[FERREMAS]

Documentación Proyecto integración de plataformas

Fecha 29-08-2025

Docente: Miguel Acevedo

Sección : 004V

Integrantes :

Freddy Neilaf

Carlos Seaman

Sylvain Delice

**1 Overview del Proyecto Ferremas**

Ferremas es una **aplicación web full-stack** diseñada para modernizar la gestión de ferreterías. Combina una **plataforma de e-commerce** orientada al cliente con robustas **capacidades de administración interna**, incluyendo gestión de inventario y usuarios. El objetivo principal del proyecto es proporcionar una solución integral que mejore tanto la experiencia de compra para los clientes como la eficiencia operativa para el personal de la ferretería.

La aplicación está construida sobre una arquitectura moderna, con una clara **separación entre el frontend y el backend**, facilitando el desarrollo modular y la escalabilidad. Utiliza **React.js** para una interfaz de usuario dinámica y **Django** con **Django REST Framework** para un backend potente y seguro. La comunicación entre ambos se realiza mediante **APIs RESTful** protegidas con **autenticación JWT**. Además, Ferremas se integra con servicios externos como **Supabase** para la base de datos y almacenamiento de archivos, y **Mercado Pago** para procesar los pagos.

**Arquitectura del Sistema Ferremas**

La plataforma Ferremas se basa en una **arquitectura full-stack** que divide claramente las responsabilidades entre el lado del cliente y el lado del servidor, comunicándose de manera eficiente y segura.

Diagrama

El contenido generado por IA puede ser incorrecto.

Texto

El contenido generado por IA puede ser incorrecto.

**Componentes Clave del Sistema**

* **Capa Cliente (Client Layer):** La interfaz de usuario reside en el **Navegador Web**, donde se ejecuta la **Aplicación React (ferremas\_frontend)**. Esta capa se encarga de la interacción directa con el usuario, la renderización de la UI y la gestión del estado local.
* **Capa de Aplicación (Application Layer):** Compuesta por elementos del frontend que facilitan la experiencia de usuario. Incluye **React Router DOM** para la navegación, **AuthContext** y **CarritoContext** para la gestión centralizada del estado de autenticación y del carrito de compras, y **Material-UI Components** para una UI consistente y moderna. La comunicación con el backend se realiza vía **HTTP/REST**.
* **Capa API (API Layer):** Es el punto de entrada al backend. La **Aplicación Django (ferremas\_backend)** expone las APIs a través de **Django REST Framework**. La seguridad se maneja con **SimpleJWT Authentication** para la gestión de tokens, y **CORS Headers Middleware** asegura la compatibilidad entre dominios.
* **Capa de Lógica de Negocio (Business Logic Layer):** Dentro de la aplicación Django, la lógica de negocio se organiza en "apps" separadas: usuarios (gestión de usuarios y autenticación), productos (catálogo y stock) y carrito (gestión de pedidos y pagos). Estas apps interactúan con la capa de datos.
* **Capa de Datos (Data Layer):** La persistencia de los datos se gestiona con **Supabase PostgreSQL** como base de datos principal, y **Supabase File Storage** para el almacenamiento de archivos (como imágenes de productos). El **ORM de Django** facilita la interacción con PostgreSQL.
* **Servicios Externos (External Services):** Ferremas se integra con **Mercado Pago API** para el procesamiento de pagos, ofreciendo una solución de e-commerce completa.
* **Tecnologías utilizadas**

La elección de tecnologías apunta a un sistema moderno, eficiente y mantenible:

* **Frontend:** Desarrollado con **React.js (v18.2.0)** y **Vite (v6.3.5)**. Se usa **Material-UI (v7.1.0)** y **Bootstrap (v5.3.6)** para los componentes UI y estilos. **React Router DOM (v6.22.3)** maneja la navegación, y **Axios (v1.6.7)** la comunicación con la API.
* **Backend:** Construido con **Django** y **Django REST Framework**. Utiliza **psycopg2-binary** para la conexión a PostgreSQL, **djangorestframework-simplejwt** para la autenticación JWT, **django-cors-headers** para manejar CORS y **Pillow** para el procesamiento de imágenes.

Esta arquitectura asegura una clara separación de preocupaciones, facilitando el desarrollo, mantenimiento y escalabilidad del proyecto Ferremas.

descripción frontend

**1 Aplicación Frontend de Ferremas**

La aplicación frontend de Ferremas es una **Single-Page Application (SPA)** robusta y responsiva, construida con **React.js**. Su propósito principal es servir como la **interfaz de usuario principal** para todos los tipos de usuarios: clientes que exploran productos, empleados que gestionan el inventario y administradores que manejan configuraciones y usuarios.

**Arquitectura de la Aplicación Frontend**

El corazón del frontend es el archivo **App.jsx**, que actúa como el punto de entrada principal y el controlador de enrutamiento. La aplicación se basa en una **arquitectura de componentes**, aprovechando las capacidades de React para construir una interfaz modular y mantenible.

Diagrama

El contenido generado por IA puede ser incorrecto.

Tecnológias del Frontend

Interfaz de usuario gráfica, Aplicación

El contenido generado por IA puede ser incorrecto.

Enrutamiento

El sistema de enrutamiento es fundamental para la experiencia del usuario, implementando un control de acceso basado en roles robusto.

Interfaz de usuario gráfica, Texto

El contenido generado por IA puede ser incorrecto.

Las rutas protegidas se envuelven en el componente PrivateRoute, que verifica el rol del usuario antes de conceder el acceso. Las rutas anidadas (/admin/productos/\*) utilizan el componente ProductosLayout para mantener una estructura consistente en las secciones administrativas.

* Sistema de Navegación

El componente ResponsiveNavbar es el cerebro de la navegación. Se adapta al tamaño de la pantalla (escritorio y móvil) y al rol del usuario autenticado.

