

1. ¿Cuáles son las partes de la arquitectura de la aplicación? (con capturas)

La arquitectura de la aplicación se basa en el patrón MVC (Modelo-Vista-Controlador) y está estructurada de la siguiente forma:

- **Modelo (Model):** Los modelos son responsables de interactuar con la base de datos y manejar los datos de la aplicación.

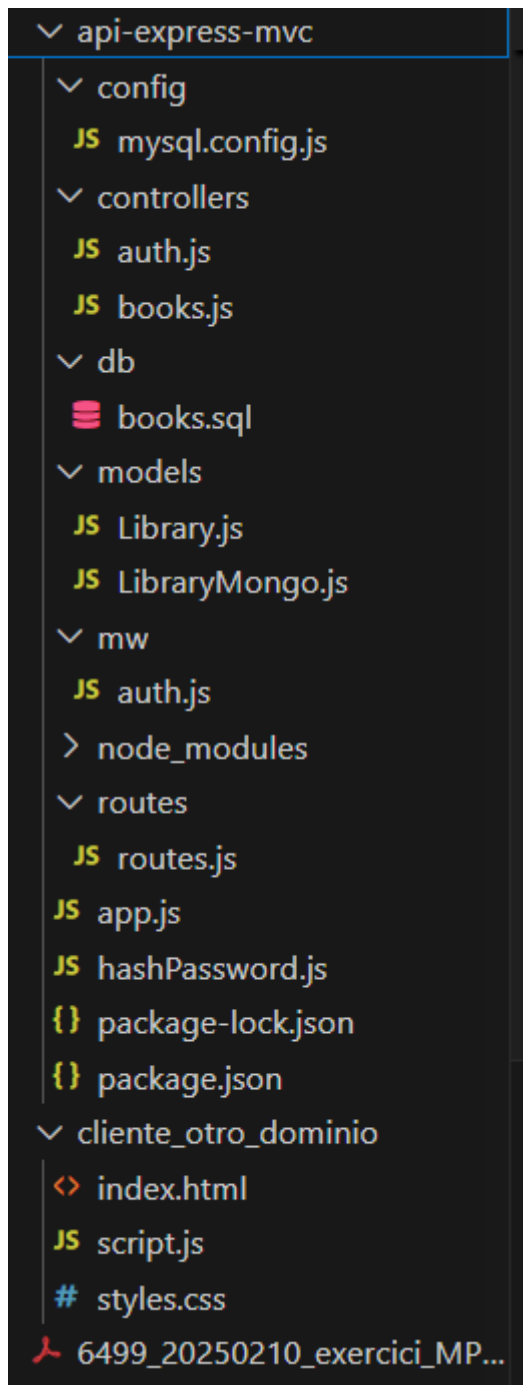
Utilizamos los modelos **Usuario.php** y **Anuncio.php**. Aquí es donde se define la lógica de negocio y la estructura de los datos.

- **Vista (View):** Las vistas están basadas en Twig, un motor de plantillas que permite separar la lógica de presentación del backend. Se incluyen archivos como layout.twig, dashboard.twig, home.twig, crear_anuncio.twig, entre otros. Las vistas se encargan de mostrar los datos a los usuarios de manera dinámica.

- **Controlador (Controller):** Los controladores como **AnuncioController.php** y **AuthController.php** manejan la lógica de la aplicación. Reciben las solicitudes del usuario, interactúan con los modelos para obtener o modificar los datos y luego devuelven la respuesta adecuada a las vistas.

- **Base de Datos:** Inicialmente, la aplicación utilizaba MySQL para almacenar los anuncios y los usuarios. Luego, se adaptó a MongoDB, lo que implicó cambios en la estructura y acceso a la base de datos, ya que MongoDB es una base de datos NoSQL y no usa esquemas rígidos como MySQL.

Captura:



2. ¿Cómo se ha hecho la adaptación del modelo de la biblioteca de MySQL a MongoDB?

Para adaptar el modelo de MySQL a MongoDB, fue necesario cambiar la forma en que se estructuran los datos y cómo se accede a ellos. Aquí están los pasos principales que se siguieron:

Cambio de Motor de Base de Datos: Se pasó de MySQL a MongoDB. MongoDB es una base de datos NoSQL que almacena datos en formato BSON (similar a JSON), lo que permite mayor flexibilidad en cuanto a las relaciones entre los datos.

Modelos Adaptados: En lugar de usar tablas, se definieron colecciones en MongoDB. Los modelos como `Anuncio.php` se adaptaron para trabajar con documentos en lugar de registros.

Manejo de Conexión: Se modificaron los controladores y el modelo para que utilizaran una conexión con MongoDB en lugar de MySQL. Para esto, se configuró el cliente MongoDB en el archivo `config.php`.

Consultas MongoDB: Las consultas SQL fueron reemplazadas por consultas específicas de MongoDB usando su sintaxis, como `.find()`, `.insertOne()`, `.updateOne()`, y `.deleteOne()`.

Captura:

Interactuar con la DB en MYSQL en Library.js:

Interactuar con la DB en MONGODB en LibraryMongo.js:

3. Capturas de la funcionalidad completa usando MongoDB (listado, adición, modificación y eliminación de libros en la base de datos).

Una vez adaptado el modelo y la base de datos a MongoDB, se implementaron las siguientes funcionalidades:

Listado de Libros (Anuncios): Los anuncios ahora se listan desde MongoDB usando la consulta `.find()`.

