Documento Técnico

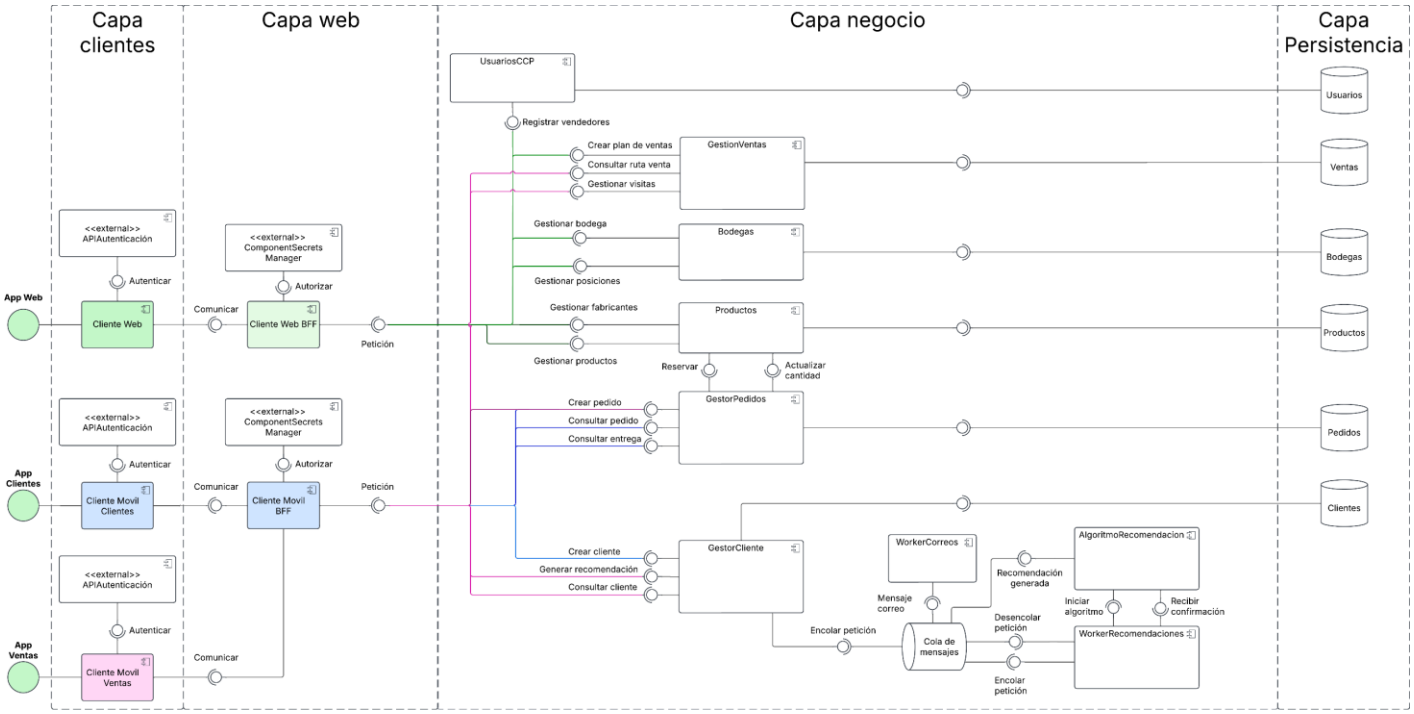
Proyecto: Logística CCP

1. Descripción General

Logística CCP es una aplicación web desarrollada para la gestión logística de inventarios, ventas y bodegas. Permite a los usuarios administrar productos, controlar el stock, gestionar planes de ventas y visualizar reportes en tiempo real. El sistema está construido con React (Next.js), utiliza Material UI para la interfaz y se comunica con microservicios backend mediante peticiones HTTP.

Logística también cuenta con un par de aplicaciones móviles construidas en React-Native que integran las funcionalidades para la fuerza móvil de ventas y los clientes de CCP respectivamente. En estas pueden realizar pedidos, realizar el seguimiento de estos, gestionar visitas técnicas y solicitar recomendaciones del estado y organización de una tienda

2. Arquitectura



Frontend cliente web:

- **Framework**: Next.js (React)
- **UI**: Material UI (MUI)
- **Internacionalización**: next-intl
- **Testing**: Jest, React Testing Library, Playwright, Cypress

Frontend app movil ventas y app movil clientes:

- **Framework**: React-native

- **UI**: Material UI (MUI)
- **Internacionalización**: next-intl
- **Testing**: Jest, Maestro

Backend:

- Microservicios en Python 3.10 utilizando el framework Flask.
- Comunicación vía API REST.

