**Vistas Arquitectónicas 4+1 del Proyecto Ferremas**

Las "vistas 4+1" son un marco arquitectónico bien establecido para describir la arquitectura de un sistema de software desde múltiples perspectivas concurrentes, abordando las preocupaciones de diferentes stakeholders (usuarios finales, desarrolladores, gestores, personal de operaciones, etc.). Para el proyecto Ferremas, estas vistas se aplican de la siguiente manera:

1. **Vista Lógica (Logical View)**

**Interfaz de usuario gráfica, Diagrama

El contenido generado por IA puede ser incorrecto.**

**Descripción:** Esta vista se centra en la funcionalidad que el sistema proporciona a los usuarios finales y cómo se descompone el sistema en componentes de software más pequeños, abstracciones y elementos lógicos. Se ocupa de las responsabilidades funcionales y las interacciones entre los componentes clave.

**Componentes Clave en Ferremas:**

* **Módulos de Aplicación (Django Apps):**
  + usuarios: Gestiona todo lo relacionado con los usuarios, incluyendo su registro, autenticación (basada en JWT), perfiles, y la asignación de roles (admin, trabajador, cliente).
  + productos: Maneja el catálogo de productos, la definición de categorías y marcas, y la gestión del inventario por tienda.
  + carrito: Encapsula la lógica del carrito de compras, la creación y gestión de órdenes, y la integración con la pasarela de pagos.
* **Modelos de Datos (Django Models):** Representan las entidades de negocio persistentes y sus relaciones, según se define en el esquema de la base de datos: CustomerUser, Producto, Tienda, StockTienda, Carrito, ItemCarrito, Orden, ItemOrden. Se definen sus atributos, tipos de datos, claves primarias (UUID/Integer) y relaciones (ej., OneToOne, ForeignKey con políticas CASCADE, SET\_NULL, PROTECT).
* **Vistas de la API (Django REST Framework ViewSets/APIViews):** Son los puntos de entrada para la interacción externa. Encapsulan la lógica de negocio para las operaciones CRUD y personalizadas (ej., UserViewSet, ProductViewSet, CartViewSet).
* **Serializadores (Django REST Framework Serializers):** Definen cómo los datos se convierten entre formatos Python y JSON (o viceversa) para las peticiones y respuestas de la API, y manejan la validación de datos.
* **Clases de Permiso y Autenticación:** Lógicas de autorización (IsAuthenticated, EsAdministrador) y mecanismos de autenticación (JWT) que controlan el acceso a los recursos.
* **Módulos de Servicios/Utilidades:** Cualquier lógica auxiliar que no encaje directamente en un modelo o vista, como la inicialización del SDK de Mercado Pago o funciones para el manejo de archivos.

**Relaciones y Abstracciones:** Las relaciones entre los componentes lógicos se establecen a través de las relaciones de claves foráneas en los modelos, la invocación de métodos entre vistas y serializadores, y el uso de permisos. Las abstracciones se manifiestan en la clara separación de responsabilidades funcionales en cada aplicación de Django, siguiendo el principio de la cohesión y el bajo acoplamiento.

**Referencias en la Documentación:**

* **Database Schema:** Para la estructura detallada de los modelos de datos y sus relaciones.
* **API Configuration & Routing:** Para la definición de los *endpoints* y su mapeo a las vistas de la API.
* **User Management System, Product Management System, Shopping Cart & Orders:** Para la funcionalidad específica y la lógica de negocio de cada dominio.
* **Authentication & Authorization:** Para el diseño de la seguridad basada en roles y JWT.

**2. Vista de Procesos (Process View)**

**Una captura de pantalla de una red social

El contenido generado por IA puede ser incorrecto.**

**Descripción:** Esta vista se enfoca en los aspectos dinámicos del sistema, incluyendo los procesos concurrentes, la distribución, la comunicación entre procesos, el rendimiento y la escalabilidad. Describe cómo los procesos interactúan y cómo se ejecutan las operaciones del sistema.

