**PennyFlow Backend – Sprint 3 Detailed Report**

**1. Overview and Objectives**

During Sprint 3, our backend team focused on stabilizing and enhancing the server-side functionality of the PennyFlow application. Our objectives were as follows:

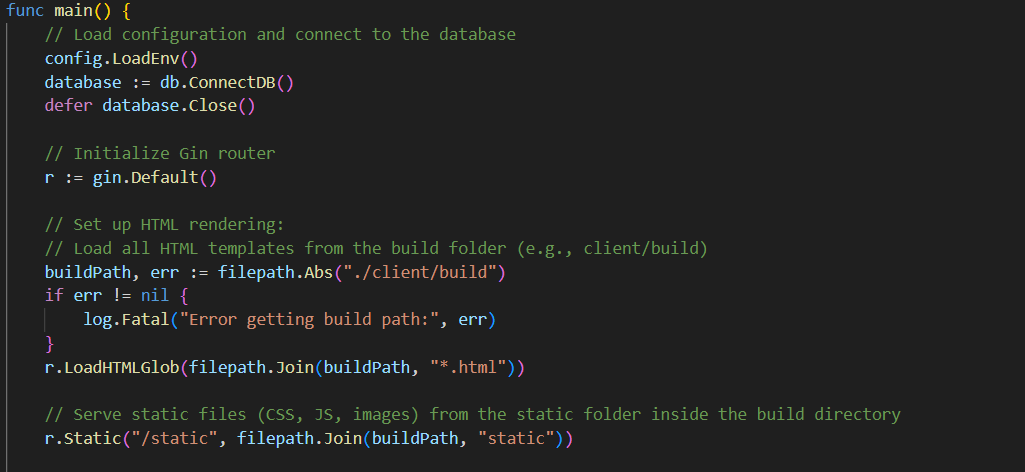
* **Bug Resolution and Feature Polishing:** Complete and refine any unfinished tasks while eliminating known bugs.
* **Security and Validation Enhancements:** Enforce strict validation rules and improve user authentication using secure practices such as bcrypt password hashing.
* **Enhanced Test Coverage:** Develop a comprehensive suite of unit tests to ensure the reliability of user management, expense tracking, and middleware functionalities.
* **Improved Documentation and Maintainability:** Update code documentation and restructure the project to facilitate future scalability and maintenance.

**2. System Architecture and Component Breakdown**

The backend is implemented in **Go** using the **Gin** web framework. The architecture is modular, with separate responsibilities for initialization, routing, database access, and middleware.

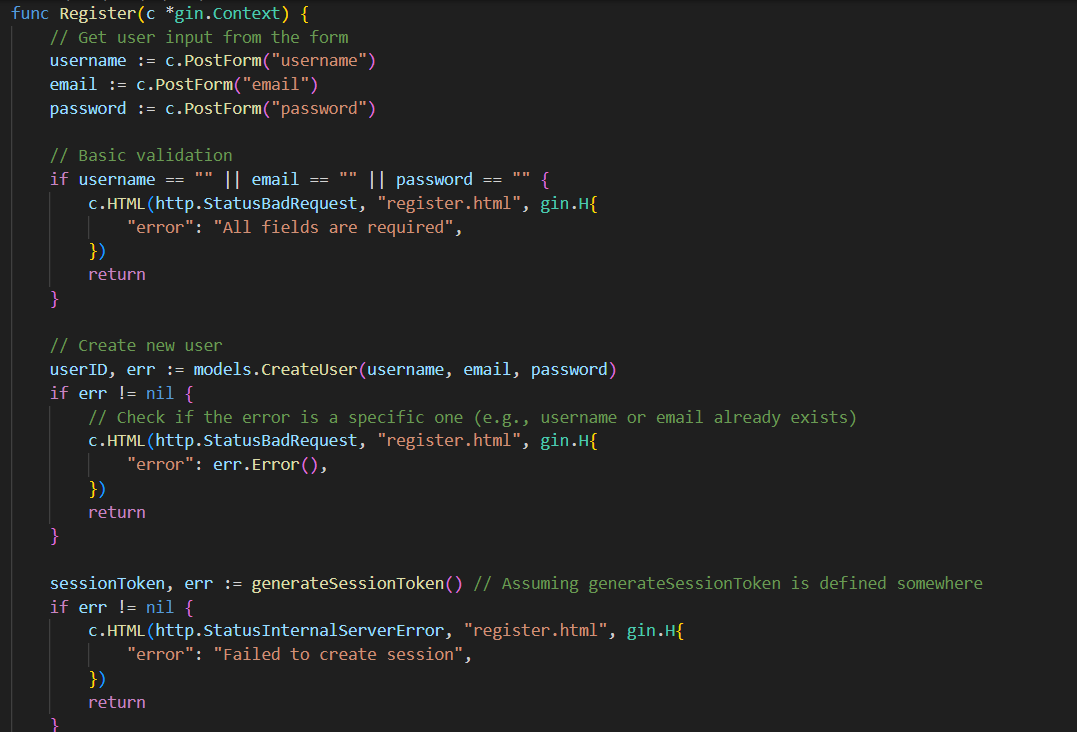
**2.1 Application Initialization (File: main.go)**

