CSCU9YW

Contact Database Web Service

2636515

March 2020

Table of Contents

1 Problem Overview

This project aims to create a web service that acts as a contacts database. The web service at a basic level should provide methods to retrieve, add, edit and delete contacts. Contacts should store details such as first and last name, address, town/city, postcode and telephone number. The following subsections will describe the project listing any assumptions made throughout the development.

1.1 Description

The objectives specified in the assignment outline are achieved using REST, based on Spring Boot. The contact details are stored in an external MySQL database. The Spring Boot application will manage the contact information stored in the MySQL contact table. A Spring MVC controller is implemented with relevant URL mappings and handler methods. These handler methods provide the functionality to create, update, delete and retrieve contacts. Within Spring Boot these CRUD operations are made easier through JPA repositories; this application implements a CrudRepository. The form-based client interface is built on thymeleaf templates supported by the MVC controller. The web application is created with three views: contacts view, edit contact and add contact view. The contacts view displays a list of the contacts from the database in tabular format. The user can either find all contacts or search for groups of contacts using the search bar feature. The screen also has options for deleting or editing the retrieved contacts. The edit/add contact screens allow users to add a new contact or edit the details of an existing contact. The Spring application also provides validation checks used to ensure form input when adding or editing a contact is correct. Any errors are handled both in the application and the interface allowing users to make the appropriate corrections. The web service, therefore, provides a client interface that facilitates the retrieval, editing, creating and deleting of contacts. The web service also allows users to interact with groups of contacts while ensuring error cases are handled appropriately. This is a general overview of some of the basic and additional functionality implemented in the web service. The upcoming sections will explore the structure and main classes used in greater detail.

1.2 Assumptions

There were no major assumptions made throughout the development of the web service.

2 Project Solution

This section provides a detailed discussion of the web service implementation. It provides information on the overall structure of the application with a description of the main classes used. Appropriate screenshots of the web service and the messages exchanged are also included in this section.

2.1 Structure

The Spring Boot application consists of a three-layer architecture. The three separate layers each have distinctive responsibilities and communicate using a hierarchical structure.

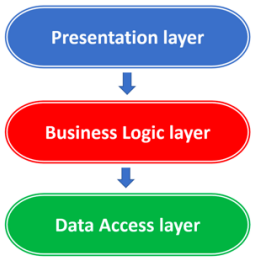

Figure 1. Layers of the Spring Boot Architecture

Presentation Layer: This is the front-end interface that presents the features of the application to the user. The interface consists of views and is used to handle HTTP requests.

Business Logic Layer: This layer contains the business logic that includes the primary functions of the application. It also handles any validation and processes data passing between the other layers.

Data Access Layer: This layer is responsible for interacting with the database to retrieve, edit, add and delete contact data.

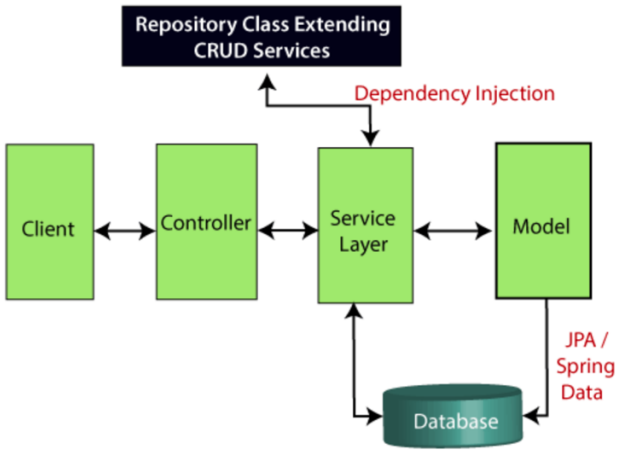

Figure 2. Structure of the Spring Boot Web Service

When the user interacts with the web service client the following processes are initiated:

1. The controller receives an HTTP request from the client – GET, POST, etc.
2. The request is processed by the controller and forwarded to the service layer
3. The Service layer preforms the business logic on the data from the database mapped with the entity model class
4. The MVC controller returns the thymeleaf view to the client if no error has occurred

2.2 Main Classes

The functionality described in the previous section is achieved through implementing a number of classes some of which are mentioned in Figure 2. This section discusses the core classes implemented to achieve the service and front-end client functionality. The structure and content of the classes can be found in the appendices at the end of the report.

2.2.1 MySQL Database

A MySQL table 'contact' is created within the 'contactsdb' database schema. The data structure contains each contact and their associated details which include first name, last name, telephone number, address, city and postcode. The telephone number uniquely defines each contact and is therefore set as the primary key. The table is executed using the MySQL script in appendix 1. An overview of the table is displayed in Figure 3.

#	Field	Schema	Table	Type
1	telephoneNumber	contactsdb	contact	VARCHAR
2	lastName	contactsdb	contact	VARCHAR
3	firstName	contactsdb	contact	VARCHAR
4	address	contactsdb	contact	VARCHAR
5	city	contactsdb	contact	VARCHAR
6	postcode	contactsdb	contact	VARCHAR

Figure 3. Overview of the MySQL 'contact' table

2.2.2 Maven Dependencies

The pom.xml file in Appendix 3 specifies a number of dependencies that are managed in this project using Maven. Some of the core dependencies that facilitate the functionality of the contacts web service are described below.

- **Spring-boot-starter-web:** starter for building RESTful services and applications using Spring MVC. Utilises Tomcat as the default embedded container
- **Spring-boot-starter-thymeleaf:** starter for building MVC web applications using Thymeleaf views
- **Spring-boot-starter-data-rest:** Exposes Spring data repositories over REST using Spring Data REST
- **Mysql-connector-java:** Allows connection between Spring Boot application and the external MySQL database
- **Spring-boot-starter-data-JPA:** It includes spring data, hibernate and JPA functionality
- **Spring-boot-starter-validation:** Uses Java Bean validation with hibernate validator
- **Bootstrap:** Facilitates bootstrap implementation on the thymeleaf views

2.2.3 Entity Class

To enable the storage of objects in the database using JPA an entity class is defined. A JPA entity class can represent objects in the database and the underlying table structure. As the web service is based on the 'contact' table, a 'contact' entity object is created. The entity class in Appendix 4 is annotated with @Entity, which marks the class as an entity object for the table 'contact'. There are six fields marked with @Column that represent the database table columns. Every JPA entity must contain a primary key; the telephone number is the primary key and is therefore annotated with @Id. Each field is also annotated with @Pattern which is used for Bean Validation. Bean Validation works by defining constraints to the fields of the entity class. This is achieved by marking them with certain annotations, @Pattern for example. The constraints in the contact class define regex expressions used to validate form input from the client.

2.2.4 Repository Class

JPA repositories are java interfaces that associate with a JPA entity – the contact class described in the previous section. The repository can only perform data access operations on the contact entity class and its data attributes. In Appendix 6 the 'ContactRepository' extends the 'CrudRepository' supplied with the JPA 'contact' entity and the telephone number primary key. The repository is now aware of the database columns it can operate with. The 'CrudRepository' provides operations to retrieve, add, edit and delete contact records from the database. Additional methods, findAll() and search(), are implemented in the repository interface. findAll() is utilised to find the list of all contacts from the database. search () executes a database query to find the list of contacts that contain the searchString inputted in the client web interface.

2.2.5 Service Class

The business logic of the contact applications is contained within this class – the @Service annotation is used to indicate that this class resides within the service layer. The class contains several CRUD methods that provide the functionality to retrieve, edit, add and delete contacts. getSearch() either returns the list of contacts containing the searchString or the list of all contacts if the searchString is empty. The other methods implement the save(), findById() and deleteById() methods of 'CrudRepository' to create, find and delete a contact.

2.2.6 Controller Class

The contact controller is a REST controller with CRUD endpoints. The controller is responsible for handling HTTP requests, preparing the model and returning the view to be rendered as a response. The handler methods are based on the @RequestMapping annotation and can serve multiple different requests that include retrieving, editing, adding and deleting contacts The saveNewContact() and saveUpdateContact() methods are used to validate form input. The @ModelAttribute annotation maps a form's input to a bean and the @Valid annotation validates the bean against the constraints defined in the entity class.

2.2.7 Views

The spring MVC returns separate three views that combine to represent the web service client. The views include an index, edit-contact and add-contact page. Thymeleaf provides the functionality that enables the creation of the client web application. Thymeleaf is a Java-based library that provides support for serving HTML in web applications. Thymeleaf is combined with Bootstrap to create the final web design. Bootstrap is a front-end framework utilised to create modern applications.

2.3 Web Application Design and Functionality

Figure 4 gives an overview of the web Application. The application facilitates CRUD operations through a number of user interface features. The output and how the features relate to the CRUD operations will be demonstrated in the following Figures.

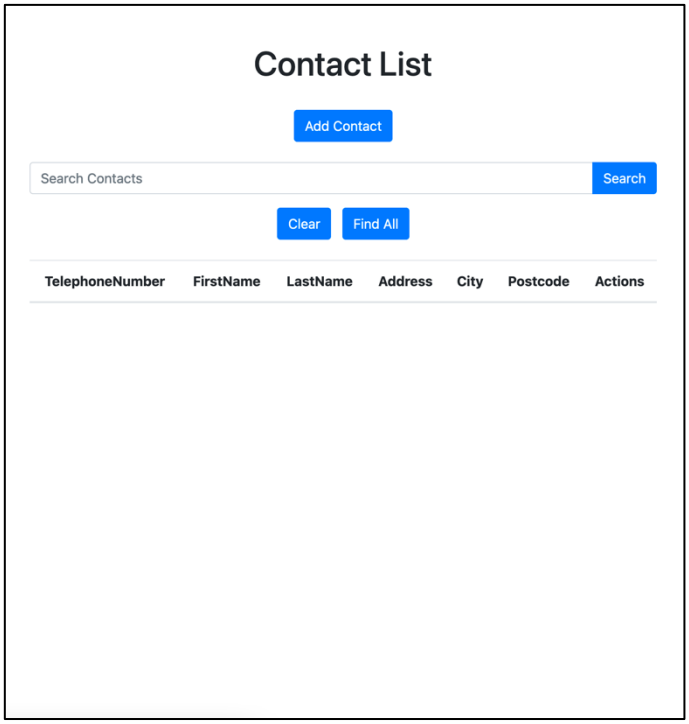

Figure 4.

