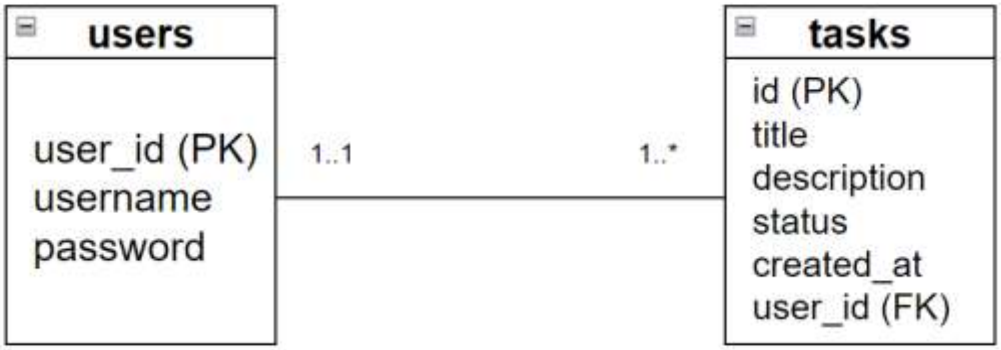
Full Stack Coding Assignment: Task Manager App

❖ Backend :

1. Project Structure and Explanation:

- **com.app.backend.config:**
  - Contains Spring Security configuration (SecurityConfig).
  - This class sets up the security filter chain, CORS configuration, authentication provider, and password encoder.
  - It is crucial for JWT authentication and API security.
- **com.app.backend.controller:**
  - Contains REST controllers (AuthController, TaskController).
  - AuthController handles user registration and login.
  - TaskController handles task creation, retrieval, updating, and deletion.
  - Controllers receive HTTP requests, delegate to services, and return responses.
- **com.app.backend.dto:**
  - Contains Data Transfer Objects (UserDto, TaskDto).
  - DTOs are used to transfer data between the client and server, separating the API layer from the model.
- **com.app.backend.model:**
  - Contains entity classes (User, Task).
  - These classes represent database tables and are used by JPA for database interaction.
- **com.app.backend.repository:**
  - Contains Spring Data JPA repositories (UserRepository, TaskRepository).
  - Repositories provide methods for database access.
- **com.app.backend.security:**
  - Contains JWT utility class (JwtUtil) and JWT authentication filter (JwtAuthFilter).
  - JwtUtil handles JWT generation, validation, and extraction.
  - JwtAuthFilter intercepts requests and authenticates users based on the JWT token.
- **com.app.backend.service:**
  - Contains service classes (AuthService, TaskService).
  - AuthService handles user registration and login logic.
  - TaskService handles task management logic.
  - Services encapsulate business logic and interact with repositories.

2. Database Diagram and Explanation:



Users Table:

- id (PK): The primary key, uniquely identifying each user.
- username: The user's login name.
- password: The user's password.

Tasks Table:

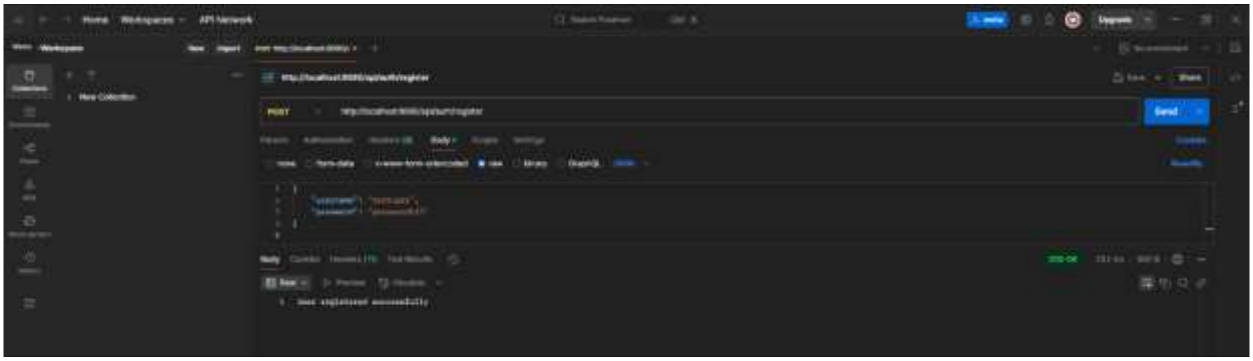
- id (PK): The primary key, uniquely identifying each task.
- title: The task's title.
- description: The task's description.
- status: The task's status (e.g., "To Do," "In Progress," "Done").
- created_at: the time the task was created.
- user_id (FK): The foreign key, linking each task to a specific user in the "Users" table. This is how the tasks and users are related. The (FK) shows that it is a foreign key.

The arrow between the tables represents the relationship between them, showing that a task belongs to a user.

3. API Endpoint Testing in Postman with Screenshots:

Auth Controller:

- **POST /api/auth/register:**
  - Request body: { "username": "testuser", "password": "password123" }
  - Expected response: 200 OK, "User registered successfully"
  - Postman screenshot: include a screenshot of the request and response.



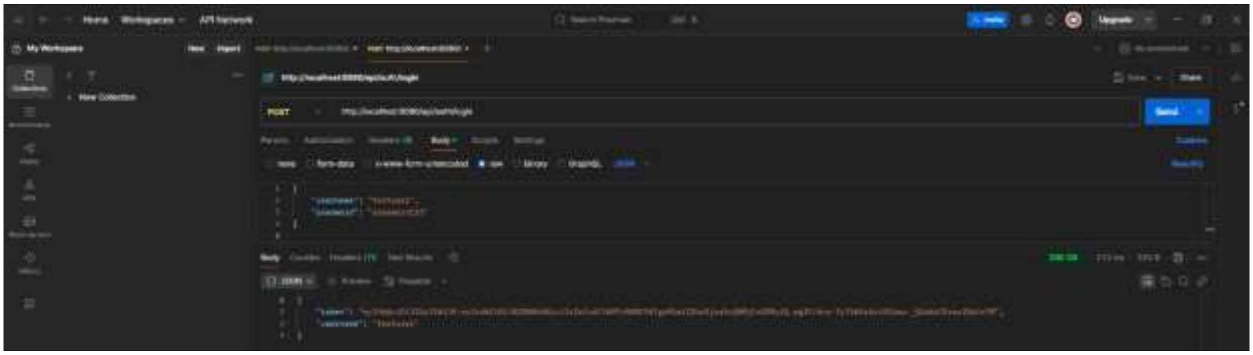
POST http://localhost:8080/api/auth/register

```
{
    "username": "testuser",
    "password": "password123"
}
```

200 OK

```
User registered successfully
```

- **POST /api/auth/login:**
  - Request body: { "username": "testuser", "password": "password123" }
  - Expected response: 200 OK, { "token": "...", "username": "testuser" }
  - Postman screenshot: include a screenshot of the request and response.



