

DOCUMENTACIÓN TÉCNICA

Aplicación Móvil CargaME

Plataforma Oficial de Estaciones de Carga de Vehículos Eléctricos

Código del documento	SERVI-KFW-109993-DT-CARGAME-v2.0
Cliente	Ministerio de Minas y Energía de Colombia
Proveedor / Desarrollador	Servinformación (Información Localizada S.A.S.)
Contrato	KfW N.º 109993
Producto	CargaME — Aplicación móvil (Android e iOS)
Versión del aplicativo	0.0.1
Documento	Documentación técnica del desarrollo del proyecto
Versión del documento	2.0
Fecha de emisión	19 de junio de 2026

Documento de carácter técnico — Confidencial

Tabla de Contenido

Control de Versiones del Documento	4
Glosario de Términos y Siglas	5
1. Introducción	6
1.1. ¿Por qué y para qué existe CargaME?	6
2. Objetivos	6
2.1. Objetivo general	6
2.2. Objetivos específicos	6
3. Estado Actual del Proyecto	7
4. Stack Tecnológico Seleccionado	7
5. Arquitectura de Software: Clean Architecture (Hexagonal)	8
5.1. ¿Por qué y para qué implementamos esta arquitectura?	8
5.2. Puntos positivos y fortalezas de la arquitectura	8
6. Estructura Detallada del Repositorio	9
7. Gestión de Sesión, Seguridad y Roles Protegidos	9
7.1. ¿Por qué y para qué?	9
7.2. Decisiones de diseño y argumentación técnica en seguridad	10
8. Diseño Visual, Tipografía y UX Premium	10
8.1. ¿Por qué y para qué?	10
8.2. Fortalezas del sistema de diseño y UX	11
9. Características Destacadas de la Plataforma	11
9.1. Wizard de creación de estaciones en 4 pasos (SIVEEIC)	11
9.2. Autocomplete y filtros multi-selección con chips	11
9.3. SecureImage (carga de imágenes eficiente y segura)	12
9.4. Sistema de comentarios y reseñas integrado (Habeas Data)	12
9.5. Integración con navegadores GPS nativos (Intent Chooser)	12
10. Mapeo Detallado de Integración de Endpoints (SIVEEIC API)	13
10.0. Resumen consolidado de endpoints	13
10.1. Mapeo y consulta de estaciones	13
10.2. Autenticación y perfil del operador (CPO)	14
10.3. Registro de operador y cuenta (Sign-up)	15
10.4. Formulario de creación de estaciones (Wizard de 4 pasos)	16
10.5. Gestión y actualización de estaciones (Edición)	17
10.6. Interacción ciudadana (Calificaciones, reseñas y Habeas Data)	18
10.7. Documentación y consultas ciudadanas	19
11. Pruebas Automatizadas y Aseguramiento de Calidad	19
11.1. Cobertura de pruebas por capa	20
12. Guía de Instalación y Ejecución Local	20
12.1. Requisitos previos del sistema	20
12.2. Clonación e instalación de dependencias	20
12.3. Configuración de variables de entorno	20
12.4. Ejecución del servidor Metro y la aplicación	21



13. Generación de Compilados para Producción	21
13.1. Generar APK de pruebas (Debug)	21
13.2. Generar APK de Release (sin firmar)	21
13.3. Generar paquete AAB para despliegue en Google Play Store	21
14. Conclusiones	21
14.1. Elaboración y revisión del documento	22



Control de Versiones del Documento

La siguiente tabla registra el historial de versiones y cambios del presente documento técnico:

Versión	Fecha	Elaborado por	Descripción del cambio
1.0	19/06/2026	Servinformación	Versión inicial de la documentación técnica del desarrollo de CargaME a partir del repositorio del proyecto.
2.0	19/06/2026	Servinformación	Reincorporación de las rutas de código fuente en los puntos de disparo; incorporación de control de versiones, glosario de siglas, tabla resumen de endpoints y bloque de elaboración.

Glosario de Términos y Siglas

Las siguientes siglas y términos técnicos se emplean a lo largo del documento. Todas corresponden a conceptos presentes en la descripción del desarrollo de CargaME:

Sigla / Término	Significado
VE	Vehículo Eléctrico.
CPO	Charge Point Operator — Operador de estaciones (puntos) de carga.
SIVEEIC	Sistema de Información de Vehículos Eléctricos y Estaciones de Carga del Ministerio de Minas y Energía de Colombia (backend oficial del aplicativo).
RETIE	Reglamento Técnico de Instalaciones Eléctricas (certificación técnica exigida a las estaciones de carga).
JWT	JSON Web Token — token de seguridad usado para autenticar la sesión del operador.
UI	User Interface — Interfaz de usuario.
UX	User Experience — Experiencia de usuario.
DTO	Data Transfer Object — objeto que refleja el contrato de datos de la API.
PBT	Property-Based Testing — pruebas basadas en propiedades.
PQRS	Peticiones, Quejas, Reclamos y Sugerencias.
Habeas Data	Derecho y normativa de protección y tratamiento de datos personales.
Clean Architecture	Arquitectura hexagonal que aísla las reglas de negocio de los detalles de infraestructura.
Fabric	Nueva arquitectura de renderizado de React Native.
AAB	Android App Bundle — paquete de publicación para Google Play Store.
APK	Android Package — instalable de la aplicación Android.

1. Introducción

El presente documento constituye la documentación técnica del desarrollo de la aplicación móvil CargaME, plataforma oficial del Ministerio de Minas y Energía de Colombia para la visualización, consulta, búsqueda y gestión de estaciones de carga de vehículos eléctricos (VE) a nivel nacional. El desarrollo fue ejecutado por Servinformación en el marco del contrato KfW N.º 109993.

CargaME fue desarrollada en React Native con TypeScript e implementa una arquitectura hexagonal pura (Clean Architecture) orientada a producción, garantizando separación estricta de responsabilidades, seguridad de grado gubernamental, resiliencia ante fallas de red y una suite de tests robusta con cobertura completa.

1.1. ¿Por qué y para qué existe CargaME?

¿Por qué? Colombia se encuentra en una transición energética activa. La regulación nacional (Resoluciones 40223 de 2021 y 40123 de 2024 del Ministerio de Minas y Energía) exige que la infraestructura de carga de vehículos eléctricos esté debidamente registrada, cumpla con estándares técnicos estrictos (RETIE, certificados de conformidad) y sea transparente para los ciudadanos conductores.

¿Para qué? CargaME centraliza toda esta información, ofreciendo a los ciudadanos un mapa confiable en tiempo real para localizar cargadores compatibles, y a los operadores de estaciones de carga (CPOs - Charge Point Operators) una herramienta móvil ágil para registrar, actualizar e inventariar sus estaciones de carga, agilizando el flujo de validación y habilitación legal por parte del Ministerio.

2. Objetivos

2.1. Objetivo general

Documentar técnicamente el desarrollo de la aplicación móvil CargaME, describiendo su arquitectura de software, stack tecnológico, mecanismos de seguridad, diseño de experiencia de usuario, integración de servicios, características funcionales y aseguramiento de calidad, como soporte de la entrega al Ministerio de Minas y Energía de Colombia.

2.2. Objetivos específicos

- Presentar el stack tecnológico seleccionado y la justificación técnica de cada componente.
- Describir la arquitectura Clean Architecture (hexagonal) implementada y su organización en capas.
- Documentar la gestión de sesión, seguridad y control de roles protegidos del aplicativo.
- Detallar el sistema de diseño visual, tipografía y experiencia de usuario (UX).
- Especificar las características destacadas de la plataforma y su argumentación técnica.
- Mapear de forma exhaustiva la integración de los endpoints con la API SIVEEIC.
- Reportar la estrategia de pruebas automatizadas y los resultados de aseguramiento de calidad.
- Entregar la guía de instalación, ejecución local y generación de compilados para producción.

