



# CSCU9YW – WEB SERVICES ASSIGNMENT

Student ID Number – 263615  
University of Stirling

## Contents

1. Introduction .....	3
2. Outline of the Problem .....	4
2.1. Description .....	4
2.2. Assumptions.....	4
2.2.1. Assumption One.....	4
2.2.2. Assumption Two .....	4
2.2.3. Assumption Three .....	4
3. Solution .....	5
3.1. Implementation .....	5
3.2. Graphical User Interface .....	5
3.2.1. Main Menu.....	5
3.2.1.1. Error Handling.....	6
3.2.2. Single Contact Display .....	8
3.2.3. New Contact Display .....	9
3.2.3.1. Contact Added .....	10
3.2.3.2. Error Handling.....	11
3.2.4. Edit Contact Display .....	13
3.2.4.1. Contact Edited.....	14
3.2.4.2. Error Handling.....	15
3.2.5. Delete Contact .....	16
3.3. Web Service Requests.....	17
3.3.1. GET .....	17
3.3.1.1. Sending out a GET Request.....	17
3.3.1.2. Recieveing a GET Response.....	18
3.3.2. PUT .....	19
3.3.2.1. Sending out a PUT Request.....	19
3.3.1.2. Retrieving a PUT Response .....	20
3.3.3. POST .....	21
3.3.3.1. Sending out a POST Request.....	21
3.3.3.2. Retrieving a POST Response .....	22
3.3.4. DELETE.....	23
3.3.4.1. Sending out a DELETE Request .....	23
3.3.4.2. Retrieving a DELETE Response .....	24
3.3.2. Error Response .....	25
4. Key Coding Constructs .....	26

4.1. Web Service and Database Connections .....	26
4.2. Graphical User Interface .....	26
5. Project Completion .....	27
5.1. Completed Implementation.....	27
5.2. Incomplete Implementation .....	27
5.3. Alternative Approach and Additional Functionality .....	27
6. Appendices.....	28
6.1. Appendix One: Main.java.....	28
6.2. Appendix Two: Contact.java .....	29
6.3. Appendix Three: ContactsController.java.....	32
6.4. Appendix Four: ContactService.java .....	36
6.5. Appendix Five: Requests.java .....	39
6.6. Appendix Six: ContactRepository.java .....	46
6.7. Appendix Seven: MainMenuUserinterface.java .....	47
6.8. Appendix Eight: NewContactUserInterface.java .....	57
6.9. Appendix Nine: EditContactUserInterface.java .....	65
6.10. Appendix Nine: SingleContactUserInterface.java.....	71
6.11. Application.properties .....	75

## 1. Introduction

The task of the of the assignment is to create a web service that has to be developed to act as a contacts database, this could be implemented using various technologies that have been previously used throughout the practical exercises. These being either OAP, REST based on javax, REST based on Jakarta EE, or REST based on Spring io. If the REST services were to be used, URI parameters, XML or JSON formatting could be used for transferring the data to and from the web service. Using SOAP requests, RPC documentation or Document literal ending could be used. For the SOAP and REST solutions based on a javax and client should be provided alongside, this being a simple user interface to allow client to interact with the service.

The contact database service that is required must be able to store people's details into a java structure, this being a HashMap or an external database. Each contact that is stored must include their name, address (street, town, postcode) and telephone number. The telephone number will be used as the unique identifier for the contact. The web service side of the database requires methods that retrieve, add, edit and delete contacts. Advanced features such as interacting with a group of contacts and adding how many edits have been carried out could be added to build on the functionality. Also, implementing error handling such as preventing a duplicate contact being added or deleting a contact that does not exist could be tried.

## 2. Outline of the Problem

### 2.1. Description

The aim for the assignment was to create and develop a contact database service that would take in details of people and store them as contacts. The details that are required to be taken in include the persons first name, surname, address (street, town/cit. postcode) and telephone number, with the assumption that the telephone number will be unique to identify a contact.

The web service requirements include methods the retrieve, add, edit, and delete the contact along with advanced features which include interacting with a group of clients such as a group of contacts with the same information such as town or how many edits have been carried out on a contact etc. The webservice may also provide a robust take on potential error handling cases, an example of this being trying to add a contact that already exists or deleting a user that does not exist. Some extra credit is available for such of these features.

### 2.2. Assumptions

These are the assumptions that I made throughout the development process of the application.

#### 2.2.1. Assumption One

The user will be starting with an empty database so they will need to add contacts from scratch, there will need to be a way to see what is currently stored in the data base.

#### 2.2.2. Assumption Two

The user will need to know the phone number of a contact in order to find them through the use of a search feature. All contacts could be displayed to see if there is anything present in the database so the user can see the contact they are looking for.

#### 2.2.3. Assumption Three

The user interface will need to be simple and easy to understand in order to apply to any user that uses the application, anyone who might not use computers a lot will still be able to work the application the same as someone who has a better understanding of using computers.

### 3. Solution

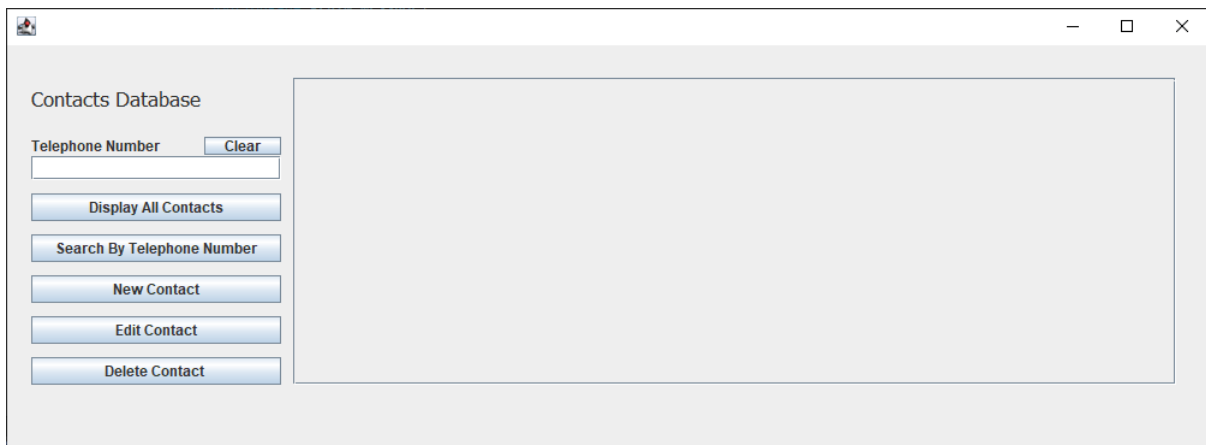
#### 3.1. Implementation

For the implementation of the project, I decided to use the lab five practical which uses Spring, I decided to use this as I felt as though I have a better understanding of how RESTful web services work as I have previously used them in the past for other projects. I was able to implement all the required features that were asked and stated in the brief for the project (can be found in section 1. Introduction), along with a few extra advanced features such as a graphical user interface, additional display features and error handling for special cases.

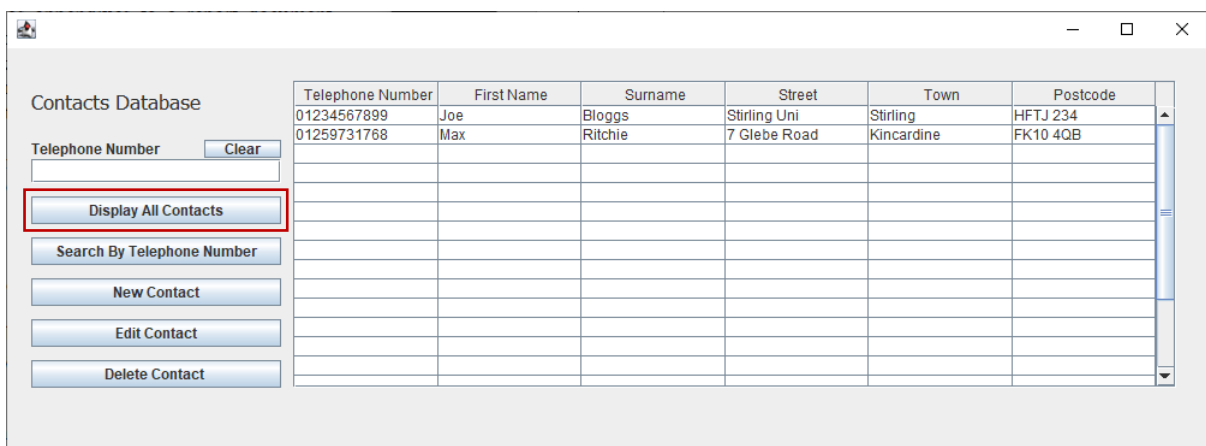
#### 3.2. Graphical User Interface

##### 3.2.1. Main Menu

This is what the user will see when the application is started, it displays all the relevant buttons for the functionality of the application along with a scroll pane that holds a table, it only displays an empty box until the user decides to display all contacts (**Figure 1.**). When the display all contacts button has been clicked, all contacts that are currently being held in the database are displayed as a table view (**Figure 2.**). The clear button will clear any input from the telephone number text field when clicked, a shortcut for clearing the text field.



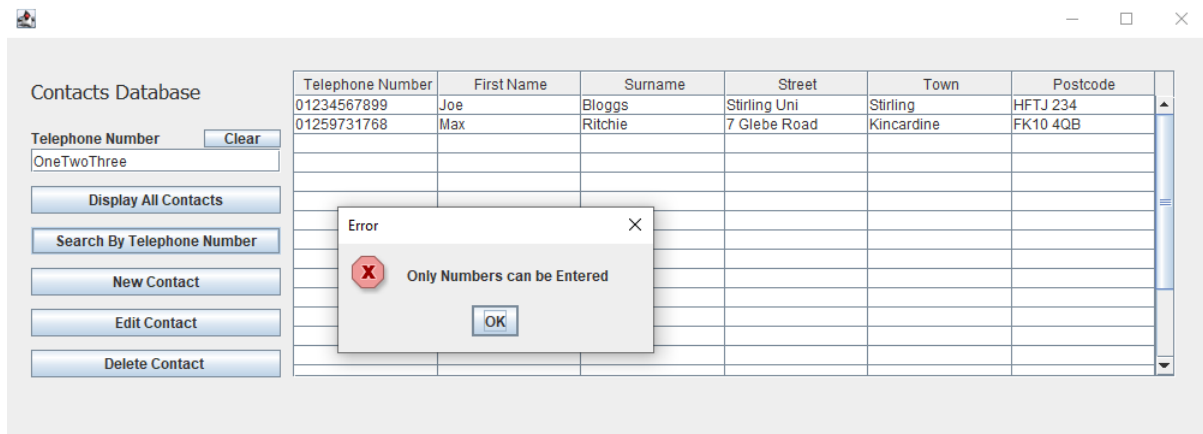
**Figure 1. Main Menu**



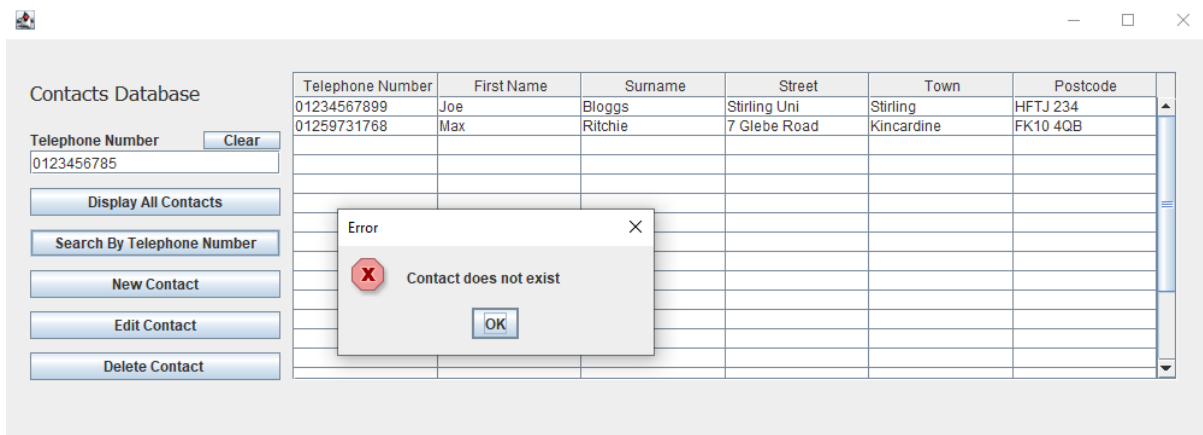
**Figure 2. Main Menu with Contacts Displayed**

#### 3.2.1.1. Error Handling

The only thing that needed error handling on the main menu of the application is the input of a telephone number in the text field. When either the search by telephone, edit contact or delete contact button is clicked, it takes the input that has been entered in the text field. It checks the input for only numbers, if anything that is not a number is searched, an error message stating this will be displayed to the screen (**Figure 3.**). Is a telephone number that is currently not being held in the database is search, the relevant error message will be displayed stating that the contact does not exist (**Figure 4.**). Is there is not input in the text field when one of the buttons are clicked, an error message stating that nothing has been entered to search for will be displayed to the screen (**Figure 5.**).