* Características Clave
* Navegación Dinámica: Usa la función getNavItems() para mostrar ítems de menú específicos según el rol del usuario (cliente, trabajador, admin).
* Integración de Autenticación: Se conecta con AuthContext para gestionar el estado de sesión del usuario, mostrando opciones de login/registro o un menú de perfil personalizado. El logout vacía la sesión y redirige al inicio.
* Integración del Carrito: Siempre visible, el ícono del carrito muestra una insignia con el número total de ítems, vinculado directamente a la página del carrito.
* Diseño Responsivo: Implementa una barra de navegación para escritorio y un *drawer* (cajón lateral) para dispositivos móviles, utilizando *breakpoints* de Material-UI.
* Páginas Basadas en Roles

Ferremas ofrece interfaces dedicadas para cada rol de usuario, personalizando la experiencia post-autenticación.

* ClienteHome.jsx: Un *dashboard* enfocado en el cliente, mostrando secciones para pedidos recientes y productos favoritos (con implementaciones futuras planificadas), y un acceso directo al catálogo.
* TrabajadorHome.jsx: Un *dashboard* para el personal, organizado en áreas clave como inventario, ventas y reportes, con enlaces directos a las herramientas administrativas correspondientes.

Ambas páginas utilizan el AuthContext para mostrar información del usuario y useNavigate() para la navegación programática, manteniendo una estética consistente con Material-UI.

* Interfaces Específicas
* Interfaz del Catálogo de Productos (ProductosPublicos.jsx)

Esta página es la vitrina de Ferremas para los clientes.

* Visualización: Muestra productos en una cuadrícula responsiva con tarjetas de Material-UI, incluyendo detalles como marca, nombre, precio, stock total y disponibilidad por tienda.
* Manejo de Imágenes: Incluye un manejo robusto de imágenes con *placeholders* para casos de imágenes faltantes o con errores.
* Filtrado por Categorías: Permite a los usuarios filtrar productos mediante una barra lateral interactiva, consultando categorías directamente desde el backend.
* Integración con Carrito: Permite agregar productos al carrito de compras con validación de sesión (requiere login) y notificaciones de éxito/error.
* API: Realiza llamadas a /productos/ y /productos/categories/ para obtener los datos.
* Interfaz de Gestión de Usuarios (Administración)

Destinada a usuarios con roles admin para la gestión de cuentas de trabajador.

* TrabajadoresLista.jsx: Muestra una tabla de trabajadores con opciones de edición y eliminación. Incorpora un diálogo de confirmación para la eliminación de usuarios y gestiona los estados de carga y errores.
* EditarTrabajador.jsx: Proporciona un formulario detallado para modificar la información de un trabajador (nombre de usuario, email, teléfono, dirección, etc.). Carga los datos existentes al montar y envía los cambios con una solicitud PUT.
* API: Se comunica con endpoints como /usuarios/, /usuarios/trabajador/{id}/ y /usuarios/trabajador/{id}/eliminar/ para realizar operaciones CRUD.

Ambas interfaces de gestión de usuarios ofrecen retroalimentación visual (indicadores de carga, alertas de éxito/error) para una experiencia de usuario clara y eficiente.

* 1. **Documentación del Backend de Ferremas**

**Visión General del Backend**

El backend de Ferremas es el motor de la plataforma, desarrollado con **Django** y **Django REST Framework**. Actúa como una **API RESTful** que maneja toda la lógica de negocio, la persistencia de datos y la integración con servicios externos. Está diseñado para ser modular y escalable, comunicándose eficientemente con el frontend de React a través de **APIs REST** protegidas con **JWT**.

### Módulos Principales del Backend

A continuación, se detallan las secciones clave de la API del backend:

**Arquitectura de la API**

La API de Ferremas sigue una **arquitectura modular**, organizada en torno a tres dominios de negocio principales: usuarios, productos y carrito. La configuración principal de URLs (config/urls.py) enruta las solicitudes a la aplicación Django correspondiente, asegurando una separación clara de responsabilidades y un flujo de trabajo eficiente.

Diagrama

El contenido generado por IA puede ser incorrecto.

* **Configuración de la API y Enrutamiento**

La configuración de la API define cómo se manejan las solicitudes HTTP. Esto incluye:

* La **estructura jerárquica de URLs** que delega el enrutamiento a cada aplicación Django.
* **Ajustes de Django** como ROOT\_URLCONF, WSGI\_APPLICATION y DEBUG.
* La configuración de **Django REST Framework** para la autenticación por **JWT**.
* Una **pila de *middleware*** robusta que procesa cada solicitud, manejando **CORS**, autenticación y otras funcionalidades clave.
* El manejo de **archivos estáticos y multimedia** durante el desarrollo.

Diagrama

El contenido generado por IA puede ser incorrecto.

* **Sistema de Gestión de Usuarios**

Implementado en la aplicación usuarios, este sistema es fundamental para la seguridad y el control de acceso:

* **Autenticación JWT personalizada**: Permite a los usuarios iniciar sesión y obtener tokens que incluyen su ID, rol y un indicador si necesitan cambiar la contraseña.
* **Registro de usuarios**: Facilita la creación de nuevas cuentas de cliente.
* **Gestión de contraseñas**: Permite a los usuarios cambiar sus claves de forma segura.
* **Sistema de permisos**: Utiliza clases personalizadas como EsAdministrador para aplicar un **control de acceso basado en roles** (admin, trabajador, cliente) en los *endpoints* de la API.
* **Gestión administrativa de trabajadores**: Los administradores pueden crear, listar, ver, editar y eliminar cuentas de trabajador.

Tabla

El contenido generado por IA puede ser incorrecto.

Tabla

El contenido generado por IA puede ser incorrecto.