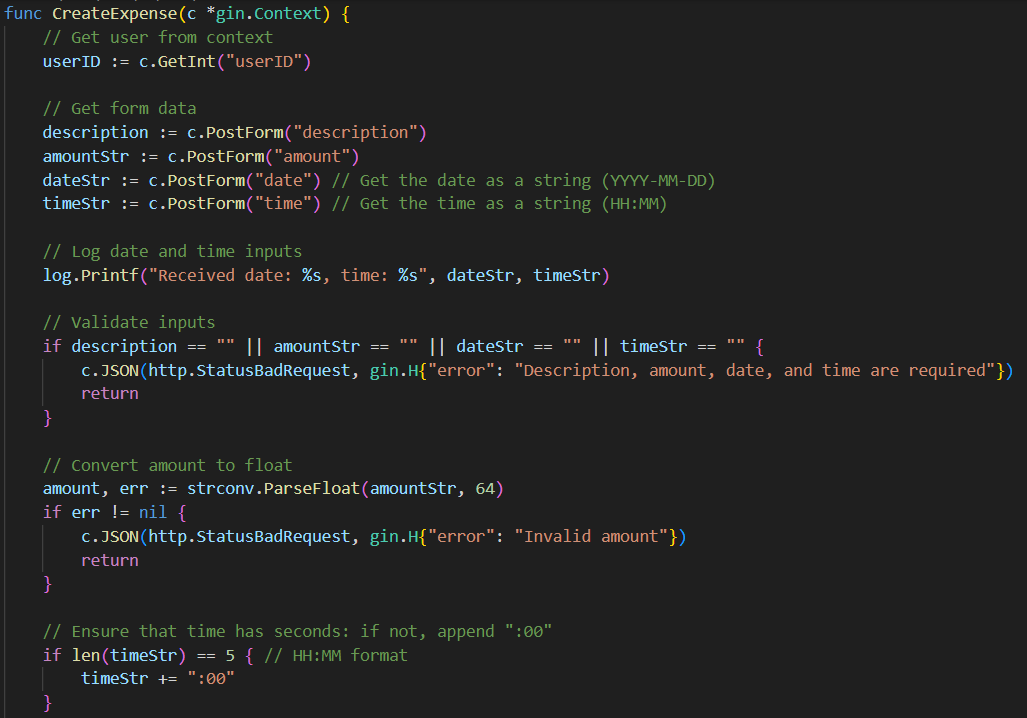
* **Description:** This file is the entry point of the application. It handles environment configuration, database connection setup, and defines all HTTP routes.



**2.2 Routing and Controllers (Files: e.g., handlers/auth.go and handlers/expense.go)**

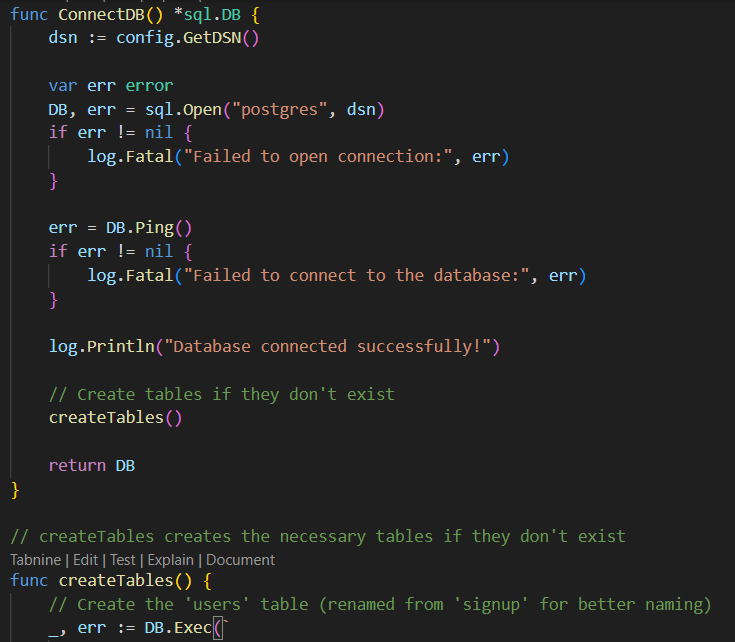
* **Description:** The backend routes are organized into separate handlers for pages, authentication, and expense management.

