Assignment 3, golang

Daniyar Anuar
Application development in the Golang language
03.11.2024

Introduction

This project focuses on integrating a React frontend with a Go backend, covering essential concepts like API development, frontend-backend communication, and security using JSON Web Tokens (JWT). These are crucial skills for creating modern web applications that are both dynamic and secure.

Frontend-backend integration enables smooth data exchange, allowing the application to be interactive and responsive. Security measures like JWT authentication and role-based access control (RBAC) help ensure that sensitive data is protected and that only authorized users can access certain features. Together, these elements lay the groundwork for building scalable, secure applications that provide a strong user experience.

Report: Connecting Frontend with Backend (Part 1)

Overview: Role of APIs in Web Applications

APIs connect the frontend and backend of a web application. They allow the frontend to send and retrieve data from the backend, making the app dynamic and interactive.

Benefits:

- **Data Exchange**: The frontend can get and send data without reloading.
- **Security**: APIs control data access.
- **Modularity**: Frontend and backend can be developed separately.

Exercises Summary

Exercise 1: Setting Up a Simple API in Go

- **Goal**: Build a basic API in Go with CRUD (Create, Read, Update, Delete) functionality.
- **Steps**: Set up a Go project with Gin, created endpoints for CRUD operations, and used an in-memory array to store data temporarily.
- **Result**: A working API to handle data operations.

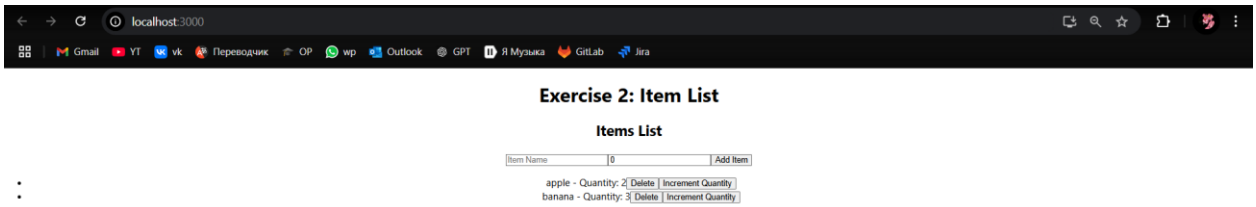
Exercise 2: Connecting React to the API

- **Goal**: Create a React app to interact with the Go API.
- **Steps**:
    1. Set up a React project and used Axios for API requests.
    2. Built components to display, add, update, and delete items.
    3. Enabled CORS on the Go server to allow requests from the React frontend.
- **Result**: React app successfully connects to the API, displaying and updating items.