POST http://localhost:8080/api/auth/login

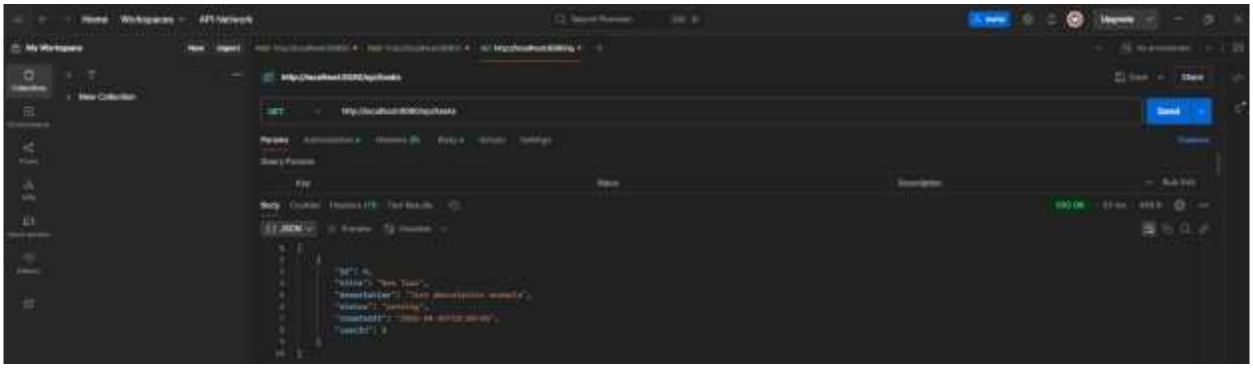
```
{
    "username": "testuser",
    "password": "password123"
}
```

200 OK

```
{
    "token": "[illegible]",
    "username": "testuser"
}
```

Task Controller:

- **GET /api/tasks:**
  - Headers: Authorization: Bearer <JWT_TOKEN>
  - Expected response: 200 OK, [{ "id": 1, "title": "...", ... }]
  - Postman screenshot: include a screenshot of the request and response.

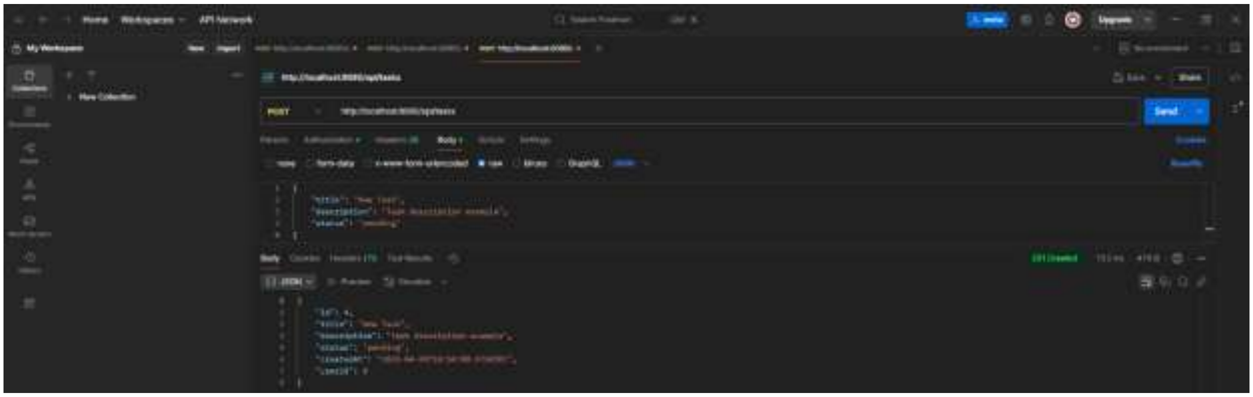


GET http://localhost:8080/api/tasks

200 OK

```
[
    {
        "id": [illegible],
        "title": "New Task",
        "description": "Task description example",
        "status": "pending",
        "createdAt": "[illegible]",
        "userId": [illegible]
    }
]
```

- **POST /api/tasks:**
  - Headers: Authorization: Bearer <JWT_TOKEN>
  - Request body: { "title": "New Task", "description": "...", "status": "TODO" }
  - Expected response: 201 Created, { "id": 2, "title": "New Task", ... }
  - Postman screenshot: include a screenshot of the request and response.



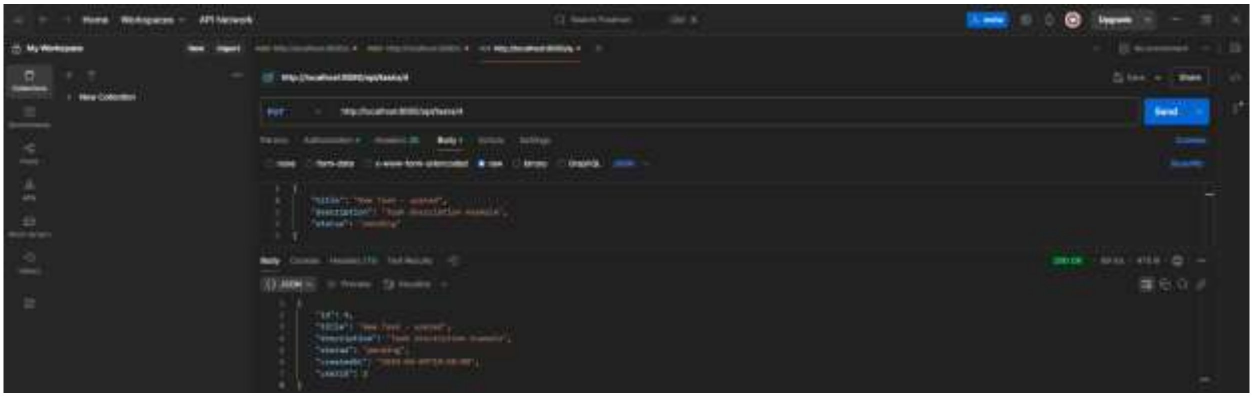
POST http://localhost:8080/api/tasks

```
{
    "title": "New Task",
    "description": "Task description example",
    "status": "pending"
}
```