**Componentes Clave en Ferremas:**

* **Peticiones HTTP/API REST:** Son el principal mecanismo de comunicación síncrona entre el cliente (Frontend React) y el servidor (Backend Django). Cada petición es manejada como un proceso que recorre la pila de la API.
* **Servidor de Aplicaciones (Django Development Server / Gunicorn/uWSGI en producción):** Escucha las peticiones entrantes, las enruta a la vista correspondiente de Django REST Framework, y gestiona el *pool* de *workers* o *threads* que procesan estas peticiones de manera concurrente.
* **Base de Datos (Supabase PostgreSQL):** Actúa como un proceso centralizado para la persistencia de datos. Gestiona la concurrencia de las transacciones para asegurar la integridad de los datos, especialmente en operaciones críticas como la gestión de stock o la creación de órdenes.
* **Flujo de Autenticación JWT:** Un proceso que implica una secuencia de pasos para la emisión y validación de tokens.
* **Flujo de Pago con Mercado Pago:** Un proceso distribuido que combina interacciones síncronas (creación de la preferencia de pago) y asíncronas (recepción de *webhooks* de notificación de pago) con un servicio externo.
* **Gestión de Archivos (Supabase Storage):** Proceso de carga y descarga de imágenes de productos, que implica interacción con un servicio de almacenamiento externo.

**Flujos de Procesos Críticos:**

* **Flujo de Inicio de Sesión:** El *frontend* envía credenciales; el backend valida y emite tokens JWT; el *frontend* almacena tokens y gestiona el estado; el *frontend* usa tokens para futuras peticiones; el backend valida tokens y otorga acceso.
* **Flujo de Proceso de Compra:** El cliente agrega ítems al carrito; el *frontend* solicita la creación de preferencia de pago; el backend interactúa con la API de Mercado Pago; el cliente es redirigido a la página de *checkout* de MP; MP procesa el pago; MP envía un *webhook* asíncrono al backend para notificar el estado final del pago; el backend actualiza el estado de la Orden.
* **Gestión Concurrente de Inventario:** Consideración de cómo múltiples peticiones para la compra o ajuste del mismo producto afectan el stock, confiando en las propiedades transaccionales de la base de datos para asegurar la consistencia.

**Referencias en la Documentación:**

* **Authentication & Authorization:** Para el flujo de autenticación basado en JWT y las políticas de acceso.
* **Payment Integration:** Para el flujo detallado de la integración con Mercado Pago y el manejo de *webhooks*.

**3. Vista de Desarrollo (Development View)**

**Interfaz de usuario gráfica, Aplicación, Word

El contenido generado por IA puede ser incorrecto.**

**Descripción:** Esta vista se enfoca en la organización del código del sistema para facilitar el desarrollo, la gestión del código fuente, el *build* y las pruebas. Es la vista de la "caja blanca" para los programadores.

**Componentes Clave en Ferremas:**

* **Estructura de Directorios del Proyecto:**
  + ferremas\_backend/: Contiene todas las aplicaciones Django (usuarios, productos, carrito), la configuración principal (config/), el script manage.py, las dependencias de Python (requirements.txt) y el entorno virtual (venv/).
  + ferremas\_frontend/: Contiene el código fuente de React (src/), los activos estáticos (public/), las dependencias y scripts de Node.js (package.json), la configuración de Vite (vite.config.js) y las dependencias de Node (node\_modules/).
* **Módulos y Paquetes (Python):** Las aplicaciones Django (usuarios, productos, carrito) están diseñadas como módulos Python cohesivos, cada uno con sus propios modelos, vistas, serializadores y urls.
* **Herramientas de Desarrollo:** Python, Node.js, pip, npm, Git para control de versiones.
* **Gestión de Dependencias:** requirements.txt especifica las librerías Python; package.json lista las dependencias de Node.js (React, Vite, etc.).
* **Entornos Virtuales:** El uso de venv es estándar para aislar las dependencias del backend, evitando conflictos entre proyectos.
* **Configuración del Proyecto:** config/settings.py centraliza la configuración del backend, mientras vite.config.js gestiona la configuración del *bundler* del frontend.
* **Variables de Entorno:** El archivo .env para el desarrollo local permite la configuración de variables sensibles sin exponerlas en el código fuente.
* **Scripts de Desarrollo:** Colección de comandos (python manage.py ..., npm run ...) para automatizar tareas como iniciar servidores, correr migraciones, instalar dependencias, etc.

**Organización y Convenciones:** El proyecto sigue las convenciones recomendadas por Django y React, promoviendo un código limpio, modular y mantenible. Se utilizan *linters* y *formatters* para mantener la consistencia del estilo de código.

