

# Instituto Politécnico Nacional

## Escuela Superior de Cómputo

Tecnologías de Desarrollo Web.

HW1 : Reporte de Proyecto Final

*Profesor: M. en C. José Asunción Enríquez Zárate*

*Alumno: Rodríguez Ramírez Fernanda*

*Alumno: Alvarez Gijón Nayeli Rubí*

*4BM2*

January 15, 2025

# Contents

<b>1</b>	<b>Introducción</b>	<b>2</b>
1.1	Backend . . . . .	2
1.2	Frontend . . . . .	2
1.3	Sincronización de Frontend y Backend . . . . .	2
1.4	Diccionario de datos . . . . .	3
<b>2</b>	<b>Conceptos</b>	<b>4</b>
2.1	<a href="#">BackEnd</a> . . . . .	4
2.2	<a href="#">FrontEnd</a> . . . . .	4
2.3	<a href="#">Diccionario de Datos</a> . . . . .	4
2.4	<a href="#">React</a> . . . . .	4
2.5	<a href="#">Modelo Entidad - Relación</a> . . . . .	4
<b>3</b>	<b>Desarrollo</b>	<b>5</b>
3.1	<a href="#">Comienzo de la creación del proyecto</a> . . . . .	5
3.2	<a href="#">Creación de bases de datos</a> . . . . .	5
3.3	<a href="#">Factorial.java</a> . . . . .	6
3.4	<a href="#">Conexión de Bases</a> . . . . .	6
3.5	<a href="#">Entorno Virtual</a> . . . . .	6
3.6	<a href="#">Páginas principales</a> . . . . .	6
3.7	<a href="#">Creación del sistema de rutas</a> . . . . .	10
3.8	<a href="#">Configuración final del Frontend</a> . . . . .	10
3.9	<a href="#">Programa Finalizado</a> . . . . .	12
<b>4</b>	<b>Conclusión</b>	<b>14</b>
<b>5</b>	<b>Referencias Bibliográficas</b>	<b>15</b>

## List of Figures

1	main.py corriendo exitosamente . . . . .	6
2	Terminal compilada para npm . . . . .	11
3	Alumnos . . . . .	12
4	Carreras . . . . .	12
5	Materias . . . . .	13
6	Profesores . . . . .	13
7	Salones . . . . .	13

List of Tables

# 1 Introducción

En el desarrollo de aplicaciones modernas, los conceptos de BackEnd y FrontEnd representan dos pilares fundamentales para la creación de soluciones tecnológicas que sean tanto funcionales como visualmente atractivas. Cada uno de estos componentes desempeña un rol específico en la estructura y operación de una aplicación, asegurando que la experiencia del usuario y la lógica del sistema trabajen de manera coordinada.

## 1.1 Backend

Es la parte "invisible" (que no es apreciable para el usuario) de una aplicación, es la responsable de manejar la lógica del negocio, el almacenamiento y la gestión de los datos, así como la interacción con el servidor. Este componente opera detrás de escena, asegurándose de que cada solicitud realizada por el usuario sea procesada correctamente y que la información entregada sea precisa y relevante.

Un sistema de Backend incluye:

- Servidor: Gestiona las solicitudes entrantes y envía respuestas al cliente.
- Base de Datos: Almacena, organiza y recupera la información necesaria para el funcionamiento de la aplicación.
- APIs: Permiten la comunicación entre el FrontEnd y el BackEnd de manera estructurada y segura.

Para implementar un BackEnd eficiente, Spring Boot es una de las tecnologías más utilizadas. Es un framework basado en Java permite construir aplicaciones escalables. Su principal ventaja es la simplificación del proceso de configuración, que ofrece herramientas preconfiguradas que aceleran el desarrollo. Spring Boot también incluye soporte para la creación de APIs, para la conexión a bases de datos, y la administración de usuarios, etc.

## 1.2 Frontend

Por otro lado, en el FrontEnd se trabaja la parte visible de la aplicación, donde el usuario interactúa directamente. Este se encarga de presentar la información de manera clara, atractiva y funcional, asegurando que el usuario pueda realizar acciones fácilmente.

Frontend trabaja usualmente con:

- HTML: Define la estructura básica de las páginas.
- CSS: Estiliza y organiza la presentación visual.
- JavaScript: Añade interactividad y dinamismo.

Angular se destaca como un framework muy bueno para el desarrollo de aplicaciones, fue desarrollado por Google y permite construir aplicaciones de una sola página (SPA, por sus siglas en inglés) que cargan dinámicamente los contenidos sin necesidad de recargar toda la página. Utiliza componentes reutilizables, enlazado de datos bidireccional y un sistema modular, lo que simplifica la gestión del código y mejora la escalabilidad.

## 1.3 Sincronización de Frontend y Backend

Esta comunicación se realiza comúnmente mediante APIs, donde el FrontEnd envía solicitudes HTTP (GET, POST, PUT, DELETE) al BackEnd, y este responde con los datos necesarios en formato JSON o XML.

Un ejemplo de los pasos que se siguen es:

1. Un usuario solicita ver la información de su perfil.
2. El FrontEnd envía una solicitud al BackEnd.
3. El BackEnd consulta la base de datos, organiza la información y la envía de regreso al FrontEnd.
4. El FrontEnd presenta los datos en una interfaz.