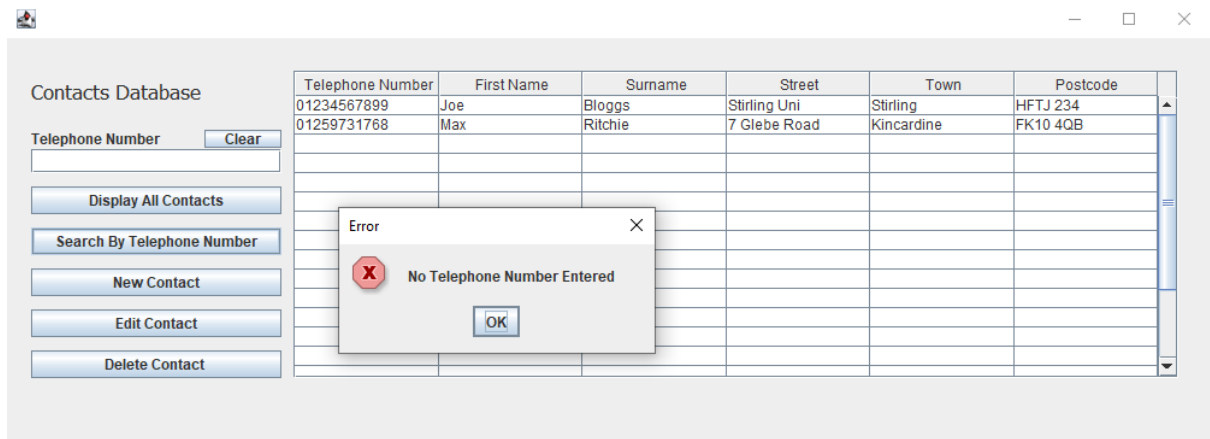


**Figure 3. String Input Error Message**



**Figure 4. Non-Existing Telephone Number Error Message**

Student ID Number  
263157

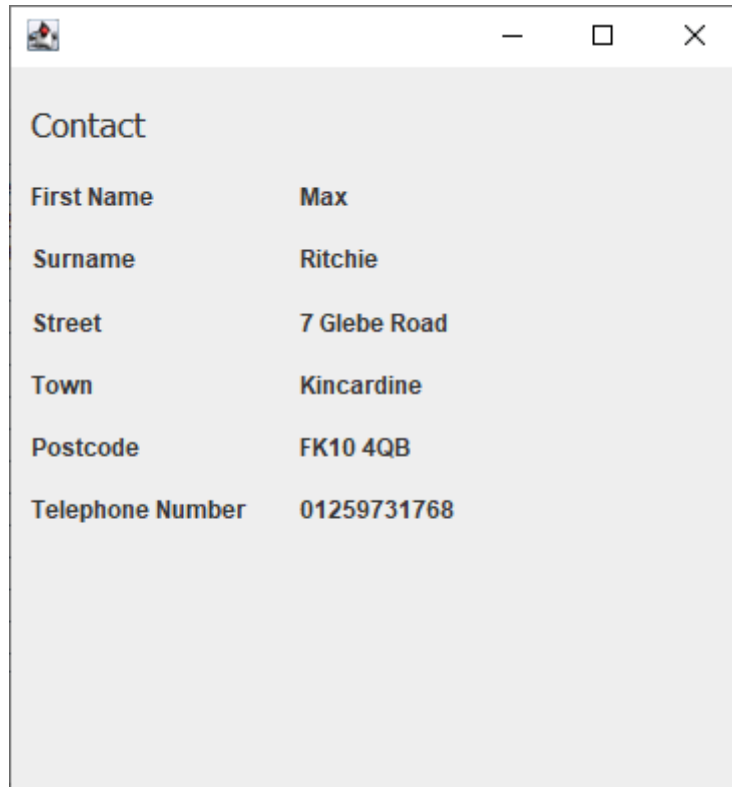


**Figure 5. No Input Error Message**



### 3.2.2. Single Contact Display

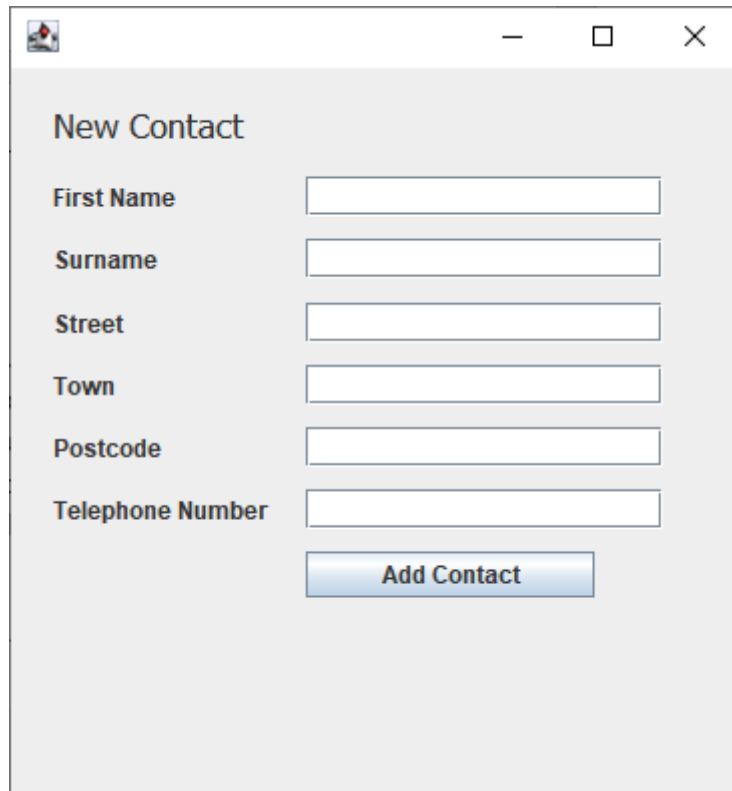
When a valid telephone number is entered and the search by telephone number button is clicked from the main menu a new display is shown holding all the information about the contact that has been searched for (**Figure 6.**). If the user wants to return to the main menu all they need to do is click the close button (x) at the top left of the screen to close and dispose of the view. The process of how the application finds a contact to be displayed is shown and explained in section 3.3.1. GET.



**Figure 6. Single Contact Display**

### 3.2.3. New Contact Display

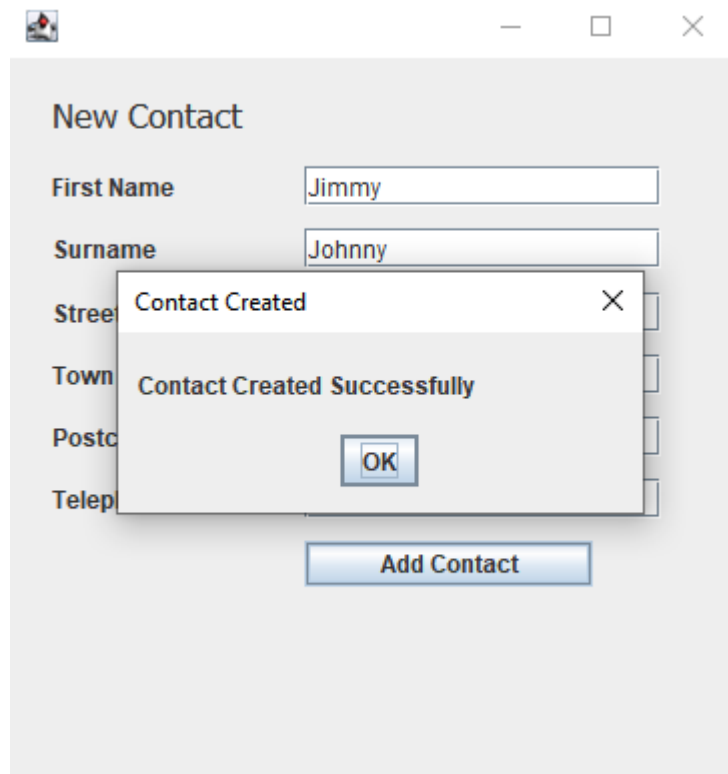
When the user clicks the create new contact button from the main menu screen, this screen is displayed to the screen (**Figure 7.**). The empty text fields that are displayed is where the user can enter in the new contacts details.

A screenshot of a software window titled "New Contact". The window has a standard title bar with a small icon on the left and minimize, maximize, and close buttons on the right. The main content area has a light gray background. At the top, the title "New Contact" is displayed in a bold, dark font. Below the title, there are six text input fields arranged vertically, each preceded by a label: "First Name", "Surname", "Street", "Town", "Postcode", and "Telephone Number". All input fields are empty. At the bottom of the form, there is a blue button with the text "Add Contact" in white.

***Figure 7. Create New Contact Display***

### 3.2.3.1. Contact Added

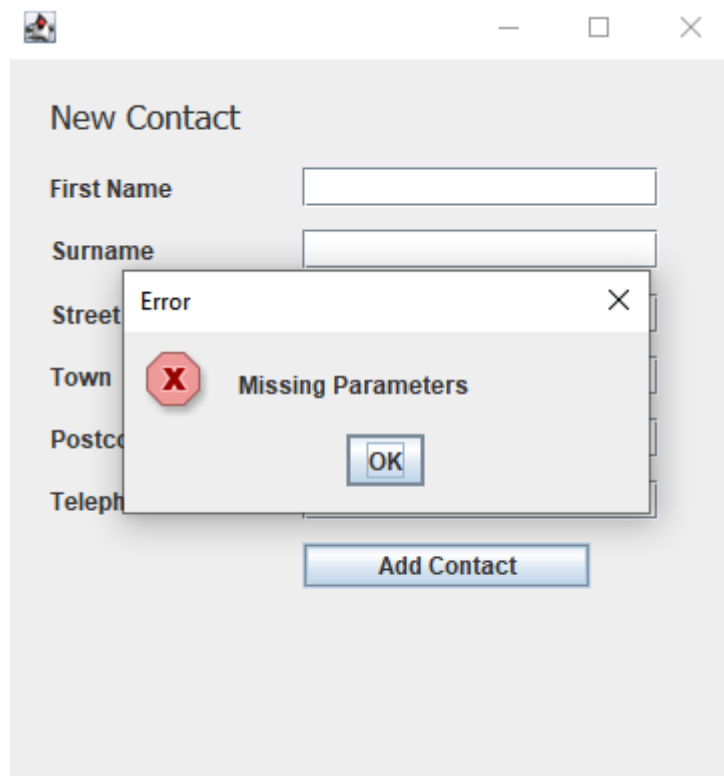
When the user has entered valid contact information and clicked the add contact button, a GET request is carried out to check if the telephone number that has been entered is already being used or not (more information about the GET requests is found in section 3.3.1. GET). If not, the contact is created and added to the database, a message also gets displayed to the screen stating that the contact has been successfully created (**Figure 7.**), the display is then closed and the user is returned back to the main menu display.



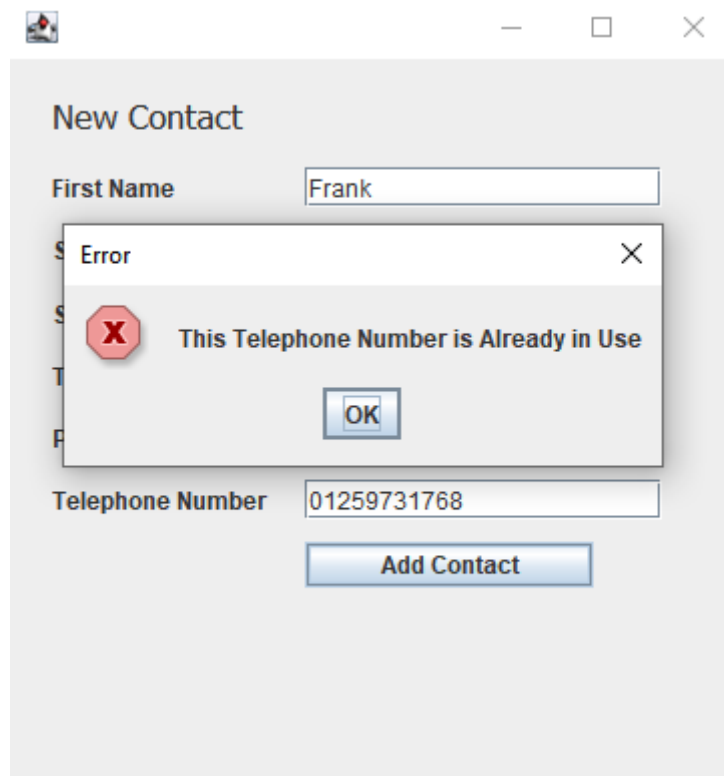
**Figure 8. Contact Successfully Added Display**

### 3.2.3.2. Error Handling

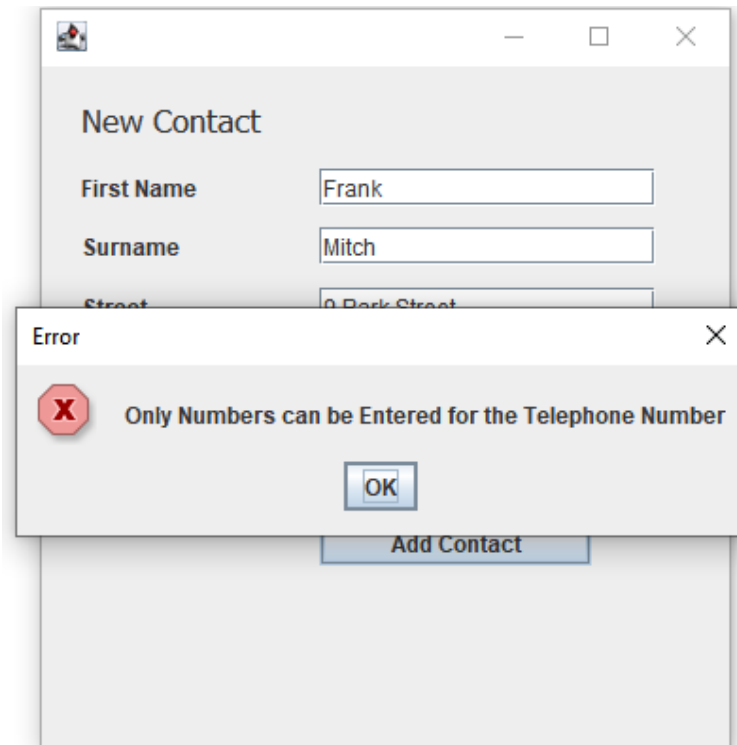
The error handling that gets carried out for creating a new contact is based mostly based around the telephone number that the user enters, it checks if the telephone number is already being used to prevent any duplicate values. It does this by sending out a GET request to check if the telephone number is already being stored in the database, it will send back a failed response if not found, more information about this can be found in section 3.3.2. Error Response. An error message gets displayed to the screen stating this, this can be seen in **Figure 10**. If the user were to input non number values into the telephone, the error message for wrong input is displayed (**Figure 11.**). Also, if the text fields were to be left empty and the add contact button is clicked, an error message is displayed to the screen stating that there are missing values (**Figure 9**).