* **Sistema de Gestión de Productos**

Este módulo maneja todo lo relacionado con el catálogo e inventario:

* **Modelos clave**: Producto, Tienda y StockTienda, que permiten un seguimiento detallado del inventario en múltiples ubicaciones.
* **Cálculo de stock**: Permite obtener el stock total de un producto en todas las tiendas de forma dinámica.
* **ViewSets de la API**: Proporciona **operaciones CRUD** completas para productos, tiendas y stock, con permisos específicos para cada acción.
* **Manejo de imágenes**: Se encarga de la subida y gestión de imágenes de productos en **Supabase Storage**.
* **Operaciones de stock**: Incluye funcionalidades avanzadas como el **ajuste de stock** en una tienda y la **transferencia de stock** entre tiendas, con validaciones robustas.
* **Gestión y filtrado por categorías**: Permite obtener categorías dinámicamente y filtrar productos por ellas.

Imagen que contiene Gráfico radial

El contenido generado por IA puede ser incorrecto.

* **Carrito de Compras y Pedidos**

Este sistema es crucial para la experiencia de compra online:

* **Modelos centrales**: Carrito, ItemCarrito, Orden e ItemOrden estructuran todo el flujo de compra, desde la selección hasta el pedido final.
* **Operaciones del carrito**: Los usuarios (autenticados o invitados) pueden agregar, remover y actualizar la cantidad de ítems en su carrito.
* **Integración de pagos**: Se conecta con **Mercado Pago** para generar preferencias de pago y procesar transacciones. Aunque el *webhook* recibe notificaciones de pago, la lógica completa de actualización de órdenes aún está en desarrollo.
* **Gestión de órdenes**: El modelo Orden captura toda la información del pedido (cliente, envío, pago) y sigue un **ciclo de vida de estados** (pendiente, pagado, enviado, entregado, cancelado).

Diagrama

El contenido generado por IA puede ser incorrecto.

**Autenticación y Autorización**

**Propósito y Alcance**

Este documento detalla el sistema de **autenticación basado en JWT (JSON Web Tokens)** y **autorización basada en roles** implementado en la plataforma Ferremas. Este sistema es crucial para gestionar el acceso de usuarios, proteger los recursos de la API y asegurar que cada usuario interactúe con la plataforma según su nivel de privilegio (admin, trabajador, cliente, invitado).

Para información sobre el esquema de la base de datos de usuarios, consulta la sección de **Base de Datos**. Para la configuración de *endpoints* de la API y el enrutamiento general, revisa **API Configuration & Routing**.

**Visión General del Sistema**

El sistema de autenticación de Ferremas se basa en **JWT** y establece tres roles de usuario definidos, además del estado de invitado (no autenticado). Este sistema integra la gestión de estado de autenticación en el **frontend de React** con los mecanismos de seguridad del **backend de Django REST Framework**.

El flujo general del sistema se ilustra a continuación:

Gráfico, Diagrama, Gráfico de cajas y bigotes

El contenido generado por IA puede ser incorrecto.

**Descripción del Flujo:**

1. **Inicio de Sesión (Frontend):** El proceso comienza en el **Frontend (A)**. El usuario interactúa con la ResponsiveNavbar (B) y, a través del AuthContext (C), intenta iniciar sesión. La solicitud **POST** a /usuarios/login/ (D) se envía al backend.
2. **Manejo de Tokens en Frontend:** Una vez que el frontend recibe los tokens JWT, los almacena en localStorage (F) y utiliza jwt-decode (E) para decodificar la información del usuario (como el rol) y gestionar el estado de autenticación de la aplicación.
3. **Autenticación en Backend:** En el **Backend (G)**, la solicitud de login es manejada por CustomTokenObtainPairView (H), que utiliza CustomTokenObtainPairSerializer (I) para validar las credenciales del usuario y emitir los tokens JWT (access y refresh tokens) junto con metadatos del usuario (user\_id, role, requiere\_cambio\_password).
4. **Roles de Usuario y Permisos:** El sistema define roles (cliente, trabajador, admin - K, L, M) y un estado de Guest (N). Estos roles, junto con las clases de permiso IsAuthenticated (S) y EsAdministrador (T), determinan el acceso a los recursos.
5. **Recursos Protegidos:** La API del backend tiene **Recursos Protegidos (O)**, que incluyen vistas administrativas (Admin API Views - P), gestión de productos (Product Management - Q), y gestión de usuarios (User Management - R).
6. **Control de Acceso:**
   * IsAuthenticated (S) protege los *endpoints* que solo pueden ser accedidos por usuarios logueados.
   * EsAdministrador (T) otorga acceso a vistas y operaciones específicas de administrador, como las Admin API Views (P) y User Management (R).
   * Los clientes (K) tienen acceso a operaciones de carrito (Cart Operations - PO).
   * Los admin (M) tienen acceso a las Admin API Views (P).
   * Los trabajador (L) tienen acceso a Product Management (Q).
   * Los Guest (N) pueden acceder a ciertas partes de Product Management (ej. ver el catálogo) y funcionalidades de User Management (ej. registro).

* **Implementación de Tokens JWT**

El backend utiliza una implementación personalizada de JWT con Django REST Framework Simple JWT.

* **Serializador de Tokens Personalizado**

La clase CustomTokenObtainPairSerializer extiende la base de Simple JWT para enriquecer la respuesta del token con información crucial:

* user\_id (Integer): Identificador único del usuario en la base de datos.
* role (String): Rol del usuario (admin, trabajador, cliente).
* requiere\_cambio\_password (Boolean): Bandera para forzar un cambio de contraseña si es necesario.

Este serializador configura el campo de autenticación a email (username\_field = 'email') y realiza validaciones adicionales de credenciales y estado de cuenta.

* **Gestión de Tokens en el Frontend**

En el frontend, los tokens JWT se almacenan en localStorage para persistir la sesión. La librería jwt-decode se utiliza para decodificar los tokens y extraer la información del rol del usuario directamente en el cliente, lo que permite una renderización dinámica de la interfaz de usuario (ej. ResponsiveNavbar) basada en los permisos del usuario.

