

|

**GUÍA 3.2.2**

Conexión de Backend Spring Boot con Frontend  
React implementando un CRUD simple

Sigla	Asignatura	Experiencia de Aprendizaje
DSY1104	Desarrollo Full Stack II	EA Integración y Comunicación REST
Tiempo	Modalidad de Trabajo	Indicadores de logro
2 h	Individual	IL 4.2



**Antecedentes generales**

Esta guía tiene como objetivo enumerar las acciones necesarias para dar solución a los problemas planteados.



**Requerimientos para esta actividad**

Para el desarrollo de esta actividad deberás disponer de:

- Computador
- AWS – EC2



**Actividad**

Esta actividad consiste en enumerar las acciones necesarias para dar solución a los casos que se verán a continuación, para ello los estudiantes deberán realizar la actividad de forma individual.

Objetivo

Crear una aplicación web full stack conectando un backend Spring Boot con un frontend React, implementando operaciones CRUD para una entidad de Libro.

Requisitos previos

- Backend Spring Boot configurado (como en la guía anterior)
- Node.js y npm instalados
- Conocimientos básicos de React y JavaScript

## Parte 1: Configuración del Proyecto Frontend

### Paso 1: Crear el proyecto React

1. Abre una terminal y navega al directorio donde quieres crear tu proyecto.
2. Ejecuta el siguiente comando:

```
...  
  
npx create-react-app frontend  
...
```

3. Navega al directorio del proyecto:

```
...  
  
cd frontend  
...
```

### Paso 2: Instalar dependencias adicionales

1. Instala Axios para realizar llamadas HTTP y react-router-dom para el enrutamiento:

```
...  
  
npm install axios react-router-dom  
...
```

### Paso 3: Configurar la estructura del proyecto

1. Dentro de la carpeta `src`, crea las siguientes carpetas:

- `components`
- `services`

## Parte 2: Implementación del Frontend

### Paso 4: Crear el servicio de API

1. Crea un nuevo archivo `src/services/BookService.js`:

```
```javascript  
import axios from 'axios';  
  
const BASE_URL = 'http://localhost:8080/api/books';  
  
class BookService {  
  getAllBooks() {  
    return axios.get(BASE_URL);  
  }  
}
```

```
}

getBookById(id) {
  return axios.get(`${BASE_URL}/${id}`);
}

createBook(book) {
  return axios.post(BASE_URL, book);
}

updateBook(id, book) {
  return axios.put(`${BASE_URL}/${id}`, book);
}

deleteBook(id) {
  return axios.delete(`${BASE_URL}/${id}`);
}
}

export default new BookService();
````
```

## Paso 5: Crear componentes React

1. Crea los siguientes componentes en la carpeta `src/components`:

a. `BookList.js`:

```
````jsx
import React, { useState, useEffect } from 'react';
import { Link } from 'react-router-dom';
import BookService from '../services/BookService';

const BookList = () => {
  const [books, setBooks] = useState([]);

  useEffect(() => {
    fetchBooks();
  }, []);

  const fetchBooks = () => {
```



```
BookService.getAllBooks().then(response => {
  setBooks(response.data);
}).catch(error => {
  console.log('Error fetching books:', error);
});

};

const deleteBook = (id) => {
  BookService.deleteBook(id).then(() => {
    fetchBooks();
  }).catch(error => {
    console.log('Error deleting book:', error);
  });
};

return (
  <div>
    <h2>Book List</h2>
    <Link to="/add">Add New Book</Link>
    <table>
      <thead>
        <tr>
          <th>Title</th>
          <th>Author</th>
          <th>Actions</th>
        </tr>
      </thead>
      <tbody>
        {books.map(book => (
          <tr key={book.id}>
            <td>{book.title}</td>
            <td>{book.author}</td>
            <td>
              <Link to={` /edit/${book.id}`}>Edit</Link>
              <button onClick={() => deleteBook(book.id)}>Delete</button>
            </td>
          </tr>
        ))}
      </tbody>
    </table>
  </div>
```

```
);  
};
```

```
export default BookList;  
...
```

b. `BookForm.js`:

```
``jsx  
import React, { useState, useEffect } from 'react';  
import { useParams, useNavigate } from 'react-router-dom';  
import BookService from '../services/BookService';  
  
const BookForm = () => {  
  const [title, setTitle] = useState("");  
  const [author, setAuthor] = useState("");  
  const { id } = useParams();  
  const navigate = useNavigate();  
  
  useEffect(() => {  
    if (id) {  
      BookService.getBookById(id).then(response => {  
        setTitle(response.data.title);  
        setAuthor(response.data.author);  
      });  
    }  
  }, [id]);  
  
  const saveOrUpdateBook = (e) => {  
    e.preventDefault();  
    const book = { title, author };  
  
    if (id) {  
      BookService.updateBook(id, book).then(() => {  
        navigate('/');  
      });  
    } else {  
      BookService.createBook(book).then(() => {  
        navigate('/');  
      });  
    }  
  }  
}
```

```
};

return (
  <div>
    <h2>{id ? 'Edit Book' : 'Add Book'}</h2>
    <form onSubmit={saveOrUpdateBook}>
      <div>
        <label>Title:</label>
        <input
          type="text"
          value={title}
          onChange={(e) => setTitle(e.target.value)}
          required
        />
      </div>
      <div>
        <label>Author:</label>
        <input
          type="text"
          value={author}
          onChange={(e) => setAuthor(e.target.value)}
          required
        />
      </div>
      <button type="submit">{id ? 'Update' : 'Save'}</button>
    </form>
  </div>
);
};

export default BookForm;
`;
```

## Paso 6: Configurar el enrutamiento

### 1. Actualiza `src/App.js`:

```
``jsx
import React from 'react';
import { BrowserRouter as Router, Route, Routes } from 'react-router-dom';
import BookList from './components/BookList';
```

```
import BookForm from './components/BookForm';

function App() {
  return (
    <Router>
      <div>
        <h1>Book Management System</h1>
        <Routes>
          <Route path="/" element={<BookList />} />
          <Route path="/add" element={<BookForm />} />
          <Route path="/edit/:id" element={<BookForm />} />
        </Routes>
      </div>
    </Router>
  );
}

export default App;
````
```

## Paso 7: Configurar CORS en el backend

1. En tu proyecto Spring Boot, crea una nueva clase `CorsConfig`:

```
````java
package com.example.demo.config;

import org.springframework.context.annotation.Configuration;
import org.springframework.web.servlet.config.annotation.CorsRegistry;
import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;

@Configuration
public class CorsConfig implements WebMvcConfigurer {

    @Override
    public void addCorsMappings(CorsRegistry registry) {
        registry.addMapping("/api/**")
            .allowedOrigins("http://localhost:3000")
            .allowedMethods("GET", "POST", "PUT", "DELETE")
            .allowedHeaders("*");
    }
}
```

```
}  
...  

```

### Parte 3: Ejecución y Pruebas

#### Paso 8: Ejecutar la aplicación

1. Inicia tu backend Spring Boot.
2. En una nueva terminal, navega al directorio del frontend y ejecuta:

```
...  
  
npm start  
...
```

3. Abre un navegador y ve a `http://localhost:3000`.

#### Paso 9: Probar la aplicación

1. Usa la interfaz web para:
  - Ver la lista de libros
  - Añadir un nuevo libro
  - Editar un libro existente
  - Eliminar un libro
2. Verifica que los cambios se reflejen tanto en el frontend como en el backend.

#### Desafíos adicionales

1. Implementa paginación en la lista de libros.
2. Añade funcionalidad de búsqueda.
3. Implementa un sistema de autenticación.
4. Mejora el diseño con CSS o un framework como Material-UI.
5. Añade validación de formularios en el frontend.
6. Implementa manejo de errores y mensajes de feedback para el usuario.

¡Felicidades! Has creado una aplicación web full stack con un backend Spring Boot y un frontend React, implementando operaciones CRUD completas. Esta aplicación proporciona una base sólida que puedes seguir expandiendo y mejorando.

#### Recursos de apoyo

*Deployment.* (s/f). Create-react-app.dev. Recuperado el 9 de agosto de 2024, de <https://create-react-app.dev/docs/deployment/>