Report for Full-Stack Web Application: Integration of Frontend with Golang Backend

Student Name: Tursun Bekzat

ID: 21B030726

Course Title: Golang

Date: 29.10.2024

Almaty, 2024

Table of Contents

Introduction

This project focuses on building a full-stack web application that integrates a React frontend with a Golang backend. The goal was to develop an application with CRUD operations, authentication using JWT, and role-based access control (RBAC). Web applications require seamless frontend-backend communication, and security measures like JWT-based authentication are essential to protect sensitive resources.

The report covers three main sections:

1. Connecting with the frontend (initial setup and API integration).
2. Improving frontend-backend communication (handling API responses and state management).
3. Implementing secure authentication and authorization with JWT and RBAC.

Connecting with Frontend, Part 1

Overview.

APIs (Application Programming Interfaces) act as bridges between frontend and backend systems, allowing applications to exchange data efficiently. In this project, I used Golang to develop a RESTful API that exposes CRUD operations for a book management system. The React frontend consumes this API to display and manage book data dynamically.

Exercises.

Exercise 1: Setting Up a Simple Golang API

- Used the Gin framework to create a Golang project.
- Developed CRUD operations:
    - Create: POST /books
    - Read: GET /books
    - Update: PUT /books/:id
    - Delete: DELETE /books/:id
- Implemented an in-memory data store for book management.

Exercise 2: Consuming the API with React

- Set up a React project using Create React App.
- Connected the React frontend to the Golang API using Axios.
- Developed a Books component to fetch and display book data.

Findings.

The initial setup of the API and frontend integration posed challenges in ensuring proper CORS configuration. There was also a learning curve in handling Axios requests correctly, especially with protected routes. Ensuring the correct structure of CRUD endpoints and managing in-memory data for testing helped streamline the development process.

Connecting with Frontend, Part 2

Overview.

The second phase involved refining the frontend-backend interaction to handle various API responses effectively. Managing application state efficiently is essential for ensuring a smooth user experience, especially in larger applications. In this project, I introduced loading indicators, error handling, and state management using React Context.

Exercises.

Exercise 3: Handling API Responses in React

- Implemented loading indicators to show progress during data fetching.
- Added error handling to manage failed API calls gracefully.
- Developed form functionality to create new book entries using the API.

Exercise 4: State Management

- Used React Context to manage the global state for books.
- Refactored components to leverage Context for centralized state management.

Findings.

One of the main challenges was managing asynchronous API calls and ensuring the UI remained responsive. Handling loading states and errors improved the user experience significantly. Implementing React Context for state management ensured that the application maintained a consistent state across components, making the code more modular and maintainable.

Authentication and Authorization

Overview.

Authentication verifies the user's identity, while authorization ensures users can access only the resources they are allowed. In this project, I implemented JWT-based authentication to secure API endpoints. Additionally, role-based access control (RBAC) was introduced to grant different levels of access to users based on their roles (e.g., admin, user).

Exercises.

Exercise 5: Implementing JWT Authentication in Golang

- Installed necessary JWT packages for Golang.
- Developed endpoints for user registration and login.
- Implemented middleware to protect routes using JWT tokens.

Exercise 6: Integrating JWT in React

- Modified the login form to capture user credentials and store JWT in localStorage.
- Implemented logic to include the JWT in the headers of protected API requests.

Exercise 7: Role-Based Access Control (RBAC)

- Extended the user model to include roles.
- Added middleware to check user roles before granting access to certain endpoints.

Findings.

JWT implementation was straightforward, but ensuring the token was passed correctly in the headers required careful attention to detail. I encountered challenges with token expiration handling and protecting routes dynamically in React. Implementing RBAC added an extra layer of security, ensuring that only authorized users could perform certain actions.

Conclusion

This project provided hands-on experience in building a full-stack web application with a focus on API integration, state management, and security. Through the exercises, I explored the complexities of managing frontend-backend communication and implementing secure authentication mechanisms.

Key takeaways:

1. Frontend-backend integration is crucial for smooth data exchange in web applications.
2. State management and error handling are essential for maintaining a responsive and robust UI.
3. JWT-based authentication ensures secure access, and RBAC enhances control over protected resources.

These concepts will be invaluable in future projects, especially when developing scalable, secure, and maintainable web applications.

References

1. Gin Web Framework Documentation. (https://gin-gonic.com/docs/);
2. React Documentation. (https://reactjs.org/docs/getting-started.html);
3. Axios Documentation. (https://axios-http.com/docs/intro);
4. JWT.io. (https://jwt.io/);
5. CORS Documentation (https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS).