**Project Report**

**Introduction:**

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

Therefore, I have created a REST API application that maintains a phonebook of names and phone numbers. The program shall be capable of receiving and storing a list of people with their full name and telephone numbers.

The application includes the following API endpoints:

* GET/Phonebook/readContacts – Produce a list of number of contacts in the database.
* POST/Phonebook/createphonebook – Add a new contact record to the database.
* PUT/Phonebook/deleteByName – Remove contact from database with given name.
* PUT/Phonebook/deleteByNumber – Remove contact from database with given phone number.

**Description on how the application works:**

I have used Spring Boot framework to create a REST API that maintains the phonebook of names and telephone numbers.

Spring Boot provides a powerful batch processing and manages REST endpoints.

In Spring Boot, everything is auto configured; no manual configurations are needed.

Spring Boot provides a flexible way to configure Java Beans and Database Transactions.

I have used SQLite database to store the data. SQLite can be easily integrated with Spring Boot to create a REST API.

All the code that has been written is in JAVA language.

The architecture I used includes: The below packages are present in the /src/main folder

* Controller Package: It has a PhonebookController class, which is used to call API endpoints. Below is the tabular format

|  |  |  |
| --- | --- | --- |
| **Method Name** | **API call** | **Path** |
| createPhonebook | RequestMethod.POST | /createphonebook |
| readContacts | RequestMethod.GET | /readContacts |
| deleteByNumber | RequestMethod.PUT | /deleteByNumber |
| deleteByName | RequestMethod.PUT | /deleteByName |

* Entity Package: It has Phonebook class, which has all the attributes for the application that would be stored in the database. Below is the tabular format

|  |  |
| --- | --- |
| **Attributes** | **Methods** |
| name | getName(), setName() |
| phoneNumber | getPhoneNumber(), setPhoneNumber() |

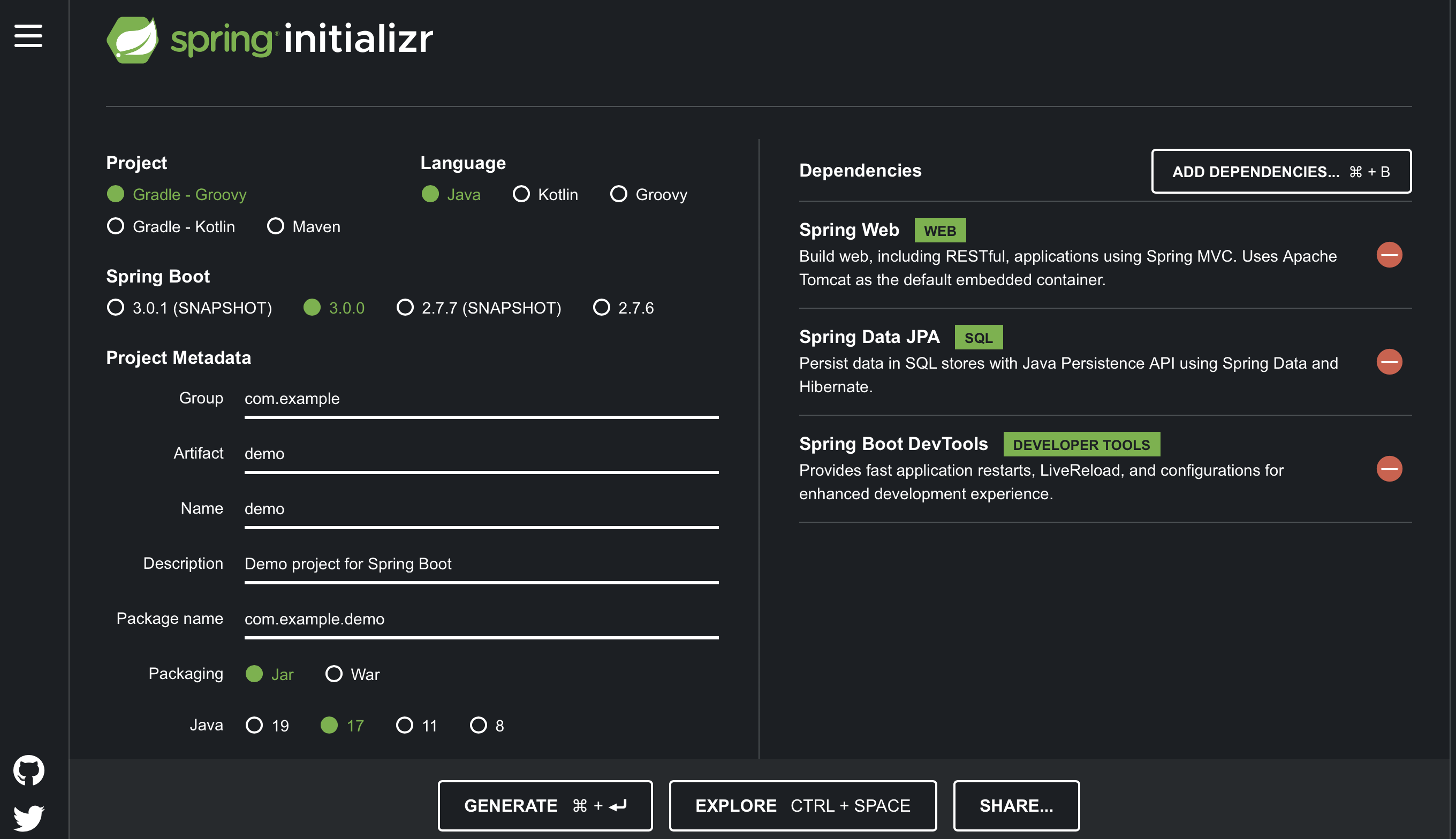
* Repository Package: It has PhonebookRepository interface which extends JPARepository. It contains the APIs for basic CRUD operations. For managing the phonebook entity and saving them to the database is done here.
* Dialect Package: It has PhoneBookDialect class which contains the mapping between java language data types and database data type. It allows Hibernate to generate SQL optimized for a particular relational database.
* Service Package: It has two classes, PhonebookService class and ValidationService Class.
  + PhonebookService: This is used to add the business logic to the application. PhonebookController class calls endpoints using service class as it has all the logic written for create, list and delete.
  + ValidationService: It has all the validations for the names and phone numbers. Used the regular expressions to validate the entities. This class has two methods(validatePhone(Phonebook contact), validateName(Phonebook contact)) which will be called in the PhonebookService class before being saved to the database.
* Exception Handling Package: It has two classes, BadRequest class and NotFound class.
  + BadRequest: This class is called in the PhoneBookService class if any bad request is given by the user and hence throws an exception with 400 status code.
  + NotFound: This class is called in the PhoneBookService class if user wants to delete a record which is not present in the database and hence throws an exception with 404 status code.
* Main Class (PhoneBookApplication): This is the class which is first called in the application that starts the API.

