Technical Exam for API Specialist

Duration: 4 hours

Instructions:

1. **Objective:** Develop a Python application that demonstrates your understanding of core Python fundamentals, object-oriented programming (OOP) principles, RESTful API development using FastAPI, role-based authorization, and database interaction using MongoDB or PostgreSQL.

2. **Tools & Technologies:**

   - **Programming Language:** Python 3.x
   - **Web Framework:** FastAPI
   - **Database:** Choose either MongoDB or PostgreSQL
   - **Database Libraries:**
     - For MongoDB: `pymongo`
     - For PostgreSQL: `psycopg2` or `asyncpg` with `aiopg`
   - **Version Control:** Git (please provide a Git repository link with your code)
   - **Optional (Bonus):** Creation of Data Flow Diagrams (DFDs) or other relevant diagrams to illustrate your application's architecture.

3. **Requirements:**

   - Design and implement a modular codebase following OOP best practices.
   - Showcase your problem-solving skills by implementing custom algorithms or data structures where appropriate.
   - Implement a RESTful API using FastAPI with role-based authorization using predefined users.
   - You can use AI tools such as code assistants and AI-driven searches for development, but ensure you understand the code you incorporate and maintain your own coding style. Avoid relying exclusively on AI-generated solutions.
   - Include error handling and input validation.
   - Write clear and concise documentation and comments in your code.
   - Provide a **README** file with instructions on how to set up and run your application.
   - **Bonus:** Include Data Flow Diagrams (DFDs) or other diagrams that illustrate the architecture and flow of your application.

4. **Submission:**

   - Upload your code to a public or private Git repository (e.g., GitHub, GitLab).
   - If the repository is private, grant access to the following usernames/emails: [Insert usernames/emails].

    - Ensure that all necessary files are committed and pushed to the repository before the deadline.
    - Send an email with the repository link and any necessary credentials or instructions.

The Task:

Build a Task Management API with Role-Based Authorization

You are tasked with creating a **Task Management** RESTful API that allows users to manage their daily tasks with role-based authorization. The application should support the following functionalities through API endpoints:

1. **Create a new task.**
2. **Retrieve all tasks with optional filtering (e.g., by due date, priority, or status).**
3. **Retrieve a specific task by its ID.**
4. **Update a task's details.**
5. **Mark a task as completed.**
6. **Delete a task.**

Each task should have the following attributes:

- **Task ID** (unique identifier)
- **Title**
- **Description**
- **Due Date**
- **Priority Level** (e.g., Low, Medium, High)
- **Status** (e.g., Pending, In Progress, Completed)
- **Creation Timestamp**
- **Owner** (the user who created the task)

User Roles:

Implement role-based authorization using **predefined users** with two roles:

- **Admin**
- **User**

Predefined Users:

Use the following hardcoded users for authentication purposes:

1. **Admin User:**
    - **Username:** `admin`

    - **Role:** `admin`
2. **Regular User:**
    - **Username:** `user`
    - **Role:** `user`

Note: For simplicity, you do not need to implement full authentication (e.g., password verification). Instead, you can simulate authentication by accepting a username in the request headers and assigning the corresponding role.

Authorization Rules:

- **Admins** can:
    - Create, retrieve, update, and delete **any** tasks.
    - Access all API endpoints.
- **Users** can:
    - Create new tasks.
    - Retrieve tasks they own.
    - Update or delete tasks they own.
    - Cannot access or modify tasks owned by others.

Additional Requirements:

- **OOP Design:**
    - Create classes that represent the main entities (e.g., `Task`, `User`, `TaskManager`).
    - Use encapsulation to protect the internal state of your objects.
    - Implement methods that provide meaningful operations on your classes.
    - Ensure that your code reflects your personal understanding and coding style.
- **API Development with FastAPI:**
    - Implement RESTful API endpoints for all required functionalities.
    - Follow RESTful design principles for URL endpoints and HTTP methods.
    - Use appropriate HTTP status codes in your responses.
    - Include input validation using Pydantic models.

- **Role-Based Authorization:**
    - Simulate authentication by accepting a username in the request headers (e.g., `X-Username`).
    - Use the predefined users to assign roles based on the provided username.

    - Enforce authorization rules for each endpoint based on the user’s role.
    - Ensure that users cannot perform actions they are not authorized to do.
- **Data Structures & Algorithms:**
    - Choose appropriate data structures to manage tasks before persisting to the database.
    - Implement sorting and filtering logic for the task retrieval endpoint.
    - Optimize your code for efficiency and scalability.
- **Database Interaction:**
    - Interact with your chosen database (MongoDB or PostgreSQL) to persist tasks.
    - Design your database schema to efficiently store task data, including ownership information.
    - Write your own queries using the selected database library (`pymongo` for MongoDB or `psycopg2` for PostgreSQL).
    - Avoid using ORMs like SQLAlchemy.
- **Error Handling & Validation:**
    - Implement input validation for all API inputs using Pydantic models.
    - Use try-except blocks to handle potential exceptions.
    - Provide meaningful error messages and appropriate HTTP status codes.
- **Documentation & Code Style:**
    - Follow Python’s PEP 8 style guide for your code.
    - Use docstrings to document your classes, methods, and API endpoints.
    - Comment your code where necessary to explain complex logic.
    - Include API documentation using OpenAPI (automatically generated by FastAPI).
- **Bonus: Diagrams:**
    - Create Data Flow Diagrams (DFDs) or other diagrams (e.g., ER diagrams, sequence diagrams) to illustrate your application’s architecture, data flow, and components.
    - Include these diagrams in a `diagrams` folder in your repository or embed them in your README.

