INPUT VALIDATION REPORT

NAME : INA KACHHAL
STUDENT ID: 1002060157

Instruction for building and running software

Docker

1. Download the zip file named ‘project_sp_1002060157.zip’.
2. Unzip the folder.
3. The folder contains the below file.
    a. Dockerfile
    b. inaAssignment.db
    c. Logs
    d. Phone (contains sql queries to create table)
    e. Src (contains all the code)
    f. Pom.xml
    g. Springbootstarter.pdf
4. Change the directory to the unzipped folder.
5. Build using the below command to generate the latest docker image.
   docker build . --tag=assignment:latest
6. Run using the below command.
   docker run -p8080:8080 assignment:latest

Intellij + Spring boot setup.

Prerequisite:

1. Install SQLite3.
2. Note down sqlite3 path.
3. Create a database using command sqlite3 inaAssignment.db.
4. Execute the create table command given in file SqlQueries in folder phone.

Installation Steps:

1. Download the zip file named ‘project_sp_1002060157.zip’.
2. Unzip the folder.
3. The folder contains the below file.

h. Dockerfile
i. inaAssignment.db
j. Logs
k. Phone (contains sql queries to create table)
l. Src (contains all the code)
m. Pom.xml
n. Springbootstarter.pdf

4. Open project in intellij.
5. Comment spring.autoconfigure.exclude property in application.properties.
6. Edit spring.datasource.url=jdbc:sqlite:/<local path to sqlite>
7. Build project using command: mvn clean install. This will execute all the test cases.

Instruction to run Test cases

Run below build command it will build all the test cases. Build will fail if any of the test cases fails.

Description of how your code works

The main source code folder are as follows:

1. src/main/java/com/cse5382/assignment
   This folder contains the subfolder as follows and the main function to kick start the project.
   a. Controller
      - The Controller serves as the entry point for handling HTTP requests related to the phonebook functionality.
      - It defines and manages all endpoints responsible for storing and retrieving data in the phonebook.
   b. Dialect
      - The Dialect specifies the SQLite variant used as an alternative SQL dialect to the OGR SQL dialect.
      - It influences the way data is stored, retrieved, and manipulated within the database.
   c. Model
      - The Model encapsulates the data structures used in the application.

        - PhonebookEntry is designed to receive incoming requests, while PhonebookResponse structures the data for outgoing responses to users.
    d. Repository
        - The Repository is responsible for database interactions, including storing, retrieving, and deleting records.
        - It acts as a bridge between the application and the underlying database, ensuring data integrity and persistence.
    e. Service
        - The Service layer contains the business logic for handling requests.
        - It receives incoming requests, validates them, coordinates the storage of data in the database through the repository, and constructs appropriate responses to be sent back to clients.
        - This layer plays a crucial role in maintaining separation of concerns and keeping the controller lightweight.
2. test/java/com/se5382/assignment
    - The test/java/com/se5382/assignment directory houses comprehensive unit test cases.
    - These test cases cover a wide range of scenarios, including valid and invalid names and numbers from user input, ensuring robust and reliable functionality.
3. logs
    - The logs directory contains an app.log file dedicated to auditing and monitoring requests entering the system.
    - This log file serves as a valuable resource for tracking system activities, diagnosing issues, and ensuring compliance with logging standards.
4. phone
    - The phone directory encompasses a SqlQueries file, specifically designed for managing database operations.
    - This file includes SQL queries responsible for creating tables and inserting initial data into the database.

Endpoint curl command

1. **Get Api:** To list all the elements

curl --location --request GET 'localhost:8080/phoneBook/list'

2. **Post Api:** To persist elements in db.

```
curl --location --request POST 'localhost:8080/phoneBook/add' \
--header 'Content-Type: application/json' \
--data-raw '{
    "name": "Bruce Schneier",
    "phoneNumber": "011 1 703 111 1234"
}'
```

3. **Put Api:** There are 2 api's one to delete record by name and another to delete record by number.

```
curl --location --request PUT
'localhost:8080/phoneBook/deleteByNumber?number=011%201%20703%
20111%201234'
curl --location --request PUT
'localhost:8080/phoneBook/deleteByName?name=Bruce%20Schneier'
```

The design of regular expressions:

I have used two regex expressions. One for name matching and other for phoneNumber matching.

1. Name

regExptNameCode = "^[A-Z][a-zA-Z']+([-][A-Z][a-zA-Z']+)?(?:, [A-Z][a-zA-Z']+((?: [A-Z]\\.)?)?)?$"

Above pattern will see whether the entered query name matches with the above regex pattern or not. If it matches with the above pattern, then it will return True Boolean value. If the return Boolean value is True, then that user entered the query name is unacceptable. After this post/put api will return an error message like 'Invalid name format has been entered' with status code 400. If the entered user query parameter value for name doesn't match with above regex pattern then it will return False as a Boolean value and that name will be considered as an accepted name value.

2. Number

regExptNumberCode = "^(?:(?:(\\+(00|011)?[1-9]\\d*[-.]?)|(0\\d{1,2}[-.]?))?(1[-.]?)?(\\(\\d{1,4}\\)|\\d{1,4})?[-.]?(\\d{3}[-.]?\\d{4}|\\d{5}|\\d{2}[-.]?\\d{8}|\\d{1,5}\\.\\d{1,5}))?(\\s?(ext|x|ext\\.)\\s?\\d{1,6})?$"

Above pattern will see whether the entered query phoneNumber matches with the above regex pattern or not. If it matches with the above pattern, then it will return True Boolean value. If the return Boolean value is True, then that user entered query phoneNumber is unacceptable. After this post/put api will return an error message like 'Invalid name format has been entered' with status code 400. If the entered user query parameter value for phoneNumber doesn't match with above regex then it will return False as a Boolean value and that phoneNumber will be considered as an accepted phoneNumber value.

Assumptions I have made

I have assumed and code according to the pattern given in the list of acceptable names and numbers while developing the regular expression. My regular expression will block the entries given in the list of unacceptable names and numbers or block the entries in similar scenarios/patterns. But I don't know the scenarios/patterns other than the patterns provided, that I should block. There might be a number of patterns other than the patterns provided for unacceptable names and phone numbers that I should block, but I have considered the patterns related to one given in the assignment-requirements pdf. However there are few numbers and names which are not handled in the regex. Names: Schneier, Bruce Wayne, O'Malley, John F., John O'Malley-Smith. Numbers: 7031111234, +1234 (201) 123-1234, (001) 123-1234, (703) 123-1234 ext 204

Pros/Cons of the approach used:

Pros:

1. This approach of building regular expression will allow us to block unwanted patterns to be passed in the query parameters for post or put api. For example, entries with the numbers (like L33t Hacker) for the 'name' query parameter will be blocked and api will return 400 status code. Similarly, entries with the alphabets (like Nr 102-123-1234) for the

‘phoneNumber’ query parameter will be blocked and api will return 400 status code.

2. This approach will block any sql query provided as a name query parameter. For example, if user provides name = “select * from users;” it will block users from accessing the database by returning 400 Bad Request responses. This can prevent Hackers from Hacking the database and accessing the crucial data from the database.
3. In short this approach is blocking hackers from doing sql injection attacks by blocking sql queries sending as a query parameters in post and put api.

Cons:

In this approach I have mentioned *. In the end of the regular expression string related to phone number. This will allow an attacker to enter a number of infinite characters in the number query parameter while calling post/put api resulting in buffer overflow attack.