* **Autorización Basada en Roles**

El sistema Ferremas define tres roles de usuario principales con niveles de acceso diferenciados:

* **admin**: Acceso completo al sistema, incluyendo gestión de usuarios, productos y configuraciones generales.
* **trabajador**: Acceso a funcionalidades operativas como gestión de productos, inventario y reportes de ventas.
* **cliente**: Acceso al catálogo de productos, carrito de compras y gestión de sus propios pedidos.
* **Guest (no autenticado)**: Acceso limitado al catálogo público de productos y a las funcionalidades de registro/login.
* **Navegación Basada en Roles en el Frontend**

El componente ResponsiveNavbar adapta su menú de navegación según el rol del usuario autenticado, utilizando la información obtenida del token JWT. Esto asegura que los usuarios solo vean las opciones de menú relevantes para sus permisos.

* **Clases de Permisos y Seguridad de la API**

El backend utiliza clases de permisos personalizadas para asegurar un control de acceso estricto en los *endpoints* de la API.

* **EsAdministrador**

Esta clase de permiso personalizada (usuarios/permissions.py) se asegura de que solo los usuarios autenticados con el rol admin puedan acceder a ciertas vistas, verificando request.user.is\_authenticated y request.user.role == 'admin'.

* **Vistas de API Protegidas**

La protección de los *endpoints* se logra combinando permisos genéricos de Django REST Framework (IsAuthenticated) con permisos personalizados (EsAdministrador), aplicados directamente en las vistas (usuarios/views.py):

* **Público:** /usuarios/registro/ (no requiere autenticación).
* **Autenticado:** /usuarios/cambiar-password/, /usuarios/protegido/ (requiere IsAuthenticated).
* **Solo Admin:** /usuarios/crear-usuario/, /usuarios/, /usuarios/trabajador/{id}/, /usuarios/trabajador/{id}/editar/, /usuarios/trabajador/{id}/eliminar/ (requieren IsAuthenticated y EsAdministrador).
* **Gestión del Estado de Autenticación**
* **Contexto en el Frontend**

El AuthContext de React gestiona el estado global de autenticación en el frontend. Proporciona funciones para el login, logout, acceso a la información del usuario y tokens, permitiendo que los componentes se suscriban a estos cambios de estado y adapten su comportamiento.

* **Validación de Tokens en el Backend**

Cada solicitud a un *endpoint* protegido en el backend pasa por un proceso de validación del token JWT. Este proceso verifica la validez del token (firma, expiración) y extrae la identidad del usuario, aplicando luego las clases de permiso para decidir si la solicitud tiene la autorización necesaria.

* **Gestión de Contraseñas**

El endpoint /usuarios/cambiar-password/ permite a los usuarios autenticados actualizar su contraseña de forma segura. Este proceso utiliza el método set\_password() de Django y, en casos específicos (como cuentas nuevas creadas por un admin), reinicia la bandera requiere\_cambio\_password para quitar la obligación de cambiar la contraseña al primer inicio de sesión.

**3 Esquema de la Base de Datos**

**Propósito y Alcance**

Este documento proporciona una documentación exhaustiva del esquema de la base de datos de Ferremas, incluyendo las definiciones de modelos, relaciones y estructuras de datos. La base de datos principal utiliza **Supabase PostgreSQL** y sigue las convenciones del **ORM de Django** para el diseño de modelos.

Para información sobre los *endpoints* de la API que interactúan con estos modelos, consulta la sección de **Backend API**. Para detalles sobre la autenticación y gestión de roles de usuario, revisa **Autenticación & Autorización**.

* **Visión General**

El esquema de la base de datos de Ferremas está organizado en torno a tres aplicaciones principales de Django, cada una manejando dominios de negocio distintos:

* **usuarios**: Gestión de usuarios y autenticación.
* **productos**: Catálogo de productos y gestión de inventario.
* **carrito**: Carrito de compras y procesamiento de órdenes.

El esquema utiliza **claves primarias UUID** para entidades sensibles como carritos y órdenes, con el fin de prevenir ataques de enumeración. Para entidades más simples,

* **🔐 Autenticación y Permisos**

| **Tabla** | **Descripción** |
| --- | --- |
| auth\_group | Grupos de usuarios. |
| auth\_permission | Permisos individuales asignables. |
| auth\_group\_permissions | Asocia permisos con grupos. |
| usuarios\_customeruser\_groups | Asocia usuarios con grupos. |
| usuarios\_customeruser\_user\_permissions | Asocia permisos directamente con usuarios. |

* **👥 Usuarios**

| **Tabla** | **Descripción** |
| --- | --- |
| usuarios\_customeruser | Usuarios registrados. Incluye rol, dirección, y control de acceso. |
| cliente\_data | Datos extendidos del cliente como historial de compras y puntos. |
| trabajador\_data | Datos extendidos de trabajadores (turno, sucursal, etc.). |

* **🛍️ Productos y Tiendas**

| **Tabla** | **Descripción** |
| --- | --- |
| productos\_producto | Detalles de los productos (SKU, precio, marca, etc.). |
| productos\_tienda | Información de tiendas físicas. |
| productos\_stocktienda | Relación producto-tienda con cantidad y stock mínimo. |

* **🛒 Carrito y Ordenes**

| **Tabla** | **Descripción** |
| --- | --- |
| carrito\_carrito | Carrito de compras vinculado a un usuario. |
| carrito\_itemcarrito | Items añadidos al carrito. |
| carrito\_orden | Orden finalizada (compra). Contiene datos del comprador y estado. |
| carrito\_itemorden | Productos que componen una orden, con precio y cantidad. |

* **✉️ Contacto**

| **Tabla** | **Descripción** |
| --- | --- |
| contacto\_mensajecontacto | Mensajes enviados desde un formulario de contacto. |

* **⚙️ Sistema Django**

| **Tabla** | **Descripción** |
| --- | --- |
| django\_admin\_log | Registro de acciones en el panel admin. |
| django\_content\_type | Mapeo de modelos Django. |
| django\_migrations | Migraciones aplicadas. |
| django\_session | Manejo de sesiones de usuario. |

* **🔗 Relaciones Clave (FK destacadas)**
* carrito\_itemcarrito y carrito\_itemorden → productos vía producto\_id.
* carrito\_carrito, carrito\_orden, cliente\_data, trabajador\_data → usuarios\_customeruser.
* productos\_stocktienda → productos\_producto y productos\_tienda.