## 1.4 Diccionario de datos

Un Diccionario de Datos es un elemento clave en el desarrollo de aplicaciones que implica el uso de bases de datos. Es un documento o recurso técnico que describe, organiza y documenta todos los elementos de datos utilizados en un sistema. El diccionario de datos es muy importante para garantizar la integridad, consistencia y claridad en el diseño y uso de la base de datos, facilitando tanto el desarrollo inicial como el mantenimiento del sistema. Incluye todo lo necesario, como los nombres de las tablas, los campos que contiene cada una, los tipos de datos que se almacenan y las relaciones entre las tablas. Tener un diccionario bien hecho no solo ayuda durante el desarrollo, sino que también facilita el mantenimiento y la comprensión del sistema en el futuro. Es como un mapa que te dice exactamente dónde está todo y cómo se conecta.

## 2 Conceptos

A continuación se enlistarán los conceptos clave:

### 2.1 Backend

- Definición: Parte del desarrollo que maneja la lógica del negocio, procesamiento de datos y gestión del servidor.
- Funciones principales: Validación de datos, manejo de bases de datos, autenticación, y exposición de APIs.

### 2.2 FrontEnd

- Definición: La interfaz visual con la que interactúa el usuario.
- Presentación de información, interactividad, experiencia de usuario.

### 2.3 Diccionario de Datos

- Elementos esenciales: Tablas, columnas, tipos de datos, relaciones entre tablas, restricciones.
- Importancia: Garantiza consistencia, facilita el desarrollo y simplifica el mantenimiento del sistema.
- Relación con el Modelo E-R: Representa el diseño lógico de la base de datos.

### 2.4 React

- Divide la interfaz en bloques reutilizables llamados componentes. Cada componente maneja su propia lógica y representación, lo que facilita la construcción y el mantenimiento del código.
- Una extensión de sintaxis de JavaScript que permite escribir código similar a HTML dentro de los archivos de JavaScript, combinando lógica y diseño en los componentes.
- State se utiliza para manejar datos dinámicos localmente en cada componente.
- Props son datos que se pasan de un componente padre a un componente hijo, permitiendo la comunicación y personalización de los componentes sin modificar su lógica interna.
- Flujo de datos unidireccional.

### 2.5 Modelo Entidad - Relación

- Definición: Es una representación gráfica utilizada para diseñar la estructura lógica de una base de datos.
- Ayuda a identificar y definir las entidades, atributos y relaciones necesarias para modelar un sistema o aplicación.
- Generalización: Proceso de abstraer atributos comunes de múltiples entidades en una entidad más general.
- Especialización: Proceso inverso, donde una entidad general se subdivide en entidades más específicas.
- Cardinalidad: Indica el número de ocurrencias de una entidad que puede estar asociada con otra entidad.
  - 1:1 (Uno a Uno): Cada registro de una entidad se relaciona con un único registro de otra entidad.
  - 1:N (Uno a Muchos): Un registro de una entidad se relaciona con múltiples registros de otra entidad.
  - M:N (Muchos a Muchos): Varios registros de una entidad se relacionan con varios registros de otra entidad.
- Clave primaria: Un atributo o combinación de atributos que identifica de manera única a cada registro en una entidad.
- ForeignKey: Es un atributo en una entidad que referencia la clave primaria de otra entidad, permitiendo establecer la relación entre ellas.

## 3 Desarrollo

El objetivo del proyecto es desarrollar un sistema de gestión web que integra al menos cinco entidades principales: Carreras, Alumnos, Materias, Profesores y Salones. El proyecto fue desarrollado utilizando React.js para el frontend y Flask con SQLite para el backend. El sistema debe permitir registrar, editar, visualizar y eliminar datos de cada una de estas entidades.

### 3.1 Comienzo de la creación del proyecto

Primero se tiene que configurar el entorno para trabajar con el Backend, entonces se crea el archivo main.py para poder configurar la API con Flask.

```
1 from flask import Flask, request, jsonify
2 import json
3 import sqlite3
4 from flask_cors import CORS
5 from dotenv import load_dotenv
6 import os
7
8 load_dotenv()
9
10 app = Flask(__name__)
11 CORS(app)
12
13 def db_connection():
14     conn = None
15     try:
16
17         conn = sqlite3.connect(os.getenv("DB_NAME"))
18         conn.row_factory = sqlite3.Row
19     except sqlite3.Error as e:
20         print(f"Error de conexión con la BD: {e}")
21     return conn
22
23 @app.route('/')
24 def index():
25     return jsonify({"message": "Todo bien API iniciada:"}), 200
```

