

DESARROLLO DE APLICACIONES WEB AVANZADO

LABORATORIO N° 13

DESARROLLO DE APLICACIONES CON MEAN.




Alumnos		Nota
Dante Samuel Rodriguez Chambi		
Grupo	A	
Fecha de Entrega		
Docente	Renato Usnayo Cáceres	

OBJETIVOS:

- Implementa aplicaciones usando MEAN.

SEGURIDAD:

	Advertencia: En este laboratorio está prohibida la manipulación del hardware, conexiones eléctricas o de red; así como la ingestión de alimentos o bebidas.
---	---

FUNDAMENTO TEÓRICO:

- Revisar el texto guía que está en el campus Virtual.

NORMAS EMPLEADAS:

- No aplica

RECURSOS:

- En este laboratorio cada alumno trabajará con un equipo con Windows 10.

METODOLOGÍA PARA EL DESARROLLO DE LA TAREA:

- El desarrollo del laboratorio es individual

PROCEDIMIENTO:**Nota:**

Las secciones en cursivas son demostrativas, pero sirven para que usted pueda instalar las herramientas de desarrollo en un equipo externo.

Procedimiento:

Paso 1: Configuración del entorno

Asegúrate de tener Node.js y MongoDB instalados en tu máquina.

Crea una nueva carpeta para tu proyecto y abre una terminal en esa ubicación.

Paso 2: Configuración del backend con Node.js y Express.js

Crea una carpeta llamada "backend" y navega hasta ella en la terminal.

Ejecuta el siguiente comando para inicializar un proyecto de Node.js:

```
npm init -y
```

Instala los paquetes necesarios ejecutando los siguientes comandos:

```
npm install express mongoose cors body-parser
```

Crea un archivo **server.js** en la carpeta "backend" y configúralo para que funcione como servidor **Express.js**. (En caso muestre algún error en ejecución, cambiar `mongoose.connect('mongodb://0.0.0.0:27017/mydatabase'` por `localhost` `mongoose.connect('mongodb://localhost:27017/mydatabase'`

```
const express = require('express');
const cors = require('cors');
const bodyParser = require('body-parser');
const mongoose = require('mongoose');
const itemController = require('./controllers/itemController');

const app = express();
const port = 3000;

// Configuración de middleware
app.use(cors());
app.use(bodyParser.json());

// Conexión a la base de datos de MongoDB
mongoose.connect('mongodb://0.0.0.0:27017/mydatabase', { useNewUrlParser: true,
useUnifiedTopology: true })
.then(() => {
  console.log('Conexión exitosa a MongoDB');
})
.catch((error) => {
  console.log('Error al conectar a MongoDB:', error);
});

// Definir rutas para CRUD
app.get('/api/items', itemController.getItems);
app.get('/api/items/:id', itemController.getItemById);
app.post('/api/items', itemController.createItem);
app.put('/api/items/:id', itemController.updateItem);
app.delete('/api/items/:id', itemController.deleteItem);

// Iniciar el servidor
app.listen(port, () => {
  console.log('Servidor backend en funcionamiento en el puerto', port);
});
```

Paso 3: Creación de modelos y controladores

Crema una carpeta llamada "**models**" dentro de la carpeta "backend".

Dentro de la carpeta "models", crea un archivo **item.js** para definir el modelo de datos del elemento que deseas manejar en el CRUD (por ejemplo, "item").

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

const itemSchema = new mongoose.Schema({
  name: { type: String, required: true },
  description: { type: String, required: true },
  // Otros campos que desees para tu modelo
});

module.exports = mongoose.model('Item', itemSchema);
```

Crema una carpeta llamada "**controllers**" dentro de la carpeta "**backend**".

Dentro de la carpeta "**controllers**", crea un archivo **itemController.js** para definir los controladores que manejarán las operaciones **CRUD** para los elementos.

```
const Item = require('../models/item');

// Obtener todos los elementos
exports.getItems = (req, res) => {
  Item.find()
    .then((items) => {
      res.json(items);
    })
    .catch((error) => {
      res.status(500).json({ error: error.message });
    });
};

// Obtener un elemento por su ID
exports.getItemById = (req, res) => {
  Item.findById(req.params.id)
    .then((item) => {
      if (!item) {
        return res.status(404).json({ message: 'Elemento no encontrado' });
      }
      res.json(item);
    })
    .catch((error) => {
```

```
res.status(500).json({ error: error.message });
});
};

// Crear un nuevo elemento
exports.createItem = (req, res) => {
const newItem = new Item({
name: req.body.name,
description: req.body.description,
// Otros campos que desees para tu modelo
});
newItem.save()
.then((item) => {
res.status(201).json(item);
})
.catch((error) => {
res.status(500).json({ error: error.message });
});
};

// Actualizar un elemento existente
exports.updateItem = (req, res) => {
Item.findByIdAndUpdate(req.params.id, req.body, { new: true })
.then((item) => {
if (!item) {
return res.status(404).json({ message: 'Elemento no encontrado' });
}
res.json(item);
})
.catch((error) => {
res.status(500).json({ error: error.message });
});
});