3. Estado Actual del Proyecto

- **Versión:** 0.0.1
- **Rama activa:** main
- **Suite de pruebas:** 28 suites / 98 tests — 100% Passing (Aprobados).
- **Calidad de código:** Libre de errores de tipado en compilación de TypeScript (`npx tsc --noEmit` limpio).
- **Compatibilidad de plataformas:** Android (entorno de producción validado con emuladores y dispositivos físicos) e iOS (arquitectura y dependencias configuradas, listo para compilación en Xcode).

4. Stack Tecnológico Seleccionado

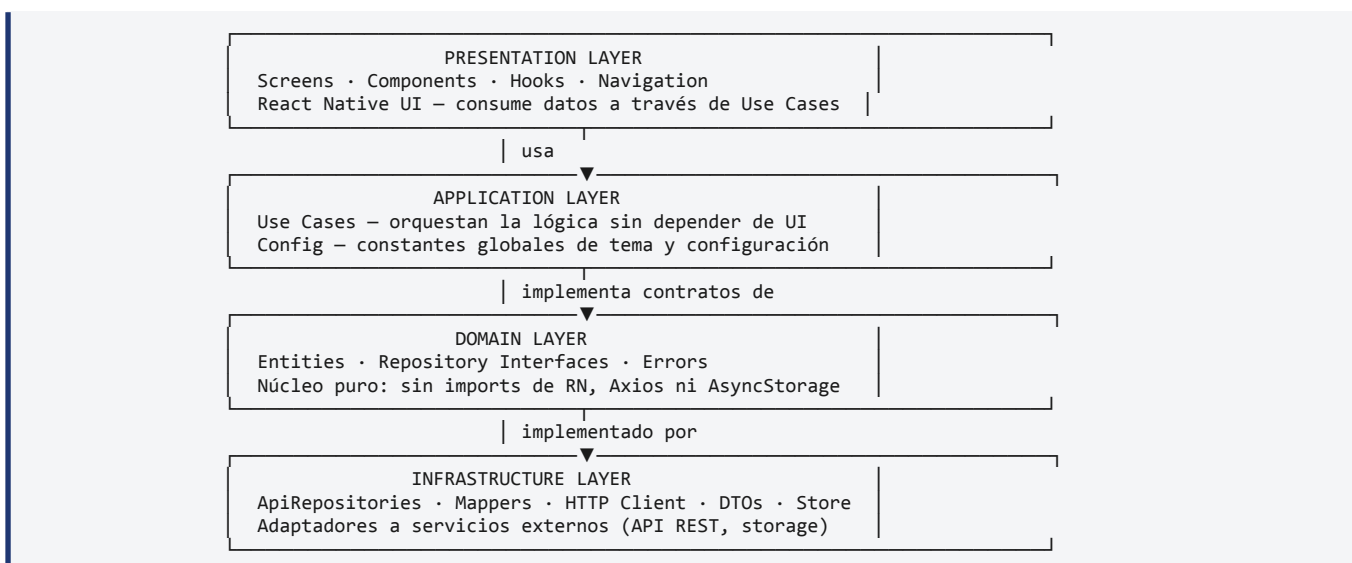
A continuación se detallan las tecnologías clave que estructuran CargaME, argumentando técnicamente la decisión de su uso:

Categoría	Tecnología	Versión	¿Por qué y para qué? / Justificación técnica
Entorno	React Native (Fabric)	0.83.1	Por qué: Permite un desarrollo multiplataforma compartiendo el 95% del código fuente entre Android e iOS. Para qué: Garantiza un rendimiento nativo fluido a 60fps, interactuando directamente con el motor gráfico nativo de mapas e inyectando componentes con la Nueva Arquitectura (Fabric).
Lenguaje	TypeScript	^5.8.3	Por qué: Evita errores comunes en tiempo de ejecución de JavaScript mediante tipado estático estricto. Para qué: Facilita la refactorización segura de modelos complejos de datos de la API de CargaME (más de 33 campos por estación de carga).
Server State	TanStack React Query	^5.90.20	Por qué: El estado del servidor es volátil y costoso de recuperar. Para qué: Administra de forma inteligente la caché local, la deduplicación de peticiones, la invalidación asíncrona de datos de cargadores y los reintentos automáticos tras microcortes de red.
Global State	Zustand	^5.0.11	Por qué: React Native requiere un manejo de estado local eficiente para la interfaz visual que no deba recargar la UI innecesariamente. Para qué: Gestiona el estado síncrono efímero de la aplicación (los formularios de 4 pasos del wizard, el estado de los filtros del mapa y las notificaciones toast).
HTTP Client	Axios	^1.13.5	Por qué: Necesitamos un control preciso sobre las peticiones HTTP, cabeceras de autorización y cookies. Para qué: Centraliza interceptores globales de seguridad, enmascara credenciales en consola de desarrollo y maneja de forma unificada las rutas base de la API del SIVEEIC.
Maps SDK	react-native-maps	^1.27.1	Por qué: La consulta espacial es el núcleo de valor para el conductor. Para qué: Renderiza de forma ultrafluida mapas de Google Maps SDK en Android y Apple Maps en iOS, soportando clústeres de pines, eventos táctiles interactivos y dibujo de rutas.

Categoría	Tecnología	Versión	¿Por qué y para qué? / Justificación técnica
Persistencia	AsyncStorage	^1.23.1	Por qué: Las credenciales del operador y las configuraciones de la aplicación deben persistir entre reinicios de la aplicación. Para qué: Permite la hidratación asíncrona de la sesión en el inicio y almacena tokens locales seguros.

5. Arquitectura de Software: Clean Architecture (Hexagonal)

CárgaME implementa una versión estricta de Clean Architecture, aislando las reglas de negocio de los detalles de infraestructura. Esto cumple la regla de oro de dependencia, donde las capas internas no conocen nada de las externas.



5.1. ¿Por qué y para qué implementamos esta arquitectura?

¿Por qué? En proyectos gubernamentales e institucionales de largo aliento, los endpoints de la API, los proveedores de base de datos o el propio framework de interfaz (React Native) pueden cambiar. Si el código de negocio está acoplado a la UI, un cambio de API obliga a reescribir toda la aplicación.

¿Para qué? Para aislar el núcleo puro de la app. La capa de Domain no contiene una sola línea de código relacionada con React Native, Axios, ni librerías nativas. Si el Ministerio decide migrar el backend a GraphQL o la UI a otro entorno, la lógica de negocio de CárgaME permanece intacta y reutilizable.