201 Created

```
{
    "id": [illegible],
    "title": "New Task",
    "description": "Task description example",
    "status": "pending",
    "createdAt": "[illegible]",
    "userId": [illegible]
}
```

- **PUT /api/tasks/{id}:**
    - Headers: Authorization: Bearer <JWT_TOKEN>
    - Request body: { "title": "Updated Task", "description": "...", "status": "DONE" }
    - Expected Response: 200 OK, {"id": 2, "title": "Updated Task", ...}
    - Postman screenshot: include a screenshot of the request and response.



http://localhost:8080/api/tasks/4

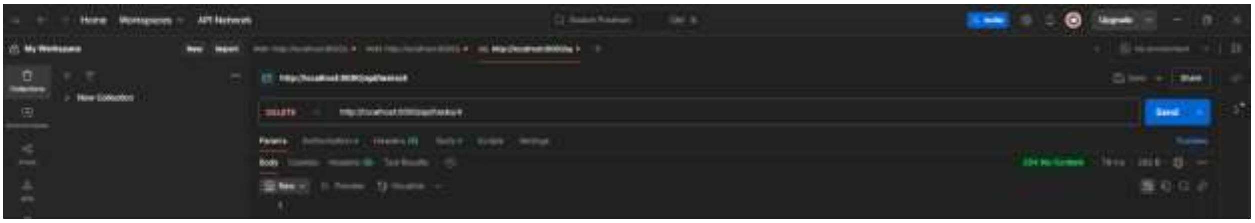
PUT http://localhost:8080/api/tasks/4

```
{
    "title": "New Task - updated",
    "description": "Task description example",
    "status": "pending"
}
```

200 OK

```
{
    "id": 4,
    "title": "New Task - updated",
    "description": "Task description example",
    "status": "pending",
    "createdAt": "[illegible]",
    "userId": 2
}
```

- **DELETE /api/tasks/{id}:**
    - Headers: Authorization: Bearer <JWT_TOKEN>
    - Expected response: 204 No Content
    - Postman screenshot: include a screenshot of the request and response.

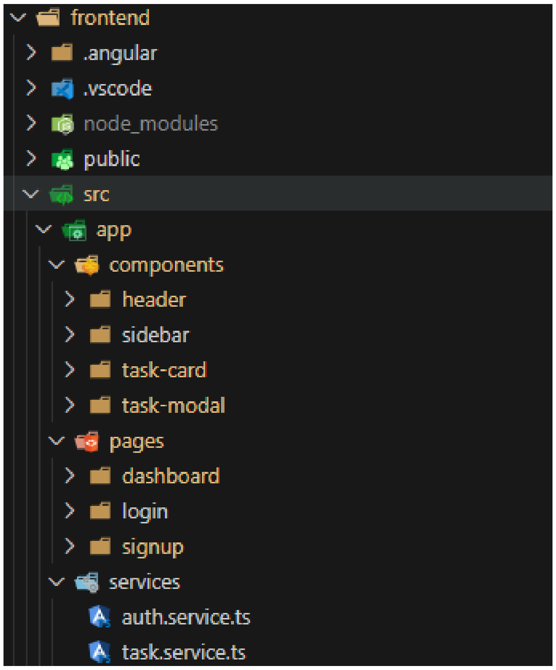


http://localhost:8080/api/tasks/4

DELETE http://localhost:8080/api/tasks/4

204 No Content

❖ Frontend :

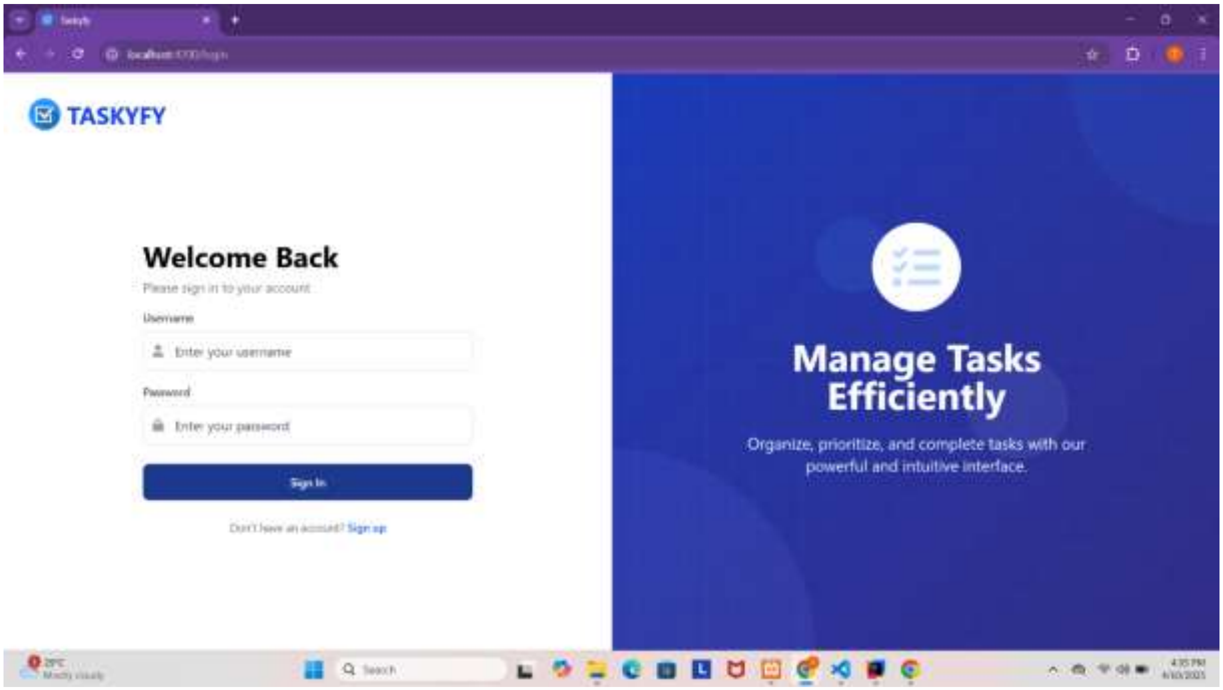


```
frontend
  .angular
  .vscode
  node_modules
  public
  src
    app
      components
        header
        sidebar
        task-card
        task-modal
      pages
        dashboard
        login
        signup
      services
        auth.service.ts
        task.service.ts
```

- **pages:** This directory contains the main screens or views of the application (e.g., dashboard, login, signup). Each folder within pages represents a distinct user interface route.
- **components:** This directory holds reusable UI elements (e.g., header, sidebar, task card, task modal). These are the building blocks that make up the pages and other parts of the application.
- **services:** This directory contains services that handle specific functionalities:
  - **auth.service:** Responsible for connecting to backend services related to user authentication (login, signup, etc.).
  - **task.service:** Responsible for connecting to backend services related to managing tasks (retrieving, creating, updating, deleting tasks).