**Figure 9. Missing Parameters Error Pop Up**



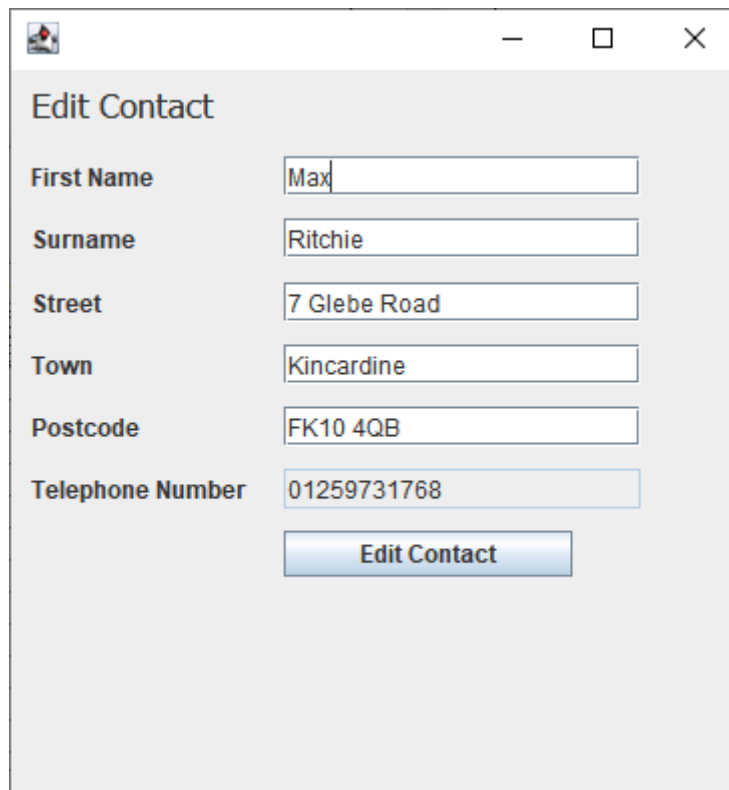
**Figure 10. Telephone Number Already in Use Error Pop Up**



**Figure 11. Invalid Values Entered Error Pop Up**

#### 3.2.4. Edit Contact Display

When the user has entered a valid phone number into the telephone number text field on the main menu and the edit contacts button gets clicked, a GET request get send to get the contact that has been requested to be search. More information can be found in section [3.3.1. GET](#). After this has been done, the edit contact display is shown on the screen with the searched contacts details displayed om text fields, this is done so the user can easily edit the contacts information. The only thing that can not be edited is the contacts telephone number as this is used to identify the contact.



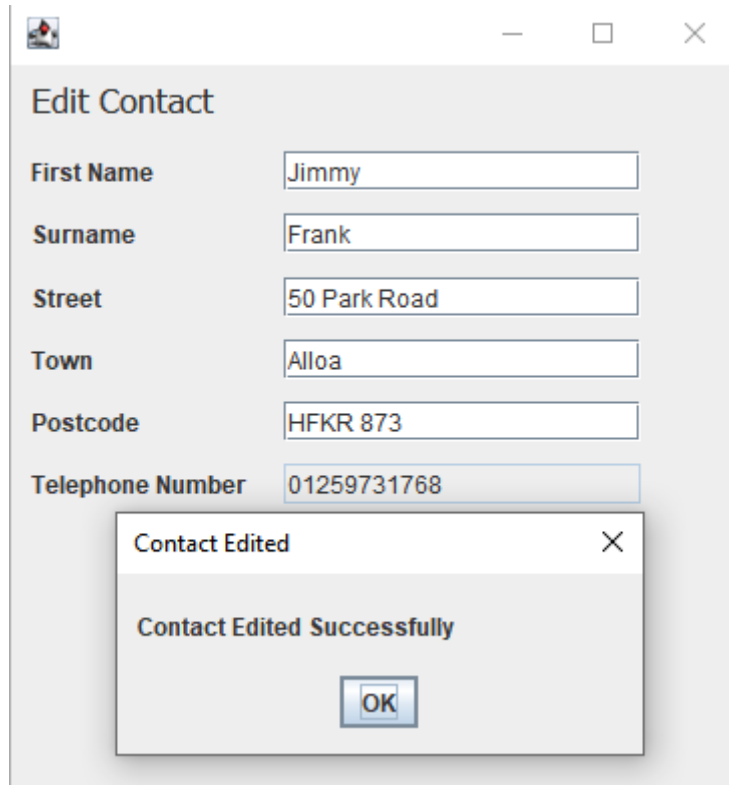
The screenshot shows a web application window titled "Edit Contact". It contains several text input fields for contact information. The fields are labeled "First Name", "Surname", "Street", "Town", "Postcode", and "Telephone Number". The values entered in these fields are "Max", "Ritchie", "7 Glebe Road", "Kincardine", "FK10 4QB", and "01259731768" respectively. Below the fields is a blue button labeled "Edit Contact".

Field	Value
First Name	Max
Surname	Ritchie
Street	7 Glebe Road
Town	Kincardine
Postcode	FK10 4QB
Telephone Number	01259731768

**Figure 12. Edit Contact Display**

#### 3.2.4.1. Contact Edited

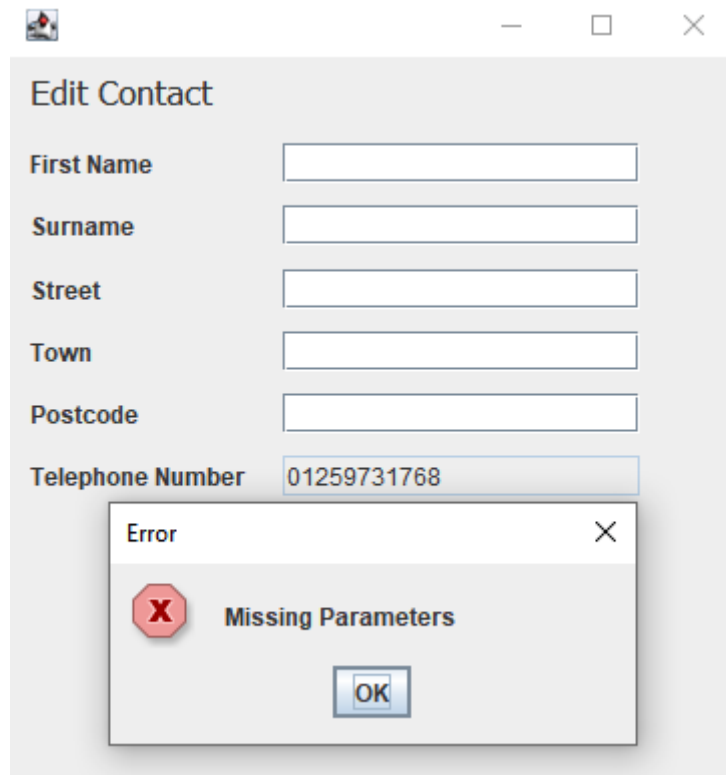
When the edit contact button is clicked with valid data present text fields, a PUT request is sent out with the new contact data that is to be used to update the contact, more information about how the PUT requests work are found in section 3.3.2. PUT. After the request has been made and the contact has been updated, a success message is displayed to the screen and the edit contacts page gets closed. The user then gets returned to the main menu screen. (**Figure 13.**)



**Figure 13. Contact Successfully Edited Display**

#### 3.2.4.2. Error Handling

If the text fields are left empty when the user clicks the edit button, an error message will be displayed to the screen stating that there are missing parameters. This is shown in **Figure 14**.

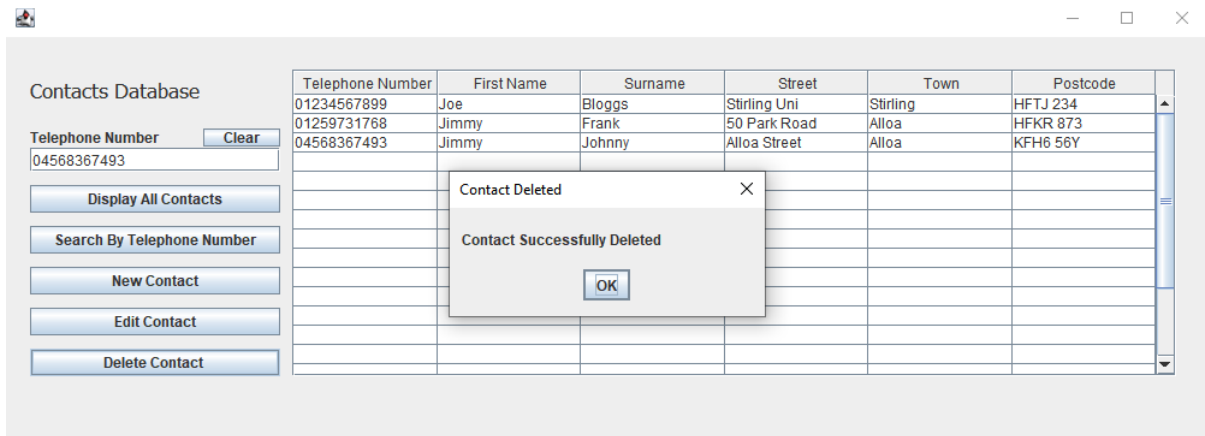


**Figure 14. Missing Parameters Error Display**



### 3.2.5. Delete Contact

When the user enters in a valid telephone number and clicks the delete button, a DELETE request is sent out to delete the contact from the database, more information about the DELETE requests is shown in section [3.3.4. DELETE](#). When the contact has been deleted, a success message gets displayed to the screen stating that the deletion of the contact has been successful. This can be seen in the screenshot in **Figure 15**. The error handling for this has already been covered in section [3.2.1.1. Error Handling](#).



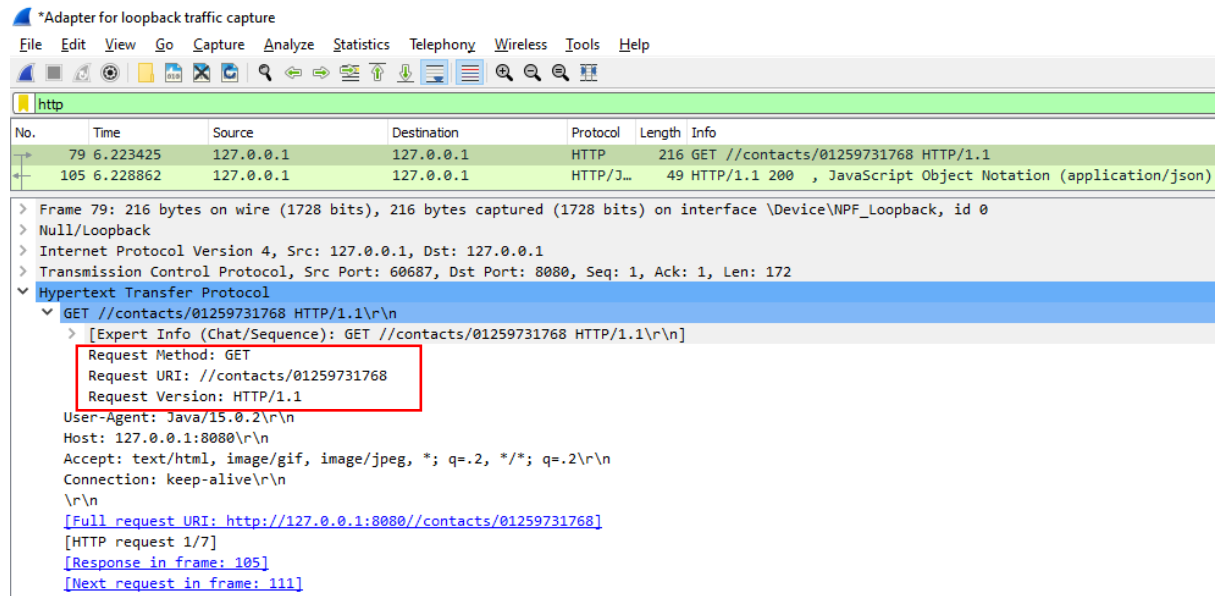
**Figure 15. Successful Deletion of a Contact.**

