CSE-5382-SECURE PROGRAMMING

Assignment 11: Input Validation

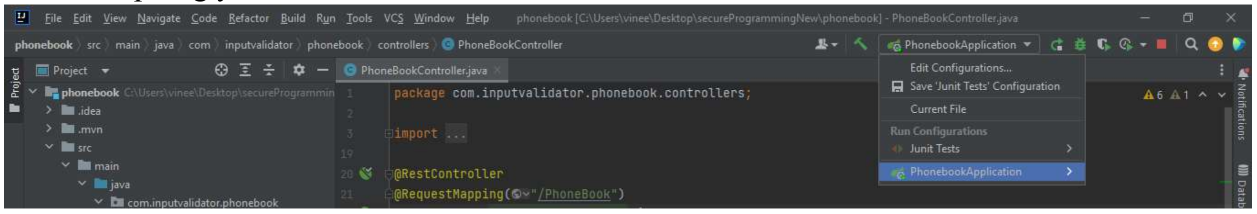
Name: Vineeth Kumar Ananthula
UTA ID: 1001953922

Tech Stack used to develop REST API:

- Java Programming Language
- Spring Boot and Maven Framework
- SQL lite database
- SQL to write parameterized queries
- Swagger UI to visualize and interact with the API
- IntelliJ IDEA Integrated development environment (IDE)
- Implemented Junit Tests

Steps to run the application:

1. Start IntelliJ IDE. Go to File -> Open.
2. Select Run Phonebook application in the popup and click the run ▶ in the gutter. The IDE starts compiling your code



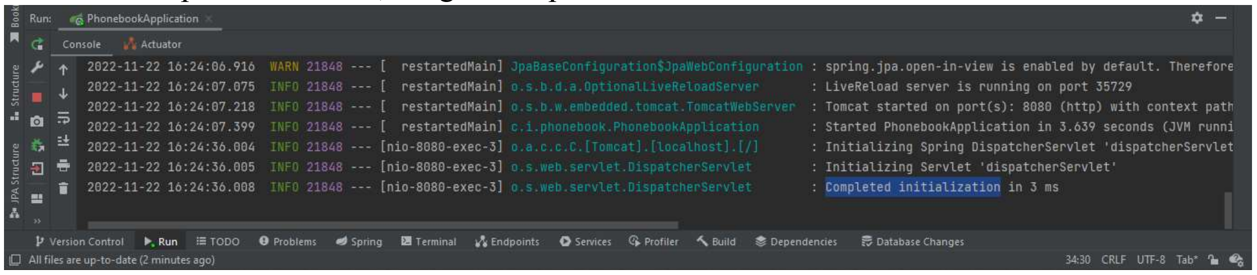
```
package com.inputvalidator.phonebook.controllers;

import ...

@RestController
@RequestMapping("/PhoneBook")
```

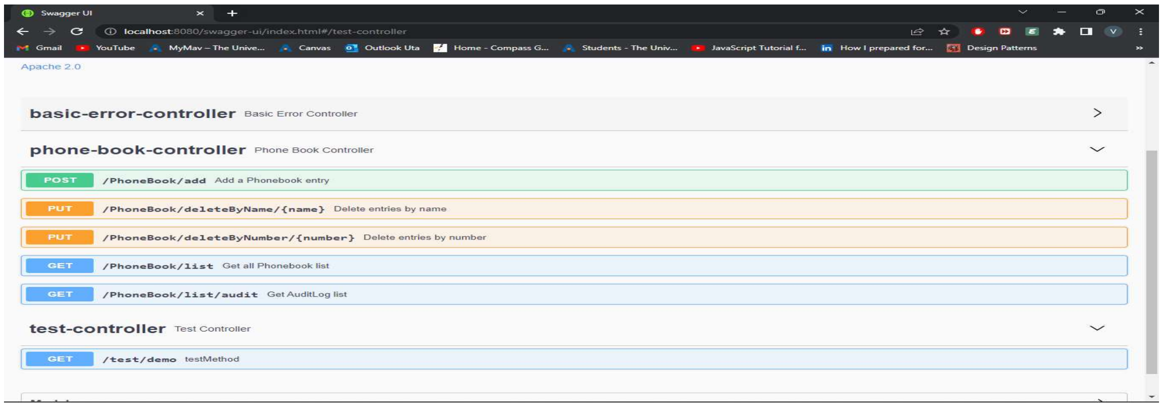
File Edit View Navigate Code Refactor Build Run Tools VCS Window Help
phonebook [C:\Users\vinee\Desktop\secureProgrammingNew\phonebook] - PhoneBookController.java
phonebook src main java com inputvalidator phonebook controllers PhoneBookController
PhonebookApplication
Project
PhoneBookController.java
phonebook C:\Users\vinee\Desktop\secureProgrammin
.idea
.mvn
src
main
java
com.inputvalidator.phonebook
Edit Configurations...
Save 'Junit Tests' Configuration
Current File
Run Configurations
Junit Tests
PhonebookApplication

3. After the compilation is done, we get Completed initialization in the terminal



```
2022-11-22 16:24:06.916  WARN 21848 --- [  restartedMain] JpaBaseConfiguration$JpaWebConfiguration : spring.jpa.open-in-view is enabled by default. Therefore
2022-11-22 16:24:07.075  INFO 21848 --- [  restartedMain] o.s.b.d.a.OptionalLiveReloadServer       : LiveReload server is running on port 35729
2022-11-22 16:24:07.218  INFO 21848 --- [  restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port(s): 8080 (http) with context path
2022-11-22 16:24:07.399  INFO 21848 --- [  restartedMain] c.i.phonebook.PhonebookApplication       : Started PhonebookApplication in 3.639 seconds (JVM runni
2022-11-22 16:24:36.004  INFO 21848 --- [nio-8080-exec-3] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring DispatcherServlet 'dispatcherServlet
2022-11-22 16:24:36.005  INFO 21848 --- [nio-8080-exec-3] o.s.web.servlet.DispatcherServlet        : Initializing Servlet 'dispatcherServlet'
2022-11-22 16:24:36.008  INFO 21848 --- [nio-8080-exec-3] o.s.web.servlet.DispatcherServlet        : Completed initialization in 3 ms
```

4. Open Swagger UI with URL http://localhost:8080/swagger-ui/index.html#/phone-book-controller, from here we can test all the API calls.



basic-error-controller Basic Error Controller

phone-book-controller Phone Book Controller

POST /PhoneBook/add Add a Phonebook entry

PUT /PhoneBook/deleteByName/{name} Delete entries by name

PUT /PhoneBook/deleteByNumber/{number} Delete entries by number

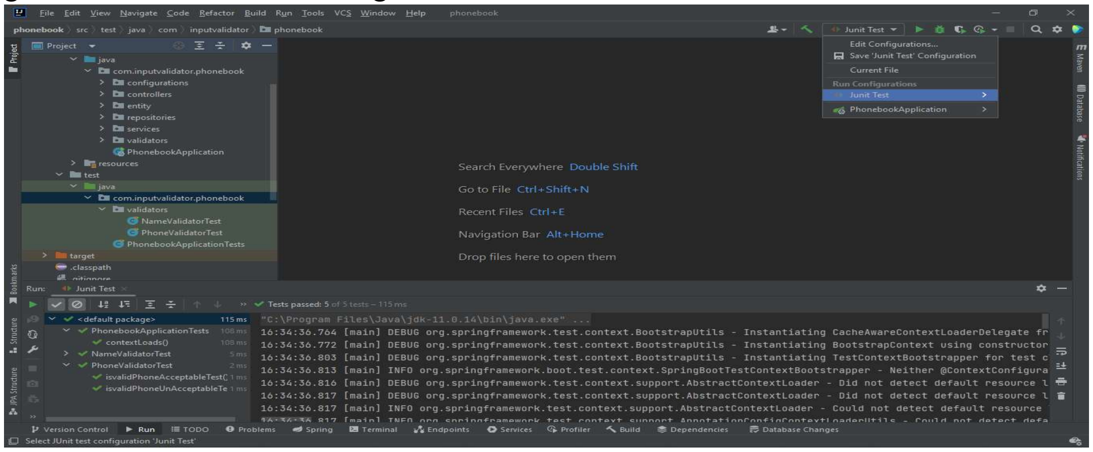
GET /PhoneBook/list Get all Phonebook list

GET /PhoneBook/list/audit Get AuditLog list

test-controller Test Controller

GET /test/demo testMethod

5. To run Junit test, select Junit test application in the popup and click the run ▶ in the gutter. The IDE starts running Junit tests



Tests passed: 5 of 5 tests – 115 ms

```
"C:\Program Files\Java\jdk-11.0.14\bin\java.exe" ...
16:34:36.764 [main] DEBUG org.springframework.test.context.BootstrapUtils - Instantiating CacheAwareContextLoaderDelegate fr
16:34:36.772 [main] DEBUG org.springframework.test.context.BootstrapUtils - Instantiating BootstrapContext using constructor
16:34:36.803 [main] DEBUG org.springframework.test.context.BootstrapUtils - Instantiating TestContextBootstrapper for test c
16:34:36.813 [main] INFO org.springframework.boot.test.context.SpringBootTestContextBootstrapper - Neither @ContextConfigura
16:34:36.816 [main] DEBUG org.springframework.test.context.support.AbstractContextLoader - Did not detect default resource l
16:34:36.817 [main] DEBUG org.springframework.test.context.support.AbstractContextLoader - Did not detect default resource l
16:34:36.817 [main] INFO org.springframework.test.context.support.AbstractContextLoader - Could not detect default resource
```

Select JUnit test configuration 'Junit Test'

Application Workflow:

1) The application starts from PhoneBookController, here the RestController and request mapping of the API calls is done. When the user selects GET/POST/PUT API calls, the application triggers this controller in this controller we validate the given input with the regex, these validators are present in this folder and all the operations are stores in audit log Validators - "phonebook\src\main\java\com\inputvalidator\phonebook\validators"
2) NameValidator class checks if the given input name is in the correct format using regex which is allowing for <first middle last>, <first last> or <last, first MI>, if its in given format it gives 200 code and saves the input to database from PhoneBookService class, for invalid it returns 400 code "phonebook\src\main\java\com\inputvalidator\phonebook\services"
3) PhoneValidator class checks if the given input number is in the correct format using regex for valid it returns 200 and for invalid it returns 400
4) addMember function in PhoneBookController class operates the inputs and adds them to database
5) deleteMemberByName function in PhoneBookController class deletes the data by name
6) deleteMemberByNumber function in PhoneBookController class deletes the data by number
7) getAuditLogList function in PhoneBookController class retrieves all the logs saved in database

8) Server returns 200 for valid input, 400 for invalid input with the error message and 404 if we try to remove non-existent name
9) SQL statements are written inside repositories folder with PhoneBookRepository class for inputting and delete the phonebook values, AuditlogRepository class to DB operations and view audit logs
10) Junit tests are written inside test folder with NameValidatorTest and PhoneValidatorTest with given validation data and CustomValidatorTest with my custom validation data

<u>Regular Expressions:</u>

- Name validation is done with the following regex expression.
  "^[A-Z]([a-zA-Z]*?\\'?[a-zA-Z]+?\\,?[ ]?\\-?\\.?){1,3}$"
- Phone validation is done with following regex expressions. These regex expressions ensure that all the scenarios are covered.

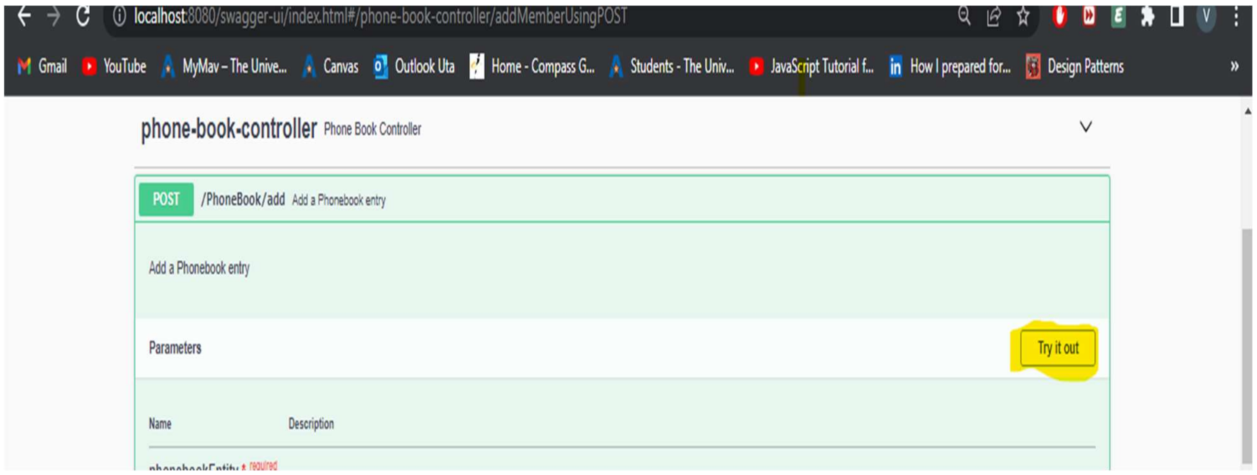
```
"^([1-9]{1}[0-9]{2}[- .]?){2}\\d{4}$"
"^((\\([1-9]{1}[0-9]{2}\\))|[1-9]{1}[0-9]{2})[- .]?\\d{3}[- .]?\\d{4}$"
"^\\d{1}((\\(\\d{3}\\))|\\d{3})[- .]?\\d{3}[- .]?\\d{4}$"
"^(\\+[1-9]{1}[0-9]?[0-9]?( )?){1}((\\(\\d{2,3}\\))|\\d{3})[- .]?\\d{3}[-
.]?\\d{4}$"
"^(\\d{5}[.]?)\\d{5}$"
"^\\d{5}$"
"^([0][1][1])[- .][1]?[- .]?(\\d{3}[- .]?){2}\\d{4}$"
"^([0][0])[- .][1]?[- .]?(\\d{3}[- .]?){2}\\d{4}$"
"^(\\d{3}[-]?)\\d{4}$"
```

