**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:**

**A line with numbers on it

Description automatically generated  
  
Users Table:**

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

**Tasks Table:**

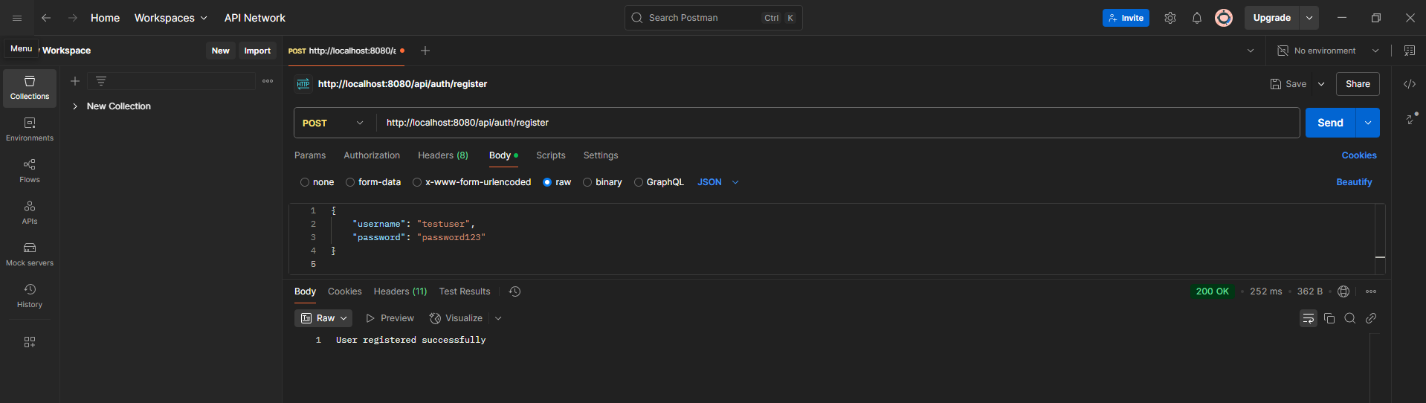
* id (PK): The primary key, uniquely identifying each task.
* title: The task's title.
* description: The task's description.
* status: The task's status (e.g., "To Do," "In Progress," "Done").
* created\_at: the time the task was created.
* user\_id (FK): The foreign key, linking each task to a specific 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.

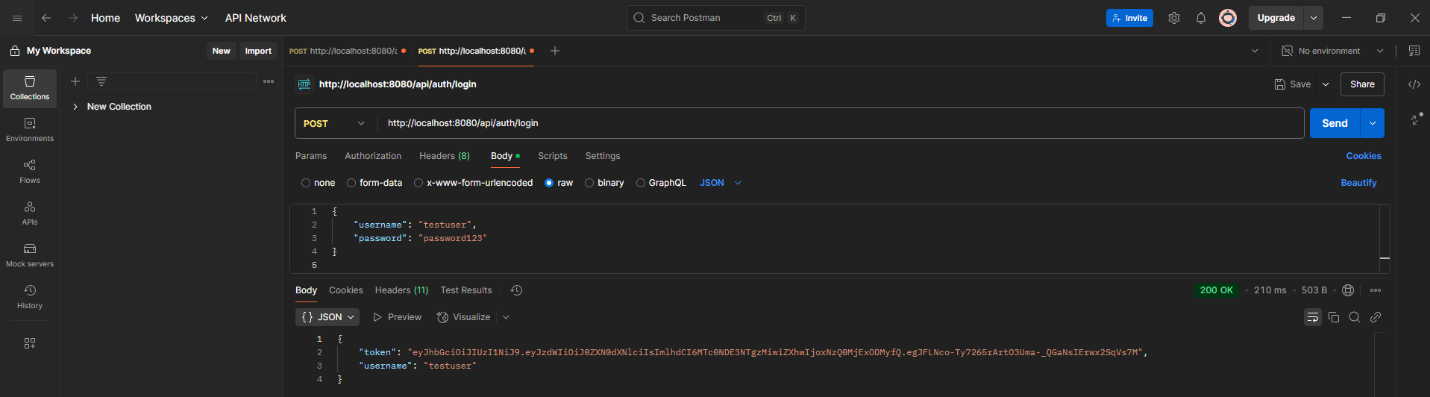
**3. API Endpoint Testing in Postman with Screenshots:**

