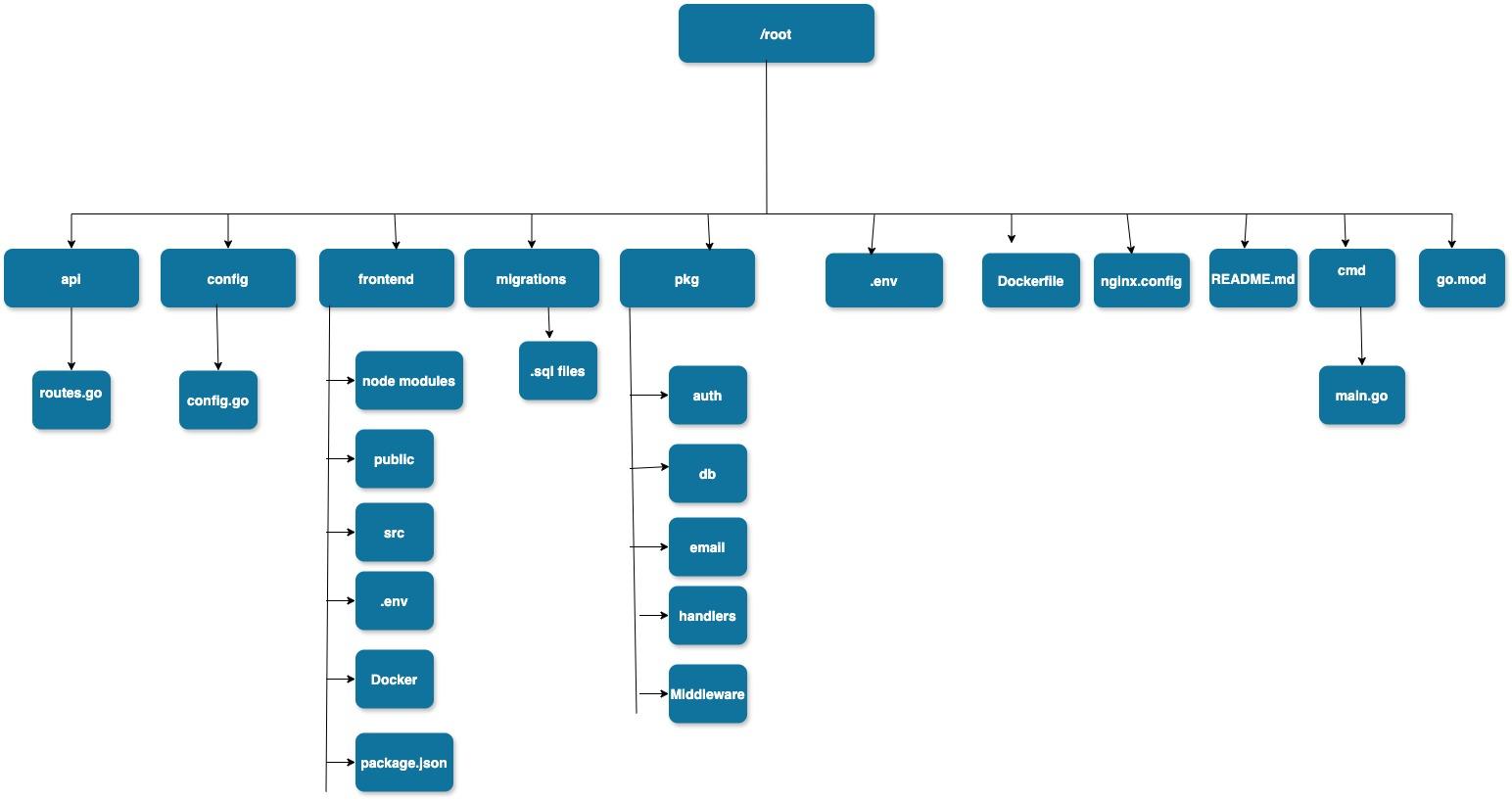
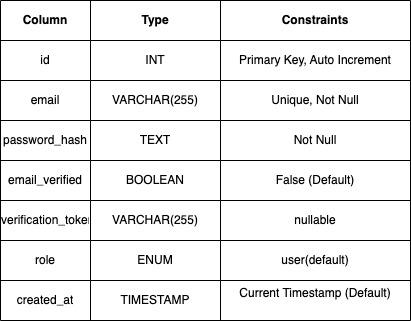
**Directory Structure**

