

## DESARROLLO DE APLICACIONES WEB AVANZADO

### LABORATORIO N° 14

### ADJUNTAR ARCHIVOS Y GENERACIÓN DE REPORTE




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

**OBJETIVOS:**

- Implementa librerías para adjuntar archivos y generar reportes en Angular y Node.js

**SEGURIDAD:**

	<b>Advertencia:</b> 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: Configurar el entorno

Asegúrate de tener Node.js instalado en tu sistema. Puedes descargarlo e instalarlo desde el sitio web oficial de Node.js.

Paso 2: Crear una nueva carpeta para el proyecto

Crea una nueva carpeta en tu sistema y ábrela en tu editor de código favorito. Esta será la carpeta raíz de tu proyecto.

Paso 3: Inicializar el proyecto Node.js

Abre una terminal en la carpeta raíz del proyecto y ejecuta el siguiente comando para inicializar el proyecto Node.js:

```
npm init -y
```

Esto creará un archivo package.json con la configuración predeterminada.

Paso 4: Instalar los módulos necesarios

En la misma terminal, ejecuta el siguiente comando para instalar los módulos necesarios:

```
npm install express multer --save
```

Esto instalará el módulo express para crear el servidor web y el módulo multer para manejar la carga de archivos.

Paso 5: Crear el servidor Express

Crea un archivo llamado **server.js** en la carpeta raíz y agrega el siguiente código para configurar el servidor Express:

```
const express = require('express');  
const multer = require('multer');  
const app = express();  
const upload = multer({ dest: 'uploads/' }); // Directorio donde se guardarán los  
archivos adjuntos  
  
app.post('/upload', upload.single('file'), (req, res) => {  
  res.send('Archivo cargado con éxito');  
});  
  
app.listen(3000, () => {  
  console.log('Servidor escuchando en el puerto 3000');  
});  
  
app.get('/', (req, res) => {  
  res.sendFile(__dirname + '/index.html');  
});
```

Paso 6: Crear un formulario HTML para cargar archivos

Crema un archivo llamado **index.html** en la carpeta raíz y agrega el siguiente código HTML básico:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Carga de archivos</title>
  </head>
  <body>
    <h1>Carga de archivos</h1>
    <form action="/upload" method="POST" enctype="multipart/form-data">
      <input type="file" name="file" />
      <input type="submit" value="Cargar" />
    </form>
  </body>
</html>
```

Paso 7: Iniciar el servidor

En la terminal, ejecuta el siguiente comando para iniciar el servidor:

```
node server.js
```

Paso 8: Probar la carga de archivos

Abre tu navegador web y accede a <http://localhost:3000>. Deberías ver un formulario para cargar archivos.

Selecciona un archivo en el formulario y haz clic en el botón "Cargar".

Si todo funciona correctamente, verás el mensaje "Archivo cargado con éxito" en tu navegador y el archivo se guardará en la carpeta uploads/ dentro de tu proyecto.

Mostrar con capturas el procedimiento de cargar una foto, cargue la foto y muestra como esta guardada en la carpeta uploads



implementar una función adicional que permita mostrar los detalles del archivo cargado, como su nombre, tamaño y tipo. Esto proporcionará más información al usuario después de cargar el archivo.

### Paso 1: Modificar el controlador de carga de archivos

En el archivo `server.js`, modifica el controlador de carga de archivos para enviar una respuesta con los detalles del archivo cargado. Puedes obtener esta información del objeto **req.file** proporcionado por Multer. Por ejemplo:

```
app.post('/upload', upload.single('file'), (req, res) => {  
  const fileInfo = {  
    filename: req.file.filename,  
    originalname: req.file.originalname,  
    size: req.file.size,  
    mimetype: req.file.mimetype  
  };  
  res.send(fileInfo);  
});
```

## Paso 2: Actualizar el formulario HTML

En el archivo **index.html**, agrega una sección para mostrar los detalles del archivo cargado. Puedes usar elementos de HTML para mostrar esta información de forma clara. Por ejemplo:

```
<body>
  <h1>Carga de archivos</h1>
  <form action="/upload" method="POST" enctype="multipart/form-data">
    <input type="file" name="file" />
    <input type="submit" value="Cargar" />
  </form>

  <div id="file-info"></div>

  <script>
    // Script para mostrar los detalles del archivo cargado
    const form = document.querySelector('form');
    const fileInfoDiv = document.querySelector('#file-info');

    form.addEventListener('submit', async (event) => {
      event.preventDefault();

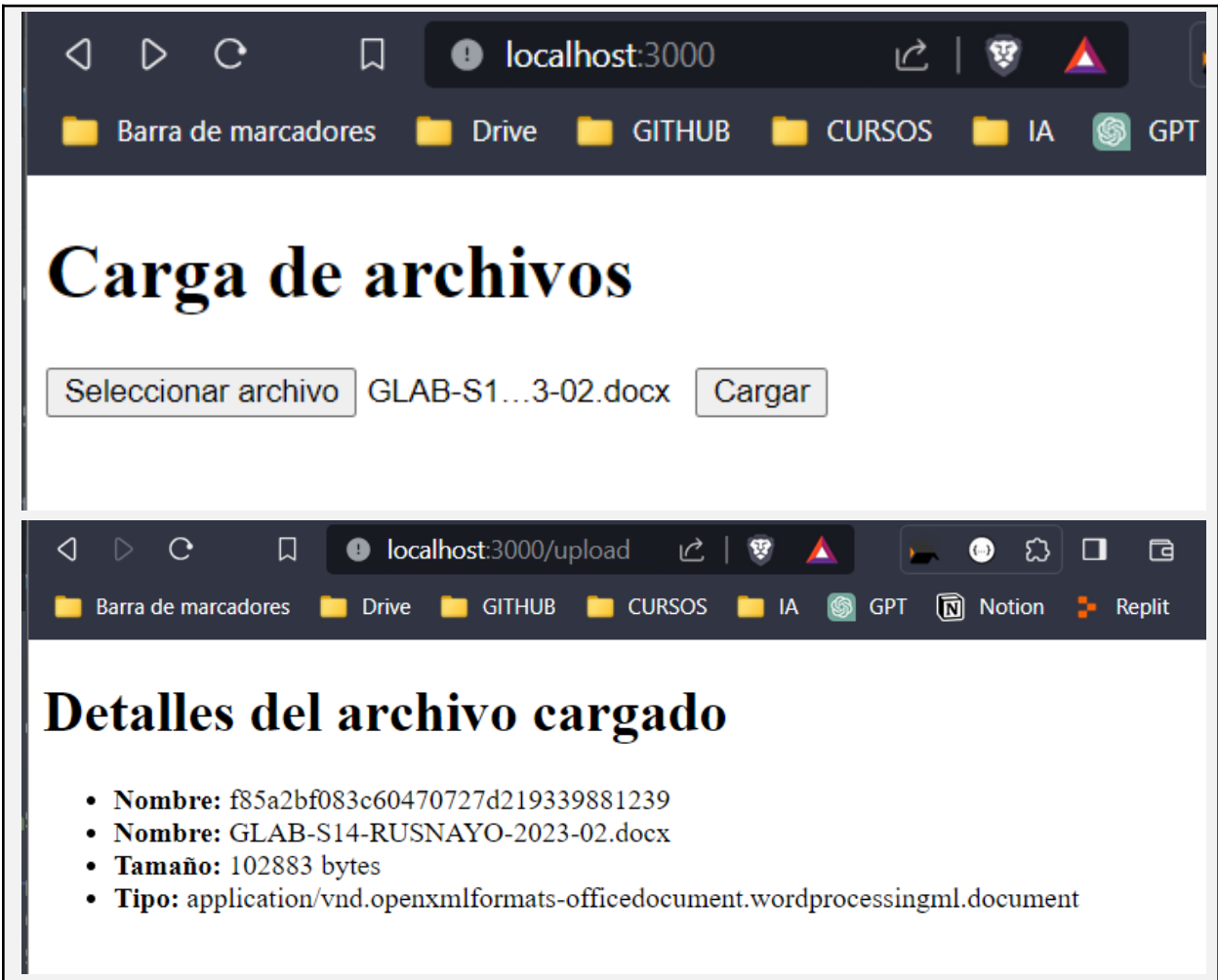
      const formData = new FormData(form);
      const response = await fetch('/upload', {
        method: 'POST',
        body: formData
      });