// Eliminar un elemento existente
exports.deleteItem = (req, res) => {
Item.findByIdAndDelete(req.params.id)
.then((item) => {
if (!item) {
return res.status(404).json({ message: 'Elemento no encontrado' });
}
res.json({ message: 'Elemento eliminado correctamente' });
})
.catch((error) => {
res.status(500).json({ error: error.message });
});
});
```

};

FRONTEND

Paso 4: Configuración del frontend con Angular

Abre otra terminal y navega hasta la carpeta raíz de tu proyecto.

Ejecuta el siguiente comando para crear un nuevo proyecto de Angular:

```
ng new frontend
```

Cambia al directorio "frontend" usando el siguiente comando:

```
cd frontend
```

Crea un componente para la interfaz de usuario del CRUD. Por ejemplo, puedes ejecutar el siguiente comando para crear un componente llamado "item":

```
ng generate component item
```

Paso 5: Configuración de las rutas en Angular

Abre el archivo **app-routing.module.ts** dentro de la carpeta **"frontend/src/app"**.

Agrega las siguientes rutas para el **CRUD**:

```
import { NgModule } from '@angular/core';
import { RouterModule, Routes } from '@angular/router';
import { ItemComponent } from './item/item.component';

const routes: Routes = [
  { path: 'items', component: ItemComponent },
  // otras rutas si las tienes
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```

Paso 6: Configuración del servicio en Angular

Crea un servicio para interactuar con el backend. Ejecuta el siguiente comando para crear un servicio llamado "item":

```
ng generate service item
```

Abre el archivo **item.service.ts** dentro de la carpeta "**frontend/src/app/**" y reemplaza su contenido con el siguiente código, también crea el archivo **item.service.ts** en "**frontend/src/app/item**" con el mismo **código**

```
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { Observable } from 'rxjs';

@Injectable({
  providedIn: 'root'
})
export class ItemService {
  private apiUrl = 'http://localhost:3000/api/items';

  constructor(private http: HttpClient) {}

  getItems() {
    return this.http.get<any[]>(this.apiUrl);
  }

  getItemById(id: string): Observable<any> {
    const url = `${this.apiUrl}/${id}`;
    return this.http.get(url);
  }

  createItem(item: any): Observable<any> {
    return this.http.post(this.apiUrl, item);
  }

  updateItem(id: string, item: any): Observable<any> {
    const url = `${this.apiUrl}/${id}`;
    return this.http.put(url, item);
  }

  deleteItem(id: string): Observable<any> {
    const url = `${this.apiUrl}/${id}`;
    return this.http.delete(url);
  }
}
```

```
}
```

Paso 7: Configuración del componente en Angular

Abre el archivo **item.component.ts** dentro de la carpeta "**frontend/src/app/item**" y reemplaza su contenido con el siguiente código:

```
import { Component, OnInit } from '@angular/core';
import { ItemService } from '../item.service';

@Component({
  selector: 'app-item',
  templateUrl: './item.component.html',
  styleUrls: ['./item.component.css']
})
export class ItemComponent implements OnInit {
  items: any[] = [];
  currentItem: any = {};

  constructor(private itemService: ItemService) {}

  ngOnInit(): void {
    this.getItems();
  }

  getItems(): void {
    this.itemService.getItems()
      .subscribe((items) => {
        this.items = items;
      });
  }

  getItemById(id: string): void {
    this.itemService.getItemById(id)
      .subscribe((item) => {
        this.currentItem = item;
      });
  }

  createItem(item: any): void {
    this.itemService.createItem(item)
      .subscribe(() => {
        this.getItems();
        this.currentItem = {};
      });
  }
}
```



```
updateItem(id: string, item: any): void {
  this.itemService.updateItem(id, item)
  .subscribe(() => {
    this.getItems();
    this.currentItem = {};
  });
}

deleteItem(id: string): void {
  this.itemService.deleteItem(id)
  .subscribe(() => {
    this.getItems();
  });
}

editItem(id: string): void {
  this.getItemById(id);
}
}
```

Paso 8: Configuración de la plantilla HTML en Angular

Abre el archivo **item.component.html** dentro de la carpeta **"frontend/src/app/item"** y reemplaza su contenido con el siguiente código:

```
<div class="container mt-4">
  <h2>Items</h2>

  <!-- Formulario para crear/editar un item -->
  <form (ngSubmit)="currentItem._id ? updateItem(currentItem._id, currentItem) :
createItem(currentItem)">
    <div class="form-group">
      <label for="name">Name:</label>
      <input type="text" class="form-control" [(ngModel)]="currentItem.name" name="name" required>
    </div>

    <div class="form-group">
      <label for="description">Description:</label>
      <input type="text" class="form-control" [(ngModel)]="currentItem.description" name="description"
required>
    </div>

    <button type="submit" class="btn btn-primary">{{currentItem._id ? 'Update' : 'Add'}}</button>
  </form>
```

```
<hr>

<!-- Lista de items -->
<table class="table">
  <thead>
    <tr>
      <th>Name</th>
      <th>Description</th>
      <th>Actions</th>
    </tr>
  </thead>
  <tbody>
    <tr *ngFor="let item of items">
      <td>{{ item.name }}</td>
      <td>{{ item.description }}</td>
      <td>
        <button class="btn btn-sm btn-primary" (click)="editItem(item._id)">Edit</button>
        <button class="btn btn-sm btn-danger" (click)="deleteItem(item._id)">Delete</button>
      </td>
    </tr>
  </tbody>
</table>
</div>
```