5.2. Puntos positivos y fortalezas de la arquitectura

- Testeabilidad en aislamiento absoluto:** Es posible probar el 100% de la lógica de negocio y las validaciones de las estaciones en Jest de forma instantánea, sin necesidad de renderizar pantallas, emular dispositivos o simular llamadas a servidores externos.
- Mapeo seguro de datos (anti-corrupción):** La capa de Infrastructure implementa mappers que transforman las respuestas crudas de la API (muchas veces inconsistentes o con tipados laxos)

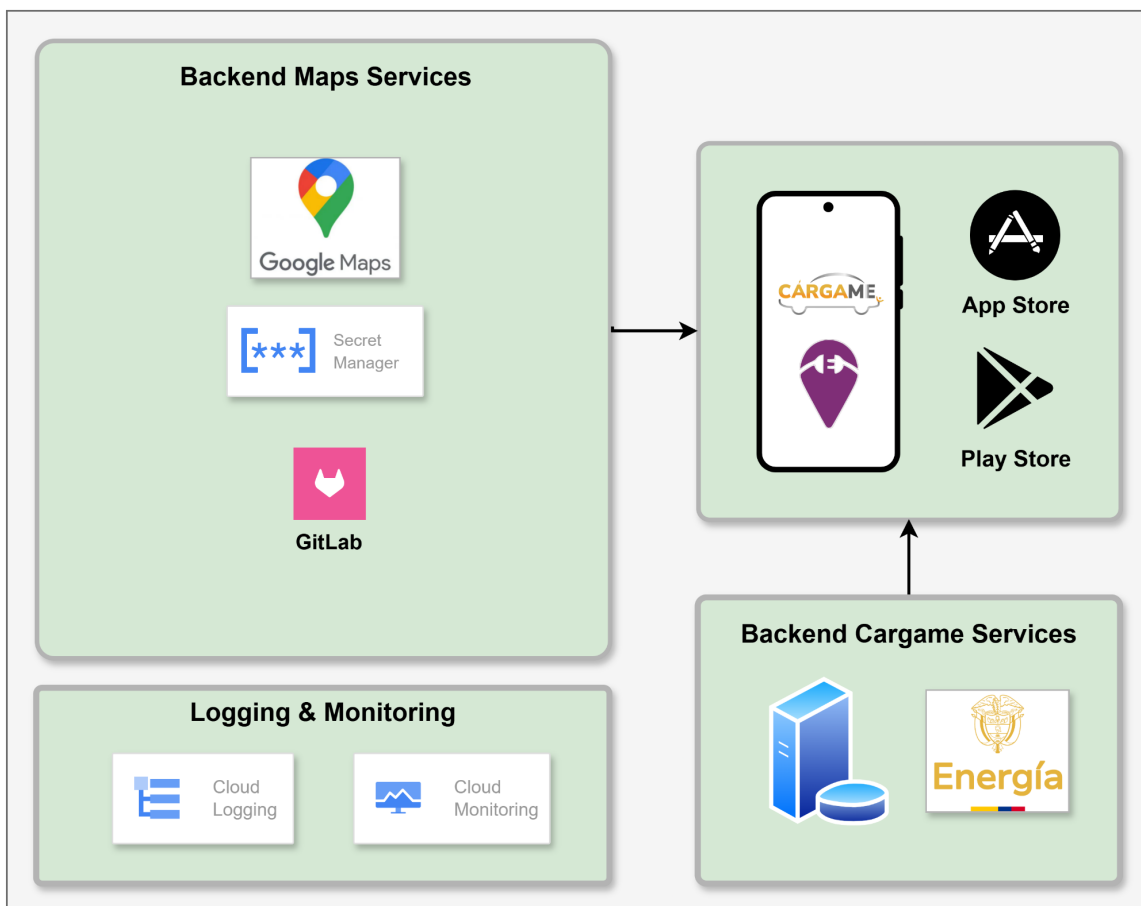
en entidades de Domain estrictas y limpias. La UI solo consume modelos seguros y perfectamente tipados.

3. **Resiliencia e independencia:** Si el servidor del Ministerio de Minas y Energía presenta fallos de red o cambia las cabeceras, sólo se modifica la capa de Infraestructura (los adaptadores), mientras que las pantallas, componentes y la navegación no sufren ninguna alteración.

6. Arquitectura de Infraestructura de la Solución

La aplicación móvil Cárgame se apoya en una infraestructura de servicios en la nube organizada en bloques funcionales claramente diferenciados, los cuales convergen hacia el aplicativo móvil para garantizar su operación, seguridad, distribución y observabilidad. A continuación se describe cada uno de los componentes que conforman la arquitectura de infraestructura de la solución.

El siguiente diagrama presenta una vista general de la arquitectura de infraestructura de la solución y la relación entre sus bloques:



6.1. ¿Por qué y para qué?

¿Por qué? Una aplicación de carácter institucional como CargaME no opera de forma aislada: requiere servicios externos de mapas, gestión segura de credenciales, control de versiones del código fuente, monitoreo continuo y canales oficiales de distribución para los usuarios finales.

¿Para qué? Para separar de forma ordenada las responsabilidades de infraestructura en bloques especializados, facilitando la operación, el mantenimiento, la trazabilidad y la seguridad de la solución a lo largo de su ciclo de vida.

6.2. Aplicación móvil CargaME

Es el núcleo de la solución y el punto de convergencia de todos los servicios de infraestructura. La aplicación móvil consume los servicios de mapas para la visualización geoespacial de las estaciones de carga y se comunica con los servicios de backend propios de CargaME para la gestión de la información. Se distribuye a los usuarios finales a través de los dos canales oficiales de las tiendas de aplicaciones móviles:

- **App Store**, para la distribución de la aplicación en dispositivos con sistema operativo iOS.
- **Play Store**, para la distribución de la aplicación en dispositivos con sistema operativo Android.

6.3. Backend Maps Services (Servicios de mapas y soporte)

Este bloque agrupa los servicios que dan soporte a la capa geoespacial, a la gestión segura de credenciales y al control de versiones del código fuente:

- **Google Maps:** Proveedor del servicio de mapas que sustenta la consulta espacial y el renderizado de las estaciones de carga, núcleo de valor para el conductor que localiza cargadores compatibles.
- **Secret Manager:** Servicio dedicado a la gestión y almacenamiento seguro de credenciales, claves de API y secretos de la solución, evitando su exposición en el código fuente o en el repositorio.
- **GitLab:** Plataforma de control de versiones y gestión del código fuente del proyecto, que permite el versionamiento, la trazabilidad y la administración del repositorio del desarrollo.

6.4. Backend Cargame Services (Servicios propios de CargaME)

Este bloque corresponde a los servicios de backend propios de la solución, que respaldan la lógica de negocio y la persistencia de la información del aplicativo. Comprende la capa de servidor y base de datos sobre la que se gestiona la información de estaciones de carga, operadores y demás entidades del sistema, en el marco institucional del Ministerio de Minas y Energía de Colombia, entidad que respalda y dirige la plataforma.

6.5. Logging & Monitoring (Registro y monitoreo)

Este bloque concentra los servicios de observabilidad de la solución, que permiten supervisar el comportamiento, el desempeño y la disponibilidad del sistema en operación:

- **Cloud Logging:** Servicio de registro centralizado de eventos y trazas de la solución, que facilita el diagnóstico y la auditoría del comportamiento del sistema.
- **Cloud Monitoring:** Servicio de monitoreo que permite observar de forma continua el estado, el desempeño y la disponibilidad de la infraestructura y los servicios de la solución.

6.6. Flujo general de la arquitectura

El flujo de la arquitectura de infraestructura se orienta hacia la aplicación móvil CargaME como punto central: el bloque de Backend Maps Services provee a la aplicación las capacidades de mapas, gestión de secretos y control de versiones, mientras que el bloque de Backend Cargame Services aporta los servicios de backend y persistencia propios de la solución. De forma transversal, el bloque de Logging & Monitoring brinda la observabilidad necesaria para garantizar la operación confiable del sistema, y la distribución a los usuarios finales se realiza a través de App Store y Play Store.