**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.

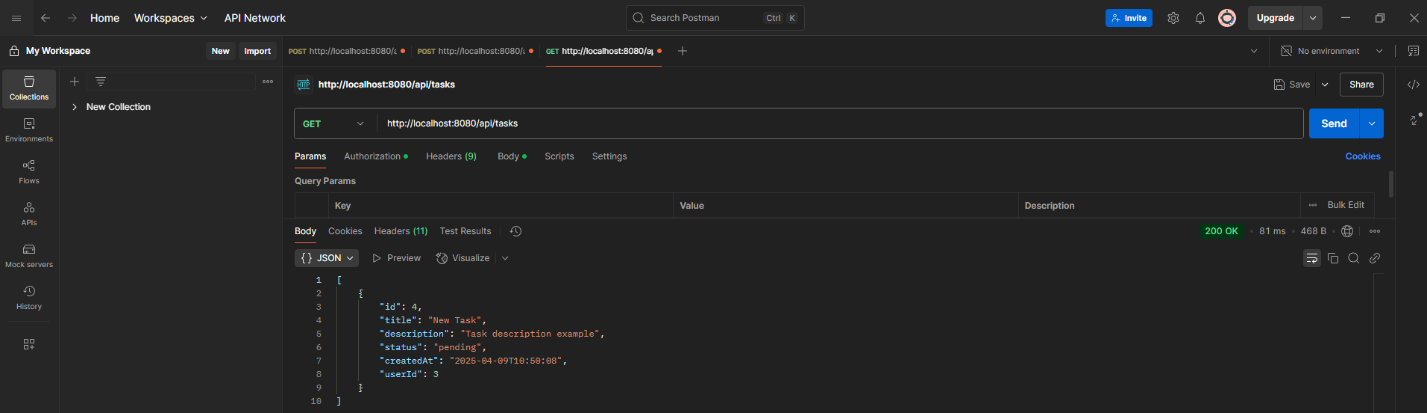


* **POST /api/auth/login:**
  + Request body: { "username": "testuser", "password": "password123" }
  + Expected response: 200 OK, { "token": "...", "username": "testuser" }
  + Postman screenshot: include a screenshot of the request and response.