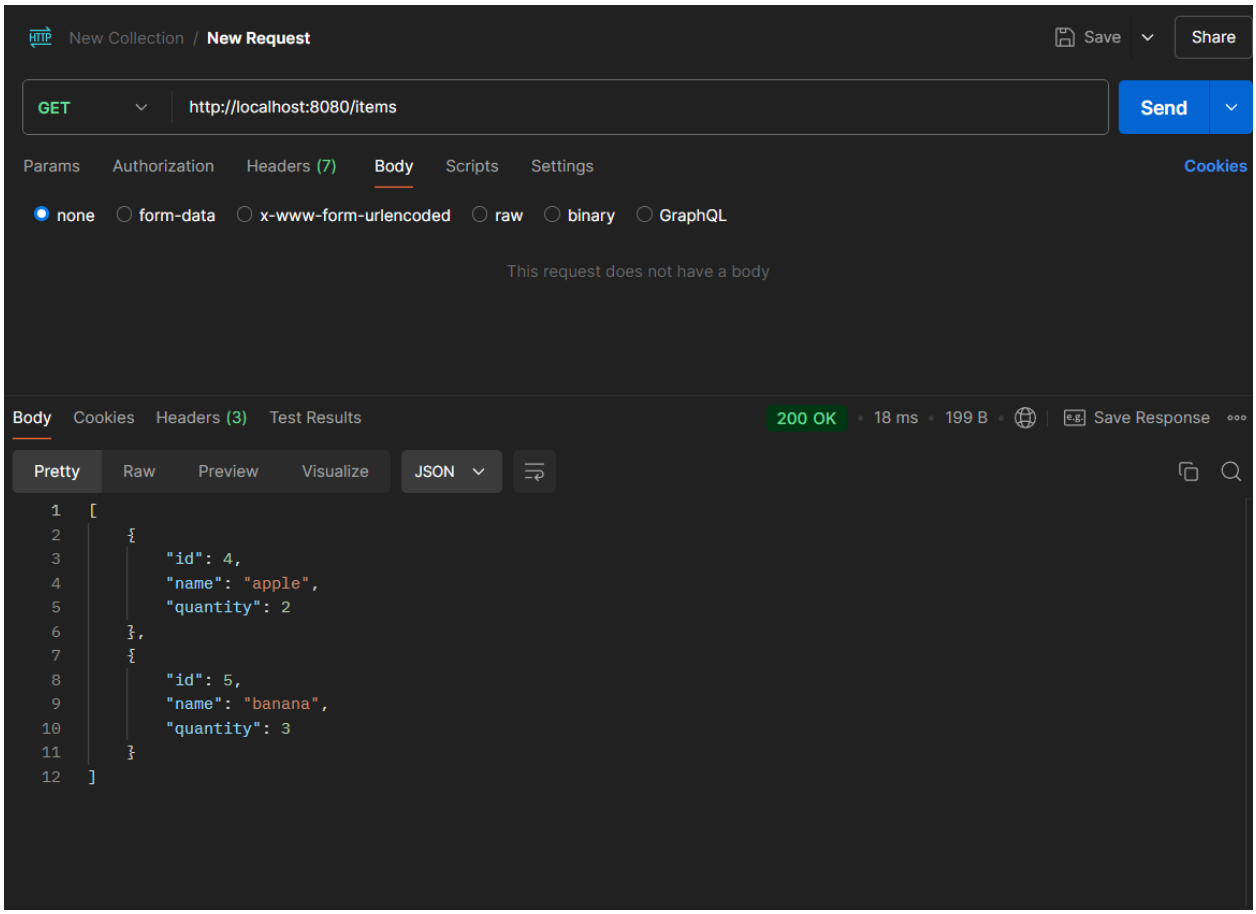
Findings: Challenges and Solutions

1. **CORS Issues**: React couldn't connect to the API initially due to CORS restrictions. Solution: Added CORS middleware on the Go server.

2. **Handling Null Data**: React showed errors when data was missing. Solution: Set default data as an empty array to prevent errors.
3. **Temporary Data Storage**: Data is stored only in memory, so it resets on server restart. For real apps, a database would be needed.



:



Report: Connecting Frontend with Backend (Part 2)

Overview: Advanced Topics in Connecting Frontend and Backend

In this part, we explored advanced techniques for effectively connecting the frontend and backend, including enhanced API response handling and centralized state management. These improvements make the application more robust, modular, and user-friendly.

Benefits:

- **Improved Error Handling**: Users receive clear feedback if something goes wrong, improving user experience.
- **State Management**: Centralized state management simplifies data flow and improves maintainability.
- **Code Modularity**: Using context or a state management library helps separate data logic from UI, making the code cleaner and more scalable.

Exercises Summary

Exercise 3: Handling API Responses in React

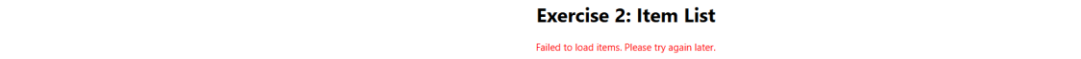
- **Goal**: Improve API response handling in React to provide better user experience.
- **Steps**:
    1. Added error handling for API calls to notify users of issues like network errors.
    2. Displayed loading indicators during data fetching, so users know when data is being loaded or added.
    3. Updated the item form to handle new entries with clear feedback (e.g., reset form fields after adding an item).
- **Result**: React app now handles API responses more gracefully, providing feedback to users during loading and error scenarios.