<u>API Operations:</u>

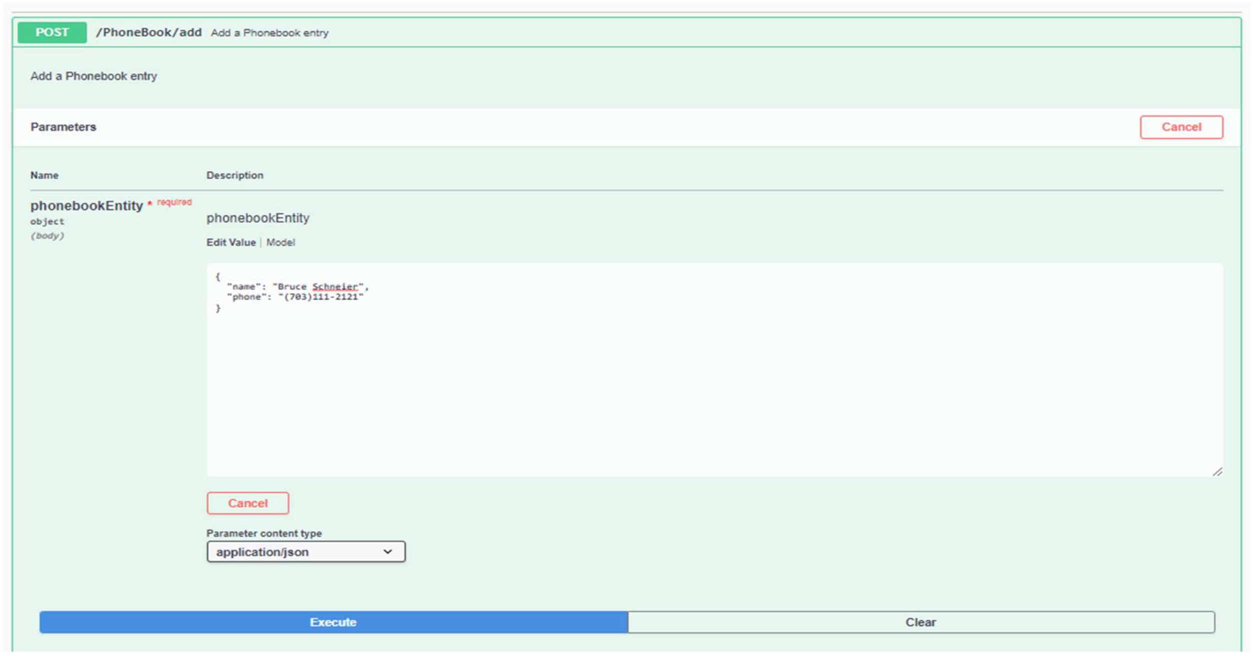
<u>POST Operation:</u>

Insert a new person into the database. Argument is a string object that contains the string elements name and phone number, and id is auto generated.

- Open swagger UI and click on phone-book-controller and in drop down select POST. Click 'Try it out'.

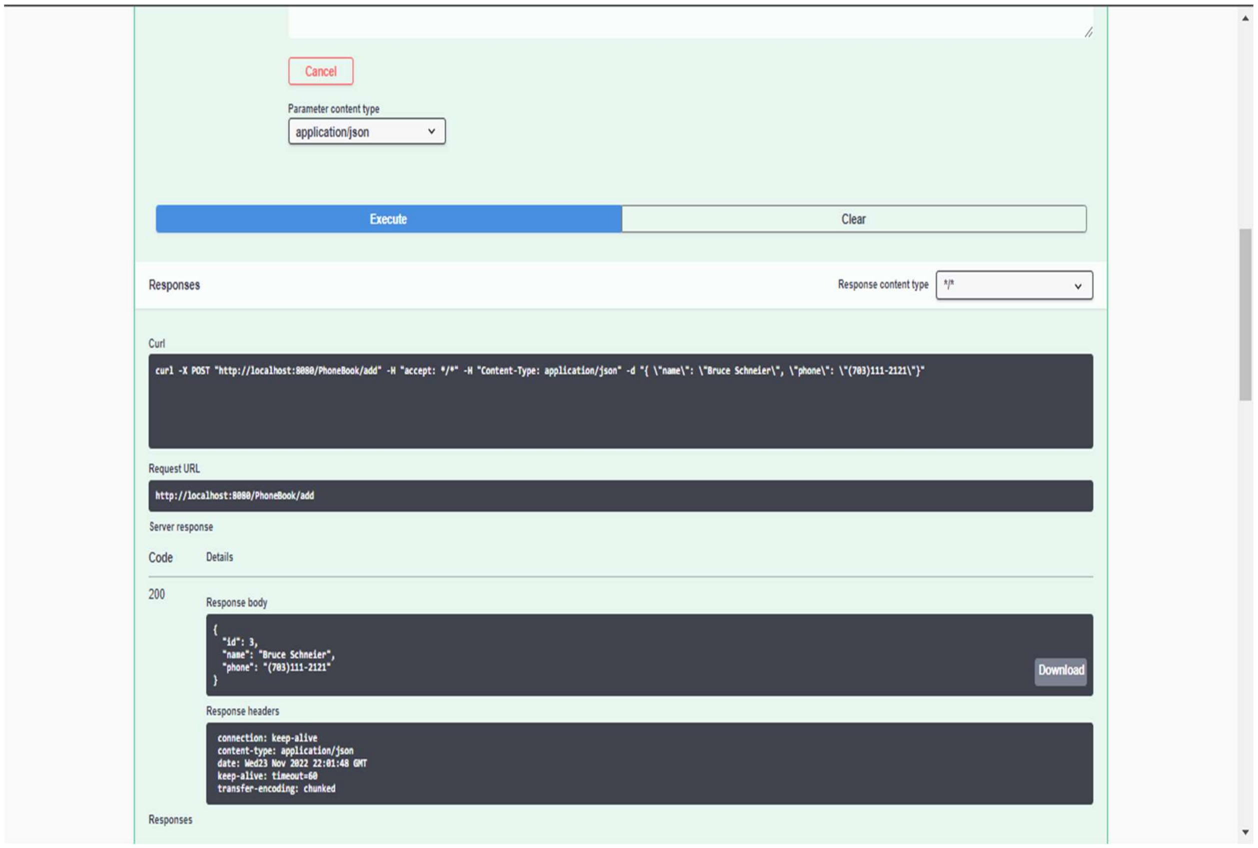


- Entry the name and phone as shown below and click execute; id is auto generated so no need to explicitly specify



```
{
  "name": "Bruce Schneier",
  "phone": "(703)111-2121"
}
```

- If all the validations are passed it will give 200 code and return the response as below



