

Name: Adam Martin

Student Number: 11021206

Module: Object Oriented and Web Programming

Assignment: Report - Java Employee RESTful Web Service Coursework

Annotated Marking Grid:

Adam Martin
11021206

✓ = Completed

Mark Scheme

Your work in this assignment will be graded in a number of areas, attracting a number of marks for each. Guidance on the assessment criteria for each area is included below. The marks given for each area will be combined to give an overall mark for 1CWK50, which is worth 50% of the final unit mark.

Base Classes		Database Connectivity		RESTful Web Service		Advanced Features		Code Quality/Report	
No/little attempt at producing base classes	0-4 marks	No/little attempt at creating the SQLite database and the EmployeeDAO class	0-4 marks	No/non-functioning attempt at a server	0-3 marks	No/little attempt at advanced features	0-1 marks	Code is poorly or moderately presented, with some or no documentation or comments.	0-3 marks
Evidence of an attempt at base classes Person and Controller	5-6 marks	CRUD methods partially implemented	5-9 marks	Server connects to the SQLite database and RESTful route for reading all employee information and output in JSON format	4-10 marks	Validation (using regular expressions) of all fields before inserting new records into database (e.g. name, gender, email, postcode, start date, department, salary)	1-10 marks	Code is well presented, with good Javadoc documentation and sensible comments	4-6 marks
Base classes Person, Employee and Controller partially implemented.	10-14 marks	CRUD methods fully implemented	10-14 marks	RESTful route to create employee record by posting JSON object	11-15 marks	Access token code partially implemented	10-12 marks	Report containing screenshots of completed application	7-15 marks
				RESTful routes to update and delete employee records	16-20 marks				
				RESTful route to retrieve an employee record based on employee ID	21-25 marks				
Base classes fully implemented and tested	15-20 marks	All CRUD tested using a class called "DatabaseTester"	15-20 marks	All services tested using a class called "WebServiceTester"	25-30 marks	Security access fully implemented (all routes)	12-15 marks		

If you need any help in understanding this assignment please arrange to see Dr Alan Crispin during his office hours.
Dr Alan Crispin (14/10/17)

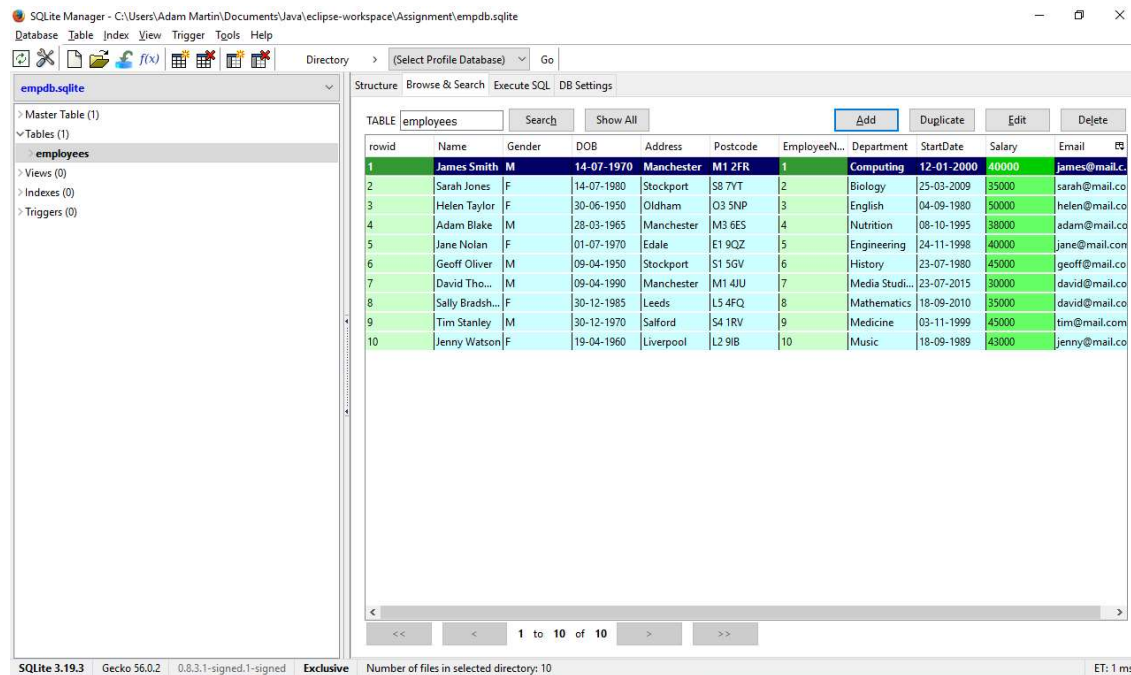
Report – Java Employee RESTful Web Service Coursework

