**Project: Secure Phone Book Rest API**



The goal of this assignment is to produce a REST API that validates its input using regular expressions and implements various security features as discussed throughout the semester.

This will be an individual assignment (no teams).

## Detail:

Produce a REST API application that maintains a phone book of names and phone numbers. The program shall be capable of receiving and storing a list of people with their full name and telephone number. The application shall include the following API endpoints:

- GET /PhoneBook/list – Produce a list of the members of the database.
- POST /PhoneBook/add – Add a new person to the database.
    - Argument is an object with name and phone number string elements (represented in JSON).
- PUT /PhoneBook/deleteByName – Remove someone from the database by name.
    - Argument is the name as a string.
- PUT /PhoneBook/deleteByNumber – Remove someone by telephone #.
    - Argument is the phone number as a string.

See the PhoneBook.json file attached to the assignment in Canvas for a full OpenAPI 3.0 spec to be used as requirements for the interface. Return values should all be in JSON per the spec.

Create regular expressions for <Person> and <Telephone #>. Use these regular expressions to verify that the user is supplying valid data. More flexible specifications will be graded higher. For example:

- Allowing for international or US format telephone numbers
- Allowing for <first middle last>, <first last> or <last, first MI>)

Reject any attempts to provide invalid data. When valid input is provided, the server shall return a status code of 200; when invalid input is provided, the server shall return a status code of 400 and an appropriate error message included in the response. An attempt to remove a non-existent name from the directory shall return a status code of 404.

You have the freedom to choose the implementation that you are comfortable with for persisting the phonebook to disk (e.g. XML, text file, binary file, CSV, database, etc.). If you are attempting the bonus at the bottom of the assignment, you might choose to use a SQL database initially to avoid rework later (our recommendation would be to use SQLite as it is portable and doesn't require excessive setup steps).

To provide an opportunity to demonstrate other skills learned this semester, the following are additional requirements that must be met:

- Create an audit log to contain timestamped log entries written anytime a user adds a name, removes a name, or lists the names. For add/remove entries, include the name of the person that was added/removed in the log.
- Add authentication and authorization capabilities to require callers to provide a credential to invoke the API endpoints. Define two roles (for example Read and Read/Write) that define two different levels of privilege. The lower privilege level (Read) only allows calls to the list endpoint while Read/Write allows calls to all endpoints (list, add, remove). The authentication capability could use a variety of schemes such as username/password, API keys, bearer token, etc. Be sure to investigate frameworks that provide these capabilities and prefer to use one instead of writing all these capabilities from scratch.

If you do use a client/server database product, you should create a config file to contain database connectivity information and have your script read that rather than hardcoding those values in the application.

Permissible languages: C/C++, Any .Net Language, Java, Perl, Python, Go, others with permission.

You may use one of the starter projects written in C#, Java, Python, or Go if you like (attached to assignment in Canvas). These implement all the basic PhoneBook functions as a REST API but do not perform any input validation or audit logging and stores the entries in an in-memory dictionary (not persisted).

You MUST use Docker container images to build and run your solution and must include the Dockerfile(s) necessary to produce the images. This is important for easing the burden of testing by the GTA when grading your assignment. These Docker containers should include all necessary dependencies for whatever technology stack you are using for your implementation.

You must also add automated security-focused unit tests using a testing framework to test at least all the inputs provided in this assignment and verify proper handling of them along with some of your own inputs. This could also be accomplished using an API testing tool such as Postman (be sure to provide the collection for Postman if you use this method).

## Requirements for Phone Numbers:

The following are some rules to follow for phone numbers:

- The country code may or may not be preceded by a + which indicates that an international dialing prefix, such as 00 or 011, must be included when dialing. If not using the plus, the dialing prefix itself may be included.
- Some organizations use 5-digit extensions only for dialing from one internal phone to another.
- North American phone numbers dialed within the countries of North America use a country code of 1, have a 3-digit area code, and a 7-digit subscriber number. Calls to local numbers in the same area code may omit the area code if not in a metro area; therefore, a subscriber only format may be used. Acceptable entry for North American phone numbers are as follows:
    - <Subscriber Number> (e.g. 123-4567)