### 3.3. Web Service Requests

#### 3.3.1. GET

##### 3.3.1.1. Sending out a GET Request

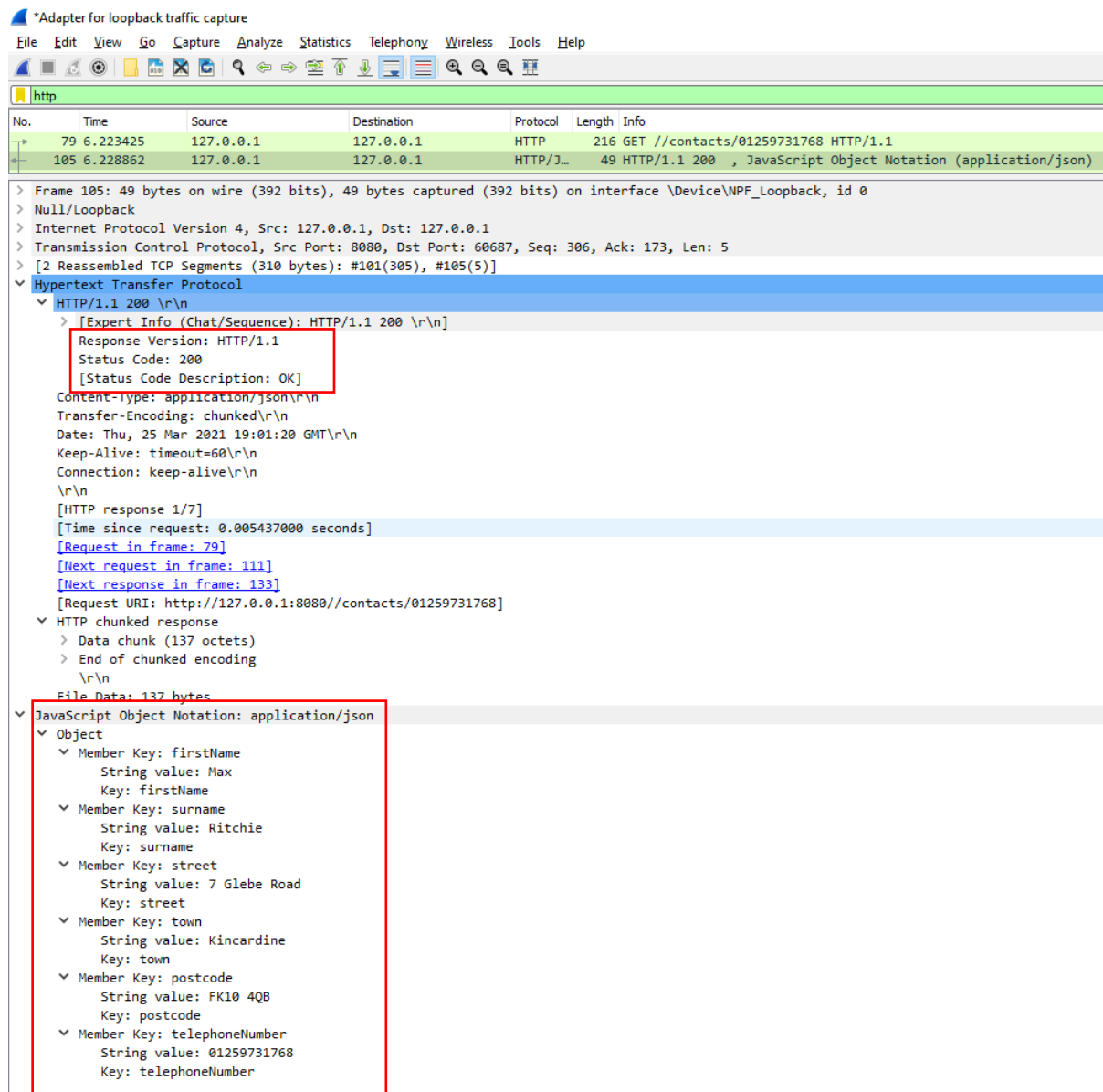
When a get request is sent from the application, it uses the telephone number that has been entered by the user to find that specific contact. This can be seen with the wire shark application in **Figure 16.**, where highlighted, you can see what the method type that has been requested is (GET) along with the URI that has been created and used for the request. This is where the users input of a telephone number is used to search for the contact.



**Figure 16. Sending out a GET Request**

### 3.3.1.2. Recieveing a GET Response

After sending out the GET request to the we service, a response is produced which returns the actions of what the GET carries out if the process is successful. As you can see in **Figure 17.**, in the first highlighted box, it shows the status code of the request which is 200 meaning that the GET request has been successful and has been carried out. The second highlighted area is the JSON object that has been returned and inside this object contains a contacts data that has been pulled from the database.

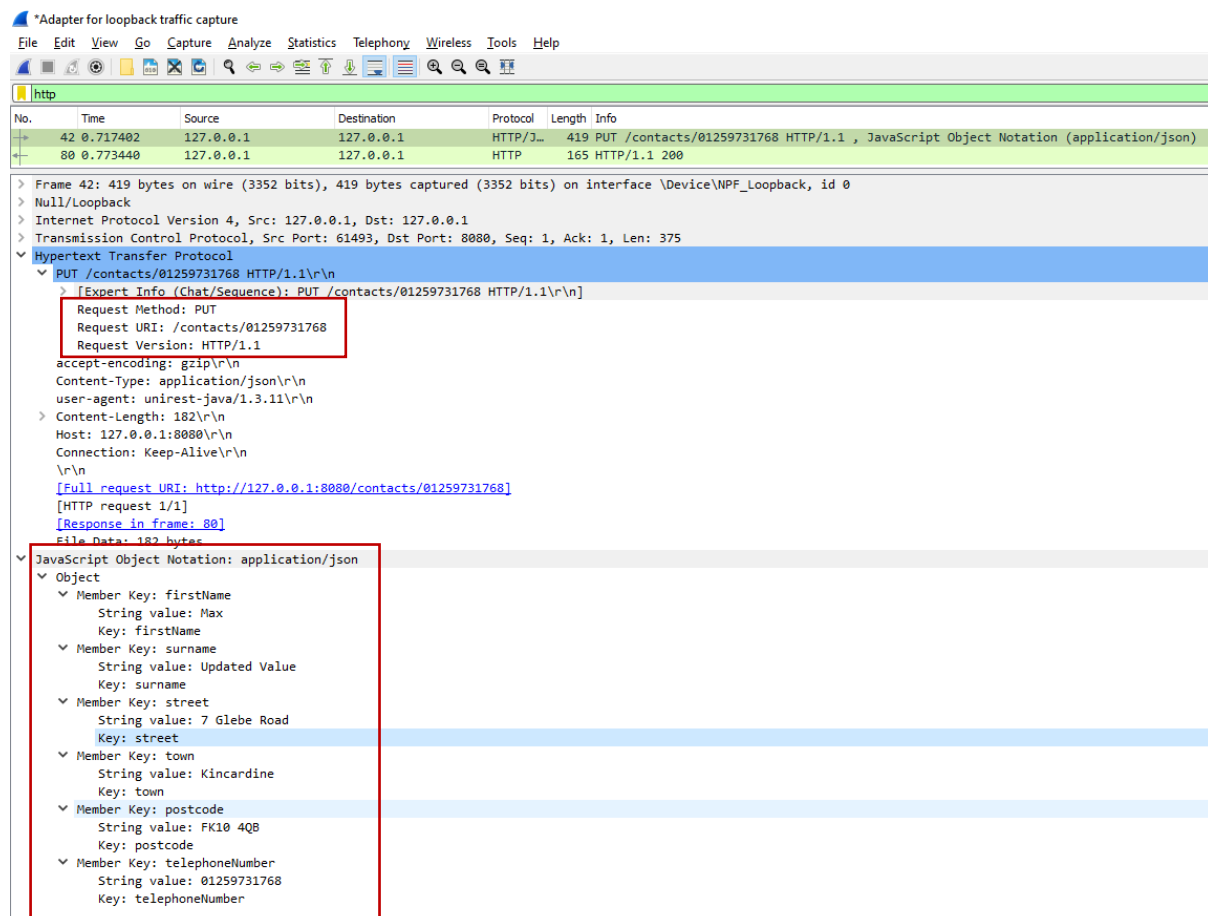


**Figure 17. Receiving a GET Response**

### 3.3.2. PUT

#### 3.3.2.1. Sending out a PUT Request

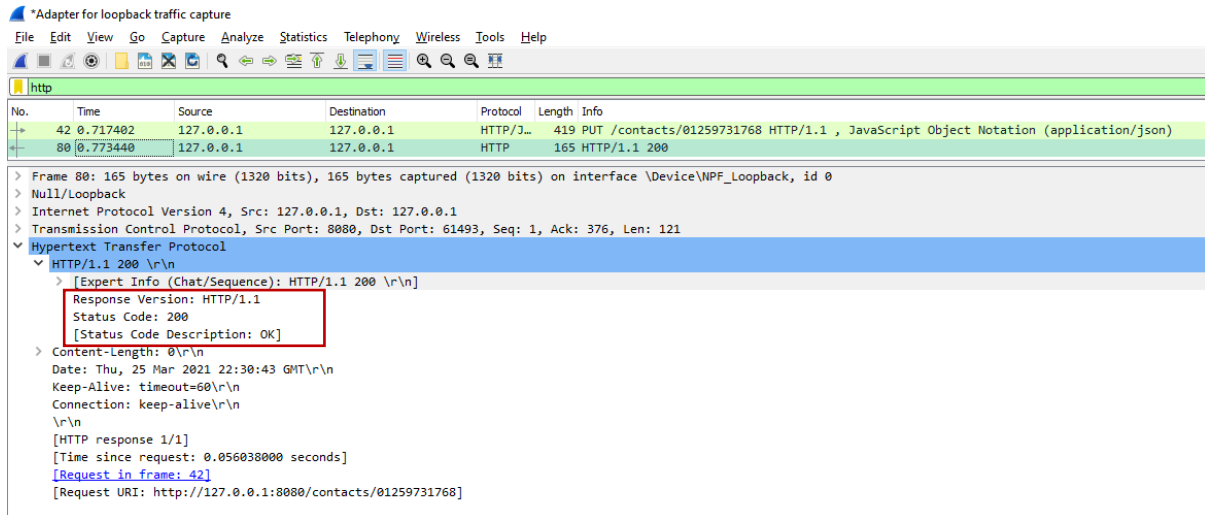
When the user edits a contact, they send out a PUT request to update the contact they have chosen with the use of the telephone number (**Figure 18.**). From the first highlighted area in the screenshot it shows that the method that has been used is PUT along with the telephone number of the contact that is to be edited. The second highlighted area is showing the new contact information that is to be used to update the contact, the example I have used is the surname field contains the only new value which is "Updated Value" to show that the PUT request actually works and has taken passed in the updated values from the application. This also shows that the contacts information is send back in a JSON format.



**Figure 18. Sending out a PUT Request**

### 3.3.1.2. Retrieving a PUT Response

When the PUT request has been successful, it returns the status code 200 which is shown in the highlight area of **Figure 19**. It also shows the description of the status code which displays “OK”, this means that the PUT request has been successful and the contact has been updated.



**Figure 19. Sending out a PUT Request**

### 3.3.3. POST

#### 3.3.3.1. Sending out a POST Request

When the user wants to create a new contact, they send out a POST request with the use of new entered data (**Figure 20.**). In the first highlighted section, it shows that the method used is POST along with the URI that is used to call the correct POST request method. The second highlighted area shows what the new contacts data will, is contains each of the values to create a new contact.

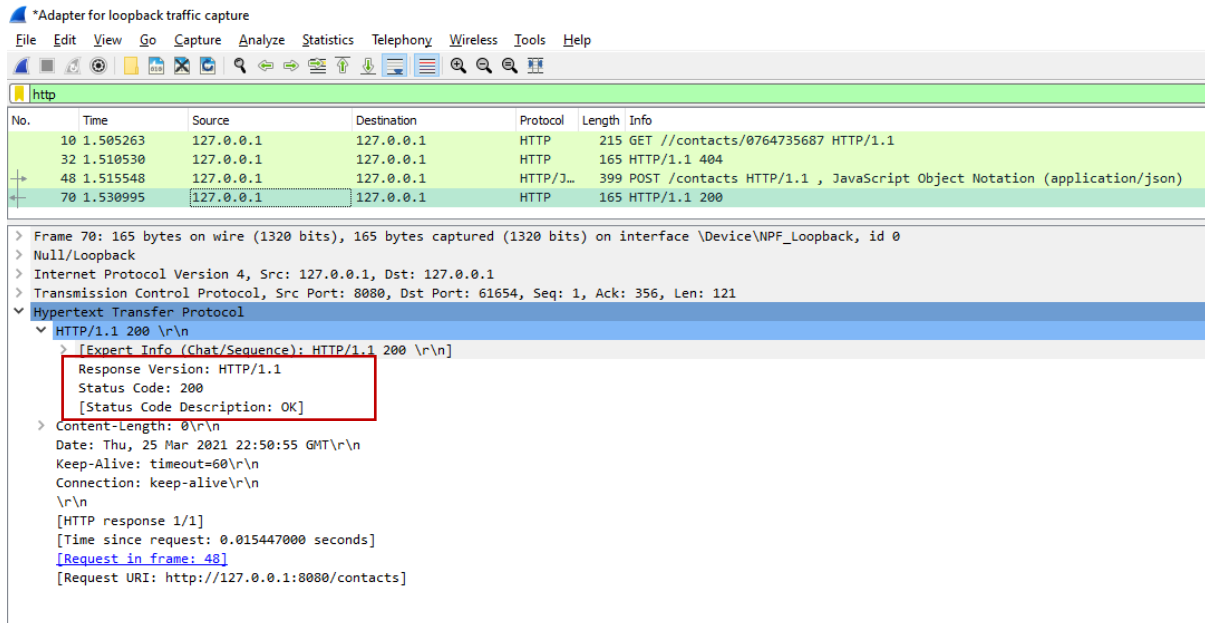
The image shows a Wireshark packet capture of an HTTP POST request. The top table lists several packets, with packet 48 (1.515548) being the selected one. The packet details pane shows the following structure:

- Frame 48: 399 bytes on wire (3192 bits), 399 bytes captured (3192 bits) on interface \Device\NPF\_{...}, id 0
- Null/Loopback
- Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
- Transmission Control Protocol, Src Port: 61654, Dst Port: 8080, Seq: 1, Ack: 1, Len: 355
- Hypertext Transfer Protocol
  - POST /contacts HTTP/1.1\r\n
    - [Expert Info (Chat/Sequence): POST /contacts HTTP/1.1\r\n]
    - Request Method: POST
    - Request URI: /contacts
    - Request Version: HTTP/1.1
  - accept-encoding: gzip\r\n
  - Content-Type: application/json\r\n
  - user-agent: unirest-java/1.3.11\r\n
  - Content-Length: 173\r\n
  - Host: 127.0.0.1:8080\r\n
  - Connection: Keep-Alive\r\n
  - \r\n
  - [Full request URI: http://127.0.0.1:8080/contacts]
  - [HTTP request 1/1]
  - [Response in frame: 70]
  - File Data: 173 bytes
- JavaScript Object Notation: application/json
  - Object
    - Member Key: firstName
      - String value: Jimmy
      - Key: firstName
    - Member Key: surname
      - String value: Jim
      - Key: surname
    - Member Key: street
      - String value: 10 Falkirk Road
      - Key: street
    - Member Key: town
      - String value: Falkirk
      - Key: town
    - Member Key: postcode
      - String value: RG56 W34
      - Key: postcode
    - Member Key: telephoneNumber
      - String value: 0764735687
      - Key: telephoneNumber

**Figure 20. Sending out a POST Request**

### 3.3.3.2. Retrieving a POST Response

After a POST request has successfully been executed, the status code of 200 will be returned (**Figure 21.**), along with the description of “OK”. This means that the POST request has been executed and the new contact has been created and inserted into the database.