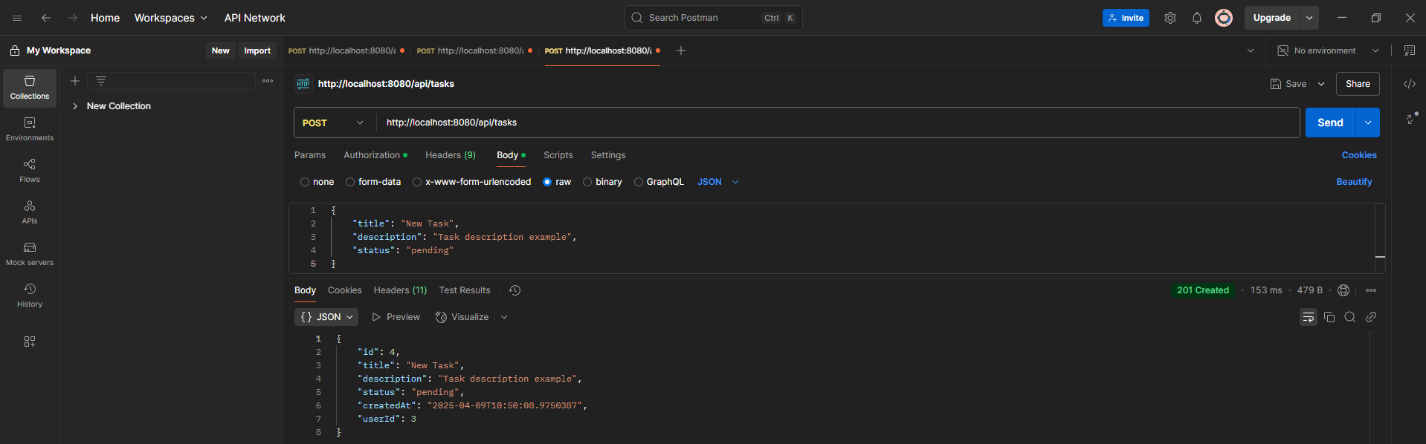


**Task Controller:**

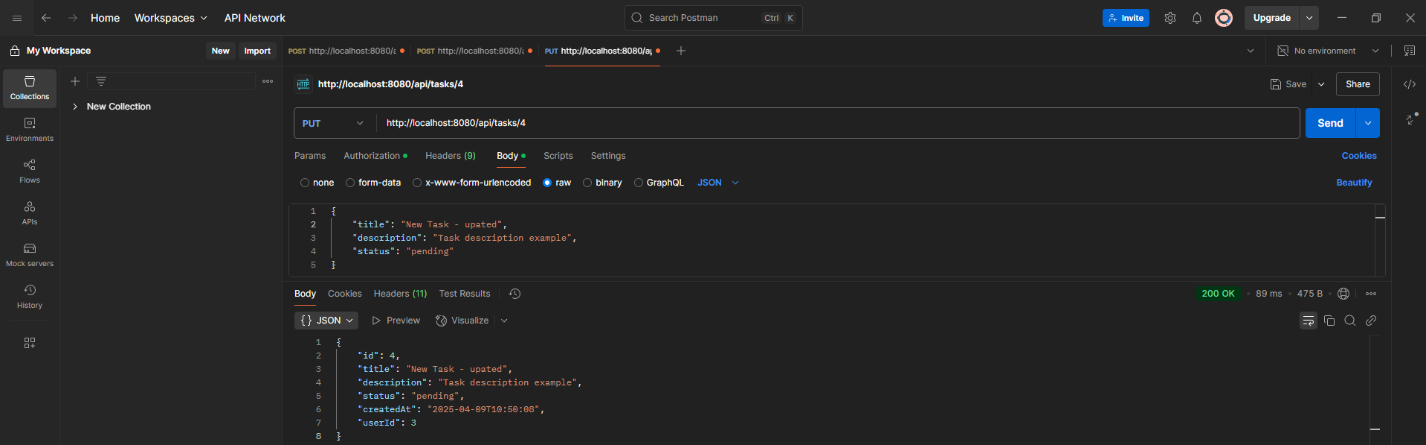
* **GET /api/tasks:**
  + Headers: Authorization: Bearer <JWT\_TOKEN>
  + Expected response: 200 OK, [ { "id": 1, "title": "...", ... } ]
  + Postman screenshot: include a screenshot of the request and response.



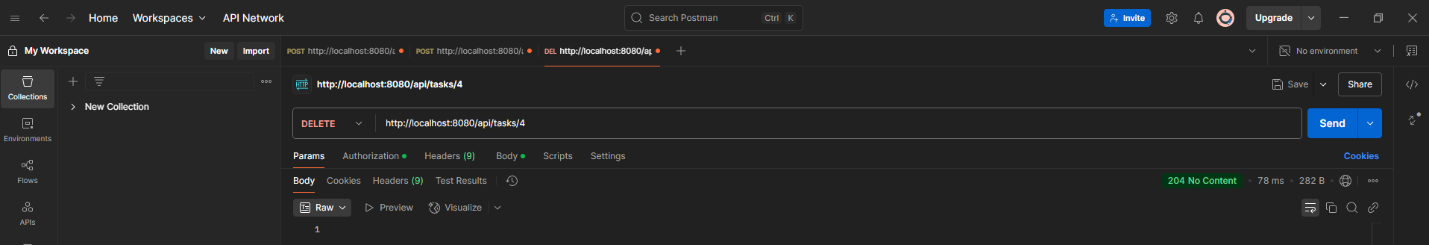
* **POST /api/tasks:**
  + Headers: Authorization: Bearer <JWT\_TOKEN>
  + Request body: { "title": "New Task", "description": "...", "status": "TODO" }
  + Expected response: 201 Created, { "id": 2, "title": "New Task", ... }
  + Postman screenshot: include a screenshot of the request and response.



* **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.