Interfaz de usuario gráfica, Aplicación

El contenido generado por IA puede ser incorrecto.

Configuración del Entorno de Desarrollo (Development Setup)

### Propósito y Alcance

Este documento proporciona instrucciones completas para configurar un entorno de desarrollo local para la plataforma de gestión de ferreterías Ferremas. Cubre los requisitos previos, la configuración del entorno, la instalación de dependencias y el flujo de trabajo de desarrollo tanto para el componente **frontend de React.js** como para el **backend de Django**.

Para información sobre la arquitectura general del sistema, consulta la sección de **Arquitectura del Sistema**. Las configuraciones de despliegue y producción no están cubiertas en este conjunto de documentación.

**Prerrequisitos**

El entorno de desarrollo de Ferremas requiere los siguientes componentes de software instalados:

Interfaz de usuario gráfica

El contenido generado por IA puede ser incorrecto.

* **Servicios Externos Requeridos:**
* **Cuenta de Supabase**: Para la base de datos **PostgreSQL** y el almacenamiento de archivos.
* **Cuenta de Desarrollador de Mercado Pago**: Para el procesamiento de pagos (opcional para desarrollo básico).

### Arquitectura del Entorno de Desarrollo

El siguiente diagrama ilustra la configuración del entorno de desarrollo local y cómo interactúan los componentes:

Imagen que contiene Diagrama

El contenido generado por IA puede ser incorrecto.

**Descripción del Flujo:**

1. **.env (A):** Archivo local que contiene variables de entorno sensibles (como los tokens de Mercado Pago). Estas variables son cargadas por el backend (Django).
2. **config/settings.py (B):** El archivo de configuración principal de Django. Carga variables del .env y define la configuración de la base de datos, los ajustes de CORS, la configuración JWT y las rutas para los archivos multimedia.
3. **package.json (C):** Define los *scripts* de *build* y desarrollo para el frontend, así como las dependencias de Node.js.
4. **vite.config.js (D):** Archivo de configuración de Vite (frontend). Configura el servidor de desarrollo, los ajustes de *build* y, crucialmente, la configuración de *proxy* para redirigir las solicitudes de la API (/api/) al servidor de desarrollo de Django (http://localhost:8000).
5. **Django Process (E):** El servidor de desarrollo de Django (python manage.py runserver) carga y utiliza la configuración de settings.py para operar la API, interactuar con la base de datos de Supabase, gestionar archivos en Supabase Storage y comunicarse con la API de Mercado Pago.
6. **Vite Process (F):** El servidor de desarrollo de Vite (npm run dev) sirve la aplicación React. Utiliza las configuraciones de package.json y vite.config.js. Las solicitudes de la API son redirigidas al proceso de Django a través del proxy.

#### Variables de Entorno Clave

Las variables de entorno son cruciales para almacenar información sensible y configuraciones específicas del entorno sin exponerlas directamente en el código fuente. Se deben configurar en el archivo ferremas\_backend/.env para el desarrollo local:

|  |  |  |
| --- | --- | --- |
| **Variable** | **Propósito** | **Requerido** |
| MP\_ACCESS\_TOKEN | Token de acceso para autenticarse con la API de Mercado Pago. Indispensable para realizar y gestionar pagos. | Sí |
| MP\_PUBLIC\_KEY | Clave pública de Mercado Pago, utilizada principalmente en el frontend para inicializar el checkout de pagos. | Sí |

**4 variables de Entorno y Configuración**

Esta sección describe cómo se gestionan las variables de entorno y las configuraciones clave en el proyecto Ferremas para ambos componentes, frontend y backend. Un manejo adecuado de estas configuraciones es fundamental para la seguridad, flexibilidad y facilidad de despliegue de la aplicación en diferentes entornos (desarrollo, producción).

* **Flujo de Configuración**

El siguiente diagrama ilustra cómo se cargan y utilizan las configuraciones y variables de entorno en los diferentes procesos de la aplicación:

Fragmento de código

graph TD

subgraph Configuration Sources

A[.env]

B[config/settings.py]

C[package.json]

D[vite.config.js]

end

subgraph Runtime Environment

E[Django Process (python manage.py runserver)]

F[Vite Process (npm run dev)]

end

A -- MP\_ACCESS\_TOKEN --> B

A -- MP\_PUBLIC\_KEY --> B

B -- Database Configuration --> E

B -- CORS Settings --> F

B -- JWT Settings --> E

B -- Media Files Settings --> E

C -- Build Scripts --> F

C -- Dev Dependencies --> F

D -- Development Server --> F

D -- Build Settings --> F

D -- Proxy Configuration (API Requests to :8000) --> F

F -- HTTP Requests --> E

E -- Database Queries --> DB(Supabase PostgreSQL Database)

E -- File Operations --> Storage(Supabase Storage)

E -- Payment Gateway --> MP(Mercado Pago API)

B --- CfgDB(DATABASES)

B --- CfgCORS(CORS\_ALLOWED\_ORIGINS)

B --- CfgJWT(SIMPLE\_JWT)

B --- CfgMedia(MEDIA\_URL, MEDIA\_ROOT)

CfgDB --> DB

CfgCORS --> F[localhost:5173]