Clicking the 'findAll' button retrieves the list of all contacts in the database in tabular format. This output can be cleared by pressing the 'Clear' button which redirects users to the screen in Figure 4.

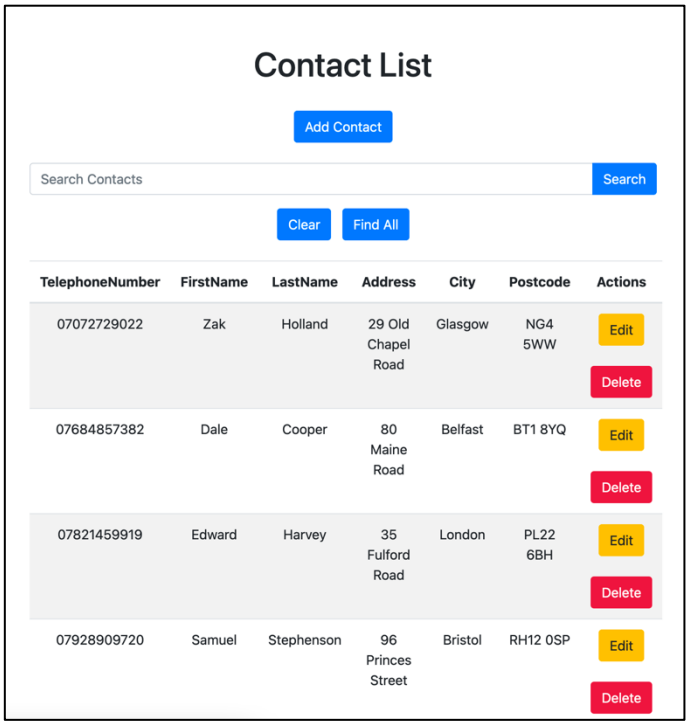

Figure 5.

Figure 6 shows the functionality of the search bar feature. This allows users to input any string and the application will return a list of contacts that contain the characters entered. The example in Figure 6 returns all contacts in a specific city – London. The search feature is case insensitive and can be invoked by pressing the ‘search’ button.

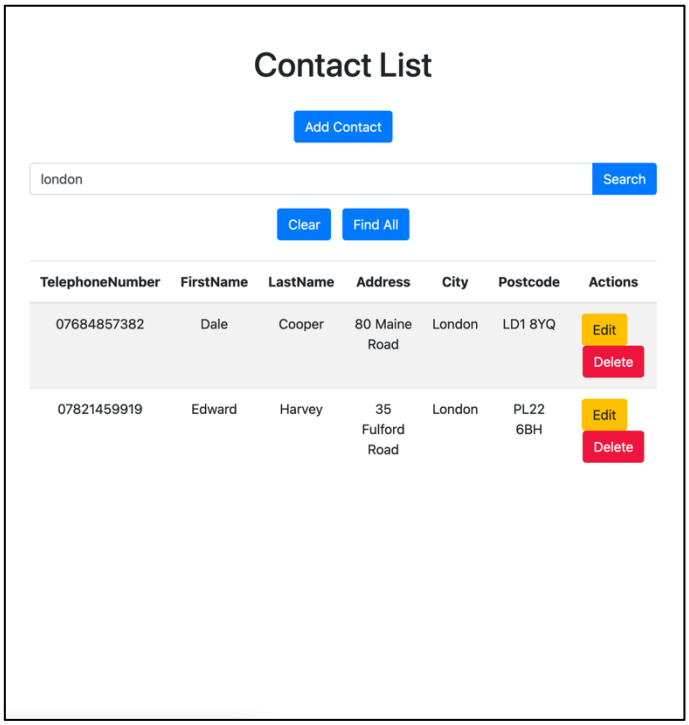

Figure 6.

Each contact retrieved from the ‘search’ or ‘findAll’ features has two separate buttons displayed under the Actions column. They allow users to delete or edit specific contacts. Deleting the contact removes the record from the database. Pressing the ‘edit’ button displays the second view in Figure 7. The view displays the current details of the contact selected which can now be changed and updated. The user is redirected to the view in Figure 4 when the ‘submit’ button is invoked.

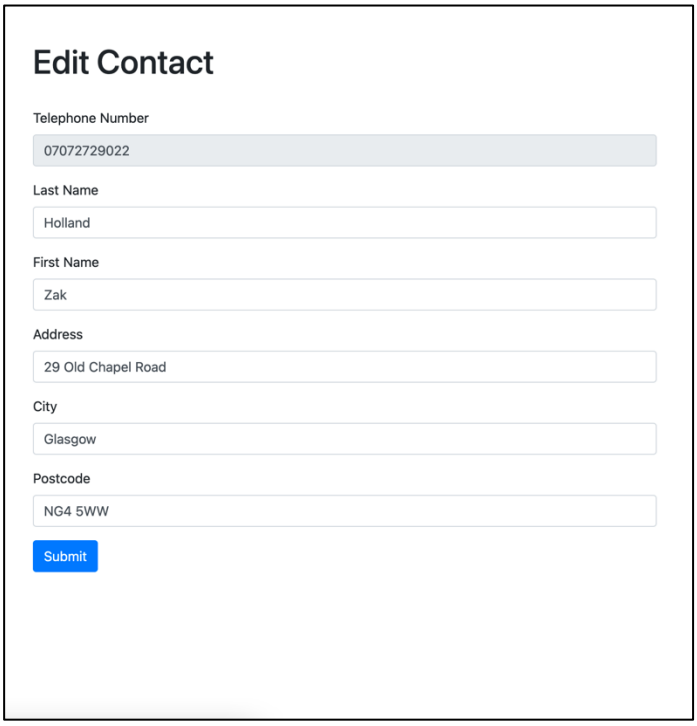

Figure 7.

