Universidad ORT Uruguay

Facultad de Ingeniería

EasyCharge

Plataforma de gestión de puntos de carga para vehículos eléctricos

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

Andrew Cooper - 253469

Felipe Crosa - 253107

Guillermo Martinicorena - 267356

María Mercedes Bañales - 252944

Tutor: Darío Macchi

2026

Declaración de autoría

Nosotros, Andrew Cooper, Felipe Crosa, Guillermo Martinicorena, María Mercedes Bañales , declaramos que el trabajo que se presenta en esa 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.

Andrew Cooper
12/03/2026

Felipe Crosa
12/03/2026

Guillermo Martinicorena
12/03/2026

María Mercedes Bañales
12/03/2026

Agradecimientos

Deseamos expresar nuestro más profundo agradecimiento a nuestro tutor, Darío Macchi, por su guía constante, su compromiso y su valioso acompañamiento a lo largo de este proyecto.

Asimismo, agradecemos a la Universidad ORT Uruguay y a sus funcionarios por el apoyo brindado para la correcta realización del trabajo. En especial, a los revisores Leonardo Scafarelli, Mariel Feder y Martín Solari, cuyas observaciones y recomendaciones fueron fundamentales para enriquecer el resultado final. Extendemos también un especial agradecimiento a Gastón Mousques y Juan Ignacio Irabedra, por su disposición para validar áreas específicas del proyecto en las que su conocimiento y experiencia resultaron claves.

A la empresa E-Mobility Solutions, gracias por abrirnos sus puertas y confiar en nuestra capacidad. En particular a Alfredo Pintos y Matías Crosa, por su profesionalismo, apoyo y compromiso, que facilitaron un entorno de aprendizaje real y desafiante.

Finalmente, a nuestras familias y amigos, por su paciencia, comprensión y aliento incondicional durante toda la carrera. Su apoyo constante fue el motor que nos impulsó a alcanzar esta meta.

A todos ellos, muchas gracias.

Abstract

En el contexto del crecimiento sostenido del mercado de la electromovilidad, este trabajo presenta el desarrollo de EasyCharge, un prototipo de plataforma tecnológica para la gestión de redes de cargadores de vehículos eléctricos, desarrollado en conjunto con la empresa E-Mobility Solutions en el marco del proyecto final de la carrera Ingeniería en Sistemas. El proyecto surge ante la necesidad de acompañar la expansión de la red de cargadores de la empresa y reducir la dependencia de plataformas de terceros de marca blanca, que limitan la automatización operativa, la personalización funcional y la calidad de la experiencia tanto para usuarios finales como para el personal interno.

La solución propuesta consiste en una plataforma basada en una arquitectura de servicios que permite gestionar la red de cargadores, supervisar su estado operativo y administrar las sesiones de carga. El sistema se comunica con los dispositivos físicos mediante el protocolo OCPP e integra mecanismos de pago digital para la monetización del servicio. Sobre esta base se desarrolló una aplicación móvil que permite a los conductores localizar cargadores, iniciar y finalizar sesiones de carga y gestionar sus pagos, junto con un panel web de administración que brinda al personal interno herramientas para supervisar la infraestructura y gestionar la red de forma eficiente.

El desarrollo se llevó a cabo siguiendo un proceso completo de ingeniería de software que incluyó relevamiento y gestión de requerimientos, diseño arquitectónico, construcción del prototipo y gestión del proyecto y de la calidad, apoyado en metodologías ágiles como Scrum y Kanban dentro del entorno de Software Factory de la universidad. Como resultado, se obtuvo un prototipo funcional de la plataforma EasyCharge que permite gestionar una red de cargadores de vehículos eléctricos y que constituye una base tecnológica para su futura evolución hacia un sistema productivo.

Palabras clave

Electromovilidad, Vehículos eléctricos, Redes de carga, OCPP, Gestión de cargadores, Sistemas distribuidos, Arquitectura de software, Ingeniería de requerimientos, Prototipo funcional, Mantenibilidad, Disponibilidad, Aplicación móvil.

Indice

1. Introducción 18

1.1. ¿Cómo surge el proyecto? 18

1.2. Descripción del equipo 19

1.3. Objetivos 19

1.3.1. Objetivos del producto 19

1.3.1.1. Valor agregado 19

1.3.1.2. Experiencia del usuario 20

1.3.1.3. Preparación para la evolución futura 20

1.3.2. Objetivos del proyecto 20

1.3.2.1. Satisfacción del cliente 20

1.3.2.2. Gestión y control del proyecto 20

1.3.2.3. Satisfacción del equipo 20

1.3.3. Objetivos académicos 21

1.3.3.1. Aplicación de conocimientos adquiridos en la carrera 21

1.3.3.2. Aprendizaje de nuevas tecnologías 21

1.3.3.3. Resolución de problemas en áreas desconocidas 21

1.3.3.4. Aprobación con excelencia académica 21

1.4. Entorno conceptual de Software Factory 21

1.5. Estructura del documento 22

1.5.1. Introducción 22

1.5.2. Problema 22

1.5.3. Solución 22

1.5.4. Marco metodológico 23

1.5.5. Ingeniería de requerimientos 23

1.5.6. Arquitectura y diseño 23
1.5.7. Gestión del proyecto 23
1.5.8. Gestión de la calidad 24
1.5.9. Gestión de la configuración 24
1.5.10. Conclusiones 24
2. Problema 25
2.1. Contexto 25
2.2. Definición del problema 26
2.3. Interesados 27
2.4. Desafíos del proyecto 28
2.4.1. Integración con cargadores eléctricos 28
2.4.2. Desarrollo app móvil 28
2.4.3. Integración con una pasarela de pagos 28
2.4.4. Alcance de la solución 29
3. Solución 30
3.1. Solución propuesta 30
3.2. Principales funcionalidades 30
3.3. Alcance del proyecto 31
4. Marco metodológico 33
4.1. Características del proyecto 33
4.1.1. Cliente comprometido 33
4.1.2. Cliente experto 33
4.1.3. Soluciones similares 33
4.1.4. Alcance extenso 34
4.2. Características del Equipo 34

4.2.1. Experiencia laboral 34
4.2.2. Desconocimiento del dominio 35
4.2.3. Incertidumbre técnica 35
4.2.4. Relacionamiento y antecedentes 35
4.3. Metodología de trabajo 35
4.3.1. Metodología en etapa de investigación 36
4.3.2. Metodología en etapa de desarrollo 36
4.4. Ciclo de vida del proyecto 36
5. Ingeniería de requerimientos 39
5.1. Proceso 39
5.1.1. Relevamiento 39
5.1.1.1. Técnicas abiertas 39
5.1.1.1.1. Entrevista semiestructurada 40
5.1.1.1.2. Observación directa 40
5.1.1.1.3. Brainstorming 40
5.1.1.2. Técnicas cerradas 40
5.1.1.2.1. Análisis de soluciones existentes (Ingeniería inversa) 40
5.1.1.2.2. Análisis de documentación 41
5.1.2. Especificación 41
5.1.2.1. Actores del sistema 41
5.1.2.1.1. Empleado de E-Mobility Solutions 41
5.1.2.1.2. Usuario final del cargador 42
5.1.3. Validación 42
5.1.3.1. Validación técnica 42
5.1.4. Verificación 43

5.2. Requerimientos funcionales 43
5.2.1. Requerimientos funcionales del panel administrativo 43
5.2.2. Requerimientos Funcionales de la Aplicación Móvil 49
5.3. Requerimientos no funcionales 51
5.3.1. RNF 1 - Mantenibilidad del código 51
5.3.2. RNF 2 - Documentación técnica completa 51
5.3.3. RNF 3 - Usabilidad e interfaz intuitiva 52
5.3.4. RNF 4 - Compatibilidad con OCPP 1.6 52
5.3.5. RNF 5 - Compatibilidad móvil 52
5.3.6. RNF 6 - Tolerancia a fallas de conectividad 52
5.3.7. RNF 7 - Sistema disponible 52
5.3.8. RNF 8 - Tiempo de inicio de carga 52
5.3.9. RNF 9 - Tiempo de actualización del estado de los cargadores 53
5.3.10. RNF 10 - Seguridad en los pagos 53
5.3.11. RNF 11 - Observabilidad del sistema 53
5.4. Conclusiones y lecciones aprendidas 53
6. Arquitectura y diseño 55
6.1. Visión general de la arquitectura 55
6.2. Atributos de calidad 56
6.2.1. Mantenibilidad 57
6.2.1.1. E1: Cobertura de componentes de flujos críticos 58
6.2.1.2. E2: Control de la complejidad ciclomática 59
6.2.2. Disponibilidad 59
6.2.2.1. E3: Disponibilidad de la operativa crítica 60
6.2.2.2. E4: Recuperación ante fallos de conexión 60

6.2.3. Interoperabilidad 61
6.2.3.1. E5: Cobertura de mensajes de transactions y status del protocolo OCPP 61
6.2.3.2. E6: Integración multi-marca 61
6.2.4. Portabilidad 62
6.2.4.1. E7: Compatibilidad multiplataforma 62
6.2.5. Usabilidad 63
6.2.5.1. E8: Navegabilidad y ejecución de flujos críticos sin asistencia 63
6.2.6. Atributos de calidad no prioritarios 63
6.3. Restricciones 64
6.4. Detalle de la arquitectura 64
6.4.1. Vista componentes y conectores 65
6.4.1.1. Decisiones de Diseño 67
6.4.2. Vista de módulos 70
6.4.2.1. Emobility Backend 71
6.4.2.1.1. Decisiones de diseño 73
6.4.2.2. Admin UI & mobile app 75
6.4.2.2.1. Decisiones de diseño 75
6.4.3. Vista de despliegue 76
6.4.3.1. Decisiones de diseño 78
6.5. Tecnologías 79
6.5.1. Lenguajes 79
6.5.2. Frameworks y librerías 79
6.6. Flujos críticos del sistema 82
6.6.1. Flujo de conexión de cargador 83

6.6.2. Flujo de carga 86
6.6.2.1. Flujo de solicitud de carga paga 86
6.6.2.2. Flujo de comienzo de carga 90
6.6.2.3. Flujo de progreso de carga 92
6.6.2.4. Flujo de fin de carga 92
6.7. Evaluación de la arquitectura 95
6.8. Conclusiones y lecciones aprendidas 96
7. Gestión del proyecto 98
7.1. Etapas y principales hitos del proyecto 98
7.1.1 Hitos Identificados 98
7.2. Gestión de la etapa de investigación 100
7.2.1. Tablero de kanban 100
7.2.2. Flujo de trabajo 101
7.3. Gestión de la etapa de desarrollo 101
7.3.1. Artefactos de Scrum 102
7.3.1.1. *Product backlog* 102
7.3.1.2. *Sprint backlog* 102
7.3.1.3. Incremento de valor 103
7.3.2 Ceremonias de scrum 103
7.3.2.1. *Sprint planning* 103
7.3.2.2. *Daily Scrum* 104
7.3.2.3. *Sprint review* 104
7.3.2.4. *Sprint retrospective* 105
7.3.3. Roles de Scrum 105
7.3.4. Definition of done 106

7.3.5. Métricas de gestión 107
7.3.5.1. Velocidad vs Compromiso 107
7.3.5.2. Distribución del esfuerzo por área de proceso 109
7.4. Herramientas de gestión 110
7.5. Gestión de la comunicación 111
7.6. Gestión de la documentación 111
7.7. Roles transversales 112
7.8. Gestión de riesgos 113
7.8.1 Clasificación de riesgos 113
7.8.1.1 Riesgos técnicos 114
7.8.1.1.1. R1: Complejidad del protocolo OCPP 114
7.8.1.1.2. R2: Integración con API de pago 116
7.8.1.1.3. R3: Aprender React Native en paralelo al desarrollo 117
7.8.1.2 Riesgos de infraestructura 118
7.8.1.2.1. R4: Acceso físico a los cargadores para pruebas 119
7.8.1.3 Riesgos organizacionales 121
7.8.1.3.1. R5: Dependencia del cliente para especificaciones o validaciones 122
7.8.1.3.2. R6: Rotación o carga académica del equipo de desarrollo 123
7.8.1.3.3. R7: Alcance de la solución 124
7.8.2 Análisis cuantitativo 126
7.8.3 Resultados 128
7.9. Conclusiones y lecciones aprendidas 129
8. Gestión de la calidad 130
8.1. Definición de calidad 131

8.2. Prácticas de aseguramiento de la calidad 131
8.2.1 Prácticas orientadas a la calidad de producto 132
8.2.1.1. Aplicación de estándares 132
8.2.1.2. Revisión del código 134
8.2.1.3. Pruebas unitarias automatizadas 135
8.2.1.4. Pruebas manuales 137
8.2.1.6 Métricas de calidad y mantenibilidad del sistema 140
8.2.1.6.1 Cobertura de código 140
8.2.1.6.2 Análisis de la complejidad ciclomática mediante SonarQube 141
8.2.2 Prácticas orientadas a la calidad de uso 143
8.2.2.1. Pruebas manuales 143
8.2.2.1.1. Reporte de Incidentes 144
8.2.2.2. Pruebas con usuarios reales 144
8.2.2.3. Instancias de *Sprint Review* 146
8.2.2.3. Análisis de la plataforma actual y criterios de diseño de interfaz 147
8.2.3 Prácticas orientadas a la calidad de proceso 148
8.2.3.2. Instancias de *Sprint Retrospective* 148
8.2.3.3 Gestión de riesgos 148
8.2.3.4. Satisfacción del cliente 149
8.3 Calidad incorporada al flujo de trabajo 150
8.4. Uso de herramientas de Inteligencia Artificial 150
8.4.1 Uso de IA en UX/UI 151
8.4.2 Uso de IA en el desarrollo de software 155
8.4.2.1 Uso de IA para generación de pruebas unitarias automatizadas 156
8.5. Conclusiones y lecciones aprendidas 156

9. Gestión de la configuración 158
9.1. Identificación de elementos de configuración 158
9.2. Herramientas utilizadas 159
9.2.1 GitHub 159
9.2.2 ClickUp 159
9.2.3 Google Drive 160
9.3. Organización de los repositorios 160
9.4. Gestión de versiones 161
9.4.1 Gitflow 162
9.4.2 *Trunk-Based Development* 163
9.5. Conclusiones y lecciones aprendidas 163
10. Conclusiones 165
10.1. Estado Actual 165
10.2. Reflexiones Generales 166
10.3. Principales Lecciones Aprendidas 166
10.3.1. Ingeniería de Requerimientos 166
10.3.2. Arquitectura y Diseño 167
10.3.3. Gestión del Proyecto 168
10.3.4. Gestión de la calidad 168
10.4. Evaluación de los Objetivos 169
10.4.1. Objetivos de Producto 169
10.4.1.1. Valor agregado 169
10.4.1.2. Experiencia del usuario 169
10.4.1.3. Preparación para la evolución futura 170
10.4.2. Objetivos de Proyecto 170

10.4.2.1. Satisfacción del cliente 170
10.4.2.2. Gestión y control del proyecto 170
10.4.2.3. Satisfacción del equipo 171
10.4.3. Objetivos Académicos 171
10.4.3.1. Aplicación de conocimientos adquiridos en la carrera 171
10.4.3.2. Aprendizaje de nuevas tecnologías 171
10.4.3.3. Resolución de problemas en áreas desconocidas 172
10.4.3.4. Aprobación con excelencia académica 172
10.4.4. Evaluación 172
10.5. Próximos pasos 172
11. Referencias bibliográficas 175
12. Anexos 180
12.1. Scrum vs. Kanban 180
12.1.1. Kanban: Flexibilidad y Flujo Continuo 180
12.1.2. Scrum: Estructura y Entrega Incremental 181
12.1.3. Conclusión 181
12.2. Entrevista semiestructurada con E-Mobility Solutions 183
12.3. Visita a las instalaciones de E-Mobility Solutions 186
12.4. Analisis de la Plataforma Actual (VoltBras) 187
12.5. Investigación de OCPP 194
12.5.1. Análisis de la Documentación 194
12.5.2. Prueba de Concepto 198
12.5.2.1. Código del PoC 199
12.6. Ejemplo de historia de usuario 203
12.7. Visualización de los flujos críticos del sistema 204

12.7.1. Visualización de la conexión del cargador 204
12.7.1.1. Visualización del panel administrativo 204
12.7.1.1. Visualización de la aplicación móvil 207
12.7.2. Visualización del flujo de carga 211
12.7.2.1. Visualización del panel administrativo 211
12.7.2.1. Visualización de la aplicación móvil 214
12.8. Modelado de Datos 227
12.8.1. Uso de ORM 227
12.8.2. Diagrama MER 229
12.8.3. Decisiones de Diseño 230
12.9. Evidencia de Ceremonias de Scrum 231
12.10. Evidencia de planilla para gestión de riesgos 236
12.11 Estándares de Código 238
12.12 Evidencia complejidad ciclomática 241
12.13 Evidencia de un reporte de *bug* 246
12.14. Plan de Validación Operativa 248
12.14.1. Objetivos del plan y criterios de evaluación 248
12.14.1.1 Evaluación 249
12.14.2. Ejecución 249
12.14.2.1. Prerrequisitos 250
12.14.2.2. Fases de Validación 250
12.14.2.3. Validaciones Complementarias 251
12.14.2.4. Equipo Mínimo Recomendado 251
12.14.3. Salida a Producción 252
12.14.3.1. Equipo de Mantenimiento 253

12.14. Costos 254
12.15. Guia de Despliegue 256
12.15.1. Despliegue local 256
12.15.2. Despliegue en la nube 257
12.15.2.1. Elección de la región 257
12.15.2.2. Consideraciones de seguridad 257
12.15.2.3. Consideraciones de rendimiento 258
12.15.2.4. Consideraciones de observabilidad 258
12.15.2.5. Consideraciones de despliegue 258
12.15.2.6. Configuración de RDS 258
12.15.2.7. Configuración de Amazon MQ 259
12.15.2.8. Configuración de Elasticache 259
12.15.2.9. Configuración de ECR 259
12.15.2.10. Configuración de ECS 260
12.15.2.11. Configuración de S3 260
12.15.2.12. Evaluación de Costos 261
12.15.3. Despliegue de las aplicaciones móviles 261
12.15.3.1. Plataforma Android (Google Play Store) 261
12.15.3.2. Plataforma iOS (Apple App Store) 262
12.15.3.3. Consideraciones del *backend* 263
12.16. Requerimientos No Implementados 264
12.17. Cartas de Satisfacción del cliente 267
12.17.1 Carta de Satisfacción de Alfredo Pintos 267
12.17.2 Carta de Satisfacción de Matías Crosa 269
12.18. Documentación de Endpoints 269

1. Introducción

Este capítulo tiene como propósito brindar contexto sobre EasyCharge, proyecto de tesis para la obtención del título de Ingeniería en Sistemas de la Universidad ORT Uruguay.

1.1. ¿Cómo surge el proyecto?

El equipo participó en la feria de proyectos de la ORT con el objetivo de identificar una consigna para el trabajo. No obstante, tras evaluar las propuestas presentadas y realizar reuniones con algunos de los clientes, no se encontró la motivación necesaria para abordar ninguna de ellas. En consecuencia, se decidió buscar una alternativa externa que se ajustara mejor a los intereses del equipo, dando lugar a la posibilidad de construir un proyecto con la empresa E-Mobility Solutions[1].

La elección de E-Mobility Solutions surge por:

- El atractivo del dominio de la movilidad eléctrica, tanto por su carácter innovador como por su creciente relevancia.
- La intriga que generaba tratar un área hasta entonces desconocida para los integrantes del equipo.
- El enfoque claro y ambicioso sobre la problemática a abordar y el impacto potencial de la solución.

Finalmente, la complejidad técnica y funcional de la propuesta representó un desafío acorde al alcance del trabajo, permitiendo aplicar y profundizar los conocimientos adquiridos a lo largo de la carrera desde una perspectiva académica y profesional.

[1] https://www.emobility-uy.com

1.2. Descripción del equipo

El equipo está conformado por cuatro estudiantes de la carrera de Ingeniería en Sistemas de la Universidad ORT Uruguay.

- María Mercedes Bañales: *Full-Stack Developer* en Light-it.
- Guillermo Martinicorena: *Full-Stack Developer* en Light-it.
- Felipe Crosa: *Full-Stack Developer* en Light-it.
- Andrew Cooper: *Full-Stack Developer* en EagerWorks.

1.3. Objetivos

Al comienzo del proyecto académico se definieron una serie de objetivos con el propósito de establecer un marco de referencia para evaluar la dirección del mismo. Estos objetivos fueron definidos en conjunto por todos los integrantes del equipo, con el fin de garantizar un compromiso compartido y una alineación en las metas a cumplir.

Los objetivos fueron categorizados según 3 áreas: de producto, de proyecto y académicos. Esto se hizo con el fin de reflejar las principales dimensiones del trabajo y facilitar su evaluación.

1.3.1. Objetivos del producto

Los objetivos de producto se centran en el impacto esperado del sistema y los resultados que la solución debe alcanzar para E-Mobility Solutions y sus usuarios finales.

1.3.1.1. Valor agregado

La solución desarrollada deberá satisfacer los requerimientos funcionales y no funcionales identificados durante la ingeniería de requerimientos, asegurando su alineación con las necesidades de E-Mobility Solutions, aportando valor.

1.3.1.2. Experiencia del usuario

La solución debe ofrecer una experiencia de uso clara, confiable para los usuarios finales del sistema.

1.3.1.3. Preparación para la evolución futura

Asegurar que la solución mantenga un nivel de calidad técnica que facilite su mantenimiento, corrección de errores y extensión futura, mediante buenas prácticas de desarrollo, pruebas y documentación.

1.3.2. Objetivos del proyecto

Los objetivos del proyecto buscan evaluar la planificación, gestión y ejecución del trabajo, asegurando un equilibrio entre los resultados entregados, el control del proceso y la motivación del equipo.

1.3.2.1. Satisfacción del cliente

Asegurar la satisfacción de E-Mobility Solutions con respecto a la gestión y la dinámica de trabajo adoptada. Garantizar una comunicación fluida, acuerdos claros y un seguimiento continuo que permitan alinear expectativas y fortalecer la relación de trabajo a lo largo de todo el proceso.

1.3.2.2. Gestión y control del proyecto

Mantener el proyecto rigurosamente gestionado y bajo control a lo largo de todo su ciclo de vida. Esto implica el cumplimiento del cronograma establecido, la correcta administración del alcance y la mitigación proactiva de riesgos, asegurando la entrega de la solución dentro de las restricciones de tiempo y esfuerzo definidas por el marco académico.

1.3.2.3. Satisfacción del equipo

Asegurar un alto nivel de motivación y compromiso por parte de todos los integrantes a lo largo del proyecto. Se busca que el equipo encuentre valor tanto en el proceso de

trabajo colaborativo como en los desafíos técnicos abordados, promoviendo un ambiente de aprendizaje y mejora continua, mientras se construye un producto final sólido que refleje el esfuerzo conjunto a nivel académico y profesional.

1.3.3. Objetivos académicos

Los objetivos académicos se orientan a los aprendizajes y competencias que se busca desarrollar a lo largo del proceso.

1.3.3.1. Aplicación de conocimientos adquiridos en la carrera

Aplicar y poner en práctica los conceptos de arquitectura e ingeniería de software, metodologías de gestión y habilidades de liderazgo adquiridos a lo largo de la carrera, incorporándolos en el análisis, diseño y desarrollo de la solución propuesta.

1.3.3.2. Aprendizaje de nuevas tecnologías

Incorporar al menos una tecnología o herramienta no abordada previamente en la formación curricular, logrando su correcta integración en la solución propuesta.

1.3.3.3. Resolución de problemas en áreas desconocidas

Desarrollar la capacidad de analizar y comprender problemáticas asociadas al dominio y a contextos de negocio ajenos al conocimiento previo del equipo.

1.3.3.4. Aprobación con excelencia académica

Obtener una calificación final igual o superior al 90%, reflejando un nivel de excelencia en el análisis, diseño, fundamentación y presentación del trabajo.

1.4. Entorno conceptual de 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.

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.

1.5. Estructura del documento

1.5.1. Introducción

En esta sección se presenta el contexto general del proyecto EasyCharge, desarrollado como trabajo de tesis para la obtención del título de Ingeniería en Sistemas en la Universidad ORT Uruguay. Se describe el origen de la iniciativa, el vínculo con la empresa E-Mobility Solutions y la motivación del equipo para abordar el desafío propuesto. Asimismo, se introduce la conformación del equipo de trabajo, los objetivos definidos para el proyecto y el entorno académico de ORT Software Factory en el cual se desarrolla la experiencia.

1.5.2. Problema

En esta sección se presenta el contexto de negocio de E-Mobility Solutions y se describen las limitaciones que enfrenta la empresa en la gestión de su red de cargadores eléctricos. Se analiza la dependencia actual de una plataforma de terceros, así como las ineficiencias operativas y los problemas de experiencia de usuario que motivan la necesidad de una nueva solución tecnológica. Asimismo, se identifican los principales interesados del proyecto y los desafíos técnicos y de alcance asociados a su desarrollo.

1.5.3. Solución

En esta sección se presenta la solución propuesta para abordar las limitaciones identificadas en la operación de E-Mobility Solutions. Se describe la plataforma diseñada, compuesta por un sistema de gestión interno y una aplicación móvil para los conductores de vehículos eléctricos. Asimismo, se detallan las principales funcionalidades del sistema y se define el alcance del proyecto, centrado en el desarrollo de un prototipo funcional junto con la documentación técnica y operativa necesaria para su futura evolución.

1.5.4. Marco metodológico

Esta sección expone el enfoque metodológico utilizado durante el desarrollo del proyecto. Se describe la metodología de trabajo adoptada por el equipo, basada en prácticas ágiles, y cómo esta permitió organizar el trabajo, gestionar cambios y mantener una comunicación continua con el cliente.

1.5.5. Ingeniería de requerimientos

En esta sección se describe el proceso de ingeniería de requerimientos seguido durante el proyecto, incluyendo las actividades de relevamiento, especificación, validación y verificación de los requisitos. Se presentan las técnicas utilizadas para comprender el dominio del problema, así como la definición de los actores del sistema y la organización de los requerimientos mediante historias de usuario. Finalmente, se detallan los requerimientos funcionales y no funcionales que guían el desarrollo de la solución propuesta.

1.5.6. Arquitectura y diseño

Esta sección presenta la arquitectura del sistema y las principales decisiones de diseño adoptadas durante el proyecto. Se describen los componentes principales de la solución, su interacción y la forma en que la arquitectura propuesta permite cumplir con los requerimientos y atributos de calidad definidos.

1.5.7. Gestión del proyecto

En esta sección se describe cómo se planificó, ejecutó y evaluó el proyecto desde la perspectiva de gestión. Se presentan las principales etapas del trabajo y los hitos más relevantes alcanzados durante su desarrollo. Asimismo, se detalla la aplicación práctica de los marcos de trabajo utilizados. Finalmente, se describen los artefactos, ceremonias, roles y métricas empleadas para dar seguimiento al progreso del proyecto y evaluar objetivamente los resultados obtenidos.

1.5.8. Gestión de la calidad

En esta sección se describe el enfoque adoptado para asegurar la calidad del sistema a lo largo del proyecto, incluyendo la definición de calidad utilizada y las principales prácticas implementadas para garantizarla durante el desarrollo.

1.5.9. Gestión de la configuración

En esta sección se presentan las prácticas y herramientas utilizadas para gestionar la evolución del sistema, incluyendo el control de versiones, la organización de repositorios, el manejo de ramas y los procesos de integración del código.

1.5.10. Conclusiones

En esta sección se presentan las conclusiones finales del proyecto, evaluando el grado de cumplimiento de los objetivos planteados. Asimismo, se reflexiona sobre los resultados obtenidos, las principales lecciones aprendidas durante el desarrollo y posibles líneas de trabajo futuro.

2. Problema

En esta sección se exploran las necesidades de E-Mobility Solutions al comenzar este proyecto y las causas que lo llevaron a buscar la solución implementada.

2.1. Contexto

E-Mobility Solutions es una compañía fundada en el año 2017, dedicada a ofrecer un servicio integral de soluciones de carga para vehículos eléctricos. Desde sus inicios, la empresa ha abarcado la cadena de valor completa del servicio, gestionando tanto la comercialización e instalación de los equipos, como su posterior administración y mantenimiento.

El modelo de negocio de E-Mobility Solutions se desarrolla en un entorno dinámico marcado por el auge de la electromovilidad. En los últimos años, el mercado automotor ha experimentado un crecimiento sostenido en la venta de vehículos eléctricos, impulsado tanto por incentivos gubernamentales como por un cambio en la preferencia de los consumidores hacia energías más limpias [1].

Este contexto favorable ha impulsado directamente la expansión de la empresa, manteniendo un crecimiento sostenido de alrededor del 30% anual compuesto en la venta de cargadores. A la fecha, la organización ha logrado comercializar una cantidad aproximada de 5.000 equipos, consolidando una base instalada significativa.

Sin embargo, este crecimiento exponencial del sector también ha traído consigo un aumento en la carga operativa y se ha transformado en un mercado sumamente atractivo para nuevos competidores. Esto obliga a las empresas como E-Mobility Solutions a optimizar procesos y ofrecer diferenciadores para sostener el ritmo de crecimiento y asegurar la calidad del servicio ante una demanda cada vez más exigente.

2.2. Definición del problema

E-Mobility Solutions enfrenta actualmente desafíos para optimizar y escalar su operativa del servicio postventa, en particular el sector de gestión de cargas e informes de operación. Esta área operativa se centra en la gestión, control y diagnóstico remoto de los equipos mediante un panel administrativo, como también en el control de acceso y la comercialización de una red distribuida de carga mediante una aplicación móvil.

La limitante radica en la infraestructura tecnológica utilizada actualmente. La gestión actual recae en una plataforma marca blanca (*white labe*l), provista por la empresa brasileña VoltBras[2] (ver sección 12.4 del Anexo). Este modelo ofrece un software predefinido que permite adaptar la identidad visual de la empresa. Pero al ser un producto estandarizado, el cliente carece de libertad técnica para modificar el código fuente o implementar funcionalidades específicas a medida. Si bien esta solución permitió un inicio rápido de operaciones y cubre las necesidades básicas, hoy presenta limitaciones para acompañar las necesidades de la empresa.

La herramienta de administración de VoltBras resulta ineficiente para las tareas que realiza habitualmente E-Mobility Solutions. La plataforma dificulta la automatización de procesos internos de diagnóstico y reporte, generando una carga operativa manual que se vuelve cada vez más compleja de sostener a medida que la empresa continúa creciendo.

En lo que respecta a la comercialización de energía, el sistema presenta ineficiencia en flujos operativos claves, como el de cargas y el procesamiento de pagos. Estas incidencias generan una experiencia de usuario degradada, lo cual resulta contraproducente en un mercado donde la calidad del servicio es el principal factor de fidelización.

VoltBras define su hoja de ruta en base a su modelo de negocio y objetivos. El margen de acción y participación que tiene E-Mobility Solutions en decisiones de producto es

[2] https://VoltBras.com

prácticamente nulo, no solo sobre las limitantes actuales sino también para necesidades futuras.

En síntesis, E-Mobility Solutions enfrenta limitaciones para escalar su servicio postventa y su red de carga debido a la dependencia de una plataforma de tercero que no se adapta a sus procesos ni a su estrategia de crecimiento, generando sobrecarga operativa interna y una experiencia de usuario final por debajo de los estándares que exige el mercado de electromovilidad.

2.3. Interesados

Para el desarrollo de este proyecto, se ha realizado un mapeo de los interesados, categorizándolos según su nivel de involucramiento y el impacto que el sistema tendrá sobre ellos.

Interesados Directos (Internos y Usuarios):

- **Directorio de E-Mobility Solutions:** su principal interés es obtener autonomía tecnológica, optimizar la operativa y asegurar la escalabilidad del negocio.
- **Personal Operativo y de Soporte:** Son los usuarios finales del sistema de gestión interno. Su interés radica en contar con una herramienta que automatice el diagnóstico de los cargadores y reduzca la carga de trabajo manual.
- **Conductores de vehículos eléctricos:** Son los usuarios finales de la aplicación móvil. Demandan una experiencia de uso intuitiva, con procesos de pago ágiles y flujos de carga sin fricciones.
- **Equipo de Desarrollo:** Quienes heredarán el sistema para su mantenimiento y evolución constituyen un grupo de interesados fundamental.

Interesados Indirectos (Externos):

- **VoltBras:** Representa un interesado externo con impacto negativo frente al éxito del proyecto, dado que la implementación de esta solución a medida implicaría la eventual rescisión de su contrato como proveedor de la plataforma de marca blanca.

- **Empresas Competidoras:** Otros actores del sector de la venta de cargas y/o cargadores como UTE. Se ven afectados indirectamente, ya que el éxito de esta plataforma otorgará a E-Mobility Solutions una ventaja competitiva significativa en términos de eficiencia operativa y calidad de servicio al usuario.

2.4. Desafíos del proyecto

2.4.1. Integración con cargadores eléctricos

Uno de los principales desafíos del proyecto radica en la integración con los cargadores eléctricos mediante el Open Charge Point Protocol (OCPP), estándar de la industria definido por la Open Charge Alliance[3]. El dominio de la carga de vehículos eléctricos es completamente nuevo para el equipo, lo que introduce una alta incertidumbre no solo en la comunicación con el hardware, sino también en la comprensión de la lógica de negocio asociada a las sesiones de carga, la medición de energía y las particularidades del ecosistema de electromovilidad.

2.4.2. Desarrollo app móvil

El desarrollo de la aplicación móvil representa un desafío significativo debido a que el equipo no posee experiencia previa en este tipo de implementaciones. Como resultado, se debe atravesar una curva de aprendizaje significativa, especialmente considerando que la app implementa flujos críticos del negocio que impactan directamente en la experiencia del usuario final.

2.4.3. Integración con una pasarela de pagos

La integración con una pasarela de pagos representa un desafío importante debido a la complejidad propia del flujo transaccional, manejando correctamente escenarios como rechazos, reintentos, confirmaciones tardías o posibles inconsistencias entre el estado del pago y el estado de la sesión de carga. El flujo exige consideraciones estrictas de

[3] https://openchargealliance.org/

seguridad en el tratamiento y transmisión de datos sensibles, ya que cualquier falla impacta de manera inmediata tanto en la confianza del cliente como en la operación general del sistema.

2.4.4. Alcance de la solución

El alcance de la solución representa un desafío en sí mismo, ya que el sistema debe cubrir múltiples dimensiones del negocio, y además, dado el contexto académico, debe desarrollarse bajo una fecha de entrega fija y con una cantidad de esfuerzo limitada. Esto exige una cuidadosa priorización y definición de límites claros desde el inicio.

3. Solución

En esta sección se detalla la propuesta técnica diseñada para responder a los desafíos identificados, sus funcionalidades clave y el alcance del prototipo desarrollado.

3.1. Solución propuesta

A partir de los problemas identificados, E-Mobility Solutions solicitó el desarrollo de una plataforma a medida que centralice la gestión de su red de cargadores y la interacción con los usuarios finales.

La solución propuesta tiene como objetivo construir un prototipo funcional que sirva como base tecnológica sólida para validar la arquitectura, los flujos críticos de negocio y las integraciones con los distintos actores del ecosistema. Debido al alcance y la complejidad del sistema, así como a las restricciones de tiempo y recursos, en esta etapa no se busca disponer de un sistema productivo definitivo. El foco está en disponer de un prototipo funcional robusto, apto para su validación operativa y para actuar como base de futuras evoluciones del producto.

La solución se estructura en dos componentes principales. Por un lado, el sistema de gestión interno permite al personal de E-Mobility Solutions centralizar la operativa mediante un panel administrativo web, facilitando tareas de monitoreo, diagnóstico y administración de la red de cargadores. Por otro lado, la aplicación móvil está orientada a los conductores, permitiéndoles interactuar de manera directa con los cargadores mediante códigos QR mediante códigos QR, gestionar sus sesiones de carga y realizar los pagos de forma autónoma. Esta herramienta fue diseñada para facilitar el acceso al servicio y proporcionar una experiencia de usuario fluida y consistente.

3.2. Principales funcionalidades

La plataforma propuesta reúne un conjunto de capacidades orientadas a cubrir las necesidades operativas y comerciales más relevantes de E-Mobility Solutions,

abarcando tanto la administración de la red de carga como la interacción con los usuarios finales.

- **Gestión de cargadores:** permite supervisar en tiempo real el estado operativo de cada punto de carga, facilitando el monitoreo y el diagnóstico remoto.
- **Gestión de cargas:** posibilita que los usuarios realicen la carga de su vehículo y registra el detalle de cada sesión (energía consumida, duración, costos asociados), permitiendo la trazabilidad técnica y la posterior facturación.
- **Gestión de usuarios y organizaciones:** permite a E-Mobility Solutions administrar los perfiles de los conductores y la estructura de las organizaciones propietarias de los cargadores, asegurando un control adecuado de accesos y permisos.
- **Gestión de pagos:** contempla la integración con una pasarela de pagos para procesar los cobros asociados a las sesiones de carga, registrando el estado de cada transacción y permitiendo la conciliación entre los pagos realizados por los usuarios y las sesiones de carga efectivamente ejecutadas.

3.3. Alcance del proyecto

Como se destacó anteriormente, el alcance de este proyecto se centra en el diseño, desarrollo y entrega de la plataforma en calidad de prototipo en una etapa de validación operativa. Al finalizar el proyecto, el equipo entrega todos los artefactos pertinentes para el desarrollo y la ejecución de la solución, los cuales incluyen:

- **Código fuente:** transferencia de todos los repositorios correspondientes a los servicios desarrollados y a las pruebas de concepto implementadas.
- **Documentación técnica:** descripción de la arquitectura del sistema, de los endpoints expuestos por la API, del modelado de datos y de los procesos de despliegue.
- **Documentación operativa:** listado de los requerimientos no implementados y un plan de acción para la puesta en operación de la plataforma en un entorno productivo.

4. Marco metodológico

En este capítulo se profundiza en las características centrales del proyecto, describiendo la metodología adoptada, el ciclo de vida completo, y la distribución de responsabilidades.

4.1. Características del proyecto

A continuación, se hace énfasis en las características del proyecto y se descomponen aquellos aspectos que influyeron en la decisión sobre qué metodología de desarrollo utilizar.

4.1.1. Cliente comprometido

Se destaca la fuerte disposición del cliente para contribuir activamente con el proyecto. Alfredo Pintos (dueño de la empresa) y Matías Crosa (ex jefe de operaciones) demostraron desde el inicio un alto nivel de compromiso con el equipo, manteniéndose disponibles para consultas y revisiones y facilitando el acceso a las instalaciones y recursos de la empresa necesarios para llevar adelante la solución.

4.1.2. Cliente experto

El cliente cuenta con un conocimiento profundo del mercado de la movilidad eléctrica y de los objetivos comerciales del producto. Sus años de experiencia en la industria se reflejan en su comprensión de los flujos de carga, las necesidades de los usuarios e incluso en el manejo técnico de los cargadores.

4.1.3. Soluciones similares

El cliente, como se mencionó en la sección Problema, cuenta actualmente con una plataforma en funcionamiento (VoltBras). Si bien presenta limitaciones y diversos problemas que han motivado el desarrollo del presente proyecto, su existencia constituye un antecedente clave a considerar. La plataforma actual sirve como referencia comparativa y contribuye a reducir incertidumbres.

4.1.4. Alcance extenso

Como se anticipó en la sección Desafíos del proyecto, el alcance es uno de los desafíos principales que enfrenta el equipo. Influyen tanto las múltiples dimensiones del negocio como la complejidad técnica que estas introducen.

El modelo de gestión “el triángulo de hierro” plantea la relación entre tres limitantes: alcance, tiempo y costo [2]. Esta herramienta nos permite analizar el impacto que tienen entre sí estas limitantes.

En el caso de este proyecto las variables tiempo y costo se consideran restringidas debido a los plazos académicos inamovibles y la cantidad de integrantes del equipo. Estas restricciones son claves a tener en cuenta al gestionar el alcance de la solución.

4.2. Características del Equipo

El equipo y sus características constituyen otra arista fundamental a la hora de definir la metodología de trabajo para llevar adelante la solución. En el capítulo de Introducción se presentó un pantallazo general de la conformación del equipo, integrado por cuatro estudiantes de la carrera de Ingeniería en Sistemas. En esta sección se profundizará en ciertos aspectos específicos de dicha conformación que influyen directamente en la metodología seleccionada.

4.2.1. Experiencia laboral

Todos los miembros del equipo cuentan con experiencia laboral previa en la industria del software. Presentan experiencia desempeñándose en proyectos de desarrollo, aplicando diversas tecnologías y adoptando marcos de trabajo ágiles, principalmente Scrum. Sin embargo, esto implica también una limitante en las horas disponibles para este proyecto debido a las responsabilidades a cumplir en los respectivos trabajos.

4.2.2. Desconocimiento del dominio

La ausencia de experiencia previa del equipo en el sector de movilidad eléctrica representa un factor que condiciona la planificación del proyecto y por ende las metodologías seleccionadas.

4.2.3. Incertidumbre técnica

A pesar de poseer experiencia técnica en diversos lenguajes de programación, el equipo no contaba con conocimientos previos en el desarrollo de aplicaciones móviles ni tampoco en el diseño de sistemas distribuidos. A esta curva de aprendizaje se le sumó un profundo desconocimiento del dominio tecnológico de la movilidad eléctrica, particularmente en lo que respecta al control de los cargadores físicos. Esta situación introdujo un doble desafío: por una parte, la necesidad de investigar e implementar desde cero el protocolo de comunicación OCPP. A su vez, a esto se le suma la complejidad inherente de lograr una integración entre el código desarrollado y los componentes de *hardware* en el mundo real.

4.2.4. Relacionamiento y antecedentes

Los miembros del equipo mantienen un vínculo personal y académico previo, habiendo cursado la mayor parte de la carrera a la par y haber colaborado en obligatorios. Este conocimiento mutuo implica un conocimiento de las fortalezas y debilidades individuales, así como también un entendimiento compartido de las formas de trabajo.

4.3. Metodología de trabajo

Dadas las características identificadas, se decidió trabajar bajo el marco de metodologías ágiles. Esta filosofía permite obtener retroalimentación continua y mantener la adaptabilidad necesaria. Esto resulta beneficioso ante el dominio de negocio complejo, la incertidumbre presente y la alta disponibilidad de E-Mobility Solutions. El análisis comparativo que fundamenta la elección de los marcos de trabajo específicos para cada etapa del proyecto se encuentra detallado en la sección 12.1 del Anexo.

La construcción de la solución se conformó de dos grandes fases: la etapa de investigación y la etapa de desarrollo. En lugar de adoptar un único marco rígido para todo el proyecto, se seleccionaron diferentes herramientas según los desafíos específicos de cada instancia. El análisis detallado de estas metodologías se aborda en la sección Gestión del proyecto.

4.3.1. Metodología en etapa de investigación

Para la etapa inicial de investigación se adoptó Kanban [3], debido a la flexibilidad que ofrece para gestionar tareas de exploración, especialmente en una fase que requería investigar, entender y validar. Este marco otorgó la agilidad de incorporar y priorizar nuevos temas a medida que iban surgiendo, con la ventaja de poder atacarlos de inmediato. La prioridad fue la profundidad de la investigación y la validación constante, esto permitió validar requerimientos, despejar algunas dudas técnicas, y contar con una base sólida antes de comprometer el esfuerzo de desarrollo definitivo.

4.3.2. Metodología en etapa de desarrollo

Posteriormente, para la etapa de desarrollo, se implementó Scrum [4], debido a su estructura orientada a la entrega frecuente de valor. Este marco de trabajo permitió partir de un *backlog* priorizado, manteniendo la flexibilidad para incorporar ajustes y mejoras a medida que surgían necesidades. Esta agilidad resultó esencial debido a la complejidad del dominio de los cargadores eléctricos, el desafío técnico de las tecnologías a implementar y el control constante que requería el alcance del proyecto. Scrum fue fundamental para habilitar un ciclo de *feedback* recurrente, permitiendo que el cliente validará el progreso de forma incremental y asegurando que el producto final se alineará con las necesidades del negocio.

4.4. Ciclo de vida del proyecto

Para el ciclo de vida del proyecto se adoptó un enfoque iterativo e incremental [5]. La etapa de investigación permitió reducir la incertidumbre y establecer una base inicial sobre la cual se fueron construyendo los siguientes incrementos del sistema. El

desarrollo se dividió en sprints, en cada uno de los cuales se incorporan incrementos de valor al sistema, obteniendo resultados incrementales en plazos relativamente cortos, facilitando la retroalimentación temprana y la aplicación de ajustes necesarios.

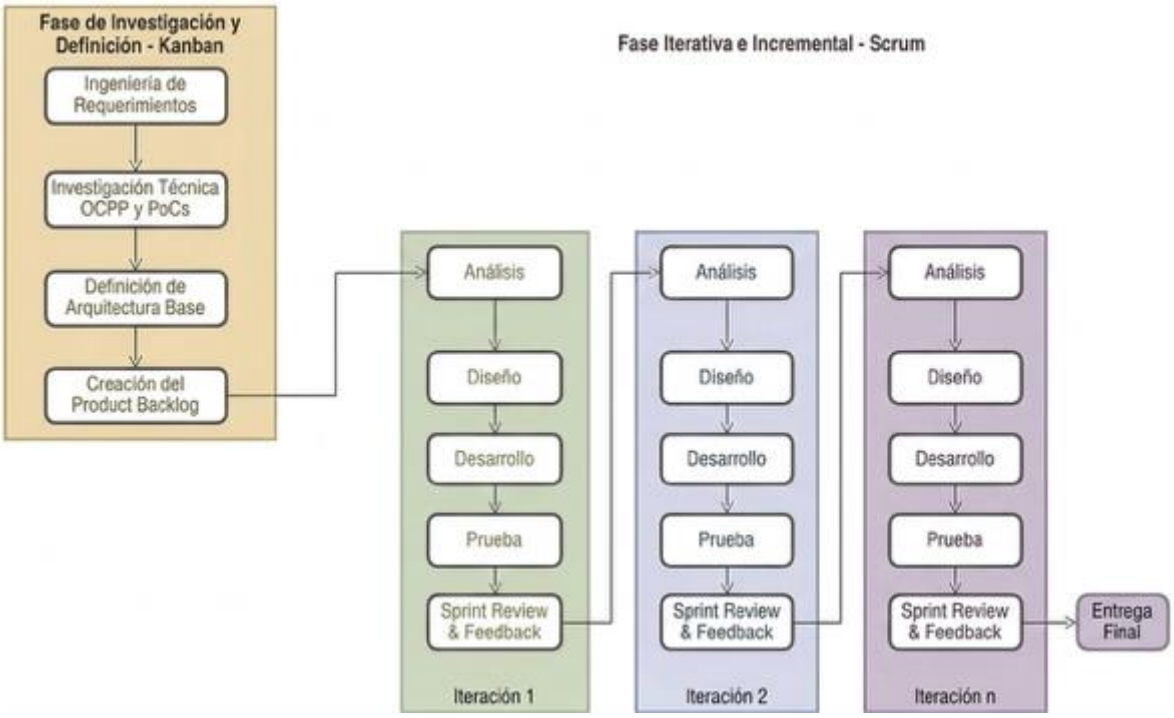


Figura 1: Ciclo de vida del proyecto

5. Ingeniería de requerimientos

En esta sección se describen los aspectos relevantes en cuanto a ingeniería de requerimientos analizando el proceso que se realizó, como también los requerimientos funcionales y no funcionales obtenidos. Estos resultados fueron la guía para asegurar el agregado de valor y que la solución satisfaga las necesidades de E-Mobility Solutions.

5.1. Proceso

El proceso de ingeniería de requerimientos seguido en este proyecto se alineó con las prácticas descritas en el estándar ISO/IEC/IEEE 29148 [6], abarcando actividades de relevamiento, especificación, validación y verificación de los requisitos.

Este proceso tuvo una gran incidencia en la etapa inicial de investigación. Esta etapa, previamente introducida en la sección Marco metodológico, se enfocó en reducir las incertidumbres del proyecto y dejar una base de requerimientos para luego iterar. Sin embargo, el proceso de ingeniería de requerimientos tomó también un rol durante la etapa de desarrollo. Esto se abordará luego en la Gestión del proyecto.

5.1.1. Relevamiento

Se presentarán las distintas técnicas de relevamiento de requerimientos utilizadas en el proyecto. El equipo se basó en el artículo Framework for Matching Requirements Elicitation Techniques to Project Characteristics que sugiere que las técnicas pueden caracterizarse según el criterio: cerrado-abierto [7].

La clasificación de las técnicas de relevamiento según el criterio cerrado-abierto permitió al equipo seleccionar herramientas adecuadas al contexto del proyecto.

5.1.1.1. Técnicas abiertas

Las técnicas abiertas fueron clave para comprender el dominio y detectar necesidades no evidentes.

5.1.1.1.1. Entrevista semiestructurada

Se realizó una entrevista semiestructurada (evidencia en la sección 12.2 del Anexo) con Matías Crosa, ex jefe de operaciones de E-Mobility Solutions. Esta instancia permitió obtener una visión general del funcionamiento operativo del negocio, comprender los principales problemas del sistema actual y conocer las necesidades reales tanto de operadores como de usuarios. El formato semiestructurado permitió guiar la conversación sobre temas clave, manteniendo flexibilidad para profundizar en aspectos relevantes que surgieron durante el intercambio.

5.1.1.1.2. Observación directa

El equipo visitó las oficinas de E-Mobility Solutions (evidencia en la sección 12.3 del Anexo) y presenció el flujo real de carga utilizando un vehículo eléctrico. Durante esta actividad se observó la interacción entre usuario, cargador y sistema, identificando tiempos de espera, estados del cargador y pasos necesarios para iniciar una sesión de carga. Esta experiencia permitió bajar a tierra la interacción con los cargadores, detectar limitaciones de usabilidad y oportunidades de mejora en el flujo.

5.1.1.1.3. Brainstorming

Se realizaron sesiones internas de análisis colaborativo dentro del equipo para interpretar la información relevada y transformarla en requerimientos concretos. Durante estas instancias se discutieron alternativas de solución, se evaluaron riesgos y se priorizaron funcionalidades. El enfoque grupal permitió integrar distintas perspectivas y generar propuestas más completas y consistentes para el prototipo.

5.1.1.2. Técnicas cerradas

Las técnicas cerradas permitieron estructurar y formalizar los requerimientos con mayor precisión técnica.

5.1.1.2.1. Análisis de soluciones existentes (Ingeniería inversa)

Se utilizó el sistema actual de la empresa para analizar su funcionamiento desde la perspectiva del usuario y del operador (evidencia en la sección 12.4 del Anexo). A partir de esta exploración se identificaron las funcionalidades claves a mantener, como también

restricciones y problemas a evitar. Este proceso permitió agilizar la definición al no tener que partir desde cero, aprovechando el conocimiento ya validado en la operación real.

5.1.1.2.2. Análisis de documentación

Se estudió y realizó un análisis, el cual se puede encontrar en la sección 12.5 del Anexo, de la documentación técnica del protocolo de comunicación utilizado para la comunicación entre los cargadores y sistemas centrales de gestión, el protocolo OCPP. El objetivo fue comprender cómo se comunican los puntos de carga con el sistema central, las restricciones de este protocolo y sus implicancias técnicas.

5.1.2. Especificación

Una vez completado el relevamiento, los requerimientos identificados fueron organizados y documentados de manera estructurada en base a historias de usuario. Esta es una forma ágil de expresar un requerimiento funcional desde la perspectiva del usuario, enfocándose en el valor que obtiene y no en la solución técnica [8] . Las historias de usuario luego fueron priorizadas y agrupadas en base a épicas (verticales importantes del dominio de negocio). Mediante la sección 12.6 del Anexo se puede visualizar un ejemplo claro del formato y contenido.

La especificación permitió transformar la información obtenida en funcionalidades concretas y condiciones claras que el sistema debía cumplir. También se establecieron requerimientos no funcionales que el sistema debe seguir y restricciones técnicas asociadas al dominio de la movilidad eléctrica y al protocolo de comunicación utilizado.

5.1.2.1. Actores del sistema

Como parte de la especificación de requerimientos, se identificaron los actores que interactúan con el sistema, con el objetivo de contextualizar las historias de usuario y delimitar responsabilidades dentro de la solución propuesta.

5.1.2.1.1. Empleado de E-Mobility Solutions

El empleado de E-Mobility Solutions es un usuario interno de la organización responsable de la operación, administración y soporte de la plataforma de gestión de cargadores

eléctricos. Utiliza el Panel Administrativo para operar y gestionar la red de carga. Se trata de un usuario con conocimientos operativos del sistema y nociones generales sobre el dominio de la movilidad eléctrica.

5.1.2.1.2. Usuario final del cargador

Persona física o jurídica que utiliza la red de carga para cargar su vehículo eléctrico mediante la aplicación móvil o RFID. Se trata de un usuario final sin conocimientos técnicos específicos, cuyo objetivo principal es cargar su vehículo de manera simple y eficiente.

5.1.3. Validación

La validación tuvo como objetivo asegurar que los requerimientos definidos reflejaran adecuadamente las necesidades de E-Mobility Solutions y el contexto operativo observado. Este proceso se llevó a cabo de forma continua a lo largo del proyecto.

Para esta etapa, el equipo mantuvo reuniones e instancias asíncronas con referentes de E-Mobility Solutions, en las que se revisaron los requerimientos y su prioridad, a fin de asegurar la alineación con las expectativas del cliente. Estas instancias resultaron especialmente importantes debido al peso que tienen el alcance y su gestión en este proyecto.

Una vez que el cliente validaba cada historia de usuario, ésta se incorporaba al *backlog* para ser abordada en una iteración futura.

5.1.3.1. Validación técnica

Parte de la validación también consistió en asegurar la viabilidad técnica de los requerimientos relacionados con la comunicación mediante el protocolo OCPP. Tras la investigación inicial, se realizó una prueba de concepto (PoC) de la comunicación entre un servidor y un cargador, cuyo proceso detallado se presenta en la sección 12.5.2 del Anexo. Esta PoC fue un hito fundamental para confirmar la factibilidad de la comunicación con los cargadores y brindar confianza para avanzar con la solución propuesta.

5.1.4. Verificación

Para la verificación de los requerimientos, el equipo se basó en los criterios definidos por el IEEE [6], revisando sistemáticamente cada requisito para asegurar su calidad. Se evaluó que los requerimientos fueran:

- Correctos y completos, sin ambigüedades ni contradicciones.
- Consistentes, tanto entre sí como con el resto de la documentación del proyecto.
- Claros, comprensibles para todos los interesados.
- Factibles, desde el punto de vista técnico y en relación con los plazos establecidos.
- Verificables, es decir, susceptibles de ser comprobados mediante pruebas o criterios objetivos.
- Rastreables, permitiendo vincular cada requisito con sus entregables correspondientes.
- Priorizados, según su importancia y valor para el proyecto.

5.2. Requerimientos funcionales

El proceso analizado dio como resultado los siguientes requerimientos funcionales que reflejan las necesidades clave del proyecto. Estos se dividieron según la arista del producto en la que agregan valor, sea el panel administrativo o la aplicación móvil. A continuación, se presenta la tabla consolidada de requerimientos funcionales para cada una. Los requerimientos que quedaron fuera del alcance de la solución se detallan en la sección 12.16 del Anexo.

5.2.1. Requerimientos funcionales del panel administrativo

ID	Descripción (User Story)	Épica	Prioridad

1	Como empleado de E-Mobility Solutions quiero agregar un nuevo cargador al sistema para gestionar su información	Gestión de Cargadores	Alta
2	Como empleado de E-Mobility Solutions quiero editar la información de un cargador existente para que quede al dia	Gestión de Cargadores	Alta
3	Como empleado de E-Mobility Solutions quiero ver el detalle de un cargador específico	Gestión de Cargadores	Alta
4	Como empleado de E-Mobility Solutions quiero configurar el precio para un cargador así cobrar el monto correspondiente por su uso	Gestión de Cargadores	Alta
5	Como empleado de E-Mobility Solutions quiero ver la auditoría de Logs OCPP del cargador para poder depurar posibles problemas	Gestión de Cargadores	Media
6	Como empleado de E-Mobility Solutions quiero ajustar las configuraciones OCPP del cargador así tener control sobre el comportamiento del cargador	Gestión de Cargadores	Media
7	Como empleado de E-Mobility Solutions quiero poner un cargador en mantenimiento para evitar su uso	Gestión de Cargadores	Baja
8	Como empleado de E-Mobility Solutions quiero ver el listado de Cargadores	Gestión de Cargadores	Alta
9	Como empleado de E-Mobility Solutions quiero visualizar cargadores en un mapa	Gestión de Cargadores	Baja

10	Como empleado de E-Mobility Solutions quiero archivar un cargador para evitar que siga apareciendo en el sistema	Gestión de Cargadores	Media
11	Como empleado de E-Mobility Solutions quiero visualizar el estado físico actual de los conectores de un cargador así tener visibilidad sobre la red eléctrica de carga	Gestión de Cargadores	Alta
12	Como empleado de E-Mobility Solutions quiero pingear el cargador para asegurar tener el estado consistente en el sistema	Gestión de Cargadores	Media
13	Como empleado de E-Mobility Solutions quiero ver las cargas de un cargador así monitorear su uso	Gestión de Cargadores	Baja
14	Como empleado de E-Mobility Solutions quiero exportar un reporte de los cargadores del sistema para realizar tareas operativas	Gestión de Cargadores	Baja
15	Como empleado de E-Mobility Solutions quiero ver el listado de todas las cargas del sistema	Gestión de Cargas	Media
16	Como empleado de E-Mobility Solutions quiero ver información detallada de una carga	Gestión de Cargas	Alta
17	Como empleado de E-Mobility Solutions quiero poder empezar una carga para un conector así poder dar soporte técnico	Gestión de Cargas	Media

18	Como empleado de E-Mobility Solutions quiero poder finalizar una carga en progreso así poder dar soporte técnico	Gestión de Cargas	Media
19	Como empleado de E-Mobility Solutions quiero exportar un reporte de las cargas del sistema para realizar tareas operativas	Gestión de Cargas	Baja
20	Como empleado de E-Mobility Solutions quiero invitar un nuevo usuario al sistema	Gestión de Usuarios	Alta
21	Como empleado de E-Mobility Solutions quiero ver los usuarios del sistema	Gestión de Usuarios	Alta
22	Como empleado de E-Mobility Solutions quiero editar la información de un usuario para tener la información actualizada	Gestión de Usuarios	Alta
23	Como empleado de E-Mobility Solutions quiero ver información detallada de un usuario	Gestión de Usuarios	Alta
24	Como empleado de E-Mobility Solutions quiero archivar un usuario para evitar que continúe usando el sistema	Gestión de Usuarios	Media
25	Como empleado de E-Mobility Solutions quiero ver estadísticas de un usuario para comprender el uso que tiene de la plataforma	Gestión de Usuarios	Baja
26	Como empleado de E-Mobility Solutions quiero ver las organizaciones de un usuario para visualizar sus permisos	Gestión de Usuarios	Baja
27	Como empleado de E-Mobility Solutions	Gestión de	Baja

	quiero ver las cargas de un usuario para entender su historial	Usuarios	
28	Como empleado de E-Mobility Solutions quiero ver los RFIDs de un usuario	Gestión de Usuarios	Baja
29	Como empleado de E-Mobility Solutions quiero crear una nueva organización para gestionar un nuevo cliente	Gestión de Organizaciones	Alta
30	Como empleado de E-Mobility Solutions quiero ver los detalles de una organización	Gestión de Organizaciones	Alta
31	Como empleado de E-Mobility Solutions quiero editar una organización para tener la información actualizada	Gestión de Organizaciones	Alta
32	Como empleado de E-Mobility Solutions quiero archivar una organización	Gestión de Organizaciones	Media
33	Como empleado de E-Mobility Solutions quiero asignar usuarios a una organización así darles los accesos correspondientes	Gestión de Organizaciones	Alta
34	Como empleado de E-Mobility Solutions quiero desasignar usuarios de una organización para evitar usos indebidos	Gestión de Organizaciones	Alta
35	Como empleado de E-Mobility Solutions quiero ver los usuarios de una organización	Gestión de Organizaciones	Alta
36	Como empleado de E-Mobility Solutions quiero ver las estadísticas de la organización para entender el impacto que tiene en el	Gestión de Organizaciones	Baja

	sistema		
37	Como empleado de E-Mobility Solutions quiero ver los cargadores de la organización	Gestión de Organizaciones	Baja
38	Como empleado de E-Mobility Solutions quiero agregar un RFID para permitir su uso en la plataforma	Gestión de RFIDs	Media
39	Como empleado de E-Mobility Solutions quiero asignar un RFID a un usuario	Gestión de RFIDs	Media
40	Como empleado de E-Mobility Solutions quiero bloquear un RFID para evitar su uso	Gestión de RFIDs	Baja
41	Como empleado de E-Mobility Solutions quiero desasignar un RFID	Gestión de RFIDs	Media
42	Como empleado de E-Mobility Solutions quiero ver el listado de RFIDs	Gestión de RFIDs	Media
43	Como empleado de E-Mobility Solutions quiero desbloquear un RFID	Gestión de RFIDs	Baja
44	Como empleado de E-Mobility Solutions quiero ver todos los pagos realizados en el sistema para tener trazabilidad de las transacciones	Gestión de Pagos	Baja
45	Como empleado de E-Mobility Solutions quiero exportar un reporte de los pagos realizados en el sistema para realizar tareas operativas	Gestión de Pagos	Baja

46	Como empleado de E-Mobility Solutions quiero permitir a usuarios autenticados entrar a la plataforma	Gestión de Perfil	Alta
47	Como empleado de E-Mobility Solutions quiero poder cambiar mi contraseña asi poder recuperar acceso a la plataforma	Gestión de Perfil	Media

5.2.2. Requerimientos Funcionales de la Aplicación Móvil

ID	Descripción (User Story)	Épica	Prioridad
48	Como usuario del cargador quiero visualizar cargadores en un mapa para entender qué puntos de carga tengo cerca	Gestión de Cargadores	Alta
49	Como usuario del cargador quiero ver la información de un conector para saber si me sirve para cargar mi vehículo	Gestión de Cargadores	Alta
50	Como usuario del cargador quiero escanear un QR para ver la información del conector	Gestión de Cargadores	Alta
51	Como usuario del cargador quiero empezar una carga para así poder tener batería en mi vehículo	Gestión de Cargas	Alta
52	Como usuario del cargador quiero visualizar el estado de una carga en progreso para poder estar informado	Gestión de Cargas	Alta

53	Como usuario del cargador quiero finalizar una carga para poder irme con mi vehículo	Gestión de Cargas	Alta
54	Como usuario del cargador quiero ver mi historial de cargas para visibilidad sobre mi uso de la plataforma	Gestión de Cargas	Media
55	Como usuario del cargador quiero realizar una carga con RFID	Gestión de Cargas	Alta
56	Como usuario del cargador quiero ver mis RFIDs asignadas para tener control sobre mis tarjetas	Gestión de RFIDs	Media
57	Como usuario del cargador quiero canjear un RFID para tener un nuevo medio de acceso	Gestión de RFIDs	Media
58	Como usuario del cargador quiero borrar un RFID para cancelar un medio de acceso	Gestión de RFIDs	Media
59	Como usuario del cargador quiero agregar Métodos de Pago para facilitar el proceso de carga	Gestión de Pagos	Media
60	Como usuario del cargador quiero borrar un Método de Pago	Gestión de Pagos	Media
61	Como usuario del cargador quiero ver mis Métodos de Pago	Gestión de Pagos	Media
62	Como usuario del cargador quiero ver mis deudas para poder entender su causa	Gestión de Pagos	Alta
63	Como usuario del cargador quiero pagar mis deudas para volver a poder usar el sistema	Gestión de Pagos	Alta

64	Como usuario del cargador quiero ver la información de mi Perfil	Gestión de Perfil	Baja
65	Como usuario del cargador quiero editar la información de mi Perfil para que quede consistente	Gestión de Perfil	Baja
66	Como usuario del cargador quiero autenticarme para usar el sistema para usar la plataforma	Gestión de Perfil	Alta
67	Como usuario del cargador quiero registrarme para así poder usar la red de carga	Gestión de Perfil	Alta
68	Como usuario del cargador quiero cambiar mi contraseña para recuperar el acceso al sistema	Gestión de Perfil	Media

5.3. Requerimientos no funcionales

Los requerimientos no funcionales definen las características de calidad que debe cumplir el sistema, más allá de las funcionalidades que ofrece. A continuación, se detallan los requerimientos no funcionales que se identificaron para este prototipo.

5.3.1. RNF 1 - Mantenibilidad del código

El código fuente del sistema deberá estar estructurado de forma clara, modular y siguiendo buenas prácticas de desarrollo, con el fin de facilitar su comprensión, mantenimiento y crecimiento.

5.3.2. RNF 2 - Documentación técnica completa

El sistema deberá contar con documentación técnica que describa su arquitectura, estructura de datos, endpoints utilizados y funcionamiento general.

5.3.3. RNF 3 - Usabilidad e interfaz intuitiva

La interfaz de usuario deberá ser clara, simple e intuitiva, permitiendo a los usuarios comprender rápidamente el estado de los cargadores y realizar una carga.

5.3.4. RNF 4 - Compatibilidad con OCPP 1.6

El sistema deberá ser compatible con la versión 1.6 del protocolo OCPP, permitiendo interpretar correctamente los mensajes enviados por los cargadores.

5.3.5. RNF 5 - Compatibilidad móvil

La aplicación móvil deberá estar disponible y ser funcional en dispositivos Android 15 e iOS 18.

5.3.6. RNF 6 - Tolerancia a fallas de conectividad

El sistema deberá ser capaz de manejar interrupciones en la conexión con los cargadores y recuperarse de manera controlada cuando la comunicación sea restablecida.

5.3.7. RNF 7 - Sistema disponible

Tanto la aplicación móvil y el panel administrativo tiene que estar siempre disponibles para poder llevar a cabo los flujos críticos del sistema

5.3.8. RNF 8 - Tiempo de inicio de carga

El sistema debe permitir que los usuarios inicien una sesión de carga en un tiempo no mayor a 10 segundos desde la confirmación de la acción de inicio.

5.3.9. RNF 9 - Tiempo de actualización del estado de los cargadores

El sistema debe reflejar el estado actual de los cargadores registrados en un tiempo menor a 30 segundos desde que se produce un cambio en dichos estados.

5.3.10. RNF 10 - Seguridad en los pagos

El sistema debe garantizar la confidencialidad, integridad y protección de los datos de pago de los usuarios durante su transmisión y almacenamiento.

5.3.11. RNF 11 - Observabilidad del sistema

El sistema debe proporcionar mecanismos de monitoreo y registro que permitan disponer de visibilidad sobre su comportamiento, facilitando la identificación de fallas y el análisis de su causa.

5.4. Conclusiones y lecciones aprendidas

La Ingeniería de Requerimientos permitió ordenar las ideas iniciales y transformar necesidades generales en definiciones concretas para el prototipo. A través de entrevistas, observación y análisis técnico, se logró comprender mejor el funcionamiento del entorno de E-Mobility Solutions y traducirlo en requerimientos claros, alcanzables y sobre todo que agreguen valor.

Como principal aprendizaje, se destaca la importancia de dedicar tiempo a entender el problema antes de comenzar a desarrollarlo. Trabajar en un dominio poco conocido demostró que relevar y validar información de manera temprana evita retrabajos y mejora la coherencia del sistema. También se confirmó que combinar distintas técnicas de relevamiento enriquece la perspectiva del equipo y contribuye a una solución más sólida y alineada con la realidad.

6. Arquitectura y diseño

En esta sección se presentan los principales aspectos de la arquitectura del sistema, destacando las decisiones que guiaron su diseño. Como se anticipó en secciones previas, la parte técnica, la arquitectura y diseño, es una de las principales complejidades de este proyecto, por ende requiere un análisis detallado. Se abordan los atributos de calidad considerados, las restricciones que condicionaron la solución, las principales decisiones de diseño adoptadas, las tecnologías seleccionadas y los flujos críticos que estructuran el funcionamiento del sistema.

6.1. Visión general de la arquitectura

La arquitectura del sistema fue el resultado de un proceso iterativo que realizó el equipo, a partir del análisis y toma de decisiones, en el cual se evaluaron diversas alternativas y sus implicancias sobre los atributos de calidad priorizados. Dicho proceso implicó múltiples instancias de revisión, ajuste y validación con referentes, como Gastón Mousques.

EasyCharge se compone por múltiples componentes que interactúan entre sí para gestionar la red de cargadores eléctricos. Incluye servicios *backend*, aplicación móvil, aplicación web, integraciones con servicios como pasarelas de pago y con los mismos cargadores. A su vez, la arquitectura se desarrolla en el contexto de un sistema distribuido, donde los distintos componentes se ejecutan de forma independiente y se comunican a través de la red.

Esta configuración introduce desafíos como la presencia de múltiples puntos de falla, latencias de comunicación, coordinación de servicios y dispositivos que operan en entornos heterogéneos bajo distintas condiciones de conectividad.

Como ya se anticipó en la sección Solución, el sistema se presenta en esta etapa como un prototipo funcional listo para una etapa de validación operativa. Es un primer

incremento funcional para el futuro sistema productivo, por lo que requiere una arquitectura mantenible y preparada para su continuidad.

6.2. Atributos de calidad

La arquitectura del sistema se definió partiendo de los requerimientos no funcionales que surgieron en la etapa de Ingeniería de requerimientos. Dichos requerimientos fueron analizados y asociados a atributos de calidad, con el objetivo de identificar su impacto en las decisiones de diseño y guiar la construcción de la solución.

Para estructurar este análisis, el equipo tomó como referencia el modelo de calidad de producto definido en la norma ISO/IEC 25010 [9], que establece un conjunto de características y subcaracterísticas para evaluar la calidad del *software*. Esta referencia permitió utilizar un marco conceptual ampliamente aceptado en la disciplina para categorizar y analizar los requerimientos de calidad del sistema.

La siguiente tabla remarca la asociación de los RNF con sus respectivos atributos de calidad y su correspondiente prioridad.

RNF	**Atributo de Calidad**	**Prioridad**
RNF 1 - Mantenibilidad del Código	Mantenibilidad	Alta
RNF 2 - Documentación Técnica Completa	Mantenibilidad	Alta
RNF 3 - Usabilidad e Interfaz Intuitiva	Usabilidad	Media
RNF 4 - Compatibilidad con OCPP 1.6	Integrabilidad	Media

RNF 5 - Compatibilidad Móvil	Portabilidad	Media
RNF 6 - Tolerancia a Fallas de Conectividad	Disponibilidad	Alta
RNF 7 - Sistema Disponible	Disponibilidad	Alta
RNF 8 – Tiempo de inicio de carga	Rendimiento	Baja
RNF 9 - Tiempo de actualización del estado de los cargadores	Rendimiento	Baja
RNF 10 – Seguridad en los pagos	Seguridad	Baja
RNF 11 – Observabilidad del sistema	Observabilidad	Baja

Para los atributos de calidad prioritarios, el equipo definió escenarios de calidad, siguiendo el enfoque metodológico propuesto en Software Architecture in Practice [10]. Estos escenarios describen cómo el sistema debería responder ante determinados estímulos en un contexto específico. De esta forma, los escenarios constituyen una base empírica y verificable para evaluar si la arquitectura diseñada satisface efectivamente los atributos de calidad priorizados, reduciendo ambigüedades y facilitando la validación de las decisiones tomadas.

6.2.1. Mantenibilidad

Este atributo se posiciona como el más importante para el sistema. Tal como se detalló en la sección Problema, esta priorización consiste en la necesidad primordial que E-

Mobility Solutions tiene de superar las limitaciones de su sistema actual mediante una plataforma que permita la evolución continua y la adaptación a cambios. Una alta mantenibilidad es, por lo tanto, un factor estratégico esencial.

El carácter de la solución como un proyecto académico y su enfoque de implementación como un prototipo, requiere que la arquitectura diseñada priorice la facilidad de comprensión y modificación. Esto es crucial para facilitar la transferencia de conocimiento y permitir que un equipo futuro pueda asumir las tareas de desarrollo y soporte sin afrontar deuda técnica, costos innecesarios de aprendizaje, o hasta el posible riesgo de regresión.

Los siguientes escenarios de calidad guían las decisiones de diseño y se establecen como métricas de éxito en el desarrollo:

6.2.1.1. E1: Cobertura de componentes de flujos críticos

Fuente	**Desarrollador**
Estímulo	Se modifica un componente asociado a un flujo crítico del sistema
Artefacto	Módulo de lógica de negocio crítico
Entorno	Durante el desarrollo
Respuesta	El sistema ejecuta automáticamente la suite de pruebas unitarias asociadas a dicho módulo
Medida	El cambio debe validarse con una cobertura superior al 95% en los componentes que implementan la lógica de negocio del flujo crítico y sin introducir fallos en las pruebas existentes.

La principal barrera para la modificación en un sistema complejo es el riesgo de introducir defectos en funcionalidades ya existentes. Exigir un 95% de cobertura en los componentes que implementan la lógica de negocio de los flujos críticos reduce significativamente este riesgo, ya que permite detectar rápidamente comportamientos inesperados ante modificaciones. De esta manera, cualquier desarrollador puede realizar cambios, ya sea para corregir errores o extender funcionalidades, con la confianza de

que la suite de pruebas automatizadas validará el comportamiento esperado de los componentes críticos del sistema.

6.2.1.2. E2: Control de la complejidad ciclomática

Fuente	Desarrollador
Estímulo	Se implementa o modifica una función del sistema
Artefacto	Modulo de Negocio
Entorno	Durante el análisis
Respuesta	Se ejecuta análisis estático de código
Medida	Ninguna función del flujo crítico del sistema debe superar una complejidad ciclomática de 10

La complejidad ciclomática mide el número de caminos de ejecución linealmente independientes a través del código fuente. McCabe destaca en su ensayo A Complexity Measure [11], ensayo que introduce el concepto de complejidad ciclomática, que un valor menor a 10 es una medida que refleja legibilidad y simplicidad del código. Al mantener la complejidad baja, se reduce la carga cognitiva sobre los futuros desarrolladores, haciendo que las funciones sean más fáciles de comprender, modificar y probar de forma unitaria. Esto en combinación con una alta cobertura en flujos críticos juega un rol fundamental para asegurar la mantenibilidad.

6.2.2. Disponibilidad

El sistema desarrollado es un sistema crítico. Su correcto funcionamiento es esencial para la operativa diaria de E-Mobility Solutions. Una indisponibilidad del sistema podría impedir la gestión de cargadores, la autorización de sesiones o el monitoreo de estados. Asimismo, desde la perspectiva del usuario final, la imposibilidad de iniciar una sesión de carga puede impactar directamente en el día de un usuario, por ejemplo, impidiéndole cargar su auto para ir a su trabajo.

En este contexto, se definieron los siguientes escenarios de calidad asociados al atributo de disponibilidad:

6.2.2.1. E3: Disponibilidad de la operativa crítica

Fuente	Usuario o cargador
Estímulo	Se solicita iniciar una sesión de carga o consultar el estado de un cargador
Artefacto	Sistema central
Entorno	Operación normal
Respuesta	El sistema procesa correctamente la solicitud
Medida	Estos flujos deben estar disponibles al menos el 99,9% del tiempo anual.

Asegurar una disponibilidad del 99,9% en los flujos de inicio de carga y en la visualización del estado de los cargadores implica un máximo de 43 minutos de indisponibilidad por mes, 8 horas 45 minutos en un año. Este margen de disponibilidad está alineado con las expectativas del cliente y la operativa de la empresa para el alcance de este prototipo.

6.2.2.2. E4: Recuperación ante fallos de conexión

Fuente	Falla de conectividad de red.
Estímulo	Recuperación tras pérdida de comunicación entre cargador y sistema central
Artefacto	Base de datos
Entorno	Operación normal
Respuesta	Restablecimiento automático de la conexión y sincronización de mensajes pendientes.
Medida	Garantizar la eventual consistencia de los datos asociados a la sesión de carga en menos de 30 segundos

Dado que se trata de un sistema distribuido, con cargadores desplegados en distintas ubicaciones y bajo condiciones de red heterogéneas, es fundamental contemplar mecanismos que permitan manejar desconexiones transitorias sin comprometer la integridad de la información.

6.2.3. Interoperabilidad

El objetivo funcional de la solución es la gestión y operación de una red distribuida de carga compuesta por cargadores de distintos fabricantes. En este contexto, la interoperabilidad constituye un atributo de calidad fundamental, ya que operar con múltiples marcas de cargadores resulta clave tanto para el correcto funcionamiento de la red actual como para su crecimiento futuro. Este atributo fue evaluado por medio de:

6.2.3.1. E5: Cobertura de mensajes de transactions y status del protocolo OCPP

Fuente	Cargador compatible con OCPP 1.6.
Estímulo	Envío de mensajes relacionados con transacciones o notificaciones de estado.
Artefacto	Sistema central.
Entorno	Operación normal
Respuesta	El sistema valida, procesa y responde a los mensajes conforme a la especificación del protocolo OCPP adoptada.
Medida	Soporte del 100% de los mensajes de transacciones y estado. Estos deben procesarse sin errores de protocolo ni inconsistencias en el estado interno del sistema.

6.2.3.2. E6: Integración multi-marca

Fuente	Cargador compatible con OCPP 1.6.
Estímulo	El cargador ejecuta los flujos críticos del sistema utilizando su propia implementación del protocolo OCPP.

Artefacto	Sistema central
Entorno	Operación normal
Respuesta	El sistema procesa correctamente todos los mensajes intercambiados
Medida	El 100% de los flujos críticos deben ejecutarse exitosamente con al menos tres marcas distintas de cargadores

Garantizar esta cobertura cubre al sistema para poder gestionar el flujo crítico de carga de cualquier cargador que respete el protocolo.

6.2.4. Portabilidad

La portabilidad constituye un atributo de calidad relevante para la solución, dado que los usuarios finales utilizan distintos sistemas operativos móviles. En Uruguay, existe un uso significativo tanto de dispositivos Android como iOS [12], por lo que limitar el acceso a la plataforma a un único sistema operativo implicaría restringir notoriamente el alcance del servicio.

Desde el punto de vista de la arquitectura, esto implica diseñar la aplicación móvil minimizando dependencias específicas de cada sistema operativo y buscando un comportamiento consistente en ambos entornos.

6.2.4.1. E7: Compatibilidad multiplataforma

Fuente	Usuario final.
Estímulo	Ejecuta la aplicación móvil en un dispositivo con sistema operativo iOS 18 o superior, o Android 15 o superior.
Artefacto	Aplicación móvil.
Entorno	Operación normal
Respuesta	La aplicación permite ejecutar correctamente todos los flujos críticos del sistema

Medida	El 100% de los flujos críticos se completan sin fallas funcionales ni errores bloqueantes en dispositivos con dichas versiones de sistema operativo.

6.2.5. Usabilidad

En el contexto de esta solución, la usabilidad de la aplicación móvil adquiere un rol central, ya que el usuario interactúa con el sistema en situaciones sensibles y muchas veces condicionadas por el entorno físico.

Dado que los flujos de inicio y finalización de carga son críticos al momento de uso, la aplicación debe minimizar fricciones, reducir la cantidad de pasos necesarios y evitar ambigüedades en la interacción. En busca de esto se establece el siguiente escenario de calidad:

6.2.5.1. E8: Navegabilidad y ejecución de flujos críticos sin asistencia

Fuente	Usuario nuevo sin experiencia previa con el sistema.
Estímulo	Intenta realizar una carga
Artefacto	Interfaz de usuario de la aplicación móvil.
Entorno	Pruebas de usabilidad controladas.
Respuesta	El sistema permite completar la tarea sin necesidad de asistencia externa.
Medida	Los participantes completan correctamente cada tarea crítica sin asistencia.

Una experiencia poco clara en este punto puede generar frustración en el usuario y afectar negativamente la percepción del servicio.

6.2.6. Atributos de calidad no prioritarios

Se vieron presentes otros atributos de calidad que impactan en la solución pero que, debido a la naturaleza de prototipo del proyecto, fueron considerados de menor prioridad

en esta etapa. Entre ellos se encuentran el rendimiento, la escalabilidad, despliegue, observabilidad y seguridad.

Esto no implica que hayan sido descartados durante el diseño y desarrollo de la solución. Estos fueron considerados de manera general, tomando participación en toma de decisiones pero con menor influencia frente a otros atributos críticos. El análisis y evaluación de dichas dimensiones quedan fuera del alcance de este proyecto y forman parte de las validaciones operativas pendientes a realizar previo a una salida a producción.

6.3. Restricciones

Es importante analizar las restricciones que condicionan el desarrollo del sistema y su resultante arquitectura. La única restricción que se presentó por parte del cliente fue el uso de Mercado Pago[4] como pasarela de pagos. El resto de las restricciones que se enfrentó el equipo recaen en el contexto académico en el cual se sitúa el proyecto, previamente adelantado en la sección Marco metodológico.

El proyecto cuenta con una fecha de entrega establecida para el 12 de marzo de 2026, lo que impone un marco temporal fijo para el diseño, desarrollo y validación de la solución.

El equipo está conformado de manera fija por cuatro integrantes, sin posibilidad de incorporar nuevos recursos ni ampliar mucho la dedicación horaria. Esto condiciona el alcance del proyecto y exige una adecuada priorización de funcionalidades y decisiones arquitectónicas acordes a la capacidad disponible.

6.4. Detalle de la arquitectura

La descripción de la arquitectura se apoya en el modelo Views and Beyond [13], desarrollado por el Software Engineering Institute de la Carnegie Mellon University. Este

[4] https://www.mercadopago.com.uy

modelo propone documentar la arquitectura mediante un conjunto de vistas complementarias, cada una enfocada en distintos intereses, lo que permite comunicar y analizar de manera sistemática las decisiones arquitectónicas del sistema.

6.4.1. Vista componentes y conectores

La vista de componentes y conectores ofrece una representación clara y concisa de la estructura interna del sistema, mostrando los principales componentes y los mecanismos de comunicación entre ellos. Esta vista permite comprender cómo se distribuyen las responsabilidades, cómo interactúan las distintas partes del sistema y su impacto en los atributos de calidad.

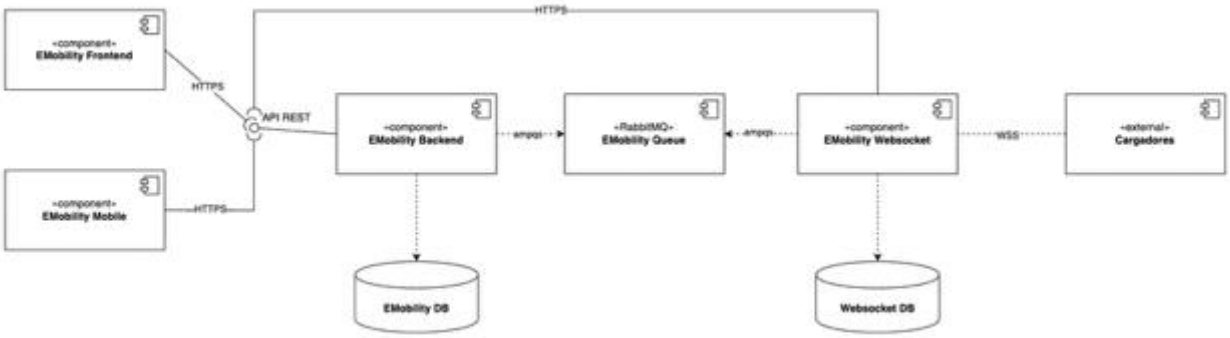


Figura 2: Vista de componentes y conectores

Catálogo de elementos:

Componente/Conector	**Responsabilidad**
E-Mobility Backend	Monolito de NestJS[5] que expone una API Rest con la lógica de negocio del sistema.
E-Mobility Mobile	Aplicación móvil iOS/Android utilizada por usuarios

[5] https://nestjs.com

	que quieren cargar su auto eléctrico. Está desarrollada en React Native[6] y se comunica con el *backend* mediante el protocolo HTTPs.
E-Mobility Frontend	Aplicación web que expone el panel administrativo donde los empleados de E-Mobility Solutions pueden gestionar la red de cargadores. Está desarrollada en React[7] y se comunica con el *backend* mediante el protocolo HTTPs.
E-Mobility DB	Base de datos relacional (PostgreSQL[8]) que almacena los datos del sistema.
E-Mobility Websocket	Servicio desarrollado en NodeJs[9] encargado de gestionar la conexión de *websockets* con los cargadores.
Websocket DB	Base de datos clave/valor (Redis[10]) utilizada para almacenar datos de las conexiones de los cargadores.
E-Mobility Queue	*Broker* de RabbitMQ[11] que gestiona la comunicación asíncrona entre los servicios de E-Mobility Backend y E-Mobility Websockets

6 https://reactnative.dev
7 https://react.dev
8 https://www.postgresql.org
9 https://nodejs.org
10 https://redis.io
11 https://www.rabbitmq.com

E-Mobility Storage	Almacenamiento externo que contiene imágenes de cargadores para mostrar en la app
Cargadores	Dispositivos que los usuarios utilizan para cargar sus autos.

6.4.1.1. Decisiones de Diseño

Servicio de Websockets:

Como se observa en el diagrama, una de las principales decisiones arquitectónicas fue designar un servicio dedicado a la gestión de comunicación de *websockets*: E-Mobility Websocket.

Inicialmente el E-Mobility Backend fue planteado como el encargado tanto de la lógica de negocio como también de la gestión de conexiones con los cargadores. La comunicación mediante *websockets*, a diferencia de una API Rest, tiene la particularidad de ser conexiones *stateful*. Esto tiene varias implicancias. Por un lado, supone un consumo intensivo de recursos constante que crecerá a medida que aumente la cantidad de cargadores conectados, lo cual tiene un efecto en el rendimiento del servicio y su API. Por otro lado, también limita la naturaleza descartable del servicio, ya que una caída del mismo provocaría la pérdida de todas las conexiones con los cargadores impactando en la disponibilidad del servicio.

Teniendo esto en consideración, se optó por separar la responsabilidad del manejo de conexiones con los cargadores a un servicio independiente. Este servicio fue diseñado para ser robusto y de baja frecuencia de cambios, mientras que el E-Mobility Backend mantiene un ciclo de evolución más dinámico. Cabe destacar que esta decisión refleja un aumento de costos y complejidad del sistema al tener que gestionar más infraestructura y la comunicación entre los procesos. Sin embargo, mejora la disponibilidad del sistema al aislar los posibles puntos de falla asociados en la lógica de

negocio, y permite además un uso más eficiente de los recursos de cada servicio, favoreciendo el rendimiento y la escalabilidad de la arquitectura.

Monolito vs. Microservicios:

La decisión de separar el servicio E-Mobility Websocket del E-Mobility Backend llevó al equipo a evaluar si correspondía continuar segmentando el monolito en servicios adicionales. Se consideró la implementación de una arquitectura de microservicios, dividiendo dominios específicos en servicios independientes con el objetivo de mejorar la escalabilidad, la performance, la disponibilidad y la flexibilidad en los despliegues.

No obstante, se concluyó que el continuar con el incremento de complejidad y costo no se justificaba en el contexto y la escala actual del proyecto, por lo que se optó permanecer con un monolito. La elección de un monolito no descarta una futura evolución hacia microservicios, sino que se alinea con la evolución incremental de la arquitectura, priorizando simplicidad y mantenibilidad en las primeras etapas del sistema.

Comunicación Asincrónica:

Como se destacó previamente, el separar los websockets a un servicio independiente incrementa la complejidad de la solución y da lugar a un nuevo dilema: la comunicación entre los procesos del E-Mobility Websocket y el E-Mobility Backend. En la Figura 2, se visualiza la decisión de implementar una comunicación asíncrona bidireccional mediante un *broker* de cola de mensajes. Este flujo de comunicación se ve más en detalle al analizar los Flujos críticos del sistema.

La decisión tomada se basó en la importancia de la disponibilidad en este sistema crítico, la cual es favorecida por medio de la tolerancia a fallos y persistencia que RabbitMQ brinda. Como contrapartida, este enfoque introduce un *trade-off* en términos de *performance*, ya que la comunicación asíncrona a través de un *broker* agrega latencia adicional en comparación con una comunicación directa síncrona. No obstante, se priorizó la disponibilidad y resiliencia del sistema por sobre la latencia mínima, dado el carácter crítico de la solución.

Base de Datos Relacional:

Mediante el diagrama se visualiza la interacción entre el E-Mobility Backend y una base de datos relacional. La elección de una base de datos relacional se fundamentó en la necesidad de almacenar datos altamente estructurados, con múltiples relaciones y restricciones de integridad entre las distintas entidades del dominio. Detalles sobre estas relaciones y entidades pueden verse en la sección 12.8 del Anexo. Este contexto favorece su uso, ya que permite modelar de forma explícita dichas relaciones, garantizar la consistencia de los datos mediante mecanismos transaccionales y facilitar consultas complejas a través de un lenguaje declarativo (SQL).

Base de Datos No Relacional:

En la sección Flujos críticos se ve la persistencia con alta frecuencia de datos crudos para observabilidad, como *logs* de OCPP y métricas de carga. Se evaluó la posibilidad de separar estos tipos de datos, en una base de datos no relacional independiente, con el objetivo de reducir la carga sobre la base de datos principal y mejorar el rendimiento general del sistema. Si bien esta alternativa presenta ventajas en términos de escalabilidad y rendimiento, se concluyó que la complejidad adicional asociada a la incorporación y mantenimiento de una nueva fuente de almacenamiento no se justificaba en función de la demanda esperada del sistema y del contexto del proyecto. En consecuencia, se optó por mantener una única base de datos relacional, priorizando la simplicidad operativa y dejando abierta la futura posibilidad del cambio.

Almacenamiento Externo para Conexión de Cargadores:

En cuanto a almacenamiento de datos, se observa como el Websocket Backend presenta su propia instancia de persistencia, independiente a la base de datos relacional. La comunicación con los cargadores no solo requiere mantener activa la conexión *websocket*, sino también preservar el contexto asociado a dicha comunicación. Este contexto incluye información como el estado de la sesión e identificadores del cargador.

Esta información se almacena en el componente Websocket DB, cuyo principal objetivo es desacoplar el estado de la comunicación del proceso que gestiona las conexiones. De esta forma, se favorece la disponibilidad del sistema, ya que ante una caída del servicio de *websockets* o una interrupción en la conexión con los cargadores, el contexto

previamente almacenado permite restablecer la comunicación sin pérdida de información relevante.

Dada la naturaleza simple, agnósticos al negocio y altamente volátil de estos datos, se optó por utilizar un almacenamiento independiente de tipo no relacional clave/valor. Esta elección permite accesos rápidos y eficientes, minimizando la latencia en operaciones de lectura y escritura, y contribuyendo positivamente al rendimiento del servicio de *websockets* y por ende la comunicación con cargadores.

Confianza en los cargadores:

Los cargadores eléctricos resultaron ser un componente fundamental dentro de la arquitectura del sistema. Cuentan con capacidad de cómputo y almacenamiento local, lo que les permite mantener estados internos, ejecutar lógica propia y operar temporalmente de forma desacoplada del sistema central. Esta característica, junto con los requisitos de OCPP, contribuye de manera crucial a la disponibilidad y tolerancia a fallos de la solución global, ya que ante interrupciones de conectividad pueden continuar ciertas operaciones y deben sincronizar su estado una vez restablecida la comunicación. Si bien esto favorece la disponibilidad, es importante destacar el *trade off* que presenta en cuanto a la consistencia inmediata de la información del sistema.

Los cargadores deben ser considerados nodos activos dentro de un sistema distribuido, con responsabilidades propias y con un impacto directo en atributos de calidad como disponibilidad y consistencia del sistema.

6.4.2. Vista de módulos

La vista de módulos ofrece una representación clara y estructurada de la organización estática del sistema, mostrando la descomposición del *software* en módulos y la distribución de responsabilidades a nivel de código. Esta vista permite comprender cómo se agrupan las funcionalidades, cómo se establecen las dependencias entre módulos y cómo estas decisiones influyen en atributos de calidad como la mantenibilidad.

6.4.2.1. Emobility Backend

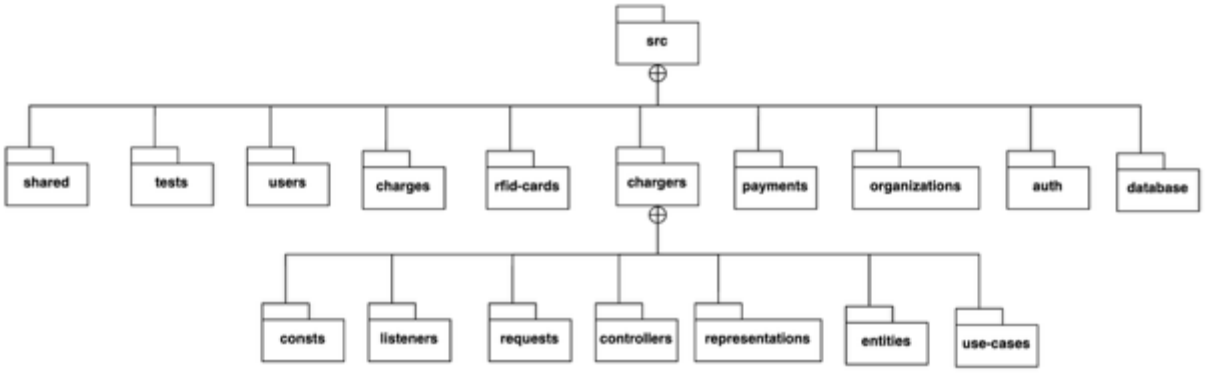


Figura 3: Vista de Módulos Backend

Catálogo de elementos:

Módulo	**Responsabilidad**
users	Gestión de usuarios del sistema. Incluye operaciones CRUD, invitaciones a la plataforma, administración del perfil del usuario y gestión de las organizaciones y cargadores asociados.
charges	Administración de sesiones de carga. Incluye el inicio y finalización de cargas, generación de estadísticas, exportación de datos y asociación de los cobros correspondientes a cada sesión.
consts	Definición de constantes del protocolo OCPP, como estados de conectores, códigos de error y mapeos utilizados en los mensajes.

listeners	Componentes que escuchan eventos externos (RabbitMQ/OCPP) y delegan su procesamiento a los casos de uso.
requests	DTOs utilizados para validar y mapear los datos de entrada de los endpoints.
controllers	Controladores HTTP que exponen los *endpoints* REST del módulo de cargas.
representations	Objetos encargados de transformar entidades en respuestas JSON para la API.
entities	Definición de las entidades del modelo de datos utilizadas por TypeORM[12].
use-cases	Implementación de la lógica de negocio asociada a la gestión de cargas.
rfid-cards	Administración de tarjetas RFID utilizadas para autorizar sesiones de carga.
chargers	Gestión de cargadores eléctricos.
payments	Gestión de pagos integrados con Mercado Pago.
organizations	Gestión de organizaciones y sus miembros.
auth	Gestión de autenticación y registro de usuarios.

[12] https://typeorm.io

database	Configuración de persistencia de datos.
tests	Conjunto de pruebas automatizadas del sistema.
shared	Servicios compartidos utilizados por distintos módulos del sistema.

6.4.2.1.1. Decisiones de diseño

Domain Driven Design:

Como se analizó previamente, el E-Mobility Backend es un monolito que contiene toda la lógica de negocio del sistema. Con el objetivo de mejorar la mantenibilidad de este servicio con tantas responsabilidades, se tomaron como referencia principios de *Domain-Driven Design* (DDD) [14] para organizar su estructura. Se definieron módulos que encapsulan los dominios de negocio específicos, agrupando sus componentes y responsabilidades. Esto permitió mantener una buena cohesión dentro de cada módulo y reducir el acoplamiento entre ellos, favoreciendo una separación clara de responsabilidades. Estos módulos se ven reflejados en el diagrama: *ChargersModule*, *UsersModule*, *ChargesModule*, etc.

Esta organización reduce la complejidad general del servicio y facilita su evolución frente a futuros cambios, como el pasaje a una arquitectura de microservicios. Además, la separación por módulos mejora la testabilidad y facilita su cobertura ya que permite probar la lógica de negocio de forma más aislada.

Arquitectura Hexagonal:

En el diseño del E-Mobility Backend también se adoptaron principios alineados con la arquitectura hexagonal [15] (también conocida como *Ports and Adapters*), con el objetivo de desacoplar la lógica de negocio de los detalles de infraestructura y facilitar la evolución del sistema. Si bien no se implementó el patrón de forma estricta, varios de sus principios

fueron aplicados de manera consistente y dieron frutos para la mantenibilidad del sistema.

Los módulos, definidos por medio de DDD, presentan una estructura interna homogénea, compuesta por carpetas como *controllers/*, *use-cases/*, *entities/*, *representations/* y *requests/*. Esta organización permite distinguir claramente la capa de presentación (controllers), la capa de aplicación (casos de uso) y los elementos del dominio (entities), reduciendo el acoplamiento entre responsabilidades y facilitando la comprensión del sistema.

Los casos de uso constituyen el núcleo de la capa de aplicación. Se componen de clases como RegisterAction o CreateChargerAction que encapsulan la lógica de negocio específica asociada a una operación concreta del sistema. Esto favorece el principio de *Single Responsibility* de SOLID, una lista de principios presentados por Robert C. Martin y altamente consideradas en la industria [16]. Este enfoque permite centralizar las reglas del dominio en unidades atómicas, bien definidas, reutilizables y fácilmente testeables, evitando que la lógica quede dispersa en controladores o servicios de infraestructura y favoreciendo así los escenarios de calidad planteados para la mantenibilidad.

Se aplica consistentemente el principio SOLID de inversión de dependencias mediante el sistema de inyección de dependencias provisto por NestJS. Los casos de uso no dependen directamente de implementaciones concretas, sino de abstracciones, como repositorios o servicios externos que son inyectados en tiempo de ejecución.

Los servicios de infraestructura se ubican en la carpeta *shared/services/*, manteniendo separada la lógica de negocio de los detalles técnicos de integración. Esto permite reemplazar o modificar proveedores externos con impacto controlado sobre el resto del sistema.

Abstracción de capa de datos:

Para la capa de acceso a datos se decidió utilizar un *Object-Relational Mapping* (ORM)[17]. El uso de esta abstracción desacopla la lógica de negocio de los detalles específicos de persistencia. Los casos de uso de no tiene dependencia directa con la

capa de datos. La adopción de un ORM favorece la mantenibilidad, ya que facilita la integración con la capa de datos, permite desacoplarse de detalles de implementación y mejora la legibilidad del código. Detalles de su implementación pueden verse en la sección 12.8 del Anexo.

6.4.2.2. Admin UI & mobile app

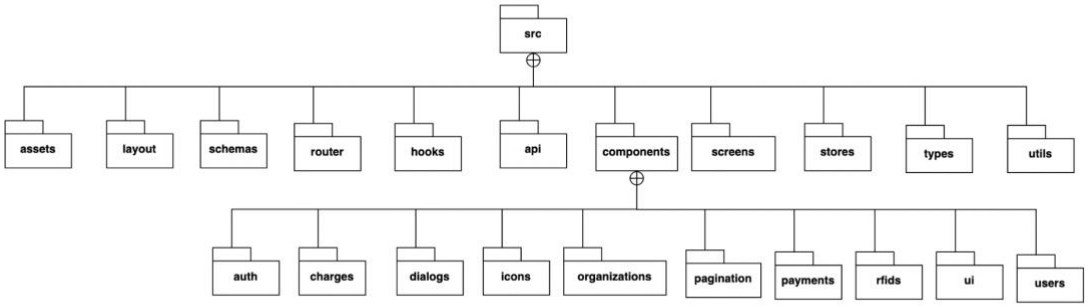


Figura 4: Vista de Módulos Admin UI

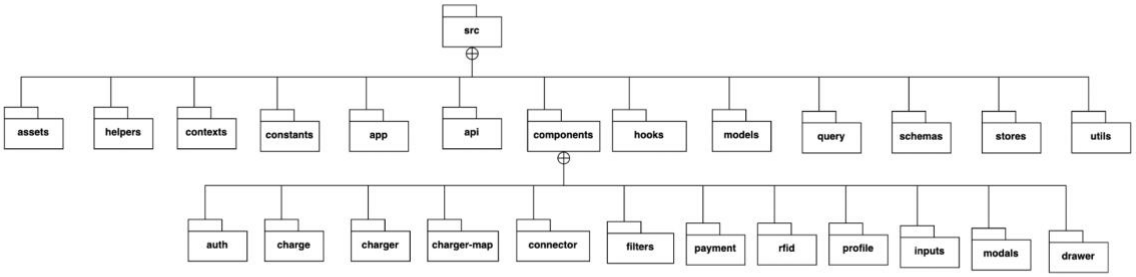


Figura 5: Vista de Módulos Mobile App

6.4.2.2.1. Decisiones de diseño

Componentización de la Interfaz

En los diagramas ejemplificados se visualiza la presencia de directorios */components*. Para el desarrollo de la aplicación móvil y web se adoptó un enfoque basado en componentes reutilizables para la construcción de las interfaces. Esta estrategia permite dividir la UI en pequeñas piezas, autocontenidas y con responsabilidades claras, reduciendo la complejidad del código y facilitando su mantenimiento.

La reutilización de componentes comunes, como botones, tarjetas, encabezados y tablas, disminuye la duplicación de lógica y estilos, lo que simplifica futuras tareas que impliquen modificaciones o nuevas implementaciones.

Además, el uso consistente de componentes a lo largo de toda la aplicación favorece una estructura visual uniforme. Esta coherencia mejora la experiencia del usuario y facilita el aprendizaje de la interfaz, ya que mantiene patrones de interacción previsibles en los distintos flujos del sistema.

6.4.3. Vista de despliegue

La vista de despliegue tiene como objetivo mostrar cómo los componentes del sistema se distribuyen sobre la infraestructura donde se ejecutan. Permite entender cómo el sistema opera en producción y analizar el impacto de las decisiones de infraestructura.

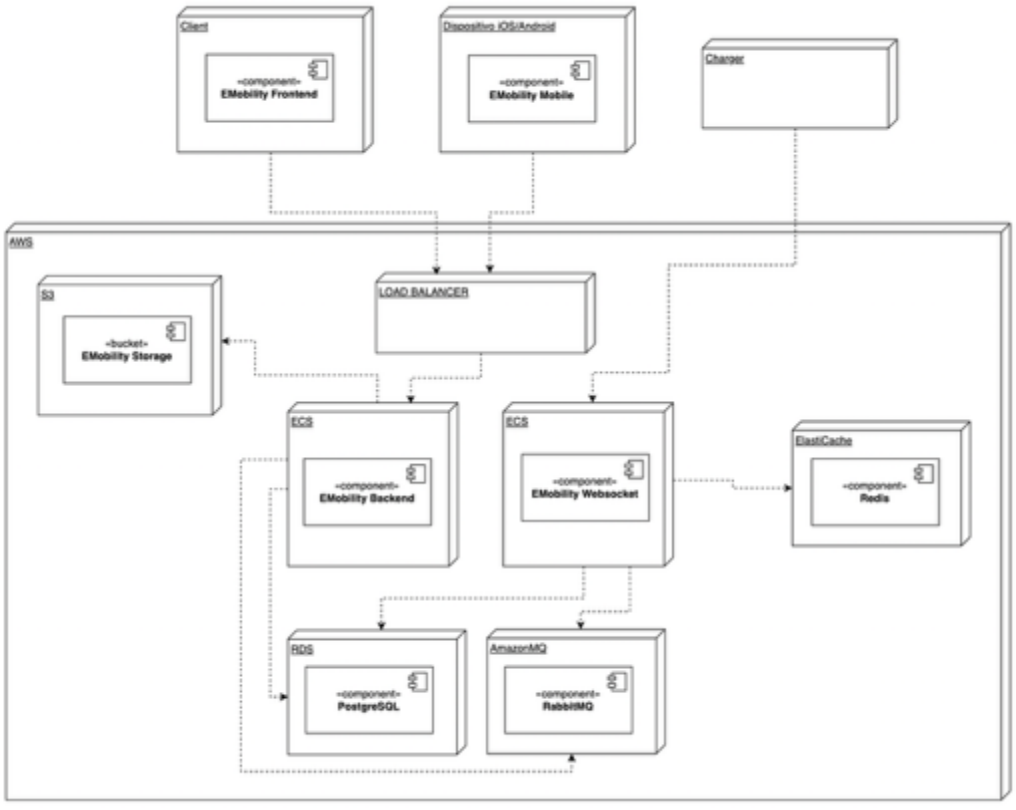


Figura 6: Vista de despliegue

Catálogo de elementos

Se puede visualizar como se usaron diversos servicios de AWS[13] para gestionar el sistema:

Servicio	**Propósito**
AWS (Amazon Web Services)	Nube pública donde se aloja toda la infraestructura del sistema.
Client	Dispositivo del usuario donde se renderiza el EMobility Frontend
Dispositivo iOS/Android	Dispositivo que permite la interacción del usuario final en tiempo real.
Charger	Representa el cargador físico (hardware) que envía y recibe datos en tiempo real.
Load Balancer	Distribuye automáticamente el tráfico entrante de la web y móvil entre múltiples instancias para mejorar la disponibilidad.
ECS (Elastic Container Service)	Servicio administrado de AWS que administra los contenedores de los servicios de EMobility Backend y EMobility Websockets.
RDS (Relational Database Service)	Servicio administrado para la base de datos relacional
S3	Almacenamiento de objetos escalable, utilizado para guardar archivos, logs o documentos externos.
ElastiCache (Redis)	Gestiona el almacenamiento en memoria (caché)
Amazon MQ (RabbitMQ)	Broker de mensajería que facilita la comunicación asíncrona y el intercambio de eventos entre servicios.

[13] https://aws.amazon.com

6.4.3.1. Decisiones de diseño

Cloud Hosting:

El sistema fue desplegado en un entorno de infraestructura en la nube con el objetivo de validar su comportamiento en condiciones similares a un entorno productivo real. Dado que la solución gestiona una red distribuida de cargadores eléctricos que se comunican con un sistema central a través de internet, resultaba fundamental probar la arquitectura en un entorno accesible externamente, con condiciones reales de latencia, conectividad y disponibilidad.

Los proveedores de infraestructura en la nube contribuyen directamente a la disponibilidad. La elección de AWS como proveedor garantiza una disponibilidad del 99,99% en los servicios seleccionados. Ofrecen entornos redundantes y acuerdos de nivel de servicio (SLA) [18] que reducen el riesgo de interrupciones prolongadas. Al mismo tiempo, mediante el uso de *health checks*, el entorno *cloud* permite redesplegar o reemplazar los servicios *backend* de manera rápida ante fallos, reduciendo el tiempo de recuperación y asegurando la disponibilidad.

Si bien se identificaron factores críticos para el entorno de producción, los siguientes elementos se mantuvieron fuera del alcance del prototipo:

- Escalabilidad a gran escala
- Redundancia multi-zona.
- Mecanismos avanzados de seguridad.
- Estrategias de despliegue (*blue-green*).

Estas consideraciones y otras mejoras de la infraestructura del sistema se ven detalladas en la sección 12.15 del Anexo.

6.5. Tecnologías

En esta sección se describen las tecnologías seleccionadas para acompañar las decisiones arquitectónicas adoptadas.

6.5.1. Lenguajes

El cliente no impuso restricciones respecto al lenguaje de programación a utilizar en la construcción de la solución. En este contexto, y con el objetivo de favorecer la futura adopción y continuidad del sistema por parte de nuevos desarrolladores, se evaluaron tecnologías con alta adopción en la industria y proyección a largo plazo.

Se optó por utilizar TypeScript. Según el informe State of Developer Ecosystem de JetBrains 2025 [19], TypeScript lidera el *ranking* de lenguajes con más proyección para los próximos años y actualmente se presenta como el principal lenguaje de programación del 22% de los desarrolladores.

Al incorporar tipado estático y contratos explícitos entre componentes, permite detectar errores en tiempo de compilación y mejorar la claridad del código, contribuyendo a la mantenibilidad y reduciendo defectos.

Adicionalmente, la elección de TypeScript permitió unificar el lenguaje a lo largo de todo el *stack* tecnológico. Esta homogeneidad disminuye la curva de aprendizaje para nuevos integrantes, simplifica el intercambio de conocimiento y favorece la coherencia conceptual entre *frontend* y *backend*, contribuyendo a la evolución futura del sistema.

6.5.2. Frameworks y librerías

Los *frameworks* y librerías fueron seleccionados en función de su contribución a los atributos de calidad definidos y su madurez tecnológica. Su amplia adopción en la industria reduce el riesgo técnico, aumenta la base de desarrolladores capaces de mantenerlo y favorece la estabilidad en producción. La decisión se respaldó en reportes como el JetBrains Developer Ecosystem Report [19] y el State of JS [20].

Express:

Express[14] fue el *framework* elegido para el desarrollo del servicio E-Mobility WebSocket. Su enfoque minimalista permite implementar únicamente los componentes necesarios sin sobrecarga estructural, lo que se asociaba de gran manera con la robustez y simplicidad que se buscaba para este servicio.

NestJS:

NestJS fue el *framework* elegido para el desarrollo del E-Mobility Backend. Dado que se optó por una arquitectura monolítica con múltiples responsabilidades, resultaba fundamental contar con una herramienta que promoviera una estructura clara y organizada. En este sentido, NestJS, a diferencia de Express, proporciona una base sólida que favorece la mantenibilidad. Su enfoque modular facilita la aplicación de prácticas de *Domain-Driven Design*.

React:

React fue la librería seleccionada para el desarrollo del *frontend* web. Lidera consistentemente los rankings de adopción y uso en implementaciones web. La alta adopción de la librería en la industria implica una alta disponibilidad de recursos y paquetes que facilitan la integración con servicios externos. Su popularidad y el hecho que el equipo ya tiene experiencia con esta tecnología fueron factores decisivos para su elección sobre otras opciones como Angular[15] o VueJs[16].

React Native:

React Native fue la tecnología elegida para el desarrollo de la aplicación móvil. La principal motivación fue la necesidad de garantizar portabilidad, permitiendo mantener una única base de código para los sistemas operativos de iOS y Android. Esto evita la duplicación de esfuerzos y simplifica el mantenimiento del sistema. La elección se realizó por sobre otras opciones como Flutter[17], considerando que el equipo ya contaba con experiencia previa en React. Este conocimiento previo permitió reducir la curva de

[14] https://expressjs.com
[15] https://angular.dev
[16] https://vuejs.org/
[17] https://flutter.dev

aprendizaje, acelerar el desarrollo inicial, alineándose con los objetivos de tiempo del proyecto.

Postgres:

PostgreSQL fue seleccionado como motor para la E-Mobility DB por su robustez, estabilidad y amplio soporte de funcionalidades avanzadas. Su cumplimiento de estándares SQL, capacidad para manejar consultas complejas y posibilidad de extensión lo convierten en una solución flexible y confiable, adecuada para sostener la evolución del sistema a largo plazo.

Redis:

Redis fue seleccionado como mecanismo de almacenamiento clave valor para la gestión de datos asociados a las conexiones activas, *Websocket DB*. Redis ofrece un buen equilibrio entre velocidad, y simplicidad, lo que lo hace especialmente adecuado para manejar datos temporales como estados de conexión de los cargadores.

RabbitMQ:

Se optó por RabbitMQ como broker para la E-Mobility Queue debido a que ofrece un modelo claro basado en colas y exchanges que resulta adecuado para los requerimientos del sistema, donde la prioridad es la confiabilidad en la entrega y disponibilidad más que el procesamiento masivo de eventos.

Docker:

La solución se desarrolló utilizando Docker encapsulando en contenedores los distintos servicios junto con sus dependencias. Docker fue seleccionado como tecnología de contenedorización debido a su capacidad para garantizar entornos consistentes y reproducibles a lo largo de todo el ciclo de vida del sistema.

Esta portabilidad contribuye directamente al despliegue de la solución. Al mismo tiempo facilita la experiencia de los desarrolladores, simplificando el levantamiento del proyecto, el trabajo del día a día y la incorporación de nuevos desarrolladores, factor clave para el equipo que mantenga esta solución a futuro.

AWS:

AWS fue seleccionado como el proveedor de infraestructura en la nube por la amplitud de su ecosistema de servicios, el cual cubre adecuadamente todas las tecnologías utilizadas en el sistema, permitiendo integrarlas dentro de una misma plataforma. Asimismo, el equipo contaba con experiencia previa en el uso de AWS. Sus capacidades de alta disponibilidad y redundancia contribuyen a garantizar la continuidad operativa del sistema, alineándose con los atributos de calidad definidos y los beneficios de usar infraestructura en la nube.

Mercado Pago:

Mercado Pago fue seleccionado como pasarela de pago por decisión del cliente. Desde el punto de vista técnico, la plataforma ofrece altos estándares de seguridad en el procesamiento de transacciones, reduciendo así el riesgo del sistema. Además, la disponibilidad de SDK oficiales facilitó una integración simple, segura y mantenible dentro de la solución.

Google Maps:

Google Maps[18] fue seleccionado como servicio de geolocalización para los cargadores principalmente por la calidad de su información geográfica, garantizando precisión en la visualización y ubicación de los puntos de carga. Su facilidad de integración con las tecnologías utilizadas mediante sdks oficiales, tanto en entornos web como iOS y Android, permitió una implementación simple, portable y eficiente.

6.6. Flujos críticos del sistema

En esta sección serán identificados y descompuestos los flujos críticos del sistema, entendidos como aquellos procesos cuyo correcto funcionamiento resulta esencial para el cumplimiento de los objetivos del producto. El análisis detallado de estos flujos permite comprender cómo interactúan los distintos componentes de la arquitectura y cómo se

[18] https://maps.google.com

materializan, en la práctica, los atributos de calidad definidos previamente. Las pantallas visuales que acompañan estos flujos se encuentran disponibles en la sección 12.7 del Anexo.

Asimismo, se examinará de qué manera las decisiones arquitectónicas tomadas, tanto las presentadas en las vistas anteriores como otras decisiones de diseño específicas, impactan en estos flujos, contribuyendo a satisfacer los escenarios de calidad planteados.

De este modo, se establece una conexión explícita entre la arquitectura definida y el comportamiento esperado del sistema en situaciones relevantes.

6.6.1. Flujo de conexión de cargador

Este flujo refiere al proceso de comunicación que se lleva a cabo para que un cargador se conecte al sistema y su estado quede consistente.

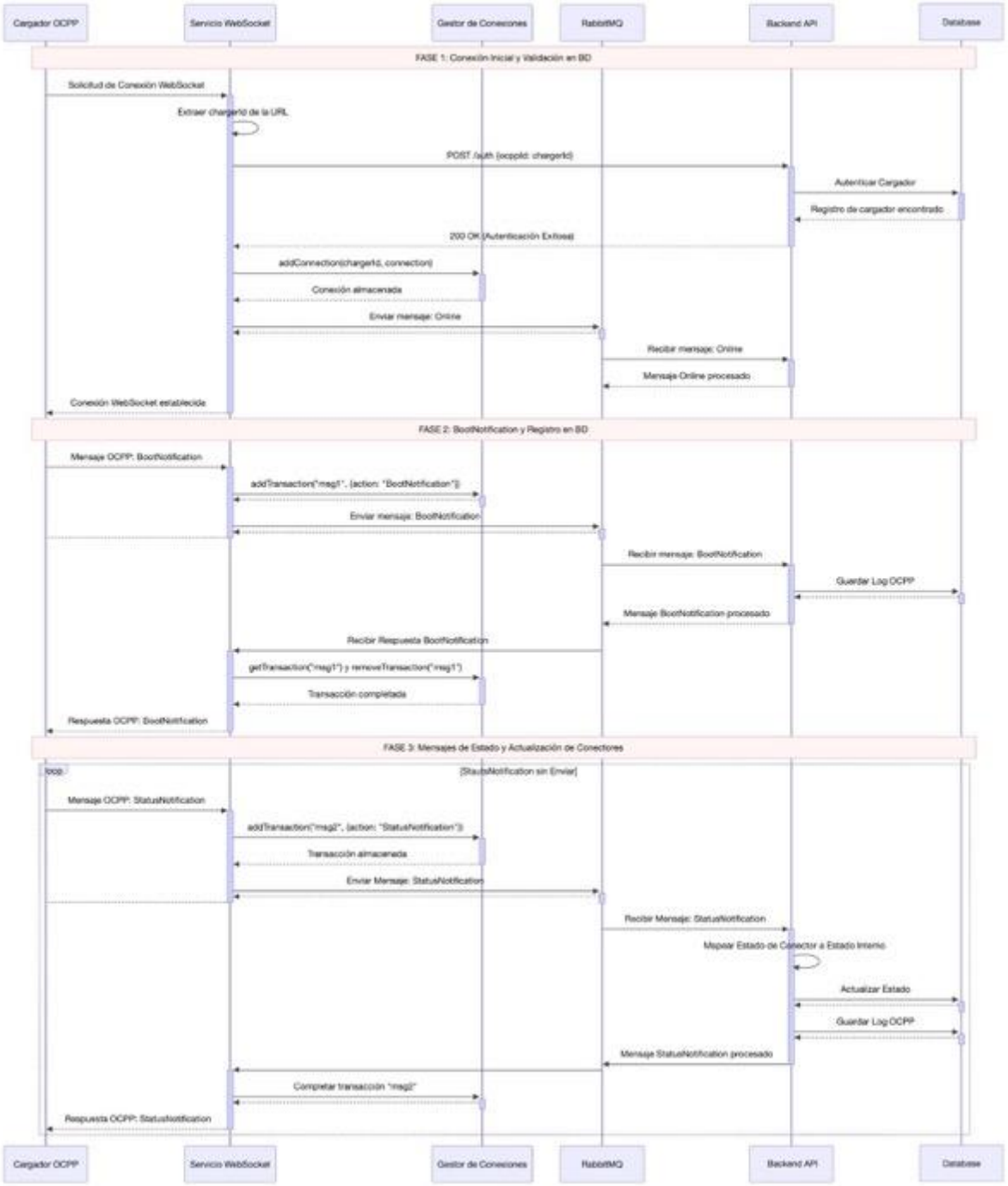


Figura 7: Diagrama de secuencia del flujo de conexión del cargador

Desde el inicio del flujo, se visualizan mecanismos de seguridad implementados, asegurando que únicamente cargadores autorizados puedan interactuar con el sistema y formar parte de esta red. Cabe destacar que a pesar de que la seguridad no fue unos de los atributos de calidad priorizados para este prototipo, si fue tomada en cuenta y se realizaron medidas para asegurar un cumpliemento mínimo y evitar riesgos. Los

cargadores no autorizados son rechazados y su conexión terminada. Esta es una excepción en la comunicación entre estos servicios, ya que se realiza de manera sincrónica mediante una API Rest así favorecer la performance y evitar *timeouts* en la conexión de los *sockets*.