En el archivo **app.module.ts** agregar las líneas necesarias para que quede como se muestra a continuación:

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { FormsModule } from '@angular/forms'; // Agrega esta línea
import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';
import { ItemComponent } from './item/item.component';
import { HttpClientModule } from '@angular/common/http';

@NgModule({
  declarations: [
    AppComponent,
    ItemComponent
  ],
  imports: [
    BrowserModule,
    FormsModule, // Agrega esta línea
    HttpClientModule,
```

```
    AppRoutingModuleModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule {}
```

Coloque el siguiente código en **app.component.html**

```
<router-outlet></router-outlet>
```

Colocar el siguiente código en **item.services.spec.ts**

```
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { Observable } from 'rxjs';

@Injectable({
  providedIn: 'root'
})
export class ItemService {
  private apiUrl = 'http://localhost:3000/api/items';

  constructor(private http: HttpClient) {}

  getItems(): Observable<any> {
    return this.http.get(this.apiUrl);
  }

  getItemById(id: string): Observable<any> {
    const url = `${this.apiUrl}/${id}`;
    return this.http.get(url);
  }

  createItem(item: any): Observable<any> {
    return this.http.post(this.apiUrl, item);
  }

  updateItem(id: string, item: any): Observable<any> {
    const url = `${this.apiUrl}/${id}`;
    return this.http.put(url, item);
  }

  deleteItem(id: string): Observable<any> {
    const url = `${this.apiUrl}/${id}`;
    return this.http.delete(url);
  }
}
```

INSTALACIÓN DE BOOTSTRAP

Abre una ventana de terminal en la raíz de tu proyecto Angular.

Instala Bootstrap utilizando el administrador de paquetes **npm (Node Package Manager)** ejecutando el siguiente comando:

```
npm install bootstrap
```

Una vez completada la instalación, ve al archivo **angular.json** en la raíz de tu proyecto Angular.

Busca la sección "**styles**" dentro del archivo **angular.json**. Debería tener una estructura similar a esta:

```
"styles": [  
  "src/styles.css"  
]
```

Agrega la ruta al archivo CSS de Bootstrap en la matriz "styles". Puedes utilizar la ruta relativa desde la raíz de tu proyecto. Por ejemplo:

```
"styles": [  
  "src/styles.css",  
  "node_modules/bootstrap/dist/css/bootstrap.min.css"  
]
```

Guarda el archivo angular.json para aplicar los cambios.

Paso 9: Ejecución de la aplicación

Abre una CMD y ejecuta

```
mongod
```

Abre una terminal en la carpeta raíz del proyecto (donde se encuentra la carpeta "backend" y "frontend").

Inicia el servidor backend ejecutando el siguiente comando dentro de la carpeta "backend":

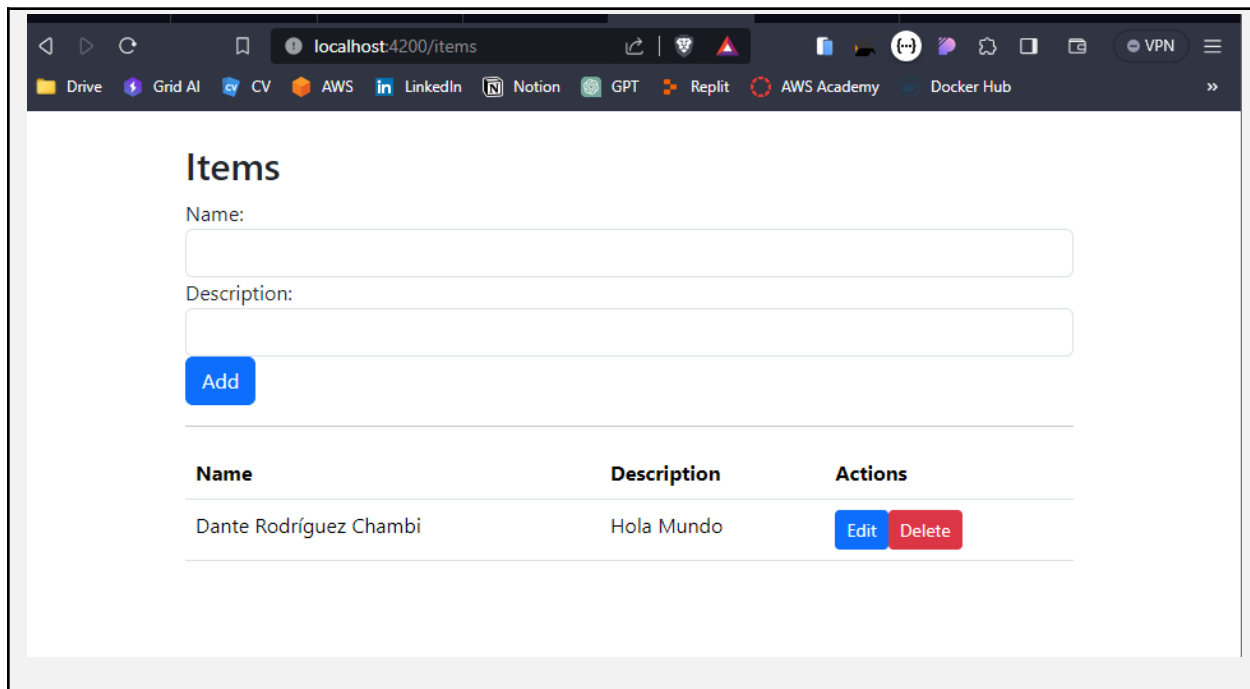
```
node server.js
```

Abre otra terminal en la carpeta "frontend" y ejecuta el siguiente comando para iniciar el servidor de desarrollo de Angular:

```
ng serve
```

Abre tu navegador web y accede a la URL **http://localhost:4200/items** para ver la aplicación en funcionamiento.

Mostrar una captura del funcionamiento de la aplicación



TAREA

Modifique la aplicación enfocada a un tema de su preferencia (películas, música, videojuegos, etc) aumente más campos de acuerdo con el enfoque de su tema

Validación de formularios:

- Implementa validaciones de formularios utilizando las características de validación de Bootstrap.
- Asegúrate de que los campos obligatorios se validen antes de enviar los datos al servidor.
- Muestra mensajes de error o indicadores visuales cuando los datos del formulario no cumplan con los requisitos de validación.

Mejorar la experiencia de usuario:

- Agrega confirmaciones antes de realizar acciones como eliminar un elemento.
- Proporciona retroalimentación visual al usuario cuando se realicen operaciones exitosas, como la creación o actualización de un elemento.
- Agrega un campo de búsqueda para permitir al usuario buscar elementos en la lista.
- Agrega características adicionales a la aplicación, como la capacidad de cargar imágenes para cada elemento, ordenar la lista por diferentes campos, etc.

BACKEND

Server.js

```
1  const express = require('express');
2  const cors = require('cors');
3  const bodyParser = require('body-parser');
4  const mongoose = require('mongoose');
5  const multer = require('multer');
6
7  const bibliotecaController = require('./controllers/bibliotecaController');
8
9  const app = express();
10 const port = 3000;
11
12 // Configuración de middleware
13 app.use(cors());
14 app.use(bodyParser.json());
15
16 // Configuración de multer para la subida de archivos
17 const storage = multer.diskStorage({
18   destination: (req, file, cb) => {
19     cb(null, 'uploads/'); // Directorio donde se guardarán los archivos
20   },
21   filename: (req, file, cb) => {
22     cb(null, file.fieldname + '-' + Date.now() + '.' + file.originalname.split('.').pop());
23   }
24 });
25
26 const upload = multer({ storage: storage });
27
28 // Conexión a la base de datos de MongoDB
29 mongoose.connect('mongodb://0.0.0.0:27017/mydatabase', {
30   useNewUrlParser: true,
31   useUnifiedTopology: true,
32 })
33 .then(() => {
34   console.log('Conexión exitosa a MongoDB');
35 })
36 .catch((error) => {
37   console.log('Error al conectar a MongoDB:', error);
38 });
39
40 // Definir rutas para CRUD de Biblioteca
41 app.get('/api/bibliotecas', bibliotecaController.getBibliotecas);
42 app.get('/api/bibliotecas/:id', bibliotecaController.getBibliotecaById);
43 app.post('/api/bibliotecas', upload.single('imagen'), bibliotecaController.createBiblioteca);
44 app.put('/api/bibliotecas/:id', upload.single('imagen'), bibliotecaController.updateBiblioteca);
45 app.delete('/api/bibliotecas/:id', bibliotecaController.deleteBiblioteca);
46
47 // Ruta para servir archivos estáticos (opcional, dependiendo de tus necesidades)
48 app.use('/uploads', express.static('uploads'));
49
50 // Iniciar el servidor
51 app.listen(port, () => {
52   console.log('Servidor backend en funcionamiento en el puerto', port);
53 });
```

FRONTEND

BibliotecaComponent.ts

```
1 import { Component, OnInit } from '@angular/core';
2 import { FormBuilder, FormGroup, Validators } from '@angular/forms';
3 import { BibliotecaService } from '../biblioteca.service';
4
5 import { MatDialog } from '@angular/material/dialog';
6 import { ConfirmDialogComponent } from '../confirm-dialog/confirm-dialog.component';
7
8 @Component({
9   selector: 'app-biblioteca',
10  templateUrl: './biblioteca.component.html',
11  styleUrls: ['./biblioteca.component.css']
12 })
13 export class BibliotecaComponent implements OnInit {
14   bibliotecas: any[] = [];
15   filteredBibliotecas: any[] = []; // Lista filtrada para búsqueda
16   currentBiblioteca: any = {};
17   bibliotecaForm: FormGroup;
18   selectedFile: File | null = null;
19   searchQuery: string = ''; // Variable para almacenar la consulta de búsqueda
20
21   constructor(
22     private bibliotecaService: BibliotecaService,
23     private fb: FormBuilder,
24     private dialog: MatDialog
25   ) {
26
27     this.bibliotecaForm = this.fb.group({
28       title: ['', Validators.required],
29       descripcion: ['', Validators.required],
30       imagen: [null, Validators.required],
31       fecha: ['', Validators.required]
32     });
33   }
```