Adición de Libros (Anuncios): Se puede agregar un anuncio a la base de datos utilizando `.insertOne()` o `.insertMany()`.

Modificación de Libros (Anuncios): Los anuncios se pueden actualizar usando `.updateOne()` para cambiar los campos como el título, descripción, precio, etc.

Eliminación de Libros (Anuncios): Los anuncios se eliminan usando `.deleteOne()` o `.deleteMany()` dependiendo del caso.

Captura:

Iniciar conexión con la DB en MYSQL en Library.js

```
1  const mysql = require("mysql2");
2  const dbConfig = require("../config/mysql.config.js");
3
4  class Library {
5  constructor() {
6      // 1. Declaramos la conexión
7      this.connection = mysql.createPool({
8          host: dbConfig.HOST,
9          user: dbConfig.USER,
10         password: dbConfig.PASSWORD,
11         database: dbConfig.DB,
12         waitForConnections: true,
13         connectionLimit: 10,
14         queueLimit: 0
15     }).promise();
16 }
```

Iniciar conexión con la DB en MONGODB en LibraryMongo.js

```
const { MongoClient, ObjectId } = require("mongodb");

class Library {
  constructor() {
    this.mongoUrl = "mongodb://localhost:27017"; // URL de conexión
    this.dbName = "library"; // Nombre de la base de datos
    this.client = new MongoClient(this.mongoUrl);
    this.database = null;
  }

  async connect() {
    if (!this.database) {
      await this.client.connect();
      this.database = this.client.db(this.dbName); // Conectar a la base de datos
      console.log(`Conectado a la base de datos: ${this.dbName}`);
    }
  }

  async close() {
    await this.client.close();
    this.database = null;
    console.log("Conexión cerrada");
  }
}
```

Interactuar con la DB en MYSQL en Library.js:

```
// Crear un nuevo libro
create = async (newBook) => {
  try {
    const [result] = await this.connection.query("INSERT INTO books SET ?", newBook);
    return result.insertId; // Devolver el ID del libro insertado
  } catch (error) {
    console.error("Error al crear libro:", error);
    throw error;
  }
};

// Actualizar un libro existente
update = async (updatedBook) => {
  try {
    const { id, title, author, year } = updatedBook;
    const [result] = await this.connection.query(
      "UPDATE books SET title = ?, author = ?, year = ? WHERE id = ?",
      [title, author, year, id]
    );
    return result.affectedRows > 0;
  } catch (error) {
    console.error("Error al actualizar libro:", error);
    throw error;
  }
};
```

Interactuar con la DB en MONGODB en LibraryMongo.js:

```
async create(newBook) {
  await this.connect();
  const result = await this.database.collection("books").insertOne(newBook);
  return result.insertedId;
}

async update(updatedBook) {
  await this.connect();
  const result = await this.database.collection("books").updateOne(
    { _id: new ObjectId(updatedBook.id) },
    { $set: { title: updatedBook.title, author: updatedBook.author, year: updatedBook.year } }
  );
  return result.modifiedCount > 0;
}

async delete(id) {
  await this.connect();
  const result = await this.database.collection("books").deleteOne({ _id: new ObjectId(id) });
  return result.deletedCount > 0;
}
```

4. Explicación de los cambios que se han tenido que hacer (backend y frontend) para implementar la autenticación JWT.

La implementación de la autenticación JWT (JSON Web Token) requirió una serie de cambios tanto en el backend como en el frontend:

Backend:

- Instalación de la librería jsonwebtoken para manejar los tokens JWT.
- Al iniciar sesión, se genera un token con `jsonwebtoken.sign(payload, secret, options)` y se envía al cliente.
- El token se utiliza en las solicitudes subsiguientes para acceder a recursos protegidos, siendo verificado con `jsonwebtoken.verify(token, secret)`.
- Se modificaron los controladores de autenticación (por ejemplo, `AuthController.php`) para crear un JWT cuando el usuario inicia sesión correctamente. Este token se genera a partir de la información del usuario y se firma con una clave secreta.
- Los endpoints que requieren autenticación fueron protegidos. Se verificó el token JWT en cada solicitud para asegurar que el usuario esté autenticado antes de acceder a ciertos recursos (como crear o editar anuncios).

Frontend:

- En el frontend, el token se guarda en el almacenamiento local (`LocalStorage`) usando `Authorization: Bearer <token>` y se incluye en las cabeceras de las solicitudes mediante el encabezado `Authorization`.
- Se modificaron las peticiones AJAX para incluir el token JWT en el encabezado `Authorization` de cada solicitud.

Captura:

Generación y verificación del token JWT

```
1  const jwt = require('jsonwebtoken');
2
3  const SECRET_KEY = "tu_clave_secreta";
4
5  const verifyToken = (req, res, next) => {
6    const token = req.headers['authorization'];
7
8    if (!token) {
9      return res.status(403).json({ message: "Token requerido" });
10   }
11
12   try {
13     const decoded = jwt.verify(token.split(" ")[1], SECRET_KEY);
14     req.userId = decoded.id;
15     next();
16   } catch (err) {
17     return res.status(401).json({ message: "Token inválido" });
18   }
19 };
20
21 const generateToken = (userId) => {
22   return jwt.sign({ id: userId }, SECRET_KEY, { expiresIn: "2h" });
23 };
24
25 module.exports = { verifyToken, generateToken };
26
```