The ‘add contact’ button in Figure 4, when clicked, displays the third view in Figure 8. This provides the functionality to add a new contact to the database. Once the details of the contact are entered the ‘submit’ button is pressed and the user is redirected back to the home view in Figure 4.

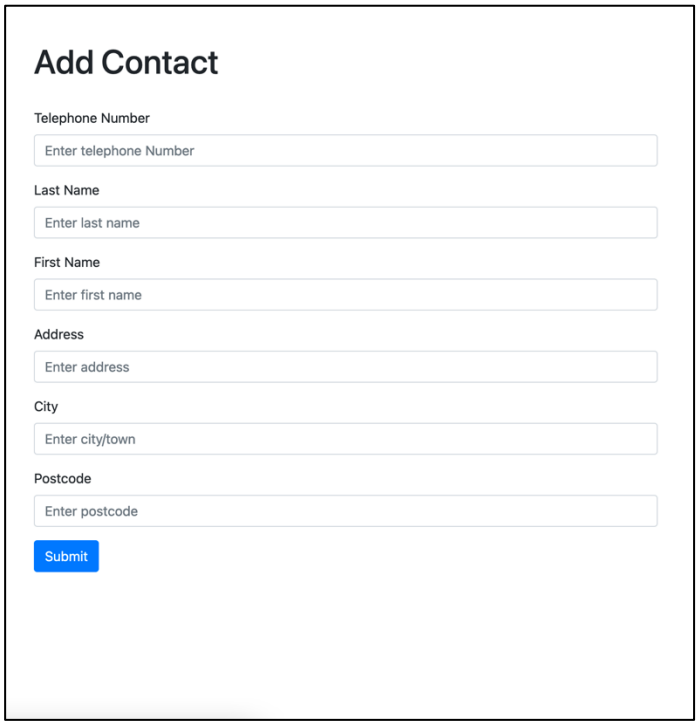

Figure 8.

Figure 9 shows how the web application is designed to be robust in handling error cases. Each field in the form must contain information in an acceptable format before submitting. If the field is blank or the entry is in an invalid format when the ‘submit’ button is pressed the following error messages are displayed in red. The form validation is present in both the ‘add contact’ and ‘edit contact’ views. The form input is validated against constraints defined in the contact entity class.

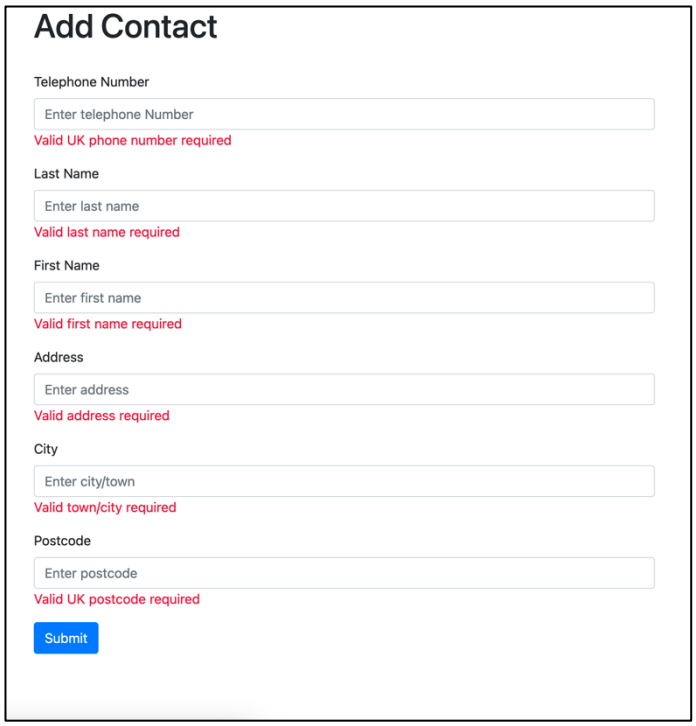

Figure 9.

2.4 Messages Exchanged

Wireshark is used to show the messages exchanged that relate to the CRUD operations of the web service. This section details the messages that were captured.