Introduction:

This report will use screenshots and accompanied descriptions to display and describe the functionality of my Java Employee RESTful Web Service project, and the steps taken to implement this functionality. The aim of the project was to develop a Java RESTful web service, which sent requests to a back-end SQLite database of employees.

Functionality, and Steps Taken to Achieve Functionality:

Step 1 – Creation of the ‘Employees’ SQLite Database:



The screenshot shows the SQLite Manager interface with the 'employees' table selected. The table contains 10 records with columns: rowid, Name, Gender, DOB, Address, Postcode, EmployeeN..., Department, StartDate, Salary, and Email. The data is as follows:

rowid	Name	Gender	DOB	Address	Postcode	EmployeeN...	Department	StartDate	Salary	Email
1	James Smith	M	14-07-1970	Manchester	M1 2FR	1	Computing	12-01-2000	40000	james@mail.c...
2	Sarah Jones	F	14-07-1980	Stockport	S8 7YT	2	Biology	25-03-2009	35000	sarah@mail.co
3	Helen Taylor	F	30-06-1950	Oldham	O3 5NP	3	English	04-09-1980	50000	helen@mail.co
4	Adam Blake	M	28-03-1965	Manchester	M3 6ES	4	Nutrition	08-10-1995	38000	adam@mail.co
5	Jane Nolan	F	01-07-1970	Edale	E1 9QZ	5	Engineering	24-11-1998	40000	jane@mail.com
6	Geoff Oliver	M	09-04-1950	Stockport	S1 5GV	6	History	23-07-1980	45000	geoff@mail.co
7	David Tho...	M	09-04-1990	Manchester	M1 4JU	7	Media Studi...	23-07-2015	30000	david@mail.co
8	Sally Bradsh...	F	30-12-1985	Leeds	L5 4FQ	8	Mathematics	18-09-2010	35000	david@mail.co
9	Tim Stanley	M	30-12-1970	Salford	S4 1RV	9	Medicine	03-11-1999	45000	tim@mail.com
10	Jenny Watson	F	19-04-1960	Liverpool	L2 9JB	10	Music	18-09-1989	43000	jenny@mail.co

The first step was to create the database, using the Firefox ‘SQLite Manager’ tool. The database contains a single ‘Employees’ table, populated with 10 fictitious records. In this scenario, the fictitious people are employees of a local university.

Step 2 – 'Person' Class:

```
eclipse-workspace - Assignment/src/Person.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help

Person.java x
1
2 /**
3  * The Person public base class establishes the attributes of a person (name, gender, date of birth, address, postcode), and
4  * sets and gets their values. All of the attributes are of type String. These attributes are used in the 'Employee'
5  * class.
6  *
7  * @author Adam Martin
8  * @version 1.0
9  *
10 */
11
12 public class Person {
13
14     // Person attributes (name, gender, date of birth, address, and postcode)
15     private String name;
16     private String gender;
17     private String dob;
18     private String address;
19     private String postcode;
20
21
22     /**
23      * The Person public constructor constructs a person using the attributes established above. A person has a name,
24      * gender, date of birth, address and postcode.
25      *
26      * @param newName, newGender, newDOB, newAddress, newPostcode
27      *
28      * @author Adam Martin
29      * @version 1.0
30      */
31
32     // Person 'Constructor' - constructs a Person using attributes established above
33     public Person(String newName, String newGender, String newDOB, String newAddress, String newPostcode)
34     {
35         // Initialization of Person attributes
36         this.name = newName;
37         this.gender = newGender;
38         this.dob = newDOB;
39         this.address = newAddress;
40         this.postcode = newPostcode;
41     }
42 }
```

```
43
44 // 'Setter and Getter' methods to 'set' and 'get' person attributes
45
46 /**
47  * This person 'setter' sets the name of the person
48  *
49  * @param newName
50  *
51  * @author Adam Martin
52  * @version 1.0
53  */
54
55 // Sets the name
56 public void setName(String newName)
57 {
58     this.name = newName;
59 }
60
61
62 /**
63  * This person 'getter' gets the name of the person
64  *
65  * @return the person's name
66  *
67  * @author Adam Martin
68  * @version 1.0
69  */
70
71 // Gets the name
72 public String getName()
73 {
74     return this.name;
75 }
76
77
```

```
/**
 * This person 'setter' sets the gender of the person
 *
 * @param newGender
 *
 * @author Adam Martin
 * @version 1.0
 */
// Sets the gender
public void setGender(String newGender)
{
    this.gender = newGender;
}

/**
 * This person 'getter' gets the gender of the person
 *
 * @return the person's name
 *
 * @author Adam Martin
 * @version 1.0
 */
// Gets the gender
public String getGender()
{
    return this.gender;
}
```

```
112  * This person 'setter' sets the date of birth of the person
113  *
114  * @param newDOB
115  *
116  * @author Adam Martin
117  * @version 1.0
118  */
119
120  // Sets the date of birth
121  public void setDOB(String newDOB)
122  {
123      this.dob = newDOB;
124  }
125
126
127  /**
128   * This person 'getter' gets the date of birth of the person
129   *
130   * @return the person's name
131   *
132   * @author Adam Martin
133   * @version 1.0
134   */
135
136  // Gets the date of birth
137  public String getDOB()
138  {
139      return this.dob;
140  }
141
```

```
143  /**
144   * This person 'setter' sets the address of the person
145   *
146   * @param newAddress
147   *
148   * @author Adam Martin
149   * @version 1.0
150   */
151
152  // Sets the address
153  public void setAddress(String newAddress)
154  {
155      this.address = newAddress;
156  }
157
158
159  /**
160   * This person 'getter' gets the address of the person
161   *
162   * @return the person's name
163   *
164   * @author Adam Martin
165   * @version 1.0
166   */
167
168  // Gets the address
169  public String getAddress()
170  {
171      return this.address;
172  }
173
```

```
174
175  /**
176   * This person 'setter' sets the postcode of the person
177   *
178   * @param newPostcode
179   *
180   * @author Adam Martin
181   * @version 1.0
182   */
183
184  // Sets the postcode
185  public void setPostcode(String newPostcode)
186  {
187      this.postcode = newPostcode;
188  }
189
190
191  /**
192   * This person 'getter' gets the postcode of the person
193   *
194   * @return the person's name
195   *
196   * @author Adam Martin
197   * @version 1.0
198   */
199
200  // Gets the postcode
201  public String getPostcode()
202  {
203      return this.postcode;
204  }
205  }
206
```

The person base class sets the attributes of a person (name, gender, date of birth, address, postcode), which are all of type string. The class also implements 'setter' and 'getter' methods to set and get the person's values. This class underpins the 'Employee' class, which inherits its values.

