

Universidad ORT Uruguay
Facultad de Ingeniería

Coopify

Plataforma de intercambio de servicios apoyada en blockchain

Entregado como requisito para la obtención del
título de Ingeniero en Sistemas

Agustin Caétano - 151997
Camilo Martínez - 180961
Ismael Puricelli - 187086
Juan Rodríguez - 171983

Tutor: Darío Macchi

2019

Declaración de Autoría

Nosotros, Agustin Caétano, Camilo Martínez, Ismael Puricelli y Juan Rodríguez, declaramos que el trabajo que se presenta en esta obra es de nuestra propia mano. Podemos asegurar que:

- La obra fue producida en su totalidad mientras realizábamos el proyecto de grado de la carrera de Ingeniería en Sistemas;
- Cuando hemos consultado el trabajo publicado por otros, lo hemos atribuido con claridad;
- Cuando hemos citado obras de otros, hemos indicado las fuentes. Con excepción de estas citas, la obra es enteramente nuestra;
- En la obra, hemos acusado recibo de las ayudas recibidas;
- Cuando la obra se basa en trabajo realizado conjuntamente con otros, hemos explicado claramente qué fue contribuido por otros, y qué fue contribuido por nosotros;
- Ninguna parte de este trabajo ha sido publicada previamente a su entrega, excepto donde se han realizado las aclaraciones correspondientes.



Agustin Caétano
5 de setiembre de 2019



Camilo Martínez
5 de setiembre de 2019



Ismael Puricelli
5 de setiembre de 2019



Juan Rodríguez
5 de setiembre de 2019

Agradecimientos

Queremos aprovechar esta sección para agradecer a todas las personas que nos brindaron su apoyo, comprensión y colaboración a lo largo de todo este proceso.

En primer lugar a nuestras familias y amigos, no queremos dejar de agradecerles por el apoyo constante durante el desarrollo del proyecto así como también desde el primer día y a lo largo de toda la carrera.

A nuestro tutor Darío Macchi, por habernos acompañado, motivado y guiado en la toma de decisiones permitiéndonos un mejor desarrollo del proyecto. Agradecemos también por habernos brindado su tiempo, donde pudimos resolver problemas de una magnitud alta, por compartir sus ideas ante nuestras inquietudes y por estar siempre dispuesto para cualquier duda.

También queremos agradecer a la empresa *Create & Thrive*, a Tomás Piaggio y Joaquín Varela, el cliente del proyecto. Siempre estuvieron atentos y dispuestos ante cualquier duda o consulta por parte del equipo, su actitud y cooperación fue de una gran ayuda. Esto hizo que la experiencia de realizar el proyecto fuera ampliamente satisfactoria por nuestra parte con resultados al mismo nivel.

A su vez, les damos las gracias a los revisores por sus críticas y comentarios, fundamentales para aprender, mejorar y luego aplicar sobre el proyecto con el objetivo de mejorar la calidad del mismo.

Por último pero no menos importante, agradecemos a la Universidad ORT Uruguay por la formación académica, el uso de las instalaciones y herramientas. Por formarnos como profesionales, permitiéndonos desarrollar este proyecto con la actitud y responsabilidad que el mismo requiere.

A todos aquellos mencionados anteriormente, muchas gracias.

Abstract

Hoy en día, el mercado de servicios es el más importante dentro de una sociedad. Los servicios son brindados por particulares o empresas y con la llegada de la tecnología y la *web*, hasta son exportados y consumidos de una manera más sencilla y a escala global. El objetivo de este proyecto es crear una plataforma para el intercambio de servicios entre los usuarios sin tener que manejar dinero real. En la plataforma se cuenta con una moneda virtual llamada “coopy” que resuelve el problema del intercambio “a trueque” de servicios, donde no es fácil lograr una conexión. Esta moneda virtual será descentralizada y regulada con tecnologías *blockchain*, para asegurar la transparencia y confiabilidad de la plataforma.

Podemos ilustrar lo anterior con los siguientes dos ejemplos:

1. Juan es guitarrista y ofrece su servicio de “clases de guitarra” en Coopify. Maria es diseñadora, y ofrece su servicio de “diseño de logo” en Coopify. A Maria le interesa aprender a tocar la guitarra y le ofrece a Juan intercambiar servicios. Juan piensa que es una buena idea ya que él precisa un logo para su emprendimiento musical, por lo que intercambian servicios.
2. Carla es masajista y ofrece “sesiones de terapia” en Coopify. A Carla le interesa asesorarse en la parte contable y publica una solicitud. Jorge es estudiante de contador y ofrece “asesoramiento en balances” en Coopify. Jorge ve la solicitud de Carla y ofrece sus servicios, pero él no está interesado en sesiones de terapia, por lo que Carla “paga” el servicio de Jorge por 20 coopies.

En resumen, Coopify es una plataforma con el objetivo de fomentar la cooperación dentro de la comunidad. Los usuarios pueden publicar sus servicios por un valor en coopies, negociar posibles intercambios y dejar reseñas sobre su experiencia para futuros consumidores. Si dos usuarios ofrecen servicios que interesan a ambas partes, lo pueden intercambiar. En caso contrario, se puede contratar el mismo abonando su valor en coopies. Los usuarios podrán hacerse de coopies cumpliendo metas o ayudando al sistema a crecer.

Este sistema consta de tres componentes principales: un sitio web y dos servidores para el manejo de lógica de negocio y comunicación con *blockchain* respectivamente.

Para la gestión del proceso de desarrollo de esta aplicación se optó por utilizar metodologías ágiles por lo que la arquitectura y diseño de la solución fueron contruidos de forma incremental con el correr de las iteraciones.

Palabras clave

Coopify, Coopi, *Create & Thrive*, *Blockchain*, Servicio, React, NodeJS, Ethereum, *Smart Contract*, Scrum, Rinkeby, Kanban, MVP, PWA, Solidity, ERC20

Glosario

A

Action plan: Actividades surgidas a partir de las ideas planteadas en la *sprint retrospective* para poner en marcha en el siguiente *sprint*.

ANII: Agencia Nacional de Investigación e Innovación [6]

Address (Ethereum): Se define como una serie de números y letras, distinguiéndose entre mayúsculas y minúsculas. Dicha serie va a identificar a una *Wallet*.

B

Bitbucket: Servicio de alojamiento para proyectos que utilizan control de versionado. [76]

Blockchain: “Una cadena de bloques conocida en inglés como *blockchain*, es una estructura de datos en la que la información contenida se agrupa en conjuntos (bloques) a los que se les añade meta-información relativa a otro bloque de la cadena anterior en una línea temporal, de manera que gracias a técnicas criptográficas, la información contenida en un bloque solo puede ser repudiada o editada modificando todos los bloques posteriores”. [82]

BPM: “Es una disciplina de gestión compuesta de metodologías y tecnologías, cuyo objetivo es mejorar el desempeño (eficiencia y eficacia) y la optimización de los procesos de una organización” [78]

C

CircleCI: Herramienta que permite a los equipos construir proyectos de calidad a gran escala.

Code Review: Actividad de aseguramiento de calidad donde uno o más integrantes de un equipo revisar el código fuente en busca de una mejor calidad de código, defectos, transferir conocimientos y encontrar posibles mejores soluciones.

Commit: Es la confirmación de un conjunto de cambios provisorios de forma permanente.

Continuous integration: “Es un modelo informático propuesto inicialmente por Martin Fowler que consiste en hacer integraciones automáticas de un proyecto lo más a menudo posible para así poder detectar fallos cuanto antes. Entendemos por integración la compilación y ejecución de pruebas de todo un proyecto.” [91]

Continuous deployment: Es una estrategia utilizada en el *software* donde cada nueva funcionalidad es liberada a través de procesos automatizados.

Coopies: Plural de la moneda descentralizada Coopi, Token ERC20 en la *block-chain* Ethereum.

Criptomoneda: Es un medio digital de intercambio que utiliza criptografía fuerte para asegurar las transacciones financieras, controlar la creación de unidades adicionales y verificar la transferencia de activos.

D

Deploy: Son todas las actividades que hacen a un sistema disponible para usar.

Dyno: “Los Dynos son contenedores Linux aislados y virtualizados que están diseñados para ejecutar código basado en un comando especificado por el usuario.” [85]

DOM Virtual: “El DOM virtual (VDOM) es un concepto de programación donde una representación ideal o “virtual” de la IU se mantiene en memoria y en sincronía con el DOM “real”, mediante una biblioteca como ReactDOM.” [58]

E

Ethereum: Es una plataforma *open source*, descentralizada que permite la creación de acuerdos de contratos inteligentes entre pares, basada en el modelo *block-chain*. [86]

Ether (ETH): “Al igual que otras blockchain, Ethereum tiene una criptomoneda nativa llamada Ether (ETH). ETH es dinero digital. Si has oído hablar de Bitcoin, ETH tiene muchas de las mismas características. Es puramente digital y se puede enviar a cualquier persona en cualquier parte del mundo al instante. El suministro de ETH no está controlado por ningún gobierno o empresa, está descentralizado y es escaso. Las personas de todo el mundo usan ETH para realizar pagos, como depósito de valor o como garantía.” (Traducido desde [25])

F

Feature: Es una funcionalidad.

G

GAS: El GAS es el coste que tiene el realizar una operación o un conjunto de operaciones en la red Ethereum. Estas operaciones pueden ser varias: desde realizar una transacción hasta ejecutar un contrato inteligente o crear una aplicación descentralizada. [47]

Git: Herramienta utilizada para el control del versionado.

Gitflow: Es un modelo de flujo de trabajo de Git que nos permite un mejor control de los proyectos.

GitHub: Provee alojamiento para productos de software de control de versiones Git.

Google Drive: Servicio de almacenamiento de datos en la red.

H

Heroku: *“Plataforma como servicio de computación en la Nube que soporta distintos lenguajes de programación.”* [89]

HTTPS: *“El Protocolo seguro de transferencia de hipertexto (en inglés: Hypertext Transfer Protocol Secure o HTTPS), es un protocolo de aplicación basado en el protocolo HTTP, destinado a la transferencia segura de datos de Hipertexto, es decir, es la versión segura de HTTP.”* [101]

I

IDE online Remix: IDE para programar en el lenguaje Solidity, para desarrollar *Smart Contracts*. [61]

J

JSON RCP: Protocolo de llamada a procedimiento remoto (RPC) sin estado y ligero. Principalmente, esta especificación define varias estructuras de datos y las reglas en torno a su procesamiento. Es independiente del transporte, ya que los conceptos se pueden usar dentro del mismo proceso, a través de sockets, a través de http o en muchos entornos de envíos de mensajes. [31]

K

Kanban: Es un método para gestionar el trabajo intelectual, con énfasis en la entrega justo a tiempo, mientras no se sobrecarguen los miembros del equipo. En este enfoque, el proceso, desde la definición de una tarea hasta su entrega al cliente, se muestra para que los participantes lo vean y los miembros del equipo tomen el trabajo de una cola. [93]

L

Lintern: Herramienta con distintas reglas precargadas de buenas prácticas para un buen desarrollo tanto en *Backend* como en *Frontend*.

M

Man-in-the-middle attack: *“En criptografía, un ataque de intermediario, tam-*

bién conocido como ataque Janus, es un ataque en el que se adquiere la capacidad de leer, insertar y modificar a voluntad. El atacante debe ser capaz de observar e interceptar mensajes entre las dos víctimas y procurar que ninguna de las víctimas conozca que el enlace entre ellos ha sido violado”. [75]

Merge: Se utiliza para fusionar una o más ramas dentro de otra.

Metamask: Es un puente que le permite visitar la red distribuida en su navegador. Incluye un almacén de identidad seguro, que proporciona una interfaz de usuario para administrar sus identidades en diferentes sitios y firmar transacciones de *blockchain*. [46]

MVP: *“El concepto se popularizó a partir de “The Lean Startup”, libro de Eric Ries, en el que hace alusión a un producto que cumple con las funcionalidades mínimas para poder validarlo con potenciales clientes utilizando el menor esfuerzo posible.”. [45]*

N

Node.js: entorno de ejecución para JavaScript construido con el motor de JavaScript V8 de Chrome. [99]

P

PaaS: *Platform as a Service* - Plataforma que provee servicios similares a los de infraestructura

Paw JS - The plugable framework: *“Un framework de aplicaciones web progresivo altamente escalable y conectable facilitando una mejor experiencia para el desarrollador. Utiliza el poder de React con Webpack y está construido sobre webpack / tapable que le permite una capacidad de extensión ilimitada.”* (traducido de [68])

PostgreSQL: Poderoso sistema de base de datos relacional de código abierto con más de 30 años de desarrollo activo con características de confiabilidad, solidez y rendimiento. [32]

Private Key (PK) (Ethereum): es un número aleatorio muy grande que se utiliza para generar una clave pública. Esta clave pública es su dirección en la cadena de bloques Ethereum. La clave privada le permite firmar transacciones. [69]

Product backlog: Es una lista priorizada de todas las funcionalidades que necesita un producto.

Pull request: Actividad que permite validar el código que se va a mergear de una rama o otra.

Pusher Channels: proporciona comunicación en tiempo real entre servidores, aplicaciones y dispositivos. Los canales se utilizan para notificaciones, chat, juegos, actualizaciones de páginas web, IoT y muchos otros sistemas que requieren comunicación en tiempo real. [57]

PWA: *Progressive Web Apps (PWA)* son un tipo de aplicaciones que se podrían definir como: “Página web que se aprovecha de las tecnologías *webs* a las cuales tiene acceso para proponer una experiencia móvil similar a una aplicación nativa”. Al final, una *Progressive Web App*, es una nueva generación de aplicaciones que están a medio camino entre una web y una *app*, ya que tienen lo mejor de los dos mundos. [22]

R

React.js: biblioteca de JavaScript para construir interfaces de usuario. [37]

Redis: es un almacén de estructura de datos en memoria de código abierto (licencia BSD), que se utiliza como agente de base de datos, caché y mensaje. [60]

Rinkeby: Es una red de prueba de Ethereum. Todo proyecto de Ethereum comienza en las redes de prueba. Antes de lanzar al público una aplicación, debe ser verificada en la red de pruebas. De igual manera el propio algoritmo de Ethereum. En conclusión, las redes de prueba de Ethereum, las vamos a usar constantemente. [59]

Rinkeby.Etherscan: Permite explorar y buscar en la cadena de bloques de Rinkeby transacciones, direcciones, tokens, etc. [24]

S

SCRUM: “*Es un proceso en el que se aplican de manera regular un conjunto de buenas prácticas para trabajar colaborativamente, en equipo, y obtener el mejor resultado posible de un proyecto. Estas prácticas se apoyan unas a otras y su selección tiene origen en un estudio de la manera de trabajar de equipos altamente productivos.*” [3]

Selenium: Entorno de pruebas automatizado para *frontend*. [63]

Service Worker: “*Un service worker es una secuencia de comandos que tu navegador ejecuta en segundo plano, separado de una página web, abriéndoles la puerta a funciones que no necesitan una página web ni interacción de usuario.*” [19]

Slack: Herramienta de comunicación en equipo.

SOLID: En ingeniería de software, SOLID (*Single responsibility, Open-closed, Liskov substitution, Interface segregation* y *Dependency inversion*) es un acrónimo mnemónico introducido por Robert C. Martin a comienzos de la década del 2000 que representa cinco principios básicos de la programación orientada a objetos y el diseño. Cuando estos principios se aplican en conjunto es más probable que un desarrollador cree un sistema que sea fácil de mantener y ampliar con el tiempo. [105]

Solidity: Lenguaje de alto nivel orientado a contratos. Su sintaxis es similar a la de JavaScript y está enfocado específicamente a la Máquina Virtual de Ethereum (EVM). [65]

Smart Contracts: Un contrato inteligente (en inglés *Smart contract*) es un programa informático que facilita, asegura, hace cumplir y ejecuta acuerdos registrados entre dos o más partes (por ejemplo personas u organizaciones). Como tales ellos les ayudarían en la negociación y definición de tales acuerdos que causarían que ciertas acciones sucedan como resultado de que se cumplan una serie de condiciones específicas. [104]

Sprint: Un *sprint* es cada una de las iteraciones que encontramos dentro de un proyecto *SCRUM*. Al finalizar el mismo se va a contar con un incremento del producto que aporta valor al cliente.

Sprint review: Ceremonia realizada al finalizar cada *sprint* entre el equipo y el cliente, donde se obtiene *feedback* del producto.

Startup: Es una empresa emergente con una estructura empresarial orientada a conseguir un rápido crecimiento en base a una innovación. [107]

Svn: Es una herramienta de control de versiones

T

Tokens de autenticación: “Para evitar que en cada llamada el usuario deba entregar su clave y contraseña se usa un token, que no es más que una cadena de texto bien larga, encriptada con una clave. Según se hace el login, el servidor devuelve al cliente un token y el cliente en futuros accesos entrega ese token para certificar que está autenticado.” [5]

Tokens ERC20: Son contratos inteligentes que se ejecutan en la *blockchain* de Ethereum. La mayoría de los tokens creados en Ethereum cumplen con el estándar ERC20, aunque también existen otros estándares como por ejemplo ERC223 y ERC721.

W

Wallet: Es una billetera digital segura, la cual se utiliza para guardar, enviar y recibir dinero digital. Almacena las claves públicas y privadas para poder realizar las acciones previamente mencionadas.

Whatsapp: Herramienta de comunicación que permite enviar y recibir mensajes.

web3js: (Ethereum JavaScript API): es una colección de librerías que le permite interactuar con un nodo Ethereum local o remoto, utilizando una conexión HTTP, WebSocket o IPC.

Índice

1. Introducción	18
1.1. Presentación del equipo	18
1.2. Estructura del documento	19
1.3. Selección de proyecto	21
1.4. Objetivos	21
1.5. Descripción del cliente	23
1.6. Entorno Conceptual de ORT Software Factory	23
2. Problema	24
2.1. Contexto del problema	24
2.1.1. Oportunidades	25
2.1.2. Público objetivo	25
2.2. Análisis de competidores	26
3. Solución	27
3.1. Solución propuesta	27
3.2. Plan de negocio	29
3.3. Alcance del proyecto	29
3.4. Principales funcionalidades	29
4. Marco metodológico	31
4.1. Características del proyecto	31
4.2. Características del equipo	34
4.3. Ciclo de vida	35
4.4. Metodología ágil	38
4.4.1. Kanban	38
4.4.2. Scrum	39
5. Ingeniería de requerimientos	40
5.1. Técnicas de relevamiento	40

5.1.1.	Entrevistas con el cliente	41
5.1.2.	Estudio de herramientas similares	42
5.2.	Técnicas de validación	42
5.2.1.	En conjunto con el cliente	42
5.2.2.	Prototipos de baja fidelidad	43
5.3.	Especificación	44
5.3.1.	Requerimientos funcionales	45
5.3.2.	Requerimientos no funcionales	49
5.4.	Conclusiones de ingeniería de requerimientos	51
5.5.	Lecciones aprendidas	52
6.	Arquitectura	53
6.1.	Descripción de la arquitectura	53
6.1.1.	Arquitectura en capas	54
6.2.	Vistas de arquitectura	54
6.2.1.	Vista de despliegue	55
6.2.2.	Vista de implementación	56
6.2.3.	Vista lógica	57
6.2.4.	Vista de escenarios	62
6.3.	Atributos de calidad	63
6.3.1.	Performance	63
6.3.2.	Modificabilidad	68
6.3.3.	Seguridad	72
6.3.4.	Usabilidad	74
6.4.	Conclusión de Arquitectura	82
6.5.	Lecciones aprendidas	82
7.	Análisis del problema tecnológico	84
7.1.	Selección de tecnología Blockchain	84
7.2.	Coopies (Token ERC20 / CPI)	87
7.3.	Rinkeby Test Network	89
7.4.	Ether	90
7.5.	Backend	90
7.5.1.	NodeJS	90
7.5.2.	Ruby on Rails	91
7.5.3.	Framework seleccionado y justificación	91
7.6.	Frontend	92
7.6.1.	Para la creación de web SPA	92
7.6.2.	Para la creación de interfaces móviles	93
7.6.3.	Tecnologías seleccionadas y justificación	94
7.6.4.	Redux-Saga middleware	94

7.7.	Servicios de alojamiento	96
7.8.	Ambientes y proceso de despliegue	96
7.9.	Servicios de terceros	97
7.9.1.	Pusher	98
7.9.2.	Cloudinary	98
7.9.3.	Sendgrid	99
7.9.4.	Etherscan	99
8.	Gestión de Proyecto	100
8.1.	Etapas del proyecto	100
8.1.1.	Investigación	101
8.1.2.	Desarrollo	103
8.2.	Etapas de investigación	104
8.2.1.	Aplicación de Kanban	104
8.2.2.	Popcorn flow	106
8.2.3.	Entregable	108
8.3.	Etapas de desarrollo	108
8.3.1.	Implementación de Scrum	109
8.3.2.	Planificación	110
8.3.3.	Ejecución	116
8.3.4.	Evaluación	118
8.4.	Análisis de resultados obtenidos	121
8.4.1.	Investigación	121
8.4.2.	Desarrollo	123
8.5.	Gestión de riesgos	129
8.5.1.	Estrategia y categorización	129
8.5.2.	Identificación	130
8.5.3.	Evolución	137
8.6.	Gestión del tiempo	139
8.6.1.	Lecciones aprendidas	141
8.7.	Gestión de la comunicación	142
8.7.1.	Lecciones aprendidas	144
9.	Gestión de Calidad	145
9.1.	Actividades de calidad	146
9.2.	Plan de calidad	148
9.3.	Aseguramiento de la calidad	149
9.4.	Conclusiones de calidad	160
9.5.	Lecciones aprendidas	160
10.	Gestión de la configuración	162

10.1. Elementos de configuración	162
10.2. Herramienta de versionado	163
10.2.1. Etapa de investigación	163
10.2.2. Etapa de desarrollo de software	164
10.3. Organización de los repositorios	165
10.3.1. Etapa de investigación	165
10.3.2. Etapa de desarrollo de software	166
10.4. Gestión de versiones	167
10.4.1. Convención de branching	168
10.5. Gestión de cambios	168
10.6. Gestión de incidentes	169
10.7. Conclusiones de gestión de configuración	170
10.8. Lecciones aprendidas	170
11. Conclusiones del proyecto	172
11.1. Estado actual	172
11.2. Conclusiones	174
11.3. Lecciones aprendidas	176
11.4. Próximos pasos	178
11.4.1. Posibles mejoras	178
12. Referencias Bibliográficas	182
13. Anexos	191
13.1. Ingeniería de requerimientos	191
13.1.1. Evidencias de relevamiento	191
13.1.2. Contrapropuesta realizada	193
13.1.3. Comparación herramientas similares	197
13.1.4. Prototipos de interfaz gráfica	198
13.2. Arquitectura	206
13.2.1. Estructura de los archivos del sistema	206
13.2.2. Diagramas de Secuencia	212
13.3. Documentación API	217
13.3.1. API Rest en servidor de Backend	218
13.3.2. API Rest en servidor de Ethereum	222
13.4. Gestión de proyecto	222
13.4.1. Comparación metodología tradicional y ágil	222
13.4.2. Comparación Kanban y Scrum	224
13.4.3. Kanbanize	225
13.4.4. Evidencia de uso de Teamwork	227
13.4.5. Valores de impacto y probabilidad de riesgos	228

13.4.6. Evidencia de la gestión de comunicación	229
13.5. Gestión de calidad	231
13.5.1. Ejemplo de pull request	231
13.5.2. Revisiones	233
13.6. Gestión de la configuración	242
13.6.1. Comparación GitHub y Bitbucket	242
13.7. Investigación de tecnologías para implementar una Blockchain	243
13.7.1. Introducción	243
13.7.2. Estado del arte	244
13.7.3. Desarrollo	247
13.7.4. Diferencias entre tecnologías	252
13.7.5. Conclusión	255
13.7.6. Glosario	256

1 Introducción

El presente documento tiene como objetivo describir el proyecto de grado **Coopify** entregado como requisito para la obtención del título de Ingeniero en Sistemas, el cuál tuvo lugar desde Agosto 2018 hasta Septiembre 2019.

El proyecto estuvo bajo la tutoría del Ing. Darío Macchi, fue realizado a través del laboratorio *ORT Software Factory* y para la empresa *Create & Thrive*.

1.1 Presentación del equipo

El equipo se compone de la siguiente de la manera:

- Agustín Caétano
- Camilo Martínez
- Ismael Puricelli
- Juan Rodríguez

Dos de los integrantes del equipo ya se conocían desde el sexto semestre de la carrera lo cuál propició una base sólida a la hora de iniciar con un trabajo de tal magnitud como lo es un proyecto de fin de carrera. El resto de los integrantes se unieron a través del foro brindado por la facultad.

El grupo adquirió afinidad desde un primer momento, esto permitió lograr una situación ideal para afrontar un proyecto de un año de duración.

1.2 Estructura del documento

Problema

Aquí el lector encontrará la oportunidad que intenta aprovechar *Create & Thrive* con el producto a desarrollar.

Solución

Se detalla la propuesta realizada para dar forma a la oportunidad percibida por el cliente.

Marco metodológico

Se exponen las características del proyecto y del grupo, en base a esto definimos las metodologías a utilizar a lo largo del proyecto.

Ingeniería de requerimientos

Aquí encontraremos el resultado del relevamiento y validación de los requerimientos funcionales y no funcionales, así como las técnicas utilizadas en cada una de estas etapas.

Arquitectura

Se especifica la arquitectura de la solución planteada en conjunto con las justificaciones del diseño y como atacamos los atributos de calidad.

Análisis del problema tecnológico

Se detallan los principales problemas tecnológicos que comprenden la solución.

Gestión de Proyecto

Se detallan las actividades correspondientes a la gestión del proyecto, como: gestión de riesgos, tiempos y comunicación. Agregado a esto también se detalla como se implementaron las metodologías a utilizar, concluyendo con el análisis de los resultados obtenidos.

Gestión de Calidad

Entraremos en detalle en las diferentes actividades que llevamos a cabo para asegurarnos de obtener un producto de software de calidad.

Gestión de la configuración

Se detalla el manejo del versionado de los elementos de configuración de *software*.

Conclusiones del proyecto

En esta parte se exponen las conclusiones generales así como también las lecciones aprendidas a lo largo del proyecto, en conjunto con los próximos pasos a dar sobre el mismo.

Referencias Bibliográficas

Detalle de las referencias utilizadas a lo largo de la documentación.

Anexos

Datos que complementan a los temas expuestos en los capítulos anteriores.

1.3 Selección de proyecto

A comienzos de agosto de 2018 la universidad ORT Uruguay presentó una serie de proyectos propuestos por empresas con el objetivo de que estudiantes próximos a recibirse pudieran escoger uno e implementarlo a modo de tesis final.

Para dar contexto a la situación en la que se encontraba el equipo, inicialmente se habían seleccionado tres propuestas en la feria de exposición de proyectos llevada a cabo en la universidad ORT Uruguay.

El equipo contaba con interés en las tecnologías emergentes, en particular *blockchain*. Por lo que este proyecto contaba con un atractivo especial.

Luego procedimos a una etapa de entrevistas con los que captaron nuestro interés. En la primera entrevista con Tomás Piaggio y Joaquín Varela se mostraron muy afines a seleccionarnos como equipo responsable de este proyecto. Luego esa afinidad se formalizó eligiéndonos como grupo oficial.

1.4 Objetivos

A continuación se detallan los principales objetivos del proyecto.

Conocer y aprender nuevas tecnologías

Como mencionamos brevemente de forma previa, los integrantes del equipo contaban con interés en aprender y especializarse en el uso de las tecnologías de vanguardia. Principalmente el uso de tecnologías *blockchain* nos fue atractivo. Este objetivo generó múltiples desafíos a lo largo del proyecto, ellos conformaron una oportunidad para crecer profesionalmente.

Obtener un excelente resultado académico

Desde un principio el principal objetivo del equipo fue completar esta instancia para obtener un resultado académico de excelencia. Esto es importante ya que el proyecto nos otorga la posibilidad de aplicar los conocimientos adquiridos a lo largo de toda la carrera, y es un reflejo de que tan buenos estudiantes hemos sido en el transcurso de este ciclo.

Brindar una solución a un problema real

Nos da la posibilidad como equipo, de salir de lo que significan el trabajos obligatorios, en donde se manejan contextos básicos que simplifican la realidad. El construir un producto real, prácticamente desde cero, implica múltiples desafíos y a su vez nos da la posibilidad de aprender en áreas que tenemos poco desarrolladas.

Crear una plataforma de intercambio de servicios confiable y segura

El atractivo del proyecto fue en gran medida la implementación y uso de la tecnología *blockchain* para el manejo de las transacciones en el sistema, así como también la inclusión de una moneda virtual en la misma. La aplicación de esta tecnología se traduce en una mayor seguridad y confiabilidad de la solución, por lo tanto es importante lograr un producto que la aproveche al máximo.

Buena usabilidad y fácil aprendizaje

Otro de los desafíos que implica este proyecto radica en la construcción de un producto que sea sencillo de utilizar y con una curva de aprendizaje baja o nula. Existe un gran desafío en este punto, dado que nuestra experiencia consiste en el desarrollo de soluciones en donde se hace un mayor foco en otros atributos de calidad, por lo que la usabilidad de la mismas queda generalmente relegada.

Construcción de un mínimo producto viable

El cliente manifestó la intención de presentar el proyecto Coopify a la ANII con el fin de lograr un emprendimiento a nivel personal. Para esto es importante que el producto construido sea un MVP [45], o producto mínimo viable, esto se traduce en un producto que cumpla con las funcionalidades mínimas que permitan presentar y validar la idea de **Coopify**.

Cumplimiento de expectativas

Otro objetivo planteado por el equipo es el cumplimiento de las expectativas del cliente. Dado que el confió en nosotros para el desarrollo de la solución, consideramos imprescindible que se encuentre satisfecho con el trabajo realizado.

1.5 Descripción del cliente

Nuestro cliente es la empresa *Create & Thrive*, una *startup* fundada por Tomás Piaggio y Joaquín Varela, ambos ingenieros en sistemas graduados de la Universidad ORT Uruguay. La misma opera con clientes situados en *Silicon Valley* y planea emprender mediante la presentación de este proyecto a la ANII.

1.6 Entorno Conceptual de ORT Software Factory

El Laboratorio de Ingeniería de Software de la Universidad ORT Uruguay, denominado ORT *Software Factory* (ORTsf) se dedica a la enseñanza de Ingeniería de Software y a la producción de software en forma industrial [71].

ORTsf está abocada fundamentalmente a desarrollar en los alumnos las habilidades que un profesional de las Tecnologías de la Información debe dominar y aplicar. Para esto se ha diseñado un método de enseñanza para estudiantes de fin de carrera, que apoyados por tutores especializados, trabajan en equipos de desarrollo aplicando prácticas avanzadas de Ingeniería de Software en proyectos reales.

Estos proyectos surgen en colaboración con la industria o como apoyo a las líneas de investigación del departamento. Buscan construir productos que satisfagan a sus clientes, promover el aprendizaje de prácticas reales de ingeniería de software y proveer tecnología probada al mercado.

2 Problema

En la siguiente sección se pasará a detallar la oportunidad que el proyecto busca aprovechar, quienes pueden llegar a ser sus interesados, y el desafío que implica a nivel de requerimientos funcionales y no funcionales.

Luego de tener los primeros acercamientos con la empresa *Create & Thrive*, se le informó al equipo del objetivo del proyecto y la oportunidad a explotar. La misma es la de agregar confiabilidad en el área de intercambio de servicios mediante el uso de tecnologías *blockchain*.

2.1 Contexto del problema

Para interiorizarnos en el problema presentado cabe destacar que existen particulares que tienen excelentes habilidades y conocimientos acerca de las cosas que comúnmente necesitamos o deseamos a diario. Pero no siempre cuentan o tienen la posibilidad de acceder a un medio en el que pueden exponer dichas virtudes.

Estas habilidades y conocimientos son las que conoceremos como servicios, las cuáles serán ofrecidas a los interesados. Dichos servicios son los que le irán brindando más notoriedad en el sistema a los usuarios. Aquí se presenta otro problema, el mismo trata de en que forma podemos hacer que los usuarios logren esa exposición ante el resto de los usuarios del sistema.

2.1.1 Oportunidades

Importancia del mercado de servicios

No es una novedad que los servicios han adquirido una gran popularidad en estos días. Se han expandido globalmente y los consumimos a diario. Vemos un valor importante en ésta área ya que podemos concentrar a los mismos en un solo lugar donde cada usuario busque y consuma el servicio de su agrado.

Internacionalización y democratización de los servicios

Con la globalización en la que vivimos hoy en día, podemos acceder y obtener cualquier cosa que queramos a solo un par de pasos de distancia, y el mercado de servicios no es la excepción. El uso de las tecnologías de la información ha abierto las puertas para que los servicios sean consumidos a nivel mundial. Esto se apoya en el conocimiento y la adopción tecnológica del público general.

De no contar con este alcance globalizado, el servicio expuesto por un usuario alcanzaría un menor conjunto de mercados. Sin embargo con ella desde la comodidad de nuestro hogar podemos acceder a múltiples mercados que eran previamente desconocidos.

Aparición de tecnologías *Blockchain*

Hoy en día esta tecnología ha adquirido una relevancia muy grande en el mundo entero, debido a su estructura descentralizada que asegura la permanencia e integridad de las transacciones. Dichas características otorgan a estas tecnologías un gran atractivo para sustituir instituciones reguladoras (bancos, instituciones financieras y gubernamentales, etc.).

2.1.2 Público objetivo

Serán aquellos habituados al uso de la tecnología en general, a aplicaciones de similar estilo y/o que poseen interés en intercambiar servicios. Personas asiduas a realizar compras vía *online* y a interactuar e intercambiar opiniones con otros.

2.2 Análisis de competidores

En esta sección se muestra un análisis de una alternativa diferente a Coopify. Se recolectó información tanto lógica (funcionamiento) como también desde el punto de vista de la usabilidad esperada. Se notaron diferencias en las funcionalidades ofrecidas, lo que permitió poder explotar esos puntos y sacar ventaja al respecto.

A continuación se detalla lo mencionado anteriormente y cuáles fueron los diferentes resultados obtenidos.

Simbi

Se decidió analizar a Simbi ya que es una empresa establecida en el mercado con un sitio estable y en funcionamiento. Permite a sus usuarios exponer servicios e intercambiarlos utilizando una moneda propia.

La primer gran diferencia es que en Coopify se maneja una moneda virtual descentralizada, regulada por *blockchain*, brindando seguridad y transparencia a la hora de los pagos por los servicios. Esto le permite a los usuarios tener confianza en el sitio ya que en Simbi no se sabe como garantizan el uso de su respectiva moneda.

Dicha diferencia puede parecer algo menor pero sin embargo aporta seguridad y confiabilidad al usuario ya que impide que alguien realice cambios en la transacciones ya efectuadas, y cualquier transacción (normal o anómala) que se realice queda registrada en todos los bloques de la *blockchain*. En el caso del competidor al contar con un registro de transacciones tradicional, la permanencia, completitud e integridad de las mismas no esta asegurada.

Otra diferencia importante es la posibilidad de calificar tanto al usuario como al servicio brindado de forma independiente, se pensó en esta posibilidad ya que la experiencia con el usuario no siempre iba a coincidir con la del servicio en sí. Esto puede ser debido a que el trato brindado por el usuario que brinda el servicio no haya sido el adecuado pero que el servicio en sí contratado fuera completamente lo contrario, en otras palabras haya colmado por completo las expectativas del usuario que contrata el mismo.

Para finalizar, algunos problemas y funcionalidades básicas se encuentran ya resueltos por Simbi. Con esto nos referimos a por ejemplo la conversación entre las dos partes y el intercambio o pago por el servicio en cuestión.

3 Solución

En las siguientes secciones detallaremos la solución al problema identificado, luego de una serie de reuniones con el cliente.

3.1 Solución propuesta

Se propuso la creación de una plataforma donde se pueda intercambiar servicios entre usuarios, sin la necesidad del uso de dinero real.

Teniendo dos modalidades a la hora intercambio, mediante:

- **Trueque** (Intercambio del servicio de un usuario por el de otro).
- **Coopi** (Moneda virtual, para resolver el intercambio de servicios, cuando estos no son comparables, o un usuario necesita de un servicio brindado por otro y paga por el).

Para asegurar la transparencia y confiabilidad de la plataforma, la moneda virtual sera descentralizada y regulada por tecnologías *blockchain*.

Una vez que se encuentre definido el perfil del usuario podrá comenzar a intercambiar servicios. Para ello deberá de realizar una publicación indicando los siguientes detalles:

- Qué servicio brinda
- Que tipo de hito está disponible para el servicio prestado (Hora, Sesión y/o Producto final)
 - Hora: Servicios que se cobren por hora. Por ejemplo, cursos particulares con horario predefinido.

- Sesión: Servicios que se cobren por sesión y no por hora. Por ejemplo, servicio de reparación a domicilio.
- Producto Final: Servicios que se cobren por producto final realizado. Por ejemplo, un producto de software.
- Cuantos coopies se espera obtener por cada uno de los hitos propuestos (ejemplo: 3 por Hora, 5 por Sesión o 30 por Producto final).

Una vez publicado el servicio dentro de Coopify es posible que dos usuarios comiencen el proceso de negociación para llegar a un acuerdo. Para esto el sistema proveerá de una herramienta integrada donde la negociación entre ambas partes tomará forma (a modo visual una ventana de *chat* donde sera posible negociar).

Aquí los usuarios podrán resolver si se llega a un intercambio equivalente de servicios (trueque) o si se recurre a los coopies para abonar por el servicio prestado. Cuando los términos del intercambio de servicio se encuentren definidos, los usuarios procederán con el mismo.

El intercambio de servicios no se limita únicamente a un usuario y un proveedor, en efecto, varios usuarios podrán contratar un mismo servicio, es decir, un determinado servicio ofrecido por un cierto usuario podría ser contratado y pagado en conjunto por varios otros usuarios, permitiendo dividir el costo entre varios usuarios.

Es importante señalar que Coopify se encarga únicamente de conectar a los usuarios y no se hace responsable si un usuario no brinda el servicio luego de ser contratado, esto se verá impactado en el *rating* de los mismos.

Un usuario cuenta con la posibilidad de gastar coopies para adquirir servicios pero para ello debe de poder adquirirlos de alguna manera. En esta instancia los medios por los cuales un usuario puede adquirir coopies son los siguientes:

- Al registrarse se le adjudicará un monto predefinido.
- Realizando intercambios de servicios con otros usuarios del sistema.
- Sistema de metas y logros
- Con toda acción que enriquezca a la plataforma (por ejemplo: darle visibilidad en las redes sociales, referir a amigos que luego se unan a la plataforma)

3.2 Plan de negocio

Previo a abordar un proyecto, es importante hacer un análisis del mismo con el fin de determinar su posible viabilidad y rentabilidad a futuro. Para esto se indagó en conjunto con el cliente sobre un posible plan de negocio para Coopify.

Dado que se trata de un *MVP* el cliente manifestó que no era necesaria la inclusión de medios de pago durante esta primera etapa. Sin embargo, este nos comentó que su idea a futuro es incluir la posibilidad de que los usuarios puedan comprar *coopies* a través de dinero real en donde dichas transacciones se delegarían a un proveedor de pagos de terceros.

En cuanto a la monetización de Coopify, el cliente expresó interés en la inclusión de anuncios en la plataforma a través de *Google AdSense* ¹, de esta manera se contaría con dos modalidades para la misma:

- **Free:** Plataforma con anuncios incluidos y la posibilidad de comprar *coopies*.
- **Premium:** Plataforma libre de anuncios junto a la posibilidad de comprar *coopies*.

3.3 Alcance del proyecto

Las funcionalidades están descritas con más detalle en los requerimientos funcionales (ver 5.3.1), son las que conforman el alcance.

3.4 Principales funcionalidades

Dentro de todas las funcionalidades pensadas para la plataforma, podemos destacar como más importantes o que conforman lo que sería el núcleo de la misma:

- **Publicar servicio:** El usuario puede publicar un servicio, proporcionando los datos necesarios y cuales son sus preferencias en cuanto a si acepta trueque o cuantos *coopies* cobra por el servicio.
- **Negociar servicio:** Un usuario puede negociar con otro, por un servicio,

¹Se trata de uno de los productos de la red de publicidad en línea de Google. Básicamente, permite a los editores obtener ingresos mediante la colocación de anuncios en sus sitios web, ya sean de texto, gráficos o publicidad interactiva avanzada. [88]

mediante un *chat*. Y además enviar en cualquier momento desde allí una proposición. Esta proposición puede ser libre (proposición con los parámetros que se le ocurran al que quiere adquirir el servicio), o limitada (a las opciones que plantea el usuario que publico el servicio).

- **Pagar servicio con coopies:** Cuando un usuario acepta una proposición enviada por otro usuario, inmediatamente se dispara la transacción (en caso de que sea mediante coopies).
- **Saber cuantos coopies tiene el usuario:** Saber fácilmente el saldo de coopies que tiene el usuario al momento.
- **Ver las transacciones realizadas por el usuario:** Poder obtener un listado de transacciones realizadas por el usuario, un histórico.

4 Marco metodológico

En este capítulo se analizarán las características del proyecto y del equipo. Este análisis servirá como base para posteriormente detallar las metodologías y ciclos de vida escogidos y aplicados durante el transcurso del mismo.

4.1 Características del proyecto

Uno de los factores que influyen sobre la elección del marco metodológico y por ende resulta de gran importancia analizar son las características del proyecto escogido. A continuación se destacan las principales características del mismo en base al análisis realizado por el equipo.

Utilizar y aprender nuevas tecnologías

Este proyecto supone la posibilidad de aprender y capacitarse en un conjunto de tecnologías emergentes y cada vez más relevantes, entre ellas destaca el uso de una plataforma *blockchain*. Eso fue un factor muy interesante que motivó al equipo y se demostró como una gran oportunidad.

Producto final funcional mínimo (MVP)

Al comenzar el proyecto el cliente fue bien claro sobre lo que esperaba del mismo, obtener un producto con las funcionalidades mínimas y que aporte el máximo valor posible. Esto significó una oportunidad para poder construir algo que realmente va a ser utilizado por usuarios finales y que a su vez generó otros desafíos los cuales se listan a continuación:

- Poder gestionar un proyecto desde su comienzo hasta su posible puesta en producción.
- El hecho de contar con un cliente real con un proyecto el cual su éxito y presentación a la ANII depende en gran medida de nuestro compromiso con el mismo.
- Manejo de tecnologías desconocidas inicialmente por el equipo como lo fueron *blockchain* y *PWA*¹.
- División de responsabilidades en el equipo.

Desarrollo profesional en diversidad de roles

Si bien todos los integrantes del equipo tienen la mayoría de sus materias rendidas y aprobadas, también carecen de experiencia real a la hora de registrar, gestionar y controlar un proyecto de estas características. Teniendo esto en cuenta el proyecto es una gran oportunidad para desarrollar habilidades como gestión de proyectos, coordinación y optimización del tiempo, habilidades blandas, etc.

Etapas bien definidas

Este proyecto se caracteriza por contar con una gran parte de investigación inicial sobre las tecnologías de desarrollo, en especial *blockchain*, y cual plataforma del mismo utilizar.

Para esto el equipo optó por definir las siguientes cuatro etapas:

- **Inicio del proyecto:** Se trata de la etapa inicial del proyecto, incluye la selección del mismo, introducción del cliente y el problema, etc.
- **Investigación:** Es una de las etapas principales del proyecto donde el equipo investigó las tecnologías y se capacitó para utilizarlas.
- **Desarrollo:** La otra etapa principal del proyecto donde el equipo tomó el resultado de la etapa previa y comenzó a construir la solución requerida.
- **Documentación:** Se trata de la etapa final del proyecto en donde el equipo se dedica a realizar la documentación final del mismo.

¹*Progressive Web App*

En la siguiente imagen se puede apreciar la división del proyecto en sus respectivas etapas.

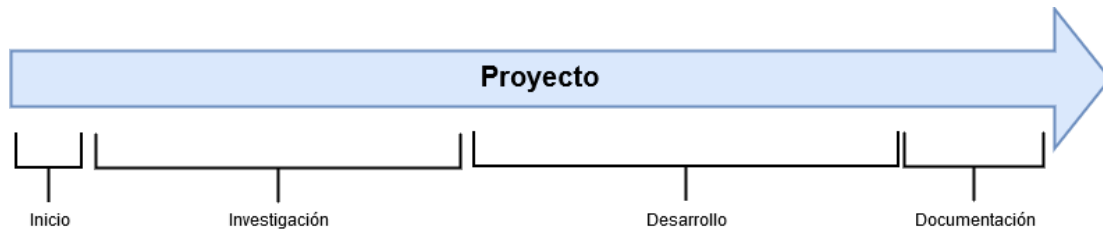


Figura 4.1: Etapas del proyecto

En la sección 8.1, se explicará más en detalle la justificación de las etapas de investigación y desarrollo, haciendo un análisis más granular sobre que consiste cada una.

Requerimientos ambiguos

Si bien la mayoría de los requerimientos fueron validados en la etapa de ingeniería de requerimientos, estos pueden ir variando conforme transcurre el proyecto, ya sea por parte del cliente como por parte nuestra al hacer pruebas de usuarios.

A su vez los requerimientos se pueden ir priorizando a medida que avanza al proyecto para la inclusión de otros elementos que el cliente o el equipo haya considerado pertinente, por lo que los requerimientos de este proyecto se caracterizan por no ser sólidos ni fijos.

4.2 Características del equipo

Otro factor que incide sobre la elección del marco metodológico son las características del equipo, por lo que en los siguientes párrafos se detallarán las principales.

Buen relacionamiento entre los integrantes

Algunos miembros del equipo ya se conocían de materias previas en donde realizaron trabajos juntos, otros sin embargo no se conocían entre sí. A pesar de esto, el equipo mantuvo una excelente relación desde el comienzo del proyecto logrando de esta manera un muy buen clima de trabajo.

Cantidad óptima de integrantes

El equipo cuenta con cuatro integrantes, un número el cual consideramos adecuado para un proyecto de grado, y en particular para el nuestro considerando el alcance del mismo, dado que permite que todos los integrantes trabajen de manera tranquila sobre el mismo con una carga horaria normal, y a su vez, facilita que el trabajo se encuentre balanceado entre todos los miembros del mismo.

Progreso en la carrera

Todos los integrantes del equipo se encuentran bastante avanzados en la carrera y próximos a recibirse, la mayoría se recibirá una vez aprobado el proyecto, por lo que contamos con los conocimientos y experiencia necesarios para encarar un proyecto de grado.

Experiencia previa

Todos los integrantes del equipo se encuentran trabajando para alguna empresa de *software*, la carga horaria varía entre 6 a 8 horas diarias según el integrante.

Camilo cuenta con bastante experiencia en el uso de la tecnología *Node.js* para desarrollo *backend*, así como también posee conocimiento y manejo de las metodologías ágiles, en particular *Scrum*, gracias a su experiencia laboral en una *Software Factory* ² en donde se emplean dichas tecnologías y metodologías de trabajo.

Agustín cuenta con algo de experiencia en *Angular* y *.NET* debido a su experiencia laboral y materias previas en la universidad como “Diseño de aplicaciones 2”.

Por último, Juan e Ismael cuentan con experiencia en el manejo de *Ruby on Rails* debido a la materia “Arquitectura de software en la practica” cursada en semestres anteriores en la universidad.

4.3 Ciclo de vida

“El ciclo de vida es la definición sobre cómo se va a organizar el proyecto, generalmente en fases, desde que se comienza hasta que termina.” [51].

En esta sección describiremos los modelos de ciclo de vida utilizados durante las principales etapas del proyecto.

Investigación

En el transcurso de la etapa de investigación el equipo decidió utilizar un ciclo de vida evolutivo, cuyo objetivo se basa en ir evolucionando el producto durante el transcurso del proyecto [14].

A continuación se muestra un diagrama ilustrando lo previamente mencionado.

²Una *Software Factory* es una empresa de software cuya misión es el desarrollo de software para clientes de acuerdo a los requisitos específicos que este solicita.

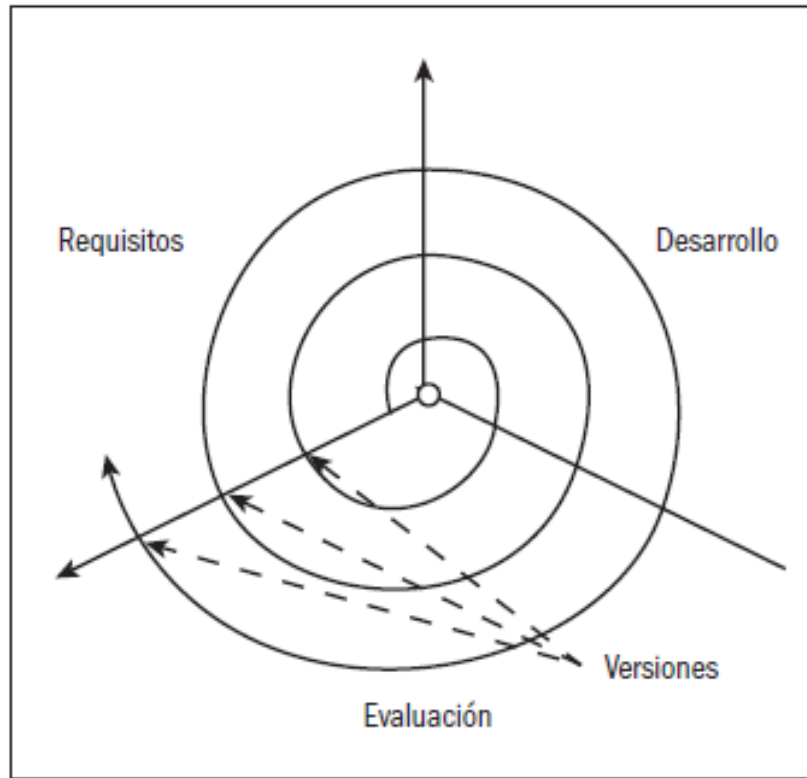


Figura 4.2: Ciclo de vida evolutivo [16]

Dado que contamos con una única iteración, vamos evolucionando la investigación realizada, aumentando el alcance de la misma y nuestro conocimiento durante el transcurso de esta etapa.

Esta decisión surge también en base a un proceso de investigación aplicado durante esta etapa denominado *Popcorn flow*, el cual se explica con más detalle en la sección 8.2.2.

A modo de resumen, se define un proceso que nos permite atacar por un lado la incertidumbre de *qué* investigar, y por el otro *cómo* hacerlo.

Desarrollo

Durante la etapa de desarrollo, sin embargo, el equipo optó por un ciclo de vida iterativo incremental.

Esto consiste en planificar varias iteraciones en las que se repite un proceso de tra-

bajo análogo con el fin de obtener un resultado completo sobre el producto final, de manera que el cliente obtenga los beneficios del proyecto de forma incremental. En cada iteración el equipo evoluciona el producto a partir de los resultados completados en las iteraciones anteriores, combinando de esta manera los ciclos iterativo e incremental [2][66].

La siguiente figura ilustra el funcionamiento de este ciclo de vida.

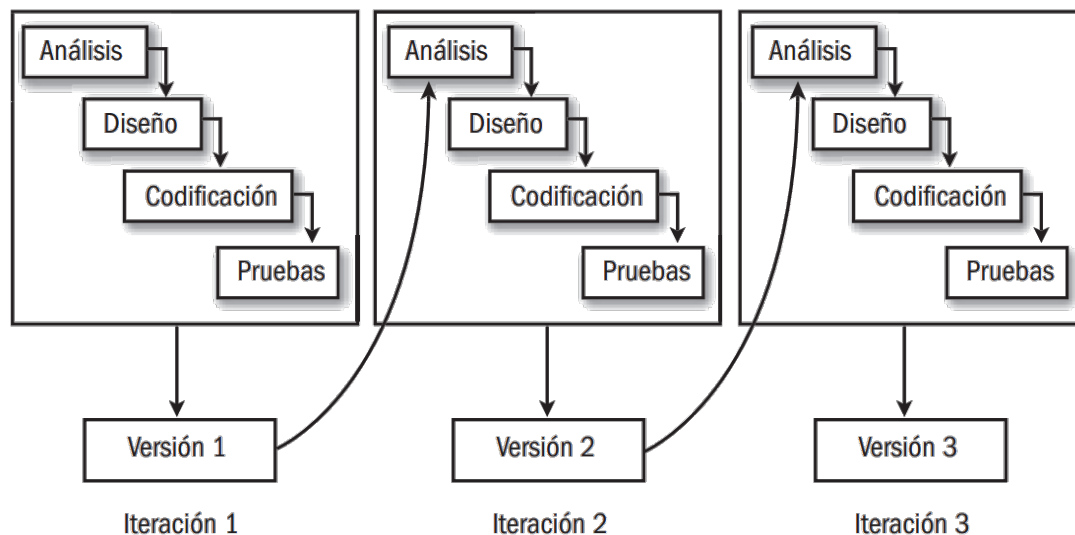


Figura 4.3: Ciclo de vida iterativo e incremental [16]

Esta decisión se consolidó luego de considerar los siguientes elementos relacionados a la etapa de desarrollo:

- Requerimientos y funcionalidades del sistema sujetos a cambios en el futuro
- Aporte de valor temprano al cliente
- Mayor aprendizaje y experiencia en todas las áreas del desarrollo durante cada iteración
- Mayor posibilidad de mitigar posibles errores

4.4 Metodología ágil

A la hora de tomar la decisión sobre que metodología utilizar, si optar por una gestión ágil o una tradicional, tuvimos en cuenta los siguientes factores:

- Volatilidad e inmadurez de las tecnologías de *blockchain*
- Requerimientos cambiantes a lo largo del proyecto
- Tener una comunicación y retroalimentación constante por parte del cliente
- Entrega de valor temprana al cliente

En base a estos factores el equipo decidió utilizar una metodología ágil durante el transcurso del proyecto.

Se puede observar una comparación de ambas metodologías en el Anexo 13.4.1.

4.4.1 Kanban

Dentro de un entorno ágil, existen varias metodologías que se pueden aplicar, sin embargo, depende del tipo de proyecto y las condiciones del mismo cual metodología conviene mas utilizar.

En nuestro caso optamos por utilizar *Kanban* como metodología de trabajo para la investigación por su fácil adaptación a los cambios y sencillez de la misma.

“Kanban es una palabra japonesa que significa “tarjetas visuales”, donde Kan es “visual”, y Ban corresponde a “tarjeta”. Las principales ventajas de esta metodología es que es muy fácil de utilizar, actualizar y asumir por parte del equipo. Además, destaca por ser una técnica de gestión de las tareas muy visual, que permite ver a golpe de vista el estado de los proyectos, así como también pautar el desarrollo del trabajo de manera efectiva.”[30].

Se puede obtener mas información sobre la manera en la cual fue aplicada, así como también las herramientas utilizadas en la sección 8.2.1.

4.4.2 Scrum

Dado que las necesidades de esta etapa eran diferentes a las de la primera etapa de investigación, el equipo creyó conveniente cambiar la metodología ágil utilizada. Considerando que en esta etapa nos interesa:

- Centrarnos en obtener un producto de software funcional
- Aportar cuanto valor sea posible al cliente
- Obtener *feedback* de manera continua para reaccionar a cambios en los requerimientos o expectativas del cliente
- Darle al cliente una muestra de lo que ha logrado el equipo en periodos cortos de tiempo

Teniendo estos intereses en mente optamos por utilizar la metodología *Scrum* ya que se alinea a los objetivos de esta etapa. La aplicación de esta ultima se detallará en la sección 8.3.1.

5 Ingeniería de requerimientos

Se denomina ingeniería de requerimientos al proceso de recopilar, analizar y verificar las necesidades del cliente para un determinado sistema de software. Es una etapa con un alto grado de importancia ya que sienta las bases del alcance del proyecto y sobre los requerimientos en los que se basará el mismo, por lo que una mala definición de estos puede traer como consecuencia los siguientes problemas:

- retrasos en el cronograma y costo adicional al cliente
- alto nivel de defectos en el software
- la entrega de un software que no satisface las necesidades del cliente

En este capítulo se ahondará sobre las principales técnicas llevadas a cabo por el equipo para el relevamiento de dichos requerimientos, así como también sus técnicas de validación, culminando con la especificación de estos.

5.1 Técnicas de relevamiento

En primer lugar, el equipo debió indagar sobre los requerimientos de nuestro cliente. Cabe destacar que dada la naturaleza de nuestro proyecto, de realizar un sistema para un cliente, el mismo nos proveyó de un listado de funcionalidades de alto nivel esperadas del sistema. Sin embargo, debimos adaptarlas para hacerlas más tangibles, analizarlas, priorizarlas y de esta manera definir un alcance para el proyecto.

En esta sección detallaremos las principales técnicas y métodos utilizados durante esta etapa.

5.1.1 Entrevistas con el cliente

Durante esta primera etapa del proyecto, el equipo se vió en la necesidad de aclarar los requerimientos provistos por el cliente, evaluar su viabilidad y poder aportar su punto de vista sobre los mismos permitiendo modificarse e incluso brindar nuevas funcionalidades que destaquen al sistema frente a otras opciones disponibles en el mercado.

La primer entrevista fue durante la feria de proyectos llevada a cabo en la Universidad *ORT* Uruguay, en donde tuvimos una instancia inicial de conocer al cliente, aprender un poco más en detalle el objetivo del sistema y sus principales funcionalidades a alto nivel.

Inmediatamente tuvimos una primera reunión presencial en la cual el equipo se limitó a detallar y discutir en conjunto con el cliente las funcionalidades previamente especificadas por ellos con el fin de tener un claro conocimiento del negocio de *Coopify*.

Se puede consultar el Anexo 13.1.1 para obtener más información sobre lo discutido y acordado durante dicha reunión.

Finalmente tuvimos una última reunión en donde presentamos una contrapropuesta inicial en base a lo acordado en la primer reunión, en conjunto con la introducción de un cambio en el *stack* tecnológico a utilizar, una *Progressive Web App*, con el objetivo de simplificar dos capas de presentación por una única y ser más mantenible a futuro. Para mas información se puede referir a la sección 7.6.2.

De estas entrevistas surgió la contrapropuesta final, con requerimientos mas detallados y concisos sobre las funcionalidades del sistema así como también las restricciones del mismo.

Contrapropuesta

Como se dijo anteriormente, luego de las reuniones mantenidas con el cliente, el equipo consideró armar un documento formal que consta de la contrapropuesta realizada detallando las funcionalidades esperadas por el sistema a alto nivel, con las modificaciones pertinentes en base a las reuniones mantenidas previamente y a su vez, con la inclusión de nuevos requerimientos.

Entre los principales cambios e inclusiones que surgieron durante el relevamiento de requerimientos surgieron las siguientes funcionalidades:

- Pago grupal de un determinado servicio
- Manejo de una *Progressive Web App* como parte del *stack* tecnológico

Por más información se puede consultar el Anexo 13.1.2.

5.1.2 Estudio de herramientas similares

Durante la etapa de relevar requerimientos, el equipo se vio en la necesidad de complementar la información relevada con el cliente a través de la investigación de herramientas similares existentes en el mercado. Esto se hizo con el fin de poder realizar ingeniería inversa ¹ sobre las mismas y poder obtener nuevas funcionalidades que complementaran a las ya propuestas en *Coopify*.

Se investigaron tres sitios con características similares a la plataforma propuesta. Todos estos sitios ofrecían funcionalidades interesantes que nos sirvieron para complementar las ya existentes y armar la contrapropuesta final.

Se puede observar una comparación de un sitio utilizado en el Anexo 13.1.3.

5.2 Técnicas de validación

En la siguiente sección expondremos como se llevó a cabo la validación de los requerimientos, que técnicas se utilizaron y las bondades de cada una de ellas.

5.2.1 En conjunto con el cliente

Para validar los requerimientos relevados, el equipo optó por validarlos en conjunto con el cliente tanto en las instancias de reunión mantenidas en forma presencial como remotas.

Por otro lado, al finalizar cada *sprint*, el equipo realiza otra validación junto al cliente con respecto a aquellas funcionalidades potenciales a desarrollar en el si-

¹La ingeniería inversa persigue el objetivo de obtener la mayor cantidad de información técnica de un producto, del cual no se tiene la más mínima información técnica de su diseño, construcción y funcionamiento, de modo que se debe partir de un todo para comprender cada pieza del sistema [38].

guiente *sprint*. Esto se hace con el fin de evaluar si el cliente desea realizar algún cambio sobre un determinado requerimiento. De este modo nos aseguramos que podamos abarcar casos en donde el cliente pueda cambiar de opinión sobre una determinada funcionalidad durante el transcurso del proyecto.

5.2.2 Prototipos de baja fidelidad

Otra manera de validar los requerimientos fue mediante la creación de prototipos o *mockups* de baja fidelidad.

Se trata de una manera sencilla y barata de probar las funcionalidades e interfaz gráfica de nuestra aplicación sin necesidad de desarrollar ningún módulo de la misma [34]. En nuestro caso utilizamos dos herramientas *web* para generarlos:

- **Marvel:** Esta herramienta nos permite realizar prototipos de una manera rápida y sencilla, brindándonos la posibilidad de definirle comportamiento a los mismos.
- **Balsamiq Mockups:** Esta herramienta nos permite diseñar prototipos de manera sencilla con una interfaz gráfica mas “realista” y similar a la de una aplicación comercial.