7. Estructura Detallada del Repositorio

La organización del código refleja de manera explícita la división de Clean Architecture para facilitar la integración de nuevos desarrolladores:

CargameApp/	
__tests__/	# Pruebas integrales del sistema y Property-Based Testing
android/	# Directorio nativo Android (inyección de API Keys y permisos)
ios/	# Directorio nativo iOS (estructura CocoaPods y certificados)
src/	
domain/	# CAPA DE DOMINIO (Núcleo de Negocio - Puro)
entities/	# Modelos de datos del negocio (ChargingStation, Operator, etc.)
errors/	# Errores específicos tipados de negocio
repositories/	# Contratos (interfaces) de repositorios que la UI consume
application/	# CAPA DE APLICACIÓN (Lógica y Orquestación)
config/	# Variables de tema institucional, timeouts y límites de red
useCases/	# Casos de uso (CreateStation, Login, filterStations)
infrastructure/	# CAPA DE INFRAESTRUCTURA (Adaptadores e Implementación)
http/	# Singleton de Axios con interceptores de seguridad y enmascaramiento
mappers/	# Traductores de objetos crudos de API a entidades limpias de dominio
models/	# Data Transfer Objects (DTOs) que reflejan el contrato de la API
repositories/	# Implementaciones concretas de las interfaces del dominio
store/	# Almacenes de Zustand (Sesión, Wizard de 4 pasos, Toasts, Filtros)
presentation/	# CAPA DE PRESENTACIÓN (Interfaz de Usuario)
assets/	# Recursos estáticos: fuentes Nunito Sans, iconos y logos
components/	# Componentes interactivos reutilizables (SecureImage, Toasts, Autocomplete)
hooks/	# Ganchos personalizados que conectan React Query con los Casos de Uso
navigation/	# Navegación nativa fluida con React Navigation (AppNavigator)
screens/	# Pantallas de la aplicación (MapScreen, CreateStationScreen, etc.)
utils/	# Utilidades puras transversales (jwt.ts, navigation.ts, validators.ts)

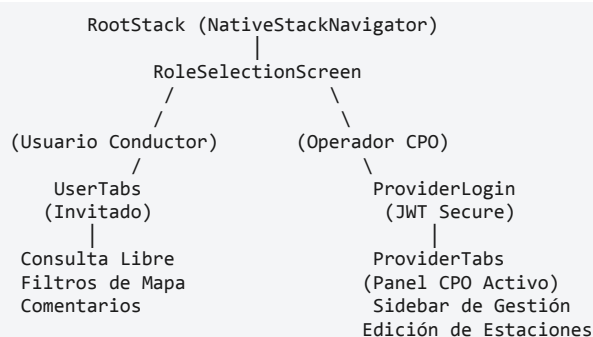
8. Gestión de Sesión, Seguridad y Roles Protegidos

CárgaME implementa una arquitectura de seguridad multinivel para salvaguardar la privacidad de la información y la integridad de los datos de la red de carga nacional.

8.1. ¿Por qué y para qué?

¿Por qué? El público general debe acceder libremente a consultar cargadores sin registrarse, pero los operadores CPO deben identificarse con altos niveles de seguridad para registrar o editar sus cargadores, previniendo alteraciones maliciosas o robo de datos.

¿Para qué? El árbol de navegación bifurca la aplicación en dos flujos independientes basándose en tokens JWT (JSON Web Tokens) persistidos localmente de forma segura.



8.2. Decisiones de diseño y argumentación técnica en seguridad

- Bypass de sesión inteligente:** Si un CPO cuenta con sesión activa en AsyncStorage, al seleccionar el perfil de Prestador en RoleSelectionScreen se ejecuta un desvío automático hacia ProviderTabs. De igual manera, si aterriza en LoginScreen, un hook de efecto redirige instantáneamente al panel principal de CPO, previniendo la visualización de pantallas de inicio de sesión intermedias innecesarias para una experiencia de usuario (UX) ágil.
- Cierre de sesión simplificado y seguro:** Se reemplazó la confusa opción oculta en el mapa por una pestaña dedicada y destacada en color rojo en la barra de navegación del CPO. Al presionarla, limpia sincronamente el almacén global useAuthStore y remueve los tokens de sesión de AsyncStorage, redireccionando sincronamente al usuario a RoleSelectionScreen. Esto es de vital importancia, ya que permite que múltiples CPOs utilicen de forma segura el mismo dispositivo móvil sin dejar tokens huérfanos.
- Control dinámico de vistas (isProviderView):** El mapa principal MapScreen.tsx recibe parámetros de navegación (isProviderView: true/false). Si un CPO con sesión activa decide ingresar a consultar cargadores como usuario general, el mapa desactiva de forma dinámica las vistas protegidas de administración, el filtrado restrictivo por su id_operador y el panel de edición, brindándole una experiencia de exploración limpia idéntica a la de un ciudadano común.
- Protección de documentos RETIE y cumplimiento:** Por directrices legales de confidencialidad, las declaraciones de cumplimiento técnica y los certificados RETIE son datos confidenciales. El botón para abrirlos e inspeccionarlos en la tarjeta de detalles se oculta de forma automática para usuarios anónimos, y solo se revela a CPOs debidamente autenticados en el sistema.

9. Diseño Visual, Tipografía y UX Premium

La aplicación ha sido diseñada con un estilo moderno, estético, institucional y altamente interactivo, logrando una de las mejores cartas de presentación visual del Ministerio.

9.1. ¿Por qué y para qué?

¿Por qué? Las aplicaciones móviles gubernamentales a menudo sufren de diseños planos, toscos y obsoletos que desincentivan su uso.

¿Para qué? CargaME rompe este paradigma introduciendo un sistema de diseño pulido con micro-animaciones reactivas, layouts de pantalla completa (full-width) y una paleta cromática de alto contraste basada en el manual de marca oficial.

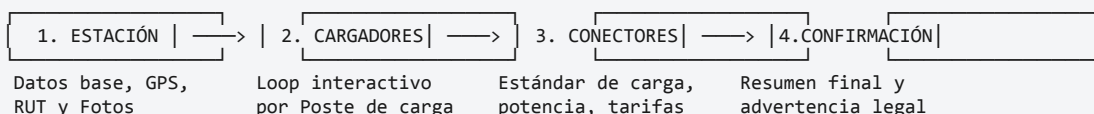
9.2. Fortalezas del sistema de diseño y UX

- **Identidad institucional elegante:** Utiliza una combinación armoniosa de colores con el Azul Ministerio (#1F3772) para encabezados, el Azul Oscuro (#002e5d) como tono principal y el Dorado de Acento (#E4B33C) para botones interactivos de acción y marcadores clave del mapa.
- **Tipografía sofisticada:** Integra de forma nativa la familia tipográfica Nunito Sans de Google Fonts, la cual se compila directamente en los recursos nativos de Android e iOS. Esto elimina las fuentes del sistema del navegador genérico y aporta un aspecto pulcro, legible y premium a cada etiqueta.
- **Prevención de bloqueo de teclado:** Se envolvió de manera sistemática cada formulario de la aplicación (LoginScreen, RegisterScreen, RecoverPasswordScreen y ContactScreen) con un contenedor KeyboardAvoidingView dinámico:
 - **En iOS:** Usa behavior="padding" para desplazar síncronamente los campos hacia arriba al emerger el teclado.
 - **En Android:** Usa behavior={undefined} en sintonía con un ScrollView de scroll vertical despejado. Esto previene dobles espaciados saltos de pantalla indeseados, permitiendo que el usuario lea fluidamente lo que escribe sin tapar los inputs.

10. Características Destacadas de la Plataforma

10.1. Wizard de creación de estaciones en 4 pasos (SIVEEIC)

Para registrar una estación de carga, la app móvil implementa un asistente de registro que cumple rigurosamente con la relación jerárquica del Ministerio (Una Estación contiene Cargadores, y cada Cargador contiene Conectores):



¿Por qué y para qué? El envío a la API requiere que la estación ya exista para obtener su ID, y luego asociar los cargadores y conectores correspondientes. El asistente guía al CPO de forma sencilla en lugar de presentarle un formulario masivo de 50 campos.

Argumentación técnica: Gracias al store reactivo `useChargingStationWizardStore` en Zustand, la navegación bidireccional es 100% segura. El usuario puede retroceder de Conectores a Cargadores para corregir una potencia, y al volver hacia adelante, los inputs conservan intactos sus datos sin perderse. La orquestación y envío final de las APIs se ejecuta únicamente en el Paso 4, manejando transacciones en bloque y reportando con precisión en qué paso falló la API para un diagnóstico claro.