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



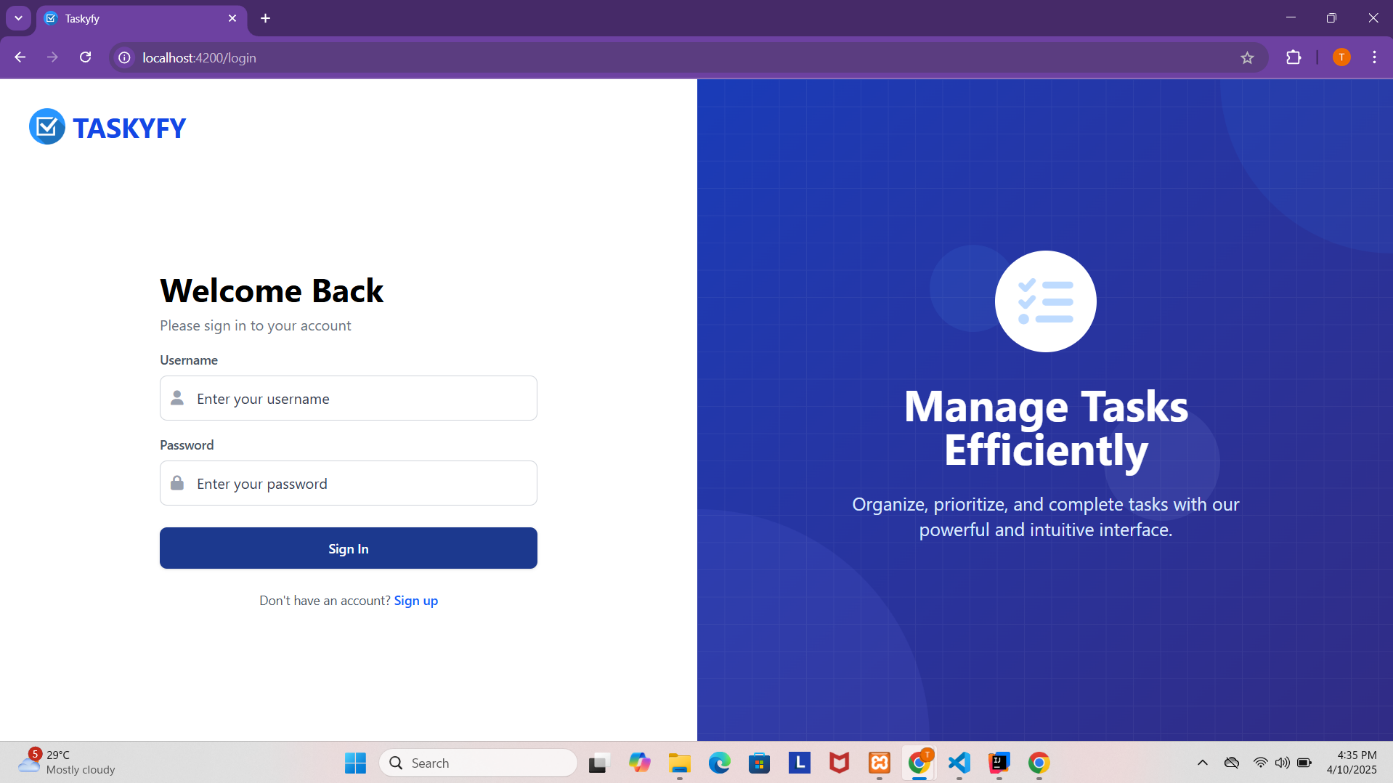
* **Frontend :**

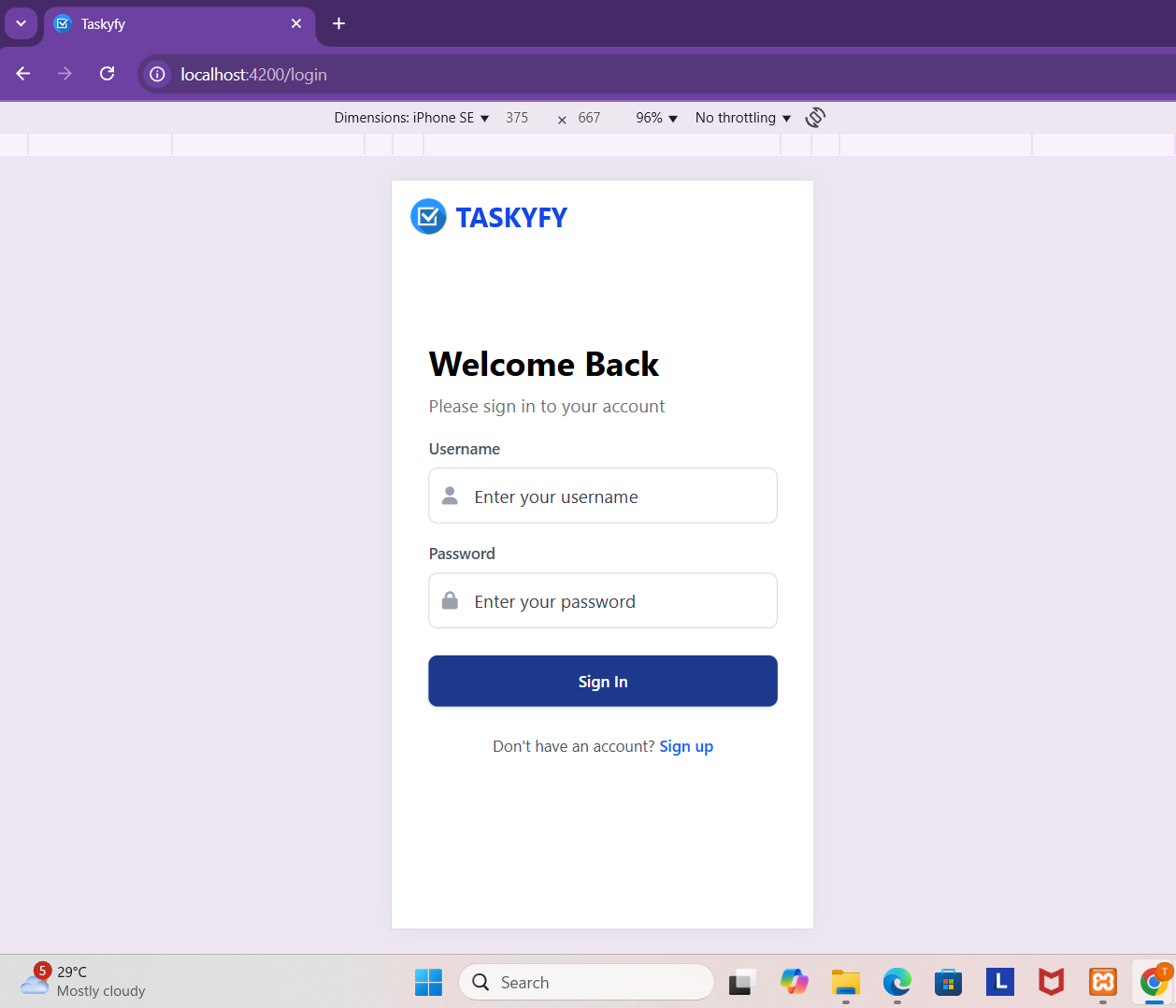
**A screenshot of a computer