Esto nos permitió evaluar y validar los requerimientos y las interfaces propuestas tanto con el cliente como con distintos usuarios de una manera rápida y sencilla.

Un ejemplo de estos prototipos se ilustra a continuación.

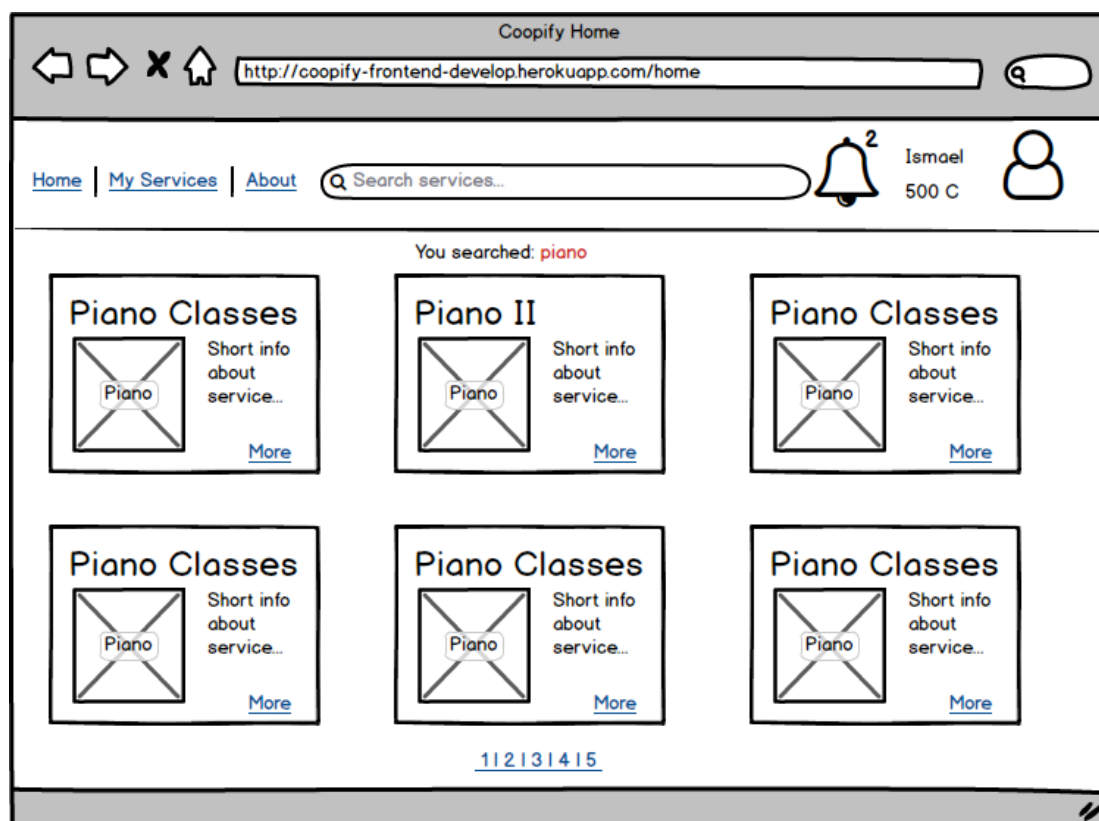


Figura 5.1: Prototipo de la pantalla principal de *Coopify*

En el anexo 13.1.4 se pueden observar otros prototipos diseñados durante esta etapa. Estos prototipos fueron validados con el cliente, que además dieron pie a mejoras o profundizaciones en varios de los requerimientos inicialmente requeridos.

5.3 Especificación

Lo que describiremos a continuación, no es lo que se conoce como un *ESRE*². Sino que al ser un proyecto con una metodología ágil, son a grandes rasgos, funcionalidades que el cliente definió, algunas de las cuales fueron épicas y otras se tradujeron directamente a *user stories*.

²*ESRE*: Especificación de requerimientos

5.3.1 Requerimientos funcionales

RF1 Registrar usuario

Cualquier persona deberá de poder registrarse en la aplicación utilizando una combinación de *email* y contraseña o permitiéndole utilizar su cuenta de los proveedores de identidad *Facebook* o *Google*.

RF2 Inicio de sesión

Prerrequisito: usuario registrado previamente.

Un usuario que se haya registrado en el sistema deberá de poder iniciar su sesión proveyendo sus credenciales o utilizando alguno de los proveedores de identidad *Facebook* o *Google*.

RF3 Perfil de usuario

Prerrequisito: usuario registrado previamente y que ha iniciado sesión.

Un usuario deberá de poder editar su información en cualquier momento.

RF4 Obtener listado de servicios

Un usuario deberá de poder obtener el listado de ofertas creadas hasta el momento.

RF5 Obtener detalle de un servicio

Dado una oferta registrada en el sistema un usuario deberá de poder obtener el detalle de la misma.

RF6 Crear servicio

Prerrequisito: usuario registrado previamente y que ha iniciado sesión.

Un usuario deberá de poder registrar una oferta, para ello deberá de indicar que medios de pago acepta (*coopies* y/o trueque) y que instancias de intercambio le interesan (Hora, Sesión y/o Producto final).

RF7 Filtrar servicios

Prerrequisito: usuario registrado previamente y que ha iniciado sesión.

El usuario deberá de poder filtrar las ofertas disponibles mediante diferentes criterios (instancias de intercambio, precio, nombre y/o medios de pago).

RF8 Contactar y negociar

Prerrequisito: usuario registrado previamente, que ha iniciado sesión y que ha encontrado una oferta que le interesa.

El usuario podrá ponerse en contacto con el proveedor de la oferta de su interés. Luego de establecido el mismo podrá entrar en una etapa de intercambio de mensajes e información con el fin de negociar con la contra parte. Todo esto deberá realizarse a través de un *chat* integrado en la aplicación. Como parte de la negociación se podrá realizar ofrecimientos de intercambio entre partes, dichos ofrecimientos pueden ser abonados intercambiando servicios directamente o mediante un pago con *coopies*.

RF9 Pagos con coopies

Prerrequisito: usuario registrado previamente, que ha iniciado sesión, ha recibido una oferta que involucra intercambio de *coopies*.

Llegada esta situación el usuario al que se le ha ofrecido el intercambio podrá aceptar o rechazar la oferta. En caso de aceptarla se deberá de generar una transferencia de *coopies* entre cuentas por el monto especificado en la oferta aceptada. Esta transacción se deberá de ver reflejada en el saldo y listado de transacciones.

RF10 Pagos grupales

Prerrequisito: usuario registrado previamente y que ha iniciado sesión

Un conjunto de usuarios deberían de poder contribuir en la adquisición de un servicio a compartir entre ellos. Para esto uno de los usuarios generará una propuesta, que una vez acordada cualquier usuario del sistema podrá contribuir con ella. Una vez alcanzado el monto acordado no se podrá seguir contribuyendo y el servicio será brindado.

RF11 Transacciones respaldadas con Ethereum

Todo movimiento de *coopies* deberá de estar respaldada por una transacción dentro de la plataforma de *blockchain Ethereum*. Esto aplica a pagos (individuales y grupales) y recompensas.

RF12 Dejar una reseña

Prerrequisito: usuario registrado previamente, que ha iniciado sesión y ha intercambiado un servicio con éxito.

Luego de que los usuarios intercambien servicios el solicitante deberá de poder realizar una reseña evaluando la calidad del servicio brindado y la calidad de su contra parte en el intercambio. La idea detrás de esto es que la comunidad se pueda auto regular y que los servicios de calidad puedan diferenciarse y para defender a los usuarios.

RF13 Ver reseñas

Prerrequisito: usuario registrado previamente y que ha iniciado sesión

Un usuario debe de poder informarse de experiencias previas de otros usuarios que han consumido este servicio, para ello debe de poder ver las reseñas sobre usuarios y servicios.

RF14 Solicitar intercambio abierto

Prerrequisito: usuario registrado previamente, que ha iniciado sesión y que ha encontrado un servicio de su interés.

Un usuario del sistema debe de poder realizar ofertas ignorando las restricciones del servicio ofrecido con el fin de permitir al solicitante un mayor abanico de posibilidades a la hora de negociar entre las partes.

RF15 Recompensas al usuario

Prerrequisito: usuario registrado previamente y que ha iniciado sesión

Un usuario debe de ser recompensado por realizar acciones que enriquezcan a la plataforma. Por acciones que enriquezcan a la plataforma entendemos las siguientes interacciones: - Referir a un nuevo usuario a la plataforma - Compartir un servicio en la red social *Facebook* - Registrarse en la plataforma Las recompensas siempre serán un numero predefinido de *coopies* que serán agregados a la cuenta del usuario beneficiado.

RF16 Preguntas generales

Prerrequisito: usuario registrado previamente, que ha iniciado sesión y encontró un servicio de su interés

Un usuario debe de ser capaz de realizar una pregunta concisa respecto a un servicio.

RF17 Responder preguntas generales

Prerrequisito: usuario registrado previamente, que ha iniciado sesión, ha publicado un servicio con una pregunta sin responder

El sistema debe de permitirle a un usuario debe de ser capaz de contestar las preguntas realizadas sobre los servicios que cree.

RF18 Cerrar sesión

Prerrequisito: usuario registrado previamente y que ha iniciado sesión

El sistema permite cerrar sesión una vez realizadas las tareas pertinentes para mantener segura y privada la cuenta personal.

RF19 Compartir servicio en Facebook

Prerrequisito: usuario registrado previamente, que ha iniciado sesión y selecciona un servicio

El sistema debe permitir compartir el servicio en la red social *Facebook*, en caso tratarse de un servicio propio que no ha sido compartido previamente debe de disparar la recompensa pertinente.

5.3.2 Requerimientos no funcionales

RNF1 Usabilidad

Un usuario debe de encontrar la aplicación entendible e intuitiva.

Por esto entendemos que la aplicación debe ser absolutamente sencilla de utilizar para el usuario, con una de curva de aprendizaje mínima para poder utilizar todas sus funcionalidades.

Para garantizar esto último, toda función compleja no debe de llevarle al usuario más de siete *clicks* o tres pantallas.

Se tratan de números experimentales que el equipo definió y acordó en conjunto con el cliente en base a investigaciones sobre herramientas similares y pruebas de validación con usuarios, en donde concluimos que el patrón se mantiene entorno a dicha media.

RNF2 Mantenibilidad

Al cliente le interesa que el resultado sea mantenible y fácil de comprender.

La mantenibilidad se define como el impacto que tiene introducir un nuevo cambio sobre un código ya existente, a mayor impacto, menos mantenible es el sistema.

Por ende, para cumplir con este objetivo intentaremos respetar, en la medida de lo posible, los principios *SOLID* [106].

Por otro lado, para facilitar la comprensión por terceros, utilizaremos *Linter* [18] para aplicar las mejores prácticas al formato del código desarrollado.

Dicha herramienta aplicará las reglas impuestas por *Airbnb* dado que las mismas se encuentran estandarizadas, son muy populares y utilizadas por muchas empresas alrededor del mundo [50].

RNF3 Seguridad de transacciones

Las transacciones realizadas por los usuarios de la aplicación deberán de mantenerse de forma segura en alguna plataforma de *blockchain*, por lo que la misma sera garantizada a través del respaldo brindado por dicha tecnología, por el manejo de encriptado de claves y sesiones a nivel del sistema.

RNF4 Performance

La aplicación deberá de responder de manera rápida y eficaz.

La métrica utilizada para medir la rapidez es el tiempo en segundos que demora el sistema en realizar una tarea y permitir al usuario retomar el control sobre la misma.

Para esto utilizamos distintos valores dependiendo del tipo de operación realizado en el sistema. Dada la naturaleza de nuestro sistema identificamos dos tipos de operaciones:

- Operaciones normales del sistema (creación de perfil de usuario, publicación de un servicio, etc.)
- Operaciones del sistema que involucran de manera directa o indirecta consultas o transacciones sobre la red de *Ethereum* (consulta de saldo, pago de un servicio, etc.)

Para aquellas operaciones normales del sistema definimos un rango de entre uno a tres segundos máximo como aceptable para realizar la misma.

En cambio, aquellas operaciones que involucran la red de *Ethereum* manejamos un rango de hasta un minuto como máximo debido la complejidad de la misma. Cabe destacar que si bien la tarea puede llegar a demorar hasta un minuto, la misma se realiza de forma asincrónica por lo que el usuario tendrá total control del sistema fomentando así la usabilidad del mismo.

RNF5 Stack tecnológico

Inicialmente la aplicación deberá de ser desarrollada ciñéndose a algunas de las tecnologías expuestas a continuación:

- *Backend: Ruby on rails o NodeJS*
- *Frontend web: React, Angular o VueJS*
- *Mobile: ReactNative, Ionic, Flutter o NativeScript*
- *Blockchain: Ethereum o HypreledgerFabric*

Luego de conversar con el cliente se le planteo la posibilidad de desarrollar una PWA para unificar el *frontend web* y *mobile*. Luego de pensarlo el cliente decidió aceptar este ofrecimiento por lo que el *stack* tecnológico sería el siguiente:

- *Backend: Ruby on rails o NodeJS*
- *Frontend web: React, Angular o VueJS*
- *Blockchain: Ethereum o HypreledgerFabric*

5.4 Conclusiones de ingeniería de requerimientos

Como se dijo al principio del capítulo, esta etapa es fundamental en el proceso de desarrollo de *software*. En base a los requerimientos relevados y validados en la misma obtuvimos como resultado el *Product Backlog*, que nos sirvió de entrada³ para la etapa de desarrollo del proyecto.

La etapa inicial de relevamiento fue muy positiva ya que pudimos plasmar las funcionalidades planteadas inicialmente por el cliente. De dicha actividad surgió una contrapropuesta por parte del equipo conformada por algunos cambios sobre las funcionalidades propuestas inicialmente, junto con algunas adiciones. En consecuencia nos permitió definir requerimientos mas concretos y concisos.

Por otro lado, las técnicas de validación nos sirvieron para corroborar los reque-

³Las entradas de un proceso responden a criterios de aceptación definidos [12]

rimientos obtenidos anteriormente. Esto nos brindó una confianza inicial de que el sistema colmara las expectativas de los usuarios finales, y a su vez asegurarnos cierto grado de usabilidad debido a la definición y prototipación de un conjunto de pantallas de la aplicación.

5.5 Lecciones aprendidas

Como lecciones aprendidas del proceso de ingeniería de requerimientos podemos destacar tres grandes actividades realizadas que aportaron gran valor durante el transcurso del proyecto, y que muy probablemente puedan implementarse de igual manera para posteriores proyectos.

Las validaciones con el cliente fueron una parte fundamental en esta etapa ya que pudimos relevar y a su vez aportar nuevas funcionalidades al sistema. Las reuniones mantenidas con el cliente fueron fructíferas, aunque consideramos que para encarar futuros proyectos sería provechoso contar con más instancias de este estilo.

El proceso de ingeniería inversa resultó muy útil para poder explorar otras aplicaciones ya existentes en el mercado permitiéndonos no solo relevar nuevas funcionalidades sino también validarlas.

Si bien pudimos haber realizado una mayor cantidad de reuniones de relevamiento con el cliente, lo pudimos solventar a través de distintas instancias puntuales al finalizar cada *sprint*. Donde confirmamos el flujo esperado para los requerimientos próximos a implementar, esto mitigó la posibilidad de cambios en los mismos.

Por último, destacamos el uso de prototipos de baja fidelidad como método para validar los requerimientos, dándonos la posibilidad de encarar las funcionalidades desde otro punto de vista. Esto nos hizo rever las ya planteadas, resultando en la modificación de las mismas, o incluso agregar nuevas.

6 Arquitectura

Una vez definidos los requerimientos funcionales y no funcionales se procedió a definir la arquitectura del producto.

6.1 Descripción de la arquitectura

La arquitectura utilizada consta de varias capas, las cuales pasaremos a detallar.

- Una capa de presentación provista por una aplicación web progresiva (*PWA*¹).
- Un servidor que encapsula y se encarga de aplicar la lógica de negocio (*Backend*)
- Un servidor que encapsula la interacción con el servicio de *blockchain* seleccionado (*Ethereum*)

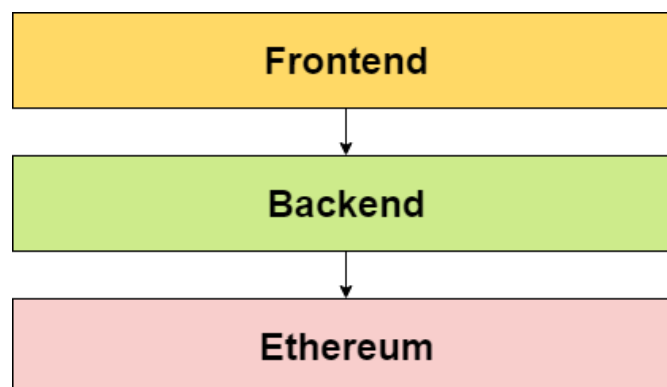


Figura 6.1: Arquitectura en capas

¹Ver referencia bibliográfica [22]

6.1.1 Arquitectura en capas

Creímos conveniente implementar la lógica referente a la comunicación con la red *blockchain*, en otro servidor. Esto más que nada, por los tiempos de respuesta, que no se ajustan a los esperados por el usuario. Eso sucede pues *blockchain* necesita tiempo para que la transacción sea guardada en un bloque, este se mine y se guarde en los nodos de la red. Además brinda la posibilidad de cambiar en un futuro las implementaciones referentes a *blockchain*, sin tener que cambiar nada en *backend*. Siempre y cuando no se modifique la interfaz que hace posible la comunicación entre los mismos.

6.2 Vistas de arquitectura

A continuación se detalla como se encuentra construido el sistema desde el punto de vista estructural y funcional. Para eso nos apoyaremos en diagramas de secuencia para ejemplificar el comportamiento de ciertos flujos principales y diagramas de clase para mostrar la estructura de clases lograda. En consecuencia examinamos determinadas vistas que integran el modelo de arquitectura 4+1.

“4+1 es un modelo diseñado por Philippe Kruchten para (describir la arquitectura de sistemas software, basados en el uso de múltiples vistas concurrentes). Las vistas suelen describir el sistema desde el punto de vista de diferentes interesados, tales como usuarios finales, desarrolladores o directores de proyecto.”[97].

Vamos a basarnos en la: vista de despliegue (o también llamada vista física), vista de implementación (o también llamada vista de desarrollo), vista lógica y vista de escenarios.

Descartamos la vista de procesos, por no brindarnos nada extra, ya que no manejamos hilos diferentes sino que *NodeJS* utiliza un solo hilo de ejecución, siendo muy eficiente con la concurrencia, para mas detalles, ver sección 6.3.1.

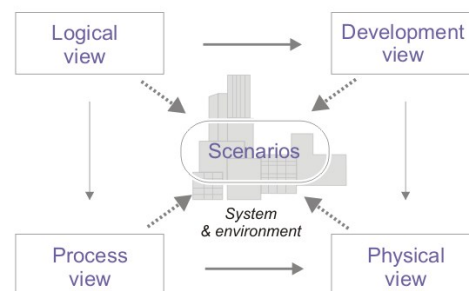


Figura 6.2: Modelo 4+1 [97]

6.2.1 Vista de despliegue

En la figura 6.3, vemos la manera en que están desplegadas las aplicaciones en cada servidor.

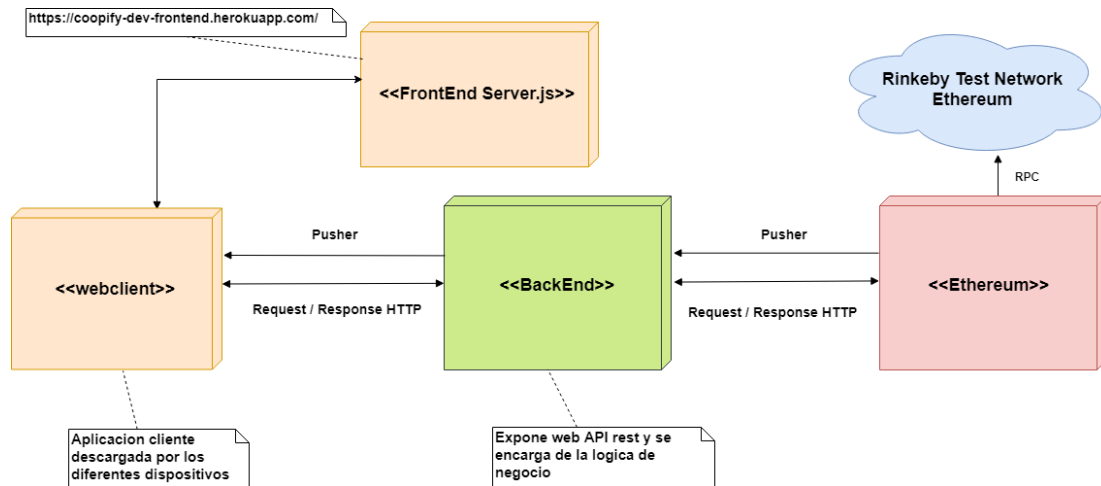


Figura 6.3: Diagrama de despliegue

El servidor de *Frontend* contiene la aplicación que se descarga al acceder a la dirección `https://coopify-dev-frontend-herokuapp.com`, esa aplicación se ejecuta en el dispositivo y es la que llamamos *webclient*.

La aplicación *webclient* se comunica con el servidor *backend* mediante *request* y *response* https, para resolver las solicitudes necesarias.

A su vez el servidor de *backend* se comunica con el servidor de *blockchain* (en este caso implementado con *Ethereum*), mediante *request* y *response* https, para resolver las solicitudes necesarias.

El servidor *Ethereum* es el encargado de efectuar las transacciones con la red de *blockchain*, utilizando *Rinkeby Test Network de Ethereum*.

6.2.2 Vista de implementación

En esta vista intentamos ilustrar el sistema de la perspectiva del programador. Está enfocada en la administración de los artefactos de software, comúnmente llamados componentes.

El lenguaje de programación utilizado fue *TypeScript* sobre *NodeJS*, por lo tanto el código desarrollado es transpilado ² a *JavaScript*. El resultado de este proceso dista de la definición de un componente tradicional, ya que no genera un único producto a partir de varias clases o archivos (como sucede en otros lenguajes como *.NET*), sino que genera uno por cada archivo de *typescript* construido.

Como aclaración tomamos como un componente a una carpeta dentro de la estructura de archivos porque de forma similar a los *namespaces* en *.NET*, las carpetas nos permiten agrupar archivos con finalidad similar. En la figura 6.4 se pueden ver los componentes de *backend*, así como sus dependencias.

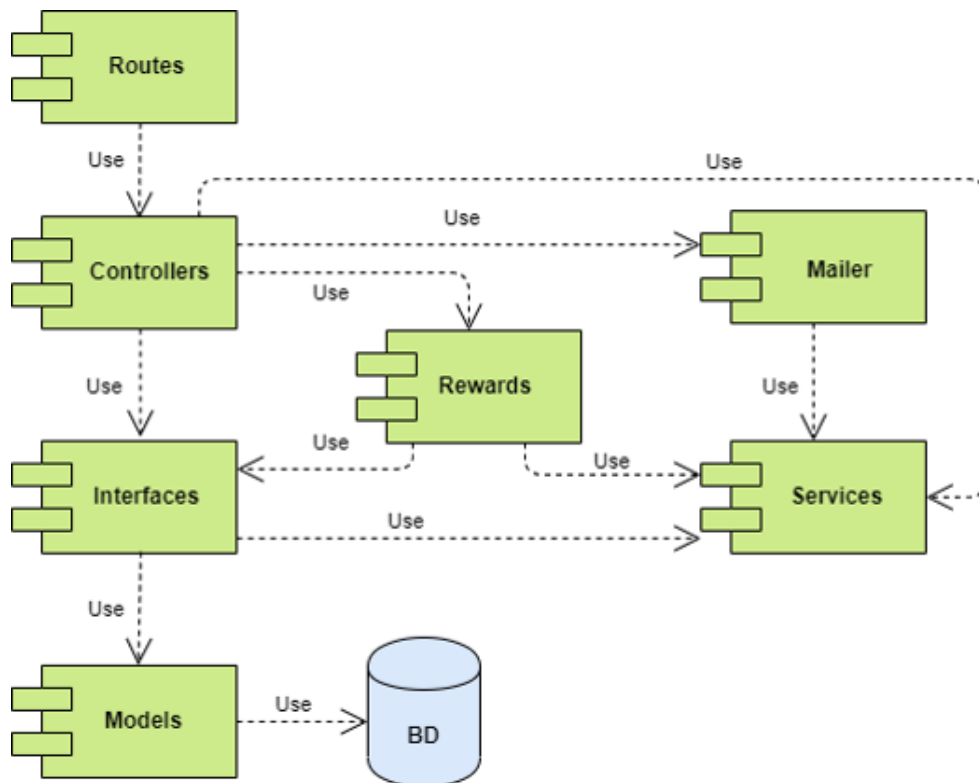


Figura 6.4: Diagrama de componentes que integran Backend

²Transpilar: traducir código entre dos lenguajes que están al mismo nivel de abstracción. Ver referencia bibliográfica [11]

6.2.3 Vista lógica

Backend: A continuación (figura 6.5) mostraremos las clases mas importantes que componen *backend* y la comunicación entre los componentes internos del mismo. Vale la pena aclarar que en el diagrama no se muestran relaciones internas entre los archivos y clases de un paquete o carpeta. Esto para simplificar el entendimiento de como se comunican entre capas de alto nivel hacia bajo nivel y que cada capa tiene su representante a más bajo nivel.

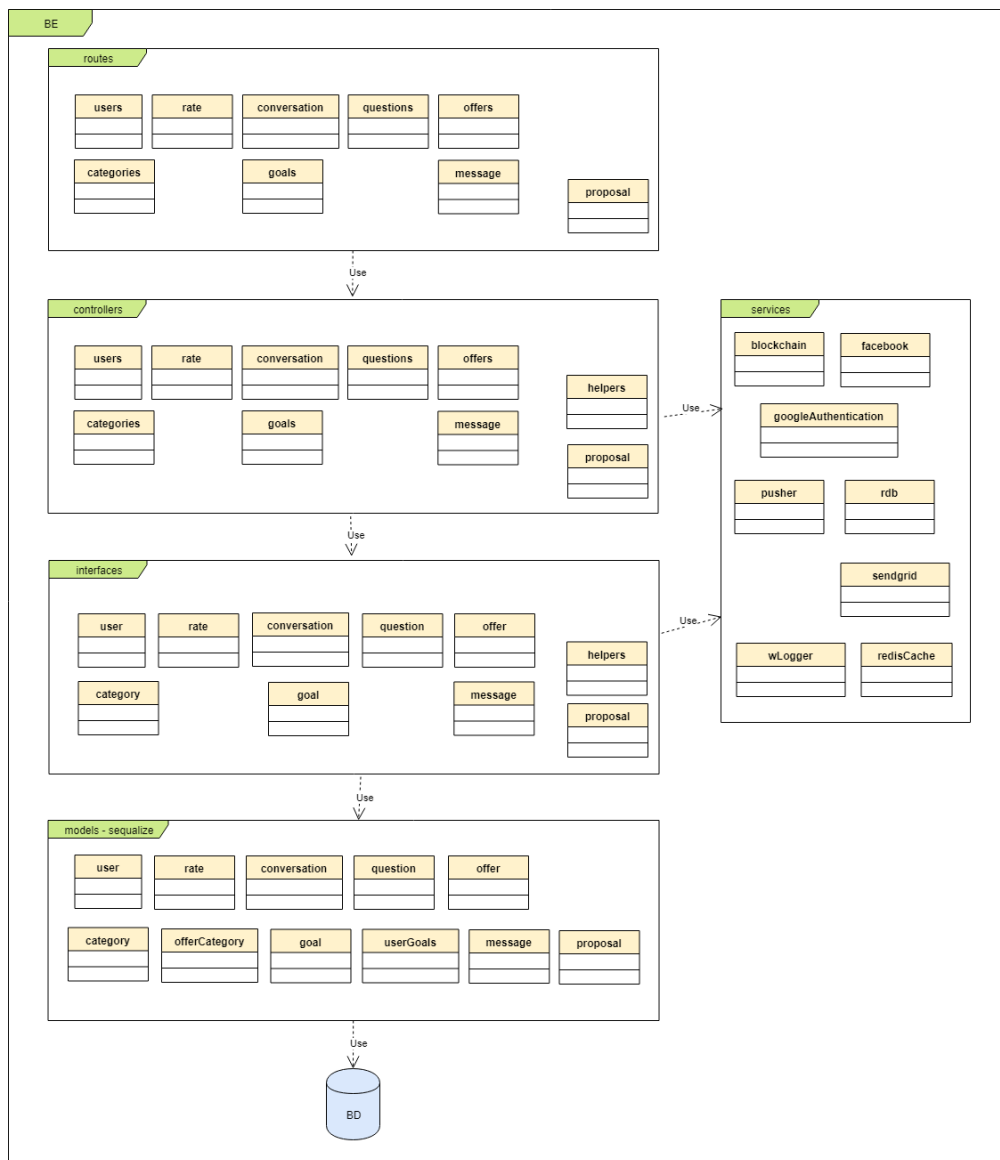


Figura 6.5: Diagrama de las clases existentes en Backend

Offers en Backend: Para ir un poco más en detalle de como se componen y comunican las clases que componen el dominio del sistema, detallaremos las que corresponden a las *Offers* o servicios, ver figura 6.6

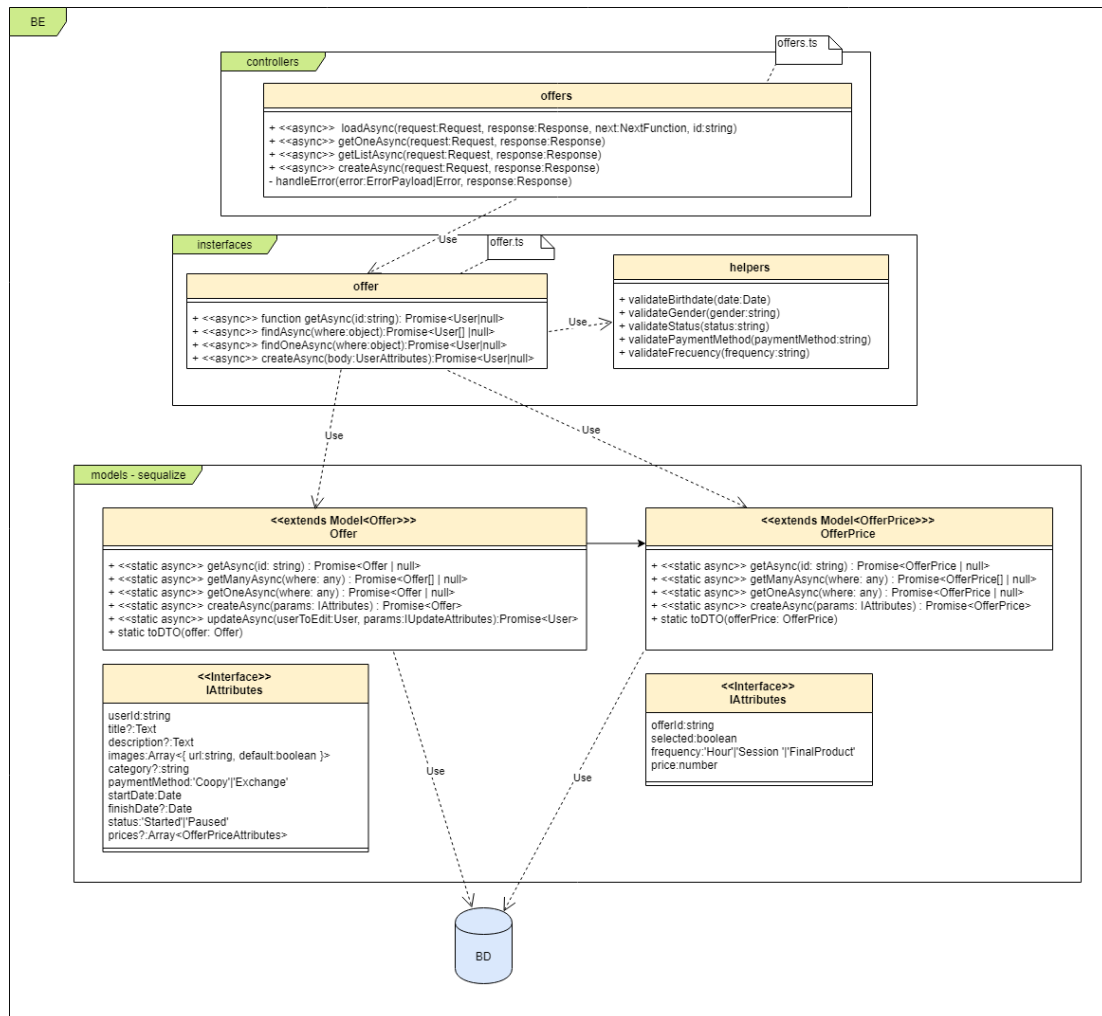


Figura 6.6: Diagrama de clases offers en Backend

Es importante destacar que en realidad la tecnología de *Backend* que estamos utilizando, no genera un único binario como pasa en *.Net* o *Java*.

Ethereum: A continuación (figura 6.7) mostraremos las clases mas importantes que componen *Ethereum* y la comunicación entre los componentes internos del mismo.

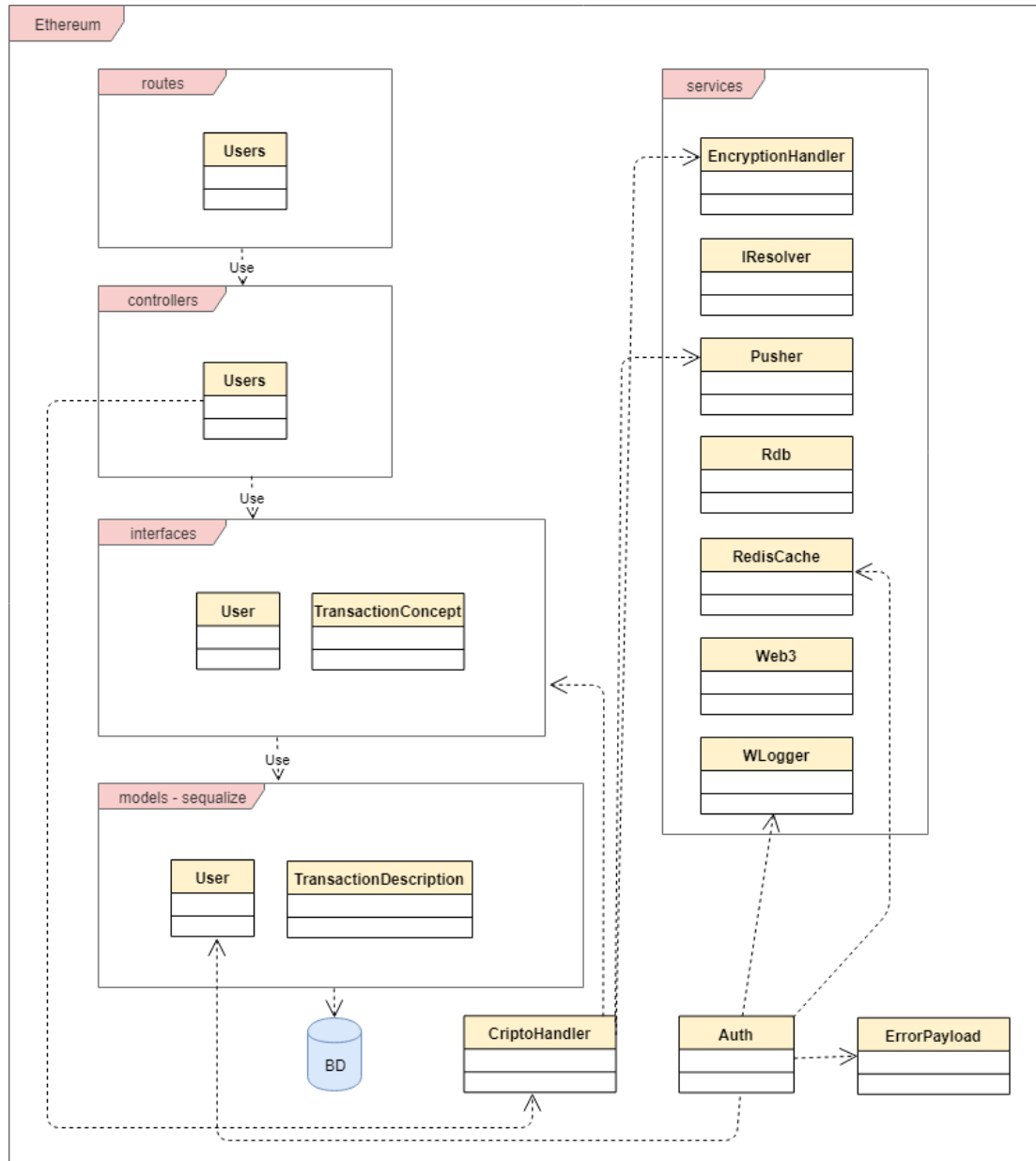


Figura 6.7: Diagrama de clases existentes en Ethereum

Users en Ethereum: Uno de los puntos principales del servidor *Ethereum*, es llevar el control y registro de las *address* y *Private Key* de los usuarios, para poder efectuar las transacciones en la *blockchain*. Si bien en *Backend* están registrados los usuarios en la base de datos, en *Ethereum* también existen, pero con datos esenciales y referentes a la *Blockchain*. Detallaremos como se componen y comunican las clases que corresponden a *users* en *Ethereum*. Ver figura 6.8.

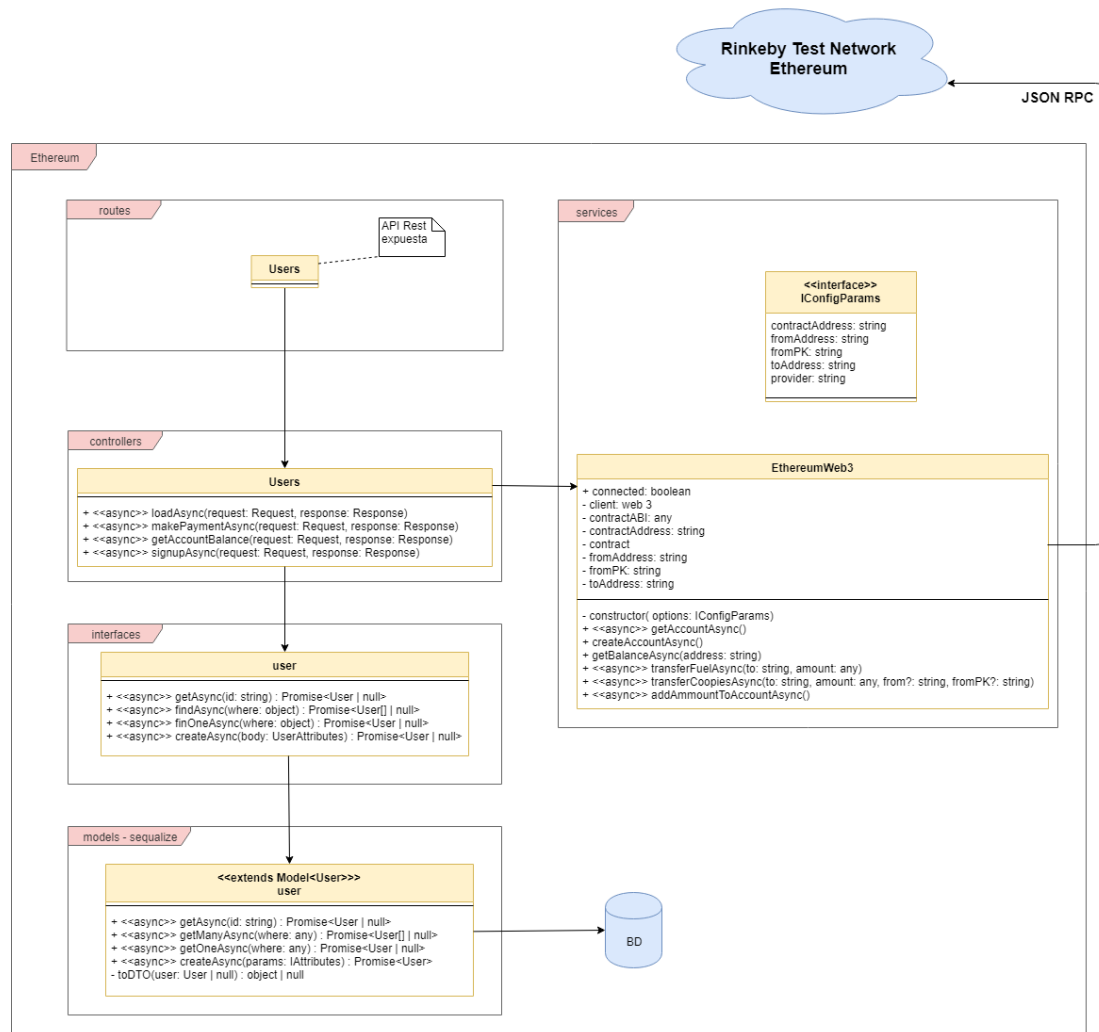


Figura 6.8: Diagrama de clases users en Ethereum

Vale la pena aclarar que tener acceso a las transacciones respaldadas con *Ethereum* no quiere decir que este puedan modificarse, ya que las mismas quedan respaldadas en la *blockchain* de forma descentralizada.

Al utilizar un servicio especializado para obtener el saldo y transacciones realizadas por un usuario nos aseguramos de obtener esta información íntegra. Incluso si alguien decidiera tratar de movilizar los saldos de la cuenta de un usuario a otra, contamos con trazabilidad de cada uno de estos movimientos y a donde han sido dirigidos estos fondos.

Diagramas de secuencia: Dentro de los diagramas correspondientes a la vista lógica, tenemos los diagramas de secuencia. Agregamos en forma de anexo, los más importantes del proyecto, que son los que intentan mostrar el funcionamiento de las peticiones referentes al uso de *blockchain*.

- Diagrama de secuencia para obtener el balance de *Coopies* de un usuario. Ver figura en anexo **13.15**
- Diagrama de secuencia para efectuar una transacción entre dos usuarios. Ver figura en anexo **13.16**
- Diagrama de secuencia para abastecer de *Ether* a la cuenta de un usuario. Ver figura en anexo **13.17**

Cuando se intenta efectuar una transacción, se hace de forma asíncrona, por los tiempos que insume la red *Blockchain*. Por lo tanto necesitamos de un mecanismo que nos permita notificar al usuario si la transacción se efectuó de manera correcta o si por algún motivo no se pudo hacer. Para esto utilizamos *Pusher*.

Básicamente lo utilizamos para el manejo de los mensajes en las conversaciones del *chat*³ y para el aviso al usuario sobre las transacciones como dijimos antes. Ver diagrama de secuencia **13.18**

³Desde el *chat* se puede negociar, según el requerimiento funcional 5.3.1

6.2.4 Vista de escenarios

En la figura 6.9, mostraremos los casos de uso del sistema.

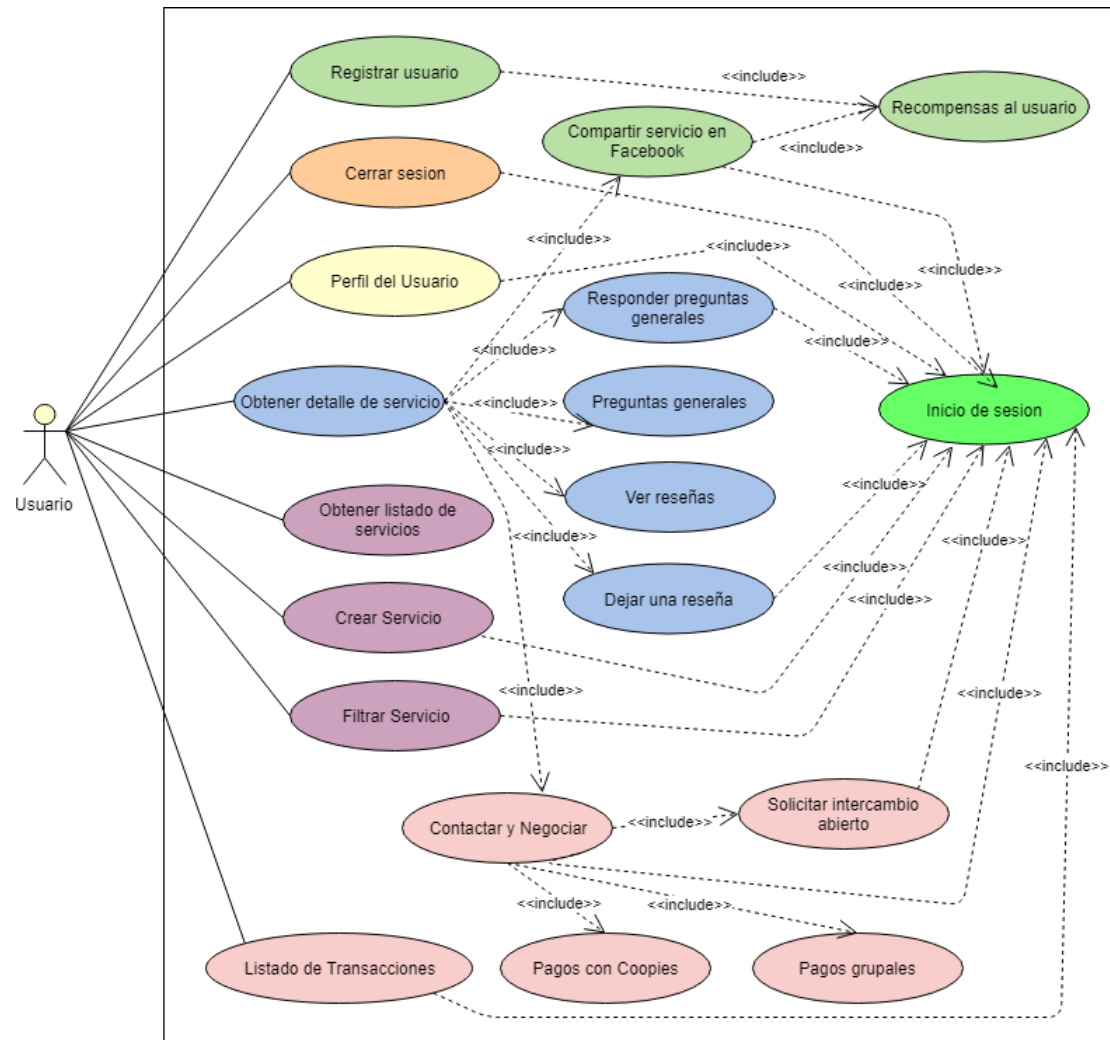


Figura 6.9: Diagrama de casos de uso

En el diagrama de casos de uso intentamos expresar como los usuarios tienen acceso sin iniciar sesión a la funcionalidad “*Obtener listado de servicios*”, y las que necesitan de inicio de sesión tienen incluida la funcionalidad “*Inicio de sesión*”.

6.3 Atributos de calidad

Para establecer la arquitectura que necesita el proyecto, tuvimos en cuenta distintos atributos de calidad que se establecieron en los requerimientos tanto funcionales (detallados en 5.3.1) como no funcionales (detallados en 5.3.2). En los siguientes puntos iremos detallando qué decisiones fuimos tomando para cumplir con esos atributos de calidad relevados.

6.3.1 Performance

Para adaptarnos al requerimiento no funcional detallado en 5.3.2, tomamos diferentes caminos favoreciendo ese atributo de calidad.

React

La aplicación de *Frontend* se encuentra en un servidor aparte, desde donde se descarga la aplicación *webClient*⁴ que se ejecuta en el dispositivo.

Existe un único tiempo de descarga y se da al principio al acceder por primera vez, donde se descarga toda la estructura de la aplicación. Luego al navegar por la misma se tiene todo en local, solo ejecutándose *request https* para comunicarse con *backend*.

Con el uso del DOM virtual, React hace las actualizaciones UI (Interfaz de Usuario) tan eficientes como nunca, haciendo al mismo tiempo que la aplicación web sea más precisa. [74]

Node.js

Tanto la aplicación de *backend* como la de *Ethereum*, se implementaron utilizando *Node.js*. Uno de los motivos principales es su alta performance. Nos referimos a la performance de *Node.js*, más que nada a nivel de el manejo de la concurrencia, ver [99].

⁴Versión comprimida del código que permite ejecutar la aplicación desde el dispositivo



Figura 6.10: Diagrama de concurrencia NodeJS [42]

Node.js es uno de los mejores en cuanto a este punto, importante a la hora de responder rápidamente a las peticiones recibidas desde *frontend*. Esto es vital, porque *Backend* es el encargado de la mayor parte de la lógica del sistema y *Ethereum* de la lógica de las comunicaciones con la *blockchain*. Digamos, son los que marcan los tiempos de respuesta de todo el sistema.

Otro de los motivos para la utilización, tanto de *ReactJS* como *NodeJS*, fue de mantener el mismo stack tecnológico de *JavaScript*.

PostgreSQL

El sistema tiene dos bases de datos, una en el servidor donde esta *backend* y otra en el servidor *Ethereum*. El motor de base de datos que utilizamos es *PostgreSQL* por ser un gestor de bases de datos multiplataforma, potente y robusto. Es relativamente lento en inserciones y actualizaciones ⁵ en bases de datos pequeñas, pero está diseñado para ambientes de alto volumen. Esto hace que la velocidad de

⁵Ver referencia bibliográfica [62]

respuesta pueda parecer lenta en comparación con bases de datos de pequeño tamaño. Si bien este detalle podría parecer un detrimento en cuanto a performance, a la larga cuando la plataforma crezca en cuanto a volumen de datos, va a ser algo favorable.

Pusher

Existen muchas comunicaciones entre las aplicaciones que componen la plataforma, que no se hacen de manera sincrónica y que podrían demorar bastante en relación a lo que el usuario podría esperar sin ver cambios y pensar que algo malo paso.

Se necesita de comunicaciones que van en sentido opuesto al tradicional, iniciando comunicaciones desde el servidor hacia otro componente. Por ejemplo al solicitar una transacción desde *backend* a *Ethereum*, esta ultima efectúa la transacción a en la red *blockchain* y cuando realmente esa transacción quedo efectiva en la red, *Ethereum* se comunica con *backend* mediante *Channels Pusher* (Ver figura 13.18). Del lado de *backend* se esta suscrito a ese *channel* y cuando se recibe el *push* que se espera, se despliega una notificación al usuario, ya sea un *mail* o un aviso en pantalla de que la transacción se efectuó con éxito o si tuvo algún problema.

Otro lugar en que utilizamos *Pusher* es en las conversaciones (*chat*) que posee la plataforma, para la comunicación entre usuarios. De esta forma los mensajes enviados de un usuario a otro puede mostrarse en tiempo real entre las dos ventanas de *chat*.

Redis

Utilizamos un servidor *Redis* como almacén de estructura de datos en memoria con el fin de manejar los tokens generados⁶ para las sesiones de usuario.

Una de las ventajas mas importantes que tiene es la de una velocidad muy por encima de la media respecto a otras bases de datos de su tipo gracias al almacenamiento en memoria y su alta disponibilidad.

⁶Ver referencia bibliográfica [5]

Heroku

Alojar de las aplicaciones en *Heroku*⁷ ayuda mucho en cuando a *performance* por el uso de los *Dynos*⁸. Cuando se utiliza la aplicación, no se ejecuta un solo *Dyno* para esa aplicación, sino que se ejecuta mas de una instancia a la vez. Estas actúan como contingencia en casos puntuales y hacen que el funcionamiento de la aplicación en general sea fluido. Otro punto importante es que sus *Dynos* tienen la capacidad de recuperarse de errores.

Otro atributo importante que favorece el uso de los *Heroku*, es referente a la escalabilidad. Según el uso, se puede optar por distintos planes que posibiliten desplegar mas *Dynos* para esa aplicación.

Resultados obtenidos en cuanto a performance

Para verificar que nuestro producto de software cumple con los objetivos definidos decidimos medir la velocidad de respuesta a operaciones comunes y operaciones que involucran *blockchain*. Para la primera utilizamos la consola integrada del navegador con el objetivo de medir el tiempo de respuesta de un subconjunto de solicitudes *https* realizadas a nuestro servidor.

⁷Ver referencia bibliografica, [89]

⁸Son componentes Linux aislados y virtualizados, diseñados para ejecutar código basado en un comando especificado por el usuario, (Ver referencia [85])

Name	Status	Type	Initiator	Size	Time	Waterfall
5a8f1100-72c5-11e9-...	200	fetch	workbox-cor...	1.1 KB	231 ms	
?reviewedUserId=5a8f...	200	fetch	workbox-cor...	7.0 KB	193 ms	
balance	200	xhr	VM168:1	(Servi...	294 ms	
offers?limit=18&skip=0	200	xhr	VM168:1	(Servi...	423 ms	
balance	200	fetch	workbox-cor...	297 B	280 ms	
offers?limit=18&skip=0	200	fetch	workbox-cor...	299 B	218 ms	
balance	200	xhr	VM168:1	(Servi...	512 ms	
a1aed4b0-b16a-11e9-9d8...	200	xhr	VM168:1	(Servi...	397 ms	
a1aed4b0-b16a-11e9-9d8...	200	xhr	VM168:1	(Servi...	383 ms	
?offerId=a1aed4b0-b16a...	200	xhr	VM168:1	(Servi...	513 ms	
reviewable	200	xhr	VM168:1	(Servi...	575 ms	
balance	200	fetch	workbox-cor...	297 B	336 ms	
a1aed4b0-b16a-11e9-...	200	fetch	workbox-cor...	297 B	223 ms	
?offerId=a1aed4b0-b1...	200	fetch	workbox-cor...	296 B	335 ms	
a1aed4b0-b16a-11e9-...	200	fetch	workbox-cor...	732 B	208 ms	
reviewable	200	fetch	workbox-cor...	297 B	243 ms	
balance	200	xhr	VM168:1	(Servi...	474 ms	
proposals/	200	xhr	VM168:1	(Servi...	460 ms	
proposals/	200	fetch	workbox-cor...	13.7 KB	291 ms	
balance	200	fetch	workbox-cor...	297 B	303 ms	
balance	200	xhr	VM168:1	(Servi...	560 ms	
balance	200	xhr	VM168:1	(Servi...	1.05 s	
transactions	200	xhr	VM168:1	(Servi...	586 ms	
balance	200	fetch	workbox-cor...	297 B	344 ms	
balance	200	fetch	workbox-cor...	297 B	834 ms	
transactions	200	fetch	workbox-cor...	1.7 KB	372 ms	

Figura 6.11: Tiempos de respuesta a operaciones comunes

Como se puede apreciar en la columna Time todas las operaciones responden en un entorno menor a un segundo. Por lo que estamos cumpliendo con el objetivo que nos habíamos planteado.

Entendemos que esta prueba es realizada en ambientes ideales, en caso de contar con una mayor cantidad de datos estos tiempos pueden aumentar. Para ese caso se podrían agregar índices en la base de datos para optimizar búsquedas, aumentar la cantidad y/o calidad de los *dynos* para incrementar el poder de procesamiento.

Para medir el tiempo de respuesta de las operaciones que involucran la red de *blockchain* utilizamos la consola integrada en el servidor de *Heroku*, la cual al igual que la anterior, nos permite medir el tiempo de respuesta de cada solicitud *https* al servidor de *Ethereum*. Para analizar la rapidez de las respuestas se deben comparar las marcas de tiempo (*timestamp*) de los siguientes eventos:

- Al llegar la solicitud de un pago al servidor de *Ethereum*.
- Al mismo obtener la respuesta por parte de dicha red.

- Principio de inversión de la dependencia (*Dependency inversion principle*)
 - Depender de abstracciones, no depender de implementaciones.

Ethereum

La aplicación *Ethereum* fue desarrollada tomando en cuenta el **Principio de Inversión de Dependencias** “Depender de abstracciones y no de implementaciones”.

Ethereum implementa una interface *IResolver*, mediante la clase concreta *EthResolver*. Los métodos de la interface *IResolver* cubren las necesidades para poder efectuar las transacciones con la red *blockchain*.

La implementación *EthResolver* se comunica con la red *blockchain* de *Ethereum* mediante una clase *webClient* que a su vez utiliza una librería llamada *web3js*.

Lo dicho anteriormente, permite que el sistema sea modificable y mantenible, dando la posibilidad a cambiar de implementación de forma fácil y cercando a que las modificaciones se tengan que hacer con una guía bien marcada de lo que se necesita para el sistema. Esta guía la brinda la interface *IResolver*.

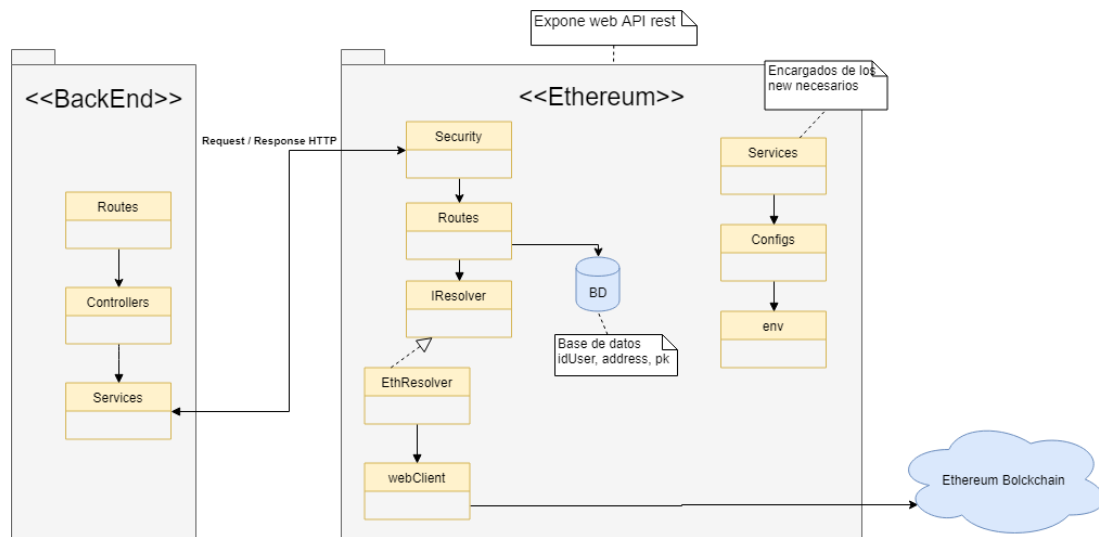


Figura 6.13: Diagrama Ethereum

Progressive Web App (PWA)

Uno de los motivos importantes del porque utilizar *PWA*, es justamente favorecer la mantenibilidad, ya que no tenemos que desarrollar aplicaciones diferentes para *desktop*, *iOS* y *Android*. Sino que la misma aplicación desarrollada con *PWA*, se puede utilizar en esos dispositivos. Enmascarando el navegador e incluso pudiendo acceder a los elementos nativos del dispositivo en cuestión.

Uso de patrón de diseño para implementación de las Rewards

Para la implementación de las recompensas, a usuarios, requerimiento funcional detallado en 5.3.1, tomamos la decisión de utilizar un patrón de diseño que nos permitiera un crecimiento a futuro de forma sencilla, ver figura 6.14.

Se trata de una cadena de responsabilidades (patrón *Chain of Responsibility*)⁹ si el día de mañana se quisieran agregar mas recompensas, seria solo crear una nueva implementación de la interface *IReward* para agregarlo a la cadena. Con esto favorecemos el **Principio de Abierto/Cerrado** “Entidades abiertas a su extensión, pero cerradas a modificaciones”.