Step 3 – 'Employee' Class:

```

1 | Employee.java X
2 |
3 | /**
4 |  * The Employee public base class 'extends' the person class, and therefore inherits from the Person class. This
5 |  * inheritance relationship states that an Employee 'is a' Person. The class establishes the attributes of an employee
6 |  * (employee number, department, start date, salary, email), and sets and gets their values. All of the attributes are
7 |  * of type String, apart from employee number (int) and salary (float).
8 |  *
9 |  * @author Adam Martin
10 |  * @version 1.0
11 |  */
12 |
13 | public class Employee extends Person {
14 |
15 |     // Employee attributes (employee number, department, start date, salary, email)
16 |     private int employeeNumber;
17 |     private String department;
18 |     private String startDate;
19 |     private float salary;
20 |     private String email;
21 |
22 |
23 |     /**
24 |     * The Employee public constructor constructs a person using the attributes established above, and the attributes of
25 |     * the 'Person' class, which this class inherits from. The 'super' class is used to obtain the Person values and establish
26 |     * them as 'Employee' values also. An employee has a name, gender, date of birth, address, postcode, employee number,
27 |     * department, start date, salary and email address.
28 |     *
29 |     * @param name, gender, dob, address, postcode, newEmployeeNumber, newDepartment, newStartDate, newSalary, newEmail
30 |     *
31 |     * @author Adam Martin
32 |     * @version 1.0
33 |     */
34 |
35 |     // Employee 'Constructor' - Constructs an Employee using 'Person' + 'Employee' values (name, gender, date of birth, address, postcode, employee number, de
36 |     public Employee(String name, String gender, String dob, String address, String postcode,
37 |                     int newEmployeeNumber, String newDepartment, String newStartDate, float newSalary, String newEmail)
38 |     {
39 |         // 'super' class is called to tell java to use the 'Person' attributes in the 'Employee'
40 |         super(name, gender, dob, address, postcode);
41 |

```

```

42 |         // Initialization of Employee attributes
43 |         this.employeeNumber = newEmployeeNumber;
44 |         this.department = newDepartment;
45 |         this.startDate = newStartDate;
46 |         this.salary = newSalary;
47 |         this.email = newEmail;
48 |     }
49 |
50 |     // 'Setter and Getter' methods to get and return values
51 |
52 |
53 |     /**
54 |     * This employee 'setter' sets the employee's employee number number.
55 |     *
56 |     * @param newEmployeeNumber
57 |     *
58 |     * @author Adam Martin
59 |     * @version 1.0
60 |     */
61 |
62 |     // Sets the employee number
63 |     public void setEmployeeNumber (int newEmployeeNumber)
64 |     {
65 |         this.employeeNumber = newEmployeeNumber;
66 |     }
67 |
68 |
69 |     /**
70 |     * This employee 'getter' gets the employee number of the employee
71 |     *
72 |     * @return the employee's employee number
73 |     *
74 |     * @author Adam Martin
75 |     * @version 1.0
76 |     */
77 |

```

```

78 |
79 |     /**
80 |     * This employee 'getter' gets the employee's salary
81 |     *
82 |     * @return the employee's salary
83 |     *
84 |     * @author Adam Martin
85 |     * @version 1.0
86 |     */
87 |
88 |     // Gets the salary
89 |     public float getSalary()
90 |     {
91 |         return this.salary;
92 |     }
93 |
94 |     /**
95 |     * This employee 'setter' sets the employee's email address
96 |     *
97 |     * @param newEmail
98 |     *
99 |     * @author Adam Martin
100 |    * @version 1.0
101 |    */
102 |
103 |    // Sets the email
104 |    public void setEmail(String newEmail)
105 |    {
106 |        this.email = newEmail;
107 |    }
108 |
109 |    // Gets the department
110 |    public String getDepartment()
111 |    {
112 |        return this.department;
113 |    }
114 |
115 |
116 |

```



```

118  /**
119   * This employee 'setter' sets the employee's start date
120   *
121   * @param newStartDate
122   *
123   * @author Adam Martin
124   * @version 1.0
125   */
126
127  // Sets the start date
128  public void setStartDate(String newStartDate)
129  {
130      this.startDate = newStartDate;
131  }
132
133
134  /**
135   * This employee 'getter' gets the employee's start date
136   *
137   * @return the employee's start date
138   *
139   * @author Adam Martin
140   * @version 1.0
141   */
142
143  // Gets the start date
144  public String getStartDate()
145  {
146      return this.startDate;
147  }
148
149
150  /**
151   * This employee 'setter' sets the employee's salary
152   *
153   * @param newSalary
154   *
155   * @author Adam Martin
156   * @version 1.0
157   */