    - (<Area Code>)<Subscriber Number> (e.g. (670)123-4567)
    - <Area Code>-<Subscriber Number> (e.g. 670-123-4567)
    - 1-<Area Code>-<Subscriber Number> (e.g. 1-670-123-4567)
    - 1(<Area Code>)<Subscriber Number> (e.g. 1(670)123-4567)
    - <Area Code> <Subscriber Number> (e.g. 670 123 4567)
    - <Area Code>.<Subscriber Number> (e.g. 670.123.4567)
    - 1 <Area Code> <Subscriber Number> (e.g. 1 670 123 4567)
    - 1.<Area Code>.<Subscriber Number> (e.g. 1.670.123.4567)
- Danish telephone numbers are 8 digits long, and normally written in four groups of two separated by spaces, AA AA AA AA. In recent years it has also become common to write them in two groups of four, AAAA AAAA. Also, allow dots instead of spaces. Denmark's country code is 45 and may be included as well for international formats.
- Some notations use 2-digit area codes.
- Some notations with 10 digits in two groups of five separated by either a space or a dot.

## Sample Inputs:

*Acceptable inputs for name:*

- `Bruce Schneier`
- `Schneier, Bruce`
- `Schneier, Bruce Wayne`
- `O'Malley, John F.`
- `John O'Malley-Smith`
- `Cher`

*Unacceptable inputs for name:*

- `Ron O''Henry`
- `Ron O'Henry-Smith-Barnes`
- `L33t Hacker`
- `<Script>alert("XSS")</Script>`
- `Brad Everett Samuel Smith`
- `select * from users;`

*Acceptable inputs for phone: *remember these are also international numbers*

- `12345`
- `(703)111-2121`
- `123-1234`
- `+1(703)111-2121`
- `+32 (21) 212-2324`
- `1(703)123-1234`
- `011 701 111 1234`
- `12345.12345`
- `011 1 703 111 1234`

*Unacceptable inputs for phone:*

- `123`
- `1/703/123/1234`
- `Nr 102-123-1234`
- `<script>alert("XSS")</script>`
- `7031111234`
- `+1234 (201) 123-1234`
- `(001) 123-1234`
- `+01 (703) 123-1234`
- `(703) 123-1234 ext 204`

The TA will utilize a script to test your program using all the above acceptable and unacceptable inputs for name and phone number, as well as additional ones not listed here.

## Submission instructions:

You are required to provide the source code including concise instructions on how to build and run the software and the unit tests. All build time and run time dependencies should be provided in the form of Docker containers so that the GTA doesn't have to install anything other than Docker. In addition to the code and instructions, you must also turn in a report that describes your submission. This report should include a description of how your code works, the design of your regular expressions, any assumptions you have made, and the pros/cons of your approach.

## Bonus (10 points):

Assuming a database backend wasn't used, change to use a SQL database engine, such as SQLite, to persist the phonebook to disk. For reading and writing to the database, use an API that supports parameterized queries (also known as prepared statements) in SELECT and INSERT statements which will avoid SQL insertion vulnerabilities.

## Grading Criteria:

- The source code should be provided along with submission (**0 point out of 100 if there is no code attached**)
- Report **<<30 points>>**:
  - Instructions for building and running software and unit tests (5 points)
  - Description of how your code works (10 points)
  - Assumptions you have made (10 points)
  - Pros/Cons of your approach (5 points)
- Program is running successfully and implements all required functionality **<<60 point>>**:
  - ADD operation (10 points)
  - DEL by name operation (10 points)
  - DEL by number operation (10 points)
  - LIST operation (10 points)
  - Audit log functionality (10 points)
  - Authentication/Authorization (10 points)

        - Authentication mechanism (4 points)
        - Read Role on list operation (3 points)
        - Read/Write Role on all operations (3 points)
- Security Tests **<<10 points>>:**
    - Using automated unit tests to verify stated good inputs (4 points)
    - Using automated unit tests to verify stated bad inputs (4 points)
    - Using automated unit tests to verify student provided inputs (2 points)
- Bonus **<<10 points>>**:
    - Using database to store the input data (5 points)
    - Using an API that supports parameterized queries (prepared statements) (5 points)