*Código:* Creación de main.py

### 3.2 Creación de bases de datos

En este caso, al utilizar el sistema operativo 'macOS', se emplea SQLite, que viene preinstalado de manera nativa en el sistema. Para iniciar el programa, basta con ejecutar el comando sqlite3 en la terminal. Es necesario acceder a la carpeta donde se encuentra el Backend del proyecto y, desde allí, crear la base de datos con el nombre deseado. En este mismo lugar, se deben agregar las tablas que serán utilizadas, asegurándose de que estén previamente definidas en la carpeta del Backend.

```
1 nayelialvarez@MacBook-Pro-de-Nayeli ~ % sqlite3
2 SQLite version 3.43.2 2023-10-10 13:08:14
3 Enter ".help" for usage hints.
4 Connected to a transient in-memory database.
5 Use ".open FILENAME" to reopen on a persistent database.
6 sqlite> .open /Users/nayelialvarez/Downloads/DaniProyecto/PROYECTO.WEB/Backend/materias.db
7 sqlite> SELECT * FROM materias;
8 1|Introduccion a la IA|IA101|1|1
9 3|Redes Neuronales|IA202|3|1
10 4|Procesamiento de Lenguaje Natural|IA303|5|1
11 5|Estadística Descriptiva|CD101|1|2
12 6|Base de Datos para Big Data|CD202|3|2
13 7|Minería de Datos|CD303|5|2
14 8|Visualización de Datos|CD404|7|2
15 10|Instrumentación Médica|BM202|3|3
```

*Código:* Ejemplo de Base de Datos



### 3.3 *Factorial.java*

### 3.4 Conexión de Bases

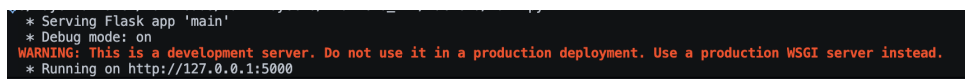
El archivo .env (abreviatura de environment) es un archivo de configuración que se utiliza para definir variables de entorno necesarias para que la aplicación funcione correctamente. En este se tiene que modificar para que se pueda acceder para permitir cambiar configuraciones.

```
1 DB_HOST=  
2 DB_USER=  
3 DB_PASSWORD=  
4 DB_NAME= /Users/nayelialvarez/Downloads/DaniProyecto/PROYECTO.WEB/Backend/materias.db
```

*Algoritmo:* Archivo .env

### 3.5 Entorno Virtual

El siguiente paso es activar el entorno virtual, y se instala **requirements.txt**, este archivo contiene una lista de todas las bibliotecas y versiones específicas que el proyecto necesita. Se debe también verificar en la terminal que el main.py funcione correctamente.



```
* Serving Flask app 'main'  
* Debug mode: on  
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.  
* Running on http://127.0.0.1:5000
```

Figure 1: main.py corriendo exitosamente

### 3.6 Páginas principales

Para el proceso del Frontend se tienen que definir las páginas para las entidades que se quieren.

Por ejemplo para el archivo AppCarreras.jsx, el cuál es un componente de React que sirve para gestionar las carreras dentro del proyecto. Al comenzar, es importante importar las dependencias necesarias como React, axios y algunos íconos para las acciones visuales. React se utiliza para estructurar y gestionar el comportamiento dinámico de la página. Axios, por su parte, permite realizar las peticiones HTTP necesarias para interactuar con el backend. Estas peticiones permiten cargar las carreras desde la base de datos, enviarlas cuando se crean nuevas o actualizan, y eliminarlas cuando es necesario.

El estado del componente es fundamental para manejar los datos. Por ejemplo, se utiliza useState para mantener la lista de carreras, un indicador para mostrar o no el formulario y una estructura para almacenar los datos de una carrera cuando se edita o se crea una nueva. Este estado es dinámico y cambia según las acciones del usuario, como hacer clic en "Nueva Carrera" o en el botón para editar.

El componente incluye funciones clave que interactúan con el backend. Una de ellas se encarga de obtener las carreras a través de una petición GET, otra guarda o actualiza una carrera dependiendo de si ya existe o es nueva, y otra elimina una carrera enviando una petición DELETE al servidor. Estas funciones aseguran que los datos en la interfaz y en el backend estén sincronizados.

La interfaz de usuario se construye utilizando JSX, que permite definir el diseño de la página y los elementos visuales. La estructura incluye un título principal, una tabla para mostrar las carreras y un botón para agregar una nueva. También hay un formulario que aparece dinámicamente para capturar la información de una carrera cuando el usuario decide crear o editar una. El diseño utiliza clases de Tailwind CSS para proporcionar un estilo visual moderno y profesional, haciendo que la página sea agradable a la vista y fácil de usar.

El formulario para agregar o editar una carrera es un elemento clave. Incluye campos para ingresar el nombre, la descripción y el número de semestres de la carrera. Cuando el formulario se envía, se llama a la función correspondiente para guardar los datos en el backend. Si el usuario decide cancelar, el formulario se cierra sin realizar ningún cambio.