CfgJWT --> E

CfgMedia --> Storage

**Descripción del Flujo:**

* **.env (A):** Archivo local que contiene variables de entorno sensibles (como los tokens de Mercado Pago). Estas variables son cargadas por el backend (Django).
* **config/settings.py (B):** El archivo de configuración principal de Django. Carga variables del .env y define la configuración de la base de datos, los ajustes de CORS, la configuración JWT y las rutas para los archivos multimedia.
* **package.json (C):** Define los *scripts* de *build* y desarrollo para el frontend, así como las dependencias de Node.js.
* **vite.config.js (D):** Archivo de configuración de Vite (frontend). Configura el servidor de desarrollo, los ajustes de *build* y, crucialmente, la configuración de *proxy* para redirigir las solicitudes de la API (/api/) al servidor de desarrollo de Django (http://localhost:8000).
* **Django Process (E):** El servidor de desarrollo de Django (python manage.py runserver) carga y utiliza la configuración de settings.py para operar la API, interactuar con la base de datos de Supabase, gestionar archivos en Supabase Storage y comunicarse con la API de Mercado Pago.
* **Vite Process (F):** El servidor de desarrollo de Vite (npm run dev) sirve la aplicación React. Utiliza las configuraciones de package.json y vite.config.js. Las solicitudes de la API son redirigidas al proceso de Django a través del proxy.
* **Variables de Entorno Clave**

Las variables de entorno son cruciales para almacenar información sensible y configuraciones específicas del entorno sin exponerlas directamente en el código fuente. Se deben configurar en el archivo ferremas\_backend/.env para el desarrollo local:

|  |  |  |
| --- | --- | --- |
| **Variable** | **Propósito** | **Requerido** |
| MP\_ACCESS\_TOKEN | Token de acceso para autenticarse con la API de Mercado Pago. Indispensable para realizar y gestionar pagos. | Sí |
| MP\_PUBLIC\_KEY | Clave pública de Mercado Pago, utilizada principalmente en el frontend para inicializar el checkout de pagos. | Sí |

Exportar a Hojas de cálculo

**Fuentes:** ferremas\_backend/.env líneas 1-4

**Nota:** En entornos de producción, estas variables no deben estar en un archivo .env en el repositorio, sino que deben gestionarse a través del sistema de secretos de la plataforma de *hosting* (ej., variables de entorno en Heroku, AWS Secrets Manager, etc.).

* **Configuración de Desarrollo Importante**

Además de las variables de entorno, los desarrolladores deben estar al tanto de las siguientes configuraciones clave en config/settings.py y vite.config.js:

* **Configuración CORS (CORS\_ALLOWED\_ORIGINS)**:
  + En settings.py, se configura una lista de orígenes permitidos para solicitudes *cross-origin*. Durante el desarrollo, http://localhost:5173 (el puerto de Vite) debe estar incluido para permitir la comunicación entre el frontend y el backend.
  + Esta configuración es vital para evitar problemas de seguridad del navegador al realizar solicitudes entre dominios.
* **Conexión a la Base de Datos (DATABASES)**:
  + Los detalles de conexión a la instancia de **Supabase PostgreSQL** se definen en el diccionario DATABASES dentro de settings.py. Esto incluye ENGINE, NAME, USER, PASSWORD, HOST, PORT.
  + Se recomienda el uso de variables de entorno para estos datos sensibles en producción.
* **Archivos Multimedia (MEDIA\_URL, MEDIA\_ROOT)**:
  + En settings.py, MEDIA\_URL define la URL base para servir archivos multimedia (como imágenes de productos), y MEDIA\_ROOT especifica la ruta del sistema de archivos donde se almacenarán.
  + La integración con **Supabase Storage** se gestiona mediante la lógica de la aplicación para cargar y recuperar archivos utilizando su **API**.
* **Configuración JWT (SIMPLE\_JWT)**:
  + Dentro de settings.py, el diccionario SIMPLE\_JWT contiene ajustes como la expiración de los tokens de acceso y refresco, la firma de claves y otras configuraciones de seguridad relacionadas con la autenticación **JWT**.
* Dependencia claves

| **Package** | **Purpose** |
| --- | --- |
| **Django>=4.2** | Web framework |
| **djangorestframework** | REST API framework |
| **psycopg2-binary** | PostgreSQL database adapter |
| **djangorestframework-simplejwt** | JWT authentication |
| **django-cors-headers** | Cross-origin resource sharing |
| **Pillow** | Image processing |

1. **Project Structure**

El entorno de desarrollo tiene dos carpetas principales:

ferremas\_backend/

├── config/ # Django settings and configuration

├── productos/ # Product management app

├── usuarios/ # User management app

├── carrito/ # Shopping cart app (inferred from API structure)

├── manage.py # Django management script

├── requirements.txt # Python dependencies

└── venv/ # Virtual environment

ferremas\_frontend/

├── src/ # React source code

├── public/ # Static assets

├── package.json # Node.js dependencies and scripts

├── vite.config.js # Vite configuration

└── node\_modules/ # Node.js dependencias

**5 Integración de Pagos (Payment Integration)**

**Propósito y Alcance**

Este documento cubre la implementación de la integración con la pasarela de pagos **Mercado Pago** en el sistema Ferremas. La integración de pagos permite a los clientes completar sus compras a través de la plataforma segura de Mercado Pago, manejando tanto los flujos de pago para usuarios autenticados como para invitados.

Para información sobre la funcionalidad del carrito de compras y la gestión de órdenes, consulta la sección de **Shopping Cart & Orders**. Para detalles sobre la autenticación y la gestión de usuarios, consulta **Authentication & Authorization**.

**Visión General**

El sistema Ferremas se integra con **Mercado Pago**, una destacada pasarela de pagos en Latinoamérica, para procesar los pagos de los clientes. La integración se implementa en el **backend de Django** utilizando el **SDK de Mercado Pago** y sigue un flujo de pago basado en preferencias, donde se crean sesiones de pago y los usuarios son redirigidos a las páginas de *checkout* alojadas por Mercado Pago.