Exercise 4: State Management

- **Goal**: Centralize application state using React Context to manage data from the API effectively.
- **Steps**:
    1. Created a global ItemContext and ItemProvider for managing the item list, loading states, and error states.
    2. Refactored ItemList and ItemForm components to use context instead of local state.
    3. Integrated functions for fetching, adding, and deleting items directly into the context, allowing any component to access the same data and actions.
- **Result**: The application now uses a single source of truth for data, improving consistency and making state updates easier to manage across components.

Findings: Improvements in API Handling and State Management

1. **Better User Feedback**: Thanks to error handling and loading indicators, users now see when data is loading or if something goes wrong. This makes the app feel more responsive.
2. **Simplified State Management**: Using React Context for global state made data flow easier and cleaner. Now we don't have to pass data through multiple components, making the code more organized. If needed, we could also scale up to a tool like Redux.
3. **Cleaner Code Structure**: By moving data logic into a central context, components focus only on UI. This makes it easier to add new features or make changes without breaking other parts of the app.



Overview: Why Authentication and Authorization Matter

In any app, authentication and authorization are essential for keeping data secure and ensuring that users only have access to what they're allowed to see or do. Authentication checks a user's identity, while authorization determines what that user can access. Together, they help protect sensitive information, personalize user experience, and control access to features in a structured way.

Benefits:

- **Security**: Keeps data safe by restricting access to authorized users.
- **Controlled Access**: Lets us assign roles (e.g., "admin" vs. "user") for different levels of access.
- **Personalization**: Users only see features relevant to them, improving their experience.

Exercises Summary

Exercise 5: Setting Up JWT Authentication in Golang

- **Goal**: Add secure login using JWT to the Go API.
- **Steps**:
    1. Created registration and login endpoints.
    2. Generated JWT tokens after successful login for users to access protected routes.
    3. Built middleware to check JWT tokens for certain routes.
- **Result**: Users log in to get a JWT, which lets them access parts of the app that require authentication.

Exercise 6: Adding JWT Support in React

- **Goal**: Set up the React frontend to use JWT tokens from the backend.
- **Steps**:
    1. Built a login form in React that collects user credentials and fetches a JWT token from the backend.

2. Saved the token in local storage to keep users logged in during their session.
3. Configured Axios to add the JWT token to requests for any API call to protected routes.