Frontend UI

- The application implements a responsive design, ensuring a consistent and user-friendly experience across various screen sizes and devices (web & mobile). Below images are some pages of the app.

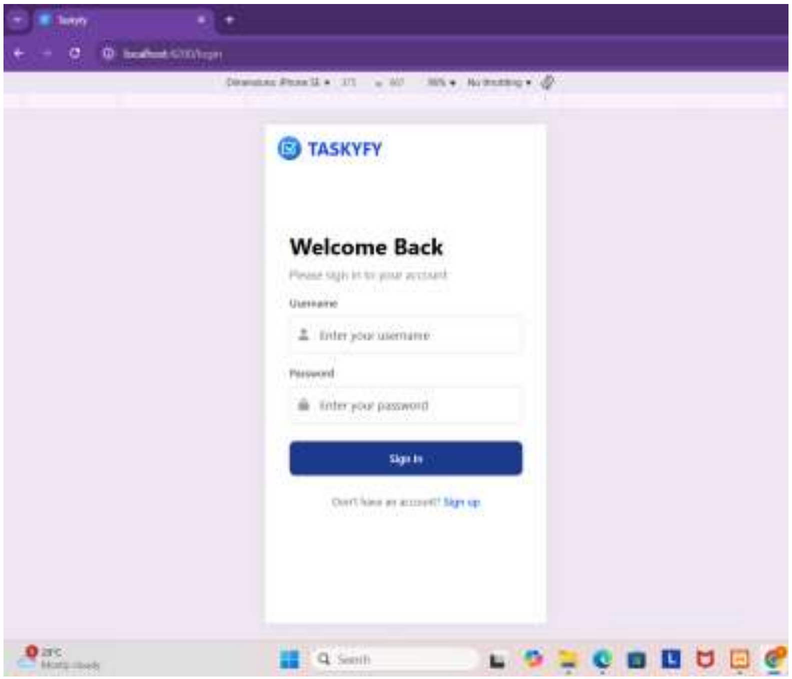


Welcome Back

Please sign in to your account

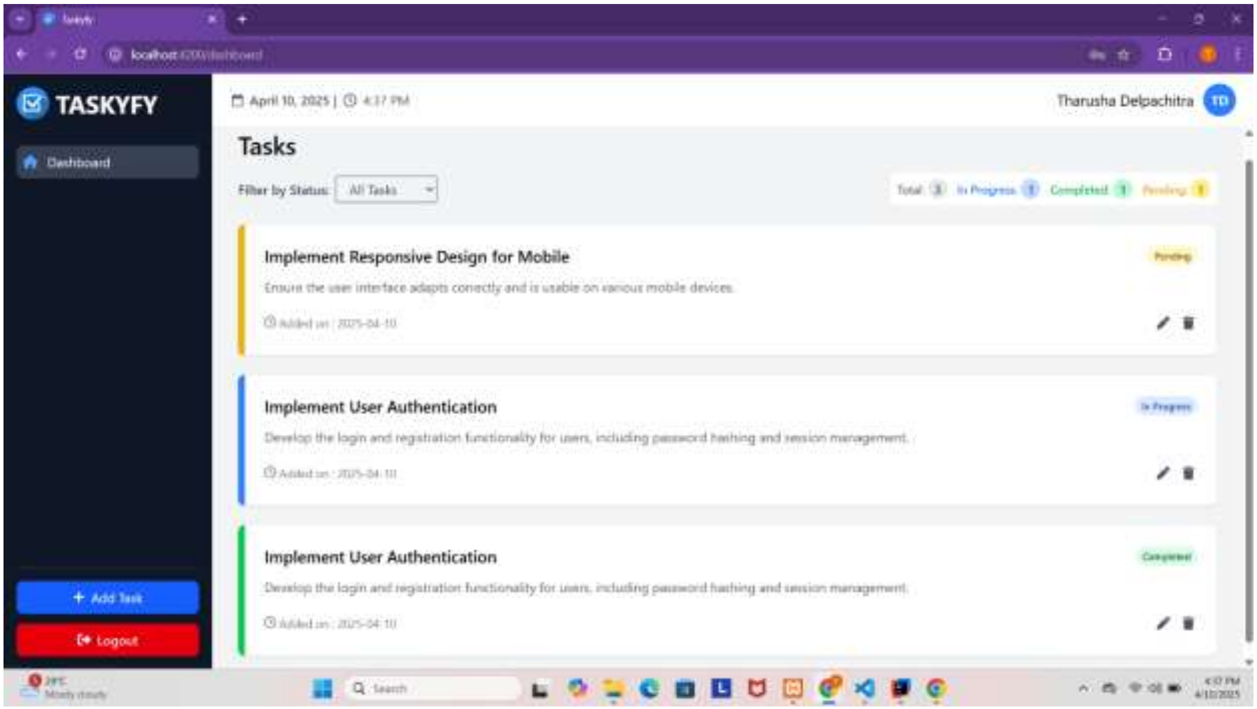
Manage Tasks Efficiently

Organize, prioritize, and complete tasks with our powerful and intuitive interface.



Welcome Back

Please sign in to your account



Tasks

Filter by Status: All Tasks

Total: 3 In Progress: 1 Completed: 1 Pending: 1

Implement Responsive Design for Mobile Pending

Ensure the user interface adapts correctly and is usable on various mobile devices.