Finalmente, todo el componente se encapsula y se exporta para que pueda integrarse fácilmente en el sistema de rutas del proyecto. Este archivo no solo gestiona la lógica de las carreras, sino que también asegura que la interacción del usuario con la aplicación sea fluida y eficiente. Es un ejemplo claro de cómo React y herramientas adicionales pueden trabajar juntas para crear una experiencia de usuario dinámica y bien estructurada.

```
1 import React, { useState, useEffect } from "react";  
2 import { AlertCircle, Plus, Edit2, Trash2, ArrowLeft } from "lucide-react";  
3 import axios from "axios";  
4 import { Link } from "react-router-dom";  
5 import { PenTool } from "lucide-react";  
6
```

```

7
8  const API_URL = "http://127.0.0.1:5000/carreras";
9
10 const AppCarreras = () => {
11   const [carreras, setCarreras] = useState([]);
12   const [error, setError] = useState(null);
13   const [showCarreraForm, setShowCarreraForm] = useState(false);
14   const [selectedCarrera, setSelectedCarrera] = useState(null);
15
16   const [newCarrera, setNewCarrera] = useState({
17     carrera: "",
18     descripcionCarrera: "",
19     semestres: "",
20     plan: "",
21   });
22
23
24   const fetchCarreras = async () => {
25     try {
26       const response = await axios.get(API_URL);
27       setCarreras(response.data);
28     } catch (err) {
29       setError("error al cargar las carreras");
30     }
31   };
32
33   useEffect(() => {
34     fetchCarreras();
35   }, []);
36
37
38   const handleSaveCarrera = async () => {
39     try {
40       if (selectedCarrera) {
41
42         await axios.put(`${API_URL}/${selectedCarrera.idCarrera}`, newCarrera);
43       } else {
44
45         await axios.post(API_URL, newCarrera);
46       }
47       fetchCarreras();
48       setShowCarreraForm(false);
49       setSelectedCarrera(null);
50       setNewCarrera({ carrera: "", descripcionCarrera: "", semestres: "", plan: "" });
51     } catch (err) {
52       setError("error al guardar la carrera");
53     }
54   };
55
56
57   const handleDeleteCarrera = async (idCarrera) => {
58     try {
59       await axios.delete(`${API_URL}/${idCarrera}`);
60       fetchCarreras();
61     } catch (err) {
62       setError("error al eliminar la carrera");
63     }
64   };
65
66
67   const handleEditCarrera = (carrera) => {
68     setSelectedCarrera(carrera);
69     setNewCarrera({
70       carrera: carrera.carrera,
71       descripcionCarrera: carrera.descripcionCarrera,
72       semestres: carrera.semestres,

```

```

73     plan: carrera.plan,
74   });
75   setShowCarreraForm(true);
76 };
77
78 return (
79   <div className="bg-gradient-to-br from-purple-200 via-pink-200 to-red-200 min-h-screen p-8">
80     <header className="text-center mb-8">
81       <h1
82         className="text-4xl font-extrabold text-black-600 drop-shadow-lg"
83         style={{ fontFamily: "'Amatic SC', cursive" }}
84       >
85         <h1 className="text-[60px] font-extrabold drop-shadow-lg">
86           <span className="text-black">
87             Carreras
88           </span>
89         </h1>
90       </h1>
91       <p className="text-sm text-gray-600 mt-2">Modifica carreras disponibles</p>
92     </header>
93
94     {error && (
95       <div className="bg-red-100 text-red-600 p-4 rounded shadow mb-4">
96         <AlertCircle className="inline-block mr-2" />
97         {error}
98       </div>
99     )}
100
101     <div className="bg-white shadow-lg rounded-lg p-6 mb-8">
102       <h2 className="text-2xl font-semibold text-gray-800 mb-4">Lista de Carreras</h2>
103       <table className="w-full table-auto border-collapse border border-gray-300">
104         <thead className="bg-pink-200 text-gray-800">
105           <tr>
106             <th className="p-4 text-left">Carrera</th>
107             <th className="p-4 text-left">Descripci n </th>
108             <th className="p-4 text-left">Semestres</th>
109             <th className="p-4 text-left">Plan</th>
110             <th className="p-4 text-left">Acciones</th>
111           </tr>
112         </thead>
113         <tbody>
114           {carreras.map((carrera) => (
115             <tr key={carrera.idCarrera} className="border-b hover:bg-blue-100 transition">
116               <td className="p-4">{carrera.carrera}</td>
117               <td className="p-4">{carrera.descripcionCarrera}</td>
118               <td className="p-4">{carrera.semestres}</td>
119               <td className="p-4">{carrera.plan}</td>
120               <td className="p-4 flex gap-4">
121                 <button onClick={() => handleEditCarrera(carrera)} className="text-pink-600 hov
122                   <PenTool /><p>Editar</p>
123                 </button>
124                 <button
125                   onClick={() => handleDeleteCarrera(carrera.idCarrera)}
126                   className="text-black-900 hover:underline"
127                 >
128                   <Trash2 /><p>Eliminar</p>
129                 </button>
130               </td>
131             </tr>
132           ))}
133         </tbody>
134       </table>
135     </div>
136
137     <div className="flex justify-center">
138