```

```

158
159  // Sets the salary
160  public void setSalary(float newSalary)
161  {
162      this.salary = newSalary;
163  }
164
165
166  /**
167   * This employee 'getter' gets the employee's salary
168   *
169   * @return the employee's salary
170   *
171   * @author Adam Martin
172   * @version 1.0
173   */
174
175  // Gets the salary
176  public float getSalary()
177  {
178      return this.salary;
179  }
180
181
182  /**
183   * This employee 'setter' sets the employee's email address
184   *
185   * @param newEmail
186   *
187   * @author Adam Martin
188   * @version 1.0
189   */
190
191  // Sets the email
192  public void setEmail(String newEmail)
193  {
194      this.email = newEmail;
195  }
196

```

```

197
198  /**
199   * This employee 'getter' gets the employee's email address
200   *
201   * @return the employee's email address
202   *
203   * @author Adam Martin
204   * @version 1.0
205   */
206
207  // Gets the email
208  public String getEmail()
209  {
210      return this.email;
211  }
212
213
214
215
216  /**
217   * This toString() method puts all of the 'Employee' attributes in a string, to avoid the Employee being omitted
218   * as byte code. The 'super' class is used when getting Person attributes to call them from the 'Person' class.
219   *
220   * @author Adam Martin
221   * @version 1.0
222   *
223   */
224
225
226  public String toString()
227  {
228      return "Employee Name: "+super.getName()+" Gender: "+super.getGender()+
229             " Date of Birth: "+super.getDOB()+" Address: "+super.getAddress()+
230             " Postcode: "+super.getPostcode()+" Employee Number "+this.getEmployeeNumber()+
231             " Department "+this.getDepartment()+" Start Date: "+this.getStartDate()+
232             " Salary: "+this.getSalary()+" Email: "+this.getEmail();
233  }
234
235
236 }
237

```

The employee base class 'extends' the person class, to inherit its attributes. The class contains setter and getter methods for employee-specific attributes (employee number, department, start date, salary email). These attributes are placed in a toString method, in order to print them to the console in a readable text format. This successful implementation of person and employee base classes allows to program to manipulate database records.

Step 3 – 'EmployeeDAO' Class:

```

EmployeeDAO.java
21 public class EmployeeDAO {
22
23
24 /**
25  * The public getDBConnection class connects to the JDBC driver, and then to the 'empdb' SQLite database file. This enables
26  * the program to manipulate the data in this database. If the connection is successful, the method returns the
27  * database connection. Otherwise, the method throws an SQLException.
28  *
29  * @return dbConnection
30  * @throws SQLException
31  *
32  * @author Adam Martin
33  * @version 1.0
34  */
35
36 // Method for getting the database connection
37 public Connection getDBConnection() {
38
39     Connection dbConnection = null;
40
41     try {
42         Class.forName("org.sqlite.JDBC"); // Links to the JDBC driver file
43     } catch (ClassNotFoundException e) {
44         System.out.println(e.getMessage());
45     }
46     try {
47         String dbURL = ("jdbc:sqlite:empdb.sqlite"); // Links the program to the 'empdb' database file
48         dbConnection = DriverManager.getConnection(dbURL);
49         return dbConnection;
50     } catch (SQLException e) {
51         System.out.println(e.getMessage());
52     }
53     return dbConnection;
54 }
55
56