```
1 ngOnInit(): void {
2   this.getBibliotecas();
3 }
4
5 getBibliotecas(): void {
6   this.bibliotecaService.getBibliotecas()
7     .subscribe((bibliotecas) => {
8     this.bibliotecas = bibliotecas;
9     this.filteredBibliotecas = bibliotecas; // Inicializa la lista filtrada
10    this.applyFilter(); // Aplica el filtro inicial
11  });
12 }
13
14 createBiblioteca(): void {
15   this.bibliotecaService.createBiblioteca(this.bibliotecaForm.value)
16     .subscribe(() => {
17     this.getBibliotecas();
18     this.bibliotecaForm.reset();
19     this.currentBiblioteca = {};
20   });
21 }
22
23 updateBiblioteca(id: string): void {
24   this.bibliotecaService.updateBiblioteca(id, this.bibliotecaForm.value)
25     .subscribe(() => {
26     this.getBibliotecas();
27     this.bibliotecaForm.reset();
28     this.currentBiblioteca = {};
29   });
30 }
```



```

1  deleteBiblioteca(id: string): void {
2    const dialogRef = this.dialog.open(ConfirmDialogComponent, {
3      data: {
4        title: 'Confirmación',
5        message: '¿Estás seguro de que deseas eliminar este elemento?',
6      },
7    });
8
9    dialogRef.afterClosed().subscribe((result: boolean) => {
10     if (result) {
11       this.bibliotecaService.deleteBiblioteca(id)
12         .subscribe(() => {
13           this.getBibliotecas();
14         });
15     }
16   });
17 }
18 editBiblioteca(id: string): void {
19   this.bibliotecaService.getBibliotecaById(id)
20     .subscribe((biblioteca) => {
21     this.currentBiblioteca = biblioteca;
22     this.bibliotecaForm.patchValue({
23       title: biblioteca.title,
24       descripcion: biblioteca.descripcion,
25       imagen: null,
26       fecha: biblioteca.fecha
27     });
28   });
29 }
30
31 onFileChange(event: any): void {
32   const fileInput = this.bibliotecaForm.get('imagen');
33   if (fileInput) {
34     const file = (event.target as HTMLInputElement).files?.[0];
35     if (file) {
36       fileInput.setValue(file);
37       fileInput.updateValueAndValidity();
38     }
39   }
40 }
41 // Método para aplicar el filtro de búsqueda
42 applyFilter(): void {
43   this.filteredBibliotecas = this.bibliotecas.filter((biblioteca) => {
44     return (
45       biblioteca.title.toLowerCase().includes(this.searchQuery.toLowerCase()) ||
46       biblioteca.descripcion.toLowerCase().includes(this.searchQuery.toLowerCase()) ||
47       biblioteca.fecha.toLowerCase().includes(this.searchQuery.toLowerCase())
48     );
49   });
50 }
51 orderList(field: string): void {
52   this.filteredBibliotecas.sort((a, b) => {
53     if (a[field] < b[field]) return -1;
54     if (a[field] > b[field]) return 1;
55     return 0;
56   });
57 }
58 }

```

biblioteca.component.html

```

1  <div class="container mt-4">
2    <h2>Bibliotecas</h2>
3
4    <form [formGroup]="bibliotecaForm" (ngSubmit)="currentBiblioteca_id ? updateBiblioteca(currentBiblioteca_id) : createBiblioteca()" enctype="multipart/form-data">
5      <div class="form-group">
6        <label for="title">Title:</label>
7        <input type="text" class="form-control" formControlName="title" required>
8      </div>
9
10     <div class="form-group">
11       <label for="descripcion">Description:</label>
12       <textarea class="form-control" formControlName="descripcion" required></textarea>
13     </div>
14
15     <div class="form-group">
16       <label for="imagen">Image:</label>
17       <input type="file" accept=".png" (change)="onFileChange($event)" formControlName="imagen" required>
18     </div>
19
20     <div class="form-group">
21       <label for="fecha">Fecha:</label>
22       <input type="date" class="form-control" formControlName="fecha" required>
23     </div>
24
25     <button type="submit" class="btn btn-primary">Add</button>
26   </form>

```

biblioteca-edit.component.ts

```
1 import { Component, Inject } from '@angular/core';
2 import { FormBuilder, FormGroup, Validators } from '@angular/forms';
3 import { MAT_DIALOG_DATA, MatDialogRef } from '@angular/material/dialog';
4
5 @Component({
6   selector: 'app-biblioteca-edit',
7   templateUrl: './biblioteca-edit.component.html',
8   styleUrls: ['./biblioteca-edit.component.css']
9 })
10 export class BibliotecaEditComponent {
11   bibliotecaForm: FormGroup;
12
13   constructor(
14     private fb: FormBuilder,
15     public dialogRef: MatDialogRef<BibliotecaEditComponent>,
16     @Inject(MAT_DIALOG_DATA) public data: any
17   ) {
18     this.bibliotecaForm = this.fb.group({
19       title: [data.title, Validators.required],
20       descripcion: [data.descripcion, Validators.required],
21       imagen: [null, Validators.required],
22       fecha: [data.fecha, Validators.required]
23     });
24   }
25
26   onFileChange(event: any): void {
27     const fileInput = this.bibliotecaForm.get('imagen');
28     if (fileInput) {
29       const file = (event.target as HTMLInputElement).files?.[0];
30       if (file) {
31         fileInput.setValue(file);
32         fileInput.updateValueAndValidity();
33       }
34     }
35   }
36
37   saveChanges(): void {
38     this.dialogRef.close(this.bibliotecaForm.value);
39   }
40
41   closeDialog(): void {
42     this.dialogRef.close();
43   }
44 }
```

biblioteca-list.component

