MF0492_3 · PROGRAMACIÓN WEB EN EL ENTORNO SERVIDOR

Guía para el proyecto full stack (inspirado en Bazar)

Vas a diseñar y planificar una aplicación web full stack que incluya:

- Una base de datos relacional (MySQL o MariaDB).
- Un backend en Node.js + Express (u otro entorno servidor indicado en clase).
- Un frontend (HTML + CSS + JavaScript, o framework autorizado por el docente) que consuma la API del backend.

No es obligatorio que el proyecto sea una tienda online como Bazar: puedes elegir otro contexto (gestor de eventos, biblioteca, reservas, tareas, agenda, etc.), pero debe cumplir los requisitos mínimos de una aplicación full stack.

✅ Requisitos mínimos

1. Base de datos

- Al menos 2 tablas relacionadas
  Ejemplos:
  - `usuarios` – `tareas`
  - `clientes` – `pedidos`
  - `usuarios` – `reservas`
- Clave primaria (PK) en cada tabla.

- Al menos una relación 1:N o N:1 (por ejemplo, un usuario tiene muchas tareas, un cliente tiene muchos pedidos).

- Opcional: relaciones N:M usando tabla intermedia.

2. Backend (entorno servidor)

- Servidor desarrollado con Node.js + Express (o la tecnología acordada en clase).

- Debe exponer una API REST básica, que permita al menos:

    - Listar registros de la tabla principal.

    - Crear registros (ej. nueva tarea, nueva reserva…).

3. Frontend

- HTML + CSS + JavaScript (o framework autorizado).

- Debe consumir la API del backend mediante fetch u otra librería.

- Debe mostrar, como mínimo:

    - Un listado de la tabla principal (tareas, libros, eventos, etc.).

    - Algún tipo de formulario para crear o gestionar esos elementos o para un login de cliente.

---

1. PLANTEAMIENTO INICIAL DEL PROYECTO

Esta primera parte es solo de diseño y planificación, sin código. Te ayudará a tener claro qué vas a hacer antes de programar.

1.1. Nombre del proyecto

Elige un nombre corto, identificable y relacionado con el objetivo del proyecto.

1.2. Objetivo del proyecto

Responde: **¿Cuál es la finalidad de tu proyecto?**

Ejemplos:

- Dar de alta tareas y marcarlas como hechas.
- Compra de productos.
- Biblioteca de música.
- Gestionar reservas de habitaciones.
- Gestionar inscripciones a eventos.

Redacta 2–3 frases explicando qué problema resuelve tu aplicación y qué hace el usuario principal en ella.

1.3. ¿Hacia quién va dirigido?

Indica si desarrollas la aplicación para:

- Un cliente final (usuario normal: comprador, lector, alumno, asistente a eventos…).
- Un administrador (gestor de contenidos, dueño de la tienda, personal del hotel…).
- Ambos (cliente y admin), aunque en esta primera versión conviene centrarse en uno.

Indica el tipo de usuario principal que usará tu app y por qué.

1.4. ¿Qué va a poder hacer tu cliente? (funcionalidades básicas)

Marca 2 o 3 objetivos sencillos. Si quieres, después puedes añadir una sección de “mejoras futuras”.

Ejemplos de objetivos mínimos:

- Dar de alta un producto y mostrarlo en una página, y permitir que se pueda reservar a través de un formulario.
- Crear una tarea, verla en una lista y marcarla como completada.
- Dar de alta un evento, verlo en la web y permitir que un usuario se apunte.
- Mostrar un catálogo de canciones y permitir a un usuario registrado dar “me gusta”.

Escribe una lista de 2–3 acciones que tu usuario podrá hacer de principio a fin.

1.5. Flujo de negocio (circuito sencillo)

Describe el recorrido típico del usuario en 2–4 pasos.

Ejemplos:

- Al entrar en mi página, cualquier usuario ve una biblioteca de canciones; un usuario registrado va a poder dar a “me gusta”.

- Un usuario verá un catálogo de productos, pero solo un usuario registrado podrá añadir productos al carrito.

- Cualquier visitante ve una lista de eventos; un usuario registrado puede inscribirse a uno de ellos.