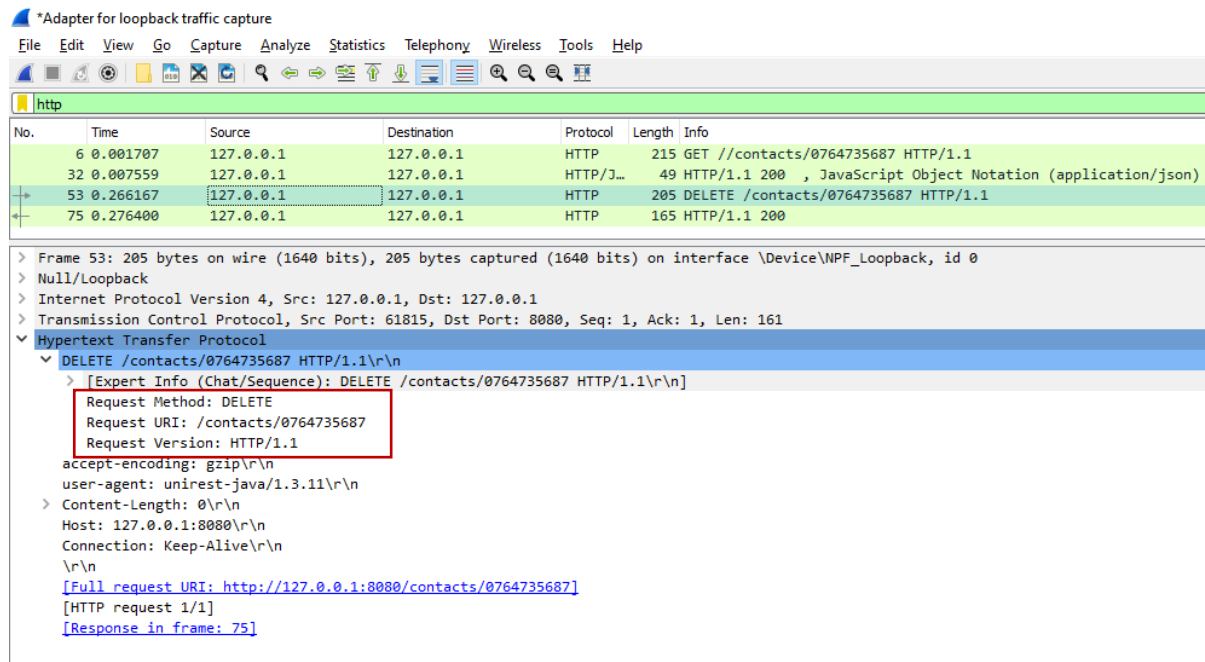


**Figure 21. Receiving a POST Response**

### 3.3.4. DELETE

#### 3.3.4.1. Sending out a DELETE Request

When the user wants to delete a contact, they send out a DELETE request when the process is activated (**Figure 22.**). In the highlighted section on the screenshot of the request, it shows the telephone number used for the contact that is to be deleted in the URI. It also shows the method that is being used in the call which is DELETE.

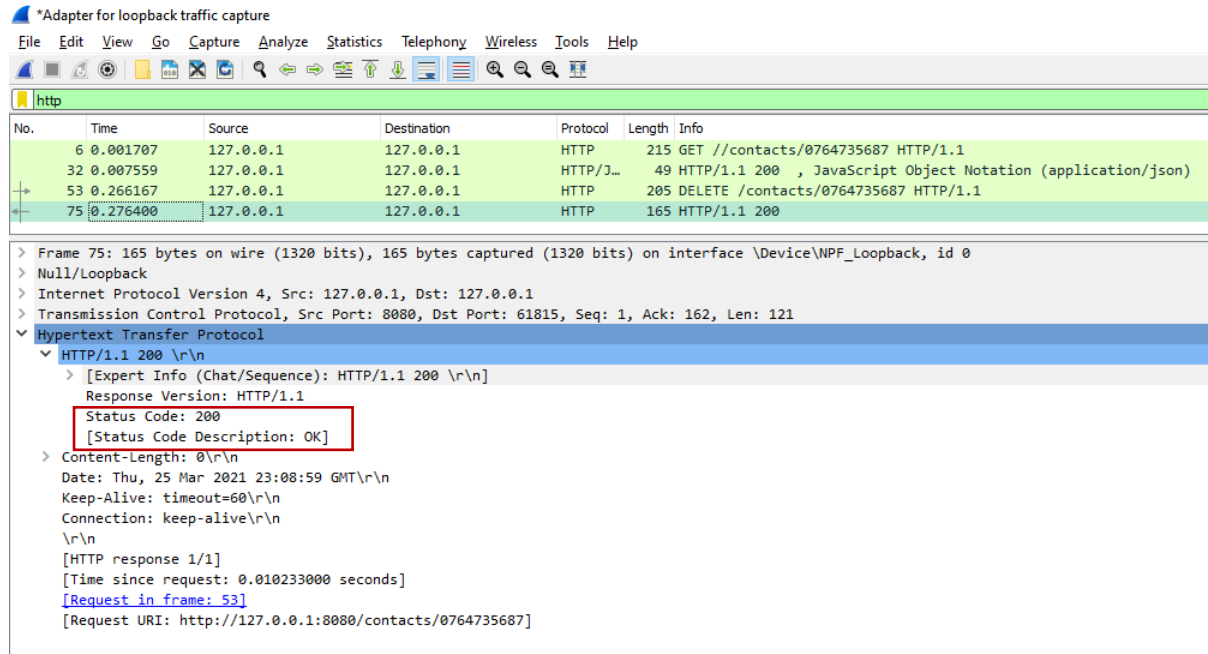


**Figure 22. Sending out a DELETE Request**



### 3.3.4.2. Retrieving a DELETE Response

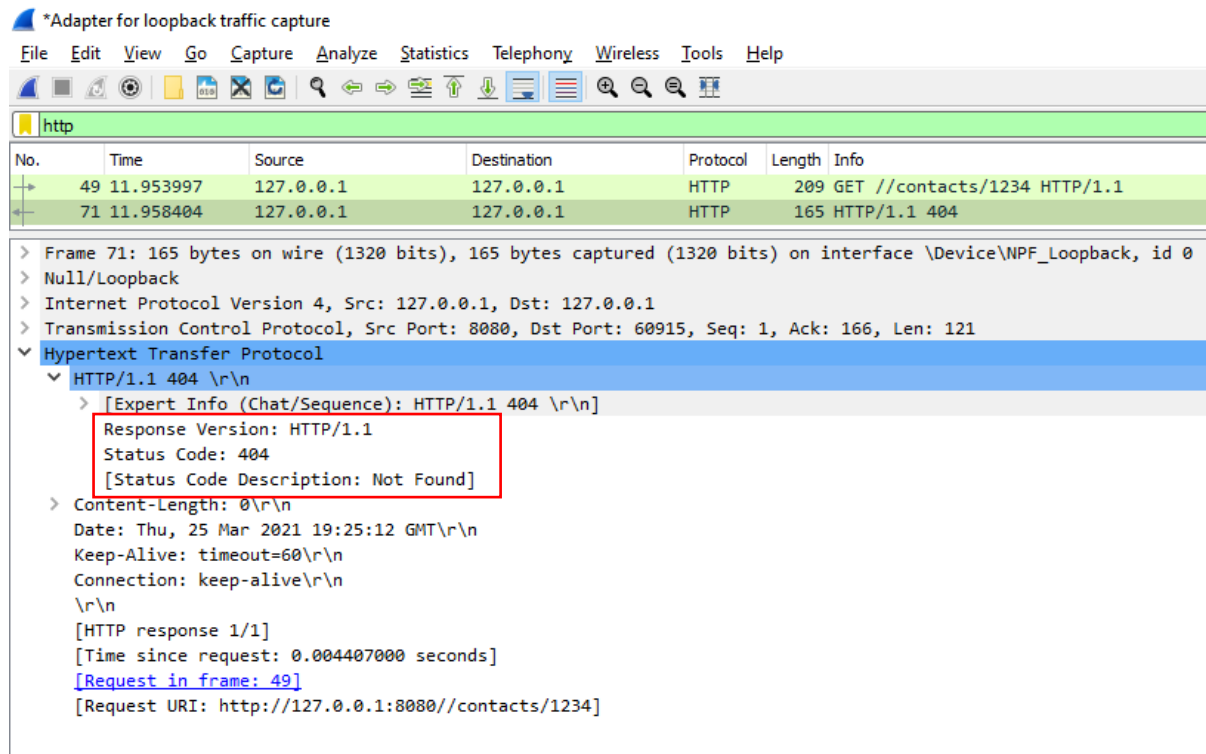
After the DELETE request has been made and it has been successful, the status code code of 200 is displayed back along with the description “OK” which means that the specific contact has been deleted out of the data base (**Figure 23.**).



**Figure 23. Receiving a DELETE Response**

### 3.3.2. Error Response

Not all requests that are sent return successfully, this is what gets returned is a telephone number is searched for that is not currently held within the database (**Figure 24.**). As you can see in the highlighted area in the response, it shows the status code of the request which is 404 along with the description of “Not Found” meaning that the contact could not be found and does not exist. Which ends up returning a nothing (null) back to the application.



**Figure 24. Receiving a Failed GET Response**

## 4. Key Coding Constructs

### 4.1. Web Service and Database Connections

These two classes contain the main features that control the requests for the RESTful web service that allows the application to communicate with the web service and the database simultaneously. The code for these two classes found in appendix [6.3. Appendix Three: ContactsController.java](#) and appendix [6.5. Appendix Five: Requests.java](#), in the controller class it uses the `@RestController` which enables the class to use the RESTful web service annotations that allow the application to send out different requests to interact with the web service. Doing this also makes the application communicate with the database using the contact service class working with the contact repository class, these are shown in appendix [6.4. Appendix Four: ContactService.java](#) and appendix [6.6. Appendix Six: ContactRepository.java](#) which use the `JpaRepository` that allows the web service to interact with the database to retrieve and modify the data.

### 4.2. Graphical User Interface

The interface classes consist of a main menu ([6.7. Appendix Seven: MainMenuUserinterface.java](#)), a new contact interface ([6.8. Appendix Eight: NewContactUserInterface.java](#)), an edit contact interface ([6.9. Appendix Nine: EditContactUserInterface.java](#)) and a single contact display ([6.10. Appendix Nine: SingleContactUserInterface.java](#)). These are what the user will see when they start the java application and interact with the built-in features.

## 5. Project Completion

### 5.1. Completed Implementation

I was able to implement all the basic required features and functionality into the application along with some additional features that were not listed to build upon the usability of the application. These functions and features have been discussed and explained along with the relevant screenshot in section 3. Solution. Here is a list of all the features and functionality that has been implemented into the application:

- Spring RESTful web service.
- An external database as the persistence storage to store the contacts, this was made with MySQL through the MySQL Workbench.
- The ability to create, edit, retrieve, and delete contacts.
- Error handling for special cases from user input, prevents duplicate values of data along with preventing the application from crashing or running into any errors when searches are carried out on contacts that do not exist.
- Displays relevant messages to the screen when an action has been carried out whether it is a successful or failed action.
- A display of all contacts currently being held in the database.
- A contact can be searched for using their telephone number.
- A contacts details include first name, surname, street, town, postcode, and telephone number.
- Each contact has a unique identifier which has been set as their telephone number, meaning that a telephone number can only be assigned to one contact and cannot be changed unless the contact is edited or deleted.
- Duplicate contacts cannot be added into the database.
- GET, PUT, POST and DELETE requests can be sent out and received with correct handling for the responses.
- Real time updates with the database through the RESTful web service when a request has been sent.
- A graphical user interface has been created to service as the client side of the application, it allows the user to easily modify the contacts database.

### 5.2. Incomplete Implementation

The only features I was unable to implement were some of the advanced features like grouping up contacts, other than that everything else has been implemented.

### 5.3. Alternative Approach and Additional Functionality

With more time allocated for the project I would have implemented a lot more advanced features such as adding a new field into the database containing the number of edits that have been made for a contact. I would also improve the security of the application surrounding the database actions as the database can easily be exploited through SQL injection. A different approach I would take would work from if I were to do this again or had more time would be to use a different lab practical to see how the process of communicating with the database as well as the web service with the use of SOAP messages.