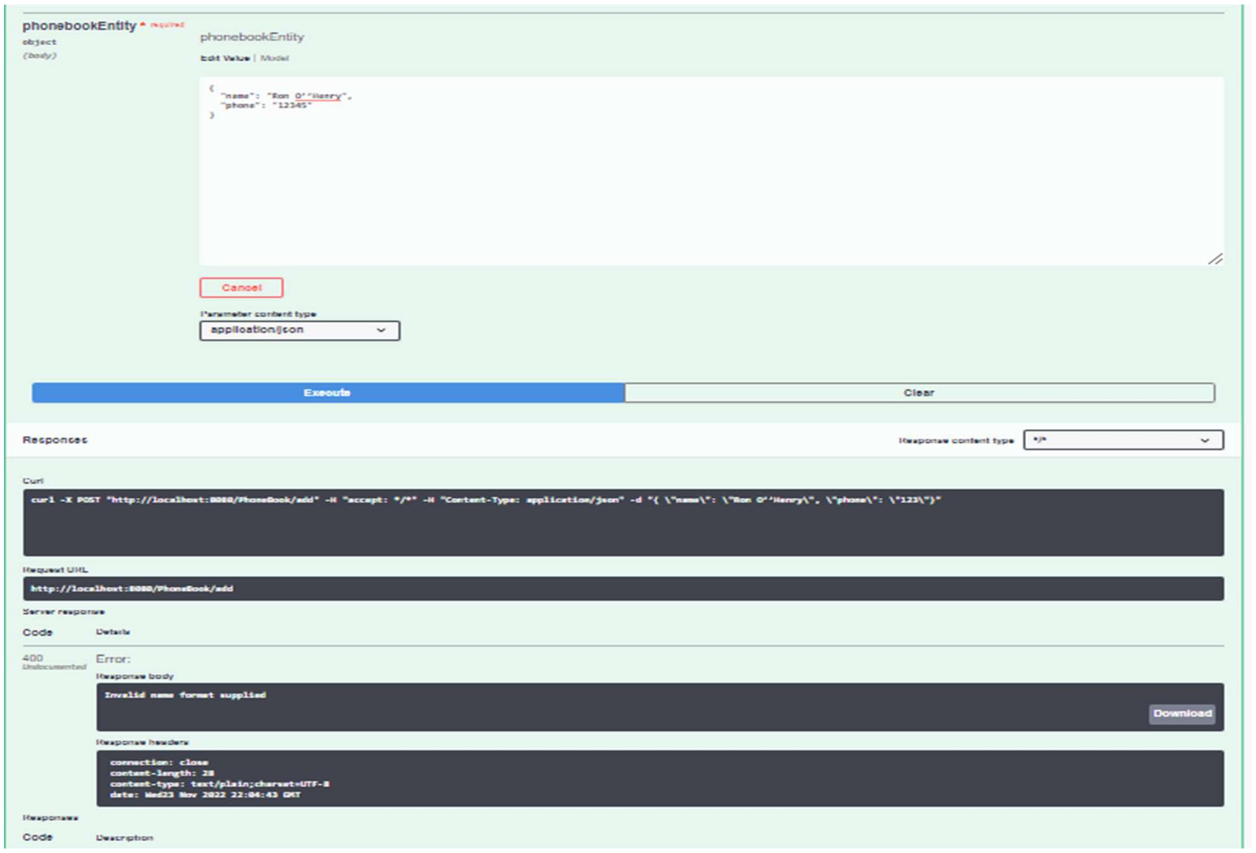
```
curl -X POST "http://localhost:8080/PhoneBook/add" -H "accept: */*" -H "Content-Type: application/json" -d "{ \"name\": \"Bruce Schneier\", \"phone\": \"(703)111-2121\"}"
```

```
http://localhost:8080/PhoneBook/add
```

```
{
  "id": 3,
  "name": "Bruce Schneier",
  "phone": "(703)111-2121"
}
```

```
connection: keep-alive
content-type: application/json
date: Wed23 Nov 2022 22:01:48 GMT
keep-alive: timeout=60
transfer-encoding: chunked
```

- If the name and phone number format are invalid, it will return the response as 400 with the error message as shown below.
- Invalid name



```
{
  "name": "Ron O''Henry",
  "phone": "12345"
}
```

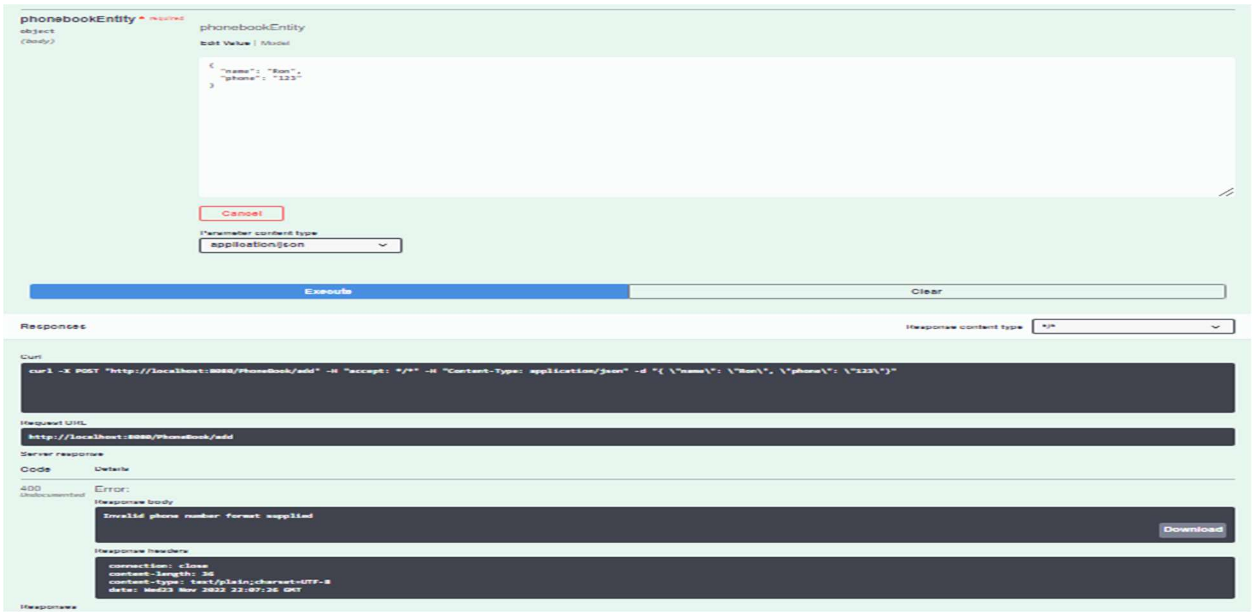
```
curl -X POST "http://localhost:8080/PhoneBook/add" -H "accept: */*" -H "Content-Type: application/json" -d "{ \"name\": \"Ron O''Henry\", \"phone\": \"123\"}"
```

```
http://localhost:8080/PhoneBook/add
```

```
Invalid name format supplied
```

```
connection: close
content-length: 28
content-type: text/plain;charset=UTF-8
date: Wed23 Nov 2022 22:04:43 GMT
```

- Invalid number



```
{
  "name": "Ron",
  "phone": "123"
}
```

```
curl -X POST "http://localhost:8080/PhoneBook/add" -H "accept: */*" -H "Content-Type: application/json" -d "{ \"name\": \"Ron\", \"phone\": \"123\"}"
```

