Laboratorio 6

Sebastián Cardona, Laura Gil, Zayra Gutiérrez

Ingeniero de Sistemas

Javier Toquica Barrera

Universidad Escuela Colombiana de ingeniería Julio Garavito

2025-1

Contenido

Introducción 3

Desarrollo del Laboratorio 4

Front-End – Vistas 4

Front-End – Lógica 7

Conclusiones 11

Introducción

El presente informe describe el desarrollo y los resultados del Laboratorio 6, cuyo objetivo principal fue la creación de una aplicación web utilizando React y SpringBoot, con el fin de implementar funcionalidades interactivas para la visualización y manipulación de planos asociados a autores. A lo largo del laboratorio, se trabajó en el desarrollo del Front-End, específicamente en la creación de vistas y la implementación de la lógica que permite la actualización dinámica de la interfaz en función de las acciones del usuario.

Implementación del laboratorio en el repositorio: https://github.com/LaaSofiaa/ARSW-Lab6

Desarrollo del Laboratorio

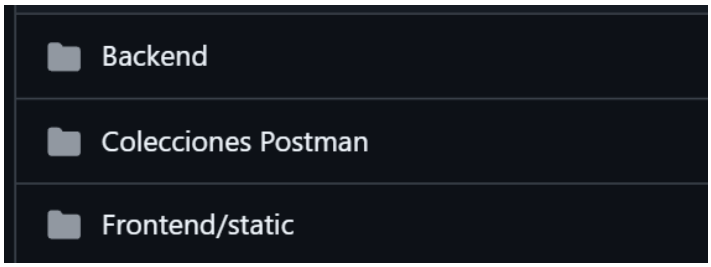
Front-End – Vistas

1. Cree el directorio donde residirá la aplicación JavaScript. Como se está usando SpringBoot, la ruta para poner en el mismo contenido estático.

Para este caso el laboratorio fue realizado con React, para crear el proyecto React utilizamos el comando

“npm create vite@latest static --template react”

Por lo tanto, se creó un directorio que se encargaría de contener todo el frontend.



2. Cree, en el directorio anterior, la página index.html, sólo con lo básico: título, campo para la captura del autor, botón de 'Get blueprints', campo donde se mostrará el nombre del autor seleccionado, la tabla HTML donde se mostrará el listado de planos (con sólo los encabezados), y un campo donde se mostrará el total de puntos de los planos del autor. Recuerde asociarle identificadores a dichos componentes para facilitar su búsqueda mediante selectores.

```
static > index.html > ...
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <link rel="icon" type="image/svg+xml" href="/icono.png" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Blueprints</title>
  </head>
  <body>
    <div id="root"></div>
    <script type="module" src="/src/main.tsx"></script>
  </body>
</html>
```

Index.html

```
static > src > main.tsx
import { StrictMode } from 'react'
import { createRoot } from 'react-dom/client'
import './index.css'
import App from './App.tsx'

createRoot(document.getElementById('root')!).render(
  <StrictMode>
    <App />
  </StrictMode>,
)
```

main.tsx

```
import Blueprints from "./components/Blueprints";
import "./App.css";

function App() {
  return (
    <div>
      <Blueprints />
    </div>
  );
}

export default App;
```

app.tsx

static > src > components > Blueprints.tsx > Blueprints

```jsx
const Blueprints = () => {
  return (
    <div>
      <h1>Gestión de Blueprints</h1>

      {/* Campo para capturar el autor */}
      <input
        type="text"
        placeholder="Ingrese el nombre del autor"
        value={author}
        onChange={(e) => setAuthor(e.target.value)}
      />
      <button onClick={handleGetBlueprints}>Get Blueprints</button>

      {/* Mostrar error si existe */}
      {error && <p style={{ color: "red" }}>{error}</p>}
```

Blueprints.tsx

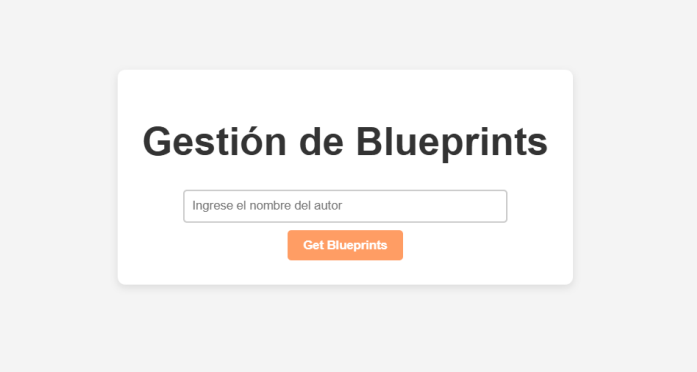
3. Suba la aplicación (mvn spring-boot:run), y rectifique:

   3.1 Que la página es accesible desde: http://localhost:5173/

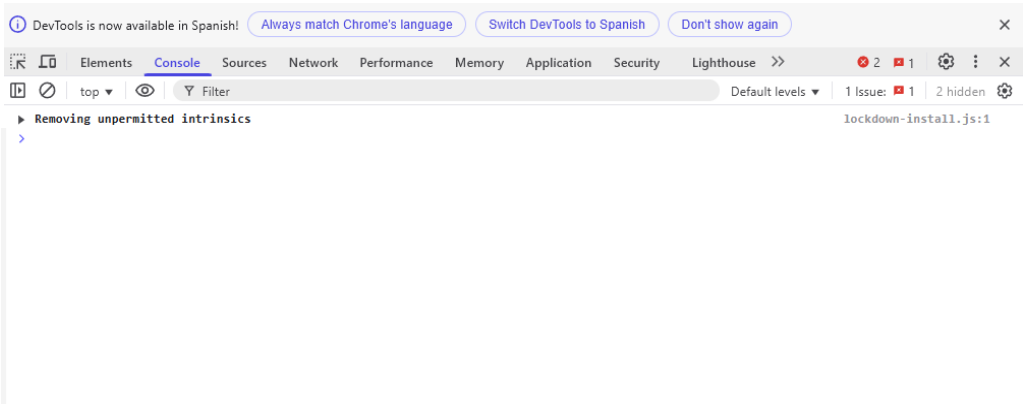
   Comando para ejecutar el proyecto

   “npm install”

   “npm run dev”



   3.2 Al abrir la consola de desarrollador del navegador, NO deben aparecer mensajes de error 404 (es decir, que las librerías de JavaScript se cargaron correctamente).



Front-End – Lógica

1. Agregue al módulo 'app.js' una operación pública que permita actualizar el listado de los planos, a partir del nombre de su autor (dado como parámetro). Para hacer esto, dicha operación debe invocar la operación 'getBlueprintsByAuthor' del módulo 'apimock' provisto, enviándole como callback una función que:

1.1 Tome el listado de los planos, y le aplique una función 'map' que convierta sus elementos a objetos con sólo el nombre y el número de puntos.

1.2 Sobre el listado resultante, haga otro 'map', que tome cada uno de estos elementos, y a través de jQuery agregue un elemento <tr> (con los respectvos <td>) a la tabla creada en el punto 4. Tenga en cuenta los selectores de jQuery y los tutoriales disponibles en línea. Por ahora no agregue botones a las filas generadas.

```jsx
{/* Mostrar nombre del autor seleccionado */}
{blueprints.length > 0 && (
  <>
    <h2>Autor: {author}</h2>

    <table border={1}>
      <thead>
        <tr>
          <th>Nombre</th>
          <th>Puntos</th>
          <th>Plano</th>
        </tr>
      </thead>
      <tbody>
        {blueprints.map((bp) => (
          <tr key={bp.name}>
            <td>{bp.name}</td>
            <td>{bp.points.length}</td>
            <td>
              <button onClick={() => openModal(bp)}>Ver</button>
            </td>
          </tr>
        ))}
      </tbody>
    </table>

    <h3>Total de puntos: {totalPoints}</h3>
  </>
)}
```

1.3 Sobre cualquiera de los dos listados (el original, o el transformado mediante 'map'), aplique un 'reduce' que calcule el número de puntos. Con este valor, use jQuery para actualizar el campo correspondiente dentro del DOM.

```jsx
const total = data.reduce(
  (sum: number, blueprint: Blueprint) =>
    sum + (blueprint.points?.length || 0),
  0
);
setTotalPoints(total);
```

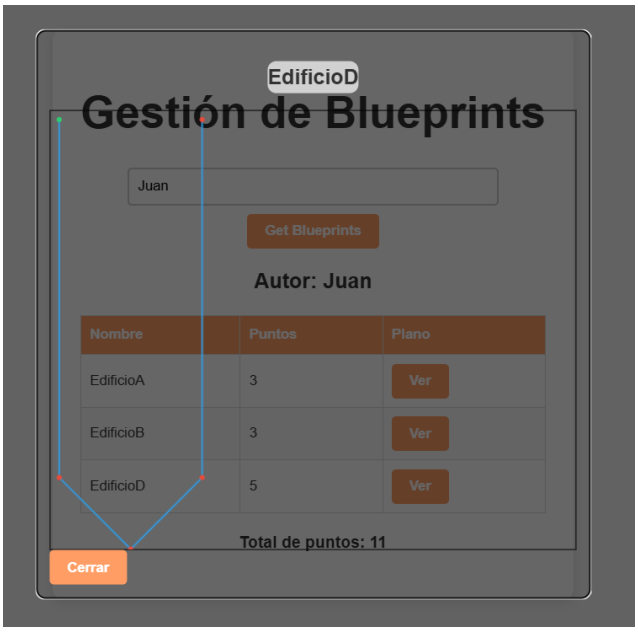
2. Verifique el funcionamiento de la aplicación. Inicie el servidor, abra la aplicación HTML5/JavaScript, y rectifique que, al ingresar un usuario existente, se cargue el listado del mismo.

Comando para prender el back

“mvn spring-boot:run”

Funcionalidad ver plano

Se implementó modal para mostrar una pantalla sobre la existente.



Conclusiones

Se logró una correcta visualización de los planos según el autor seleccionado, y se implementó de manera efectiva la actualización dinámica del listado de planos y el cálculo del total de puntos.

Este laboratorio no solo permitió aplicar conceptos de desarrollo web, sino también fortalecer la comprensión de la importancia de la separación entre Front-End y Back-End, y cómo las distintas capas de una aplicación interactúan para proporcionar una experiencia de usuario fluida y eficiente.