Escribe tu flujo de negocio en forma de pasos: Paso 1, Paso 2, Paso 3.

1.6. Aspecto de la web (diseño básico)

No hace falta que lo maquetes aún, pero sí que tengas una idea de distribución y estilo:

- Número de contenedores / zonas principales:
    - Cabecera (logo + menú).
    - Zona principal (listado, tarjetas, formularios).
    - Barra lateral o footer (opcional).
- ¿Será de una sola página con secciones o varias vistas?
- ¿Tendrás vista de lista y vista de detalle?

Paleta de colores:

- ¿Fondo claro u oscuro?
- 1 o 2 colores principales para botones y elementos destacados.

Describe brevemente cómo será tu página: qué se ve arriba, qué se ve en el centro, qué se ve abajo, y qué colores vas a usar.

2. PLANTEAMIENTO DE LA BASE DE DATOS

La base de datos es la columna vertebral del proyecto: de aquí saldrán los datos que luego consultarás desde el backend y mostrarás en el frontend. Te recomiendo no usar más de 3 tablas en esta primera versión.

2.1. Elegir las entidades principales (tablas)

Piensa primero en qué cosas quieres guardar en tu aplicación. Cada "tipo de cosa" será normalmente una tabla.

- Si es una biblioteca → seguramente tendrás:
    - libros
    - usuarios (lectores)
- Si es una web de música → podrías tener:
    - canciones
    - álbum o playlist
    - usuarios
- Si es un hotel → podrías tener:
    - habitaciones
    - reservas
    - clientes

★ ¿Cuáles son tus tablas? Escríbelas y define el objetivo de cada una.

2.2. Tabla principal

Tu proyecto debe tener una tabla principal, que será la “estrella” de tu aplicación:
lo que el usuario viene a ver, gestionar o reservar.
Algunos ejemplos:

- Biblioteca → tabla principal: libros
- Gestor de tareas → tabla principal: tareas
- Web de música → tabla principal: canciones
- Hotel → tabla principal: habitaciones
- Gestor de eventos → tabla principal: eventos

★ ¿Cuál es tu tabla principal y cómo se relacionan las demás con ella?

★ Define los atributos (columnas) de tu tabla.

Recuerda que debes tener una PK y definir la FK en las tablas relacionadas. Si la relación entre dos tablas es n:m tendrás que crear una tabla intermedia.

2.2 Segunda tabla: usuarios, clientes o similar (no obligatorio)

★ ¿Vas a crear una tabla de usuarios?

★ ¿El usuario se podrá registrar?

★ ¿Qué privilegios tendrá el cliente registrado?

3. PLANTEAMIENTO DE MODELO Y CONTROLADORES. ENDPOINT.

Conforme a lo que hemos planteado piensa que peticiones vamos a necesitar. No es necesario que escribas el código si no que plantees las rutas que necesitas. Por ejemplo: necesito una ruta que muestre todas las obras, necesito una ruta que permita dar de alta eventos… Recuerda que los filtros por ejemplo categorías o autores los podemos aplicar en frontend si hemos recibido todos los datos.

3.1. Modelo (Model)

Por cada tabla importante, tendrás normalmente un **modelo** que se encargará de hablar con la base de datos.

Por ejemplo, si tu tabla principal es libros:

- libros.model.js podría incluir funciones como:
    - findAll() → devuelve todos los libros.
    - findById(id) → devuelve un libro por id.
    - create(datosLibro) → inserta un libro nuevo.

    - update(id, datosLibro) → actualiza un libro.
    - remove(id) → borra un libro.

★ **Haz una lista de las funciones que necesitarán tus modelos para la tabla principal (y, si aplica, para la tabla de usuarios).**

3.2. Controladores (Controller) y endpoints

Los controladores reciben la petición HTTP, llaman al modelo y devuelven una respuesta.

Piensa qué rutas necesitas. No hace falta que las programes, solo que las definas.