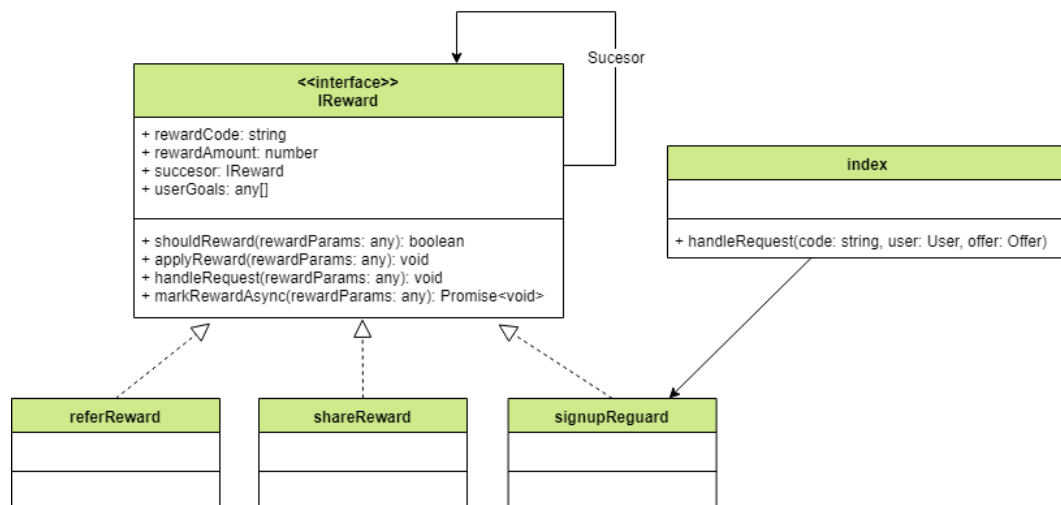


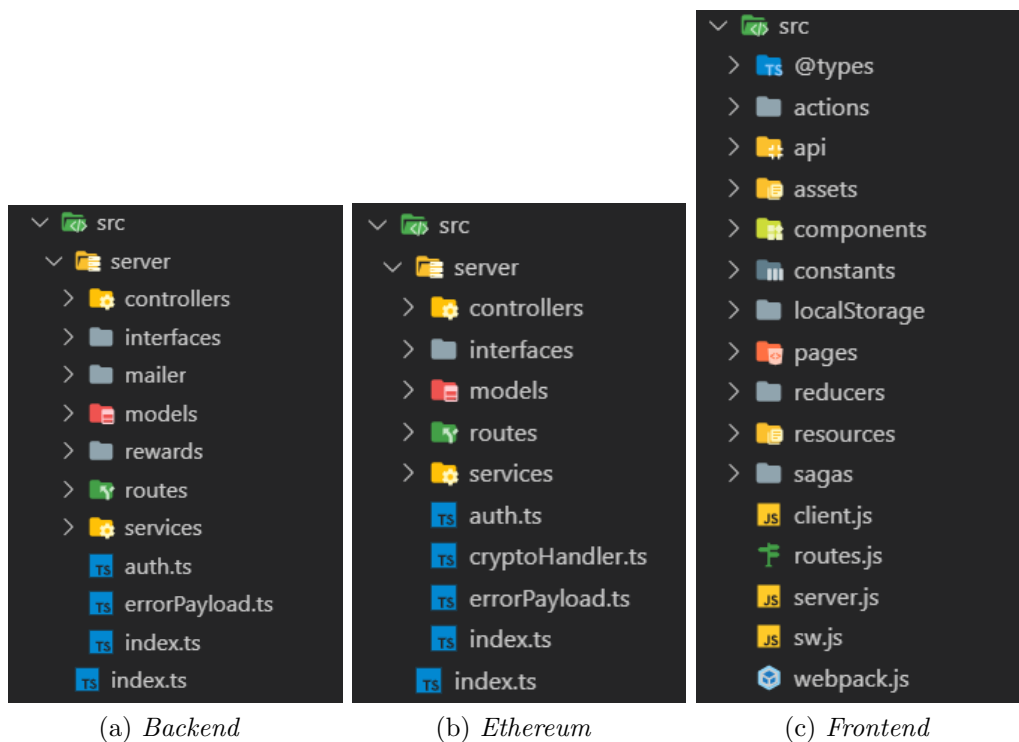
Figura 6.14: Diagrama patrón *Chain of responsibility*

⁹ *Chain of Responsibility*: “evita acoplar el emisor de una petición a su receptor dando a más de un objeto la posibilidad de responder a una petición. Para ello, se encadenan los receptores y pasa la petición a través de la cadena hasta que es procesada por algún objeto”. Ver [83]

Estructura de archivos del sistema

Para la separación de responsabilidades, la estructura de archivos es muy importante ya que justamente marca los límites de cada una de ellas. Y favoreciendo el **Principio de Responsabilidad Única**, “Objetos con una sola responsabilidad” Esto debido a que los archivos que componen las carpetas intentan cumplir el principio.

En la siguiente figura se muestran las carpetas que componen la carpeta *src*, de cada uno de los proyectos desarrollados.



El detalle de la estructura de archivos la colocamos en forma de anexo en la sección 13.2.1.

Como dijimos antes, tratamos de apegarnos a los principios *SOLID* y en forma de resumen, cumplimos con tres de los mismos, ya que hay dos que no aplican:

- **Principio de sustitución de *Liskov*:** al no tener herencia dentro del diseño de la solución, no aplica el principio.

- **Principio de segregación de interfaces:** Dada la cantidad de funcionalidades y por consiguiente de interfaces que creamos, no fue necesario tener varias específicas. De todas formas, creemos que en un futuro, cuando el sistema escape de lo que es un *MVP*, sea conveniente hacerlo.

6.3.3 Seguridad

Para adaptarnos al requerimiento no funcional detallado en 5.3.2, tomamos diferentes caminos para favorecer ese atributo de calidad.

A nivel de *backend* las claves de usuario se guardan cifradas, en una base de datos en *PostgreSQL*. Las claves que se guardan son las de los tipos de usuario que cuando se registraron lo hicieron de la manera tradicional, con usuario¹⁰. Para estos el sistema crea un *token* y se lo retorna al cliente y este token sirve de firma en cada *request* que este usuario haga desde *frontend* a *backend*, que necesiten la verificación.

En cambio para los registros de usuario mediante *google* o *facebook*, utilizamos *OAuth 2.0* conservando el token que retorna para el uso en cada *request*.

Ethereum

Uno de los aspectos mas importantes en cuanto a seguridad, esta enfocado básicamente en como guardar las *Private Key (PK)*¹¹ de cada usuario.

La *PK* se utiliza al momento de efectuar una transacción en la *blockchain*, ya sea para efectuar una transferencia de un usuario a otro, como así también para consultar el balance de *coopies* de un usuario. Para esto el sistema utiliza las *address* y *PK* de cada usuario cada vez que se le solicita efectuar una transacción, esto lo implementamos mediante una base de datos en *PostgreSQL*.

Las *PK* y *address* se guardan cifradas en la base de datos, y solo este cifrado/descifrado lo efectúa la aplicación *Ethereum*. Esto quiere decir que la aplicación *backend* no conoce ningún dato referente a credenciales para efectuar transacciones en la *blockchain*. *Backend* solo le envía una solicitud a *Ethereum* y este último se encarga de traducir de que usuarios esta hablando *backend* y disparar la transacción correspondiente con los datos que solo el conoce.

¹⁰Por detalles ver 5.3.1

¹¹Ver referencia bibliográfica [69]

Validamos nuestra arquitectura mediante reunión con Gastón Mousques ¹². En base a unas consultas que le hicimos respecto a seguridad nos recomendó reunirnos con Santiago Paz ¹³, para evacuar las dudas respecto al tema.

Eso hicimos y según lo conversado, se nos ocurrió una doble autenticación para el momento preciso de disparar una transacción.

Doble autenticación

Una idea que comenzamos a implementar pero no llegamos incorporar al código fuente del entregable ¹⁴, porque surgieron otras funcionalidades más importantes, es la de una doble autenticación al momento de efectuar específicamente una transacción.

La idea es disminuir las posibilidades de que alguien pueda disparar transacciones si logra saber a que *endpoint* enviar las *request* necesarias para efectuar esas transacciones. Se basa en una solicitud de clave extra al aceptar una proposición, esta clave es auto generada previamente.

Heroku

La utilización de *Heroku* nos brinda seguridad gracias a el uso del canal cifrado *HTTPS*. Esto ayuda a que si alguien se interpusiera en la comunicación, por ejemplo en la modalidad *man-in-the-middle attack*. Los paquetes de datos que pudiera interceptar están cifrados y es un gran impedimento para acceder a los verdaderos datos que van del emisor al receptor.

Resultados obtenidos en cuanto a Seguridad

En la figura siguiente podremos ver un reporte en *Etherscan* [24] con los datos importantes del *Smart Contract* y parte del listado con las ultimas transacciones en que el mismo participó.

¹²Gastón Mousques: Experto en arquitectura de software

¹³Santiago Paz, experto en seguridad informática

¹⁴Lo tenemos en una rama separada para una posible futura implementación

Etherscan
Rinkeby Testnet Network

Contract: 0xe3b35EB49c56CC93314624f4563287B103101Ad6

Sponsored: Advertise to blockchain developers - sponsored slots available. [Book your slot here!](#)

Contract Overview

Balance: 0 Ether

More Info

My Name Tag: Not Available

Contract Creator: 0xcee267634114c3... at txn 0x4f49c8ffdd3f7e35...

Token Tracker: Coopi v3 (CPI)

Transactions | Contract | Events

Latest 25 from a total of 159 transactions

Txn Hash	Block	Age	From	To	Value	[Txn Fee]
0x1c004ca6b4bdbc...	5027722	21 hrs 46 mins ago	0x0a3bfb97aa7a14c...	0xe3b35eb49c56cc...	0 Ether	0.00038303
0x96ddfcdb8213e29...	4860913	29 days 23 hrs ago	0x40f1e8d522cf704...	0xe3b35eb49c56cc...	0 Ether	0.00038303
0x8cf2df37a31112b...	4860767	29 days 23 hrs ago	0x40f1e8d522cf704...	0xe3b35eb49c56cc...	0 Ether	0.00038239
0x9f137d2e1b89d9b...	4855637	30 days 21 hrs ago	0xcee267634114c3...	0xe3b35eb49c56cc...	0 Ether	0.00053303
0x84763c693ab058...	4855628	30 days 21 hrs ago	0xcee267634114c3...	0xe3b35eb49c56cc...	0 Ether	0.00053303

Figura 6.15: Reporte de Smart Contract de Coopify en rinkeby.etherscan.io

De esta forma validamos que las transacciones realizadas por el sistema no se encuentran almacenadas en él, sino que son parte de los bloques de la red de *Ethereum*. De esta forma, aunque el sistema se vea comprometido, las transacciones jamás podrán ser modificadas.

6.3.4 Usabilidad

Para adaptarnos al requerimiento no funcional detallado en 5.3.2, tomamos diferentes caminos para favorecer ese atributo de calidad.

En cuanto a funcionalidades complejas llegamos a la conclusión que se tienen que poder efectuar en menos de siete *clicks* o tres pantallas. Esos fueron números experimentales que tomamos en conjunto con el cliente.

Esto podría hacer que la aplicación fuera sencilla o amigable de utilizar por parte del usuario. Pero si bien pueden efectuarse las tareas necesarias bajo esas cantidad de *click* o pantallas, un aspecto muy importante que también afecta la usabilidad

son los tiempos de respuesta de *backend* y por consiguiente de *Ethereum*. Por lo tanto lo descripto en el atributo de calidad performante, se tomo en cuenta para la elección de esas tecnologías a modo de que las respuestas sean justamente rápidas, por ejemplo la elección de *Node.js*.

Progressive Web App (PWA)

El uso de esta tecnología, nos brinda aspectos positivos en cuanto a la usabilidad.

Uno de los puntos más importantes es que el usuario puede seguir utilizando la aplicación al perder conexión a internet, gracias a estrategias de como manejar la desconexión, que poseen los *service workers*.

La estrategia que maneja nuestro *service worker* es de primero ir lo que esta en internet y si no hay conexión ir por lo que esta en memoria cache.

Los instantes en que el usuario intenta realizar operaciones que no pueden hacerse sin conexión internet, el sistema se encarga de notificarlo al respecto.

Otro punto destacable de contar con una *PWA*, es que cuenta con una única interface compartida a lo largo de diferentes plataformas. En consecuencia los usuarios pueden cambiar el medio por el cual acceden a la aplicación sin tener que aprender nada nuevo.

Heurísticas de Nielsen

Para identificar posibles problemas en cuanto a usabilidad del sistema, podemos hacer un estudio en base a las “10 Reglas Heurísticas de usabilidad de Jakob Nielsen”¹⁵. Es lo que pasaremos a detallar a continuación.

1. **Visibilidad del estado del sistema:** *“el sistema siempre debería mantener informados a los usuarios de lo que está ocurriendo, a través de retroalimentación apropiada dentro de un tiempo razonable”.*

Las funcionalidades más importantes como por ejemplo la negociación que desemboca en una transacción, tiene su aviso cuando sucede, para los dos o mas usuarios que intervienen. Como la transacción se dispara al momento

¹⁵“Jakob estudió 249 problemas de usabilidad y a partir de ellos diseñó lo que denominó las “reglas generales” para identificar los posibles problemas de usabilidad” [28]

de aceptar una proposición, los usuarios pueden estar haciendo cualquier otra operación en el sistema al momento que realmente queda efectiva la transacción real en la red *blockchain*. Por lo tanto en ese momento es cuando se notifica a los usuarios mediante un aviso en pantalla.

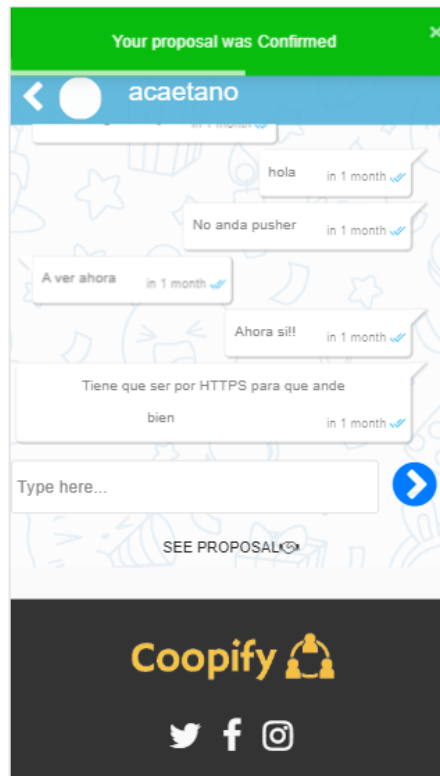


Figura 6.16: Pago de servicio realizado en *Coopify*

Lo mismo para otras funcionalidades menos importantes, como registro, inicio de sesión, crear servicio, etc. Todas ellas cuentan con avisos en pantalla al usuario en casos de error, campos incompletos, datos erróneos, etc.

2. **Relación entre el sistema y el mundo real:** *“el sistema debería hablar el lenguaje de los usuarios mediante palabras, frases y conceptos que sean familiares al usuario, más que con términos relacionados con el sistema. Seguir las convenciones del mundo real, haciendo que la información aparezca en un orden natural y lógico”.*

En este sentido se intento justamente de que el lenguaje utilizado sea sencillo y con palabras familiares al usuario. De todas formas hay lugares donde

sabemos que podríamos mejorar eso, como por ejemplo algunos mensajes o notificaciones al usuario que manejan palabras que utilizamos dentro del sistema y que podrían cambiar logrando una mejor sintonía entre el usuario y el sistema.

3. **Control y libertad del usuario:** *“hay ocasiones en que los usuarios elegirán las funciones del sistema por error y necesitarán una “salida de emergencia” claramente marcada para dejar el estado no deseado al que accedieron, sin tener que pasar por una serie de pasos. Se deben apoyar las funciones de deshacer y rehacer”.*

Si bien no tenemos funciones como “deshacer y rehacer”, independientemente de donde se encuentre el usuario, siempre tiene en pantalla la posibilidad de acceder al menú para moverse a otra funcionalidad rápidamente, saliendo de lo que esta haciendo de inmediato. En el caso de aceptar una proposición por error, esto va a disparar una transacción (en caso de que sea por *Coopies*), proceso que no podemos parar al dispararse. Y al crear por error un servicio, como al momento no tenemos funcionalidad para eliminarlo, no se puede deshacer, de todas formas prevemos que en un futuro se tenga que implementar una funcionalidad al respecto y ya lo teníamos previsto como posible mejora porque escapaba del alcance pactado para el MVP, ver sección 11.4.1.4.

4. **Consistencia y estándares:** *“los usuarios no deberían cuestionarse si acciones, situaciones o palabras diferentes significan en realidad la misma cosa; siga las convenciones establecidas”.*

En este sentido, manejamos siempre las mismas palabras para cosas similares.

5. **Prevención de errores:** *“mucho mejor que un buen diseño de mensajes de error es realizar un diseño cuidadoso que prevenga la ocurrencia de problemas”.*

Un ejemplo para prevenir errores es que cuando un usuario completa un campo con un formato incorrecto darle aviso inmediato y brindarle asistencia sobre como corregirlo. Sino directamente que los campos a completar por el usuario sean acotados en cuanto a los posibles datos que se pueden cargar, un ejemplo de esto se da en las fechas, estos campos que solo aceptan formatos correctos de las mismas y hasta podemos abrir un calendario para seleccionar la fecha como se ve en la imagen.

Birthdate

28/07/1993

julio de 1993

lun.	mar.	mié.	jue.	vie.	sáb.	dom.
28	29	30	1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	1

Figura 6.17: Campo fecha en perfil de usuario

Si bien el sistema cuenta con múltiples confirmaciones para constatar que no sea realizada ninguna acción irregular, entendemos que existe espacio para mejora en asistir al usuario en evitar que esos errores sucedan en primera instancia.

6. **Reconocimiento antes que recuerdo:** *“se deben hacer visibles los objetos, acciones y opciones, El usuario no tendría que recordar la información que se le da en una parte del proceso, para seguir adelante. Las instrucciones para el uso del sistema deben estar a la vista o ser fácilmente recuperables cuando sea necesario”.*

A no ser en la funcionalidad de creación de servicio, que cuenta con dos pantallas para cargar el mismo (de todas formas el usuario no tiene que recordar nada del proceso), no tenemos problemas en este sentido.

Para facilitar la tarea del usuario en caso de tener que recordar contamos con navegabilidad a las entidades involucradas. Digamos, cuando se está en la pantalla de detalle de un servicio, se puede navegar al perfil del usuario dueño de la misma. Cuando se esta en la pantalla de una proposición, se puede navegar al servicio. Al estar en una conversación, se puede navegar a la proposición, etc.

7. **Flexibilidad y eficiencia de uso:** *“la presencia de aceleradores, que no son vistos por los usuarios novatos, puede ofrecer una interacción más rápida*

a los usuarios expertos que la que el sistema puede proveer a los usuarios de todo tipo. Se debe permitir que los usuarios adapte el sistema para usos frecuentes.”

La implementación de atajos de teclado no es algo que tengamos desarrollado porque escapa al alcance del MVP. A futuro nos parecería correcto, proveer de algunos atajos para alguna funcionalidad que realmente lo necesite.

Destacamos la inclusión de tutoriales que ayuden al usuario a comprender para que sirven determinados elementos de la interfaz del sistema¹⁶.

En esta etapa se invirtió esfuerzo en captar y mantener usuario novato y no tanto en facilitarle el accionar a un usuario experto.

8. **Estética y diseño minimalista:** *“los diálogos no deben contener información que es irrelevante o poco usada. Cada unidad extra de información en un diálogo, compite con las unidades de información relevante y disminuye su visibilidad relativa.”*

Creemos que los diálogos mostrados son bastante cortos y que cumplen con la regla. La interfase es a nuestro entender bastante minimalista.

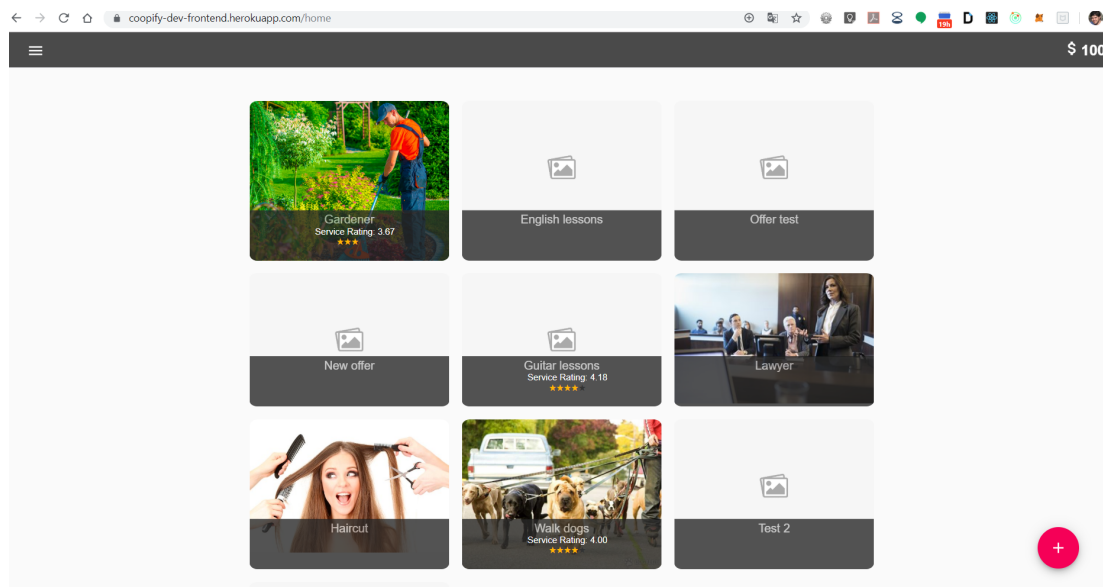


Figura 6.18: Servicios en *Coopify*

¹⁶Este tutorial no lo tenemos activo al momento, pero con unos pequeños cambios podría quedar funcional y ayudaría mucho al usuario

De todas las pantallas que posee el sistema, la que menos se adapta a esto puede ser la del detalle de servicio, porque posee mucha información. De todas formas esta desplegada correctamente y con la respiración necesaria para entender lo que muestra la misma.

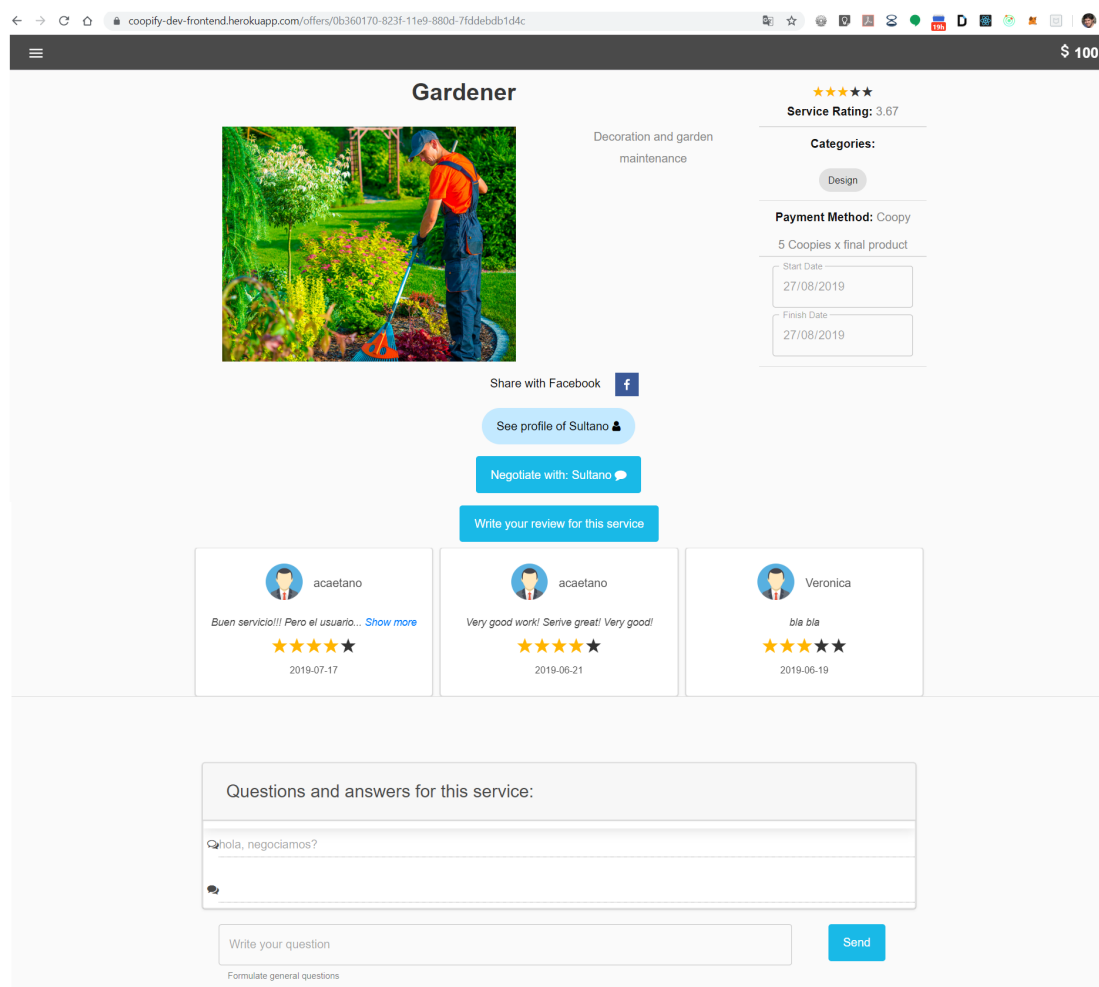


Figura 6.19: Detalle de servicio en *Coopify*

9. **Ayudar a los usuarios a reconocer, diagnosticar y recuperarse de errores:** *“los mensajes de error se deben entregar en un lenguaje claro y simple, indicando en forma precisa el problema y sugerir una solución constructiva al problema”*

Los avisos en cuanto a errores se despliegan de manera correcta, por que

el usuario se encuentra informado de lo que esta sucediendo. En lo que si se podría profundizar, es en sugerencias en base a problemas, para que el usuario sepa que hacer en cada caso.

A modo de ejemplo, para hacer una transacción la cuenta del usuario tiene que tener una cantidad necesaria de *Ether*¹⁷. En caso contrario la transacción nunca sería minada o confirmada. Esto dejaría el usuario en un estado donde no podría utilizar una de las funcionalidades centrales como el realizar pagos con *coopies*, lo cual sería extremadamente frustrante.

Por lo tanto el sistema esta preparado, para en el caso de que se quiera hacer una transacción sin contar con el saldo necesario, se despliega un mensaje de error y el sistema automáticamente se encarga de cargar *Ether* a la cuenta de ese usuario para que no le vuelva a ocurrir.

10. **Ayuda y documentación:** *“incluso en los casos en que el sistema pueda ser usado sin documentación, podría ser necesario ofrecer ayuda y documentación. Dicha información debería ser fácil de buscar, estar enfocada en las tareas del usuario, con una lista concreta de pasos a desarrollar y no ser demasiado extensa.”*

Es un debe que no entra dentro del alcance que establecimos con el cliente. Generar una documentación básica de uso del sistema sería una buena opción para desplegar en los lugares precisos que el usuario lo necesite.

¹⁷Entraremos más en detalle sobre el *Ether* en la sección 7.4

6.4 Conclusión de Arquitectura

La arquitectura planteada cubre todos los atributos de calidad especificados en los requerimientos no funcionales, detallados en 5.3.2. Y lo hace gracias a los puntos destacados anteriormente.

Una de las dificultades mas importantes del proyecto es la implementación de la *Blockchain* y que los tiempos insumidos por los procesos de la misma, no interfieran en los tiempos del usuario. Digamos que el usuario no tenga que preocuparse por esperar cuando se dispara una transacción.

Por este motivo fue que implementamos el servidor de *Ethereum* de forma separada al de *Backend*. Si bien podíamos haberlo hecho juntos creemos que la solución actual nos da la posibilidad de, cambiar la implementación elegida para *blockchain* sin obligar a quienes interactúan con este servidor a recompilarse en consecuencia.

Esto que detallamos anteriormente sumado a la ayuda brindada por el uso de *Pusher*, para notificar al usuario en transacciones, separó de forma óptima, los tiempos de la red *blockchain*, con los requeridos por el usuario.

Otro foco importante que hicimos, fue en la posibilidad de escalar dependiendo del uso del sistema. Todo esto gracias al uso de *Heroku* y sus *Dynos*. Si bien el punto escalabilidad no estaba establecido como un requerimiento no funcional, creemos que es de extrema importancia en un sistema de estas características.

Importante en cuanto a usabilidad fue haber decidido utilizar *PWA* en lugar de crear aplicaciones nativas para cada sistema operativo. Ya que nos resumió bastante el manejo de la desconexión de internet, para que el usuario sienta la sensación de seguir conectado casi siempre.

Mantener el mismo *stack* tecnológico *Javascript* tanto en *Backend* como en *Frontend*, fue importante ya que nos permitió reducir el tiempo invertido en capacitarnos en el uso de nuevas tecnologías.

6.5 Lecciones aprendidas

Respecto al uso de *React* para desarrollar el *frontend*, creemos que debido a nuestra inexperiencia con ella y la elevada curva de aprendizaje que la caracteriza, contribuyeron a que el equipo no utilizará su enfoque orientado a construir componentes

de manera óptima. Si bien el equipo invirtió un esfuerzo para lograr componentes reutilizables y atómicos, en caso de comenzar el proyecto con el conocimiento que tenemos ahora, el código que terminaríamos logrando gozaría de una mantenibilidad ampliamente superior.

Otra de las lecciones aprendidas es la ayuda en cuanto a configuraciones que nos brinda *Heroku*. En un principio, habíamos pensado en la utilización de *Amazon AWS*, pero gracias a investigar nos dimos cuenta que nos complicaríamos bastante en configuraciones. Cosa que para la etapa de creación de un *MVP* y desarrollando un proyecto que por lo menos en nuestro alcance no llegaría a estar en producción, no era necesario. Digamos que fue algo bueno, el quitar complejidad para centrarnos en lo verdaderamente importante.

Si bien utilizamos variada tecnología para la implementación de las diferentes partes del sistema, mantenemos en gran medida el *stack* tecnológico de *Javascript*. Más que nada a nivel de *Backend* y *Ethereum*, partes centrales del sistema. Eso es favorable en cuanto a mantenibilidad.

Por último pero no menos importante, el contar con etapas de validación con expertos (Gastón Mosques y Santiago Paz) nos permitió expandir nuestra percepción de la solución construida, permitiéndonos ver múltiples oportunidades de mejora y mostrándonos que siempre hay trabajo por delante.

7 Análisis del problema tecnológico

En este capítulo analizaremos las posibles tecnologías a utilizar a lo largo del proyecto. Tanto para la implementación a nivel de *blockchain*, como así también las tecnologías para desarrollar las aplicaciones que componen el sistema en su totalidad.

7.1 Selección de tecnología Blockchain

Para decidir cual de las tecnologías *Blockchain* utilizar para implementar los *Copies*, hicimos una investigación tomando en cuenta las tecnologías en las cuales el cliente estaba interesado. Estas tecnologías son *Ethereum* e *Hyperledger Fabric*.

Para esto dividimos el proyecto en dos etapas bien marcadas, una etapa de investigación y luego la etapa de desarrollo.

Para la etapa de investigación utilizamos diferentes metodologías que desarrollaremos en detalle más adelante, en el capítulo de gestión de proyecto 8. El resultado detallado de esta etapa se puede encontrar en el anexo 13.7.

En el documento de investigación se hizo una introducción en cuanto a la historia de *Blockchain* y las posibilidades que brinda en cuanto a la creación de redes descentralizadas. Luego hacemos una comparación resaltando las ventajas y desventajas de las tecnologías *Ethereum* e *Hyperledger Fabric*. Para luego plantear las diferencias entre las tecnologías, en base a puntos precisos que detallamos a modo de titulares:

- Propósito

Son plataformas creadas con propósitos totalmente diferentes. *Ethereum* apro-

vecha sus *Smart Contracts* con el propósito de ser descentralizada y utilizada por gran cantidad de usuarios. Por otro lado *Hyperledger Fabric* aprovecha la tecnología *blockchain* para el ámbito de negocios.

- Confidencialidad

Ethereum es transparente a todos los usuarios de la red, mientras que en *Hyperledger Fabric*, es posible la existencia de transacciones confidenciales entre determinados usuarios de la red.

- Modo de participación

Ethereum puede ser utilizado mediante dos modalidades: una red pública o una privada sin ningún tipo de permisos, mientras que *Hyperledger Fabric* es una red privada y con permisos.

- Mecanismo de consenso

En *Ethereum* todos los participantes de la red (también llamados nodos) deben llegar a un consenso sobre todas las transacciones. *Hyperledger Fabric* sin embargo, tiene un control más granular sobre el consenso y acceso restringido a las transacciones.

- Lenguaje de programación

El uso de *Smart Contracts* en *Ethereum* escritos en un lenguaje de alto nivel denominado *Solidity*. En *Hyperledger Fabric* se utiliza el termino “*chaincode*” escrito en *GoLang*, un lenguaje de programación creado por *Google*.

- Criptomoneda

Hyperledger Fabric no requiere una criptomoneda para las transacciones, por lo que tampoco proviene con una criptomoneda incorporada. Por lo tanto no requiere de minería, en cambio *Ethereum* posee una criptomoneda propia llamada *Ether*, la cual se utiliza como recompensa a aquellos nodos que contribuyen a lograr un consenso mediante la minería de bloques.

Conclusión de investigación

En base al proceso llegamos a la conclusión de que la tecnología que mas se adapta a lo que necesitamos, es *Ethereum* y esto en base a determinadas bondades que pasamos a detallar como anexo 13.7.

En resumen

- Genérica y adaptable.
- No controlada (descentralizada).
- Menor tiempo para validar transacciones.
- Utilización de *Smart Contracts*.
- Uno de los puntos mas importantes que no cubre *Hyperledger Fabric* es que no tiene una manera sencilla de implementar una criptomoneda, cosa que si hace *Ethereum*.

7.2 Coopies (Token ERC20 / CPI)

Dada la necesidad de contar con una criptomoneda que nos permita realizar las transacciones. Conseguimos implementar una mediante un *Smart Contract* ¹.

Para desarrollar uno tenemos que construir un archivo utilizando el lenguaje de programación *Solidity* ² para que la lógica de ese contrato se ejecute cada vez que se haga una transacción que involucre a ese contrato.

Para programar ese contrato existen múltiples alternativas, estas dependen del ambiente:

- Ambiente local: (con *Ganache*, *Truffle* y un *IDE* de texto cualquiera)³
- Ambiente remoto de prueba: (*Rinkeby test network*) ⁴
- Ambiente remoto en producción: (*Main network Ethereum*), ver figura 7.1

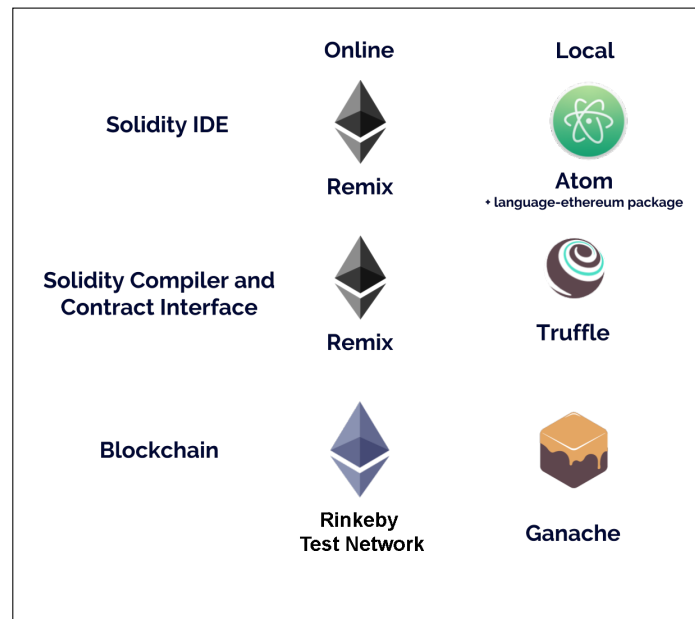


Figura 7.1: IDE Online / IDE Local [29]

¹Código *Smart Contract* estándar, ver referencia bibliográfica [26]

²*Solidity*, ver referencia bibliográfica [65]

³*Ganache* y *Truffle*: Ver referencia bibliográfica [1]

⁴*Rinkeby*

Hasta que un contrato no sea publicado no lo podremos utilizar. Para eso utilizamos *Remix* ⁵ que tiene incorporadas herramientas con las que agregar el contrato a la red de prueba o producción.

Al publicar el contrato (hacer *deploy*) en la red de prueba perteneciente a *Ethereum* (*Rinkeby*), este perdura y es inmodificable. La diferencia principal que tiene trabajar inicialmente en local es poder hacer *deploy* del *Smart contract* varias veces ya que se esta en un entorno mas cerrado. Como el *Smart contract* que utilizamos es prácticamente un estándar dentro de los existentes para la creación de los *Tokens*, no necesitábamos de mucho trabajo a nivel local, por lo que directamente lo hicimos en la red de prueba *Rinkeby*. *Token* identificado como *CPI*.

La publicación resulta en una dirección única que identifica a nuestro contrato y un *ABI* ⁶ que podemos usar para que nuestro servidor conozca los contenidos del contrato. Este permite saber el nombre de las funciones del *Smart Contract* y conocer el retorno de esas funciones, para poder trabajar en base a estos.

Utilizando el *ABI* como receta de interacción con el contrato ya publicado y la dirección del contrato, podemos lograr que nuestro servidor comience a realizar transacciones reales con nuestro contrato y distribuya nuestra criptomoneda. Para eso el equipo utilizó *web3js*, una librería proporcionada por el repositorio de *Ethereum* con el fin de simplificar la interacción con el contrato.

Tanto para hacer la publicación como para hacer cualquier transacción en la red de *Blockchain* se consume una cierta cantidad de gas. Ese gas se traduce en un gasto de *Ether* (de prueba en la red *Rinkeby*), conseguido a través del grifo (*faucet*) brindado por la red *Rinkeby*, que nos da la posibilidad de adquirir de forma gratuita *Ether* de prueba.

Para una depuración de situaciones particulares de las cuentas de usuario (*accounts*) usamos *Metamask* ⁷ y para ver todas las transacciones usamos *RinkebyEtherscan*.

⁵IDE online Remix, ver referencia bibliográfica [61]

⁶ “La Application Binary Interface (Interfaz Binaria de Aplicación) o ABI es el modo estándar de interactuar con contratos en el ecosistema Ethereum, tanto desde fuera de la blockchain como en interacciones contrato-contrato. Los datos se codifican siguiendo su tipo acorde a esta especificación” [23]

⁷*Metamask*: ver referencia bibliográfica [46]

7.3 Rinkeby Test Network

La utilización de la red *Ethereum* tiene costos en *Ether*, por lo que para la etapa de desarrollo utilizamos una red de prueba llamada *Rinkeby Test Network*.

Rinkeby Test Network pertenece a *Ethereum*, digamos, funciona de la misma manera que la red *Main* de *Ethereum*, pero con la posibilidad de hacer transacciones gastando *GAS*⁸ y por consiguiente *Ether* ficticios, que solo corresponden a la red de prueba. Ya que de otra forma tendríamos que disponer de *Ether* real para pagar el gas correspondiente para efectuar las transacciones.

El *Ether* manejado en esta red de prueba se consigue por intermedio de publicaciones de la *Address* de la cuenta general que tienen *Coopify*, en una red social, por ejemplo en (*Facebook*, *Twitter*, *Google Plus*, etc). Luego de haber publicado, se copia el *link* de la publicación y mediante (<https://www.rinkeby.io/#faucet>), pegamos el *link* de la publicación y damos *click* en ***Give me Ether***.

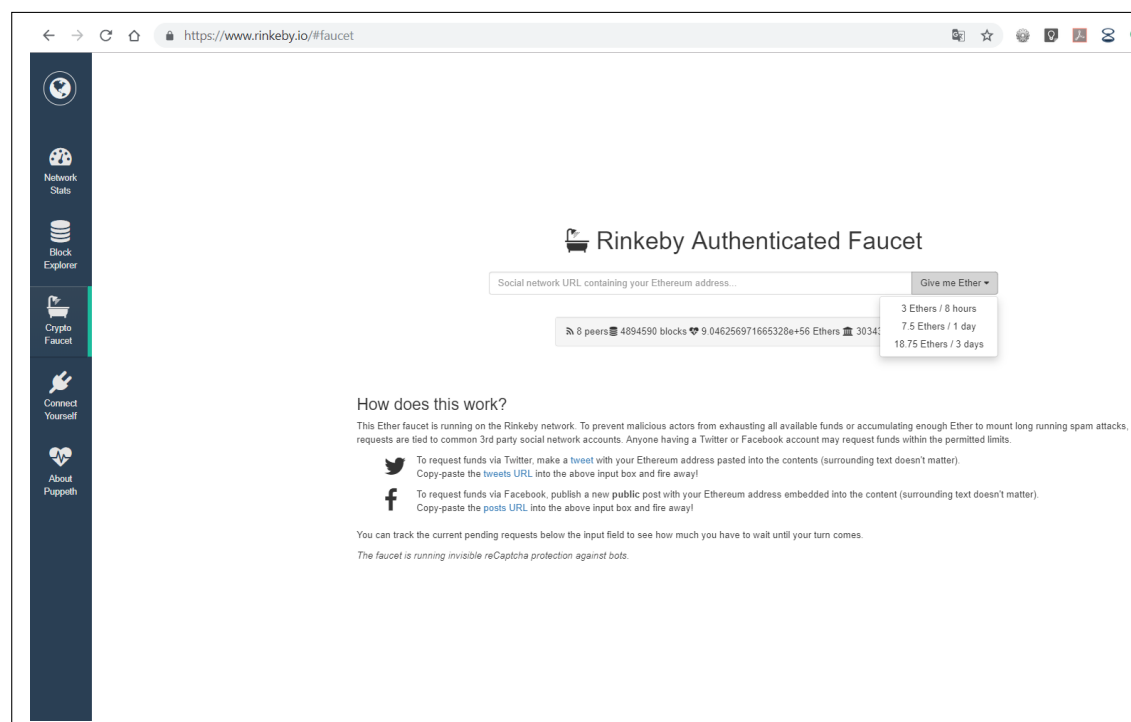


Figura 7.2: <https://www.rinkeby.io/#faucet>

⁸ *GAS*, ver referencia bibliográfica [47]

7.4 Ether

Para utilizar *Ethereum*, es necesario usar su criptomoneda llamada *Ether*, ella hace funcionar la red. Mas precisamente para efectuar las transacciones entre usuarios. Cada vez que se quiere efectuar una transacción se utiliza *GAS*, esto de fondo lo que necesita es *Ether*.

Dicho esto, cada cuenta de usuario en la plataforma *Coopify* necesita de *Coopies*, pero a su vez pequeñas cantidades de *Ether*. Para pagar ese *GAS* utilizado en la computación de esos cambios en la red *Ethereum*.

Carga de Ether a cuenta de usuario

Cuando un usuario quiere efectuar una transacción, el sistema intenta efectuarla y si no pudiese por falta de *Ether* para solventar el *GAS* necesario, el sistema en respuesta carga *Ether* a la cuenta del usuario. Para que en un segundo intento pueda efectuar esa transacción. Por mas detalle, ver **13.17**.

Ese *Ether* sale de una cuenta general que posee *Coopify* en la red *Ethereum*. Por tanto, en esta cuenta se debe de tener *Ether* disponible para poder efectuar las cargas a cuentas de usuarios que contábamos previamente.

7.5 Backend

Al comenzar el desarrollo contamos con dos alternativas tecnológicas para construir el servidor centralizado; *NodeJS* y *Ruby on Rails*, en esta sección las compararemos y justificaremos la seleccionada por el equipo.

7.5.1 NodeJS

NodeJS es un entorno de ejecución programado en *Javascript* y *C++* que fue construido para ser altamente escalable utilizando el motor V8 de *Google* que le permite correr *Javascript* en el servidor.

Si bien es una tecnología más joven que *Ruby on Rails*, la comunidad y grado de

aceptación que ha logrado en el mercado desde su aparición en 2009 lo respaldan. Agregado su comunidad es una tecnología que cuenta con un aprovechamiento excelente de un conjunto de recursos bastante limitado para responder rápidamente a un gran número de solicitudes asincrónicas.

En la realidad de este proyecto si bien la mayoría no cuenta con experiencia utilizando este lenguaje y *framework*, un recurso se desarrolla como programador *backend* con algunos años de experiencia utilizando esta tecnología.

Agregado a esto dado que las tecnologías a utilizar en las capas superiores son en su mayoría basadas en *Javascript*, elegir *NodeJS* para el *backend* tiene sentido para manejar un único lenguaje de programación a lo largo de todo el proyecto.

7.5.2 Ruby on Rails

Ruby on Rails es un marco de trabajo construido con el objetivo de simplificar el proceso de desarrollo haciendo gran foco en la reutilización de código a lo largo de solución, agregado a esto hay un esfuerzo para disminuir la configuración de los entornos de desarrollo a un mínimo.

Es uno de los principales *frameworks* para el desarrollo de pequeñas aplicaciones y prototipos funcionales, esto se debe principalmente a su comunidad madura y las gemas que ofrece. Estas gemas tienen como objetivo resolver algún problema que fue encontrado repetitivamente a lo largo de diferentes proyectos.

En la realidad de este proyecto la mayoría de los integrantes no cuenta o cuenta con muy poca experiencia utilizando este lenguaje y *framework*.

7.5.3 Framework seleccionado y justificación

Teniendo en cuenta lo expuesto previamente, el equipo se inclinó por utilizar *NodeJS* para el desarrollo del servidor centralizado. Los factores principales que nos llevaron a tomar esta decisión fueron;

- Utilizar un mismo lenguaje a lo largo de *Frontend* y *Backend*. Esto nos permite que todo el grupo pueda trabajar y colaborar de una manera mucho más sencilla.
- Experiencia de uno de los recursos. Esto nos permitió contar con mayor experiencia en la resolución de problemas más comunes y dedicarle mayor esfuerzo a los problemas que conforman el núcleo de este proyecto.

7.6 Frontend

Para desarrollar la aplicación cliente, teníamos disponibles una serie de alternativas en cuanto a tecnologías. En esta sección las compararemos y justificaremos la elección tomada.

Las posibles tecnologías a utilizar, tomadas como requerimientos no funcionales son:

- Para la creación de la web *SPA*: *React*, *Angular* o *Vue*.
- Para la creación interfaces móviles nativas: *ReactNative*, *Ionic*, *Flutter* o *NativeScript*.

7.6.1 Para la creación de web SPA

7.6.1.1 ReactJS

“React es una biblioteca Javascript de código abierto diseñada para crear interfaces de usuario con el objetivo de facilitar el desarrollo de aplicaciones en una sola página (SPA). Es mantenido por Facebook y la comunidad de software libre” [103].

Esta basado en componentes encapsulados que manejan su propio estado, permitiendo combinarlos para crear interfaces de usuario complejas. La lógica de estos componentes esta escrita en *Javascript*. La sintaxis utilizada en *React* es *JSX*, sintaxis propia siendo una mezcla entre *Javascript* y *XML*.

Los componentes utilizan un *state*, donde se almacenan los datos del contexto general de los componentes, y pueden recibir datos de los componentes padres a través de las *props* del componente.

React esta siendo utilizado en muchos sitios importantes como por ejemplo *Airbnb*, entre otros.

7.6.1.2 AngularJS

“Es un framework frontend mantenido por Google que posee una gran cantidad de objetos para la realización de una webapp”.[55]

“Algo interesante es que Angular funciona con Typescript, destacándose como una solución de sintaxis fuertemente tipada”[55]

En *Angular* en lugar de inyectar *JS* como en el caso de *React*, las directivas y utilidades que trae son más que suficientes para realizar cualquier tarea, utilizando *Templates*.

Si bien algunos de los integrantes del proyecto cuenta con mínima experiencia en la utilización de Angular, fue con las primeras versiones (*Angular 1* y *Angular 2*) y si tenemos en cuenta que la versión al momento es *Angular 7* y que si bien es compatible mas que nada con la 2 (no tanto con la 1). La curva de aprendizaje del equipo seria prácticamente igual a las otras tecnologías.

7.6.1.3 VueJS

Vue es un *framework* de código abierto, desarrollado en *Javascript*, para construir interfaces y aplicaciones de una sola pagina (*SPA*).

Fue creado simplificando o resumiendo Angular para desarrollar algo mas liviano.

Si bien sabemos que la evolución de *VueJS* va en progreso y resume muchas de las bondades de las tecnologías antes planteadas, nos apoyamos mas en lo robusto de las que ya tienen ya un poco mas de uso.

7.6.2 Para la creación de interfaces móviles

En primera instancia, como requerimiento no funcional se nos estableció por parte del cliente la utilización de *ReactNative*, *Ionic*, *Flutter* o *NativeScript*. Pero en base a un consejo del tutor Darío Macchi, estudiamos con el cliente la posibilidad de incursionar en la tecnología *Progressive Web Apps (PWA)*.

“Una PWA utiliza las últimas tecnologías disponibles en los navegadores para ofrecer una experiencia en móviles lo más parecida a la de una aplicación nativa”[4].

Esta nueva tecnología, nos brinda la posibilidad de hacer un único desarrollo para los distintos tipos de tecnologías, *iOS* y *Android*.

Al cliente le pareció muy buena idea que incursionáramos en esa tecnología y eso hicimos.

7.6.3 Tecnologías seleccionadas y justificación

Por lo tanto en base a lo antes dicho, la tecnología seleccionada es *React*. Una de las principales razones de la elección fue de mantener el stack tecnológico *Javascript* y algún caso de éxito importante como *Airbnb* como credenciales.

Además la curva de aprendizaje seria prácticamente igual en cualquiera de las tecnologías planteadas dada la poca experiencia del grupo en las mismas.

Boilerplate PawJS

Investigando sobre las tecnologías seleccionadas, *React* y *PWA*.

Comenzamos a buscar algún *boilerplate* que nos facilitara un poco el trabajo ya que no teníamos experiencia en ninguna de esas tecnologías más que conocimientos sólidos de alguno de los integrantes en cuanto a *Javascript*. Y fue cuando encontramos *PawJS*.

Al cliente le pareció bien la utilización de este *boilerplate* y por tanto fue nuestra base para comenzar a desarrollar nuestro *frontend*.

7.6.4 Redux-Saga middleware

El modelo de aplicación utilizado para el desarrollo del *frontend* es mediante la utilización de *Redux-saga middleware*.

Redux se trata de una librería en *Javascript* que se encarga de desacoplar el estado global de una aplicación web de la parte visual, es decir los componentes.

El uso de esta librería nos ofreció una forma sencilla, eficiente y prolija de poder manejar un estado global a nivel de la aplicación para luego interactuar con los datos desde distintos componentes.

Sagas se define como un *middleware* diseñado para manejar los efectos secundarios de la aplicación de manera sencilla y prolija.

El uso de *sagas* nos permitió un mejor manejo de la asincronía y efectos secundarios en las llamadas a la *API*, así como también favorecer la modificabilidad del sistema a través de módulos independientes que se encargan de procesar las llamadas asíncronas.

El flujo completo de la combinación de ambas herramientas puede apreciarse en el siguiente diagrama.

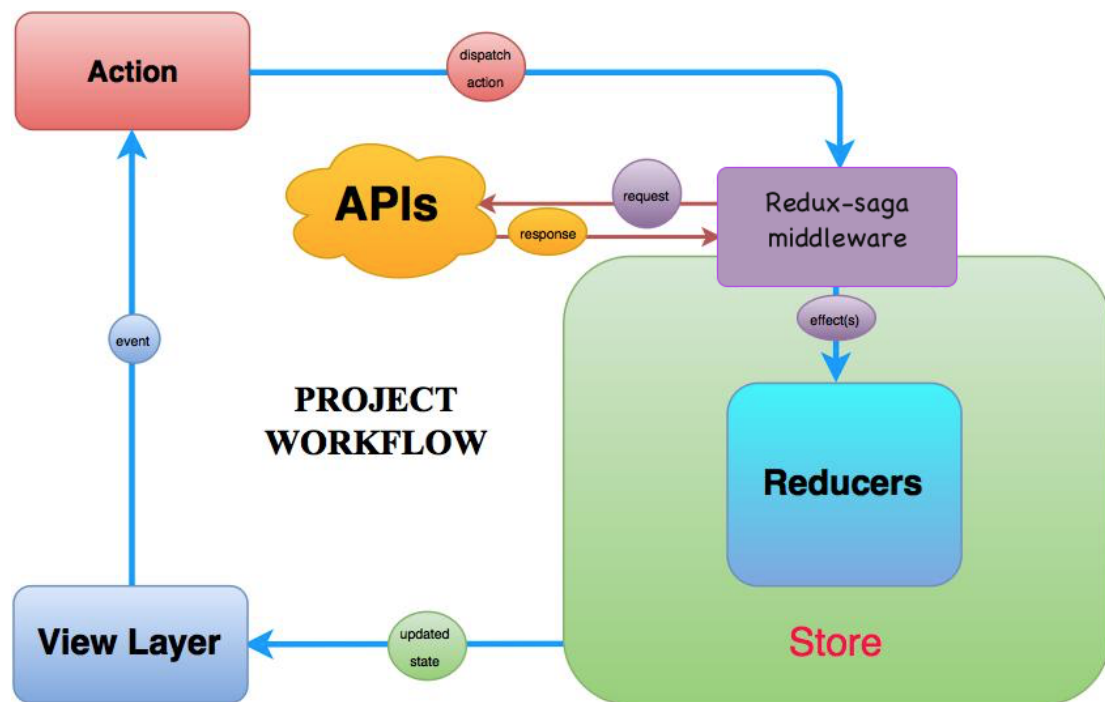


Figura 7.3: Diagrama de flujo de *Redux-Saga middleware* [41]

Cualquier cambio en una *View Layer* dispara un evento.

Estos eventos a los que están suscritos los componentes de las vistas (*View Layer*), disparan a su vez acciones (*Action*), haciendo que se despache (*dispatch*) lo necesario para que el *Redux-Saga middleware* pueda efectuar las *requests* correspondientes y obtener la respuesta de la *API* del *backend*.

Para esto ultimo el *sagas* intercepta la *request* y se encarga de comunicarse con la *API* enviando los datos pertinentes.

Al recibir los mismos mediante los *effect* envía la respuesta a los *Reducers*, para que estos modifiquen las variables que conforman el *Store*.

Luego el *Store* es el encargado de hacer *Update* en el estado, para que la *View Layer* actualice las vistas correspondientes según las modificaciones.

7.7 Servicios de alojamiento

En la siguiente sección vamos a discutir las decisiones que se tomaron para alojar el código en ejecución en un ambiente remoto. Para satisfacer esta necesidad evaluamos dos alternativas: *Amazon Web Services* y *Heroku*. Luego de investigar un poco nos inclinamos por *Heroku* pues su orientación *PAAS* (*Platform as a Service*) que nos permitía centrarnos en el desarrollo. De esta manera pudimos invertir la mayoría del tiempo en el proceso de desarrollo, en vez de gastarlo configurando maquinas virtuales, *firewalls*, *security groups* o *vpns*.

Otro aspecto positivo de elegir *Heroku* como plataforma de alojamiento es que nos permite escalar rápidamente cambiando el tipo de *dyno* utilizado (para escalar horizontalmente) y la cantidad disponible (escalando verticalmente). Un *dyno* es la representación ofrecida por *Heroku* para ofrecer a sus usuarios poder de computo cada uno con sus recursos físicos asociados.

En adición a las prestaciones previamente mencionadas *Heroku* presenta otra ventaja respecto a la alternativa, la capacidad de contar con *WebHooks* que nos permitieran tener en ejecución todo cambio que sea llevado a determinada rama del repositorio.

7.8 Ambientes y proceso de despliegue

Al comienzo del proyecto el equipo conversó con el cliente llegando a la conclusión de que dadas las particularidades del proyecto lo mejor era contar con un único ambiente remoto alojado. El principal motivo para esto es que el objetivo del cliente es obtener un prototipo funcional, él no llevaría a cabo un conjunto de pruebas extenso y por lo tanto tener un ambiente dedicado a ello parecía un desperdicio.

Proceso de despliegue

Como se comentó previamente el proceso de despliegue utiliza la integración ofrecida por *Heroku* con los *Webhooks* de los repositorios de *GitHub*. Eso nos permitió que cada cambio que tenga una determinada rama del repositorio dispararán un proceso de despliegue automatizado.

Utilizando los ganchos de proceso ofrecidos por *Heroku* fuimos capaces de definir dos recetas personalizadas y una receta estandarizada para automatizar el despliegue de la versiones al ambiente remoto.

Estas recetas son las siguientes:

- Proceso de instalación de dependencias: en este paso se descargan las dependencias de pertinentes.
- Proceso de compilado: es el proceso por el cual se toman las dependencias y el código fuente para lograr una versión minimizada de esa versión del código para ser más performante.
- Proceso de inicio: es el proceso por el cual se pone en ejecución esa versión del código.

7.9 Servicios de terceros

Con el fin de construir una aplicación robusta y eficiente, el equipo decidió acoplarse a ciertos servicios provistos por terceros.

Durante la etapa de desarrollo el equipo observó que habían determinados problemas que eran independientes de la lógica de negocio del sistema. Decidimos clasificar estos problemas en los siguientes tres tipos:

1. **Arquitectura elegida:** Son todos aquellos problemas los cuales surgieron en base a la arquitectura escogida, por ejemplo la comunicación en tiempo real entre los componentes de nuestra aplicación.
2. **UI/UX:** Son todos aquellos problemas que implican una mejora en la experiencia de usuario y usabilidad en cuanto a la interfaz gráfica, por ejemplo la subida de imágenes en el perfil de los usuarios.
3. **Otros** Son todos aquellos problemas que no encajan en ninguno de los dos anteriores, por ejemplo el envío de correos electrónicos como notificación a los usuarios registrados en el sistema.

De este modo resulta mucho mas sencillo y eficiente que nuestro sistema se acople a la implementación provista por un tercero que tener que implementar una propia nosotros.

A continuación se detallan los servicios externos utilizados durante el desarrollo del sistema.

7.9.1 Pusher

“Los canales de Pusher proveen una comunicación en tiempo real entre servidores, aplicaciones y dispositivos. Los canales son utilizados para notificaciones, chat, videojuegos, actualización de paginas web, IoT y muchos otros sistemas que requieran comunicación en tiempo real.” (traducido de [57]).

Pusher nos brinda una comunicación en tiempo real entre componentes permitiéndonos detectar cambios en uno de manera automática e inmediata. De esta forma podemos actuar en base a los mismos sin necesidad de consultar manualmente cada determinado período de tiempo (*polling*).

Se utilizó en los siguientes componentes:

- comunicación desde *backend* hacia *frontend*
- comunicación desde *ethereum* hacia *backend*

Este servicio nos permitió:

- Lograr comunicaciones al completarse las transacciones (ya sea en caso positivo, como negativo), debido a los tiempos manejados por *blockchain* era importante permitir que el usuario siguiera pudiendo usar el sistema mientras las transacciones eran procesadas.
- Lograr comunicaciones desde *backend* cuando un usuario envía un mensaje a una conversación para lograr un sensación de inmersión e inmediatez.

7.9.2 Clouldinary

Clouldinary es un servicio de alojamiento y gestión de imágenes el cual a través de su *API* provee los mecanismos para subir una imagen a su servidor y obtenerla posteriormente.

Utilizamos este servicio dado que dicha *API* resulta bastante sencilla en comparación con otros servicios, y a su vez cuenta con un modo gratuito que nos permitió durante el desarrollo poder subir imágenes y obtenerlas sin necesidad de implementar su funcionamiento.

7.9.3 Sendgrid

“Sendgrid es un servicio de correo electrónico basado en la nube que ofrece un sistema confiable de entrega de correo electrónico transaccional, escalabilidad y análisis en tiempo real junto, con APIs flexibles que facilitan la integración personalizada”. [20].

Optamos por utilizar este servicio para manejar el envío de correos electrónicos en todas las funcionalidades que requieran una notificación por parte del sistema a usuarios registrados en el mismo.

7.9.4 Etherscan

“Etherscan es el líder en la exploración de la red de Ethereum. La clave de Etherscan consiste en extraer información de datos del libro mayor distribuido de Ethereum, indexar y mostrar los datos procesados de manera concisa y legible para las personas”. [15].

Etherscan se trata de una plataforma de análisis y exploración de bloques para la red *Ethereum*, la misma provee una interfaz gráfica donde se pueden visualizar todas las transacciones realizadas así como también información específica de una determinada *address* en la red.

Asimismo, provee de una *API* para desarrolladores que permite realizar consultas a la red de *Ethereum*, esto facilita la interacción con dicha red permitiendo resolver consultas complejas sobre la misma de manera sencilla y con un buen grado de *performance*.

El equipo utilizó dicha *API* para la obtención del listado de transacciones de una determinada *address* en un tiempo cercano a nuestros requerimientos.

8 Gestión de Proyecto

8.1 Etapas del proyecto

Como se hizo mención anteriormente en la sección 4.1, el núcleo del sistema radica en la correcta implementación de transacciones reguladas con la tecnología *blockchain*.

Sin embargo, ningún integrante del equipo contaba con experiencia o conocimiento en esta área y dado que el distintivo de nuestro sistema y los requerimientos funcionales más importantes del mismo se basan en estas tecnologías, era de vital importancia escoger una plataforma que se adecuase a nuestros requerimientos ya que su elección impactaría luego en el desarrollo.

De esta manera el equipo consideró conveniente reservar una etapa del proyecto a la investigación de las distintas plataformas de *blockchain* y la elaboración de un entregable final al cliente con la decisión de esta última.

Una vez culminada la etapa anterior, se comenzó con la etapa de desarrollo del proyecto, en donde se hizo un énfasis especial en el diseño y desarrollo de la solución propuesta. Cabe destacar que para esta etapa utilizamos una metodología distinta a la de la investigación, siendo esta *Scrum* como se menciona en la sección 4.4.2, esto se detallará más adelante en este capítulo.