```

```

139     <button
140       onClick={() => setShowCarreraForm(true)}
141       className="bg-pink-400 text-gray-700 px-6 py-3 rounded-lg shadow-md hover:bg-pink-400 t
142     >
143       Nueva Carrera
144     </button>
145   </div>
146
147   {showCarreraForm && (
148     <div className="fixed inset-0 bg-gray-800 bg-opacity-50 flex justify-center items-center">
149       <div className="bg-white p-6 rounded-lg shadow-lg w-1/3">
150         <h3 className="text-2xl font-bold mb-4">{selectedCarrera ? "Editar Carrera" : "Nueva Carrera"}
151         <input
152           type="text"
153           placeholder="Nombre de la carrera"
154           value={newCarrera.carrera}
155           onChange={(e) => setNewCarrera({ ...newCarrera, carrera: e.target.value })}
156           className="w-full p-2 border border-gray-300 rounded mb-4"
157         />
158         <textarea
159           placeholder="Descripción"
160           value={newCarrera.descripcionCarrera}
161           onChange={(e) => setNewCarrera({ ...newCarrera, descripcionCarrera: e.target.value })}
162           className="w-full p-2 border border-gray-300 rounded mb-4"
163         />
164         <input
165           type="number"
166           placeholder="Semestres"
167           value={newCarrera.semestres}
168           onChange={(e) => setNewCarrera({ ...newCarrera, semestres: e.target.value })}
169           className="w-full p-2 border border-gray-300 rounded mb-4"
170         />
171         <input
172           type="text"
173           placeholder="Plan"
174           value={newCarrera.plan}
175           onChange={(e) => setNewCarrera({ ...newCarrera, plan: e.target.value })}
176           className="w-full p-2 border border-gray-300 rounded mb-4"
177         />
178         <div className="flex justify-end gap-4">
179           <button onClick={handleSaveCarrera} className="bg-pink-400 text-gray-700 px-6 py-3 rounded">
180             Guardar
181           </button>
182           <button
183             onClick={() => {
184               setShowCarreraForm(false);
185               setSelectedCarrera(null);
186             }}
187             className="bg-gray-500 text-white px-4 py-2 rounded hover:bg-gray-600"
188           >
189             Cancelar
190           </button>
191         </div>
192       </div>
193     </div>
194   )}
195   <footer className="text-center mt-8">
196     <Link to="/" className="text-pink-900 hover:underline text-lg">
197       <ArrowLeft className="inline-block mr-2" /> Volver a alumnos
198     </Link>
199   </footer>
200 </div>
201 );
202 };
203
204 export default AppCarreras;

```

Este mismo proceso se tiene que repetir con cada una de las entidades en la base de datos.

### 3.7 Creación del sistema de rutas

Se tiene que ejecutar en la terminal donde se encuentra la carpeta Frontend el comando `npm install react-router-dom` el cuál dará un archivo llamado "RouterConfig.jsx" donde se tienen que modificar las rutas de cada App creada para las entidades. Se tienen que crear los Endpoints básicos y asegurar la conexión con la base de datos para la extracción de estos.

---

```

1
2 import Alumnos from "../src/pages/AppAlumnos";
3 import Carreras from "../src/pages/AppCarreras";
4 import Materias from "../src/pages/AppMaterias";
5 import Profesores from "../src/pages/AppProfesores";
6 import Salones from "../src/pages/AppSalones";
7
8 const routes = [
9   {
10     path: '/',
11     component: (
12       <
13         <Alumnos />
14       </>
15     ),
16   },
17   {
18     path: '/Carreras',
19     component: (
20       <
21         <Carreras />
22       </>
23     ),
24   },
25   {
26     path: '/Materias',
27     component: (
28       <
29         <Materias />
30       </>
31     ),
32   },
33   {
34     path: '/Profesores',
35     component: (
36       <
37         <Profesores />
38       </>
39     ),
40   },
41   {
42     path: '/salones',
43     component: <Salones />,
44   }
45 ];
46
47
48 export default routes;

```

---

*Algoritmo:* Código de RouterConfig.jsx

### 3.8 Configuración final del Frontend

Se abre una terminal en la ubicación de la carpeta Frontend para asegurar que todos los comandos ejecutados se apliquen específicamente a esa parte del proyecto. El comando `npm install` se utiliza para instalar todas las dependencias necesarias que están listadas en el archivo package.json. Este archivo actúa como un registro

de todas las bibliotecas y herramientas que el frontend requiere para funcionar. Cuando ejecutas este comando, el sistema descarga e instala esas dependencias en una carpeta llamada `node_modules`. Esta carpeta contiene todo el código y las librerías externas que utiliza el proyecto. Sin estas dependencias, el Frontend no podría ejecutarse correctamente, ya que faltaría el soporte de las bibliotecas esenciales como React.