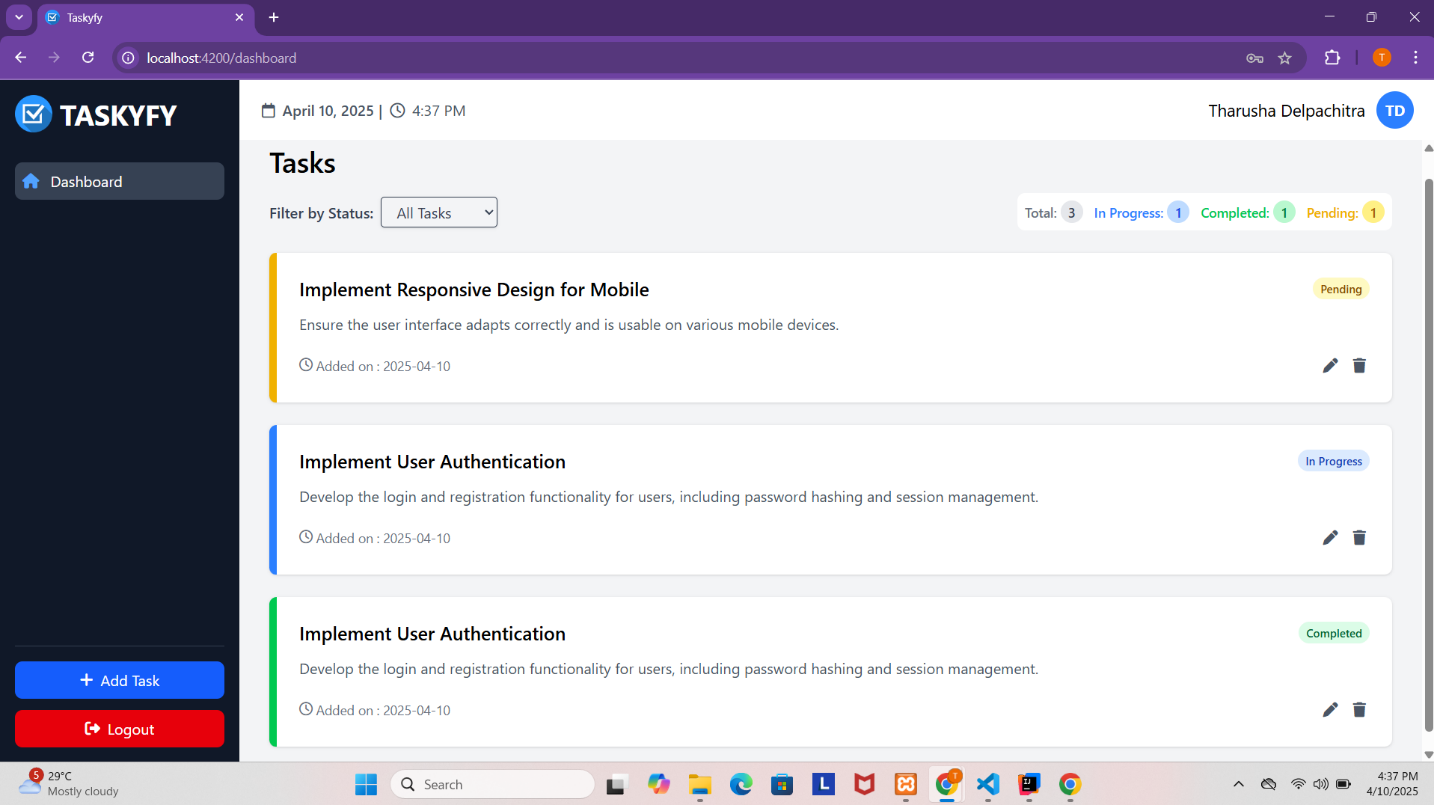
Description automatically generated**

* **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**

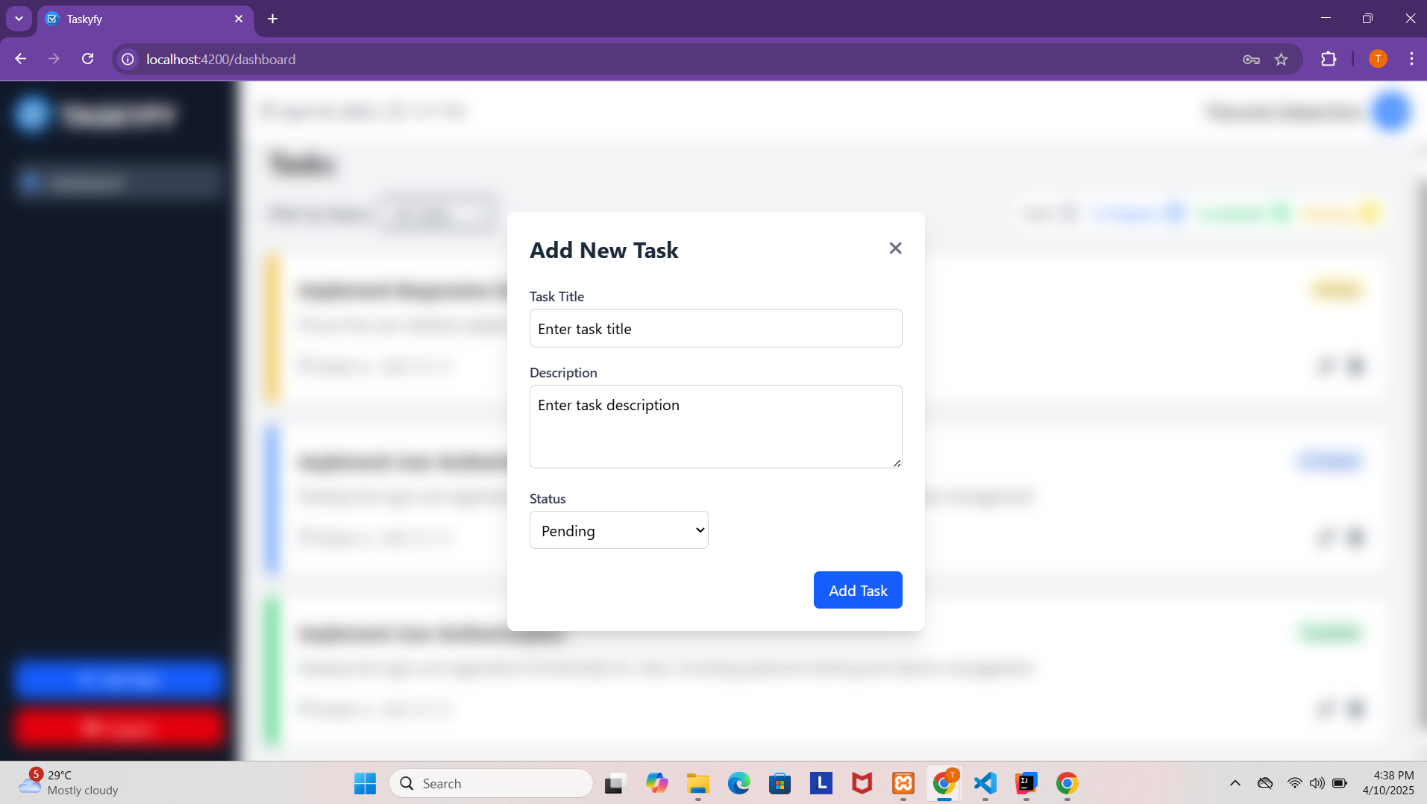
* ****The application implements a responsive design, ensuring a consistent and user-friendly experience across various screen sizes and devices (web & mobile). Below images are some pages of the app.