Asimismo, el diagrama refleja la principal comunicación asincrónica mediante RabbitMQ. Ante eventuales fallas, los mensajes se encolan y procesan posteriormente, evitando la pérdida de información, asegurando la disponibilidad y eventual consistencia. Al estar encolados estos son enviados de manera oportuna cuando el servicio vuelve a estar disponible, favoreciendo así el escenario de calidad planteado para la disponibilidad.

En relación con el protocolo OCPP, se destaca la manipulación de los mensajes BootNotification y StatusNotification, fundamentales para mantener la consistencia de la información entre los cargadores y el sistema central. Cuando un cargador se conecta, envía su estado actualizado al servidor, notificando su disponibilidad y condiciones operativas.

Durante los períodos de desconexión, el dispositivo almacena localmente los mensajes generados y los transmite posteriormente en orden cronológico, priorizando aquellos de mayor relevancia, como BootNotification y StatusNotification, sobre otros de menor prioridad, tales como MeterValues.

Este mecanismo permite al sistema re-sincronizar el estado operativo de los cargadores una vez restablecida la conexión, tomando como fuente de verdad el cargador. Esto garantiza una consistencia eventual entre la información persistida en el sistema central y la realidad del dispositivo, asegurando que la plataforma refleje en todo momento la disponibilidad y condición actualizada de cada punto de carga.

El diagrama muestra claramente la persistencia de todos estos mensajes en la tabla ocpp_logs. Esta persistencia favorece la observabilidad del sistema y ayuda a los empleados de E-Mobility Solutions tener contexto del funcionamiento de la red. Como se destacó previamente, a futuro estos podrían ser separados a una base de datos no relacional para reducir la carga sobre la base de datos relacional.

Otro aspecto a resaltar es el manejo de la gestión de conexiones, se puede ver con más precisión el rol que tiene este almacenamiento externo. Se evidencia la importancia de su disponibilidad y persistencia al procesar mensajes, como también la importancia de su *performance* debido a la participación en toda comunicación.

6.6.2. Flujo de carga

El flujo de carga presenta un alto nivel de complejidad, ya que involucra la comunicación entre múltiples componentes y servicios de forma asíncrona y distribuida. En este proceso intervienen la aplicación móvil, la API *backend*, el servicio de *websockets*, el sistema de pagos, la base de datos y el cargador, cada uno con responsabilidades específicas y distintos tiempos de respuesta.

Debido a esta complejidad y con el objetivo de realizar un análisis claro que refleje adecuadamente las decisiones arquitectónicas adoptadas, el flujo fue dividido en múltiples subflujos. Esta descomposición permite aislar responsabilidades, entender mejor las interacciones entre componentes y analizar de forma detallada aspectos como seguridad, disponibilidad y consistencia.

De esta manera, cada subflujo puede estudiarse individualmente sin perder la visión integral del proceso completo de inicio a fin de una sesión de carga.

6.6.2.1. Flujo de solicitud de carga paga

Consiste en desde que el usuario solicita y paga la carga hasta que se visualiza la carga en progreso en la aplicación.

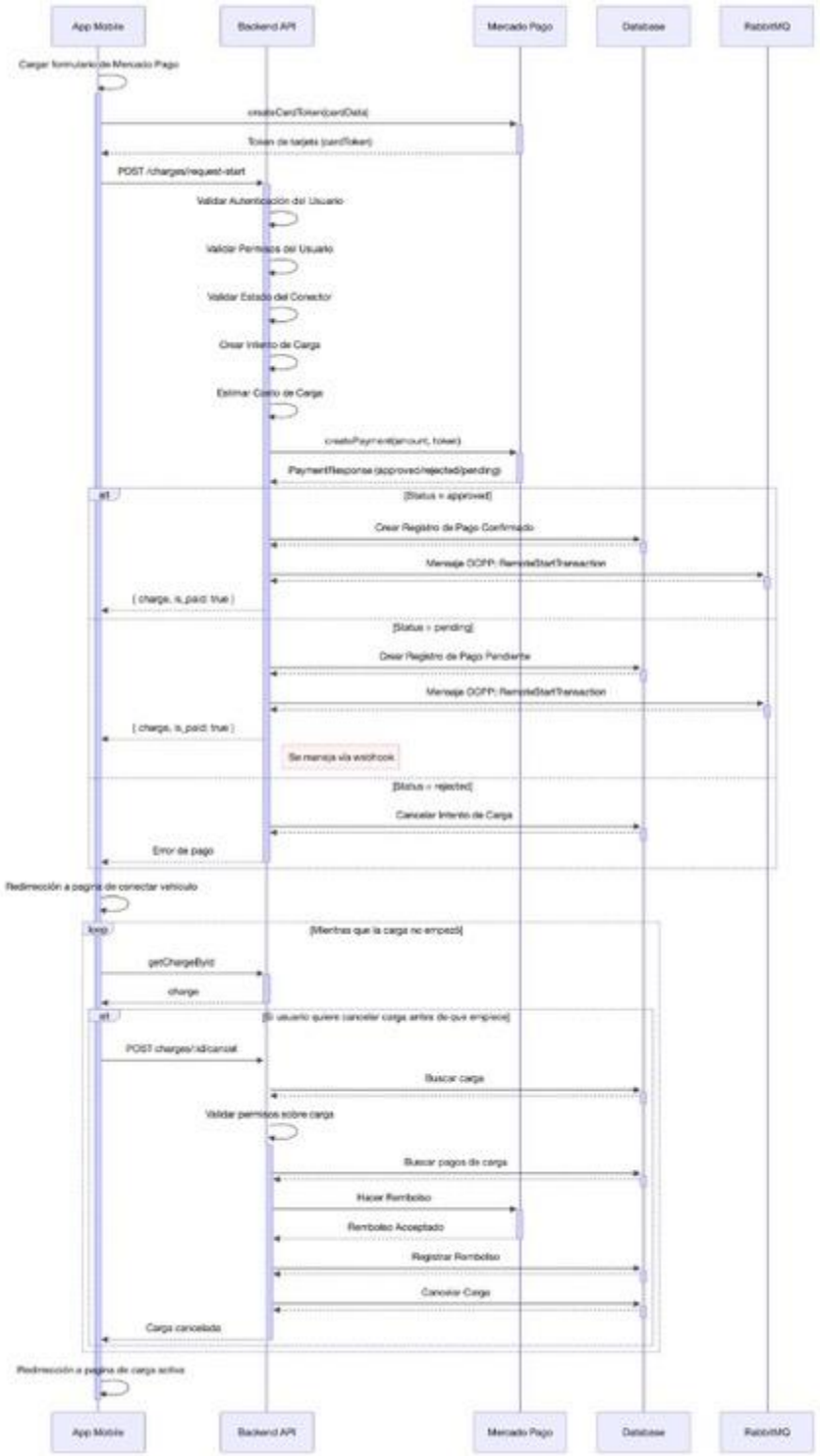


Figura 8: Diagrama de secuencia de flujo de solicitud carga paga

En primer lugar, nuevamente vemos el foco en la seguridad. Las *requests* a la API se autentican mediante el uso de JWT, lo que permite asegurar la identidad de los actores

en cada una de ellas. A nivel de pagos, el sistema no manipula ni almacena datos sensibles de tarjetas, sino que opera exclusivamente con un *payment token*, reduciendo riesgos de seguridad y asegurando cumplimiento normativo.

Tras las medidas de autenticación y autorización, el sistema instancia un intento de carga. Esto refiere a entidad persistida de carga pero que aún no ha sido confirmada por el cargador. El propósito de esta entidad parcial es la trazabilidad de la solicitud a lo largo del proceso asincrónico.

Una vez confirmado el intento de carga se procede con uno de los flujos más críticos, la gestión del pago. En busca de minimizar la generación de deudas dentro del sistema debido a cargas sin fondos, el flujo contempla una preautorización previo a empezar la carga basada en un costo estimado que queda como crédito a favor para el costo final. Esta estimación se realiza tomando en cuenta los costos fijos, variables y el consumo promedio de energía de una carga. Se realiza una transacción con Mercado Pago para confirmar este cobro.

El diseño contempla explícitamente el manejo de pagos síncronos y diferidos. Este cobro puede resolverse de forma inmediata (aprobado o rechazado) o quedar en estado pendiente. Para estos últimos casos, se implementa un mecanismo de *webhooks* que permite recibir notificaciones asíncronas con la actualización del estado del pago, garantizando consistencia entre el proveedor externo y el sistema.

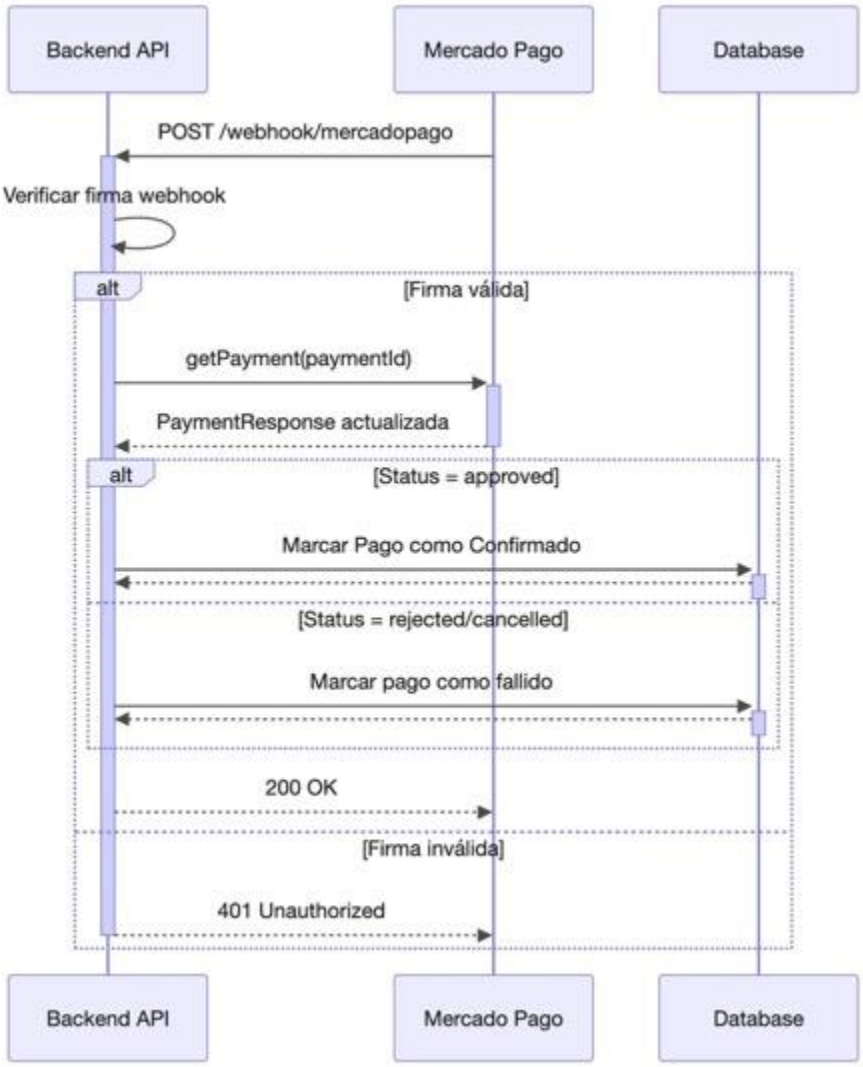


Figura 9: Diagrama de secuencia del sistema de webhooks para el flujo de pago

Los *webhooks* son validados mediante verificación de firma, asegurando que las notificaciones recibidas provengan efectivamente del proveedor y evitando la intervención de actores maliciosos. Cabe notar que para estos casos se adopta una postura permisiva, en la cual se le permite al usuario continuar con el flujo de inicio de carga aun cuando el pago no esté confirmado de forma definitiva, lo cual fue requerido por el cliente.

Toda transacción monetaria se registra formalmente mediante una tabla (*payments*) basada en eventos, lo que permite mantener trazabilidad completa de cada acción (credito, debito y reembolso) y facilita tareas de auditoría sobre los pagos las cargas.

Tras la evaluación del pago, en caso de ser exitosa (pago aceptado o pendiente) se envía el mensaje correspondiente al cargador para iniciar la sesión (RemoteStartTransaction), lo que da lugar a nuestro próximo subflujo en paralelo, el flujo de comienzo de carga.

Hasta este punto, el usuario visualiza la confirmación de su solicitud de carga pero la sesión queda pendiente hasta que el cargador efectivamente confirme el inicio de la transacción. Esta confirmación del cargador está sujeta a la latencia de la red pero previamente también a que el usuario confirme la carga al conectar su auto al cargador.

Mientras el sistema se encuentra en este estado pendiente, propio de la naturaleza asíncrona del protocolo y de un sistema distribuido, el usuario debe esperar a la confirmación o decidir cancelar el flujo. El sistema realiza un *polling* durante un máximo de 60 segundos sobre el estado de la carga para rápidamente dar *feedback* sobre cambios al usuario.

En caso de cancelación antes de que la carga comience o que la carga no sea confirmada en el tiempo dado, se ejecuta el proceso de reembolso correspondiente de la totalidad de la carga, se registra el evento monetario y se marca como cancelada la carga en el sistema.

6.6.2.2. Flujo de comienzo de carga

Este flujo hace foco en la comunicación que se lleva a cabo entre entre el cargador y el sistema para confirmar el inicio de carga y reflejar la misma en el sistema.

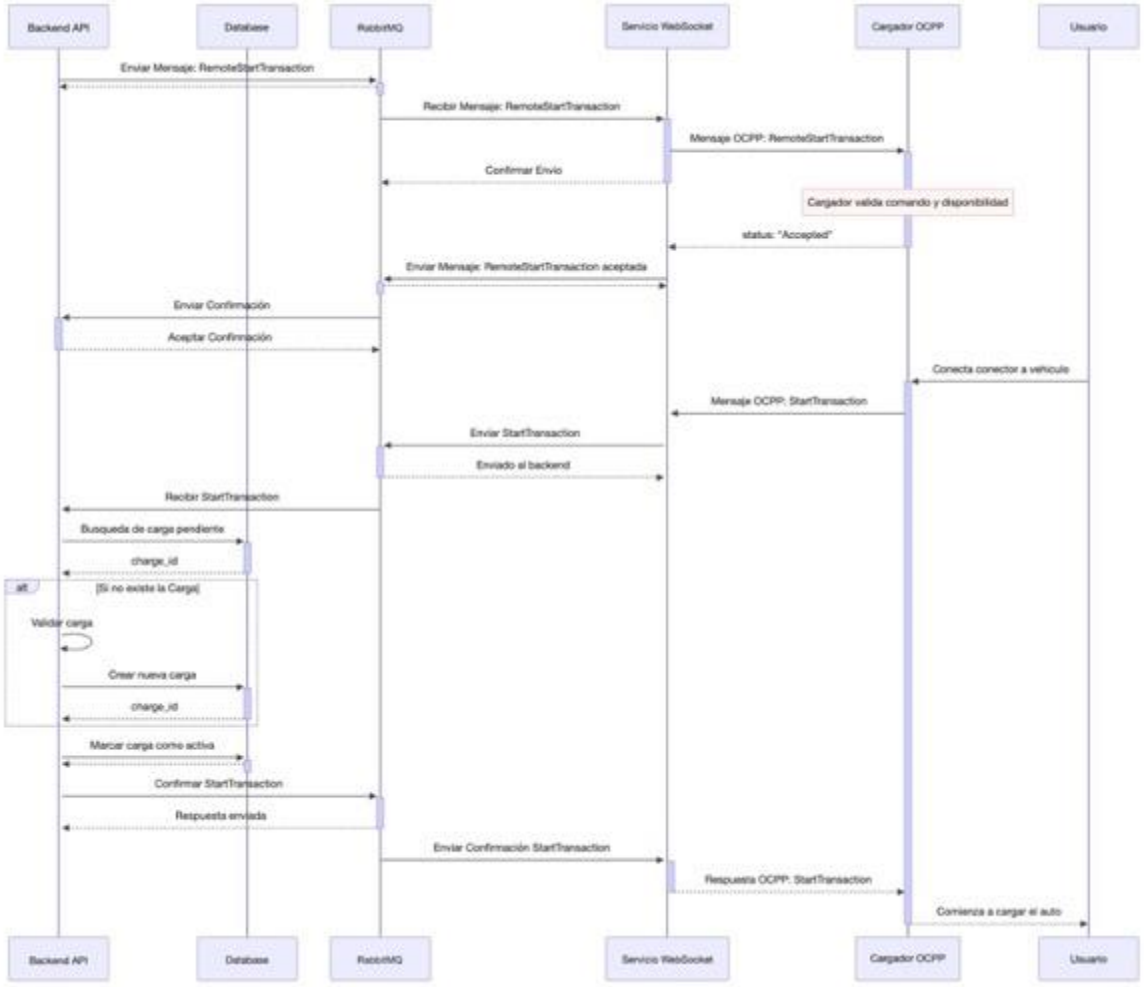


Figura 10: Diagrama de secuencia del flujo de comienzo de carga

El flujo comienza alineado con el anterior, tras la solicitud de carga que realiza el sistema dado el pago de un usuario. Esta solicitud se le envía al cargador y este en consecuencia comienza el flujo de carga una vez el usuario conecta su vehículo.

No obstante, el flujo de comienzo de carga no está limitado únicamente a peticiones realizadas por la API. Como se menciona en la Ingeniería de requerimientos, el sistema también contempla el inicio de cargas mediante el uso de RFIDs. Por ende, el sistema debe también contemplar estas cargas alternativas y su complejidad agregada. En el diagrama se ve reflejado mediante el flujo condicional donde le intentó de carga no existe todavía y el sistema debe validar este intento y de ser aceptado crear un carga nueva. Esto también tiene repercusiones en el pago de estas cargas ya que no hay una interacción mediante una UI donde el usuario pueda interactuar con Mercado Pago. En

estos el pago se gestiona tras la finalización de la carga, esto se explica al analizar el subflujo de fin de carga.

6.6.2.3. Flujo de progreso de carga

Este flujo hace foco en la comunicación que se lleva a cabo durante la transmisión de energía entre el cargador y el sistema.

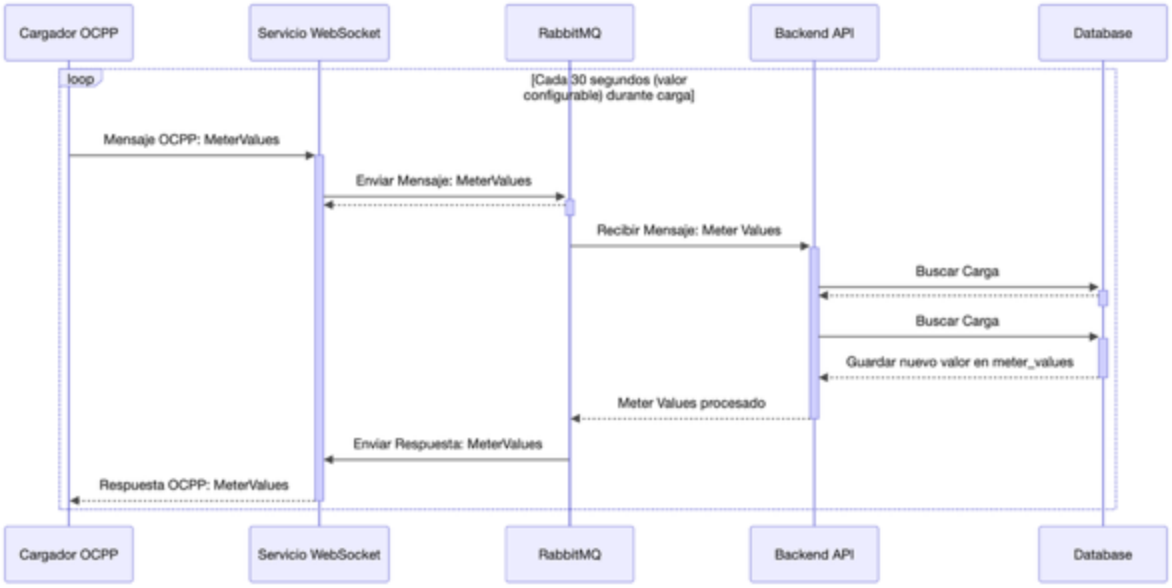


Figura 11: Diagrama de secuencia del flujo de progreso de carga

Durante la carga de un vehículo podemos ver como el cargador constantemente emite datos sobre su estado y también la energía consumida. La persistencia de esta información es clave para la operativa de E-Mobility Solutions, no solo porque condiciona el cargo final realizado por una carga, sino también debido a que esta observabilidad es crucial para entender el estado de la red eléctrica y poder identificar posibles problemas.

6.6.2.4. Flujo de fin de carga

Este flujo hace foco en la comunicación que se lleva a cabo al finalizar la carga entre el cargador y el sistema.

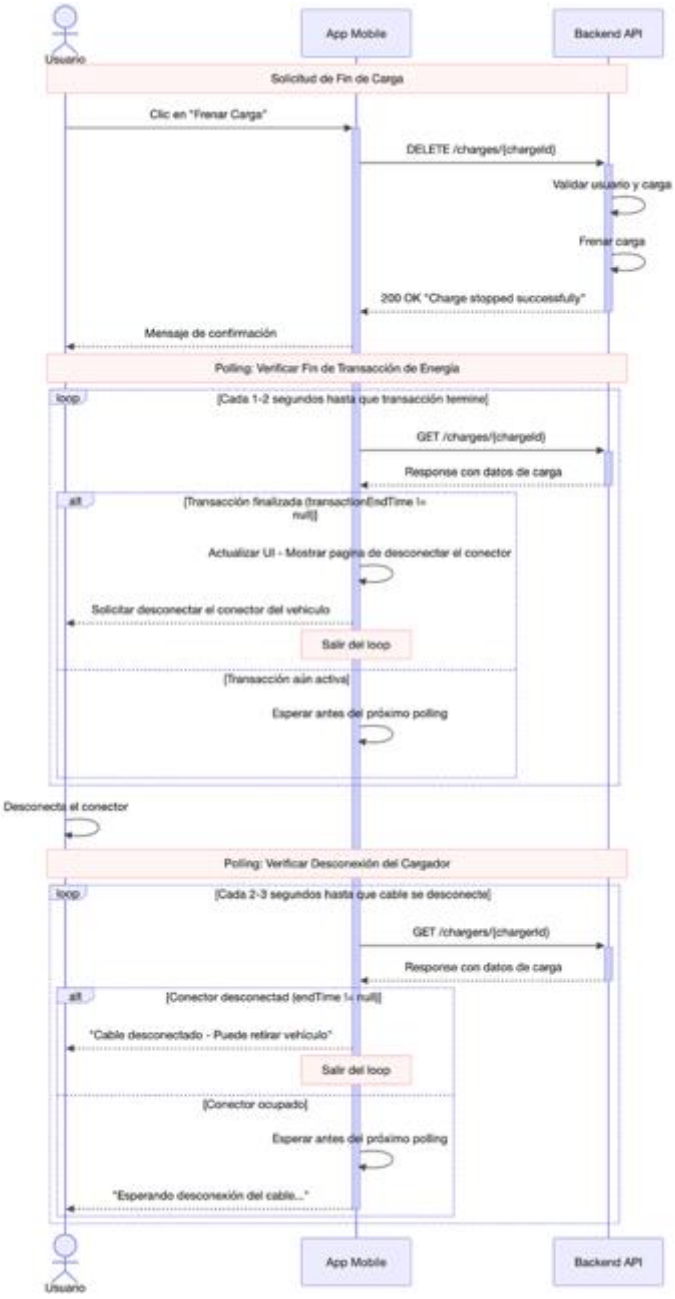


Figura 12: Diagrama de secuencia del flujo de fin de carga de la perspectiva de un usuario

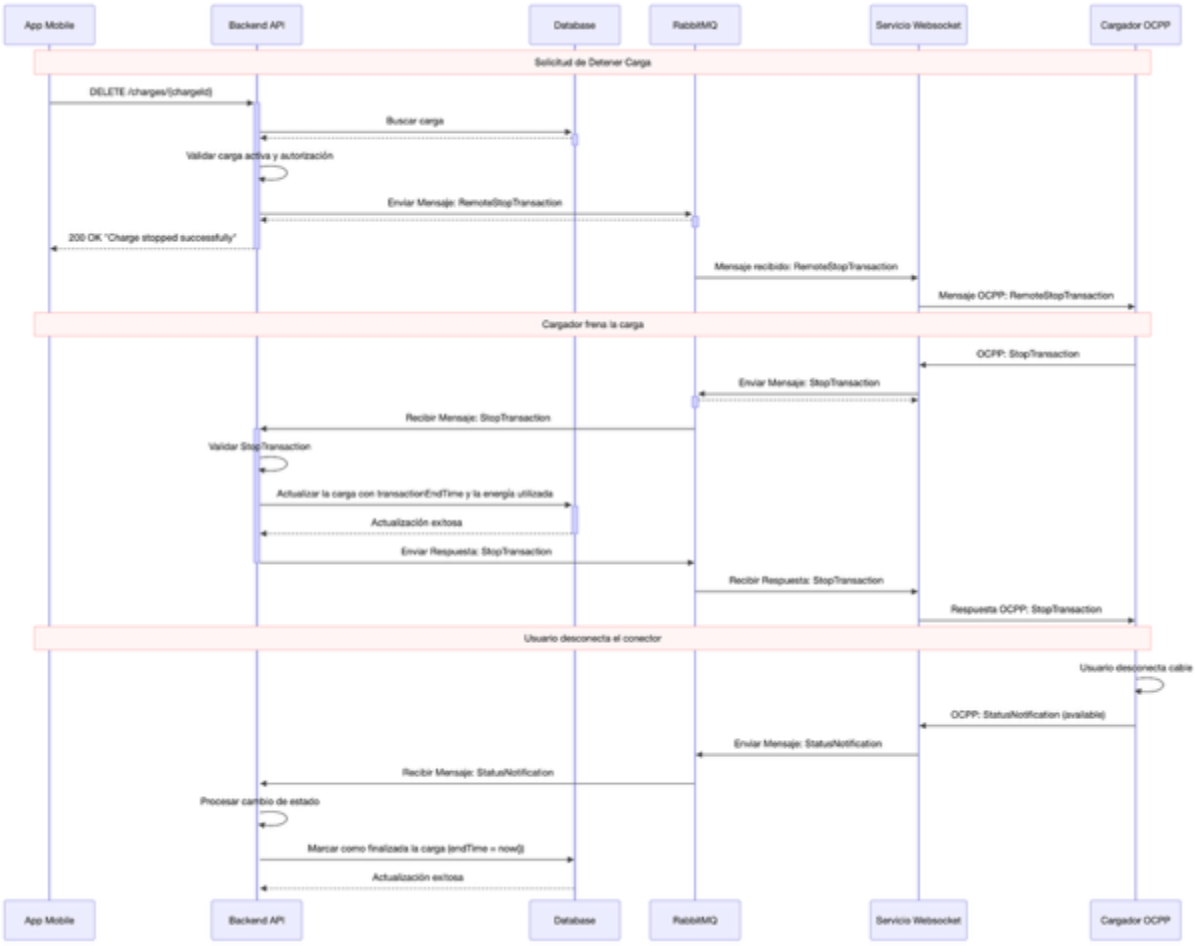


Figura 13: Diagrama de secuencia del flujo de fin de carga de la perspectiva del backend

Al igual que al inicio, el flujo de finalización de una carga puede originarse tanto desde el servidor como de forma externa. Esto introduce múltiples caminos posibles y exige una correcta sincronización entre el estado físico del cargador y el estado lógico en el sistema.

Una particularidad relevante es que, aunque el cargador informe el fin de la transacción, la carga continua activa en el sistema mientras el vehículo permanezca conectado. Desde la perspectiva del usuario, la sesión se considera finalizada únicamente cuando se libera el conector. Este estado se determina a partir de las *status updates* enviadas por el cargador mediante el protocolo OCPP, lo que refuerza el carácter asíncrono y distribuido del flujo.

Una vez confirmada la finalización, con el cargador liberado, el sistema marca la carga como finalizada, calcula el costo total en función del consumo y el tiempo que estuvo utilizando el punto de carga, y registra el débito asociado a la carga. En base a la diferencia entre el crédito inicial y el débito final se evalúa si corresponde hacer un reembolso parcial al usuario o si el usuario queda en deuda que deberá saldar previo a su próxima carga. Cuando salden la deuda se registra el evento crédito correspondiente así cerrar la trazabilidad de la cuenta y asegurar la consistencia de la información. Los usuarios que operan mediante RFID se encuentran sistemáticamente con este último caso, ya que no cuentan con un mecanismo de crédito al momento de iniciar la carga.

6.7. Evaluación de la arquitectura

A partir del análisis de las vistas definidas según el enfoque Views and Beyond, de las tecnologías seleccionadas y de los flujos críticos del sistema, se identificaron y analizaron las principales decisiones arquitectónicas y su impacto. Todas estas decisiones se tomaron con el objetivo de satisfacer los atributos de calidad establecidos y, en consecuencia, cumplir con las expectativas del cliente expresadas mediante los requerimientos no funcionales. En este contexto, la arquitectura fue evaluada mediante la verificación del cumplimiento de los escenarios de calidad definidos.

La validación se llevó a cabo mediante herramientas de análisis de código estático, ejecuciones de pruebas automatizadas, pruebas manuales, y pruebas con usuarios. Estas actividades permitieron validar satisfactoriamente los escenarios de calidad definidos. Los resultados detallados y métricas obtenidas se desarrollan en la sección Gestión de la calidad.

Cabe destacar que estas evaluaciones, dado el alcance del proyecto, se realizaron en un entorno controlado. La validación definitiva de la arquitectura en términos de disponibilidad, rendimiento, escalabilidad y usabilidad requiere validación empírica en un entorno productivo bajo condiciones reales de carga y demanda real, lo que permitirá confirmar la viabilidad operativa integral de la solución.

6.8. Conclusiones y lecciones aprendidas

La arquitectura propuesta logró dar soporte a los escenarios de calidad definidos para el prototipo y se mantuvo alineada con las necesidades y expectativas del cliente. A lo largo del proceso iterativo de definición e implementación de la arquitectura, el equipo adquirió experiencia valiosa tanto en aspectos técnicos como en el manejo de la complejidad propia de sistemas distribuidos.

En términos tecnológicos, el equipo incorporó nuevas competencias, entre ellas el uso de React Native para el desarrollo de la aplicación móvil y la comprensión y aplicación del protocolo OCPP para la comunicación con los cargadores.

Asimismo, se enfrentaron de primera mano los desafíos asociados a sistemas distribuidos. La gestión de actualizaciones de estado en tiempo real, consideración de los tiempos de comunicación entre servicios y manejo de posibles fallas en distintos puntos del flujo.

Este proyecto evidenció cómo un flujo que en apariencia es simple, como conectar y cargar un vehículo, puede involucrar una alta variabilidad de escenarios y múltiples puntos críticos de falla, lo que resalta la importancia de una arquitectura cuidadosamente diseñada.

7. Gestión del proyecto

En esta sección se detalla la planificación, ejecución y evaluación del proyecto, abordando las metodologías aplicadas, herramientas utilizadas y métricas clave para su evaluación. Se retoma el marco metodológico presentado previamente, profundizando en su aplicación práctica y en la evaluación de los resultados obtenidos.

7.1. Etapas y principales hitos del proyecto

La ejecución del proyecto se abordó a través de diversas etapas bien definidas. En primer lugar, se desarrollaron dos etapas orientadas a la construcción de la solución; investigación y desarrollo, cuya estructura y enfoque metodológico se presentaron en la sección Marco metodológico. Finalmente, se llevó a cabo una etapa intrínseca al carácter académico del proyecto, centrada en la elaboración de la documentación de la tesis.

7.1.1 Hitos Identificados

Al observar el recorrido del proyecto en retrospectiva, resulta evidente que hubo hitos que fueron verdaderos puntos de inflexión. Más allá de pautar el avance del desarrollo, cada uno de estos hitos fundamentales funcionó como una validación clave para confirmar las decisiones tomadas a lo largo del proyecto. En su conjunto, estas instancias representaron grandes victorias para el equipo, consolidando la confianza en el producto construido y demostrando, paso a paso, que la visión inicial era un objetivo alcanzable:

1. **Prueba de concepto del protocolo OCPP:** El primer gran hito técnico del proyecto surgió en junio de 2025. Consistió en la primera comunicación bidireccional exitosa con un cargador físico de E-Mobility Solutions, logrando validar la investigación teórica del protocolo OCPP.

2. **Primera revisión académica y ajuste de alcance:** Esta evaluación inicial funcionó simultáneamente como una validación académica del trabajo realizado y como un punto de inflexión en la expectativa del proyecto (detallado en la sección Gestión de riesgos).

3. **Presentación del *dashboard*:** Primera demostración funcional integral orientada al personal operativo E-Mobility Solutions. Fue la primera instancia en la cual se fue más allá del incremento funcional del *sprint* y se realizó una demostración completa de una primera versión del *dashboard* y realizando primeras pruebas de usuario sobre este. El hito ocurrió en octubre de 2025.

4. **Integración de pasarela de pagos:** En noviembre de 2025 se logró validar la arquitectura transaccional al ejecutar con éxito el primer cobro mediante la integración con Mercado Pago, garantizando la seguridad y trazabilidad de los fondos.

5. **Despliegue de la infraestructura:** En diciembre de 2025 se dio la transición del entorno de desarrollo local a un entorno desplegado en la nube para la realización de pruebas.

6. **Distribución inicial de la aplicación móvil:** En enero de 2026 se dio el primer despliegue de la aplicación nativa para dispositivos iOS en un entorno controlado, utilizando TestFlight y utilizando *builds* en formato APK para Android. Este paso habilitó el comienzo formal de las pruebas de usuario directamente en los dispositivos de uso diario.

7. **Validación del flujo *End-to-End* en la nube:** El hito integrador definitivo del proyecto. Consistió en la ejecución exitosa de un ciclo completo: un usuario iniciando un flujo de carga desde la aplicación móvil, procesando el pago correspondiente, y el *backend* alojado en la nube comunicándose en tiempo real con el cargador físico para autorizar la sesión mientras el portal de gestión supervisaba todo en tiempo real.

Figura 14: Línea de tiempo con las etapas del proyecto

7.2. Gestión de la etapa de investigación

Tal como se expuso en la sección Marco metodológico, durante la fase de investigación el equipo optó por implementar Kanban como marco de trabajo. A continuación, se describe cómo se gestionó esta etapa, detallando la organización de las tareas, el flujo de trabajo y los mecanismos de seguimiento utilizados.

7.2.1. Tablero de kanban

Para gestionar la etapa de investigación se utilizó un tablero de Kanban digital, estructurado en columnas que representaban los estados del flujo de trabajo: *to do*, *in progress*, *documenting*, y *completed*.

Figura 15: Tablero Kanban utilizado para la etapa de investigación del proyecto

Cada tarea de investigación se representaba mediante una tarjeta, que avanzaba de izquierda a derecha a medida que progresaba su ejecución. Esta configuración permitió tener visibilidad sobre el estado de cada actividad, identificar rápidamente tareas bloqueadas y mantener una trazabilidad simple de las investigaciones realizadas, desde su definición inicial hasta su cierre y documentación.

7.2.2. Flujo de trabajo

El equipo trabajó con un modelo de entrega de valor ininterrumpido. A medida que surgían nuevas necesidades de investigación, las tareas ingresaban al tablero y se abordaban según la prioridad. Esta dinámica otorgó una excelente capacidad de respuesta frente a los nuevos descubrimientos, permitiendo priorizar y ejecutar tareas críticas de manera inmediata.

Para asegurar el enfoque y evitar la dispersión técnica, se estableció un límite estricto de tareas concurrentes en la columna de ejecución. Específicamente, se definió un *Work In Progress* (WIP) de máximo de 4 tareas simultáneas. WIP refiere a la cantidad de ítems que el equipo está trabajando actualmente [21]. Se determinó un límite máximo de tareas exactamente igual a la cantidad de integrantes del equipo. La implementación de este límite garantizó que el esfuerzo se concentrara en concluir y documentar las investigaciones activas antes de iniciar nuevos análisis, resultando fundamental para identificar cuellos de botella de manera temprana y mantener un ritmo de avance sostenible.

7.3. Gestión de la etapa de desarrollo

Durante la fase de desarrollo, el equipo implementó Scrum como marco de trabajo, decisión cuya justificación detallada se presenta en la sección Marco metodológico. En esta sección se describe específicamente cómo se llevó a la práctica dicha decisión, abarcando la organización del equipo, la planificación de los *sprints*, la realización de los eventos de Scrum y el uso de sus artefactos para gestionar y dar seguimiento al avance del proyecto.

El ciclo de vida del desarrollo se organizó en un cronograma total de 14 *sprints*, cada uno de dos semanas, comenzando el 13 de julio de 2025 y finalizando el 24 de enero del 2026. Esta extensión temporal permitió empaquetar un incremento de producto significativo sin perder la capacidad de adaptación ante posibles cambios.

7.3.1. Artefactos de Scrum

7.3.1.1. *Product backlog*

El *Product Backlog* es el listado priorizado de todo el trabajo pendiente del producto (requisitos funcionales, mejoras, correcciones y tareas técnicas) [22]. Su propósito fue concentrar y ordenar los requisitos identificados, servir de base para la planificación de los *sprints* y permitir que el equipo ajuste el desarrollo a medida que surgen nuevos hallazgos y necesidades.

El proceso inicial de ingeniería de requerimientos llevado a cabo en la etapa de investigación dejó una base de funcionalidades principales. Pero, como se destacó en la sección Ingeniería de requerimientos, el proyecto presentó un proceso de ingeniería de requerimientos constante. El descubrimiento técnico y la retroalimentación continua generaron nuevos requerimientos emergentes. En consecuencia, el *Product Backlog* fue evolucionando y ajustándose a lo largo de todo el proyecto.

7.3.1.2. *Sprint backlog*

El *Sprint Backlog* es el conjunto de ítems del *Product Backlog* seleccionados para un *sprint*, junto con el plan de trabajo necesario para completarlos [23]. Representa el compromiso del equipo para cada iteración, ya que consolida las funcionalidades priorizadas, las descompone en tareas concretas y proporciona una visión clara de qué se espera entregar al finalizar el *sprint*.

El *Sprint Backlog* se gestionó mediante un tablero de Scrum, en el que se representaron las tareas seleccionadas para cada *sprint* y su estado de avance. El tablero contenía los siguientes estados: *to do, in progress, pr pending, testing, blocked, rejected, done*.

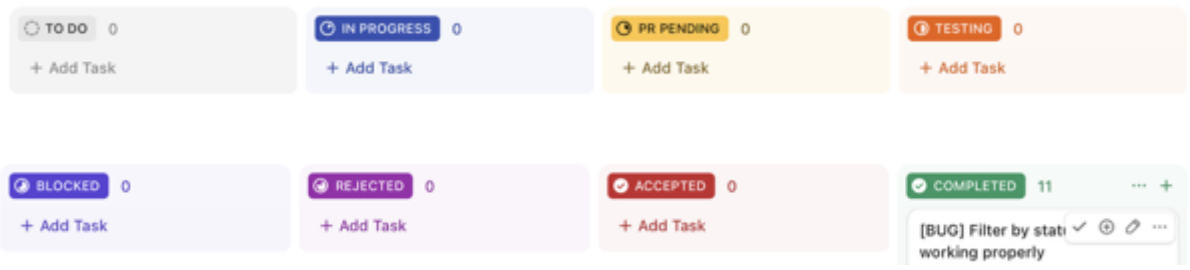


Figura 16: Tablero de Scrum utilizado para gestionar el Sprint Backlog

Este tablero centralizó las tareas correspondientes a cada *sprint*. De esta manera, se garantizó que cada integrante del equipo tuviera visibilidad en tiempo real sobre el estado de las tareas. Esta práctica fue clave para identificar cuellos de botella rápidamente, asegurando un proceso de desarrollo transparente y ordenado.

7.3.1.3. Incremento de valor

El incremento de valor correspondió al conjunto de user stories completadas y probadas, dando lugar a una nueva versión funcional de la solución.

7.3.2 Ceremonias de scrum

7.3.2.1. *Sprint planning*

En cada iteración se realizaba una sesión de planificación de *sprint* [24]. En esta instancia se analizaba el *Product Backlog* y, a partir de un *Sprint Goal* previamente definido, se seleccionaban los elementos a abordar en la iteración [22]. El *Sprint Goal* tenía como propósito orientar el trabajo del equipo y delimitar el incremento de valor que se buscaba alcanzar al finalizar el *sprint*.

Los elementos seleccionados se incorporaban al *Sprint Backlog* y eran descompuestos en subtareas técnicas más granulares, lo que facilitaba su comprensión, estimación y ejecución por parte del equipo. Las tareas ingresaban al *Sprint Backlog* sin una asignación previa a un desarrollador específico. Por el contrario, a medida que los integrantes se liberaban y había capacidad operativa asumían de manera proactiva la responsabilidad de las tareas pendientes. Esta dinámica no solo agilizó la distribución del esfuerzo y evitó cuellos de botella, sino que también fortaleció el compromiso y la autonomía de cada miembro del equipo.

Para determinar la carga de trabajo que el equipo podía asumir en cada *sprint* se consideraban tanto la velocidad promedio como la disponibilidad efectiva de sus integrantes (véase la sección Velocidad vs compromiso). Teniendo esto en cuenta, la estimación de las tareas resultó fundamental para evaluar si existía capacidad para

incorporar más trabajo o, por el contrario, si era necesario reducir el alcance del *sprint*. Para la cuantificación del esfuerzo se utilizaron *Story Points* [25] mediante la técnica de *Planning Poker* [26], empleando una escala de *Fibonacci* [27]. A fin de garantizar la calidad y la precisión de estas estimaciones, se exigía que todo el equipo comprendiera plenamente cada requerimiento antes de proceder a la votación y que la tarea contará con criterios de aceptación claramente redactados.

7.3.2.2. *Daily Scrum*

Durante la ejecución del desarrollo, se optó por adaptar la frecuencia de las reuniones de sincronización establecidas por el marco de trabajo original. Dado que la disponibilidad horaria del equipo era asimétrica durante la semana y se contaba con canales de comunicación asincrónica permanentes, se determinó que realizar las *Daily Scrum* tres veces por semana (martes, jueves y sábados) resultaba más eficiente que mantener una frecuencia diaria [28]. En los días restantes, la coordinación se realizaba principalmente a través de mensajes en WhatsApp, lo que permitió mantener un flujo de comunicación continuo sin requerir reuniones sincrónicas adicionales

Estas sesiones se llevaron a cabo de forma sincrónica mediante videollamadas en Google Meet y se gestionaron bajo un estricto bloque de tiempo de 15 minutos. Para garantizar el dinamismo y el enfoque de la reunión, cada integrante exponía de manera concisa tres puntos fundamentales: los avances logrados desde el último encuentro, las tareas a abordar a continuación y la identificación de cualquier impedimento que requiriera asistencia. El registro documental de estas instancias puede consultarse en la sección 12.9 del Anexo.

7.3.2.3. *Sprint review*

Cada dos viernes se realizaba la *Sprint Review* para presentar el incremento de valor logrado durante el *sprint* [29]. Estas ceremonias se llevaban a cabo con la participación del cliente y, cuando este no estaba disponible, se realizaban con el *Product Owner*. Esta instancia permitió obtener retroalimentación inmediata sobre las funcionalidades y su integración técnica, facilitando ajustes oportunos en el *Product Backlog*.

7.3.2.4. *Sprint retrospective*

Siguiendo la lógica del marco de trabajo, una vez verificado el producto en la *Review*, se procedía a analizar el proceso en busca de aprendizajes y oportunidades de mejora [30]. Para esta ceremonia se implementó la técnica DAKI [31], estructurada en cuatro categorías de análisis:

- ***Drop*:** Identificación de elementos o prácticas que no resultaron efectivas.
- ***Add*:** Propuestas de nuevas dinámicas o herramientas a incorporar.
- ***Keep*:** Reconocimiento de procesos funcionales que deben mantenerse.
- ***Improve*:** Aspectos con potencial de optimización.

La dinámica consistía en que cada integrante aportaba sus observaciones de manera abierta y transparente dentro de las categorías de análisis. Posteriormente, se realizaba una puesta en común y discusión de los puntos expuestos con el fin de definir acciones de mejora concretas para el siguiente *sprint*. Este ejercicio de introspección sirvió para levantar problemas de comunicación, de planificación, de calidad y más. Fue clave para garantizar la optimización continua del flujo de trabajo a lo largo de todo el proyecto.

7.3.3. Roles de Scrum

El marco de trabajo Scrum define tradicionalmente tres roles fundamentales: *Product Owner*, *Scrum Master* y *Developers* [32]. Todos los integrantes participaron activamente en la construcción técnica del *software* tomando el rol de desarrolladores, pero se asignaron las responsabilidades de gestión a miembros específicos para garantizar el cumplimiento y la integridad del marco ágil.

El rol de *Product Owner* fue asumido por Andrew Cooper. Su responsabilidad principal bajo este rol consistió en actuar como nexo directo entre el cliente y el equipo técnico. Se encargó de la gestión del alcance, la redacción de las historias de usuario, la priorización del *Product Backlog* y la validación de los agregados de valor, asegurando que el esfuerzo de desarrollo se centrara en maximizar el valor del producto entregado a E-Mobility Solutions.

La facilitación del proceso ágil recayó sobre Guillermo Martinicorena. Su función fue velar por la correcta aplicación de la metodología Scrum, facilitando las ceremonias, gestionando la métrica de velocidad del equipo, y actuando como facilitador para remover impedimentos técnicos u operativos que pudieran bloquear el avance del *sprint*.

7.3.4. Definition of done

Con el fin de establecer criterios claros para determinar cuándo una funcionalidad podía considerarse realmente finalizada, el equipo definió explícitamente una *Definition of Done* (DoD) aplicable a todos los incrementos desarrollados durante el proyecto.

La *Definition of Done* establecía que una funcionalidad sólo podía considerarse completa cuando cumplía simultáneamente con las siguientes condiciones:

- El desarrollo estaba finalizado y alineado con los requerimientos definidos en la historia de usuario.
- El código respetaba los estándares establecidos y no presentaba errores de linting ni de tipado.
- Se habían ejecutado correctamente las pruebas automatizadas sin fallos.
- El *Pull Request* correspondiente había sido revisado y aprobado por al menos otro integrante del equipo.
- La funcionalidad había sido validada manualmente en el entorno de pruebas.
- La funcionalidad se encontraba integrada en la rama principal de desarrollo sin errores.

Este conjunto de criterios permitió evitar ambigüedades respecto al estado real de cada tarea, asegurando que ningún incremento fuera considerado terminado sin haber atravesado instancias formales de validación técnica y funcional.

7.3.5. Métricas de gestión

Durante la etapa de desarrollo del proyecto, se definieron y monitorearon un conjunto de métricas para evaluar el progreso de forma objetiva y basada en datos reales. El uso de estos indicadores fue fundamental para evitar estimaciones subjetivas, permitiendo gestionar el alcance con mayor seguridad y analizar cómo se distribuyó el esfuerzo del equipo a lo largo del tiempo. Esto es un factor no menor teniendo en cuenta el desafío de alcance que se identificó para este proyecto. A continuación, se presentan las métricas más relevantes obtenidas durante la ejecución.

7.3.5.1. Velocidad vs Compromiso

Para evaluar la madurez en la estimación del equipo y la previsibilidad del proyecto, se realizó un seguimiento continuo de la relación entre los *Story Points* planificados durante la *Sprint Planning* (barras naranjas) y los puntos efectivamente completados al cierre de cada iteración (barras azules). En la Figura 17, la línea de tendencia roja representa la métrica de velocidad promedio calculada sobre las últimas tres iteraciones.

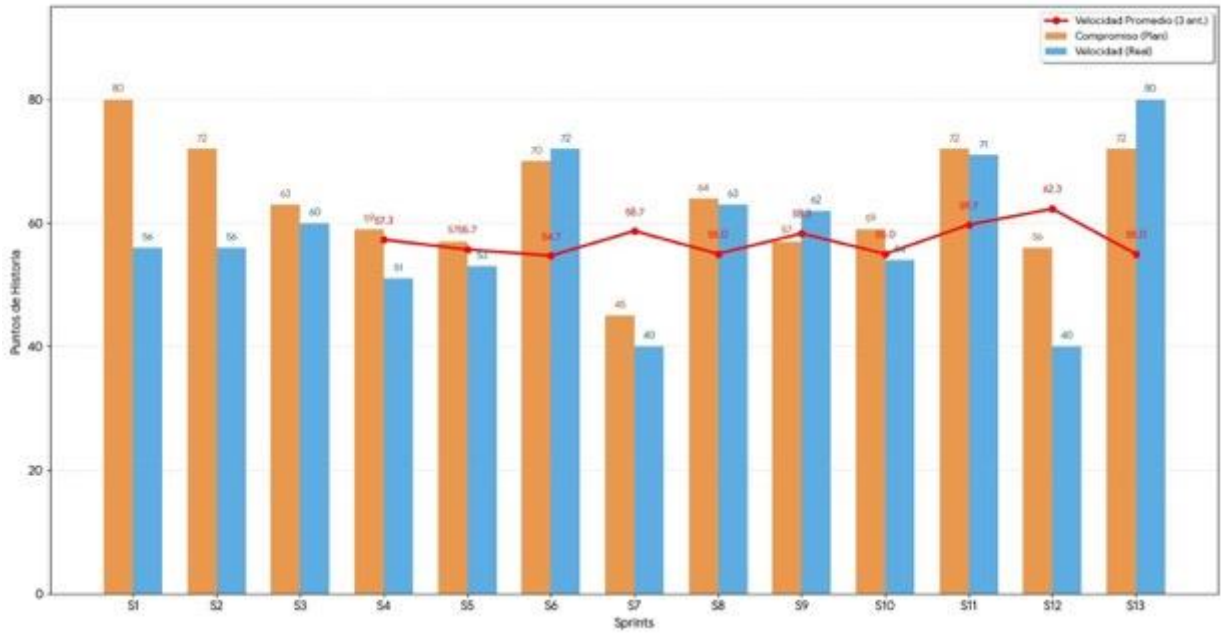


Figura 17: Gráfica que refleja la velocidad vs. el compromiso durante el proyecto

El análisis retrospectivo de esta gráfica refleja claramente la evolución y consolidación del equipo de desarrollo. Durante los primeros *sprints*, se evidencia una brecha entre los puntos comprometidos y la velocidad real alcanzada. Esta desviación inicial no representa una falla de planificación, sino que responde a una etapa natural de aprendizaje y adaptación. Durante este período, el equipo debió transitar una curva de asimilación pronunciada: familiarizándose de manera simultánea con las nuevas herramientas tecnológicas, interiorizando la lógica de un dominio de negocio complejo (como la movilidad eléctrica y el protocolo OCPP) y consolidando la dinámica operativa del propio proyecto.

A partir del *sprint* 4, el equipo contó con el primer dato estadístico consolidado de la velocidad promedio. Desde entonces, se utilizó este dato como línea base para planificar con precisión la capacidad de trabajo y gestionar las expectativas de entrega. Sin embargo, este promedio se gestionó de forma flexible para adaptarse a las distintas circunstancias del proyecto:

- ***Sprints* 6 y 7:** En el *sprint* 6 se tomó la decisión excepcional de elevar el compromiso a 70 puntos para asegurar que el incremento de *software* estuviera completo de cara a la etapa de pruebas con usuarios del *dashboard*. El equipo era consciente de que a estas tareas del *sprint* 7 también se le sumaría la segunda revisión académica, y que como se refleja en la Figura 17, experimentaría una caída de velocidad.

- **Sprints 11 y 12:** Frente a la variación de disponibilidad programada del equipo por motivo de vacaciones en el *sprint* 12 (donde la entrega cayó a 40 puntos), se ejecutó una estrategia de mitigación en el *sprint* 11, llevando la entrega a un máximo histórico de 71 puntos para adelantar trabajo crítico y no comprometer el cronograma general.

- **Sprint 13 (Cierre del Desarrollo):** Finalmente, el desarrollo concluyó con un récord productivo de 80 *Story Points* reales entregados en el *sprint* 13. Este último esfuerzo intensivo garantizó el cumplimiento íntegro del alcance previo al fin del desarrollo.

La evolución de estas métricas, y en particular la baja variabilidad observada en la línea de velocidad promedio, constituye un indicador directo de un proyecto bien gestionado. Esto demuestra que el equipo no solo logró dominar el *stack* tecnológico y el contexto del negocio, sino que alcanzó la madurez metodológica necesaria para utilizar el *feedback* continuo y la gestión de riesgos, adaptando su capacidad operativa a las exigencias del calendario y culminando el desarrollo de manera exitosa.

7.3.5.2. Distribución del esfuerzo por área de proceso

Complementando el análisis de velocidad, el equipo llevó a cabo un registro cuantitativo de las horas efectivas invertidas a lo largo de todo el ciclo de vida del proyecto. Esta métrica tuvo como objetivo visibilizar y auditar la estructura del esfuerzo a través de las distintas áreas de proceso de la Ingeniería de Software.

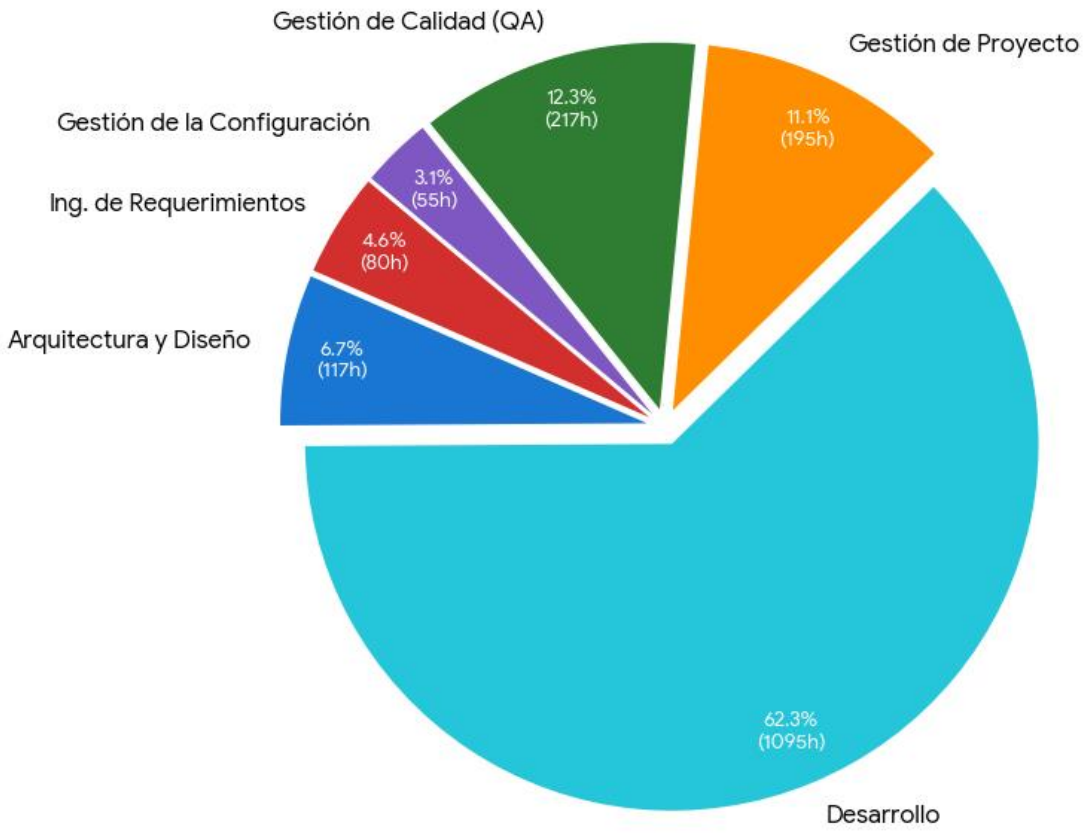


Figura 18: Gráfica que refleja las horas invertidas en cada área identificada durante el proyecto

El análisis revela un volumen acumulado de 1.758 horas de trabajo, cuya proporcionalidad refleja un sólido equilibrio técnico sustentado en un ratio aproximado de 3:1 entre las tareas de ejecución directa y las actividades de soporte y validación:

- **Desarrollo:** Como es esperable en la etapa de construcción de una solución de software tan grande, la mayor parte del esfuerzo se concentró en la codificación de las funcionalidades, ocupando el 62,3% del tiempo.
- **Soporte y Validación Simétrica:** El bloque de Desarrollo cuenta con un respaldo casi simétrico por parte de las áreas de Gestión de la calidad y Gestión del proyecto, con 12,3% y 11,1% respectivamente. Esta paridad fue estratégica para el éxito del proyecto, ya que aseguró que el ritmo de desarrollo y escritura de código nunca superará la capacidad del equipo para validar los incrementos.
- **Arquitectura, Requerimientos y Configuración:** Las áreas de Arquitectura y Diseño, Ingeniería de Requerimientos y Gestión de la Configuración agrupan el esfuerzo restante.

En síntesis, si bien el equipo priorizó la entrega de producto funcional, dedicando el 62% del esfuerzo al desarrollo, el 38% restante se destinó a actividades no directamente vinculadas con la codificación. Esta proporción evidencia la aplicación de un proceso de ingeniería maduro, en el que la ejecución técnica se mantuvo constantemente balanceada por la planificación, la gestión rigurosa y la verificación continua de la calidad.

7.4. Herramientas de gestión

Para la administración de las tareas, el seguimiento de los tableros y la centralización de la documentación, se utilizaron diversas herramientas digitales que facilitaron la colaboración:

- **Gestión de tareas:** Se empleó ClickUp para la implementación del tablero visual tanto en la fase inicial de investigación, como en la fase de desarrollo, permitiendo visualizar el flujo de trabajo y los estados de cada historia de usuario.

- **Comunicación:** Se utilizaron canales de mensajería instantánea (como WhatsApp[19]) y videollamadas mediante Google Meet[20]. para las reuniones de sincronización internas y las instancias con el tutor y el cliente.

7.5. Gestión de la comunicación

La gestión de la comunicación interna resultó ser un factor clave para el éxito del proyecto, especialmente considerando las restricciones de disponibilidad horaria de los integrantes. Al conformar un equipo donde los cuatro miembros tenían compromisos académicos y jornadas laborales extensas, la comunicación efectiva tuvo gran importancia. Se adoptó un modelo híbrido que combinó canales sincrónicos y asincrónicos.

Por un lado, la comunicación asincrónica se gestionó utilizando WhatsApp como herramienta principal para el día a día. Este medio permitió realizar consultas técnicas de rápida resolución, coordinar las revisiones de código y notificar sobre bloqueos en caso de ser necesario. Esta dinámica garantizó la fluidez de la información y la toma de decisiones ágiles, sin interrumpir el flujo de trabajo de cada desarrollador.

