

Richard Olu-Jordan

Nov 15th, 2024

CSE 4382 – Secure Programming

Prof. Trey Jones

Project - PhoneBook Starter Project Report

Table of Contents

1. Introduction
2. Assumptions
3. Project Setup Instructions
4. Detailed Feature Descriptions
5. File Descriptions
6. Postman Testing Guide
7. Automated Testing with Docker
8. Error Handling and Expected Error Messages
9. Test Cases and Postman Collection
10. Troubleshooting and Common Issues
11. Appendices

1. Introduction

This code implements a simple phone book API using FastAPI. It includes functionalities for adding, listing, and deleting phone book entries, with authentication and role-based access controls. The following components are used:

- **Database Models:** SQLAlchemy is used to define models for User and PhoneBook entries, each stored in a SQLite database.
- **Authentication:** JWT tokens are generated for registered users, providing access to protected endpoints.

2. Regular Expressions Design

Name Validation (full_name)

```
r"^(?:[A-Za-z]+(?:['-]?[A-Za-z]+),?\\s){1,3}(?:\\s[A-Z].)? $"
```

This regex allows:

- Alphabetical characters with optional hyphens (-) and apostrophes (') for names like "Ron O'Henry" and "Ron O'Henry-Smith-Barnes".
- A comma-separated format with spaces.
- Up to three name segments, with an optional middle initial.

Phone Number Validation (phone_number)

A general regex pattern (general_validation) is used to validate various phone number formats, including:

- **5-digit patterns:** Accepts patterns like 12345 or 12345-12345.
- **North American numbers:** Validates standard U.S. formats with optional country code 1.
- **International formats:** Matches numbers with international prefixes +, 00, or 011.

- **Danish numbers:** Validates formats specific to Denmark with the country code +45.

Specific Format Check

The function `is_valid_phone_number` includes additional checks for the North American formats `(XXX)XXX-XXXX` and `1(XXX)XXX-XXXX`, which are validated separately if they do not pass the main regex.

Pros

- **Comprehensive Input Validation:** The use of detailed regular expressions for both phone numbers and names reduces the chance of invalid inputs.
- **Structured Role-Based Access Control:** The application enforces access restrictions based on user roles, ensuring data integrity and security.
- **Automated Testing:** The `app_test.py` script provides comprehensive testing for multiple scenarios, allowing easy validation of API functionalities.

Cons

- **Complexity in Regex Maintenance:** The complexity of the regex patterns, particularly for phone number validation, could make maintenance difficult if new formats are introduced.
- **Limited Flexibility in Name and Phone Number Patterns:** The strict regular expressions might inadvertently reject valid entries, such as names with unconventional characters or phone numbers in rare formats.
- **Static Secret Key:** The `SECRET_KEY` is hardcoded, which may pose a security risk if not managed securely.

3. Assumptions

- Users have roles: **READ** (view only) and **READ_WRITE** (view and modify).
- Database persistence is enabled with volume mapping for phonebook.db and test.db.
- Log files audit.log is used to track API actions and is set with the appropriate permissions for writing.
- Postman is used for endpoint testing, and Docker is used to manage the environment and automate tests.

4. Project Setup Instructions

- **Cloning the Repository:**

Use ***git clone https://github.com/RMO1749/PhoneBookAPITesting-.git*** to get the project files.

- **Creating Required Files:**

Run the following commands to create necessary files:

touch audit.log phonebook.db test.db

Update docker-compose.yml paths to match the correct file locations for phonebook.db, audit.log, and test.db.

- **Setting Permissions:**

Set file permissions to ensure Docker can write to logs and databases:

chmod 777 audit.log phonebook.db test.db

- **Running Docker Compose:**

Use the following command to build and start the Docker containers:

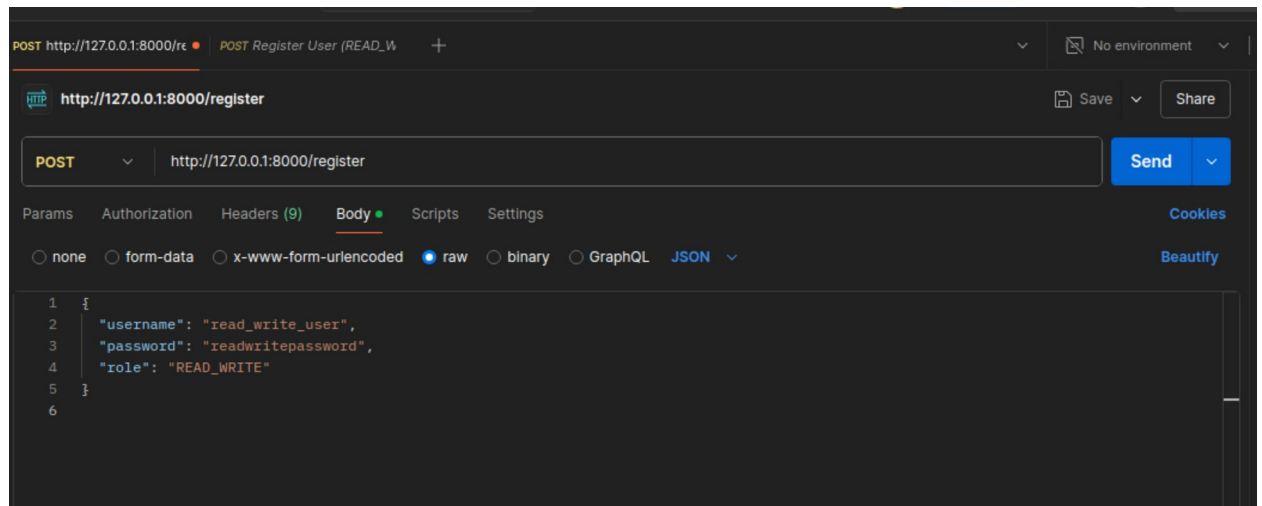
docker compose build && docker compose up

5. Detailed Feature Descriptions

- **User Registration (/register):**

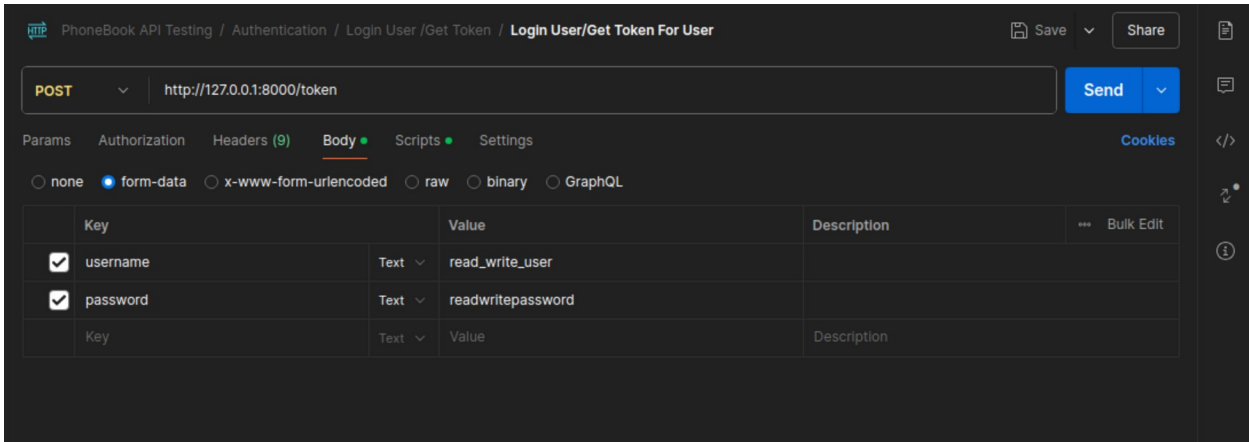
Allows a new user to register by sending a POST request. Requires a JSON body with username, password, and role (either READ or READ_WRITE).

Response: JWT token if registration is successful.



