1002161475
Varad Nair

INPUT VALIDATION: CSE CSE-5382-001 : Project Report: Secure Phone Book REST API

Overview

This project is a Secure Phone Book REST API developed using FastAPI, implementing input validation, authentication, and audit logging features. The API manages a simple phone book with names and phone numbers, with operations to add, delete, and list contacts, all while ensuring data security and input integrity.

API Endpoints

The project includes the following RESTful API endpoints:

1. **GET /PhoneBook/list**
    - **Functionality**: Lists all the contacts stored in the phone book.
    - **Response**: Returns a JSON array of phone book entries.
    - **Access Control**: Accessible by users with "Read" or "Read/Write" roles.
2. **POST /PhoneBook/add**
    - **Functionality**: Adds a new contact to the phone book.
    - **Request Body**: Expects a JSON object with full_name and phone_number.
    - **Input Validation**: Validates the name and phone number using regular expressions to ensure data integrity.
    - **Audit Log**: Records the event with a timestamp in the audit log.
    - **Response**: Returns a success message if the contact is added successfully.
    - **Access Control**: Accessible only by users with the "Read/Write" role.
3. **PUT /PhoneBook/deleteByName**
    - **Functionality**: Deletes a contact from the phone book using the full name.
    - **Query Parameter**: The full name of the contact to be deleted.
    - **Input Validation**: Ensures the provided name is valid.
    - **Audit Log**: Records the deletion event in the audit log.

- **Response**: Returns a success message or a 404 error if the contact is not found.
- **Access Control**: Accessible only by users with the "Read/Write" role.

4. **PUT /PhoneBook/deleteByNumber**
    - **Functionality**: Deletes a contact using the phone number.
    - **Query Parameter**: The phone number of the contact to be deleted.
    - **Input Validation**: Ensures the provided phone number is valid.
    - **Audit Log**: Records the deletion event in the audit log.
    - **Response**: Returns a success message or a 404 error if the contact is not found.
    - **Access Control**: Accessible only by users with the "Read/Write" role.

Input Validation

Input validation is crucial to ensure data integrity and security. The project uses regular expressions (regex) to validate names and phone numbers.

Name Validation (name_check Function)

The name_check function uses regex patterns to ensure that names are free from malicious characters and follow acceptable formats.

- **Regex Pattern**:
    - "^[a-zA-Z(')?\-]+(,)?(\s)?([a-zA-Z\-]+(')?[a-zA-Z\-]+)?(\s)?([a-zA-Z'\-]+)?(.)?$"
- **Description**:
    - This pattern allows names with alphabets, optional punctuation (such as hyphens and apostrophes), commas for surname-first formats, and handles various spacing rules.
    - Examples of valid names: "Bruce Schneier", "O'Malley, John F.", "Schneier, Bruce Wayne".
    - The function also explicitly rejects names with consecutive apostrophes or multiple hyphens.

```python
def name_check(name):
    if not re.match("^[a-zA-Z(')?\-]+(,)?(\s)?([a-zA-Z\-]+(')?[a-zA-Z\-]+)?(\s)?([a-zA-Z'\-]+)?(.)?$", name):
        return False
    if  re.match(".*''.*", name):
        return False
    if  re.match(".*\-.*\-.*", name):
        return False

    return True
```

Phone Number Validation (phone_check Function)

The phone_check function uses various regex patterns to allow multiple phone number formats, including international and local numbers.