Retrieve all contacts:

```
  127.0.0.1          127.0.0.1          HTTP      470 GET /contacts?searchString= HTTP/1.1
  127.0.0.1          127.0.0.1          HTTP       61 HTTP/1.1 200   (text/html)
```

Retrieve all contacts in London:

```
127.0.0.1          127.0.0.1          HTTP      476 GET /contacts?searchString=london HTTP/1.1
127.0.0.1          127.0.0.1          HTTP       61 HTTP/1.1 200   (text/html)
```

Edit contact:

```
127.0.0.1          127.0.0.1          HTTP      492 GET /edit/07821459919 HTTP/1.1
127.0.0.1          127.0.0.1          HTTP       61 HTTP/1.1 200   (text/html)
127.0.0.1          127.0.0.1          HTTP      687 POST /update HTTP/1.1  (application/x-www-f
```

Add contact:

```
 127.0.0.1          127.0.0.1          HTTP      674 POST /add HTTP/1.1  (application/x-www-for
 127.0.0.1          127.0.0.1          HTTP      236 HTTP/1.1 302
```

Delete contact:

```
127.0.0.1          127.0.0.1          HTTP      488 GET /delete/07718675667 HTTP/1.1
127.0.0.1          127.0.0.1          HTTP      236 HTTP/1.1 302
```

3 Project Summary

This section provides a summary as to how complete the web service is and any cases where the program is not functioning correctly or could be improved.

3.1 Completed Implementation

The majority of the requirements outlined in the assignment specification have been implemented and function correctly within the web service. In addition to the basic CRUD operations, some of the advanced functions and error handling also feature in the web service solution. A complete list of the implemented features is detailed below.

- RESTful web service using Sprint Boot
- Contact data stored in a MySQL database
- Spring MVC controller with a record of requests mapped to exercise the service
- Client web application created through bootstrap and thymeleaf templates
- Service layer methods that can retrieve, add, edit and delete contacts
- Contact retrieval using search bar feature. This matches contacts that 'contain' the search string and therefore satisfies some of the advanced features outlined in the assignment specification. For example, the search feature can be used to retrieve all contacts in a specific town/city.
- The web service is also robust in handling error cases as validation checks are performed on the form input when adding and editing a contact.

3.2 Incomplete Implementation

The web service solution has no significant implementation processes that are incomplete. Despite this, with more time additional service methods and a more sophisticated client could be incorporated into the web service. Further testing could also be carried out to determine any special error cases that the current solution cannot handle.

4 Code Appendices

4.1 Appendix 1 – contacts.sql

```
CREATE TABLE `contactsDB`.`contact` (
  `telephoneNumber` VARCHAR(45) NOT NULL,
  `lastName` VARCHAR(45) NULL,
  `firstName` VARCHAR(45) NULL,
  `address` VARCHAR(45) NULL,
  `city` VARCHAR(45) NULL,
  `postcode` VARCHAR(45) NULL,
  PRIMARY KEY (`telephoneNumber`));
```

4.2 Appendix 2 - application.properties

```
# JPA settings
spring.jpa.hibernate.ddl-auto=update
spring.jpa.hibernate.naming.implicit-strategy=org.hibernate.boot.model.naming.ImplicitNamingStrategyLegacyJpaImpl
spring.jpa.hibernate.naming.physical-strategy=org.hibernate.boot.model.naming.PhysicalNamingStrategyStandardImpl

# The database connection URL
spring.datasource.url=jdbc:mysql://localhost:3306/contactsDB

# Username
spring.datasource.username=root

# Password
spring.datasource.password=password1&
```

4.3 Appendix 3 – pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.4.3</version>
        <relativePath /> <!-- lookup parent from repository -->
    </parent>

    <groupId>com.assignment</groupId>
    <artifactId>contacts</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>contacts</name>
    <description>Contacts Database Assignment</description>

    <properties>
        <java.version>11</java.version>
    </properties>
```

```xml
    <dependencies>

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-thymeleaf</artifactId>
        </dependency>

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-jpa</artifactId>
        </dependency>

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-rest</artifactId>
        </dependency>

        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <scope>runtime</scope>
        </dependency>

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>

        <dependency>
            <groupId>org.webjars</groupId>
            <artifactId>bootstrap</artifactId>
            <version>4.0.0-2</version>
        </dependency>

        <dependency>
            <groupId>org.webjars</groupId>
            <artifactId>webjars-locator</artifactId>
            <version>0.30</version>
        </dependency>

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-validation</artifactId>
        </dependency>

    </dependencies>
```

```xml
    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>

</project>
```

4.4 Appendix 4 – Contact.java

```java
package com.assignment.demo;

import javax.persistence.*;
import javax.validation.constraints.Pattern;

// specifies the class Contact is an entity and is mapped to a database table
@Entity

// specifies the name of the database table to be used for mapping
@Table(name = "contact")
public class Contact {

    // specifies the primary key of the entity
    @Id
    // specifies the column name
    @Column(name = "telephoneNumber")
    // used to validate regex pattern - phone number must be 11 consecutive digits
    @Pattern(regexp = "^\\d{11}$", message = "Valid UK phone number required")
    private String telephoneNumber;

    @Column(name = "lastName")
    // used to validate regex pattern - lastName must only contain alphabetic
    // characters
    @Pattern(regexp = "[A-Za-z]+", message = "Valid last name required")
    private String lastName;

    @Column(name = "firstName")
    // used to validate regex pattern - firstName must only contain alphabetic
    // characters
    @Pattern(regexp = "[A-Za-z]+", message = "Valid first name required")
    private String firstName;

    @Column(name = "address")
    // used to validate regex pattern - address can contain alphanumeric characters
    // and whitespace
    @Pattern(regexp = "[A-Za-z0-9 ]+", message = "Valid address required")
    private String address;

    @Column(name = "city")
    // used to validate regex pattern - city can contain alphanumeric characters and
    // whitespace
    @Pattern(regexp = "[A-Za-z ]+", message = "Valid town/city required")
    private String city;

