Name: Bao Chau

Mav ID: 1001668479

PHONEBOOK API INSTRUCTION

- ## Description of project and how the code work:
  - This is a Python code for a RESTful API built with FastAPI that interacts with a SQLite database. The API allows users to add and retrieve and delete phonebook entries, which consist of a person's full name and phone number.
  - It uses regular expression (re) for data validation and SQLAlchemy as the ORM.
  - The application uses FastAPI, a Python web framework, for creating the REST API.
  - The SQLite database is used for storing the phone book data.
  - Files: main.py, test.py, testData.py, loginInfo.py, apiTest.py
  - ### Stack Used:
    - Python: Multipurpose programming language with rich library collection
    - fastapi: FastAPI is a modern, fast (high-performance) web framework for building APIs with Python 3.7+ based on standard Python type hints.
    - uvicorn: Uvicorn is a lightning-fast ASGI server implementation, using uvloop and httptools.
    - httpx: An HTTP client for Python, providing asynchronous operations, compatible with HTTP/1.1 and HTTP/2. Use for testing and consuming APIs in FastAPI applications.
    - pytest: A testing framework for Python that allows you to write simple and scalable test cases for unit and integration testing.
    - sqlalchemy: SQLAlchemy is a popular SQL toolkit and ORM for Python. It provides a set of high-level API to work with relational databases.
    - jose: A library for handling JSON Web Tokens (JWT) and other cryptographic operations. Used in authentication and authorization workflows.
    - passlib: A password hashing library for Python, supporting various secure hashing algorithms like bcrypt and Argon2, used to safely store and verify passwords.
    - python-multipart: A library for parsing multipart/form-data. Used in handling file uploads and form submissions in web applications.

        - pytest-asyncio: A plugin for pytest that adds support for testing asynchronous code. Allowing us to write and execute tests for async functions in FastAPI and other async frameworks.
    - **Data flow:**
        - Login: Login information provided -> log the user in (admin or readonly) -> Denied or Allowed -> call function (list, add, delete)
        - List: call list endpoint -> fetch current user -> fetch authorization -> fetch current session -> if allowed: fetch the list of phone number, if not: denied-> return the necessary information and status code -> log the action
        - Add: call endpoint with request body-> fetch current user -> fetch authorization -> fetch current session -> if allowed: perform operation, if not: denied-> return the necessary information and status code -> log the action
        - Delete by number/name: call endpoint with request body-> fetch current user -> fetch authorization -> fetch current session -> if allowed: perform operation, if not: denied-> return the necessary information and status code -> log the action
    - **Login information:**
        - Readonly (read):
            - Username: readonlyuser
            - Password: readonlypassword
        - Admin (read/write):
            - Username: adminuser
            - Password: adminpassword
- **Instructions for building, running software and unit tests:**
    - **Build Software:**
        - Open Visual Studio.
        - Click on "File" in the top left corner and select "Open Folder".
        - Navigate to the folder where your Python code is located and select it.
        - Open the "Terminal" tab in Visual Studio by clicking on "View" and then "Terminal".
        - Install python and check for the version (only if you don't have it):
            - Install python through: https://www.python.org/downloads/
            - Check python version: `python --version`
        - Create Virtual Environment:
            - `python -m venv venv `

        - ` venv\Scripts\activate`
    - Install the dependencies
        - `pip install fastapi uvicorn[standard] sqlalchemy jose passlib python-multipart`
    - (Optional) Install Testing Dependencies:
        - `pip install pytest httpx pytest-asyncio`
    - In the terminal, navigate to the folder containing your Python code.
    - In our case keep the main.py tab open.
- **Run Software:**
    - To run the app, type the command in terminal: `uvicorn main:app --reload`
    - Access the fastAPI Swagger UI: `http://127.0.0.1:8000/docs#/`
- **Run Unit Test:**
    - Testing environment must be setup first, refer to Build Software section
    - There are 2 test files, one for functions test and one for api test:
        - Run functions test: `pytest test.py`
        - Run api test: `pytest apiTest.py`
- **Docker Setup:**
    - Docker files are created and setup.
    - Build it using command: `docker build -t fastapi-app .`
    - Run the image: `docker run -d -p 8000:8000 fastapi-app`
    - Once build and run finishes, open browser window.
    - Access the fastAPI Swagger UI: `http://127.0.0.1:8000/docs

- **Assumptions made:**
    - Assume that the given test cases are the core focus and the app should pass all test cases for it
    - Assume that I will not be graded on the status code of login
    - Assume that other test cases will be similar to the given test cases
    - Assume that any testing tool will work for this assignment
    - Assume that the default case of all operation is denial, until given the proper authorization
    - Assume users will always try to submit harmful/malicious input
    - Assume that every operation that need authorization need logging, failed or not
    - Assume that performance is not a concern in this assignment
    - Assume login and test data files need to encryption/protection in this assignment
    - Assume that TA will use Docker and Docker Desktop to grade the project

- **Pros/Cons of the application:**
  - Pros
    - Simple and Readable: use fastapi, which provide a quick and easy way to create APIs.
    - Separation of concern: Different functions are designated to handle specific tasks such as authentication, database management, validation, logging, and endpoints, with each having its own dedicated handler.
    - Role-Based access control: Differentiates between users with read and read/write permissions for secure access.
    - Logging: Tracks API actions, creating a basic mechanism for accountability and debugging.

  - Cons
    - Bad Scalability: not design to scale, SQLite is not suitable for production/high-concurrency.
    - Not Production ready: Hardcoded data and configurations make it unsuitable for deployment.
    - Performance: The use of synchronous SQLAlchemy operations may limit performance under high loads.
    - Advance analytics: logs may expose personal data without encryption/protection.
    - UX: No frontend or client app is provided, limiting the usability of the app