**Componentes Clave**

|  |  |  |
| --- | --- | --- |
| **Componente** | **Propósito** | **Implementación/Referencia** |
| CarritoViewSet | Controlador principal de la API para operaciones de pago. | ferremas\_backend/carrito/views.py líneas 14-177 |
| **Mercado Pago SDK** | SDK de terceros para la comunicación con la API de Mercado Pago. | ferremas\_backend/carrito/views.py líneas 7-12 |
| **Configuración de Entorno** | Almacenamiento seguro de credenciales. | ferremas\_backend/.env líneas 1-4 |
| **Webhook Handler** | Procesamiento de notificaciones de estado de pago. | ferremas\_backend/carrito/views.py líneas 135-142 |

**Configuración**

**Variables de Entorno**

La integración de pagos requiere dos credenciales de Mercado Pago configuradas como variables de entorno:

|  |  |  |
| --- | --- | --- |
| **Variable** | **Propósito** | **Ubicación** |
| MP\_ACCESS\_TOKEN | Token de acceso para autenticación en la API del lado del servidor. | ferremas\_backend/.env línea 1 |
| MP\_PUBLIC\_KEY | Clave pública de Mercado Pago (para uso futuro en integración del lado del cliente). | ferremas\_backend/.env línea 3 |
| El token de acceso se carga en las vistas de pago utilizando os.getenv() con un valor predeterminado para desarrollo (ferremas\_backend/carrito/views.py línea  **Inicialización del SDK**  El SDK de Mercado Pago se inicializa globalmente en el módulo de vistas (ferremas\_backend/carrito/views.py línea  Python  mp = mercadopago.SDK(MP\_ACCESS\_TOKEN)  ARQUITECTURA DEL FLUJO DE PAGO    **Gestión del Estado del Carrito y Pago**  Las acciones clave del CarritoViewSet orquestan el estado del carrito y el proceso de pago:   1. **add\_item()**: Permite al usuario añadir productos al carrito, transicionando el carrito de un estado vacío a Carrito with ItemCarrito objects. 2. **get\_cart()**: Recupera el contenido actual del carrito para que el usuario lo revise. 3. **create\_mercadopago\_preference()**: Valida el carrito y genera la preferencia de pago en Mercado Pago, llevando el estado a MercadoPago Preference Created y luego Payment in Progress una vez que el usuario es redirigido. 4. **Redirección a init\_point**: El usuario es enviado a la plataforma de Mercado Pago para completar la transacción. 5. **webhook()**: Recibe notificaciones de Mercado Pago sobre el estado del pago (Payment Completed o Payment Failed). Esta es la etapa donde se debe actualizar el estado final de la Orden.   ENDPOINT API PAGO   |  |  |  |  | | --- | --- | --- | --- | | Acción | Método HTTP | Propósito | Líneas de Código | | create\_mercadopago\_preference | POST | Crea la preferencia de pago en Mercado Pago. | ferremas\_backend/carrito/views.py líneas 90-133 | | webhook | POST | Maneja las notificaciones de estado de pago de Mercado Pago. | ferremas\_backend/carrito/views.py líneas 135-142 | | add\_item | POST | Añade productos al carrito. | ferremas\_backend/carrito/views.py líneas 16-46 | | get\_cart | GET | Recupera el contenido del carrito. | ferremas\_backend/carrito/views.py líneas 77-88 | | remove\_item | POST | Elimina ítems del carrito. | ferremas\_backend/carrito/views.py líneas 48-75 | | update\_quantity | POST | Actualiza las cantidades de los ítems en el carrito. | ferremas\_backend/carrito/views.py líneas 144-176 | |  |  |
| * **Lógica de Creación de Preferencia**   La acción create\_mercadopago\_preference (ferremas\_backend/carrito/views.py líneas 91-133) es el corazón de la integración de pagos:   * **Validación del Carrito**: Verifica que el carrito no esté vacío. * **Mapeo de Ítems**: Convierte los objetos ItemCarrito del modelo de Django al formato de ítem requerido por Mercado Pago.   Python  items\_mp = [{  "title": item.producto.nombre,  "quantity": item.cantidad,  "unit\_price": float(item.producto.precio),  "currency\_id": "CLP" # Moneda de Chile  } for item in carrito.items.all()]   * **Configuración de Preferencia**: Establece las URLs de redirección para los diferentes resultados del pago.   + **Éxito:** http://localhost:5173/checkout/success   + **Fallo:** http://localhost:5173/checkout/failure   + **Pendiente:** http://localhost:5173/checkout/pending * **Manejo de la Respuesta**: Retorna el preference\_id y las URLs de *checkout* (sandbox\_init\_point/external\_resource) al frontend. * **Integración de Webhooks** * **Implementación del *Webhook Handler***   La acción webhook (ferremas\_backend/carrito/views.py líneas 135-142) procesa las notificaciones de estado de pago enviadas por Mercado Pago:  Python  @action(detail=False, methods=['post'])  def webhook(self, request):  payment\_id = request.data.get("data", {}).get("id")  if payment\_id:  payment\_info = mp.payment().get(payment\_id)["response"]  # Lógica de actualización del estado de la orden se implementaría aquí  return Response({"status": "ok"})  return Response({"status": "ignored"})   * **Procesamiento del Estado del Pago**   El *webhook* extrae el payment\_id del *payload* de la notificación y recupera los detalles completos del pago utilizando mp.payment().get(payment\_id). La información de estado del pago (ej. approved, pending, rejected) se utilizaría para actualizar el estado correspondiente de la Orden en la base de datos, aunque se observa que esta funcionalidad está TODO en el código fuente actual.  **Fuentes:** ferremas\_backend/carrito/views.py líneas 135-142   * **Soporte a Usuarios y Clientes Invitados**   El sistema de pagos de Ferremas está diseñado para soportar tanto a usuarios autenticados como a clientes invitados:   * **Usuarios Autenticados**: El carrito está asociado directamente con request.user a través del campo Carrito.usuario. La recuperación del carrito se realiza automáticamente utilizando Carrito.objects.get\_or\_create(usuario=user). * **Usuarios Invitados**: El carrito se identifica mediante un guest\_cart\_id (UUID) enviado por el frontend. Estos carritos se crean con usuario=None. El guest\_cart\_id se retorna al frontend para mantener la persistencia de la sesión del invitado.   