## 6. Appendices

### 6.1. Appendix One: Main.java

```
package stirring.cscu9yw.main;
```

```
import org.springframework.boot.SpringApplication;
```

```
import org.springframework.boot.autoconfigure.SpringBootApplication;
```

```
/**
```

```
 * This is the main class for the applications web service, when this is  
run the
```

```
 * spring service is started.
```

```
 *
```

```
 * @author 2636157
```

```
 *
```

```
 */
```

```
@SpringBootApplication
```

```
public class Main {
```

```
    /**
```

```
     * Starts the spring application service.
```

```
     *
```

```
     * @param args An array of sequence of characters (Strings) that are  
passed to
```

```
     *         the "main" function.
```

```
     */
```

```
    public static void main(String[] args) {
```

```
        SpringApplication.run(Main.class, args);
```

```
    }
```

```
}
```

## 6.2. Appendix Two: Contact.java

```
package stirling.cscu9yw.main;

import javax.persistence.*;

// Specifies that the class is an entity. This annotation is applied to the entity
class.
@Entity
// Identifies the table name that will be used from the database.
@Table(name = "contacts")

/**
 * This class is used to create a new contact, it contains the getters and
 * setters to retrieve a contacts information or create a completely new one.
 *
 * @author 2636157
 */
public class Contact {

    // The @Column tags are used to identify the column from the database and
link // it to its relevant String, the @Id identifies the primary key.

    @Column(name = "first_name")
    private String firstName;
    @Column(name = "surname")
    private String surname;
    @Column(name = "street")
    private String street;
    @Column(name = "town")
    private String town;
    @Column(name = "postcode")
    private String postcode;
    @Column(name = "telephone_Number")
    @Id
    private String telephoneNumber;

    /**
    this * This is a constructor for the class without anything being passed in,
    * allows the contact class to be used and accessed throughout the
    application * without creating a new contact.
    */
    public Contact() {

    }

    /**
    contact is * This is the constructor for the class, when this is called, a new
    * created with the data that has been passed in.
    *
    * @param firstName Passes in the first name value for the contact.
    * @param surName Passes in the surname value for the contact.
    * @param street Passes in the street value for the contact.
    * @param town Passes in the town value for the contact.
    * @param postcode Passes in the post code value for the contact.
```

```
* @param telephoneNumber Passes in the telephone number value for the
contact.
*/
public Contact(String firstName, String surName, String street, String
town, String postcode,
                String telephoneNumber) {

    this.telephoneNumber = telephoneNumber;
    this.firstName = firstName;
    this.surname = surName;
    this.street = street;
    this.town = town;
    this.postcode = postcode;
}

public String getFirstName() {
    return firstName;
}

public void setFirstName(String firstName) {
    this.firstName = firstName;
}

public String getSurname() {
    return surname;
}

public void setSurName(String surName) {
    this.surname = surName;
}

public String getStreet() {
    return street;
}

public void setStreet(String street) {
    this.street = street;
}

public String getTown() {
    return town;
}

public void setTown(String town) {
    this.town = town;
}

public String getPostcode() {
    return postcode;
}

public void setPostCode(String postcode) {
    this.postcode = postcode;
}

public String getTelephoneNumber() {
    return telephoneNumber;
}
```

Student ID Number  
263157

```
public void setTelephoneNumber(String telephoneNumber) {  
    this.telephoneNumber = telephoneNumber;  
}  
  
@Override  
public String toString() {  
    return "Contact First Name : " + firstName + ", Surname : " +  
surname + ", Street : " + street + ", Town : "  
        + town + ", Postcode : " + postcode + ", Telephone  
Number : " + telephoneNumber;  
}  
}
```



### 6.3. Appendix Three: ContactsController.java

```
package sterling.cscu9yw.main;
```

```
import java.util.List;
```

```
import java.util.NoSuchElementException;
```

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

```
import org.springframework.http.HttpStatus;
```

```
import org.springframework.http.ResponseEntity;
```

```
import org.springframework.web.bind.annotation.*;
```

```
/**
```

```
 * The controller holds of all of the RESTful web services, any action that is  
 * carried out with communicating with the web service is done through this  
 * class.
```

```
 *
```

```
 * @author 2636157
```

```
 *
```

```
 */
```

```
// The Spring RestController annotation is used to create RESTful web services using the Spring MVC.
```

```
// The RestController takes care of mapping request data of the defined request.
```

```
@RestController
```

```
public class ContactsController {
```

```
    // The @Autowired annotation is added to the ContactService to be used by
```

```
    // Spring's dependency injection facilities. This allows method calls to be made
```

```
    // to class that holds all of the requests actions.
```

```
    @Autowired
```

```
    ContactService contactService;
```

```
/**
 * Annotation for mapping HTTP GET requests, retrieves and returns all contacts.
 *
 * @return All contacts.
 */
@GetMapping("/contacts")
public List<Contact> list() {
    return contactService.listAllContacts();
}

/**
 * Annotation for mapping HTTP GET requests, retrieves a single contact using a
 * telephone number.
 *
 * @param telephoneNumber Passed in telephone number, the unique identifier.
 * @return A single contact and a HTTP response.
 */
@GetMapping("/contacts/{telephoneNumber}")
public ResponseEntity<Contact> get(@PathVariable String telephoneNumber) {
    try {
        Contact contact = contactService.getContact(telephoneNumber);
        // Returns the contact along with a successful HTTP response.
        return new ResponseEntity<Contact>(contact, HttpStatus.OK);
    } catch (NoSuchElementException e) {
        // Only returns a failure HTTP response.
        return new ResponseEntity<Contact>(HttpStatus.NOT_FOUND);
    }
}

/**
```

\* Annotation for mapping HTTP POST requests, creates a new contact, takes in a  
\* request body that holds all of the required data to create a new contact.

\*

\* @param contact Passed in request body holding a new contacts information.

\*/

@PostMapping("/contacts")

public void add(@RequestBody Contact contact) {

    contactService.saveContact(contact);

}

/\*\*

\* Annotation for mapping HTTP PUT requests, retrieves and returns a contact by

\* taking in a request body that holds the data for a contact and a telephone

\* number to find the specific contact to be updated.

\*

\* @param contact      Passed in request body holding updated information for

\*                      the contact.

\* @param telephoneNumber Passed in telephone number, the unique identifier.

\* @return A HTTP response.

\*/

@PutMapping("/contacts/{telephoneNumber}")

public ResponseEntity<?> update(@RequestBody Contact contact, @PathVariable String  
telephoneNumber) {

    try {

        Contact existContact = contactService.getContact(telephoneNumber);

        contact.setTelephoneNumber(telephoneNumber);

        // Saves the contact information.

        contactService.saveContact(contact);

        // Returns a successful HTTP response.

        return new ResponseEntity<>(HttpStatus.OK);

    } catch (NoSuchElementException e) {

Student ID Number  
263157

```
// Returns a failure HTTP response.
return new ResponseEntity<>(HttpStatus.NOT_FOUND);
    }
}

/**
 * Annotation for mapping HTTP DELETE requests, retrieves the required contact
 * using the passed in phone number and deletes it.
 *
 * @param telephoneNumber Passed in telephone number, the unique identifier.
 */
@DeleteMapping("/contacts/{telephoneNumber}")
public void delete(@PathVariable String telephoneNumber) {
    contactService.deleteContact(telephoneNumber);
}
}
```

#### 6.4. Appendix Four: ContactService.java

```
package sterling.cscu9yw.main;
```

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

```
import org.springframework.stereotype.Service;
```

```
import javax.transaction.Transactional;
```

```
import java.util.List;
```

```
/**
```

```
 * This class holds all the method calls for the services that are used to  
 * control the RESTful services with the database, it allows the actions being  
 * carried out on the web service to be done to the database at the same time.
```

```
 *
```

```
 * It does this by calling the ContactRepository which holds the JpaRepository,  
 * this controls and handles the actions required for the web service and the  
 * database to work together.
```

```
 *
```

```
 * @author 2636157
```

```
 *
```

```
 */
```

```
// Indicates that the class is to be used as a service.
```

```
@Service
```

```
// The @Transactional annotation provides the application the ability to declaratively control
```

```
// transaction boundaries on CDI managed beans, as well as classes defined as managed beans.
```

```
@Transactional
```

```
public class ContactService {
```

```
    // The @Autowired annotation marks the constructor for the class that is to be
```

```
    // used by Spring's dependency injection facilities.
```

@Autowired

private ContactRepository contactRepository;

/\*\*

\* Calls the ContactRepository to retrieve all contacts.

\*

\* @return Returns all contacts.

\*/

```
public List<Contact> listAllContacts() {  
    return contactRepository.findAll();  
}
```

/\*\*

\* Calls the ContactRepository to save a contact.

\*

\* @param contact Passes in a contact.

\*/

```
public void saveContact(Contact contact) {  
    contactRepository.save(contact);  
}
```

/\*\*

\* Calls the ContactRepository to get a single contact with the use of a phone  
\* number.

\*

\* @param telephoneNumber Passes in the telephone number that is used to search  
\* for the contact.

\* @return The searched for contact.

\*/

```
public Contact getContact(String telephoneNumber) {  
    return contactRepository.findById(telephoneNumber).get();  
}
```

```
}

/**
 * Calls the ContactRepository to delete a contact using the contacts telephone
 * number.
 *
 * @param telephoneNumber Passes in the telephone number of the contact that is
 * used to delete the contact.
 */
public void deleteContact(String telephoneNumber) {
    contactRepository.deleteById(telephoneNumber);
}
}
```

6.5. Appendix Five: Requests.java

```
package stirling.cscu9yw.main;

import java.io.IOException;
import java.io.InputStream;
import java.net.URL;
import java.nio.charset.StandardCharsets;

import com.mashape.unirest.http.HttpResponse;
import com.mashape.unirest.http.Unirest;
import com.mashape.unirest.http.exceptions.UnirestException;

import org.json.JSONObject;

/**
 * This class holds all the requests used for the web service, it works
 * along
 * side the ContactController and Unirest to send out requests and
 * retrieve
 * data.
 *
 * @author Max 2636157
 *
 */
public class Requests {
    private static final String serviceURI = "http://127.0.0.1:8080/";

    /**
     * The get method retrieves a contact which gets converted into
     * string through
     * the convertString method, it then gets converted into a JSON
     * object by
     * getting passed through the convertData method.
     */
}
```



```
*
* @param telephoneNumber Passed in for searching a specific
contact.
* @param enitity          Passed in String value to get the specific
piece of
*                          data to be returned.
* @return A single value that from a contact that has been
searched.
*/
public static String get(String telephoneNumber, String enitity) {
    String output = null;

    // Try catch statement used to catch any IOExceptions if
nothing is returned
    // after searching, if an exception is thrown, the output is
set to nothing
    // instead of null to prevent the application from crashing.
    try {
        // Passes in the uri that is used for sending out a
request.
        output = (convertString(serviceURI + "/contacts/" +
telephoneNumber).toString());
    } catch (IOException e) {
        output = "";
    }

    // If statement used to check if the output is not empty, if
not, the output can
    // be converted into JSON format.
    if (!output.equals("")) {
        output = convertData(output, enitity);
        return output;
        // Else, the output is returned back empty.
    } else {
        return output;
    }
}
```

```
    }

}

/**
 *
 * The put method grants the ability to edit a contact, it takes in
the
 * parameters from a contact and sends out a HTTP request contain a
request body
 * with the updated contact data through the Unirest put call.
 *
 * @param firstName      Passes in the contacts first name value.
 * @param surname        Passes in the contacts surname value.
 * @param street         Passes in the contacts street value.
 * @param town           Passes in the contacts town value.
 * @param postcode       Passes in the contacts town value.
 * @param telephoneNumber Passes in the contacts telephone value.
 */

public static void put(String firstName, String surname, String
street, String town, String postcode,
                        String telephoneNumber) {
    Unirest.setTimeouts(0, 0);
    try {
        HttpResponse<String> response = Unirest.put(serviceURI +
"/contacts/" + telephoneNumber)
            .header("Content-Type", "application/json")
            .body("{\r\n    \"firstName\": \"" +
firstName + "\", \r\n    \"surname\": \"" + surname
                        + "\", \r\n    \"street\": \"" +
street + "\", \r\n    \"town\": \"" + town
                        + "\", \r\n    \"postcode\": \"" +
postcode + "\", \r\n    \"telephoneNumber\": \""
                        + telephoneNumber + "\"\r\n}")
    }
```

```
.asString();

    } catch (UnirestException e) {
        e.printStackTrace();
    }
}

/**
 * The post method carries out the creation of a new contact, it
 * does this by
 *   * taking in the values for the contact and sending them out in
 *   * request body
 *   * from the Unirest post call.
 *
 * @param firstName      Passes in the contacts first name value.
 * @param surname         Passes in the contacts surname value.
 * @param street          Passes in the contacts street value.
 * @param town            Passes in the contacts town value.
 * @param postcode        Passes in the contacts town value.
 * @param telephoneNumber Passes in the contacts telephone number
 * value.
 */
public static void post(String firstName, String surname, String
street, String town, String postcode,
                        String telephoneNumber) {

    Unirest.setTimeouts(0, 0);
    try {
        HttpResponseMessage<String> response = Unirest.post(serviceURI
+ "/contacts")

                                .header("Content-Type", "application/json")

                                .body("{\r\n    \"firstName\":\"" +
firstName + "\", \r\n    \"surname\":\"" + surname
```

```
street + "\",\r\n    \"town\": \"" + town
+ "\",\r\n    \"street\": \"" +
postcode + "\",\r\n    \"telephoneNumber\": \""
+ telephoneNumber + "\",\r\n}");

    .asString();
    } catch (UnirestException e) {
        e.printStackTrace();
    }
}

/**
 * The delete method deletes a contact with the use of the contacts
telephone
 * number value number, it does with the Unirest delete call.
 *
 * @param telephoneNumber Passes in the contacts telephone number.
 */
public static void delete(String telephoneNumber) {
    Unirest.setTimeouts(0, 0);
    try {
        HttpResponse<String> response =
Unirest.delete(serviceURI + "/contacts/" + telephoneNumber).asString();
    } catch (UnirestException e) {
        e.printStackTrace();
    }
}

/**
 * This method gets used with the get method to convert the output
from the get
 * request into String format, it uses the inputStream to do it and
returns it
 * with the UTF_8 charsets.
```

```
*  
  
* @param requestURL Passed in url that will be used to search and  
retrieve a  
  
*          contact.  
  
* @return A string value of the contact retrieved.  
  
* @throws IOException Signals that an I/O exception of some sort  
has occurred.  
  
*          This class is the general class of exceptions  
produced by  
  
*          failed or interrupted I/O operations.  
  
*/  
  
private static String convertString(String requestURL) throws  
IOException {  
    URL url = new URL(requestURL);  
    try (InputStream inputStream = url.openStream()) {  
        return new String(inputStream.readAllBytes(),  
StandardCharsets.UTF_8);  
    }  
}  
  
/**  
  
* This method converts the String value of the contact passed  
through into a  
  
* JSON object, this then gets used to search for a specific piece  
of data from  
  
* the contact using a passed in string value called entity.  
  
*  
  
* @param forConversion Passes in the contact that needs to be  
converted.  
  
* @param entity          Passes in a String value for what piece of  
data is to be  
  
*          pulled and returned.  
  
* @return A single piece of data from the contact.  
  
*/
```

```
public static String convertData(String forConversion, String
entity) {

    String model = null;
    JSONObject contact = new JSONObject(forConversion);

    model = contact.getString(entity);

    return model;
}

}
```