- **Accepted Formats**:
    - ##### (5-digit numbers)
    - ###-###-#### (US format)
    - (###) ###-#### (Parentheses format)
    - #####.##### (International extensions with a dot)
    - ### ### #### (Space-separated)
    - +## (##) ###-#### (International format with country code)
- **Rejected Formats**: Patterns with slashes, non-numeric characters (e.g., "Nr"), and consecutive extensions are flagged as invalid.

```python
def phone_check(phone_number):
    ''' check number is in this format: #####'''
    if  re.match("^\d{5}$", phone_number):
        return True

    ''' check number is in this format: ###-###-####'''
    if  re.match("^\d{3}-\d{3}-\d{4}$", phone_number):
        return True

    ''' check number is in this format: (###) ###-####'''
    if  re.match("^\(\d{3}\)\s\d{3}-\d{4}$", phone_number):
        return True

    ''' check number is in this format: (###)###-####'''
    if  re.match("^\(\d{3}\)\d{3}-\d{4}$", phone_number):
        return True

    ''' check number is in this format: ### ### ####'''
```

```
    if  re.match("^\d{3}\s\d{3}\s\d{4}$", phone_number):
        return True

    '''check number if is in this format:#####.#####'''
    if  re.match("^\d{5}\.\d{5}$", phone_number):
        return True

    '''check number if is this format: ### # ### ### ####'''
    if  re.match("^\d{3}\s\d{1}\s\d{3}\s\d{3}\s\d{4}$", phone_number):
        return True

    '''check number if is this format: # (###) ###-####'''
    if  re.match("^\d{1}\s\(\d{3}\)\s\d{3}-\d{4}$", phone_number):
        return True

    '''check number if is this format: +32 (##) ###-####'''
    if  re.match("^\+\d{2}\s\(\d{2}\)\s\d{3}-\d{4}$", phone_number):
        return True

    '''check number if is this format: ### ### ### ####'''
    if  re.match("^\d{3}\s\d{3}\s\d{3}\s\d{4}$", phone_number):
        return True

    ''''check number if is this format: ###-####'''
