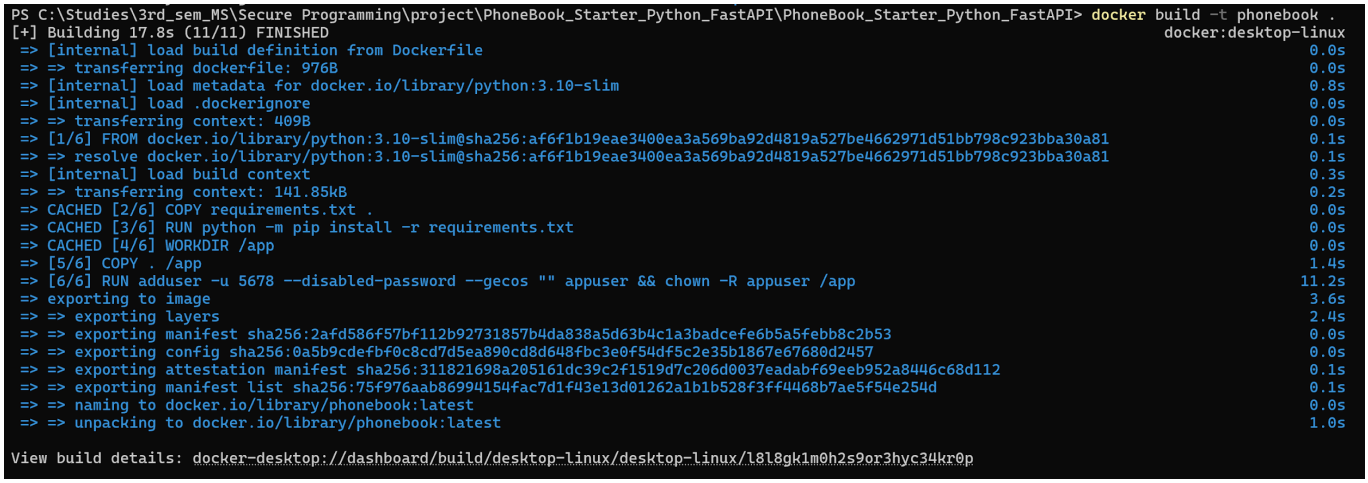
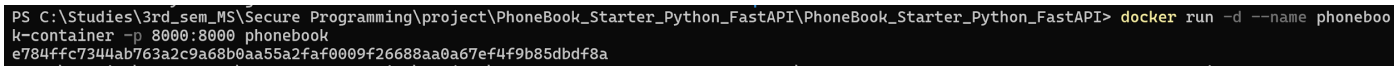
Secure PhoneBook REST API - Implementation Report

Instructions for Building and Running

1. Docker Integration:
    - Ensure you are in the correct project folder
    - Build: **docker build -t phonebook .**



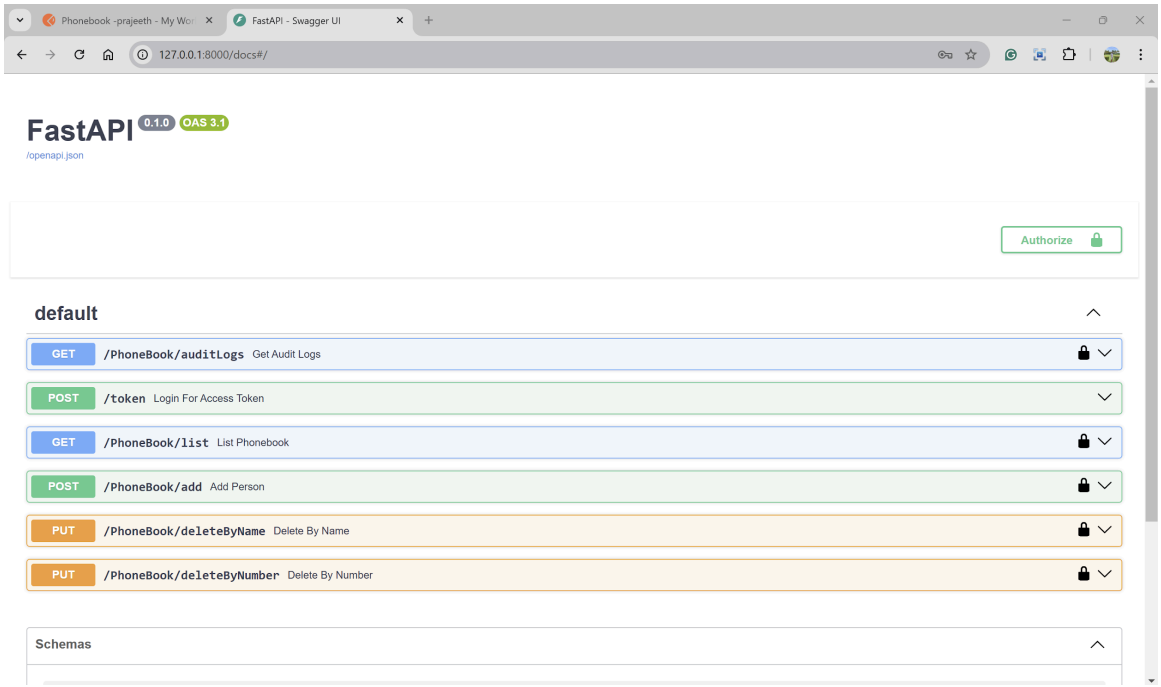
    - Run: **docker run -p 8000:8000 phonebook-api.**