Una vez que las dependencias están instaladas, el siguiente paso es ejecutar el comando `npm run dev`. Este comando inicia un servidor de desarrollo para tu aplicación frontend. Este servidor es proporcionado por herramientas como Vite, que es rápida y optimizada para el desarrollo moderno de aplicaciones web. Cuando se ejecuta, el servidor compila y sirve tu aplicación, permitiéndote verla en el navegador en tiempo real en la dirección `http://localhost:5173`. Esto significa que ya se puede ver y probar los cambios en el código de inmediato, sin necesidad de procesos de compilación manuales.

```
virtual_envnayeliavarez@MacBook-Pro-de-Nayeli frontend % npm install
npm warn ERESOLVE overriding peer dependency
npm warn While resolving: @typescript-eslint/utils@5.62.0
npm warn Found: eslint@9.18.0
npm warn   node_modules/eslint
npm warn   dev eslint@"^9.9.0" from the root project
npm warn   9 more (@babel/eslint-parser, ...)
npm warn
npm warn Could not resolve dependency:
npm warn peer eslint@"^6.0.0 || ^7.0.0 || ^8.0.0" from @typescript-eslint/utils@5.62.0
npm warn   node_modules/@typescript-eslint/type-utils/node_modules/@typescript-eslint/utils
npm warn   @typescript-eslint/utils@"5.62.0" from @typescript-eslint/type-utils@5.62.0
npm warn   node_modules/@typescript-eslint/type-utils
npm warn
npm warn Conflicting peer dependency: eslint@8.57.1
npm warn   node_modules/eslint
npm warn peer eslint@"^6.0.0 || ^7.0.0 || ^8.0.0" from @typescript-eslint/utils@5.62.0
npm warn   node_modules/@typescript-eslint/type-utils/node_modules/@typescript-eslint/utils
npm warn   @typescript-eslint/utils@"5.62.0" from @typescript-eslint/type-utils@5.62.0
npm warn   node_modules/@typescript-eslint/type-utils

up to date, audited 1405 packages in 2m

277 packages are looking for funding
  run `npm fund` for details

8 vulnerabilities (2 moderate, 6 high)

To address all issues (including breaking changes), run:
  npm audit fix --force

Run `npm audit` for details.
virtual_envnayeliavarez@MacBook-Pro-de-Nayeli frontend % npm run dev
> sayproject@0.0.0 dev
> vite

VITE v5.4.11 ready in 160 ms
  → Local:   http://localhost:5173/
  → Network: use --host to expose
  → press h + enter to show help
```