```
1 import { Component, OnInit } from '@angular/core';
2 import { BibliotecaService } from '../biblioteca.service';
3 import { MatDialog } from '@angular/material/dialog';
4 import { Router } from '@angular/router';
5 import { ConfirmDialogComponent } from '../confirm-dialog/confirm-dialog.component';
6 import { BibliotecaEditComponent } from '../biblioteca-edit/biblioteca-edit.component';
7
8 @Component({
9   selector: 'app-biblioteca-list',
10  templateUrl: './biblioteca-list.component.html',
11  styleUrls: ['./biblioteca-list.component.css']
12 })
13 export class BibliotecaListComponent implements OnInit {
14   bibliotecas: any[] = [];
15   filteredBibliotecas: any[] = [];
16   searchQuery: string = '';
17
18   constructor(
19     private bibliotecaService: BibliotecaService,
20     private dialog: MatDialog,
21     private router: Router
22   ) {}
23
24   ngOnInit(): void {
25     this.getBibliotecas();
26   }
27
28   getBibliotecas(): void {
29     this.bibliotecaService.getBibliotecas()
30       .subscribe((bibliotecas) => {
31         this.bibliotecas = bibliotecas;
32         this.filteredBibliotecas = bibliotecas;
33         this.applyFilter();
34       });
35   }
36
37   applyFilter(): void {
38     this.filteredBibliotecas = this.bibliotecas.filter((biblioteca) => {
39       return (
40         biblioteca.title.toLowerCase().includes(this.searchQuery.toLowerCase()) ||
41         biblioteca.descripcion.toLowerCase().includes(this.searchQuery.toLowerCase()) ||
42         biblioteca.fecha.toLowerCase().includes(this.searchQuery.toLowerCase())
43       );
44     });
45   }
46
47   orderList(field: string): void {
48     this.filteredBibliotecas.sort((a, b) => {
49       if (a[field] < b[field]) return -1;
50       if (a[field] > b[field]) return 1;
51       return 0;
52     });
53   }
54
55   deleteBiblioteca(id: string): void {
56     const dialogRef = this.dialog.open(ConfirmDialogComponent, {
57       data: {
58         title: 'Confirmación',
59         message: '¿Estás seguro de que deseas eliminar este elemento?',
60       },
61     });
62   }
63 }
```

```
27     dialogRef.afterClosed().subscribe((result: boolean) => {
28         if (result) {
29             this.bibliotecaService.deleteBiblioteca(id)
30                 .subscribe(() => {
31                     this.getBibliotecas();
32                 });
33         }
34     });
35 }
36
37 editBiblioteca(id: string): void {
38     this.bibliotecaService.getBibliotecaById(id)
39         .subscribe((biblioteca) => {
40             const dialogRef = this.dialog.open(BibliotecaEditComponent, {
41                 width: '400px',
42                 data: biblioteca
43             });
44
45             dialogRef.afterClosed().subscribe((result: any) => {
46                 if (result) {
47                     this.bibliotecaService.updateBiblioteca(id, result)
48                         .subscribe(() => {
49                             this.getBibliotecas();
50                         });
51                 }
52             });
53         });
54     }
55 }
```

confirm-dialog.component.ts

```
1 import { Component, Inject } from '@angular/core';
2 import { MAT_DIALOG_DATA, MatDialogRef } from '@angular/material/dialog';
3
4 @Component({
5     selector: 'app-confirm-dialog',
6     templateUrl: './confirm-dialog.component.html',
7     styleUrls: ['./confirm-dialog.component.css']
8 })
9 export class ConfirmDialogComponent {
10     constructor(
11         public dialogRef: MatDialogRef<ConfirmDialogComponent>,
12         @Inject(MAT_DIALOG_DATA) public data: { title: string; message: string },
13     ) {}
14
15     onNoClick(): void {
16         this.dialogRef.close(false);
17     }
18 }
19
```

biblioteca.service.ts

```
1 import { Injectable } from '@angular/core';
2 import { HttpClient } from '@angular/common/http';
3 import { Observable, throwError } from 'rxjs';
4 import { catchError } from 'rxjs/operators';
5
6 @Injectable({
7   providedIn: 'root'
8 })
9 export class BibliotecaService {
10   private apiUrl = 'http://localhost:3000/api/bibliotecas';
11
12   constructor(private http: HttpClient) { }
13
14   getBibliotecas(): Observable<any[]> {
15     return this.http.get<any[]>(this.apiUrl)
16       .pipe(
17         catchError(this.handleError)
18       );
19   }
20
21   getBibliotecaById(id: string): Observable<any> {
22     const url = `${this.apiUrl}/${id}`;
23     return this.http.get<any>(url)
24       .pipe(
25         catchError(this.handleError)
26       );
27   }
```