8.1.1 Investigación

Se trata de la primera etapa del proyecto. El objetivo de la misma es que el equipo adquiriera conocimientos relacionados a aquellas tecnologías en las cuales no cuenta con experiencia o conocimiento como *blockchain*, *PWA* y *React*.

Por otro lado, se trata también de determinar que plataforma de *blockchain* utilizar en la etapa de desarrollo. Esta decisión quedará documentada en un entregable al cliente con los detalles de la investigación y la justificación de dicha elección.

Esta etapa se extendió entre la primera semana de octubre de 2018 hasta mediados, fines de enero de 2019 en donde se realizó la transición a la etapa de desarrollo.

FASE	DETALLES	INVESTIGACIÓN												TRANSICIÓN			
		2018															
		OCTUBRE			NOVIEMBRE				DICIEMBRE					ENERO			
		KANBAN												KANBAN / AGILE			
	SEMANA DEL PROYECTO:	Introduce la fecha del primer lunes de cada mes -->															
		15	22	29	5	12	19	26	3	10	17	24	31	7	14	21	28
1	Concepción e inicio del proyecto																
2	Investigacion																

Figura 8.1: Calendario etapa de investigación

Desglose del calendario

Concepción e inicio del proyecto

- Coordinación con el tutor
- Revisión de los objetivos, herramientas y procesos
- Inicio

Investigación

- Investigación de tecnologías *blockchain*
- Investigación de tecnologías *backend*
- Investigación de tecnologías *frontend*
- Investigación sobre *PWA*
- Elaboración de documentos intermedios para presentación de avance a los clientes
- Gestión de riesgos

Registro de horas

Para llevar un control sobre las horas invertidas en esta etapa (y a lo largo de todo el proyecto) se decidió utilizar la herramienta *Toggl*. Decidimos utilizarla principalmente por su facilidad de uso, agregado a esto el hecho de que su sección de reportes es muy flexible e intuitiva.

8.1.2 Desarrollo

Se trata de la segunda etapa del proyecto. Durante la misma el equipo se dedicó exclusivamente al desarrollo del sistema.

Esta etapa contempla a su vez, varias decisiones de la arquitectura y diseño del mismo para su posterior implementación.

Esta etapa se extendió entre la última semana de enero de 2019 hasta mediados, fines de julio de 2019 en donde se realizó la reunión de cierre con el cliente y se procedió a realizar la documentación final.

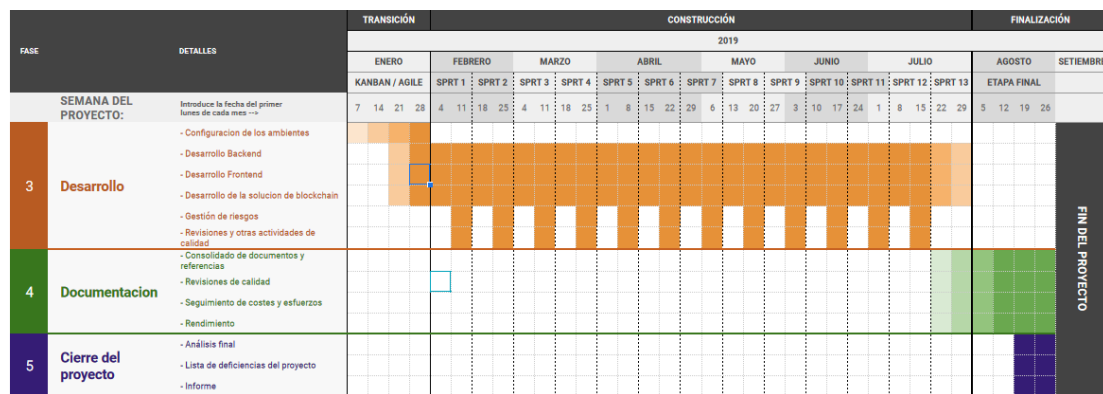


Figura 8.2: Calendario etapa de desarrollo

Desglose del calendario

Desarrollo

- Configuración de los ambientes
- Desarrollo *backend*
- Desarrollo *frontend*
- Desarrollo de la solución de *blockchain*
- Revisiones y otras actividades de calidad

Documentación

- Consolidado de documentos y referencias
- Revisiones de calidad
- Seguimiento de costos y esfuerzos
- Rendimiento

Cierre del proyecto

- Análisis final
- Lista de deficiencias del proyecto
- Informe

8.2 Etapa de investigación

8.2.1 Aplicación de Kanban

Como se dijo en la sección 4.4.1, utilizamos *Kanban* como metodología de trabajo ágil durante esta etapa.

Dado que en esta etapa de investigación no aplican o no aportan mucho al equipo ni a la investigación las reuniones obligatorias, esta metodología nos brinda una mayor flexibilidad con respecto a esto a diferencia de otras, como por ejemplo *Scrum*.

Una comparativa en detalle entre *Kanban* y *Scrum* puede observarse en el Anexo 13.4.2.

Kanbanize

Kanbanize ¹ se define como una herramienta para gestionar los tableros *Kanban* de uno o varios proyectos.

¹<https://kanbanize.com/es/>

En nuestro caso decidimos utilizar esta herramienta ya que nos ofrece un mayor control sobre las tareas en proceso de investigación y una visualización mucho mas eficiente de las mismas.

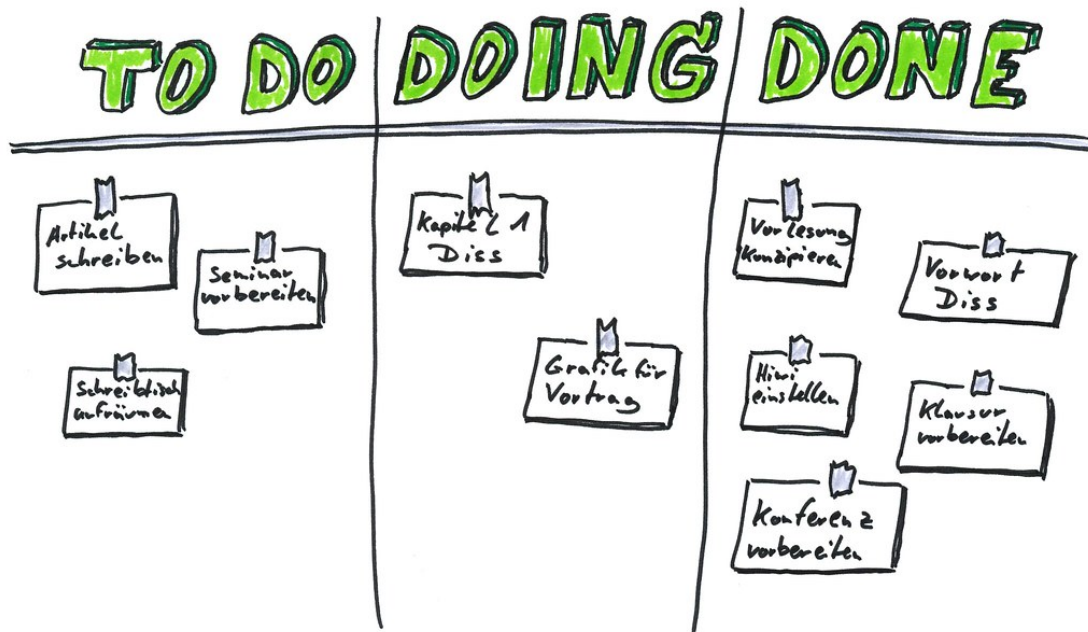


Figura 8.3: Tablero Kanban [13]

De esta forma cada integrante del proyecto, puede ejecutar tareas que se encuentran en *To do*, pasándolas a *Doing* y cuando las terminan pasan a *Done*.

Por otro lado, nos permite definir relaciones entre las tareas y a su vez definir subtareas dentro de cada una, lo cual fue un diferencial con respecto a otras herramientas similares como por ejemplo *Trello* [8].

En su modalidad paga ofrece a su vez la posibilidad de incluir andariveles por cada tablero de *Kanban*, lo que nos ayudó a que cada integrante del equipo trabajara sobre su propio tablero de investigación en una primera fase, para finalmente unificar todas las tareas en el andarivel principal y trabajar sobre el mismo.

Esto último fue acordado por el equipo durante la primera *Kanban retrospective*, lo cual se explica con mayor detalle en el siguiente apartado.

Kanban retrospective

Si bien la metodología *Kanban* brinda una mayor flexibilidad en cuanto a las reuniones, permitiéndonos optar por ellas, el equipo considero importante realizar una reunión cada dos semanas con el objetivo de realizar una puesta en común de todo lo investigado por cada integrante durante ese tiempo. Esta reunión nos permite:

- Llevar un control sobre el avance de la investigación
- Discutir e intercambiar ideas de lo aprendido
- Mejorar el proceso de investigación
- Definir que investigar y como proceder durante las siguientes dos semanas

8.2.2 Popcorn flow

Durante la etapa de investigación surgieron varias dudas en el equipo. Dado que ningún miembro del mismo tenia conocimientos sobre el área de *blockchain* la mayor incógnita al momento de comenzar con esta primera etapa fue que investigar, que alcance tener y con que profundidad.

Es por eso que el equipo se vio en la necesidad de utilizar una metodología existente para ayudarnos con este proceso. En base a estas dudas, el tutor Darío Macchi nos recomendó la metodología *Popcorn flow*.

Popcorn flow [64] se define como la evolución continua a través de la experimentación ultrarrápida. La siguiente imagen ilustra el flujo completo de dicha metodología.

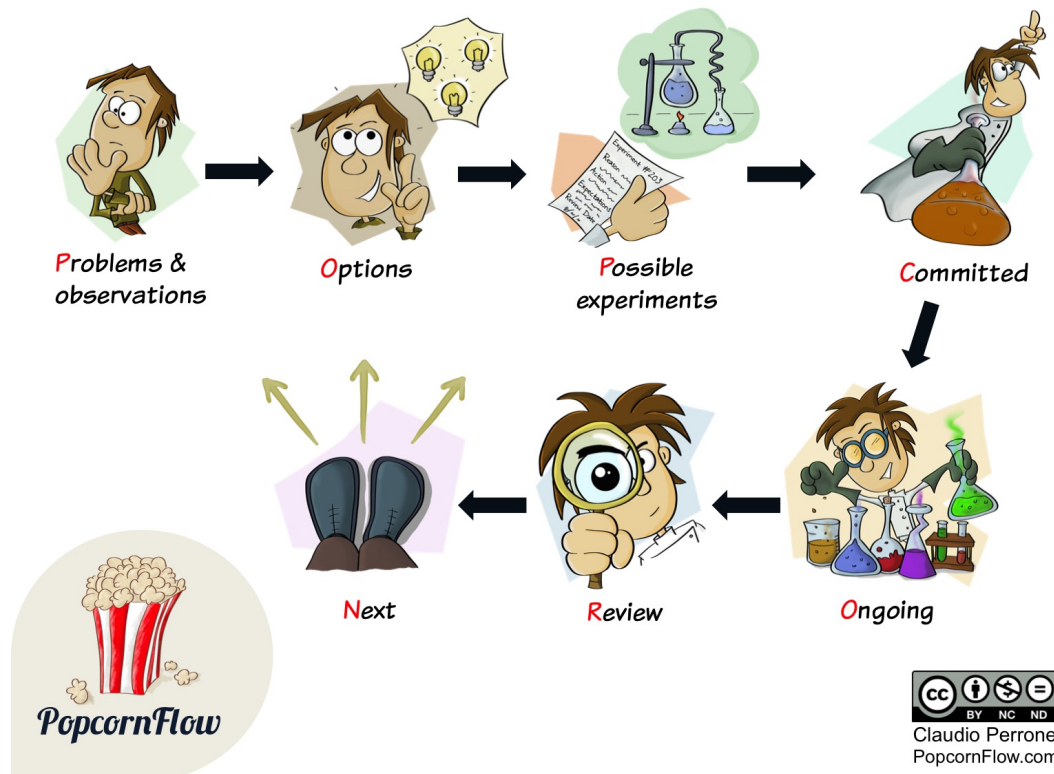


Figura 8.4: Etapas del método Popcorn flow [64]

La metodología cuenta con los siguientes pasos:

- **Problems & observations:** Lo primero que debemos hacer es encontrar problemas y observaciones al problema que queremos investigar.
- **Options:** Luego generar opciones para resolver dichos problemas u observaciones.
- **Possible experiments:** Luego establecer posibles experimentos para esas opciones.
- **Committed:** Tomar uno de esos experimentos establecidos y comprometerse a realizarlo.
- **Ongoing:** Efectuar esos experimentos.
- **Review:** Revisar y evaluar los resultados obtenidos.

- **Next:** Si en base a ese experimento efectuado, encontramos nuevos problemas u observaciones, nuevas opciones u otros posibles experimentos. Volvemos a iterar sobre la metodología.

Esta iteración termina cuando resolvemos los experimentos planteados.

Utilizamos esta metodología en conjunto con *Kanban* definiendo para cada columna del tablero de *Kanban* un paso de la metodología *Popcorn flow*, en donde cada miembro del equipo registra tareas en la columna *Problems & observations* y luego las va moviendo entre las columnas a medida que avanza la metodología.

Se pueden observar capturas de esto en funcionamiento en el Anexo 13.4.3.

8.2.3 Entregable

Con el fin de documentar las decisiones tomadas a nivel de las distintas plataformas de *blockchain*, hicimos un documento con los resultados de la etapa de investigación junto con la elección de la plataforma, en nuestro caso *Ethereum* por las bondades que nos brinda.

Este entregable se puede apreciar en el Anexo 13.7.

8.3 Etapa de desarrollo

Es la segunda etapa del proyecto. El objetivo de ella es construir el producto de software utilizando los conocimientos adquiridos en la etapa anterior. Por las notorias diferencias en la naturaleza de las etapas del proyecto el equipo optó por continuar utilizando una metodología ágil pero con otra implementación.

Temporalmente este periodo del proyecto comienza a fines de enero de 2019 y finaliza en las últimas semanas de julio. Es importante destacar que si bien el principio de esta etapa es en enero, el equipo hará una transición a lo largo de un mes. Es decir que durante el mes de enero de 2019 los integrantes del equipo comenzarán a dejar de investigar y capacitarse para comenzar a construir el producto de software deseado.

8.3.1 Implementación de Scrum

Como se mencionó en el apartado 4.4.2, decidimos utilizar *Scrum* como metodología de trabajo. Al ser una metodología ágil se centra en obtener software funcional sobre documentación, lo que nos permite centrarnos en obtener un producto final funcional que cumpla los requerimientos del cliente.

Al contar con la *daily meeting* podemos sacarle mas provecho al tiempo permitiéndonos reaccionar ante desvíos rápidamente, encontrarle soluciones a problemas ni bien aparecen y en consecuencia aportar cuanto valor sea posible al cliente.

Por otro lado, al finalizar cada iteración contamos con una instancia programada para que el cliente confirme que se está construyendo un producto que cumple con lo que necesita, si hay algún requerimiento o punto a mejorar. De esta forma evitamos que el cliente se lleve sorpresas negativas con el producto final entregado luego de esta etapa. Otro punto a favor de esta práctica es que nos permite tomar cualquier solicitud de mejora que el cliente sienta es necesaria para estimarla e incorporarla a la solución rápidamente.

Con estos puntos en mente concluimos que debíamos de utilizar *Scrum* como metodología para la etapa de desarrollo. Optamos por tener *sprints* de dos semanas por lo cual contamos con trece *sprints* para construir el producto deseado.

8.3.1.1 Roles

Una vez definido que íbamos a utilizar *Scrum* para llevar a cabo esta etapa del proyecto, debimos de asignar los roles competentes a esta etapa, es decir los roles transversales a todo proceso de *software* (gestión de riesgos, arquitecto, gestión de calidad, gestión de la configuración y gestión de proyecto) y los roles relativos a *Scrum* (*Scrum master* y *product owner*). Agregado a estos roles bajo recomendación del tutor se decidió incorporar un rol de *proxy-product-owner* que fuera un rol rotativo y se encargará de llevar a cabo las comunicaciones con el *product owner*.

La designación de roles de *scrum* fue la siguiente:

- **Product owner:** Tomás Piaggio
- **Scrum master:** Camilo Martínez
- **Proxy-product-owner [39](inicialmente):** Agustín Caétano
- **Equipo:** Camilo Martínez, Agustín Caétano, Ismael Puricelli y Juan Rodríguez

Queremos detenernos en el rol de *Proxy-product-owner*, ya que entendemos que se distancia de una implementación cotidiana de *Scrum*. El principal papel de este rol fue el de canalizar las interacciones con el *product owner*, organizar las reuniones e instancias de intercambio.

Mientras que los roles transversales fueron divididos de la siguiente manera:

- **Arquitecto:** Agustín Caétano
- **Gestión de calidad:** Juan Rodríguez
- **Gestión de la configuración:** Ismael Puricelli
- **Gestión de riesgo:** Camilo Martínez
- **Gestión del proyecto:** Agustín Caétano, Camilo Martínez, Ismael Puricelli y Juan Rodríguez

8.3.2 Planificación

Para comenzar con esta sección, es necesario recalcar que los requerimientos a realizar en cada *sprint* fueron expresados en base a *user stories*, que tienen como objetivo representar una funcionalidad de forma atómica y entendible. Excepcionalmente también fueron incluidas tareas para solucionar algún *bug* o configurar algún producto de *software*.

Para escribir las *user stories* utilizamos el siguiente formato: *Como [actor] me gustaría que el sistema/aplicación cuente con [característica] para cumplir/lograr [objetivo]*.

Al comienzo del proyecto se definió el *product backlog* a partir de un conjunto de funcionalidades deseadas por el cliente. Una vez establecido este artefacto el equipo pudo conocer el alcance del proyecto trazando así una meta, agregado a esto el *backlog* nos sirvió de guía a lo largo del proyecto permitiéndonos saber cuantas funcionalidades quedaban por completar.

Si bien se obtuvo el *product backlog* al comienzo, es claro que al tratarse de una estimación este fue sufriendo modificaciones a lo largo del proyecto.

Sprint planning meetings

Para llevar a cabo la planificación de cada iteración utilizamos la ceremonia propuesta por *Scrum*, la *sprint planning*. En primera instancia se estudia el *product backlog* para definir que conjunto de tareas serán las que el equipo va a afrontar en este sprint constituyendo de esta manera el *sprint backlog*.

Una vez definido el mismo se procede a descomponerlos en tareas afrontables para luego realizar una estimación, determinando de esa manera que va a ser logrado dentro del sprint que comienza. Agregado a esto es la ceremonia que da comienzo al sprint.

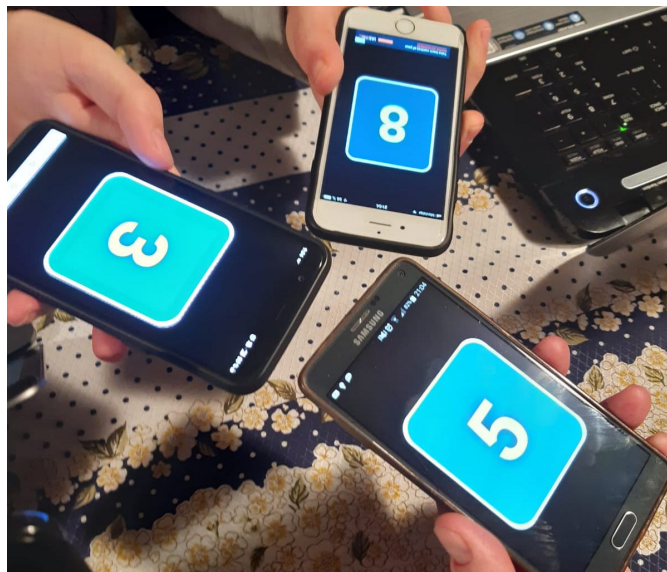


Figura 8.5: Estimación utilizando *poker planning*

Para llevar a cabo la estimación el equipo utilizó la estrategia *planning poker* con la escala de *Fibonacci*. Esta estrategia nos permitió conocer la percepción de cada uno de los integrantes del equipo sobre el desafío que implicaba cada tarea, una vez compartida esa percepción se comenzaba a discutir para llegar a un acuerdo sobre el valor asignado a cada tarea.

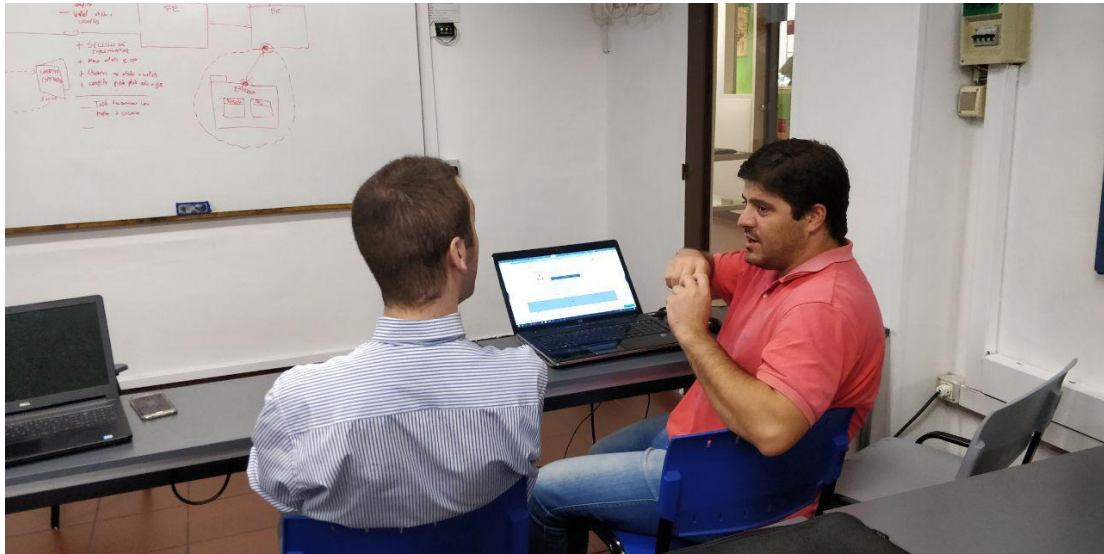


Figura 8.6: *Planning*

La planificación del trabajo sprint a sprint fue de vital importancia para comprometernos a realizar la cantidad justa de tareas, evitando tomar desafíos muy grandes para ser completados en un solo sprint, y sin volverse lo suficiente laxo como para no cumplir con el alcance deseado.

Product backlog

A continuación se detallan las épicas e historias de usuario utilizadas a lo largo del proyecto. Una vez construido, el equipo pudo comenzar a realizar estimaciones para definir el tamaño de las diferentes funcionalidades y luego comenzar su construcción.

Tabla 8.1: *Product Backlog*

Épica	Historia de usuario	Prioridad	SP ²
Registro de usuarios	Como usuario me gustaría tener la posibilidad de registrarme en el sistema proveyendo credenciales para posteriormente poder iniciar sesión en la plataforma y acceder a sus servicios	Prescindible	5
	Como usuario me gustaría tener la posibilidad de registrarme en el sistema mediante redes sociales para posteriormente poder iniciar sesión utilizando dichas credenciales en la plataforma y acceder a sus servicio	Imprescindible	13
Inicio de Sesión	Como usuario quiero iniciar sesión para acceder al sistema	Imprescindible	5
Perfil de usuario	Como usuario me gustaría poder crear y editar mi perfil personal para que otros usuarios de la plataforma puedan reconocerme	Imprescindible	5
Publicaciones de usuarios	Como usuario me gustaría poder publicar un servicio en la plataforma para que otros usuarios puedan visualizarlo y eventualmente contratarlo	Imprescindible	5
	Como usuario de la aplicación me gustaría poder visualizar todos los servicios registrados en la plataforma por otros usuarios para poder eventualmente contratar uno	Imprescindible	3

²Estimación en *Story Points*

	Como cliente quiero poder realizar consultas generales sobre los servicios listados en la aplicación	Deseable	8
Contactar y negociar intercambio	Como usuario me gustaría poder ponerme en contacto con el usuario proveedor de un determinado servicio y negociar con él mediante una ventana de <i>chat</i> con el fin de llegar a un acuerdo y contratar su servicio	Imprescindible	13
Pagos de servicios	Como usuario quiero poder pagar por un servicio mediante <i>coopis</i> o trueque con el fin de completar la compra y poder recibir el mismo	Imprescindible	21
	Como usuario quiero poder dividir el pago de un servicio entre varios usuarios de la plataforma con el fin de comprar un servicio de manera colectiva	Imprescindible	21
Seguridad y <i>confiabilidad</i> de las transacciones	Como cliente quiero que las transacciones realizadas dentro de la plataforma se ejecuten mediante <i>block-chain</i> utilizando la plataforma <i>Ethereum</i>	Imprescindible	21
Reseña de servicio recibido	Como usuario quiero poder realizar una reseña de un servicio recibido en base a un sistema de puntuación de estrellas y comentarios con el fin de dejar mi opinión visible a otros usuarios del sistema sobre el servicio provisto	Imprescindible	8
	Como usuario quiero poder visualizar las reseñas de los distintos usuarios de la plataforma con el fin de poder tener un anticipo de la calidad del servicio antes de contratarlo	Imprescindible	5

Obtención de <i>co-opis</i> por actividades extra	Como usuario quiero poder recibir recompensas por tareas que enriquezcan a la aplicación, por ejemplo: registrarse, invitar a un usuario, compartir un servicio en redes sociales, etc	Imprescindible	34
Usabilidad de la aplicación	Como cliente me gustaría que la aplicación se pueda utilizar de manera sencilla e intuitiva, teniendo una buena <i>UX</i> , con el fin de facilitar y agilizar el uso de la misma	Imprescindible	21
Modificabilidad de la aplicación	Como cliente quiero que la aplicación desarrollada sea fácilmente modificable y extensible a modo de poder agregar nuevas funcionalidades a futuro de una manera sencilla sin afectar lo ya desarrollado	Imprescindible	34
<i>PWA</i>	Como cliente quiero que la aplicación desarrollada cumpla con los estándares planteados por <i>PWA</i>	Imprescindible	21

8.3.3 Ejecución

Aquí se describen las actividades realizadas a lo largo del *sprint*. A medida que el *sprint* avanzaba, el equipo registraba el avance. A continuación se describe como ocurría el registro y control en conjunto con los artefactos de *Scrum* que se seleccionaron para la ejecución.

Teamwork

Para llevar la gestión de esta etapa del proyecto el equipo decidió incorporar una herramienta orientada a la gestión de proyectos. En ella se encontraba el *backlog* priorizado que era utilizado al comienzo de cada sprint para definir al alcance en el mismo.

Luego de definido el alcance, desglosado en tareas y estimado en esfuerzo se designaba un responsable encargado de llevar la tarea adelante. Al finalizar la tarea asignada el responsable debía de marcarla como completada. Se puede observar la evidencia del uso de esta herramienta en el anexo 13.4.4.

En cuanto al registro de horas, al estar acostumbrados a utilizar la herramienta *Toggl* para la gestión del tiempo se decidió continuar utilizándola para esta etapa.

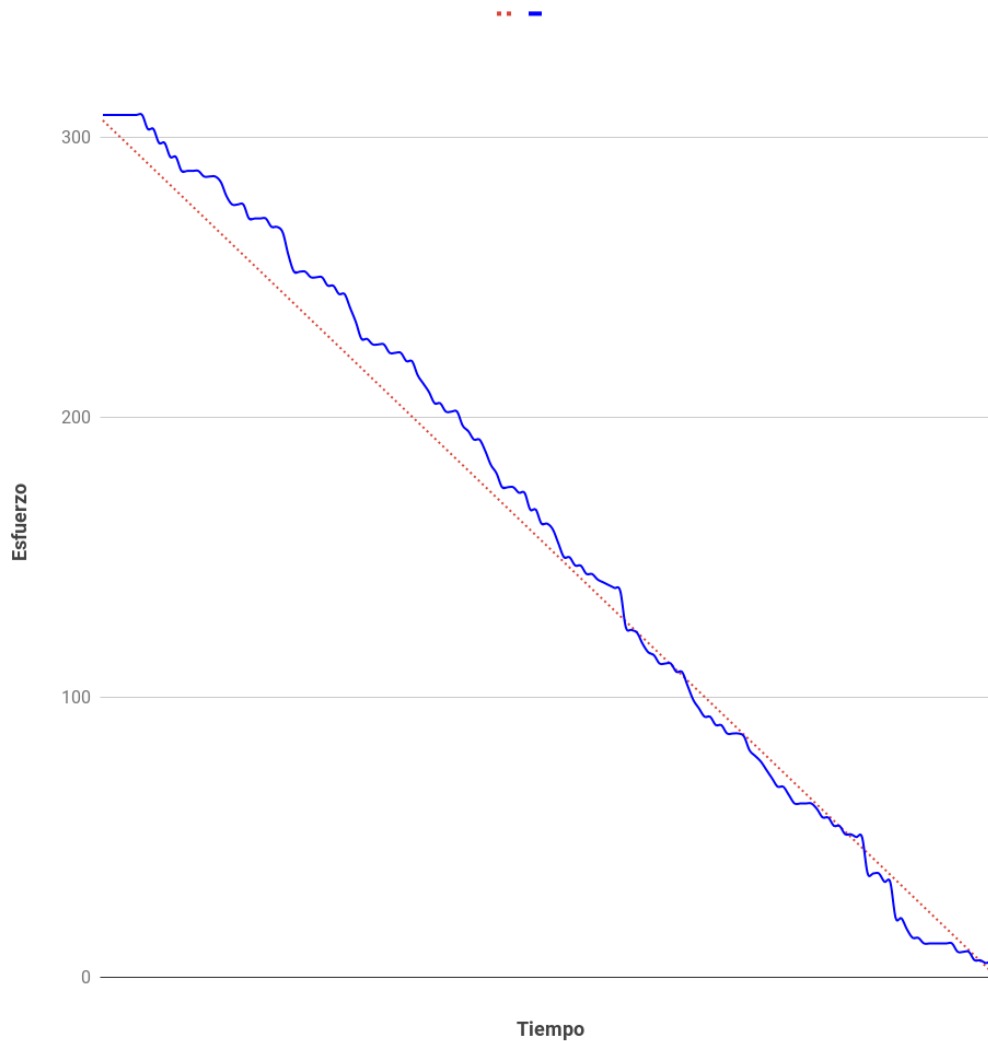
Daily meeting

En esta ceremonia cada integrante del equipo comentaba brevemente en que había estado trabajando en el día anterior, dificultades a las que enfrentó y que solución ideó, en caso de haberlo solucionado. Esta es una instancia vital para encontrar desvíos y solucionarlos. Esta reunión se podía dar en dos modalidades; presenciales o remotas. En caso de ser remotas se daba intercambiando mensajes en alguno de los medios de intercambios utilizados por el equipo.

Registro de avance

Para llevar constancia del grado de avance logrado por el equipo se llevo un registro de horas utilizando la plataforma *Toggl*, agregado a esto también llevamos el registro de las tareas completadas en cuanto a puntos de esfuerzo, para ello marcamos como terminada la tarea en la plataforma *Teamwork*.

Sprint burndown chart



Del gráfico mostrado previamente podemos destacar los siguientes puntos:

- El equipo comenzó lento, principalmente por el desconocimiento de tecnologías y metodologías. Conforme avanzaron los *sprints* el mismo tendió a alinearse a la línea de tendencia.
- Luego de ganar experiencia con las tecnologías y metodologías el equipo logró sentirse a gusto y como consecuencia logramos compensar la diferencia

inicial entre el esfuerzo entregado y el ideal. Agregado a esto en este momento el equipo ya había finalizado gran parte de la integración con *Ethereum*, disminuyendo de esta manera la incertidumbre generada por interactuar con ella.

- Para finalizar el equipo demostró haber entendido y saber utilizar las tecnologías y metodologías planteadas, en consecuencia podemos ver que el esfuerzo está por debajo de la línea de tendencia. Sin embargo el equipo encontró un problema técnico que no pudo solventar con rapidez lo que tuvo como consecuencia que el esfuerzo real se aproxime a dicha línea de tendencia en el tramo final.

Reuniones de seguimiento con tutor

Otra instancia para controlar nuestro progreso a lo largo del proyecto fueron las reuniones periódicas con el tutor. Estas instancias se daban cada dos semanas, con una semana de diferencia respecto al inicio del sprint, de esa manera siempre contábamos con alguna inquietud o inconveniente a resolver y se le podía sacar el máximo provecho a esas instancias.

En esas reuniones normalmente se trataban problemas del presente sprint de toda índole, relacionados a investigación, tecnología, gestión del proyecto, problemas internos, etc. En varias instancias la experiencia y perspectiva externa del tutor nos permitió encontrar soluciones que de otra manera no hubieran sido alcanzadas.

8.3.4 Evaluación

Aquí se describen las actividades realizadas para dar cierre al *sprint*. Principalmente llevamos a cabo tres tareas:

- *Sprint reviews*
- *Sprint retrospective*
- Actualización de documentos intermedios de gestión

Es una etapa crucial del proceso que nos permite detectar desvíos y encontrar posibles oportunidades de mejora del proceso.

Sprint reviews

La *sprint review* es la ceremonia en la que se muestra el avance obtenido hasta el momento haciendo foco en las últimas funcionalidades que han sido incorporadas o editadas.

En un principio tratamos de organizar estas instancias para que sean presenciales y poder aprovecharlas al máximo, sin embargo se demostró muy complicado conseguir que los horarios del equipo y cliente coincidieran por lo que se recurrió a realizarlas de manera remota.

Fue una instancia sumamente importante a lo largo del proyecto ya que nos permitió validar el progreso del mismo hasta el momento, conocer la opinión del cliente del producto logrado y detectar posibles oportunidades de mejora al producto y/o proceso.

Agregado a esto, el equipo aprovechó esta instancia para confirmar las expectativas del cliente para el siguiente *sprint*, eso nos permitió mantener las funcionalidades más importantes para la siguiente iteración, definiendo y/o confirmando el alcance del siguiente *sprint*.

Al finalizar la última *review* se compartió la última versión del código fuente en ejecución en un ambiente remoto, así como las credenciales de los usuarios de prueba utilizados para demostrar el comportamiento alcanzado.



Figura 8.7: Instancia de revisión con el cliente

Sprint retrospective

Luego de realizada la *review*, procedíamos a realizar la retrospectiva de esa iteración. Tal cual como plantea *Scrum*, en esta instancia el equipo tiene la posibilidad de retraerse en el tiempo para encontrar mejoras o puntos a discutir que no han sido conversados en su momento.

Entendiendo la importancia y, por lo tanto, el posible impacto de esta actividad el equipo creyó conveniente reunirse para llevarlas a cabo de manera presencial.



Figura 8.8: Evidencia de *Sprint retrospective*

A lo largo del proyecto se utilizaron múltiples metodologías para llevar a cabo esta ceremonia, todas ellas con el objetivo de encontrar nuevas oportunidades de mejora, puntos débiles a mejorar y nuevas practicas que nos ayuden a evitar desviarnos de lo estimado. En algunas de estas metodologías, luego de que cada integrante del equipo exponía su sentir e ideas, se elaboraba un plan de acción. Dicho plan debía de cumplirse antes de finalizar el siguiente *sprint*, para ello se le asignaba un responsable encargado de velar por el cumplimiento del plan ideado.

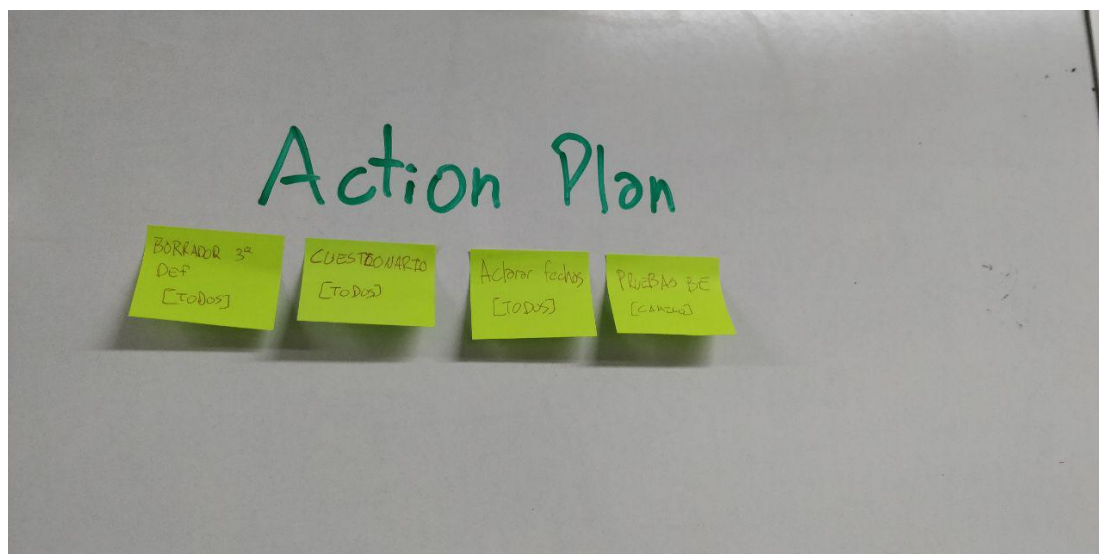


Figura 8.9: *Action plan* construido en base a la *sprint retrospective*

8.4 Análisis de resultados obtenidos

En esta sección se van a examinar los resultados obtenidos de la gestión en ambas etapas del proyecto. Comenzaremos hablando de la etapa de investigación y luego procederemos a la etapa de desarrollo.

8.4.1 Investigación

Dada la metodología utilizada, inicialmente, nos planteamos utilizar las siguientes métricas para evaluar nuestra velocidad a lo largo del proceso. [111]

- Capacidad (*Throughput*): Nos indica cuantas tarjetas el equipo logró entregar en un tiempo definido, usualmente una iteración de una o dos semanas. Nos sirve para saber cuantas tarjetas han sido procesadas, que tan activo está el equipo, etc.
- Procesamiento (*Responsiveness*): Nos indica cuanto tiempo promedio una tarjeta necesita para dar una vuelta por todos los andariveles planteados por *Kanban*.

Sin embargo luego de conversar con Darío respecto a nuestra selección de métricas a analizar, nos sugirió utilizar un enfoque más retrospectivo y autocrítico para

analizar nuestros puntos débiles y fuertes en esta etapa. Esa sugerencia se justifica considerando que el objetivo de esta etapa, principalmente, era investigar tecnologías emergentes o recientes. Eso implica dificultades a la hora de determinar los resultados obtenidos y trasladarlos a futuras instancias de investigación, agregado a esto, al tratarse de una actividad de investigación, el tiempo que se le puede llegar a dedicar a cada tecnología investigada puede variar enormemente dependiendo de la persona y tecnología. Teniendo esto en cuenta no realizamos un análisis de métricas, sino que utilizamos nuestro criterio para realizar un análisis retrospectivo de esta etapa.

Lecciones aprendidas

Teniendo en cuenta los resultados obtenidos pudimos encontrar múltiples puntos de mejora a futuro, entre los principales encontramos:

- **Mala inversión de los tiempos:** Estimamos un tiempo demasiado grande para estudiar múltiples tecnologías a un nivel que no terminamos logrando y hubiera sido mas conveniente utilizar un tiempo mas corto para lograr una capacitación mas básica, ganar experiencia en los primeros *sprints* y luego tomar algunos periodos para profundizar el conocimiento cuando fuera necesario.
- **Falta de instancias colaborativas y falta de comunicación interna:** Quizá fue por el poco conocimiento que teníamos entre los integrantes del grupo de las formas de trabajar de cada uno. Eso incurrió en que el sincronizar lo que cada uno aprendía (compartir el conocimiento) se volvió un proceso engorroso. Respecto a este punto aprendimos a comunicarnos de manera mas eficiente e inteligente con nuestros pares lo cual es algo que puede parecer trivial pero no lo es.
- **Exceso de herramienta/metodología y uso poco consistente de la misma:** Una vez finalizada la etapa de investigación y con un buen tramo del desarrollo completado el equipo fue lo suficientemente maduro para determinar que se había realizado un error importante en la investigación. Si bien se construyeron pruebas de concepto sobre una diversidad de elementos que terminaron formando parte del producto final, se podría haber sido más auto exigente en esta etapa, lo que hubiera generado mejores resultados.

En conclusión si bien creemos que el resultado final de la etapa de investigación fue

bueno, el mismo resultado se podría haber conseguido en una fracción del tiempo de haber sido más pro activos y haber afrontado los desafíos de una forma más escalonada o progresiva.

8.4.2 Desarrollo

Dada la metodología utilizada en esta etapa utilizamos las siguientes métricas para evaluar nuestra velocidad a lo largo del proceso.

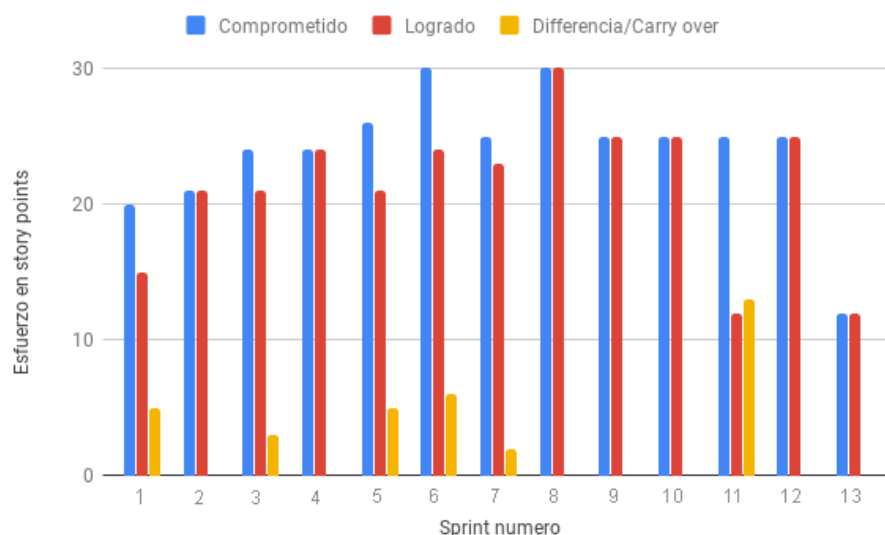
- Velocidad
- Esfuerzo
- Tiempo (horas) invertido

Exposición de métricas relevantes

Esfuerzo

A continuación se puede observar la gráfica comparando el esfuerzo comprometido, logrado y sobrannte obtenida a lo largo del proyecto *sprint* a *sprint*. Esto se obtuvo como resultado directo de la estimación de los integrantes del equipo.

Nos da una medida de cuanto trabajo realizó el equipo a lo largo del proyecto.



De la gráfica podemos interpretar que el equipo enfrentó desafíos cada vez más grandes hasta llegar a la mitad del proyecto donde hay un evidente pico de actividad. Luego los desafíos fueron disminuyendo de manera casi continua.

Es importante destacar que en los *sprints* 1, 3, 5, 6, 7 y 11 el equipo no completó el esfuerzo estimado. Siendo la ocurrencia más importante de este evento la del *sprint* 11, en dicho *sprint* el equipo no logró completar una tarea de alta complejidad.

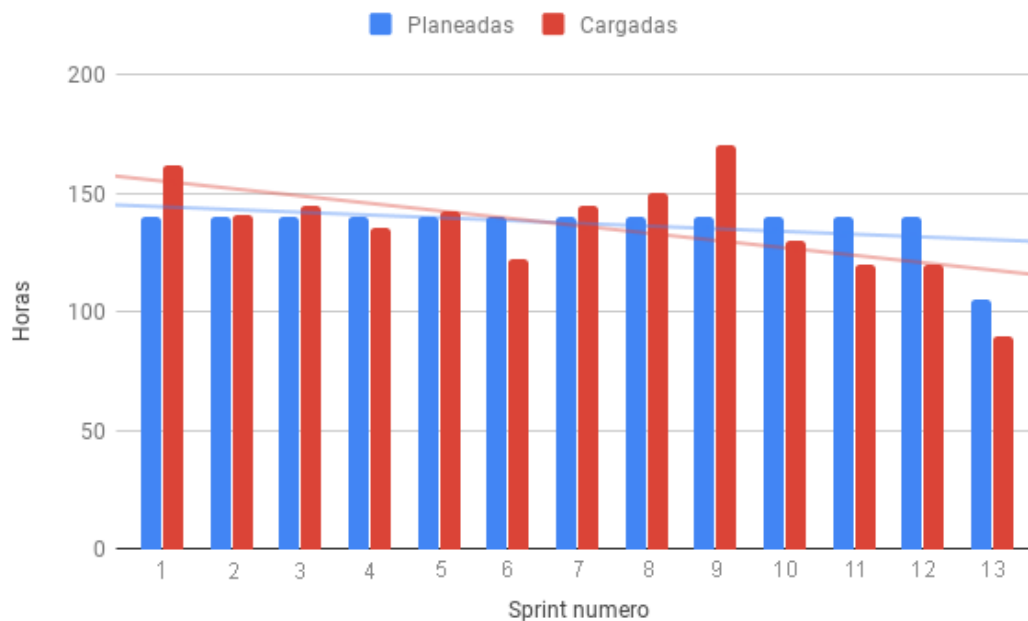
En cada una de estas ocasiones el equipo incorporó el sobrante del *sprint backlog* finalizado al siguiente. Estos elementos contaban con mayor prioridad que las nuevas tareas pues era prioritario culminar con el restante del sprint anterior.

Tiempo invertido

A continuación se puede observar la gráfica comparando las horas estimadas contra las horas realmente trabajadas. Utilizando esta gráfica podemos ver donde fue necesario invertir más tiempo del estimado y realizar un análisis para entender el porque de esa situación. Esto se obtuvo como resultado directo del trabajo llevado a cabo por los integrantes del equipo a lo largo del *sprint*.

Es importante destacar que las horas estimadas no fueron calculadas en función a las tareas a realizar en el *sprint*. En cada sprint este numero se mantuvo constante con excepción del último sprint por hacer una transición con el proceso de documentación.

Nos da una medida de cuanto tiempo se invirtió a lo largo del proyecto y como fue distribuido el mismo.



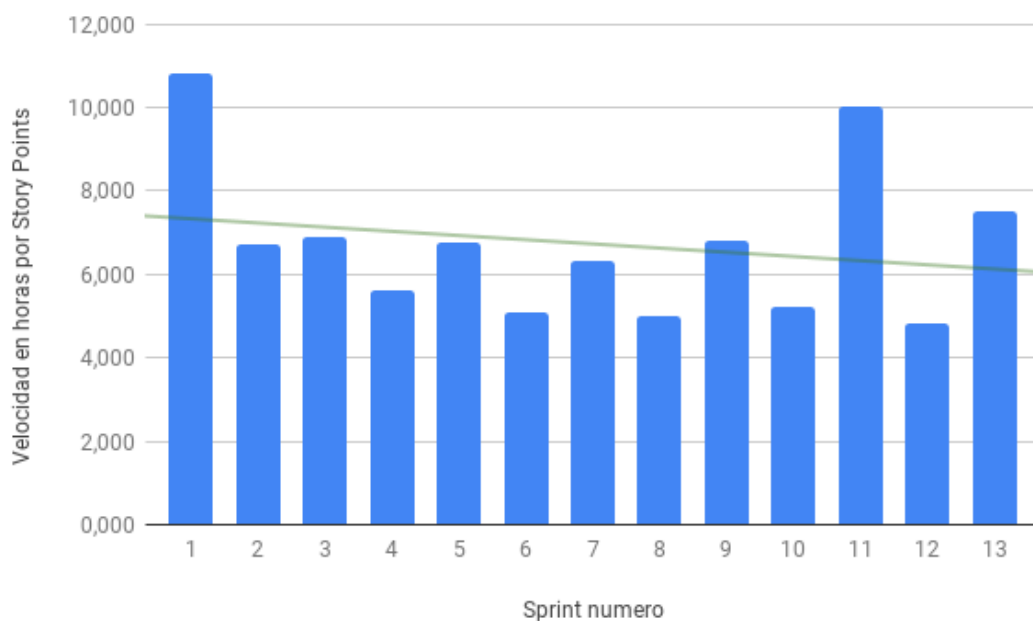
De la gráfica podemos interpretar que el equipo distribuyo su tiempo de acuerdo a lo estimado en la mayoría de los *sprints*, con excepción del 10. En ese *sprint* el equipo decidió invertir un gran esfuerzo en mejorar la usabilidad de la aplicación, entendiendo que estábamos por debajo de los aceptable. Como consecuencia se puede ver que las horas invertidas superan a las estimadas por un buen margen.

En estas gráficas se puede apreciar además de la comparación *sprint* a *sprint* del tiempo invertido y el tiempo estimado, las líneas de tendencia correspondientes.

Velocidad

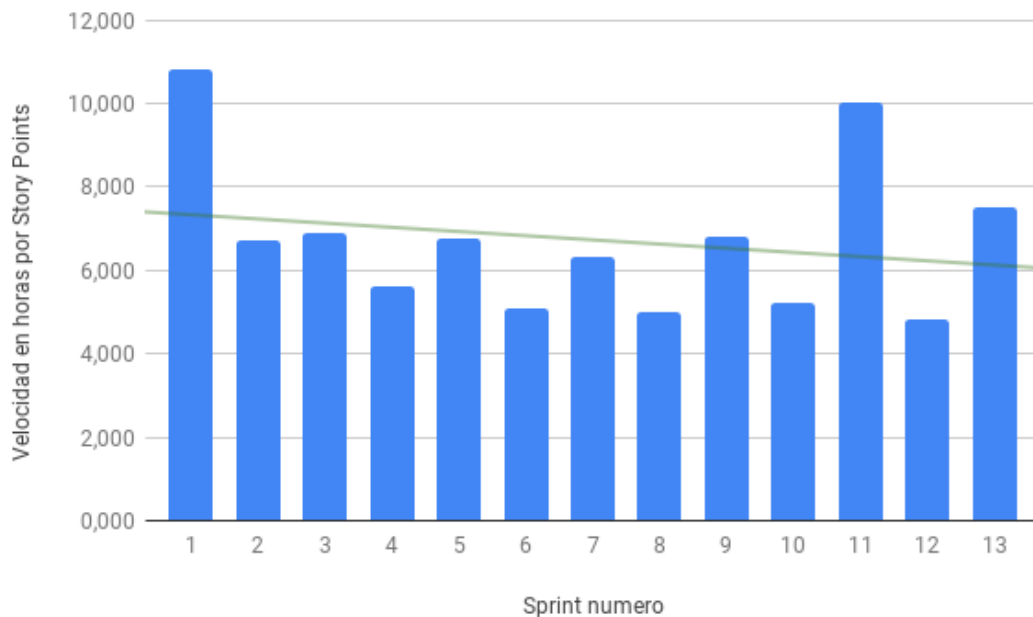
A continuación se puede observar la gráfica de velocidad obtenida a lo largo del proyecto *sprint* a *sprint*. Para eso simplemente medimos cuantos *story points* logramos alcanzar en cada *sprint*.

Nos da una medida de cuanto trabajo realizó el equipo a lo largo del proyecto.



De la gráfica podemos interpretar que el equipo enfrentó desafíos cada vez más grandes hasta llegar a los últimos *sprints* de desarrollo. En esa etapa el equipo fue enfrentando una serie de desafíos que llevaron a que se disminuya la cantidad de *story points* logrados *sprint* a *sprint*. Entre los retos que incidieron en este cambio encontramos: fatiga por la continua ejecución del proyecto, construir una solución cada vez más grande, etc.

A continuación se puede observar otra interpretación gráfica sobre la distribución del esfuerzo a lo largo del proyecto. Creímos que sería interesante saber cuanto tiempo era necesario que el equipo invirtiera para completar un *story point* en cada *sprint*.



Vemos que la gráfica se encuentra bastante uniforme, a excepción de los *sprints* 1 y 11. En esos *sprints* hay un pico en la cantidad de horas invertidas para lograr un *story point*. Para el primer *sprint* puede ser algo más normal teniendo la falta de práctica del equipo con las tecnologías y metodologías. En el *sprint* 11 se dio que el equipo no logró completar una tarea que impidió dar como completada una funcionalidad bastante compleja (estimada en 13 *story points* por el equipo), eso conllevó que no se completarán muchos puntos del *sprint* y como la gráfica utiliza los *story points* completados la gráfica se ve afectada.

8.4.2.1 Lecciones aprendidas

Teniendo en cuenta los resultados obtenidos pudimos encontrar múltiples puntos de mejora a futuro, entre los principales encontramos:

- Costo de calidad: Si bien el equipo aplicó un plan de calidad, algunas de las actividades fueron incorporadas a lo largo del proyecto. En consecuencia, surgieron algunos desvíos y problemas que nos impidieron cumplir con lo acordado en algunas instancias.
- Consecuencias a utilizar plantillas de código: Si bien fue una decisión que simplificó gran parte de los problemas relacionados a una épica (*PWA*), el

uso de plantillas acarreo múltiples problemas que nos afectaron lo largo del proyecto. A futuro, en caso de decir utilizar una, invertir mas tiempo en investigar y experimentar previamente con ella nos ahorraría problemas a futuro.

- *Dailies* y su registro Si bien el equipo utilizó la metodología *Scrum*, no todas sus ceremonias fueron aprovechadas la máximo, concretamente creemos que se podría haber explotado más las *daily meetings*. Otro punto donde se podría haber mejorado en esta ceremonia es en el registro de la misma, si bien contábamos con un canal dedicado a eso no se lo utilizó asiduamente.

8.5 Gestión de riesgos

En esta sección se muestra la gestión de riesgos del proyecto. Desde la estrategia utilizada, la identificación de los mismos y por último el plan de acción a realizar. Cabe destacar que dado el contexto del proyecto, un contexto cambiante, esta gestión fue clave para llevar el proyecto de la mejor manera posible.

El objetivo principal de esta gestión fue evitar que los riesgos se manifiesten y generen problemas inesperados. Para ello se elaboró un plan de mitigación, y en caso de que el riesgo de manifestará se contó con un plan de respuesta.

Se presenta a continuación la estrategia utilizada gestionar los riesgos con su descripción y evolución a lo largo del proyecto.

8.5.1 Estrategia y categorización

A continuación se pueden encontrar los riesgos detectados a lo largo del proyecto junto a un identificador que será utilizado en las representaciones gráficas evolutivas de los mismos.

Una vez identificados los riesgos realizamos un estudio del impacto y la probabilidad de manifestación de cada uno. Encontrar un método para poder determinar cada una de estas magnitudes de manera objetiva fue por demás complejo. Teniendo esto en cuenta optamos por utilizar nuestra experiencia previa para medir de manera subjetiva ambas características para cada uno de los riesgos. Para ver el detalle de dicho estudio ir al anexo 13.4.5.

Dado que no contábamos con una herramienta objetiva para llevar a cabo la estimación de impacto, mantener actualizada la magnitud y los planes relativos a cada riesgo jugó un rol vital. Teniendo eso en cuenta decidimos actualizar el plan luego de cada sprint, sin embargo ese enfoque no se demostró compatible con la metodología ágil seleccionada.

Utilizando una recomendación de Álvaro Ortas a partir de la segunda revisión se cambió el enfoque. El mismo pasó a ser que luego de cada planificación se revisan detalladamente los riesgos que aplican en ese sprint revelándose la probabilidad e impacto de cada uno, mientras que los restantes riesgos se revisan rápidamente. En esa instancia se discutía la planificación llevada a cabo el sprint pasado y se actuaba en consecuencia a los aciertos o errores tomados pasados.

8.5.2 Identificación

Para poder evaluar de manera más concreta y sencilla los diferentes riesgos detectados decidimos agrupar y categorizar los riesgos detectados. De acuerdo a lo detectado agrupamos los riesgos en las siguientes categorías

- Equipo
- Infraestructura
- *Ethereum*
- *PWA*
- Gestión
- Código

Equipo

1. Retrasos por desconocimiento de tecnologías: teniendo en cuenta los requerimientos planteado por el cliente los integrantes del equipo debieron de manejar tecnologías que les eran ajenas. Esto puede incurrir en retrasos y/o desvíos.

Como plan de mitigación se decidió complementar la etapa de investigación, con algunas horas de investigación y capacitación en estas tecnologías.

Sin embargo este plan no fue suficiente ya que el riesgo asociado se manifestó en un par de ocasiones (*sprints* 5 y 11), en esas situaciones el plan de contingencia que se ideó consto de dedicarle más tiempo del planificado.

2. Problemas de relacionamiento entre integrantes del equipo: dado que el grupo se conformó utilizando el grupo de *Google* facilitado por la Universidad *ORT*, los integrantes del equipo no contaban con experiencias previas trabajando en conjunto. Eso puede derivar en que las formas de trabajo de los integrantes puedan entrar en conflicto y generar un clima de trabajo adverso.

Como plan de mitigación se decidió hacer especial énfasis en las algunas de las ceremonias de *Scrum* para detectar estos problemas, discutirlos y llegar a una solución.

3. Baja performance de uno o varios de los integrantes del equipo: este riesgo fue el que el equipo asumió en caso de que alguno de los integrantes del equipo no le dedicara al proyecto el tiempo y esfuerzo que el mismo exige.

Como plan de mitigación se decidió hacer especial énfasis en las algunas de las ceremonias de *Scrum* para detectar estos problemas, discutirlos y llegar a una solución.

Lamentablemente el plan de contingencia no fue suficiente ya que en el *sprint* 8 el riesgo se manifestó. Como plan de contingencia el equipo decidió recurrir al tutor en busca de su intervención para subsanar esta situación.

4. Inconvenientes por fallas en la comunicación: este riesgo fue el que el equipo asumió para reflejar los problemas de comunicación entre el equipo o del equipo al cliente y/o tutor. Considerando la falta de experiencia del equipo con las tecnologías y metodologías la comunicación entre las partes tomó mayor importancia.

Como plan de mitigación se decidió tener un rol rotativo “*Proxy product owner*” encargado de realizar llevar a cabo las comunicaciones entre el equipo y cliente.

5. Inconvenientes por falta de practica con las metodologías ágiles: como se ha comentado previamente el equipo no contaba con experiencias previas utilizando metodologías ágiles, eso genera el presente riesgo.

Como plan de mitigación se decidió incluir en la etapa de investigación un conjunto de horas para llevar a cabo una puesta a punto de las mejores practicas por parte del tutor. En esta puesta a punto el equipo llevo a cabo lectura de documentos con buenas practicas y consejos para aprovechar al máximo la/s metodología/s seleccionadas.

Infraestructura

1. Utilizar plataformas para el *hosting* del producto que nos son ajenas: como se comentó previamente el código desarrollado se encuentra “almacenado” en la plataforma *Heroku*. Dicha plataforma nos era ajena el comienzo del proyecto, por lo que la creación y mantenimiento de cada uno de los ambientes donde se ejecutaría el producto de software son la razón de ser del presente riesgo.

Como plan de mitigación se decidió dedicar algunas horas de investigación y experimentación en la etapa de investigación y capacitación.

2. Diferencias de ejecución entre ambiente local y remoto: diferencias entre el código desarrollado por el equipo en su ambiente local y el código presente en *Heroku*. Si bien utilizamos *scripts* para llevar a cabo los despliegues de manera automática y replicable, las instrucciones dentro de ellos causaban que la versión almacenada en *Heroku* falle y no se terminara de desplegar. Este riesgo fue el primero en manifestarse sin estar incluido en nuestras estimaciones iniciales.

Como plan de respuesta se decidió desplegar una versión compilada localmente y revisar la bitácora del ambiente remoto en busca del error.

3. Retrabajo por costos asociados a mantener la solución *hosteada* remotamente: este es otro riesgo que no estaba presente en nuestra estimación inicial, al finalizar el quinto sprint el cliente nos solicito si seria posible realizar un cambio de plan sobre los recursos utilizados para el mantenimiento de la solución en forma remota para abaratar costos.

Como plan de respuesta se decidió destruir el ambiente elaborado en etapas tempranas del proyecto y construir uno similar utilizando recursos gratuitos dentro de *Heroku*.

Ethereum

1. Ataques de mayoría a *Ethereum*: al comenzar el proyecto la plataforma de *blockchain Ethereum classic* sufrió un ataque en el cual un atacante logro hacerse con la mayoría del poder minero de la red. Eso le otorgó un grado de control sobre la red que dejó a la misma vulnerable.

Dado a que este riesgo es un riesgo asociado a la elección de la tecnología *blockchain* a utilizar decidimos asumir este riesgo sin más.

2. Posibles *bugs* nuevos o que nadie enfrento antes: al comenzar el proyecto la interfaz elegida para interactuar con *Ethereum* estaba pasando por un cambio de versión mayor (de la versión 0.X a la 1.0), eso podría haber incurrido en que el equipo tuviera que afrontar *bugs* sin reportar o no solucionado al momento.

Como plan de mitigación se decidió dedicar algunas horas de investigación y comparación entre ambas versiones para ver cual era la mas conveniente utilizar. Agregado a esto se decidió utilizar algunas de estas horas para construir algunos prototipos de los diferentes componentes que integrarían la solución final.

3. No existe tanta documentación sobre *Ethereum* como las otras tecnologías: de la mano del riesgo anterior al tratarse de una tecnología joven, los registros de experiencias previas y documentación de las herramientas utilizadas para interactuar con la plataforma son escasos, eso representó una barrera de entrada a la hora de llevar a cabo nuestra investigación.