10.2. Autocomplete y filtros multi-selección con chips

¿Por qué y para qué? El mapa de consulta puede poblarse con cientos de estaciones de carga a nivel nacional. Un conductor necesita segmentar de forma veloz para no saturar su pantalla y buscar por municipio, operador o tipo de conector compatible.

Argumentación técnica: Implementamos un modal de filtros en `FilterModal.tsx` que integra el componente personalizado `AutocompleteChipInput`. Mientras el usuario escribe en un input con `debounce` (200ms) para no fatigar la memoria, la app filtra síncronamente los municipios y entidades disponibles mostrando un menú desplegable de sugerencias. Al tocar una sugerencia, se genera un Chip Dorado (`#E4B33C`) autodescartable con botón X. Los filtros internos dentro de una misma categoría operan bajo una lógica OR, lo que le permite al usuario buscar cargadores simultáneamente en “Bogotá” y “Cali”.

10.3. SecureImage (carga de imágenes eficiente y segura)

¿Por qué y para qué? El backend gubernamental protege las imágenes de las estaciones mediante cabeceras de autorización. La carga ingenua en dispositivos móviles descargando bytes en arrays de buffer y convirtiéndolos a base64 consumía gigabytes de memoria RAM de JavaScript, produciendo crasheos constantes de la app con fotos de alta resolución.

Argumentación técnica: `SecureImage.tsx` utiliza la ruta de imagen relativa devuelta por el mapeador, construye la URL absoluta y le inyecta las cabeceras `Authorization` y `Cookie` directamente al cargador nativo de la plataforma:

```
<Image source={{ uri: fullUrl, headers: { Authorization: token, Cookie: cookie } }} />
```

Esto delega la decodificación, el renderizado de formatos (JPEG, PNG, WebP) y el almacenamiento en caché al motor nativo del sistema operativo (iOS/Android), reduciendo el consumo de RAM en JS a prácticamente cero y logrando un scroll de alto rendimiento sin parpadeos. Si la imagen falla por red, despliega un placeholder “Sin imagen”; si el campo es nulo de base, muestra de forma elegante “Imagen no disponible”.

10.4. Sistema de comentarios y reseñas integrado (Habeas Data)

¿Por qué y para qué? Un ecosistema saludable de movilidad eléctrica necesita la retroalimentación de la comunidad sobre el estado real de los cargadores, pero la ley colombiana exige la protección de datos personales y prohíbe calificaciones anónimas.

Argumentación técnica: Se construyó el componente `CommentSection` integrado en `StationDetailCard.tsx`. Para calificar de 1 a 5 estrellas y escribir un comentario, se exige de forma mandatoria un formulario con Nombre, Tipo de Identificación, Correo, Teléfono y la casilla de aceptación expresa de tratamiento de datos personales de Habeas Data. Para mantener la descentralización, los comentarios creados se guardan en `AsyncStorage` con su respectivo

delete_token retornado por la API, permitiendo identificar localmente cuáles comentarios pertenecen al propio dispositivo para mostrar opciones de eliminación segura.

10.5. Integración con navegadores GPS nativos (Intent Chooser)

¿Por qué y para qué? Una app de mapas no debe competir contra Waze o Google Maps para la guía de rutas giro a giro. La app móvil debe facilitar de forma inmediata la apertura de la herramienta preferida del conductor.

Argumentación técnica: La utilidad buildNavigationUrl genera esquemas de URL universales basados en la geolocalización de la estación (geo:lat,lng?q=lat,lng en Android y maps://?q=lat,lng en iOS) y los dispara mediante Linking.openURL(). Esto invoca de forma directa el menú de opciones del sistema operativo móvil (Intent Chooser / Activity View) para que el usuario elija su app de mapas favorita con un solo tap.

11. Mapeo Detallado de Integración de Endpoints (SIVEEIC API)

La comunicación con el backend oficial de SIVEEIC (Sistema de Información de Vehículos Eléctricos y Estaciones de Carga del Ministerio de Minas y Energía de Colombia) está estructurada siguiendo los patrones de Clean Architecture en la capa de Infraestructura, utilizando Axios para las peticiones HTTP y TanStack React Query para la sincronización y almacenamiento en caché de los datos del servidor.

A continuación, se detalla de forma exhaustiva cada uno de los 20 endpoints integrados en el aplicativo móvil, especificando su propósito, ubicación exacta en el código fuente (repositorios, casos de uso y hooks de React) y el momento exacto en el que se disparan en la interfaz de usuario:

11.0. Resumen consolidado de endpoints

La siguiente tabla resume los 20 endpoints integrados, agrupados por su categoría funcional, para una consulta rápida antes del detalle individual:

Método	Ruta (endpoint)	Categoría funcional
GET	/crg/resumenestacionescarga/0/0	Mapeo y consulta de estaciones
GET	/crg/estacioncarga/{id}	Mapeo y consulta de estaciones
POST	/crg/logueo	Autenticación y perfil del operador (CPO)
POST	/crg/solicitud	Autenticación y perfil del operador (CPO)
GET	/crg/usuario	Autenticación y perfil del operador (CPO)
GET	/crg/operador/{id}	Autenticación y perfil del operador (CPO)
POST	/crg/operador	Registro de operador y cuenta (Sign-up)
POST	/crg/usuario	Registro de operador y cuenta (Sign-up)
GET	/crg/municipiofull	Creación de estaciones (Wizard 4 pasos)
POST	/crg/estacioncarga	Creación de estaciones (Wizard 4 pasos)
POST	/crg/puntocarga	Creación de estaciones (Wizard 4 pasos)
POST	/crg/conectorcarga	Creación de estaciones (Wizard 4 pasos)
PUT	/crg/estacioncarga/{id}	Gestión y actualización (Edición)

Método	Ruta (endpoint)	Categoría funcional
PUT	/crg/puntocarga/{id}	Gestión y actualización (Edición)
PUT	/crg/conectorcarga/{id}	Gestión y actualización (Edición)
GET	/crg/puntos-carga/{id}/comentarios	Interacción ciudadana (Habeas Data)
POST	/crg/puntos-carga/{id}/comentarios	Interacción ciudadana (Habeas Data)
DELETE	/crg/puntos-carga/{id}/comentarios/{commentId}	Interacción ciudadana (Habeas Data)
GET	/crg/documentosmovil	Documentación y consultas ciudadanas
POST	/crg/contactoestaciones	Documentación y consultas ciudadanas

11.1. Mapeo y consulta de estaciones

GET /crg/resumenestacionescarga/0/0

Propósito: Recuperar el listado completo y consolidado de todas las estaciones de carga eléctrica registradas en el territorio nacional de Colombia.

Ubicación en el código:

- **Repositorio:** `ApiChargingStationRepository.ts`
(`file:///c:/Users/jpatino/CargameApp/src/infrastructure/repositories/ApiChargingStationRepository.ts#L8-L18`)
- **Mapeador (Mapper):** `ChargingStationMapper.ts` (`src/infrastructure/mappers/ChargingStationMapper.ts`)
- **Hook de React:** `useChargingStations.ts` (`src/presentation/hooks/useChargingStations.ts`)

Punto de disparo (UI): Se invoca de forma automática al montar la pantalla principal del mapa (`MapScreen.tsx` (`file:///c:/Users/jpatino/CargameApp/src/presentation/screens/MapScreen.tsx`)). Descarga las coordenadas GPS, nombres, operadores y estados técnicos para pintar todos los marcadores interactivos (`StationMarker.tsx`) en el mapa.

GET /crg/estacioncarga/{id}

Propósito: Consultar la información técnica y estructural detallada de una estación de carga específica, incluyendo su desglose jerárquico de cargadores y mangueras (conectores).

