Social Media API Client \_ Frontend - Project Report

# Project Overview

## Introduction

The Social Media API Client is a React-based application that allows users to authenticate using Google or GitHub OAuth and perform CRUD operations on an API. The app demonstrates authentication, authorization, responsive UI, error handling, and API integration.

## Key Features

* Social media login (Google & GitHub)
* CRUD operations via Axios and JSONPlaceholder API
* Authentication & authorization
* Responsive design
* Error handling and user feedback

## Objectives

1. Build a functional social media login system.
2. Implement API CRUD functionality with token-based authentication.
3. Ensure a responsive and user-friendly UI.
4. Include error handling and feedback.
5. Verify application functionality through manual testing.

# UI Design and Component Architecture

## Front-End Framework

* React.js with React Router DOM for navigation.

## Component Structure

| Component | Responsibility |
| --- | --- |
| Login.js | Handles Google/GitHub login and token storage |
| Dashboard.js | Displays welcome message and navigation |
| ApiCrud.js | Handles CRUD operations |
| api.js | Axios instance and CRUD endpoints |

## Design Features

* Flexbox for centering content.
* Inline CSS for styling buttons with hover effects.
* Responsive design for desktop and mobile.

**Screenshots:** - Placeholder for Login page screenshot - Placeholder for Dashboard screenshot - Placeholder for CRUD page screenshot

# Social Media Login & API Integration

## Google Login

* Implemented with @react-oauth/google.
* Stores credential token in localStorage.
* Redirects to /dashboard on success.

## GitHub Login

* OAuth integration via URL redirection.
* Redirects to /dashboard on success.

## API Integration

### Axios Setup

const api = axios.create({  
 baseURL: 'https://jsonplaceholder.typicode.com',  
 headers: { 'Content-Type': 'application/json' }  
});  
  
api.interceptors.request.use(config => {  
 const token = localStorage.getItem('token');  
 if(token) config.headers.Authorization = `Bearer ${token}`;  
 return config;  
});

### CRUD Functions

export const getItems = () => api.get('/posts');  
export const createItem = data => api.post('/posts', data);  
export const updateItem = (id, data) => api.put(`/posts/${id}`, data);  
export const deleteItem = id => api.delete(`/posts/${id}`);

# Error Handling and Security

## Error Handling

* Login errors: console logs if OAuth fails.
* CRUD errors: try-catch blocks display error messages.
* Route protection: unauthorized access redirects to login.