- **Login and Token Generation** (/token):

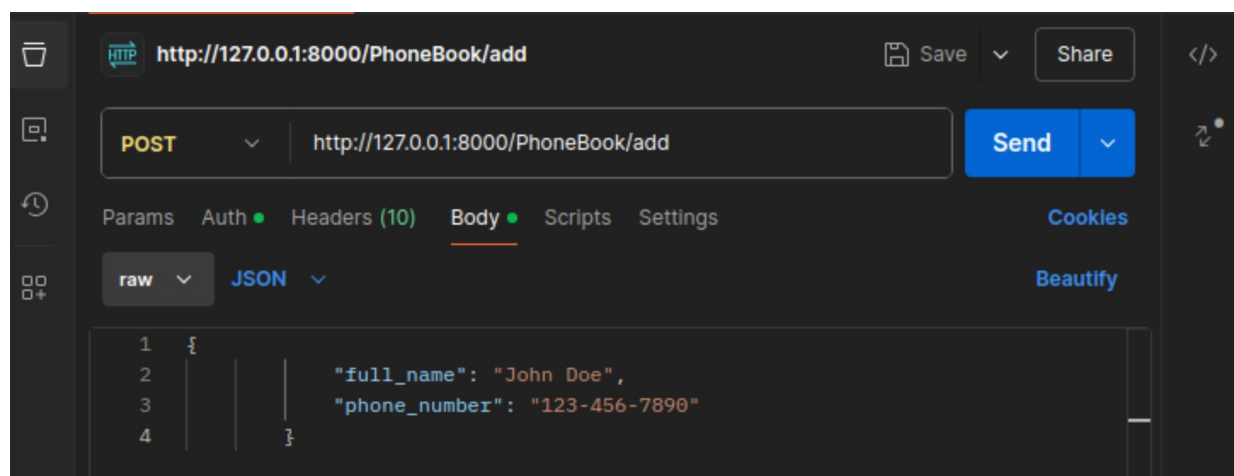
Users obtain a JWT token by posting username and password. This token is used to authenticate and authorize further actions.



Response: The access token associated with the requested username and password

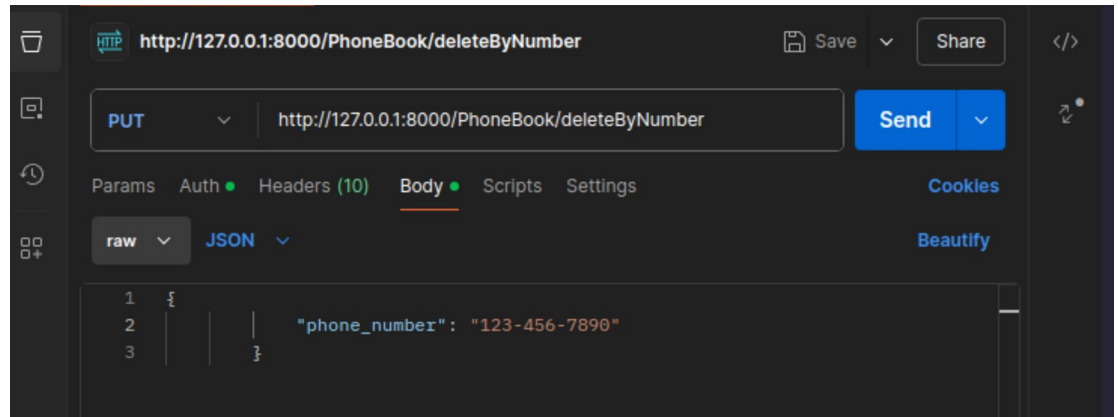
- **PhoneBook CRUD Operations:**

- o **List Entries** (/PhoneBook/list): GET request accessible with READ or READ_WRITE role, returning all entries.
- o **Add Entry** (/PhoneBook/add): POST request available to users with READ_WRITE role, adds a new entry.



- o **Delete by Name** (/PhoneBook/deleteByName): PUT request to delete entries by name, available to READ_WRITE users.

- o **Delete by Phone Number** (/PhoneBook/deleteByNumber): PUT request to delete entries by phone number, available to READ_WRITE users.