Figure 2: Terminal compilada para npm

### 3.9 Programa Finalizado

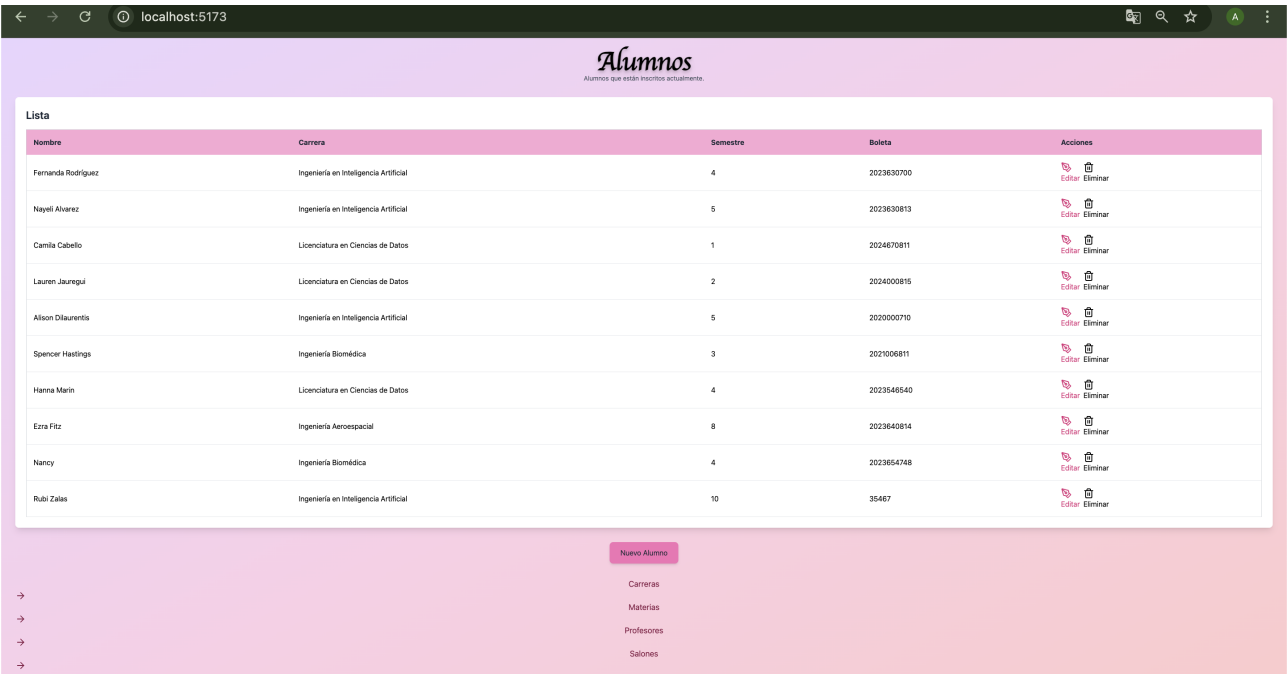


Figure 3: Alumnos

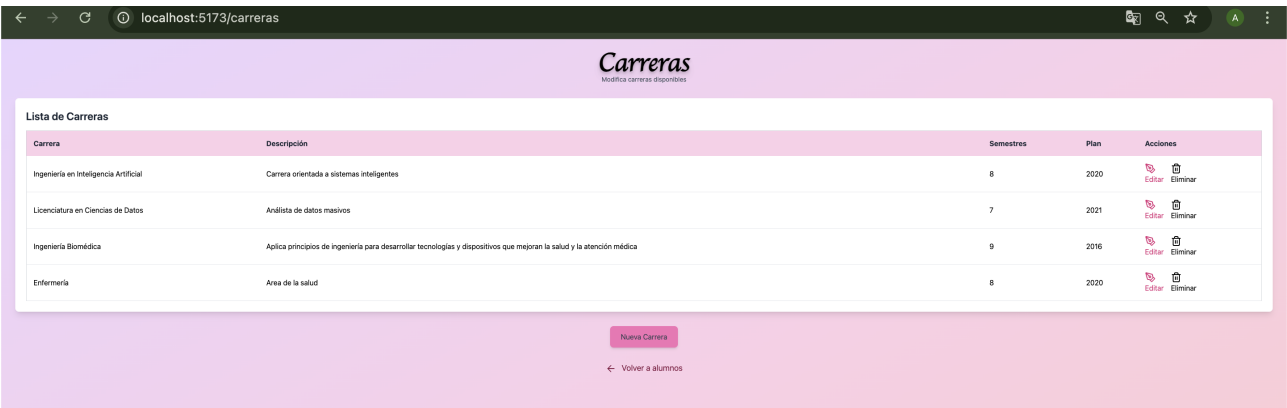


Figure 4: Carreras

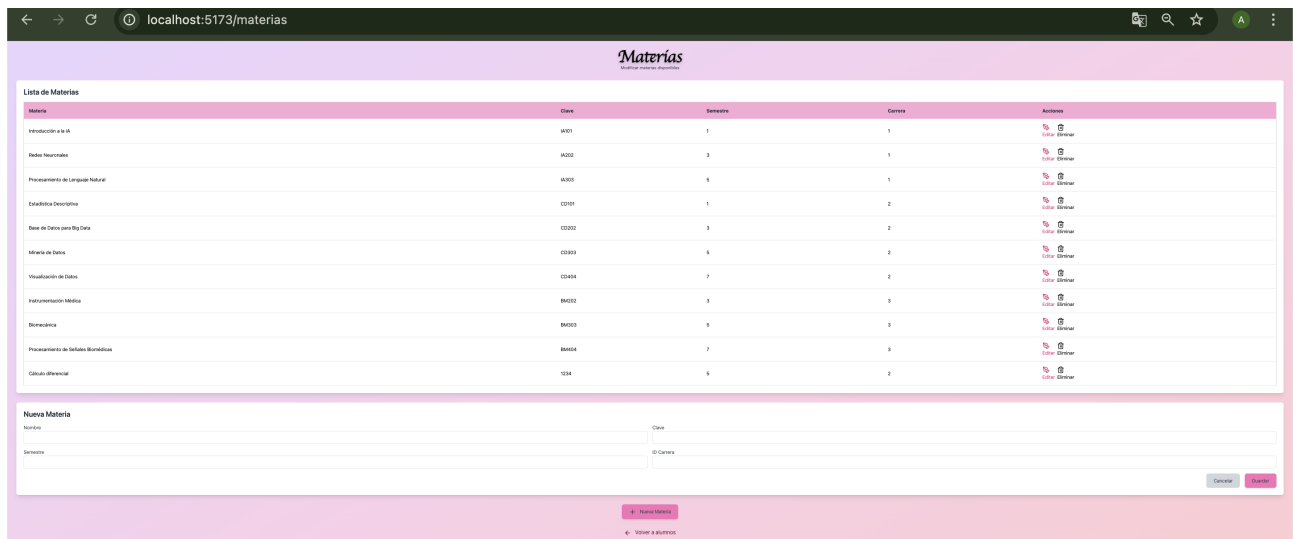


Figure 5: Materias

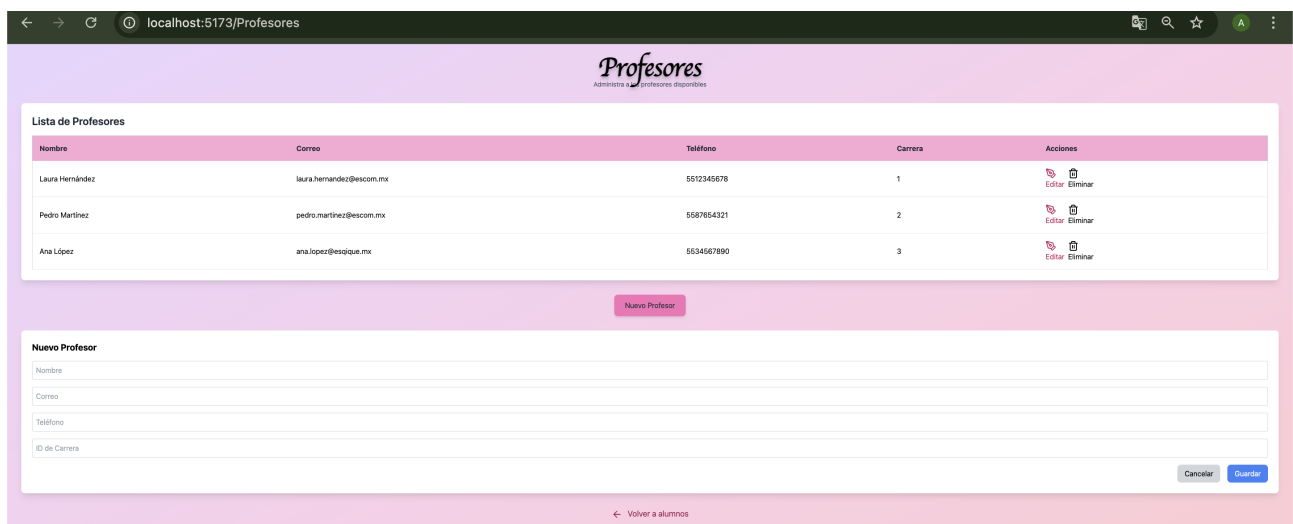


Figure 6: Profesores

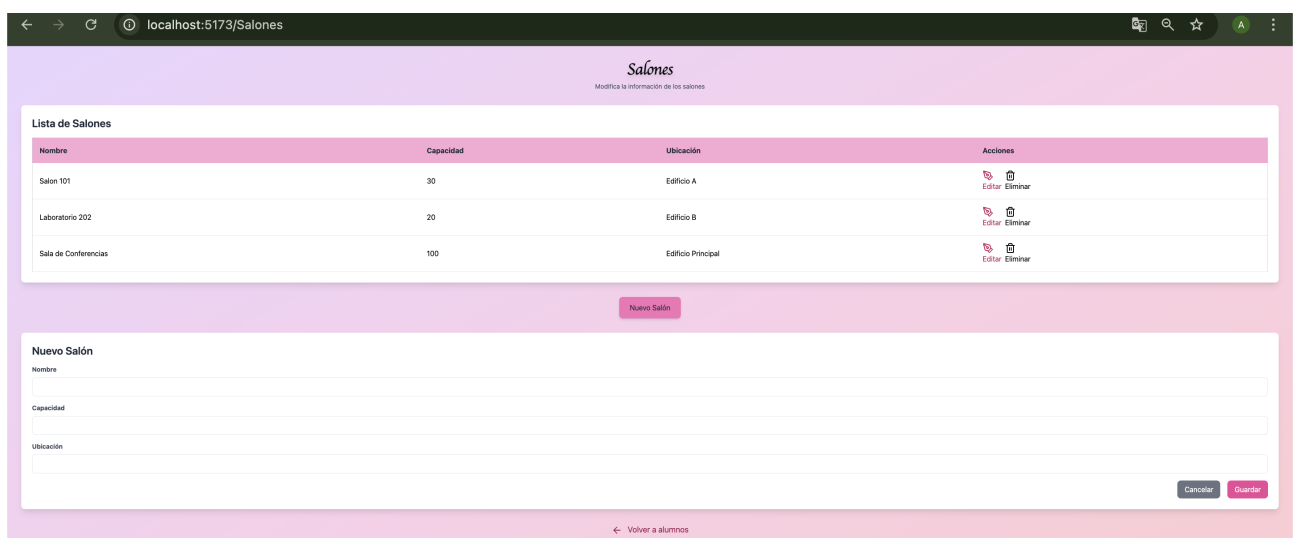


Figure 7: Salones



## 4 Conclusión

Trabajar con React para el frontend, nos permitió crear una interfaz visual muy fácil de usar. En el backend utilizamos Flask junto con SQLite, lo que hizo posible manejar los datos de manera eficiente y estructurada. Desde la creación de las tablas en la base de datos hasta la implementación de las operaciones CRUD (Crear, Leer, Actualizar y Eliminar), cada paso fue reto ya que no conocíamos a profundidad sobre cómo conectar y sincronizar las diferentes partes de una aplicación y con este proyecto lo comprendimos.

Además, aprendimos la importancia de las herramientas de desarrollo como los entornos virtuales, la instalación de dependencias y el uso de comandos como `npm run dev` para correr el servidor. Esto nos dio una visión más clara de cómo funciona un proyecto desde su configuración inicial hasta su ejecución final. También me enfrenté a errores y desafíos que me ayudaron a desarrollar habilidades para solucionarlos de manera más eficiente.