    @Column(name = "postcode")
    // used to validate regex pattern - postcode must be in a correct UK format
    @Pattern(regexp = "[A-Za-z]{1,2}[0-9Rr][0-9A-Za-z]? [0-9][ABD-HJLNP-UW-Zabd-hjlnp-uw-z]{2}", message = "Valid UK postcode required")
    private String postcode;

    public Contact() {
    }

    /**
     * creates an object of the Contact class containing all the information
     * associated with a contact
     */
    public Contact(String telephoneNumber, String lastName, String firstName, String address, String city,
            String postcode) {
        this.telephoneNumber = telephoneNumber;
        this.lastName = lastName;
        this.firstName = firstName;
        this.address = address;
        this.city = city;
        this.postcode = postcode;
    }

    /**
     * @return the telephone number
     */
    public String getTelephoneNumber() {
        return telephoneNumber;
    }

    /**
     * @param set the telephone number
     */
    public void setTelephoneNumber(String telephoneNumber) {
        this.telephoneNumber = telephoneNumber;
    }

    /**
     * @return the last name of the contact
     */
    public String getLastName() {
        return lastName;
    }

    /**
     * @param set the last name
     */
    public void setLastName(String lastName) {
        this.lastName = lastName;
    }

    /**
     * @return the first name of the contact
     */
    public String getFirstName() {
        return firstName;
    }
```

```java
    /**
     * @param set the first name
     */
    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }

    /**
     * @return the current address
     */
    public String getAddress() {
        return address;
    }

    /**
     * @param set the address
     */
    public void setAddress(String address) {
        this.address = address;
    }

    /**
     * @return the current city/town
     */
    public String getCity() {
        return city;
    }

    /**
     * @param set the city/town
     */
    public void setCity(String city) {
        this.city = city;
    }

    /**
     * @return the current postcode
     */
    public String getPostcode() {
        return postcode;
    }

    /**
     * @param set the postcode
     */
    public void setPostcode(String postcode) {
        this.postcode = postcode;
    }

    /**
     * @return the string representation of the contact object
     */
    @Override
    public String toString() {
        return "Contact{" + "telephoneNumber='" + telephoneNumber + '\'' + ", lastName='" + lastName + '\''
                + ", firstName='" + firstName + +'\'' + ", address='" + address + '\'' + ", city='" + city + '\''
                + ", postcode='" + postcode + '\'' + '}';
    }
}
```

4.5 Appendix 5 – ContactApplication.java

```java
package com.assignment.demo;

import org.springframework.boot.SpringApplication;

/**
 * class is used to launch the Spring application from a java main method
 * helps with launching the Spring MVC used for the web service front end
 */
@SpringBootApplication
public class ContactApplication {

    public static void main(String[] args) {
        SpringApplication.run(ContactApplication.class, args);
    }
}
```

4.6 Appendix 6 – ContactRepository.java

```java
package com.assignment.demo;

import org.springframework.data.jpa.repository.Query;

/**
 * repository interface is a mechanism for encapsulating storage, retrieval, and
 * search behaviour repository emulates a collection of objects
 */
@Repository
public interface ContactRepository extends CrudRepository<Contact, String> {

    // creates a database query that selects Contacts with similar entries in any
    // column to that of the searchString
    @Query("SELECT p FROM Contact p WHERE CONCAT(p.telephoneNumber, p.lastName, p.firstName, p.address, p.city, p.postcode) LIKE %?1%")
    public List<Contact> search(String searchString);

    // returns a list of all Contacts within the database
    public List<Contact> findAll();

}
```

4.7 Appendix 7 – ContactService.java

```java
package com.assignment.demo;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

// defines the business logic
@Service
public class ContactService {

    // injects an instance of ContactRepository
    @Autowired
    ContactRepository repository;

    /**
     * @return the contact records that match the searchString
     *  using the search() method of CrudRepository if searchString is not null
     *
     * @return all the contact records using the findAll() method of CrudRepository
     *  if the searchString is null
     */
    public List<Contact> getSearch(String searchString) {
        if (searchString != null) {
            return repository.search(searchString);
        }
        return repository.findAll();
    }

    /**
     * saving a contact using the save() method of CrudRepository
     */
    public void createContact(Contact contact) {
        repository.save(contact);
    }

    /**
     * getting a specific contact using the findById method of CrudRepository
     */
    public Contact getContact(String telephoneNumber) {
        return repository.findById(telephoneNumber).get();
    }

    /**
     * deleting a specific contact using deleteById() method of CrudRepository
     */
    public void deleteContact(String telephoneNumber) {
        repository.deleteById(telephoneNumber);
    }
}
```

4.8 Appendix 8 – ContactController.java

```java
package com.assignment.demo;

import org.springframework.beans.factory.annotation.Autowired;

// defines the class as a Spring MVC controller
@RestController
// indicates all handling methods are relative to the '/' path
@RequestMapping("/")
public class ContactController {

    // injects an instance of ContactService
    @Autowired
    private ContactService service;