Ubicación en el código:

- **Hook de React:** `useStationDetail.ts`
(`file:///c:/Users/jpatino/CargameApp/src/presentation/hooks/useStationDetail.ts#L5-L16`)

Punto de disparo (UI):

1. Al presionar sobre el marcador de una estación en el mapa, abre la tarjeta inferior de detalles (`StationDetailCard.tsx` (`file:///c:/Users/jpatino/CargameApp/src/presentation/components/StationDetailCard.tsx`)), disparando la petición para listar sus especificaciones técnicas de carga.
2. Al ingresar a la pantalla de edición de estación (`EditStationScreen.tsx` (`file:///c:/Users/jpatino/CargameApp/src/presentation/screens/EditStationScreen.tsx`)) para precargar el wizard con la información actual de la base de datos antes de permitir cambios.

11.2. Autenticación y perfil del operador (CPO)

POST /crg/logueo

Propósito: Autenticar las credenciales del operador (CPO) y retornar un token de seguridad JWT.

Ubicación en el código:

- **Repositorio:** `ApiAuthRepository.ts`
(`file:///c:/Users/jpatino/CargameApp/src/infrastructure/repositories/ApiAuthRepository.ts#L10`)
- **Caso de Uso:** `LoginUseCase.ts` (`src/domain/useCases/LoginUseCase.ts`)
- **Hook de React:** `useLogin.ts` (`src/presentation/hooks/useLogin.ts`)

Punto de disparo (UI): Se dispara en la pantalla de inicio de sesión (`LoginScreen.tsx` (`file:///c:/Users/jpatino/CargameApp/src/presentation/screens/LoginScreen.tsx`)) al rellenar el formulario de usuario/contraseña y pulsar el botón “Ingresar”. Encripta la sesión localmente y redirige al panel exclusivo del CPO.

POST /crg/solicitud

Propósito: Solicitar el restablecimiento de contraseña para cuentas de operador.

Ubicación en el código:

- **Repositorio:** `ApiRecoverPasswordRepository.ts`
(`file:///c:/Users/jpatino/CargameApp/src/infrastructure/repositories/ApiRecoverPasswordRepository.ts#L18-L21`)
- **Caso de Uso:** `RecoverPasswordUseCase.ts` (`src/domain/useCases/RecoverPasswordUseCase.ts`)
- **Hook de React:** `useRecoverPassword.ts` (`src/presentation/hooks/useRecoverPassword.ts`)

Punto de disparo (UI): Se invoca en la pantalla de recuperación de contraseña (`RecoverPasswordScreen.tsx` (`file:///c:/Users/jpatino/CargameApp/src/presentation/screens/RecoverPasswordScreen.tsx`)) cuando el CPO ingresa su correo electrónico o usuario registrado y presiona “Enviar enlace”.

GET /crg/usuario

Propósito: Listar usuarios registrados en la plataforma para realizar auditoría y emparejamiento de cuentas de CPO.

Ubicación en el código:

- **Hook de React (Secuencial):** `useGetUserEntity.ts`
(`file:///c:/Users/jpatino/CargameApp/src/presentation/hooks/useGetUserEntity.ts#L29`)

Punto de disparo (UI): Se dispara automáticamente en la pestaña “Mi Entidad” (`EntityDataScreen.tsx` (`file:///c:/Users/jpatino/CargameApp/src/presentation/screens/EntityDataScreen.tsx`)) de la barra de navegación del CPO. Busca la información del usuario logueado actualmente para extraer dinámicamente el `id_operador` de su propiedad (`Coordinacionoperativa.Id`).

GET /crg/operador/{id}

Propósito: Recuperar la información legal, comercial y de contacto de una entidad operadora (CPO) específica.

Ubicación en el código:

- **Hook de React (Secuencial):** `useGetUserEntity.ts`
(`file:///c:/Users/jpatino/CargameApp/src/presentation/hooks/useGetUserEntity.ts#L45`)

Punto de disparo (UI): Es el paso final disparado automáticamente en la pantalla de perfil del operador (`EntityDataScreen.tsx` (`file:///c:/Users/jpatino/CargameApp/src/presentation/screens/EntityDataScreen.tsx`)) tras resolver el endpoint `/crg/usuario`. Renderiza la información oficial de la empresa (RUT, dirección, teléfono, correo oficial) en modo de solo lectura.

11.3. Registro de operador y cuenta (Sign-up)

POST /crg/operador

Propósito: Crear una nueva entidad jurídica o de operador de carga (CPO) en el sistema del Ministerio, adjuntando soporte de representación legal.

Ubicación en el código:

- **Repositorio:** `ApiOperatorRepository.ts`
(`file:///c:/Users/jpatino/CargameApp/src/infrastructure/repositories/ApiOperatorRepository.ts#L55-L59`)
- **Caso de Uso:** `RegisterOperatorUseCase.ts` (`src/domain/useCases/RegisterOperatorUseCase.ts`)
- **Hook de React:** `useRegisterOperator.ts` (`src/presentation/hooks/useRegisterOperator.ts`)

Punto de disparo (UI): Se ejecuta en la pantalla de registro (`RegisterScreen.tsx` (`file:///c:/Users/jpatino/CargameApp/src/presentation/screens/RegisterScreen.tsx`)) tras rellenar el formulario de datos de la empresa y pulsar “Crear Cuenta”. Al enviar el archivo RUT en formato PDF, requiere cabecera de tipo `multipart/form-data`. Devuelve el identificador del operador creado (`id_operador`).

POST /crg/usuario

Propósito: Crear la cuenta de acceso de usuario asociada al operador creado anteriormente.

Ubicación en el código:

- **Repositorio:** `ApiOperatorRepository.ts`
(`file:///c:/Users/jpatino/CargameApp/src/infrastructure/repositories/ApiOperatorRepository.ts#L64-L80`)

Punto de disparo (UI): Es una operación transaccional encadenada que se dispara automáticamente en la pantalla de registro (`RegisterScreen.tsx` (`file:///c:/Users/jpatino/CargameApp/src/presentation/screens/RegisterScreen.tsx`)) al recibir respuesta exitosa de `/crg/operador`. Registra el correo electrónico, contraseña hasheada y asocia el `Idoperador` para habilitar el login inmediato.

11.4. Formulario de creación de estaciones (Wizard de 4 pasos)

GET /crg/municipiofull

Propósito: Consultar el catálogo nacional completo de divisiones político-administrativas (Departamentos y Municipios) de Colombia.

Ubicación en el código:

- **Repositorio:** `ApiLocationRepository.ts`
(`file:///c:/Users/jpatino/CargameApp/src/infrastructure/repositories/ApiLocationRepository.ts#L10`)
- **Caso de Uso:** `GetLocationsUseCase.ts` (`src/domain/useCases/GetLocationsUseCase.ts`)
- **Hook de React:** `useLocations.ts` (`src/presentation/hooks/useLocations.ts`)

Punto de disparo (UI): Se dispara automáticamente al cargar la pantalla de creación de estación (`CreateStationScreen.tsx` (`file:///c:/Users/jpatino/CargameApp/src/presentation/screens/CreateStationScreen.tsx`)) - Paso 1). Llena de manera dinámica los selectores desplegables de Departamento y Municipio obligatorios para georreferenciar la estación.

POST /crg/estacioncarga

Propósito: Registrar una nueva estación de carga con su metadata base (nombre, coordenadas GPS, horarios, observaciones) y adjuntar archivos de soporte (Foto de la estación, Certificado RETIE, Declaración de Cumplimiento en PDF).

