Task Management App Assignment

Objective

Your task is to create a task management system with both backend and frontend components. The system will allow users to register, log in, and manage tasks (such as adding, updating, retrieving, and deleting tasks). The main functionality will be implemented via a REST API, which can be built using Flask or FastAPI. The tasks will be associated with individual authenticated users, ensuring that each user can only manage their own tasks. Additionally, you have the option to create a frontend using React to provide a user interface for interacting with the API.

The project will focus on key aspects of software development, including authentication, data handling, error management, and security. This assignment provides an opportunity to showcase your backend development skills, as well as your ability to design clean and maintainable code. For those who choose to implement the optional frontend, you will also be evaluated on your UI/UX and frontend design skills.

Part 1: Backend Development (Mandatory)

You must develop an API that allows users to manage their tasks.

Endpoints

The following endpoints are required:

1. User Registration
   - Endpoint: POST /register
   - Description: This endpoint registers a new user with a username and password.
   - Request Body:
     ```
     {
       "username": "user1",
       "password": "mypassword"
     }
     ```
   - Response:
     ```
     {
       "id": 1,
       "username": "user1"
     }
     ```
     Explanation: The response should include the user's ID and username, confirming successful registration.
2. User Login
   - Endpoint: POST /login
   - Description: This endpoint allows a registered user to log in using their username and password.
   - Request Body:
     ```
     {
       "username": "user1",
       "password": "mypassword"
     }
     ```
   - Response:
     ```
     {
       "access_token": "your_jwt_token_here"
     }
     ```
     Explanation: The response includes a JSON Web Token (JWT) or session token that the user must include in the request headers for authenticated operations.

3. Add a Task (Authenticated Users Only)
    - Endpoint: POST /tasks
    - Description: This endpoint allows an authenticated user to add a new task to their list.
    - Request Header: Include the authentication token in the Authorization header.
      ```
      Authorization: Bearer your_jwt_token_here
      ```
    - Request Body:
      ```
      {
        "description": "Buy groceries"
      }
      ```
    - Response:
      ```
      {
        "id": 1,
       "description": "Buy groceries",
        "completed": false,
        "user_id": 1
      }
      ```
      Explanation: The response includes the task ID, description, its completion status (defaulted to false), and the ID of the user who created the task.
4. Get All Tasks (Authenticated Users Only)
    - Endpoint: GET /tasks
    - Description: This endpoint returns all tasks belonging to the authenticated user.
    - Request Header: Include the authentication token in the Authorization header.
      ```
      Authorization: Bearer your_jwt_token_here
      ```
    - Response:
      ```
       [
        {
          "id": 1,
          "description": "Buy groceries",
          "completed": false,
          "user_id": 1
        }
      ]
      ```
      Explanation: The response is a list of tasks that belong to the authenticated user. Each task includes its ID, description, completion status, and the user ID.

5. Update a Task (Authenticated Users Only)
    - Endpoint: PUT /tasks/<id>
    - Description: This endpoint allows an authenticated user to update the description or completion status of a task.
    - Request Header: Include the authentication token in the Authorization header.
      ```
      Authorization: Bearer your_jwt_token_here
      ```
    - Request Body (partial updates allowed):
      ```
      {
        "description": "Buy groceries and fruits",
        "completed": true
      }
      ```
    - Response:
      ```
      {
        "id": 1,
        "description": "Buy groceries and fruits",
        "completed": true,
        "user_id": 1
      }
      ```
      Explanation: The response includes the updated task details, including its ID, updated description, completion status, and the user ID.
6. Delete a Task (Authenticated Users Only)
    - Endpoint: DELETE /tasks/<id>
    - Description: This endpoint allows an authenticated user to delete a task.
    - Request Header: Include the authentication token in the Authorization header.
      ```
      Authorization: Bearer your_jwt_token_here
      ```
    - Response:
      ```
      {
        "message": "Task deleted successfully"
      }
      ```
      Explanation: The response confirms the task has been deleted.