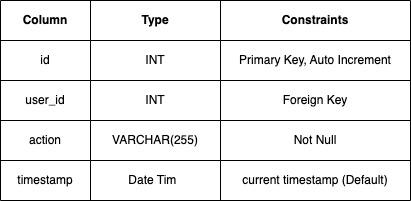


**Database Schema**

**Users Table**



**Activity Logs Table**



**Video**

[**https://youtu.be/8gx1j1gOPN8**](https://youtu.be/8gx1j1gOPN8)

**Running the application.**

git clone <https://github.com/DimanthaG/GO-SQL-UMS.git>

cd GO-SQL-UMS

cd frontend

npm install

npm run build

cd ../

docker-compose up –build

open localhost:80 on your browser

**if you have issues switch docker-compose2.txt and rename it to docker-compose.yaml and vice versa.**

**this would use the docker containers i pushed onto docker hub.**

[**https://hub.docker.com/repository/docker/dimanthag/go-sql-ums-app/general**](https://hub.docker.com/repository/docker/dimanthag/go-sql-ums-app/general)

[**https://hub.docker.com/repository/docker/dimanthag/go-sql-ums-db/general**](https://hub.docker.com/repository/docker/dimanthag/go-sql-ums-db/general)

[**https://hub.docker.com/repository/docker/dimanthag/go-sql-ums-frontend/general**](https://hub.docker.com/repository/docker/dimanthag/go-sql-ums-frontend/general)

[**https://hub.docker.com/repository/docker/dimanthag/go-sql-ums-nginx/general**](https://hub.docker.com/repository/docker/dimanthag/go-sql-ums-nginx/general)

**Code Logic for User Management System**

**Overall Flow**

**Database Connection**

**FrontEnd**

**Backend (Go)**

**Dockerized application**: to ensure the consistency across different environments

Here I have described the flow of the system and function of each component from both backend and frontend.

**FrontEnd**

For the FrontEnd I chose the React library to provide a user-friendly interface.

1. State Management

Uses React Context to manage user authentication state. With the help of these hooks and context, it ensures smooth experience between authenticated and unauthenticated views.

2. API Integration

Fetches and posts data using the Fetch API..

3. Components

Signup: This component handles user registration.

Login Forms: This component handles user authentication.

Reset Password: Handles the reset of password when user try to reset password.

**BackEnd**

For the backend for this management system, I chose Golang. Here are some of the snippets which I did for this system.

1. Database Connection (db package)

* Establish database connection
* Authenticate user login credentials

2. RunMigrations():

* SQL queries for migrating the table.

**Database Logic**

Schema

There are two tables for this management system, the user table stores the information of users and the activity\_log table monitors the activity of users.

Relationship

Activity\_log id is referenced with user\_id from users table.

Queries

Here I have listed some of the database queries for this management system.

INSERT INTO users (email, password\_hash, email\_verified, role) VALUES (?, ?, ?, ?)

SELECT password\_hash FROM users WHERE email = ?

**Docker Integration**

The multi-stage Dockerfile ensures separation of concerns:

**Frontend Build Stage:**

Compiles the React frontend into static assets.

Outputs to /app/build.

**Backend Build Stage:**

Compiles the Go application

Outputs the binary as main.

**Final Container:**

Copies the Go binary and frontend assets into an Alpine container.

Ensures the lightweight deployment of the application.

**Contributions:**

**Dimantha Goonewardena (500237785) – Handle Both Backend and Frontend for this project.**

**Sujan Thapa (500236989) – Handled Frontend and Documented everything for this project.**