Desarrollo de una Plataforma E-Commerce Full Stack (Frontend + Backend + Base de Datos)

Lenguajes	HTML, CSS, JavaScript, Node.js.
Objetivo del Proyecto:	El objetivo del proyecto es diseñar y desarrollar una plataforma completa de comercio electrónico que permita a los usuarios visualizar productos tecnológicos en un catálogo dinámico, acceder a la ficha individual de cada producto, registrarse o iniciar sesión, y gestionar un carrito de compra simple. La plataforma deberá consumir datos reales desde una o varias API REST desarrollada por el propio equipo, e incluir operaciones básicas de gestión de productos (para administradores). Además, deberá cumplir con los principios de diseño responsive y usabilidad moderna, funcionando correctamente en escritorio, tablet y móvil.
Descripción:	**Frontend (cliente web):** Desarrollado con HTML, CSS y JavaScript. Incluye vistas públicas (catálogo, login, registro, carrito) y secciones privadas según rol de usuario (por ejemplo, panel de administración solamente para administradores). **Backend (API REST):** Desarrollado con Node.js + Express. Gestiona productos, usuarios y pedidos, así como autenticación y autorización básica, y responde a las peticiones del frontend. El proyecto contará con una entrega final que incluirá el código en ZIP, un documento explicativo y la presentación realizada para la defensa del proyecto (se detallarán los criterios de presentación más adelante).
Requisitos Funcionales:	**1. Registro e inicio de sesión** • Implementar un sistema de autenticación entre frontend y backend. • Alta de usuarios (registro) con al menos los campos username, password y tipo de usuario. Adicionalmente, se deberán crear los campos necesarios (nombre, dirección, teléfono, etc) para el envío del pedido final al cliente. • Validación de credenciales mediante una petición de login a una API de usuarios. • Gestión básica de sesión (con persistencia del usuario actual y almacenamiento en localStorage). • Redirección según el rol (user o admin) al catálogo o a la página de administración.

2. Roles de usuario

- Admin: Puede crear, editar y eliminar productos. Además, podrá gestionar pedidos realizados por los usuarios y confirmarlos.
- User: Puede consultar el catálogo, ver los detalles y gestionar su carrito así como sus pedidos.

3. Catálogo de productos

- Carga dinámica de productos desde la API de productos (por ejemplo GET /api/products).
- Vista individual de producto con:
  - Imagen
  - Nombre y categoría
  - Precio
  - Stock
  - Descripción breve y/o extendida
- Filtros por categoría seleccionada(s).
- Las imágenes de los productos deberán estar disponibles mediante una URL accesible desde el frontend. Pueden almacenarse localmente (por ejemplo, en una carpeta pública del proyecto) o en un servicio externo de almacenamiento como AWS S3 o Firebase Storage. No es obligatorio desplegar el proyecto en la nube, pero sí garantizar que las imágenes se muestren correctamente al ejecutar el servidor en local.

4. Carrito de compra

- Los usuarios pueden:
  - Añadir productos al carrito.
  - Ver el listado, cantidades y total acumulado.
  - Eliminar productos del carrito.
  - Gestionar sus pedidos ya realizados.
- El carrito se almacenará en una tabla pedidos, en la base de datos, con estados según sea "CART" (pedido actual temporal) o "ORDER" (pedido definitivo).

5. Gestión de productos (solo Admin)

Operaciones disponibles:

- Crear producto (POST /api/products)
- Actualizar producto (PUT /api/products/:id)
- Eliminar producto (DELETE /api/products/:id)

6. Interfaz de usuario

- Diseño claro, moderno y responsive.
- Compatible con dispositivos móviles, tabletas y escritorio.
- Navegación intuitiva entre secciones.
- Coherencia visual entre pantallas (login, catálogo, carrito, administración).

	7. API REST y consumo desde el frontend
	• Todas las peticiones se realizarán con fetch() y promesas / async-await.
	• La aplicación deberá comunicarse únicamente a través de la API para obtener o modificar datos (productos, usuarios, carritos, pedidos, etc.).
	• El frontend no gestionará datos locales salvo la sesión del usuario activo (almacenada en localStorage).
	• El backend será responsable de procesar, validar y persistir toda la información de la aplicación.
	• APIs principales:
	  ○ Productos: gestión del catálogo (consultas y CRUD).
	  ○ Usuarios: registro, autenticación y roles.
	  ○ Pedidos / Carritos: gestión de carritos activos y pedidos confirmados (manejo de estados)
	• El backend deberá:
	  ○ Manejar errores correctamente (404, 500, etc.).
	  ○ Usar middlewares para el control de acceso.
	  ○ Separar rutas, controladores y modelos.
	
	8. Documentación técnica
	Debe incluir:
	• Estructura de carpetas del proyecto.
	• Descripción de la arquitectura y flujo cliente-servidor.
	• Esquema de base de datos (tablas y campos).
	• Rutas de la API (métodos y endpoints).
	• Explicación de la autenticación.
	• Retos y soluciones encontrados durante el desarrollo.

Requisitos Técnicos:	**Frontend** • HTML semántico. • CSS con Flexbox y/o Grid. • Diseño responsive (@media queries). • Formularios de login/registro. • Generación dinámica del catálogo. • Uso de módulos JS (import/export). • Consumo de API REST con fetch(). **Backend** • Node.js + Express. • Rutas separadas por entidad (products, users, login). • CRUD de productos. • Autenticación de usuarios. • Middlewares para validación y control de acceso. • Gestión básica de errores. **Base de datos** Motor sugerido: SQLite (básico) o MySQL (avanzado).

