Laboratorio 6 - Desarrollo Full Stack

Julian F. Latorre

16 de septiembre de 2024

Índice

1. Introducción

Este laboratorio está diseñado para evaluar tus habilidades en el desarrollo fullstack con React, abarcando desde el manejo de datos hasta el despliegue en producción. Completa cada sección siguiendo las instrucciones proporcionadas.

2. Manejo de datos con React usando Hooks, Fetch y Axios

2.1. Objetivo

Crear una aplicación React que muestre una lista de usuarios obtenidos de una API externa, utilizando tanto Fetch como Axios.

2.2. Tareas

1. Crea un nuevo proyecto React usando Create React App.
2. Implementa un componente funcional llamado `UserList` que use el hook `useState` para manejar el estado de los usuarios.
3. Utiliza el hook `useEffect` para cargar los datos de usuarios al montar el componente.
4. Implementa dos funciones:
   - `fetchUsersWithFetch()`: Usa la API Fetch para obtener usuarios de `https://jsonplaceholder.typicode.com/users`.
   - `fetchUsersWithAxios()`: Usa Axios para obtener la misma lista de usuarios.
5. Muestra la lista de usuarios en el componente, incluyendo nombre, email y empresa.
6. Agrega botones para alternar entre Fetch y Axios para la obtención de datos.

2.3. Código de ejemplo

```
import React, { useState, useEffect } from 'react';
import axios from 'axios';

function UserList() {
  const [users, setUsers] = useState([]);
  const [useAxios, setUseAxios] = useState(false);

  useEffect(() => {
    if (useAxios) {
      fetchUsersWithAxios();
```

```
    } else {
      fetchUsersWithFetch();
    }
  }, [useAxios]);

  // Implementa fetchUsersWithFetch() y fetchUsersWithAxios() aquí

  return (
    <div>
      {/* Implementa la renderización de la lista y los botones aqu
    í */}
    </div>
  );
}
```

3. Implementación de Lógica de Negocio en aplicaciones fullstack

3.1. Objetivo

Extender la aplicación anterior para incluir funcionalidades CRUD (Crear, Leer, Actualizar, Eliminar) para los usuarios.

3.2. Tareas

1. Crea un backend simple usando Express.js que sirva como API para los usuarios.
2. Implementa endpoints para:
   - GET /users (obtener todos los usuarios)
   - POST /users (crear un nuevo usuario)
   - PUT /users/:id (actualizar un usuario existente)
   - DELETE /users/:id (eliminar un usuario)
3. En el frontend, crea formularios y funciones para:
   - Agregar un nuevo usuario
   - Editar un usuario existente
   - Eliminar un usuario
4. Implementa la lógica de negocio necesaria en el backend para validar los datos antes de realizar operaciones CRUD.
5. Asegúrate de manejar los errores tanto en el frontend como en el backend.

3.3. Código de ejemplo (Backend)

```
const express = require('express');
const app = express();
app.use(express.json());

let users = [/* ... */];

app.get('/users', (req, res) => {
  res.json(users);
});

app.post('/users', (req, res) => {
  // Implementa la lógica para crear un usuario
});

// Implementa los demás endpoints (PUT, DELETE)

app.listen(5000, () => console.log('Server running on port 5000'));
```

4. Testing y pruebas

4.1. Objetivo

Implementar pruebas unitarias para los componentes React y pruebas end-to-end (E2E) para la aplicación.

4.2. Tareas

1. Escribe pruebas unitarias para el componente `UserList` usando Jest y React Testing Library.
2. Implementa pruebas para las funciones de obtención de datos (`fetchUsersWithFetch` y `fetchUsersWithAxios`).
3. Crea pruebas E2E con Cypress que:
   - Verifiquen que la lista de usuarios se carga correctamente.
   - Prueben la funcionalidad de agregar, editar y eliminar usuarios.
   - Comprueben la navegación entre diferentes partes de la aplicación.

4.3. Código de ejemplo (Prueba unitaria)

```
import { render, screen, waitFor } from '@testing-library/react';
import UserList from './UserList';

test('renders user list', async () => {
  render(<UserList />);
  await waitFor(() => {
    expect(screen.getByText(/User List/i)).toBeInTheDocument();
```

```
    expect(screen.getAllByRole('listitem')).toHaveLength(10);
  });
});
```

4.4. Código de ejemplo (Prueba E2E con Cypress)

```
describe('User List', () => {
  it('loads users and allows CRUD operations', () => {
    cy.visit('/');
    cy.contains('User List').should('be.visible');
    cy.get('li').should('have.length', 10);

    // Implementa pruebas para agregar, editar y eliminar usuarios
  });
});
```

5. Optimización y SEO

5.1. Objetivo

Optimizar el rendimiento de la aplicación y mejorar su SEO.

5.2. Tareas

1. Implementa lazy loading para los componentes que no son críticos en la carga inicial.
2. Utiliza React.memo() para evitar renderizaciones innecesarias en componentes que no cambian frecuentemente.
3. Optimiza las imágenes utilizadas en la aplicación.
4. Implementa meta tags dinámicos para SEO usando React Helmet.
5. Asegúrate de que la aplicación sea accesible, siguiendo las mejores prácticas de WCAG.
6. Implementa server-side rendering (SSR) o static site generation (SSG) para mejorar el tiempo de carga inicial y el SEO.

5.3. Código de ejemplo (Lazy Loading)

```
import React, { Suspense, lazy } from 'react';

const UserDetails = lazy(() => import('./UserDetails'));

function App() {
  return (
    <div>
```

```
      <Suspense fallback={<div>Loading...</div>}>
        <UserDetails />
      </Suspense>
    </div>
  );
}
```

6. Despliegue a producción

6.1. Objetivo

Desplegar la aplicación fullstack en diferentes plataformas de hosting.

6.2. Tareas

1. Prepara tu aplicación para producción:
   - Optimiza el build de React (`npm run build`).
   - Configura variables de entorno para las URLs de API y otras configuraciones sensibles.
2. Despliega el frontend en Vercel:
   - Conecta tu repositorio de GitHub a Vercel.
   - Configura las variables de entorno necesarias en Vercel.
3. Despliega el frontend en Netlify:
   - Usa el CLI de Netlify o conecta tu repositorio directamente.
   - Configura el comando de build y el directorio de publicación.
4. Despliega el backend en fly.io:
   - Instala el CLI de fly.io y autentica tu cuenta.
   - Crea un archivo `fly.toml` para tu aplicación.
   - Despliega usando `fly deploy`.
5. Despliega el backend en railway.app:
   - Conecta tu repositorio de GitHub a Railway.
   - Configura las variables de entorno y los comandos de inicio.

6.3. Ejemplo de configuración para Vercel (vercel.json)

```
{
  "version": 2,
  "builds": [
    {
      "src": "package.json",
      "use": "@vercel/node"
    }
  ],
  "routes": [
    {
      "src": "/(.*)",
      "dest": "/"
    }
  ]
}
```

7. Entrega y Evaluación

- Sube tu código a un repositorio de GitHub.
- Proporciona URLs para los despliegues en cada plataforma.
- Incluye un README.md con instrucciones para ejecutar la aplicación localmente y realizar las pruebas.
- Realiza un informe completo del laboratorio y súbelo a la carpeta "Actividades estudiantes"
- La evaluación se basará en la completitud de las tareas, la calidad del código, las pruebas implementadas y el rendimiento de la aplicación desplegada.