There is an audit log which is generated for all the operations done in the application (Create, List and Delete) with the timestamp. This is generated at the PhoneBookService class, used File functionality of java to generate a log that is saved on the disc with an extension as log.txt

Additionally, added Junit test cases to the project to ensure the project code is working as expected or not. This is present in the path /src/test folder.

**Instructions for Installation, Setup and Compilation/Build:**

* Must have Java version 17.x, the latest one.
* Download IntelliJ IDEA Community version to develop your application and Postman for API call testing.
* To create a Rest API using Spring Boot framework, we need Spring Boot dependencies.
* Go to <https://start.spring.io> website and create a project by adding the below mentioned dependencies



* Then, open IntelliJ IDEA and import the spring boot project which you have generated by adding the above dependencies and select maven builder.
* Also, add the below mentioned dependencies directly to the pom.xml file.

Text

Description automatically generated

Graphical user interface, text

Description automatically generated

* + These are the dependency for SQLite Database

Text

Description automatically generated

* + This is the dependency for spring boot annotations.

Text

Description automatically generated with medium confidence

* + This is the dependency for Junit test cases.

Text

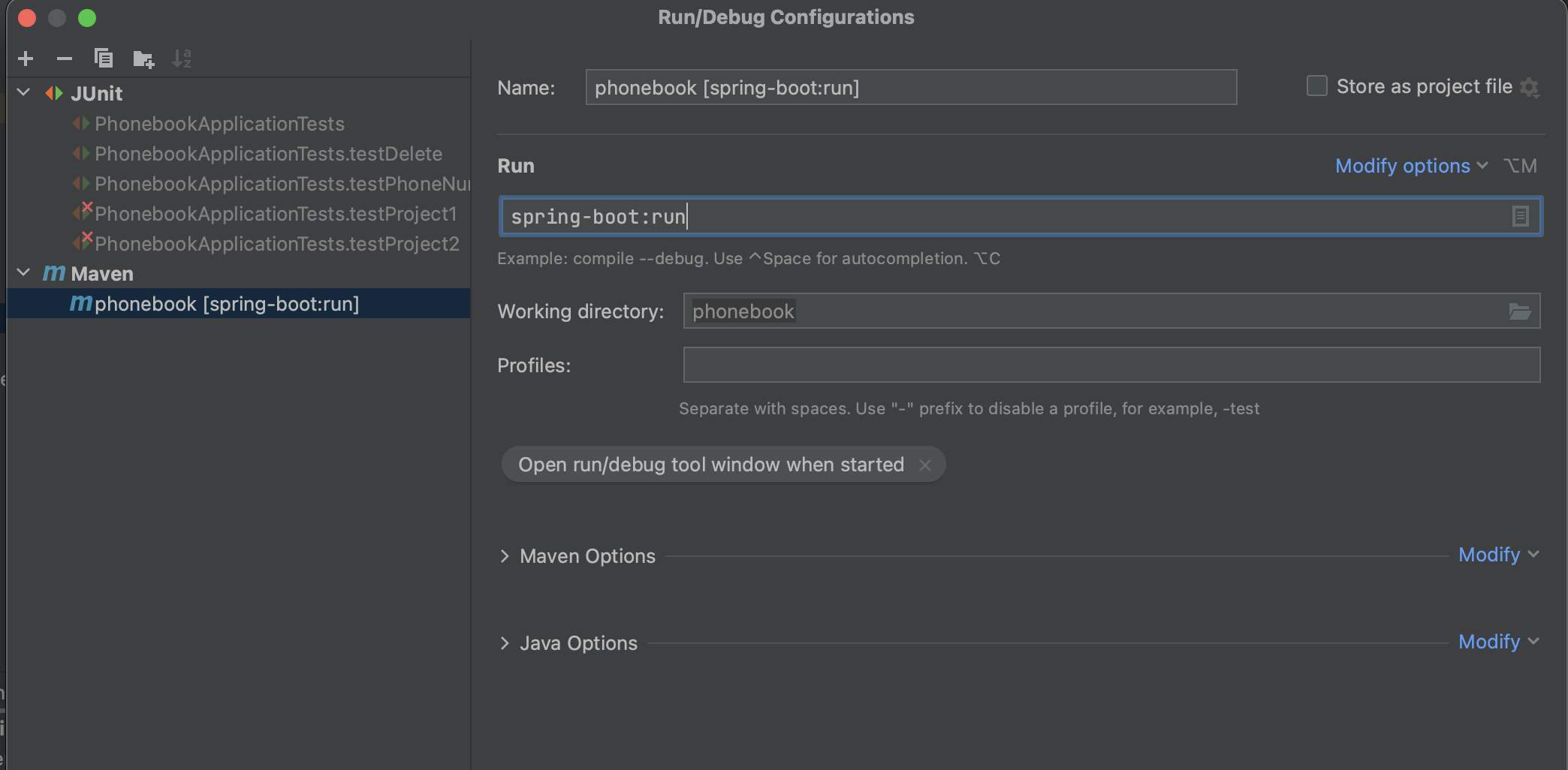
Description automatically generated

* + This is the dependency used for validation of NotNull and NotBlank fields.
* After all the above steps, use maven builder to build the project for all the dependencies to work as expected. Below is the screenshot for the same.

Text

Description automatically generated

* Then, make sure to edit the configuration to start the application properly. Below are the instructions for the same



* + Go to run configuration and give any name in the Name field and make sure to give maven boot:run in the Run field as mentioned above in the screenshot.
* Now, run the application in maven boot:run mode as shown below

Graphical user interface, text, application

Description automatically generated

* + Start the application by pressing the Green play button.
* Application starts and you can see the console as shown below

Text

Description automatically generated

**Execution Instructions:**

* Now, once the application is started in the boot:run mode, go to postman and give the appropriate endpoints to test your application.
