FastAPI

Exercise 1: Basic FastAPI Blog Application

Objective: To build a basic FastAPI application for a blog.

Instructions:

1. **Blog Application:** You are going to build a simple blog application with two primary entities: Users and Posts.
2. **Data Storage:** Use a Python dictionary to simulate data storage. Each User and Post will be an entry in their respective dictionary. Remember, the data in these dictionaries will only persist while the server is running.
3. **User Model:** The User model should have the following fields:
    - `username`: a unique identifier for the user.
    - `email`: the email address of the user.
    - `password`: the password of the user.
4. **Post Model:** The Post model should have the following fields:
    - `title`: the title of the blog post.
    - `content`: the content of the blog post.
    - `author`: the username of the post's author.
5. **Users Route:** Implement the following endpoints for the User model:
    - `GET /users`: return a list of all users.
    - `POST /users`: add a new user.
    - `GET /users/{username}`: return a specific user.
    - `PUT /users/{username}`: update a specific user's information.
    - `DELETE /users/{username}`: delete a specific user.
6. **Posts Route:** Implement the following endpoints for the Post model:
    - `GET /posts`: return a list of all posts.
    - `POST /posts`: add a new post.
    - `GET /posts/{title}`: return a specific post.
    - `PUT /posts/{title}`: update a specific post's information.
    - `DELETE /posts/{title}`: delete a specific post.

FastAPI

Exercise 2: Authentication, Authorization, Access Control, and Permissions

Objective: To secure your blog application with user roles and JWT-based authentication.

Instructions:

1. **User Roles:** Each user should have a role of either "regular" or "admin". Regular users should only have the ability to view posts, while admin users should be able to create, update, and delete posts.
2. **JWT Authentication:** Implement a login endpoint for your users. When a user successfully logs in using their username and password, generate a JWT token for them and include it in the response. This token should be sent with subsequent requests in the Authorization header to authenticate the user.
3. **Registration Endpoint:** Implement an endpoint where new users can register. Make sure to hash the user's password before storing it. Do not store the plaintext password.
4. **Access Control:** Implement access control so that only authenticated users can access your endpoints. An unauthenticated request should result in an HTTP 401 Unauthorized status code. Regular users should only be able to access the `GET /posts` and `GET /posts/{title}` endpoints, while admin users should have access to all endpoints.

FastAPI

Exercise 3: Session and Cookies

Objective: To implement session management using cookies for user authentication.

Instructions:

1. **Session Management:** When a user logs in, create a session for them. The session should include the user's username and the login timestamp. Store this session information in your in-memory data structure.
2. **Cookies:** Upon successful login, send a cookie containing the session ID back to the client. The client will need to send this cookie with each subsequent request to authenticate the user.
3. **Logout Endpoint:** Implement a logout endpoint. When this endpoint is accessed, invalidate the user's session by removing it from your in-memory data structure and send a response to the client instructing it to delete the cookie.

FastAPI

Exercise 4: FastAPI vs Flask

Objective: To compare and contrast FastAPI with Flask.

Instructions:

1. **Comparison:** Write a brief report comparing FastAPI and Flask. Highlight the key differences in terms of performance, ease of use, and unique features. Mention which framework you find more intuitive and why.
2. **Usage Scenarios:** Discuss different scenarios where you might prefer to use FastAPI over Flask, or vice versa. Justify your choices.
3. **Code Examples:** Provide simple code examples showing how each framework would handle the creation of a blog post in your application. Specifically, demonstrate how to define a route for creating a post in both FastAPI and Flask.