    if  re.match("^\d{3}-\d{4}$", phone_number):
        return True

    '''check number if is in this format: # (###) ###-####'''
    if  re.match("^\d{1}\s\(\d{3}\)\s\d{3}-\d{4}$", phone_number):
        return True

    '''check number if is in this format: +# (###) ###-####'''
    if  re.match("^\+\d{1}\s\(\d{3}\)\s\d{3}-\d{4}$", phone_number):
        return True

    '''check number if is in this format: ###'''
    if  re.match("^\d{3}$", phone_number):
        return False

    '''check number if is in this format: #/###/###/####'''
    if  re.match("^\d{1}\/\d{3}\/\d{3}\/\d{4}$", phone_number):
        return False

    '''check number if is in this format: (###) ###-#### ext ###'''
    if  re.match("^\(\d{3}\)\s\d{3}-\d{4}\sext\s\d{3}$", phone_number):
```

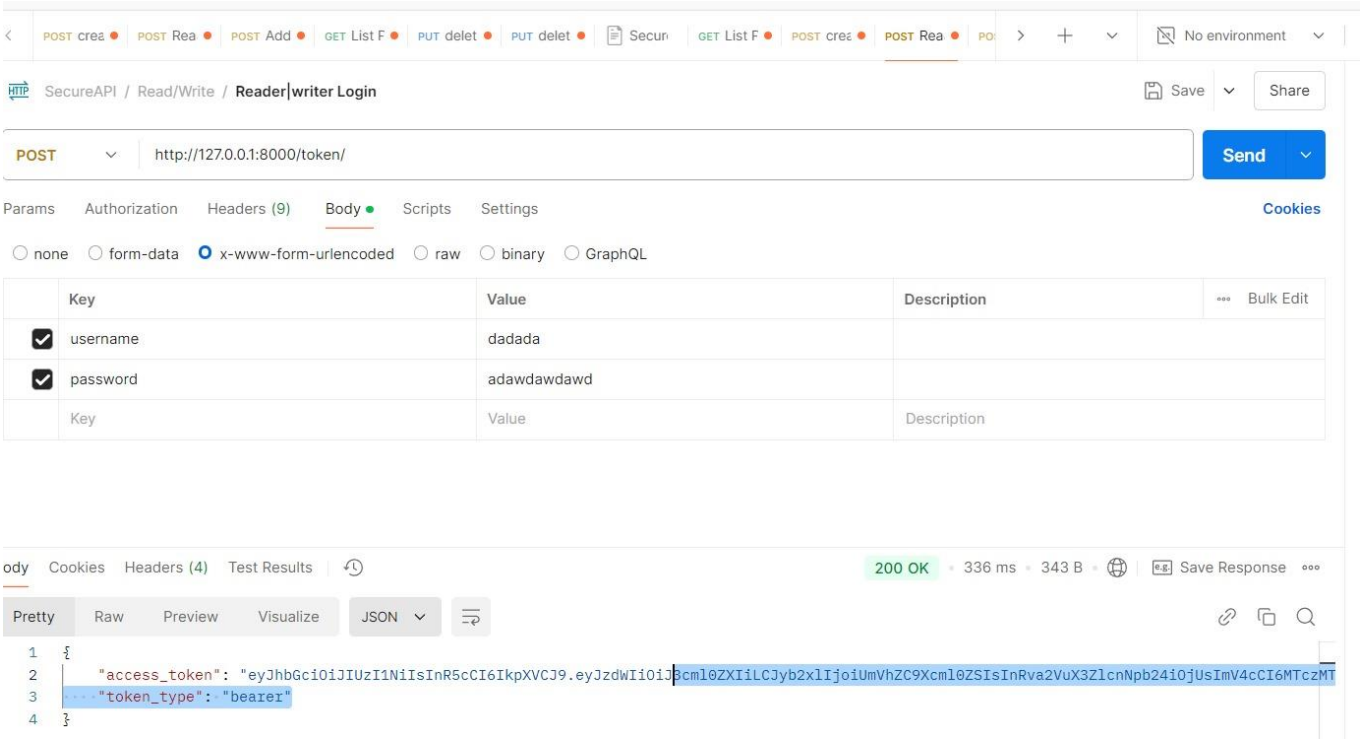
```
        return False

    '''check number if is in this format: +## (###) ###-####'''
    if  re.match("^\+\d{2}\s\(\d{3}\)\s\d{3}-\d{4}$", phone_number):
        return False

    '''check number if is in this format: Nr ###-###-####'''
    if  re.match("^Nr\s\d{3}-\d{3}-\d{4}$", phone_number):
        return False

    '''check number if is in this format: ##########'''
    if  re.match("^\d{10}$", phone_number):
        return False

    '''check number if is in this format: +#### (###) ###-####'''
    if re.match("^\+\d{4}\s\(\d{3}\)\s\d{3}-\d{4}$", phone_number):
        return False

    '''check number if is in this format: (###) ###-####'''
    if re.match("^\(\d{3}\)\s\d{3}-\d{4}$", phone_number):
        return False

    '''check number if is in this format: <###> ###-####'''
    if re.match("^\<\d{3}\>\s\d{3}-\d{4}$", phone_number):
        return False

    if re.match("/(\+\d{1,3}\s?)?((\(\d{3}\)\s?)|(\d{3})(\s|-?))(\d{3}(\s|-?))(\d{4})(\s?(([E|e]xt[:|.|]?)|x|X)(\s?\d+))?/g", phone_number):
        return True
```