6.6. Appendix Six: ContactRepository.java

```
package stirling.cscu9yw.main;

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

/**
 * This interface is used work with the database and the web service
 * simultaneously, it does this by using the JpaRepository which is a standard
 * way of persisting objects into a database as well as allowing requests to be
 * sent to the web service.
 *
 * It does this by using the contact object along with a String value which is
 * used as the unique identifier for finding a contact, which is the telephone
 * number in this case.
 *
 * @author 2636157
 */
public interface ContactRepository extends JpaRepository<Contact, String> {

}
```

6.7. Appendix Seven: MainMenuUserinterface.java

```
package stirring.cscu9yw.userinterfaces;
```

```
import java.awt.EventQueue;
```

```
import javax.swing.JFrame;
```

```
import javax.swing.JLabel;
```

```
import javax.swing.JOptionPane;
```

```
import java.awt.BorderLayout;
```

```
import java.awt.FlowLayout;
```

```
import javax.swing.JTextField;
```

```
import stirring.cscu9yw.database.ConnectDB;
```

```
import stirring.cscu9yw.main.Requests;
```

```
import javax.swing.JButton;
```

```
import javax.swing.JTable;
```

```
import java.awt.event.ActionListener;
```

```
import java.util.regex.Matcher;
```

```
import java.util.regex.Pattern;
```

```
import java.awt.event.ActionEvent;
```

```
import javax.swing.JScrollPane;
```

```
import java.awt.Font;
```

```
import javax.swing.JTextPane;
```

```
/**
```

```
 *
```

```
 * This user interface is displayed when the java application is launched;
```

```
 *
```

```
 * @author 2636157
```

```
 *
```



```
*/  
public class MainMenuUserInterface {  
  
    private JFrame frame;  
    private JTextField textFieldTelephoneNumber;  
  
    private String telephoneNumber;  
    private JTable table;  
  
    private JScrollPane scrollPane = new JScrollPane();  
  
    public JButton btnDisplayAllContacts;  
  
    /**  
     * Launch the application.  
     */  
    public static void main(String[] args) {  
        EventQueue.invokeLater(new Runnable() {  
            public void run() {  
                try {  
                    MainMenuUserInterface window = new  
MainMenuUserInterface();  
                    window.frame.setVisible(true);  
                } catch (Exception e) {  
                    e.printStackTrace();  
                }  
            }  
        });  
    }  
  
    /**  
     * Create the application.
```

```
*/  
  
public MainMenuUserInterface() {  
    initialize();  
}  
  
/**  
 * Initialize the contents of the frame.  
 */  
private void initialize() {  
    frame = new JFrame();  
    frame.setBounds(100, 100, 1017, 375);  
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
    frame.getContentPane().setLayout(null);  
    frame.setLocationRelativeTo(null);  
  
    JLabel lblNewLabel = new JLabel("Telephone Number");  
    lblNewLabel.setBounds(20, 73, 134, 20);  
    frame.getContentPane().add(lblNewLabel);  
  
    textFieldTelephoneNumber = new JTextField();  
    textFieldTelephoneNumber.setBounds(20, 92, 208, 20);  
    frame.getContentPane().add(textFieldTelephoneNumber);  
    textFieldTelephoneNumber.setColumns(10);  
  
    scrollPane.setBounds(238, 27, 735, 255);  
    frame.getContentPane().add(scrollPane);  
  
    btnDisplayAllContacts = new JButton("Display All Contacts");  
    btnDisplayAllContacts.addActionListener(new ActionListener() {  
        public void actionPerformed(ActionEvent e) {  
            String[] column = { "Telephone Number", "First  
Name", "Surname", "Street", "Town", "Postcode" };
```

```
ConnectDB connectDB = new ConnectDB();

table = new JTable(connectDB.db_select(), column);
scrollPane.setViewportViewView(table);
    }
});

btnDisplayAllContacts.setBounds(20, 123, 208, 23);
frame.getContentPane().add(btnDisplayAllContacts);

JButton btnNewButton = new JButton("New Contact");
btnNewButton.addActionListener(new ActionListener() {

    /**
     * Launches and displays the new contact interface when
the new contact button
     * is clicked.
     */
    public void actionPerformed(ActionEvent e) {

        NewContactUserInterface newcontactUserInterface =
new NewContactUserInterface();
        newcontactUserInterface.setVisible(true);

    }
});

btnNewButton.setBounds(20, 191, 208, 23);
frame.getContentPane().add(btnNewButton);

JButton btnNewDelete = new JButton("Delete Contact ");

/**
```

```
        * Deletes a contact using the telephone number entered when
the delete button

        * is clicked. Carries out error checking to ensure that the
value that has been

        * entered is valid, if not error messages will be displayed.
*/
btnNewDelete.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {

        telephoneNumber =
textFieldTelephoneNumber.getText().toString();

        if (validation(telephoneNumber)) {
            Requests.delete(telephoneNumber);
            JOptionPane.showMessageDialog(new JFrame(),
"Contact Successfully Deleted", "Contact Deleted",
JOptionPane.PLAIN_MESSAGE);

            textFieldTelephoneNumber.setText("");

            btnDisplayAllContacts.doClick();

        }
    }
});

btnNewDelete.setBounds(20, 259, 208, 23);
frame.getContentPane().add(btnNewDelete);

/**
 * Searches a contact using the telephone number entered when
the search by

 * telephone number button is clicked. Carries out error
checking to ensure that
```

Student ID Number  
263157

```
        * the value that has been entered is valid and displays the
contacts info, if
        * not error messages will be displayed.
    */

    JButton btnSearchByTelephoneNumber = new JButton("Search By
Telephone Number");

    btnSearchByTelephoneNumber.addActionListener(new
ActionListener() {

        public void actionPerformed(ActionEvent e) {

            SingleContactUserInterface
singleContactUserInterfacen = new SingleContactUserInterface();

            telephoneNumber =
textFieldTelephoneNumber.getText().toString();

            if (validation(telephoneNumber)) {

                singleContactUserInterfacen.setVisible(true);

                singleContactUserInterfacen.getContactDetails(telephoneNumber);

                textFieldTelephoneNumber.setText("");

            }

        }

    });

    btnSearchByTelephoneNumber.setBounds(20, 157, 208, 23);
    frame.getContentPane().add(btnSearchByTelephoneNumber);

    /**
    * Launches and displays the edit contact interface when the
edit contact button
    * is clicked. Carries out error checking to ensure that the
value that has been
```

```
        * entered is valid and displays the contacts info, if not
error messages will
        * be displayed.
        */
        JButton btnEditContact = new JButton("Edit Contact");
        btnEditContact.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {

                EditContactUserInterface editContactUserInterface
= new EditContactUserInterface();

                telephoneNumber =
textFieldTelephoneNumber.getText().toString();

                if (validation(telephoneNumber)) {
                    editContactUserInterface.setVisible(true);

                    editContactUserInterface.setContactDetails(telephoneNumber);

                    textFieldTelephoneNumber.setText("");
                }
            }
        });
        btnEditContact.setBounds(20, 225, 208, 23);
        frame.getContentPane().add(btnEditContact);

        JLabel lblContactsDatabase = new JLabel("Contacts Database");
        lblContactsDatabase.setFont(new Font("Tahoma", Font.PLAIN,
17));

        lblContactsDatabase.setBounds(20, 27, 164, 35);
        frame.getContentPane().add(lblContactsDatabase);

        JButton btnClear = new JButton("Clear");
        btnClear.addActionListener(new ActionListener() {
```

```
        public void actionPerformed(ActionEvent e) {
            textFieldTelephoneNumber.setText("");
        }
    });
    btnClear.setBounds(164, 76, 64, 15);
    frame.getContentPane().add(btnClear);

}

/**
 * Handles the validation for the input of the telephone number, it
check if the
 * input is empty, if the telephone number exists , if it's already
being used
 * and if the input is only numbers.
 *
 * @param telephoneNumber Passes in the inputed telephone number.
 * @return True if the input passes all the validation, false if the
input gets
 *      caught.
 */
public Boolean validation(String telephoneNumber) {

    String contact;

    if (checkDigits(telephoneNumber)) {

        contact = Requests.get(telephoneNumber,
"telephoneNumber");

        if (!contact.equals("")) {
            return true;
        }
    }
}
```

```
        } else {
            JOptionPane.showMessageDialog(new JFrame(),
"Contact does not exist", "Error",
            JOptionPane.ERROR_MESSAGE);

            return false;
        }

        } else if (telephoneNumber.equals("")) {
            JOptionPane.showMessageDialog(new JFrame(), "No
Telephone Number Entered", "Error",
            JOptionPane.ERROR_MESSAGE);

            return false;
        } else {
            JOptionPane.showMessageDialog(new JFrame(), "Only
Numbers can be Entered", "Error",
            JOptionPane.ERROR_MESSAGE);

            return false;
        }
    }

    /**
     * Checks the input for the telephone number to ensure that only
     numbers have
     * been inputed.
     *
     * @param telephoneNumber Passes in the telephone number to be
     checked.
     * @return A Boolean value whether the telephone number is valid or
     not.
     */
    public static boolean checkDigits(String telephoneNumber) {
```



```
String regex = "[0-9]+";  
Pattern pattern = Pattern.compile(regex);  
  
if (telephoneNumber == null) {  
    return false;  
}  
  
Matcher matcher = pattern.matcher(telephoneNumber);  
return matcher.matches();  
}  
}
```

6.8. Appendix Eight: NewContactUserInterface.java

```
package stirring.cscu9yw.userinterfaces;

import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import java.awt.Font;
import javax.swing.JTextField;

import stirring.cscu9yw.main.Requests;

import javax.swing.JButton;
import java.awt.event.ActionListener;
import java.util.regex.Matcher;
import java.util.regex.Pattern;
import java.awt.event.ActionEvent;

/**
 *
 * This user interface is displayed when the new contact button has been
 * clicked
 * on the main menu user interface.
 *
 * @author 2636157
 *
 */
public class NewContactUserInterface extends JFrame {

    private JPanel contentPane;
    private JTextField textFieldFirstName;
    private JTextField textFieldSurname;
```

```
private JTextField textFieldStreet;
private JTextField textFieldTown;
private JTextField textFieldPostcode;
private JTextField textFieldTelephoneNumber;

String firstName;
String surname;
String street;
String town;
String postcode;
String telephoneNumber;

/**
 * Create the frame.
 */
public NewContactUserInterface() {

    setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
    setBounds(100, 100, 380, 400);
    contentPane = new JPanel();
    setContentPane(contentPane);
    contentPane.setLayout(null);

    JLabel lblNewLabel = new JLabel("New Contact");
    lblNewLabel.setFont(new Font("Tahoma", Font.PLAIN, 17));
    lblNewLabel.setBounds(21, 11, 100, 35);
    contentPane.add(lblNewLabel);

    JLabel lblNewLabel_1 = new JLabel("First Name");
    lblNewLabel_1.setBounds(21, 54, 124, 20);
    contentPane.add(lblNewLabel_1);
```

```
textFieldFirstName = new JTextField();
textFieldFirstName.setBounds(147, 54, 178, 20);
contentPane.add(textFieldFirstName);
textFieldFirstName.setColumns(10);

textFieldSurname = new JTextField();
textFieldSurname.setColumns(10);
textFieldSurname.setBounds(147, 85, 178, 20);
contentPane.add(textFieldSurname);

textFieldStreet = new JTextField();
textFieldStreet.setColumns(10);
textFieldStreet.setBounds(147, 117, 178, 20);
contentPane.add(textFieldStreet);

textFieldTown = new JTextField();
textFieldTown.setColumns(10);
textFieldTown.setBounds(147, 148, 178, 20);
contentPane.add(textFieldTown);

textFieldPostcode = new JTextField();
textFieldPostcode.setColumns(10);
textFieldPostcode.setBounds(147, 179, 178, 20);
contentPane.add(textFieldPostcode);

textFieldTelephoneNumber = new JTextField();
textFieldTelephoneNumber.setColumns(10);
textFieldTelephoneNumber.setBounds(147, 210, 178, 20);
contentPane.add(textFieldTelephoneNumber);

JLabel lblNewLabel_1_1 = new JLabel("Surname");
lblNewLabel_1_1.setBounds(21, 85, 124, 20);
```

```
contentPane.add(lblNewLabel_1_1);
```

```
JLabel lblNewLabel_1_2 = new JLabel("Street");  
lblNewLabel_1_2.setBounds(21, 117, 124, 20);  
contentPane.add(lblNewLabel_1_2);
```

```
JLabel lblNewLabel_1_3 = new JLabel("Town");  
lblNewLabel_1_3.setBounds(21, 148, 124, 20);  
contentPane.add(lblNewLabel_1_3);
```

```
JLabel lblNewLabel_1_4 = new JLabel("Postcode");  
lblNewLabel_1_4.setBounds(21, 179, 124, 20);  
contentPane.add(lblNewLabel_1_4);
```

```
JLabel lblNewLabel_1_5 = new JLabel("Telephone Number");  
lblNewLabel_1_5.setBounds(21, 210, 124, 20);  
contentPane.add(lblNewLabel_1_5);
```

```
JButton btnAddContact = new JButton("Add Contact");
```

```
/**
```

```
    * A new contact is created when the add contact button is  
    clicked, it takes the
```

```
    * values from the text fields and uses them to create the new  
    contact. Error
```

```
    * handling is done to ensure that the inputs are valid and  
    there is actually
```

```
    * input there and not left empty. Error messages are  
    displayed relevant to the
```

```
    * error that has been caught.
```

```
    *
```

```
    * The interface closes and returns to the main menu interface  
    when the contact
```

```
    * has been added.
```

```
*/  
btnAddContact.addActionListener(new ActionListener() {  
    public void actionPerformed(ActionEvent e) {  
  
        String contact;  
  
        firstName =  
textFieldFirstName.getText().toString();  
        surname = textFieldSurname.getText().toString();  
        street = textFieldStreet.getText().toString();  
        town = textFieldTown.getText().toString();  
        postcode = textFieldPostcode.getText().toString();  
        telephoneNumber =  
textFieldTelephoneNumber.getText().toString();  
  
        if (!firstName.equals("") && !surname.equals("")  
&& !street.equals("") && !town.equals("")  
        && !postcode.equals("") &&  
!telephoneNumber.equals("")) {  
  
            if (!(telephoneNumber.length() > 11)) {  
  
                if (checkDigits(telephoneNumber)) {  
  
                    contact =  
Requests.get(telephoneNumber, "telephoneNumber");  
  
                    if (contact.equals("")) {  
  
                        Requests.post(firstName,  
surname, surname, street, town, postcode, telephoneNumber);  
  
                        textFieldFirstName.setText("");
```

```
textFieldSurname.setText("");

textFieldStreet.setText("");

                                textFieldTown.setText("");

textFieldPostcode.setText("");

textFieldTelephoneNumber.setText("");


JOptionPane.showMessageDialog(new JFrame(), "Contact Created
Successfully",

                                "Contact
Created", JOptionPane.PLAIN_MESSAGE);

                                setVisible(false);
                                dispose();

                                } else {

JOptionPane.showMessageDialog(new JFrame(), "This Telephone Number
is Already in Use",

                                "Error",
JOptionPane.ERROR_MESSAGE);

                                }

                                } else if (telephoneNumber.equals(""))
{

JOptionPane.showMessageDialog(new JFrame(), "No Telephone Number
Entered", "Error",

JOptionPane.ERROR_MESSAGE);

                                } else {

JOptionPane.showMessageDialog(new JFrame(), "Only Numbers can be
Entered for the Telephone Number", "Error",
```

```
JOptionPane.ERROR_MESSAGE);
    }

    } else {
        JOptionPane.showMessageDialog(new
JFrame(), "Telephone Number must be 11 or less digits",
        "Error",
JOptionPane.ERROR_MESSAGE);
    }
    } else {
        JOptionPane.showMessageDialog(new JFrame(),
"Missing Parameters", "Error",
        JOptionPane.ERROR_MESSAGE);
    }
    }

    });
    btnAddContact.setBounds(147, 241, 144, 23);
    contentPane.add(btnAddContact);
}

/**
 * Checks the input for the telephone number to ensure that only
numbers have
 * been inputed.
 *
 * @param telephoneNumber Passes in the telephone number to be
checked.
 * @return A Boolean value whether the telephone number is valid or
not.
 */
public static boolean checkDigits(String telephoneNumber) {
    String regex = "[0-9]+";
    Pattern pattern = Pattern.compile(regex);
```



```
        if (telephoneNumber == null) {  
            return false;  
        }  
  
        Matcher matcher = pattern.matcher(telephoneNumber);  
        return matcher.matches();  
    }  
}
```