      if (response.ok) {
        const fileInfo = await response.json();
        fileInfoDiv.innerHTML = `
          <p>Nombre: ${fileInfo.filename}</p>
          <p>Nombre original: ${fileInfo.originalname}</p>
          <p>Tamaño: ${fileInfo.size} bytes</p>
          <p>Tipo MIME: ${fileInfo.mimetype}</p>
        `;
      } else {
        fileInfoDiv.innerHTML = 'Error al cargar el archivo.';
      }
    });
  </script>
```

</body>

Con estos cambios, después de cargar un archivo, se mostrarán los detalles del archivo en la sección correspondiente en la página.

Volver a cargar un archivo y mostrar los datos adicionales que se muestran



### Tarea:

Implementar la funcionalidad de validación de archivos:

- **Agrega validaciones para asegurarte de que solo se carguen ciertos tipos de archivos o que los archivos no superen un tamaño máximo determinado.**

Añadir una interfaz de usuario más amigable:

- Mejora el diseño y la experiencia de usuario del formulario HTML y la página de resultados. Puedes utilizar CSS y JavaScript para agregar estilos y funcionalidades interactivas.

Implementar la opción de carga de múltiples archivos:

- Permite al usuario seleccionar y cargar varios archivos al mismo tiempo. Modifica el formulario y el controlador para admitir la carga de múltiples archivos y mostrar los detalles de cada archivo cargado.

***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***

## PARTE 2: GENERACIÓN DE REPORTES

Paso 1: Configurar el proyecto Angular

Asegúrate de tener Angular CLI instalado en tu sistema.

Crea un nuevo proyecto Angular ejecutando el siguiente comando en tu terminal:

```
ng new Lab14Reporte
```

Ve al directorio del proyecto:

```
cd Lab14Reporte
```

Paso 2: Crear un componente para el informe

Crea un nuevo componente llamado "reporte-peliculas" utilizando el siguiente comando:



```
ng generate component reporte-peliculas
```

Esto generará una carpeta llamada "reporte-peliculas" con los archivos necesarios para el componente.

### Paso 3: Preparar los datos

Crea un archivo JSON llamado "peliculas.json" en src/assets

Agrega algunos datos de ejemplo en el archivo JSON. Por ejemplo:

```
[
  {
    "titulo": "Película 1",
    "genero": "Acción",
    "lanzamiento": 2020
  },
  {
    "titulo": "Película 2",
    "genero": "Comedia",
    "lanzamiento": 2018
  },
  {
    "titulo": "Película 3",
    "genero": "Drama",
    "lanzamiento": 2021
  }
]
```

### Paso 4: Leer los datos en el componente

Abre el archivo "reporte-peliculas.component.ts" ubicado en la carpeta del componente.

Importa el módulo HttpClient de Angular agregando la siguiente línea al comienzo del archivo:

```
import { HttpClient } from '@angular/common/http';
```

En el constructor del componente, inyecta la dependencia del HttpClient:

```
constructor(private http: HttpClient) {}
```

Crea una variable para almacenar los datos de las películas:

```
películas: any[] = [];
```

En el método `ngOnInit()` del componente, realiza una llamada HTTP para leer los datos del archivo JSON:

```
ngOnInit() {  
  this.http.get<any[]>('./assets/peliculas.json').subscribe(data => {  
    this.películas = data;  
  });  
}
```

Reporte-películas.component.ts debería quedar así:

```
import { Component, OnInit } from '@angular/core';  
import { HttpClient } from '@angular/common/http';  
  