Por otro lado, la comunicación sincrónica se reservó estrictamente para las ceremonias formales del marco ágil, utilizando Google Meet La estructura que se planteó en la sección Ceremonias, aseguró que el equipo mantuviera una visión unificada sobre los objetivos de la iteración, facilitando la identificación y remoción de bloqueos técnicos de manera conjunta, sin incurrir en una sobrecarga innecesaria de reuniones.

7.6. Gestión de la documentación

Para garantizar la trazabilidad y el orden de la documentación a lo largo del proyecto, se estableció una estrategia de gestión centralizada. Desde el inicio, se configuró un

[19] https://www.whatsapp.com

[20] https://meet.google.com

repositorio compartido en Google Drive[21], estructurado jerárquicamente mediante carpetas categorizadas según su dominio (arquitectura, requerimientos, revisiones, entre otros). Esta organización aseguró la accesibilidad inmediata y el correcto versionado de cada uno de los documentos.

El uso de Google Docs[22] como herramienta principal facilitó la edición colaborativa en tiempo real, mitigando los riesgos de duplicación de información o pérdida de datos entre los cuatro integrantes.

7.7. Roles transversales

Durante el proyecto se consolidaron distintos roles dentro del equipo que fueron instrumentales para la organización y la distribución de responsabilidades. La asignación de estos roles se dio de manera natural, en función de las fortalezas y virtudes que presentaba cada integrante. Algunos surgieron a partir del marco metodológico implementado, como se vio con los roles de Scrum, mientras que otros se configuraron como roles transversales, necesarios para responder a las demandas organizativas de la construcción del producto de *software*. El objetivo de estos roles fue asignar referentes específicos encargados de liderar y unificar criterios en áreas críticas del proyecto. Los roles asignados fueron los siguientes.

Rol	**Asignado**	**Responsabilidad**
Arquitecto de *Software*	Felipe Crosa	Liderar la toma de decisiones sobre las decisiones arquitectónicas.
Referente UX/UI	Mercedes Bañales	Liderar el diseño visual y la usabilidad del sistema.

[21] https://drive.google.com

[22] https://docs.google.com

Referente QA	Mercedes Bañales	Liderar la definición de la estrategia de *testing* y los criterios de aceptación
Project Manager	Guillermo Martinicorena	Asegurar el cumplimiento del cronograma, facilitar la remoción de impedimentos operativos y mantener la sincronización del equipo.
Business Analyst	Andrew Cooper	Liderar el relevamiento y refinamiento de requerimientos.

7.8. Gestión de riesgos

La gestión de riesgos se llevó adelante mediante un registro estructurado que acompañó la evolución del proyecto desde sus primeras etapas hasta el cierre. Una vez identificados, los riesgos fueron documentados en una planilla de seguimiento donde se definió su categoría, nivel de criticidad (despreciable, leve, grave, crítico o catastrófico), impacto potencial y acciones asociadas. Un ejemplo de esta planilla de seguimiento puede observarse en la sección 12.10 del Anexo.

Este registro no fue un elemento estático, sino que se revisó periódicamente. Cada dos semanas, generalmente durante la *Sprint Planning*, se analizaba el estado de los riesgos activos para evaluar si su nivel había aumentado o disminuido en función del contexto real del proyecto. En caso necesario, se definían nuevas acciones preventivas o correctivas.

Este seguimiento permitió que la gestión fuera iterativa y dinámica, alineada con el enfoque ágil adoptado y facilitando la toma de decisiones informadas a lo largo del desarrollo.

7.8.1 Clasificación de riesgos

Para facilitar su análisis y seguimiento, los riesgos fueron agrupados en tres categorías principales, en función de su naturaleza y del tipo de impacto que podían generar en el

proyecto. Esta clasificación permitió organizar su evaluación y comprender mejor el tipo de exposición al que se enfrentaba el equipo en cada etapa del desarrollo.

7.8.1.1 Riesgos técnicos

Corresponden a aquellos riesgos vinculados a decisiones tecnológicas, complejidad de implementación, integración con sistemas externos y curva de aprendizaje de nuevas herramientas. Estos riesgos podían afectar directamente la viabilidad técnica de la solución o generar retrasos significativos en el desarrollo.

Con el objetivo de analizar su comportamiento a lo largo del proyecto, en la Figura 19 se presenta la evolución de los riesgos técnicos durante los distintos *sprints*.

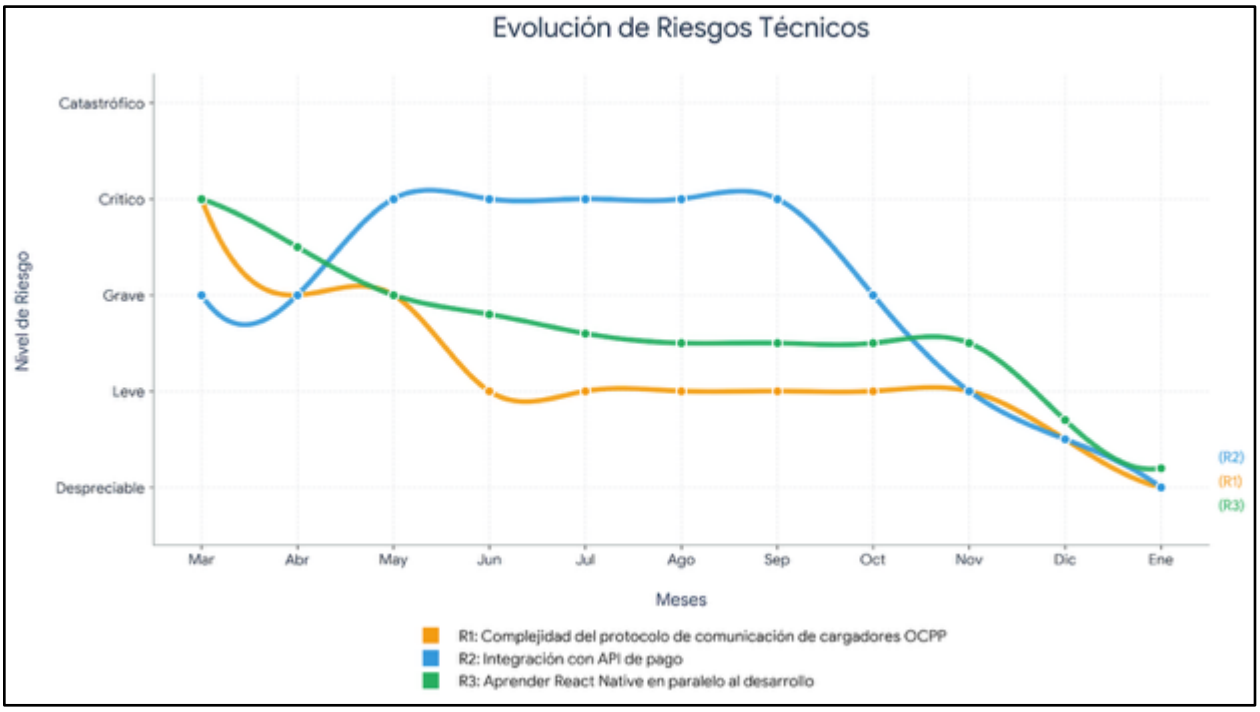


Figura 19: Gráfica que refleja la evolución de riesgos técnicos durante el proyecto

A continuación, se detalla la evolución particular de cada uno.

7.8.1.1.1. R1: Complejidad del protocolo OCPP

Este riesgo estuvo asociado a la complejidad técnica del protocolo de comunicación OCPP, fundamental para la conexión y control de los cargadores eléctricos.

A continuación, se presenta su especificación utilizando la plantilla BDD adoptada para la gestión de riesgos.

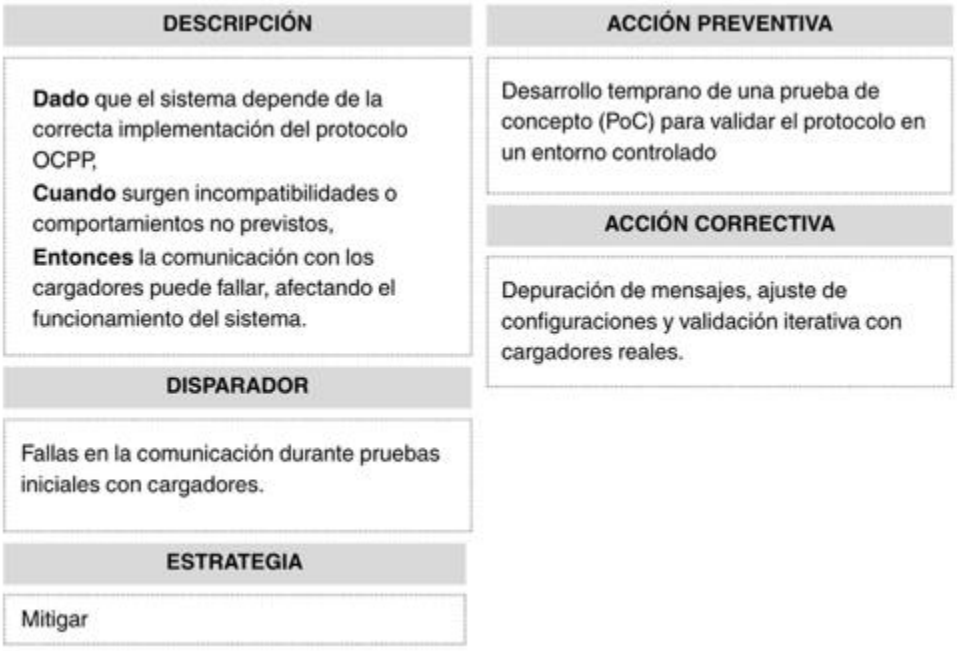


Figura 20: Especificación del riesgo R1

Tal como se observa en la Figura 19, este riesgo comenzó en un nivel alto debido a la complejidad técnica del protocolo y la falta de experiencia inicial del equipo.

Para mitigarlo tempranamente, se realizó un estudio exhaustivo de la documentación oficial del protocolo OCPP, lo que permitió comprender su estructura, los tipos de mensajes involucrados y los flujos de comunicación esperados. A su vez, se desarrolló una PoC en las primeras etapas del proyecto, con el objetivo de validar la factibilidad técnica en un entorno controlado. Más detalles de estas instancias pueden encontrarse en la sección Ingeniería de requerimientos.

A medida que se implementaron las funcionalidades asociadas y se consolidó la comunicación con los cargadores mediante pruebas iterativas, el riesgo fue disminuyendo progresivamente hasta estabilizarse en un nivel bajo. Finalmente, tras completar todas las funcionalidades vinculadas al protocolo y verificar su correcto funcionamiento en escenarios reales de uso, el riesgo pasó a considerarse despreciable.

7.8.1.1.2. R2: Integración con API de pago

Este riesgo estuvo asociado a la integración con una pasarela de pago externa, necesaria para permitir a los usuarios realizar transacciones dentro de la aplicación. Su definición formal se detalla en la siguiente figura.

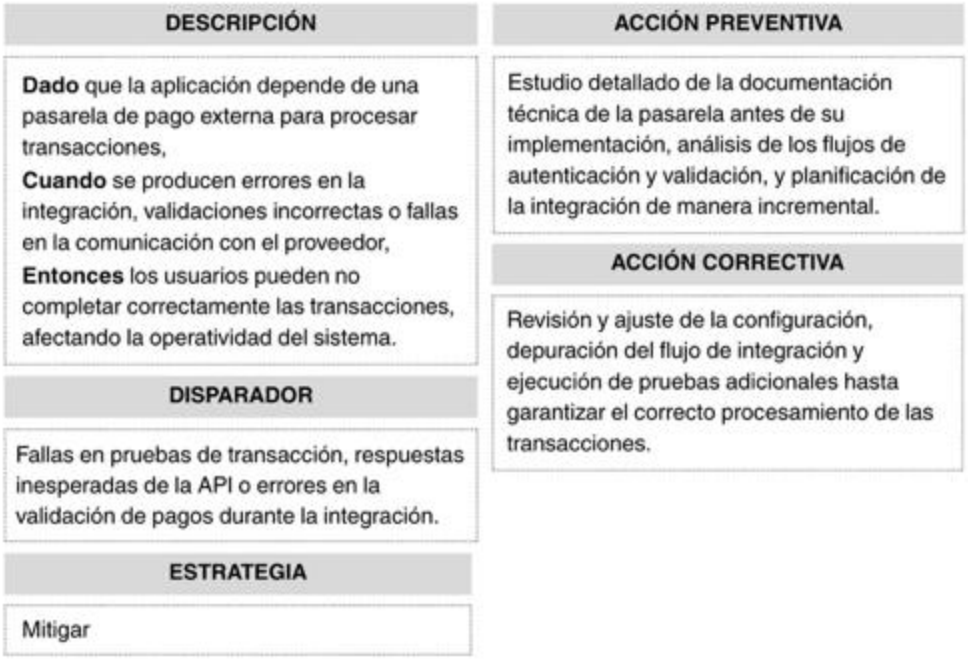


Figura 21: Especificación del riesgo R2

Inicialmente, el riesgo se clasificó como alto debido a la complejidad inherente a los sistemas de pago, que implican el manejo de datos sensibles, estrictos requisitos de seguridad, validaciones específicas y el cumplimiento de normativas aplicables. Asimismo, la dependencia de un proveedor externo incrementaba la incertidumbre técnica en las primeras etapas.

Dado que la pasarela de pago fue definida por el cliente, la estrategia adoptada consistió en mitigar el riesgo mediante un estudio detallado de su documentación técnica antes de iniciar la implementación. Se analizaron los flujos de autenticación, los mecanismos de validación y las restricciones de seguridad requeridas, con el objetivo de anticipar posibles inconvenientes.

La integración se llevó a cabo de manera incremental, validando cada componente del flujo de pago mediante pruebas controladas y verificando el correcto procesamiento de las transacciones en distintos escenarios.

Tal como se observa en la Figura 19, una vez completada la integración y superadas las pruebas funcionales, la incertidumbre asociada disminuyó significativamente, reduciendo el riesgo a nivel bajo y estabilizándose en ese estado durante las iteraciones posteriores.

7.8.1.1.3. R3: Aprender React Native en paralelo al desarrollo

Este riesgo estuvo asociado a la incorporación de herramientas y tecnologías nuevas para el equipo de desarrollo, necesarias para la implementación de determinados componentes del sistema.

En la siguiente figura se expone la formulación estructurada del riesgo según la plantilla BDD adoptada en el proyecto.

DESCRIPCIÓN	ACCIÓN PREVENTIVA
Dado que la aplicación móvil debía desarrollarse utilizando React Native, **Cuando** el equipo no posee experiencia previa en dicha tecnología al inicio del proyecto, **Entonces** pueden producirse demoras en el desarrollo, retrabajo o dificultades técnicas durante las primeras iteraciones.	Investigación inicial sobre la tecnología, revisión de documentación y buenas prácticas, intercambio de conocimientos dentro del equipo.
DISPARADOR	**ACCIÓN CORRECTIVA**
Falta de conocimiento práctico evidenciada en dificultades durante las primeras implementaciones o demoras en tareas vinculadas a la aplicación móvil.	Refactorización de componentes implementados incorrectamente y adopción de mejoras técnicas a medida que se adquiría mayor experiencia en React Native.
ESTRATEGIA	
Mitigar	

Figura 22: Especificación del riesgo R3

Como se puede ver en la Figura 19, el riesgo se clasificó inicialmente en un nivel alto, dado que ninguno de los integrantes del equipo contaba con experiencia previa en React Native y se sabía que la aplicación móvil debía desarrollarse utilizando esta tecnología.

Con el transcurso de los sprints, el equipo realizó tareas de investigación, compartió conocimientos adquiridos y comenzó a incorporar buenas prácticas en el desarrollo. Asimismo, dos integrantes obtuvieron experiencia adicional en otros proyectos utilizando la misma tecnología, lo que contribuyó a acelerar el proceso de aprendizaje colectivo.

Como resultado de este crecimiento progresivo en el dominio técnico, la incertidumbre asociada fue disminuyendo gradualmente hasta que el riesgo pasó a considerarse despreciable hacia las últimas etapas del proyecto.

7.8.1.2 Riesgos de infraestructura

Esta categoría contempla los riesgos relacionados con la disponibilidad de infraestructura física necesaria para la realización de pruebas. En el marco del proyecto, el principal riesgo identificado estuvo vinculado al acceso a cargadores eléctricos para validar la comunicación y el funcionamiento integral del sistema en un entorno real.

La Figura 23 ilustra cómo evolucionó este riesgo a lo largo de los distintos *sprints* del proyecto.

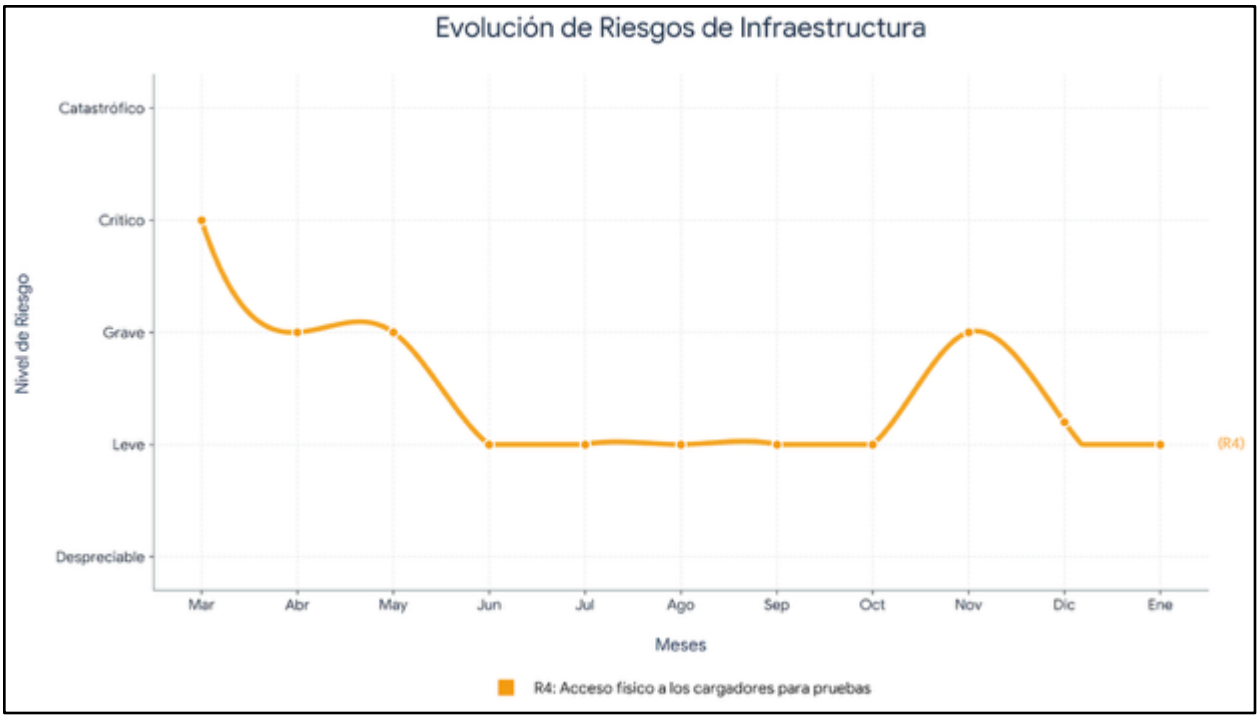


Figura 23: Gráfica que refleja la evolución de riesgos de infraestructura durante el proyecto

A continuación, se describe su comportamiento y las acciones adoptadas para su tratamiento.

7.8.1.2.1. R4: Acceso físico a los cargadores para pruebas

La disponibilidad de hardware específico resultó un factor crítico para validar la comunicación con los cargadores mediante el protocolo OCPP.

A continuación, se define su estructura en la Figura 24.

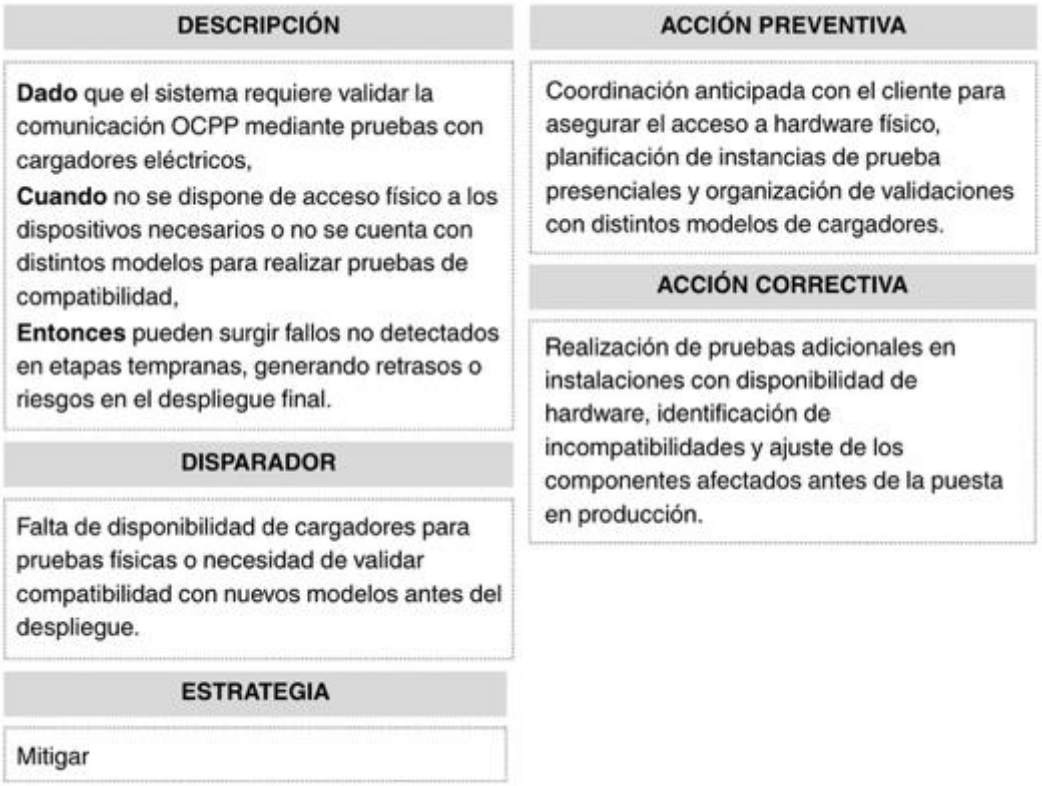


Figura 24: Especificación del riesgo R4

Según indica la Figura 23, el riesgo se clasificó inicialmente en un nivel alto, dado que el equipo no disponía de hardware para probar la comunicación OCPP. Esta limitación generaba una alta incertidumbre respecto al comportamiento del sistema fuera del entorno de desarrollo.

A mediados del proyecto, el nivel de riesgo disminuyó cuando se obtuvo acceso a un cargador provisto por el cliente. Esto permitió comenzar a realizar pruebas físicas concretas y validar la integración entre los distintos componentes del sistema, reduciendo significativamente la incertidumbre inicial.

No obstante, durante los meses de noviembre y diciembre el riesgo volvió a incrementarse hasta alcanzar un nivel grave. En esta etapa fue necesario realizar pruebas con distintos modelos de cargadores para asegurar el cumplimiento del escenario de calidad E6: Integración multi-marca. La posibilidad de detectar

incompatibilidades en una instancia avanzada del proyecto implicaba un impacto potencial considerable.

Para abordar esta situación, el equipo llevó a cabo pruebas presenciales en las oficinas de E-Mobility Solutions, utilizando otros cargadores disponibles en dichas instalaciones. Estas instancias permitieron detectar posibles fallos con anticipación y realizar los ajustes necesarios.

Una vez completadas las validaciones con múltiples dispositivos y corregidos los casos identificados, el riesgo se redujo nuevamente, al confirmarse el correcto funcionamiento del sistema en un entorno real y bajo diferentes configuraciones de *hardware*.

7.8.1.3 Riesgos organizacionales

Esta categoría comprende los riesgos vinculados a factores humanos, coordinación y planificación del proyecto, tales como la dependencia del cliente para validaciones, la disponibilidad del equipo y la definición del alcance. A diferencia de los riesgos técnicos o de infraestructura, estos no estaban asociados directamente a decisiones tecnológicas, sino a dinámicas de trabajo y gestión que podían impactar en el ritmo de avance y en el cumplimiento de los plazos establecidos.

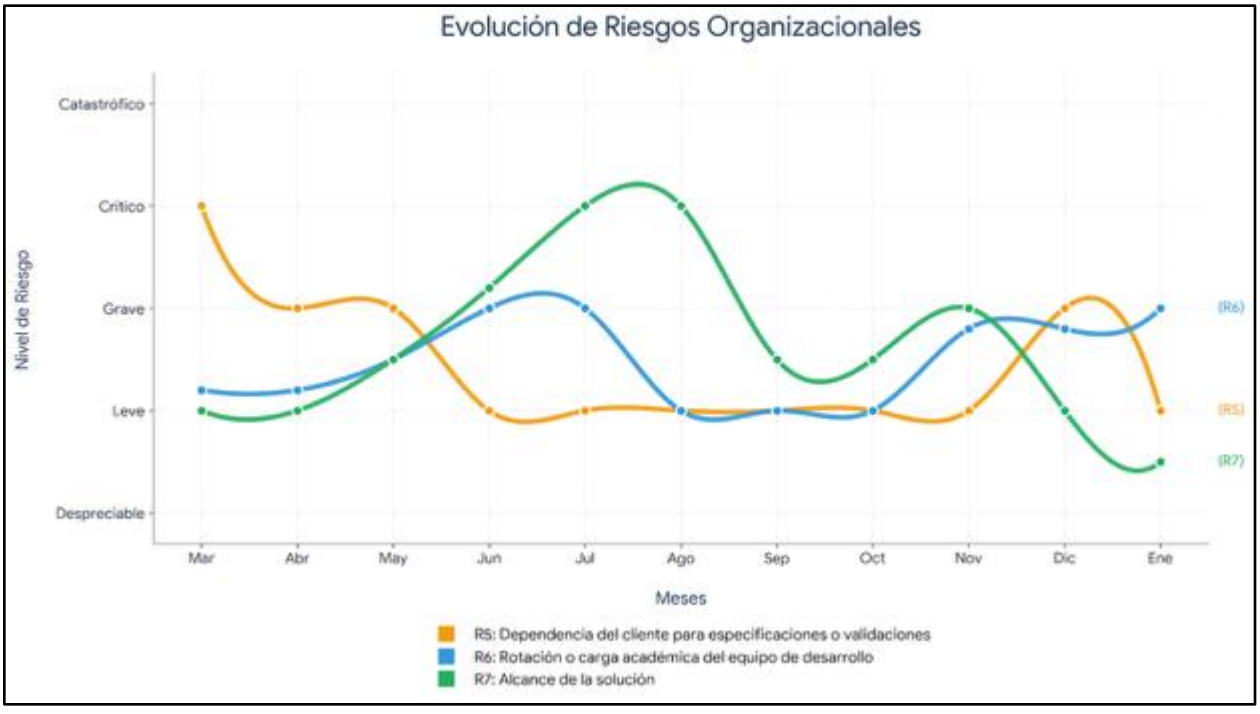


Figura 25: Gráfica que refleja la evolución de riesgos organizacionales durante el proyecto

A continuación, se detalla el comportamiento de cada uno.

7.8.1.3.1. R5: Dependencia del cliente para especificaciones o validaciones

La necesidad de contar con *feedback* frecuente del cliente para definir reglas de negocio y validar funcionalidades representó un factor de incertidumbre relevante durante el proyecto.

La Figura 25 resume la caracterización estructurada del riesgo y las acciones previstas para su tratamiento.

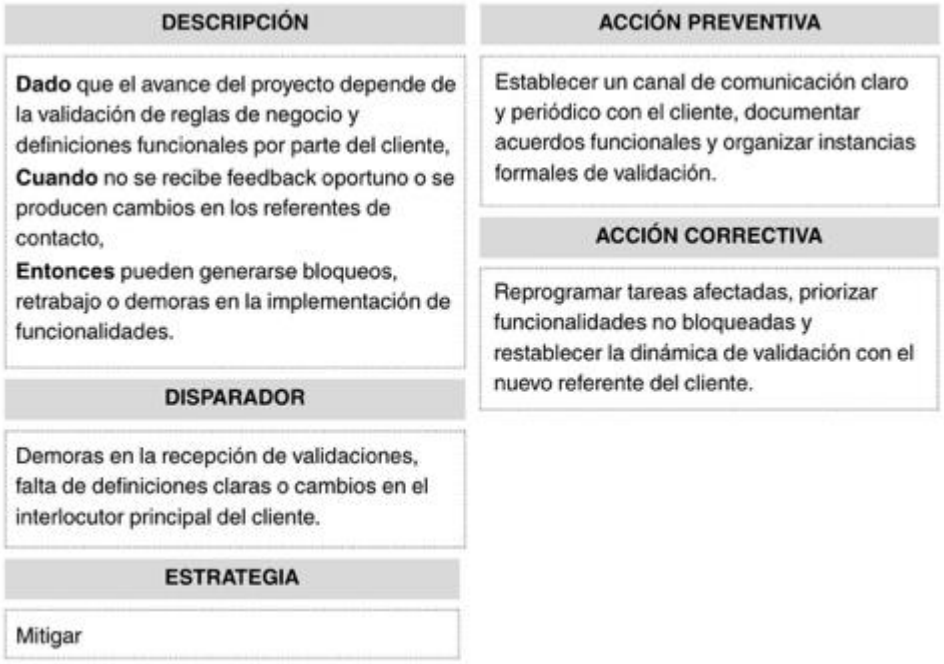


Figura 26: Especificación del riesgo R5

Inicialmente, el riesgo se clasificó en un nivel alto debido a la dependencia constante de validaciones externas para poder avanzar en determinadas funcionalidades. La falta de definiciones claras podía generar bloqueos o retrabajo.

A medida que los requerimientos fueron consolidándose y se estableció un flujo de validaciones más ordenado, la incertidumbre disminuyó progresivamente hasta ubicarse en un nivel bajo.

Sin embargo, en diciembre el riesgo aumentó de manera temporal cuando Matías, quien había sido el principal contacto del equipo durante todo el proyecto, anunció su salida de la empresa. Esta situación generó preocupación respecto a la continuidad en la comunicación y la transferencia de conocimiento.

Posteriormente, el riesgo volvió a reducirse al retomar el contacto con Alfredo Pintos, otro referente del cliente con quien ya existía una relación previa. Esto permitió mantener la dinámica de validaciones y mantener la continuidad del trabajo.

7.8.1.3.2. R6: Rotación o carga académica del equipo de desarrollo

La disponibilidad efectiva del equipo estuvo condicionada por compromisos académicos y personales que coexistían con el desarrollo del proyecto.

En la siguiente figura se detallan los elementos que conforman este riesgo y su estrategia de tratamiento.

DESCRIPCIÓN	ACCIÓN PREVENTIVA
Dado que los integrantes del equipo poseen compromisos académicos y personales además del proyecto, **Cuando** se presentan parciales, entregas, vacaciones u otras situaciones que reducen la disponibilidad, **Entonces** puede verse afectado el ritmo de desarrollo y el cumplimiento de los plazos establecidos.	Planificar considerando el calendario académico, revisar la disponibilidad en cada Sprint Planning y distribuir tareas de forma equilibrada.
DISPARADOR	**ACCIÓN CORRECTIVA**
Reducción temporal de la disponibilidad de integrantes durante determinados períodos del calendario académico o vacacional.	Reasignar tareas según disponibilidad real, ajustar la planificación del sprint y redefinir prioridades para evitar sobrecargas y retrasos críticos.
ESTRATEGIA	
Mitigar	

Figura 27: Especificación del riesgo R6

Inicialmente, el riesgo se ubicó en un nivel medio debido a parciales, entregas y otras responsabilidades que podían afectar la dedicación al proyecto.

A medida que avanzaron los meses, el equipo logró organizarse de manera más eficiente, redistribuyendo tareas y planificando en función de los calendarios académicos. Esta adaptación permitió disminuir el riesgo de forma moderada, aunque sin eliminarlo completamente.

Sin embargo, durante diciembre y enero se registró un aumento significativo, alcanzando un nivel grave. En este caso, la causa no fue la carga académica, sino la rotación temporal asociada a las vacaciones, lo que redujo la disponibilidad de integrantes en un momento crítico.

Para mitigar este impacto, en cada *Sprint Planning* se revisó explícitamente la disponibilidad real del equipo y se ajustó la asignación de tareas. Esta práctica permitió evitar sobrecargas individuales y sostener el avance sin comprometer los plazos acordados.

7.8.1.3.3. R7: Alcance de la solución

La dimensión del alcance definido para el proyecto constituyó otro factor organizacional relevante, dado que debía ser compatible con el tiempo y los recursos disponibles.

A continuación, se define el riesgo junto con las medidas previstas para su gestión.

DESCRIPCIÓN	ACCIÓN PREVENTIVA
Dado que el proyecto posee un conjunto definido de requerimientos y funcionalidades a implementar, **Cuando** el alcance resulta excesivo en relación con el tiempo y los recursos disponibles, **Entonces** puede ponerse en riesgo la finalización del proyecto dentro del plazo acordado.	Revisar periódicamente el backlog, priorizar funcionalidades críticas y validar el alcance junto con el cliente y tutor.
DISPARADOR	**ACCIÓN CORRECTIVA**
Observaciones sobre la viabilidad del alcance, acumulación de funcionalidades pendientes o desalineación entre el backlog y el tiempo disponible.	Ajustar el alcance, redefinir prioridades y reorganizar el backlog para asegurar la entrega dentro de los plazos establecidos.
ESTRATEGIA	
Mitigar	

Figura 28: Especificación del riesgo R7

Como se observa en la Figura 25, en un comienzo, el riesgo se clasificó como bajo, ya que los requerimientos parecían abordables dentro del cronograma previsto.

Sin embargo, durante la primera instancia de revisión en julio, un revisor externo señaló que el alcance propuesto resultaba demasiado ambicioso en relación con el tiempo y los recursos disponibles. A partir de esta observación, y en conjunto con el cliente, se decidió revisar el *backlog*, redefinir prioridades, y alinear en las expectativas.

Esta reorganización permitió reducir nuevamente el nivel de riesgo y mantenerlo bajo durante varios *sprints*.

En noviembre, no obstante, volvió a incrementarse debido a que, al aproximarse el cierre del desarrollo, aún restaba una cantidad considerable de funcionalidades por completar en relación con el tiempo disponible.

Finalmente, tras implementar las funcionalidades restantes y completar el alcance acordado, el riesgo pasó a considerarse despreciable al cierre del proyecto.

7.8.2 Análisis cuantitativo

Con el objetivo de priorizar y gestionar adecuadamente los riesgos identificados, se realizó una evaluación sistemática considerando dos dimensiones: la probabilidad de ocurrencia y el impacto potencial en el proyecto.

Para cada riesgo se definieron los siguientes atributos:

- **Probabilidad de ocurrencia (P):** nivel que indica la posibilidad de que el riesgo se materialice en un determinado momento. Se utilizó una escala cualitativa de cinco niveles: Raro, Poco probable, Posible, Muy probable y Casi seguro.
- **Impacto o consecuencias (I):** nivel que representa la magnitud del efecto que el riesgo tendría en caso de concretarse. Se establecieron cinco categorías: Despreciable, Leve, Grave, Crítico y Catastrófico.

La combinación de ambos factores permitió ubicar cada riesgo dentro de una matriz de evaluación, clasificándolo en zonas diferenciadas mediante una codificación por colores según su criticidad. Dicha combinación se denomina magnitud.

La Figura 29 presenta la matriz utilizada durante todo el proyecto para el seguimiento de los riesgos.

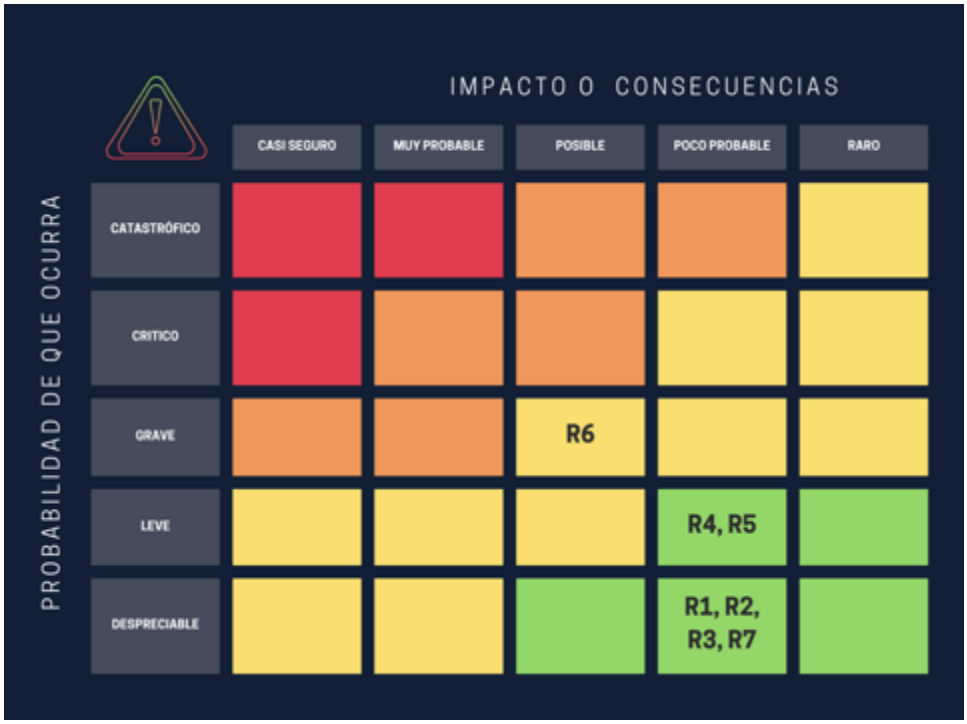


Figura 29: Matriz de riesgos

Esta matriz no fue una representación estática, sino una herramienta de monitoreo continuo. A lo largo de los distintos *sprints*, los riesgos fueron reubicándose dentro de la matriz en función de su evolución y de la efectividad de las acciones de mitigación implementadas.

La posición que se observa actualmente corresponde al estado final de los riesgos al cierre del proyecto. En este punto:

- **R6** se ubica en una zona de riesgo grave, reflejando que, si bien fue gestionado, mantuvo una criticidad relevante hacia las últimas etapas.
- **R4 y R5** se encuentran en una zona de riesgo leve.
- **R1, R2, R3 y R7** se sitúan en la zona de riesgo despreciable, evidenciando que las estrategias adoptadas permitieron reducir significativamente su nivel de exposición inicial.

Este enfoque dinámico permitió no solo identificar y priorizar riesgos en cada etapa, sino también visualizar su disminución progresiva como resultado de las medidas implementadas.

7.8.3 Resultados

La aplicación sistemática del proceso de identificación, evaluación y seguimiento de riesgos permitió reducir progresivamente la exposición del proyecto a eventos que pudieran comprometer su viabilidad técnica u organizacional.

A lo largo de los *sprints*, la matriz de riesgos funcionó como una herramienta de monitoreo continuo, facilitando la priorización de acciones y la toma de decisiones informadas. La revisión periódica en instancias de planificación permitió detectar variaciones en la probabilidad o el impacto de determinados riesgos y ajustar las estrategias en consecuencia.

Como resultado:

- Los riesgos técnicos inicialmente clasificados como altos (R1, R2 y R3) fueron mitigados hasta alcanzar niveles bajos o despreciables.
- El riesgo asociado a infraestructura (R4) logró estabilizarse tras la realización de pruebas físicas con distintos dispositivos.
- Los riesgos organizacionales (R5, R6 y R7) fueron gestionados mediante ajustes en la planificación, reorganización del alcance y mejora en la comunicación con el cliente.

Al cierre del proyecto, la mayoría de los riesgos se encuentran en zonas de baja criticidad dentro de la matriz, lo que evidencia la efectividad de las acciones preventivas y correctivas implementadas.

En términos generales, la gestión activa de riesgos contribuyó a sostener el avance del proyecto, minimizar retrabajos y asegurar el cumplimiento del alcance acordado dentro de los plazos establecidos.

7.9. Conclusiones y lecciones aprendidas

La gestión del proyecto fue un pilar central para transformar una idea inicial en una solución funcional y validada, permitiendo ordenar el trabajo, priorizar el alcance y sostener el avance en un contexto de alta complejidad técnica.

El uso combinado de Kanban y Scrum, con tableros visuales, retrospectivas y *sprint reviews*, facilitó la inspección y adaptación continúa, habilitando ajustes oportunos tanto en el producto como en la forma de trabajo del equipo. Las métricas de seguimiento, en particular las relacionadas con velocidad y cumplimiento de compromisos, aportaron una base objetiva para la toma de decisiones, alejándose de percepciones subjetivas sobre el progreso.

La comunicación constante, apoyada en instancias sincrónicas y canales asincrónicos, fue clave para coordinar esfuerzos, anticipar impedimentos y mantener alineados a los distintos actores involucrados.

Finalmente, la gestión sistemática de riesgos permitió identificar tempranamente amenazas técnicas y organizacionales, mitigar su impacto y asegurar que el proyecto pudiera completarse dentro de los plazos y alcances definidos.

8. Gestión de la calidad

En este capítulo se describen los criterios, enfoques y prácticas implementadas para asegurar que el sistema desarrollado cumpliera con los estándares técnicos y funcionales definidos para el proyecto. La gestión de la calidad se integró de manera transversal al proceso de desarrollo, incorporando instancias de validación, control y mejora continua.

A continuación, se presenta la definición de calidad adoptada y el marco teórico que fundamenta las decisiones metodológicas y técnicas aplicadas.

8.1. Definición de calidad

La calidad en EasyCharge fue concebida como un concepto integral que abarca el cumplimiento de los requisitos, la solidez técnica del producto y la experiencia real de uso. Desde la perspectiva de Philip Crosby, la calidad se define como el cumplimiento de los requisitos, entendiendo que un producto es de calidad cuando se ajusta a las especificaciones definidas y previene defectos mediante procesos adecuados.

Complementariamente, el estándar ISO/IEC 25010 [9] establece que la calidad del software puede analizarse a partir de características como adecuación funcional, eficiencia de desempeño, compatibilidad, usabilidad, confiabilidad, seguridad, mantenibilidad y portabilidad. Este modelo permite traducir el cumplimiento de requisitos en atributos concretos y evaluables del producto.

A partir de estos marcos, en el proyecto se adoptó una visión estructurada en tres dimensiones:

- **Calidad de producto:** vinculada a las características técnicas internas y externas del software.
- **Calidad en uso:** relacionada con la experiencia, eficacia y satisfacción del usuario en contexto real.
- **Calidad de proceso:** referida a la forma en que el equipo organiza y ejecuta el desarrollo.

8.2. Prácticas de aseguramiento de la calidad

Con el objetivo de garantizar un enfoque integral, las prácticas de aseguramiento de la calidad se organizaron en función de las tres dimensiones definidas previamente. Esta clasificación permitió vincular cada práctica implementada con los atributos de calidad que buscaba fortalecer, asegurando coherencia entre el marco teórico adoptado y las acciones concretas realizadas durante el desarrollo del proyecto.

8.2.1 Prácticas orientadas a la calidad de producto

Esta dimensión se centró en asegurar que EasyCharge cumpliera de manera robusta y consistente con los requisitos no funcionales definidos desde el inicio del proyecto. En particular, se buscó validar durante el desarrollo que el sistema satisficiera los atributos de calidad establecidos en la sección de Arquitectura, tales como mantenibilidad, disponibilidad, interoperabilidad y portabilidad.

8.2.1.1. Aplicación de estándares

Para garantizar un código limpio, consistente y fácil de mantener, el equipo acordó un conjunto de buenas prácticas y convenciones de desarrollo que guiaron tanto la escritura como la revisión del código a lo largo de todo el proyecto. Estos lineamientos permitieron establecer criterios compartidos desde las etapas iniciales de desarrollo, evitando divergencias de estilo entre los integrantes del equipo y contribuyendo a reducir la introducción de deuda técnica.

Dado que el sistema fue desarrollado íntegramente en TypeScript, se adoptó un conjunto uniforme de convenciones aplicables tanto al *frontend* como al *backend*. Esta homogeneidad tecnológica facilitó la aplicación consistente de reglas en los diferentes repositorios. Las convenciones y buenas prácticas de codificación adoptadas en el proyecto se describen en detalle en la sección 12.11 del Anexo. Estas reglas permiten que el código sea fácilmente comprensible, reduciendo errores y facilitando futuras modificaciones.

La adopción de estos estándares contribuye directamente al atributo de mantenibilidad priorizado en la arquitectura del sistema (Sección 6.2.1), el cual establece como objetivo facilitar la evolución del *software* sin incrementar su complejidad interna.

Para complementar estos estándares, se utilizaron herramientas de automatización y control de calidad orientadas al análisis estático y al formateo automático del código, con el fin de mantener consistencia y prevenir errores desde etapas tempranas del desarrollo.

En particular, se empleó ESLint[23] como herramienta de análisis estático, permitiendo detectar patrones problemáticos, malas prácticas y posibles inconsistencias antes de la ejecución del programa. Asimismo, se utilizó Prettier como formateador automático de código, asegurando una estructura uniforme y coherente en todos los archivos del proyecto. Estas herramientas se configuraron desde el inicio del desarrollo para que cada cambio realizado se ajustara automáticamente a los estándares definidos, reduciendo la probabilidad de introducir errores relacionados con estilo o estructura.

Asimismo, se implementaron *hooks* de Git mediante Husky[24] que aprovechan estas herramientas y refuerzan buenas prácticas en commits, ramas y pushes:

- El hook de *commit-msg* valida que los mensajes de *commit* sigan un formato uniforme y descriptivo.
- El hook de *pre-commit* ejecuta ESLint y Prettier[25], garantizando que solo se integren cambios que cumplan con los estándares de calidad.
- El hook de *pre-push* valida que el nombre de la rama cumpla con los patrones establecidos (como *feature/*, *bugfix/* o *refactor/*) y ejecuta los *linters*, los test automatizados y un *smoke test* antes de permitir la integración en el repositorio.

Esto aseguró que los estándares técnicos se mantuvieran de manera consistente incluso si alguna validación local fuera omitida, centralizando el control de calidad antes de la integración en las ramas principales y garantizando uniformidad en todo el equipo de desarrollo.

[23] https://eslint.org

[24] https://typicode.github.io/husky

[25] https://prettier.io

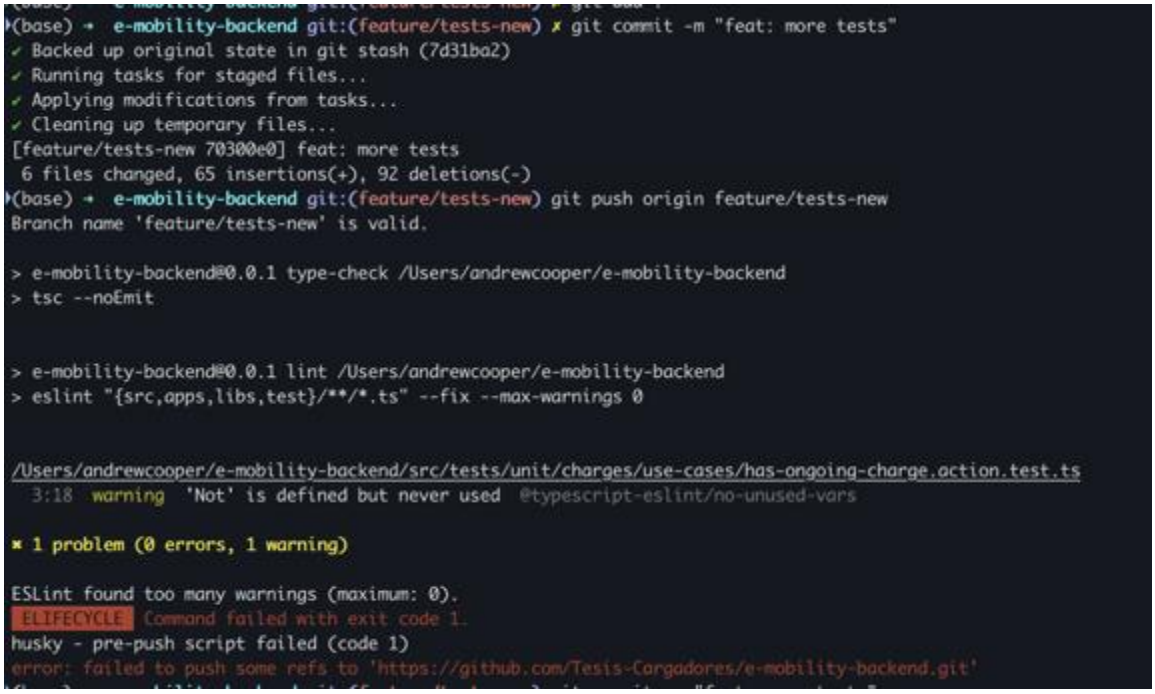


Figura 30: Flujo de validaciones automáticas del pipeline de integración continua

Gracias a esta combinación de estándares, herramientas automatizadas, *hooks* y validaciones centralizadas, el equipo logró mantener un código legible, consistente y fácil de mantener. Esto permitió reducir la deuda técnica, prevenir errores desde etapas tempranas del desarrollo y facilitar la colaboración y escalabilidad del sistema a largo plazo.

8.2.1.2. Revisión del código

El equipo implementó un proceso de revisiones sistemático para garantizar la calidad técnica del proyecto, la coherencia del código y la alineación con los requerimientos del cliente.

Como será explicado en mayor detalle en la sección 9.4.1 Gitflow, todo cambio en el código se realizó siguiendo el flujo de Gitflow: cada nueva funcionalidad se desarrollaba en una rama *feature*, sobre la cual se creaba un *Pull Request* para ser revisado por al menos otro integrante del equipo. Esta práctica permitió detectar errores de manera temprana, asegurar el cumplimiento de los estándares de desarrollo y fomentar el aprendizaje mutuo entre los integrantes del equipo.

Una vez aprobada la revisión, los cambios se integraban a la rama correspondiente, asegurando la coherencia técnica y la consistencia arquitectónica en todo el proyecto. Este enfoque no sólo reforzó la calidad del código, sino que también promovió la colaboración y la alineación entre los desarrolladores, garantizando que cualquier modificación siguiera un proceso controlado y predecible.

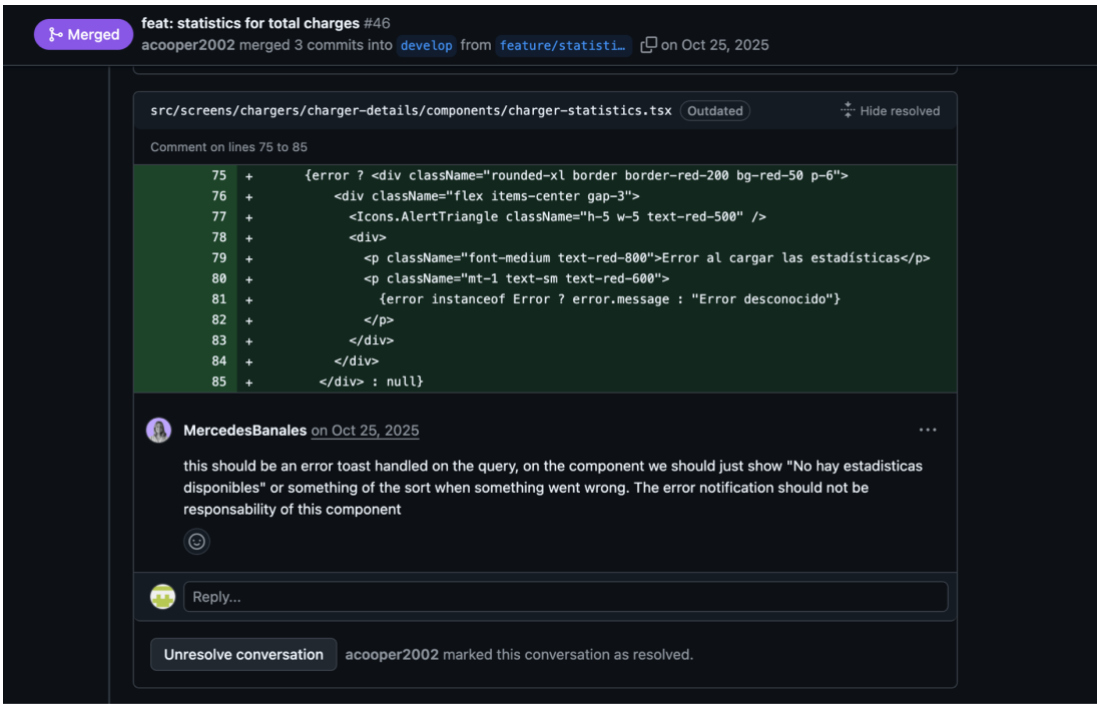


Figura 31: Ejemplo de revisión de un Pull Request

8.2.1.3. Pruebas unitarias automatizadas

Se desarrollaron pruebas unitarias automatizadas utilizando el *framework* Jest para los componentes del *backend* responsables de implementar la lógica de negocio del sistema. Estas pruebas permitieron validar de forma aislada el comportamiento de las reglas de negocio asegurando que las funcionalidades críticas se mantuvieran correctas ante modificaciones futuras.

```
describe('successful scenarios', () => {
  it('should successfully start a charge', async () => {
    await expect(
      action.execute(mockCharger.ocppId, startChargePayload),
    ).resolves.not.toThrow();

    expect(chargerRepository.findOneOrFail).toHaveBeenCalledWith(
      expect.objectContaining({
        where: {
          ocppId: mockCharger.ocppId,
          status: 'active',
          deletedAt: IsNull(),
        },
        relations: ['connectors'],
      }),
    );

    expect(authorizeChargerAction.execute).toHaveBeenCalledWith(
      mockCharger.ocppId,
      startChargePayload.idTag,
    );
    expect(chargeRepository.findOne).toHaveBeenCalledWith({
      where: {
        connector: { id: connectorId },
      },
    });
    //to have been called with at least these params
    expect(chargeRepository.save).toHaveBeenCalledWith(
      expect.objectContaining({
        userId: mockUser.id,
        connectorId: mockConnector.id,
        startTime: startChargePayload.timestamp,
        powerStart: startChargePayload.meterStart,
      }),
    );
  });

  it('should override price rate if new charge', async () => {
    await expect(
      action.execute(mockCharger.ocppId, startChargePayload),
    ).resolves.not.toThrow();

    expect(chargeRepository.save).toHaveBeenCalledWith(
```

Figura 32: Ejemplo de pruebas unitarias para el caso de uso ‘Iniciar una carga’

La automatización permitió detectar errores de forma temprana y reducir el riesgo de regresiones. Asimismo, facilitó la ejecución repetida de validaciones durante el desarrollo y antes de cada integración de código, contribuyendo a mantener un alto estándar de calidad técnica.

Al finalizar el proyecto, el conjunto de pruebas automatizadas alcanzó un total de 163 test suites y 2198 pruebas ejecutadas exitosamente, validando el correcto funcionamiento de los componentes evaluados y reduciendo el riesgo de introducir regresiones (ver *Figura 33*).

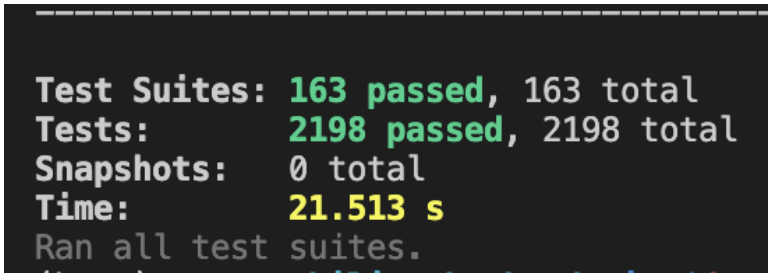


Figura 33: Resultado de la ejecución completa de pruebas automatizadas

8.2.1.4. Pruebas manuales

Además de las pruebas unitarias automatizadas, se realizaron pruebas manuales centradas en la validación de los escenarios de calidad definidos en la arquitectura del sistema (Sección 6.2). Estas instancias permitieron evaluar la correcta implementación de atributos como disponibilidad, interoperabilidad y portabilidad en un entorno de pruebas controlado.

El protocolo de prueba manual para estas instancias se diseñó para contrastar el sistema contra los escenarios de calidad previamente definidos en arquitectura, estableciendo pasos, entradas y resultados esperados de manera estructurada. De esta forma, cada prueba valida que los componentes del sistema cumplan con los requisitos de estabilidad, recuperación y consistencia de datos, asegurando que las funcionalidades técnicas se implementen según lo previsto.

Compatibilidad en dispositivos Android e iOS

Se realizaron pruebas manuales utilizando dispositivos reales con sistemas operativos Android e iOS, validando el correcto funcionamiento de los flujos críticos de la aplicación móvil. Entre estos flujos se incluyeron la autenticación de usuarios, la visualización de cargadores disponibles, el inicio de una sesión de carga y la finalización de la misma.

Estas pruebas permitieron comprobar que la aplicación se comporta de manera consistente en ambos sistemas operativos, cumpliendo con el escenario de calidad E7 (Compatibilidad Multiplataforma), definido en la Sección 6.2.4, el cual establece que el 100% de los flujos críticos debe ejecutarse correctamente en dispositivos Android e iOS.

Integración con cargadores de distintas marcas

Dado que el sistema debe operar con cargadores eléctricos de distintos fabricantes, se realizaron pruebas manuales validando la interacción del sistema con diferentes modelos de cargadores compatibles con el protocolo OCPP 1.6. Para estas pruebas, se utilizaron los cargadores disponibles en las oficinas del cliente E-Mobility Solutions, lo que permitió evaluar la interoperabilidad con distintos modelos de dispositivos.

Figura 34: Evidencia del equipo probando el modelo de cargador TonHe G26 en las oficinas de E-Mobility Solutions

Figura 35: Evidencia del equipo probando la conexión con el modelo de cargador Wallbox, usando un simulador de vehículo para ello

Durante estas pruebas se validó el correcto procesamiento de los mensajes intercambiados entre el sistema central y los cargadores, incluyendo notificaciones de estado e inicio y finalización de sesiones de carga. Estas instancias permitieron comprobar el cumplimiento de los escenarios de calidad asociados a la interoperabilidad, en particular E5 (Cobertura de Mensajes del Protocolo OCPP) y E6 (Integración Multi-Marca), definidos en la Sección 6.2.3.

Los resultados confirmaron que el sistema puede gestionar correctamente los flujos críticos de carga con distintos cargadores compatibles con el protocolo adoptado.

Comportamiento ante situaciones de disponibilidad

Finalmente, se realizaron pruebas manuales orientadas a validar el comportamiento del sistema ante situaciones que afectan la disponibilidad del servicio. Entre los escenarios evaluados se incluyeron desconexiones de red, interrupciones temporales en la comunicación con los cargadores y cambios inesperados en el estado de los dispositivos.

Estas pruebas permitieron validar que el sistema puede recuperar la comunicación con los cargadores y restablecer el estado de las sesiones de carga sin inconsistencias en

los datos registrados. De esta forma, se valida el cumplimiento del escenario de calidad E4 (Recuperación ante Fallos de Conexión), definido en la Sección 6.2.2.