Como plan de mitigación se decidió dedicar algunas horas de investigación y comparación entre ambas versiones para ver cual era la mas conveniente utilizar. Agregado a esto se decidió utilizar algunas de estas horas para construir algunos prototipos de los diferentes componentes que integrarían la solución final.

4. Dificultades para desarrollar múltiples partes que deben de interactuar entre si correctamente: la solución propuesta de *Ethereum* cuenta con varias partes que deben de poder comunicarse e intercambiar información. Concretamente se debieron de desarrollar contratos inteligentes, oráculos e incluso un servidor dedicado a la interacción con *Ethereum*. Todo esto debe de poder comunicarse con el servidor central de lógica de negocio.
5. Tiempos de *blockchain*: si bien la descentralización que nos da *Ethereum* es algo muy positivo, el interactuar con este plataforma tiene consecuencias, principalmente en la diferencia de tiempos de ejecución. Esta diferencia se hace evidente a la hora de obtener el saldo de *copies* de los usuarios, para hacerlo por nuestros medios debíamos de comenzar a iterar en cada uno de los bloques registrados y revisar transacción a transacción si fue efectuada desde la dirección de la billetera digital del usuario. Eso no cumplía con los requerimientos de performance ³ requeridos por el cliente.

Para atacar este riesgo decidimos utilizar el servicio *EtherScan* que nos permite obtener un listado de transacciones y el saldo actual de una dirección en tiempos adecuados, es decir que en realidad transferimos este riesgo a un proveedor de servicios especializado en este problema.

PWA

1. Problemas con tecnología que no son tan conocidas aun por el usuario: este riesgo corresponde al hecho de que si bien las *PWA* no son una tecnología reciente, para el usuario común es una tecnología bastante nueva e innovadora. Por lo que contamos con un riesgo de dejarle en claro al usuario que se trata de una *PWA* y no una *App* o una pagina

³Requerimiento no funcional de performance detallado en 5.3.2

Como plan de mitigación se incluyo información especifica para educar a los usuario en como descargarse el producto logrado y entender las diferencias entre una aplicación móvil, una pagina web y una *PWA*.

2. Aspecto similar/único en diferentes plataformas (*iOS, Android, Web*): dado a que la *PWA* será accedida a través de varios dispositivos (ordenadores, dispositivos móviles, etc), surge el riesgo de uniformizar la experiencia ofrecida a nuestros usuarios.

Como plan de mitigación invertimos tiempo de cada sprint para revisar el comportamiento del producto entregado en diversidad pantallas y dispositivos.

3. No tanta documentación sobre las funcionalidad de *PWA* en el *boilerplate*: dada nuestra elección de utilizar una plantilla de código personalizable, debimos de enfrentarnos al problema de que algunos de los comportamientos definidos en ella no eran suficientes y debíamos extenderlos, en respuesta a eso surge este riesgo.

Como plan de mitigación consumimos y utilizamos ávidamente los medios de contacto disponibles por los creadores de la plantilla.

8.5.2.1 Gestión

1. Poca experiencia en desarrollo de proyectos: este riesgo corresponde al hecho de que ninguno de los integrantes del equipo cuenta con experiencia previa en la gestión de proyectos. Esto agrega desafíos a una experiencia que de por si un reto.

Como plan de mitigación la universidad *ORT* nos asigno un tutor (Darío Macchi) para acompañarnos y asesorarnos a lo largo de este proyecto. Recurrir a él conscientemente para evacuar nuestras dudas y buscar ayuda cuando sea necesario fue de vital importancia para mitigar este riesgo.

2. Sobrecarga de trabajo en alguno de los integrantes del equipo: ya sea por el desafío (técnico, humano, etc.) que este proyecto impone es probable que las responsabilidades se distribuyan de maneara desigual, causando que uno o mas de los integrantes equipo se sobrecarguen de trabajo.

Como plan de mitigación se decidió distribuir los roles para que sea claro que labor de llevar a cabo cada uno de los integrantes del equipo.

Sin embargo este plan no fue suficiente ya que algunas áreas fueron más exigentes que otras por lo que se decidió ser transparente y en caso de encontrarse sobre exigido comunicarlo al equipo y encontrar entre todos la solución.

3. Problemas con comunicación - disponibilidad horaria con tutor: este riesgo corresponde a posibles problemas de coordinación y/o comunicación con el tutor a lo largo del proyecto.

Como plan de mitigación se decidió organizar reuniones quincenales y otros canales de comunicación directa con el tutor para mantener el contacto.

4. Alcance demasiado amplio para el tiempo de desarrollo: este riesgo corresponde a que el equipo no pueda entregar un producto final con todas las funcionalidades requeridas por el cliente. Teniendo en cuenta los otros riesgos de inexperiencia con tecnologías y metodologías utilizadas, es una realidad que no puede ser ignorada.

Como plan de mitigación se decidió tener un *backlog* priorizado por parte del cliente y realizar estimaciones de esfuerzo y tiempos coherentes teniendo en cuenta la velocidad del equipo hasta el momento.

Si bien el riesgo no llegó a manifestarse en una de las instancias de revisión se nos aconsejó remover algunas de las funcionalidades deseables con el fin de mejorar la usabilidad de la aplicación. Esa recomendación fue discutida con el cliente quien aprobó ese accionar.

8.5.2.2 Código

1. Consecuencias a utilizar *boilerplates*: este riesgo surge de una idea del equipo de utilizar repositorios con versiones mínimas funcionales de los diferentes componentes como paso inicial del desarrollo. La idea de esta decisión es disminuir el tiempo invertido por el equipo para obtener resultados y entregarle más valor al cliente. Sin embargo esta decisión puede incurrir en múltiples problemas a futuro.

Como plan de mitigación se decidió consultar frecuentemente los repositorios de dichos productos personalizables para estar al tanto de cambios de versiones y evacuar cualquier duda que surja con la comunidad que lo respalda/construye.

2. Enfoque en usabilidad: este riesgo surge del requerimiento del cliente de

obtener un producto de *software* usable y entendible. El cliente no proveyó de diseños o prototipos por lo que recayó en los integrantes del equipo en pensar como se debían dar las interacciones de los usuarios con el sistema.

Como plan de mitigación se decidió consultar con el cliente un flujo de actividad y pantallas al final de cada *sprint* y validarlo para el siguiente, además se estudiaron paginas que proveen funcionalidades similares para detectar buenas practicas y posibles puntos de mejora.

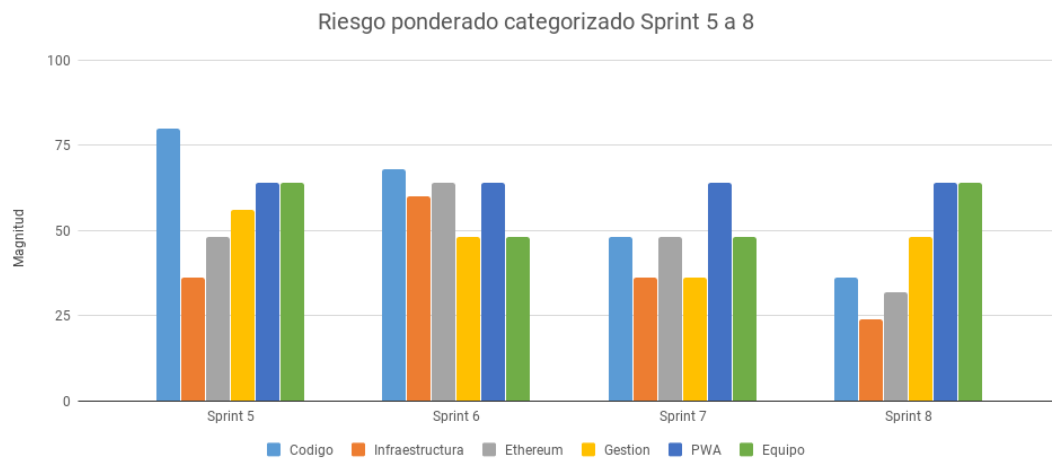
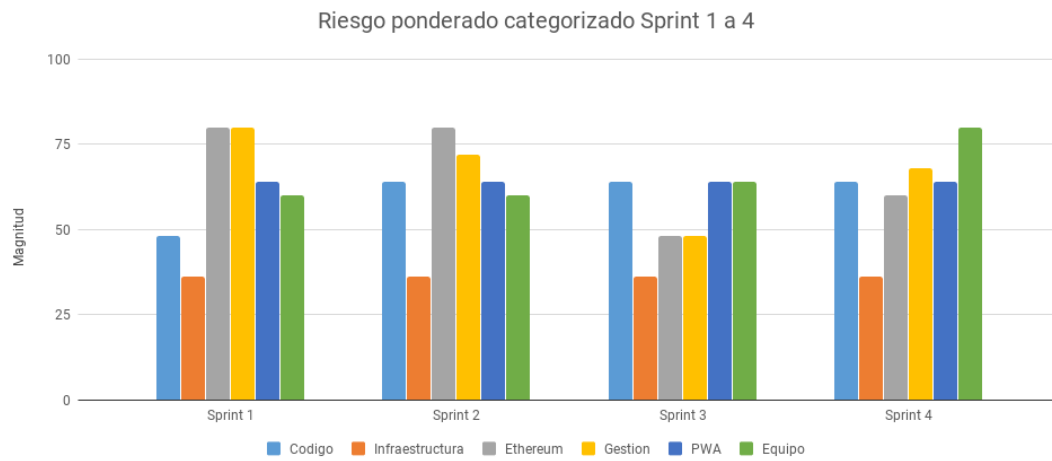
3. Enfoque en mantenibilidad: este riesgos surge del requerimiento introducido por el cliente para poder obtener código entendible, extensible y modificable como resultado final. Si bien el cumplimiento de ese requerimiento no es tan apreciable con el resultado en ejecución, creemos que le aporta mucho valor al cliente el poder continuar con el desarrollo del producto una vez finalizado el proyecto.

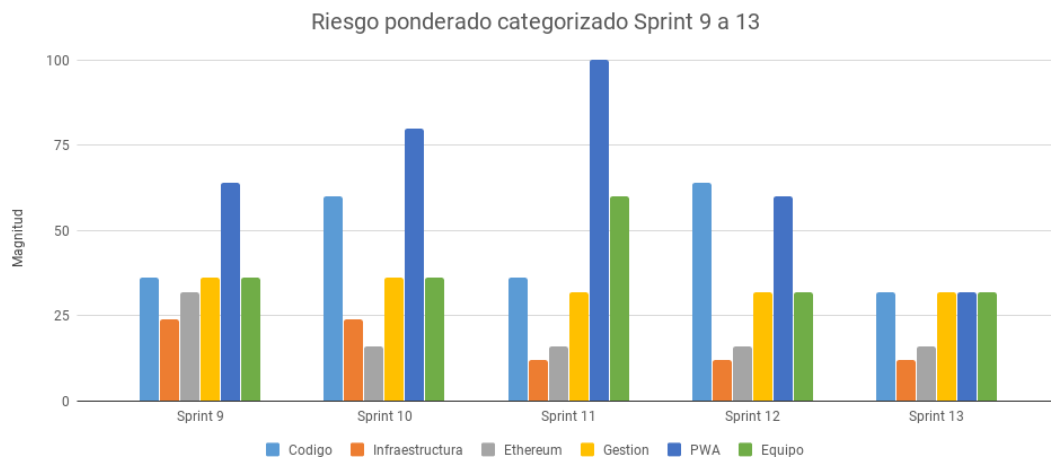
Para cumplir a lo mejor de nuestro criterio con este riesgo decidimos incorporar diferentes practicas (revisiones de código cruzadas), y herramientas (*linterns*), para asegurarnos de construir código que sea entendible y extensible.

4. Retrabajo por malas decisiones: si bien somos estudiantes avanzados de la carrera, eso no impide que cometamos errores no previstos a la hora de tomar decisiones arquitectónicas o de diseño, por ese motivo existe este riesgo.

Este fue otro de los riesgos que no fue detectado inicialmente hasta el quinto sprint donde el riesgo se hizo evidente. Al manifestarse el equipo debió de reaccionar invirtiendo tiempo extra en algunas funcionalidades para lograr el comportamiento deseado.

8.5.3 Evolución





Como se puede observar en la gráfica anterior la mayoría de los riesgos fueron disminuyendo su magnitud a lo largo del proyecto. La razón detrás fue que el equipo logró encontrar planes para mitigarlos y/o reaccionar ante su ocurrencia.

Sin embargo queremos detenernos en las categorías cuyos riesgos no disminuyeron con el pasaje del tiempo, concretamente vamos a hablar de las categorías: Código y Equipo.

Respecto a la primera podemos ver que la magnitud del riesgo comienza media y tiene un pico en el *sprint* 5. Esto se debe a que se manifiesta el riesgo “Retrabajo por malas decisiones” a causa de algunos escenarios que no fueron tenidos en cuenta por el equipo a la hora de realizar la funcionalidad de filtros en las ofertas del mercado. Por suerte esto pareció ser un incidente aislado y no se repitió luego de esa incidencia.

Respecto al segundo, Equipo, vemos que comienza con una magnitud media, y a medida que pasa el tiempo dicha magnitud sube hasta que luego de la cuarta iteración el riesgo “Baja performance” de uno o varios de los integrantes del equipo. En esa instancia se conversó entre los integrantes del equipo y se llegó a una solución aceptable al momento, eso se refleja en la disminución de la magnitud del riesgo. Sin embargo la situación se repitió en el octavo *sprint*, en esta ocasión se activó el plan de respuesta recurriendo al tutor para que intervenga en la situación. Gracias a su intervención la situación se pudo normalizar y el riesgo volvió a disminuir.

Al comenzar el desarrollo afrontando los desafíos tecnológicos en una etapa temprana, se pudo disminuir el impacto de los riesgos asociados al uso de *Ethereum*.

Es destacable el caso del riesgo “**Dificultades para desarrollar múltiples partes que deben de interactuar entre si correctamente**”, ya que se debieron construir múltiples componentes con alto grado de incertidumbre. Tener desvíos en alguno de ellos o en la comunicación entre los mismos implicaba un impacto no menor en el equipo, y por tanto en el producto final.

El buen trabajo realizado para la disminución de la magnitud de los riesgos fue clave para que no aparezcan nuevos riesgos de alto impacto o probabilidad. Sin esta actividades no se puede asegurar que el resultado hubiera sido el mismo. Por esto es que también las actividades de identificación y ponderación fueron claves sumados por supuesto a los distintos planes de solución.

8.6 Gestión del tiempo

En esta sección se procederá a mostrar como se organizó el registro de horas a lo largo del proyecto. Como se comentó previamente se utilizó la herramienta *Toggl*. Dentro de ella organizamos cuatro categorías para facilitarnos esta tarea.

Los categorías son los siguientes:

- **Investigación:** En esta categoría podemos encontrar toda hora que fue invertida por el equipo para investigar y capacitarse en el uso de las tecnologías requeridas por el sistema.
- **Desarrollo:** En esta categoría podemos encontrar todos los registros de tiempo invertidos en el desarrollo de la segunda etapa del proyecto. Cabe destacar que las actividades de prueba, configuración de ambientes, configuración de integración continua, etc. se encuentran comprendidos en este categoría.
- **Gestión:** En esta categoría encontramos toda hora invertida en tareas relacionadas a la gestión del proyecto, reuniones con el tutor, ceremonias de *Scrum*, etc.
- **Documentación:** En esta categoría podemos encontrar toda hora invertida en tareas relativas a la elaboración de la documentación final.

A continuación se puede observar una gráfica con la distribución de horas durante el transcurso del proyecto.

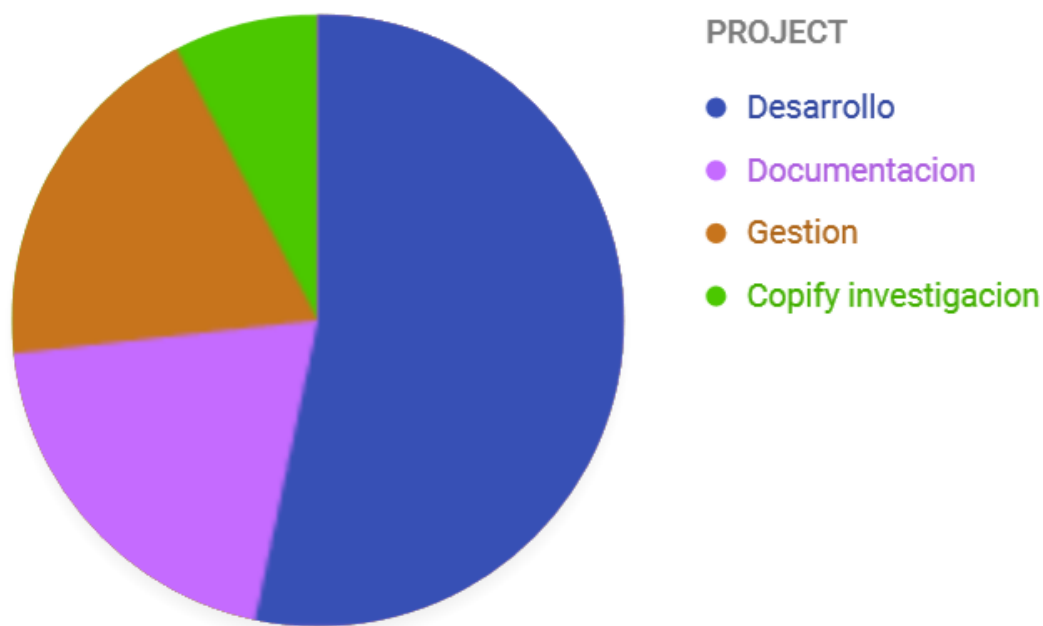


Figura 8.10: Distribución de horas totales

En esta gráfica podemos observar que la etapa en donde más esfuerzo y tiempo se le dedicó al proyecto fue durante la etapa de desarrollo. Esto es coherente debido a que es la actividad principal del proyecto y por ende se le dedico un mayor tiempo en la planificación inicial.

Luego se encuentra la categoría de gestión del proyecto, al ser un ítem transversal al proyecto, es decir que transcurre desde el inicio hasta el final del mismo (aunque con mas foco en la etapa de desarrollo), tiene sentido que esto se vea reflejado en su alta carga horaria.

Por ultimo, las etapas de documentación e investigación son las que menos tiempo se le dedicó en todo el proyecto, esto también es coherente ya que el lapso de tiempo para dichas etapas fue de uno y cuatro meses respectivamente.

Esto último se puede comprobar más detalladamente en la siguiente gráfica en donde se ilustra la distribución de horas durante el transcurso del proyecto por cada mes.

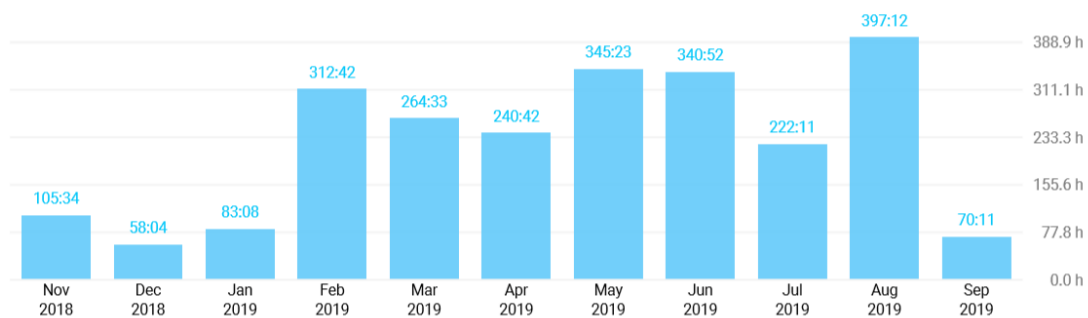


Figura 8.11: Distribución de horas por mes

Como se mencionó anteriormente, los primeros tres meses corresponden a la etapa de investigación y gestión del proyecto, mientras que el ultimo corresponde a la etapa de documentación. Entre Febrero y Julio se dedicó a la etapa de desarrollo y gestión del proyecto.

8.6.1 Lecciones aprendidas

Como crítica constructiva realizada por parte del equipo, una vez sobre la etapa final del proyecto, fue que podríamos haber explotado más las funcionalidades provistas por la herramienta *Toggl*.

Esto se ve reflejado en que contamos con cuatro categorías (o proyectos) en donde registramos las horas invertidas, algo lo cual nos percatamos que sería más fructífero un detalle más granular en el tiempo invertido durante todo el proyecto.

Como posible mejora podríamos haber llevado una trazabilidad de las horas estimadas para cada funcionalidad haciendo uso de las “etiquetas” provistas por la misma.

De esta manera podemos tener un registro más fidedigno de las horas invertidas, y a su vez, poder comparar las horas estimadas para una determinada funcionalidad contra las realmente invertidas en la misma.

8.7 Gestión de la comunicación

La comunicación del equipo fue un pilar fundamental durante el transcurso del proyecto, ya que el éxito del mismo radica en parte en una comunicación fluida y constante entre cada miembro del mismo.

En esta sección detallaremos los medios y herramientas las cuales el equipo utilizó para mantenerse comunicado. En el anexo 13.4.6 se evidencia el uso de estas herramientas.

WhatsApp

Para la comunicación diaria, el equipo decidió utilizar *WhatsApp* debido a que es una herramienta muy sencilla y todos los miembros de mismo tienen experiencia en su uso cotidiano.

Utilizando esta herramienta, los integrantes del equipo fueron capaces de canalizar la mayoría de la comunicación interna, coordinaciones para reuniones, recordatorios, etc. De manera similar también llevaron la mayoría de las comunicaciones internas con el tutor por este medio.

Slack

Para la comunicación técnica decidimos utilizar la herramienta *Slack*, entendiéndose por comunicación técnica a los registros diarios de actividad de cada integrante, *links* interesantes que aporten al proyecto, entre otros. Dado que Camilo contaba con bastante experiencia en el uso de esta herramienta, optamos por utilizarla.

Contamos con los siguientes cinco canales:

- **general:** Canal dedicado a compartir *links* a sitios interesantes que aportan al proyecto.
- **develop:** Canal en donde se publica toda aquella información pertinente a la etapa de desarrollo, como por ejemplo usuarios de registro, *API Keys* ⁴, etc.
- **daily:** Canal en donde se registran las *daily*s de cada integrante.

⁴Una *API Key* es un identificador que sirve como el medio de autenticación de un usuario para el uso de los servicios proporcionados por un determinado *Web Service* o *API Rest*.

- **reuniones:** Canal en donde se suben evidencias o material relacionado a las reuniones mantenidas con el tutor u otros entes de la universidad.
- **documentación:** Canal en donde se sube información relevante sobre la etapa de la documentación final del proyecto.

Google Hangouts

Utilizamos *Google Hangouts* para realizar reuniones de manera remota mediante videollamada dado que el equipo cuenta con bastante experiencia en el uso de la misma.

Se trata de una herramienta *web* en donde el único requisito para utilizarla es contar con una cuenta de *Google*, por lo que no es necesario instalar ninguna aplicación extra. Por otro lado nos ofrece la posibilidad de realizar conferencias, compartir pantalla de escritorio, y una muy buena calidad de llamada.

Reuniones

El equipo acordó en conjunto con el tutor del proyecto Darío Macchi una reunión presencial en la universidad, cuyo objetivo fue llevar a cabo una puesta en común sobre el avance del proyecto, y por otro lado, de resolver consultas específicas sobre el mismo.

Estas reuniones se realizaron cada dos semanas los días jueves a las 18:30 horas en la universidad *ORT* Uruguay, y su duración fue de aproximadamente una hora cada una.

Por otro lado, el equipo también decidió juntarse los días lunes siguientes a terminar un *sprint* para poder realizar la *sprint retrospective* y la *sprint planning* del siguiente.

Al margen de esto, también se definieron reuniones presenciales para la etapa de desarrollo durante los fines de semana con el fin de avanzar sobre dicha etapa con una mayor eficiencia. La ubicación de dichas reuniones iba rotando entre cada integrante.

8.7.1 Lecciones aprendidas

Consideramos que no aprovechamos todo el potencial que nos ofrece la herramienta *Slack*.

Durante el transcurso del proyecto el equipo generó varios canales en donde algunos cuentan con actividad discontinuada. Por otro lado, fuera del ámbito de registro de las *dailys*, mucha información compartida por este medio también era compartida por *WhatsApp*.

9 Gestión de Calidad

Con el objetivo de lograr un producto de calidad se consideró necesario definir distintas actividades que permitieran controlar que tanto el proceso como el producto cumplieran con los objetivos planteados.

Para esto, en primera instancia, se definió que entendemos por calidad y cuáles eran los objetivos en cuanto a calidad del producto a desarrollar, así como los objetivos del proceso.

Los objetivos de esta gestión fueron definidos de la manera *SMART*¹, estos objetivos se caracterizan por contar con las siguientes propiedades:

- **S:** Específico
- **M:** Medible
- **A:** Alcanzable
- **R:** Relevante
- **T:** Duradero en el tiempo

A continuación se enumeran los objetivos planteados:

- Objetivos de calidad del producto
 - Proveer un sistema que cumpla las expectativas del cliente, comprobándolo a través de encuestas y recibiendo *feedback* del mismo
 - Proveer un sistema confiable y seguro en el manejo de los datos donde aquellos relevantes se encuentren encriptados

¹Los objetivos *SMART*, modelo creado por George T.Dorian, se basa en un modelo que define unos objetivos inteligentes, posibles y eficaces. [9]

- Brindarle al usuario una experiencia de uso intuitiva siguiendo las Heurísticas de Nielsen, verificando esto con pruebas de usuario
 - Entregarle al cliente un producto con la menor cantidad de defectos posible
 - Entregar un producto compatible con *PWA* (*Progressive Web App*), dicho cumplimiento será verificado por medio de *Lighthouse* (ver 9.3)
- Objetivos de calidad del proceso
- Se busca tener un proceso de desarrollo definido, respetando *SCRUM* y sus ceremonias. Poder utilizar los resultados de las retrospectivas y las manifestaciones del cliente en las revisiones
 - Poder tomar acciones correctivas a tiempo reaccionando ante situaciones anómalas de forma rápida y eficaz, para esto tratamos de que no se manifiesten riesgos a lo largo del proyecto.
 - Completar el alcance del *sprint backlog* a lo largo del proyecto, manteniendo al mínimo las instancias en las que no se cumpla.
 - Introducir la menor cantidad de defectos posibles a lo largo de todo el proyecto. Para esto evaluamos cuantos escapan a las tareas de prueba en su primera instancia y deben ser solucionados en el próximo *sprint*

9.1 Actividades de calidad

Validaciones

El proceso de validación consiste en asegurar que lo que se está construyendo se corresponde con el uso que se le quiere dar al producto.

Al finalizar cada sprint el equipo se reunió y evaluó si esto se está cumpliendo. A su vez cuando el equipo lo considere necesario se podrá evaluar con distintos usuarios mostrándoles prototipos o demos de las funcionalidades desarrolladas.

Producto a evaluar	Técnica	Registro de resultados	Cuando se realiza
<i>Product Backlog</i>	Validación con el equipo de desarrollo y cliente	Consentimiento del cliente. Registro en <i>teamwork</i>	Periódicamente
Producto de software	Validación con el cliente	Consentimiento del cliente.	Periódicamente

Revisiones

Las revisiones permiten obtener defectos, desviaciones y retroalimentación sobre el software que se está desarrollando para así poder aplicar en caso de ser necesario acciones correctivas. Las mismas fueron llevadas a cabo por el equipo y se dividieron de la siguiente manera:

Producto a evaluar	Técnica	Registro de resultados	Cuando se realiza
Código fuente	Revisiones cruzadas	<i>Pull requests</i> cerrados, cancelados, comentados, etc, en <i>GitHub</i>	Al finalizar una funcionalidad
Gestión de riesgos	Revisión de los documentos	Plan de riesgos actualizado	Al finalizar cada <i>sprint</i>

Verificaciones

Consiste en verificar que cada entregable de software cumple con los requerimientos especificados. El equipo en esta instancia verificó internamente que se haya cumplido con todas las tareas planificadas para cada *sprint* de la siguiente manera:

Producto a evaluar	Técnica	Registro de resultados	Cuando se realiza
Producto de software	Evaluación del funcionamiento y completitud de las funcionalidades	Registro en <i>teamwork</i>	Al finalizar cada <i>sprint</i>

9.2 Plan de calidad

Teniendo en cuenta estos objetivos se elaboró el plan de calidad encontrado a continuación:

Áreas	Actividad	Entrada	Salida	Herramientas y tecnología	Roles
Ingeniería de requerimientos	Relevamiento	Solución	Requerimientos	Entrevistas con el cliente	Equipo
	Priorizar requerimientos	Requerimientos	Requerimientos priorizados	Entrevistas con el cliente	Equipo
	Determinar alcance del proyecto	Requerimientos priorizados	<i>Product backlog</i>	<i>Teamwork</i>	Equipo
	Determinar alcance del <i>sprint</i>	Requerimientos estimados	<i>Product backlog</i>	<i>Poker planning</i>	Equipo
Arquitectura y diseño	Priorización de requerimientos no funcionales	Requerimientos no funcionales	Atributos de calidad, tácticas, bocetos y diagramas	Artículos, documentos	Arquitecto/Equipo
	Diseño de arquitectura	Bocetos y diagramas	Diagramas concretos	UML, consulta con expertos	Arquitecto/Equipo
	Revisión y validación de la arquitectura	Diagramas concretos	Arquitectura validada y verificada	Revisión con expertos	Arquitecto/Equipo
	Diseño de la solución	Requerimientos funcionales	Documento de diseño	UML	Arquitecto/Equipo
Desarrollo	Asignación de los recursos	Arquitectura, diseño y tecnologías Experiencia previa de los integrantes del equipo	Responsabilidades asignadas	Evaluación de habilidades y preferencias de los integrantes del equipo	Equipo
	<i>Clean code</i>	Código fuente en tiempo real	Sugerencias para acop	Reglas específicas y <i>lint</i>	Equipo
	Revisión de código	Código fuente	Código fuente corregido	<i>Pull requests</i> , estándares de código	Equipo
Pruebas	Pruebas unitarias	Funcionalidad terminada	Resultados esperados y obtenidos	<i>Mocha</i> y <i>CirdeCI</i>	Calidad
	Verificación de cumplimiento de estándares de <i>Clean Code</i>	Código fuente	Resultados esperados y obtenidos	<i>CircleCI</i>	Calidad
	Pruebas de usabilidad	Sistema creado y requerimientos terminados	Resultados	Excel (planilla casos de pruebas)	Calidad
Universidad ORT	Reuniones con el tutor	Dudas e inquietudes	Consejos y recomendaciones basadas en la experiencia del tutor	Reuniones presenciales/remotas	Equipo
	Revisiones	Avance al momento	Oportunidades de mejora	Presentaciones	Equipo

Figura 9.1: Plan de calidad

9.3 Aseguramiento de la calidad

En esta sección se continuará a detallar el proceso de llevar a cabo la calidad en los puntos mencionados del plan de calidad.

Requerimientos

Las interacciones con el cliente al comienzo del proyecto permitieron realizar un relevamiento de los mismos claros y concisos, esto llevó a describir historias de usuario (ver *Product Backlog* 8.1) detalladas las cuales le permitieron al equipo interpretar mejor la funcionalidad a llevar a cabo.

Diseño

El equipo desarrolló una serie de ideas y soluciones al problema enfrentado, las mismas fueron validadas con expertos. Para esto, se coordinó una reunión con Gastón Mousques para exponerle dichas ideas y obtener comentarios con respecto a lo que se podía mejorar y sacarle aún un mejor provecho.

Uso de estándares de codificación

Con el fin de obtener código que se adapte a los principios básicos de diseño y que sea limpio se definieron estándares a seguir en cuanto a la codificación del mismo.

- Al ser un producto con el objetivo de comercializarse en el exterior, se decidió que el código fuente fuera escrito totalmente en el idioma inglés. Esto incluye comentarios.
- El nombre de las clases debe empezar siempre con mayúscula, es decir, se debe utilizar *UpperCamelCase*
- Los métodos deben comenzar con minúscula. Se debe utilizar *lowerCamelCase*
- En las clases primero se deben declarar los atributos, luego los constructores y por último los métodos de la misma
- Se debe dejar una línea por medio entre cada método y la declaración de los atributos y métodos

- Usar *try-catch-finally* para el manejo de excepciones
- El nombre de los métodos debe ser auto explicativo
- Métodos pequeños que cumplan una sola función
- Seguir las buenas prácticas de cada lenguaje en particular
- Se deben seguir los principios de diseño básicos como Abierto-Cerrado y Responsabilidad Única

Estos estándares definidos por el equipo fueron verificados, comentados (en caso de ser necesario) y validados en cada *pull request*. En caso de realizarse alguna observación debía ser resuelta y confirmada antes de incorporarse a la rama destinataria. Se puede observar la traza de un *pull request* realizado en el siguiente anexo 13.5.1.

Uso de estándares de documentación

Se definieron los estándares de documentación 302 [72], 303 [70], 306 [73] publicados por Universidad *ORT* Uruguay.

Buenas prácticas utilizadas

Práctica	Desarrollo	Objetivos / Funcionamiento
Ceremonias de <i>Scrum</i>	Cumplir con las diversas pautas y ceremonias de <i>Scrum</i>	Para la correcta utilización de <i>Scrum</i> (ver sección 4.4.2) se realizarán las siguientes ceremonias: - <i>Sprint retrospective</i> - <i>Daily meeting</i> - <i>Sprint review</i> - <i>Sprint planning</i>
<i>Pair programming</i>	Escribir el código de a pares en una misma computadora	Mejorar la calidad del código, tener mayor disciplina
<i>Refactoring</i>	Contar con tiempo para analizar el código y realizar mejoras	Obtener código que cumpla con los estándares definidos y que se encuentre alineado a los principios de diseño
<i>Code review</i>	Generar instancias para la revisión de código	Obtener código que cumpla con los estándares y mejorar la calidad del mismo. También es una manera de aprender nuevas formas de implementar cierta funcionalidad al tener que buscar alternativas más eficientes. Ayuda a que el código sea más mantenible
<i>Spikes</i>	Realizar tareas de estudio e investigación	Obtener el conocimiento necesario para desarrollar las distintas funcionalidades. Sobre todo teniendo en cuenta el reto de desarrollar sobre tecnologías nuevas para todo el equipo
Obtención y medición de métricas	Medir y documentar a lo largo del desarrollo las distintas métricas que se establecieron que eran importantes	Tener un mayor control sobre el proyecto. Tomar medidas correctivas

Pruebas de software

Las pruebas de *software* son fundamentales para determinar el nivel de calidad de un producto, tienen como objetivo principal el detectar las posibles fallas o defectos del producto durante la etapa de desarrollo. Evitando así que las mismas sean descubiertas por los usuarios finales.

A continuación pasaremos a detallar los distintos tipos de pruebas utilizados durante la etapa de desarrollo:

Pruebas unitarias

Con el objetivo e importancia de mitigar los riesgos de forma temprana, es que surge la necesidad de que para cada funcionalidad desarrollada se implementen pruebas correspondientes a las entidades, servicios y controladores respectivamente.

Este tipo de pruebas nos ayudaron a mitigar posibles errores que pudieran surgir al momento de integrar dichas funcionalidades al resto del sistema.

```
describe('#POST /api/offers/', async () => {
  context('No offer created', async () => {
    let createOfferClone, user, createCategoryClone, categoryId
    beforeEach(async () => {
      user = await factory.create('user', createUser)
      createOfferClone = _.cloneDeep(createOffer)
      createOfferClone.userId = user.id
      createCategoryClone = _.cloneDeep(createCategory)
      const cat = await factory.create('category', createCategoryClone)
      categoryId = cat.name
    })
    it('Should create the new offer', async () => {
      const token = (await loginUser(user)).accessToken
      const res = await request.post('/api/offers/').set('Authorization', `bearer ${token}`)
        .send(createOfferClone).expect(200)
      expect(res.body.offer.userId).to.eq(createOfferClone.userId)
      expect(res.body.offer.status).to.eq(createOfferClone.status)
    })
    it('Should create the new offer with its prices', async () => {
      const token = (await loginUser(user)).accessToken
      createOfferClone.prices = new Array()
      createOfferClone.prices.push({ selected: true, frequency: 'FinalProduct', price: 20 })
      const res = await request.post('/api/offers/').set('Authorization', `bearer ${token}`)
        .send(createOfferClone).expect(200)
      expect(res.body.offer.userId).to.eq(createOfferClone.userId)
      expect(res.body.offer.status).to.eq(createOfferClone.status)
    })
  })
})
```

Figura 9.2: Ejemplo de prueba unitaria

Gran parte de la calidad se basa en la automatización de tareas ya que quitan el factor humano, el cuál es propenso a errores. Por esto se decidió la utilización de *CircleCI*² para ejecutar de manera automática pruebas y *linterns*.

² *CircleCI* es una plataforma de integración continua y entrega de resultados para automatizar los procesos de creación, prueba e implementación

Jobs » coopify » coopify » develop » 45 (LintAndTest)

3 (01:21)	TEST	
Spin up Environment		00:13
Container circleci/postgres:9.6.2-alpine		01:08
Container redis		01:08
Checkout code		00:00
Wait for redis		00:00
Install apt-transport-https		00:01
Apt-get Update		00:01
Install Yarn		00:00
Yarn install dependencies		00:10
Run Lint		00:04

Figura 9.3: Ejemplo de *CircleCI*

Jobs » coopify » coopify » filters-fixes » 36 (LintAndTest)

3 (00:24)

Run Lint 00:04

```

$ #!/bin/bash -eo pipefail
yarn lint

yarn run v1.3.2
$ yarn run lint:js && yarn run lint:ts
$ node_modules/.bin/eslint .
$ node_modules/.bin/talint -t codeFrame 'src/**/*.ts' 'lib/**/*.ts' 'bin/**/*.ts' 'config/**/*.ts' 'test/**/*.ts'
src/server/models/sequelize/offer.ts
Unnecessary semicolon (semicolon)
5 | import { Proposal } from '..'
6 | import { Transaction, Op, OrderItem } from 'sequelize'
> 7 | import { logger } from '../../services';
    |                                     ^
8 |
9 | interface IAttributes {
10 |   userId: string
error Command failed with exit code 2.
info Visit https://yarnpkg.com/en/docs/cli/run for documentation about this command.
error Command failed with exit code 1.
info Visit https://yarnpkg.com/en/docs/cli/run for documentation about this command.
Exited with code 1

```

Figura 9.4: Ejemplo de *CircleCI* con advertencia de *Lint*

Pruebas de sistema

Para llevar un control sobre las pruebas realizadas y los defectos encontrados se decidió llevar un registro al finalizar cada *sprint* con las funcionalidades del mismo probadas con sus respectivos resultados.

Este registro consta de los siguientes datos para cada caso de prueba:

- Número del caso de prueba
- Nombre del caso de prueba
- Ejecutado

- Esperado
- Obtenido
- Corregido
- Observación

De esta manera pudimos registrar casos de prueba con resultados positivos y negativos. Documentar observaciones sobre un posible camino para la solución del error, y llevar el control por *sprint* del resultado de cada prueba. Esto último con el fin de tener la traza sobre cuando se corrigieron y reincidencias de los mismos.

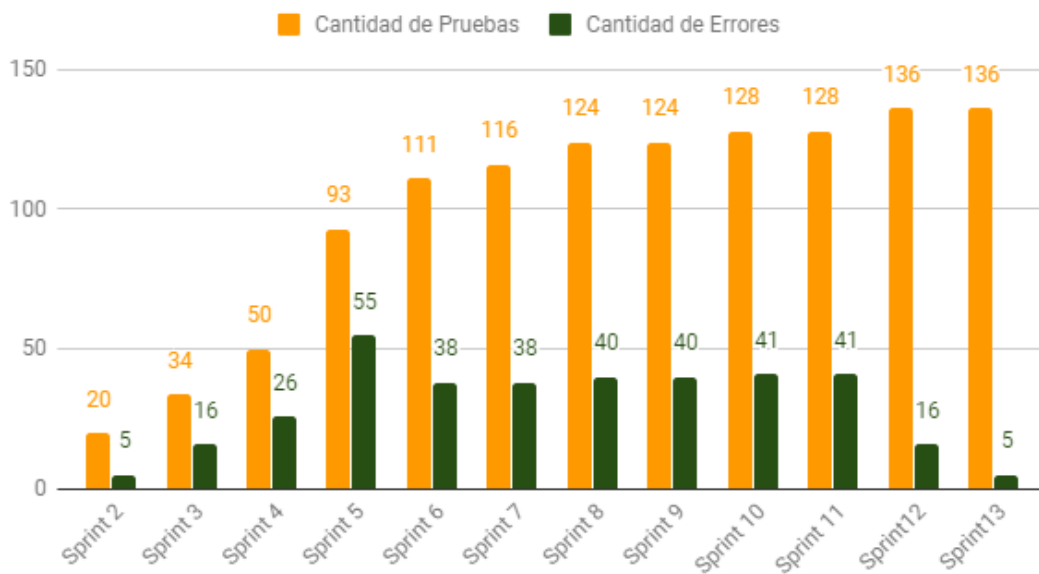


Figura 9.5: Gráfica de pruebas y errores por *sprint*

En los primeros *sprints* se dispararon los errores debido a la inexperiencia del equipo en las tecnologías seleccionadas. El equipo decidió estabilizar la cantidad de defectos, adquirir conocimientos de buenas prácticas y poder bajar la cantidad de los mismos a medida que iban transcurriendo los próximos *sprints*.

Cumplimiento de PWA

Para asegurar el cumplimiento de estándares propuestos por *Google* con respecto a las aplicaciones web progresivas optamos por utilizar la herramienta *Lighthouse*,³ la cual nos permite generar un reporte con métricas sobre el rendimiento de la aplicación.

A continuación se puede comprobar que nuestra aplicación cumple en gran medida con los estándares requeridos para ser una *PWA*.

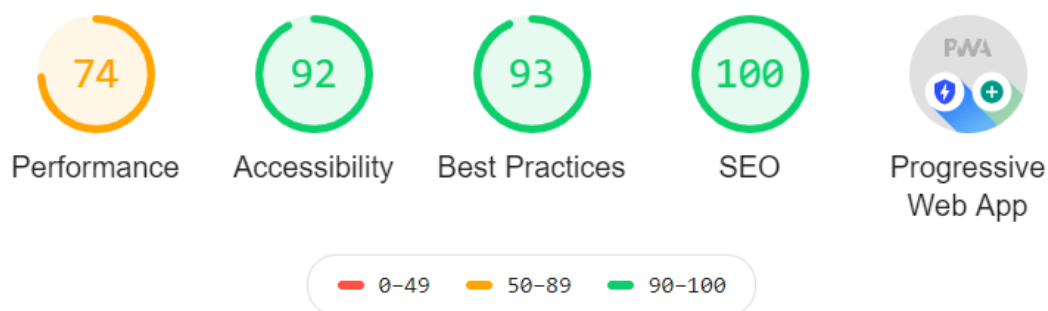


Figura 9.6: Reporte de cumplimiento de estándares de *PWA* por parte de *Lighthouse*

Pruebas con usuarios

Se decidió realizar pruebas con usuarios finales con el objetivo de obtener datos que evalúen la calidad de diseño de la interfaz de usuario, recolectar comentarios y poder aplicarlos para una eventual mejora.

Se plantearon 11 casos de prueba, cada uno con determinados prerequisites. Durante la ejecución de los casos, se evaluaron 4 cosas:

- Completar la funcionalidad
- Completar la funcionalidad en el tiempo deseado
- Completar la funcionalidad en un número máximo de pantallas
- Completar la funcionalidad en un rango determinado de *clicks*

³*Lighthouse* es una herramienta automatizada de código abierto diseñada para mejorar la calidad de las *apps web*.

Descripción	Prerrequisitos	Resultado esperado	Tiempo esperado en minutos
Registrarse		complete la funcionalidad complete la funcionalidad en el tiempo deseado complete la funcionalidad en un maximo de 2 pantallas complete la funcionalidad en un rango bajo de clics	2
Ver cualquier Servicio	Haberse registrado y tener sesion iniciada	complete la funcionalidad complete la funcionalidad en el tiempo deseado complete la funcionalidad en un maximo de 2 pantallas complete la funcionalidad en un rango bajo de clics	1
Buscar servicio de su interes	Haberse registrado y tener sesion iniciada	complete la funcionalidad complete la funcionalidad en el tiempo deseado complete la funcionalidad en un maximo de 3 pantallas complete la funcionalidad en un rango alto de clics	4
Negociar y proponer	Haberse registrado y tener sesion iniciada. Seleccionado un servicio	complete la funcionalidad complete la funcionalidad en el tiempo deseado complete la funcionalidad en un maximo de 3 pantallas complete la funcionalidad en un rango alto de clics	4
Ver conversaciones	Haberse registrado y tener sesion iniciada	complete la funcionalidad complete la funcionalidad en el tiempo deseado complete la funcionalidad en un maximo de 3 pantallas complete la funcionalidad en un rango medio de clics	1
Crear servicio (crear proposicion nosotros)	Haberse registrado y tener sesion iniciada	complete la funcionalidad complete la funcionalidad en el tiempo deseado complete la funcionalidad en un maximo de 2 pantallas complete la funcionalidad en un rango alto de clics	5
Ver saldo	Haberse registrado y tener sesion iniciada	complete la funcionalidad complete la funcionalidad en el tiempo deseado complete la funcionalidad en un maximo de 2 pantallas complete la funcionalidad en un rango bajo de clics	1

Figura 9.7: *Template* resumido de las pruebas de usuarios

Métrica		Mínimo	Maximo
Cantidad de clicks	Bajo	1	4
	Medio	4	8
	Alto	8	12
	Muy alto	12	
Cantidad de pantallas	Bajo	1	2
	Medio	2	4
	Alto	4	5
	Muy alto	6	

Figura 9.8: Glosario con los valores determinados por el equipo

Encuestas al cliente

Realizamos una encuesta para evaluar el grado de satisfacción del cliente con el equipo y conocer su opinión personal del proyecto y sus resultados hasta el mo-

mento. Ésta encuesta fue completada en momentos clave en la etapa de desarrollo: luego de implementación de la integración *Ethereum* y en etapas culmines del desarrollo.

No se pueden editar las respuestas

Sprint customer feeling feedback

*Obligatorio

El producto entregado cumple con lo esperado *

1 2 3 4 5 6 7 8 9 10

Extremadamente en desacuerdo ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☒ Extremadamente de acuerdo

El equipo respondió bien ante el desafío planteado *

1 2 3 4 5 6 7 8 9 10

Extremadamente descontento ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☒ Extremadamente contento

Se siente conforme con el desarrollo del proyecto al momento *

1 2 3 4 5 6 7 8 9 10

Extremadamente descontento ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☒ Extremadamente contento

¿Cuál es la razón principal de tu puntuación?

Estoy muy conforme con el equipo, se nota el esfuerzo y la dedicación, la comunicación es excelente!

¿Qué fue lo que faltó o decepcionó en tu experiencia con nosotros?

La verdad no veo que haya fallado nada, estoy muy conforme con el equipo

¿Cómo podemos mejorar su experiencia y/o el producto entregado?

El componente de Blockchain es lo mas rico del proyecto, hay que remarcarlo de alguna manera en el software o con algún informe de investigación con posibilidad de compartirlo con el CIE tal vez

Enviado: 19/5/19 15:25

Figura 9.9: Evidencia de satisfacción del cliente con el equipo

Retrospectivas

Actividad realizada al terminar cada *sprint* donde el equipo analizó lo acontecido a lo largo del mismo, buscando posibles puntos de mejora. Luego de exponer sus opiniones e ideas, el equipo ideaba un *action plan* a llevar a cabo en el siguiente *sprint*. Todo esto siempre persiguiendo la mejora continua.

Se planificó realizar las retrospectivas remotas utilizando la herramienta de *Hangouts* de *Google* para la comunicación e *IdeaBoardz* para simular un pizarrón en los casos que los miembros no pudieran coordinar para reunirse y contar con una instancia presencial. El resto de las retrospectivas fueron planificadas realizarlas en la Universidad, para sacarle el mejor provecho a las mismas.

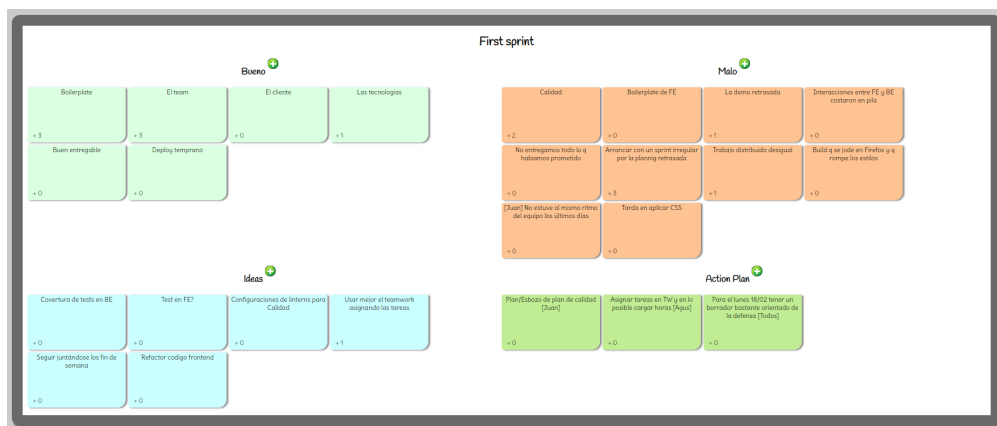


Figura 9.10: Retrospectiva remota

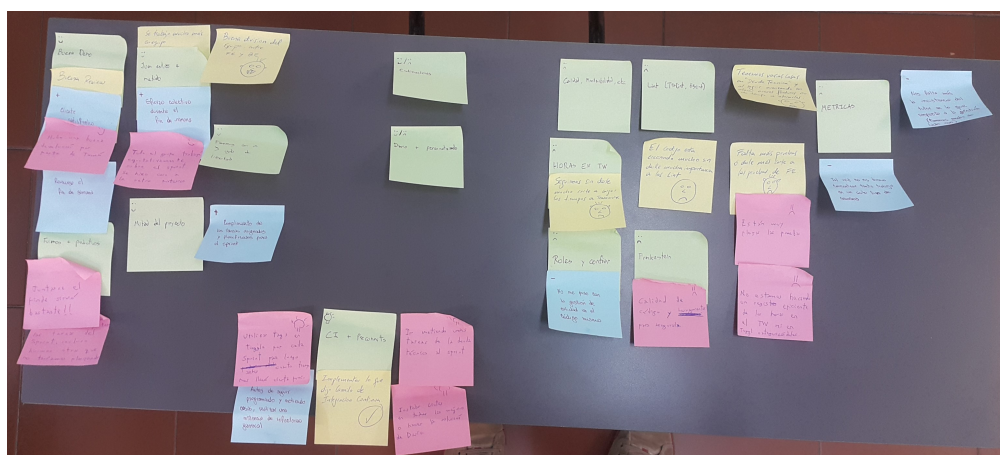


Figura 9.11: Retrospectiva presencial

Revisiones con el cliente

Cada *sprint* terminaba con una demostración al cliente de las funcionalidades implementadas en el mismo. Es una práctica realizada habitualmente en el ámbito profesional la cuál nos iba a servir también para conocer la opinión del cliente y obtener la retroalimentación correspondiente (Ver evidencia 8.7)

Reuniones de tutoría

Actividad realizada cada dos semanas entre los integrantes del equipo y el tutor. En estas instancias se comentó y se evaluó el trabajo realizado hasta el momento al tutor, para que luego éste pudiera comentarnos los siguientes pasos a tomar con el fin de continuar avanzando correctamente con el proyecto.

Las mismas fueron de mucha ayuda ya que en un principio no se contaba con una idea a ciencia cierta de como proceder en un proyecto de tal magnitud y tener una opinión de una persona con más experiencia siempre es muy bueno.

Esto nos sirvió a la hora de la toma de decisiones importantes tales como: la división del proyecto en las etapas de investigación y desarrollo, las metodologías a utilizar en cada una de ellas y la forma correcta en la que abarcarlas.

Revisiones de avance

A lo largo del proyecto el equipo contó con tres instancias de validación del trabajo realizado por parte de revisores de la Universidad *ORT* Uruguay.

En dichas instancias el equipo debía realizar una presentación al revisor en la cual se exponía el avance del proyecto hasta el momento. Posteriormente el revisor nos daba una devolución sobre el estado y grado de avance del mismo junto a lo positivo y puntos a mejorar de este último.

Luego de cada instancia realizamos un informe en el cual se detallan tres puntos importantes:

- Fortalezas del equipo en cuanto al trabajo realizado hasta el momento.
- Oportunidades de mejora con respecto al proyecto.
- Acciones para implementar las oportunidades de mejora anteriores.

El detalle de estas revisiones se puede encontrar en el anexo 13.5.2.

9.4 Conclusiones de calidad

En líneas generales el objetivo de lograr un producto de calidad lo consideramos cumplido, para esto utilizamos una variedad de actividades.

Dentro de ellas creemos importantes destacar:

- La variedad de pruebas utilizadas nos permitió asegurar de entregar un producto final con la menor cantidad de defectos posibles. Agregado a esto nos dio la posibilidad de detectar cuando eran introducidos defectos y solucionarlos pro activamente.
- Las retrospectivas nos dieron la posibilidad de detectar donde el equipo estaba fallando, exponer problemas e inquietudes, así como encontrar soluciones. Fue una de las instancias a las que el equipo le saco mayor provecho. De las mismas surgieron variedad de planes de acción que fueron implementados a lo largo del proyecto, con el fin de continuar mejorando la calidad del producto entregado.
- Las instancias de intercambio con el tutor y los revisores nos permitieron detectar problemas y oportunidades que de otra forma no habríamos detectado. Por tanto jugaron un papel importantísimo para lograr un producto de calidad.
- Las revisiones con el cliente en conjunto con la encuesta de satisfacción nos ayudo a validar el proceso y trabajo realizado hasta el momento, y de esta manera asegurar la calidad del producto desarrollado.

9.5 Lecciones aprendidas

Todo el proceso de desarrollo nos permitió adquirir conocimiento, este aprendizaje lo pudimos poner en práctica en actividades como: ser autocrítico a la hora de realizar *Code review*, aplicar correctamente las ceremonias de *Scrum*, comprender, entender y aplicar buenas prácticas en las distintas tecnologías.

Podríamos haber utilizado alguna herramienta para automatizar las pruebas de

interfaz gráfica, como por ejemplo *Selenium* ⁴. Nos hubiera permitido aprovechar mejor las bondades de *circleCI* para que cada vez que se fuera a hacer *deploy* saber si estas pruebas llegaron a buen puerto o no. De todas formas pudimos solventar esta deficiencia mediante la planilla de pruebas funcionales de sistema para probar el comportamiento en cada *sprint*.

Si bien al construir las funcionalidades fuimos creando pruebas unitarias en *backend*, consideramos que podríamos haber construido escenarios mas exhaustivos. Ya que hay varios casos de borde que se hicieron visibles con la ejecución de pruebas funcionales de sistema.

Seria importante un mejor uso de los correctores automáticos de código (*linters*), ya que las reglas utilizadas por ellos no fueron lo suficientemente exhaustivas desde un principio. Eso genero que el equipo terminara invirtiendo un tiempo considerable en subsanar la situación.

Son cosas que el equipo al finalizar el trabajo valoró como importante, ya que ayudaron al crecimiento del grupo y a desarrollarnos tanto en forma profesional como personal.

⁴Entorno de pruebas automatizado para *frontend*, ver [63]

10 Gestión de la configuración

La gestión de la configuración consiste en asegurar la calidad de un producto mediante un control de los cambios realizados y de la disponibilidad constante de una versión estable del mismo.

Es importante tener una gestión en esta área para que cada integrante del equipo conozca la versión actual del sistema en desarrollo y poder acceder a los cambios incorporados en diferentes versiones del producto.

En este capítulo se detalla el procedimiento utilizado para la gestión de configuración en todas las etapas del proyecto, a nivel de investigación, desarrollo y documentación.

10.1 Elementos de configuración

Se define un elemento de configuración de software (*ECS*) como toda información creada como parte del proceso de ingeniería de software.

En nuestro caso identificamos dos tipos de elementos de configuración como se observa a continuación:

Código Fuente

- Coopify Frontend
- Coopify Backend
- Coopify Ethereum

Documentación

- Documento de investigación de tecnologías de *blockchain*
- Informes de revisión de *ORT*
- Diagramas de diseño y arquitectura
- Documentos de pruebas del sistema
- Documentos de métricas
- Documentos varios que complementan la documentación
- Documentos de reuniones
- Documentación final

10.2 Herramienta de versionado

10.2.1 Etapa de investigación

Para el versionado de todos los elementos de configuración relacionados con documentos, el equipo decidió utilizar *Google Drive* como repositorio y herramienta de versionado, dado que es una herramienta sencilla de utilizar, la cual todos los integrantes del equipo conocen y tienen experiencia, y a su vez nos brinda numerosas bondades que hacen al proceso de documentación mucho mas eficiente y colaborativo.

Entre estas bondades se encuentra la posibilidad de edición de varios usuarios en simultaneo de un documento, lo que permite al equipo trabajar en paralelo sobre un mismo archivo, aumentando la colaboración entre el mismo y manteniéndonos informados acerca del trabajo del resto de los integrantes.

Por ser una herramienta web, resulta completamente independiente de la plataforma de la cual se este accediendo, lo que nos otorga la posibilidad de revisar o editar documentos en cualquier momento desde una computadora o celular sin necesidad de considerar el sistema operativo o navegador instalado.

Entre otras de las ventajas que nos ofrece se encuentran:

1. Agregar comentarios sobre un bloque de texto seleccionado permitiéndonos generar hilos de discusiones hasta resolver el mismo

2. Versionado automático de documentos
3. Posibilidad de revertir cambios introducidos a un estado previo del documento (*rollback*)

10.2.2 Etapa de desarrollo de software

Al momento de elegir la herramienta de versionado del código fuente, el equipo se presentó ante el desafío de elegir entre dos modelos de versionado, centralizado o descentralizado, básicamente entre *Git* o *Svn*

Dada la naturaleza ágil de nuestro proyecto, los requerimientos variantes sobretodo en la etapa de la *sprint review* con el cliente y las oportunidades que brinda *Git* para una gestión de incidentes mas prolija y eficiente, junto con la posibilidad de que cada miembro del equipo trabaje de manera independientemente, concluimos que dentro de un proyecto de enfoque ágil como el nuestro *Git* nos brinda un entorno mas colaborativo y eficiente al momento de gestionar las versiones del producto.

En cuanto al proveedor de repositorio, existe una lista numerosa de proveedores que utilizan *Git*, entre los mas populares y que el equipo tiene experiencia previa se encuentran *Bitbucket* y *GitHub*.

Dado que el cliente ya nos proveía una organización en *GitHub* y a su vez los miembros del equipo contaban con experiencia previa con este proveedor, se decidió por utilizar *GitHub* como repositorio.

Una comparativa mas en detalle puede observarse en el Anexo 13.6.1.

10.3 Organización de los repositorios

10.3.1 Etapa de investigación

Con el objetivo de llevar una organización eficiente y prolija de la documentación, decidimos separar y agrupar en carpetas distintas los archivos relacionados. A continuación se evidencia la estructura realizada:

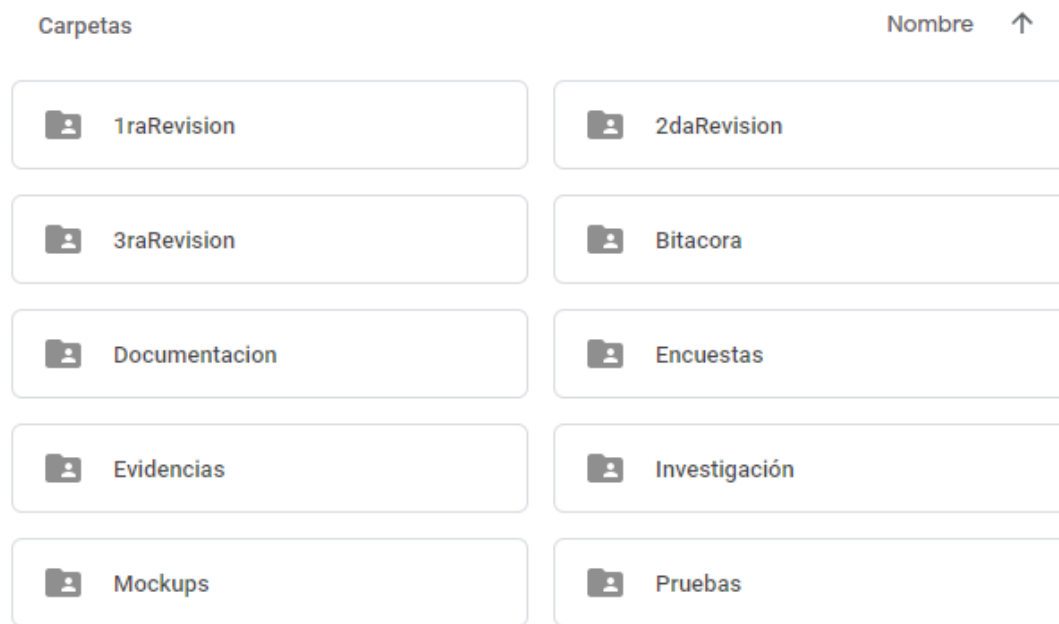


Figura 10.1: Estructura de carpetas en Google Drive

El criterio utilizado por el equipo para incluir archivos dentro de una determinada carpeta fue el siguiente:

- **1ra Revisión:** Carpeta donde se guardan todos los documentos y archivos relacionados a la primera revisión de *ORT*.
- **2da Revisión:** Carpeta donde se guardan todos los documentos y archivos relacionados a la segunda revisión de *ORT*.
- **3ra Revisión:** Carpeta donde se guardan todos los documentos y archivos relacionados a la tercera revisión de *ORT*.