2. Setup:
    - Start the API using uvicorn app:app --reload.

```
(venv) C:\Studies\3rd_sem_MS\Secure Programming\project\PhoneBook_Starter_Python_FastAPI\PhoneBook_Starter_Python_FastAPI>uvicorn app:app --reload
INFO:     Will watch for changes in these directories: ['C:\\Studies\\3rd_sem_MS\\Secure Programming\\project\\PhoneBook_Starter_Python_FastAPI\\PhoneBook_Starter_Python_Fast
API']
INFO:     Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)
INFO:     Started reloader process [10160] using StatReload
2024-11-15 18:52:08,957 INFO sqlalchemy.engine.Engine BEGIN (implicit)
2024-11-15 18:52:08,957 INFO sqlalchemy.engine.Engine PRAGMA main.table_info("users")
2024-11-15 18:52:08,957 INFO sqlalchemy.engine.Engine [raw sql] ()
2024-11-15 18:52:08,957 INFO sqlalchemy.engine.Engine PRAGMA main.table_info("phonebook")
2024-11-15 18:52:08,957 INFO sqlalchemy.engine.Engine [raw sql] ()
2024-11-15 18:52:08,957 INFO sqlalchemy.engine.Engine COMMIT
2024-11-15 18:52:08,957 INFO sqlalchemy.engine.Engine BEGIN (implicit)
2024-11-15 18:52:08,957 INFO sqlalchemy.engine.Engine PRAGMA table_info(users)
2024-11-15 18:52:08,957 INFO sqlalchemy.engine.Engine [generated in 0.00033s] ()
2024-11-15 18:52:08,957 INFO sqlalchemy.engine.Engine ROLLBACK
2024-11-15 18:52:08,957 INFO sqlalchemy.engine.Engine BEGIN (implicit)
2024-11-15 18:52:08,957 INFO sqlalchemy.engine.Engine SELECT users.id AS users_id, users.username AS users_username, users.hashed_password AS users_hashed_password, users.rol
e AS users_role
FROM users
WHERE users.username = ?
 LIMIT ? OFFSET ?
2024-11-15 18:52:08,957 INFO sqlalchemy.engine.Engine [generated in 0.00028s] ('read_user', 1, 0)
2024-11-15 18:52:08,957 INFO sqlalchemy.engine.Engine SELECT users.id AS users_id, users.username AS users_username, users.hashed_password AS users_hashed_password, users.rol
e AS users_role
FROM users
WHERE users.username = ?
 LIMIT ? OFFSET ?
2024-11-15 18:52:08,957 INFO sqlalchemy.engine.Engine [cached since 0.001759s ago] ('write_user', 1, 0)
2024-11-15 18:52:08,957 INFO sqlalchemy.engine.Engine COMMIT
INFO:     Started server process [12940]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
```

3. Execution:
    - Access endpoints at http://127.0.0.1:8000.



4. Unit Tests:

- Place API inputs into a Postman collection.
- Validate both positive and negative test cases using provided sample inputs.

Code Overview

Overview

This code implements a secure REST API using FastAPI to manage a phone book. It includes:

1. **User Authentication:** JWT-based login system.
2. **Role-based Authorization:** Supports two roles:
    - **Read:** Can only list phonebook entries.
    - **Read/Write:** Can add, delete, and list entries.
3. **Data Validation:** Ensures valid names and phone numbers using regular expressions.
4. **Logging:** Tracks user actions (e.g., add, delete) in an audit log.
5. **Database Persistence:** Uses SQLite for storing data.