Added on : 2025-04-10

Implement User Authentication In Progress

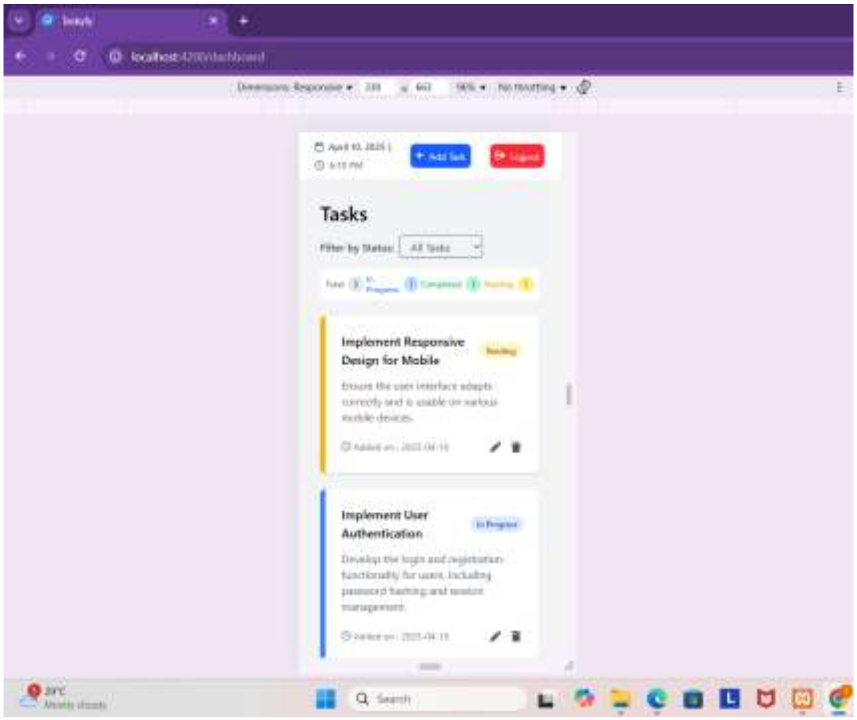
Develop the login and registration functionality for users, including password hashing and session management.

Added on : 2025-04-10

Implement User Authentication Completed

Develop the login and registration functionality for users, including password hashing and session management.

Added on : 2025-04-10



Tasks

Filter by Status: All Tasks

Total: 3 In Progress: 1 Completed: 1 Pending: 1

Implement Responsive Design for Mobile Pending

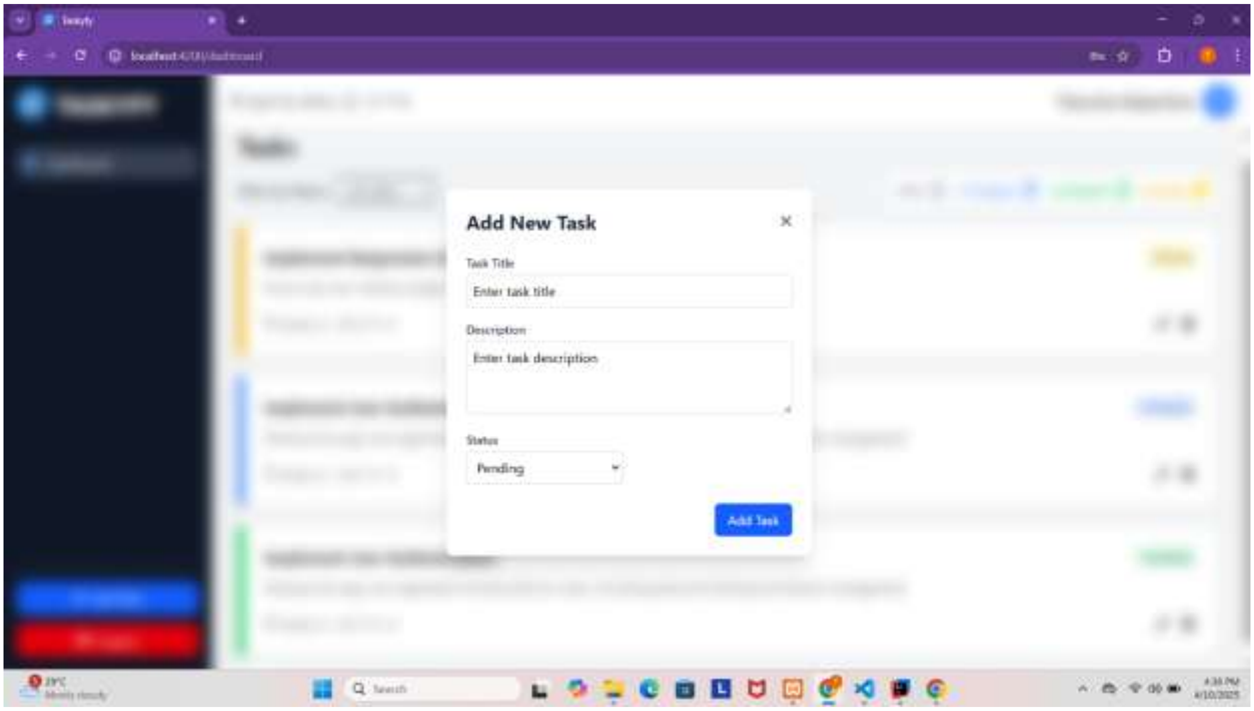
Ensure the user interface adapts correctly and is usable on various mobile devices.

Added on : 2025-04-10

Implement User Authentication In Progress

Develop the login and registration functionality for users, including password hashing and session management.