@Component({  
  selector: 'app-reporte-peliculas',  
  templateUrl: './reporte-peliculas.component.html',  
  styleUrls: ['./reporte-peliculas.component.css']  
})  
export class ReportePeliculasComponent implements OnInit {  
  películas: any[] = [];  
  
  constructor(private http: HttpClient) { }  
  
  ngOnInit() {  
    this.http.get<any[]>('./assets/peliculas.json').subscribe(data => {  
      this.películas = data;  
    });  
  }  
}
```

**Paso 5: Mostrar los datos en el template**

Abre el archivo "reporte-películas.component.html" ubicado en la carpeta del componente.

Utiliza la directiva `*ngFor` para iterar sobre la lista de películas y mostrar sus propiedades:

```
<table>  
  <tr>  
    <th>Título</th>  
    <th>Género</th>  
    <th>Año de lanzamiento</th>  
  </tr>  
  <tr *ngFor="let pelicula of películas">  
    <td>{{ pelicula.titulo }}</td>
```

```
<td>{{ pelicula.genero }}</td>
<td>{{ pelicula.lanzamiento }}</td>
</tr>
</table>
```

#### Paso 6: Agregar estilos

Aplicar estilos al informe, puedes agregar CSS al archivo "reporte-peliculas.component.css" ubicado en la carpeta del

```
table {
  width: 100%;
  border-collapse: collapse;
}

th, td {
  padding: 8px;
  text-align: left;
}

tr:nth-child(even) {
  background-color: #f2f2f2;
}

tr:hover {
  background-color: #ddd;
}
```

#### Paso 7: Agregar el componente al módulo principal

Abre el archivo "app.module.ts" ubicado en la carpeta raíz del proyecto.

Importa el módulo HttpClientModule agregando la siguiente línea al comienzo del archivo:

```
import { HttpClientModule } from '@angular/common/http';
```

Agrega HttpClientModule al arreglo de imports en el decorador @NgModule:

```
imports: [
  // ...
  HttpClientModule
],
```

#### Paso 8: Mostrar el informe en la aplicación

Abre el archivo "app.component.html" ubicado en la carpeta raíz del proyecto.

Reemplaza el contenido existente con el siguiente código para mostrar el componente del informe de películas:

```
<h1>Informe de Películas</h1>
<app-reporte-peliculas></app-reporte-peliculas>
```

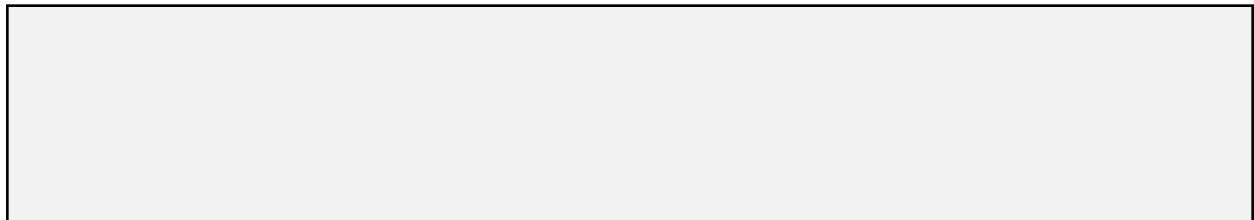
#### Paso 9: Ejecutar la aplicación

En la terminal, ejecuta el siguiente comando para iniciar la aplicación:

```
ng serve
```

Abre un navegador web y navega a la dirección <http://localhost:4200>. Deberías ver el informe de películas con los datos mostrados en la tabla.

Mostrar una captura de la ejecución



Para exportar el informe de películas a un archivo PDF en Angular, se utilizará la librería pdfmake. A continuación, los pasos para integrar pdfmake y generar el informe en formato PDF.

#### Paso 1: Instalar pdfmake

En tu proyecto de Angular, abre una terminal y ejecuta el siguiente comando para instalar pdfmake:

```
npm install pdfmake --save
npm install @types/pdfmake --save-dev
```

#### Paso 2: Importar las dependencias necesarias

Abre el archivo "reporte-peliculas.component.ts", modificar de la siguiente manera:

```
import { Component, OnInit } from '@angular/core';
import { HttpClient } from '@angular/common/http';
```

```
import * as pdfMake from 'pdfmake/build/pdfmake';
import * as pdfFonts from 'pdfmake/build/vfs_fonts';

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

  constructor(private http: HttpClient) {
    (<any>pdfMake).vfs = pdfFonts.pdfMake.vfs;
  }

  ngOnInit() {
    this.http.get<any[]>('./assets/peliculas.json').subscribe(data => {
      this.peliculas = data;
    });
  }

  generarPDF() {
    const contenido = [
      { text: 'Informe de Películas', style: 'header' },
      { text: '\n\n' },
      {
        table: {
          headerRows: 1,
          widths: ['*', '*', '*'],
          body: [
            ['Título', 'Género', 'Año de lanzamiento'],
            ...this.peliculas.map(pelicula => [pelicula.titulo, pelicula.genero, pelicula.lanzamiento.toString()])
          ]
        }
      }
    ];

    const estilos = {
      header: {
        fontSize: 18,
        bold: true
      }
    };

    const documentDefinition = {
      content: contenido,
      styles: estilos
    };
  }
}
```

```
pdfMake.createPdf(documentDefinition).open();  
}  
}
```

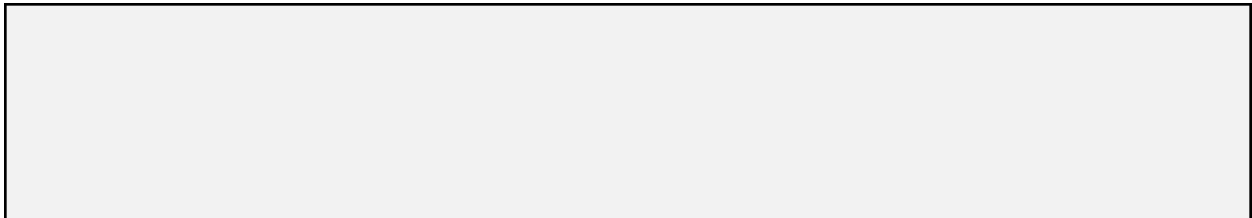
Paso 4: Agregar un botón para generar el PDF

Abre el archivo "reporte-peliculas.component.html".

Agrega un botón en el lugar deseado para permitir al usuario generar el archivo PDF:

```
<button (click)="generarPDF()">Generar PDF</button>
```

Mostrar una captura de la ejecución del PDF



## TAREA

- Mejorar la presentación del informe: Agrega estilos adicionales al informe PDF, como colores, fuentes personalizadas o tamaños de texto distintos. Experimenta con la configuración de estilos en la función generarPDF() para lograr un diseño más atractivo.
- Filtrar películas: Agrega funcionalidad para permitir al usuario filtrar las películas según diferentes criterios, como género o año de lanzamiento. Actualiza el informe PDF para mostrar solo las películas que coincidan con los filtros seleccionados.
- Permitir exportación a otros formatos: Además del formato PDF, agrega la posibilidad de exportar el informe a otros formatos, como Excel o CSV. Investiga e implementa las librerías adecuadas para generar estos formatos de archivo.

**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**



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

- \_\_\_\_\_  
\_\_\_\_\_
- \_\_\_\_\_  
\_\_\_\_\_
- \_\_\_\_\_  
\_\_\_\_\_
- \_\_\_\_\_  
\_\_\_\_\_
- \_\_\_\_\_  
\_\_\_\_\_

**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)*

- \_\_\_\_\_  
\_\_\_\_\_
- \_\_\_\_\_  
\_\_\_\_\_
- \_\_\_\_\_  
\_\_\_\_\_
- \_\_\_\_\_  
\_\_\_\_\_
- \_\_\_\_\_  
\_\_\_\_\_