Ejemplos:

- Para una tabla principal obras:
    - GET /api/obras → mostrar todas las obras.
    - GET /api/obras/:id → mostrar detalle de una obra.
    - POST /api/obras → dar de alta una nueva obra.
    - PUT /api/obras/:id → editar una obra.
    - DELETE /api/obras/:id → borrar una obra.
- Para gestionar eventos:
    - GET /api/eventos → listado de eventos.
    - POST /api/eventos → alta de nuevo evento.
    - (Opcional) POST /api/eventos/:id/inscripciones → inscribir a un usuario.

Recuerda que muchos filtros (por categoría, autor, tipo…) se pueden aplicar en frontend si has recibido todos los datos, aunque también es posible aplicarlos en la consulta SQL.

- ★ Establece tus rutas necesarias y los métodos del controlador.

5. Planificación y cronograma de proyecto.

¿Conoces Trello? Trello

Trello es una herramienta visual donde organizamos tareas en columnas (listas) y dentro añadimos tarjetas (tasks).
Nos ayuda a dividir el proyecto en etapas claras y manejables, igual que hacen los equipos profesionales.

Aquí tienes un ejemplo de cinco posibles sprints. Un sprint es una etapa corta de trabajo con objetivos concretos y resultados claros.

SPRINT 1 — Diseño y planteamiento inicial
SPRINT 2 — Diseño de base de datos
SPRINT 3 — Backend: API y lógica de negocio
SPRINT 4 — Frontend y consumo de la API
SPRINT 5 — Documentación y README.
SPRINT 6 — Pruebas, mejoras y presentación final

5. Iniciamos proyecto.

- ★ *Crea la carpeta backend*

mkdir backend

cd backend

- ★ *Instala las librerías mínimas para comenzar.*

npm init -y (inicaliza proyecto node)

npm install express mysql2 dotenv cors

npm install --save-dev nodemon

★ Recuerda añadir "type": "module" para usar import en lugar de require.

Los 3 scripts más típicos para un backend moderno:

```
"scripts": {
"start": "node server.js",
 "dev": "nodemon server.js",
  "doc": "jsdoc -c jsdoc.json"
}
```

6. Documentación.

Aunque no es obligatorio codificar documentando si es lo más recomendable. Acostumbrate a documentar cuando terminamos un archivo así tendrás fresca la funcionalidades y podrás documentar correctamente.
Jsdoc.
¿Cómo lo instalamos? Ejecuta npm install jsdoc --save-dev en la carpeta de backend o donde tengas package json.

3. (Opcional pero recomendado) Crear un script en package.json

Dentro de `"scripts"` añade:

json Copiar código

```
"docs": "jsdoc ./ -d docs"
```

Así puedes generar la documentación ejecutando:

bash Copiar código

```
npm run docs
```

¿Cómo se genera documentación?

```js

/**
 * Comentario JSDoc aquí
 */
```

2. Comentario básico para una función

```js
/**
 * Suma dos números.
 * @param {number} a - Primer número.
 * @param {number} b - Segundo número.
 * @returns {number} Resultado de la suma.
 */
function sumar(a, b) {
  return a + b;
}
```

3. Documentar un endpoint Express

```js
/**
 * GET /api/productos
 * Obtiene la lista de productos.
 * @route GET /productos
 * @returns {Array<Object>} Lista de productos.
 */
router.get("/productos", productosController.getTodos);
```

4. Documentar un controlador

```js

/**
 * Controlador: obtiene todos los productos.
 * @async
 * @function getTodos
 * @param {Request} req - Objeto de petición Express.
 * @param {Response} res - Objeto de respuesta Express.
 */
export async function getTodos(req, res) {
  const productos = await productosModel.getAll();
  res.json({ data: productos });
}
```

7. Iniciamos proyecto.

Una vez que has planteado tu documentación y la estructura general del proyecto, ya puedes empezar a crear tus primeros archivos.
Este es un orden lógico y recomendado para construir cualquier backend en Node.js + Express:

Un orden lógico podría ser .env
db.js (creamos pool)

Creamos server.js
Creamos y lanzamos initdb.js Recuerda crear tu bbdd en phpmyadmin. En el propio archivo initdb.js puedes incluir datos de ejemplo y si no haz otro archivo y llamalo seeds.js para tus datos de ejemplo.