Added on : 2025-04-10



Add New Task

Task Title

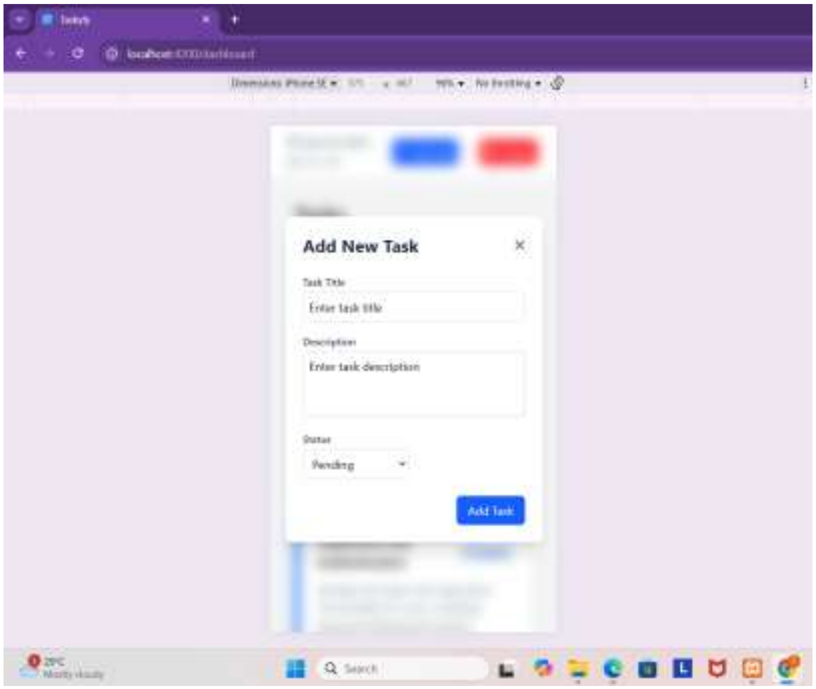
Enter task title

Description

Enter task description

Status

Pending



Add New Task

Task Title

Enter task title

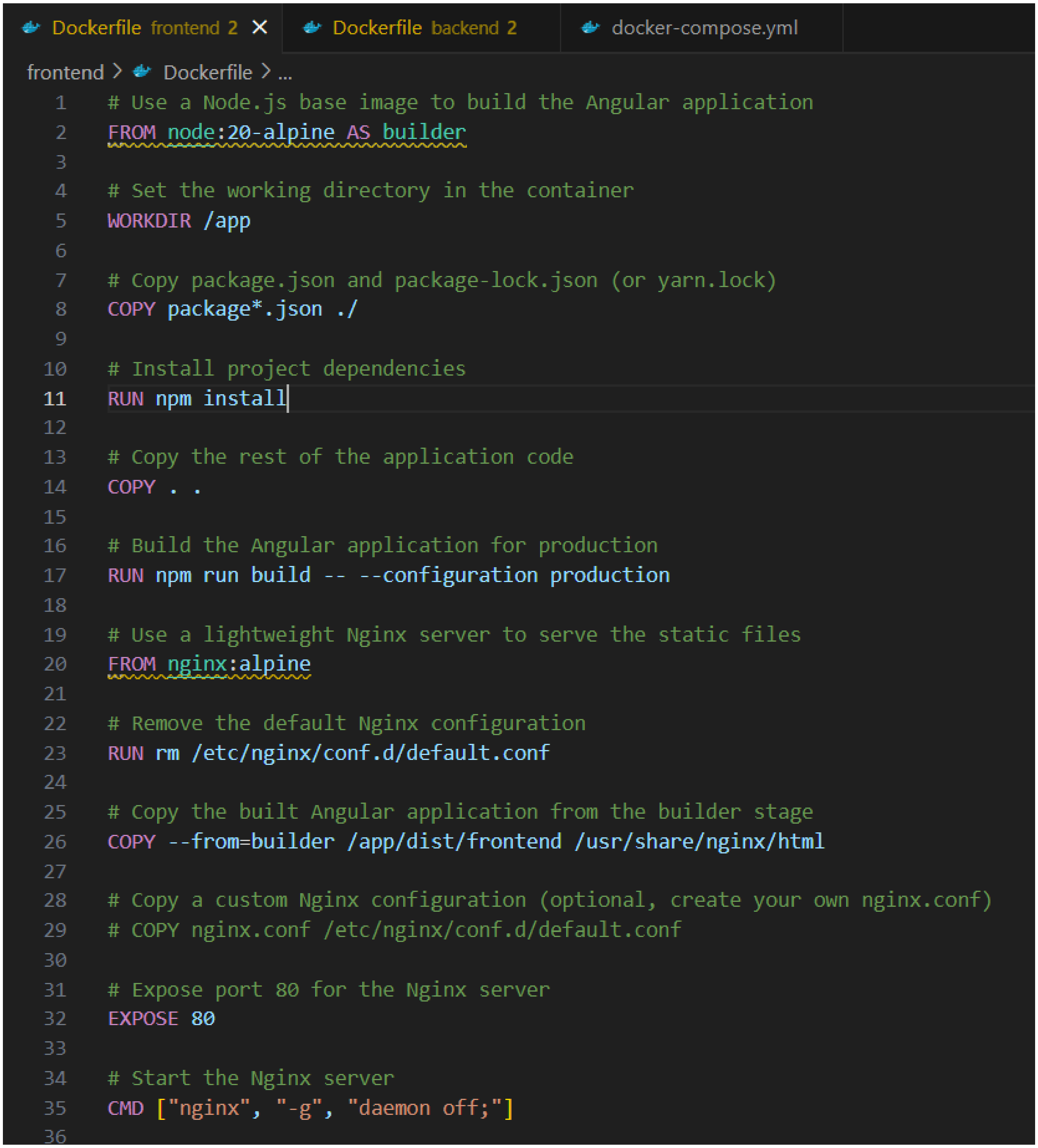
Description

Enter task description

Status

Pending

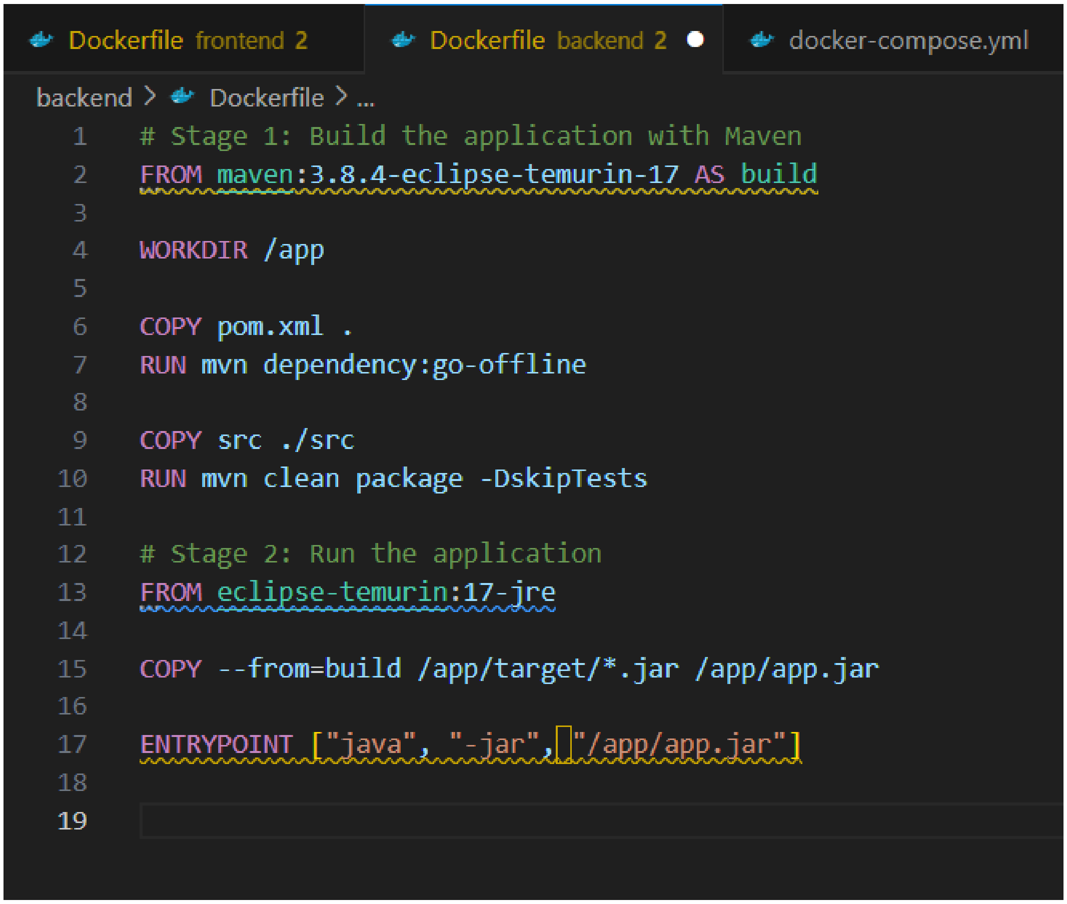
❖ Docker :



```
# Use a Node.js base image to build the Angular application
FROM node:20-alpine AS builder

# Set the working directory in the container
WORKDIR /app

# Copy package.json and package-lock.json (or yarn.lock)
COPY package*.json ./

# Install project dependencies
RUN npm install

# Copy the rest of the application code
COPY . .

# Build the Angular application for production
RUN npm run build -- --configuration production

# Use a lightweight Nginx server to serve the static files
FROM nginx:alpine

# Remove the default Nginx configuration
RUN rm /etc/nginx/conf.d/default.conf

# Copy the built Angular application from the builder stage
COPY --from=builder /app/dist/frontend /usr/share/nginx/html