**Referencias en la Documentación:**

* **Development Setup:** Para una guía paso a paso sobre cómo configurar el entorno de desarrollo y las herramientas.
* **Environment Variables and Configuration:** Para el manejo de las variables de entorno y los archivos de configuración.

**4. Vista Física (Physical View / Deployment View)**

**Diagrama

El contenido generado por IA puede ser incorrecto.**

**Descripción:** Esta vista describe la topología de hardware del sistema, cómo los componentes de software se mapean a los nodos físicos o virtuales de despliegue, y la distribución del sistema en un entorno de producción. Se enfoca en la infraestructura, el rendimiento, la disponibilidad y la escalabilidad del sistema desplegado.

**Componentes Clave y Su Mapeo en Ferremas (Considerando un entorno de producción):**

* **Frontend (Aplicación React):**
  + **Mapeo:** Desplegado como archivos estáticos en un servicio de *hosting* de objetos o CDN (ej. Vercel, Netlify, AWS S3 + CloudFront).
  + **Consideraciones:** Baja latencia, alta disponibilidad, escalabilidad horizontal automática, cacheo en el borde.
* **Backend (Aplicación Django):**
  + **Mapeo:** Alojado en uno o varios servidores de aplicaciones (ej. un servidor virtual en un proveedor cloud como AWS EC2, Google Cloud Run, o plataformas PaaS como Heroku). Utilizará un servidor WSGI como Gunicorn o uWSGI, posiblemente detrás de un *proxy* inverso (ej. Nginx) para manejar peticiones, *load balancing* y SSL.
  + **Consideraciones:** Escalabilidad vertical (más recursos en una máquina) u horizontal (más máquinas), monitoreo de rendimiento, gestión de procesos.
* **Base de Datos (Supabase PostgreSQL):**
  + **Mapeo:** Servicio de base de datos administrado por Supabase.
  + **Consideraciones:** Alta disponibilidad, *backups* automáticos, escalabilidad de almacenamiento y cómputo, seguridad de red.
* **Almacenamiento de Archivos (Supabase Storage):**
  + **Mapeo:** Servicio de almacenamiento de objetos proporcionado por Supabase.
  + **Consideraciones:** Escalabilidad masiva para imágenes y otros archivos, redundancia de datos.
* **Pasarela de Pagos (Mercado Pago API):**
  + **Mapeo:** Servicio externo de terceros al que el backend se conecta a través de HTTPS.
  + **Consideraciones:** Dependencia de la disponibilidad y latencia del servicio externo, seguridad de la comunicación (OAuth/tokens).
* **Servicio de Webhooks (Backend webhook endpoint):**
  + **Mapeo:** Un *endpoint* público y accesible en el servidor de backend, configurado en Mercado Pago para recibir notificaciones asíncronas.
  + **Consideraciones:** Debe ser accesible desde el exterior, y la lógica de manejo de *webhooks* debe ser robusta y idempotente.
* **DNS y Balanceadores de Carga:** (Implícito para sistemas escalables)
  + **Mapeo:** Configurados para dirigir el tráfico de los dominios personalizados a los respectivos servidores del frontend y backend.

**Consideraciones de Despliegue:**

* Uso de **HTTPS** para toda la comunicación entre componentes y con los clientes.
* Gestión segura de variables de entorno de producción (ej. sistemas de secretos del proveedor cloud).
* Implementación de pipelines de **CI/CD** (Integración Continua/Despliegue Continuo) para automatizar los *builds* y despliegues.
* Estrategias de *logging* y monitoreo para la visibilidad del estado del sistema en producción.

**Referencias en la Documentación:**

* Aunque la documentación actual se enfoca en el desarrollo local, esta vista proporciona una base para la futura sección de "Despliegue e Infraestructura".
* Las secciones sobre **Database Schema** y **Payment Integration** tocan aspectos de servicios externos utilizados en producción.