Authentication and Authorization

The API uses Bearer Token authentication to ensure secure access. The Bearer Token mechanism provides two levels of authorization:

1. **Read**: Users with this role can only access the /PhoneBook/list endpoint.
2. **Read/Write**: Users with this role can access all endpoints, including add and delete operations.

By using tokens, the API ensures that only authenticated and authorized users can make changes to the phone book. This implementation is crucial for protecting sensitive data and preventing unauthorized modifications.



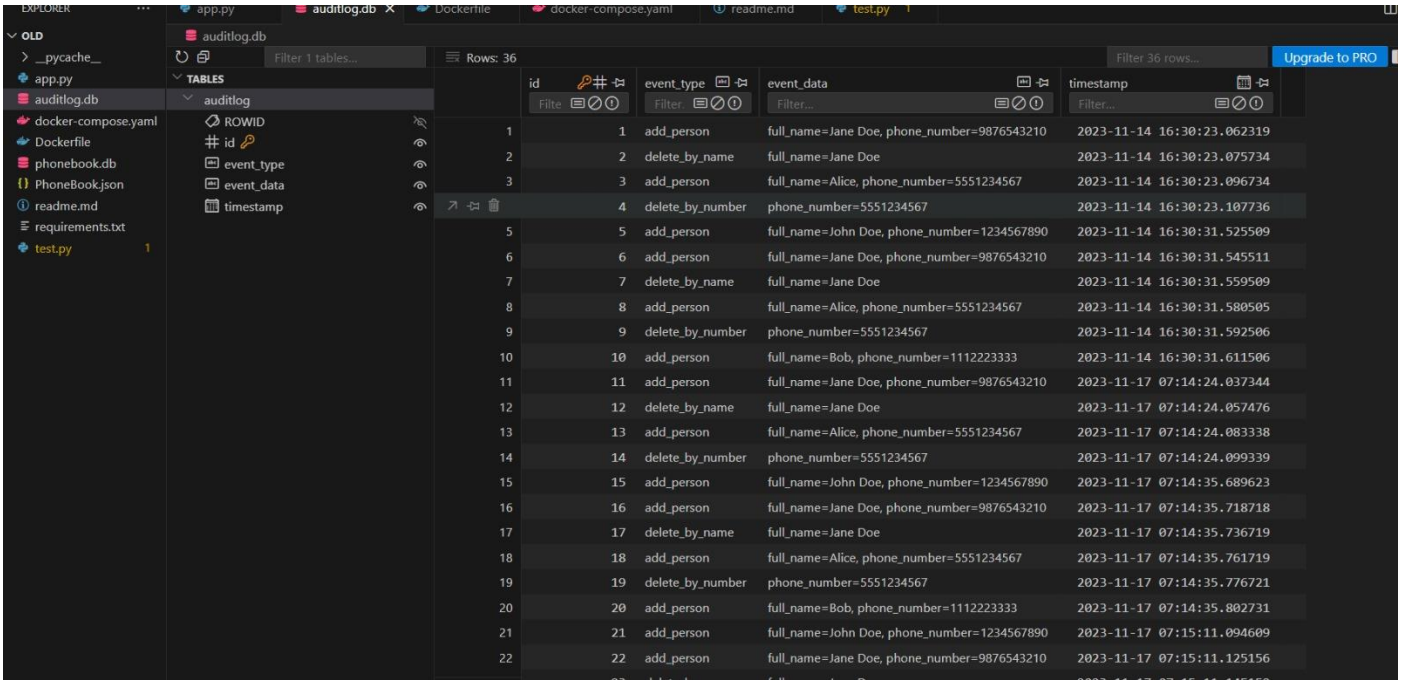
Audit Logging

The project includes an audit logging system to track all significant actions. Each log entry contains:

- **Event Type**: Describes the action performed (e.g., "add_person", "delete_by_name").
- **Event Data**: Details about the action, such as the name or phone number involved.
- **Timestamp**: The exact time when the event occurred.

The write_to_audit_log function handles logging by recording entries in a separate SQLite database (auditlog.db). This system is essential for monitoring and debugging and can help trace actions in case of a security breach.

1002161475
Varad Nair



Database and Persistence

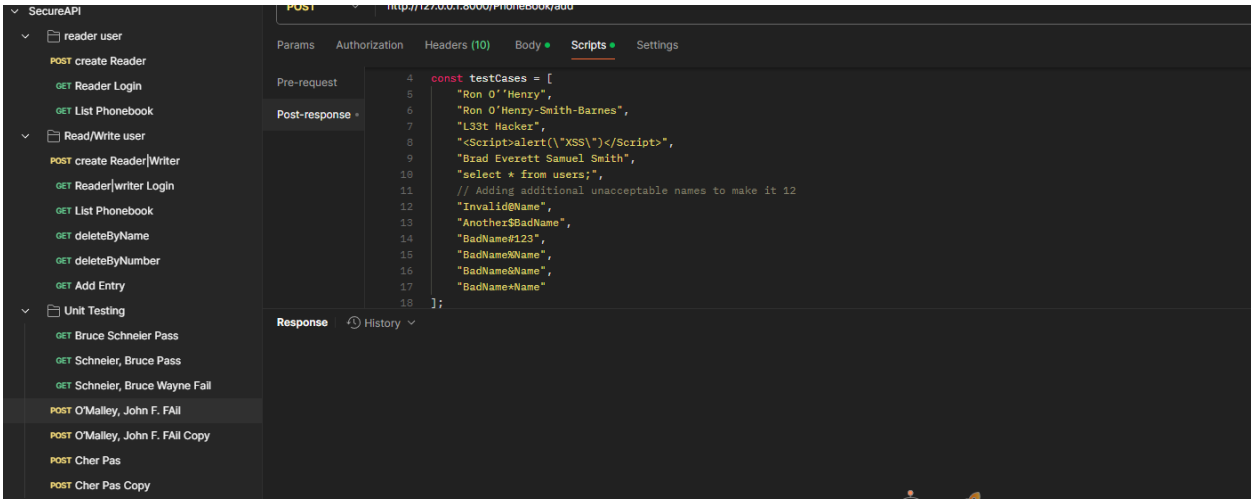
- **SQLite**: The project uses SQLite for data persistence, with separate databases for the phone book and audit logs.
- **Models**: SQLAlchemy ORM models define the structure of the phone book and audit log tables.
- **Session Management**: SQLAlchemy sessions are used to interact with the databases, ensuring efficient data handling and transaction management.

Testing

The project includes a test.py file for automated testing using pytest. Go to Postman for Unit testing. The tests cover:

- Adding valid and invalid contacts.

- Deleting contacts by name and number.
- Listing all contacts.
- Ensuring proper handling of edge cases and invalid input.



Conclusion

This Secure Phone Book REST API demonstrates robust input validation, efficient data management using SQLite, and secure authentication using Bearer Tokens. The implementation balances functionality with security, making it a practical example of a real-world application.