Se podrá obtener una calificación extra de 1.5 puntos, por implementar adicionalmente a lo indicado anteriormente:

- Documentación Swagger UI de la API, explicado en clase.
- Proceso de hash seguro para las contraseñas de usuario utilizando la librería bcryptjs antes de almacenarse en la base de datos.
- Despliegue del servidor(es) en la nube (Render, Railway, Vercel, AWS, Google Cloud, Azure, etc). Este despliegue se deberá investigar por cuenta de cada grupo, si decidís implementarlo, elegid la forma más sencilla.

Entrega y Evaluación

Cada grupo entregará:

1. **Proyecto completo comprimido (.zip)**
    - Frontend + Backend + Base de datos.
2. **Documento PDF explicativo** (máx. 10 páginas):
    - Arquitectura técnica.
    - Estructura de carpetas.
    - Descripción de rutas API.
    - Explicación del frontend y sus funciones.
    - Esquema de base de datos.
    - Principales retos y Soluciones aportadas (al menos 2 que marquen cierta diferencia con respecto a lo que hemos estado desarrollando en clase).
3. **Presentación oral (15 minutos)**
    - Demostración del funcionamiento.
    - Explicación del código y estructura.
    - Roles y contribuciones del equipo.

Fecha de entrega y defensa:

El miércoles, día 7 de enero se hará una primera validación de requisitos en clase, debéis tener todo listo solamente a falta de correcciones finales y presentación/defensa.

El proyecto se presentará y defenderá el jueves, día 8 de enero. El orden de presentación se realizará por sorteo ese mismo día.

Este proyecto se debe aprobar con la nota mínima de 5 para poder aprobar y ponderar con las demás notas de la asignatura. Es también obligatorio para poder presentarse al examen de C. Ordinaria y C. Extraordinaria.

Rúbrica de evaluación:

(Siguiente página)

Criterio	Excelente	Adecuado	Necesita mejorar	Insuficiente	Peso
1. Estructura y arquitectura del proyecto	La estructura del proyecto está perfectamente organizada, separando frontend, backend y base de datos. Se usan carpetas y módulos de forma coherente, con código limpio y bien comentado. Arquitectura clara, funcional y documentada.	Estructura bien definida con ligeras inconsistencias o redundancias. Se aprecia organización y modularidad general.	Organización parcial o confusa: mezcla de responsabilidades o falta de claridad entre frontend y backend.	Estructura desordenada, sin separación lógica de componentes. Difícil de mantener o ejecutar.	10%
2. Frontend (HTML, CSS, JS, responsive)	HTML semántico impecable. CSS moderno y consistente (Flexbox y/o Grid). JS estructurado con funciones y módulos claros. Diseño completamente responsive, accesible y coherente con las heurísticas de Nielsen.	HTML y CSS correctos, con pequeños fallos de maquetación o consistencia. Diseño responsive con leves errores de escala o alineación.	HTML y CSS funcionales pero básicos. Diseño poco coherente o adaptabilidad parcial a móviles. Interactividad limitada con la API.	HTML desordenado, sin semántica ni diseño adaptativo. Sin conexión o interacción con la API.	20%
3. Backend y API REST (Node.js + Express)	API REST completa, modular y estable. Implementa correctamente rutas, controladores y middlewares. Gestiona errores (404, 500) con respuestas JSON coherentes. CRUD funcional para productos, usuarios y pedidos.	API funcional y estable, con estructura correcta. Pequeños fallos en validación o manejo de errores. Endpoints principales implementados.	API básica o con endpoints incompletos. CRUD parcial o sin validación.	API no funcional o ausente. El frontend no obtiene datos desde la API.	25%
4. Base de datos (SQLite / MySQL)	Base de datos correctamente diseñada, con tablas y relaciones claras entre productos, usuarios y pedidos. Tipos de datos adecuados y persistencia verificada. Consultas y conexiones eficientes.	Estructura adecuada con pequeños errores en tipos o relaciones. Persistencia funcional.	Base de datos mínima o con errores en la conexión. No refleja correctamente las entidades principales.	No se conecta correctamente la base de datos o no hay persistencia real.	10%
5. Autenticación y roles de usuario	Sistema de autenticación totalmente funcional y conectado a la API. Registro, login y control de acceso por roles (admin/user) bien implementados. Gestión de sesión persistente en localStorage y redirección según rol.	Login y registro operativos con ligeras deficiencias (validaciones o persistencia). Roles básicos aplicados parcialmente.	Autenticación funcional pero incompleta. Falta diferenciación de roles o persistencia.	Sin autenticación o control de acceso. Fallos graves de validación o credenciales.	10%
6. Carrito de compra y lógica de cliente	Carrito completamente funcional e integrado con la API. Permite añadir, eliminar y actualizar productos, mostrando cantidades y total acumulado. Sincroniza correctamente estados “CART” y “ORDER”.	Carrito operativo con algunas limitaciones (problemas al actualizar o persistir). Cálculo total correcto.	Carrito parcial o básico. Solo permite añadir o listar productos. No gestiona estados ni pedidos.	Carrito no implementado o no funcional.	15%
7. Documentación técnica y claridad del código	Documento PDF claro y completo. Explica arquitectura, rutas de la API, esquema de base de datos y flujo cliente-servidor. Código bien comentado, legible y con convenciones de nombres uniformes.	Documentación comprensible con ligeras omisiones o errores de formato. Código en general claro.	Documentación superficial o incompleta. Faltan explicaciones clave sobre la API o la estructura.	Sin documentación o con errores graves. Código difícil de leer o sin comentarios.	10%