Authentication :  Expense management:



**2.3 Database and Models (Files: e.g., db.go, models/user.go, models/expense.go)**

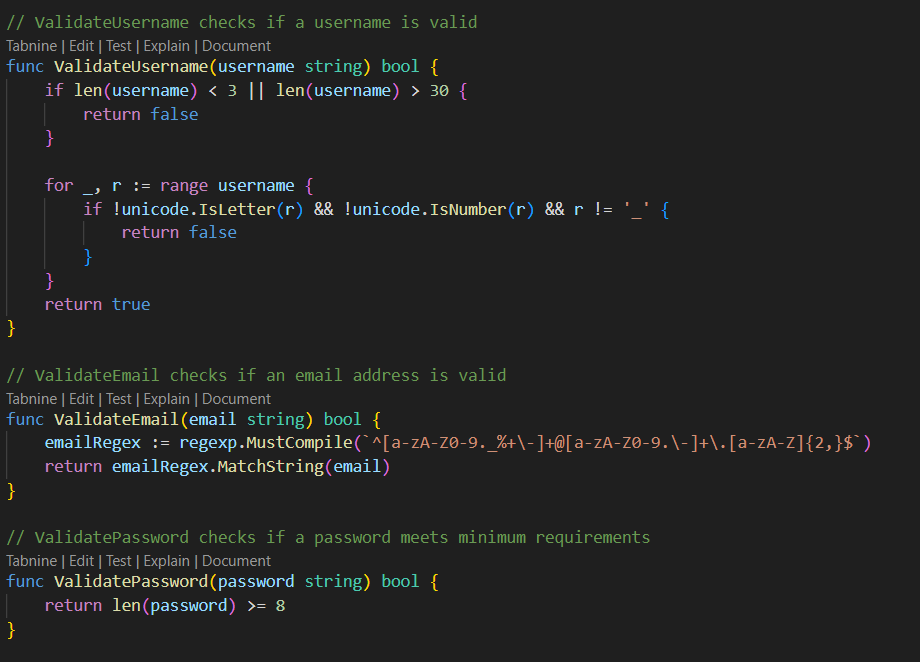
* **Description:** Modules that handle PostgreSQL connections, schema migrations, and encapsulate business logic for users and expenses.



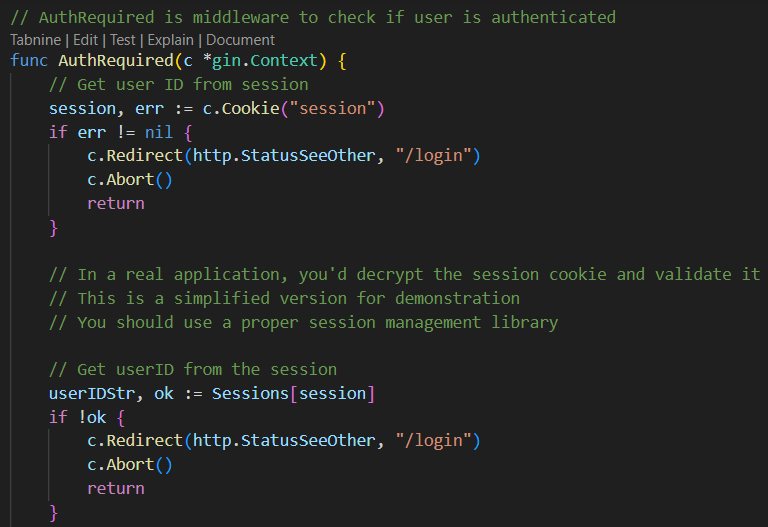
**2.4 Utilities and Middleware**

**Description:**  
Validation utilities (e.g., in validator.go) enforce proper input formatting (e.g., regex-based email and password validation). In parallel, middleware (e.g., in middleware/auth.go) checks for valid sessions before granting access to protected routes.

**Key Code (Validation Utility):**



**Key Code (Authentication Middleware):**



**3. Feature Enhancements**

**3.1 Authentication and Session Management**

* **Description:**
  + Implementation of secure registration and login using robust input validation.
  + Session handling via HTTP-only cookies.
* **Reference:**
  + See the screenshots and code excerpts from handlers/auth.go.

**3.3 Database Integration Improvements**

* **Description:**
  + Automatic schema migration during startup.
  + Use of parameterized SQL queries for enhanced security.