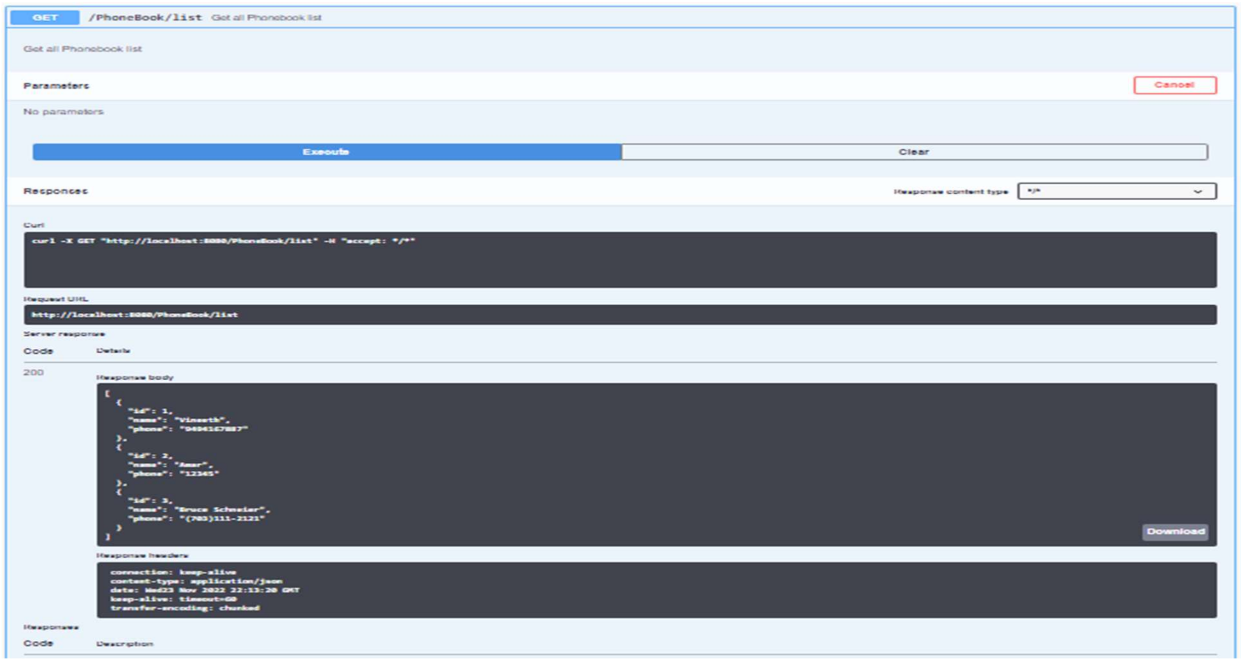
```
http://localhost:8080/PhoneBook/add
```

```
Invalid phone number format supplied
```

```
connection: close
content-length: 36
content-type: text/plain;charset=UTF-8
date: Wed23 Nov 2022 22:07:26 GMT
```

GET Operation:

This function returns a list from the database.



```
curl -X GET "http://localhost:8080/PhoneBook/list" -H "accept: */*"
```

```
http://localhost:8080/PhoneBook/list
```

200

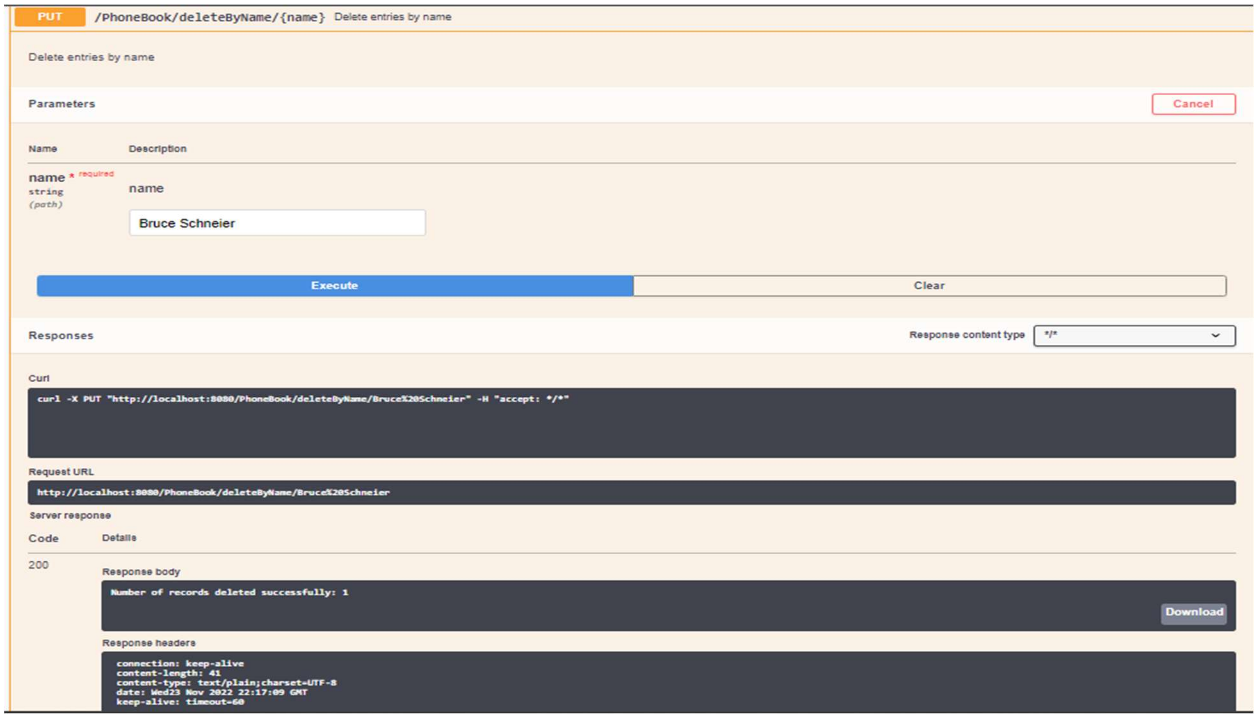
```
[
  {
    "id": 1,
    "name": "Vineeth",
    "phone": "9494167887"
  },
  {
    "id": 2,
    "name": "Amar",
    "phone": "12345"
  },
  {
    "id": 3,
    "name": "Bruce Schneier",
    "phone": "(703)111-2121"
  }
]
```

```
connection: keep-alive
content-type: application/json
date: Wed23 Nov 2022 22:13:20 GMT
keep-alive: timeout=60
transfer-encoding: chunked
```

PUT Delete Member by Name Operation:

Remove a specific person from the database. The name as a string is the argument.