Estructura de carpetas principal:

3. Principales Módulos y Componentes

3.1 Inventario

Permite visualizar, registrar y actualizar productos en stock, así como asociarlos a bodegas.

3.2 Plan de ventas

Gestión de planes de ventas, asignación de metas y productos a vendedores, cierre de planes.

3.3 Productos

Administrado por el área de compras de los productos ingresados a la plataforma. Permite la carga masiva de productos por medio de archivos csv, json o tablas de Excel.

3.4 Proveedores

Administrado por el área de compras para tener control y registro de los proveedores de CCP.

3.5 Vendedores

Administrado por el equipo de ventas permite tener control en tiempo real de los vendedores de CCP.

3.6 Pedidos

Registro clave para la visualización de los pedidos generados por los clientes de CCP. Tabla actualizada en tiempo real que permite evidenciar el estado de los pedidos y la evolución de la operación del área logística.

4. Integración con Microservicios

Cada módulo del frontend cuenta con un archivo de adaptadores (adapters/microservice*.ts) que encapsula la lógica de comunicación con los microservicios backend. Estos adaptadores utilizan fetch para consumir los endpoints REST y devolver los datos al frontend.

Inventario

Obtener productos en stock
Endpoint: GET /api/stock
Función: getStock()
Descripción: Devuelve la lista de productos actualmente en inventario.

Obtener bodegas

Endpoint: GET /api/bodegas
Función: getBodegas()
Descripción: Devuelve la lista de bodegas disponibles.

Registrar producto en stock

Endpoint: POST /api/stock
Función: createStock(data)
Descripción: Registra un nuevo producto en el inventario.

Planes de Ventas

Obtener planes de venta
Endpoint: GET /api/vendedor/planes
Función: getSalesPlan()
Descripción: Devuelve la lista de planes de venta activos y finalizados.

Crear plan de venta

Endpoint: POST /api/vendedor/crear_plan
Función: createPlan(data)
Descripción: Crea un nuevo plan de ventas para un vendedor.

Cerrar plan de venta

Endpoint: POST /api/vendedor/finalizar_plan
Función: closePlan({ id })
Descripción: Marca un plan de ventas como finalizado.

Obtener vendedores

Endpoint: GET /api/vendedor
Función: getSellers()
Descripción: Devuelve la lista de vendedores registrados.

Bodegas

Obtener bodegas
Endpoint: GET /api/bodegas
Función: getBodegas()
Descripción: Devuelve la lista de bodegas.

Obtener productos por bodega

Endpoint: GET /api/bodegas/:id/productos
Función: getBodegaProducts(id)
Descripción: Devuelve los productos almacenados en una bodega específica.

Notas adicionales

Todos los endpoints pueden requerir autenticación (por ejemplo, mediante tokens JWT).

Los endpoints pueden variar según la configuración del entorno (apiURI).

Los adaptadores están diseñados para manejar errores y devolver mensajes claros al frontend.

5. Internacionalización

El sistema soporta múltiples idiomas usando next-intl.

Las traducciones se gestionan por carpetas [locale] y archivos de mensajes.

6. Pruebas

- Las pruebas unitarias y de integración se encuentran en la carpeta __tests__. Se utiliza Jest y React Testing Library para simular la interacción de usuario y mockear peticiones HTTP.
- Se realizaron varios ciclos de pruebas tipo Monkey test utilizando cypress.
- Se desarrollaron pruebas e2e automaticas para todos los componentes importantes de la aplicación web utilizando Cypress.

7. Seguridad y Validaciones

Los formularios validan campos obligatorios y tipos de datos (por ejemplo, solo números en campos como meta_venta).

El acceso a los microservicios puede requerir autenticación.

8. Ejecución y Desarrollo

Se recomienda el uso de node la versión 11.1.0.

Instalación de dependencias:

```
$ npm install
```

Ejecución en desarrollo:

```
$ npm run dev
```

Ejecución de pruebas:

```
$ npm run test
```

9. Consideraciones Finales

El proyecto está diseñado para ser modular y escalable. Por tal razón cuenta con todos los archivos para ser desplegado en un ambiente en la nube

Se recomienda mantener actualizadas las dependencias y revisar las pruebas al agregar nuevas funcionalidades.

La integración con microservicios debe manejar correctamente los errores y estados de carga.