```

```

71 // Method for getting a list of all employees in the database
72 public ArrayList<Employee> getAllEmployees() throws SQLException {
73
74     Connection dbConnection = null;
75     Statement statement = null;
76     ResultSet resultset = null;
77     String query = "SELECT * FROM employees;"; // Selects everything from the 'employees' database table
78     Employee tempEmp = null;
79
80     ArrayList<Employee> allEmployees = new ArrayList<>(); // Creates an 'Array List' (an array with an undefined length) for the database records
81
82     try {
83         // Opens the database connection by calling the getDBConnection method
84         dbConnection = getDBConnection();
85         System.out.println("-----");
86         System.out.println("Get All Employees - Database successfully opened");
87
88         // Executes the SQL statement
89         statement = dbConnection.createStatement();
90         System.out.println("SQL Statement: " + query);
91         resultset = statement.executeQuery(query);
92
93         // Adds the table columns to the result set
94         while (resultset.next()) {
95             String name = resultset.getString("name");
96             String gender = resultset.getString("gender");
97             String dob = resultset.getString("dob");
98             String address = resultset.getString("address");
99             String postcode = resultset.getString("postcode");
100
101             int employeeNumber = resultset.getInt("employeeNumber");
102             String department = resultset.getString("department");
103             String startDate = resultset.getString("startDate");
104             float salary = resultset.getFloat("Salary");
105             String email = resultset.getString("email");
106
107             tempEmp = new Employee(name, gender, dob, address, postcode, employeeNumber,
108                                   department, startDate, salary, email);
109             allEmployees.add(tempEmp);
110         }
111

```

```

113 // Catches any SQL exceptions and closes the result set
114 } catch (SQLException e) {
115     System.out.println(e.getMessage());
116 } finally {
117     if (resultset != null) {
118         resultset.close();
119     }
120     if (statement != null) {
121         statement.close();
122     }
123     if (dbConnection != null) {
124         dbConnection.close();
125     }
126 }
127
128 // Console output for users
129 System.out.println("Get All Employees - Records successfully located");
130 System.out.println("-----");
131 System.out.println();
132 System.out.println("Get All Employees - All Employee Records:");
133 System.out.println();
134
135 return allEmployees;
136
137 }
138