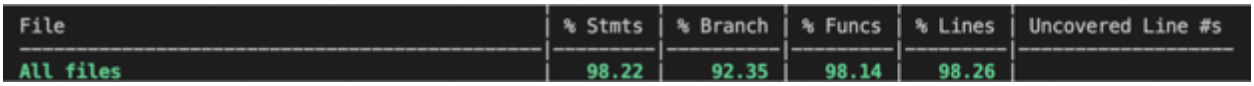
En conjunto, estas instancias de validación manual permitieron comprobar que el sistema mantiene un comportamiento robusto reforzando los atributos de calidad considerados en el diseño de la arquitectura.

8.2.1.6 Métricas de calidad y mantenibilidad del sistema

Con el objetivo de evaluar la calidad del proyecto de manera objetiva, se utilizaron métricas técnicas que permitieron analizar la mantenibilidad del sistema. Estas métricas complementaron las prácticas de aseguramiento de calidad descritas anteriormente, proporcionando evidencia cuantitativa del estado del código y permitiendo identificar áreas de mejora de forma temprana.

8.2.1.6.1 Cobertura de código

Se midió la cobertura del código, alcanzando un valor superior al 98% de cobertura de líneas (ver *Figura 36*). Estos resultados reflejaron un alto nivel de validación del comportamiento del sistema y una baja probabilidad de defectos no detectados en los flujos principales.



File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Line #s
All files	98.22	92.35	98.14	98.26	

Figura 36: Reporte de cobertura de pruebas automatizadas

Esto permite validar empíricamente el escenario de calidad E1 definido en la sección 6.2.1 de Arquitectura y Diseño, el cual establece como criterio alcanzar una cobertura superior al 95% en los componentes que implementan la lógica de negocio asociada a los flujos críticos del sistema. De esta forma, la cobertura obtenida no constituye únicamente una métrica técnica aislada, sino evidencia concreta del cumplimiento de un requisito arquitectónico previamente formalizado.

Un sistema ampliamente cubierto por pruebas unitarias reduce la probabilidad de defectos no detectados y permite realizar refactorizaciones con mayor seguridad, ya que

cualquier comportamiento inesperado en los componentes evaluados es detectado tempranamente por el conjunto de pruebas automatizadas.

La alta cobertura de pruebas refuerza la robustez estructural del sistema, asegurando no solo que el código sea comprensible y mantenible.

8.2.1.6.2 Análisis de la complejidad ciclomática mediante SonarQube

Para el monitoreo continuo de la calidad técnica se utilizó SonarQube, herramienta de análisis estático que permitió evaluar indicadores clave vinculados a la mantenibilidad y complejidad del software.

El análisis global del proyecto arrojó resultados altamente satisfactorios en términos de calidad estructural. La evaluación indicó calificación A en mantenibilidad, junto con calificaciones A en confiabilidad y seguridad, sin presencia de bugs, vulnerabilidades ni duplicación significativa de código.

Estos resultados evidencian que el sistema presenta una arquitectura clara, bajo nivel de deuda técnica y ausencia de problemas estructurales relevantes. La calificación obtenida respalda que las prácticas adoptadas tuvieron un impacto directo en la calidad del producto final.

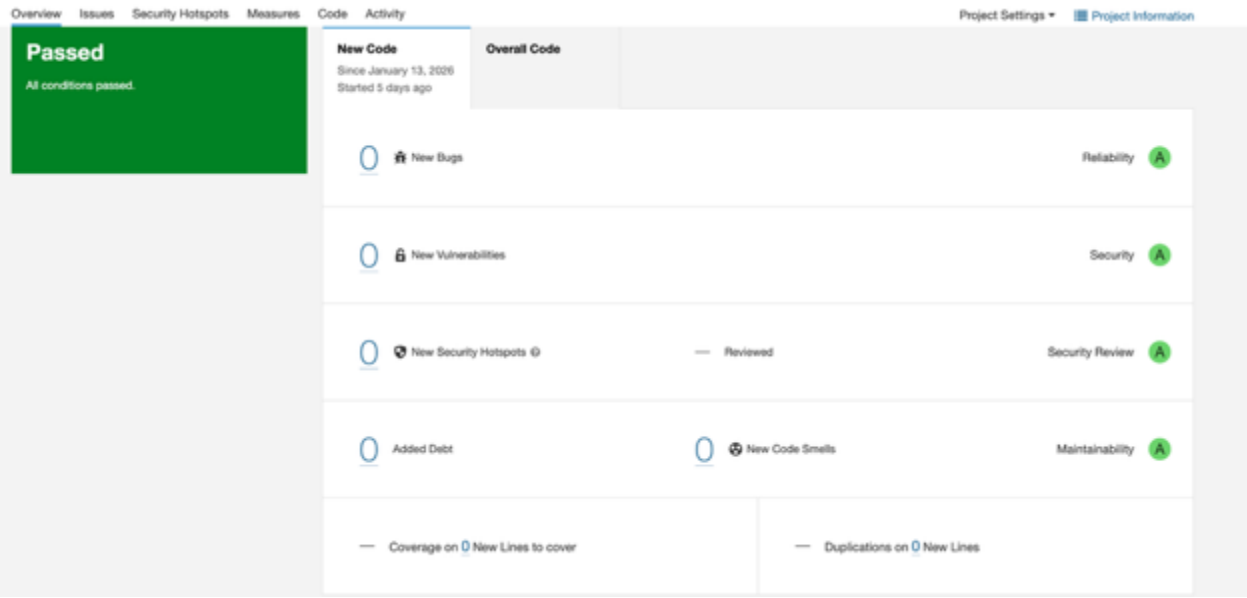


Figura 37: Resultado general del análisis de calidad del proyecto en SonarQube.

Estos resultados se encuentran alineados con las prácticas de aseguramiento de calidad aplicadas durante el desarrollo, particularmente con la estrategia de pruebas automatizadas y el control de complejidad del código descritos en esta sección. El análisis de SonarQube permitió complementar estas prácticas mediante una evaluación automatizada de la calidad estructural del sistema, verificando que la implementación mantiene niveles altos de mantenibilidad.

Se midió la complejidad ciclomática de los módulos críticos del sistema asegurando que los módulos mantuvieran un nivel de ramificación controlado, facilitando su comprensión, prueba y evolución, al mismo tiempo .

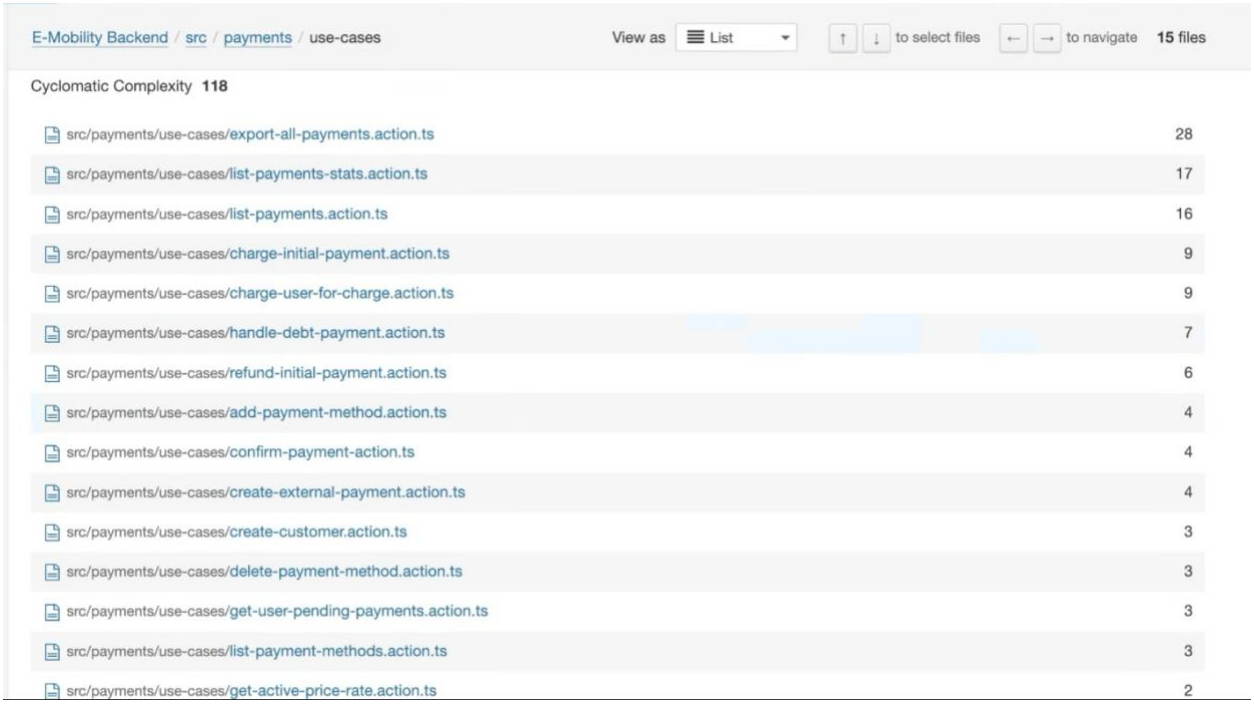


Figura 38: Análisis de complejidad ciclomática de los casos de uso del módulo de Pagos en SonarQube.

El análisis puntual del módulo de pagos refuerza esta tendencia. Como se muestra en la Figura 38, todas las acciones críticas de pago mantienen una complejidad ciclomática menor a 10, lo que refleja una distribución equilibrada de la lógica de negocio y evita concentraciones de decisiones en puntos únicos del sistema.

Este comportamiento es consistente a lo largo de los flujos críticos del sistema, cumpliendo con el escenario de calidad E2 definido en sección 6.2.1 y asegurando que

las funcionalidades esenciales puedan comprenderse, probarse y modificarse con un esfuerzo controlado. El análisis detallado del resto de los componentes críticos puede visualizarse en la sección 12.12 del Anexo.

El monitoreo de estas métricas mediante SonarQube permitió identificar posibles incrementos de complejidad y aplicar refactorizaciones preventivas, garantizando que la mantenibilidad y la estabilidad del sistema no se vean comprometidas a medida que evoluciona.

8.2.2 Prácticas orientadas a la calidad de uso

Esta dimensión se enfocó en asegurar que el sistema no solo funcionara correctamente desde el punto de vista técnico, sino que resultara comprensible, intuitivo y eficiente para los usuarios finales. La calidad en uso implicó validar la experiencia real de interacción con la aplicación, considerando la claridad de los flujos, la facilidad para completar tareas y la adecuación del sistema al contexto de utilización.

8.2.2.1. Pruebas manuales

La validación del sistema desde la perspectiva de uso y funcionalidades se realizó mediante instancias de testing manual. Cada funcionalidad desarrollada pasaba por una columna de *Testing* en el tablero Scrum (ver sección 7.3.1.2 de Gestión de proyecto), donde otro integrante del equipo validaba de forma independiente los flujos principales antes de considerarla finalizada.

Las pruebas manuales se ejecutaron de manera exploratoria, siguiendo un protocolo basado en los criterios de aceptación definidos para cada tarea, que especificaban pasos, acciones del usuario y resultados esperados. Esto permitió reproducir de forma consistente los flujos principales del sistema y validar que su comportamiento cumpliera con los requerimientos funcionales (ver sección 12.6 del Anexo para un ejemplo de tarea y criterios correspondientes).

Cuando se identificaba algún fallo o comportamiento inesperado, la tarea se movía a la columna *Rejected* y luego regresaba a *In Progress* para que la persona que la desarrolló realizara la corrección correspondiente.

La revisión independiente reforzó el control de calidad y aseguró que cada incremento entregado cumpliera con los criterios de aceptación establecidos para el sprint, manteniendo coherencia entre lo diseñado, lo implementado y lo validado.

8.2.2.1.1. Reporte de Incidentes

En aquellos casos donde durante el proceso de prueba se detectaban problemas adicionales no contemplados en la tarea original, estos se registraban como *bugs* en nuevas tareas. Cada *bug* se documentaba indicando su prioridad y nivel de severidad, lo que permitía evaluar su impacto en el sistema y determinar el orden en que debía ser corregido. En la sección 12.13 del Anexo, se presenta un ejemplo de registro de un *bug*. De esta manera, los defectos detectados podrían ser gestionados y priorizados de forma independiente dentro del proceso de desarrollo.

Estas nuevas tareas se trasladaban al *Product Backlog*, a excepción de casos puntuales de alta severidad y alta prioridad, que debieron ser atacados dentro del mismo sprint.

8.2.2.2. Pruebas con usuarios reales

Se realizaron pruebas con usuarios para evaluar la experiencia de uso y la claridad de los flujos principales. Durante estas sesiones se observaron tareas concretas dentro de la aplicación, particularmente relacionadas con el inicio y seguimiento de una sesión de carga.

Se llevaron a cabo pruebas de validación con tres usuarios diferentes, quienes interactuaron con los flujos principales del sistema sin recibir instrucciones adicionales y con autonomía.

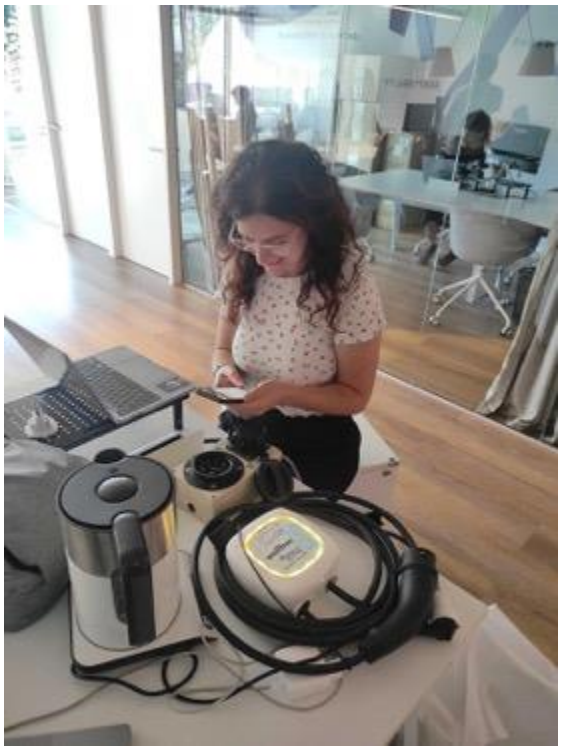


Figura 39: Instancia de un usuario real probando el sistema

La retroalimentación obtenida permitió identificar oportunidades de mejora en usabilidad, claridad de mensajes y simplificación de ciertos pasos del proceso. Estas instancias fueron clave para asegurar que el sistema no solo funcionara correctamente desde el punto de vista técnico, sino que también resultara intuitivo y alineado con las expectativas de los usuarios finales.

No obstante, el equipo reconoce que la muestra utilizada, compuesta por tres participantes, es reducida y no resulta estadísticamente representativa del universo de usuarios objetivo. En consecuencia, se considera que la usabilidad del sistema, y por ende el escenario de calidad E8 definido en la Sección 6.2.5, ha sido validada de forma parcial, dado que los resultados obtenidos permiten detectar problemas relevantes pero no son suficientes para generalizar conclusiones firmes. Como trabajo futuro, se propone continuar realizando pruebas con usuarios y recopilando retroalimentación en la fase de validación operativa, lo cual se detalla en la sección 12.14 del Anexo.

Adicionalmente, se realizaron pruebas de usuario con empleados de E-Mobility Solutions, quienes probaron el *admin dashboard* del sistema. Estas pruebas abarcaron

las principales funcionalidades de gestión del sistema, incluyendo la administración de usuarios, organizaciones, cargadores, sesiones de carga y tarjetas RFID, entre otras. Las sesiones permitieron evaluar la claridad de la interfaz, la comprensión de la información presentada y la eficiencia de los flujos administrativos en el contexto de uso esperado por la organización.

En conjunto, la estrategia de pruebas implementada garantizó un producto robusto, estable y validado tanto técnica como funcionalmente, reduciendo significativamente el riesgo de fallas en entornos reales.

Figura 40: Evidencia de instancia de pruebas del admin dashboard con el cliente

8.2.2.3. Instancias de *Sprint Review*

Como fue mencionado en la Sección 7.3.2 (Ceremonias), cada dos semanas el equipo realizó *Sprint Reviews* con el cliente y, cuando correspondía, con el Product Owner, en las cuales se presentaron los avances desarrollados durante el *sprint*. Estas sesiones

permitieron validar que las funcionalidades cumplieran con los requisitos y expectativas, recibir retroalimentación directa y detectar desviaciones antes de que se convirtieran en problemas mayores, asegurando que el producto final se mantuviera alineado con las necesidades del usuario.

8.2.2.3. Análisis de la plataforma actual y criterios de diseño de interfaz

Desde el punto de vista de la experiencia de usuario, se realizó un análisis de VoltBras, la plataforma actualmente utilizada por el cliente para la gestión de infraestructura de carga eléctrica. Este análisis permitió comprender los flujos de interacción existentes y las convenciones de uso presentes en el sistema previo antes de iniciar el diseño de las nuevas interfaces.

A partir de este estudio se observaron patrones de interacción relevantes para el contexto de uso, particularmente en tareas como la localización de cargadores, el inicio de una sesión de carga y el seguimiento de su estado. Estas observaciones sirvieron como punto de partida para definir criterios de diseño orientados a mejorar la claridad y la simplicidad de los flujos principales del sistema.

Durante las iteraciones de diseño se aplicaron además heurísticas de usabilidad ampliamente reconocidas en el diseño de interfaces, tales como la visibilidad del estado del sistema, la prevención de errores mediante confirmaciones y mensajes claros, la consistencia visual entre pantallas y componentes, y el control y libertad del usuario en los flujos principales. Estas consideraciones aseguraron que las interfaces no solo fueran visualmente atractivas, sino también comprensibles, intuitivas y alineadas con buenas prácticas de experiencia de usuario.

Para el análisis de las interfaces existentes se utilizaron los prototipos y diseños disponibles en Figma[26] proporcionados por el cliente, los cuales permitieron comprender la estructura de navegación y los flujos de interacción presentes en la plataforma actual.

[26] https://figma.com

Estos criterios fueron utilizados posteriormente como base para la generación y evaluación de nuevos prototipos de interfaz, incluyendo propuestas generadas con asistencia de inteligencia artificial, tal como se describe en la Sección 8.3.1.

Este enfoque se encuentra alineado con el escenario de calidad E8 definido en la Sección 6.2.5, el cual establece como objetivo que los usuarios puedan completar los flujos críticos sin asistencia. La aplicación de estas prácticas contribuyó a reducir fricciones en la navegación y facilitar la ejecución autónoma de tareas sensibles como el inicio y finalización de una sesión de carga.

8.2.3 Prácticas orientadas a la calidad de proceso

Esta dimensión se centró en asegurar que la calidad no dependiera exclusivamente de validaciones finales, sino que estuviera integrada en la forma de trabajo del equipo. Siguiendo un enfoque preventivo, se establecieron instancias formales de revisión y mejora continua que permitieron detectar desvíos tempranamente y mantener la alineación con el cliente durante todo el ciclo de desarrollo.

8.2.3.2. Instancias de *Sprint Retrospective*

Tal como fue comentado en la sección 7.3.2 Ceremonias, al cierre de cada *sprint*, se llevaron a cabo *Sprint Retrospectives*, reuniones internas enfocadas en analizar el proceso de desarrollo, identificar oportunidades de mejora y optimizar la colaboración y comunicación del equipo. Estas instancias permitieron ajustar metodologías, refinar prácticas de trabajo y aumentar la eficiencia, contribuyendo indirectamente a la calidad del producto y del código.

8.2.3.3 Gestión de riesgos

La gestión de riesgos formó parte del proceso de seguimiento del proyecto y contribuyó indirectamente al aseguramiento de la calidad, al permitir identificar de forma temprana posibles factores que pudieran afectar el desarrollo del sistema o el cumplimiento de los objetivos definidos.

Para una descripción detallada del proceso de identificación, clasificación y seguimiento de riesgos aplicado durante el proyecto, consultar la sección 7.8 Gestión de riesgos.

8.2.3.4. Satisfacción del cliente

Como complemento a las evaluaciones técnicas del sistema, se consideró la percepción del cliente como un indicador relevante de la calidad del proceso y del producto desarrollado. En este sentido, se recibieron cartas de satisfacción por parte de referentes del cliente, en las cuales se expresa su valoración positiva sobre el trabajo realizado durante el proyecto. Dichas cartas pueden consultarse en la Sección 12.5: Cartas de Satisfacción del cliente.

En estas comunicaciones se destaca la colaboración mantenida a lo largo del desarrollo, la disposición del equipo para comprender las necesidades del negocio y la calidad de la solución implementada. Asimismo, se reconoce que el sistema desarrollado cumple con los objetivos planteados y constituye una base sólida para futuras mejoras y ampliaciones.

Estas instancias de retroalimentación directa por parte de los principales stakeholders permiten documentar el grado de satisfacción alcanzado y constituyen un indicador cualitativo del valor percibido del sistema desarrollado.

Figura 41: Integrantes del equipo junto con los referentes del cliente, Matías y Alfredo

8.3 Calidad incorporada al flujo de trabajo

Desde la perspectiva de la gestión de la calidad, la *Definition of Done*, previamente definida en la sección Gestión de proyecto, actuó como un mecanismo preventivo. Integró estándares de código, pruebas automatizadas, revisión por pares y validación funcional dentro de un único criterio formal de aceptación. De esta manera, la calidad no dependía únicamente de controles posteriores, sino que quedaba incorporada estructuralmente en el flujo de trabajo del equipo.

8.4. Uso de herramientas de Inteligencia Artificial

Las herramientas basadas en inteligencia artificial (IA) pueden acelerar significativamente los procesos de desarrollo de software, permitiendo generar prototipos, explorar alternativas de diseño y producir soluciones preliminares en tiempos reducidos. Sin embargo, su uso también introduce riesgos potenciales para la calidad del

sistema, ya que las propuestas generadas automáticamente pueden contener inconsistencias, decisiones de diseño inapropiadas o soluciones que no se ajusten a los requerimientos específicos del proyecto.

En este contexto, el equipo adoptó un enfoque de uso controlado y supervisado de estas herramientas. La inteligencia artificial fue utilizada como apoyo para agilizar tareas de exploración, generación de propuestas y asistencia al desarrollo, pero nunca como un reemplazo del criterio técnico del equipo. Todas las soluciones sugeridas mediante herramientas de IA fueron evaluadas, revisadas y adaptadas manualmente antes de ser incorporadas al sistema.

Este enfoque permitió aprovechar las ventajas de la IA en términos de productividad y velocidad de iteración, al mismo tiempo que se preservaban atributos clave de calidad como la mantenibilidad, la coherencia arquitectónica y el cumplimiento de los requisitos funcionales y no funcionales del sistema.

8.4.1 Uso de IA en UX/UI

En complemento al proceso de diseño realizado en Figma, se utilizaron herramientas de generación asistida por inteligencia artificial, particularmente v0[27], para explorar variantes de interfaz y generar propuestas iniciales de algunas pantallas. Esta combinación permitió generar variantes de diseño de forma rápida, explorar alternativas visuales y validar decisiones de manera más eficiente.

[27] https://v0.app

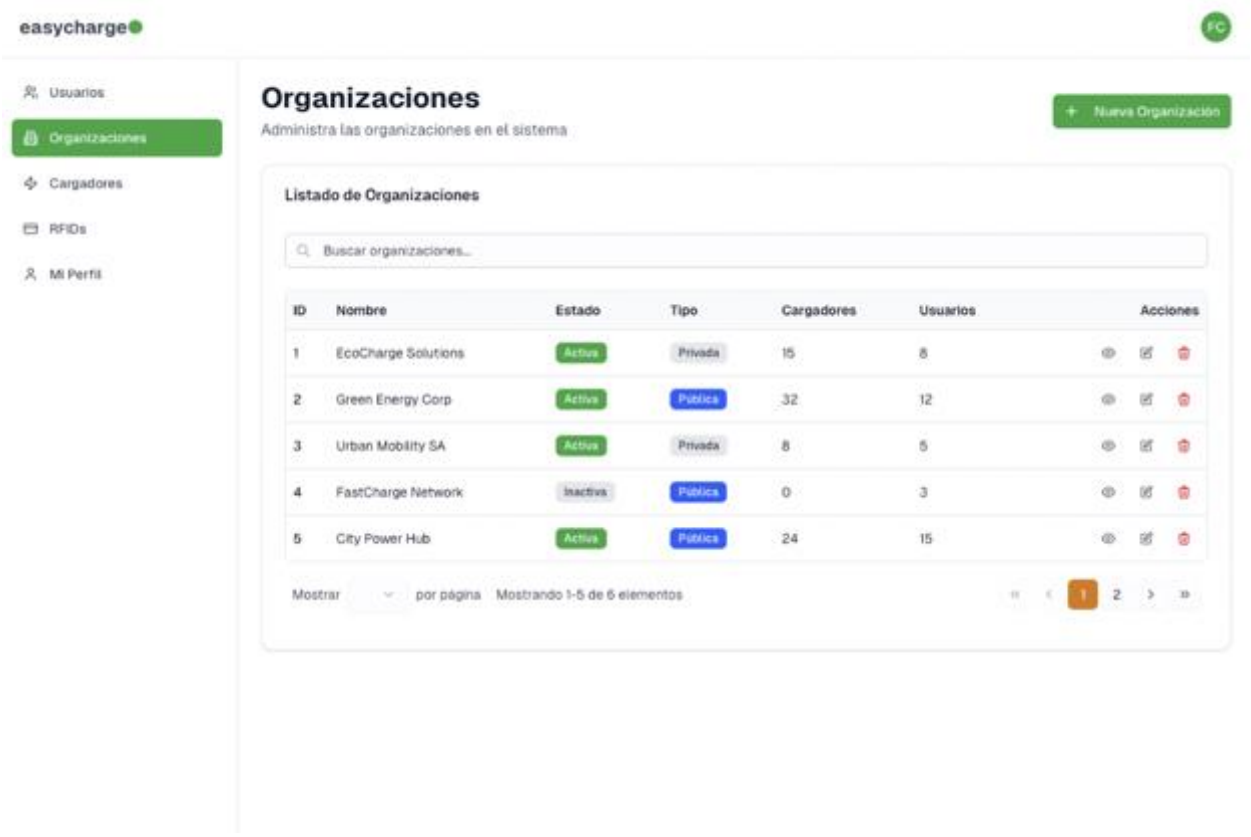
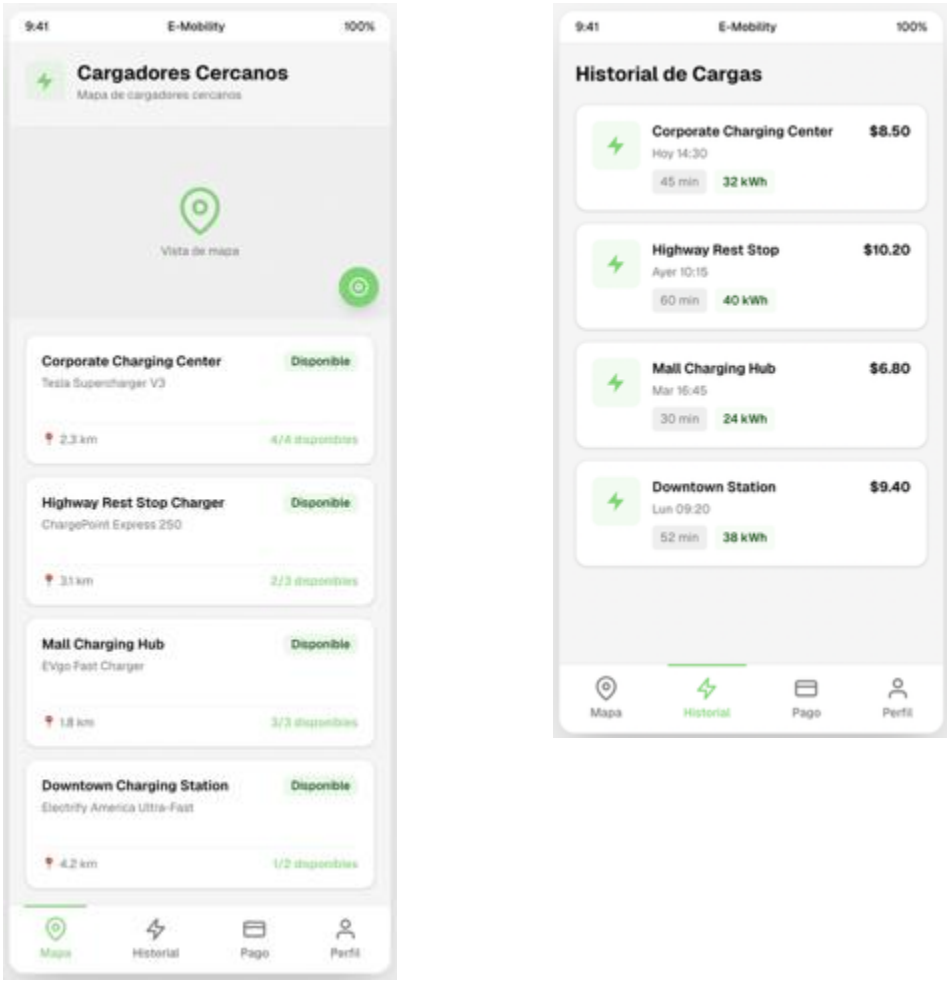


Figura 42: Prototipo inicial del dashboard generado mediante asistencia de IA (v0).



Figuras 43 y 44: Propuestas de interfaz móvil generadas con asistencia de IA

Las propuestas generadas mediante asistencia de inteligencia artificial fueron posteriormente revisadas, refinadas y adaptadas por el equipo, asegurando su coherencia con los requerimientos funcionales y técnicos definidos para el sistema. Ninguna interfaz fue adoptada de manera automática; cada diseño fue evaluado críticamente y ajustado en función de las necesidades reales del proyecto. El proceso de validación se realizó de forma iterativa junto al cliente, lo que permitió contrastar las decisiones de diseño con expectativas concretas de uso y realizar mejoras progresivas antes de la etapa de implementación.

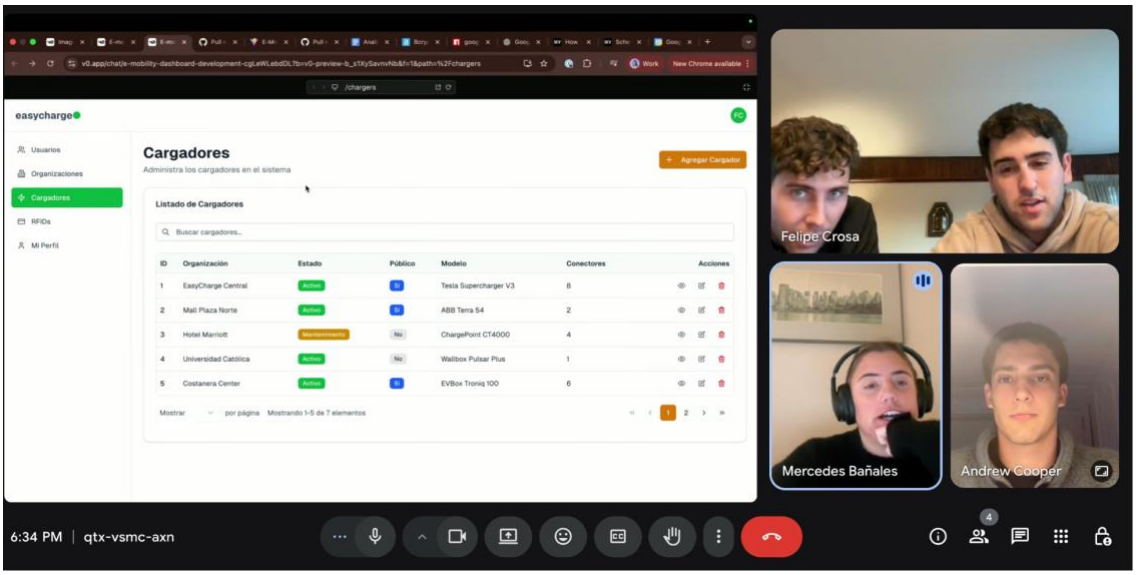


Figura 45: Instancia de evaluación del diseño propuesto por v0 con el cliente

El uso de herramientas de IA en esta etapa permitió detectar inconsistencias de diseño en fases tempranas al generar rápidamente prototipos y variantes de interfaz. Esto facilitó comparar alternativas, identificar diferencias en la disposición de componentes y validar patrones de navegación antes de avanzar a la implementación. Además, redujo retrabajos y aceleró la generación de propuestas visuales para su discusión con el cliente. Al disponer rápidamente de prototipos concretos que materializaban ideas abstractas, se facilitó la comunicación y se mejoró la calidad de las decisiones tomadas. Como resultado, se logró mantener uniformidad visual y coherencia estructural entre los distintos componentes de la interfaz.

De esta manera, la incorporación de inteligencia artificial en el proceso de diseño contribuyó principalmente a fortalecer la dimensión de calidad de uso y la calidad percibida del producto, al mejorar tanto la claridad de interacción como la consistencia general del sistema antes de su desarrollo definitivo.

8.4.2 Uso de IA en el desarrollo de software

Para la implementación se utilizaron herramientas de asistencia basadas en modelos de lenguaje integradas al entorno de desarrollo, tales como Cursor[28], Claude[29] y GitHub Copilot[30]. Estas herramientas funcionaron como apoyo técnico en distintas tareas del proceso de programación.

Su utilización permitió agilizar tareas repetitivas y acelerar la implementación de funcionalidades estándar, optimizando los tiempos de desarrollo. Esto fue posible al emplear estas herramientas principalmente en la generación de código *boilerplate*, estructuras iniciales de funciones, refactorizaciones menores y apoyo en la creación de pruebas unitarias.

Sin embargo, para asegurar que el uso de inteligencia artificial no afecte negativamente la calidad ni la mantenibilidad del sistema, se establecieron criterios claros y consistentes. Todo código generado con asistencia de IA fue revisado manualmente antes de su integración, garantizando que su funcionamiento fuera plenamente comprendido por los desarrolladores. Esta revisión se realizó en el marco del proceso de *Pull Requests* definido para el proyecto, donde cada contribución debía ser revisada en primer lugar por el propio autor del cambio, responsable de su correcta implementación, y posteriormente analizada y aprobada por al menos otro integrante del equipo antes de su incorporación al repositorio principal. Ninguna sugerencia fue incorporada sin un análisis crítico previo, y se exigió que cada aporte respetara estrictamente las convenciones de estilo, los lineamientos arquitectónicos y las decisiones de diseño adoptadas en el proyecto.

Las decisiones de arquitectura y modelado estructural permanecieron exclusivamente bajo responsabilidad del equipo técnico, evitando delegar aspectos estratégicos del diseño a herramientas automatizadas. De esta manera, el uso de herramientas asistidas

[28] https://cursor.com

[29] https://claude.com

[30] https://github.com/features/copilot

por IA se mantuvo alineado con las decisiones arquitectónicas establecidas en la Sección 6.3, evitando introducir dependencias, acoplamientos innecesarios o estructuras que fueran en contra del modelo modular adoptado.

Este enfoque permitió aprovechar la inteligencia artificial como acelerador de productividad sin introducir deuda técnica ni inconsistencias estructurales. Desde la perspectiva del aseguramiento de la calidad, su uso controlado contribuyó a reducir errores sintácticos y omisiones frecuentes, mejorar la claridad de ciertos bloques de código mediante refactorizaciones sugeridas y facilitar la generación de pruebas automatizadas, manteniendo siempre la supervisión técnica del equipo sobre el resultado final.

8.4.2.1 Uso de IA para generación de pruebas unitarias automatizadas

Uno de los principales usos estratégicos de la IA fue la generación preliminar de pruebas unitarias, con el objetivo de alcanzar un alto porcentaje de cobertura en los flujos críticos del sistema. Dado que dichas funcionalidades ya se encontraban validadas manualmente durante el desarrollo, la generación asistida de tests permitió formalizar el comportamiento esperado y asegurar que, ante futuras modificaciones, cualquier regresión pudiera detectarse tempranamente mediante la suite automatizada, reforzando así los criterios de mantenibilidad definidos en la arquitectura.

8.5. Conclusiones y lecciones aprendidas

La calidad en EasyCharge fue abordada como un proceso transversal al desarrollo, integrando estándares de código, revisiones, pruebas automatizadas y validaciones con usuarios. Esto permitió construir un sistema robusto tanto desde el punto de vista técnico como funcional.

Las métricas obtenidas evidenciaron baja complejidad estructural, alta mantenibilidad y elevados niveles de cobertura de pruebas, lo que redujo el riesgo de errores y facilita la evolución del sistema. Asimismo, las instancias de validación realizadas durante el proyecto permitieron verificar el correcto comportamiento del sistema respecto a otros

atributos relevantes, como la disponibilidad, la interoperabilidad y la portabilidad. Estos resultados constituyen evidencia concreta del cumplimiento de los escenarios de calidad definidos, confirmando que la arquitectura propuesta logró materializar en la implementación los atributos técnicos priorizados en su diseño.

Un aspecto relevante observado durante el proyecto fue la importancia de mantener criterios estrictos de calidad al utilizar herramientas basadas en inteligencia artificial como apoyo al desarrollo. Si bien estas herramientas permiten acelerar tareas como la generación de prototipos, código o pruebas automatizadas, su utilización requiere una revisión crítica constante para evitar introducir inconsistencias, decisiones de diseño inapropiadas o soluciones que no se ajusten a los requerimientos del sistema.

Como principales lecciones aprendidas, se destaca la importancia de integrar la calidad desde el inicio, respaldarla con métricas objetivas y apoyarse en la automatización para sostenerla en el tiempo. Esta combinación permitió entregar un producto mantenible, estable y alineado con las expectativas del cliente.

9. Gestión de la configuración

La gestión de la configuración en el proyecto tuvo como objetivo asegurar la trazabilidad, coherencia y control de los distintos artefactos generados durante el desarrollo del sistema de gestión de cargadores eléctricos.

Dado que el sistema involucra múltiples componentes (*backend*, *frontend web*, aplicación móvil y servicio de WebSockets), así como documentación técnica y funcional asociada, fue fundamental definir mecanismos claros para identificar y versionar los cambios realizados sobre cada uno de estos elementos.

La correcta aplicación de estas prácticas permitió mantener coherencia entre código y documentación, reducir riesgos y facilitar el trabajo colaborativo, asegurando estabilidad en las versiones del sistema a lo largo del desarrollo.

9.1. Identificación de elementos de configuración

Se identificaron como elementos de configuración todos aquellos artefactos cuyo cambio puede impactar el comportamiento, arquitectura o documentación del sistema. Entre ellos:

- Código fuente de los distintos componentes del sistema: backend (API principal), frontend web (panel administrativo), aplicación móvil, servicio de webSockets
- Código fuente de la prueba de concepto de OCPP
- Archivos de configuración de entorno (.env)
- Estructura de base de datos y migraciones
- Documentación técnica y funcional
- *Backlog* de historias de usuario
- Matriz de riesgos
- Definición de arquitectura

Cada uno de estos elementos fue versionado o almacenado en herramientas que permiten trazabilidad y control de cambios.

9.2. Herramientas utilizadas

Para gestionar los elementos de configuración identificados, el equipo seleccionó diversas herramientas que permiten centralizar información, mantener historial de cambios y facilitar la colaboración.

9.2.1 GitHub

Se utilizó GitHub[31] como plataforma principal para el versionado del código fuente. A través de esta herramienta se gestionaron:

- Repositorios independientes por componente
- *Pull Requests* con revisión de código
- Historial completo de cambios
- Control de versiones por ramas
- Integraciones con automatizaciones

El uso de GitHub permitió mantener un registro auditable de cada modificación realizada en el sistema, asociando cambios a tickets específicos y responsables.

9.2.2 ClickUp

Para la gestión del trabajo y trazabilidad funcional se utilizó ClickUp. En esta herramienta se administraron:

- Historias de usuario

[31] https://github.com/

- Incidentes
- Tareas técnicas
- *Backlog*
- Estados de avance

9.2.3 Google Drive

Para la documentación formal y entregables académicos se utilizó Google Drive. Esta herramienta permitió:

- Trabajo colaborativo en documentos
- Control de versiones automático
- Almacenamiento de imágenes y diagramas
- Gestión de presentaciones de avance
- Respaldo centralizado de información relevante

Posteriormente, la documentación final fue consolidada en el formato requerido para la entrega del proyecto.

9.3. Organización de los repositorios

El proyecto fue estructurado en repositorios independientes, siguiendo un criterio de separación por responsabilidades técnicas y dominio funcional. Cada componente del sistema, tal como se describe en la sección Arquitectura y diseño, cuenta con su propio repositorio.

Esta organización permitió que cada componente tuviera su propio ciclo de desarrollo y versionado, reduciendo el riesgo de cambios cruzados y manteniendo mayor control sobre la evolución del sistema. Esta decisión se alinea con los principios propuestos por

la metodología The Twelve-Factor App [33], referencia ampliamente aceptada en la industria para aplicaciones *cloud*.

En este sentido, la estructura de repositorios refleja directamente la arquitectura definida:

- **emobility-backend** → Servicio API principal y lógica de negocio.
- **emobility-frontend** → Panel administrativo web.
- **emobility-mobile** → Aplicación móvil para usuarios.
- **emobility-websockets** → Servicio de comunicación en tiempo real.

También se crearon repositorios para herramientas auxiliares que asistieron en la construcción de la solución.

- **ocpp-poc** → Prueba de concepto para la comunicación con cargadores mediante OCPP.

9.4. Gestión de versiones

La gestión de versiones tuvo como objetivo asegurar estabilidad, trazabilidad y control sobre la evolución de los distintos componentes del sistema. Dado que el proyecto se estructuró en múltiples repositorios independientes, fue necesario definir una estrategia de versionado clara y consistente que permitiera organizar el desarrollo y mantener versiones estables del producto.

Para ello, se estableció un esquema formal de ramificación y control de cambios, adaptado a las características de cada componente del sistema. Esta estrategia permitió ordenar el ciclo de vida del código, facilitar la integración de nuevas funcionalidades y mantener una separación clara entre versiones en desarrollo y versiones estables.

En las subsecciones siguientes se detalla el enfoque adoptado en cada caso.

9.4.1 Gitflow

El modelo Gitflow [34] fue adoptado como estrategia principal de versionado en el proyecto, estableciendo una estructura clara para la gestión del ciclo de vida del código en la mayoría de los repositorios. Este ofrece una estructura clara para la gestión de versiones y liberaciones mediante el uso de ramas dedicadas.

Este modelo permitió organizar el desarrollo mediante ramas específicas para nuevas funcionalidades y correcciones, manteniendo una separación clara entre código en evolución y versiones estables del sistema.

La rama *main* fue definida como la referencia estable del sistema. Esta rama concentraba únicamente aquellas funcionalidades que habían sido integradas, revisadas y validadas por el equipo, representando en todo momento la versión consolidada y funcional del producto.

Por su parte, la rama *develop* funcionó como espacio de integración de nuevas funcionalidades antes de su consolidación definitiva. Las implementaciones se desarrollaban en ramas de tipo *feature*, creadas a partir de *develop*, mientras que las correcciones de errores se gestionaban mediante ramas de tipo *fix*.

La integración de cada rama se realizaba mediante *Pull Requests*, los cuales eran sometidos a revisión antes de su aprobación. Una vez validados, los cambios se incorporaban a *develop* y, posteriormente, eran consolidados en *main* cuando alcanzaban un estado de estabilidad adecuado. Este proceso de revisión mediante *Pull Requests*, tal como se detalla en la sección Gestión de la calidad, permitió validar los cambios antes de su integración y mantener control sobre el código incorporado.

Esta estructura permitió mantener una separación clara entre código en evolución y código estable, reduciendo riesgos asociados a integraciones desordenadas y facilitando el control de versiones a lo largo del proyecto.

9.4.2 *Trunk-Based Development*

Si bien el modelo formal adoptado para la gestión de versiones fue Gitflow, en el desarrollo del servidor de *websockets* se aplicaron prácticas asociadas a *Trunk-Based Development*. Este enfoque promueve la integración frecuente de cambios en una única rama principal compartida [35]. Dado que se trataba de un componente acotado y de baja complejidad, se optó por un esquema de integración más directo y frecuente, evitando la sobrecarga de un flujo de ramas más estructurado.

En particular, para este módulo se procuró que las ramas de tipo *feature* y *fix* tuvieran una duración muy acotada, evitando que permanecieran activas por períodos prolongados. La integración hacia la rama *main*, utilizada como trunk en este contexto, se realizaba de forma directa y frecuente, priorizando commits pequeños y continuos una vez que la funcionalidad alcanzaba un estado estable y validado.

Este enfoque resultó adecuado dado el tamaño y simplicidad del componente. Al tratarse de un servicio acotado y con bajo nivel de complejidad, las integraciones frecuentes permitieron mantener el código actualizado sin necesidad de estructuras de ramificación más formales. Esto facilitó su mantenimiento y permitió realizar ajustes de manera rápida y controlada.

9.5. Conclusiones y lecciones aprendidas

La gestión de la configuración fue importante para mantener orden, control y trazabilidad durante todo el desarrollo del sistema. Definir claramente qué elementos debían versionarse y utilizar herramientas adecuadas permitió trabajar de forma organizada y reducir errores.

La estrategia de versiones basada en Gitflow, junto con el uso de prácticas de *Trunk-Based Development* en el servicio E-Mobility WebSockets, ayudó a mantener un equilibrio entre control y rapidez en la integración de cambios. Esto permitió adaptarse a las necesidades de cada componente sin perder estabilidad.

En general, tanto la gestión de versiones como la gestión de cambios contribuyeron a que las modificaciones se realizaran de forma ordenada y planificada. Esto facilitó el trabajo en equipo, redujo conflictos en el código y permitió que el sistema evolucionara de manera controlada a lo largo del proyecto.

10. Conclusiones

La siguiente sección busca dar cierre al proyecto, sintetizando los principales resultados obtenidos y las lecciones aprendidas a lo largo de su desarrollo. Propone una reflexión sobre el proceso recorrido, marcando el cierre de esta etapa, no solo del proyecto, sino también de nuestra etapa como estudiantes de la carrera de Ingeniería de Sistemas.

10.1. Estado Actual

Finalizado el proyecto, la plataforma ha alcanzado el nivel de prototipo funcional, capaz de cubrir las principales funcionalidades definidas y lista para iniciar una etapa de validación operativa en un entorno productivo.

Se ha realizado la entrega formal de todo el código fuente junto con la documentación elaborada durante el proyecto asegurando que el cliente cuente con los insumos necesarios para dar continuidad al desarrollo y eventual puesta en producción de la solución.

Documentación técnica:

- Documento de arquitectura y diseño (sección 6)
- Documento de modelado de datos (sección 12.8 del Anexo)
- Documento de guía de despliegue (sección 12.15 del Anexo)
- Documentacion de endpoints mediante coleccion de Postman (sección 12.18 del Anexo)

Documentación operativa:

- Sección Requerimientos no implementados
- Sección Plan de validación operativa

El proyecto deja una base sólida sobre la cual continuar evolucionando el producto. La valoración positiva del resultado por parte del cliente refuerza tanto la validez de las decisiones técnicas adoptadas como la efectividad de la forma de trabajo y de la comunicación sostenida durante el proceso.

10.2. Reflexiones Generales

Todos los integrantes del equipo contábamos con experiencia previa en el desarrollo de proyectos de software gracias a nuestros roles profesionales. Sin embargo, dicha experiencia no fue comparable con lo que implicó este proyecto. A lo largo de este proceso tuvimos la oportunidad de involucrarnos activamente en todas las áreas de la ingeniería de software, algo que, debido a las estructuras jerárquicas y a la especialización de roles presentes en el ámbito laboral, no suele ocurrir en la práctica profesional.

La exposición a estas situaciones evidenció, en algunos casos, un recuerdo limitado de ciertos conceptos abordados en materias previamente cursadas, pero al mismo tiempo abrió nuevamente la puerta al aprendizaje y al crecimiento profesional.

En el contexto actual, donde el desarrollo de herramientas basadas en inteligencia artificial ha incrementado significativamente la productividad en la escritura de código, el valor diferencial de los profesionales del software se vuelve cada vez más evidente en otras áreas de la ingeniería.

Los aprendizajes y experiencias obtenidos a lo largo de este proyecto contribuyeron significativamente a fortalecer estas capacidades y a consolidar nuestro crecimiento como profesionales de la ingeniería de software.

10.3. Principales Lecciones Aprendidas

Detenerse a reflexionar sobre el proceso vivido a lo largo del último año permite reconocer todo lo aprendido y cómo cada área de la ingeniería de software aportó a la consolidación del proyecto. A continuación, se resumen las principales lecciones aprendidas durante el ciclo de vida del proyecto.

10.3.1. Ingeniería de Requerimientos

Una de las principales lecciones aprendidas fue la importancia de la ingeniería de requerimientos, especialmente al enfrentarse a un dominio nuevo para el equipo, como

lo es el de la electromovilidad. El equipo comenzó el proyecto sin conocimientos previos en este dominio, un sector en constante crecimiento pero altamente complejo por la multiplicidad de actores e interdependencias.

A través de un proceso sistemático de relevamiento, análisis y validación de requerimientos, el equipo logró comprender progresivamente el dominio del problema y las necesidades reales de E-Mobility Solutions. Este proceso permitió reducir ambigüedades, alinear expectativas y establecer una base sólida para las etapas posteriores del desarrollo.

10.3.2. Arquitectura y Diseño

Desde el punto de vista arquitectónico, se reafirmó el valor de diseñar la estructura del sistema a partir de los requerimientos no funcionales, de modo que las funcionalidades agreguen el valor buscado y la solución cumpla con los objetivos. Este enfoque permitió alinear las decisiones de diseño con atributos de calidad como mantenibilidad, disponibilidad e interoperabilidad, evitando re-trabajos y facilitando la evolución futura del sistema.

Durante este proceso, el equipo atravesó un marcado crecimiento técnico, incorporando nuevas tecnologías y profundizando en conceptos avanzados de ingeniería de software. El uso de React Native para el desarrollo móvil y el estudio del protocolo OCPP representaron desafíos que exigieron investigación, experimentación y adaptación, fortaleciendo las capacidades individuales y colectivas del equipo para enfrentar proyectos en el futuro.

Asimismo, trabajar sobre un sistema distribuido evidenció su complejidad inherente. Procesos que en apariencia parecían simples involucraron múltiples interacciones, servicios y puntos de falla que debieron anticiparse y gestionarse cuidadosamente. Esta experiencia reforzó la necesidad de contar con una arquitectura robusta, con responsabilidades bien definidas y decisiones de diseño justificadas, que reduzca el riesgo de fallas y facilite el diagnóstico y la corrección de problemas.

10.3.3. Gestión del Proyecto

A lo largo del proyecto se comprobó que no existe una única metodología adecuada para todos los contextos, sino que resulta necesario adaptar el enfoque de trabajo según la etapa y las necesidades del equipo. En particular, si bien Scrum brindó una estructura clara para la planificación y el seguimiento, en otros momentos fue conveniente apoyarse en prácticas de Kanban para gestionar de forma más flexible el flujo de trabajo y las prioridades, evitando rigideces innecesarias.

También se evidenció que, incluso tratándose de un equipo reducido y conformado por amigos, es fundamental reservar espacios de retrospectiva para reflexionar sobre el proceso, identificar mejoras y trabajar por la satisfacción de todos los integrantes. En este marco, la definición explícita de roles y responsabilidades fue clave para gestionar la alta carga de trabajo, prevenir sobrecargas individuales y asegurar que cada tarea tuviera un responsable claro.

Otro aspecto central fue la importancia de contar con visibilidad constante sobre el progreso del proyecto mediante métricas. El uso sistemático de indicadores permitió entender mejor la capacidad real del equipo, realizar compromisos más realistas y gestionar las expectativas del cliente de forma transparente, reduciendo desalineaciones y replanificaciones de último momento.

Finalmente, la gestión de riesgos se consolidó como una práctica imprescindible a lo largo de todo el ciclo de vida del proyecto. Mantener los riesgos identificados, analizados y monitoreados en forma continua ayudó a anticipar posibles retrasos y problemas, habilitando acciones preventivas y correctivas que contribuyeron a un desarrollo más controlado y predecible.

10.3.4. Gestión de la calidad

La experiencia también dejó en claro que la calidad del software trasciende el cumplimiento funcional. No se limita a que el código funcione, sino que abarca aspectos como la mantenibilidad, la documentación, la experiencia del usuario y la eficiencia de

los procesos. Incorporar la calidad como un eje transversal del desarrollo permitió obtener resultados más confiables y sostenibles en el tiempo.

Al mismo tiempo, el avance de la inteligencia artificial plantea un nuevo escenario. Si bien permite agilizar ciertas etapas del desarrollo, también refuerza la importancia del rol de la gestión de la calidad. Resulta esencial asegurar que la velocidad y automatización que trae consigo la IA no comprometan la solidez ni la confiabilidad del producto final.

10.4. Evaluación de los Objetivos

Al inicio del proyecto se definieron una serie de objetivos que guiaron el trabajo y sirvieron como referencia para la toma de decisiones. Al finalizar el proceso, resulta importante revisar estos objetivos y evaluar en qué medida se han cumplido y qué prácticas contribuyeron a su cumplimiento.

10.4.1. Objetivos de Producto

10.4.1.1. Valor agregado

El objetivo consistía en desarrollar un prototipo que respondiera a las necesidades de E-Mobility Solutions.

A lo largo del proyecto se definieron y priorizaron requerimientos funcionales y no funcionales, que fueron validados regularmente con el cliente. Como resultado, el prototipo permite gestionar la red de cargadores y los flujos de carga de forma alineada con el negocio y establece una base propia sobre la cual evolucionar la solución. En función de estos elementos, el objetivo se considera cumplido.

10.4.1.2. Experiencia del usuario

El objetivo de experiencia de usuario buscaba ofrecer una interacción clara y confiable para los distintos actores del sistema.

Se trabajó en el diseño de flujos simples, en la implementación de heurísticas de usabilidad, en la realización de pruebas con usuarios y en revisiones iterativas con el

cliente, ajustando la interfaz a partir del *feedback* recibido. Como resultado, el objetivo se considera parcialmente cumplido. Se logró una experiencia adecuada al contexto del prototipo, pero aún quedan pendientes validaciones en un entorno productivo con una base de usuarios más amplia.

10.4.1.3. Preparación para la evolución futura

Este objetivo apuntaba a que la solución pudiera evolucionar y escalarse con un esfuerzo razonable.

Para la solución se definió una arquitectura modular, se controló la complejidad de los componentes, se incorporó una alta cobertura de pruebas y se generó documentación técnica que facilite su extensión futura. En base a esto, el objetivo se considera cumplido a nivel de prototipo, ya que la solución presenta una base técnica ordenada y documentada que habilita su evolución.

10.4.2. Objetivos de Proyecto

10.4.2.1. Satisfacción del cliente

El objetivo de satisfacción del cliente buscaba que el resultado y la forma de trabajo fueran percibidos como valiosos por el cliente.

Se promovió su cumplimiento mediante reuniones periódicas, demostraciones de avance, gestión explícita de cambios de alcance y la incorporación sistemática del *feedback*, utilizando estas instancias como indicadores de alineación y satisfacción. La manifestación de conformidad del cliente tanto con el prototipo obtenido como con la dinámica de trabajo mantenida durante el proyecto (evidencia en la sección 12.17 del Anexo) refleja el fruto de estas decisiones y el cumplimiento del objetivo.

10.4.2.2. Gestión y control del proyecto

Este objetivo apuntaba a mantener el proyecto organizado y bajo control en términos de alcance, tiempos y seguimiento.

Para favorecerlo se aplicaron prácticas ágiles, se utilizaron tableros para dar visibilidad al trabajo, se planificaron sprints, se monitorearon métricas de avance y se registraron riesgos y acciones de mitigación, tomando estos elementos como mecanismos de medición del control del proyecto. Como resultado, el objetivo se considera cumplido, ya que el proyecto se desarrolló con un seguimiento claro de tareas e hitos y sin desvíos críticos no gestionados.

10.4.2.3. Satisfacción del equipo

El objetivo de satisfacción del equipo buscaba que los integrantes mantuvieran motivación, compromiso y una percepción positiva del proceso.

Mediante las retrospectivas, la distribución equilibrada de la carga de trabajo y la asignación de roles se logró mantener un proceso de trabajo cómodo para todos. Esto, en conjunto con las oportunidades de aprendizaje que se presentaron, mantuvo al equipo motivado durante el proyecto y derivó en una valoración positiva en su cierre.

10.4.3. Objetivos Académicos

10.4.3.1. Aplicación de conocimientos adquiridos en la carrera

Este objetivo buscaba aplicar de forma integrada los conocimientos de la carrera en un proyecto real con cliente.

El objetivo se considera cumplido. Como se destacó en las reflexiones anteriores, el proyecto dio lugar a aplicar y recordar conocimientos adquiridos a lo largo de la carrera en las diversas áreas de la Ingeniería de Software.

10.4.3.2. Aprendizaje de nuevas tecnologías

El objetivo de aprendizaje de nuevas tecnologías buscaba incorporar herramientas y marcos no utilizados previamente por el equipo.

Se considera cumplido, dado que tecnologías previamente desconocidas, como Mercado Pago y la comunicación OCPP, fueron integradas en flujos críticos y se consolidó conocimiento práctico sobre su uso.

10.4.3.3. Resolución de problemas en áreas desconocidas

Este objetivo apuntaba a poder abordar un dominio de negocio inicialmente desconocido.

El proyecto se desarrolló en el contexto de la electromovilidad, un dominio totalmente nuevo para los integrantes del equipo. Se realizaron actividades de investigación, reuniones con el cliente para comprender sus procesos y restricciones, y experimentos técnicos. Como resultado, el objetivo se considera cumplido, ya que el equipo alcanzó un nivel de comprensión suficiente del mismo para diseñar e implementar un prototipo viable.

10.4.3.4. Aprobación con excelencia académica

El objetivo de aprobación con excelencia académica buscaba alcanzar una calificación mayor a 90% por parte del tribunal.

Al momento de redactar esta sección, este objetivo se encuentra pendiente de validación, ya que depende de la calificación final otorgada por el tribunal académico. No obstante, se trabajó con este objetivo como guía, apuntando a altos estándares de calidad técnica, documental y de presentación del proyecto.

10.4.4. Evaluación

En conjunto, los objetivos definidos se cumplieron a nivel de producto, proyecto y académicos, alcanzando una integración equilibrada entre valor técnico, gestión efectiva y aprendizaje académico.

10.5. Próximos pasos

Concluido el proyecto y realizada la entrega formal del código y la documentación el equipo queda a disposición de E-Mobility Solutions para atender consultas relacionadas con la solución desarrollada. Esto incluye aclaraciones sobre el diseño, el funcionamiento de los distintos componentes y los documentos entregados, de modo de facilitar una eventual continuidad del desarrollo o una futura puesta en producción por parte del cliente.

Al mismo tiempo, este proyecto representa para el equipo el cierre de una etapa académica y el cumplimiento de un objetivo central: recibirse como Ingenieros en Sistemas. Con la experiencia adquirida, la intención es proyectar estos aprendizajes hacia nuevos contextos, aplicando los conocimientos técnicos y de gestión desarrollados aquí en otros proyectos, ya sea en el ámbito profesional, emprendedor o de investigación.