Ubicación en el código:

- **Repositorio:** `ApiStationCreationRepository.ts`
(`file:///c:/Users/jpatino/CargameApp/src/infrastructure/repositories/ApiStationCreationRepository.ts#L82-L84`)
- **Caso de Uso:** `CreateStationUseCase.ts` (`src/domain/useCases/CreateStationUseCase.ts`)
- **Hook de React:** `useCreateStation.ts` (`src/presentation/hooks/useCreateStation.ts`)

Punto de disparo (UI): Se dispara en el Paso 4: Confirmación de la pantalla de creación (`CreateStationScreen.tsx` (`file:///c:/Users/jpatino/CargameApp/src/presentation/screens/CreateStationScreen.tsx`)). Envía todos los campos mediante `multipart/form-data` y retorna un `id_estacion` para asociar los subcomponentes.

POST /crg/puntocarga

Propósito: Crear e individualizar un cargador físico (Poste de carga) asociado a la estación creada previamente.

Ubicación en el código:

- **Repositorio:** `ApiStationCreationRepository.ts`
(`file:///c:/Users/jpatino/CargameApp/src/infrastructure/repositories/ApiStationCreationRepository.ts#L89-L97`)

Punto de disparo (UI): Se ejecuta automáticamente dentro del bucle de transacciones asíncronas en el Paso 4: Confirmación (`CreateStationScreen.tsx` (`file:///c:/Users/jpatino/CargameApp/src/presentation/screens/CreateStationScreen.tsx`)). Por cada cargador agregado por el usuario en el Paso 2, realiza un POST enviando la marca, potencia máxima, protocolo y lo asocia al `id_estacion` obtenido. Retorna el identificador del poste (`id_punto`).

POST /crg/conectorcarga

Propósito: Registrar e individualizar cada conector físico (manguera de carga) y asociarlo a su respectivo cargador (poste).

Ubicación en el código:

- **Repositorio:** `ApiStationCreationRepository.ts`
(`file:///c:/Users/jpatino/CargameApp/src/infrastructure/repositories/ApiStationCreationRepository.ts#L102-L113`)

Punto de disparo (UI): Es el último paso de la transacción asíncrona encadenada en el Paso 4: Confirmación (`CreateStationScreen.tsx` (`file:///c:/Users/jpatino/CargameApp/src/presentation/screens/CreateStationScreen.tsx`)). Itera sobre los conectores definidos por el CPO en el Paso 3 y realiza un POST con la potencia de salida, tipo de conector (estándar), modalidad de cobro (sesión, kWh), tarifa y precio actual, asociándolo al respectivo `id_punto`.

11.5. Gestión y actualización de estaciones (Edición)

PUT /crg/estacioncarga/{id}

Propósito: Actualizar la metadata base y archivos adjuntos de una estación de carga ya existente (siempre y cuando se encuentre en estado borrador o edición).

Ubicación en el código:

- **Repositorio:** `ApiStationCreationRepository.ts`
(`file:///c:/Users/jpatino/CargameApp/src/infrastructure/repositories/ApiStationCreationRepository.ts#L203-L205`)

Punto de disparo (UI): Se ejecuta en la pantalla de edición (EditStationScreen.tsx (file:///c:/Users/jpatino/CargameApp/src/presentation/screens/EditStationScreen.tsx) - Paso 4) al guardar las modificaciones. Detecta dinámicamente si los archivos multimedia son locales nuevos (vía URI file:// o content://) para subirlos mediante multipart/form-data, o si se conservan los existentes en el servidor.

PUT /crg/puntocarga/{id}

Propósito: Modificar las especificaciones de un cargador físico (poste) ya existente en el sistema.

Ubicación en el código:

- **Repositorio:** ApiStationCreationRepository.ts
(file:///c:/Users/jpatino/CargameApp/src/infrastructure/repositories/ApiStationCreationRepository.ts#L210-L218)

Punto de disparo (UI): Se invoca síncronamente durante la confirmación de cambios en (EditStationScreen.tsx (file:///c:/Users/jpatino/CargameApp/src/presentation/screens/EditStationScreen.tsx) - Paso 4) para actualizar la información técnica de los cargadores previamente registrados.

PUT /crg/conectorcarga/{id}

Propósito: Actualizar las tarifas, potencias o estándares de un conector de carga existente.

Ubicación en el código:

- **Repositorio:** ApiStationCreationRepository.ts
(file:///c:/Users/jpatino/CargameApp/src/infrastructure/repositories/ApiStationCreationRepository.ts#L222-L234)

Punto de disparo (UI): Es la última llamada en la cascada de actualización del Paso 4 de (EditStationScreen.tsx (file:///c:/Users/jpatino/CargameApp/src/presentation/screens/EditStationScreen.tsx)) para guardar precios de cobros actualizados o cambios en los tipos de conector.

11.6. Interacción ciudadana (Calificaciones, reseñas y Habeas Data)

GET /crg/puntos-carga/{id}/comentarios

Propósito: Consultar el histórico de comentarios, estrellas de calificación y reseñas dejadas por conductores en una estación de carga.

Ubicación en el código:

- **Repositorio:** ApiCommentRepository.ts
(file:///c:/Users/jpatino/CargameApp/src/infrastructure/repositories/ApiCommentRepository.ts#L6-L12)
- **Hook de React:** useStationComments.ts (src/presentation/hooks/useStationComments.ts)

Punto de disparo (UI): Se dispara automáticamente en la sección de comentarios (CommentSection.tsx (file:///c:/Users/jpatino/CargameApp/src/presentation/components/CommentSection.tsx)) integrada en la tarjeta detallada del marcador del mapa. Permite visualizar la retroalimentación de la comunidad de conductores.

POST /crg/puntos-carga/{id}/comentarios

Propósito: Registrar un comentario y calificación obligatoria (1-5 estrellas), validando datos del ciudadano y requiriendo aceptación legal de Habeas Data para tratamiento de datos personales.

Ubicación en el código:

- **Repositorio:** ApiCommentRepository.ts
(file:///c:/Users/jpatino/CargameApp/src/infrastructure/repositories/ApiCommentRepository.ts#L14-L20)

- **Hook de React:** `useCreateComment.ts` (`src/presentation/hooks/useCreateComment.ts`)

Punto de disparo (UI): Invocado al rellenar el formulario de reseña en la tarjeta detallada (`CommentSection.tsx` (`file:///c:/Users/jpatino/CargameApp/src/presentation/components/CommentSection.tsx`)) y dar tap a “Enviar Reseña”. Requiere la aceptación explícita de tratamiento de datos personales según normatividad legal.

DELETE `/crg/puntos-carga/{id}/comentarios/{commentId}`

Propósito: Eliminar un comentario propio registrado previamente por el usuario conductor desde su dispositivo móvil.

Ubicación en el código:

- **Repositorio:** `ApiCommentRepository.ts`
(`file:///c:/Users/jpatino/CargameApp/src/infrastructure/repositories/ApiCommentRepository.ts#L22-L24`)

Punto de disparo (UI): El aplicativo guarda de forma local en `AsyncStorage` un token de borrado seguro retornado por el servidor (`delete_token`) al crear el comentario. En (`CommentSection.tsx` (`file:///c:/Users/jpatino/CargameApp/src/presentation/components/CommentSection.tsx`)), la app evalúa si el comentario pertenece al usuario y le revela un botón interactivo “Eliminar” que gatilla este endpoint.

11.7. Documentación y consultas ciudadanas

GET `/crg/documentosmovil`

Propósito: Consultar el catálogo de documentos oficiales, manuales de usuario, plantillas descargables y normatividad del Ministerio de Minas y Energía.

Ubicación en el código:

- **Repositorio:** `ApiDocumentRepository.ts`
(`file:///c:/Users/jpatino/CargameApp/src/infrastructure/repositories/ApiDocumentRepository.ts#L6-L16`)

Punto de disparo (UI): Se dispara en el sidebar del menú del CPO bajo la sección “Documentos” (anteriormente “Mis archivos”). Abre el modal interactivo `DocumentsModal.tsx` listando las guías oficiales para descarga y revisión técnica del CPO.