## 5 Referencias Bibliográficas

### References

- [Oracle, 2018] Oracle. *Web Component Development With Servlet and JSP Technologies*. Oracle, 2018.
- [Devlin, 2016] Edgar Martinez, Tom McGinn, Eduardo Moranchel, Anjana Shenoy, Michael Williams. *Java EE 7: Back-end Server Application Development*. Oracle, 2016.
- [Jendrock, 2017] Eric Jendrock, Ricardo Cervera-Navarro, Ian Evans, Kim Haase, William Markito. *Java Platform, Enterprise Edition (Java EE) 8 The Java EE Tutorial*. Oracle, 2017.
- [Fain, 2020] Yakov Fain. *Angular Development with TypeScript*. Manning Publications, 2020.
- [Flanagan, 2020] David Flanagan. *JavaScript: The Definitive Guide*. O'Reilly Media, 2020.
- [Sedlacek, 2021] Nick Sedlacek, Cory Rylan. *Full-Stack React Projects*. Packt Publishing, 2021.
- [Frain, 2019] Benjamin Frain. *Responsive Web Design with HTML5 and CSS*. Packt Publishing, 2019.
- [Tan, 2022] Dan Abramov, Andrew Clark, and Dominic Tancredi. *React: Up and Running*. O'Reilly Media, 2022.
- [Duckett, 2014] Jon Duckett. *HTML and CSS: Design and Build Websites*. Wiley, 2014.