Este enfoque de doble modo se aplica en todas las acciones del carrito, incluyendo la creación de la preferencia de pago (ferremas\_backend/carrito/views.py líneas 92-98).  **Fuentes:** ferremas\_backend/carrito/views.py líneas 23-29, ferremas\_backend/carrito/views.py líneas 92-98   * **Manejo de Errores**   El sistema implementa un manejo de errores robusto para la creación de preferencias de pago y otras operaciones relacionadas:   * **Errores en la Creación de Preferencia de Pago** * **Validación de Carrito Vacío**: Retorna un error 400 Bad Request si el carrito está vacío al intentar crear una preferencia (ferremas\_backend/carrito/views.py líneas 99-100). * **Validación de Respuesta de la API de MP**: Comprueba la estructura de la respuesta de Mercado Pago y gestiona errores (500 Internal Server Error) si la respuesta es inválida (ferremas\_backend/carrito/views.py líneas 124-127). * **Registro de Depuración**: Imprime detalles de la respuesta de Mercado Pago para facilitar la depuración (ferremas\_backend/carrito/views.py línea 119).   Escenarios Comunes de Error   |  |  |  |  |  | | --- | --- | --- | --- | --- | | Tipo de Error | Código HTTP | Condición | Implementación (Líneas de Código) |  | | Carrito Vacío | 400 Bad Request | No hay ítems en el carrito al intentar pagar. | ferremas\_backend/carrito/views.py líneas 99-100 |  | | Error de la API de Mercado Pago | 500 Internal Server Error | Respuesta inválida o error interno de Mercado Pago. | ferremas\_backend/carrito/views.py líneas 124-127 |  | | Producto No Encontrado | 404 Not Found | ID de producto inválido en add\_item. | ferremas\_backend/carrito/views.py líneas 45-46 |  | | Ítem No Encontrado | 404 Not Found | ID de ítem inválido en operaciones del carrito. | ferremas\_backend/carrito/views.py líneas 171-172 |  |   **6 Conclusiones del Backend de Ferremas**   * **Resumen del Proyecto y Logros Clave**   La plataforma Ferremas ha sido diseñada y desarrollada como una solución integral para la gestión de una ferretería, implementando un sólido backend basado en **Django** y **Django REST Framework**. A lo largo de este proceso de documentación, hemos detallado un sistema robusto y modular que soporta las operaciones esenciales del negocio.  Los logros clave del backend incluyen:   * **Arquitectura RESTful Robusta**: Se ha implementado una **API RESTful** bien definida, organizada modularmente, que permite una comunicación eficiente y escalable con el *frontend* y otros servicios. * **Gestión de Usuarios y Seguridad**: Un sistema de autenticación **JWT** personalizado con roles de usuario (admin, trabajador, cliente) y clases de permiso granular (EsAdministrador) garantiza un control de acceso seguro y flexible. * **Gestión Integral de Productos e Inventario**: Funcionalidades completas para el manejo del catálogo de productos, seguimiento de stock por tienda, y operaciones avanzadas como ajustes y transferencias entre sucursales. * **Flujo de Carrito de Compras y Pedidos**: Un sistema estructurado para el proceso de compra online, desde la gestión del carrito hasta la creación de órdenes, con soporte para usuarios autenticados e invitados. * **Integración de Pagos con Mercado Pago**: Una integración funcional con Mercado Pago, que permite el procesamiento de transacciones y la gestión de preferencias de pago, sentando las bases para una experiencia de compra fluida. * **Infraestructura de Datos Resiliente**: El uso de **Supabase PostgreSQL** como base de datos principal, junto con **Supabase Storage** para archivos multimedia, proporciona una solución de persistencia de datos fiable y escalable. * **Entorno de Desarrollo Bien Definido**: Una configuración clara para el entorno de desarrollo local, incluyendo la gestión de dependencias y variables de entorno, facilita la incorporación de nuevos desarrolladores y el mantenimiento del proyecto. * **Estado Actual y Próximos Pasos**   El backend de Ferremas se encuentra en un estado funcional y sienta una base sólida para futuras expansiones. Sin embargo, como todo proyecto en desarrollo, existen áreas identificadas para mejora y futuras implementaciones que serán abordadas en próximas etapas:   * **Completar la Actualización del Estado de Órdenes vía Webhook**: Finalizar la lógica para actualizar completamente el estado de la orden en la base de datos basándose en la información recibida del *webhook* de Mercado Pago (payment\_id, estado de la transacción). * **Monitoreo y Logging Detallado**: Implementar herramientas de monitoreo y un sistema de *logging* centralizado para mejorar la observabilidad, la detección temprana de problemas y la depuración en entornos de producción. * **Estrategias de Testing Exhaustivas**: Desarrollar y aplicar pruebas unitarias, de integración y funcionales de forma sistemática para garantizar la estabilidad y fiabilidad del código. * **Optimización de Rendimiento**: A medida que la plataforma crezca y la carga de usuarios aumente, será necesario revisar y optimizar el rendimiento de las consultas a la base de datos y los *endpoints* de la API más críticos. * **Seguridad Adicional**: Aunque ya se cuenta con una base de seguridad sólida (JWT, permisos), se pueden explorar y aplicar medidas adicionales como *rate limiting*, protección CSRF en formularios no-API si aplica, y auditorías de seguridad periódicas.   En conclusión, el backend de Ferremas es una pieza central y funcional que demuestra una arquitectura bien pensada y una implementación efectiva de las funcionalidades clave para una plataforma de gestión de ferreterías. El camino a seguir implica consolidar las funcionalidades existentes y expandir la plataforma para atender futuras necesidades del negocio y asegurar su operación robusta y segura. |  |  |