- **Bitácora:** Carpeta donde se guardan datos generales del sistema como usuarios de prueba, claves, etc.
- **Documentación:** Carpeta donde se guardan todos los documentos relacionados a la documentación final.
- **Encuestas:** Carpeta donde se guardan encuestas, validaciones y *mockups* tanto a usuarios como al cliente.
- **Evidencias:** Carpeta donde se guardan evidencias generales de *sprints*, reuniones con el cliente, tutor, entre otros.
- **Investigación:** Carpeta donde se guarda información general recopilada durante la etapa de investigación, en conjunto con el entregable al cliente sobre la elección de la plataforma de *blockchain*.
- **Pruebas:** Carpeta donde se guardan archivos y documentos relacionados a pruebas del sistema.

10.3.2 Etapa de desarrollo de software

Para lograr una mayor gestión y organización del código fuente, optamos por separar los repositorios por cada componente de la solución de *Coopify*. A continuación se detalla cada repositorio con su funcionalidad:

- **coopify:** Repositorio encargado de almacenar las distintas versiones relacionadas al *core* o *backend* de la aplicación.
- **ethereum:** Repositorio encargado de almacenar las distintas versiones relacionadas a la integración con *blockchain* por medio de la plataforma *ethereum*.
- **coopify-frontend:** Repositorio encargado de almacenar las distintas versiones relacionadas a la parte visual o de interfaz gráfica (*frontend*) de la aplicación.

10.4 Gestión de versiones

Con el objetivo de lograr una eficiente gestión de versionado, el equipo opto por manejar ramas en *Git* durante cada *sprint* (*Gitflow* [7]) en donde cada miembro trabaja sobre una determinada funcionalidad, para luego unificarlo al finalizar el mismo en una rama en común.

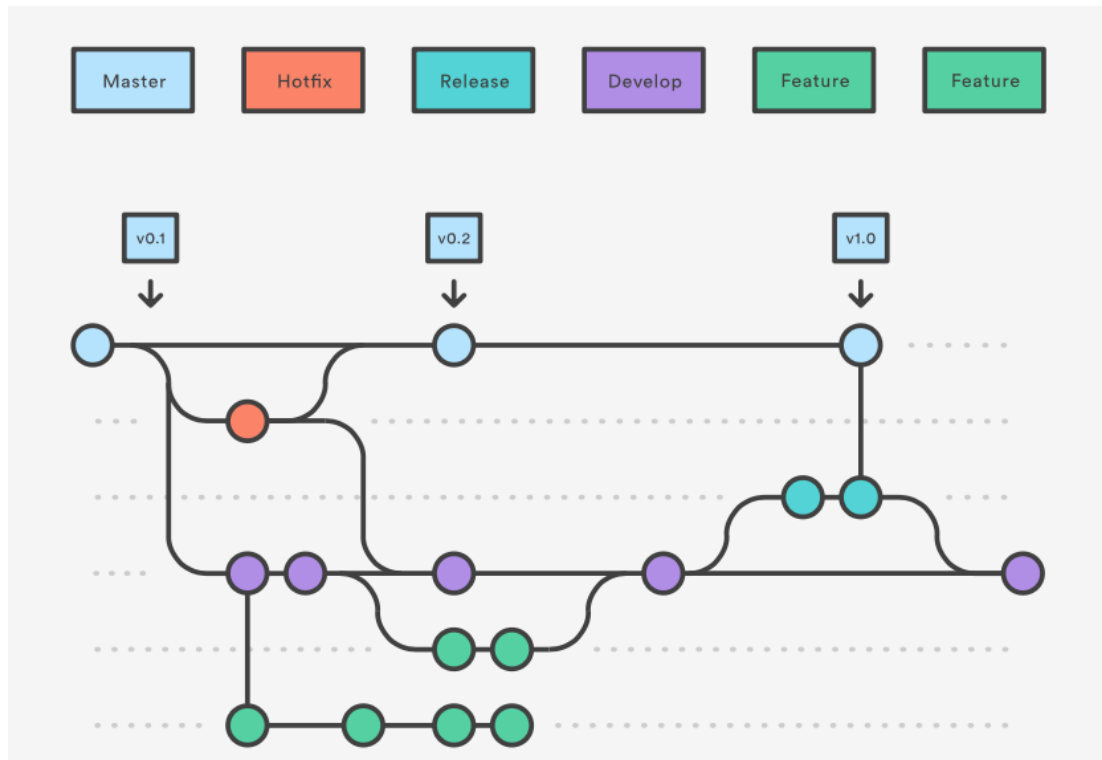


Figura 10.2: Gitflow [7]

10.4.1 Convención de branching

Master

Rama dedicada a la versión en producción de la aplicación. Al culminar el desarrollo de la misma, se impactan los cambios sobre esta rama, dejando el código del sistema en producción.

Develop

Rama con el código estable durante el desarrollo con todas las funcionalidades comprometidas al finalizar el *sprint*. Es importante aclarar que en esta rama la aplicación debe ser funcional en todo sentido de acuerdo a lo comprometido en el *sprint*.

Por otro lado, al utilizar *Heroku* como plataforma de servicio, nos permitió habilitar la opción de *deploy* automático del código (*Automatic deploy*) asociándolo a esta rama. Esto nos ayudó a automatizar el *deploy* en *Heroku* simplemente subiendo los cambios en esta rama, lo que se conoce como *continuous deployment*.

Feature

Por cada funcionalidad distinta del *sprint* cada miembro trabaja en una rama con el nombre de la *feature* a desarrollar.

Al finalizar el *sprint* se realiza un *Pull Request* en donde todos los miembros validan los cambios implementados, y si se logra un consenso se realiza el *merge* sobre la rama *develop* para que todas las funcionalidades del *sprint* queden impactadas sobre dicha rama, a esta práctica se le denomina *continuous integration*.

Hotfix

Esta rama tiene como función realizar cambios rápidos que resuelvan algún defecto específico que haya surgido tanto a nivel de código que se encuentre en desarrollo en una *feature* como a nivel de código existente en *develop*.

10.5 Gestión de cambios

Para llevar una correcta gestión de cambios el equipo optó por definir un proceso el cual seguir en caso de que surjan nuevos cambios ya sea al finalizar el *sprint* o durante el mismo.

A continuación se muestra el proceso en notación *BPM* [78] definido por el equipo.

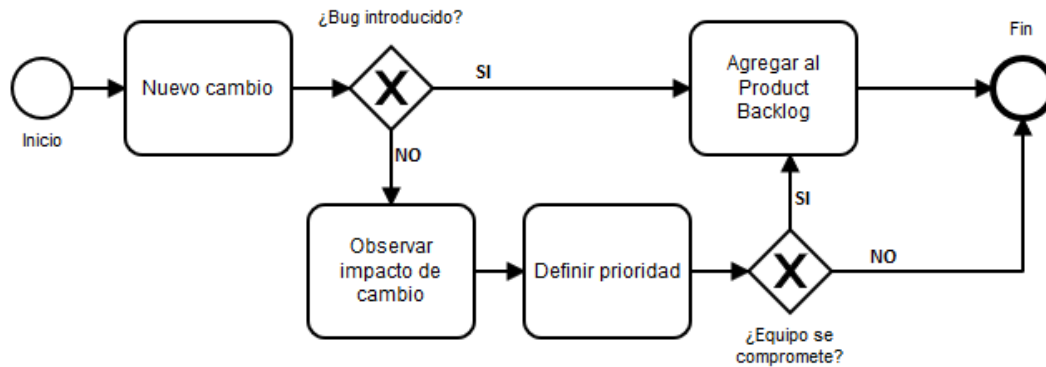


Figura 10.3: Gestión de cambios

Al surgir un nuevo cambio, ya sea por petición del cliente como por petición de algún integrante del equipo, ya sea durante el *sprint* o al finalizar el mismo, el equipo evalúa en primera instancia el tipo de cambio.

Si se trata de corrección de algún defecto, entonces el cambio es aprobado y agregado al *Product backlog* para su posterior desarrollo.

En cambio, si se trata de una nueva funcionalidad, el equipo evalúa primero el impacto que tendrá sobre la aplicación, acto seguido define una prioridad al mismo en base al impacto y valor aportado al cliente.

Si el equipo aprueba dicho cambio entonces se agrega al *Product backlog* para su posterior desarrollo, en caso contrario el cambio se descarta.

10.6 Gestión de incidentes

Para realizar una eficiente gestión y llevar un registro de los incidentes durante el *sprint* y el desarrollo el equipo se comunica mediante *Whatsapp* y/o *Slack* dejan-

do un registro del incidente surgido para que el equipo posteriormente pueda en primera instancia priorizarlo y si es viable el cambio entonces resolverlo durante el *sprint*, en caso contrario se deje abierto para resolver en el siguiente, siguiendo el proceso de gestión de cambios de la Figura 10.3.

De esta manera si surge un incidente similar, el equipo cuenta con la posibilidad de verificar el histórico permitiendo una respuesta rápida ante el mismo.

10.7 Conclusiones de gestión de configuración

La gestión de configuración fue de gran importancia para poder llevar un control de las distintas versiones de cada elemento de configuración de software (*ECS*).

El uso de *Git* como sistema de control de versiones de código fuente, permitió al equipo poder trabajar en paralelo sobre distintas *features* de manera ordenada, y a su vez mantener distintas versiones del sistema en paralelo.

Por otro lado, *GitHub* nos ayudó al momento de obtener métricas sobre la cantidad de *commits* por miembro del equipo, así como también la frecuencia de actividad durante el proyecto.

Cabe destacar la sencillez de vincular *GitHub* con *Heroku* [89] para poder realizar *continuous deployment* y automatizar el proceso de *deploy*. De esta forma nos aseguramos que el equipo contara con la última versión del código fuente ejecutándose en un ambiente remoto.

10.8 Lecciones aprendidas

El uso de *gitflow* como metodología de trabajo nos resultó favorable dado que permitió a cada integrante trabajar en una determinada funcionalidad o *feature* de manera independiente. Esto le permitió al equipo desarrollar con mayor velocidad, pudiendo enfocarse en funcionalidades específicas y de esta manera avanzar sobre el *sprint* en paralelo.

El uso de *pull requests* nos ayudó a verificar los cambios introducidos por cada integrante, para posteriormente ser validados por el resto del equipo. Esta instancia nos permitió detectar defectos y evitar introducirlos a la rama principal.

Como contrapartida, consideramos que la gestión de incidentes desaprovechó la oportunidad de utilizar la funcionalidad de manejo de incidentes provista por *GitHub*. Teniendo un control y un registro más organizado y fino por cada incidente detectado en el sistema. Esto ayudaría a evitar futuros *bugs*.

11 Conclusiones del proyecto

11.1 Estado actual

Se entregó un producto que cumple con todos los requerimientos funcionales a excepción del pago grupal. Para implementar este requerimiento pendiente se documentó un posible enfoque que se encuentra disponible en la sección 11.4.1.

Dentro de los requerimientos ¹ implementados se destacan los siguientes:

Contactar y negociar con un proveedor

Para esta funcionalidad se desarrollo un *chat* y además efectuar proposiciones. Estas proposiciones pueden ser ajustadas a los parámetros establecidos en la oferta o a merced del consumidor del servicio, dándole la posibilidad al dueño de la oferta de aceptar o rechazar la oferta.

Pagos con coopies utilizando blockchain

Al momento de aceptar la proposición, se procesa el pago disparando una transacción en la red *blockchain* entre los usuarios involucrados en el intercambio. Dada la naturaleza de *blockchain*, hacerse efectiva la transacción puede demorar un tiempo que no condice con los del usuario, por lo que al terminar esta se notifica a los involucrados.

Para cumplir con esto se desarrollaron los siguientes componentes:

- Aplicación *PWA* (*frontend*)
- Servidor central (*backend*)

¹Requerimientos funcionales: 5.3.1

- Servidor dedicado a *blockchain* (*ethereum*)

A nivel de requerimientos no funcionales, dentro de los relevados durante la ingeniería de requerimientos 5.3.2, destacamos los siguientes:

Usabilidad

Se desarrollo una *Progressive Web App* (PWA) 7.6.2, lo que nos permitió desarrollar una única interfaz gráfica la cual puede ser utilizada en todo tipo de dispositivos. Agregado a esto, nos brinda la posibilidad de asegurar la funcionalidad cuando no se cuenta con conectividad a internet.

Gracias a las recomendaciones de una de las revisiones, decidimos corroborar nuestra usabilidad mediante pruebas de usuario. Estas mostraron buenos resultados y oportunidades de mejora, que fueron estudiadas e implementadas en el producto final.

Seguridad

Nuestro foco a nivel de seguridad se basa en la estructura que provee *blockchain* para registrar las transacciones en sus bloques, gracias a su descentralización podemos asegurar la permanencia e integridad de las mismas. Cada vez que un usuario en *Coopify* recibe *coopies* existe una transacción detrás que respalda ese movimiento.

Con respecto a la seguridad “tradicional” ², o las buenas practicas entendemos que hay un margen de mejora. Como el objetivo del proyecto era desarrollar un *MVP*, estas no entraron en el alcance previsto. Sin embargo, el equipo dio los primeros pasos para remediar esta situación desarrollando parcialmente ³ algunas de estas prácticas.

²registro, autenticación, doble verificación de identidad, etc.

³Se dejo implementado en una rama en *Git* sin *mergear* con la ultima solución

11.2 Conclusiones

Ingeniería de requerimientos

Gracias a la contrapropuesta realizada por el equipo, pudimos ponernos en sintonía con el cliente y sus necesidades. Para validar los requerimientos inicialmente se utilizaron técnicas como: prototipos de baja fidelidad, entrevistas con el cliente e ingeniería inversa de competidores. Otra técnica utilizada posteriormente, antes de comenzar cada *sprint*, fue validar estos requerimientos a implementar mediante reuniones con el cliente.

Arquitectura

Nos basamos en una arquitectura que cumpliera con todos los atributos de calidad especificados en los requerimientos no funcionales.

- **Performance:** Separar el servidor de *Etherum* respecto del de *Backend*, nos sirvió en cuanto al manejo de los tiempos, haciendo que el mismo se encargue solo de lo concerniente a las transacciones y consultas de balances de las cuentas. Y que cuando realmente queda la transacción en la red *blockchain*, se notifique a los usuarios que intervienen en el intercambio mediante *Pusher*, para que sea de forma asíncrona.
- **Mantenibilidad:** La separación de los servidores detallada en el punto anterior, de performance, nos ayuda en cuanto a mantenibilidad porque nos da la posibilidad de que se pueda cambiar lo relacionado a la red *blockchain* de forma independiente, sin tener que recompilar lo más utilizado que son las funcionalidades provistas por *Backend*.
- **Usabilidad:** Importante en cuanto a usabilidad fue haber decidido utilizar *PWA* en lugar de crear aplicaciones nativas para cada sistema operativo. Ya que nos resumió bastante el manejo de la desconexión de internet, para que el usuario sienta la sensación de seguir conectado casi siempre.
- **Seguridad:** Tener encapsulado el manejo de lo relacionado a *Ethereum*, así como los datos sensibles que nos permiten realizar transacciones, dentro de un servidor dedicado solo a eso, es importante en cuanto a la seguridad. De todas formas somos conscientes que el alcance que nos propusimos sobre el proyecto, no llegaba a cubrir un nivel de seguridad mayor al que implementamos. Por medio de consultas a expertos ideamos una implementación en base a

doble autenticación, que no introducimos en el código fuente a entregar, por haber surgido implementaciones más importantes. De todas formas se puede utilizar ese código en un futuro por parte del cliente, ya que esta en una rama aparte en el repositorio.

Gestión del proyecto

La gestión del proyecto se llevo a cabo en dos etapas, una dedicada a la investigación y capacitación del equipo en las tecnologías a utilizar, y otra a la elaboración del producto. Para ambas etapas decidimos utilizar metodologías ágiles, principalmente debido a la volatilidad de los requerimientos. En la primera parte utilizamos *Kanban* mientras que para la segunda parte del proyecto utilizamos *Scrum*.

El equipo invirtió un gran esfuerzo en este proceso, en consecuencia el resultado del mismo fue satisfactorio. Se destaca la dedicación que el equipo mostró a lo largo del proyecto permitiéndonos cumplir, casi en la totalidad de los *sprints*, con los objetivos planificados. En términos de: esfuerzo comprometido y logrado, tiempo estimado e invertido o velocidad, el equipo estuvo a la altura del desafío.

Gestión de calidad

La variedad de pruebas utilizadas nos permitió asegurar que se entregaba un producto final con la menor cantidad de defectos posibles, posibilitando detectar la introducción de defectos y solucionarlos pro activamente.

De las retrospectivas surgieron variedad de planes de acción que fueron implementados a lo largo del proyecto, con el fin de continuar mejorando la calidad del producto entregado.

11.3 Lecciones aprendidas

Ingeniería de requerimientos

Respecto a los requerimientos y su análisis, para futuros proyectos sería provechoso:

- Incrementar las instancias iniciales de relevamiento. Durante el desarrollo, al planificar sucedió en múltiples ocasiones encontrar requerimientos especificados sin la profundidad esperada por el equipo.
- Continuar utilizando prototipos de baja fidelidad para detectar problemas y desafíos de manera temprana. No es necesario contar con un prototipo detallado para obtener buenos resultados de esta técnica.

Arquitectura

En *Frontend*, nos faltó encapsular en componentes generales algunos de los que usamos en forma reiterativa. En algunos casos lo hicimos obteniendo buenos resultados y favoreciendo la mantenibilidad. Por ejemplo en el caso del listado de proposiciones y listado de servicios, generalizando su comportamiento en un mismo componente.

La utilización de *Heroku* fue una buena elección por la facilidad que nos dio para los procesos de despliegue de las aplicaciones implementadas. No tener que preocuparnos demasiado para su configuración fue algo bueno.

Mantener el mismo *stack* tecnológico (*Javascript*) más que nada en los servidores de *Backend* y *Ethereum* fue una buena decisión por ser partes centrales del sistema y es favorable en cuanto a mantenibilidad.

Gestión de Proyecto

El mismo se dividió en dos secciones, una para investigación y otra para desarrollo.

- **Etapas de Investigación:**
 - Utilizar metodologías ágiles como *Kanban*, nos permitió encontrarle un ciclo de vida y proceso acorde a lo que requería esta etapa.
 - Si bien logramos obtener buenos resultados pero a costa de tiempo, de repente un enfoque más práctico hubiera hecho que se consiguieran

mejores resultados al finalizar el proyecto. Por ejemplo comenzar con la etapa de desarrollo con una capacitación parcial pero suficiente para afrontar los primeros desafíos e intercalar pequeñas instancias para capacitarnos en algún tema particular.

■ Etapa de Desarrollo:

- Entendemos que si bien no hubieron mayores desvíos respecto a lo planificado, la causa de estos siempre la pudimos trazar hasta la gestión de la calidad. El ser laxos al comienzo en esta categoría nos implico invertir tiempo de futuros *sprints* en ponernos al día. En este aspecto existe un amplio margen de mejora en como se llevo a cabo esta gestión.
- El llevar la gestión del proyecto a lo largo del mismo nos permitió enfrentarnos a problemas que ni siquiera concebíamos. Desafíos dentro y fuera del grupo y lo tecnológico que nos obligaron a crecer como personas y como profesionales. (relacionado a los múltiples desafíos que se evidencian en cada una de las gestiones que están dentro de gestión de proyecto).

Gestión de calidad

Podríamos haber utilizado alguna herramienta para automatizar las pruebas de interfaz gráfica, como por ejemplo *Selenium*⁴. Nos hubiera permitido aprovechar mejor las bondades de *CircleCI* para que cada vez que se fuera a hacer *deploy* saber si estas pruebas llegaron a buen puerto o no. De todas formas pudimos solventar esta deficiencia mediante la planilla de pruebas funcionales de sistema, para probar el comportamiento en cada *sprint*.

Si bien al construir las funcionalidades fuimos creando pruebas unitarias en *backend*, consideramos que podríamos haber construido escenarios mas exhaustivos. Ya que hay varios casos de borde que se hicieron visibles con la ejecución de pruebas funcionales de sistema.

Otro punto de mejora seria un mejor uso de los correctores automáticos de código (*linters*), ya que las reglas utilizadas por ellos no fueron lo suficientemente exhaustivas desde un principio. Eso genero que el equipo terminara invirtiendo un tiempo considerable en subsanar la situación.

⁴Entorno de pruebas automatizado para *frontend*, ver [63]

11.4 Próximos pasos

La idea del cliente se basó en la implementación de un *MVP*, para luego en base al mismo presentar el proyecto a la *ANII* ⁵. El motivo de esa presentación, es para conseguir un financiación para poder llevar el proyecto a otra etapa, poder seguir implementando otras funcionalidades necesarias, que no entraban en el alcance a cubrir en primera instancia y ponerlo en producción.

A continuación detallaremos posibles mejoras a lo ya implementado, para que se puedan tomar en cuenta a futuro.

11.4.1 Posibles mejoras

11.4.1.1 Borrar usuario

Si bien el sistema no tiene implementada una funcionalidad para borrar usuarios (no es la idea poder borrar un usuario), si por cualquier motivo se borra la tupla correspondiente al usuario en la base de datos de *Backend*, eso borra en cascada todo lo correspondiente a ese usuario, pero hay cosas del usuario que no están bajo el contexto exclusivo de *Backend*.

Nos referimos a la base de datos perteneciente al servidor *Ethereum*, este guarda la información de *Id* de usuario y *address* (correspondiente a la dirección o billetera del usuario en *blockchain*).

Esto significa que al borrar el usuario de la base de datos de *Backend*, tendríamos una *address* en *blockchain* con *copies* que quedan vagando por la red y que no se podrían utilizar más mediante la aplicación *Coopify*. Por lo tanto lo bueno sería en ese caso, transferir los *copies* de la *address* de ese usuario a la *address* de la cuenta principal de *Coopify* desde donde se extraen *copies* siempre, al utilizar la plataforma.

11.4.1.2 Pago grupal

Una de las funcionalidades que atacaría el equipo en caso de contar con uno o dos *sprints* más de desarrollo es la de realizar pagos grupales. La idea de esta funciona-

⁵ *ANII*: Agencia Nacional de Investigación e Innovación [6]

lidad es la de darle la posibilidad a un conjunto de usuarios que puedan contribuir entre todos para comprar algún servicio. Por ejemplo un grupo de amigos que desean celebrar alguna ocasión y deciden pagar algún servicio de entretenimiento.

A lo largo del proyecto el equipo fue conversando y discutiendo para encontrar la mejor solución para implementar funcionalidad. De estos intercambios identificamos que:

- Una solución que reserve saldo actual no es viable pues no está provista por *Ethereum* y tendríamos que idear una solución casera que tome en cuenta estos saldos reservados.
- Una solución que realice pagos directos entre ambas partes tampoco es viable por que existen varios casos en los que una parte puede pagar y no llegar al monto adecuado, lo ideal seria devolverle el monto después de un periodo de tiempo predeterminado.

Por lo tanto la solución que planteamos es la siguiente:

- Crear billeteras intermedias para los pagos grupales, de esa forma no tenemos que reservar saldo de los usuarios que están colaborando, pero tampoco estamos adelantándole saldo al proveedor del servicio.
- Crear un sistema de procesos automatizados que se ejecuten periódicamente y detecten pagos grupales inactivos y retorne el saldo a los usuarios que colaboraron, evitando que pierdan saldo por no concretarse el negocio.

El diseño de la solución se realizó con el supuesto de que la forma más sencilla de implementar esta funcionalidad es no restringir a los usuarios sobre el monto que van a aportar cada uno. Tomando un enfoque similar a los de otras plataforma de *crowdfunding* ⁶ esta funcionalidad puede ser mucho más exitosa que restringiéndola a un saldo predefinido por usuario. Además de esto se impulsa realmente la cooperación entre iguales para completar una meta común, que es uno de los valores centrales de *Coopify* como plataforma, la cooperación.

⁶ *Crowdfunding*: financiación de masas, ver referencia bibliográfica [96]

11.4.1.3 Abastecimiento de Ether a cuenta principal de Coopify

Como ya hemos explicado antes, para que *Coopify* funcione, digamos, para que se puedan efectuar las transacciones de un usuario a otro, necesitamos que cada cuenta tenga una cantidad de *Ether* para poder solventar el *GAS* que consume la transacción. Para profundizar sobre el tema, ver referencia 7.4.

Por lo que necesitamos que la cuenta principal⁷ que posee *Coopify*, tenga *Ether*. Como lo que estamos desarrollando es un *MVP* y la red *blockchain* que estamos utilizando es de prueba, la carga de *Ether* de prueba se efectúa de manera gratuita. Pero el día de mañana, cuando se quiera poner en producción, se necesitara contar con *Ether* real en una cuenta de la red principal (*Main Chain*)⁸ de *Ethereum*.

Esta cuenta principal se podría cargar con alguna billetera virtual como *Metamask* y sus servicios externos como *Wyre* y *Coinswitch* para la compra de esos *Ether* mediante tarjeta de crédito por ejemplo [46].

11.4.1.4 Eliminación de servicios

Al tratarse de un *MVP*, en muchos casos los requerimientos y consiguiente desarrollo de los mismos no conforman lo que sería un *ABM* completo. Este es el ejemplo de los servicios. Donde desarrollamos la creación del mismo y no la eliminación. Requerimientos importantes a la hora del manejo en producción.

Por lo que a futuro sería algo a implementar.

11.4.1.5 Backoffice para usuarios administradores

Vemos como una posible mejora, la creación de un *Backoffice*, que permita a los usuarios administradores gestionar varios *ítems*, por ejemplo:

- Manejo de las categorías. Para poder crear, modificar o eliminar las mismas, según su uso.
- Eliminación de servicios.

⁷Address en la red *blockchain* de *Ethereum*. *Rinkeby test network*. Es la que posee *Ether* y *Coopies* para distribuir

⁸*Ethereum* cuenta con diferentes redes *blockchain*, unas de prueba (*Rinkeby Test Network* y *Ropsten Test Network*) y una principal (*Main Chain*)

- Administración de usuarios.
- Verificación de Saldo de *Ether* y *Coopies* de la cuenta principal.
- Consumir servicio externo que permita cargar *Ether* mediante pago con tarjeta de crédito, similar al funcionamiento de *Metamask*.

12 Referencias Bibliográficas

- [1] Truffle Blockchain Group 2019. “Ganache, one click blockchain”. [En línea] Disponible: <https://www.trufflesuite.com/ganache>, [Último acceso 02 de Setiembre de 2019].
- [2] Proyectos Ágiles. “Desarrollo iterativo e incremental”. [En línea] Disponible: <https://proyectosagiles.org/desarrollo-iterativo-incremental/>, [Último acceso 02 de Setiembre de 2019].
- [3] Proyectos Ágiles. “SCRUM”. [En línea] Disponible: <https://proyectosagiles.org/que-es-scrum/>, [Último acceso 02 de Setiembre de 2019].
- [4] José Manuel Alarcón. “Qué son las Aplicaciones Web Progresivas o Progressive Web Apps”. [En línea] Disponible: <https://www.campusmvp.es/recursos/post/Que-son-las-Aplicaciones-Web-Progresivas-o-Progressive-Web-Apps.aspx>, [Último acceso 02 de Setiembre de 2019].
- [5] Miguel Angel Alvarez. “Qué es la autenticación por token”. [En línea] Disponible: <https://desarrolloweb.com/articulos/autenticacion-token.html>, [Último acceso 02 de Setiembre de 2019].
- [6] ANII. “Agencia Nacional de Investigación e innovación”. [En línea] Disponible: <https://www.anii.org.uy/>, [Último acceso 02 de Setiembre de 2019].
- [7] Atlassian. “Gitflow Workflow”. [En línea] Disponible: <https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow>, [Último acceso 02 de Setiembre de 2019].
- [8] Atlassian. “Trello”. [En línea] Disponible: <https://trello.com/>, [Último acceso 02 de Setiembre de 2019].
- [9] Betty Avanzini. “Objetivos SMART: cómo definirlos y 4 ejemplos reales”. [En línea] Disponible: <https://blog.cool-tabs.com/es/objetivos-smart-definirlos-ejemplos/>, [Último acceso 02 de Setiembre de 2019].
- [10] Víctor Martínez Bahillo. “Hyperledger. Una breve introducción”. [En línea] Disponible: <https://www.versia.com/blog/hyperledger-una-breve-introduccion.html>, [Último acceso 02 de Setiembre de 2019].

- [11] Jordi Cabot. “La diferencia entre transpilación y compilación”. [En línea] Disponible: <https://ingenieriadesoftware.es/diferencia-transpilacion-compilacion/>, [Último acceso 02 de Setiembre de 2019].
- [12] ISO 9001 Calidad. “Elementos de un proceso”. [En línea] Disponible: <https://iso9001calidad.com/elementos-de-un-proceso-30.html>, [Último acceso 02 de Setiembre de 2019].
- [13] Manuel Guerrero Cano. “Método Kanban. Herramienta Kaizen que podemos utilizar en nuestro grupo Scrum”. [En línea] Disponible: <https://manuelguerrerocano.wordpress.com/2014/11/09/metodo-kanban-herramienta-kaizen-que-podemos-utilizar-en-nuestro-grupo-scrum/>, [Último acceso 02 de Setiembre de 2019].
- [14] Javier Chamorro. “Ciclo de vida del software”. [En línea] Disponible: <https://ingsw.pbworks.com/f/Ciclo+de+Vida+del+Software.pdf>, [Último acceso 02 de Setiembre de 2019].
- [15] Kaven Choi. “Etherscan blog”. [En línea] Disponible: <https://medium.com/etherscan-blog/about>, [Último acceso 02 de Setiembre de 2019].
- [16] CicloDeVidaSoft. “Ciclo de Vida del software”. [En línea] Disponible: <http://ciclodevidasoft.blogspot.com/2012/07/modelos.html>, [Último acceso 02 de Setiembre de 2019].
- [17] Gonzalo Sánchez D. “¿Qué es MVC?” [En línea] Disponible: <https://medium.com/tecnologia-digital/que-es-mvc-a84cdc2ed088>, [Último acceso 02 de Setiembre de 2019].
- [18] DCCoder. “Code Lint – What is it? What can help?” [En línea] Disponible: <https://dccoder.com/2018/08/code-lint/>, [Último acceso 02 de Setiembre de 2019].
- [19] Google Developers. “Service Worker”. [En línea] Disponible: <https://developers.google.com/web/fundamentals/primers/service-workers/?hl=es>, [Último acceso 02 de Setiembre de 2019].
- [20] Microsoft Docs. “Uso del servicio de correo electrónico SendGrid”. [En línea] Disponible: <https://docs.microsoft.com/es-es/azure/sendgrid-dotnet-how-to-send-email>, [Último acceso 02 de Setiembre de 2019].
- [21] Bill Brock- VP of Engineering Nashville TN. “The Pros and Cons of Hyperledger Fabric”. [En línea] Disponible: <https://www.verypossible.com/blog/the-pros-and-cons-of-hyperledger-fabric>, [Último acceso 02 de Setiembre de 2019].
- [22] Antoine Eripret. “Introducción a las Progressive Web Apps (PWA)”. [En línea] Disponible: <https://seocom.agency/es/blog/introduccion-pwa/>, [Último acceso 02 de Setiembre de 2019].

- [23] Ethereum. “Especificación de Application Binary Interface”. [En línea] Disponible: <https://solidity-es.readthedocs.io/es/latest/abi-spec.html>, [Último acceso 02 de Setiembre de 2019].
- [24] Ethereum. “Etherscan”. [En línea] Disponible: <https://rinkeby.etherscan.io/>, [Último acceso 02 de Setiembre de 2019].
- [25] ethereum.org. “Ethereum Beginners”. [En línea] Disponible: <https://www.ethereum.org/beginners/>, [Último acceso 02 de Setiembre de 2019].
- [26] ethereum-org. “Homestead Release”. [En línea] Disponible: <https://github.com/ethereum/ethereum-org/blob/master/solidity/token-erc20.sol>, [Último acceso 02 de Setiembre de 2019].
- [27] fabrictestdocs.readthedocs.io. “What is chaincode?” [En línea] Disponible: <https://fabrictestdocs.readthedocs.io/en/latest/chaincode.html>, [Último acceso 02 de Setiembre de 2019].
- [28] Fernando Finelli. “10 reglas heurísticas de usabilidad de Jakob Nielsen”. [En línea] Disponible: <http://www.braintive.com/10-reglas-heuristicas-de-usabilidad-de-jakob-nielsen/>, [Último acceso 02 de Setiembre de 2019].
- [29] Paul Gadi. “Let’s Build A Decentralized Game Economy Using Blockchains [Part 2]”. [En línea] Disponible: <https://medium.com/@polats/lets-build-a-decentralized-game-economy-using-blockchains-part-2-ec72f6e1b865>, [Último acceso 02 de Setiembre de 2019].
- [30] Laia Gilibets. “Qué es la metodología Kanban y cómo utilizarla”. [En línea] Disponible: <https://www.iebschool.com/blog/metodologia-kanban-agile-scrum/>, [Último acceso 02 de Setiembre de 2019].
- [31] JSON-RPC Working Group. “JSON-RPC 2.0 Specification”. [En línea] Disponible: <https://www.jsonrpc.org/specification>, [Último acceso 02 de Setiembre de 2019].
- [32] PostgreSQL Global Development Group. “PostgreSQL: The World’s Most Advanced Open Source Relational Database”. [En línea] Disponible: <https://www.postgresql.org/>, [Último acceso 02 de Setiembre de 2019].
- [33] Ruby Hanson. “Advantages and disadvantages of ethereum”. [En línea] Disponible: <http://riversihaveknown.com/advantages-and-disadvantages-of-ethereum/>, [Último acceso 02 de Setiembre de 2019].
- [34] James Hom. “Prototipado de Baja Fidelidad”. [En línea] Disponible: <http://www.sidar.org/recur/desdi/traduc/es/visitable/tecnicas/Low.htm>, [Último acceso 02 de Setiembre de 2019].
- [35] Ivan Flores Hurtado. “Hyperledger Fabric — En 4 minutos”. [Último acceso 02 de Setiembre de 2019].
- [36] IBM. “Hyperledger”. [En línea] Disponible: <https://www.ibm.com/es-es/blockchain>, [Último acceso 02 de Setiembre de 2019].

- [37] Facebook Inc. “Una biblioteca de JavaScript para construir interfaces de usuario”. [En línea] Disponible: <https://es.reactjs.org/>, [Último acceso 02 de Setiembre de 2019].
- [38] Blog Ingeniería. “¿Qué es la ingeniería inversa?” [En línea] Disponible: <https://blogingenieria.com/general/ingenieria-inversa/>, [Último acceso 02 de Setiembre de 2019].
- [39] Alex Ballarin Latre. “What is a Proxy Product Owner? Why it is found so often?” [En línea] Disponible: <https://www.scrum.org/resources/blog/what-proxy-product-owner-why-it-found-so-often>, [Último acceso 02 de Setiembre de 2019].
- [40] lll-docs.readthedocs.io. “LLL Introduccion”. [En línea] Disponible: https://lll-docs.readthedocs.io/en/latest/lll_introduction.html, [Último acceso 02 de Setiembre de 2019].
- [41] Ravinder Mahajan. “Why Use Redux Saga?” [En línea] Disponible: <https://medium.com/@ravindermahajan/why-use-redux-saga-f3413a3f7e34>, [Último acceso 02 de Setiembre de 2019].
- [42] Azat Mardan. “You Don’t Know Node: Quick Intro to Core Features”. [En línea] Disponible: <https://webapplog.com/you-dont-know-node/>, [Último acceso 02 de Setiembre de 2019].
- [43] Philipp Sandner Martin Valenta. “Comparison of Ethereum, Hyperledger Fabric and Corda”. [En línea] Disponible: <https://medium.com/@philippsandner/comparison-of-ethereum-hyperledger-fabric-and-corda-21c1bb9442f6>, [Último acceso 02 de Setiembre de 2019].
- [44] Raúl Jaime - Director del Master en Blockchain y Fintech en IEBS Business School. “Blockchain y desarrollo Ethereum”. [En línea] Disponible: <https://www.territoriobitcoin.com/wp-content/uploads/2018/10/Blockchain-y-desarrollo-Ethereum.pdf>, [Último acceso 02 de Setiembre de 2019].
- [45] Gonzalo Amuchastegui Mayoral. “Por qué es importante desarrollar un MVP”. [En línea] Disponible: <https://medium.com/tmp-app/por-qu%C3%A9-es-importante-desarrollar-un-mvp-e54dc4e43096>, [Último acceso 02 de Setiembre de 2019].
- [46] metamask.io. “MetaMask, Brings Ethereum to your browser”. [En línea] Disponible: <https://metamask.io/>, [Último acceso 02 de Setiembre de 2019].
- [47] miethereum. “¿Que es el GAS en Ethereum?” [En línea] Disponible: <https://www.miethereum.com/ether/gas/#toc1>, [Último acceso 02 de Setiembre de 2019].

- [48] miethereum. “¿Que es una DAPP?” [En línea] Disponible: <https://www.miethereum.com/smart-contracts/dapps/#toc1>, [Último acceso 02 de Setiembre de 2019].
- [49] Pablo Olóndriz - miethereum. “Todos los recursos sobre el lenguaje de programación de los smart contracts”. [En línea] Disponible: <https://miethereum.com/smart-contracts/solidity/>, [Último acceso 02 de Setiembre de 2019].
- [50] Alex Moldovan. “Breathing air into AirBnB’s JavaScript Style Guide”. [En línea] Disponible: <https://www.freecodecamp.org/news/adding-some-air-to-the-airbnb-style-guide-3df40e31c57a/>, [Último acceso 02 de Setiembre de 2019].
- [51] Carlos Monreal. “Ciclos de vida iterativo e incremental, ¿Qué son?” [En línea] Disponible: <https://www.cursodireccionproyectos.com/2018/07/ciclos-de-vida-iterativo-e-incremental-que-son/>, [Último acceso 02 de Setiembre de 2019].
- [52] Neel. “A Beginner’s Guide to Blockchain Technology”. [En línea] Disponible: <https://www.edureka.co/blog/blockchain-tutorial>, [Último acceso 02 de Setiembre de 2019].
- [53] Javier Pastor. “Qué es blockchain: la explicación definitiva para la tecnología más de moda”. [En línea] Disponible: <https://www.xataka.com/especiales/que-es-blockchain-la-explicacion-definitiva-para-la-tecnologia-mas-de-moda>, [Último acceso 02 de Setiembre de 2019].
- [54] Carlos de Pedro. “El minado en Blockchain”. [En línea] Disponible: <https://cysae.com/el-minado-en-blockchain/>, [Último acceso 02 de Setiembre de 2019].
- [55] polbac. “React vs Angular. La historia detrás de una elección”. [En línea] Disponible: <https://medium.com/trocafone/react-vs-angular-la-historia-detras-de-una-eleccion-5b955ad9be70>, [Último acceso 02 de Setiembre de 2019].
- [56] Prerna. “Hyperledger vs Ethereum – Which Blockchain Platform Will Benefit Your Business?” [En línea] Disponible: <https://www.edureka.co/blog/hyperledger-vs-ethereum/>, [Último acceso 02 de Setiembre de 2019].
- [57] Pusher. “Channels overview || *Pusherdocs*”. [En línea] Disponible: <https://pusher.com/docs/channels>, [Último acceso 02 de Setiembre de 2019].
- [58] React. “¿Qué es el DOM virtual?” [En línea] Disponible: <https://es.reactjs.org/docs/faq-internals.html>, [Último acceso 02 de Setiembre de 2019].

- [59] redislabs. “Ethereum Testnet Blockchain: Ropsten, Rinkeby, Kovan”. [En línea] Disponible: <https://bit2main.com/ethereum-testnet/>, [Último acceso 02 de Setiembre de 2019].
- [60] redislabs. “Redis”. [En línea] Disponible: <https://redis.io/>, [Último acceso 02 de Setiembre de 2019].
- [61] Remix. “Welcome to Remix documentation!” [En línea] Disponible: <https://remix-ide.readthedocs.io/en/latest/>, [Último acceso 02 de Setiembre de 2019].
- [62] Jose Segovia. “Ventajas y Desventajas de PostgreSQL”. [En línea] Disponible: <https://todopostgresql.com/ventajas-y-desventajas-de-postgresql/>, [Último acceso 02 de Setiembre de 2019].
- [63] Selenium. “Selenium Web Browser Automation”. [En línea] Disponible: <https://www.seleniumhq.org/>, [Último acceso 02 de Setiembre de 2019].
- [64] Agile Sensei. “Continuous Innovation Change with PopcornFlow”. [En línea] Disponible: <https://popcornflow.com/>, [Último acceso 02 de Setiembre de 2019].
- [65] solidity-es.readthedocs.io. “Solidity”. [En línea] Disponible: <https://solidity-es.readthedocs.io/es/latest/>, [Último acceso 02 de Setiembre de 2019].
- [66] Jeff Sutherland. “Iterative vs. Incremental Development”. [En línea] Disponible: <https://www.scruminc.com/iterative-vs-incremental-development/>, [Último acceso 02 de Setiembre de 2019].
- [67] Andrew Tar. “Prueba de Trabajo (PoW), Explicado”. [En línea] Disponible: <https://es.cointelegraph.com/explained/proof-of-work-explained>, [Último acceso 02 de Setiembre de 2019].
- [68] Atyantik technologies. “Paw JS - The plugable framework”. [En línea] Disponible: <https://github.com/Atyantik/pawjs>, [Último acceso 02 de Setiembre de 2019].
- [69] theethereum.wiki. “Accounts, Addresses, Public And Private Keys, And Tokens”. [En línea] Disponible: https://theethereum.wiki/w/index.php/Accounts,_Addresses,_Public_And_Private_Keys,_And_Tokens, [Último acceso 02 de Setiembre de 2019].
- [70] Universidad ORT Uruguay. “Hoja de verificación de pautas de presentación de trabajos finales de carreras de la facultad de ingeniería - excepto biotecnología”. [En línea] Disponible: <http://www.ort.edu.uy/fi/documentos/hoja-de-verificacion-de-pautas-de-presentacion-de-trabajos-finales-de-carrera.pdf>, [Último acceso 02 de Setiembre de 2019].
- [71] Universidad ORT Uruguay. “Laboratorio ORT Software Factory”. [En línea] Disponible: <http://fi.ort.edu.uy/innovaportal/v/3393/5/fi.ort.front/inicio.html>, [Último acceso 02 de Setiembre de 2019].

- [72] Universidad ORT Uruguay. “Normas específicas para la presentación de trabajos finales de carrera (TFDC)”. [En línea] Disponible: <https://fi.ort.edu.uy/innovaportal/file/30964/3/documento-302-et-26102018.pdf>, [Último acceso 02 de Setiembre de 2019].
- [73] Universidad ORT Uruguay. “Orientación para títulos, resúmenes o abstracts e informes de corrección de trabajos finales de carrera”. [En línea] Disponible: <https://www3.ort.edu.uy/varios/pdf/documento306.pdf>, [Último acceso 02 de Setiembre de 2019].
- [74] William Wang. “Componentes React Eficientes: Una Guía para Optimizar el Desempeño de React”. [En línea] Disponible: <https://www.toptal.com/react/componentes-react-eficientes-una-guia-para-optimizar-el-desempeno-de-react>, [Último acceso 02 de Setiembre de 2019].
- [75] Wikipedia. “Ataque de intermediario”. [En línea] Disponible: https://es.wikipedia.org/wiki/Ataque_de_intermediario, [Último acceso 02 de Setiembre de 2019].
- [76] Wikipedia. “Bitbucket”. [En línea] Disponible: <https://es.wikipedia.org/wiki/Bitbucket>, [Último acceso 02 de Setiembre de 2019].
- [77] Wikipedia. “Bitcoin”. [En línea] Disponible: <https://es.wikipedia.org/wiki/Bitcoin>, [Último acceso 02 de Setiembre de 2019].
- [78] Wikipedia. “Business process management”. [En línea] Disponible: https://en.wikipedia.org/wiki/Business_process_management, [Último acceso 02 de Setiembre de 2019].
- [79] Wikipedia. “Business to business”. [En línea] Disponible: <https://es.wikipedia.org/wiki/Business-to-business>, [Último acceso 02 de Setiembre de 2019].
- [80] Wikipedia. “Business to consumer”. [En línea] Disponible: <https://es.wikipedia.org/wiki/Business-to-consumer>, [Último acceso 02 de Setiembre de 2019].
- [81] Wikipedia. “C++”. [En línea] Disponible: <https://es.wikipedia.org/wiki/C%2B%2B>, [Último acceso 02 de Setiembre de 2019].
- [82] Wikipedia. “Cadena de bloques”. [En línea] Disponible: https://es.wikipedia.org/wiki/Cadena_de_bloques, [Último acceso 02 de Setiembre de 2019].
- [83] Wikipedia. “Cadena de responsabilidad”. [En línea] Disponible: https://es.wikipedia.org/wiki/Cadena_de_responsabilidad, [Último acceso 02 de Setiembre de 2019].
- [84] Wikipedia. “Criptomoneda”. [En línea] Disponible: <https://es.wikipedia.org/wiki/Criptomoneda>, [Último acceso 02 de Setiembre de 2019].

- [85] Wikipedia. "Dynos: the heart of the Heroku platform". [En línea] Disponible: <https://www.heroku.com/dynos>, [Último acceso 02 de Setiembre de 2019].
- [86] Wikipedia. "Ethereum". [En línea] Disponible: <https://es.wikipedia.org/wiki/Ethereum>, [Último acceso 02 de Setiembre de 2019].
- [87] Wikipedia. "Go (lenguaje de programación)". [En línea] Disponible: [https://es.wikipedia.org/wiki/Go_\(lenguaje_de_programaci%C3%B3n\)](https://es.wikipedia.org/wiki/Go_(lenguaje_de_programaci%C3%B3n)), [Último acceso 02 de Setiembre de 2019].
- [88] Wikipedia. "Google AdSense". [En línea] Disponible: https://es.wikipedia.org/wiki/Google_AdSense, [Último acceso 02 de Setiembre de 2019].
- [89] Wikipedia. "Heroku". [En línea] Disponible: <https://en.wikipedia.org/wiki/Heroku>, [Último acceso 02 de Setiembre de 2019].
- [90] Wikipedia. "Hyperledger". [En línea] Disponible: <https://es.wikipedia.org/wiki/Hyperledger>, [Último acceso 02 de Setiembre de 2019].
- [91] Wikipedia. "Integración continua". [En línea] Disponible: https://es.wikipedia.org/wiki/Integracion_continua, [Último acceso 02 de Setiembre de 2019].
- [92] Wikipedia. "Java (lenguaje de programación)". [En línea] Disponible: [https://es.wikipedia.org/wiki/Java_\(lenguaje_de_programaci%C3%B3n\)](https://es.wikipedia.org/wiki/Java_(lenguaje_de_programaci%C3%B3n)), [Último acceso 02 de Setiembre de 2019].
- [93] Wikipedia. "Kanban (desarrollo)". [En línea] Disponible: [https://es.wikipedia.org/wiki/Kanban_\(desarrollo\)](https://es.wikipedia.org/wiki/Kanban_(desarrollo)), [Último acceso 02 de Setiembre de 2019].
- [94] Wikipedia. "Kit de desarrollo de software". [En línea] Disponible: https://es.wikipedia.org/wiki/Kit_de_desarrollo_de_software, [Último acceso 02 de Setiembre de 2019].
- [95] Wikipedia. "Mapeo objeto relacional". [En línea] Disponible: https://es.wikipedia.org/wiki/Mapeo_objeto-relacional, [Último acceso 02 de Setiembre de 2019].
- [96] Wikipedia. "Micromecenazgo". [En línea] Disponible: <https://es.wikipedia.org/wiki/Micromecenazgo>, [Último acceso 02 de Setiembre de 2019].
- [97] Wikipedia. "Modelo de Vistas de Arquitectura 4+1". [En línea] Disponible: https://es.wikipedia.org/wiki/Modelo_de_Vistas_de_Arquitectura_4%2B1, [Último acceso 02 de Setiembre de 2019].
- [98] Wikipedia. "Módulos de seguridad de hardware (HSM)". [En línea] Disponible: <https://safenet.gemalto.es/data-encryption/hardware-security-modules-hsms/>, [Último acceso 02 de Setiembre de 2019].
- [99] Wikipedia. "Nodejs - Aspectos técnicos - Concurrencia". [En línea] Disponible: <https://es.wikipedia.org/wiki/Node.js>, [Último acceso 02 de Setiembre de 2019].

- [100] Wikipedia. "Peer to peer". [En línea] Disponible: <https://es.wikipedia.org/wiki/Peer-to-peer>, [Último acceso 02 de Setiembre de 2019].
- [101] Wikipedia. "Protocolo seguro de transferencia de hipertexto". [En línea] Disponible: https://es.wikipedia.org/wiki/Protocolo_seguro_de_transferencia_de_hipertexto, [Último acceso 02 de Setiembre de 2019].
- [102] Wikipedia. "Python". [En línea] Disponible: <https://es.wikipedia.org/wiki/Python>, [Último acceso 02 de Setiembre de 2019].
- [103] Wikipedia. "React". [En línea] Disponible: <https://es.wikipedia.org/wiki/React>, [Último acceso 02 de Setiembre de 2019].
- [104] Wikipedia. "Smart Contract". [En línea] Disponible: https://es.wikipedia.org/wiki/Contrato_inteligente, [Último acceso 02 de Setiembre de 2019].
- [105] Wikipedia. "SOLID". [En línea] Disponible: <https://es.wikipedia.org/wiki/SOLID>, [Último acceso 02 de Setiembre de 2019].
- [106] Wikipedia. "SOLID". [En línea] Disponible: <https://es.wikipedia.org/wiki/SOLID>, [Último acceso 02 de Setiembre de 2019].
- [107] Wikipedia. "Startup". [En línea] Disponible: https://es.wikipedia.org/wiki/Empresa_emergente, [Último acceso 02 de Setiembre de 2019].
- [108] Wikipedia. "Tolerancia a faltas bizantinas". [En línea] Disponible: https://es.wikipedia.org/wiki/Tolerancia_a_faltas_bizantinas, [Último acceso 02 de Setiembre de 2019].
- [109] Sales & Marketing Manager de Addalia Wladimiro Navarro. "Historia del Blockchain, La solución a un problema". [En línea] Disponible: <https://www.addalia.com/historia-del-blockchain-la-solucion-problema/>, [Último acceso 02 de Setiembre de 2019].
- [110] Yurong Zhu. "Ethereum: qué es, cómo nació y cuáles son sus ventajas". [En línea] Disponible: <https://www.rankia.com/blog/blockchain-criptomonedas-bitcoin-ethereum/3683082-ethereum-que-como-nacio-cuales-son-sus-ventajas>, [Último acceso 02 de Setiembre de 2019].
- [111] Kanban Zone. "Kanban Metrics to Track Performance (Throughput) and Responsiveness (Cycle Time)". [En línea] Disponible: <https://kanbanzone.com/2017/throughput-cycle-time-metrics-kanban-performance-responsiveness/>, [Último acceso 02 de Setiembre de 2019].

,

13 Anexos

13.1 Ingeniería de requerimientos

13.1.1 Evidencias de relevamiento

A continuación se evidencia una minuta ¹ realizada por el equipo entre la primera y segunda reunión presencial con el cliente.

Reunión Nro.: 1

Tipo: Externa

Objetivo: Relevamiento de requerimientos.

Fecha: 3 de Setiembre de 2018.

Hora: 18:30.

Lugar: Universidad *ORT* Uruguay.

Duración: 3 hs.

Participantes: Agustin Caétano, Camilo Martínez, Ismael Puricelli, Juan Rodríguez, Tomás Piaggio, Joaquín Varela.

Temas de agenda:

1. Introducción al cliente y su empresa
2. Relevar las principales funcionalidades del sistema
3. Resolver incógnitas varias con respecto a dichas funcionalidades

¹Se le denomina minuta a las notas en borrador para dar cuenta de lo que se habló en un discurso, conferencia o reunión.

Detalle:

1. Conocer al cliente y su empresa
2. Desglosar las funcionalidades esperadas de *Coopify*
3. Monetización y ganancia de *Coopify*
4. Dudas varias referentes a los tipos de intercambios de servicio
5. Dudas de la negociación entre las partes al contratar un servicio
6. Maneras de obtener *coopies* en la plataforma
7. Principales atributos de calidad a satisfacerse

Temas tratados: Todos los puntos previstos en la agenda.

Resoluciones:

- La negociación se hace previamente a través de un *chat* entre los contactos (se accede contactarlo desde el perfil). se inicia el *chat*, estoy interesado en tus servicios y se propone el valor del mismo. A través de una negociación abierta entre ambos usuarios.
- Concepto de *Request* para ofrecer al otro usuario realizar un determinado servicio (por más de que este no lo realice)
- Concepto de Hora/Sesión/Producto Final al realizar un servicio.
- Se puede contratar varias veces el servicio, no se ha considerado “estirar” el servicio.
- Cuanto menos pasos hacer mejor en el proceso de *checkout*.
- *Coopify* se encarga de conectar ambos usuarios, si otro usuario no cumple al brindar su servicio se impacta en el *rating* y comentarios sobre dicho servicio.
- Un servicio debe tener un valor en *coopies*, si vale 5 *coopies* una hora, dos horas vale 10 *coopies*.
- Puede aceptar *coopies*, trueque o estar libre a cualquier servicio.
- El servicio podría tener distintos precios según la cantidad de horas. (descuento)

- Cada usuario tiene listado de preferencias, etc, Surge también el concepto de méritos y logros.
- *Review*: 5 estrellas y comentario, algo estándar.
- Pago grupal esta bueno para servicios caros, poder dividir el costo entre distintos usuarios.
- Medios para conseguir *coopies*: (el usuario arrancar con determinada cantidad de *coopies* al registrarse)
 1. Proponiendo servicio y que te paguen en *coopies*
 2. Con toda acción que enriquezca a la plataforma, referir amigos a unirse a la plataforma, compartir servicios en redes sociales, acciones que fomenten que la plataforma crezca.
 3. (Más a futuro) Publicidad, si se consume cierta publicidad (*ads*) se generan cierta cantidad de *coopies*.
- Ver como se puede solucionar la devaluación de la moneda. La idea es que los *coopies* pasen de usuario en usuario.
- Login con *Facebook*, *Google*, *Linkedin* aparte del común.
- Atributos de calidad:
 1. Usabilidad: los usuarios deben usarlo y decir “que bueno que es esto”
 2. Performance: No tan crítica pero esta bueno considerarla
 3. Seguridad: respecto a *Blockchain* y además autenticación, autorización.
 4. Modificabilidad (mantenible y extensible)

NOTAS:

Importante: Transmitir confianza y transparencia a los usuarios, los *coopies* siempre están en manos de los usuarios, al ser descentralizado está respaldado por *smart contracts*, la transacción está respaldada por los propios usuarios.

13.1.2 Contrapropuesta realizada

El objetivo de este proyecto es crear una plataforma para el intercambio de servicios entre los usuarios sin tener que manejar dinero real. Para ello se implementará

una moneda propia del sistema llamada *coop*i, es de particular importancia que la moneda sea confiable y transparente para los usuarios por lo que dicha moneda va a estar respaldada por la tecnología *blockchain*, los *coop*ies siempre están en manos de los usuarios, al ser descentralizado está respaldado por *smart contracts* la transacción está respaldada por los propios usuarios.

Para comenzar a publicar sus servicios es necesario que el usuario defina su perfil dentro del sistema, para ello deberá de proveer información básica de sobre su persona, los servicios que le interesan y los servicios que potencialmente puede llegar a proveer.

Los usuarios podrán crearse una cuenta en la aplicación o tendrán la posibilidad de iniciar sesión en la misma mediante redes sociales: *Facebook*, *Linkedin*, *Google*.

En base al perfil creado anteriormente, un usuario podrá realizar una *request* a otro usuario sobre un servicio que éste no provea, pero se encuentre alineado con los que sí provee.

Una vez que se encuentre definido el perfil del usuario podrá comenzar a intercambiar servicios. Para ello deberá de realizar una publicación indicando los siguientes detalles:

- Qué servicio brinda
- Que tipo de hito están disponibles para el servicios prestado (Hora, Sesión y/o Producto final)
 - Hora: Servicios que se cobre por hora. Por ejemplo, cursos particulares con horario predefinido.
 - Sesión: Servicios que se cobre por sesión y no por hora. Por ejemplo, servicio de reparación a domicilio.
 - Producto Final: Servicios que se cobre el producto final realizado. Por ejemplo, un producto de software.
- Cuantos *coop*ies se espera obtener por cada uno de los hitos propuestos (3 por Hora, 5 por Sesión o 30 por Producto Final)
- Como opcional, el descuento, si aplica, sobre el servicio brindado. (una hora por 10 *coop*ies, 2 horas por 15 *coop*ies)

Una vez publicado el servicio dentro de *Coopify* es posible que dos usuarios comiencen el proceso de negociación para llegar a un acuerdo. Para esto el sistema proveerá de una herramienta integrada donde la negociación entre ambas partes

tomará forma (a modo visual podría ser una ventana de *chat* en el perfil del usuario proveedor del servicio). Aquí los usuarios podrán resolver si se llega a un intercambio equivalente de servicios o si se recurre a los *coopies* para abonar por el servicio prestado. Cuando los términos del intercambio de servicio se encuentren definidos, los usuarios procederán con el mismo. Una vez finalizado el intercambio de servicios, el mismo se dará como finalizado y el solicitante podrá elaborar una reseña sobre su experiencia, de este manera los servicios y sus proveedores se podrían ir calificando, mediante un sistema de comentarios y clasificación por estrellas, para evitar que los usuarios finales sufran malas experiencias.

El intercambio de servicios no se limita únicamente a un usuario y un proveedor, en efecto, varios usuarios podrán contratar un mismo servicio, es decir, un determinado servicio ofrecido por un cierto usuario podría ser contratado y pagado en conjunto por varios otros usuarios, permitiendo dividir el costo entre varios usuarios.

Es importante señalar que *Coopify* se encarga únicamente de conectar a los usuarios y no se hace responsable de que un usuario no brinde el servicio luego del mismo ser contratado, esto se verá impactado en el *rating* de los mismos.

Un usuario cuenta con la posibilidad de gastar *coopies* para adquirir servicios pero para ello debe de poder adquirirlos de alguna manera. En esta instancia los medios por los cuales un usuario puede adquirir *coopies* son los siguientes:

- Al registrarse se le adjudicará un monto predefinido
- Realizando intercambios de servicios con otros usuarios del sistema
- Sistema de metas y logros
- Con toda acción que enriquezca a la plataforma (darle visibilidad en las redes sociales, referir a amigos que luego se unan a la plataforma, por nombrar algunas)

Es importante destacar, que en esta etapa **no** se desea realizar ninguna integración con un medio de pago, interesa que los intercambios de servicios se realicen con los *coopies* o con intercambio equivalente entre servicios sin involucrar dinero real.

Hasta este punto se detalló la funcionalidad de la aplicación.

Con respecto al modo de desarrollarla, en conjunto con la empresa destacamos los siguientes puntos:

En cuanto a atributos de calidad, los más importantes a tener en cuenta serán la usabilidad y la modificabilidad.

Para la usabilidad, la aplicación debe resultar lo más intuitiva, rápida y sencilla posible para los usuarios finales, con la posibilidad de lograr sus cometidos de manera rápida y sencilla.

De la mano de la usabilidad, si bien la performance no es algo tan crítico para la aplicación, estaría bueno considerarla dado que se encuentra muy ligada a la usabilidad.

Con respecto a la modificabilidad, al tratarse de un *MVP* es importante lograr una solución que sea sencilla de mantener y modificar en el futuro.