Evaluation Criteria:

- **Core Python Fundamentals:** Demonstrated understanding of Python syntax, data types, control structures, and standard libraries.
- **OOP Implementation:** Effective use of classes, objects, encapsulation, and other OOP concepts.
- **Problem-Solving Skills:** Ability to design algorithms and choose appropriate data structures.
- **API Development Skills:** Proficiency in creating RESTful APIs using FastAPI and handling HTTP requests/responses.
- **Role-Based Authorization:** Correct implementation of authorization rules based on user roles.
- **Code Quality:** Cleanliness, readability, and organization of code reflecting your personal coding style.
- **Functionality:** The API works correctly and meets all the specified requirements.
- **Database Interaction:** Proper use of database operations and efficient queries without relying on ORMs.
- **Error Handling:** Robustness of the application against invalid input and unexpected situations.
- **Documentation:** Clarity of the README file, code comments, and API documentation.
- **Integrating AI Tools Effectively:** If using AI tools, ensure effective integration of these resources to enhance development while maintaining your own understanding and coding style.
- **Creativity:** Unique solutions or features that go beyond the basic requirements.
- **Bonus Points:**
  - Inclusion and quality of Data Flow Diagrams (DFDs) or other diagrams.
  - Presence and quality of unit tests or API tests.

Guidelines:

1. **Setup Instructions:** In your **README**, include clear instructions on how to set up and run your application, including any dependencies that need to be installed.

2. **Database Configuration:**

    - Provide sample configuration files or scripts needed to set up the database schema.
    - Ensure that your application can be easily connected to a new database instance.
    - Clearly specify any environment variables or configuration settings needed.
3. **Data Persistence:**

    - All tasks should be persisted in the database. Upon restarting the application, the tasks should be available via the API endpoints.
4. **API Endpoints:**

    - **Authentication Simulation:**

        - Expect a header `X-Username` in each request containing the username (`admin` or `user`).
        - Use this username to determine the user's role and enforce authorization rules.
    - **Create Task:** `POST /tasks/`

        - **Access:** Admins and Users
        - The creator of the task should be set as the owner.
    - **Get All Tasks:** `GET /tasks/`

        - **Access:**
            - **Admins:** Can retrieve all tasks.
            - **Users:** Can retrieve only their own tasks.
        - Support query parameters for filtering (e.g., `status`, `priority`, `due_date`).
    - **Get Task by ID:** `GET /tasks/{task_id}`

        - **Access:**
            - **Admins:** Can retrieve any task.
            - **Users:** Can retrieve only their own tasks.
    - **Update Task:** `PUT /tasks/{task_id}`

        - **Access:**
            - **Admins:** Can update any task.
            - **Users:** Can update only their own tasks.

- **Mark Task as Completed:** `PATCH /tasks/{task_id}/complete`
    - **Access:**
        - **Admins:** Can mark any task as completed.
        - **Users:** Can mark only their own tasks as completed.
- **Delete Task:** `DELETE /tasks/{task_id}`
    - **Access:**
        - **Admins:** Can delete any task.
        - **Users:** Can delete only their own tasks.

5. **Input Validation:**
    - Use Pydantic models to define request and response schemas.
    - Validate all incoming data and provide informative error messages.
6. **Error Responses:**
    - Return appropriate HTTP status codes (e.g., 401 Unauthorized, 403 Forbidden, 404 Not Found, 400 Bad Request).
    - Include error messages in a consistent format.
7. **API Documentation:**
    - Utilize FastAPI's automatic documentation generation.
    - Ensure that all endpoints, models, and parameters are well-documented.
8. **Diagrams (Bonus):**
    - **Data Flow Diagrams (DFDs):** Illustrate how data moves through your application.
    - **Entity-Relationship (ER) Diagrams:** Show the relationships between different entities in your database schema.
    - **Sequence Diagrams:** Depict the sequence of operations for key functionalities.
    - **Include Diagrams:** Add these diagrams to a `diagrams` folder or embed them in your README.
    - **Tools:** You can use any diagramming tools (e.g., Draw.io, Lucidchart, Microsoft Visio) or hand-drawn diagrams scanned as images.
9. **Time Management:**
    - Be mindful of the time limit. Prioritize core functionalities first before adding optional features and the bonus diagrams.

Tips to Showcase Your Coding Style:

- **Expressive Variable and Method Names:** Use names that clearly convey purpose and make your code self-documenting.
- **Modularization:** Break down your code into smaller, reusable functions or methods.
- **Design Patterns:** If applicable, use design patterns to solve common problems.
- **Error Handling Best Practices:** Demonstrate thoughtful exception handling and user-friendly error messages.
- **Authorization Logic:** Implement clean and efficient authorization checks. Consider creating a reusable dependency or middleware in FastAPI for role-based access control.
- **Innovation:** Feel free to add any additional features or improvements that you think would enhance the application, such as:
  - Logging of actions performed by users.
  - Pagination for listing tasks.
  - Additional filters for task retrieval.
  - Implementing asynchronous database operations for improved performance.

Good Luck!

We are excited to see your approach to this task and look forward to reviewing your work. Remember, while AI tools can aid your development, the final code should be a reflection of your understanding and abilities.

If you have any questions or need clarification on the task, please reach out promptly within the allotted time.