6.9. Appendix Nine: EditContactUserInterface.java

```
package stirring.cscu9yw.userinterfaces;

import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.border.EmptyBorder;

import stirring.cscu9yw.main.Requests;

import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JButton;
import javax.swing.JTextField;
import java.awt.Font;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;

/**
 *
 * This user interface is displayed when the edit contact button has been
 * clicked on the main menu user interface.
 *
 * @author 2636157
 *
 */
public class EditContactUserInterface extends JFrame {

    private JPanel contentPane;
    private JTextField textFieldTelephoneNumber;
    private JTextField textFieldPostcode;
    private JTextField textFieldTown;
    private JTextField textFieldStreet;
```

```
private JTextField textFieldSurname;
private JTextField textFieldFirstName;

private String firstName;
private String surname;
private String street;
private String town;
private String postcode;
private String telephoneNumber;

/**
 * Creates the frame to be displayed.
 */
public EditContactUserInterface() {
    setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
    setBounds(100, 100, 380, 400);
    contentPane = new JPanel();
    contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
    setContentPane(contentPane);
    contentPane.setLayout(null);

    JLabel lblFirstName = new JLabel("First Name");
    lblFirstName.setBounds(10, 43, 124, 20);
    contentPane.add(lblFirstName);

    JLabel lblSurname = new JLabel("Surname");
    lblSurname.setBounds(10, 74, 124, 20);
    contentPane.add(lblSurname);

    JLabel lblStreet = new JLabel("Street");
    lblStreet.setBounds(10, 106, 124, 20);
    contentPane.add(lblStreet);
```

```
JLabel lblTown = new JLabel("Town");
lblTown.setBounds(10, 137, 124, 20);
contentPane.add(lblTown);

JLabel lblPostcode = new JLabel("Postcode");
lblPostcode.setBounds(10, 168, 124, 20);
contentPane.add(lblPostcode);

JLabel lblTelephoneNumber = new JLabel("Telephone Number");
lblTelephoneNumber.setBounds(10, 199, 124, 20);
contentPane.add(lblTelephoneNumber);

JButton btnEditContact = new JButton("Edit Contact");
btnEditContact.addActionListener(new ActionListener() {

    /**
     * Carries out the actions when the edit contact button
    is clicked, each text
     * field is set to the values stored in the relevant
    String. It carries out
     * error handling and send outs a message to the screen
    if one of the errors
     * occurs
     */
    public void actionPerformed(ActionEvent e) {

        firstName =
textFieldFirstName.getText().toString();
        surname = textFieldSurname.getText().toString();
        street = textFieldStreet.getText().toString();
        town = textFieldTown.getText().toString();
        postcode = textFieldPostcode.getText().toString();
```

Student ID Number  
263157

```
        telephoneNumber =  
textFieldTelephoneNumber.getText().toString();  
  
        if (!firstName.equals("") && !surname.equals("")  
&& !street.equals("") && !town.equals("")  
        && !postcode.equals("") &&  
!telephoneNumber.equals("")) {  
  
            Requests.put(firstName, surname, street,  
town, postcode, telephoneNumber);  
  
            JOptionPane.showMessageDialog(new JFrame(),  
"Contact Edited Successfully", "Contact Edited",  
JOptionPane.PLAIN_MESSAGE);  
  
            setVisible(false);  
            dispose();  
        } else {  
            JOptionPane.showMessageDialog(new JFrame(),  
"Missing Parameters", "Error",  
JOptionPane.ERROR_MESSAGE);  
        }  
    }  
});  
btnEditContact.setBounds(136, 230, 144, 23);  
contentPane.add(btnEditContact);  
  
textFieldTelephoneNumber = new JTextField();  
textFieldTelephoneNumber.setEditable(false);  
textFieldTelephoneNumber.setColumns(10);  
textFieldTelephoneNumber.setBounds(136, 199, 178, 20);  
contentPane.add(textFieldTelephoneNumber);  
  
textFieldPostcode = new JTextField();
```

```
textFieldPostcode.setColumns(10);
textFieldPostcode.setBounds(136, 168, 178, 20);
contentPane.add(textFieldPostcode);

textFieldTown = new JTextField();
textFieldTown.setColumns(10);
textFieldTown.setBounds(136, 137, 178, 20);
contentPane.add(textFieldTown);

textFieldStreet = new JTextField();
textFieldStreet.setColumns(10);
textFieldStreet.setBounds(136, 106, 178, 20);
contentPane.add(textFieldStreet);

textFieldSurname = new JTextField();
textFieldSurname.setColumns(10);
textFieldSurname.setBounds(136, 74, 178, 20);
contentPane.add(textFieldSurname);

textFieldFirstName = new JTextField();
textFieldFirstName.setColumns(10);
textFieldFirstName.setBounds(136, 43, 178, 20);
contentPane.add(textFieldFirstName);

JLabel lblNewLabel = new JLabel("Edit Contact");
lblNewLabel.setFont(new Font("Tahoma", Font.PLAIN, 17));
lblNewLabel.setBounds(10, 0, 100, 35);
contentPane.add(lblNewLabel);

}

/**
```

```
* Sets the contact details for the text fields relevant to the
contact that has
* been searched for. Each text field calls the get method from the
Requests
* class, passing in the telephone number and the entity of what
needs to be
* displayed in the corresponding text field.
*
* @param telephoneNumberToSearch Passes in the telephone number
value for the
*
* contact that will be displayed to
be edited.
*/
public void setContactDetails(String telephoneNumberToSearch) {

    textFieldFirstName.setText(Requests.get(telephoneNumberToSearch,
"firstName").toString());

    textFieldSurname.setText(Requests.get(telephoneNumberToSearch,
"surname").toString());

    textFieldStreet.setText(Requests.get(telephoneNumberToSearch,
"street"));

    textFieldTown.setText(Requests.get(telephoneNumberToSearch,
"town").toString());

    textFieldPostcode.setText(Requests.get(telephoneNumberToSearch,
"postcode").toString());

    textFieldTelephoneNumber.setText(Requests.get(telephoneNumberToSearch,
"telephoneNumber").toString());
}
}
```

6.10. Appendix Nine: SingleContactUserInterface.java

```
package stirring.cscu9yw.userinterfaces;

import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.border.EmptyBorder;

import stirring.cscu9yw.main.Requests;

import javax.swing.JLabel;
import java.awt.Font;

/**
 *
 * This user interface is displayed when the new contact button has been
 * clicked
 * on the main menu user interface.
 *
 * @author 2636157
 *
 */
public class SingleContactUserInterface extends JFrame {

    private JPanel contentPane;

    private JLabel lblFirstName;
    private JLabel lblSurname;
    private JLabel lblStreet;
    private JLabel lblTown;
    private JLabel lblPostcode;
    private JLabel lblTelephoneNumber;
```



```
/**
 * Create the frame.
 */
public SingleContactUserInterface() {
    setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
    setBounds(100, 100, 380, 400);
    contentPane = new JPanel();
    contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
    setContentPane(contentPane);
    contentPane.setLayout(null);

    JLabel lblContact = new JLabel("Contact");
    lblContact.setFont(new Font("Tahoma", Font.PLAIN, 17));
    lblContact.setBounds(10, 11, 100, 35);
    contentPane.add(lblContact);

    JLabel lbl1 = new JLabel("First Name");
    lbl1.setBounds(10, 54, 124, 20);
    contentPane.add(lbl1);

    JLabel lbl2 = new JLabel("Surname");
    lbl2.setBounds(10, 85, 124, 20);
    contentPane.add(lbl2);

    JLabel lbl3 = new JLabel("Street");
    lbl3.setBounds(10, 117, 124, 20);
    contentPane.add(lbl3);

    JLabel lbl4 = new JLabel("Town");
    lbl4.setBounds(10, 148, 124, 20);
    contentPane.add(lbl4);
}
```

Student ID Number  
263157

```
JLabel lbl5 = new JLabel("Postcode");  
lbl5.setBounds(10, 179, 124, 20);  
contentPane.add(lbl5);
```

```
JLabel lbl6 = new JLabel("Telephone Number");  
lbl6.setBounds(10, 210, 124, 20);  
contentPane.add(lbl6);
```

```
lblFirstName = new JLabel("First Name");  
lblFirstName.setBounds(144, 54, 124, 20);  
contentPane.add(lblFirstName);
```

```
lblSurname = new JLabel("Surname");  
lblSurname.setBounds(144, 85, 124, 20);  
contentPane.add(lblSurname);
```

```
lblStreet = new JLabel("Street");  
lblStreet.setBounds(144, 117, 124, 20);  
contentPane.add(lblStreet);
```

```
lblTown = new JLabel("Town");  
lblTown.setBounds(144, 148, 124, 20);  
contentPane.add(lblTown);
```

```
lblPostcode = new JLabel("Postcode");  
lblPostcode.setBounds(144, 179, 124, 20);  
contentPane.add(lblPostcode);
```

```
lblTelephoneNumber = new JLabel("Telephone Number");  
lblTelephoneNumber.setBounds(144, 210, 124, 20);  
contentPane.add(lblTelephoneNumber);
```

```
}
```

```
/**
 * This method sets all of the labels displayed to the contact
information that
 * has been searched for. It calls the get method from the Requests
calls,
 * passing through the telephone number to be searched along with
the single
 * piece of data that is to be displayed.
 *
 * @param telephoneNumberToSearch Passes in the telephone number to
be searched.
 */
public void getContactDetails(String telephoneNumberToSearch) {
    lblFirstName.setText(Requests.get(telephoneNumberToSearch,
"firstName").toString());
    lblSurname.setText(Requests.get(telephoneNumberToSearch,
"surname").toString());
    lblStreet.setText(Requests.get(telephoneNumberToSearch,
"street"));
    lblTown.setText(Requests.get(telephoneNumberToSearch,
"town").toString());
    lblPostcode.setText(Requests.get(telephoneNumberToSearch,
"postcode").toString());

    lblTelephoneNumber.setText(Requests.get(telephoneNumberToSearch,
"telephoneNumber").toString());
}
}
```

#### 6.11. Application.properties

```
spring.datasource.url=jdbc:mysql://localhost:3306/contactsDB
spring.datasource.username=root
spring.datasource.password=root1
spring.jpa.hibernate.naming.physical-
strategy=org.hibernate.boot.model.naming.PhysicalNamingStrategyStandardImpl
```