13.1.3 Comparación herramientas similares

Característica	Simbi	Coopify
Iniciar sesión	<i>Facebook, LinkedIn, Común</i>	<i>Facebook, Google, Común</i>
Comunidad	Internacional	Estados Unidos
UI/UX	Listado en grilla, Minimalista, Sencillo de utilizar	Listado en grilla, Minimalista, Sencillo de utilizar
App	<i>Web, Mobile</i>	<i>Web, Mobile</i>
Perfil	Datos generales, Ubicación, Servicios interesados, Abierto a proposición de servicios	Datos generales, Ubicación, Categorías
Publicación de servicios	Todos los usuarios registrados en el sistema pueden ver el detalle de un determinado servicio	Todos los usuarios registrados en el sistema pueden ver el detalle de un determinado servicio
Búsqueda de servicios	Filtros por categoría y nombre únicamente	Filtros de servicios más granular
Intercambio de servicios	Moneda virtual, Intercambio de servicios, Intercambio <i>in-chat</i>	Moneda virtual, Intercambio de servicios, Intercambio <i>in-chat</i>
Comunicación con proveedor	<i>Chat</i> de contacto embebido	<i>Chat</i> de contacto embebido
Respaldo <i>blockchain</i>	No	Si
Intercambio de servicios grupal	No	Si
Sistema de logros	Categorizado	Categorizado
Reviews	<i>Feedback</i> , Recomendaciones	<i>Feedback</i> , Responder preguntas, Calificación de servicio, Calificación de usuario

Tabla 13.1: Comparación entre *Simbi* y *Coopify*

13.1.4 Prototipos de interfaz gráfica

Pantalla principal

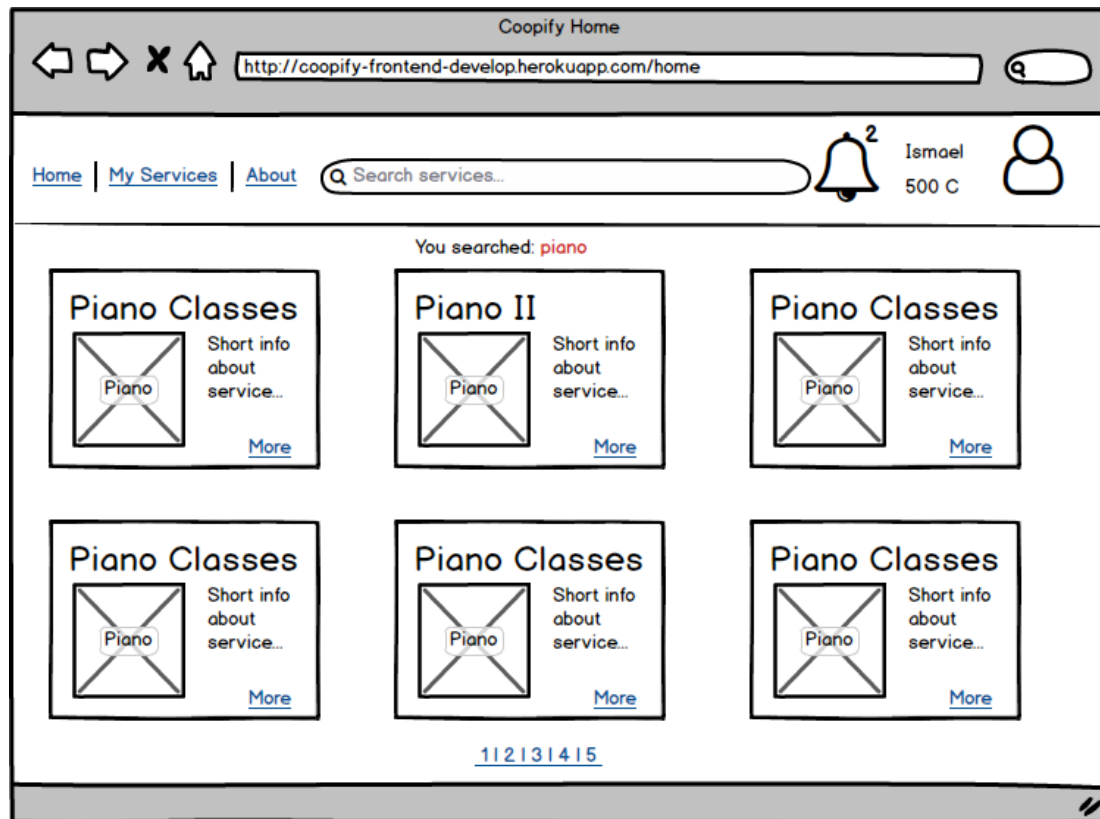


Figura 13.1: Pantalla principal de *Coopify*

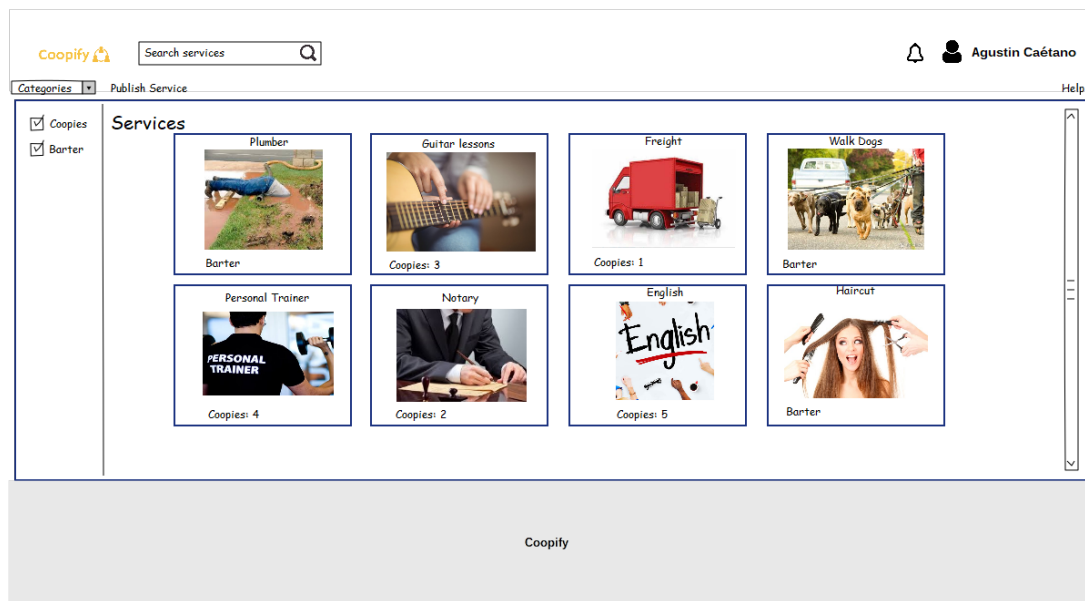


Figura 13.2: Pantalla principal de *Coopify*

Perfil de usuario

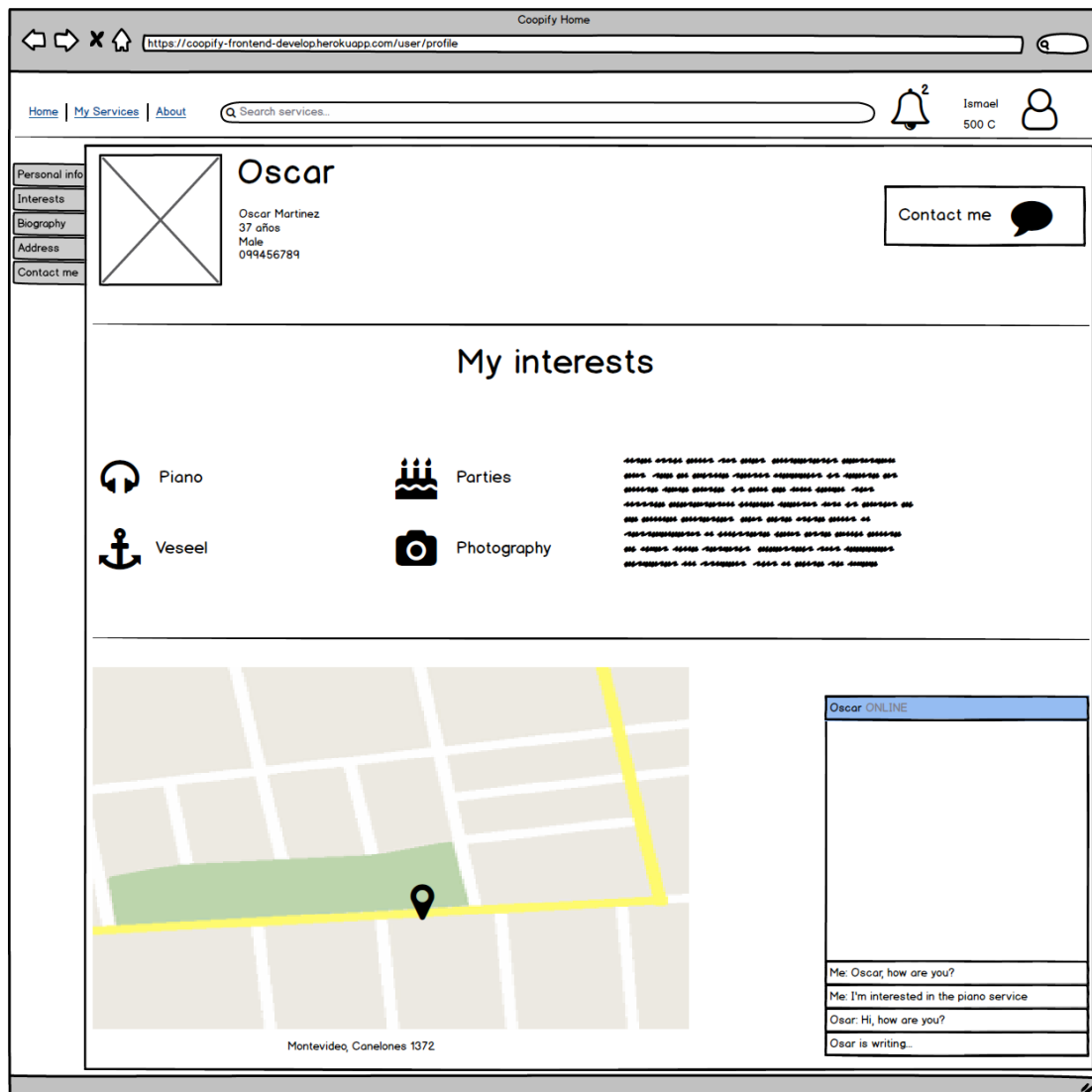


Figura 13.3: Pantalla de perfil de usuario

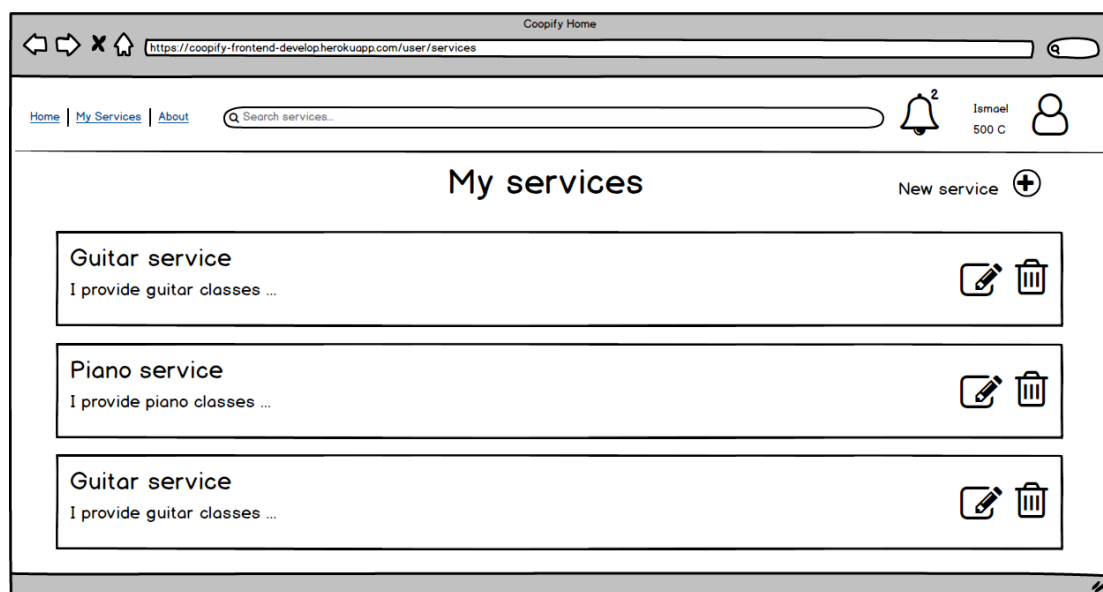


Figura 13.4: Pantalla de perfil de usuario

Publicar un servicio

Figura 13.5: Pantalla de publicación de un servicio

Coopify

Search services

Agustin Caetano

Categories Publish Service Help

Service Description Time, Iterests and Coopies

Publish Service Interest & copies

End Date 1/07/2019

Interest for barter

Dance
Percussion
Languages

Copies 3

Back Publish

Coopify

Figura 13.6: Pantalla de publicación de un servicio

Detalle de un servicio

Coopify

Search services

Agustin Caetano

Categories Publish Service Help

Guitar Lessons

I give guitar lessons, if possible in exchange for coopies, anyway I do not rule out accepting some very tempting barter.

Copies 3 Contact Share

Interests for barter

Dance
Percussion
Languages

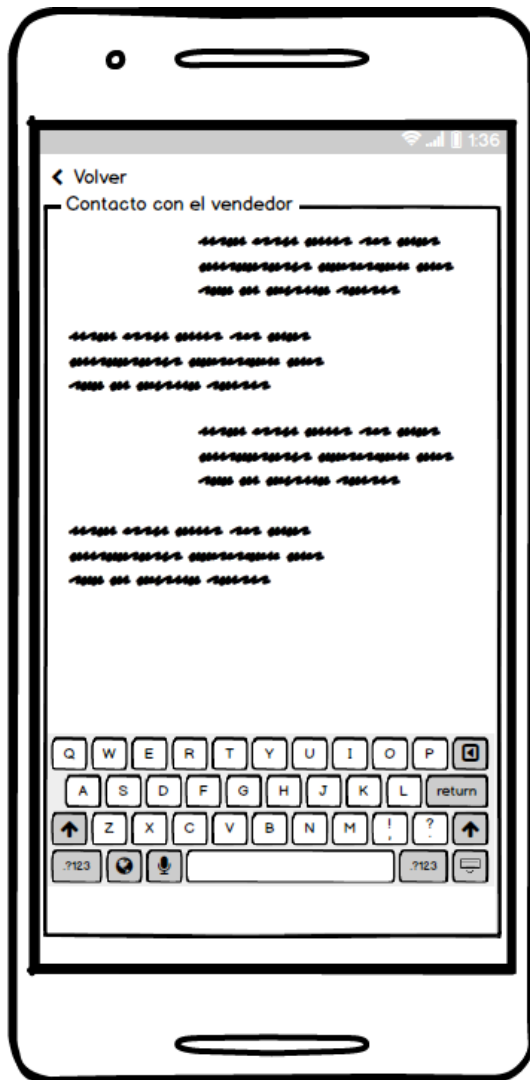
Location

7th Ave 2564

Coopify

Figura 13.7: Pantalla de detalle de un servicio

Negociación e intercambio de un servicio



(a) Pantalla de negociación con un usuario



(b) Pantalla de negociación con un usuario

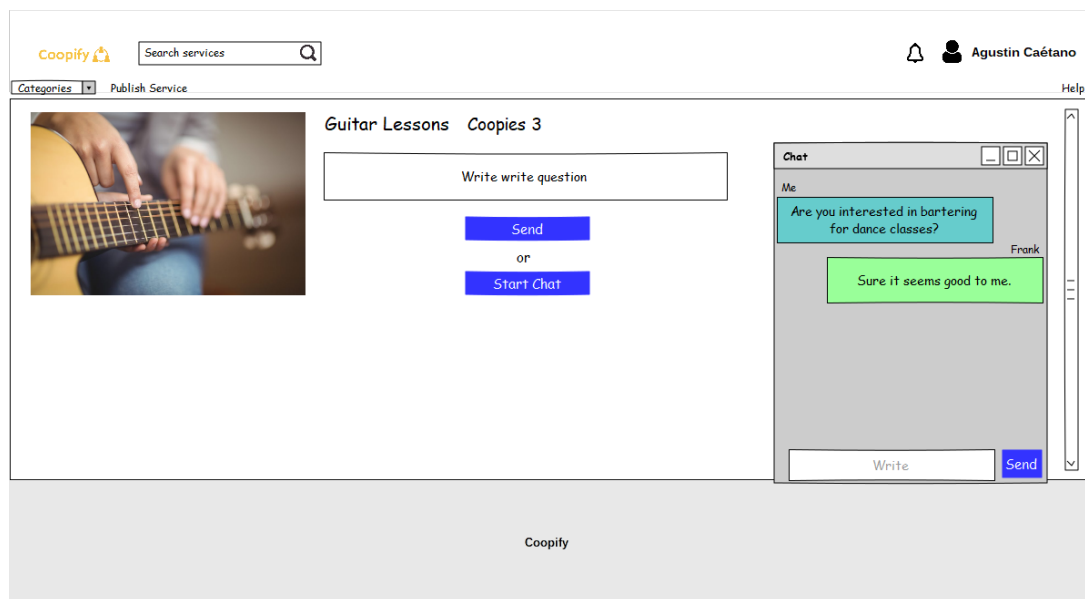


Figura 13.9: Pantalla de negociación con un usuario

Historial de transacciones

The screenshot shows a web browser window titled 'Coopify Home' with the URL `http://coopify-frontend-develop.herokuapp.com/user/profile`. The page has a navigation bar with links for 'Home', 'My Services', and 'About', along with a search bar and a user profile icon for 'Ismael' with a balance of '500 C'. The main content area is titled 'Profile' and contains a 'Purchase history' section. On the left, there is a sidebar with three menu items: 'Basic data', 'Purchase history', and 'Account Settings'. The 'Purchase history' section displays a table with the following data:

Transaction	Date	Destination	Ammount
Registration	2019/09/02	Me	500 C
Piano service	2019/09/10	Oscar	-250 C
Guitar service	2019/09/12	Me	200 C
Painting service	2019/09/13	Fred	-100 C
Guitar service	2019/09/14	Me	200 C
Piano service	2019/09/15	Oscar	-250 C
Guitar service	2019/09/16	Me	200 C

Figura 13.10: Pantalla de historial de transacciones

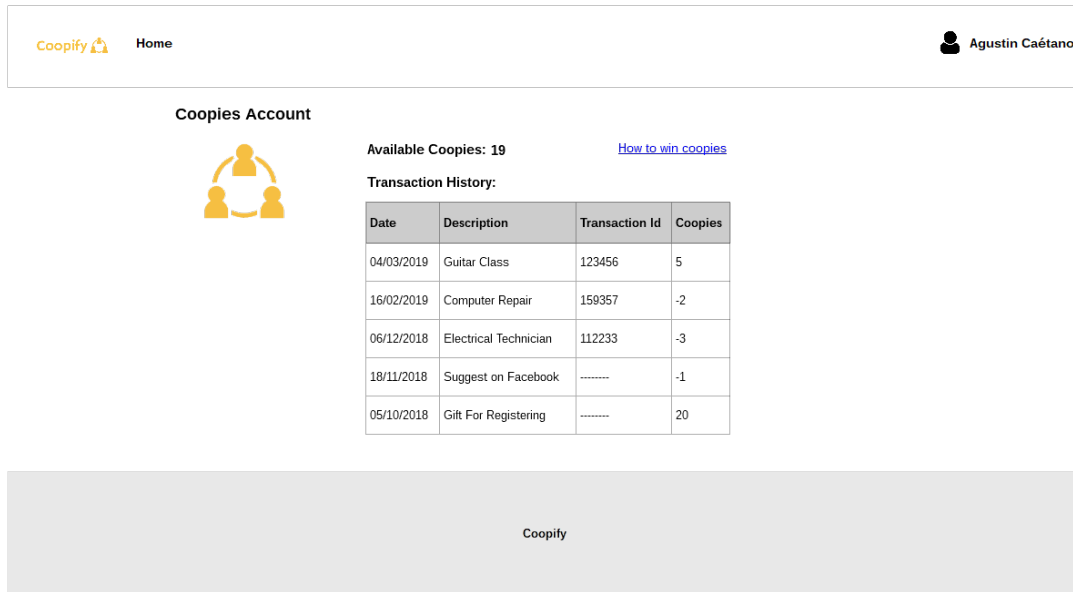


Figura 13.11: Pantalla de historial de transacciones

13.2 Arquitectura

13.2.1 Estructura de los archivos del sistema

En la presente sección detallaremos la estructura de archivos de cada uno de los proyectos que componen el sistema.

Estructura de archivos Backend

En la figura 13.12, se ven las principales carpetas ² que contienen la lógica de negocio necesaria. Como se dijo en capítulos anteriores utilizamos *NodeJS* con *Express* para la implementación de *Backend* y *Ethereum*, a modo de exponer la *API Rest* necesaria.

²Es importante resaltar que una carpeta en el lenguaje *Javascript* representa una *namespace* en el lenguaje *.NET*

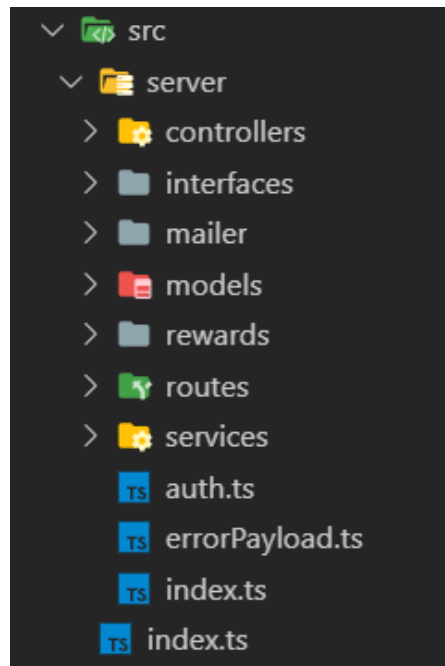


Figura 13.12: Estructura de archivos de *Backend*

La estructura se inspira en las pautas del patrón Modelo Vista Controlador (*MVC*)³.

routes

La carpeta *routes* contiene la definición de las rutas de acceso a cada uno de los recursos internos de la lógica. Es el lugar donde accede cada una de las peticiones externas al *backend*. Es donde se define la *API Rest* a consumir externamente. Para ver los detalles de la definición de la *API*, ver sección 13.3.1.

controllers

Allí se define la “conexion” entre las *routes* y la lógica interna. Nos permite que el día de mañana si se necesita algún tipo de comunicación diferente al de *request/response http*. Seria el lugar a modificar o agregar controladores para hacer esa “conexion” y no tener que modificar nada de la lógica interna.

³MVC: Modelo Vista Controlador, ver referencia bibliográfica [17]

interfaces

Lugar donde se define la lógica de negocio necesaria.

models

Contiene las clases que definen las entidades utilizadas. El *ORM*⁴ utilizado para la conexión con el motor de base de datos es *Sequelize*. Y la base de datos en *PostgreSQL*.

El día de mañana se podría cambiar el motor de base de datos y el lugar a modificar sería justamente en *models* y no en la capa de interfaces. Marcando bien las responsabilidades y no atándonos a la implementación de un solo motor de base de datos.

services

Esta contiene la conexión a servicios como:

- *Blockchain* (conexión entre *Backend* y *Ethereum*)
- *facebook* (para autenticación)
- *google* (para autenticación)
- *pusher*
- *redis*
- *sendgrid*
- *wLogger*

rewards

Aquí separamos en una carpeta lo referente a la implementación de las recompensas a usuarios.

⁴*ORM: Object-Relational mapping* “es una técnica de programación para convertir datos entre el sistema de tipos utilizado en un lenguaje de programación orientado a objetos y la utilización de una base de datos relacional como motor de persistencia”. Ver referencia bibliográfica, [95]

Se trata de un patrón de diseño llamado *Chain of Responsibility*.⁵.

mailer

Archivos encargados del envío de *mails*.

migrations

Por fuera de los que seria la carpeta *src* y a destacar esta la carpeta *migrations*, que documenta o lleva el registro, de los *scripts* necesarios para correr las migraciones en la base de datos en cuanto a cambios en la definición de las tablas.

Estructura de archivos Ethereum

Estructura similar a la de *Backend*.

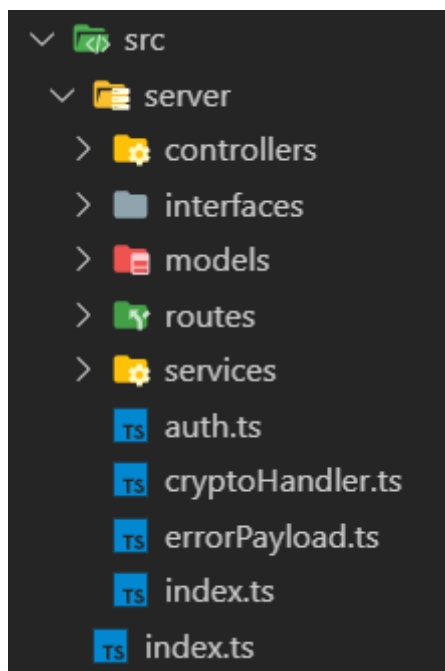


Figura 13.13: Estructura de archivos de *Ethereum*

⁵*Chain fo responsibility*: “patrón de comportamiento que evita acoplar el emisor de una petición a su receptor dando a más de un objeto la posibilidad de responder a una petición”. Ver referencia bibliográfica [83]

La diferencia radica básicamente en el archivo a nivel de la carpeta *src* llamado *cryptoHandler*, encapsulando la lógica encargada de la comunicación con la red *blockchain*.

Estructura de archivos Frontend

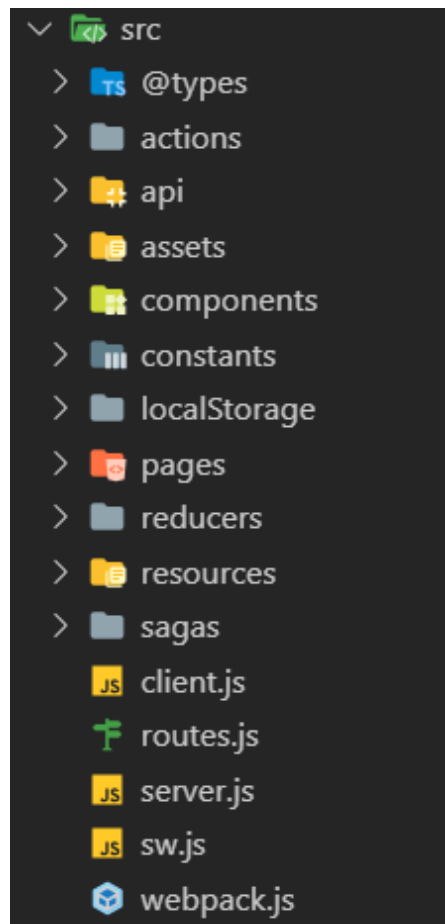


Figura 13.14: Estructura de archivos de *Frontend*

components

Una de las carpetas principales es la de componentes. Ella encapsula los archivos correspondientes a la implementación de cada uno de los componentes que juntos construyen la *UI* que ve el usuario final.

actions

Ella se compone de la definición de las acciones y el tipo de cada una de estas. Las acción al despacharse se comunica con los archivos de las carpetas *reducers* y *sagas*.

sagas

Los archivos que la componen hacen las veces de filtro para saber a que llamada de la carpeta *api* corresponde.

api

Estos disparan las *request https* a la *API Rest* de *Backend* y obteniendo las response que captura y luego comunicando a la carpeta *reducers*.

reducers

Estos se encargan de modificar el *state* que comparten todos los componentes en forma global. La modificación de este *state* hace que se recarguen los componentes conectados al *state* o mejor dicho a las variables que se actualizaron del mismo.

resources

Contiene los archivos como *CSS* para la parte gráfica de del sistema.

assets

Se cargan en ella los archivos de imagen necesarios en el funcionamiento del sistema.

src

Contiene entre las carpetas ya especificadas anteriormente, archivos importantes para el funcionamiento de *pawjs*.

Pudiendo destacar el archivo *sw.js* (*service worker*), encargado de la definición de estrategias de funcionamiento referente a *PWA*, este *sw.js* sobre escribe el *service worker* que contiene por defecto el *boilerplate pawjs*.

De esta forma se plantea la estrategia que se quiere utilizar para el manejo de la desconexión. En nuestro caso, la estrategia se basa en ir primero por lo que hay en internet y si no la hay, obtiene lo guardado en memoria *cache*. Obviamente que esta memoria *cache* se guardo al haber entrado previamente. Por lo que hay varias solicitudes que al momento de querer acceder no habiendo internet pueden mostrar un aviso al usuario de que no se puede acceder a lo solicitado por falta de internet. Esto pasa para lo que dijimos y de forma permanente para algunas de las funcionalidades que no podrían funcionar sin la presencia de internet.

13.2.2 Diagramas de Secuencia

En las siguientes paginas veremos los diagramas de secuencia más relevantes del proyecto.

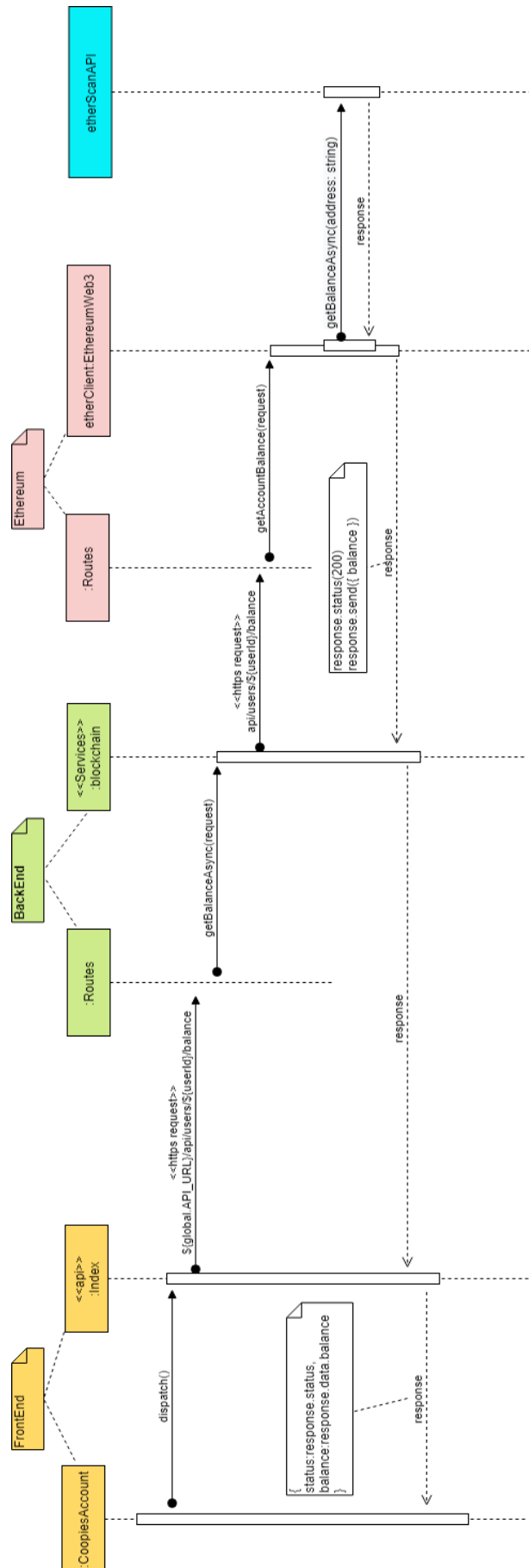


Figura 13.15: Diagrama secuencia obtener balance

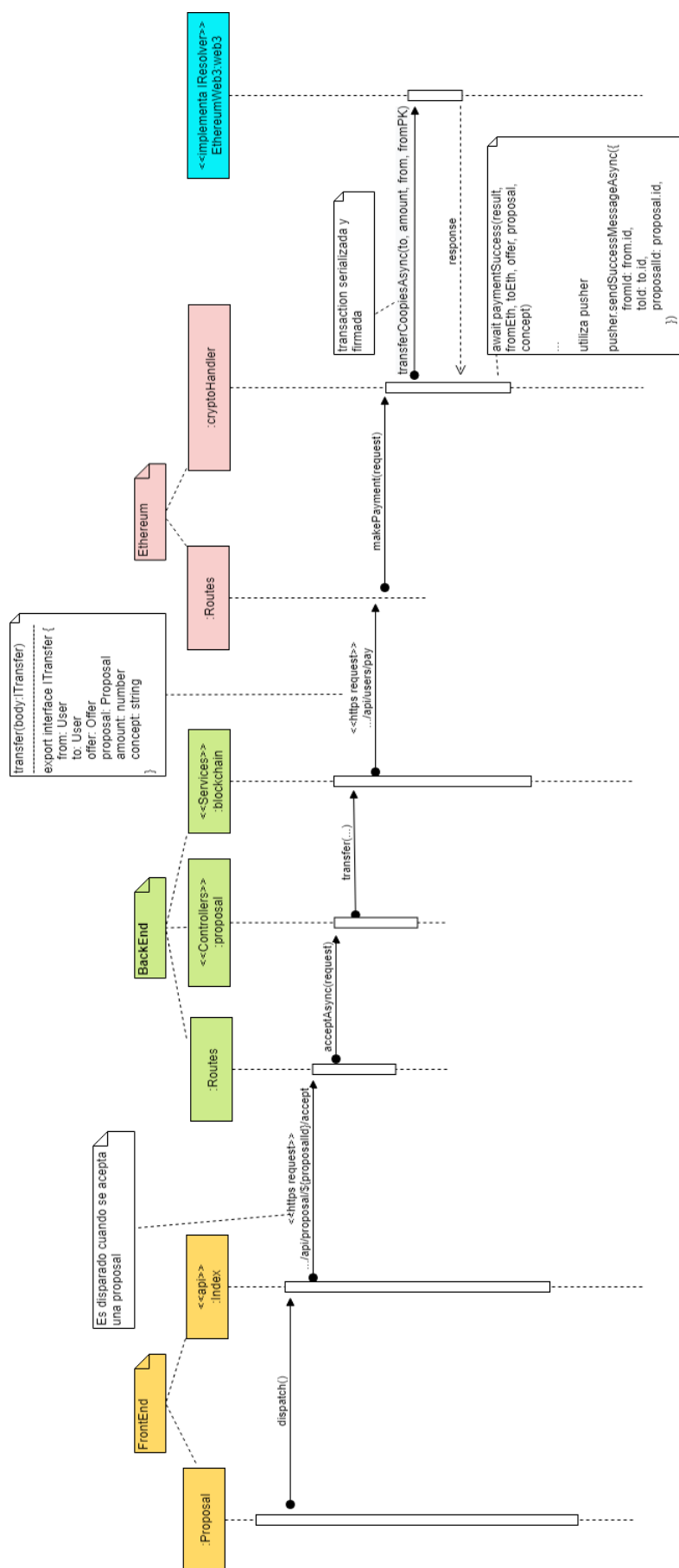


Figura 13.16: Diagrama secuencia aceptar proposición (transacción)

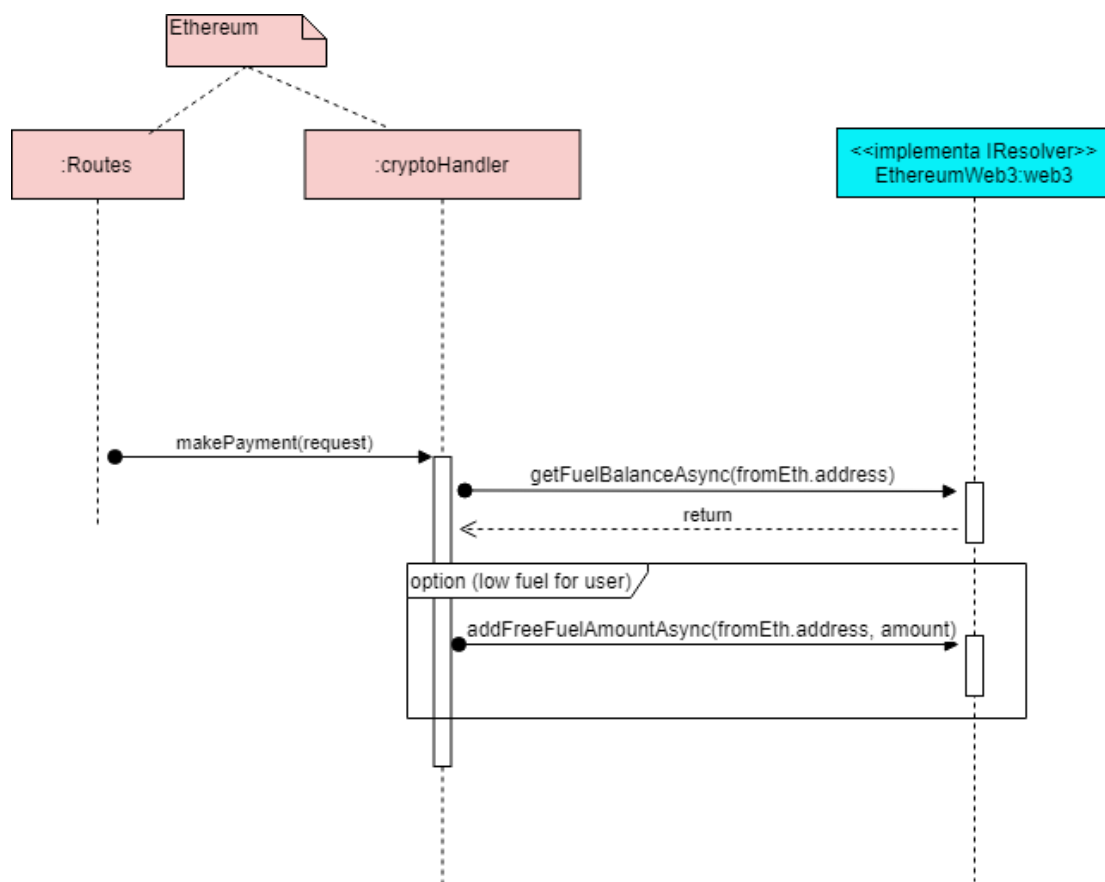


Figura 13.17: Diagrama secuencia agregar Ether a usuario

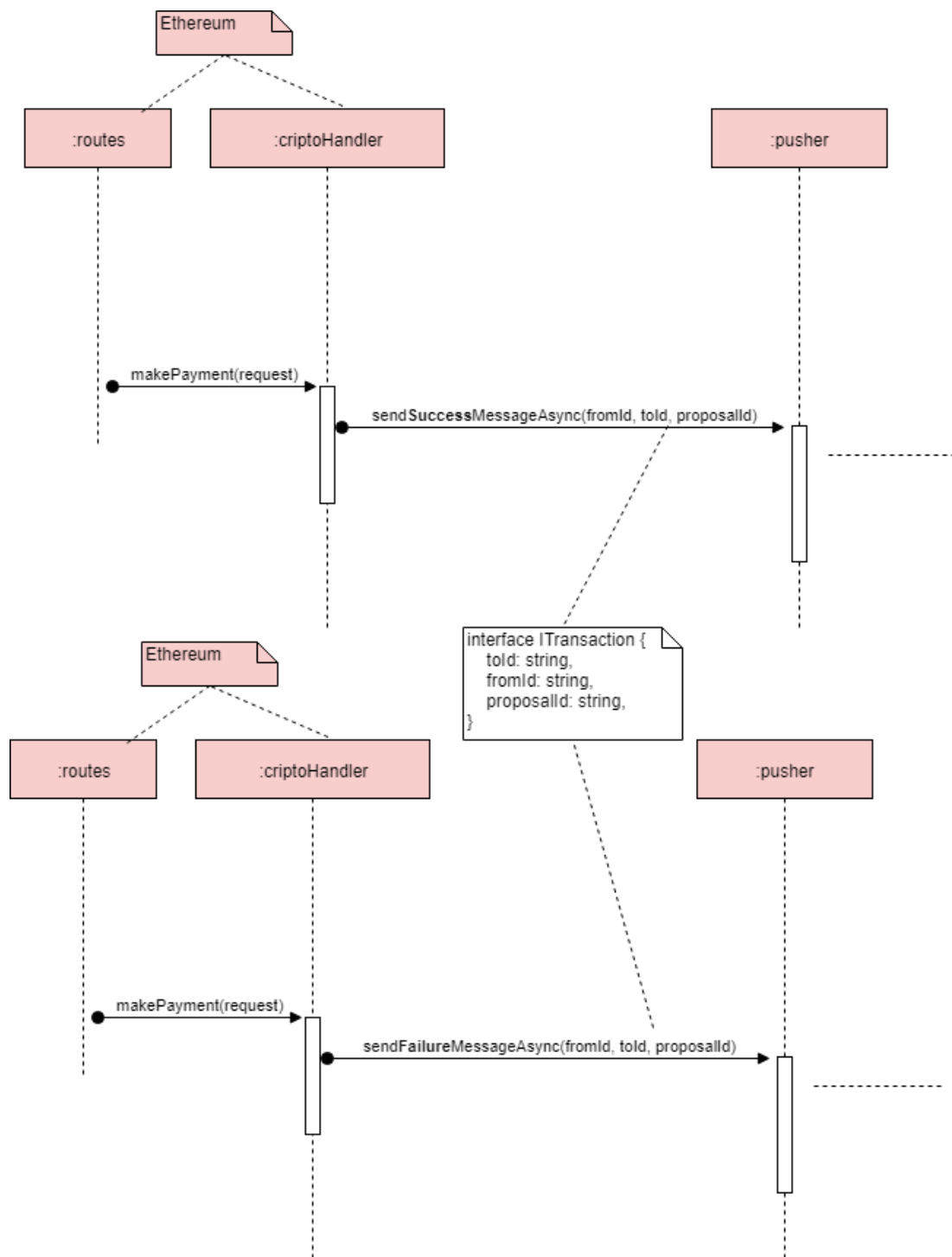


Figura 13.18: Diagrama secuencia envío de mensaje por Pusher

13.3 Documentación API

Dentro de la plataforma, como dijimos antes tenemos tres servidores. Siendo dos de ellos los encargados de la lógica de negocio. Parte de la comunicacion entre ellos es mediante *Request* y *Response https*, gracias a las *API Rest* que exponen tanto el servidor de *Backend*, como el de *Ethereum*. En las siguientes secciones documentaremos estas *API Rest*.

13.3.1 API Rest en servidor de Backend

13.3.1.1 Users

Verbo	Routes	Descripción
GET	/users/facebookURLs	Url inicial para interactuar con Facebook mediante OAuth
GET	/users/googleURL	Url inicial para interactuar con Google mediante OAuth
GET	/users/{userId}/transactions	Obtener las transacciones del usuario por su id
GET	/users/{userId}/balance	Obtener el balance de <i>Coopies</i> del usuario por su id
GET	/users/{userId}	Obtener el detalle de un usuario
PUT	/users/{userId}	Actualizar usuario (Según el body)
POST	/users/signup	Registrar usuario a modo tradicional
POST	/users/facebook/signup	Registrar usuario mediante Facebook
POST	/users/facebook/login	Iniciar sesion mediante Facebook
POST	/users/facebook/sync	Sincronizar usuario registrado de manera tradicional con su cuenta en Facebook
POST	/users/google/signup	Registrar usuario mediante Google
POST	/users/google/login	Iniciar sesion mediante Google
POST	/users/login	Hacer Login tradicional
POST	/users/{userId}/logout	Cerrar sesion
POST	/users/{userId}/share	Disparar acción porque el usuario compartió un servicio en red social

Tabla 13.2: API Rest Backend referentes a *Users*

13.3.1.2 Categories

Verbo	Routes	Descripción
GET	/categories/	Obtener lista de categorías
GET	/categories/{categoryId}	Obtener categoría por id
POST	/categories/	Crear categoría (según body)

Tabla 13.3: API Rest Backend referentes a *Categories*

13.3.1.3 Conversation

Verbo	Routes	Descripción
GET	/conversation/	Obtener lista de conversaciones
GET	/conversation/{conversationId}	Obtener conversación por id
POST	/conversation/	Crear conversación (según body)

Tabla 13.4: API Rest Backend referentes a *Conversation*

13.3.1.4 Goals

Verbo	Routes	Descripción
GET	/goals/	Obtener lista de metas
GET	/goals/{goalId}	Obtener meta por id
GET	/goals/user/{userId}	Obtener las metas del usuario por id

Tabla 13.5: API Rest Backend referentes a *Goals*

13.3.1.5 Message

Verbo	Routes	Descripción
GET	/message/{conversationId}	Obtener lista de mensajes de una conversación
POST	/message/{conversationId}	Crear un mensaje en una conversación

Tabla 13.6: API Rest Backend referentes a *Message*

13.3.1.6 Offers

Verbo	Routes	Descripción
GET	/offers/	Obtener lista de servicios
GET	/offers/count	Obtener cantidad de interacciones
GET	/offers/{offerId}	Obtener servicio por id
GET	/offers/user/{userId}	Obtener servicios de un usuario por id
GET	/offers/questions/{offerId}	Obtener lista de preguntas de un servicio
POST	/offers/	Crear un servicio (según body)

Tabla 13.7: API Rest Backend referentes a *Offers*

13.3.1.7 Proposals

Verbo	Routes	Descripción
GET	/proposals/	Obtener lista de proposiciones
GET	/proposals/{conversationId}	Obtener proposición de una conversación
GET	/proposals/{proposalId}	Obtener proposición por id
POST	/proposals/{conversationId}	Crear proposición en una conversación
PUT	/proposals/{proposalId}/accept	Aceptar una proposición (Proceso que dispara una transacción si el método de pago es por <i>coopies</i>)
PUT	/proposals/{proposalId}/reject	Rechazar una proposición
PUT	/proposals/{proposalId}/cancel	Cancelar una proposición

Tabla 13.8: API Rest Backend referentes a *Proposals*

13.3.1.8 Questions

Verbo	Routes	Descripción
GET	/questions/	Obtener lista de preguntas
GET	/questions/{questionId}	Obtener pregunta por id
PUT	/questions/{questionId}	Modificar pregunta
POST	/questions/{offerId}	Crear pregunta para el servicio

Tabla 13.9: API Rest Backend referentes a *Questions*

13.3.1.9 Rate

Verbo	Routes	Descripción
GET	/rates/	Obtener lista de reseñas
GET	/rates/{rateId}	Obtener reseña por id
GET	/rates/{offerId}/reviweable	Saber si se puede dejar reseña en el servicio
POST	/rates/{offerId}	Crear reseña en el servicio

Tabla 13.10: API Rest Backend referentes a *Rate*

13.3.2 API Rest en servidor de Ethereum

Verbo	Routes	Descripción
GET	/users/{userId}/transactions	Obtener las transacciones del usuario por su id
GET	/users/{userId}/balance	Obtener el balance de <i>Coopies</i> del usuario por su id
POST	/users/pay	Efectuar transacción (según los datos del body)
POST	/users/reward	Dar recompensa (según los datos de body)
POST	/users/signup	Registrar usuario en la red <i>blockchain</i>

Tabla 13.11: API Rest en servidor de Backend

13.4 Gestión de proyecto

13.4.1 Comparación metodología tradicional y ágil

La metodología tradicional se caracteriza por ser compatible con proyectos donde los requerimientos se encuentran previamente definidos y no varían durante el transcurso del mismo. Se ajusta a proyectos ordenados y predecibles.

En la actualidad, sin embargo, los proyectos con estas características van en disminución. Cada vez son más los clientes que solicitan cambios sobre los requerimientos durante el desarrollo del proyecto, esto trae como consecuencia que las empresas tengan que adaptar su proceso de trabajo y métodos de entrega.

En este escenarios, las metodologías ágiles son más exitosas ya que favorecen lo imprescindible y lo volátil.



Figura 13.19: Metodología tradicional vs ágil

Como se puede observar en la imagen, una de las principales diferencias radica en que las metodologías ágiles se basan en ciclos de iteración durante las etapas de diseño, desarrollo y pruebas, facilitando de esta manera la adaptación a los cambios en lugar de una estructura predefinida de las metodologías tradicionales.

A continuación se presentan las principales diferencias entre ambas metodologías.

Tradicional	Ágil
Proceso controlado con numerosas políticas y normas	Procesos menos controlado y con menos principios
Se hace énfasis en la definición del proyecto	Se hace énfasis en los recursos humanos
Grupos grandes y distribuidos	Grupos pequeños y centralizados
Mucha documentación	Poca documentación
El cliente interactúa con el equipo en instancias puntuales prefijadas	Cliente es parte del equipo de desarrollo y esta informado durante el transcurso del mismo
Mayor resistencia a cambios durante el transcurso del proyecto	Se esperan cambios durante el transcurso del proyecto
Introducción de cambios	Se agregan al tablero en el momento que surgen
Roles prefijados no intercambiables	Hay pocos roles y son mas genéricos y flexibles
La arquitectura del sistema se encuentra definida al comienzo del desarrollo y es mas resistente a cambios	La arquitectura del sistema se redefine y mejora conforme se va iterando sobre el desarrollo
Varios artefactos, el modelado es esencial	Pocos artefactos, el modelado es prescindible

Tabla 13.12: Comparación entre metodologías tradicionales y ágiles

13.4.2 Comparación Kanban y Scrum

A continuación se detalla una comparación de las características principales entre las dos metodologías ágiles *Kanban* y *Scrum*, en las que el equipo se basó en parte para la elección de la metodología de investigación.

Característica	Kanban	Scrum
Artefactos	Tablero	Tablero, <i>backlog</i> , <i>burndowns</i>
Ceremonias	<i>Daily, sprint review</i> (opcionales)	<i>Daily, sprint planning, sprint review, sprint retrospective</i>
Iteraciones	No	Si
Estimación	No	Si
Roles	Equipo, roles que el equipo considere necesarios	<i>Product owner, Scrum master</i> , equipo
Introducción de cambios	Se agregan al tablero en el momento que surgen	Se introducen en el próximo <i>sprint</i>

Tabla 13.13: Comparación entre *Kanban* y *Scrum*

13.4.3 Kanbanize

Se adjunta la evidencia del uso de *Kanbanize* con las siguientes capturas.

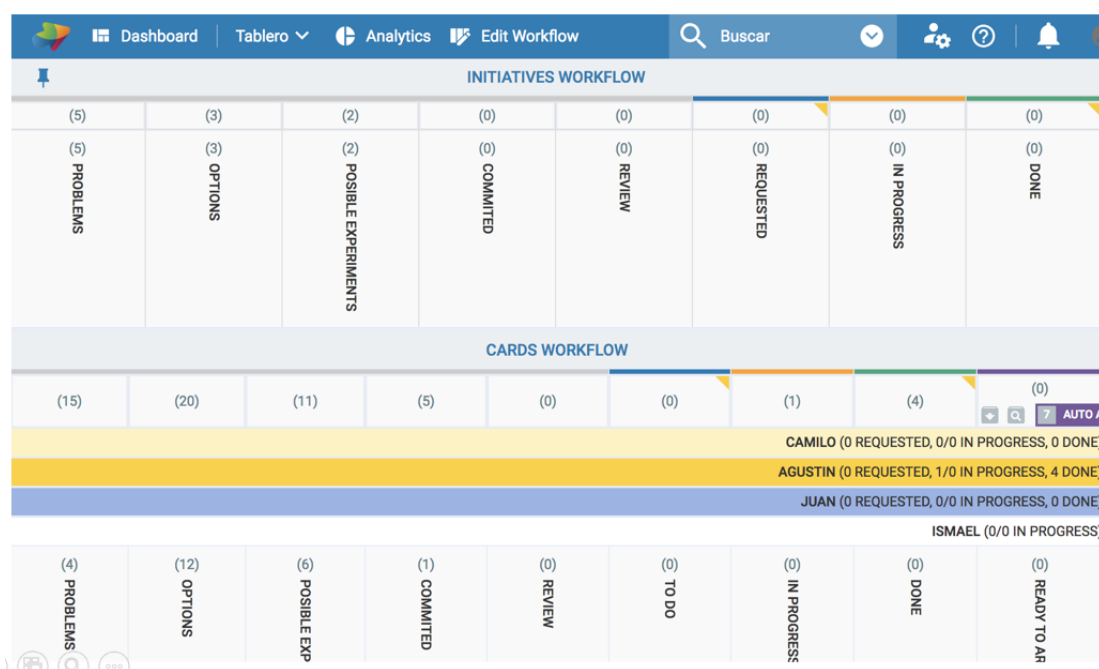


Figura 13.20: *Kanbanize* con columnas de la metodología *Popcorn flow*

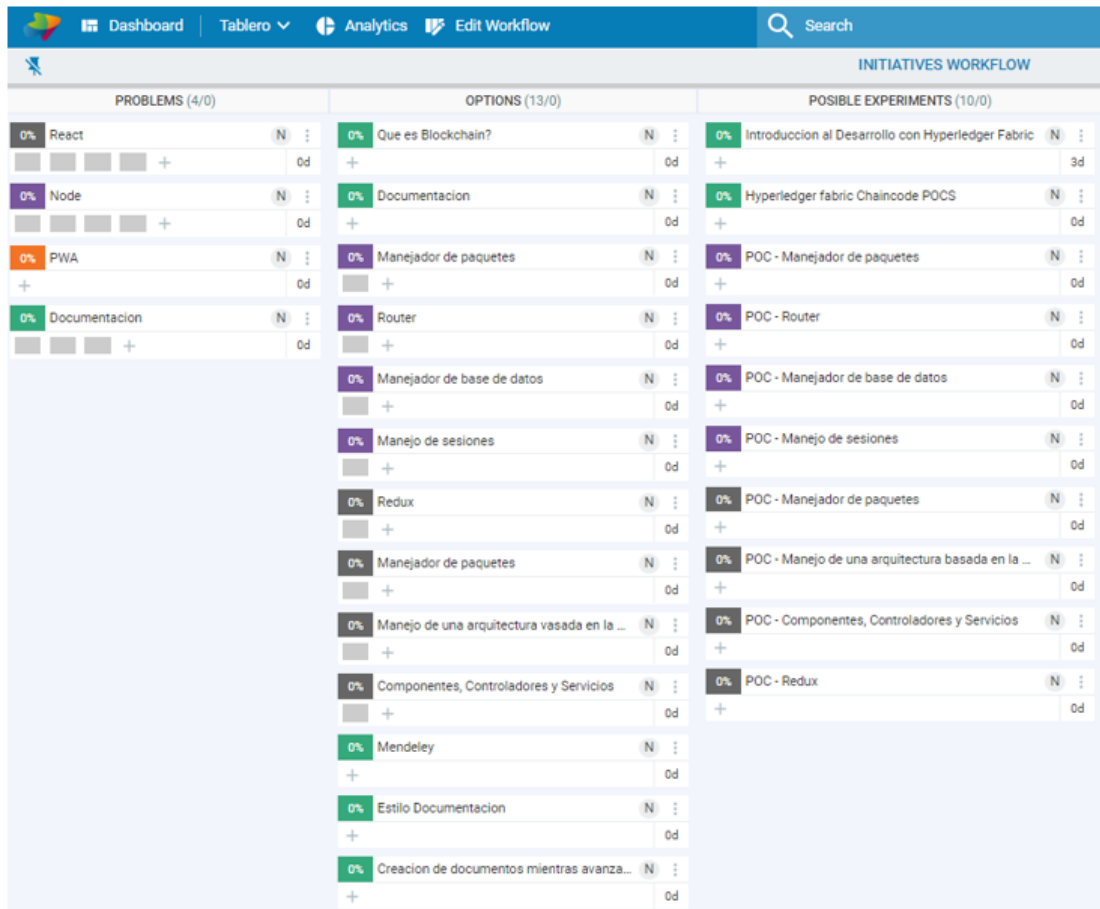


Figura 13.21: *Kanbanize* en uso con la metodología *Popcorn flow*

13.4.4 Evidencia de uso de Teamwork

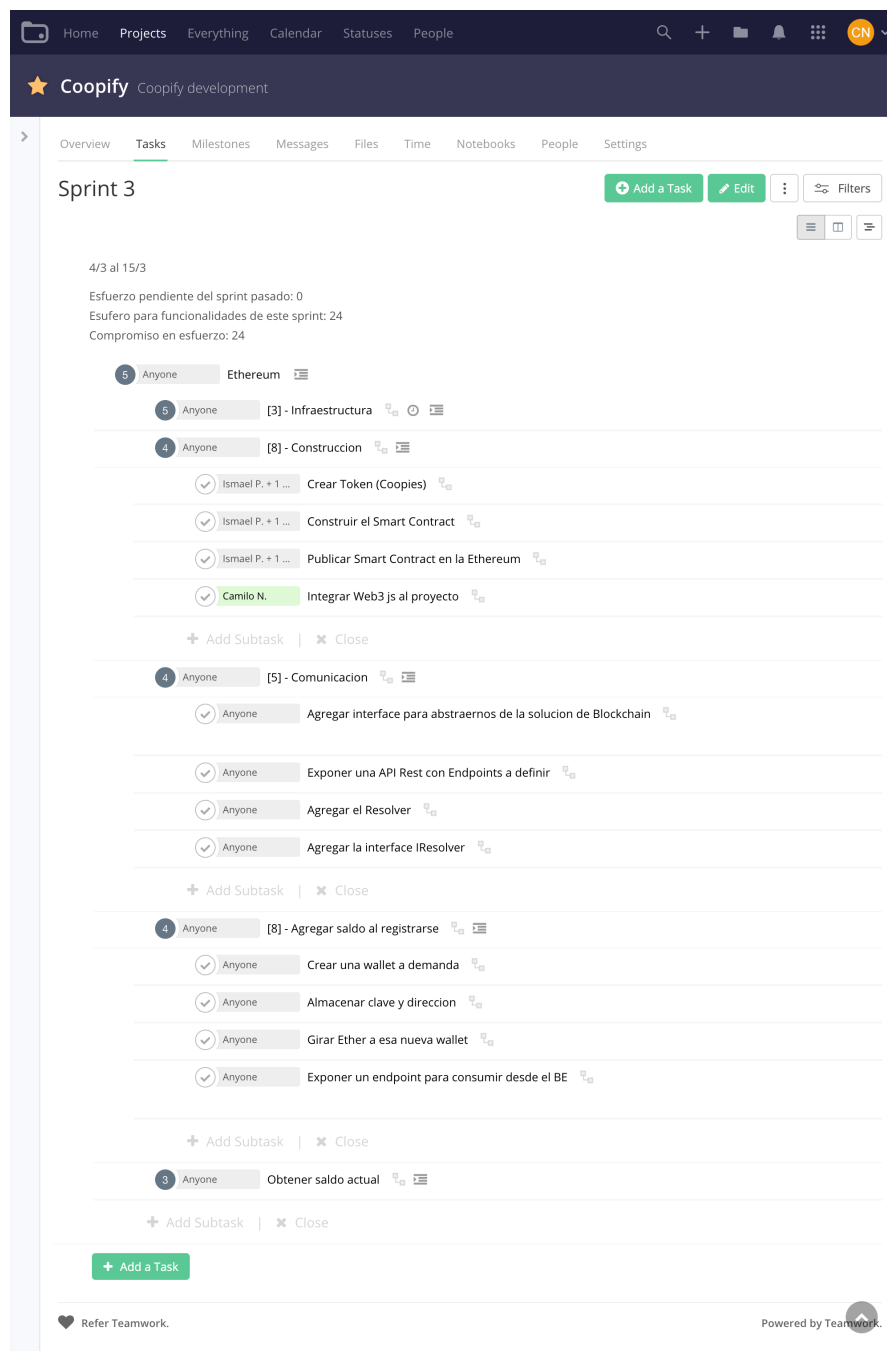


Figura 13.22: Herramienta de *Teamwork*

13.4.5 Valores de impacto y probabilidad de riesgos

		Sprints												
		1	2	3	4	5	6	7	8	9	10	11	12	13
Codigo	C01	60	60	60	60	60	60	60	60	60	60	60	60	60
	C02	0	80	60	60	60	60	60	60	60	60	60	60	60
	C03	0	40	80	80	80	80	80	80	80	80	80	80	80
	C04	0	0	0	0	80	80	80	80	80	80	80	80	80
Devops	D01	60	40	40	40	40	40	40	40	40	40	40	40	40
	D02	0	60	60	60	60	60	60	60	60	60	60	60	60
	D03	0	0	0	0	0	60	60	60	60	60	60	60	60
Ethereum	E01	80	80	60	60	60	60	60	60	60	60	60	60	60
	E02	80	80	40	40	40	40	40	40	40	40	40	40	40
	E03	80	80	60	60	60	60	60	60	60	60	60	60	60
	E04	80	80	60	60	60	60	60	60	60	60	60	60	60
	E05	0	0	0	80	80	80	80	80	80	80	80	80	80
Gestion	G01	100	80	80	80	80	60	60	60	60	60	60	60	60
	G02	60	60	60	80	80	80	80	80	80	80	80	80	80
	G03	80	60	40	40	40	40	40	40	40	40	40	40	40
	G04	80	60	60	60	60	60	60	60	60	60	60	60	60
PWA	P01	40	40	40	40	40	40	40	40	40	40	40	40	40
	P02	80	80	80	80	80	80	80	80	80	80	80	80	80
	P03	60	60	60	60	60	60	60	60	60	60	100	100	100
Equipo	T01	60	60	80	80	60	60	60	60	60	60	60	60	60
	T02	40	40	40	40	60	60	60	60	60	60	60	60	60
	T03	60	60	60	80	80	80	80	80	80	80	80	80	80
	T04	80	80	60	60	60	60	60	60	60	60	60	60	60
	T05	40	40	40	40	40	40	40	40	40	40	40	40	40