# Copy a custom Nginx configuration (optional, create your own nginx.conf)
# COPY nginx.conf /etc/nginx/conf.d/default.conf

# Expose port 80 for the Nginx server
EXPOSE 80

# Start the Nginx server
CMD ["nginx", "-g", "daemon off;"]

```

Frontend Dockerfile:

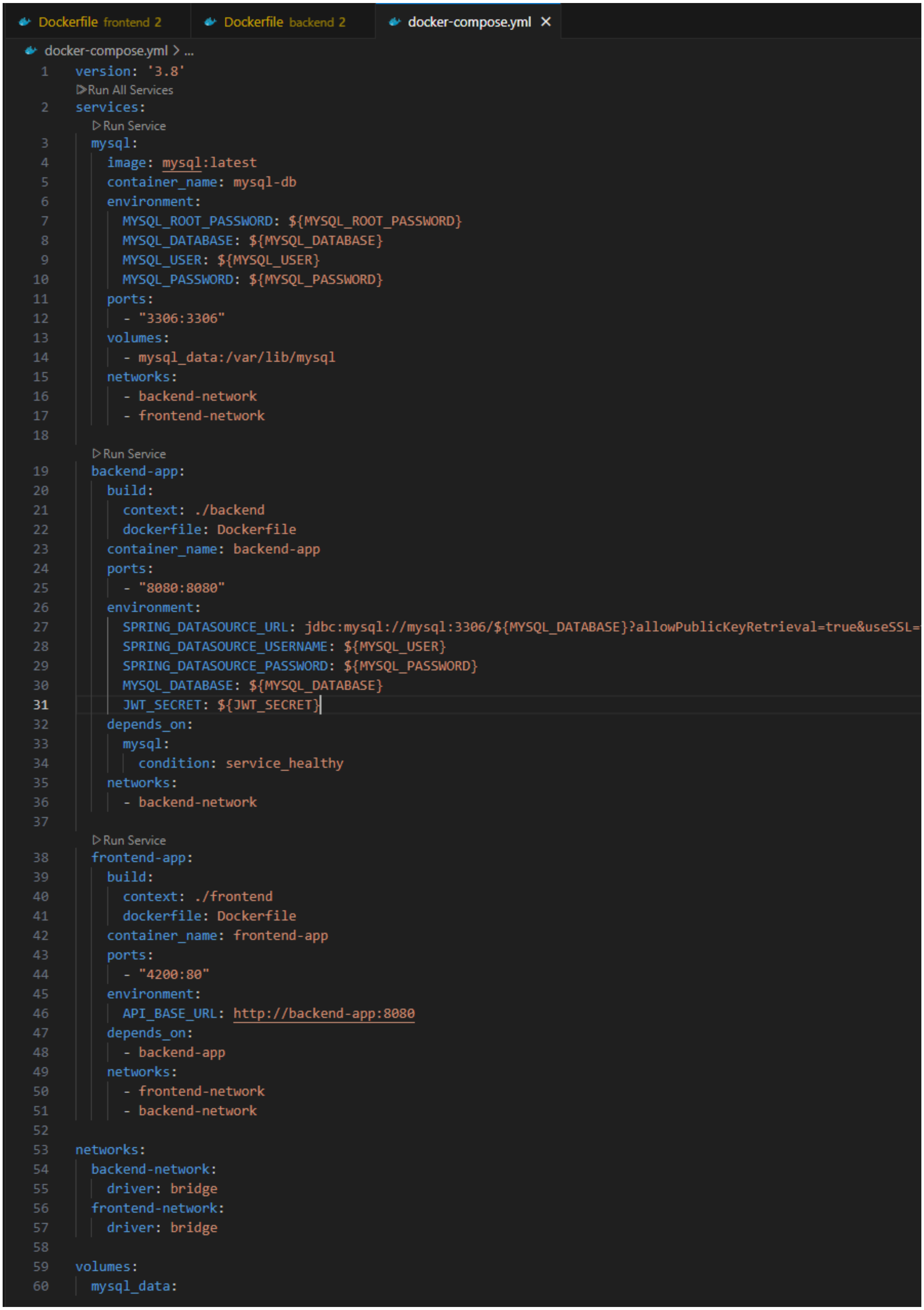
- **Builds Angular:** Uses Node.js to compile the Angular application into static files.
- **Serves with Nginx:** Uses a lightweight Nginx server to host the built static website.



```
# Stage 1: Build the application with Maven
FROM maven:3.8.4-eclipse-temurin-17 AS build

WORKDIR /app

COPY pom.xml .
RUN mvn dependency:go-offline

COPY src ./src
RUN mvn clean package -DskipTests

# Stage 2: Run the application
FROM eclipse-temurin:17-jre

COPY --from=build /app/target/*.jar /app/app.jar

ENTRYPOINT ["java", "-jar", "/app/app.jar"]

```

Backend Dockerfile:

- **Builds Java App:** Uses Maven to compile the Java backend application into a JAR file.
- **Runs Java App:** Executes the compiled JAR file using a Java runtime environment.