**4. Unit Testing Coverage**

**Test Scope:**

* **User Tests:**  
  Verify that duplicate user registration is prevented and that password operations (hashing and verification) work correctly.
* **Expense Tests:**  
  Ensure proper creation, retrieval, updating, and deletion of expense records, including safeguards for unauthorized modifications.
* **Validation Tests:**  
  Confirm that only valid inputs (e.g., correctly formatted emails, strong passwords) are accepted.
* **Middleware Tests:**  
  Check that protected endpoints reject unauthorized access.
* Placement**:**  
  Insert these images in the “Unit Testing Coverage” section alongside a brief explanation of the tests performed.

**5. Updated API Documentation**

Endpoints Summary:

* **Authentication Endpoints:**
  + POST /register: Creates a new account, validates input, and starts a session.
  + POST /login: Authenticates user credentials and initiates a session.
  + GET /logout: Clears the session and logs out the user.
* **Expense Endpoints (Protected):**
  + GET /expenses: Retrieves a JSON list of all expenses for the authenticated user along with a computed total.
  + POST /expenses: Inserts a new expense record after validating input data.
  + PUT /expenses/:id: Updates an expense record if the user is its owner.
  + DELETE /expenses/:id: Deletes an expense record given proper ownership.
* **Analytics Endpoint:**
  + GET /analysis: Delivers aggregated data for user expenses (e.g., breakdown by category, monthly summaries).
* **Backend API Documentation**

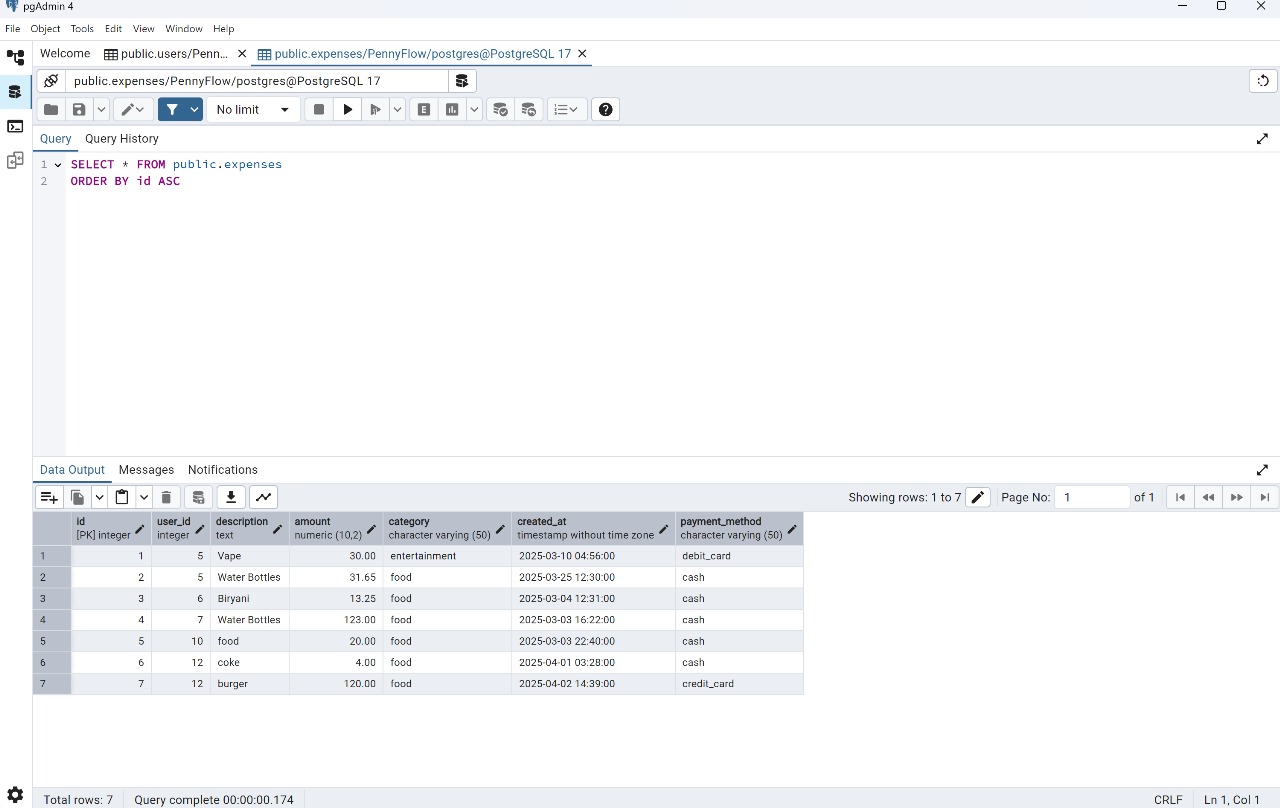
https://docs.google.com/document/d/1dR3KRYIVRCh8PQtA3aY82ERigMjkhe-Ucwjqu1pSUf0/edit?usp=sharing