****

****

**A screenshot of a computer

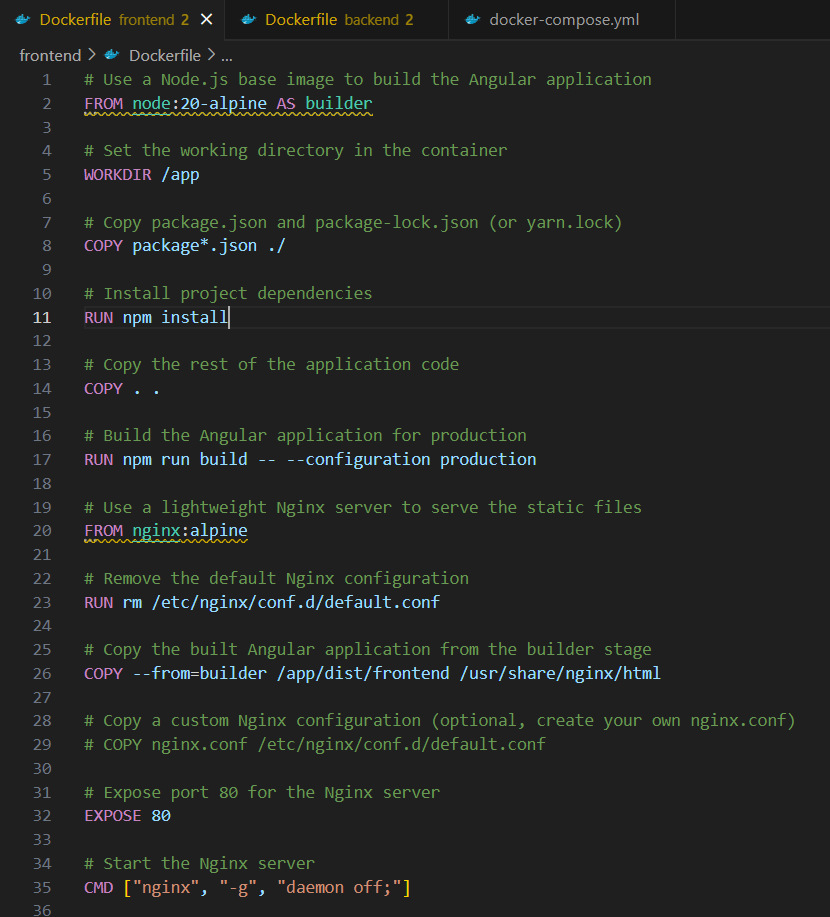
Description automatically generated**

****

**A screenshot of a computer

Description automatically generated**

* **Docker :**

****

**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.

**A screenshot of a computer program

Description automatically generated**

**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.

**A screen shot of a computer

Description automatically generated**

**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.