## **Security Measures**

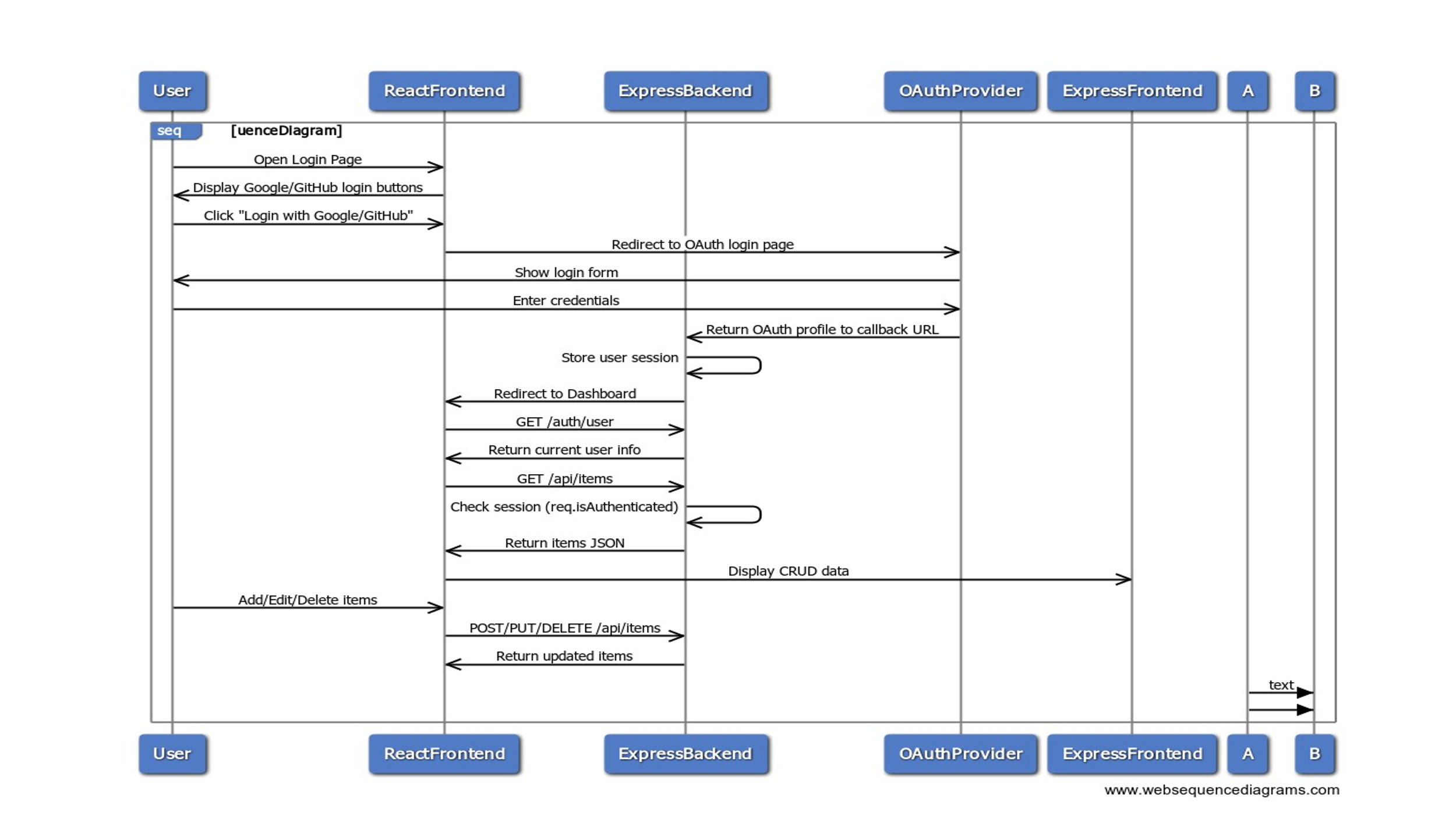
* Token-based authentication using localStorage.
* Protected routes using React Router.
* Recommended HTTPS for production.

## Responsive Design

* Tested on multiple screen sizes.
* Buttons and inputs scale appropriately.

**Screenshots/Demo:** - The rest of the screenshots and diagrams are in the GitHub Repo: <https://github.com/JPretz/sweng861-2025-FALLSocialMedialAPI_client_Frontend>

The following workflow diagram represents the frontend process from the user perspective and the Authorization mechanism.



The following illustration is the Axios configuration within the program:

A screenshot of a computer program

AI-generated content may be incorrect.

Testing and Conclusion

## Manual Testing

| Feature | Test Steps | Expected Result | Actual Result |
| --- | --- | --- | --- |
| Google Login | Click button | Redirect & token stored | ✅ Pass |
| GitHub Login | Click button | Redirect & token stored | ✅ Pass |
| CRUD Operations | Create/Read/Update/Delete | UI updates, errors handled | ✅ Pass |
| Route Protection | Access /dashboard without token | Redirect to login | ✅ Pass |
| Logout | Click logout | Token removed & redirect | ✅ Pass |
| Responsive UI | Resize window | Layout remains centered | ✅ Pass |

## 

## Conclusion

The Social Media API Client is fully functional and meets all homework requirements: - Part 1: UI implemented with responsive design. - Part 2: Social login integrated, API CRUD functional. - Part 3: Error handling, security, and testing completed.

**References**

1. React Documentation: <https://reactjs.org/docs/getting-started.html>
2. React Router Documentation: https://reactrouter.com/en/main
3. Axios Documentation: <https://axios-http.com/docs/intro>
4. Google OAuth Documentation: <https://developers.google.com/identity/gsi/web>
5. GitHub OAuth Documentation: <https://docs.github.com/en/developers/apps/building-oauth-apps>
6. JSONPlaceholder API: <https://jsonplaceholder.typicode.com/>
7. PlantUML Documentation: <https://plantuml.com/>