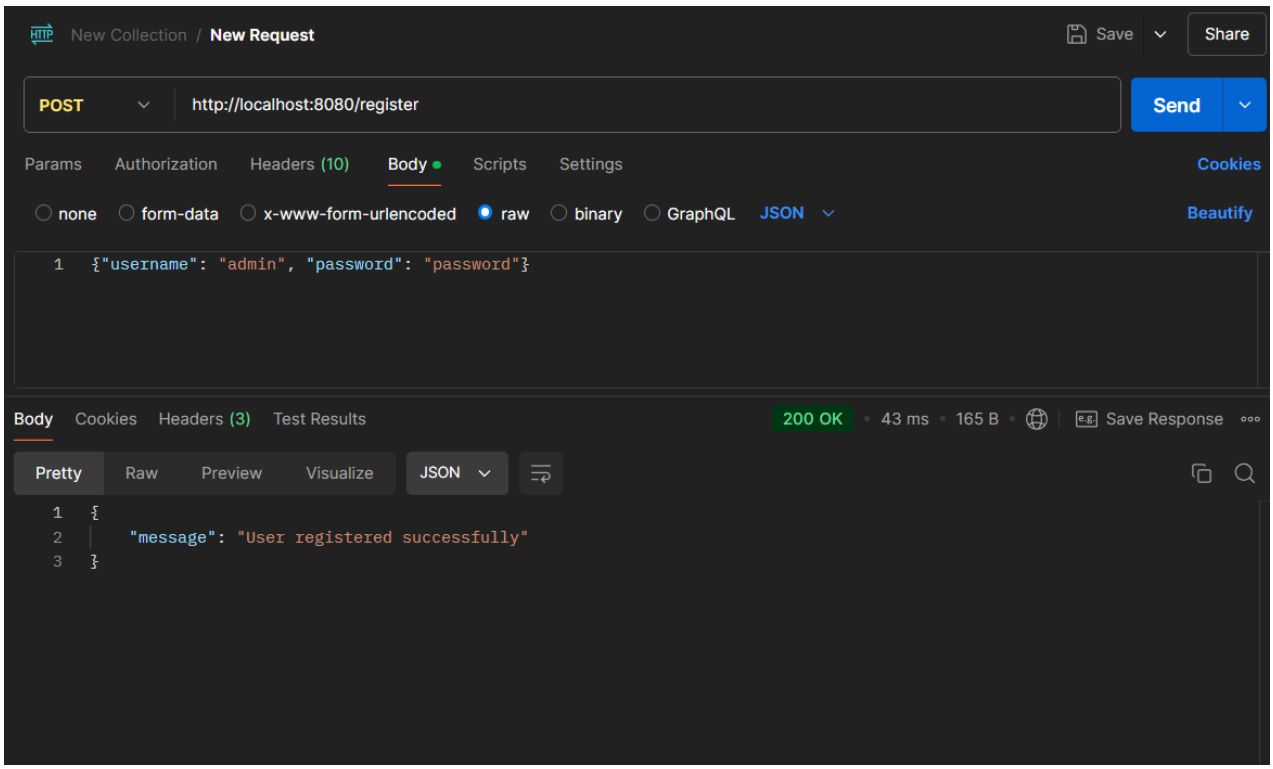
- **Result**: The React app now handles login, saves the JWT token, and includes it in each request to secure API routes.

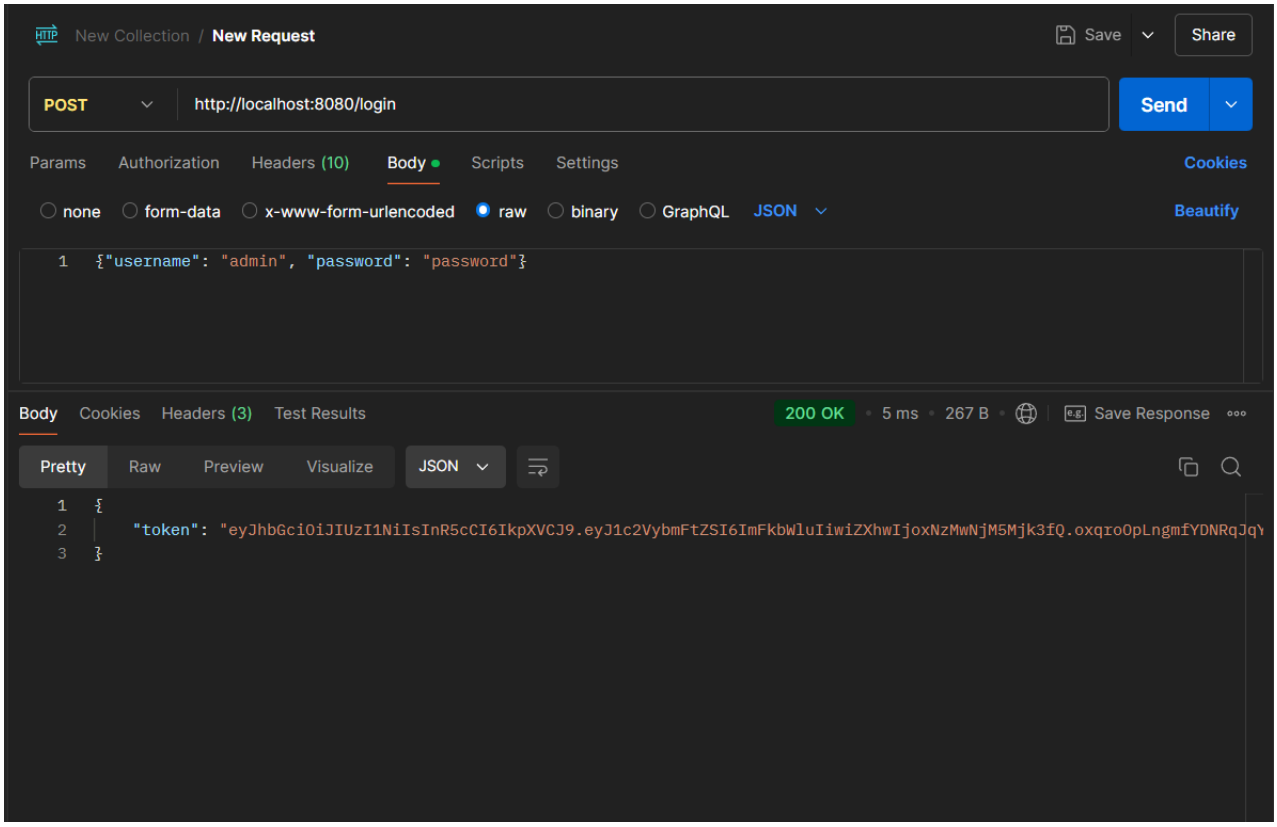
Exercise 7: Role-Based Access Control (RBAC)