Database

- Use SQLite for storing user credentials and tasks.
- Each user should have their own set of tasks.
- Tasks should have the following fields:
    i. ID (integer, auto-increment)
    ii. User ID (foreign key to the users table)
    iii. Description (string)
    iv. Completed (boolean, defaults to false)

Error Handling

Implement error handling for:

- Invalid login credentials.
- Accessing tasks without authentication.
- Accessing tasks that belong to another user.
- Non-existing tasks.

Part 2: Frontend Development with React (Optional)

Build a simple frontend using React to interact with the backend API.

1. Register and Login Forms
    - Forms for user registration and login.
    - Upon successful login, store the authentication token and use it for subsequent API calls.
2. Task Management
    - A form to add new tasks.
    - Display a list of all the user's tasks.
    - Update task descriptions and mark tasks as complete or incomplete.
    - Delete tasks from the list.
3. UI Design (Optional)
    - Apply basic CSS styling to make the interface user-friendly and visually appealing.

Submission Instructions

- Submit your code via a GitHub repository.
- Include a README file with clear instructions on how to set up and run the backend and (if applicable) the frontend.

Important Points to Focus On

Your assignment will be evaluated based on several criteria. Please ensure you pay attention to the following aspects:

1. Code Functionality:

   Make sure your API meets the assignment requirements (user registration, login, task management) and works as expected.

2. Code Quality:
   - Write clean, readable, and maintainable code.
   - Use proper naming conventions and structure your code logically.
3. Documentation:
   - Provide a clear and concise README that explains how to set up and run your project.
   - Document your code where necessary (e.g., complex functions or logic).
4. Error Handling:
   - Implement basic error handling, especially for user inputs and authentication.
   - Ensure your application handles invalid requests gracefully.
5. Security:
   - Pay attention to user authentication and secure password handling (e.g., password hashing).
   - Make sure only authorized users can access their tasks.

Verification Tests

As part of the evaluation process, we have provided a set of minimal verification tests that you can use to verify whether your implementation works as expected. These tests are not comprehensive, but they ensure that all the main endpoints in your API function correctly. Below is an explanation of the provided tests:

How to Run the Tests

1. Install pytest and httpx, You'll need these libraries to run the tests. You can install them using:

   ```
   pip install pytest httpx
   ```

2. Test Commands:
   - Ensure your server is running locally on http://127.0.0.1:8000. If you are using a different port, adjust the BASE_URL in the test file.
   - Once the server is running, run the tests using:

     ```
     pytest test_ver.py
     ```

Why Use These Tests?

These tests are designed to:

- Validate basic functionality: Each of the required endpoints is tested to ensure they work correctly.
- Provide quick feedback: You can run the tests to immediately see if there are any issues with your implementation.
- Focus on core features: The tests cover user registration, authentication, task creation, retrieval, updating, and deletion, which are the core requirements for the task management API.
- These tests are not comprehensive and do not cover edge cases or error handling in detail, but they will ensure that your solution meets the basic functional requirements.

Reading Materials and Resources

- Flask:
  - Official Flask Documentation – The official Flask documentation is a great starting point to learn how to build REST APIs with Flask.
  - Flask-JWT-Extended Documentation – Learn how to implement JWT-based authentication in Flask.
- FastAPI:
  - Official FastAPI Documentation – FastAPI's documentation includes excellent resources for building APIs with modern Python frameworks.
  - FastAPI OAuth2 and JWT Authentication – A step-by-step guide for implementing OAuth2 and JWT in FastAPI.
- SQLAlchemy (For Flask and FastAPI):
  - SQLAlchemy Documentation – SQLAlchemy is a popular ORM for working with databases in Python. This guide explains how to use it to interact with SQLite or other databases.
- SQLite (Database)
  - SQLite Python Tutorial – This is a good starting point for learning how to use SQLite with Python. It's lightweight and works well with small to medium-scale applications.

Deadline

Submit the assignment within 72 hours.