    /**
     * creates a mapping that retrieves the list of contacts similar to the searchString
     * @Return the index view
     */
    @RequestMapping("/contacts")
    public ModelAndView findContacts(Model model, @Param("searchString") String searchString) {
        List<Contact> listContacts = service.getSearch(searchString);
        model.addAttribute("listContacts", listContacts);
        model.addAttribute("searchString", searchString);

        return new ModelAndView("index");
    }

    /**
     * creates a mapping that adds a new contact
     * @Return the add contact view
     */
    @RequestMapping("/new")
    public ModelAndView addContact(Model model) {
        Contact contact = new Contact();
        model.addAttribute("contact", contact);

        return new ModelAndView("add_contact");
    }
```

```java
     * creates a mapping that edits a contact specified with its telephone number
     * @Return the edit contact view
     */
    @RequestMapping("/edit/{telephoneNumber}")
    public ModelAndView editContact(@PathVariable(name = "telephoneNumber") String telephoneNumber) {
        ModelAndView modv = new ModelAndView("edit_contact");
        Contact contact = service.getContact(telephoneNumber);
        modv.addObject("contact", contact);

        return modv;
    }

    /**
     * creates a mapping that deletes a contact specified with its telephone number
     * @return a redirection to the '/' mapping
     */
    @RequestMapping("/delete/{telephoneNumber}")
    public ModelAndView deleteContact(@PathVariable(name = "telephoneNumber") String telephoneNumber) {
        service.deleteContact(telephoneNumber);

        return new ModelAndView("redirect:/");
    }

    /**
     * @return the add contact view if validation errors are present
     * @return a redirection to the '/' mapping otherwise
     */
    @RequestMapping("/add")
    public ModelAndView saveNewContact(@Valid @ModelAttribute("contact") Contact contact, BindingResult bindRes) {

        if (bindRes.hasErrors()) {
            return new ModelAndView("add_contact");
        }

        service.createContact(contact);

        return new ModelAndView("redirect:/");
    }

    /**
     * @return the edit contact view if validation errors are present
     * @return a redirection to the '/' mapping otherwise
     */
    @RequestMapping("/update")
    public ModelAndView saveUpdateContact(@Valid @ModelAttribute("contact") Contact contact, BindingResult bindRes) {

        if (bindRes.hasErrors()) {
            return new ModelAndView("edit_contact");
        }

        service.createContact(contact);

        return new ModelAndView("redirect:/");
    }
```

4.9 Appendix 9 – index.html

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml" xmlns:th="http://www.thymeleaf.org">

  <head>
    <meta charset="utf-8" />
    <title>Contacts Database</title>
    <link rel="stylesheet" th:href="@{/webjars/bootstrap/css/bootstrap.min.css}">
  </head>

  <body>
    <div align="center" class="m-5 pb-5">
      <h1>Contact List</h1>
      <br />
      <a href="/new" class="btn btn-primary">Add Contact</a>
      <br /><br />

      <form th:action="@{/contacts}">

        <div class="input-group mb-3">
          <input type="text" class="form-control" placeholder="Search Contacts" aria-label="Search Contacts"
            aria-describedby="basic-addon2" name="searchString" id="searchString" th:value="${searchString}" />
          <div class="input-group-append">
            <input type="submit" value="Search" class="btn btn-primary" />
          </div>
        </div>

        <input type="button" value="Clear" id="btnClear" onclick="clearSearch()" class="btn btn-primary" />
         
        <input type="submit" value="Find All" class="btn btn-primary" />
      </form>

      <br />
      <table class="table table-striped text-center">
        <thead>
          <tr>
            <th>TelephoneNumber</th>
            <th>FirstName</th>
            <th>LastName</th>
            <th>Address</th>
            <th>City</th>
            <th>Postcode</th>
            <th>Actions</th>
          </tr>
        </thead>
      <tbody>
        <tr th:each="contact : ${listContacts}">
          <td th:text="${contact.telephoneNumber}">TelephoneNumber</td>
          <td th:text="${contact.firstName}">FirstName</td>
          <td th:text="${contact.lastName}">LastName</td>
          <td th:text="${contact.Address}">Address</td>
          <td th:text="${contact.City}">City</td>
          <td th:text="${contact.Postcode}">Postcode</td>
          <td>
            <a th:href="@{'/edit/' + ${contact.telephoneNumber}}" class="btn btn-warning">Edit</a>
                
            <a th:href="@{'/delete/' + ${contact.telephoneNumber}}" class="btn btn-danger">Delete</a>
          </td>
        </tr>
      </tbody>
    </table>
  </div>

  <script th:src="@{/webjars/jquery/jquery.min.js}"></script>
  <script th:src="@{/webjars/popper.js/umd/popper.min.js}"></script>
  <script th:src="@{/webjars/bootstrap/js/bootstrap.min.js}"></script>

  <script type="text/javascript">
    function clearSearch() {
      window.location = "[[@{/}]]";
    }
  </script>
</body>

</html>
```

4.10 Appendix 11 – edit_contact.html

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml" xmlns:th="http://www.thymeleaf.org">

<head>
  <meta charset="utf-8" />
  <link rel="stylesheet" th:href="@{/webjars/bootstrap/css/bootstrap.min.css}">
  <title>Update Contact</title>
</head>

<body>
  <div class="m-5 pb-5">
    <h1>Edit Contact</h1>
    <br />
    <form action="#" th:action="@{/update}" th:object="${contact}" method="post">