- Valid Data



name: Bruce Schneier

```
curl -X PUT "http://localhost:8080/PhoneBook/deleteByName/Bruce%20Schneier" -H "accept: */*"
```

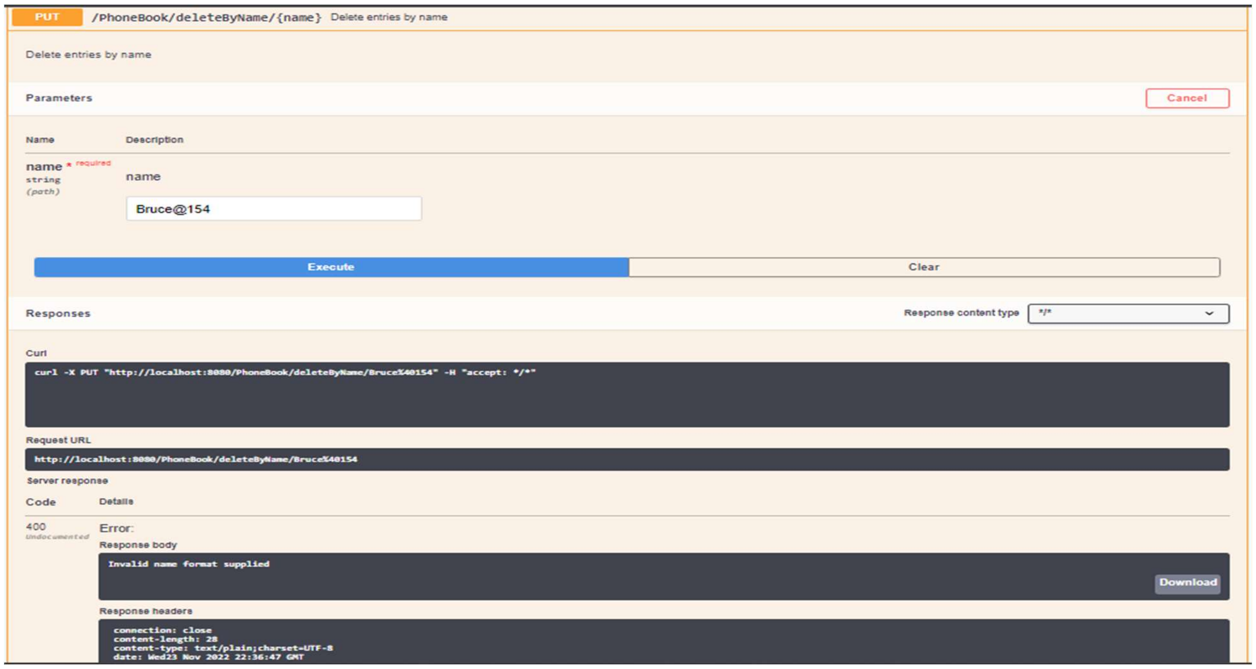
```
http://localhost:8080/PhoneBook/deleteByName/Bruce%20Schneier
```

200

```
Number of records deleted successfully: 1
```

```
connection: keep-alive
content-length: 41
content-type: text/plain;charset=UTF-8
date: Wed23 Nov 2022 22:17:09 GMT
keep-alive: timeout=60
```

- Invalid format a status 400 code will be returned with invalid name format response.



Bruce@154

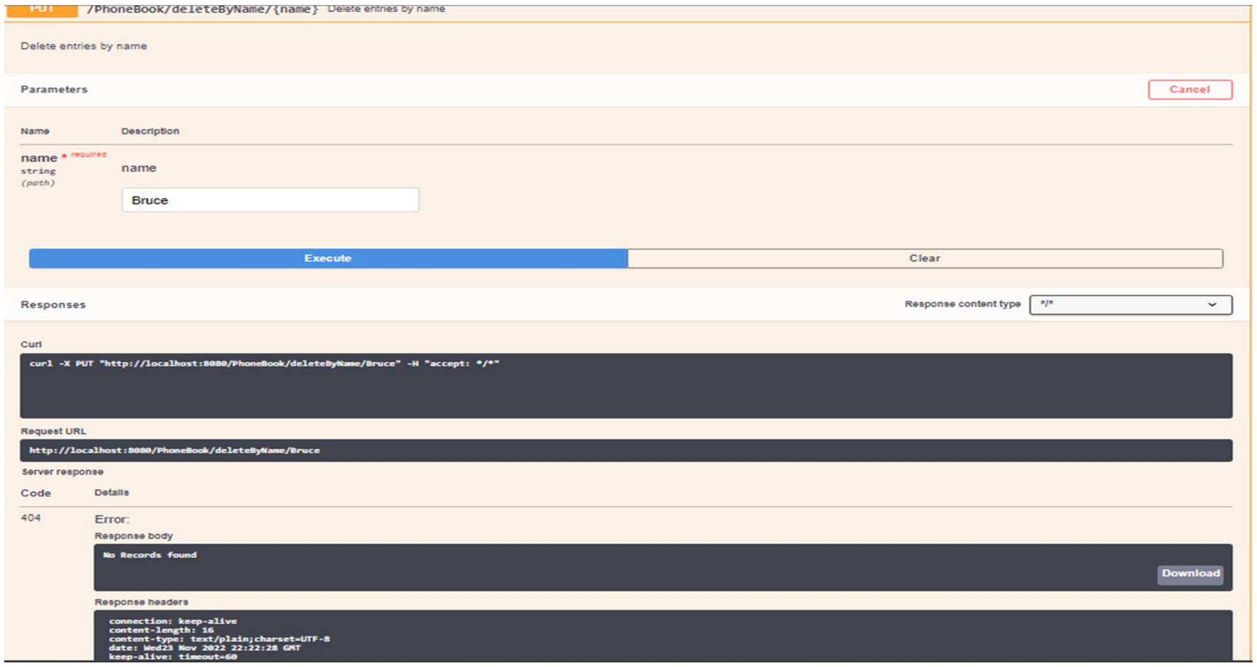
Curl
```
curl -X PUT "http://localhost:8080/PhoneBook/deleteByName/Bruce%40154" -H "accept: */*"
```
Request URL
```
http://localhost:8080/PhoneBook/deleteByName/Bruce%40154
```
Server response

Code: 400 *Undocumented*

Details: Error:

Response body
```
Invalid name format supplied
```
Response headers
```
connection: close
content-length: 28
content-type: text/plain;charset=UTF-8
date: Wed23 Nov 2022 22:36:47 GMT
```

- A 404 status code will be returned if a non-existent name is removed from the database



Bruce

Curl
```
curl -X PUT "http://localhost:8080/PhoneBook/deleteByName/Bruce" -H "accept: */*"
```
Request URL
```
http://localhost:8080/PhoneBook/deleteByName/Bruce
```
Server response

Code: 404

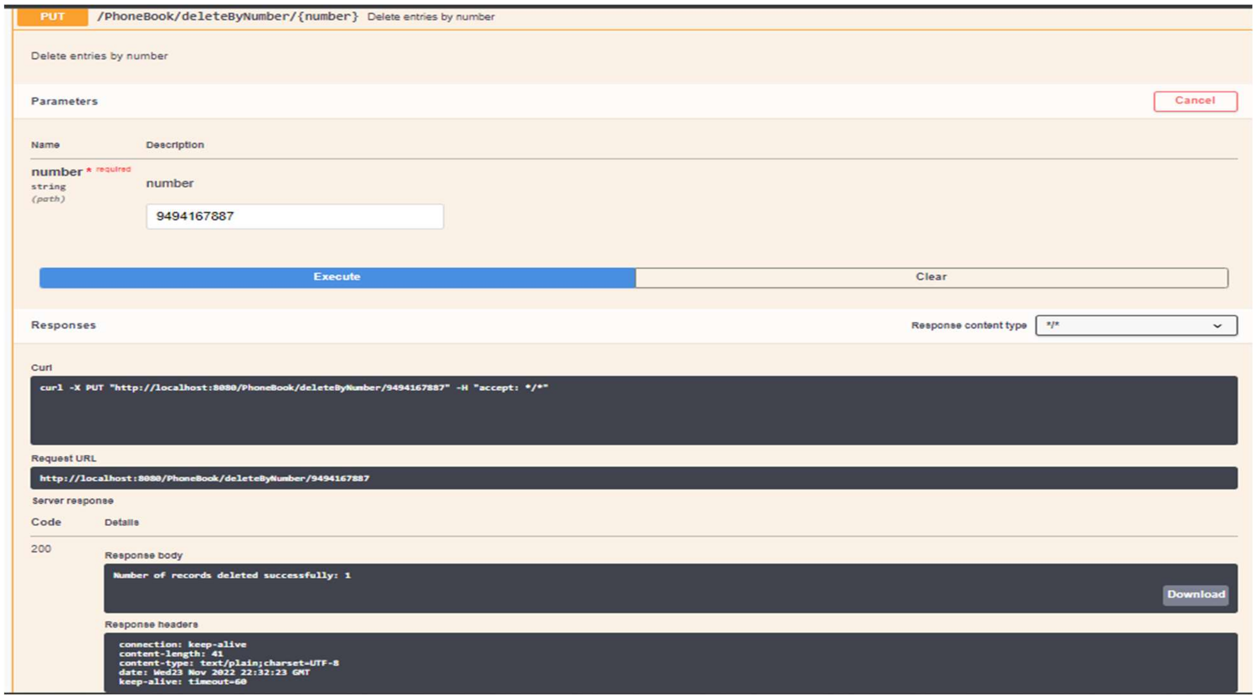
Details: Error:

Response body
```
No Records found
```
Response headers
```
connection: keep-alive
content-length: 16
content-type: text/plain;charset=UTF-8
date: Wed23 Nov 2022 22:22:28 GMT
keep-alive: timeout=60
```

PUT Delete Member by Number Operation:

Remove a specific person using their phone number The phone number as a string is the argument.

- Valid operation



Curl

```
curl -X PUT "http://localhost:8080/PhoneBook/deleteByNumber/9494167887" -H "accept: */*"
```

Request URL

```
http://localhost:8080/PhoneBook/deleteByNumber/9494167887
```

Server response

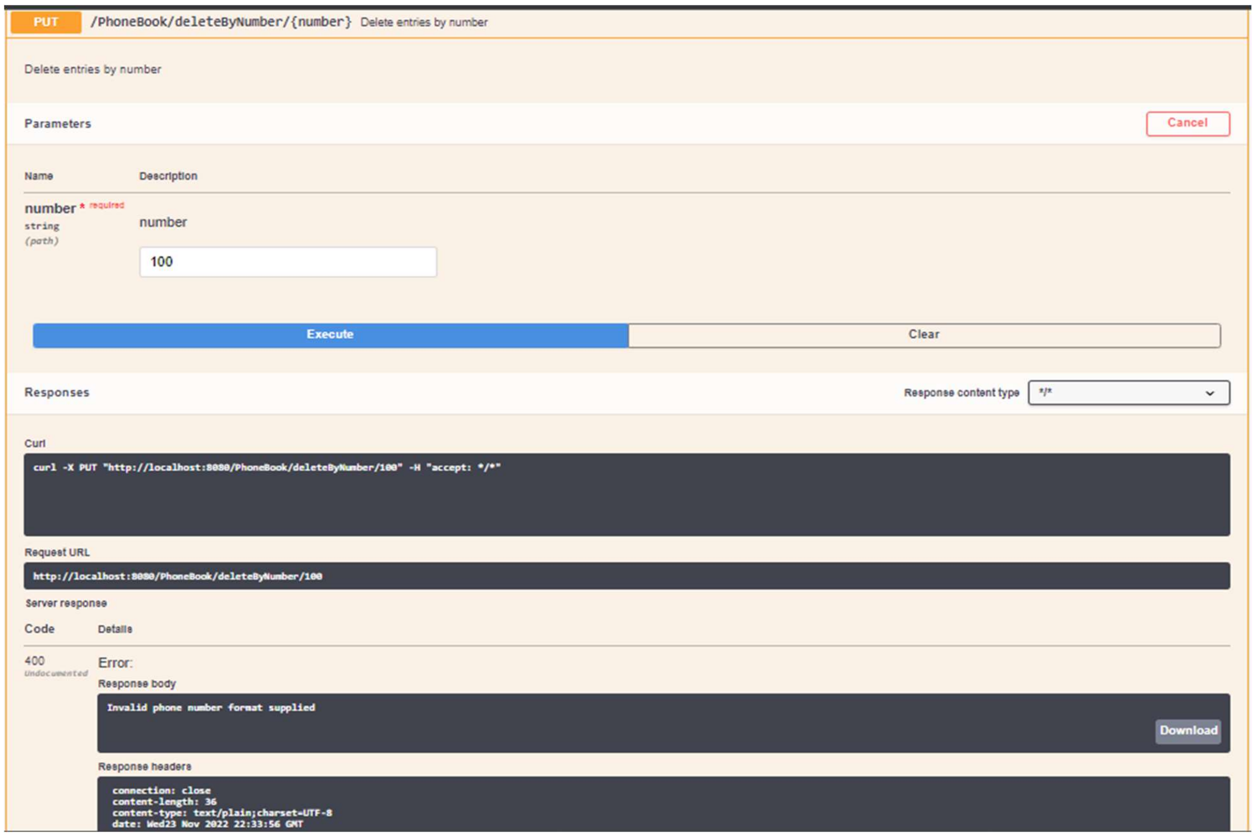
Code	Details
200	Response body

```
Number of records deleted successfully: 1
```

Response headers

```
connection: keep-alive
content-length: 41
content-type: text/plain;charset=UTF-8
date: Wed23 Nov 2022 22:32:23 GMT
keep-alive: timeout=60
```

- Invalid format a status 400 code will be returned with invalid number format response.



Curl

```
curl -X PUT "http://localhost:8080/PhoneBook/deleteByNumber/100" -H "accept: */*"
```