```
1 createBiblioteca(biblioteca: any): Observable<any> {
2     const formData = new FormData();
3     Object.keys(biblioteca).forEach(key => {
4         formData.append(key, biblioteca[key]);
5     });
6
7     return this.http.post(this.apiUrl, formData)
8         .pipe(
9             catchError(this.handleError)
10        );
11 }
12
13 updateBiblioteca(id: string, biblioteca: any): Observable<any> {
14     const formData = new FormData();
15     Object.keys(biblioteca).forEach(key => {
16         formData.append(key, biblioteca[key]);
17     });
18
19     const url = `${this.apiUrl}/${id}`;
20     return this.http.put(url, formData)
21         .pipe(
22             catchError(this.handleError)
23        );
24 }
25
26 deleteBiblioteca(id: string): Observable<any> {
27     const url = `${this.apiUrl}/${id}`;
28     return this.http.delete(url)
29         .pipe(
30             catchError(this.handleError)
31        );
32 }
33
34 private handleError(error: any): Observable<never> {
35     console.error('Error:', error);
36     return throwError('Ocurrió un error. Por favor, inténtelo nuevamente.');
```

localhost:4200/biblioteca

Search for...

Bibliotecas

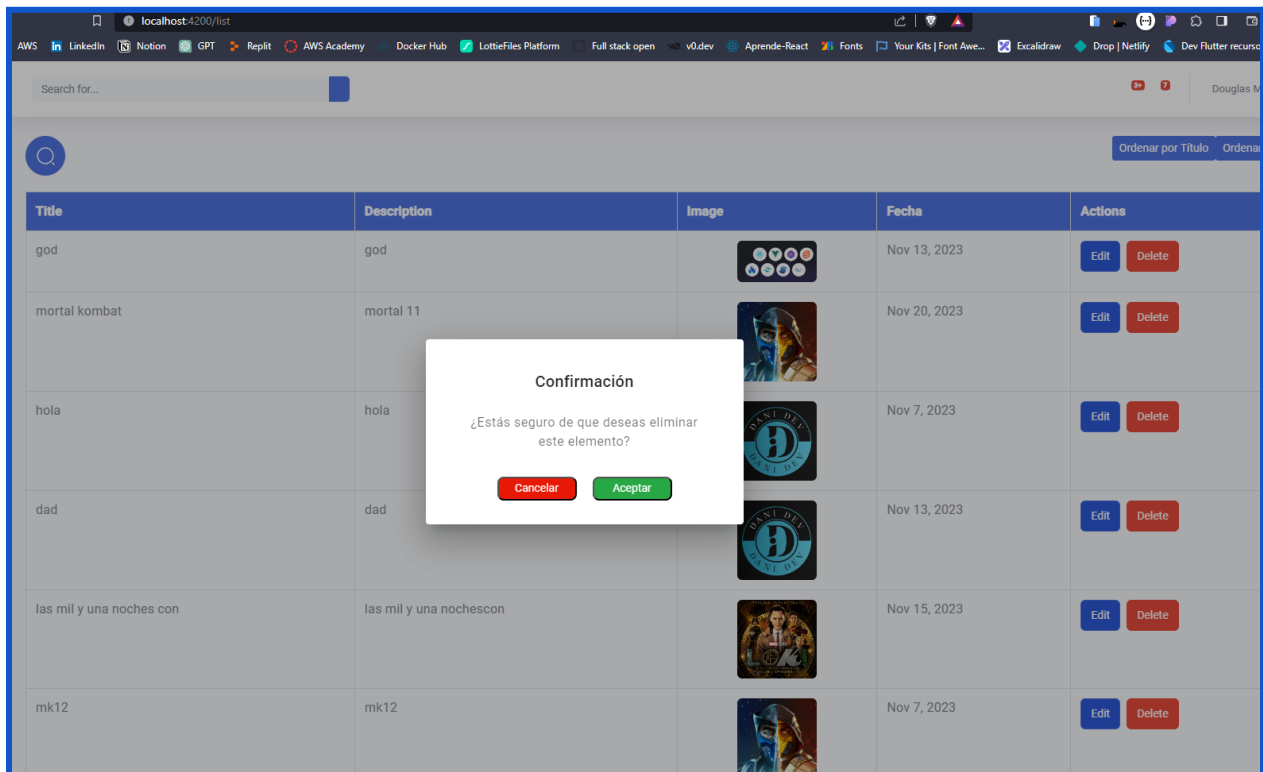
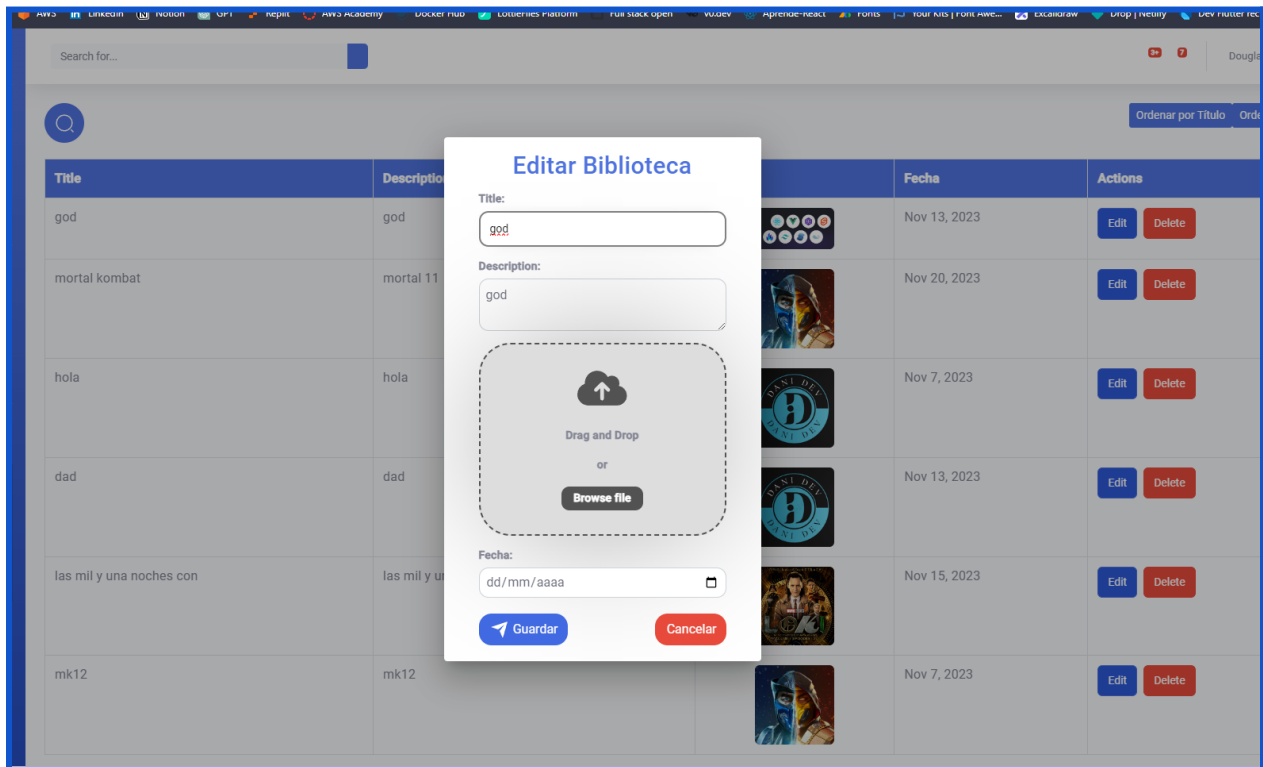
Title:

Description:

Image: Seleccionar archivo Sin archivos seleccionados

Fecha:

[illegible]



Colocar capturas del código y capturas de ejecución O (Se pueden grabar explicando el código y ejecución de este)

Adjuntar el código subido en Git o en un archivo zip

VIDEO: <https://youtu.be/FEzuadGHg74>

GITHUB: <https://github.com/DanniDevv/Biblioteca-WAV.git>

OBSERVACIONES: *(Las observaciones son las notas aclaratorias, objeciones y problemas que se pudo presentar en el desarrollo del laboratorio)*

- Es importante incorporar una gestión más robusta de errores tanto en el frontend como en el backend.