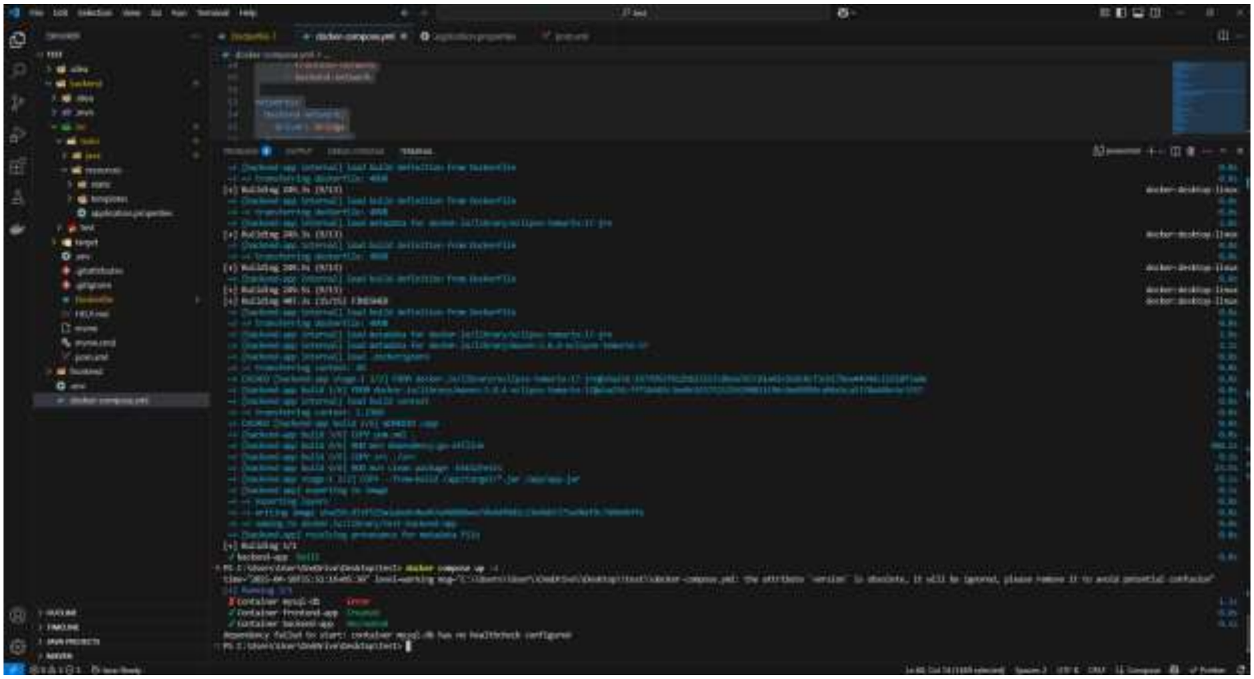


```yaml
version: '3.8'
services:
  mysql:
    image: mysql:latest
    container_name: mysql-db
    environment:
      MYSQL_ROOT_PASSWORD: ${MYSQL_ROOT_PASSWORD}
      MYSQL_DATABASE: ${MYSQL_DATABASE}
      MYSQL_USER: ${MYSQL_USER}
      MYSQL_PASSWORD: ${MYSQL_PASSWORD}
    ports:
      - "3306:3306"
    volumes:
      - mysql_data:/var/lib/mysql
    networks:
      - backend-network
      - frontend-network

  backend-app:
    build:
      context: ./backend
      dockerfile: Dockerfile
    container_name: backend-app
    ports:
      - "8080:8080"
    environment:
      SPRING_DATASOURCE_URL: jdbc:mysql://mysql:3306/${MYSQL_DATABASE}?allowPublicKeyRetrieval=true&useSSL=
      SPRING_DATASOURCE_USERNAME: ${MYSQL_USER}
      SPRING_DATASOURCE_PASSWORD: ${MYSQL_PASSWORD}
      MYSQL_DATABASE: ${MYSQL_DATABASE}
      JWT_SECRET: ${JWT_SECRET}
    depends_on:
      mysql:
        condition: service_healthy
    networks:
      - backend-network

  frontend-app:
    build:
      context: ./frontend
      dockerfile: Dockerfile
    container_name: frontend-app
    ports:
      - "4200:80"
    environment:
      API_BASE_URL: http://backend-app:8080
    depends_on:
      - backend-app
    networks:
      - frontend-network
      - backend-network

networks:
  backend-network:
    driver: bridge
  frontend-network:
    driver: bridge

volumes:
  mysql_data:
```

Docker Compose:

- **Orchestrates Services:** Defines and manages the MySQL database, backend application, and frontend application as separate containers.
- **Manages Networking & Dependencies:** Sets up communication between containers and ensures they start in the correct order (e.g., database before backend).

Implementation Issue: During the implementation of Docker Compose, I encountered an error that I was unable to resolve within the time. A screenshot of the error is included below.



```
[+] Building 1/1
 ✓ backend-app  Built
PS C:\Users\User\OneDrive\Desktop\test> docker compose up -d
time="[illegible]" level=warning msg="C:\\Users\\User\\OneDrive\\Desktop\\test\\docker-compose.yml: the attribute `version` is obsolete, it will be ignored, please remove it to avoid potential confusion"
[+] Running 3/3
 ✘ Container mysql-db      Error
 ✓ Container frontend-app  Created
 ✓ Container backend-app   Recreated
dependency failed to start: container mysql-db has no healthcheck configured
PS C:\Users\User\OneDrive\Desktop\test>
```