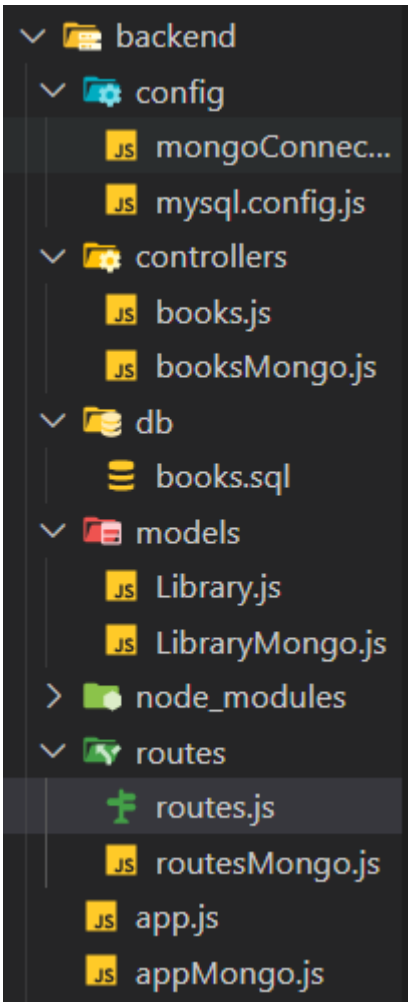
Bernat Brucet

Abans de fer servir JWT

Estructura de l'aplicació:

Backend:



- **config:** Conexio amb la bd (SQL i Mongo)
  - **mongoConnection.js:** Conexió bd Mongo
  - **mysql.config.js:** Conexió al bd SQL
- **controllers**: Controladors de les rutes api
  - **books.js:** Controlador de llibres
  - **booksMongo.js:** Controlador específic per MongoDB
- **db:** Genera la bd en SQL
  - **books.sql:** Genera la bd en SQL en cas de que no existeixi
- **models:** models de dades
  - **Library.js:** Model de MySQL
  - **LibraryMongo.js:** Model de MongoDB
- **routes:** Definició de rutes api
  - **routes.js:** Rutes de l'API TEST
- **app.js:** Servidor principal del servidor express
- **appMongo.js:** Servidor principal del servidor de mongo

ADAPTACIÓ DEL MODEL DE MySQL i MongoDB

MySQL:

```
const mysql = require("mysql2");
const dbConfig = require("../config/mysql.config.js");

class Library {
  constructor() {
    // En el constructor, creamos una conexión a la base de datos
    // y la guardamos en la propiedad connection de la clase

    // 1.Declaramos la conexión
    let connection = mysql.createConnection({
      host: dbConfig.HOST,
      user: dbConfig.USER,
      password: dbConfig.PASSWORD,
      database: dbConfig.DB
    });

    // 2.Abrimos la conexión
    connection.connect(error => {
      if (error) throw error;
      console.log("Successfully connected to the database.");
    });

    // 3.Dejamos la conexión en la propiedad connection, promisificada
    // (para poder utilizarlas más cómodamente en el resto de métodos de la
clase)
    this.connection = connection.promise();
  }

  close = () => {
    this.connection.end();
  }
  // métodos de la clase Library
  listAll = async () => {
    console.log(this.connection)
    const [results, fields] = await this.connection.query("SELECT * FROM
books");
    return results;
  }
```

```
  create = async (newBook) => {
    try {
        const [results] = await this.connection.query(
            "INSERT INTO books (title, author, year) VALUES (?, ?, ?)",
            [newBook.title, newBook.author, newBook.year]
        );
        return results.affectedRows; // Verifica si las filas fueron afectadas
    } catch (error) {
        console.error("Error inserting book:", error);
        throw new Error("Database insert failed"); // Lanzamos el error para
que se capture en el controlador
    }
  };
  update = async (updBook) => {
    const {id, title, author, year} = updBook;
    try {
        const [results] = await this.connection.query(
            "UPDATE books SET title = ?, author = ?, year = ? WHERE id = ?",
            [title, author, year, id]
        );
        return results.affectedRows > 0; // Devuelve `true` si se actualizó
algo, `false` si no
    } catch (error) {
        console.error("Error updating book:", error);
        throw new Error("Database update failed"); // Lanza el error en lugar
de devolverlo
    }
  };

  delete = async (delBook) => {
    try {
        const [results] = await this.connection.query(
            "DELETE FROM books WHERE id = ?",
            [delBook.id]  // Pasamos el ID correctamente
        );
        return results.affectedRows > 0; // Devuelve `true` si se eliminó algo
    } catch (error) {
        console.error("Error deleting book:", error);
        return false;
    }
  }
}
module.exports = Library;
```