- Aunque se controla la carga de imágenes en el servidor con Multer, se podría mejorar la seguridad, por ejemplo, validando el tipo de archivo o estableciendo límites de tamaño para prevenir posibles vulnerabilidades.
- Para un proyecto más grande o para compartir el código con otros desarrolladores, se recomienda agregar documentación y comentarios para facilitar la comprensión del código y la colaboración.
- No se abordó la implementación de pruebas unitarias e integración en este código. Introducir pruebas puede garantizar un desarrollo más robusto y una mayor confianza en la estabilidad de la aplicación.
- Aunque funcional, la interfaz de usuario podría mejorarse con estilos y diseños más atractivos. La usabilidad también puede ser optimizada para proporcionar una experiencia más intuitiva al usuario.

CONCLUSIONES: *(Las conclusiones son una opinión sobre tu trabajo, explicar cómo resolviste las dudas o problemas presentados en el laboratorio. Además de aportar una opinión crítica de lo realizado)*

- Angular y Node.js con Express y MongoDB permite la construcción de aplicaciones web de extremo a extremo, manejando tanto el frontend como el backend de manera eficiente.
- El uso de formularios reactivos en Angular facilita la creación y validación de formularios complejos. Permite una gestión más avanzada de los datos y la interacción del usuario con la aplicación.
- La utilización de ventanas modales con Angular Material, como se vio en el componente BibliotecaEditComponent, es una técnica efectiva para la creación de interfaces de usuario más limpias y amigables.
- El código muestra un proceso iterativo de desarrollo, incorporando mejoras y nuevas características a medida que avanzamos. Desde la carga inicial hasta la adición de funcionalidades como la búsqueda, ordenación y edición, el desarrollo ha sido evolutivo y basado en requisitos.
- La implementación de la carga y visualización de imágenes en la aplicación añade una dimensión multimedia importante