**Proyecto de Microservicios – Grupo 3**

* ***Microservicios para desarrollar:***
  + Products Service - Gestión de productos
  + Inventory Service - Control de stock
  + Billing Service - Facturación
  + Authentication Service (Auth)
  + API Gateway
  + Reports Service (opcional)
  + Orders Servic (opcional)
* ***Flujo de servicios:***

API Gateway (entrada)

↓

Auth Service (base para todos)

↓

Products Service

↓

Inventory Service

↓

Orders Service (opcional)

↓

Billing Service

↓

Reports Service (opcional)

* **Consideraciones en las carpetas**

**📁 project-root/**

Contiene los archivos y carpetas principales del proyecto.

* **docker-compose.yml**  
  Define todos los servicios (frontend, gateway, microservicios, base de datos).  
  Orquesta la red interna, volúmenes, puertos y dependencias entre contenedores.
* **.env**  
  Archivo global de variables de entorno: credenciales, puertos, URIs, claves JWT, etc.  
  Cada servicio puede tener su propio .env interno, pero este centraliza lo global.

**📁 gateway/**

API Gateway (Nginx) — punto de entrada único a todos los servicios.

* **nginx.conf**  
  Archivo de configuración de Nginx: define las rutas proxy hacia los microservicios (por ejemplo /api/auth → auth-service).  
  También puede gestionar encabezados CORS y balanceo simple de carga.

**📁 frontend/**

Interfaz web unificada del sistema.

* **Dockerfile**  
  Contenedor que sirve la aplicación con Nginx o un servidor estático (http-server o similar).  
  Incluye el proceso de build si se usa un framework (React/Vue) o solo copia archivos HTML/CSS/JS.
* **index.html**  
  Página principal del frontend. Puede representar el **login** o **dashboard general**.
* **pages/**  
  Contiene las páginas adicionales del sistema:
  + login.html: pantalla de inicio de sesión.
  + dashboard.html: panel principal del usuario.
  + products.html: interfaz para gestionar productos.
* **assets/**  
  Archivos estáticos del frontend:
  + **css/**
    - styles.css: define el estilo visual global del sistema (colores, fuentes, layout).
  + **js/**
    - main.js: punto de entrada principal del frontend, controla la lógica global de la interfaz.
    - api.js: módulo centralizado que contiene las funciones para comunicarse con el Gateway (GET, POST, PUT, DELETE).
    - auth.js: maneja la autenticación de usuarios (login/logout, guardado de tokens).
    - products.js: lógica específica para CRUD de productos.
    - utils.js: utilidades compartidas, como formateo de datos, manejo de errores o validación.
  + **images/**
    - Recursos visuales como logotipos o íconos (logo.png, etc.).
* **config/**
  + env.example.js: ejemplo de configuración (por ejemplo, API\_BASE\_URL del gateway).  
    Se usa como plantilla para crear un env.js local de configuración.

**📁 services/**

Contiene todos los microservicios del sistema, cada uno con su propia lógica, base de datos, rutas y dependencias.

Cada servicio tiene la misma estructura base 👇

**⚙️ Estructura común de un servicio**

📦 **Ejemplo: auth-service/**  
(*Se aplica igual a products, inventory, orders, billing, reports.*)

**Dockerfile**

Define cómo se construye la imagen del servicio:

* Base Node.js.
* Copia el código fuente.
* Instala dependencias (npm install).
* Expone el puerto interno (por ejemplo 4001).
* Ejecuta el servidor (npm start).

**package.json**

Archivo de configuración del servicio:

* Nombre, versión, scripts (start, dev, etc.).
* Dependencias (Express, Mongoose, dotenv, axios, cors, etc.).
* Metadata básica del servicio.

**📁 src/**

Código fuente del servicio.  
Contiene toda la lógica funcional y estructura interna.

**index.js**

* Punto de entrada del servicio.
* Crea el servidor Express.
* Carga middlewares globales (CORS, JSON parsing).
* Importa las rutas desde /routes.
* Conecta a la base de datos.
* Inicia el servidor en el puerto indicado por el .env.

**📁 routes/**

* Define las rutas/endpoints HTTP del servicio.
* Cada archivo agrupa las rutas por tema (por ejemplo auth.routes.js, products.routes.js).
* Importa el controlador correspondiente y asigna funciones a cada ruta (GET, POST, PUT, DELETE).

**📁 controllers/**

* Contiene la lógica de negocio o procesamiento de cada endpoint.
* Los controladores ejecutan las operaciones principales: validaciones, acceso a base de datos, generación de respuestas.
* Son llamados desde las rutas.

**📁 models/**

* Define los esquemas de datos y modelos de la base de datos.
* En MongoDB, usa Mongoose (Schema y Model).
* Representa entidades como usuarios, productos, inventario, pedidos, facturas o reportes.

**📁 middleware/**

(*Solo algunos servicios como auth lo necesitan*)

* Define funciones intermedias que se ejecutan antes o después de las rutas.
* Ejemplo típico: verificar tokens JWT, permisos o validaciones de sesión.

**📁 config/**

* Configuración interna del servicio.
  + db.js: define la conexión a la base de datos (MongoDB u otra).
  + env.js: carga las variables de entorno específicas del servicio usando dotenv.  
    (Ej: PORT, MONGO\_URI, JWT\_SECRET, etc.)

**📁 utils/**

* Funciones auxiliares reutilizables entre controladores.
  + token.js: creación y validación de JWTs (Auth service).
  + httpClient.js: cliente HTTP (por ejemplo con axios) para comunicar un microservicio con otro a través del Gateway.
  + Otras utilidades comunes (formateo, logs, etc.)

**📁 volumes/**

* Carpeta montada como volumen de Docker.
* **mongodb\_data/** almacena los datos persistentes de MongoDB, para no perder información al reiniciar contenedores.