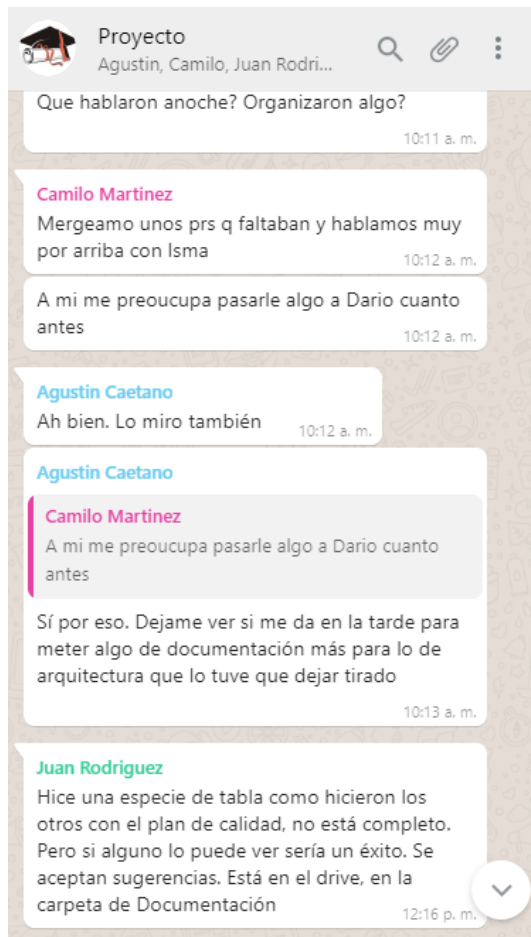
Figura 13.23: Estimación de impactos de riesgos

		Sprints												
		1	2	3	4	5	6	7	8	9	10	11	12	13
Codigo	C01	80%	60%	60%	60%	60%	60%	60%	40%	40%	40%	40%	40%	40%
	C02		80%	80%	80%	80%	80%	80%	60%	60%	100%	60%	40%	40%
	C03		100%	80%	80%	80%	80%	40%	20%	20%	20%	20%	20%	20%
	C04					100%	85%	60%	20%	20%	20%	20%	80%	80%
Devops	D01	60%	40%	40%	40%	40%	40%	40%	20%	20%	20%	20%	20%	20%
	D02		60%	60%	60%	60%	60%	60%	40%	40%	40%	20%	20%	20%
	D03						100%	60%	40%	40%	40%	20%	20%	20%
Ethereum	E01	40%	40%	40%	40%	40%	40%	20%	20%	20%	20%	20%	20%	20%
	E02	80%	80%	60%	60%	60%	60%	40%	40%	40%	20%	20%	20%	20%
	E03	100%	100%	80%	60%	60%	60%	40%	40%	40%	20%	20%	20%	20%
	E04	60%	60%	60%	60%	60%	60%	40%	40%	40%	20%	20%	20%	20%
	E05				75%	60%	80%	60%	40%	40%	20%	20%	20%	20%
Gestion	G01	80%	90%	60%	60%	60%	60%	40%	40%	40%	40%	40%	40%	40%
	G02	100%	100%	80%	85%	70%	60%	40%	40%	40%	40%	40%	40%	40%
	G03	20%	20%	20%	20%	20%	20%	20%	20%	20%	20%	20%	20%	20%
	G04	60%	60%	60%	60%	60%	60%	60%	80%	60%	60%	50%	50%	50%
PWA	P01	80%	80%	80%	80%	80%	80%	80%	80%	80%	80%	80%	80%	80%
	P02	80%	80%	80%	80%	80%	80%	80%	80%	80%	100%	100%	60%	60%
	P03	60%	60%	60%	60%	60%	60%	60%	60%	60%	60%	100%	60%	60%
Equipo	T01	100%	100%	80%	80%	60%	60%	80%	80%	60%	60%	100%	40%	40%
	T02	60%	60%	60%	60%	80%	80%	40%	40%	40%	40%	40%	40%	40%
	T03	60%	60%	80%	100%	80%	60%	40%	80%	40%	40%	40%	40%	40%
	T04	60%	60%	60%	60%	60%	60%	60%	60%	60%	60%	80%	40%	40%
	T05	80%	80%	40%	40%	40%	40%	40%	60%	60%	60%	80%	40%	40%

Figura 13.24: Estimación de probabilidades de riesgos

13.4.6 Evidencia de la gestión de comunicación

WhatsApp



(a) Grupo comunicación interna



(b) Grupo comunicación con el tutor

Slack

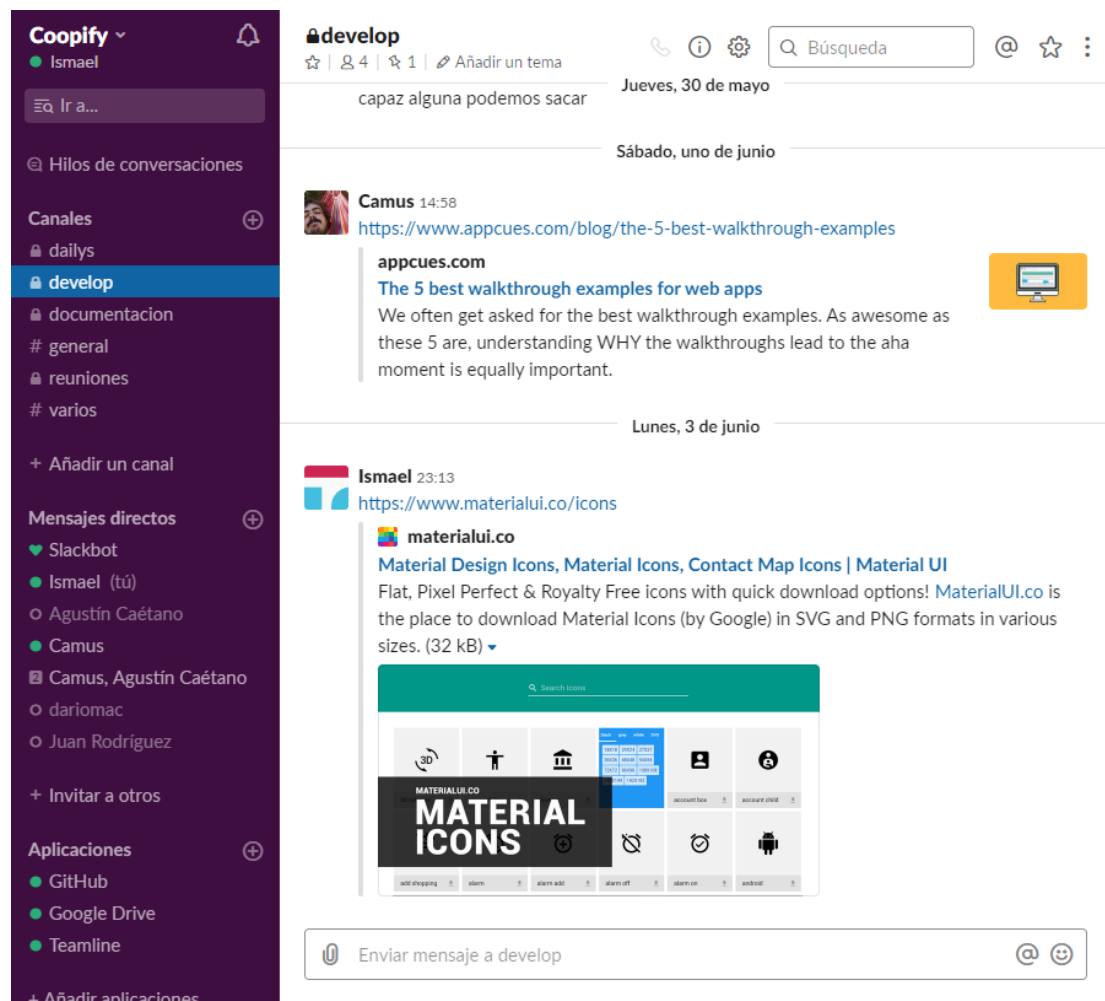


Figura 13.26: Slack

13.5.1 Ejemplo de pull request

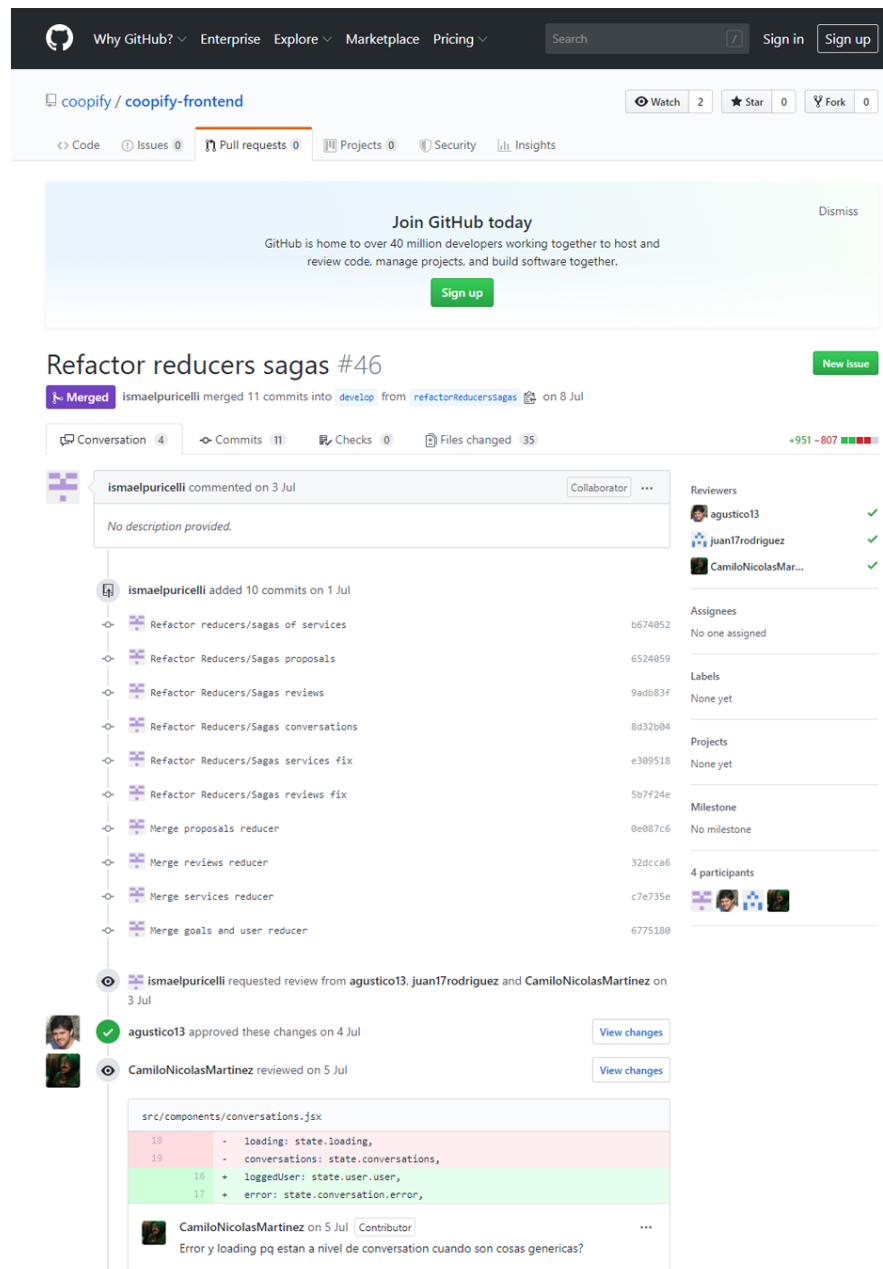


Figura 13.27: *pull request de frontend*





CamiloNicolasMartinez reviewed on 5 Jul

View changes

src/components/coopiesAccount.jsx

Outdated

18

- transactions: state.transactions,

19

- error: state.error,

20

- loading: state.loading,

16

+ loggedUser: state.user.user,



CamiloNicolasMartinez on 5 Jul

Contributor

...

Pq tenemos un State.user.user ??



ismaelpuricelli on 6 Jul

Author

Collaborator

...

Eso es porque cuando dividimos los reducers le agrega el nombre del reducer como objeto, es decir, el primer user es el nombre de UserReducer y el segundo es el objeto user que tiene al usuario logueado.





CamiloNicolasMartinez requested changes on 5 Jul

View changes

CamiloNicolasMa... left a comment

Contributor

...

Si bien estamos mejorando, tengo la sensación que podemos mejorar un poco más con respecto a las transformaciones que hacemos sobre el state. El usuario al loguearse debería de sobre escribir todo el objeto user, no solo el q está dentro del user.user. Debemos de mejorar eso cambiando el nombre del primer user o manejando bien de una eso.





juan17rodriguez approved these changes on 7 Jul

View changes



User property of reducer name changed

131fb55




ismaelpuricelli requested review from CamiloNicolasMartinez, juan17rodriguez and agustico13 on 7 Jul





juan17rodriguez approved these changes on 8 Jul

View changes





agustico13 approved these changes on 8 Jul

View changes





CamiloNicolasMartinez approved these changes on 8 Jul

View changes




ismaelpuricelli merged commit 6ef9287 into develop on 8 Jul

Figura 13.28: *pull request* de frontend

232

13.5.2 Revisiones

Revisión 1

Revisor: Mariel Feder

Fecha: 13 de diciembre de 2018

Fortalezas del grupo

1. El grupo se mostró ordenado en la etapa de investigación.
2. Definición de etapas de manera acertada. Se reconoció y separó el proyecto en 2 etapas de duración correcta con un esfuerzo invertido razonable.
3. Uso correcto de diferentes ciclos de vida ágil para cada una de las etapas del proyecto. Para la primera etapa al utilizar la metodología *Kanban* podemos hacer foco en investigar y no en entregas intermedias al cliente, o mantener ceremonias que no nos aportan nada. Para la etapa de desarrollo utilizamos *SCRUM* justamente para poder entregarle al cliente productos intermedios y en consecuencia detectar desvíos y reaccionar de manera temprana.
4. Aclarar que el objetivo del proyecto es la construcción de un *MVP* para realizar una prueba de concepto.
5. El uso de diferentes metodologías ágiles para la etapa de investigación como para la de desarrollo, si bien ambas utilizan una metodología ágil como base, ambas etapas tienen enfoques y requerimientos distintos.
6. El uso de una herramienta de gestión bibliográfica para poder mantener la bibliografía de referencia ordenada.

Oportunidades de mejora

1. Al ser un *MVP* vamos a necesitar un listado (justificado) de funcionalidades y un público objetivo (justificado).
2. Mejor definición de los RNF. Si bien contamos con un listado de requerimientos no funcionales acotados, no contamos con métricas con las cuales luego revisar que dichos requerimientos fueron cumplidos o no.

3. Plasmar algunas restricciones planteadas por el cliente desde el comienzo como RNF. Por ejemplo el *stack* tecnológico a utilizar sale de una preselección definida por el cliente, eso debería de constituir un RNF en sí mismo.
4. Elaboración de un plan de riesgo. Si bien ya designamos tiempo para el control de los riesgos aún no contamos con un borrador del mismo. Antes de comenzar en desarrollo (o en etapas tempranas del mismo) debemos de elaborar un plan de riesgos.
5. Mejor manejo de herramientas para control del tiempo y actividades. Si bien ya tenemos configurada y estamos utilizando *Toggl* para llevar cuenta de cuánto invertimos en cada tarea no utilizamos dicha información a la hora de realizar la presentación.
6. Construir un plan de *release* que nos permita (a grandes rasgos) definir qué funcionalidades deben de estar desarrolladas para que punto del tiempo.
7. Para futuras exposiciones tomarse un momento para definir que es una criptomoneda, *blockchain*, etc.
8. Plan de calidad, cómo garantizar que vamos a brindar calidad mientras se desarrolla, antes y después.
9. Terminar de definir el *stack* tecnológico a utilizarse, como por ejemplo qué tipo de base de datos vamos a utilizar, gestión de configuración, etc.
10. Para futuras exposiciones acortar el texto en la parte de requerimientos, quedaría mejor simplemente una palabra o texto breve y luego explicarlos con mayor profundidad durante la presentación.
11. Pensando más hacia la presentación final, intentar acortar el tiempo durante la parte de la presentación de los integrantes del equipo y enfocarse más en lo relevante.
12. Definir una posible gestión de riesgos para la etapa de investigación.

Acciones para implementar las oportunidades de mejora

#	Descripción de la acción	Fecha estimada
1	Mejora en el plasmado de los RNF.	Segunda defensa
2	Elaboración de un plan de riesgo.	Finales de enero
3	Mejor manejo del tiempo invertido en las actividades.	Desde ya al final del proyecto
4	Crear un plan de <i>release</i> y compartirlo con el cliente.	Finales de enero / Principios de febrero
5	Elaboración de un plan de calidad.	Mediados de enero / Principios de febrero
6	Terminar de definir el <i>stack</i> tecnológico a utilizarse.	Finales de diciembre

Tabla 13.14: Revisión 1: Acciones para implementar las oportunidades de mejora

Revisión 2

Revisor: Álvaro Ortas

Fecha: 18 de marzo de 2019

Fortalezas del grupo

1. La presentación en términos generales tuvo una muy buena devolución por parte de Álvaro.
2. El grupo se encuentra unido y organizado.
3. Las tecnologías utilizadas si bien no son de punta siguen siendo muy interesantes y le agregan valor al producto al crecimiento profesional de los estudiantes.

Oportunidades de mejora

1. Plantear la solución al problema de manera más intuitiva y con un ejemplo práctico.
2. No queda explícito el valor agregado de la plataforma frente a otras ya existentes.
3. Explicar el concepto de *Blockchain* con un ejemplo práctico y más terrenal de cómo se integra en nuestra solución, no tan teórico.
4. El uso de la metodología *Popcorn Flow* debería tener su propia diapositiva.
5. Las diapositivas referentes a los requerimientos, tanto funcionales como no funcionales, son muy extensas y se asemeja a un listado formal de requerimientos que a una presentación de los mismos.
6. Ordenar los requerimientos con un criterio de prioridad, mostrar los requerimientos más importantes primero.
7. No queda claro cual es el plan de negocio de *Coopify*.
8. No se mostraron prototipos ni una demo funcional de lo que teníamos realizado hasta el momento.
9. Explicar en más detalle la metodología realizada para el desarrollo de la plataforma.
10. Falto hablar del desarrollo de una *Progressive Web App* en lugar de una aplicación *web* y otra *mobile* como fue propuesto inicialmente por el cliente.
11. La gestión de riesgos es conveniente llevarla a nivel de todo el proyecto y no por cada sprint, a modo de no hacerla tan tradicional.
12. Falta la sección de *SQA* desde un punto de vista ágil.
13. Dar trazabilidad a los requerimientos en los diagramas de arquitectura.
14. El diagrama de arquitectura resulta de difícil visualización para el corrector.
15. Con respecto a la gestión del tiempo, no mostrar el tiempo invertido por cada integrante, y mostrar en su lugar el tiempo invertido en cada *feature*.
16. Con respecto a la gestión de configuración, mostrar la estructura general tanto de la etapa de desarrollo como de la de investigación.

17. Al momento de las preguntas, que cada integrante tenga predefinido un tema con el fin de ser más ordenados y que no todos los integrantes intenten contestar a la vez.

Acciones para implementar las oportunidades de mejora

#	Descripción de la acción	Fecha estimada
1	Mejorar y refinar la presentación en múltiples aspectos (presentación de requerimientos funcionales y no funcionales, tecnologías utilizadas, metodologías aplicadas, etc.).	Tercera revisión
2	Indagar y documentar el plan de negocios del cliente para la plataforma.	Tercera revisión
3	Ajustar el alcance de la gestión de riesgos a modo del proyecto y no de cada sprint.	Comienzos de abril
4	Profundizar tareas de <i>SQA</i> para la metodología ágil.	Tercera revisión
5	Mejorar y actualizar la presentación de la arquitectura aumentando la visibilidad y asociándola a los requerimientos no funcionales.	Tercera revisión
6	Llevar un control de tiempo invertido en cada <i>feature</i> .	Mediados de marzo / Comienzos de abril
7	Para futuras defensas practicar y dividir las presentaciones presenciales.	Tercera revisión
8	Elaborar una demo que muestre más lo que se ha elaborado hasta el momento.	Tercera revisión

Tabla 13.15: Revisión 2: Acciones para implementar las oportunidades de mejora

Revisión 3

Revisor: Martín Solari

Fecha: 30 de mayo de 2019

Fortalezas del grupo

1. La presentación mejoró notablemente con el ejemplo de José-Laura durante la introducción.
2. Los requerimientos funcionales estuvieron bien presentados.
3. La sección de arquitectura estuvo muy bien definida y presentada.
4. A pesar de los errores a corregir, la presentación a nivel general estuvo bien, con excepción de la demo.

Oportunidades de mejora

■ Oportunidades relativas a la presentación:

1. Transmitir mejor el problema durante la introducción, comenzar con más fuerza.
2. La presentación se hace muy larga en cuanto a falta de ritmo, no de duración.
3. Falta un hilo conductor durante toda la presentación, está poco organizada, estaría bueno que se siga con un mismo ejemplo como el de José-Laura, incluso para la demo.
4. Expresar mejor la estructura de la investigación y destacar o mencionar la salida o entregable del mismo (en nuestro caso el informe).
5. No queda claro cuales son los RNF a nivel general, decir cuáles son y luego en todo caso cómo llegamos a ellos. Por ej. El RNF es asegurar la mantenibilidad y performance, entonces la tecnología a utilizarse a nivel de servidor es *NodeJS*.
6. Destacar la estabilidad de *PWA* y explicar que no se trata de otra tecnología para el ámbito *mobile*, sino que seguimos utilizando *React*.

7. La demo resultó muy confusa y poco intuitiva para un usuario final. Se intentó abarcar todo y no queda claro cómo se realizan las principales funcionalidades.

8. En la demo no se vio nada desde el punto de vista tecnológico.

■ **Oportunidades relativas a la gestión del proyecto:**

1. Al ser un *MVP* deberíamos centrarnos en funcionalidades específicas pero bien explotadas, y no tanto en presentar varias funcionalidades pero poco explotadas.

2. No conformarse con lo que diga el cliente, ir más allá y ofrecerle alternativas para que sea mas critico.

3. Falta determinar y definir el perfil y grupo de usuarios que utilizará la aplicación.

4. Queda poco clara la interacción y validación de requerimientos con el cliente.

5. Intentar realizar más reuniones presenciales con el cliente y no tantas remotas.

■ **Oportunidades para el cumplimiento de atributos de calidad:**

1. Faltan hacer pruebas de validación con usuarios.

2. No queda claro cómo probamos y validamos las funcionalidades tanto a nivel interno del equipo como con el cliente.

3. La aplicación resulta muy poco usable para un usuario final.

4. Falta agregar cobertura de las pruebas automáticas realizadas.

Acciones para implementar las oportunidades de mejora

Oportunidades relativas a la presentación:

#	Descripción de la acción	Fecha estimada
1	Re estructurar la sección de investigación de la presentación para transmitir claramente que se hizo y el resultado obtenido.	Próxima presentación
2	Mantener el ejemplo con el que se introduce el problema a lo largo de toda la presentación.	Próxima presentación
3	Re estructurar la sección de RNF partiendo de las necesidades del cliente, como las encaramos y resolvimos.	Próxima presentación
4	Re hacer la demo con los tiempos que la misma requiera. Mantener el ejemplo en la demo también. Aprovechar la interacción entre actores para mostrar el ámbito web y <i>mobile</i> de una <i>PWA</i> (lucir el producto).	Próxima presentación
5	Practicar más la presentación para que sea más sencillo mantener la atención del público.	Próxima presentación

Tabla 13.16: Revisión 3: Oportunidades relativas a la presentación

Oportunidades relativas a la gestión del proyecto:

#	Descripción de la acción	Fecha estimada
1	Hacer ajustes finales y liberar la encuesta de caracterización de actores al público, luego de unos días evaluar los resultados obtenidos.	Cuanto antes
2	Conversar con el <i>Product Owner</i> para reevaluar las historias pendientes teniendo en cuenta que el proyecto se trata de un <i>MVP</i> , estimar nuevamente cuales son las historias imprescindibles, deseables y prescindibles.	Cuanto antes
3	Acordar reuniones presenciales más frecuentes.	Cuanto antes
4	Formalizar el proceso por el cual el cliente queda conforme (o no) con el resultado de cada sprint.	Cuanto antes

Tabla 13.17: Revisión 3: Oportunidades relativas a la gestión del proyecto

Oportunidades para el cumplimiento de atributos de calidad:

#	Descripción de la acción	Fecha estimada
1	Construir un plan de pruebas de usuario final y llevarlo a cabo.	Cuanto antes
2	Invertiremos gran parte del próximo sprint en mejorar la usabilidad de la aplicación. Para esto cambiaremos la estructura de algunas páginas, introduciremos tutoriales en el primero uso, unificamos los elementos del menú, entre otros.	Cuanto antes
3	Agregaremos alguna herramienta para generar reportes sobre las pruebas ya existentes de <i>backend</i> .	Próximos <i>sprints</i>

Tabla 13.18: Revisión 3: Oportunidades para el cumplimiento de atributos de calidad

13.6 Gestión de la configuración

13.6.1 Comparación GitHub y Bitbucket

A continuación se evidencia la comparación realizada por el equipo entre *GitHub* y *Bitbucket*.

Característica	<i>GitHub</i>	<i>Bitbucket</i>
Sistema de control de versiones	<i>Git</i>	<i>Mercurial, Git</i>
Repositorios privados	Ilimitados a partir de 2019	Ilimitados hasta 5 usuarios por repositorio
Manejo de incidentes	Si	Si
<i>Merge requests</i>	Si	Si
Integración continua	Si, mediante aplicaciones de terceros	Si
Integración con terceros	<i>Heroku, Windows Azure, AWS, Google Cloud, Asana, Zendesk, CloudBees, Travis, CodeClimate</i>	<i>Jira, Crucible, Jenkins, Bamboo</i>

Tabla 13.19: Comparación entre *GitHub* y *Bitbucket*

13.7 Investigación de tecnologías para implementar una Blockchain

13.7.1 Introducción

Blockchain es una tecnología que plantea un método mediante el cual se logra separar información importante en diferentes nodos. Esto tiene como objetivo descentralizar y proteger información importante por ejemplo: transacciones financieras, registros médicos, etc.

Pensemos por ejemplo en las empresas que están detrás de tarjetas de crédito como por ejemplo *Visa/Mastercard*; ellas tienen toda su información centralizada para poder responder rápidamente a la gran cantidad de transacciones que deben de procesar por segundo. Dicha centralización es riesgosa pues los usuarios de ese servicio no tienen certeza si su información financiera fue o está siendo alterada (para beneficiar a la empresa o por un error informático).

Para poder evitar los problemas inherentes a tener la información centralizada *Blockchain* plantea una estructura en donde existen diferentes nodos conectados entre sí que entre todos se comportan como un organismo regulador de las diferentes transacciones generadas a lo largo del tiempo. De esa manera si un nodo se ve afectado (por un ataque malicioso, un desperfecto en alguna tarea automatizada, etc) todavía contamos con una red de nodos que podrá validar todas las transacciones que se realicen a futuro.

Cómo sabemos gracias a *Blockchain* y su magistral forma de crear una red en donde salimos de lo convencional, del banco como servidor central y pasando a utilizar el potencial de las redes *P2P*, pasamos a tener varias opciones en cuanto a plataformas que nos brindan opciones para la creación de esas redes descentralizadas.

La investigación surge gracias a la necesidad de obtener información sobre las tecnologías existentes para implementar una red *blockchain*, y mediante esta crear una moneda virtual, que nos permita manejar las transacciones entre usuarios de la plataforma que desarrollaremos con nuestro modelo de negocio, garantizando de esta manera confiabilidad y seguridad a los usuarios en cada una de estas transacciones.

De aquí en adelante haremos una comparación resaltando las ventajas y desventa-

jas de las tecnologías *Ethereum* e *Hyperledger Fabric*. Llegando a una conclusión sobre cuál de las tecnologías nos parece que más se adapta a lo necesario en nuestro caso.

13.7.2 Estado del arte

Para entender mejor sobre lo que trataremos, tenemos que entender en detalle de qué se trata *blockchain*.

Blockchain es una red de nodos (miles o millones de dispositivos en el mundo), que se convierten en parte de un enorme banco, en el que cada nodo es partícipe y gestor de los libros de cuenta del banco.

Forman un gigantesco libro de cuentas, en los que los registros (los bloques) están enlazados y cifrados para proteger la privacidad y seguridad de las transacciones.

Forman una base de datos distribuida y segura aplicable a todo tipo de transacciones que no tienen porqué ser necesariamente económicas.

El requisito importante es que deben de haber varios usuarios (nodos) que se encarguen de verificar esas transacciones para validarlas y que el bloque correspondiente a esa transacción (existen muchas transacciones por bloque), se registre en ese gigantesco libro de cuentas.

Para que esas transacciones queden registradas de forma definitiva, necesita de que el bloque que la contiene se valide o selle, qué es lo que se hace al hacer minería.

La minería consiste en la realización de una serie de complejos cálculos que requieren tiempo y electricidad, pero que cuando este proceso termina, ese bloque queda registrado de forma permanente en la cadena de bloques y no puede ser modificado sin que se alteren todos los bloques que están enlazados con el.

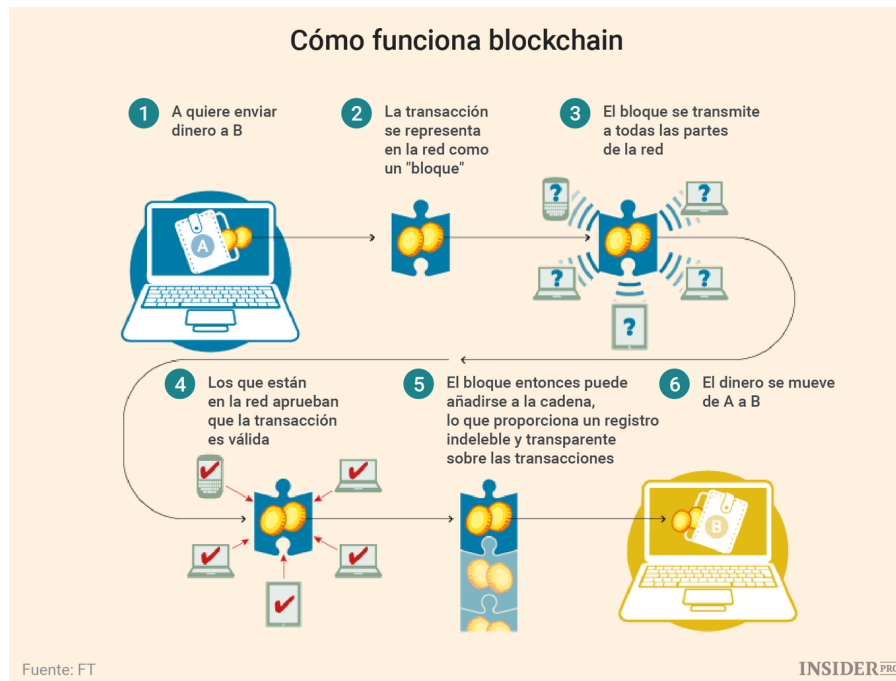


Figura 13.29: Cómo funciona Blockchain [53]

Si yo quiero pagar a alguien con dinero físico, es un proceso sencillo y bastante seguro ya que se la entrego en mano. No hay dudas que la transacción se dio y que el destinatario ya tiene el dinero.

Pero cuando esto se quiere hacer en el mundo electrónico ya es distinto. Necesitamos confiar en alguien que regule las transacciones y gestione la titularidad del dinero electrónico.

Los bancos certifican el dinero que tenemos en nuestras cuentas y ejecutan los movimientos, garantizando que las transacciones se realizan correctamente. Por encima de ellos, los gobiernos regulan y controlan que realizan correctamente su labor. Para llevar adelante estas tareas los bancos cobran intereses sobre esos movimientos. Por esto es que surge el querer quitar del medio los intermediarios (en este caso los bancos).

Esto se pretende con *Bitcoin* y de a poco lo ha logrado.

No es extraño observar que lentamente esta nueva tecnología está reemplazando el sistema bancario existente. La razón a esto, como se dijo anteriormente, es que dicho sistema actual presenta ciertas problemáticas las cuales no están presentes en un sistema *Blockchain*.

Dichos problemas suelen ser numerosos, a continuación listamos los principales.

- Altos costos de transacción
- Problema de “*Double Spending*”

El problema conocido como “*Double Spending*” se trata de un error en donde un mismo token puede ser usado más de una vez en una transacción. Por ejemplo, supongamos que un individuo tiene un balance en su cuenta de \$100 y decide enviar \$90 a otra persona, normalmente no habría problema, pero si decide realizar varias transacciones en simultáneo enviando \$90 a varias personas duplicando el token de la transacción, podría estar enviando más dinero del que posee en su balance.

- Fraudes y *hackeos*
- Crisis financiera

Entonces, ¿cómo es que la tecnología *Blockchain* logra hacer frente a estos problemas? Gracias a su estructura e implementación, *Blockchain* posee ciertas características que permiten atacar los problemas anteriormente mencionados.

- Sistema descentralizado

Blockchain posee un sistema descentralizado a diferencia del sistema bancario actual, donde se encuentra todo centralizado en uno o varios entes, bancos y organizaciones financieras. El hecho de que este sistema sea descentralizado implica que cada integrante del mismo es igualmente responsable del éxito o fracaso del mismo.

- Libros mayores públicos

El libro mayor, donde se guarda un histórico de todas las transacciones llevadas a cabo en la red *Blockchain*, es completamente abierto y accesible a todos los miembros de la misma, permitiendo a cada uno acceder a la lista de transacciones y de esta manera evitar fraudes. Sin embargo, los detalles de los miembros asociados a cada transacción permanecen anónimos.

- Verificación de cada transacción individual

Cada transacción es verificada a través de una comprobación cruzada contra el libro mayor, al cabo de unos minutos se devuelve una señal de validación al usuario. Mediante el uso de varios algoritmos de *hashing* y encriptación, el problema de “*Double Spending*” se elimina.

- Costos de transacción bajos o inexistentes

Generalmente no aplican costos extras a las transacciones, sin embargo algunas plataformas de *Blockchain* lo implementan, aun así dichos costos son relativamente más bajos que los impuestos por bancos u otras organizaciones financieras. Si una transacción tiene mayor prioridad que el resto y debe llevarse a cabo en un determinado periodo de tiempo, podrían aplicarse costos sobre la misma.[52]

En 1991 aparece el primer trabajo de una cadena de bloques segura utilizando criptografía, y que fue evolucionando hasta que en 1998, Wei Dai describe una solución descentralizada para pagos electrónicos basada en criptografía de clave pública.

Este primer trabajo es evolucionado por otros autores, hasta que en 2008 se publica con el pseudónimo de Satoshi Nakamoto, el artículo que define el mecanismo para implementar una moneda digital (*Bitcoin*). Basado en las cadenas de bloques (*blockchain*) para registrar las transacciones en una red peer-to-peer.

Nadie sabe quién es Satoshi Nakamoto, es más, se cree que es el pseudónimo que le pusieron a un grupo que lo puso en marcha.

La red de *Bitcoin* se desarrolla como un proyecto de código abierto, donde nadie es dueño de la tecnología y la evolución de la misma se realiza en consenso de la comunidad.

En 2013 un programador involucrado en el desarrollo de *Bitcoin*, llamado Vitalik Buterin propuso incorporar la posibilidad de gestionar no solo transacciones de dinero (*bitcoins*), sino que aprovechar el modelo *blockchain* para ejecutar aplicaciones distribuidas, para ejecutar contratos de forma descentralizada. Propuesta que no fue aceptada por la comunidad, por lo que este se abre del proyecto *Bitcoin* y pone en marcha el proyecto *Ethereum* para la ejecución de *Smart Contracts*. [109]

13.7.3 Desarrollo

Dentro del mundo de las criptomonedas existen múltiples monedas con una variedad de objetivos.

Existen moneda orientadas a pagos/transacciones económicas: *Bitcoin*, *Ripple* y *Stellar* son buenos ejemplos de este tipo de moneda. Existen otras basadas en *smart contracts*: *Ethereum*, *Hyperledger fabric*, *EOS*, *Cardano* y *Monero*.

Estas últimas son las que nos interesan al momento ya que nos brindan la posibilidad de definir un comportamiento que permite verificar, facilitar y realizar transac-

ciones de forma independiente, trazable e irreversible. De ser necesario podríamos definir y desarrollar uno/s *smart contracts* para llevar a cabo los objetivos de este proyecto.

Dentro de todas las criptomonedas mencionadas nos concentramos en *Ethereum* e *Hyperledger* ya que son dos de las más famosas y de las que el cliente mostró más interés.

13.7.3.1 Introducción Ethereum

En este caso nos centraremos en *Ethereum*, una plataforma que gestiona la criptomoneda (*Ether*), inventada por Vitalik Buterin, joven de origen Ruso, en un proyecto que comenzó a finales de 2014.

Ethereum, llegó como un proyecto mucho más potente y con mayores usos que el de ser un sistema de pago y una moneda digital descentralizada, como en el caso de *Bitcoin*. El gran avance de *Ethereum* respecto a éste son los llamados ‘*smart contracts*’ o contratos inteligentes.

Los contratos inteligentes son programas informáticos que ejecutan acuerdos establecidos entre dos o más partes cuando se da una condición programada con anterioridad. Es decir, son contratos que se ejecutan y se hacen cumplir a sí mismos de manera automática y autónoma.

En palabras de uno de sus fundadores:

“Los smart contracts son una forma para que personas de todo el mundo hagan negocios entre sí, incluso si no hablan el mismo idioma o no utilizan la misma moneda”.^[49]

Nos centraremos en las ventajas y desventajas de *Ethereum*.

13.7.3.2 Ventajas de Ethereum

No están controladas por ningún gobierno central, por lo que no pueden controlar su valor.

Son más difíciles de falsificar que las divisas cómo el euro o el dólar.

No existen intermediarios a la hora de hacer transacciones o pagos, por lo que es más barato.

Sistema de código abierto, lo que permite agregar mejoras.

No existe límite de *Ether*, a diferencia de *Bitcoin* que tendrá un máximo de 21 millones de unidades.

Recompensa fija de 5 *Ether* por la minería al inicio y al momento 3 *Ether* por bloque, mientras que en el caso del *Bitcoin* se reduce a la mitad por cada 210000 bloques minados.

El tiempo entre transacciones de los *Etheres* es mucho más rápido que el de los *Bitcoin*. Llegando a ejecutar operaciones en segundos, en lugar de minutos.

Smart Contracts integrados en la misma plataforma de *Ethereum* sin necesidad de ningún software externo. Estos son aplicaciones que funcionan exactamente como están programadas, sin posibilidad de inactividad, censura, fraude o interferencia de terceros.

La red *Ethereum* está basada en la comunidad y no controlada por ninguna autoridad, lo que hace que estas aplicaciones no necesitan de servidores ni otras entidades, sino que viven en la cadena de bloques, al igual que todo su contenido. [110]

Para que los desarrolladores escriban aplicaciones de calidad, *Ethereum* utiliza el *Ether* para incentivar a los desarrolladores, mediante *GAS*.

Con el fin de evitar bucles infinitos accidentales, hostiles, u otro desperdicio computacional en el código, cada transacción es obligada a establecer un límite al número de pasos computacionales de ejecución de código que ella puede utilizar. La unidad fundamental de computación es “*GAS*”. Por lo general un paso computacional cuesta 1 *GAS*, pero algunas operaciones cuestan cantidades más altas de *GAS* porque son más costosas computacionalmente, o porque aumenta la cantidad de datos que deben ser almacenados como parte del estado. También hay una tarifa de 5 *GAS* por cada byte en los datos de transacción.

La intención del sistema de comisiones es obligar a un atacante a pagar proporcionalmente por cada recurso que consume, incluyendo computación, ancho de banda y almacenamiento. Por lo tanto, cualquier operación que conduzca a la red a consumir una mayor cantidad de cualquiera de estos recursos debe tener una comisión de *GAS* más o menos proporcional al incremento.[44]

Ethereum, con su enfoque en aplicaciones descentralizadas (*DApps*), contratos inteligentes y cadenas de bloques públicas, está más alineado con el mercado de empresa a consumidor (*B2C*). [21]

13.7.3.3 Desventajas de Ethereum

Una desventaja importante, pero que es relativa a *Blockchain* y no tanto a una tecnología específica dentro de la metodología, es la falta de regulación. No hay nadie que se dedique a velar por que todo esto sea legal. Digamos, a quien habría que reclamar si algo sale mal.

La velocidad de las transacciones no es tan buena, no es tan eficiente como otras criptomonedas más rápidas.

La falta de buena documentación y la mayor parte del contenido en línea está desactualizado.[33]

13.7.3.4 Introducción Hyperledger Fabric

Hyperledger nace como un proyecto de código abierto para avanzar en el desarrollo de las tecnologías *Blockchain*. En este proyecto participan numerosas empresas coordinadas y organizadas por: *The Linux Foundation*.

Está formada por líderes del mundo de las finanzas, la banca, el Internet de las cosas, la cadena de suministro, la fabricación y la tecnología. Más de 185 miembros y 8 proyectos en marcha, entre los que se incluyen *Hyperledger Fabric* y *Hyperledger Composer*, trabajan conjuntamente para crear una base de código y una infraestructura de contabilidad distribuida abierta y estandarizada para empresas.

Hyperledger tiene un enfoque modular, en el que podemos encontrar, Infraestructura, *Frameworks* y Herramientas. La descripción detallada en el siguiente esquema:

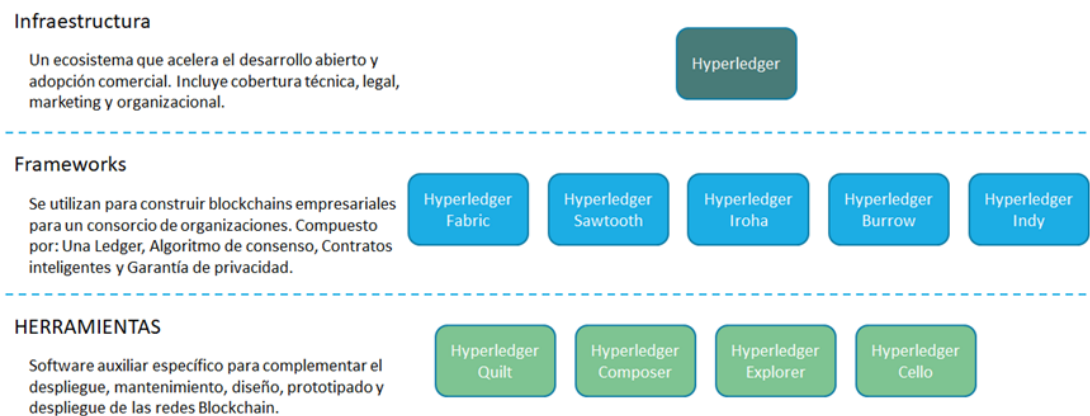


Figura 13.30: Una breve introducción a Hyperledger. [10]

Hyperledger Fabric es una infraestructura de *blockchain* para empresas y cuya finalidad es crear una base para desarrollar aplicaciones o soluciones de *blockchain* con una arquitectura modular. *Hyperledger Fabric* permite que componentes como los servicios de consenso y membresía sean de conectar y listo.

Aporta confidencialidad y escalabilidad a los entornos de negocio, aprovechando las mejores prácticas de código abierto.

13.7.3.5 Ventajas de Hyperledger Fabric

Las empresas que desean desarrollar aplicaciones de *blockchain* con privacidad sólida y soporte de permisos están mirando hacia *Hyperledger Fabric*. Es el más adecuado para desarrollar aplicaciones de *blockchain* listas para la empresa mediante contratos inteligentes con privacidad y soporte de permisos. [21]

A nivel de autorización, no es gratuito, los usuarios deben registrarse y renunciar a su anonimato. Esto es importante para una empresa, para saber con quién se está negociando.

Escalabilidad, personalización y confianza. Al tener una estructura modular, permite una mayor escalabilidad. No todos los operadores de nodos tienen que estar incluidos en una transacción. Solo las personas directamente implicadas en una operación obtienen información.

Las empresas pueden negociar precios diferentes y lo ideal es que no todo el mundo pueda verlo. Para esto permite que las empresas puedan hacer que determinadas operaciones sean visibles para un grupo determinado.

Trabaja con *SDKs* y tiene una estructura modular, lo que permite a las empresas desarrollar una red de acuerdo a sus necesidades. Haciendo que los componentes existentes se puedan combinar fácilmente con otras soluciones de *Blockchain*.

Utiliza módulos de seguridad de hardware (*MSH*), para proteger y gestionar claves e identidades digitales.

Es fácil de entender, por lo que atrae a muchas industrias, sin importar el tamaño y la experiencia en *Blockchain*. [35]

13.7.3.6 Desventajas de Hyperledger Fabric

Hyperledger Fabric es un *framework* para utilizar en cualquier industria. No es adecuado para proyectos de criptomoneda. Se puede desarrollar un token digital nativo, pero será un proyecto de desarrollo complejo que requerirá tiempo y esfuerzo significativos.

Brian Behlendorf director ejecutivo del Proyecto *Hyperledger*, prometió no crear una moneda, evitando así tantos desafíos políticos de tener que mantener una moneda globalmente consistente.

Muchos pueden ver esto como una ventaja, pero en el caso de querer utilizar una criptomoneda, se requerirá de mucho tiempo y esfuerzo para implementarla.

13.7.4 Diferencias entre tecnologías

¿En qué se diferencia *Hyperledger Fabric* de otras implementaciones de *blockchain*?

	<i>Ethereum</i>	<i>Hyperledger Fabric</i>
Criptomoneda necesaria	<i>ether</i> , criptomoneda creada por el usuario	none
Red	Transacciones	Consenso
Transacciones	anónimas o privadas	públicas o confidenciales
Consenso	prueba de trabajo	<i>PBFT</i>
Contratos inteligentes (lógica empresarial)	sí (<i>Solidity</i> , <i>Serpent</i> , <i>LLL</i>)	Sí (<i>chaincode</i>)
Lenguaje	<i>Golang</i> , <i>C++</i> , <i>Python</i>	<i>Golang</i> , <i>Java</i>

Tabla 13.20: diferencias entre *Hyperledger Fabric* de otras implementaciones de *blockchain*

[36][35]

Para comparar las plataformas *Ethereum* y *Hyperledger Fabric* tomaremos en cuenta las siguientes características de cada plataforma, si bien ambas son plataformas de *Blockchain* su enfoque sobre la implementación de *Blockchain* es distinto.

- Propósito

El punto principal de la comparación radica en el propósito de ambas plataformas y con la intención con la cual fueron diseñadas.

Ethereum ejecuta *Smart Contracts* bajo su máquina virtual (*EVM*) para aquellas aplicaciones que son pensadas en ser descentralizadas y ser consumidas por una gran cantidad de usuarios.

Hyperledger Fabric, por otro lado, aprovecha la tecnología *Blockchain* para el ámbito de negocios. Se encuentra diseñado para soportar la incorporación de implementaciones de nuevos componentes (o *plugins*) brindando altos niveles de confidencialidad, adaptabilidad y escalabilidad. De esta manera podemos observar que la plataforma *Hyperledger Fabric* posee una arquitectura modular la cual provee mucha flexibilidad al momento de decidir cómo usarlo.

- Confidencialidad

Ethereum se encuentra diseñado para ser completamente transparente con los usuarios por lo que cada transacción quedará visible para todos los usuarios de la *Blockchain*.

En *Hyperledger Fabric* sin embargo, es posible la existencia de transacciones confidenciales entre determinados usuarios de la *Blockchain*, esto permite a los negocios tener una mayor flexibilidad y una mayor seguridad para realizar transacciones visibles para un determinado grupo de usuarios de la *Blockchain*.

- Modo de participación

Ethereum puede ser una red pública o privada sin ningún tipo de permisos, mientras que *Hyperledger Fabric* es una red privada y con permisos. Esto significa que en *Ethereum* cualquier usuario puede participar en la red en cualquier momento, sin embargo en la plataforma *Hyperledger Fabric* se tiene una comunidad predefinida de participantes de la red, por lo que el acceso a la misma se encuentra restringido a dichos participantes únicamente.

- Mecanismo de consenso

En *Ethereum* todos los participantes de la red (también llamados nodos) deben llegar a un consenso sobre todas las transacciones. Esto es independiente de si un nodo a participado o no de una determinada transacción.

El mecanismo de consenso se encuentra establecido en la minería de bloques basado en el algoritmo “*Proof of Work*”, todos los nodos deben acordar un libro mayor común y de esta manera todos tendrán acceso a los registros del mismo.

Hyperledger Fabric sin embargo, tiene un control más granular sobre el consenso y acceso restringido a las transacciones lo que impacta positivamente en la performance, escalabilidad y privacidad de la aplicación. Dicho mecanismo de consenso es más amplio y abarca todo el flujo de la transacción, desde proponer una nueva transacción a la red hasta registrarla en el libro mayor, donde los nodos asumen diferentes roles y tareas durante dicho proceso.

Esto último contrasta con *Ethereum* donde los roles y tareas de los nodos se mantienen idénticos.

- Lenguaje de programación

Otra diferencia importante es el uso de *Smart Contracts* en *Ethereum* escritos en un lenguaje de alto nivel denominado *Solidity*.

En *Hyperledger Fabric* se utiliza el término “*chaincode*” como sinónimo de *Smart Contract*, un “*chaincode*” se encarga de manejar lógica de negocio acordada por los miembros de la red, por lo que se puede pensar como un *Smart Contract*. Estos últimos se encuentran escritos en *GoLang*, un lenguaje de programación creado por *Google*.

- Criptomoneda

El uso de criptomonedas entre las transacciones es otro punto de comparación importante.

Hyperledger Fabric no requiere una criptomoneda para las transacciones, por lo que tampoco proviene con una criptomoneda incorporada. Esta característica hace que la minería no sea requerida como en otras plataformas y por ende permite el uso de un algoritmo de consenso escalable capaz de manejar las altas tasas de transacciones usualmente requeridas por aplicaciones empresariales.

Ethereum sin embargo, posee una criptomoneda propia llamada *Ether*, la cual se utiliza como recompensa a aquellos nodos que contribuyen a lograr un consenso mediante la minería de bloques, o para el pago de cargos de las transacciones. Esto último puede ser una ventaja frente a *Hyperledger Fabric* en aquellos casos en los que se requiera una criptomoneda para las transacciones, aunque se podría lograr un efecto similar mediante el desarrollo de tokens en el “*chaincode*” actuando como criptomonedas. [43][56]

13.7.5 Conclusión

Los *Smart Contrat* de *Ethereum* nos brindan una solución bastante genérica y adaptable a casi cualquier tipo de aplicación, sin embargo, su transparencia en la *blockchain* y modo de operación sin permisos impacta negativamente en la performance, escalabilidad y privacidad de la aplicación. Es una plataforma dirigida a aplicaciones *B2C*.

Hyperledger Fabric, en cambio, provee una gran nivel de performance y escalabilidad gracias a su modo de operación basado en permisos y control de acceso granular. Su arquitectura permite adaptarse a varios tipos de aplicaciones gracias a sus componentes modulares fácilmente incorporables en distintas soluciones. Es una plataforma que se encuentra dirigida especialmente a aplicaciones *B2B*.

De las dos tecnologías la que nos convence más para utilizar en el proyecto es *Ethereum*.

Nos brinda la descentralización necesaria para los *Smart contracts*. Dándonos la seguridad para que los usuarios efectúen intercambios de la moneda virtual, sin posibilidad de que esos intercambios se modifiquen luego, como podría ser por ejemplo al tener todo en una base de datos única. En donde habría que confiar siempre en la administración se esos datos.

Un punto importante de porqué no utilizar *Hyperledger fabric* es que no es muy recomendable para proyectos en donde se quiera implementar una criptomoneda por la dificultad que significa en cuanto a tiempo y esfuerzo en comparación con *Ethereum*.

Ethereum nos permite crear tokens encima de su *blockchain*, lo que se adapta muy bien al propósito que tenemos en mente.

Al momento de desarrollar una aplicación basada en *blockchain*, los desarrolladores se enfrentan a la decisión de qué plataforma optar, ambas opciones son bastante flexibles pero en diferentes aspectos.

13.7.6 Glosario

Blockchain: *“Una cadena de bloques conocida en inglés como blockchain, es una estructura de datos en la que la información contenida se agrupa en conjuntos (bloques) a los que se les añade meta-información relativa a otro bloque de la cadena anterior en una línea temporal, de manera que gracias a técnicas criptográficas, la información contenida en un bloque solo puede ser repudiada o editada modificando todos los bloques posteriores”.* [82]

Redes P2P: *“Una red peer-to-peer, red de pares, red entre iguales o red entre pares (P2P, por sus siglas en inglés) es una red de ordenadores en la que todos o algunos aspectos funcionan sin clientes ni servidores fijos, sino una serie de nodos que se comportan como iguales entre sí. Es decir, actúan simultáneamente como clientes y servidores respecto a los demás nodos de la red. Las redes P2P permiten el intercambio directo de información, en cualquier formato, entre los ordenadores interconectados”.* [100]

Ethereum: es una plataforma open source, descentralizada que permite la creación de acuerdos de contratos inteligentes entre pares, basada en el modelo blockchain. [86]

Hyperledger: es una plataforma código abierto para blockchain, iniciado en diciembre de 2015 por la Fundación Linux, para apoyar a los ledgers distribuidos basados en la blockchain.[90]

Hyperledger fabric: A comienzos de 2016, el proyecto empezó a aceptar propuestas para codebases y otras tecnologías para incubar, para inclusión potencial como elementos centrales de Hyperledger. Una de las primeras propuestas fue un codebase que combinaba trabajos anteriores de Digital Asset Holdings, el libconsensus de Blockstream, y la OpenBlockchain de IBM.³ Esto fue nombrado más tarde Fabric.[90]

Criptomoneda: Una criptomoneda, criptodivisa (del inglés cryptocurrency) o criptoactivo es un medio digital de intercambio que utiliza criptografía fuerte para asegurar las transacciones financieras, controlar la creación de unidades adicionales y verificar la transferencia de activos. [84]

Ether: es una criptomoneda cuyo blockchain es generado por la plataforma Ethereum

Bitcoin: es un protocolo y red P2P que se utiliza como criptomoneda, sistema de pago y mercancía. [77]

P2P (Peer-to-Peer): Una red peer-to-peer, red de pares, red entre iguales o red entre pares (P2P, por sus siglas en inglés) es una red de ordenadores en la que todos o algunos aspectos funcionan sin clientes ni servidores fijos, sino una serie de nodos que se comportan como iguales entre sí. Es decir, actúan simultáneamente como clientes y servidores respecto a los demás nodos de la red. Las redes P2P permiten el intercambio directo de información, en cualquier formato, entre los ordenadores interconectados.[100]

Smart Contracts: Un contrato inteligente (en inglés Smart contract) es un programa informático que facilita, asegura, hace cumplir y ejecuta acuerdos registrados entre dos o más partes (por ejemplo personas u organizaciones). Como tales ellos les ayudarían en la negociación y definición de tales acuerdos que causarían que ciertas acciones sucedan como resultado de que se cumplan una serie de condiciones específicas. [104]

Minería de bloques: En el sistema de Blockchain, los mineros tienen el rol de la creación de nuevos bloques y la verificación de los bloques añadidos a la cadena. Por un lado, los mineros buscan encontrar el número mediante el cual resuelvan el bloque y encuentren el Hash correspondiente al mismo. Una vez solucionado, se notificará al resto de miembros de la red, que verificarán si el Hash obtenido es el correcto. [54]

GAS: El Gas es el coste que tiene el realizar una operación o un conjunto de operaciones en la red Ethereum. Estas operaciones pueden ser varias: desde realizar una transacción hasta ejecutar un contrato inteligente o crear una aplicación descentralizada. [47]

DApps: Una DApp (pronunciado como [Di-app], similar a como decimos “e-mail” [i-meil]) es una aplicación descentralizada, es decir, una app que no depende de un sistema central, sino que depende de la comunidad de usuarios que la utilizan. La aplicación descentralizada puede ser una app móvil o una aplicación web que interactúa con un contrato inteligente para llevar a cabo su función. [48]

B2C: Negocio a consumidor o B2C (del inglés business-to-consumer) se refiere a la estrategia que desarrollan las empresas comerciales para llegar directamente al cliente o consumidor final. B2C es el tipo de operación que realiza una compañía cuando los clientes son muchos y donde se espera un predominio de la dirección de mercadotecnia. [80]

B2B: Negocio a negocio (del inglés business-to-business o B2B) hace referencia a las transacciones comerciales entre empresas, es decir, a aquellas que típicamente se establecen entre un fabricante y el distribuidor de un producto, o entre un distribuidor y un comercio minorista. [79]

SDKs: Un kit de desarrollo de software (SDK) es generalmente un conjunto de herramientas de desarrollo de software que le permite al programador o desarrollador de software crear una aplicación informática para un sistema concreto, por ejemplo ciertos paquetes de software, entornos de trabajo, plataformas de hardware, computadoras, videoconsolas, sistemas operativos, etcétera. [94]

MSH: Un módulo de seguridad de hardware (HSM) es un procesador criptográfico diseñado específicamente para proteger el ciclo de vida de la clave criptográfica. [98]

Prueba de trabajo: Prueba de Trabajo, o PoW (por sus siglas en inglés), es el algoritmo de consenso original en una red de Blockchain. En la Blockchain, este algoritmo se usa para confirmar transacciones y producir nuevos bloques en la cadena. Con PoW, los mineros compiten entre ellos para completar transacciones en la red y obtener recompensas. [67]

PBFT: La tolerancia a faltas bizantinas (BFT) es la resistencia de un sistema informático tolerante a faltas, en particular los sistemas informáticos distribuidos, a fallas de componentes electrónicos donde hay información imperfecta sobre si un componente falla. En una "falla bizantina", un componente, como un servidor, puede aparecer de manera incoherente, fallando y funcionando para sistemas de detección de fallas, presentando diferentes síntomas a diferentes observadores. Es difícil para los otros componentes declarar que falló y cerrarlo fuera de la red, ya que primero necesitan llegar a un consenso sobre qué componente falla. El término se deriva del problema de los generales bizantinos, donde los actores deben acordar una estrategia concertada para evitar una falla catastrófica del sistema, pero algunos de los actores no son confiables. Un ejemplo de BFT en uso es bitcoin, un sistema de moneda digital peer-to-peer. La red bitcoin funciona en paralelo para generar una cadena de prueba de trabajo de estilo Hashcash. La cadena de prueba de trabajo es la clave para superar las fallas bizantinas y alcanzar una visión global coherente del estado del sistema. [108]

Solidity: Solidity es un lenguaje de alto nivel orientado a contratos. Su sintaxis es similar a la de JavaScript y está enfocado específicamente a la Máquina Virtual de Ethereum (EVM). [65]

Serpent: Lenguaje para desarrollo de Smart Contract.

LLL: es uno de los lenguajes de programación de contrato inteligente originales de Ethereum. [40]

Chaincode: es un fragmento de código que está escrito en uno de los idiomas

compatibles, como Go o Java. Se instala y crea una instancia a través de un SDK o CLI en una red de nodos pares Hyperledger Fabric, lo que permite la interacción con el libro mayor compartido de esa red. [27]

C++: es un lenguaje de programación diseñado en 1979 por Bjarne Stroustrup. La intención de su creación fue extender al lenguaje de programación C mecanismos que permiten la manipulación de objetos. [81]

Golang: Go es un lenguaje de programación concurrente y compilado inspirado en la sintaxis de C. Ha sido desarrollado por Google, y sus diseñadores iniciales son Robert Griesemer, Rob Pike y Ken Thompson. Actualmente está disponible en formato binario para los sistemas operativos Windows, GNU/Linux, FreeBSD y Mac OS X, pudiendo también ser instalado en estos y en otros sistemas con el código fuente. [87]

Python: Python es un lenguaje de programación interpretado cuya filosofía hace hincapié en una sintaxis que favorezca un código legible. Se trata de un lenguaje de programación multiparadigma, ya que soporta orientación a objetos, programación imperativa y, en menor medida, programación funcional. Es un lenguaje interpretado, usa tipado dinámico y es multiplataforma. [102]

Java: Java es un lenguaje de programación de propósito general, concurrente, orientado a objetos, que fue diseñado específicamente para tener tan pocas dependencias de implementación como fuera posible. [92]

Tokens: Representan un activo o utilidad. Son bienes transferibles y pueden representar monedas, puntos de fidelidad, activos en un juego, etc. Incluso pueden canjearse por un servicio que un emisor proporcionará en una fecha posterior. En la Blockchain, un token representa un valor financiero o un activo digital. En un futuro los tokens pueden incluso ser utilizados para representar instrumentos financieros como acciones y bonos.

Tokens ERC20: Son contratos inteligentes que se ejecutan en la blockchain de Ethereum. La mayoría de los tokens creados en Ethereum cumplen con el estandar ERC20, aunque también existen otros estandares como por ejemplo ERC223 y ERC721.