**+1. Vista de Escenarios (Scenarios View / Use Case View)**

* **Propósito:** Esta vista es transversal y complementaria a todas las demás. Describe los casos de uso principales del sistema, ilustrando cómo los componentes de software (lógicos), los procesos (dinámicos), la organización del código (desarrollo) y la topología de despliegue (físico) interactúan para satisfacer un requisito funcional clave.
* **Preocupaciones que aborda:** Casos de uso, flujo de trabajo, funcionalidad del sistema desde la perspectiva del usuario, validación de la arquitectura.
* **Para Ferremas:** Los escenarios demuestran cómo los diferentes actores (clientes, administradores, trabajadores) interactúan con el sistema para lograr sus objetivos.
* **Diagrama de la Vista de Escenarios (Ejemplo: "Realizar una Compra"):**

Diagrama

El contenido generado por IA puede ser incorrecto.

**Ejemplos de Casos de Uso Clave en Ferremas:**

1. **Registro de Nuevo Cliente:**
   * **Lógica:** Módulo usuarios, modelo CustomerUser, vista UserViewSet (registro acción).
   * **Proceso:** El *frontend* envía una petición POST con los datos de registro; el backend valida los datos, crea un nuevo CustomerUser con rol cliente, y opcionalmente autentica al usuario.
   * **Desarrollo:** Código en ferremas\_backend/usuarios/views.py y usuarios/serializers.py.
   * **Físico:** Petición desde el navegador del usuario al servidor frontend, proxy al servidor backend de Django, que interactúa con la base de datos Supabase PostgreSQL.
   * Diagrama, Esquemático

     El contenido generado por IA puede ser incorrecto.
2. **Cliente Realiza una Compra (con Pago vía Mercado Pago):**

Diagrama

El contenido generado por IA puede ser incorrecto.

* + **Lógica:** Módulos carrito y productos, modelos Producto, ItemCarrito, Orden, ItemOrden, vistas del CartViewSet.
  + **Proceso:** El cliente añade ítems al carrito (add\_item); el *frontend* solicita la creación de una preferencia de pago (create\_mercadopago\_preference); el backend se comunica con la API de Mercado Pago, redirigiendo al cliente; el cliente completa el pago en la plataforma de MP; MP envía una notificación de *webhook* asíncrona al backend; el backend procesa el *webhook* para actualizar el estado de la Orden.
  + **Desarrollo:** Código en ferremas\_backend/carrito/views.py, integración con mercadopago SDK, configuración de URLs en settings.py.
  + **Físico:** Interacción entre navegador (frontend), servidor backend de Django, la API de Mercado Pago y el servicio de *webhooks*.

1. **Trabajador de Tienda Actualiza el Stock de un Producto:**

Diagrama

El contenido generado por IA puede ser incorrecto.

* + **Lógica:** Módulo productos, modelos Producto y StockTienda, vistas del ProductViewSet (acciones para gestión de stock).
  + **Proceso:** Un trabajador (autenticado) accede al *frontend* de gestión, solicita un ajuste de stock para un producto en una tienda específica; el backend valida las credenciales y los permisos (IsAuthenticated, rol trabajador o admin), y actualiza la cantidad en el modelo StockTienda.
  + **Desarrollo:** Código en ferremas\_backend/productos/views.py, productos/serializers.py, y clases de permiso en usuarios/permissions.py.
  + **Físico:** Petición del navegador al servidor frontend, proxy al servidor backend, interacción con la base de datos Supabase PostgreSQL.

1. **Administrador Gestiona Usuarios:**

Diagrama

El contenido generado por IA puede ser incorrecto.

* + **Lógica:** Módulo usuarios, modelo CustomerUser, vistas del UserViewSet (acciones de administración de usuarios).
  + **Proceso:** Un admin (autenticado) accede al *frontend* de administración de usuarios; solicita la lista de usuarios, o la creación/edición/eliminación de un usuario; el backend valida la autenticación y el permiso EsAdministrador; realiza la operación en la base de datos.
  + **Desarrollo:** Código en ferremas\_backend/usuarios/views.py, usuarios/serializers.py, y usuarios/permissions.py.
  + **Físico:** Interacción similar al caso del "Trabajador", pero con requisitos de permisos más elevados.

**Las descripciones detalladas de los sistemas de Gestión de Usuarios, Gestión de Productos, Carrito de Compras y Pedidos, Autenticación y Autorización, e Integración de Pagos, incluyendo sus escenarios específicos e implementaciones, se encuentran en el documento adjunto "Das Ferremas" (,comunicación personal, 29 de mayo de 2025).**