11. Referencias bibliográficas

[1] El País, "Venta de vehículos eléctricos se cuadruplicó en enero de 2026, ¿cuáles fueron las unidades que marcaron este crecimiento?," elpais.com.uy, 23 de febrero de 2026. [En línea]. Disponible en: https://www.elpais.com.uy/negocios/noticias/la-venta-de-vehiculos-electricos-en-2025-crecio [Accedido: 7-feb-2026].

[2] Asana, "El triángulo de la gestión de proyectos: qué es y cómo usarlo," Asana.com. [En línea]. Disponible en: https://asana.com/es/resources/project-management-triangle [Accedido: 10-feb-2026].

[3] Atlassian, "¿Qué es la metodología kanban?," Atlassian.com. [En línea]. Disponible en: https://www.atlassian.com/agile/kanban [Accedido: 28-feb-2026].

[4] Scrum.org, "Home: The Home of Scrum," Scrum.org. [En línea]. Disponible en: https://www.scrum.org/ [Accedido: 28-feb-2026].

[5] Fing, "Ingeniería de Software: Tabla Comparativa Modelos," Facultad de Ingeniería, Universidad de la República, Montevideo, Uruguay, Material de curso, s.f. [En línea]. Disponible en: https://www.fing.edu.uy/tecnoinf/mvd/cursos/ingsoft/material/teorico/is02b-Tabla%20Comparativa%20Modelos.pdf [Accedido: 2-feb-2026].

[6] ISO/IEC/IEEE International Standard - Systems and software engineering -- Life cycle processes -- Requirements engineering, ISO/IEC/IEEE 29148:2018(E), pp. 1-104, nov. 2018. doi: 10.1109/IEEESTD.2018.8559686.

[7] L. R. Vijayasarathy y D. Turk, "Agile software development: A survey of early adopters," Softw. Process: Improv. Pract., vol. 13, no. 4, pp. 327-340, jul. 2008. doi: 10.1002/spip.293.

[8] A. Cockburn et al., “Manifesto for Agile Software Development,” Agile Alliance, 2001. [Online]. Available: https://agilemanifesto.org/ [Accessed: Mar. 9, 2026].

[9] ISO25000.com, "ISO/IEC 25010," ISO25000.com. [En línea]. Disponible en: https://iso25000.com/index.php/en/iso-25000-standards/iso-25010 [Accedido: 15-jul-2025].

[10] L. Bass, P. Clements, y R. Kazman, *Software Architecture in Practice*, 4ª ed. Boston, MA, EE. UU.: Addison-Wesley, 2021.

[11] T. J. McCabe, "A Complexity Measure," *IEEE Trans. Softw. Eng.*, vol. SE-2, no. 4, pp. 308-320, dic. 1976. doi: 10.1109/TSE.1976.233837.

[12] Statcounter Global Stats, "Mobile Operating System Market Share Uruguay," gs.statcounter.com. [En línea]. Disponible en: https://gs.statcounter.com/os-market-share/mobile/uruguay [Accedido: 10-feb-2026].

[13] Software Engineering Institute, "Views and Beyond Collection," SEI Digital Library. [En línea]. Disponible en: https://www.sei.cmu.edu/library/views-and-beyond-collection/ [Accedido: 7-feb-2026].

[14] M. Fowler, "Domain Driven Design," MartinFowler.com, 22 de abril de 2020. [En línea]. Disponible en: https://martinfowler.com/bliki/DomainDrivenDesign.html [Accedido: 15-feb-2026].

[15] Microsoft, "Use domain analysis to model microservices," Azure Architecture Center. [En línea]. Disponible en: https://learn.microsoft.com/en-us/azure/architecture/microservices/model/domain-analysis [Accedido: 3-mar-2026].

[16] S. Oloruntoba, A. S. Walia, y M. Kurup, "SOLID Design Principles Explained: Building Better Software Architecture," DigitalOcean Community, 12 de junio de 2025. [En línea]. Disponible en: https://www.digitalocean.com/community/conceptual-articles/s-o-l-i-d-the-first-five-principles-of-object-oriented-design [Accedido: 7-mar-2026].

[17] Amazon Web Services (AWS), "¿Qué es el mapeo objeto-relacional (ORM)?," AWS. [En línea]. Disponible en: https://aws.amazon.com/what-is/object-relational-mapping/ [Accedido: 7-mar-2026].

[18] Amazon Web Services (AWS), "Acuerdos de nivel de servicio (SLA) de AWS," AWS Legal. [En línea]. Disponible en: https://aws.amazon.com/es/legal/service-level-agreements/ [Accedido: 4-mar-2026].

[19] JetBrains, "The State of Developer Ecosystem 2025," JetBrains.com, 2025. [En línea]. Disponible en: https://devecosystem-2025.jetbrains.com/ [Accedido: 20-feb-2026].

[20] State of JS, "The State of JavaScript," StateOfJS.com. [En línea]. Disponible en: https://stateofjs.com/ [Accedido: 20-feb-2026].

[21] D. Molina, "¿Qué es Work in Progress (WIP) en Kanban y cómo utilizarlo?," IEBS Business School, 16 de febrero de 2022. [En línea]. Disponible en: https://www.iebschool.com/hub/que-es-work-in-progress-wip-en-kanban-y-como-utilizarlo-agile-scrum/ [Accedido: 7-mar-2026].

[22] Scrum.org, "What is a Product Backlog?," Scrum.org Resources. [En línea]. Disponible en: https://www.scrum.org/resources/what-is-a-product-backlog [Accedido: 8-mar-2026].

[23] Scrum.org, "What is a Sprint Backlog?," Scrum.org Resources. [En línea]. Disponible en: https://www.scrum.org/resources/what-is-a-sprint-backlog [Accedido: 8-mar-2026].

[24] Scrum.org, "What is Sprint Planning?," Scrum.org Resources. [En línea]. Disponible en: https://www.scrum.org/resources/what-is-sprint-planning [Accedido: 8-mar-2026].

[25] Scrum.org, "Why do we use Story Points for Estimating?," Scrum.org Blog. [En línea]. Disponible en: https://www.scrum.org/resources/blog/why-do-we-use-story-points-estimating [Accedido: 6-mar-2026].

[26] S. Laoyan, "Planning poker: la estrategia integral para la estimación ágil," Asana, 23 de enero de 2026. [En línea]. Disponible en: https://asana.com/es/resources/planning-poker [Accedido: 6-mar-2026].

[27] Scrum Alliance, "A Guide to Using the Fibonacci Sequence in Scrum," ScrumAlliance.org. [En línea]. Disponible en:

https://resources.scrumalliance.org/Article/guide-using-fibonacci-sequence-scrum [Accedido: 6-mar-2026].

[28] Scrum.org, "What is a Daily Scrum?," Scrum.org Resources. [En línea]. Disponible en: https://www.scrum.org/resources/what-is-a-daily-scrum [Accedido: 8-mar-2026].

[29] Scrum.org, "What is a Sprint Review?," Scrum.org Resources. [En línea]. Disponible en: https://www.scrum.org/resources/what-is-a-sprint-review [Accedido: 8-mar-2026].

[30] Scrum.org, "What is a Sprint Retrospective?," Scrum.org Resources. [En línea]. Disponible en: https://www.scrum.org/resources/what-is-a-sprint-retrospective [Accedido: 8-mar-2026].

[31] TeamRetro, "DAKI (Drop, Add, Keep, Improve) retrospective," TeamRetro.com. [En línea]. Disponible en: https://www.teamretro.com/retrospectives/daki-retrospective/ [Accedido: 2-mar-2026].

[32] S. Monroy, "¿Cuáles son los roles de la metodología Scrum?," Asociación para el Progreso de la Dirección (APD), 14 de diciembre de 2021. [En línea]. Disponible en: https://www.apd.es/roles-metodologia-scrum/ [Accedido: 8-mar-2026].

[33] A. Wiggins, "The Twelve-Factor App," 12factor.net, 2017. [En línea]. Disponible en: https://12factor.net/es/ [Accedido: 22-feb-2026].

[34] V. Driessen, "A successful Git branching model," nvie.com, 5 de enero de 2010. [En línea]. Disponible en: https://nvie.com/posts/a-successful-git-branching-model/ [Accedido: 8-mar-2026].

[35] J. Humble y D. Farley, *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*. Upper Saddle River, NJ, USA: Addison-Wesley, 2011.

12. Anexos

12.1. Scrum vs. Kanban

Kanban y Scrum son marcos de trabajo utilizados para gestionar el desarrollo de productos y la organización del trabajo. Ambos comparten principios del desarrollo ágil, como la transparencia, la mejora continua y la entrega de valor, pero difieren significativamente en su estructura y en el momento estratégico en que cada uno aporta mayor beneficio.

12.1.1. Kanban: Flexibilidad y Flujo Continuo

Este marco se basa en la visualización del flujo de trabajo mediante tableros donde las tareas avanzan entre distintos estados. Al no establecer iteraciones fijas ni roles específicos, permite incorporar nuevas tareas o pivotar prioridades en cualquier momento, siempre que se respeten los límites de trabajo en curso (WIP - Work In Progress).

Es más útil cuando:

- **La incertidumbre es máxima:** En fases donde no se puede predecir qué se descubrirá a corto plazo.
- **Las tareas son reactivas o de soporte:** Entornos donde surgen urgencias constantes que no pueden esperar a un ciclo de planificación.
- **El objetivo es la agilidad pura:** Resulta adecuado para actividades exploratorias o de investigación, donde las prioridades cambian con frecuencia y se requiere validar hipótesis sin la rigidez de un compromiso cerrado.

12.1.2. Scrum: Estructura y Entrega Incremental

Scrum organiza el trabajo en iteraciones de duración fija denominadas Sprints, durante las cuales el equipo se compromete a desarrollar un conjunto definido de funcionalidades. Introduce una estructura formal mediante roles definidos (Scrum Master, Product Owner, Equipo de Desarrollo), eventos periódicos (Planning, Daily, Review, Retrospective) y artefactos de planificación.

Es más útil cuando:

- **La prioridad es la predictibilidad:** Se requiere saber qué valor tangible se entregará al final de un periodo determinado.
- **Se requiere *feedback* continuo:** Las revisiones de *sprint* permiten validar el progreso con los interesados de forma constante y cadenciada.
- **El producto es complejo pero definible:** Facilita la construcción de soluciones mediante incrementos planificados y medibles, asegurando que los componentes críticos se desarrollen bajo estándares de calidad estables.

12.1.3. Conclusión

En síntesis, Kanban prioriza la flexibilidad del flujo de trabajo, siendo la herramienta ideal para gestionar la variabilidad y el descubrimiento constante. Por el contrario, Scrum introduce una mayor estructura orientada a la entrega incremental y predecible de valor, siendo preferible cuando el objetivo es transformar requisitos definidos en un producto funcional mediante un ritmo de trabajo sostenido.

12.2. Entrevista semiestructurada con E-Mobility Solutions

Durante las primeras reuniones con el cliente se realizaron entrevistas semiestructuradas con el objetivo de comprender el contexto del negocio, el uso del sistema actual y las principales necesidades de la organización. A continuación se presentan algunas de las preguntas guía utilizadas durante estas entrevistas.

- ¿Cuáles son los objetivos de la empresa a largo plazo?
- ¿Qué tan importante es esta aplicación en el día a día del negocio?
- ¿Qué tantos clientes usan la app?
- ¿Qué tan seguido se utiliza este sistema?
- ¿Qué tipos de reportes necesitan?
- ¿Cada cuanto generan reportes?
- ¿Qué es lo que más les molesta de la aplicación actual? ¿El costo? ¿la falta de funcionalidad?
- ¿Debería tener integración con servicios externos? ¿Facturación?
- ¿Tiene un costo mensual para los usuarios finales usar la app?
- ¿Preferencia en la pasarela de pagos? Solo pago con tarjeta?
- ¿Quién tiene acceso a la aplicación?
- ¿Todos los empleados tienen control total sobre el sistema?
- ¿Cuántos clientes tienen?
- ¿Cuántos cargadores tienen instalados?
- ¿Cuántos empleados son?
- ¿Tienen a alguien de marketing?
- ¿Qué tantos clientes usan la app?
- ¿Alguna preferencia en la tecnología?
- ¿Cuánto les cuesta esta app por año?
- ¿Podemos conseguir acceso a la plataforma?
- ¿Cuándo podríamos reunirnos?

Estas interrogantes constituyeron la guía base de la entrevista, funcionando como puntos de partida desde los cuales surgieron preguntas adicionales de forma dinámica durante

el intercambio con el cliente. Esta flexibilidad, propia del formato semiestructurado, permitió profundizar en aspectos específicos del negocio y captar matices operativos que no habían sido previstos inicialmente. La información recolectada resultó fundamental para identificar con precisión las ineficiencias críticas de la plataforma VoltBras. Los resultados de estas instancias sirvieron como el insumo principal para la etapa de especificación de requerimientos, garantizando que el diseño del prototipo funcional estuviera plenamente alineado con los objetivos de autonomía y escalabilidad definidos por la organización.

12.3. Visita a las instalaciones de E-Mobility Solutions

Al inicio del proyecto, el equipo realizó una visita a las instalaciones de E-Mobility Solutions con el objetivo de efectuar una observación directa del funcionamiento de los cargadores eléctricos y comprender en profundidad los procesos operativos de la empresa.

Figura 46: Integrantes del equipo realizando observación directa

Durante la visita, Matías Crosa y Alfredo Pintos guiaron al equipo en el recorrido por las áreas de trabajo y mostraron en detalle la interacción con los cargadores, explicando las diferencias entre distintos cargadores, ejemplificando su funcionamiento y respondiendo las dudas que fueron surgiendo. Esta instancia permitió al equipo obtener una visión práctica de los sistemas y procedimientos involucrados, lo que resultó fundamental para contextualizar el desarrollo posterior del proyecto.

12.4. Analisis de la Plataforma Actual (VoltBras)

Al iniciar el proyecto con E-Mobility Solutions, el equipo dedicó tiempo a analizar en profundidad la plataforma VoltBras, la solución brasileña multitenant whitelabel que la empresa utilizaba para gestionar sus cargadores eléctricos. Este análisis exploratorio, realizado mediante navegación directa por el sistema, permitió al equipo comprender sus capacidades operativas, identificar fortalezas y detectar las limitaciones que motivaron el desarrollo de una solución propia.

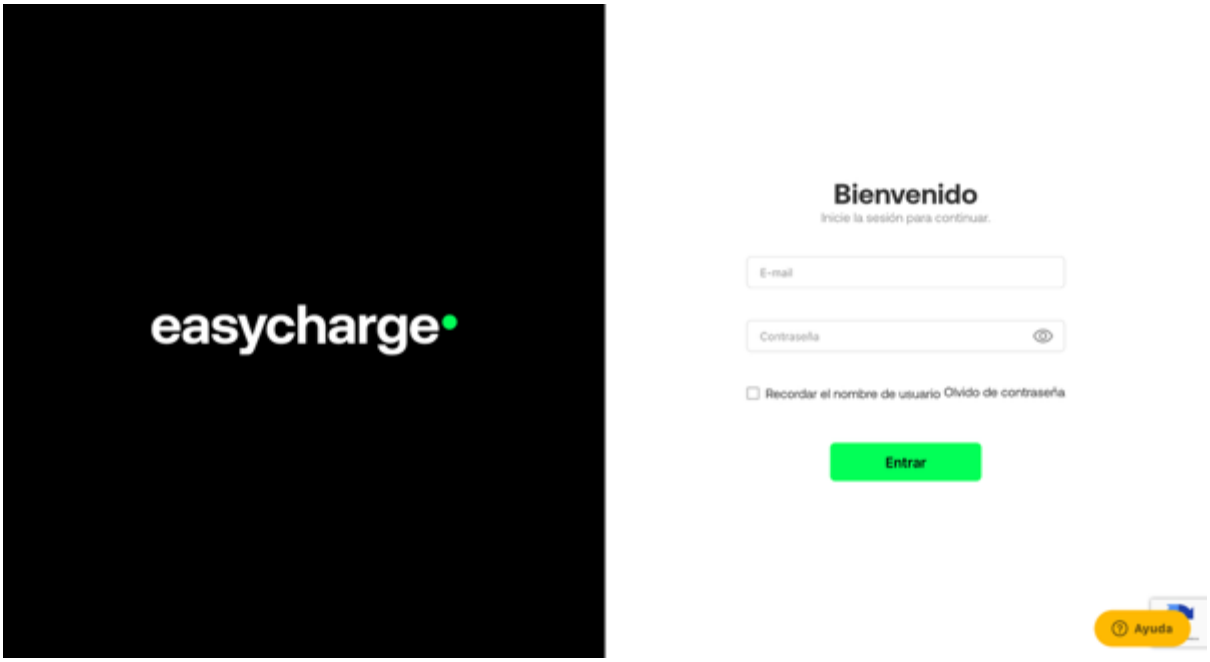


Figura 47: Pantalla de bienvenida de la plataforma Voltrbras

A continuación pasamos a ver lo que sería la pantalla principal al entrar al sistema. Se puede visualizar gráficas, métricas de consumo y un listado de los cargadores junto a algunos datos del mismo, como la ubicación, organización perteneciente y estado de sus conectores.

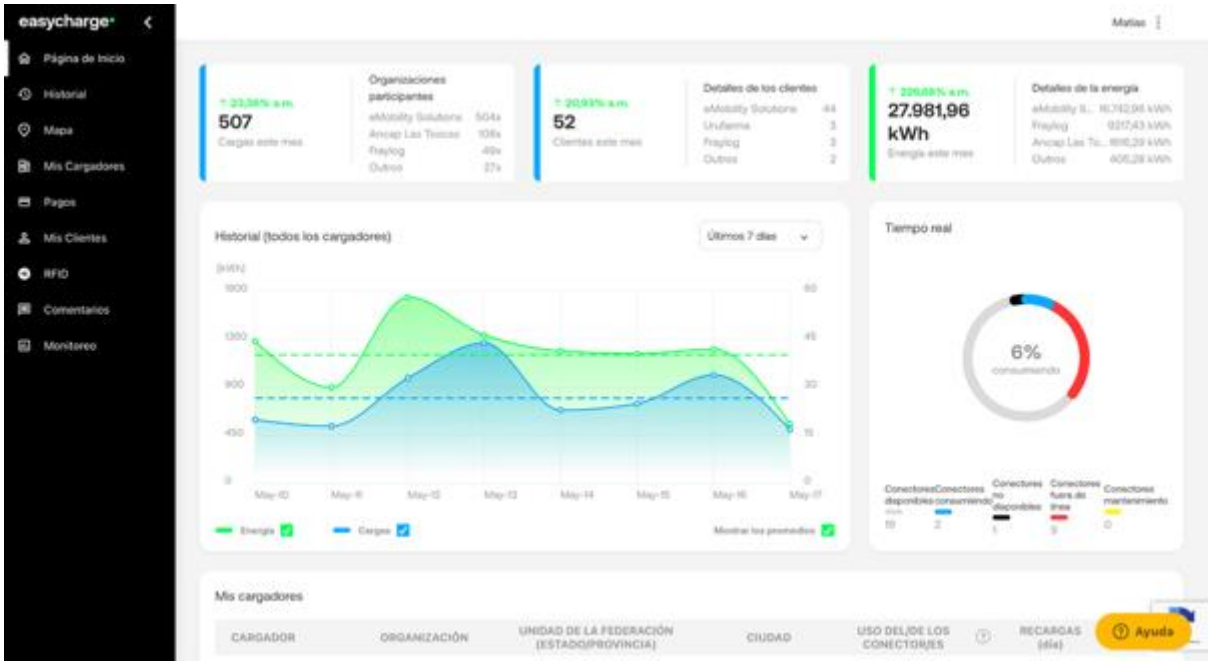


Figura 48: Dashboard principal al ingresar al sistema de VoltBras

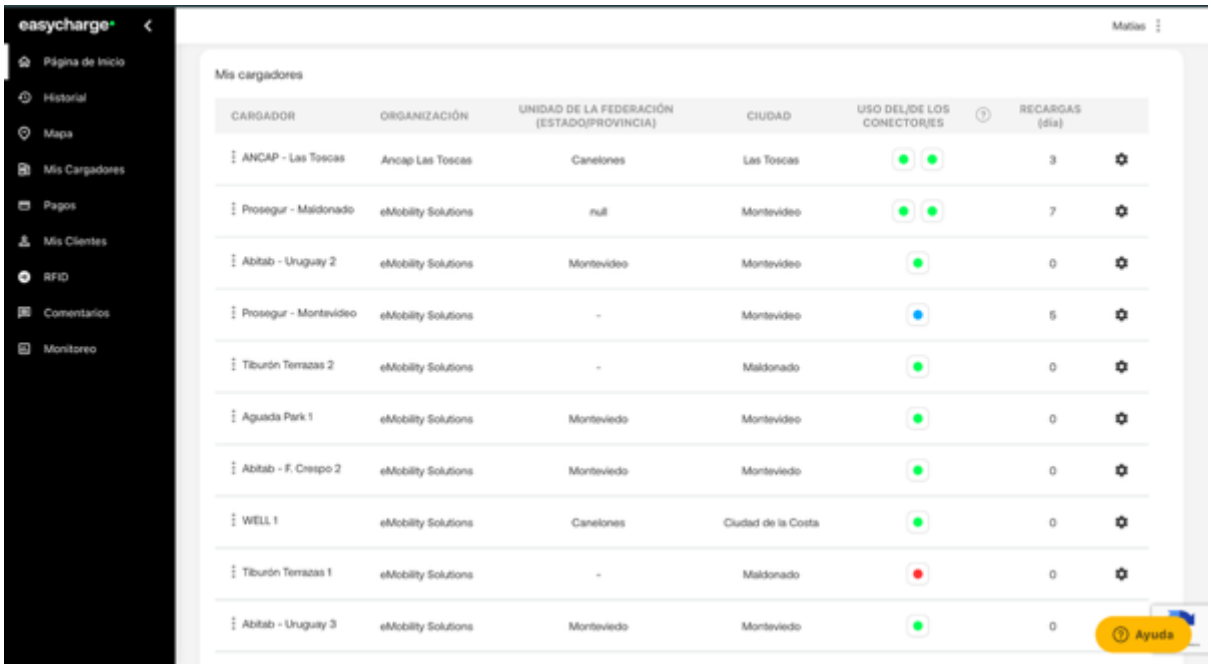


Figura 49: Visualización de la lista de cargadores en dashboard principal de VoltBras, con el estado de sus conectores

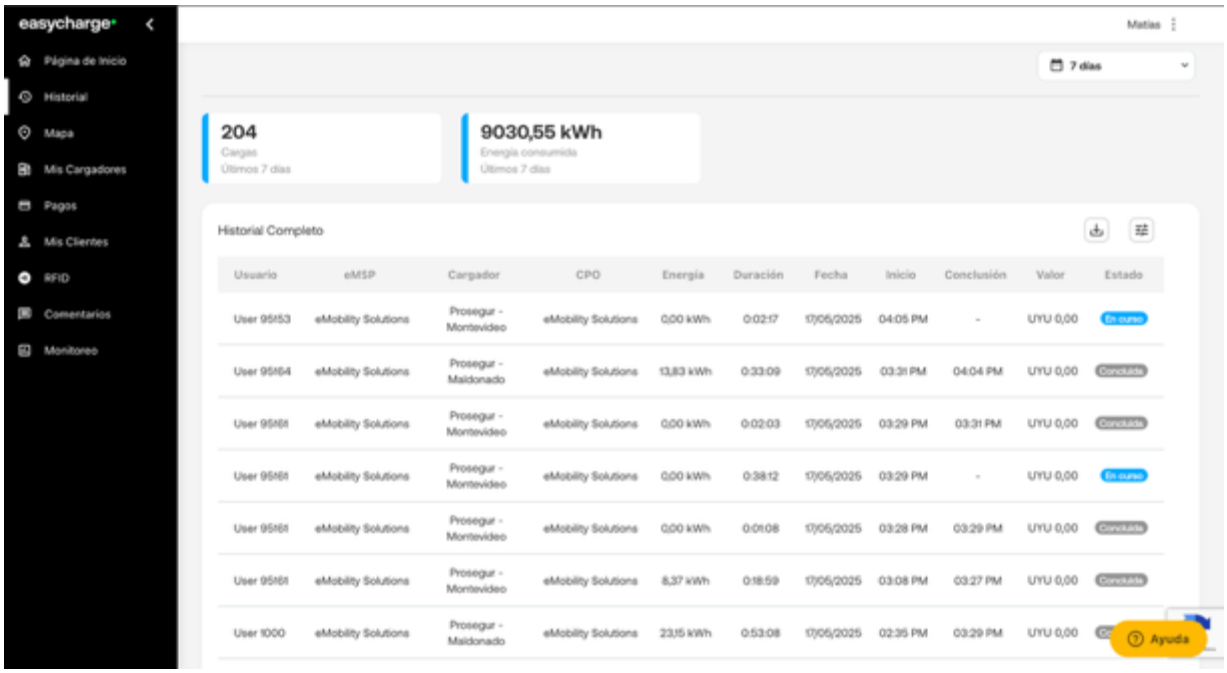


Figura 50: Historial de cargas en VoltBras

La siguiente imagen muestra una carga concluida, con información de la misma como el horario en el que estuvo en curso, su tiempo de inactividad (importante para cobrarle a los usuarios si se exceden del límite), qué conector usaron y un gráfico del consumo en tiempo real, que permite visualizar si en algún momento hubo un corte de energía o algún problema con la carga.

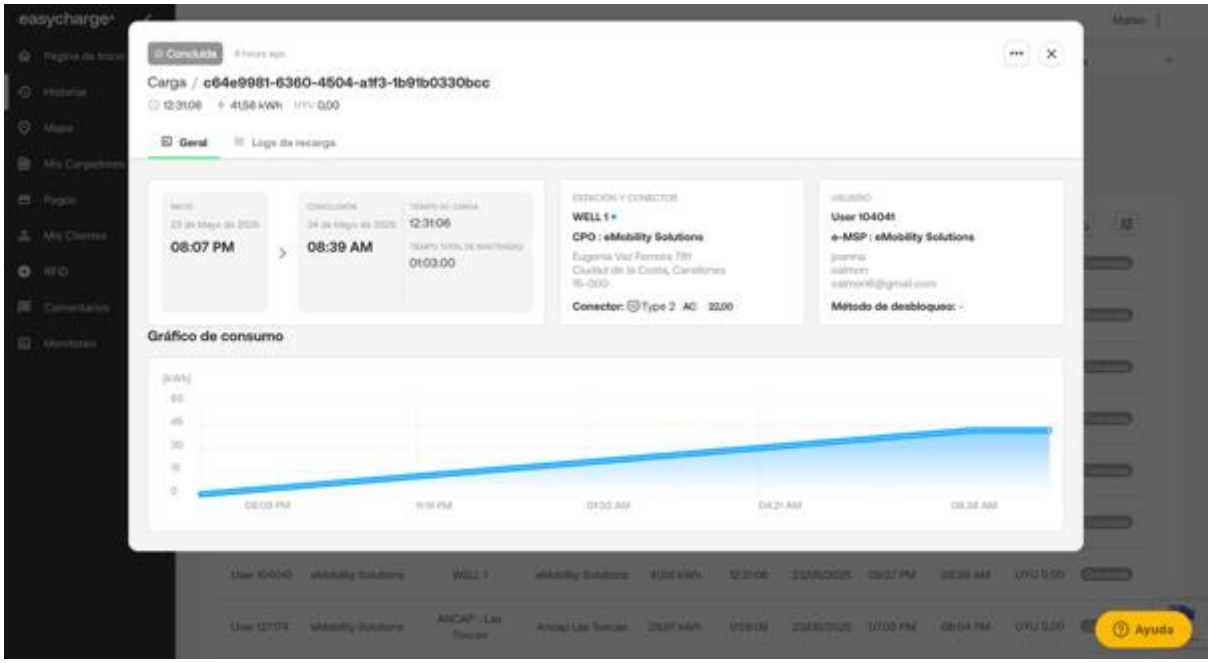


Figura 51: Visualización de una carga en particular en VoltBras

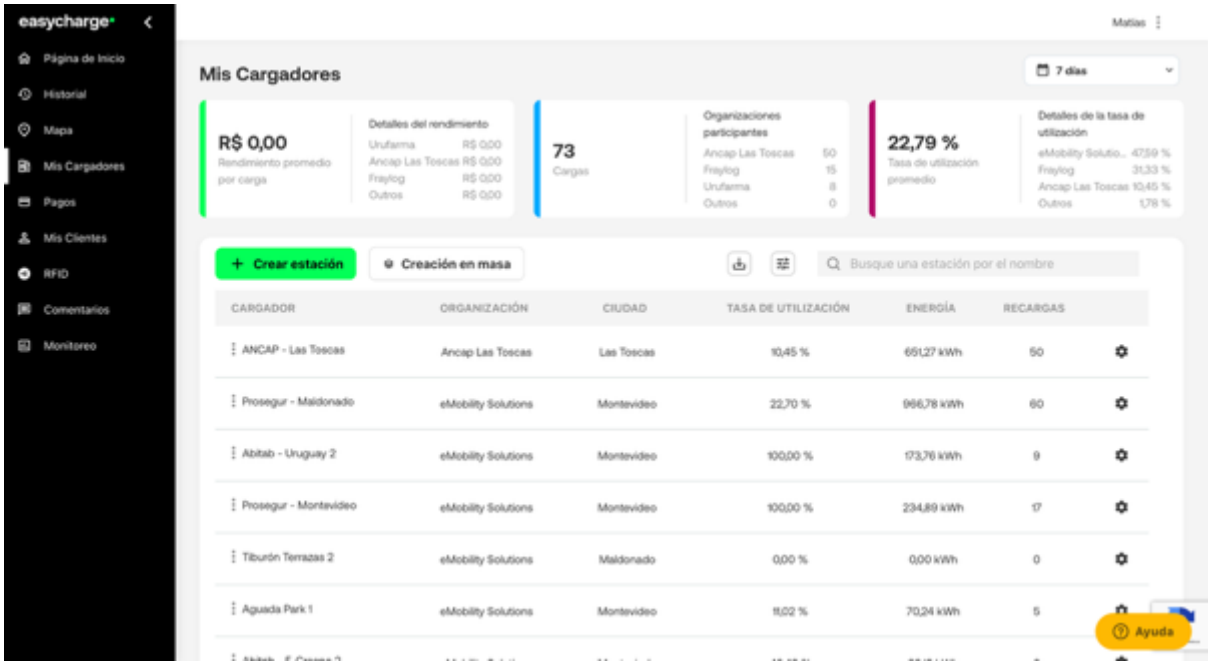


Figura 52: Visualización inicial de la pantalla de cargadores en VoltBras

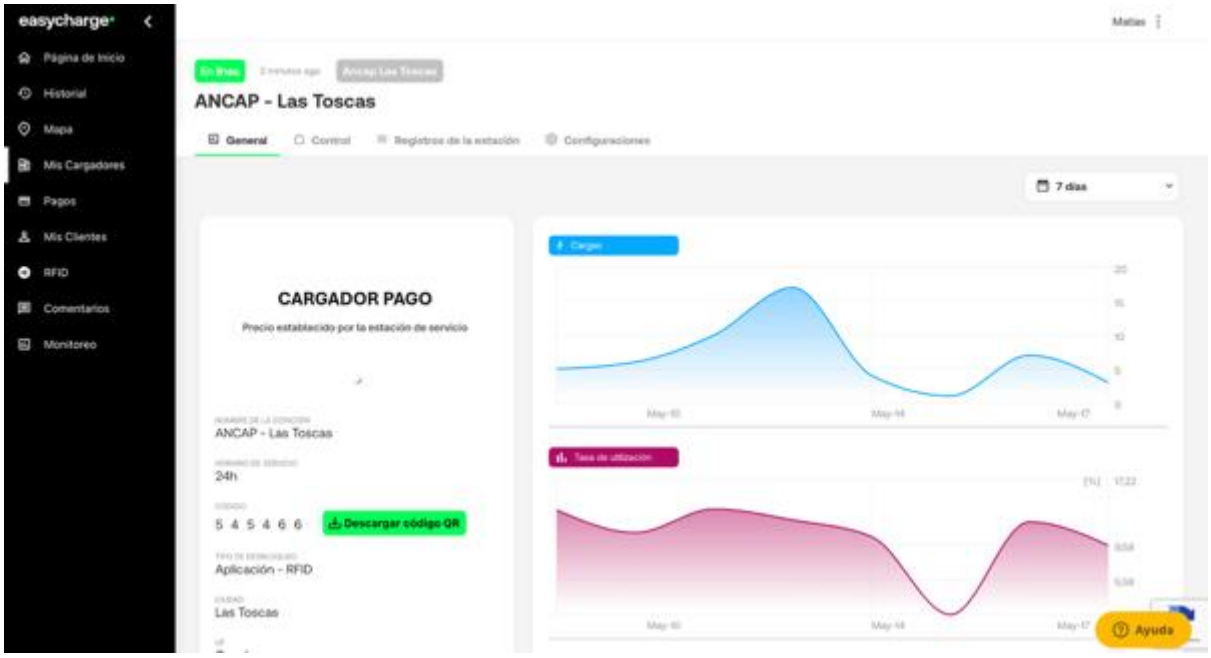


Figura 53: Información general de un cargador

Las siguiente imágenes muestran una de las partes más importantes del sistema, los cargadores. A continuación se detallan su historial de cargas y usos de la estación, al igual que datos específicos de configuración del protocolo OCPP y logs. Estos logs son cruciales ya que permiten al administrador visualizar de forma rápida si hubo algún problema de conexión entre el cargador y el servidor, fallas de energía, pérdidas de internet, entre otras.

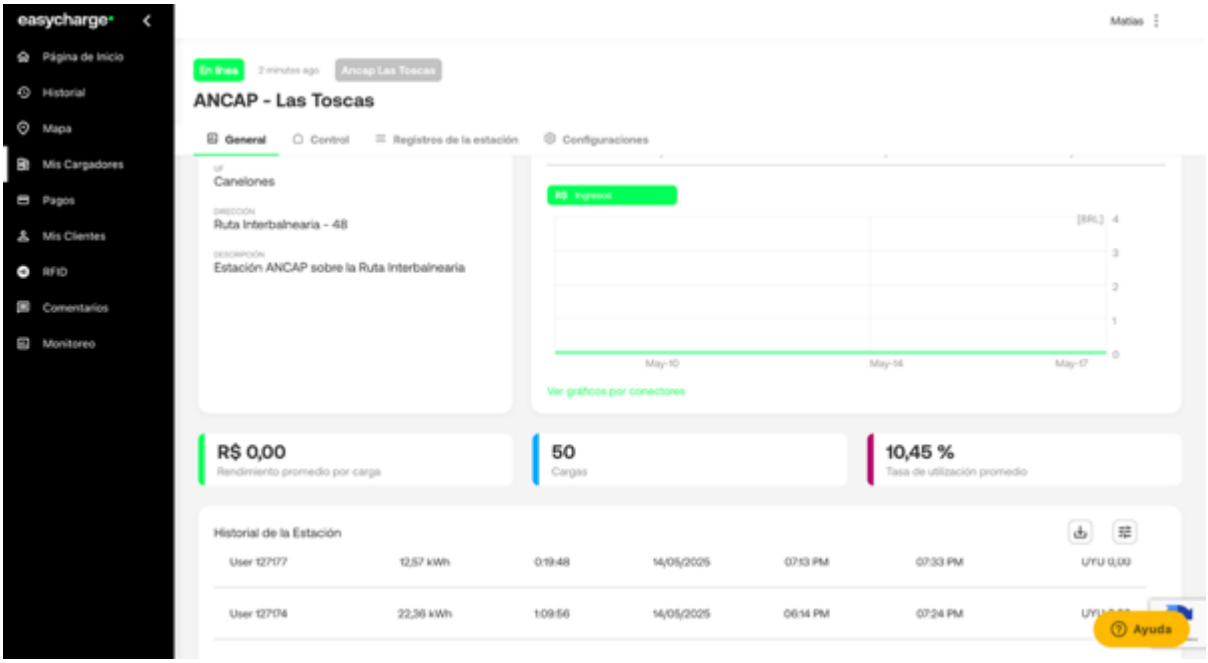


Figura 54: Métricas e historial de carga de un cargador

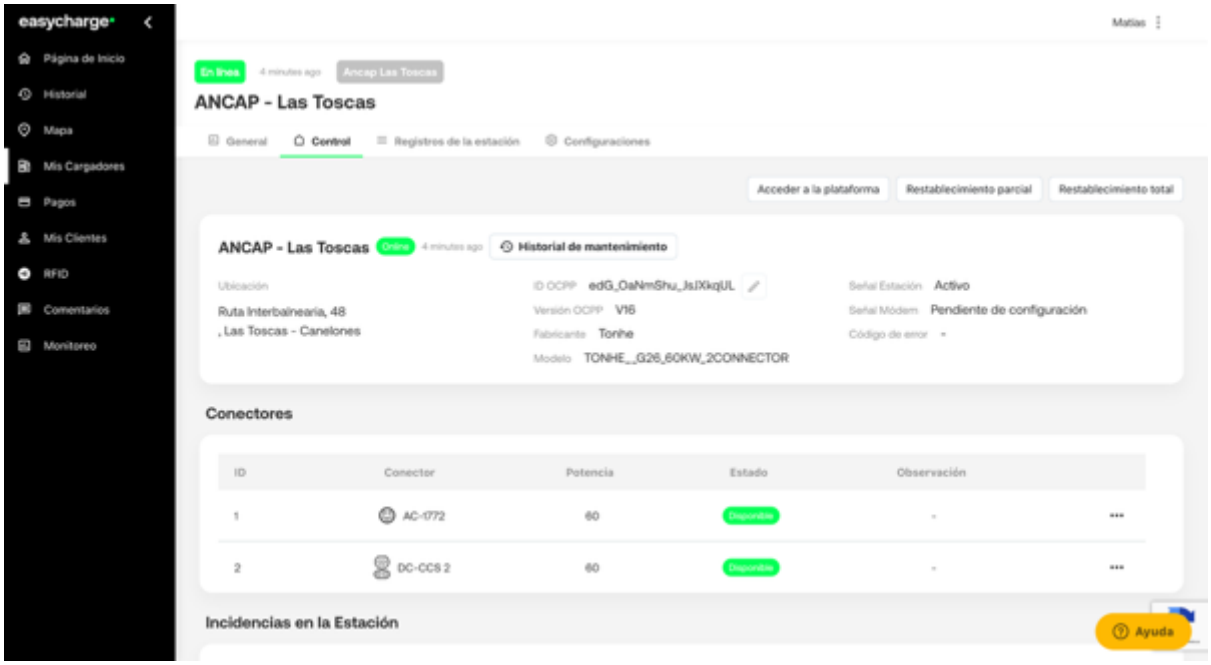


Figura 55: Sección de control de un cargador con sus conectores y datos propios

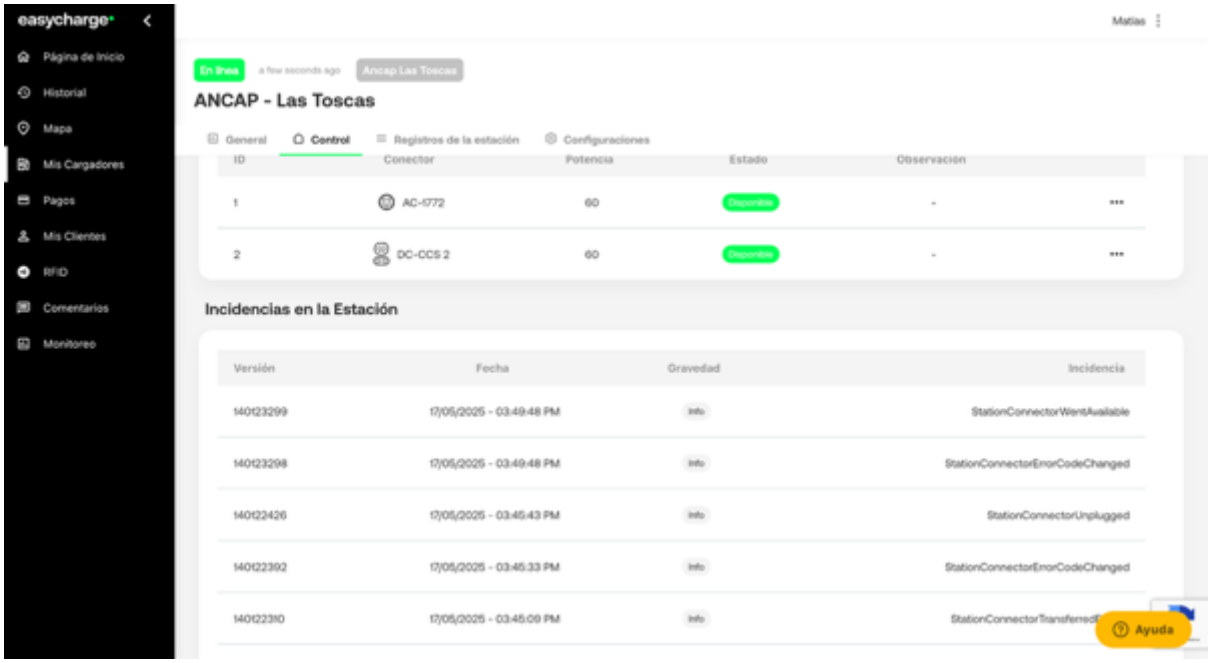


Figura 56: Visualización de incidentes de cierto cargador

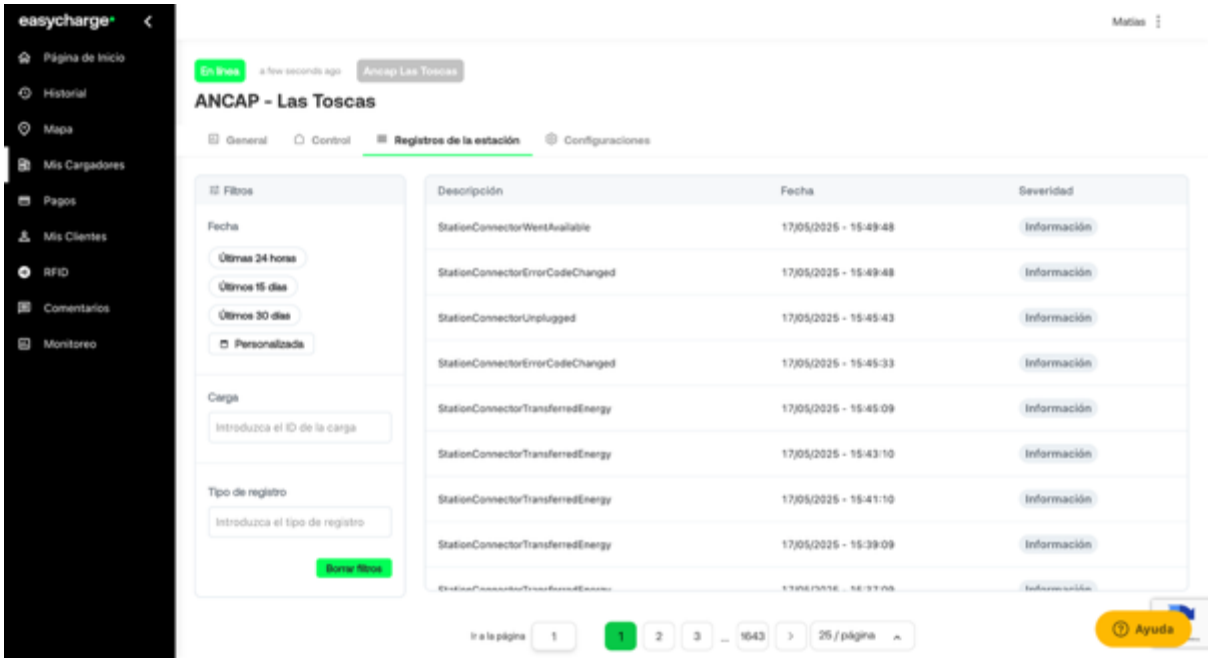


Figura 57: Registros de incidencias de un cargador

12.5. Investigación de OCPP

El equipo realizó una investigación sobre OCPP ya que el sistema a desarrollar debía interactuar con cargadores compatibles con este protocolo, y gran parte de la lógica operativa se materializa precisamente a través de mensajes OCPP. Comprender en detalle el estándar permitía validar requerimientos de la solución, reducir la incertidumbre y definir de manera informada la arquitectura.

OCPP es un protocolo de comunicación abierto diseñado específicamente para estandarizar el intercambio de mensajes entre estaciones de carga y plataformas de backend, permitiendo interoperar hardware y software de distintos fabricantes sin depender de soluciones propietarias. Este protocolo es actualmente mantenido y evolucionado por la Open Charge Alliance (OCA).

Para este proyecto, por pedido del cliente, el equipo se enfocó principalmente en la versión OCPP 1.6. Es la versión más ampliamente implementada a nivel mundial y la soportada por la mayoría de los cargadores del entorno objetivo.

El análisis consistió de dos grandes aspectos: análisis de la documentación oficial y construcción de una prueba de concepto.

12.5.1. Análisis de la Documentación

El equipo realizó un análisis detallado de la documentación oficial de OCPP, tomando como referencia las especificaciones publicadas por la Open Charge Alliance para OCPP 1.6 y el material complementario de interpretación técnica. Este trabajo se centró en entender no solo la lista de mensajes disponibles, sino también la arquitectura propuesta (Charge Point y Central System), el modelo de comunicación sobre WebSocket/JSON y las responsabilidades de cada extremo en los distintos flujos de uso.

A partir de la documentación oficial de OCPP, el equipo identificó en primer lugar el conjunto de mensajes definidos por el protocolo y las acciones que estos permiten

realizar sobre los cargadores. En base a este relevamiento, se construyó una tabla que resume todos los mensajes aceptados.

Mensaje	Descripción breve
Authorize	Autoriza una tarjeta o usuario antes de iniciar carga
BootNotification	Registro inicial del cargador en el sistema central
CancelReservation	Cancela una reserva existente
ChangeAvailability	Cambia disponibilidad de un conector o cargador
ChangeConfiguration	Modifica un parámetro de configuración
ClearCache	Limpia la caché de autorizaciones
ClearChargingProfile	Elimina perfiles de carga
DataTransfer	Envía datos personalizados (extensión)
DiagnosticsStatusNotification	Notifica estado de diagnóstico
FirmwareStatusNotification	Notifica estado de actualización de firmware
GetCompositeSchedule	Solicita cronograma compuesto de carga
GetConfiguration	Consulta parámetros de configuración
GetDiagnostics	Solicita generación de diagnóstico
GetLocalListVersion	Consulta versión de lista local de autorizaciones

Heartbeat	Mantiene sincronización y conexión activa
MeterValues	Envía valores de medición (energía, potencia, etc.)
RemoteStartTransaction	Inicia carga remotamente
RemoteStopTransaction	Detiene carga remotamente
ReserveNow	Reserva un conector
Reset	Reinicia el cargador
SendLocalList	Envía lista local de autorizaciones
SetChargingProfile	Define perfil de carga
StartTransaction	Indica inicio de sesión de carga
StatusNotification	Notifica cambio de estado del conector
StopTransaction	Indica fin de sesión de carga
TriggerMessage	Solicita al cargador que envíe un mensaje específico
UnlockConnector	Desbloquea un conector

Luego, el equipo analizó los diagramas de secuencia provistos por la documentación para comprender cómo se maneja la comunicación en los distintos flujos de interacción entre el cargador y el sistema central. Este trabajo permitió entender de forma explícita el manejo de estados del cargador, las transiciones entre ellos, así como los mecanismos previstos para tratar fallos de comunicación, desconexiones y reintentos de mensajes.

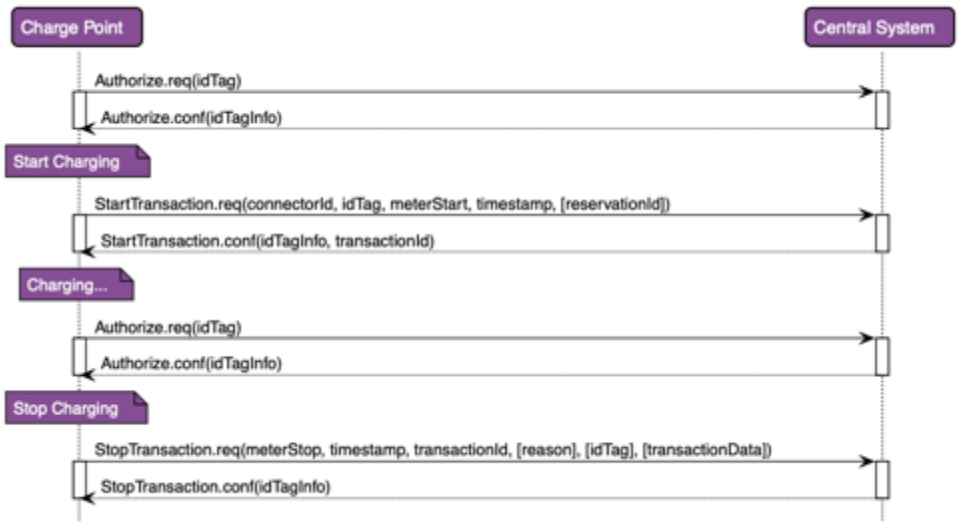


Figura 58: Ejemplo de diagrama de secuencia extraído de la documentación oficial de OCPP 1.6

Finalmente, se realizó un análisis detallado de la composición de los mensajes JSON definidos por OCPP que se envían a través de la comunicación por websockets, estudiando su estructura y el contenido de cada payload. Esto permitió comprender qué datos intercambia cada mensaje, entender si hay campos son obligatorios u opcionales y cómo se representan aspectos los códigos de error, la información de la sesión de carga y los parámetros de configuración.

Estructura de Mensaje de Solicitud: [<MessageTypeId>, "<UniqueId>", "<Action>", {<Payload>}]

Ejemplo de Mensaje de Solicitud:

```
[

 2,

 "19223201",

 "BootNotification",

 {

   "chargePointVendor": "VendorX",

   "chargePointModel": "SingleSocketCharger"
```

```
 }

]
```

Estructura de Mensaje de Respuesta: [<MessageTypeId>, "<UniqueId>", {<Payload>}]

Ejemplo de Mensaje de Respuesta:

```
[

  3,

  "19223201",

  {

    "status":"Accepted",

    "currentTime":"2013-02-01T20:53:32.486Z",

    "heartbeatInterval":300

  }

]
```

12.5.2. Prueba de Concepto

A partir del conocimiento adquirido en el análisis de la documentación de OCPP, el equipo decidió desarrollar una PoC con el objetivo de validar los requerimientos técnicos asociados a la comunicación entre el sistema y los cargadores. Esta instancia buscó comprobar en un entorno controlado que la plataforma fuera capaz de establecer y mantener una conexión compatible con el protocolo, antes de avanzar con un diseño de mayor complejidad.

La prueba de concepto tuvo una misión deliberadamente simple: lograr que el sistema se comunicara de manera exitosa con un cargador OCPP y respondiera correctamente a los mensajes recibidos. Se hizo sin implementar aún la lógica de negocio final ni reglas

avanzadas de procesamiento. El foco estuvo puesto en poder recibir, interpretar y contestar los mensajes básicos del protocolo, respetando el formato y la secuencia esperada.

12.5.2.1. Código del PoC

```
import { WebSocketServer } from 'ws';

import express from 'express';




const app = express();

const wss = new WebSocketServer({ port: 8080 });

const clients = new Set();

app.use(express.json());



const handleBootNotification = (data, ws) => {

 const currentTime = new Date().toISOString();

 const response = [3, data[1], {"status":"Accepted", "currentTime": currentTime,
"interval":1000}];

 ws.send(JSON.stringify(response));

}

const handleStatusNotification = (data, ws) => {

 const response = [3, data[1], {"status":"Accepted"}];

 ws.send(JSON.stringify(response));

}

const handleHeartbeat = (data, ws) => {

 const currentTime = new Date().toISOString();

 const response = [3, data[1], {"currentTime": currentTime}];
```

```
 ws.send(JSON.stringify(response));

}



const handleAuthorization = (data, ws) => {

 const response = [3, data[1], {"idTagInfo": {"status": "Accepted"}}];

 ws.send(JSON.stringify(response));

}



const handleStartTransaction = (data, ws) => {

 const response = [3, data[1], {"transactionId": 12345, "idTagInfo": {"status":
"Accepted"}}];

 ws.send(JSON.stringify(response));

}



const handleStopTransaction = (data, ws) => {

 const response = [3, data[1], {"idTagInfo": {"status": "Accepted"}}];

 ws.send(JSON.stringify(response));

}



const actionFactory = {

 BootNotification: handleBootNotification,

 StatusNotification: handleStatusNotification,

 Heartbeat: handleHeartbeat,

 Authorization: handleAuthorization,

 StartTransaction: handleStartTransaction,

 StopTransaction: handleStopTransaction,
```

```
}


const handleMessage = (rawData, ws) => {
 const data = JSON.parse(rawData.toString());
 console.log('Received message:', data);
 const messageType = data[2];

 if (actionFactory[messageType]) {
   actionFactory[messageType](data, ws);
 }
}


wss.on('connection', function connection(ws, request) {
 clients.add(ws);
 ws.on('error', console.error);

 ws.on('message', function message(data) {
   handleMessage(data, ws);
 });
  ws.on('close', function() {
   clients.delete(ws);
 });
});
```

```
app.post('/send', (req, res) => {

 const { message } = req.body;

  clients.forEach(client => {

   if (client.readyState === 1) {

     client.send(JSON.stringify(message));

   }

 });

  res.json({ sent: true, clientCount: clients.size, message });

});



app.listen(3000, () => {});
```

Esta PoC permitió reducir significativamente el riesgo y la incertidumbre técnica asociados a la integración con los cargadores. Una vez verificado que la comunicación se comportaba como se esperaba, el equipo contó con la confianza necesaria para continuar con el diseño detallado de la solución, sabiendo que el fundamento técnico de la conexión estaba validado.

12.6. Ejemplo de historia de usuario

Como usuario, quiero iniciar sesión en la aplicación, para poder acceder a las funcionalidades del sistema.

Ask Brain to create a summary, generate subtasks or find similar tasks

Status	COMPLETED	Assignees	Maria Bañales
Dates	Start → Due	Priority	Urgent
Time estimate	5h	Sprint points	Empty
Track time	Start	Tags	Empty

Saved

Descripción

Implementar la funcionalidad que permita a un usuario autenticarse en la aplicación móvil mediante sus credenciales. El sistema deberá validar la información ingresada contra el backend y, en caso de ser correcta, permitir el acceso a las funcionalidades de la aplicación.

Criterios de aceptación

CA1

Dado que el usuario se encuentra en la pantalla de inicio de sesión,

Cuando ingresa su correo electrónico y contraseña válidos y presiona el botón para iniciar sesión,

Entonces el sistema envía una solicitud al backend para validar las credenciales.

CA2

Dado que el sistema envía las credenciales al backend,

Cuando las credenciales son válidas,

Entonces el sistema autentica al usuario y muestra la pantalla principal de la aplicación.

CA3

Dado que el usuario intenta iniciar sesión,

Cuando las credenciales ingresadas son incorrectas,

Entonces el sistema muestra un mensaje indicando que el correo electrónico o la contraseña no son válidos.

CA4

Dado que el usuario inicia sesión correctamente,

Cuando accede a la aplicación,

Entonces el sistema mantiene la sesión activa y permite el acceso a las funcionalidades disponibles para el usuario.