      <div class="form-group">
        <label for="teleNum">Telephone Number</label>
        <input type="text" class="form-control" th:field="*{telephoneNumber}" id="teleNum" placeholder="Enter telephone Number" readonly="true" />
      </div>
      <div class="form-group">
        <label for="lname">Last Name</label>
        <input type="text" class="form-control" th:field="*{lastName}" id="lname" placeholder="Enter last name" />
        <span class="help-block" th:if="${#fields.hasErrors('lastName')}">
            <p th:errors="*{lastName}" class="text-danger">Last name error</p>
          </span>
        </div>
        <div class="form-group">
          <label for="fname">First Name</label>
          <input type="text" class="form-control" th:field="*{firstName}" id="fname" placeholder="Enter first name" />
          <span class="help-block" th:if="${#fields.hasErrors('firstName')}">
            <p th:errors="*{firstName}" class="text-danger">First name error</p>
          </span>
        </div>
        <div class="form-group">
          <label for="add">Address</label>
          <input type="text" class="form-control" th:field="*{address}" id="add" placeholder="Enter address" />
          <span class="help-block" th:if="${#fields.hasErrors('address')}">
            <p th:errors="*{address}" class="text-danger">Address error</p>
          </span>
        </div>
        <div class="form-group">
          <label for="cit">City</label>
          <input type="text" class="form-control" th:field="*{city}" id="cit" placeholder="Enter city/town" />
          <span class="help-block" th:if="${#fields.hasErrors('city')}">
            <p th:errors="*{city}" class="text-danger">City error</p>
          </span>
        </div>
        <div class="form-group">
          <label for="post">Postcode</label>
          <input type="text" class="form-control" th:field="*{postcode}" id="post" placeholder="Enter postcode" />
          <span class="help-block" th:if="${#fields.hasErrors('postcode')}">
            <p th:errors="*{postcode}" class="text-danger">Postcode error</p>
          </span>
        </div>

        <button type="submit" class="btn btn-primary">Submit</button>
      </form>

    </div>

    <script th:src="@{/webjars/jquery/jquery.min.js}"></script>
    <script th:src="@{/webjars/popper.js/umd/popper.min.js}"></script>
    <script th:src="@{/webjars/bootstrap/js/bootstrap.min.js}"></script>
  </body>

</html>
```

4.11 Appendix 10 – add_contact.html

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml" xmlns:th="http://www.thymeleaf.org">

<head>
  <meta charset="utf-8" />
  <link rel="stylesheet" th:href="@{/webjars/bootstrap/css/bootstrap.min.css}">
  <title>Add New Contact</title>
</head>

<body>
  <div class="m-5 pb-5">
    <h1>Add Contact</h1>
    <br />
    <form action="#" th:action="@{/add}" th:object="${contact}" method="post">

      <div class="form-group">
        <label for="teleNum">Telephone Number</label>
        <input type="text" class="form-control" th:field="*{telephoneNumber}" id="teleNum" placeholder="Enter telephone Number" />
        <span class="help-block" th:if="${#fields.hasErrors('telephoneNumber')}">
            <p th:errors="*{telephoneNumber}" class="text-danger">Telephone number error</p>
          </span>
        </div>
        <div class="form-group">
          <label for="lname">Last Name</label>
          <input type="text" class="form-control" th:field="*{lastName}" id="lname" placeholder="Enter last name" />
          <span class="help-block" th:if="${#fields.hasErrors('lastName')}">
            <p th:errors="*{lastName}" class="text-danger">Last name error</p>
          </span>
        </div>
        <div class="form-group">
          <label for="fname">First Name</label>
          <input type="text" class="form-control" th:field="*{firstName}" id="fname" placeholder="Enter first name" />
          <span class="help-block" th:if="${#fields.hasErrors('firstName')}">
            <p th:errors="*{firstName}" class="text-danger">First name error</p>
          </span>
        </div>
        <div class="form-group">
          <label for="add">Address</label>
          <input type="text" class="form-control" th:field="*{address}" id="add" placeholder="Enter address" />
          <span class="help-block" th:if="${#fields.hasErrors('address')}">
            <p th:errors="*{address}" class="text-danger">Address error</p>
          </span>
        </div>
        <div class="form-group">
          <label for="cit">City</label>
          <input type="text" class="form-control" th:field="*{city}" id="cit" placeholder="Enter city/town" />
          <span class="help-block" th:if="${#fields.hasErrors('city')}">
            <p th:errors="*{city}" class="text-danger">City error</p>
          </span>
        </div>
        <div class="form-group">
          <label for="post">Postcode</label>
          <input type="text" class="form-control" th:field="*{postcode}" id="post" placeholder="Enter postcode" />
          <span class="help-block" th:if="${#fields.hasErrors('postcode')}">
            <p th:errors="*{postcode}" class="text-danger">Postcode error</p>
          </span>
        </div>

        <button type="submit" class="btn btn-primary">Submit</button>
      </form>

    </div>

    <script th:src="@{/webjars/jquery/jquery.min.js}"></script>
    <script th:src="@{/webjars/popper.js/umd/popper.min.js}"></script>
    <script th:src="@{/webjars/bootstrap/js/bootstrap.min.js}"></script>
  </body>

</html>
```