POST `/crg/contactoestaciones`

Propósito: Entregar sugerencias, PQRS o dudas al canal de atención ciudadana del Ministerio de Minas y Energía.

Ubicación en el código:

- **Repositorio:** `ApiContactRepository.ts`
(`file:///c:/Users/jpatino/CargameApp/src/infrastructure/repositories/ApiContactRepository.ts#L18-L21`)
- **Caso de Uso:** `SendMessageUseCase.ts` (`src/domain/useCases/SendContactMessageUseCase.ts`)
- **Hook de React:** `useContact.ts` (`src/presentation/hooks/useContact.ts`)

Punto de disparo (UI): Invocado en la pantalla “Contáctenos” (`ContactScreen.tsx` (`file:///c:/Users/jpatino/CargameApp/src/presentation/screens/ContactScreen.tsx`)) al pulsar “Enviar Mensaje”, validando de forma rigurosa que el correo tenga estructura correcta (`validateEmail`).

12. Pruebas Automatizadas y Aseguramiento de Calidad

CárgaME posee una suite de tests robusta con cobertura del 100% de éxito, diseñada con un enfoque dual: Pruebas Unitarias para verificar la UI y Property-Based Testing (PBT) para garantizar la consistencia matemática de las utilidades.

```
# Ejecutar toda la suite de tests en Jest
npm test

# Ejecutar tests con reporte detallado de cobertura de código
npx jest --coverage
```

12.1. Cobertura de pruebas por capa

Capa / Módulo	Archivos evaluados	Escenarios probados
Application (Casos de Uso)	GetChargingStationsUseCase, CreateStation, filterStations	Filtrado correcto de estaciones de carga según CPO logueado. Lógica OR en selección múltiple de filtros de municipios.
Infrastructure (Mappers)	ChargingStationMapper, LocationMapper	Transformación precisa de JSON crudo de API a Entidades. Construcción de rutas de imágenes relativas para SecureImage.
Infrastructure (Mappers)	ApiChargingStationRepository, ApiLocationRepository	Validación de endpoints correctos de Axios y llamadas multipart/form-data.
Presentation (Componentes)	SecureImage, FilterModal, StationDetailCard	Renderizado correcto de placeholders ante fallos de red. Ocultamiento de botones de documentos RETIE para invitados.
Presentation (Screens)	LoginScreen, ContactScreen, RegisterScreen	Retroceso correcto en Android para evitar cierres de app. Activación de Toasts animados ante errores controlados.
Utils (Transversales)	jwt.ts, navigation.ts, validators.ts	Decodificación de JWT base64 sin dependencias externas. Formateo preciso de esquemas de geolocalización para mapas.

13. Guía de Instalación y Ejecución Local

Siga los siguientes pasos para configurar y arrancar la aplicación en su entorno de desarrollo local:

13.1. Requisitos previos del sistema

- Node.js **>= 20.x**
- JDK **17** (necesario para compilación Android)
- Android Studio con SDK API 34+ instalado.
- Xcode instalado (únicamente si trabaja en macOS para desarrollo iOS).

13.2. Clonación e instalación de dependencias

```
# Instalar las dependencias de Node.js
npm install
```

Si desarrolla para iOS (en macOS):

```
# Instalar dependencias nativas de CocoaPods
cd ios && pod install && cd ..
```

13.3. Configuración de variables de entorno

Cree un archivo llamado `.env` en el directorio raíz de la aplicación (este archivo está listado en `.gitignore` para proteger las claves en el repositorio):

```
ENV=development
API_URL=https://siveeic.minenergia.gov.co:3011
API_TOKEN=Bearer <inyectar_token_siveeic_aqui>
API_COOKIE=<inyectar_cookie_autenticacion_aqui>
GOOGLE_MAPS_API_KEY=<inyectar_google_cloud_maps_key_aqui>
```

13.4. Ejecución del servidor Metro y la aplicación

```
# Iniciar Metro Bundler (Servidor de empaquetado de JS)
npm start

# En una terminal adicional, lanzar en emulador o dispositivo Android
npm run android

# Lanzar en emulador o dispositivo iOS (requiere macOS)
npm run ios
```

13. Generación de Compilados para Producción

13.1. Generar APK de pruebas (Debug)

```
# Compilar el archivo APK de desarrollo
cd android && gradlew.bat assembleDebug
# El instalable se generará en: android/app/build/outputs/apk/debug/app-debug.apk
```

13.2. Generar APK de Release (sin firmar)

```
cd android && gradlew.bat assembleRelease
# El instalable se generará en: android/app/build/outputs/apk/release/app-release-unsigned.apk
```

13.3. Generar paquete AAB para despliegue en Google Play Store

```
cd android && gradlew.bat bundleRelease
# El paquete final se generará en: android/app/build/outputs/bundle/release/app-release.aab
```

14. Conclusiones

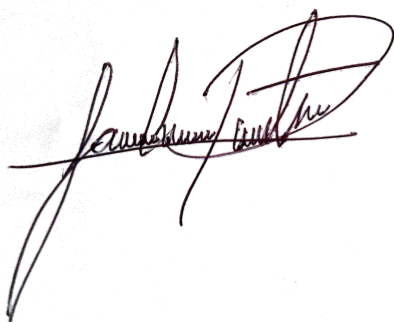
La aplicación móvil CargaME se entrega como una plataforma oficial del Ministerio de Minas y Energía de Colombia, desarrollada en React Native con TypeScript bajo una arquitectura hexagonal pura (Clean Architecture), que centraliza la consulta, búsqueda, registro y gestión de estaciones de carga de vehículos eléctricos a nivel nacional.

- La adopción de Clean Architecture aísla el núcleo de negocio de los detalles de infraestructura, garantizando testeabilidad en aislamiento absoluto, mapeo seguro de datos frente a respuestas inconsistentes de la API y resiliencia ante cambios del backend del Ministerio.
- El stack tecnológico seleccionado (React Native con Fabric, TypeScript, TanStack React Query, Zustand, Axios, react-native-maps y AsyncStorage) responde a necesidades concretas de rendimiento nativo, tipado estricto, gestión de estado y consulta espacial.
- La arquitectura de seguridad multinivel, basada en tokens JWT y en la bifurcación de flujos entre usuario conductor y operador CPO, protege la integridad de los datos de la red de carga y la confidencialidad de documentos RETIE y declaraciones de cumplimiento.
- La integración con la API SIVEEIC cubre 20 endpoints documentados de forma exhaustiva en su propósito, ubicación en el código y punto de disparo en la interfaz, abarcando consulta, autenticación, registro, creación y edición de estaciones, interacción ciudadana y documentación.
- El aseguramiento de calidad se sustenta en una suite de 28 suites / 98 tests con 100% de aprobación y un código libre de errores de tipado en TypeScript, validado en Android sobre emuladores y dispositivos físicos, con iOS listo para compilación en Xcode.

Con base en lo anterior, CargaME se encuentra en un estado funcional y técnicamente consolidado (versión 0.0.1), con la documentación, la guía de instalación y los procedimientos de generación de compilados necesarios para su despliegue y para la incorporación de nuevos desarrolladores al proyecto.

14.1. Elaboración y revisión del documento

El presente documento técnico fue elaborado por el equipo de Servinformación en el marco del contrato KfW N.º 109993, para entrega al Ministerio de Minas y Energía de Colombia.

Elaboró	Revisó	Aprobó
Juan Nicolas Patiño Rodriguez Servinformación Equipo técnico	 Servinformación Gerencia de Proyecto	Ministerio de Minas y Energía Interventoría / Supervisión