6. File Descriptions

- **app.py**: Main FastAPI application with endpoint definitions.
- **app_test.py**: Automated test cases for user registration, token authentication, and PhoneBook CRUD operations.
- **test_data.json**: JSON file containing sample test cases for valid and invalid entries.
- **docker-compose.yml**: Configuration for running the FastAPI app and test environment with volume mappings for persistent data.
- **deltatables.sql**: Command for dropping all values in an SQLite database.
- **PhoneBookApiTesting.json** - Postman collection containing several test scripts

7. Postman Testing Guide

1. **Register a New User:**
 - a. **URL:** <http://127.0.0.1:8000/register>
 - b. **Method:** POST

c. **Body (JSON):**

```
{ "username": "read_write_user",  
  "password": "readwritepassword",  
  "role": "READ_WRITE" }
```

d. **Expected Response:** Access token if registration is successful.

2. **Log In to Get a Token:**

a. **URL:** <http://127.0.0.1:8000/token>

b. **Method:** POST

c. **Body (form-data):**

i. username: read_write_user

ii. password: readwritepassword

d. **Expected Response:** Access token for authorized access.

3. **Use the Token to Access Protected Endpoints:**

a. Copy the access token and set it as the Bearer Token in Postman under Authorization for the following endpoints.

4. **Test PhoneBook Endpoints:**

a. **List PhoneBook Entries** (Requires READ role or higher):

i. URL: <http://127.0.0.1:8000/PhoneBook/list>

b. **Add a New Entry** (Requires READ_WRITE role):

i. URL: <http://127.0.0.1:8000/PhoneBook/add>

```
{  
  "full_name": "Alice Johnson",  
  "phone_number": "555-1234"  
}
```

c. **Delete Entry by Name:**

i. URL: <http://127.0.0.1:8000/PhoneBook/deleteByName>

```
{  
  "full_name": "Alice Johnson"  
}
```

d. **Delete Entry by Phone Number:**

i. URL: `http://127.0.0.1:8000/PhoneBook/deleteByNumber`

```
{  
  "phone_number": "12345"  
}
```

8. Automated Testing with Docker

- **Configure Docker Compose:**

Uncomment lines in `docker-compose.yml` to enable testing with `app_test.py` (lines are from “test service” till end of file).