Key Features

1. **Authentication with JWT:**
    - **Login:** /token endpoint generates a JWT token after verifying credentials.
    - **Token Validation:** Extracts user information and checks the token's validity.
2. **Role-based Access Control:**
    - **Roles:**
        - read: Access to list phonebook entries.
        - read_write: Access to all endpoints (add, delete, list).
    - Restricted endpoints (add, delete) check the user's role before performing actions.
3. **Input Validation:**

Name Validation: r"^(?:[A-Za-z]+(?:[
'\u2019-][A-Za-z]+)*,?\s+[A-Za-z]+(?:\s+[A-Z](?:[a-z]+|\.)?)?(?:\s+[A-Za-z]+(?:[
'\u2019-][A-Za-z]+)*)?|\b[A-Za-z]+\b)$"

Phone Number Validation: r"^(?:(?:\+(?!0)|00|011)?\s*([1-9]\d{0,2})?\s*[-. (]*(?!0)(\d{2,4})[-.)]*\s*\d{3}[-.]\d{4}|\d{5}|\d{5}[-.]\d{5}|[1-9]\d{2}-\d{4})$"

4. **Database Integration:**
    - **SQLite Database:**
        - users table: Stores user credentials and roles.
        - phonebook table: Stores phonebook entries (name, phone number).
    - SQLAlchemy is used to manage database operations.
5. **Audit Logging:**
    - Logs user actions (e.g., listing, adding, deleting entries) to phonebook_audit.log.

Endpoints

Endpoint	Role Required	Description	Error Codes
/token	None	Logs in user and returns a JWT token.	401 (Unauthorized)
/PhoneBook/auditLogs	read_write	Fetches the audit log.	403, 404, 500
/PhoneBook/list	read/read_write	Lists all phonebook entries.	401
/PhoneBook/add	read_write	Adds a new phonebook entry.	403, 400
/PhoneBook/deleteByName	read_write	Deletes an entry by name.	403, 404
/PhoneBook/deleteByNumber	read_write	Deletes an entry by phone number.	403, 404

Security Features

1. **Password Hashing:**
    - Uses bcrypt to securely hash passwords.
2. **Prepared Statements:**
    - SQL queries are parameterized to prevent SQL injection.
3. **Error Handling:**
    - Returns HTTP status codes:

- 400: Invalid input.
- 401: Unauthorized access.
- 403: Insufficient permissions.
- 404: Not found.
- 500: Internal server error.

How It Works

1. **Authentication:**
    - Users log in with /token to receive a JWT token.
    - Token is passed with requests for validation.
2. **CRUD Operations:**
    - list: Retrieve all entries.
    - add: Add a validated entry.
    - deleteByName: Remove entry by name.
    - deleteByNumber: Remove entry by phone number.
3. **Logging:**
    - Tracks all operations with timestamp, username, and action.
4. **Validation:**
    - Rejects invalid names or phone numbers.

Assumptions

1. Roles:
    - Role-based access control assumes no hierarchical structure; explicit role permissions are defined.
2. Input Validation:
    - Names and phone numbers are validated against specific formats.
    - International phone formats and special character names are supported.
3. Persistence:
    - SQLite was chosen for portability and ease of setup.
4. Security:
    - All passwords are hashed with bcrypt.
    - Parameterized queries are used to prevent SQL injection.
5. Concurrency:
    - Assumes limited simultaneous users, suitable for SQLite.

Pros and Cons

Pros:

1. Secure Authentication:
    - JWT tokens ensure robust, stateless user sessions.
2. Comprehensive Validation:
    - Prevents common input errors and injection vulnerabilities.
3. Portability:
    - SQLite-based persistence and Docker deployment make the solution lightweight.
4. Auditing:

    - Detailed log entries enhance accountability and troubleshooting.

Cons:

1. Limited Scalability:
    - SQLite may not handle high-concurrency scenarios efficiently.
2. Complex Regular Expressions:
    - Maintaining validation patterns could be challenging for diverse formats.

Bonus Features

1. Database Integration:
    - SQLite is used to persist entries and users.
2. Parameterized Queries:
    - SQL queries use prepared statements to enhance security.

Conclusion

This implementation addresses all assignment requirements:

- Authentication and role-based access control.
- Regular expression-based validation.
- Secure database access with prepared statements.
- Logging and auditing mechanisms.
- Comprehensive error handling.