* Use Get call to display all the records in the database as shown below

Graphical user interface, text, application, email

Description automatically generated

* Use Post call to add contact to the database as shown below

Graphical user interface, text, application, email

Description automatically generated

* Use Put call to delete a record from database by name as well as by phone number

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

* As you can see from the above screenshots, for all the successful calls, we are getting a status code of 200 OK, which is a proper status code for acceptable API calls.
* Now, for the bad request, such as user entering wrong details, we will get a 400 Bad Request status code with an error message as shown below

Graphical user interface, text, application, email

Description automatically generated

* Similarly, we will get a 404 Not Found status code for deletion of data which is not present in the database with an appropriate error message as shown below

Graphical user interface, text, application, email

Description automatically generated

* Audit log is generated for all the operations performed above which is shown below

Text

Description automatically generated

**Assumptions Made:**

* Assumed that the phone numbers 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.
* Assumed 5-digit extensions only for dialing from one internal phone to another.
* Assumed 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.
* Assumed 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.
* Assumed valid inputs for Name are as follows:
  + Bruce Lee
  + Lee, Bruce
  + Lee, Bruce Wayne
  + O’Malley, John F.
  + John O’Malley-Smith
  + Cheri
* Assumed each word in the name must start with a capital letter
* Assumed each word before space should have at most 10 letters
* Assumed each word in a name separated with space must have a capital letter in the beginning.

**Pros/Cons of my approach:**

**Advantages:**

* Easy Development
* No additional XML configuration
* Less Source codes
* Simple Setup and Management
* Out-of-the-box functionality
* Provided Regular expressions for validation of the user inputs could prevent input validation attack

**Disadvantages:**

* Lack of control.
* Spring Boot creates a lot of unused dependencies, resulting in a large deployment file.
* Complex and time-consuming process.
* Not suitable for large scale projects.
* Regular expressions are not completely invincible. Hence, malicious information as input from attackers could trigger input validation attack.