- **To maintain clean test runs, clear the database by running:**

`sqlite3 test.db < deltables.sql`

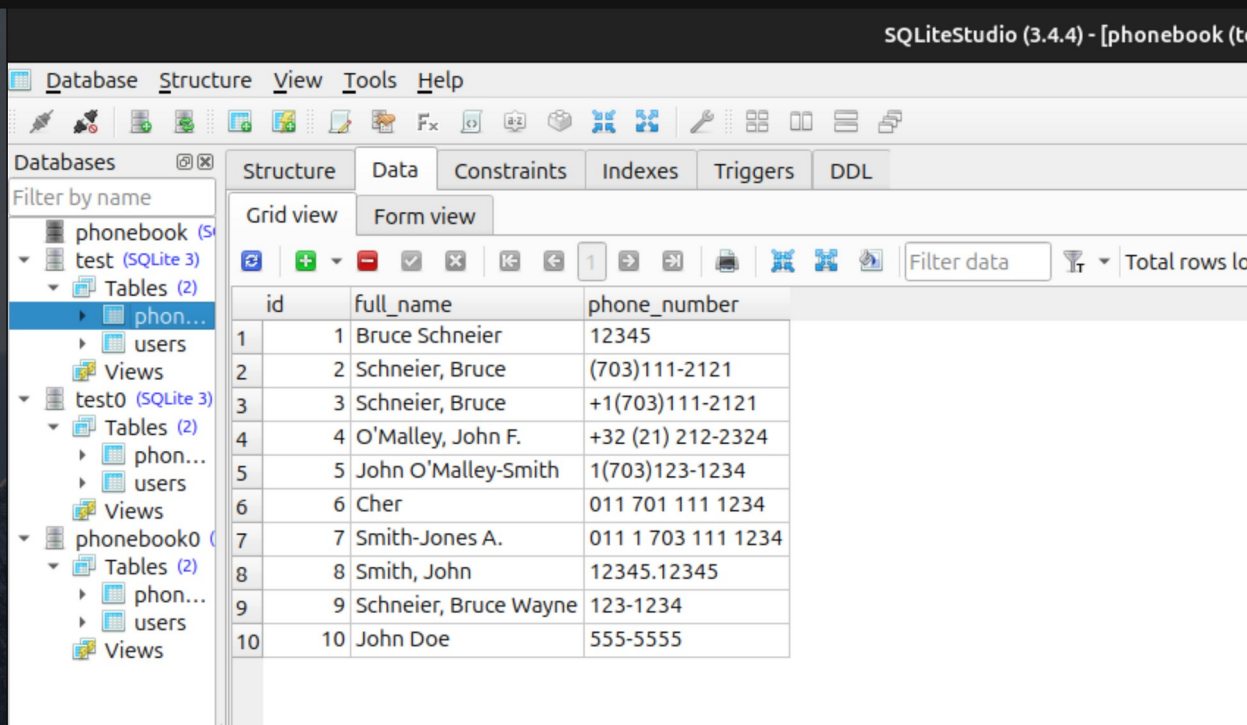
- **For fully automated testing, run:**

`docker compose up`

`test_data.json` contains the following test cases:

- **Valid Entries:** JSON entries in `PhoneBookApiTesting.json` ensure that only correct entries are added.
- **Invalid Names/Phones:** Each invalid input is tested, expecting a 400 response.
- **Duplicate Entries:** Tests for duplicate entries, expecting a specific duplicate error

Example Output Showing only Valid Entries from test_data.json



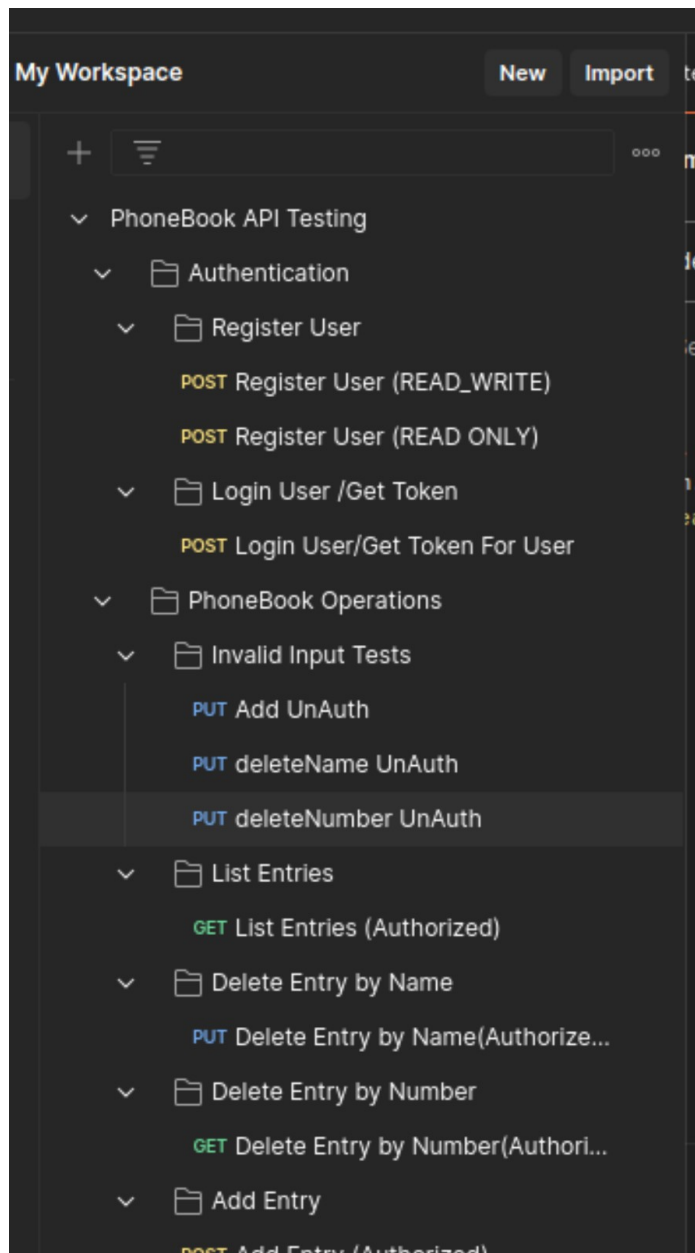
The screenshot shows the SQLiteStudio interface with the 'phonebook' database selected. The 'phon...' table is highlighted in the left sidebar. The main window displays the data in a grid view, showing 10 rows of data. The columns are 'id', 'full_name', and 'phone_number'.