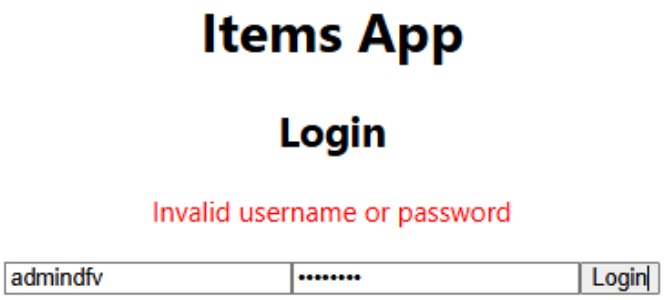
- **Goal**: Use roles to control access to specific API routes.
- **Steps**:
    1. Added a "role" field (e.g., "admin", "user") to the user model.
    2. Created middleware to check user roles before allowing access to certain endpoints.
    3. Verified that only users with the right role (e.g., "admin") could access certain routes.
- **Result**: Role-based access control now limits access to sensitive routes, allowing only specific roles to access certain parts of the API.

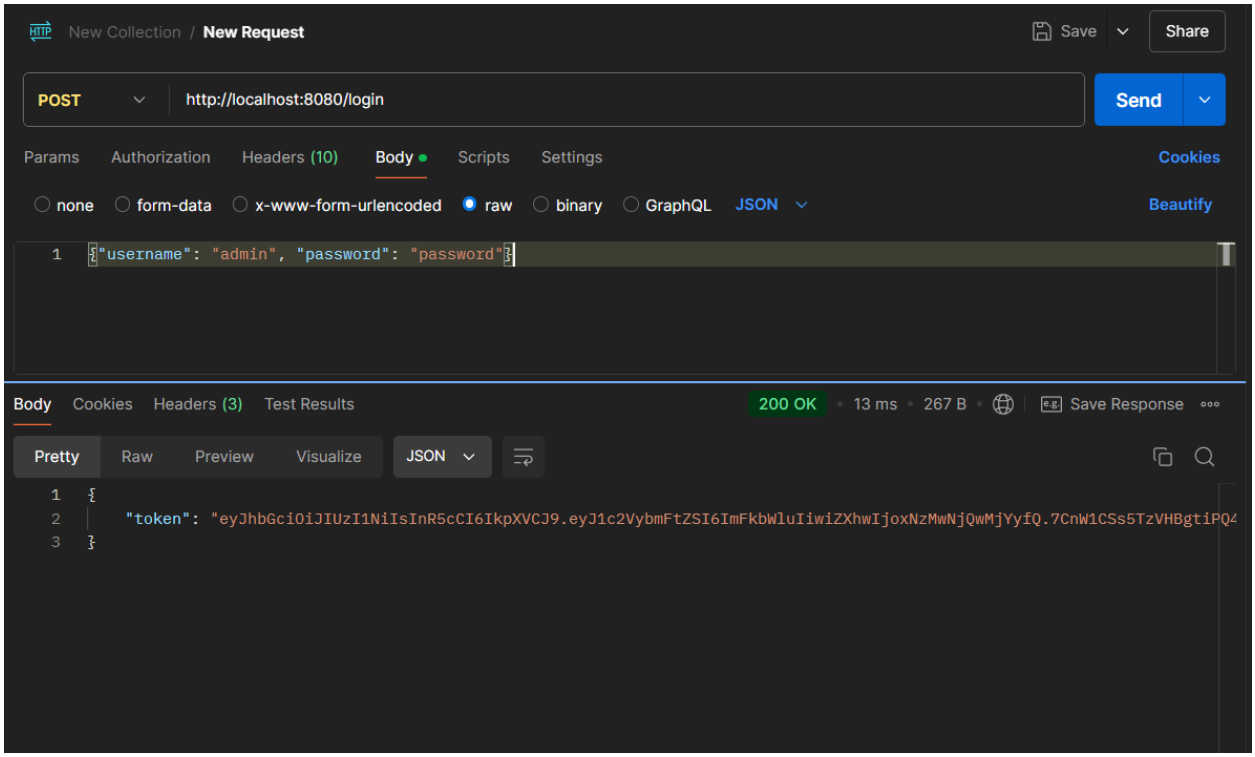
Findings: Working with JWT and Role-Based Access Control