Figura 59: Ejemplo de historia de usuario

12.7. Visualización de los flujos críticos del sistema

12.7.1. Visualización de la conexión del cargador

12.7.1.1. Visualización del panel administrativo

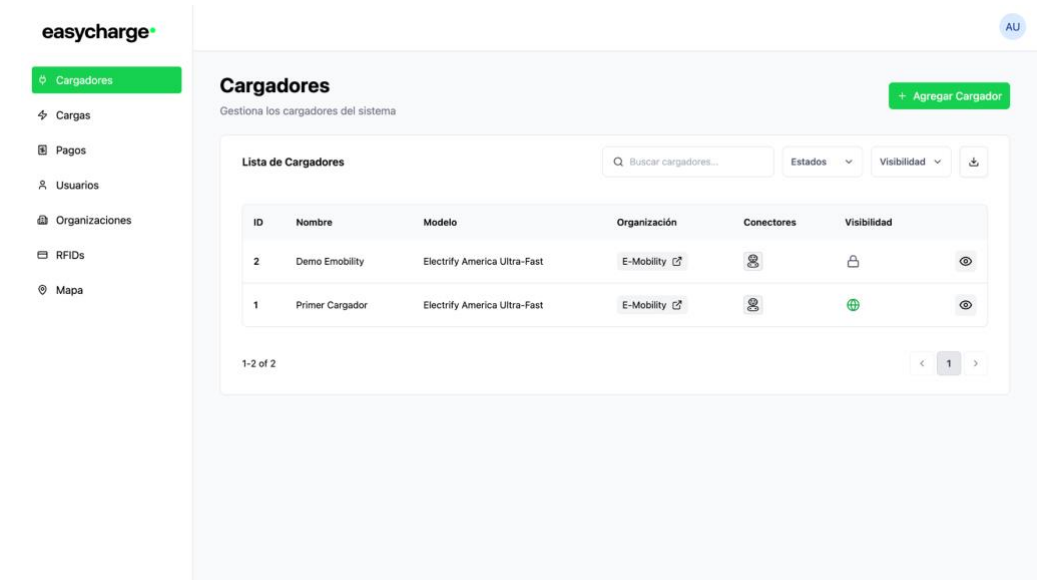


Figura 60: Área de Gestión de cargadores con cargadores desconectados

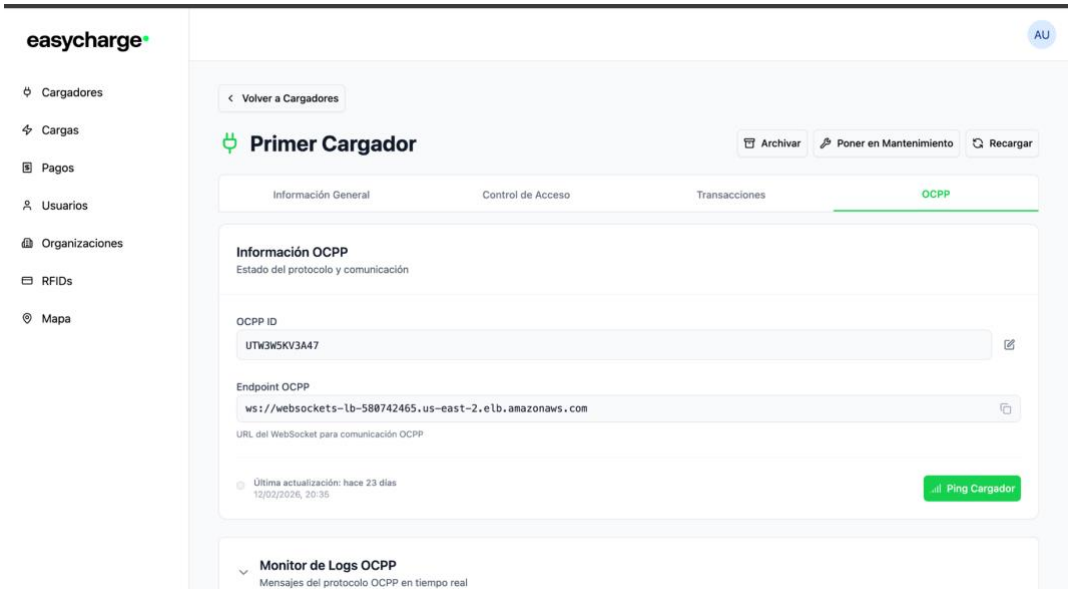


Figura 61: Detalles de conexión OCPP del cargador

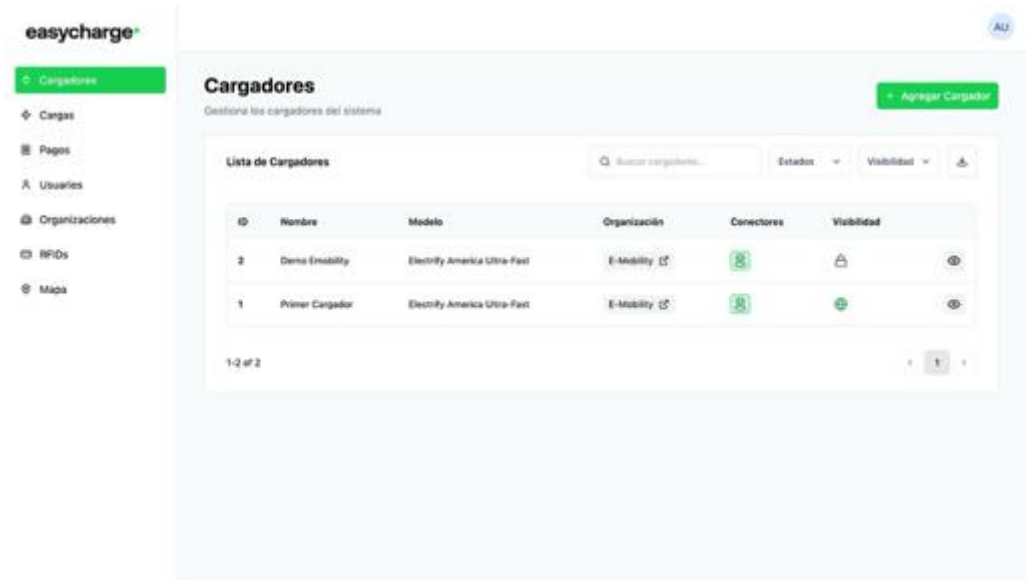


Figura 62: Área de Gestión de cargadores con cargadores conectados al sistema

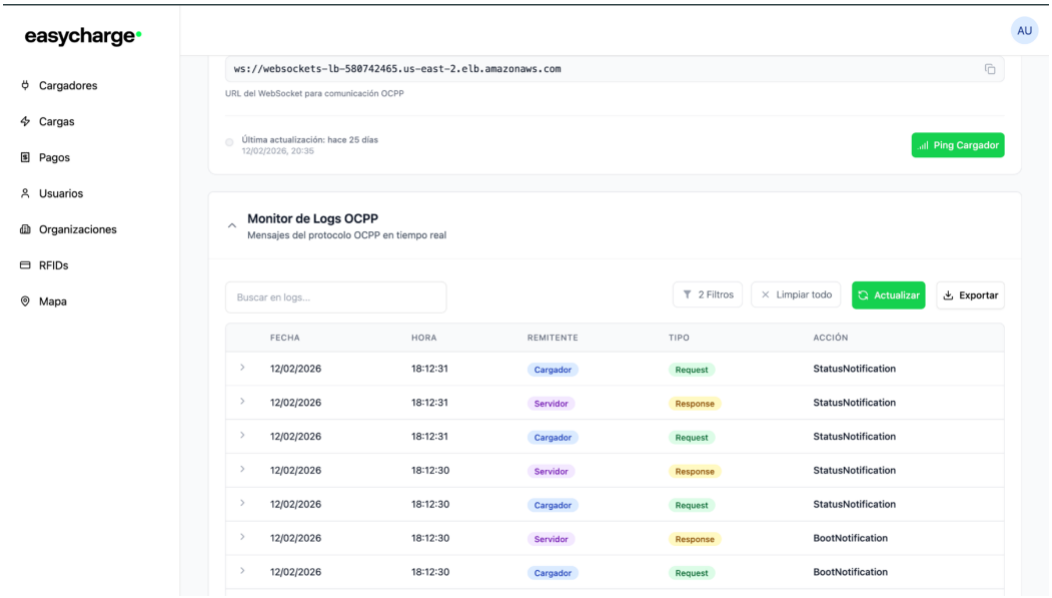


Figura 63: Visualización de los logs en la conexión de un cargador

12.7.1.1. Visualización de la aplicación móvil

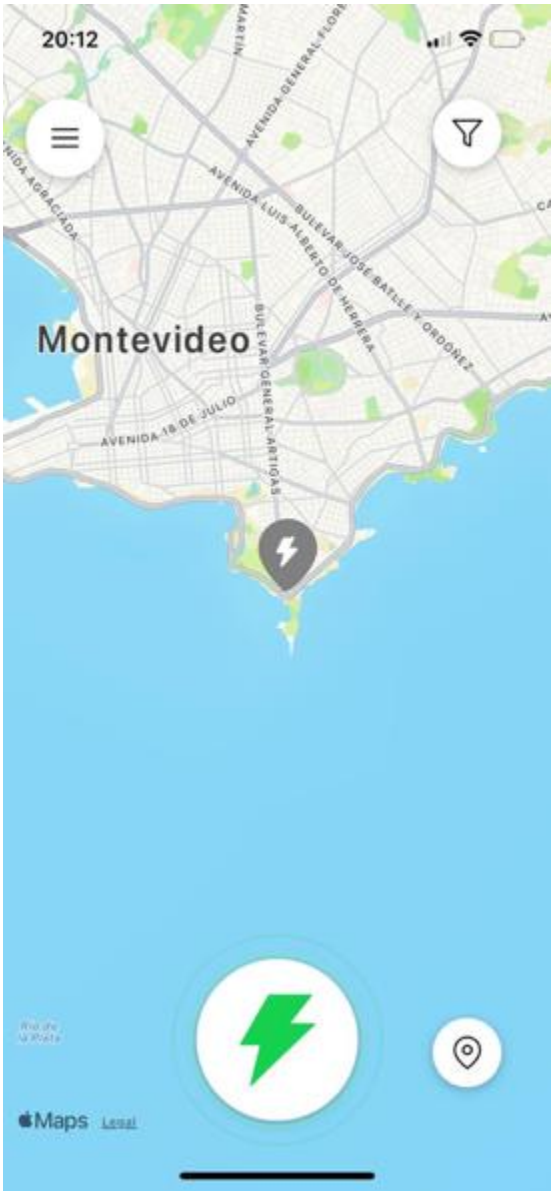


Figura 64: Visualización del cargador desconectado en el mapa

Figura 65: Visualización del cargador conectado en el mapa

Figura 66: Visualización detallada del estado del conector

12.7.2. Visualización del flujo de carga

12.7.2.1. Visualización del panel administrativo

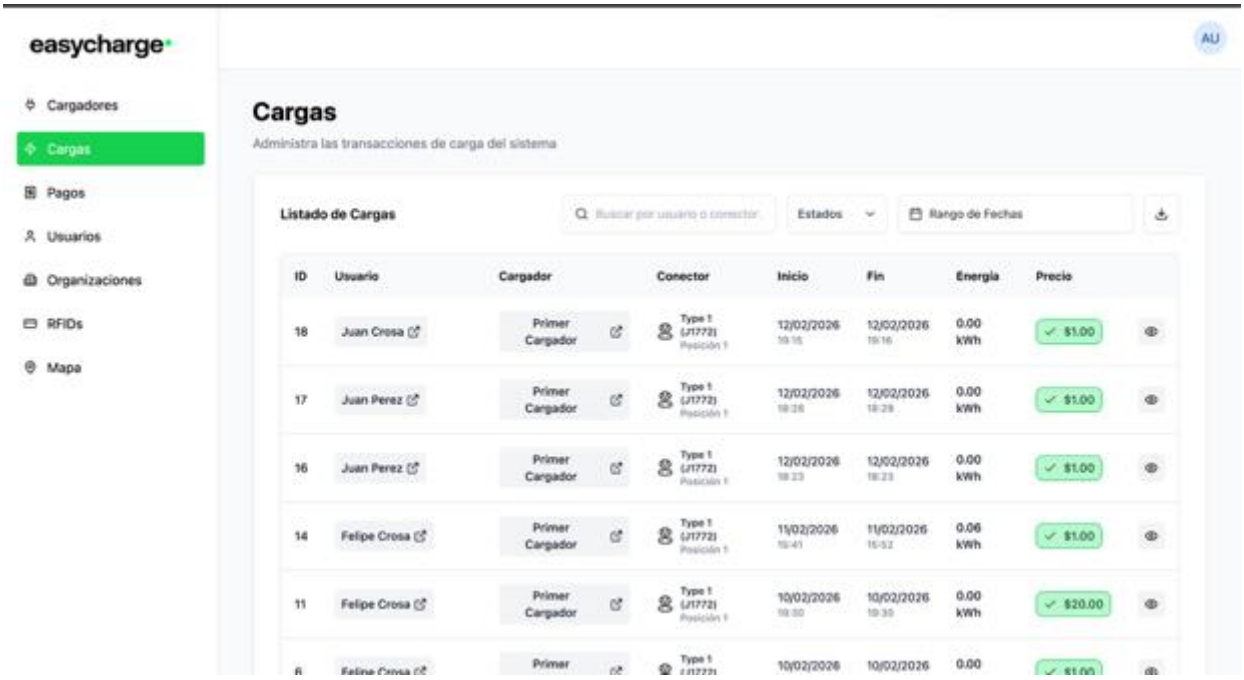


Figura 67: Área de Gestión de cargas

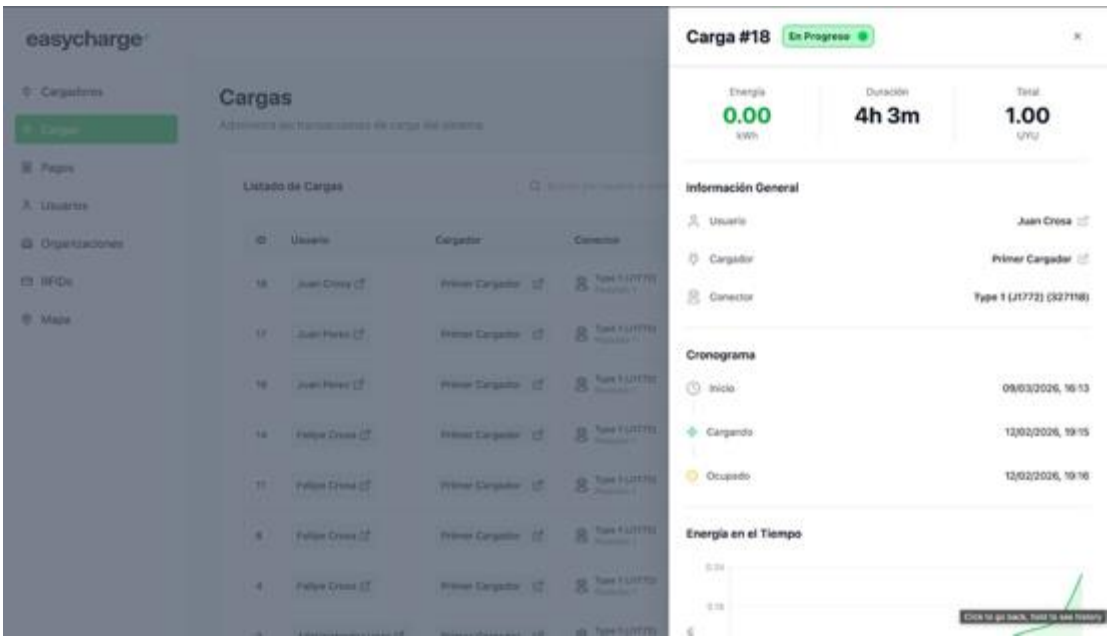


Figura 68: Detalle de una carga

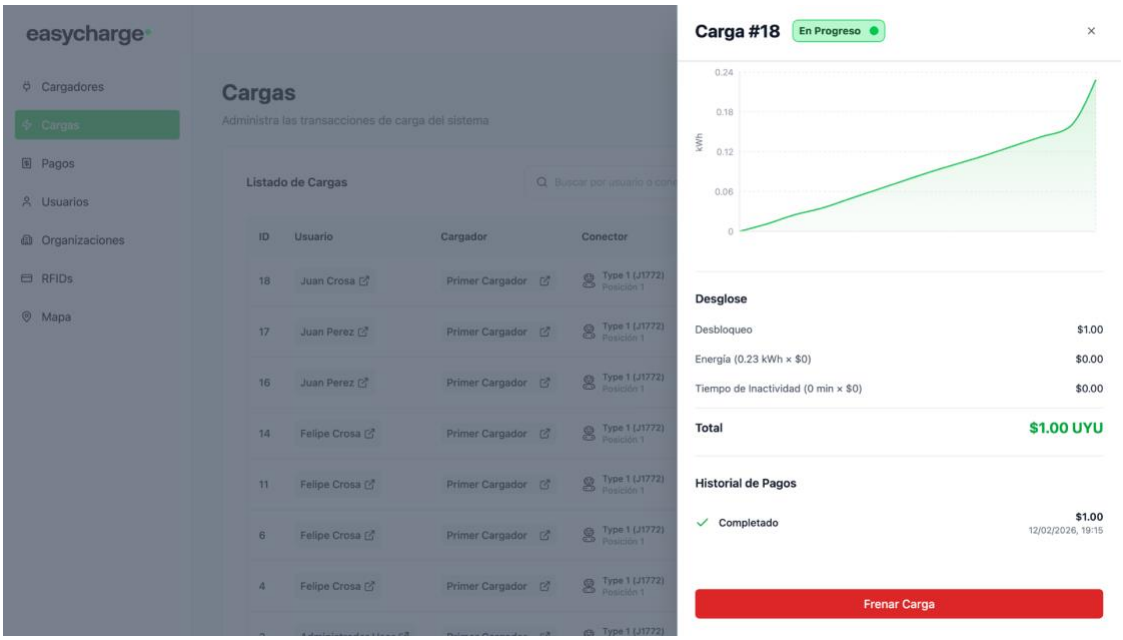


Figura 69: Detalle de una carga con visualización gráfica de la energía utilizada y opción de finalizar la carga

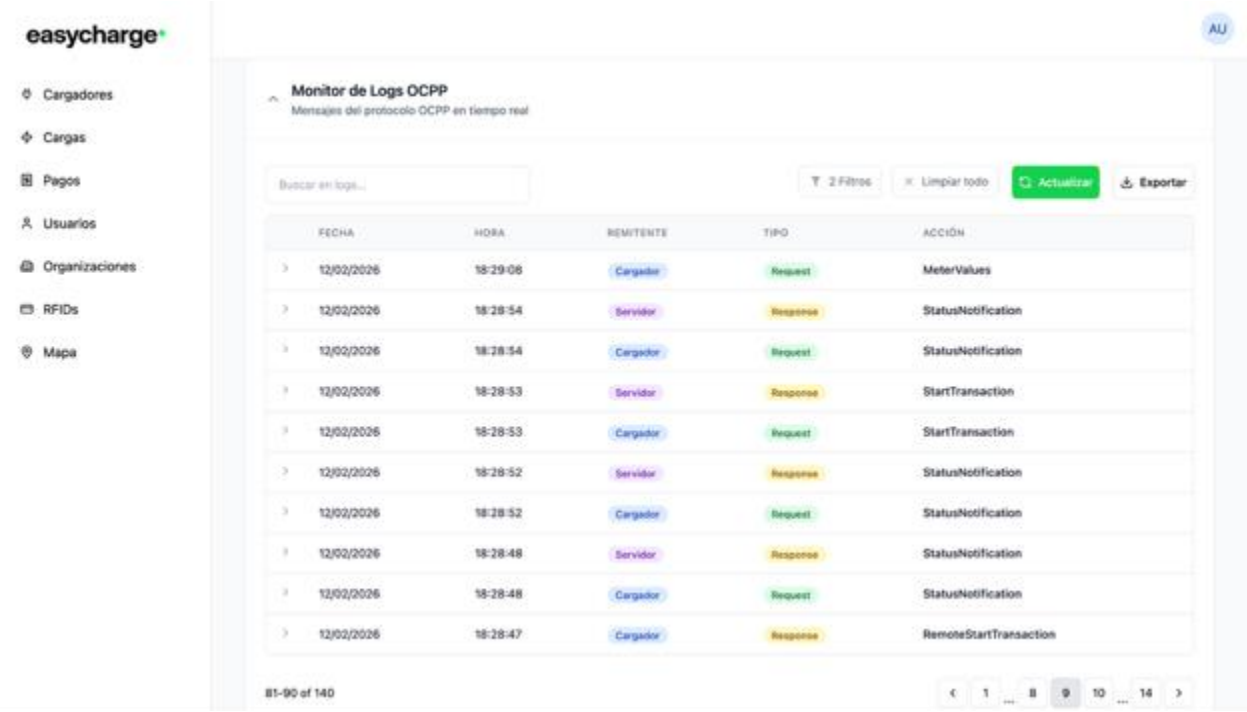


Figura 70: Visualización de logs del comienzo de la carga

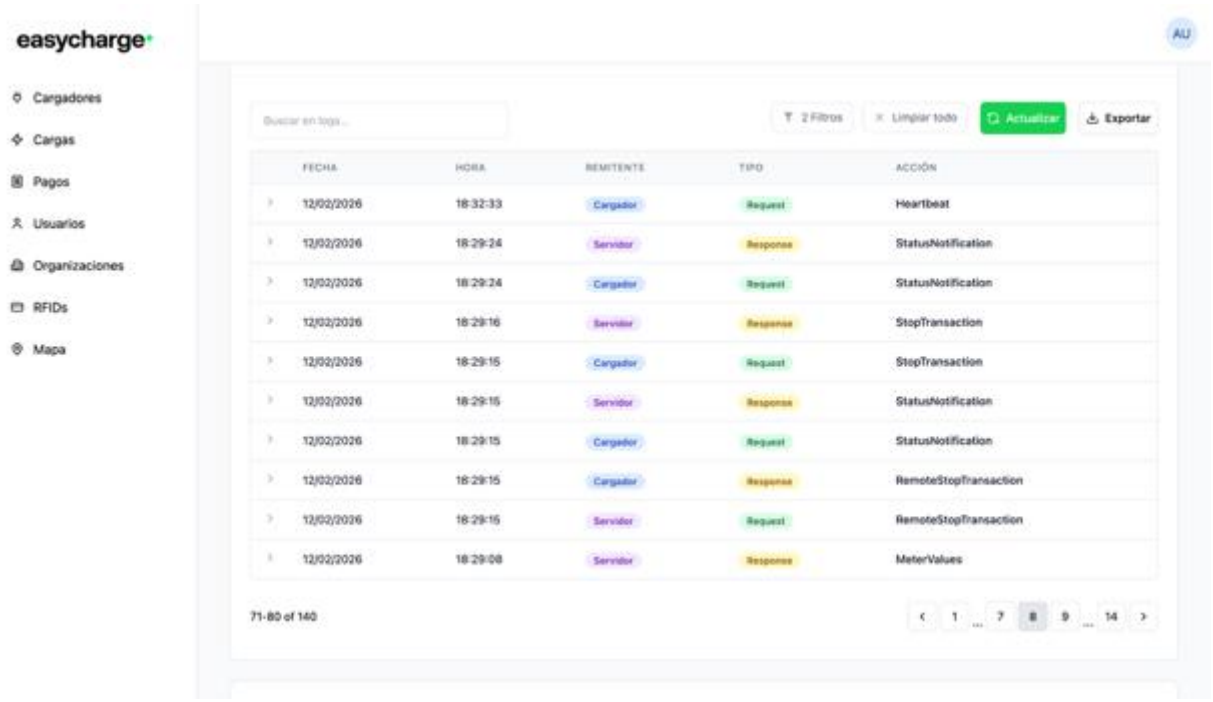


Figura 71: Visualización de logs del fin de la carga

12.7.2.1. Visualización de la aplicación móvil

Figura 72: Visualización de detalles de costos de un cargador

Figura 73: Visualización de detalles de pagos de cargas

Figura 74: Selección de métodos de pago para la carga

Figura 75: Agregar un nuevo método de pago

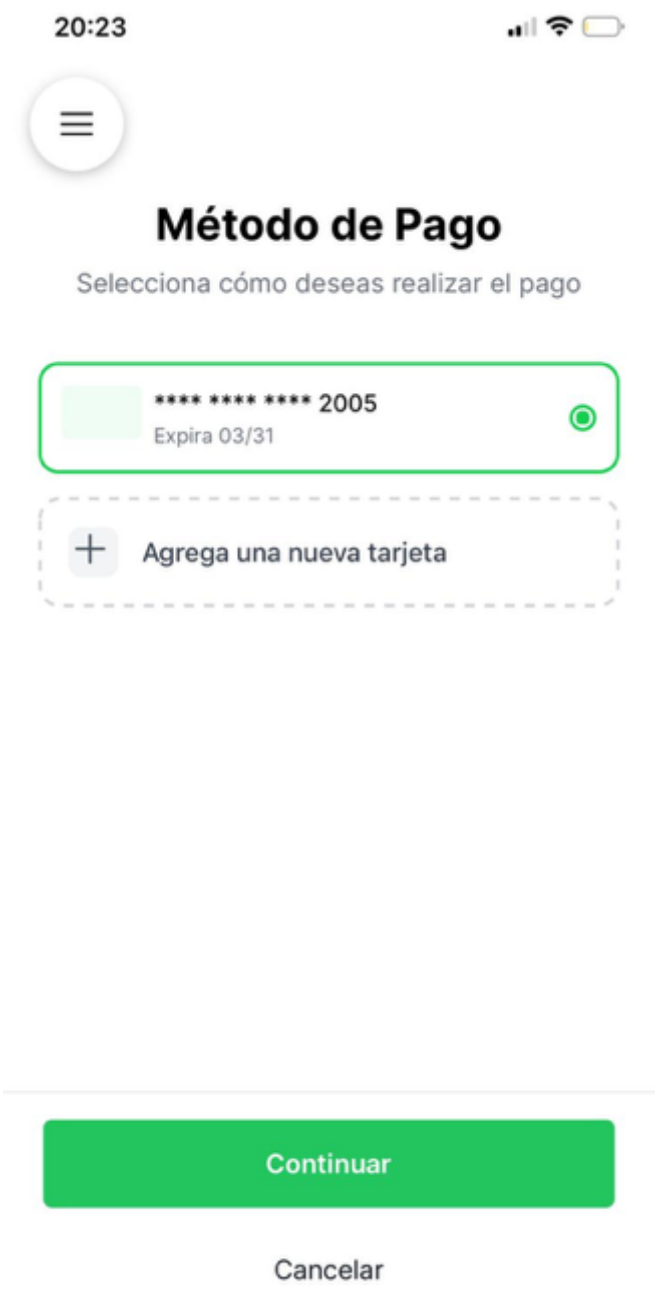


Figura 76: Visualización de los métodos de pagos guardados

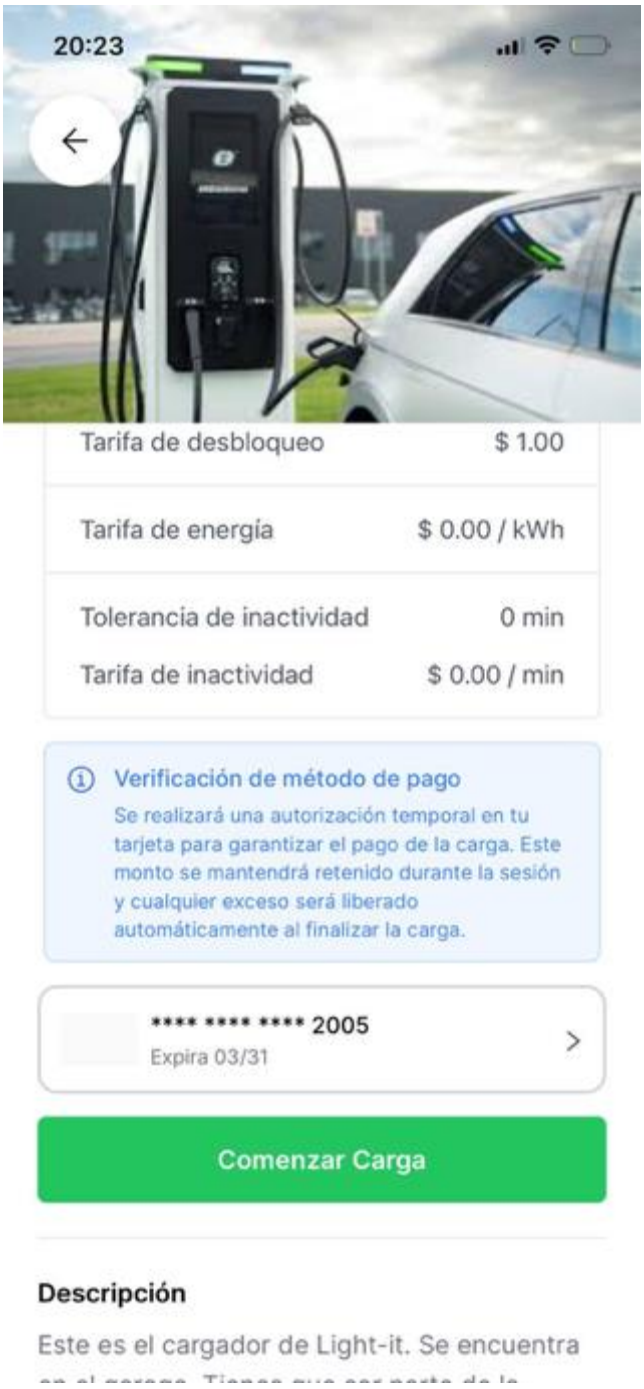


Figura 77: Visualización de carga lista para ser comenzada

Figura 78: Visualización temporizador para enchufar el vehículo

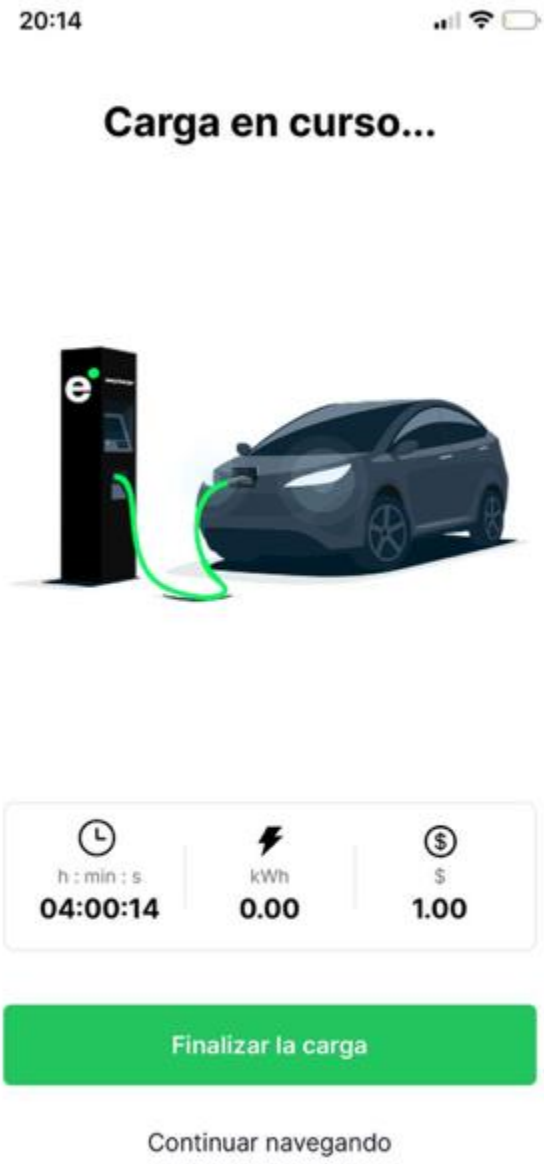


Figura 79: Visualización de detalles de carga en curso

Figura 80: Modal para detener una carga

Figura 81: Visualización de aviso para desconectar el vehículo

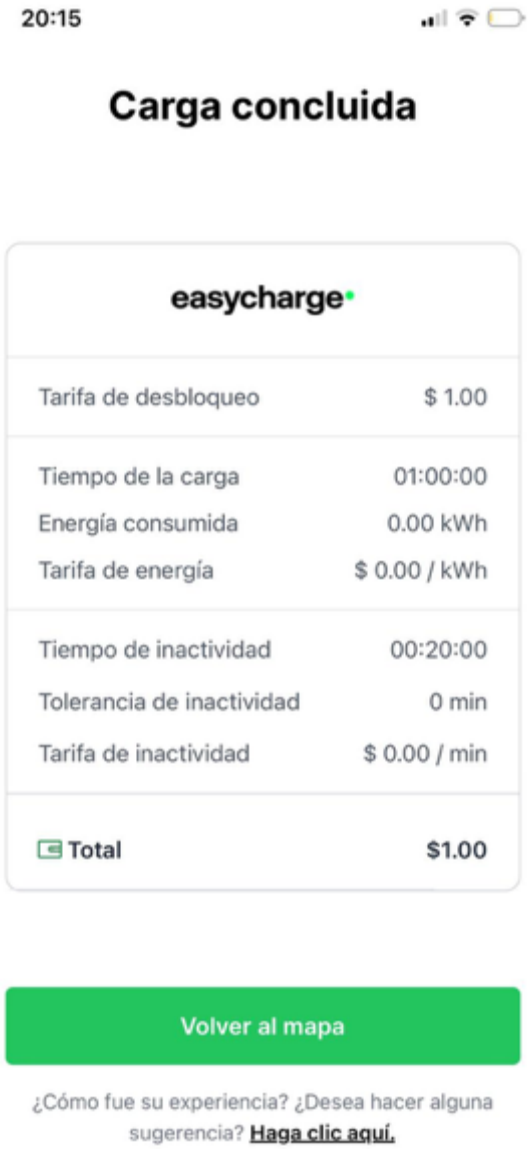


Figura 82: Visualización de resumen de costos de una carga

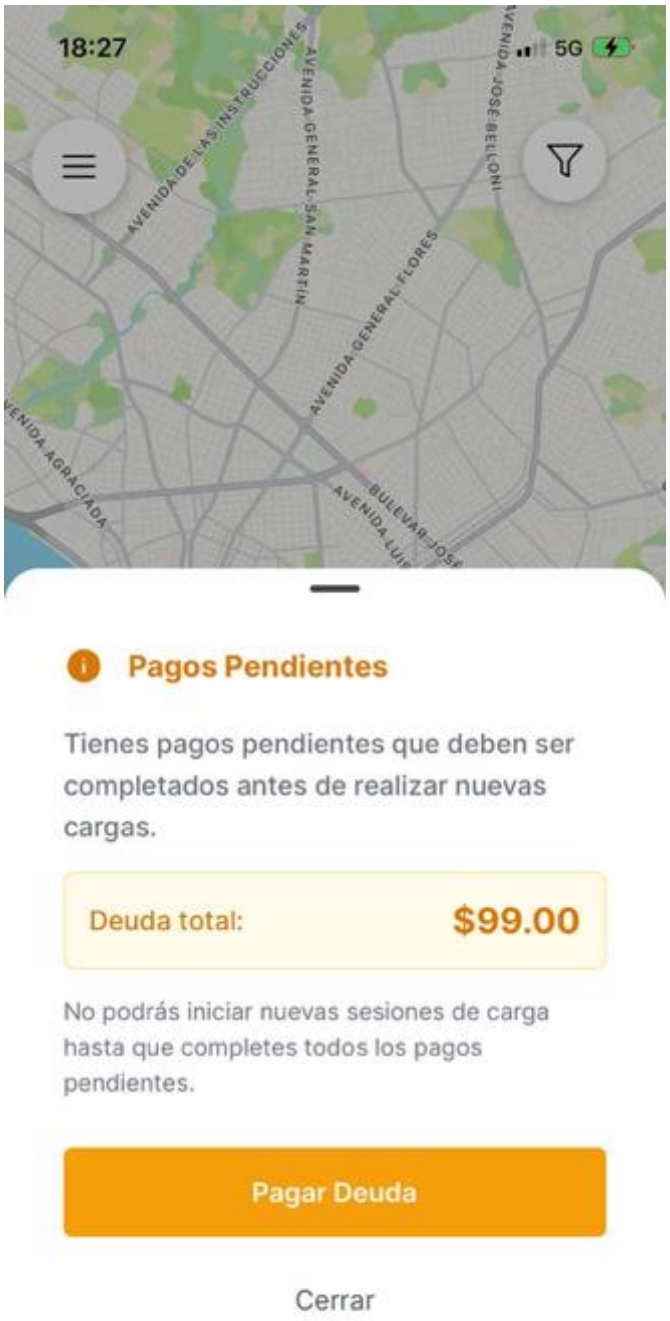


Figura 83: Mensaje de deuda a pagar

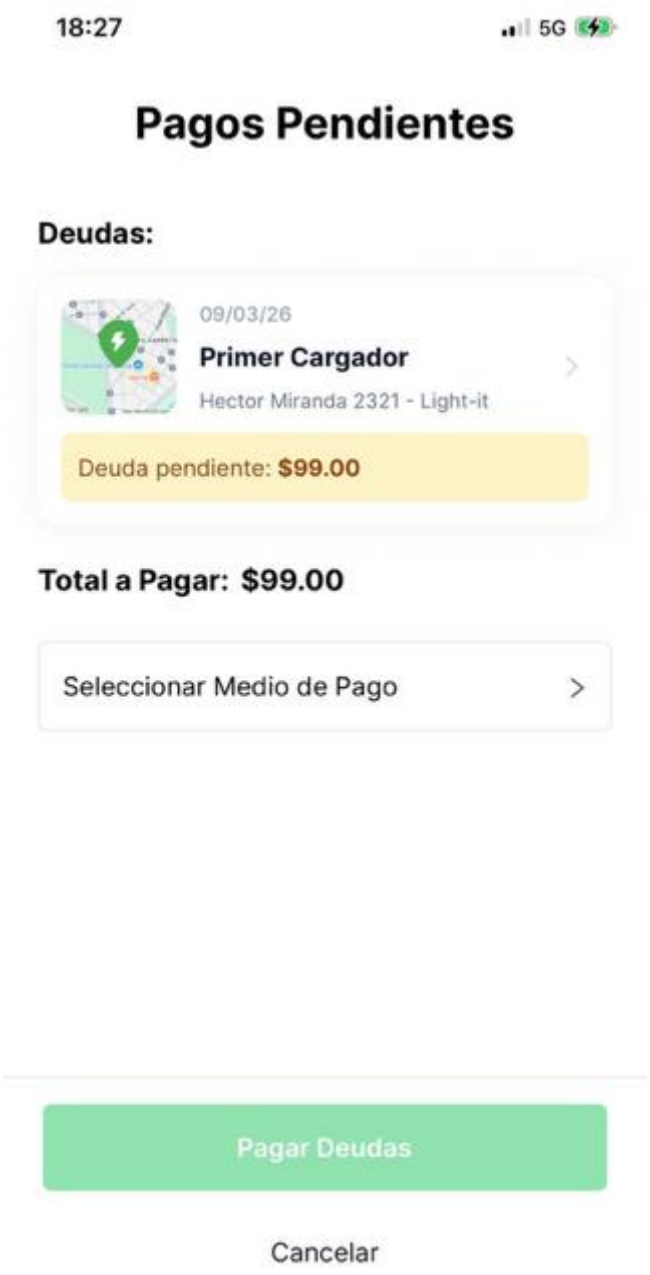


Figura 84: Página para pagar una deuda

12.8. Modelado de Datos

12.8.1. Uso de ORM

Para el modelado de datos del sistema, el equipo decidió utilizar un Object-Relational Mapper (ORM) como capa de abstracción entre el código de la aplicación y la base de datos relacional. Un ORM permite mapear clases y objetos del dominio a tablas y registros de la base de datos, de forma tal que las operaciones de lectura y escritura se realizan mediante código de alto nivel en lugar de escribir sentencias SQL.

El uso de un ORM ofrece varias ventajas. Favorece la consistencia entre el modelo de dominio y el esquema de la base de datos, reduce la cantidad de código repetitivo asociado a consultas, facilita la evolución del esquema mediante migraciones y contribuye a mejorar la mantenibilidad al centralizar la lógica de persistencia.

Dentro de las alternativas disponibles, el equipo eligió TypeORM por su integración natural con el ecosistema de TypeScript y su soporte nativo en NestJS. Esta elección permitió definir las entidades directamente como clases TypeScript decoradas con anotaciones de TypeORM, aprovechar el tipado estático en todo el ciclo de desarrollo y utilizar las herramientas de NestJS para gestionar repositorios e inyección de dependencias de manera coherente con el resto de la arquitectura.

A continuación, se presenta una tabla con todas las entidades definidas en el sistema.

Entidad	**Descripción Breve**
UserEntity	Representa a los usuarios del sistema, incluyendo sus credenciales de acceso, rol y datos básicos necesarios para operar.
OrganizationEntity	Representa a las organizaciones dueñas u operadoras de la infraestructura de carga.

MembershipEntity	Vincula usuarios con organizaciones, permitiendo que un usuario pertenezca y acceda a varias organizaciones.
ChargerBrandEntity	Registra las marcas de cargadores soportadas por el sistema.
ChargerModelEntity	Registra los modelos de cargadores soportados y los asocia a una marca.
ChargerEntity	Representa cada cargador físico instalado, con su ubicación y configuraciones técnicas principales.
ConnectorEntity	Representa cada conector individual de un cargador.
ChargeEntity	Registra cada sesión de carga, desde su inicio hasta su finalización.
MeterValueEntity	Almacena las mediciones periódicas de energía y otros valores reportados durante una sesión de carga.
PriceRateEntity	Define las tarifas aplicables a las sesiones de carga, incluyendo diferentes componentes de precio.
PaymentEntity	Registra las transacciones de pago asociadas a las sesiones de carga y otros movimientos.

RfidCardEntity	Representa las tarjetas RFID usadas para autenticación en los cargadores, incluyendo si están activas o bloqueadas.
ChargeStatusEntity	Registra los cambios de estado del cargador a lo largo de un carga

12.8.2. Diagrama MER

Se incorpora un diagrama MER (Modelo Entidad–Relación) que ilustra visualmente las entidades, sus relaciones y las claves primarias y foráneas. Este diagrama complementa la tabla de entidades, facilitando la comprensión global de la estructura de datos y del vínculo entre los distintos componentes del sistema.

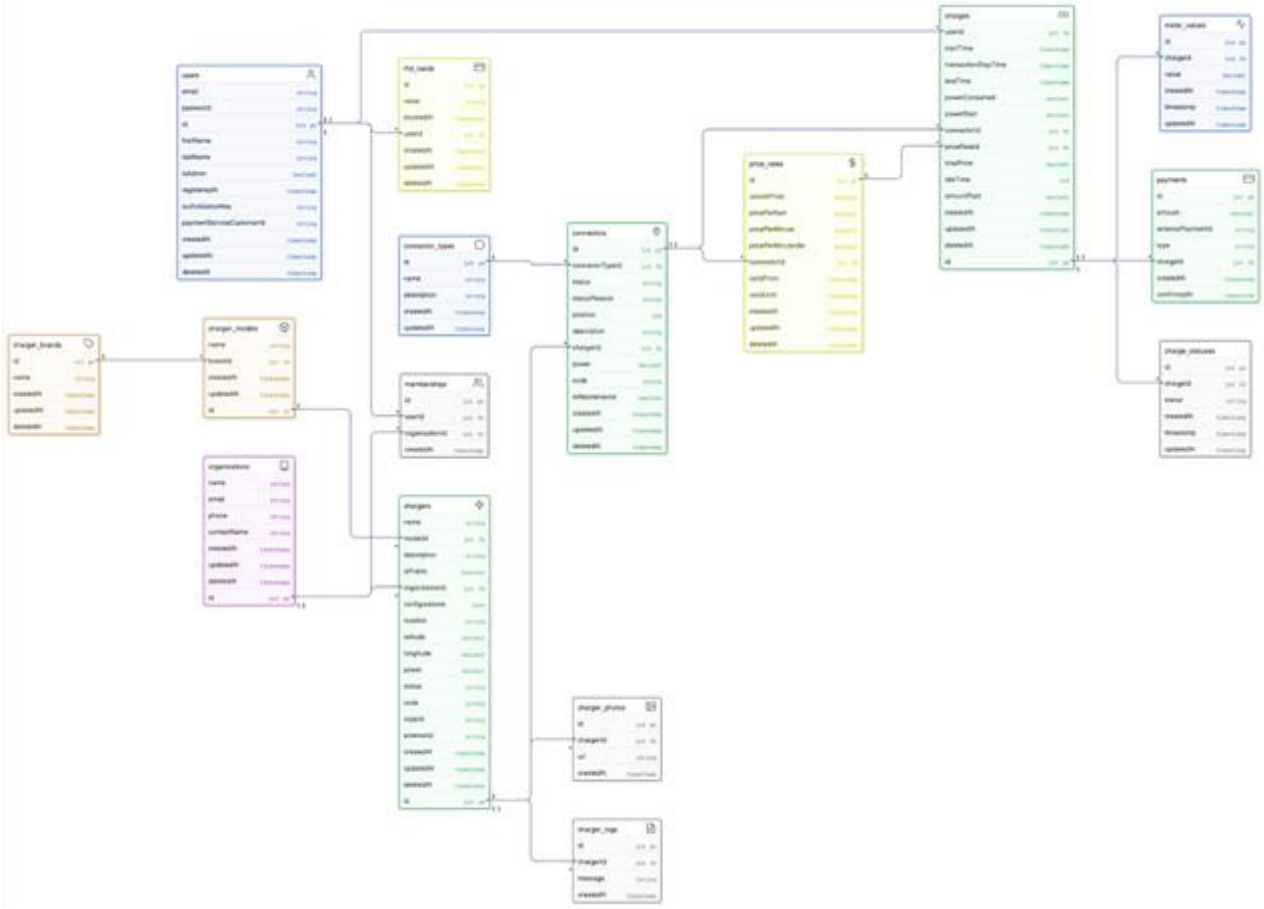


Figura 85: Diagrama MER

12.8.3. Decisiones de Diseño

En el proceso de modelado de datos, el equipo tomó una serie de decisiones de diseño orientadas a garantizar trazabilidad, consistencia e incluso buen desempeño en el sistema. Estas decisiones impactan directamente en cómo se almacenan y actualizan los datos a lo largo del ciclo de vida de las entidades.

Se optó por utilizar borrado lógico en lugar de eliminaciones físicas. Esto significa que los registros no se borran definitivamente de la base de datos, sino que se marcan como eliminados. De esta manera se preserva el historial completo de operaciones y estados, lo que resulta clave para auditoría, análisis históricos y trazabilidad.

En el caso de las tarifas, al editar un PriceRate no se modifica el registro existente, sino que se realiza un borrado lógico del registro vigente y se crea una nueva versión. Este enfoque evita inconsistencias entre los datos históricos de cargas y las reglas tarifarias que estaban vigentes en el momento de cada sesión, garantizando que los cálculos realizados en el pasado sigan siendo reproducibles.

Como se destacó en la sección Arquitectura y Diseño, el modelado de los pagos se realizó siguiendo un enfoque basado en eventos. Cada transacción queda registrada explícitamente con su tipo y su monto, lo que permite auditar el flujo completo de operaciones y disponer de un registro trazable de todos los movimientos financieros del sistema.

Se decidió introducir cierta redundancia controlada en la entidad de carga (ChargeEntity), almacenando datos que podrían calcularse a partir de otras tablas, como el precio total y la energía consumida. Aunque estos valores podrían derivarse a partir de las mediciones (MeterValueEntity), las tarifas (PriceRateEntity) y los estados (ChargeStatusEntity), guardarlos directamente mejora la performance de consultas frecuentes y simplifica reportes, reduciendo la necesidad de recomputar información en cada consulta.

12.9. Evidencia de Ceremonias de Scrum

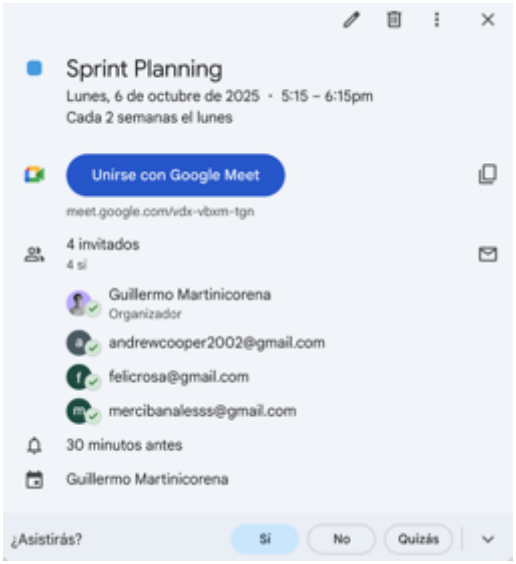


Figura 86: Invitación de Google Calendar para participar de la Sprint Planning

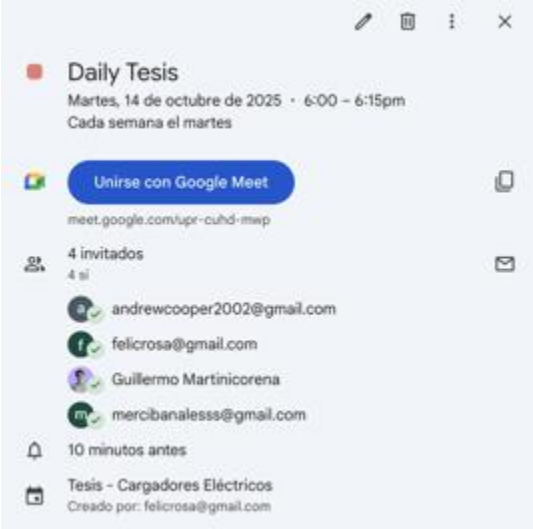


Figura 87: Invitación de Google Calendar para participar de la Daily Scrum

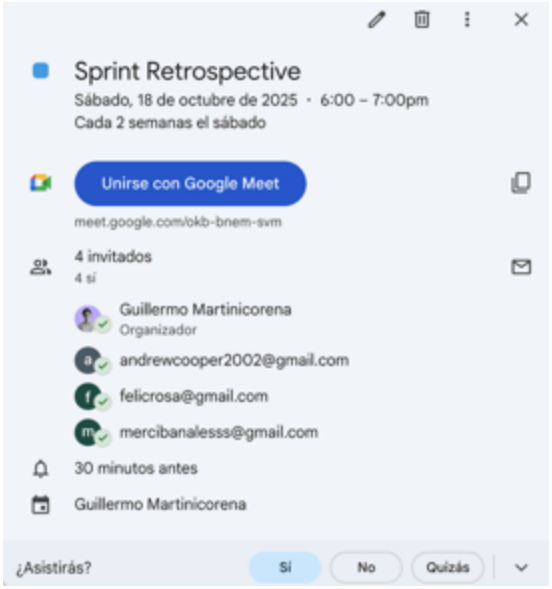


Figura 88: Invitación de Google Calendar para participar de la Sprint Retrospective

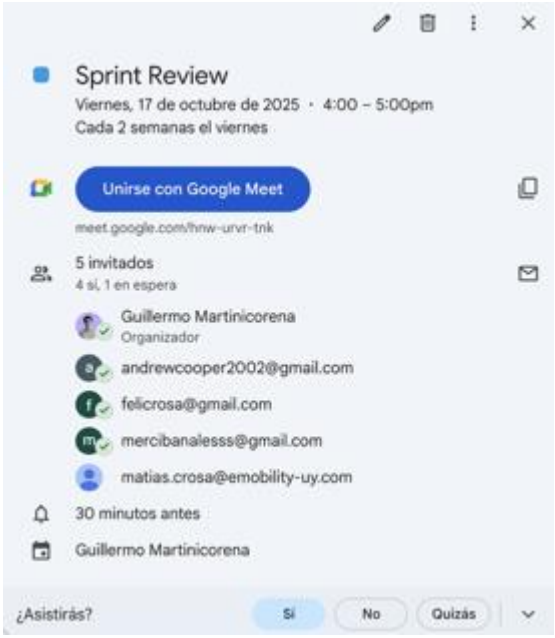


Figura 89: Invitación de Google Calendar para participar de la Sprint Review

Figura 90: Integrantes del equipo en una reunión presencial

Figura 91: Integrantes del equipo junto con los referentes del cliente, Matías y Alfredo

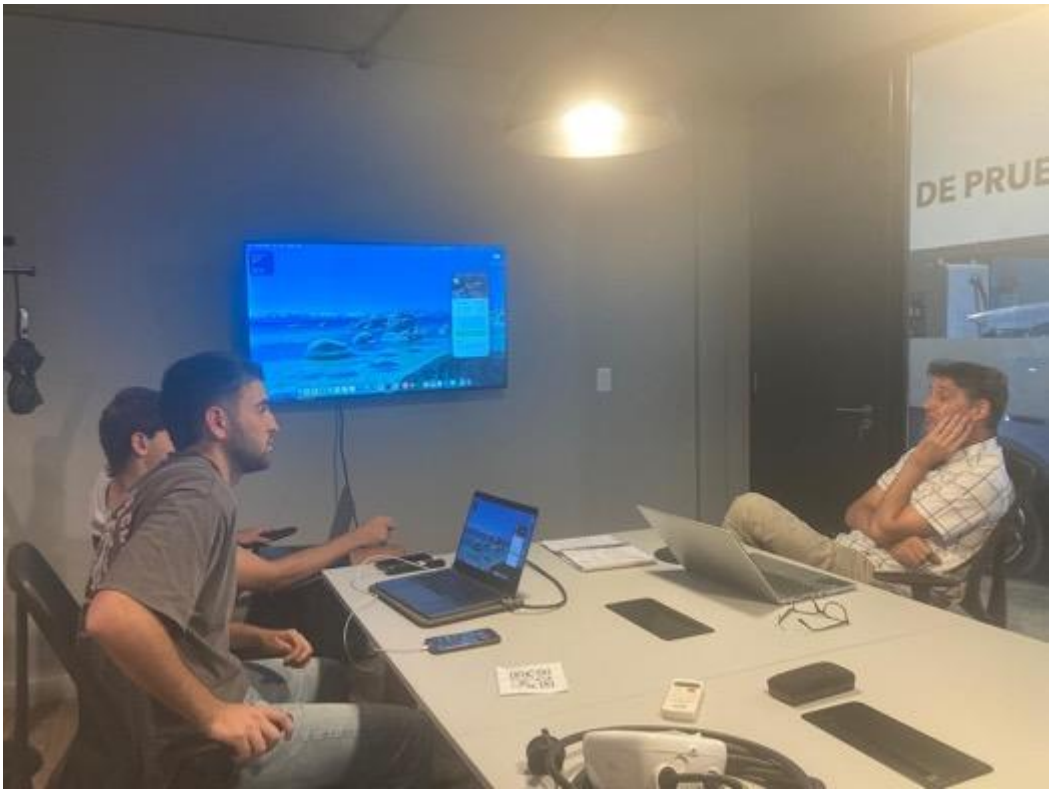


Figura 92: Integrantes del equipo junto con el referente del cliente, Alfredo

12.10. Evidencia de planilla para gestión de riesgos

Gestión de riesgos

ID	Riesgo	Categoría	Criticidad Actual (Ene)	Última Revisión	Comentarios de Seguimiento (Sprint Planning)
R1	Protocolo OCPP	Técnico	**Despreciable**	15/01/2026	Se estabilizó la comunicación tras el uso de distintos modelos de cargadores. Riesgo controlado.
R2	Integración API Pago	Técnico	**Despreciable**	15/01/26	Se estabilizó tras corregir errores en el sandbox de la pasarela.
R3	React Native	Técnico	**Despreciable**	15/01/2026	El equipo ya domina la tecnología. Se cierra el riesgo por aprendizaje cumplido.
R4	Acceso Físico	Infraestructura	**Leve**	02/01/2026	Se completaron las pruebas con distintos modelos de cargadores en las oficinas de E-Mobility Solutions. Queda residual por posibles ajustes finales.
R5	Dep. Cliente	Organizacional	**Leve**	15/01/2026	El cliente validó las últimas definiciones de UI/UX.
R6	Carga Académica	Organizacional	**Grave**	15/01/2026	Aumentó: desarrollo de la documentación técnica y arreglos finales del código coincide con vacaciones de varios integrantes.
R7	Alcance	Organizacional	**Despreciable**	15/01/2026	Alcance congelado y MVP validado. No se aceptan más cambios.

Figura 93: Planilla de gestión de riesgos

Esta planilla representa el estado de los riesgos al 15 de enero, reflejando el trabajo de seguimiento que hicimos durante todo el proyecto. No la tomamos como una foto fija, sino que la fuimos actualizando en cada *Sprint Planning* para ajustar las criticidades y acciones según lo que pasaba en cada etapa. Por eso, al final del camino, se puede ver cómo la mayoría de los riesgos técnicos terminaron controlados, mientras que otros, como la carga académica (R6), se mantuvieron en niveles más altos por la fecha de cierre.

12.11 Estándares de Código

Estándar	**Descripción**
Idioma	Todo el código, nombres de variables, funciones y comentarios se escriben en inglés.
Formato	Se mantiene un formato uniforme con indentación consistente, espacios adecuados y longitud de línea limitada para mejorar la legibilidad.
Linting	ESLint se configura utilizando las reglas `eslint:recommended` y `typescript-eslint` con type-checking habilitado.
Convención de nombres	Clases, interfaces y tipos utilizan PascalCase. Variables y funciones utilizan camelCase. Constantes utilizan UPPER_SNAKE_CASE.
Estructura de archivos	Los archivos utilizan kebab-case y un sufijo que identifica su rol dentro del sistema (controller, action, entity, request, representation, module o test).
Funciones	Las funciones deben ser *arrow functions*, tener una única responsabilidad y mantenerse lo más simples posible.
Manejo de errores	Los errores deben manejarse mediante excepciones controladas y estructuras try/catch cuando corresponda.
Comentarios	Se utilizan únicamente cuando agregan valor, por ejemplo para documentar decisiones técnicas o aclarar comportamientos no evidentes del código.
Testing	Los tests siguen el patrón Arrange / Act / Assert. Cada prueba debe validar un único

	comportamiento del sistema.
Logs	El sistema utiliza un mecanismo de logging centralizado para registrar eventos relevantes durante la ejecución del backend.
Git Hooks	Se utilizan hooks de Git para validar el formato del código, el linting y el build del proyecto antes de permitir commits o pushes.
Conventional Commits	Los mensajes de commit siguen el formato `<tipo>[alcance opcional]: <descripcion>`.
Nomenclatura de ramas	Las ramas siguen el patrón `(feature|bugfix|refactor|setup|docs)/descripcion-en-minusculas`.
Tipado	Se prioriza el uso de tipado estático de TypeScript. Las funciones deben declarar explícitamente el tipo de retorno y de sus parámetros. Se evita el uso de any siempre que sea posible, priorizando interfaces o tipos específicos.

12.12 Evidencia complejidad ciclomática

Mediante el uso de SonarQube se realizó un análisis de la complejidad ciclomática para evaluar la calidad del sistema y su cumplimiento con el escenario de calidad definido para la mantenibilidad.

En este anexo se presentan los resultados obtenidos, donde se observa que la complejidad ciclomática de los componentes críticos se mantiene por debajo de 10.

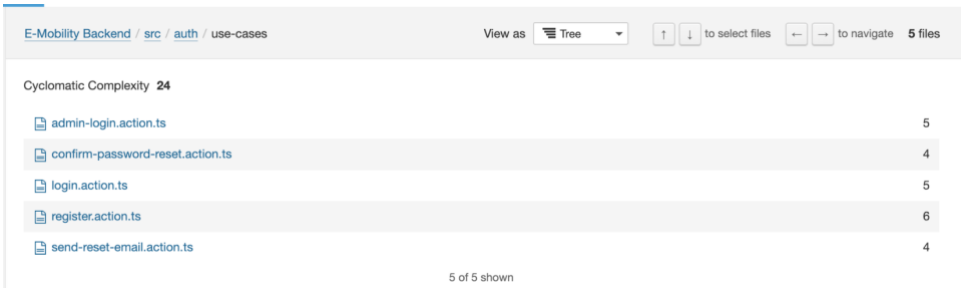


Figura 94: Complejidad ciclomática en *use-cases,* sección de autenticación.

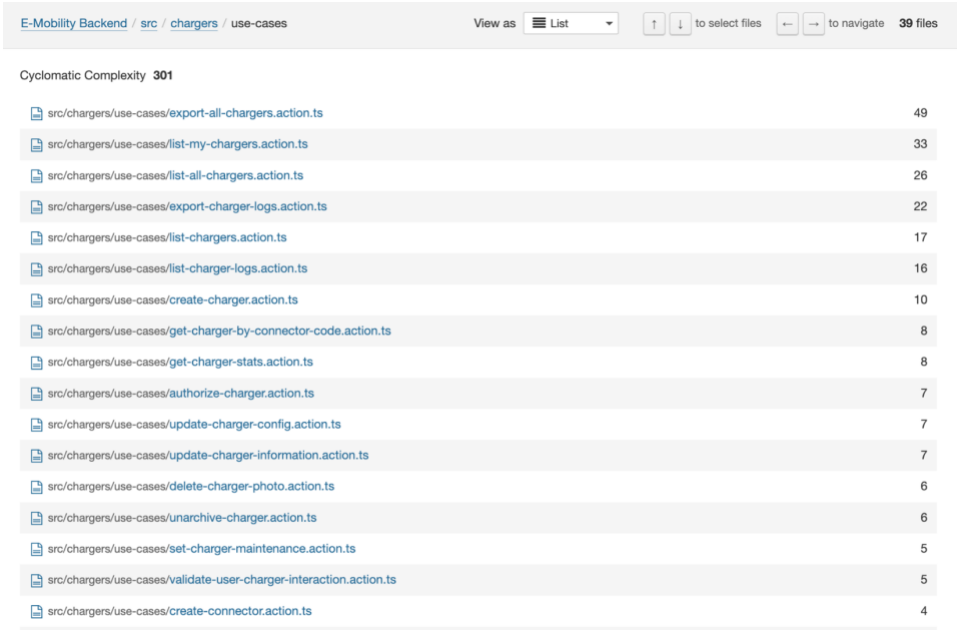


Figura 95: Complejidad ciclomática en *use-cases,* sección de cargadores.