MongoDB:

```
const mongoose = require('mongoose');

// Definir el esquema para el libro
const bookSchema = new mongoose.Schema({
    title: { type: String, required: true },
    author: { type: String, required: true },
    year: { type: Number, required: true }
});

// Crear el modelo de Book
const Book = mongoose.model('Book', bookSchema);

module.exports = Book;
```

Principals diferències:

- MySQL requereix definir els tipus de dades i la clau primària manualment.
- MongoDB usa un esquema flexible i genera automàticament un _id com a identificador.

Funcionalitat de MongoDB:

```
window.onload = () => {
    // Pedimos a la API los libros actuales en base de datos
    fetchBooks();

    // Añadimos al botón de submit del formulario un listener para enlazarlo a
la función createBook
    document.querySelector('#createButton').addEventListener('click',
createBook);

    document.querySelector('#downloadButton').addEventListener('click',
downloadVideo);
}
```

```
async function fetchBooks() {
    // Cambiar URL según si es MySQL o MongoDB
    let apiUrl = "http://localhost:5001/api/books";  // MongoDB

    let res = await fetch(apiUrl);
    let books = await res.json();
    // console.log(books);

    //Borramos el contenido de la tabla
    eraseTable();
    // Poblamos la tabla con el contenido del JSON
    updateTable(books);
}

function eraseTable() {
    // Accedemos a la lista de filas de la tabla <tr> y las borramos todas
    let filas = Array.from(document.querySelectorAll('tbody tr'));
    for (let fila of filas) {
        fila.remove();
    }
}

function updateTable(books) {
    let table = document.getElementById("book-table");

    // Iteramos books: por cada book
    for (let book of books) {
        // Creamos y añadimos a la tabla una nueva fila (<tr>)
        let row = document.createElement('tr');
        table.append(row);
        // Creamos y añadimos a la fila las celdas de id, título, autor, año,
acciones.
        // Las celdas id, título, autor, año se deben rellenar con la info del
JSON.
        // Las celdas título, autor, año deben tener el atributo
contenteditable a true.
        let celdaId = document.createElement('td');
        celdaId.innerHTML = book._id;  // MongoDB usa _id en lugar de id
        row.append(celdaId);
        let celdaTitulo = document.createElement('td');
        celdaTitulo.innerHTML = book.title;
        celdaTitulo.contentEditable = true;
        row.append(celdaTitulo);
```

```
        let celdaAutor = document.createElement('td');
        celdaAutor.innerHTML = book.author;
        celdaAutor.contentEditable = true;
        row.append(celdaAutor);
        let celdaAno = document.createElement('td');
        celdaAno.innerHTML = book.year;
        celdaAno.contentEditable = true;
        row.append(celdaAno);
        // Creamos dos botones (editar y eliminar) y los añadimos a la celda
acciones.
        // Hay que añadir a cada botón el listener correspondiente para
enlazarlos a las funciones editBook i deleteBook, respectivamente.
        let celdaAcciones = document.createElement('td');
        row.append(celdaAcciones);
        let buttonEdit = document.createElement('button');
        buttonEdit.innerHTML = "Modificar";
        buttonEdit.addEventListener('click', editBook);
        celdaAcciones.append(buttonEdit);
        let buttonDelete = document.createElement('button');
        buttonDelete.innerHTML = "Eliminar";
        buttonDelete.addEventListener('click', deleteBook);
        celdaAcciones.append(buttonDelete);
    }
}

async function deleteBook(event) {
    // Leemos el contenido de la columna id de esa fila
    let celdas = event.target.parentElement.parentElement.children;
    let id = celdas[0].innerHTML;
    // Hacemos la petición de DELETE a la API pasando un json en el cuerpo del
mensaje
    let apiUrl = "http://localhost:5001/api/books";  // MongoDB URL

    let deletedBook = {
        "_id": id  // MongoDB usa _id en lugar de id
    }

    let response = await fetch(apiUrl, {
        method: "DELETE",
        headers: {
            "Content-Type": "application/json",
        },
        body: JSON.stringify(deletedBook)
```

```
    });
    let json = await response.json()
    // Muestra respuesta de la API (JSON) por consola
    console.log(json);

    // Volvemos a pedir libros
    fetchBooks();
}

async function editBook(event) {
    // Leemos el contenido de las columnas id, título, autor, año de esa fila
    let celdas = event.target.parentElement.parentElement.children;
    let id = celdas[0].innerHTML;
    let titulo = celdas[1].innerHTML;