id	full_name	phone_number
1	Bruce Schneier	12345
2	Schneier, Bruce	(703)111-2121
3	Schneier, Bruce	+1(703)111-2121
4	O'Malley, John F.	+32 (21) 212-2324
5	John O'Malley-Smith	1(703)123-1234
6	Cher	011 701 111 1234
7	Smith-Jones A.	011 1 703 111 1234
8	Smith, John	12345.12345
9	Schneier, Bruce Wayne	123-1234
10	John Doe	555-5555

9. Error Handling and Expected Error Messages

- **Invalid Input Errors:**
 - o Example: Missing full_name or invalid phone_number triggers a **400 error**.
- **Authorization Errors:**
 - o Users with READ role attempting to add or delete entries receive a **403 Forbidden error**.
- **Duplicate Entry Errors:**
 - o Attempting to add a duplicate entry returns a 400 error with a message indicating "Person already exists."
- **Attempting to delete a non-existent entry:**
 - o Returns a **404 error** with a message indicating the action cannot be performed.

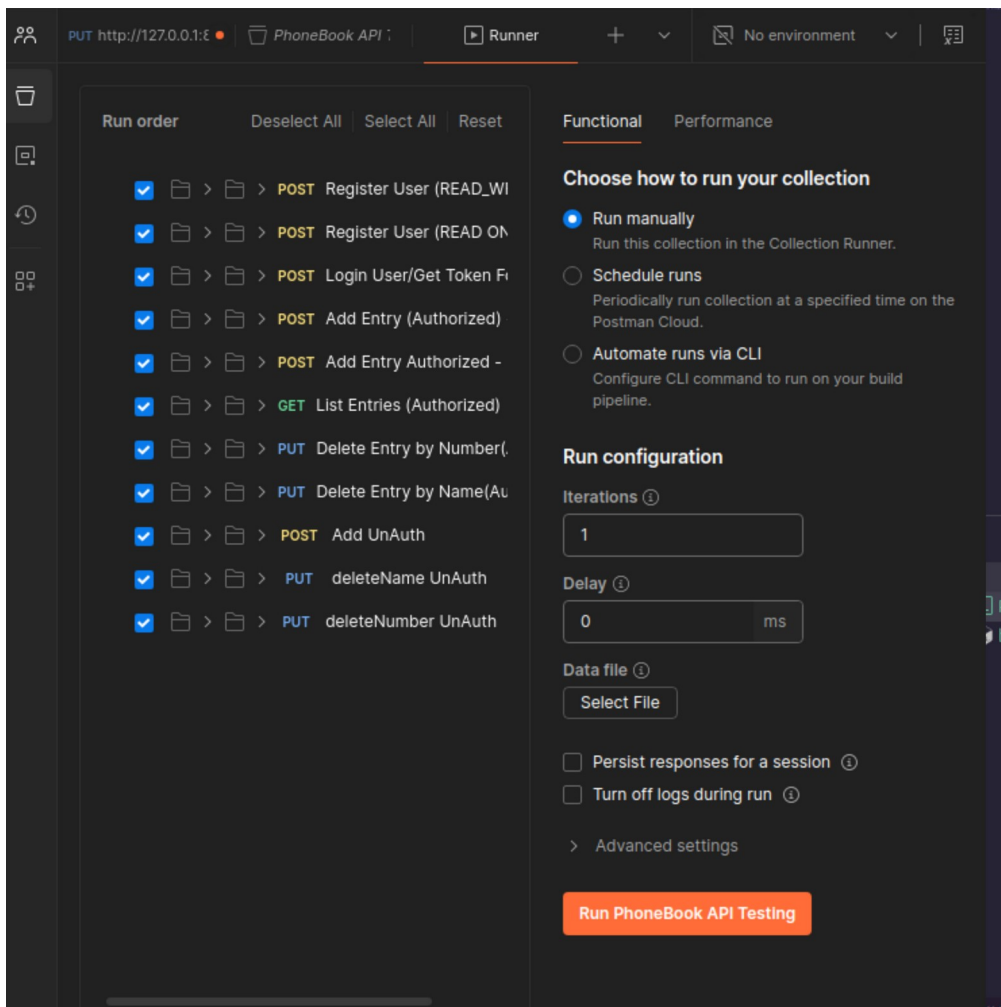
10. Test Cases and Postman Collection



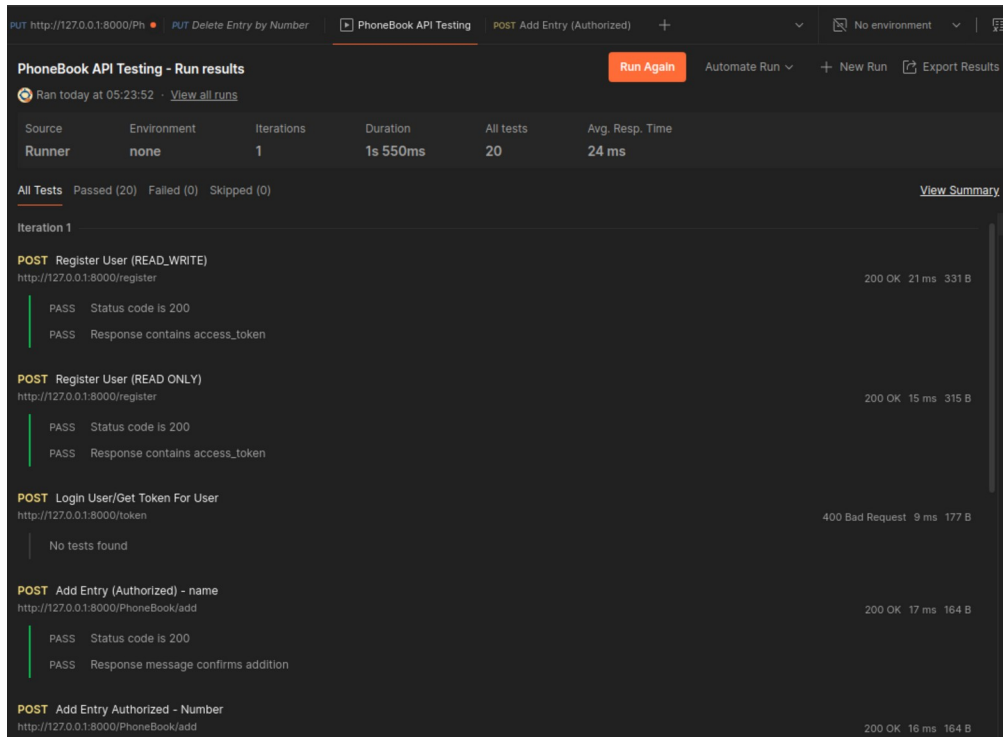
- **Postman Collection:**
 - o Run the Postman Collection to test multiple scenarios. **Reset phonebook.db before each collection run to avoid duplicates/errors.**

- o The PhoneBookApiTesting.json has been provided and it contains the structure for running automated authentication and CRUD operations.
- o The collection is divided into two main sections: **Authentication** and **PhoneBook Operations**.
 - *Authentication*: Test cases verify user registration and token retrieval for both READ and READ_WRITE roles.
 - *PhoneBook Operations*: CRUD tests for adding, listing, and deleting entries. Each test case checks for correct response codes and messages, with additional tests for error handling, such as insufficient permissions (403 Forbidden).

How to Run Postman Collection



Example Output of Postman showing all 20 test cases passing!



11. Troubleshooting and Common Issues

- **Persistent Data Management:**

If data persists between tests, either delete phonebook.db and test.db files or reset by running:

```
sqlite3 phonebook.db < deltables.sql
```

- **Log File Output:**

Ensure audit.log, phonebook.db, and test.db are correctly mapped in docker-compose.yml with the correct path (meaning change the path to reflect where data should persist) and make sure these new paths are writable.