1. **JWT Authentication**: Using JWT tokens made authentication lightweight and stateless. Middleware on the backend could validate tokens and control access, while tokens stored in the frontend allowed React to manage sessions easily.
2. **Role-Based Access Control**: Adding roles to JWT claims made it easy to create role-based restrictions in the app. Middleware could check the user's role before granting access to endpoints, enhancing security and flexibility.
3. **Challenges**:
    - **Token Expiration**: Managing token expiration was a bit tricky; we needed to handle token renewal without interrupting the user's session.
    - **Role Misconfigurations**: It was important to double-check roles, especially as the app scales with more users and permissions.
    - **Frontend Integration**: Making sure Axios sent JWT tokens with each request required some setup, especially to handle token expiration smoothly.


New Collection / New Request
Save
Share
POST
http://localhost:8080/register
Send
Params
Authorization
Headers (10)
Body
Scripts
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
Beautify
1 {"username": "admin", "password": "password"}
Body
Cookies
Headers (3)
Test Results
200 OK
43 ms
165 B
Save Response
Pretty
Raw
Preview
Visualize
JSON
1 {
2 "message": "User registered successfully"
3 }


New Collection / New Request
Save
Share
POST
http://localhost:8080/login
Send
Params
Authorization
Headers (10)
Body
Scripts
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
Beautify
1 {"username": "admin", "password": "password"}
Body
Cookies
Headers (3)
Test Results
200 OK
5 ms
267 B
Save Response
Pretty
Raw
Preview
Visualize
JSON
1 {
2 "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VybmFtZSI6ImFkbWluIiwiZXhwIjoxNzMwNjM5Mjk3fQ.oxqroOpLngmfYDNRqJq
3 }





Conclusion

In this project, we gained practical experience in connecting a React frontend with a Go backend, focusing on creating a smooth user experience and adding security through JWT authentication and role-based access control (RBAC). Starting with basic CRUD operations and progressing to managing application state and securing routes, we built a solid foundation for integrating and protecting frontend-backend interactions.

These skills are directly useful for future projects, especially when building apps that need secure user access and clear role distinctions. Knowing how to connect, protect, and manage data between the frontend and backend gives us essential tools for creating reliable, user-focused applications.

References

- Gin Web Framework Documentation: https://github.com/gin-gonic/gin
- Golang JWT Library: https://github.com/golang-jwt/jwt
- React Documentation: https://reactjs.org/docs/getting-started.html
- Axios Documentation: https://axios-http.com/docs/intro
- RESTful API Design Guidelines and Best Practices