Request URL

```
http://localhost:8080/PhoneBook/deleteByNumber/100
```

Server response

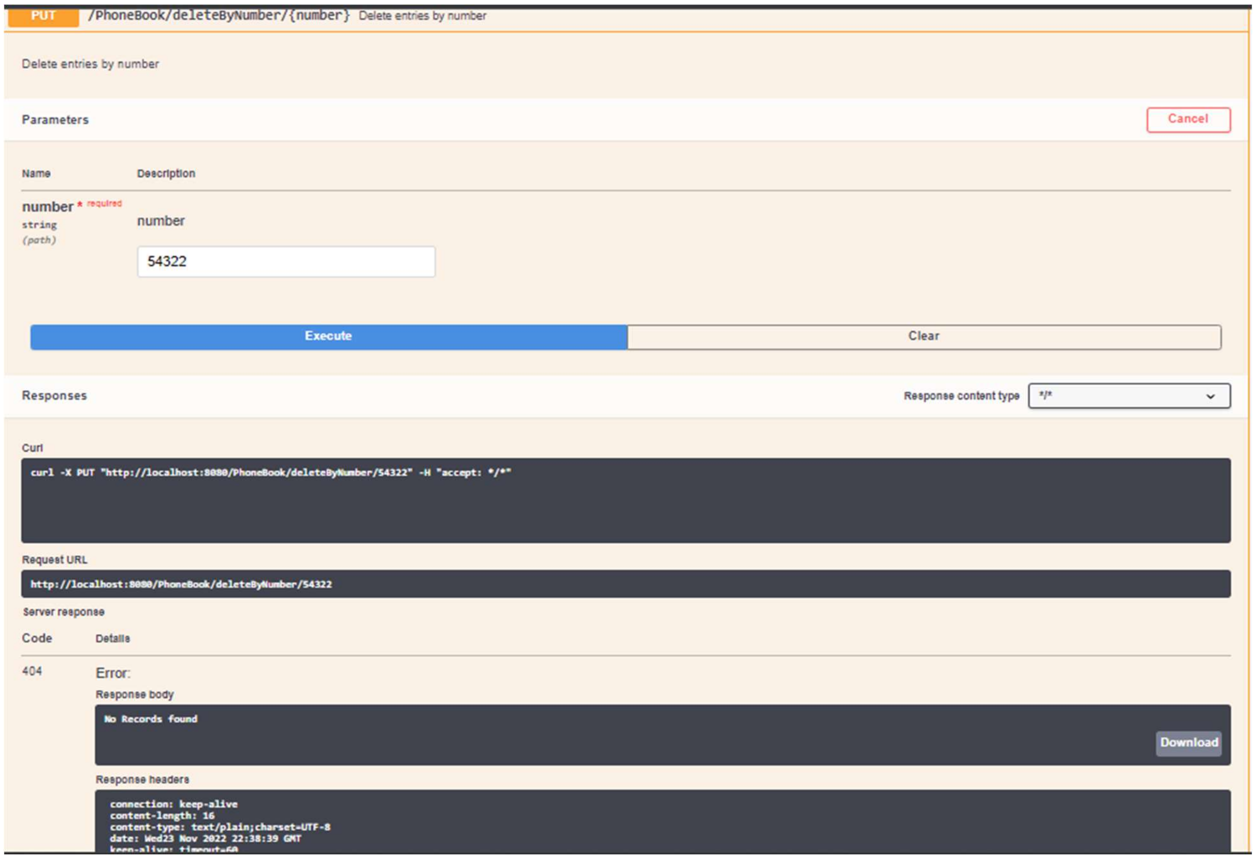
Code	Details
400 Undocumented	Error: Response body

```
Invalid phone number format supplied
```

Response headers

```
connection: close
content-length: 36
content-type: text/plain;charset=UTF-8
date: Wed23 Nov 2022 22:33:56 GMT
```

- A 404 status code will be returned if a non-existent number is removed from the database



number: 54322

Response content type: */*

Curl
```
curl -X PUT "http://localhost:8080/PhoneBook/deleteByNumber/54322" -H "accept: */*"
```

Request URL
```
http://localhost:8080/PhoneBook/deleteByNumber/54322
```

Server response

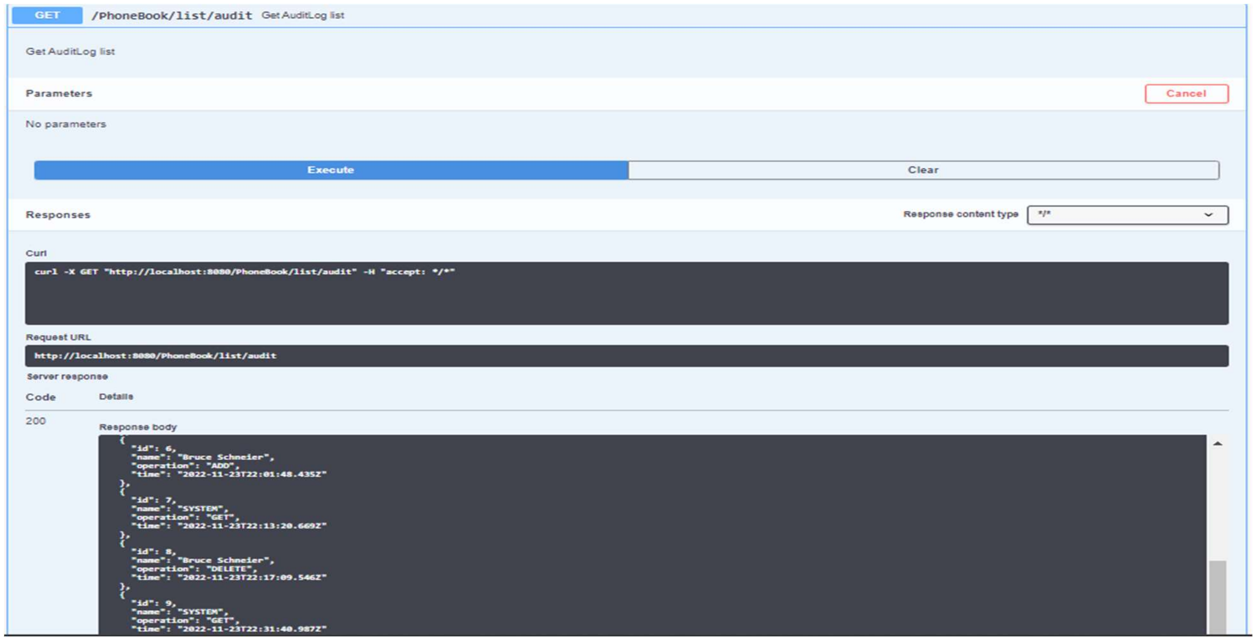
Code	Details
404	Error:

Response body
```
No Records found
```

Response headers
```
connection: keep-alive
content-length: 16
content-type: text/plain;charset=UTF-8
date: Wed23 Nov 2022 22:38:39 GMT
keep-alive: timeout=60
```

Get Audit Log List Operation:

This returns the audit log list which has been saved in the database after every operation.



Response content type: */*

Curl
```
curl -X GET "http://localhost:8080/PhoneBook/list/audit" -H "accept: */*"
```

Request URL
```
http://localhost:8080/PhoneBook/list/audit
```

Server response

Code	Details
200	Response body

```
  {
    "id": 6,
    "name": "Bruce Schneier",
    "operation": "ADD",
    "time": "2022-11-23T22:01:48.435Z"
  },
  {
    "id": 7,
    "name": "SYSTEM",
    "operation": "GET",
    "time": "2022-11-23T22:13:20.669Z"
  },
  {
    "id": 8,
    "name": "Bruce Schneier",
    "operation": "DELETE",
    "time": "2022-11-23T22:17:09.546Z"
  },
  {
    "id": 9,
    "name": "SYSTEM",
    "operation": "GET",
    "time": "2022-11-23T22:31:40.987Z"
```

Assumptions:

- The same individual can have many numbers. However, the same number cannot be used for more than one individual.
- All the formats given in assignment for phone and number are checked.
- Audit logs are saved and retrieved from database.
- Assuming the user enters the phone number with country code.

Pros:

- Fast and simple
- Implemented prepared statements in API to support parameterized queries in SQL queries. Any attacker will be unable to perform a SQL injection attack because of this.
- Implemented swagger ui to visualize and interact with the API

Cons:

- Can perform one operation at a time for PUT and POST

Bonus:

1. Used SQL Lite database to store the data. Implemented API to support parameterized queries (prepared statements).

2. Implemented automated unit tests for good inputs, bad inputs and custom inputs provided by me.