Input Validation

The goal of this assignment is to produce a REST API that validates its input using regular expressions.

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. 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.
- 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.

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.

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 (also, the config file should not be readable by users running the program).

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

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 of 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 instructions on how to compile, install, and run the software. Please indicate if your code has any dependencies (e.g. libraries, external programs, etc.) that must be installed prior to running your code. I would strongly prefer that you do not rely upon anything that is not freely available. If this is not possible, please let me or the TA know in advance so we can work out some way for us to evaluate and grade your assignment. 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 1 (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.

Bonus 2 (15 points):

Add automated unit tests using a testing framework to test at least all of the inputs provided above and verify proper handling of them along with some of your own inputs.

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>>:**
  - Description of how your code works (10 points)
  - Compilation/build instructions (5 points)
  - Installation, setup, and execution instructions (5 points)
  - Assumptions you have made (5 points)
  - Pros/Cons of your approach (5 points)
- Program is running successfully **<<70 point>>**:
  - ADD operation (15 points)
  - DEL by name operation (15 points)
  - DEL by number operation (15 points)
  - LIST operation (15 points)
  - Audit log functionality (10 points)
- Bonus 1 **<<10 points>>**:
  - Using database to store the input data (5 points)
  - Using an API that supports parameterized queries (prepared statements) (5 points)
- Bonus 2 **<<15 points>>**:
  - Using automated unit tests to verify stated good inputs (5 points)
  - Using automated unit tests to verify stated bad inputs (5 points)
  - Using automated unit tests to verify student provided inputs (5 points)