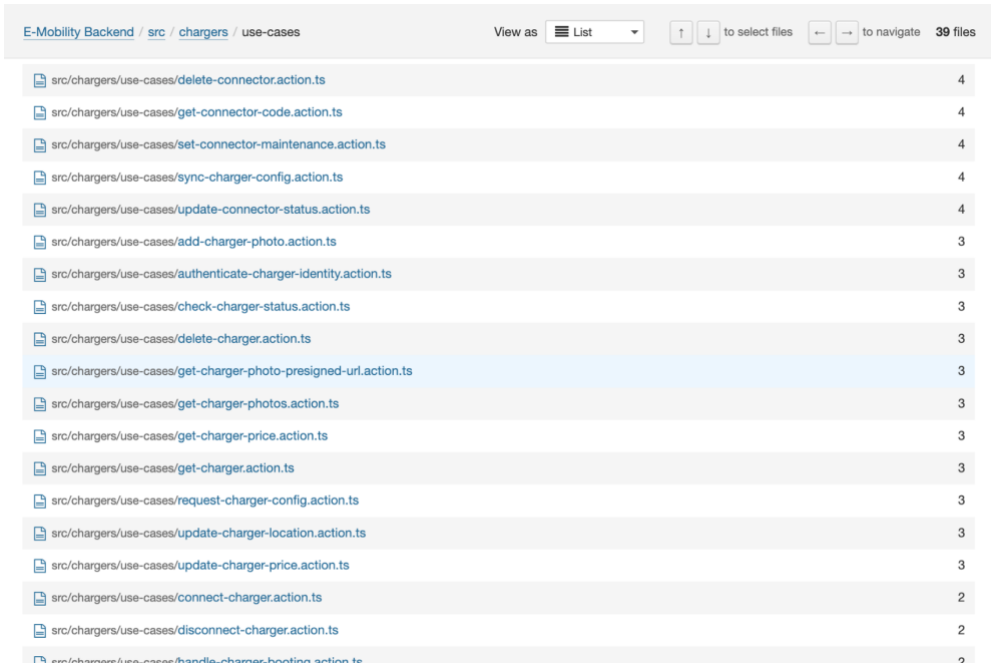


Figura 96: Complejidad ciclomática en *use-cases,* sección de cargadores.

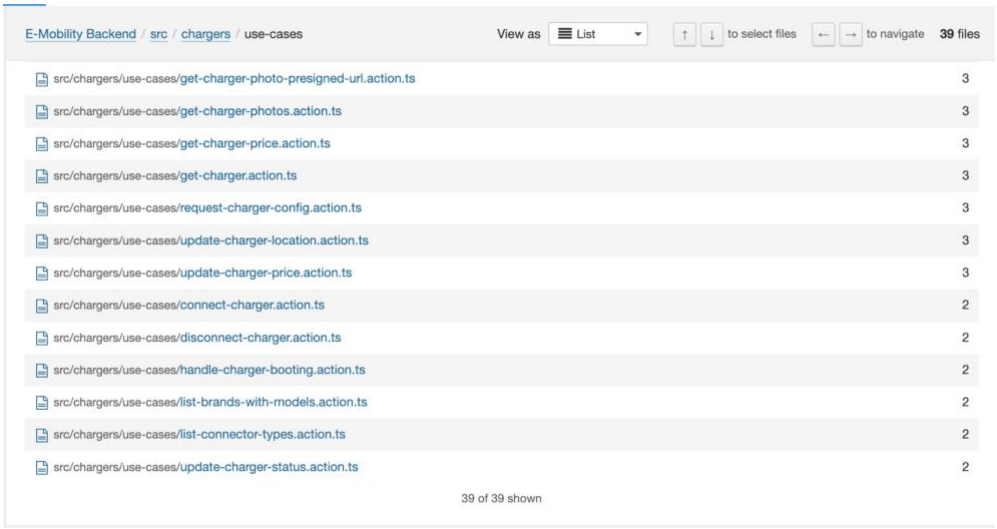


Figura 97: Complejidad ciclomática en *use-cases,* sección de cargadores.

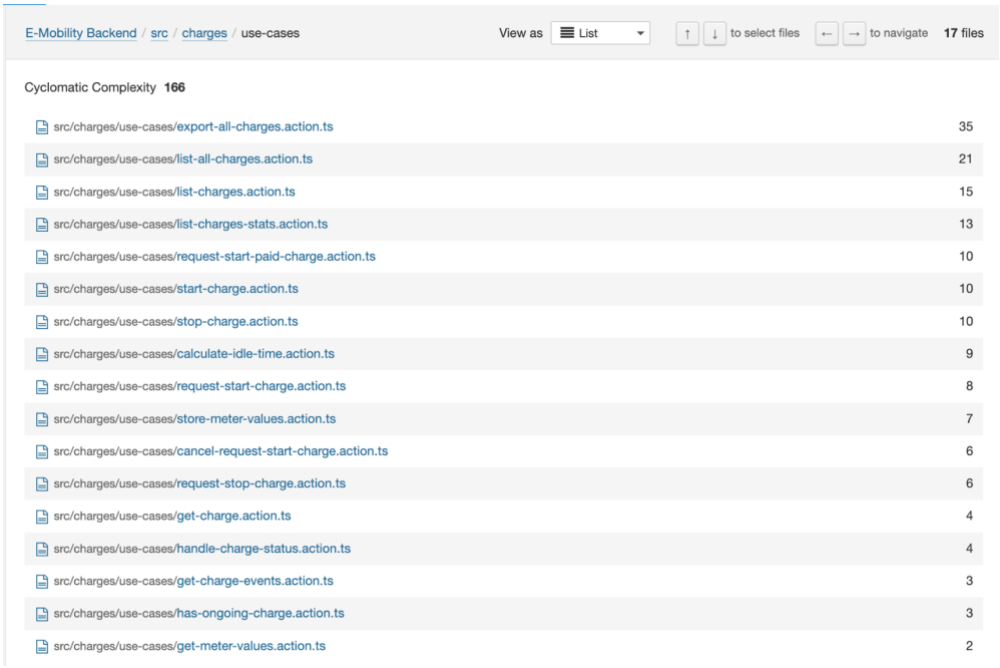


Figura 98: Complejidad ciclomática en *use-cases,* sección de cargas.

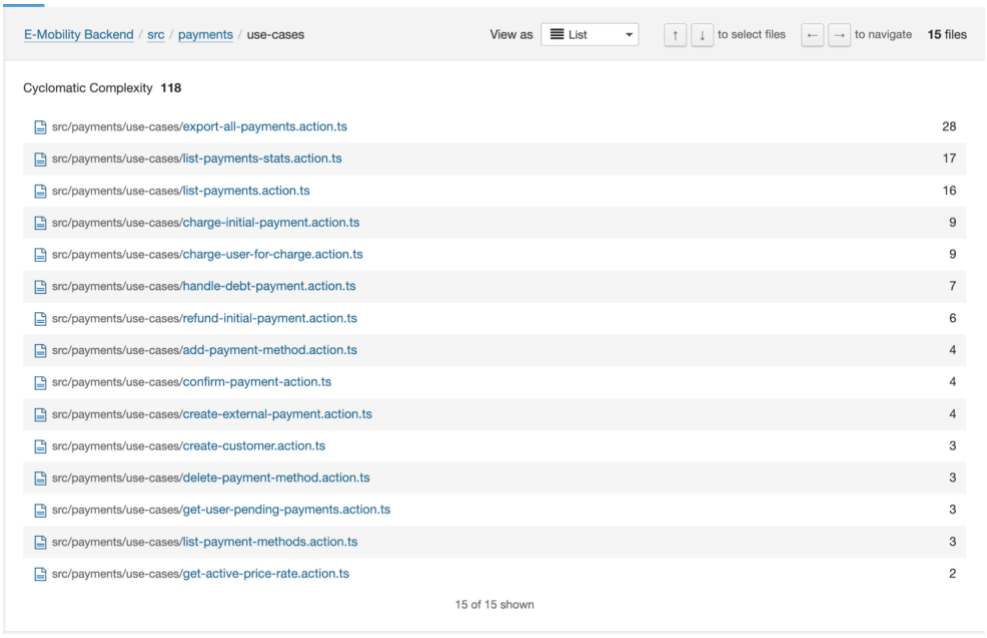


Figura 99: Complejidad ciclomática en *use-cases,* sección de pagos.

Se identifican algunos casos puntuales con valores más altos, principalmente en casos de uso asociados a listados y exportaciones. Su mayor complejidad se explica por la variabilidad introducida por múltiples filtros y combinaciones posibles. Al no corresponder

a flujos críticos del dominio, se considera que este nivel de complejidad es aceptable dentro del contexto del sistema.

12.13 Evidencia de un reporte de *bug*

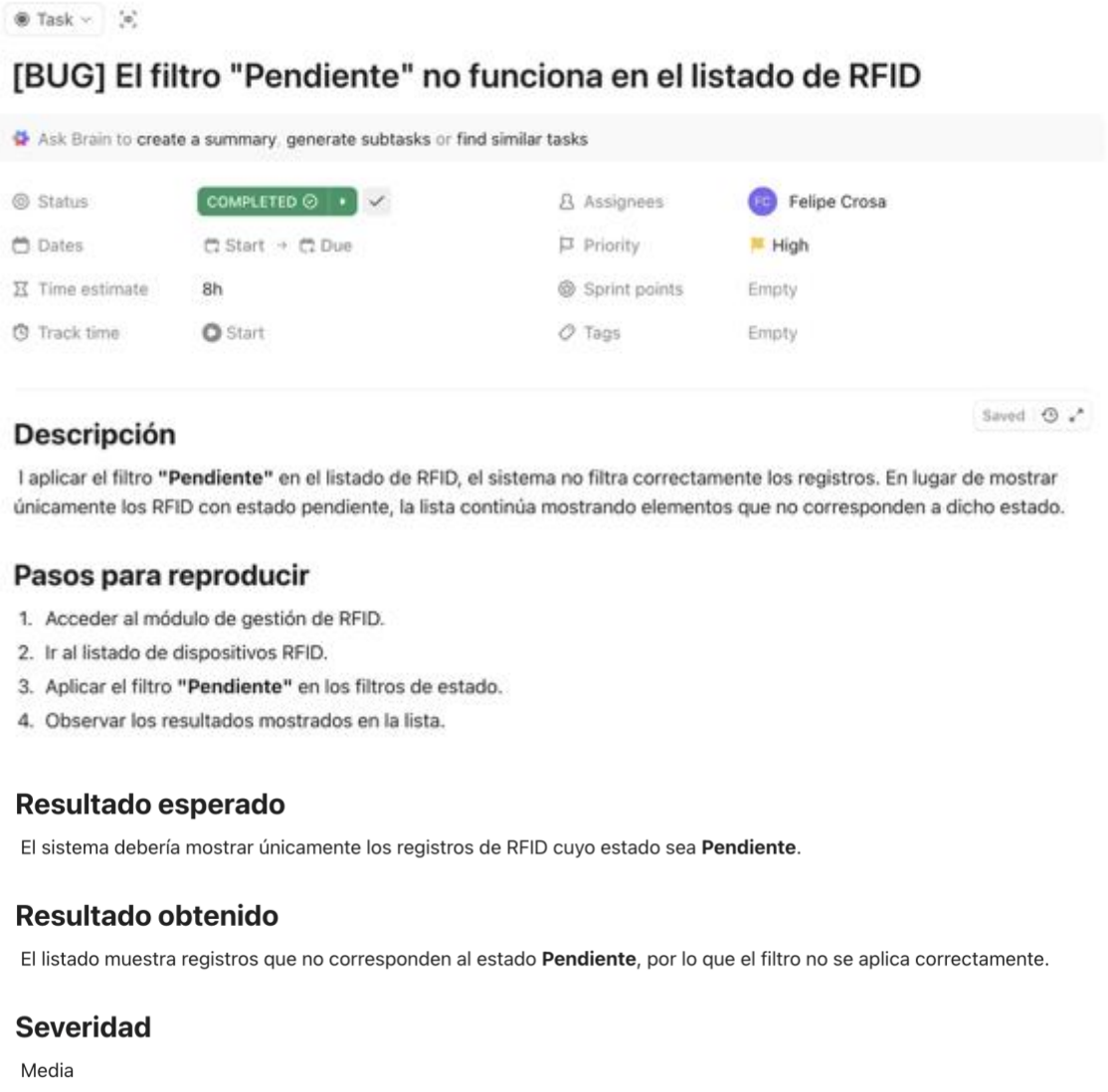


Figura 100: Reporte de bug

12.14. Plan de Validación Operativa

Este documento define un plan de validación operativa del nuevo sistema de gestión de cargadores de cara a su salida a producción y migración completa desde la plataforma actual. Su propósito es reducir el riesgo técnico y operativo, proteger la experiencia de los usuarios finales y resguardar la imagen de E-Mobility Solutions mediante una transición gradual, controlada y medible.

Este documento constituye un plan y una recomendación basada en estimaciones preliminares. Tanto los tiempos propuestos como el tamaño del equipo sugerido son orientativos y pueden no ajustarse exactamente a la realidad, por lo que deberán revisarse y adaptarse en función del contexto, la disponibilidad de recursos y las decisiones que se vayan tomando.

En el contexto actual, la solución ya validada en entornos de prueba controlados, donde se comprobó que los principales flujos funcionan y que la comunicación con cargadores OCPP es correcta. Sin embargo, todavía no se la ha sometido a un uso sostenido por usuarios reales, con múltiples cargadores en simultáneo y condiciones de conectividad variables, que son las que realmente se encuentran en producción. Por este motivo, es imprescindible realizar una instancia intermedia de validación operativa, que permita observar el rendimiento y el comportamiento del sistema bajo condiciones reales, pero con un alcance acotado y manejable.

12.14.1. Objetivos del plan y criterios de evaluación

Los objetivos principales de la validación operativa son verificar la estabilidad técnica del sistema en un entorno semejante a producción. La instancia de validación busca detectar tempranamente problemas de rendimiento, disponibilidad y usabilidad, de modo de corregirlos antes de escalar la solución a toda la red de cargadores, y generar evidencia objetiva (métricas, registros de sistema y *feedback* estructurado) que respalde la decisión de avanzar hacia una salida a producción completa.

12.14.1.1 Evaluación

Para medir el grado de cumplimiento de estos objetivos se definen una serie de indicadores clave de desempeño (KPIs), que deberán evaluarse con el sistema operando en un contexto y bajo una carga lo más representativa posible de un entorno de producción real.

- En términos de disponibilidad, se propone como meta que el sistema se mantenga operativo al menos el 99,9% del tiempo durante el período de validación.
- El tiempo de inicio de una carga debería ser siempre inferior a 10 segundos desde la acción del usuario hasta que el equipo indique que la sesión ha comenzado.
- El número de fallas críticas no resueltas que impidan sistemáticamente el uso del servicio debería ser nulo.
- Los conductores valoren la experiencia de manera positiva con un puntaje promedio igual o superior a 4 sobre 5 en las encuestas de satisfacción.
- El personal operativo de E-Mobility Solutions brinde un *feedback* favorable sobre la facilidad de operación, el monitoreo y la resolución de incidencias.

Se recomienda mantener un umbral exigente para asegurar que la salida a producción se realice con un nivel de confianza alineado con la criticidad del servicio.

12.14.2. Ejecución

En esta sección se detalla cómo se propone llevar a cabo la ejecución del plan de validación operativa. El enfoque adoptado es incremental y no de cambio total inmediato, justamente para poder abordar los problemas de manera gradual, limitar el impacto de posibles fallas y reducir el riesgo asociado a la migración. La duración total estimada para completar todas las fases del plan es de aproximadamente tres meses, sujeta a ajuste según los resultados y el contexto operativo.

12.14.2.1. Prerrequisitos

Previo al inicio de la validación, es necesario cumplir con los siguientes prerrequisitos mínimos:

- Sistema desplegado: el sistema debe estar correctamente instalado y configurado en el entorno de infraestructura definido para la validación, siguiendo la guía de despliegue.
- Aplicación móvil accesible: la app debe estar disponible para los usuarios de prueba, ya sea mediante ambientes específicos de testing o publicada en las tiendas de iOS y Android.
- Selección de cargadores: se deben elegir los cargadores que participarán de la prueba, procurando que sean puntos no críticos pero representativos, asumiendo que durante este período pueden producirse fallas y tiempos fuera de servicio.
- Usuarios de validación: es necesario contar con un conjunto de usuarios dispuestos a participar en la prueba, utilizar la nueva solución y proporcionar *feedback*.
- Equipo de desarrollo disponible: se debe definir un equipo de desarrollo responsable de esta fase, con disponibilidad para atender incidencias, analizar problemas y ajustar el sistema durante todo el período de validación.

12.14.2.2. Fases de Validación

La ejecución del plan se organiza en fases sucesivas, que permiten ir incrementando progresivamente el alcance del sistema en términos de cantidad de cargadores y usuarios involucrados. A lo largo de cada fase se monitorea de forma continua la estabilidad de las conexiones con los cargadores, la disponibilidad del sistema, los logs de la aplicación y de OCPP, y se compila *feedback* de los usuarios (encuestas y entrevistas) para identificar posibles problemas de funcionamiento o de experiencia.

Sobre la información recolectada, se analizan y solucionan las fallas que se vayan detectando y, una vez que se alcanza un punto de estabilidad aceptable en una fase determinada, se avanza a la siguiente etapa de validación con un alcance mayor.

En una primera fase se trabaja con un conjunto muy reducido de cargadores ubicados en entornos controlados, idealmente dentro de las oficinas o instalaciones internas de E-Mobility Solutions, y con un grupo pequeño de usuarios invitados. El objetivo es validar el comportamiento del sistema en condiciones reales pero con máxima facilidad de acceso físico y soporte técnico, reduciendo el impacto de posibles fallas y permitiendo ajustes rápidos.

En la segunda fase se extiende la validación a cargadores instalados en ubicaciones externas, comenzando con un número acotado de equipos y aumentando progresivamente hasta alcanzar cerca de 50 cargadores. Esta fase debe incorporar diversidad de modelos, clientes y condiciones de conectividad. y busca aproximarse al contexto real de operación, permitiendo observar cómo se comporta el sistema bajo una carga creciente y en escenarios heterogéneos similares a los de producción.

12.14.2.3. Validaciones Complementarias

Durante el período de validación operativa también es recomendable realizar pruebas adicionales sobre el sistema, como pruebas de carga con herramientas como k6 para evaluar su rendimiento bajo distintos niveles de demanda. Asimismo, resulta conveniente ejecutar pruebas orientadas a identificar vulnerabilidades y debilidades de seguridad, como escaneos de vulnerabilidades y pruebas de penetración sobre las interfaces web y APIs del sistema, con el fin de reforzar la protección antes de su puesta en producción.

12.14.2.4. Equipo Mínimo Recomendado

Para llevar adelante la fase de migración y validación operativa se propone trabajar con un equipo reducido pero claramente definido, capaz de responder con rapidez a los problemas que puedan surgir y de iterar sobre la solución durante el período del plan. En este contexto, se considera suficiente el siguiente esquema mínimo:

- 1 desarrollador junior full time (8hs) con conocimientos full-stack, dedicado al soporte diario del sistema, al análisis de logs, a la corrección de errores simples y a la implementación de pequeños ajustes o mejoras que se detecten durante la validación.
- 1 desarrollador senior part time (4hs), responsable de la toma de decisiones técnicas más complejas, los cambios de arquitectura, la gestión de infraestructura, la revisión de cambios, la resolución de incidencias críticas y la orientación del trabajo del desarrollador junior.
- 1 empleado de E-Mobility Solutions, disponible para asistir con la configuración de los cargadores, coordinar acciones en campo y canalizar el *feedback* operativo.

A esta estructura se le pueden sumar roles auxiliares que facilitan el proceso:

- 1 analista de calidad, encargado de planificar y ejecutar pruebas funcionales y de regresión, documentar incidencias y verificar el cumplimiento de los criterios de aceptación definidos.
- Horas disponibles de un diseñador, orientadas a evaluar la usabilidad del sistema con usuarios reales o escenarios representativos y proponer mejoras en los flujos de interacción y en la experiencia de uso de las aplicaciones web y móvil.

Antes de comenzar la validación es necesario que este equipo disponga de al menos una semana de capacitación para contextualizarse con la solución técnica, comprender la arquitectura y los flujos principales, conocer las herramientas y alinearse con los procesos internos de E-Mobility Solutions. De esta forma, la fase de validación puede comenzar con un equipo ya familiarizado con el sistema y en condiciones de reaccionar de manera efectiva ante los incidentes que puedan aparecer.

12.14.3. Salida a Producción

Cuando el sistema haya alcanzado un nivel de estabilidad consistente y se hayan cumplido los objetivos y KPIs definidos en el plan de validación operativa, se podrá avanzar hacia la salida total a producción.

Desde el punto de vista operativo, la salida a producción supone incorporar todos los cargadores restantes al nuevo backend, configurándolos desde el dashboard de administración y actualizando la URL OCPP de cada equipo para que apunte al nuevo sistema. A partir de ese momento, la gestión de sesiones de carga, monitoreo y facturación pasa a depender de la nueva plataforma. En paralelo, la aplicación móvil debe publicarse de forma abierta en las tiendas correspondientes, quedando disponible en los buscadores para todos los usuarios finales, y acompañarse de una campaña de comunicación que explique el cambio e invite a descargar la nueva app.

En relación con la plataforma anterior (VoltBras), se plantea como opción cortar la relación de forma definitiva una vez que el nuevo sistema esté estabilizado o, alternativamente, mantenerla durante un período acotado como mecanismo de respaldo mientras se consolida la confianza en la solución propia. Esta decisión igual queda sujeta a los costos asociados a operar ambas plataformas en paralelo y de la evidencia recogida durante la validación.

12.14.3.1. Equipo de Mantenimiento

El equipo requerido una vez que el sistema se encuentra en un entorno productivo varía según las necesidades y el nivel de servicio objetivo. En un escenario donde el foco principal sea mantener la solución existente y resolver los problemas que se vayan presentando, puede sostenerse una estructura reducida compuesta únicamente por el desarrollador senior, encargado tanto del mantenimiento correctivo como de los ajustes menores necesarios.

Si se busca continuar incorporando funcionalidades y evolucionar la solución, resulta pertinente mantener también al desarrollador junior, de modo de contar con capacidad operativa adicional para atender nuevas demandas sin comprometer las tareas de mantenimiento. Dado el carácter crítico del servicio, es importante asimismo contemplar un esquema de guardias, asegurando que haya siempre un desarrollador disponible para responder ante alertas del sistema e incidentes fuera del horario habitual.

12.14. Costos

Los costos asociados a la fase de validación operativa y a una eventual salida a producción no se incluyen en el presente plan, dado que dependen de múltiples factores externos al alcance de esta etapa académica. Entre los principales determinantes se encuentran la conformación del equipo técnico necesario para sostener las operaciones en un entorno productivo y las decisiones vinculadas a la infraestructura tecnológica, como la elección de servicios en la nube, escalabilidad de los recursos y requerimientos de monitoreo o soporte.

Estas variables pueden modificar significativamente la estimación económica final, por lo que resulta prudente excluirlas del presupuesto del presente proyecto. En consecuencia, este documento se concentra únicamente en las actividades de desarrollo, pruebas y validación funcional del prototipo, dejando la planificación detallada de costos operativos y de producción para una fase posterior, en la que se cuente con información concreta sobre las condiciones de despliegue y nivel de servicio requerido.

12.15. Guia de Despliegue

Este documento técnico describe los detalles del despliegue del sistema en los distintos ambientes definidos para el proyecto, indicando precondiciones, pasos técnicos y posibles mejoras.

12.15.1. Despliegue local

El despliegue local es importante porque permite al equipo de desarrollo trabajar de forma aislada, reproducible y consistente, reduciendo errores derivados de diferencias entre entornos y facilitando la ejecución y depuración del sistema en el día a día. Cada repositorio del proyecto incluye un archivo *markdown* README.md donde se documenta el proceso específico de instalación y despliegue, indicando prerrequisitos, comandos a ejecutar y consideraciones particulares de cada componente.

Para el despliegue local de los servicios backend se utiliza Docker mediante el uso de docker-compose, lo que permite levantar de manera orquestada todos los servicios auxiliares necesarios (bases de datos, colas de mensajes, servicios de monitoreo, entre otros) con un único comando y una configuración declarativa versionada junto al código fuente. Esta estrategia facilita que cualquier integrante del equipo pueda reproducir el entorno en su máquina con el mismo conjunto de servicios, versiones e interconexiones definidos para el proyecto.

Para la conexión con cargadores mediante el protocolo OCPP, durante el desarrollo local se puede utilizar una herramienta de túneles como ngrok para exponer el servidor OCPP que corre en la máquina del desarrollador a internet a través de una URL pública, o bien conectar los cargadores en la misma red local y configurar la dirección del servidor utilizando la IP interna del equipo que aloja el backend. De este modo es posible probar, en un entorno controlado, tanto la comunicación websocket OCPP como los distintos flujos de inicio, seguimiento y finalización de sesiones de carga antes de pasar a ambientes en la nube.

12.15.2. Despliegue en la nube

El despliegue en la nube describe los elementos y consideraciones necesarias para tener el sistema corriendo en una plataforma de infraestructura en Amazon Web Services (AWS). En este entorno se utilizan distintos servicios de AWS para alojar los componentes del sistema, por ejemplo servicios de cómputo para el backend, servicios gestionados de base de datos y servicios de almacenamiento para archivos y recursos estáticos.

La configuración detallada de cada servicio utilizado y la forma en que se conectan entre sí se desarrolla en las subsecciones siguientes, donde se describe cómo se despliega cada componente del sistema dentro de la infraestructura de AWS.

12.15.2.1. Elección de la región

Para el prototipo fue priorizado minimizar los costos y se utilizó la región us-east-1 de AWS. Para las pruebas de validación operativa y el uso productivo se debe considerar la opción de utilizar regiones más cercanas a los usuarios finales, con el objetivo de reducir la latencia y mejorar el rendimiento del sistema, evaluando el impacto en el incremento del costo de la infraestructura.

12.15.2.2. Consideraciones de seguridad

Para el prototipo, la seguridad no fue la mayor prioridad y la infraestructura resultante es relativamente permisiva en cuanto a los accesos definidos en los security groups y a la comunicación entre servicios. Para una eventual salida a producción se identifican oportunidades de mejora relevantes, entre ellas evaluar el uso de security groups más estrictos (limitando puertos y orígenes de tráfico al mínimo necesario), el despliegue de los componentes dentro de una VPC con subredes adecuadamente segmentadas y el manejo centralizado de credenciales de aplicación mediante servicios gestionados como AWS Secrets Manager, con el objetivo de reducir la superficie de ataque y mejorar la protección de la información sensible.

12.15.2.3. Consideraciones de rendimiento

Para el prototipo, al tratarse de pruebas controladas y no intensivas, se priorizó evitar costos elevados utilizando instancias livianas y de bajo cómputo. Durante la validación operativa y de cara a un uso productivo, se debe evaluar la necesidad de escalar verticalmente las instancias en caso de que su capacidad de cómputo se convierta en un cuello de botella para la carga esperada del sistema.

12.15.2.4. Consideraciones de observabilidad

En el prototipo, la observabilidad de la infraestructura se apoya principalmente en los registros generados por los servicios y recopilados en CloudWatch Logs. Existen mejoras posibles en este aspecto, como la configuración de alertas sobre métricas y eventos relevantes, y la incorporación de herramientas especializadas como Sentry[32] para centralizar errores de aplicación y obtener mayor visibilidad sobre el comportamiento del sistema en tiempo de ejecución.

12.15.2.5. Consideraciones de despliegue

El proceso de despliegue actual se realiza de forma manual, publicando las últimas versiones de los componentes a medida que se introducen cambios en el sistema. A futuro, este proceso puede mejorarse mediante la incorporación de una pipeline de integración y despliegue continuo, por ejemplo utilizando infraestructura como código con Terraform y flujos automatizados con GitHub Actions, de forma de estandarizar los pasos de despliegue, reducir errores manuales y favorecer la consistencia entre entornos.

12.15.2.6. Configuración de RDS

Se utiliza Amazon RDS como servicio de base de datos relacional administrada para alojar la base de datos principal del sistema E-Mobility Solutions. Para el prototipo se creó una nueva instancia destinada a la base de datos de la plataforma, utilizando un motor relacional (se sugiere PostgreSQL) y una clase de instancia db.t3.micro, suficiente para las necesidades de carga y prueba del entorno de desarrollo.

[32] https://sentry.io

De cara a la salida a producción, se identifican algunas consideraciones relevantes que quedaron fuera del alcance del prototipo pero resultan importantes:

- Habilitar backups automáticos para reducir el riesgo de pérdida de información.
- Configurar redundancia multizona para mejorar la disponibilidad del servicio.
- Activar la encriptación de la base de datos y de los volúmenes asociados para fortalecer la seguridad y disminuir la exposición frente a posibles vulnerabilidades.

12.15.2.7. Configuración de Amazon MQ

Para el broker de comunicación se utilizó Amazon MQ con un broker de RabbitMQ como solución de mensajería entre servicios. Para el prototipo se configuró un broker de tipo single-instance con una clase de instancia mq.t3.micro y la versión 3.13.7 de RabbitMQ, suficiente para las necesidades de carga y pruebas en esta etapa.

De cara a un uso productivo, se identifican las siguientes consideraciones relevantes:

- Evaluar el uso de un esquema de cluster-deployment en lugar de una única instancia para mejorar la disponibilidad y la tolerancia a fallos.
- Habilitar el registro de logs de Amazon MQ en CloudWatch para asegurar la observabilidad del servicio y facilitar el diagnóstico de incidentes.

12.15.2.8. Configuración de Elasticache

Para el despliegue del prototipo, dado que no se utilizaron VPC, el servicio de Redis se gestionó mediante la oferta gratuita oficial del proveedor, suficiente para las necesidades de desarrollo y prueba. Sin embargo, para una infraestructura productiva se sugiere crear un clúster de caché utilizando Amazon ElastiCache para Redis, desplegado dentro de una VPC que permita gestionar de forma controlada los permisos de acceso, restringiendo el servicio únicamente a los componentes autorizados del sistema.

12.15.2.9. Configuración de ECR

Se utiliza Amazon ECR como servicio para gestionar las imágenes Docker de los servicios de backend, para lo cual se creó un repositorio destinado a almacenar las versiones construidas de cada componente.

12.15.2.10. Configuración de ECS

Se utilizó Amazon ECS para el despliegue de los servicios de backend y de websockets. Por un lado, se definieron task definitions independientes para cada servicio, en las cuales se especificaron los recursos necesarios (Linux/ARM64, 1 vCPU, 3 GB de memoria), la imagen de contenedor a utilizar, los puertos expuestos para el tráfico, la configuración de registro de logs en CloudWatch y las variables de entorno requeridas para su correcto funcionamiento.

Por otro lado, sobre estas task definitions se configuró un clúster de ECS y se crearon los servicios correspondientes, ejecutados sobre Fargate con una tarea por servicio, utilizando una estrategia de despliegue blue/green y asociando a cada servicio su load balancer para el manejo del tráfico externo.

De cara a un escenario productivo, se identifican las siguientes mejoras:

- Habilitar políticas de autoescalado para los servicios con mayor tráfico, de forma que el número de tareas se ajuste automáticamente a la carga.
- Reemplazar los secretos definidos directamente en las task definitions por referencias a AWS Secrets Manager, reduciendo la exposición de información sensible y centralizando su gestión.

12.15.2.11. Configuración de S3

Se utilizó Amazon S3 tanto para alojar el frontend como para el almacenamiento de imágenes. Para ello se crearon dos buckets independientes, cada uno con su propósito específico, y en el contexto del prototipo ambos buckets se configuraron como públicos para simplificar el acceso desde los clientes. En el caso del bucket que disponibiliza el frontend, el despliegue se realiza de forma manual subiendo el build de la aplicación web y configurando el servicio de Static website hosting para exponer el archivo index.html como punto de entrada del sitio.

12.15.2.12. Evaluación de Costos

El equipo, utilizando la herramienta AWS Pricing Calculator, realizó una evaluación del costo estimado de la infraestructura considerando la carga prevista del sistema en un entorno de validación operativa. El costo resultante fue de aproximadamente ochenta y dos dólares mensuales para la configuración utilizada en el prototipo.

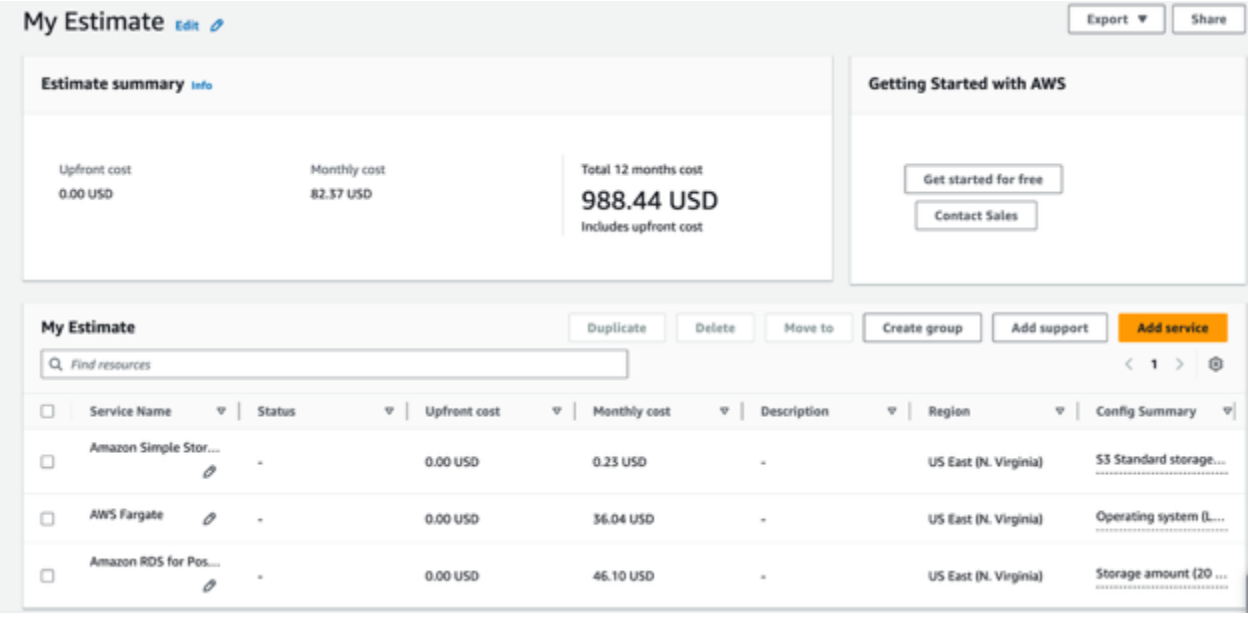


Figura 101: Costo estimado de la infraestructura

Este valor puede variar en función de la implementación de las distintas consideraciones planteadas a lo largo de esta sección (por ejemplo, habilitar alta disponibilidad, aumentar capacidades de cómputo o incorporar servicios adicionales), por lo que debe entenderse como una referencia inicial y no como un costo fijo.

12.15.3. Despliegue de las aplicaciones móviles

El despliegue de las aplicaciones móviles para el sistema E-Mobility Solutions abarca la publicación en las tiendas oficiales de Google (Play Store) y Apple (App Store). A diferencia de los servicios de *backend*, el despliegue móvil está sujeto a procesos de revisión externos y costos de membresía de desarrollador.

12.15.3.1. Plataforma Android (Google Play Store)

Para el despliegue en Android, se genera un archivo en formato Android App Bundle (.aab).

- **Proceso de Build:** Se utiliza el script automatizado configurado en el proyecto mediante el comando `npm run android:production`, el cual compila la aplicación optimizada para el entorno de producción.
- **Prototipo:** Para las pruebas iniciales, se utilizó la distribución mediante *Internal Testing* en Google Play Console, lo que permite a hasta 100 probadores específicos descargar la app antes de su revisión pública.
- **Producción:** Se debe configurar la ficha de la tienda (descripciones, capturas de pantalla) y cumplir con las políticas de privacidad. Además la aplicación debe pasar por un proceso de revisión que puede demorar entre 24 y 48 horas.
- **Costo:** Google requiere un pago único de USD 25 por la creación de la cuenta de desarrollador.

12.15.3.2. Plataforma iOS (Apple App Store)

El despliegue en iOS requiere el uso de Xcode y el cumplimiento de los estándares de la Human Interface Guidelines de Apple.

- **Proceso de Build:** Se genera el binario de producción utilizando el comando `npm run ios:production`.
- **Carga a la Tienda:** Una vez generado el build, se utiliza la herramienta Transporter de Apple para subir el paquete (.ipa) a App Store Connect de forma segura y eficiente.
- **Prototipo:** Se utilizó TestFlight para la distribución del prototipo. Este servicio permite la validación operativa antes del lanzamiento oficial.
- **Producción:** Requiere la firma de código mediante certificados de distribución y perfiles de aprovisionamiento. La aplicación debe pasar por un proceso de revisión manual por parte de Apple que puede demorar entre 24 y 48 horas.

- **Costo:** Apple requiere una suscripción anual al Apple Developer Program de USD 99. Este es un costo recurrente necesario para mantener la aplicación disponible en la tienda.

12.15.3.3. Consideraciones del *backend*

Para que las aplicaciones móviles funcionen correctamente en el entorno productivo descrito en la sección 12.15.2 del Anexo, se deben considerar los certificados SSL. Es mandatorio que el *Load Balancer* de ECS cuente con un certificado válido (vía AWS Certificate Manager) ya que tanto iOS como Android requieren conexiones HTTPS seguras.

12.16. Requerimientos No Implementados

Debido al tiempo limitado disponible, algunos requerimientos de menor prioridad no fueron implementados en esta versión del sistema. Estos elementos se mantienen en el *backlog* como posibles extensiones futuras:

- Como usuario del cargador, quiero reservar un conector para asegurarme de que haya un punto de carga disponible cuando llegue al cargador.
- Como usuario del cargador, quiero programar una carga para que el vehículo comience a cargar en un horario específico, optimizando costos y disponibilidad.
- Como empleado de E-Mobility Solutions, quiero disponer de roles en el panel administrativo para que ciertos clientes puedan ver el rendimiento de sus cargadores sin acceder a toda la configuración del sistema.
- Como empleado de E-Mobility Solutions quiero generar reportes que tengan en cuenta la tarifa variable de UTE para analizar correctamente los costos y márgenes asociados a cada sesión de carga.
- Como empleado de E-Mobility Solutions quiero configurar precios variables en función de la tarifa variable de UTE para que las tarifas de los cargadores reflejen de forma dinámica los costos de energía.
- Como empleado de E-Mobility Solutions quiero integrar el sistema con otros proveedores de red de carga, como UTE, para unificar la gestión de cargadores y ofrecer una experiencia consistente a los usuarios.
- Como usuario del cargador, quiero recibir notificaciones push cuando termina la carga para saber cuándo el vehículo está listo y liberar el conector a tiempo.
- Como operador, quiero recibir alertas cuando un usuario ocupa demasiado tiempo el cargador después de finalizar la carga, para poder gestionar mejor la rotación de los puntos de carga.

- Como responsable financiero, quiero integrar el sistema con un servicio de facturación electrónica para emitir comprobantes válidos de manera automática por las sesiones de carga realizadas.
- Como usuario del cargador quiero poder dejar reviews y comentarios sobre los cargadores después de una carga para compartir mi experiencia y ayudar a otros usuarios y a la operación a identificar problemas o mejoras.
- Como usuario del cargador quiero poder pagar por una carga usando un método existente de pago sin tener que ingresar mi CVV para tener un experiencia más fluida y sin interrupciones.

12.17. Cartas de Satisfacción del cliente

12.17.1 Carta de Satisfacción de Alfredo Pintos

Se presenta la carta de satisfacción de Alfredo Pintos, director y dueño de E-Mobility Solutions. Su perspectiva como experto en el mercado de electromovilidad y principal impulsor del proyecto resulta fundamental para confirmar que el prototipo desarrollado cumple con la visión estratégica y los objetivos de autonomía tecnológica buscados por la organización.

www.emobility-uy.com
Megenkar S.A. – eMobility Solutions
Ramón del Valle Inclán 2579
Montevideo 11800
Uruguay
Tel.: 094 370 274

11 de marzo de 2026

Ref.: Comentarios tesis plataforma de gestión de carga de vehículos eléctricos

De nuestra mayor consideración,

Desde eMobility Solutions queremos agradecer el trabajo realizado por el equipo durante el desarrollo de la plataforma de gestión de cargadores eléctricos.
Desde el comienzo se notó el interés por entender a fondo nuestros problemas y el contexto en el que operamos. No se limitaron solo a los aspectos técnicos, sino que buscaron comprender cómo funciona el día a día de nuestra operación y qué necesitábamos realmente resolver. Ese enfoque marcó una gran diferencia.
Valoramos mucho la disposición que mostraron en todo momento: siempre hubo buena comunicación, respuestas rápidas y una actitud proactiva para encontrar soluciones. El resultado final cumple con las funcionalidades que habíamos definido y nos deja con una base sólida para seguir creciendo y mejorando en el futuro.
Agradecemos el compromiso y la dedicación de todo el equipo durante el proyecto. Fue un gusto trabajar juntos.

Sin otro motivo, saluda a Ud. Atte.,

Alfredo Pintos

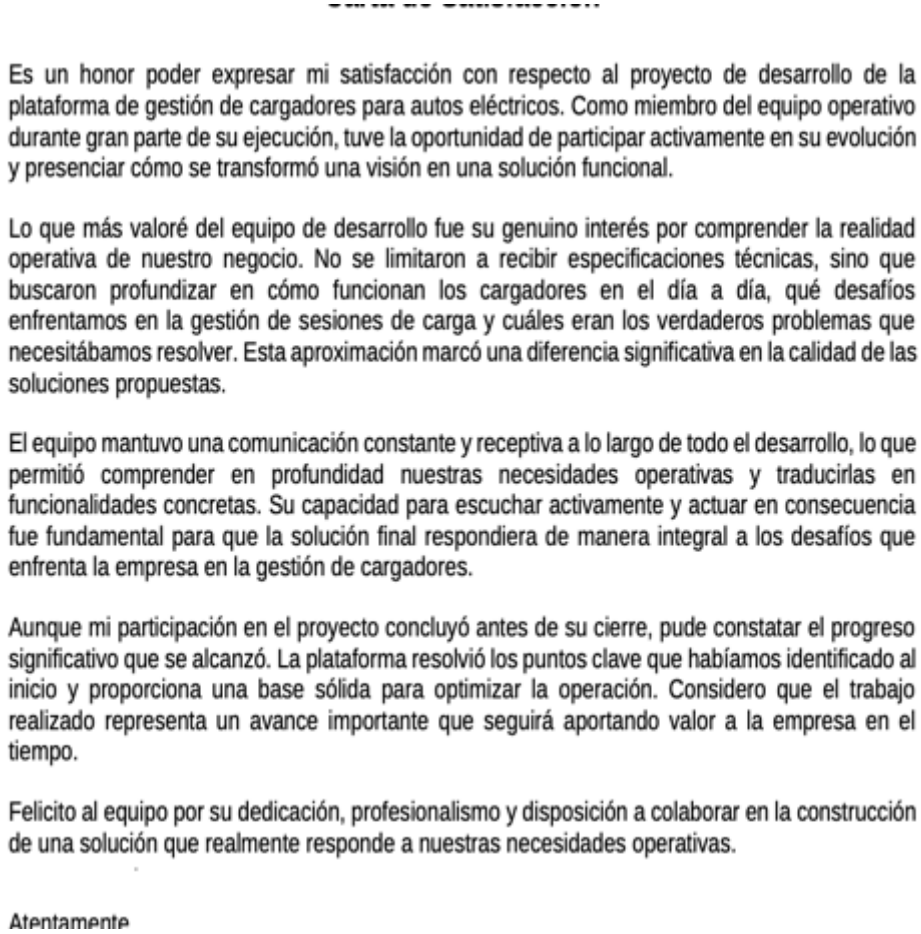
MBA Ing. Alfredo Pintos

Gerente General
alfredo.pintos@emobility-uy.com
094 370 274

Figura 102: Carta de satisfacción del referente de cliente, Alfredo Pintos

12.17.2 Carta de Satisfacción de Matías Crosa

Se le solicitó a Matías Crosa, ex jefe de operaciones de E-Mobility Solutions, su opinión sobre el transcurso del proyecto. Matías fue el principal punto de contacto a lo largo de todo el proceso, por lo que su perspectiva, aunque actualmente ya no forme parte de la organización, resulta especialmente valiosa para evaluar cómo se desarrolló el trabajo, qué tan alineado estuvo con las necesidades operativas del cliente y qué lecciones pueden extraerse.



Es un honor poder expresar mi satisfacción con respecto al proyecto de desarrollo de la plataforma de gestión de cargadores para autos eléctricos. Como miembro del equipo operativo durante gran parte de su ejecución, tuve la oportunidad de participar activamente en su evolución y presenciar cómo se transformó una visión en una solución funcional.

Lo que más valoré del equipo de desarrollo fue su genuino interés por comprender la realidad operativa de nuestro negocio. No se limitaron a recibir especificaciones técnicas, sino que buscaron profundizar en cómo funcionan los cargadores en el día a día, qué desafíos enfrentamos en la gestión de sesiones de carga y cuáles eran los verdaderos problemas que necesitábamos resolver. Esta aproximación marcó una diferencia significativa en la calidad de las soluciones propuestas.

El equipo mantuvo una comunicación constante y receptiva a lo largo de todo el desarrollo, lo que permitió comprender en profundidad nuestras necesidades operativas y traducirlas en funcionalidades concretas. Su capacidad para escuchar activamente y actuar en consecuencia fue fundamental para que la solución final respondiera de manera integral a los desafíos que enfrenta la empresa en la gestión de cargadores.

Aunque mi participación en el proyecto concluyó antes de su cierre, pude constatar el progreso significativo que se alcanzó. La plataforma resolvió los puntos clave que habíamos identificado al inicio y proporciona una base sólida para optimizar la operación. Considero que el trabajo realizado representa un avance importante que seguirá aportando valor a la empresa en el tiempo.

Felicito al equipo por su dedicación, profesionalismo y disposición a colaborar en la construcción de una solución que realmente responde a nuestras necesidades operativas.

Atentamente

Figura 103: Carta de satisfacción del referente de cliente, Matías Crosa

12.18. Documentación de Endpoints

Este anexo presenta una vista general de los endpoints implementados en la API del sistema, organizados según los distintos módulos funcionales. Las capturas fueron

obtenidas desde la herramienta Postman, utilizada durante el desarrollo para documentar y probar los servicios expuestos por el backend.

El objetivo de este anexo es ilustrar la estructura de la API y los principales recursos disponibles, mostrando las operaciones HTTP asociadas a cada uno de ellos.

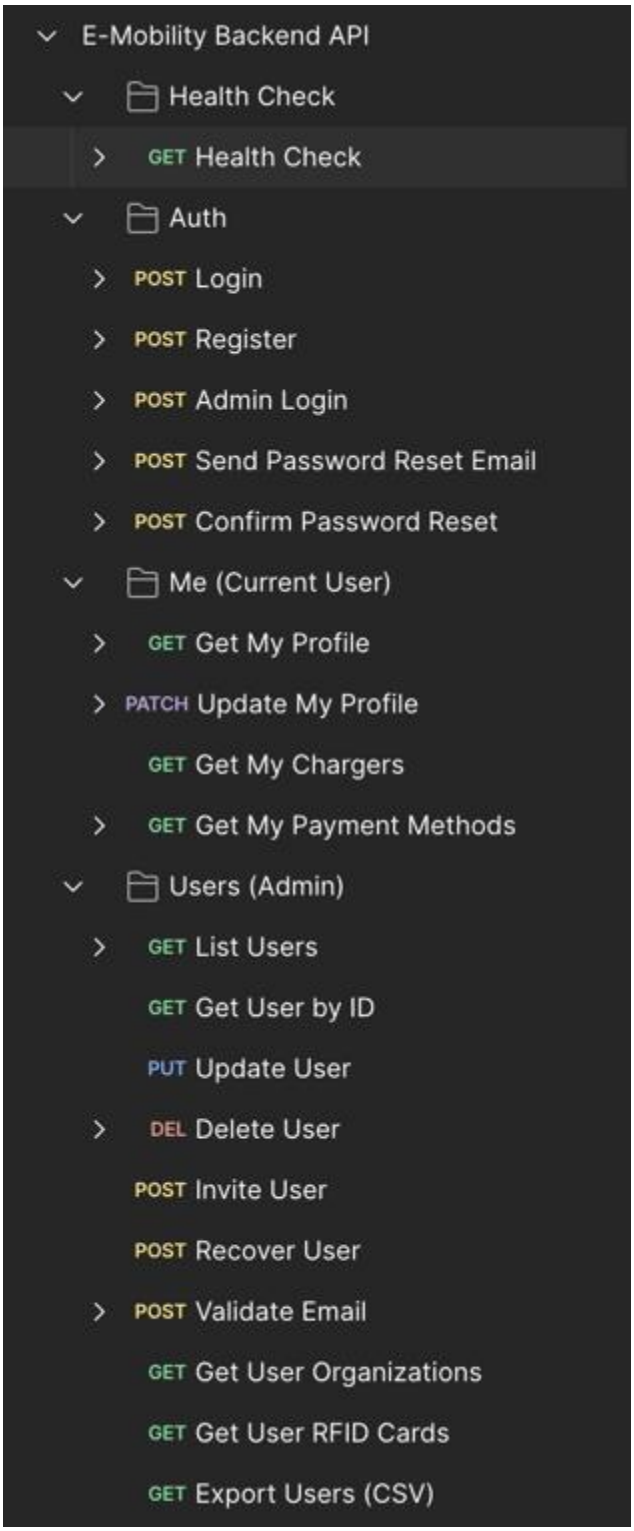


Figura 104: Colección de Postman de E-Mobility Backend

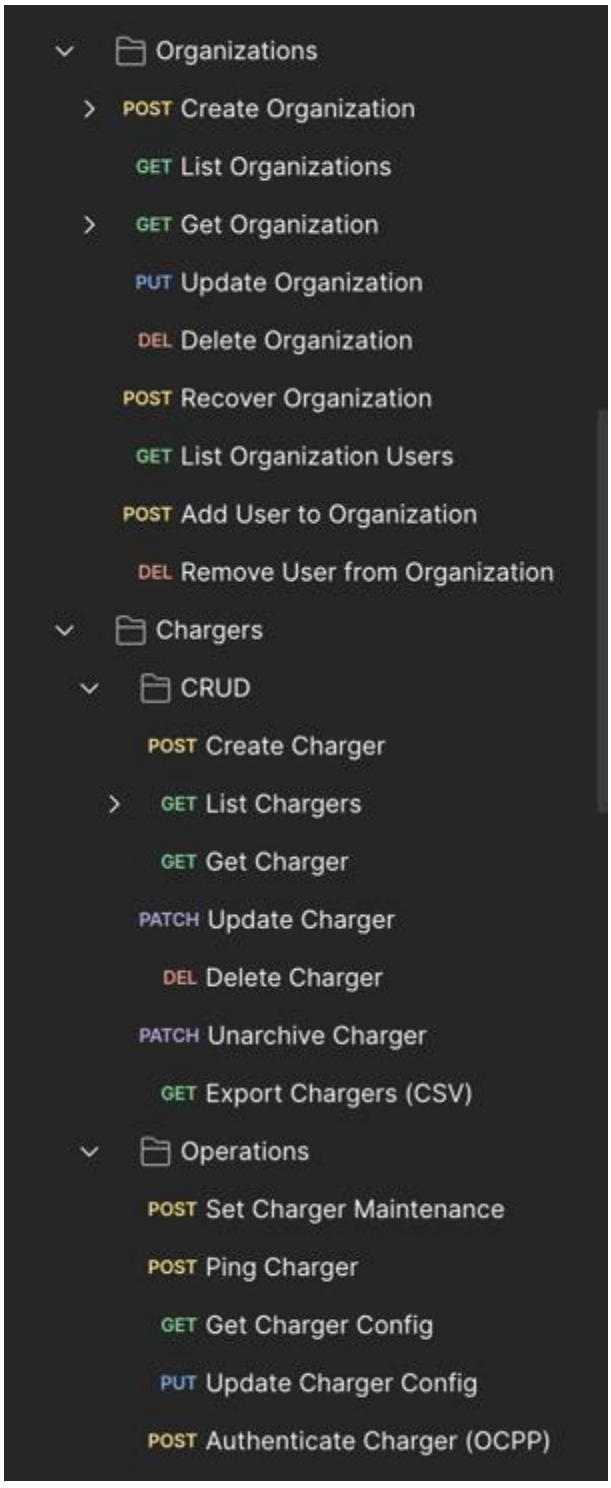


Figura 105: Colección de Postman de E-Mobility Backend

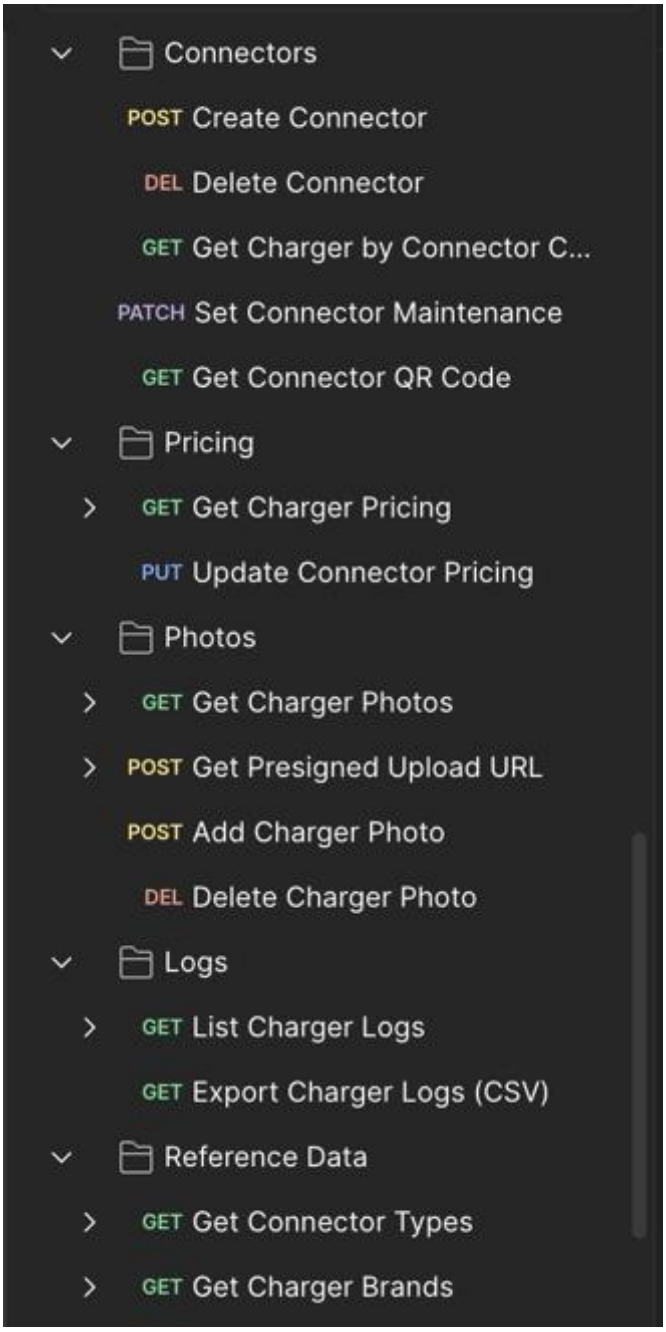


Figura 106: Colección de Postman de E-Mobility Backend

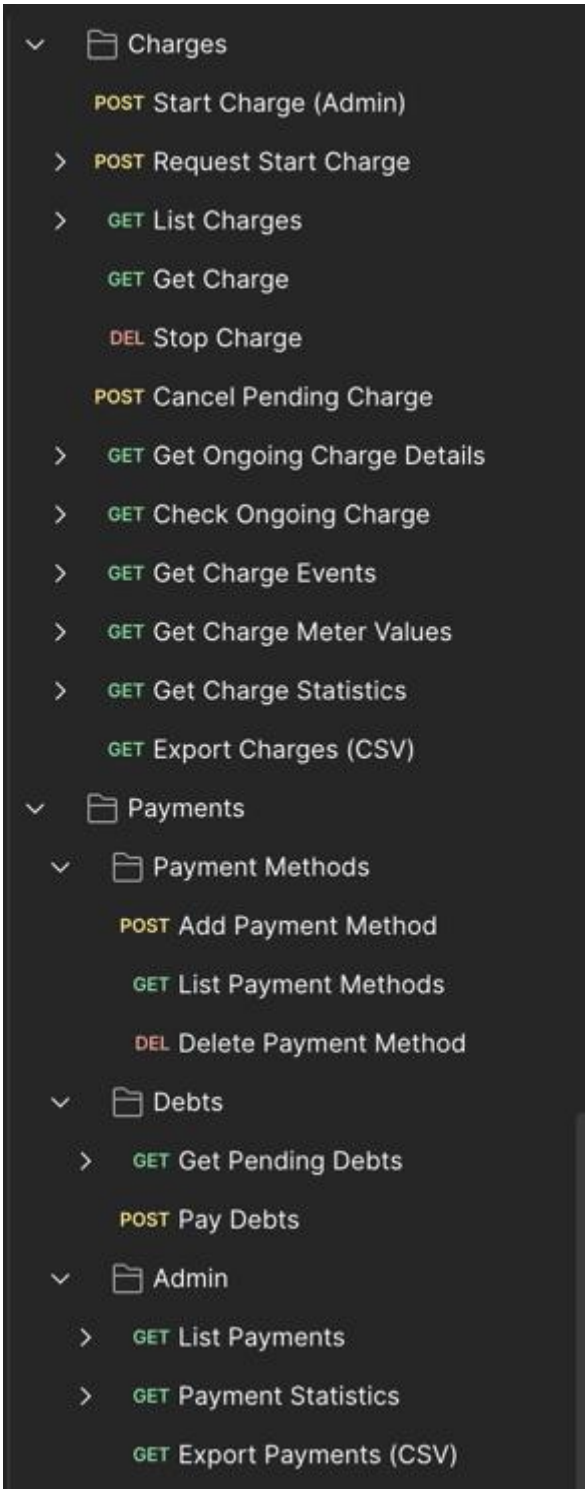


Figura 107: Colección de Postman de E-Mobility Backend

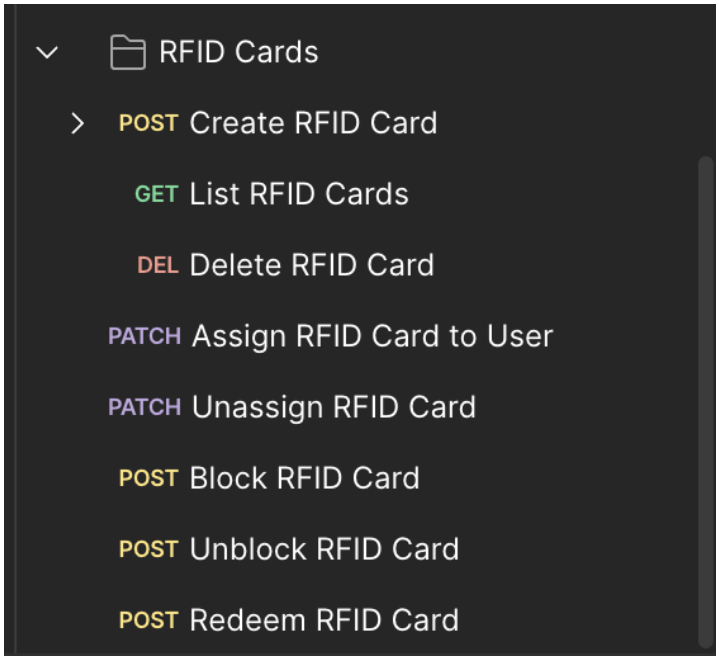


Figura 108: Colección de Postman de E-Mobility Backend