**Database Structure**

Our PostgreSQL database is a critical component of the PennyFlow backend. It is composed of two main tables that support the core functionalities of the application:

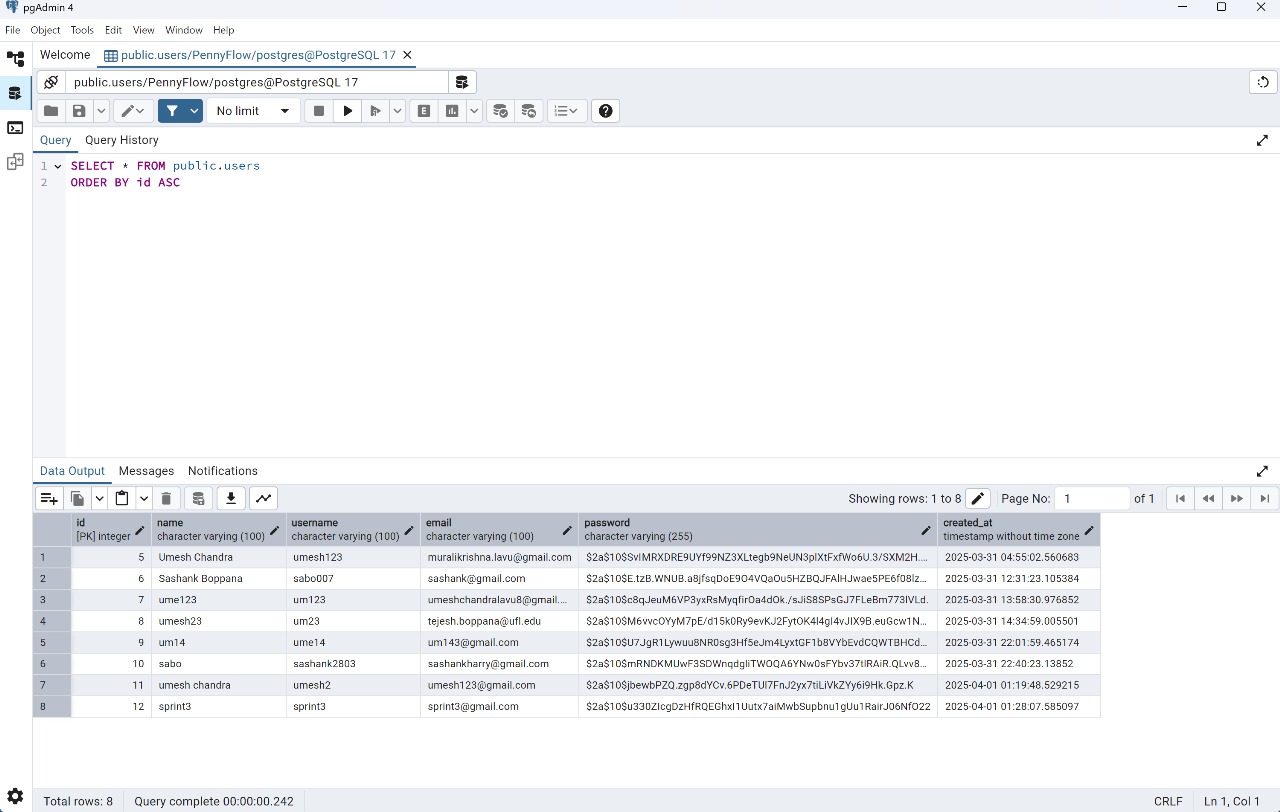
1. **Users Table:**  
   This table stores all user account data, including the username, email, and the password hash (ensuring user credentials are stored securely). Each record represents an individual user.
2. **Expenses Table:**  
   This table records individual expense entries for each user. It includes details such as expense amount, description, timestamp, and a reference to the associated user.

Figure 1: Expenses Table Screenshot



**Caption:** "Figure 1: The Expenses table, displaying fields such as expense ID, user ID, amount, description, and timestamp."

Figure 2: Users Table Screenshot



**Caption:** "Figure 2: The Users table, illustrating the storage of user credentials where passwords are securely hashed."

These images confirm that the backend code (specifically in db.go, models/user.go, and models/expense.go) interacts correctly with the expected database schema.