CAPSTONE

MODELO 4+1

Contenido

[1. Arquitectura Utilizada 1](#_Toc211200492)

[2. Vista Lógica 2](#_Toc211200493)

[DIAGRAMA DE CLASES 3](#_Toc211200494)

[DIAGRAMA DE SECUENCIA (auto asignación visita) 4](#_Toc211200495)

[3. Vista de Desarrollo 4](#_Toc211200496)

[DIAGRAMA DE COMPONENTES 5](#_Toc211200497)

[DIAGRAMA DE PAQUETES 5](#_Toc211200498)

[4. Vista de Procesos 6](#_Toc211200499)

[DIAGRAMA DE ACTIVIDAD (Técnico se auto-asigna una visita) 6](#_Toc211200500)

[5. Vista Física 7](#_Toc211200501)

[DIAGRAMA DE DESPLIEGUE 7](#_Toc211200502)

[6. Vista de Escenarios (Casos de Uso) 7](#_Toc211200503)

[DIAGRAMA DE CASOS DE USOS 8](#_Toc211200504)

# 1. Arquitectura Utilizada

El proyecto utiliza una arquitectura en capas bajo el enfoque cliente-servidor, implementada mediante un backend monolítico modular desarrollado con Django y Django REST Framework y una base de datos PostgreSQL alojada en Supabase.

La aplicación está estructurada en tres capas principales:

* Capa de Presentación: corresponde al frontend desarrollado en React/Vite, encargado de la interacción con el usuario (técnico o administrador) y de enviar solicitudes al servidor mediante peticiones HTTP.
* Capa de Lógica de Negocio: implementada en el backend con Django/DRF, donde se definen las reglas del sistema, los permisos por rol, la validación de datos y la gestión de visitas, auditorías y asignaciones.
* Capa de Datos: compuesta por la base de datos PostgreSQL (Supabase), que almacena de forma segura la información de usuarios, direcciones, visitas, auditorías e historiales.

La comunicación entre el frontend y el backend se realiza mediante servicios RESTful que intercambian información en formato JSON, utilizando JWT en cookies HttpOnly para la autenticación y protección de sesiones.  
Esta arquitectura fue elegida por su simplicidad, escalabilidad y facilidad de mantenimiento, permitiendo separar responsabilidades y garantizar la seguridad y trazabilidad del sistema.

# 2. Vista Lógica

La vista lógica describe los componentes principales del sistema y cómo se relacionan entre sí. El sistema está organizado en módulos Django que agrupan funcionalidades específicas:

* Usuarios: maneja el registro, autenticación (JWT) y control de roles (técnico, administrador).
* Asignaciones: gestiona la carga de visitas mediante archivos CSV, su autoasignación por técnicos y los estados de cada visita.
* Auditoría: registra los resultados de las visitas, las evidencias (fotos, observaciones) y los issues detectados.
* Core: define permisos, validaciones globales y utilidades compartidas.

Las relaciones entre los módulos se gestionan mediante el ORM de Django, permitiendo definir dependencias entre modelos con claves foráneas y reglas de integridad. Esto garantiza coherencia de los datos y trazabilidad completa.

## DIAGRAMA DE CLASES

Diagrama, Dibujo de ingeniería

El contenido generado por IA puede ser incorrecto.

## DIAGRAMA DE SECUENCIA (auto asignación visita)

Gráfico, Calendario

El contenido generado por IA puede ser incorrecto.

# 3. Vista de Desarrollo

La vista de desarrollo representa la estructura interna del código fuente y la organización técnica del proyecto. El backend está dividido en aplicaciones ('usuarios', 'asignaciones', 'auditoria', 'core') siguiendo la arquitectura en capas. Se utiliza Django REST Framework para construir la API RESTful, documentada automáticamente con drf-spectacular (OpenAPI/Swagger). El control de versiones se realiza con GitHub, empleando ramas por sprint. El frontend consume los endpoints del backend a través de peticiones HTTP, y la base de datos se administra en Supabase. Este diseño modular favorece la mantenibilidad y colaboración.

## DIAGRAMA DE COMPONENTES

Diagrama, Dibujo de ingeniería

El contenido generado por IA puede ser incorrecto.

## DIAGRAMA DE PAQUETES

Diagrama

El contenido generado por IA puede ser incorrecto.

# 4. Vista de Procesos

La vista de procesos describe cómo funcionan los flujos del sistema durante la ejecución:

1. Autenticación: los usuarios inician sesión y reciben tokens JWT almacenados en cookies HttpOnly.  
2. Carga de visitas: el auditor carga un archivo CSV con direcciones y datos técnicos.  
3. Autoasignación: los técnicos consultan visitas disponibles y se autoasignan tareas.  
4. Ejecución de visita: el técnico registra el resultado (autorizado, sin moradores, reagendado, etc.).  
5. Auditoría: si corresponde, se genera una auditoría con fotos y observaciones.  
6. Dashboard: el auditor visualiza métricas generales y el historial de visitas.

Los procesos se ejecutan mediante peticiones HTTP entre frontend y backend. La seguridad se garantiza mediante autenticación JWT y validaciones por rol.

## DIAGRAMA DE ACTIVIDAD (Técnico se auto-asigna una visita)

Imagen que contiene Diagrama

El contenido generado por IA puede ser incorrecto.

# 5. Vista Física

La vista física muestra cómo se despliega el sistema y qué componentes conforman su infraestructura:

* Servidor Backend: desarrollado en Django + DRF, desplegado en entornos locales y plataformas cloud (Render o Railway).
* Base de Datos: PostgreSQL alojada en Supabase.
* Frontend: construido en React/Vite.
* Almacenamiento: evidencias (fotos o CSV) en carpetas protegidas.

## DIAGRAMA DE DESPLIEGUE

Diagrama

El contenido generado por IA puede ser incorrecto.

# 6. Vista de Escenarios (Casos de Uso)

Esta vista muestra ejemplos reales del uso del sistema por parte de los distintos roles:

* Carga de visitas: el auditor carga un CSV con direcciones y técnicos disponibles.
* Autoasignación: el técnico visualiza las visitas y se autoasigna una dirección disponible.
* Registro de visita: el técnico registra el resultado de la visita y adjunta evidencias.
* Auditoría: el auditor revisa las evidencias y valida la ejecución de la visita.
* Reporte: el auditor visualiza en el dashboard los indicadores de cumplimiento y estado general.

## DIAGRAMA DE CASOS DE USOS

Diagrama

El contenido generado por IA puede ser incorrecto.