    let autor = celdas[2].innerHTML;
    let ano = celdas[3].innerHTML;

    // Hacemos la petición de PUT correspondiente pasando un json en el cuerpo
del mensaje
    // p.ej. { "id": 1, "title": "titulo", "author": "autor", "year": 1980 }
    let apiUrl = "http://localhost:5001/api/books";  // MongoDB URL

    let modifiedBook = {
        "_id": id,  // MongoDB usa _id en lugar de id
        "title": titulo,
        "author": autor,
        "year": ano
    }
    let response = await fetch(apiUrl, {
        method: "PUT",
        headers: {
            "Content-Type": "application/json",
        },
        body: JSON.stringify(modifiedBook)
    });
    let json = await response.json()
    // Muestra respuesta de la API (JSON) por consola
    console.log(json);

    //Volvemos a pedir libros
    fetchBooks();
}
```

```
async function createBook(event) {
    event.preventDefault(); // Previene la recarga de la página si se ejecuta
en un formulario

    // Leemos el contenido del formulario: título, autor, año
    let titulo = document.querySelector("#book-title").value.trim();
    let autor = document.querySelector("#book-author").value.trim();
    let ano = document.querySelector("#book-year").value.trim();

    // Validar que los campos no estén vacíos
    if (!titulo || !autor || !ano) {
        console.error("Todos los campos son obligatorios.");
        alert("Por favor, completa todos los campos.");
        return;
    }

    // Validar que el año sea un número válido
    if (isNaN(ano) || ano < 0) {
        console.error("El año debe ser un número válido.");
        alert("El año debe ser un número válido.");
        return;
    }

    // Hacemos la petición de POST
    let apiUrl = "http://localhost:5001/api/books";  // MongoDB URL

    let newBook = {
        title: titulo,
        author: autor,
        year: parseInt(ano) // Convertimos a número
    };

    try {
        let response = await fetch(apiUrl, {
            method: "POST",
            headers: {
                "Content-Type": "application/json",
            },
            body: JSON.stringify(newBook)
        });

        if (!response.ok) {
```

```
            throw new Error(`Error en la petición: ${response.status}
${response.statusText}`);
        }

        let json = await response.json();
        console.log("Respuesta del servidor:", json);

        // Volvemos a pedir libros para actualizar la lista
        fetchBooks();

        // Limpiar los campos del formulario
        document.querySelector("#book-title").value = "";
        document.querySelector("#book-author").value = "";
        document.querySelector("#book-year").value = "";
    } catch (error) {
        console.error("Error al crear el libro:", error.message);
        alert("Hubo un problema al crear el libro. Revisa la consola para más
detalles.");
    }
}

function downloadVideo() {
    console.log('Donwloading video...');
    // 1. Create a new XMLHttpRequest object
    let xhr = new XMLHttpRequest();

    // 2. Configure it: GET-request for the URL /article/.../load
    xhr.open('GET', './vid.mp4');

    // 3. Set the responseType to 'blob' to handle binary data
    xhr.responseType = 'blob';

    // 4. Send the request over the network
    xhr.send();

    // 5. This will be called after the response is received
    xhr.onload = function () {
        if (xhr.status != 200) { // analyze HTTP status of the response
            console.log(`Error ${xhr.status}: ${xhr.statusText}`); // e.g.
404: Not Found
        } else { // show the result
            console.log(`Done downloading video!`); // response is the server
response
```

```
            // CREATE A TEMPORARY DOWNLOAD LINK
            // Create a blob URL for the video
            console.log(`Creating download link!`);
            const blob = new Blob([xhr.response], { type: 'video/mp4' });
            const url = URL.createObjectURL(blob);

            // Create a temporary download link
            const a = document.createElement('a');
            a.href = url;
            a.download = 'downloaded_video.mp4'; // Suggested file name
            document.body.appendChild(a);
            a.click();

            // Remove the temporary link
            document.body.removeChild(a);
        }
    };

    xhr.onprogress = function (event) {
        if (event.lengthComputable) {
            console.log(`Received ${event.loaded} of ${event.total} bytes`);
        } else {
            console.log(`Received ${event.loaded} bytes`); // no
Content-Length
        }

    };

    xhr.onerror = function () {
        alconsole.log("Request failed");
    };
}
```