```


Adam Martin 11021206 - Java Assignment Report

```
154 // Method for getting a single employee record from the database based on a given ID (or 'Employee Number')
155 public Employee getEmployee(String id) throws SQLException {
156     Connection connection = null;
157     Statement statement = null;
158     Employee tempEmp = null;
159     String query = "SELECT * FROM employees WHERE EmployeeNumber = '" + id + "'"; // Gets an employee based a given Employee Number (or 'id')
160     ResultSet resultSet = null;
161
162     try { // Opens the database connection by calling the getDBConnection method
163         connection = getDBConnection();
164         connection.setAutoCommit(false);
165
166         // Console output for users
167         System.out.println();
168         System.out.println("Get Employee - Database successfully opened");
169
170         // Executes the SQL statement
171         statement = connection.createStatement();
172         System.out.println("SQL Statement: " + query);
173         System.out.println("Get Employee - Record successfully located");
174         resultSet = statement.executeQuery(query);
175
176         // Add the table columns to the result set
177         String name = resultSet.getString("Name");
178         String gender = resultSet.getString("Gender");
179         String dob = resultSet.getString("DOB");
180         String address = resultSet.getString("Address");
181         String postcode = resultSet.getString("Postcode");
182         int employeeNumber = resultSet.getInt("EmployeeNumber");
183         String department = resultSet.getString("Department");
184         String startDate = resultSet.getString("StartDate");
185         float salary = resultSet.getFloat("Salary");
186         String email = resultSet.getString("Email");
187
188         tempEmp = new Employee(name, gender, dob, address, postcode, employeeNumber,
189                               department, startDate, salary, email);
190
191         // Prints the chosen Employee record to the console
192         System.out.println("Employee Record: " + tempEmp);
193     }
194 }
```

```
101 // Closes the result set and catches any exceptions
102 resultSet.close();
103 statement.close();
104 connection.commit();
105 connection.close();
106
107 } catch ( Exception e ) {
108     System.err.println( e.getClass().getName() + ": " + e.getMessage() );
109     System.exit(0);
110 }
111
112 return tempEmp;
113 }
114 }
```

```
115 // Method for inserting an employee record into the database
116 public boolean insertEmployee(Employee emp) throws SQLException {
117     Boolean insertResult = false;
118     Connection connection = null;
119     Statement statement = null;
120     String query = "INSERT INTO employees (Name, Gender, DOB, Address, Postcode, EmployeeNumber, Department, StartDate, Salary, Email) "
121                   + "VALUES ('" + emp.getName() + "','" + emp.getGender() + "','" + emp.getDOB() + "','" + emp.getAddress() + "','" +
122                   + emp.getPostcode() + "','" + emp.getEmployeeNumber() + "','" + emp.getDepartment() + "','" + emp.getStartDate() + "','" +
123                   + emp.getSalary() + "','" + emp.getEmail() + "')"; // Inserts an Employee by calling 'getters' created in Person and Employee classes
124
125     try {
126         // Opens the database connection by calling the getDBConnection method
127         connection = getDBConnection();
128         connection.setAutoCommit(false);
129         System.out.println();
130         System.out.println("Insert Employee - Database successfully opened");
131
132         // Executes the SQL statement
133         statement = connection.createStatement();
134         System.out.println("SQL Statement: " + query);
135         statement.executeUpdate(query);
136
137         // Closes the result set and catches any exceptions
138         statement.close();
139         connection.commit();
140         connection.close();
141
142     } catch ( Exception ex ) {
143         System.err.println( ex.getClass().getName() + ": " + ex.getMessage() );
144         System.exit(0);
145     }
146
147     // Console output for users
148     System.out.println("Insert Employee - Records successfully created");
149     System.out.println();
150
151     return insertResult;
152 }
153 }
```

```

388 // Method for updating an employee record in the database
389 public Boolean updateEmployee(Employee emp) throws SQLException {
390 {
391 Boolean updateResult = false;
392 Connection connection = null;
393 Statement statement = null;
394 String query = "UPDATE employees SET Name = '"+emp.getName()+"', Gender = '"+emp.getGender()+"', "
395 + "DOB = '"+emp.getDOB()+"', Address = '"+emp.getAddress()+"', Postcode = '"+emp.getPostcode()+"', "
396 + "Department = '"+emp.getDepartment()+"', StartDate = '"+emp.getStartDate()+"', "
397 + "Salary = '"+emp.getSalary()+"', Email = '"+emp.getEmail()+"' "
398 + "WHERE EmployeeNumber = '" + emp.getEmployeeNumber() + "'"; // Updates an employee assigned to a certain employee number. Calls 'getters'
399
400 try {
401 // Opens the database connection by calling the getDBConnection method
402 connection = getDBConnection();
403 connection.setAutoCommit(false);
404
405 // Console output for users
406 System.out.println();
407 System.out.println("-----");
408 System.out.println("Update Employee - Database successfully opened");
409
410 // Executes the SQL statement
411 statement = connection.createStatement();
412 System.out.println("SQL Statement: "+query);
413 statement.executeUpdate(query);
414
415 // Closes the result set and catches any exceptions
416 connection.commit();
417 statement.close();
418 connection.close();
419
420 } catch (Exception e) {
421 System.err.println( e.getClass().getName() + ": " + e.getMessage() );
422 System.exit(0);
423 }
424
425 // Console output for users
426 System.out.println("Update Employee - Records successfully updated");
427 System.out.println("-----");
428
429 return updateResult;
430 }
431 }
432
433

```

```

349 // Method for deleting an employee record from the database
350 public Boolean deleteEmployee(String id) throws SQLException {
351 {
352 Boolean deleteResult = false;
353 Connection connection = null;
354 Statement statement = null;
355 String query = "DELETE FROM employees WHERE EmployeeNumber = '" + id + "'"; // Deletes an employee with a given employee number (or 'id')
356
357 try {
358 // Opens the database connection by calling the getDBConnection method
359 connection = getDBConnection();
360 connection.setAutoCommit(false);
361
362 // Console output for users
363 System.out.println();
364 System.out.println("-----");
365 System.out.println("Delete Employee - Database successfully opened");
366
367 // Executes the SQL statement
368 statement = connection.createStatement();
369 System.out.println("SQL Statement: "+query);
370 statement.executeUpdate(query);
371
372 // Closes the result set and catches any exceptions
373 connection.commit();
374 statement.close();
375 connection.close();
376
377 } catch (Exception e) {
378 System.err.println( e.getClass().getName() + ": " + e.getMessage() );
379 System.exit(0);
380 }
381
382 // Console output for users
383 System.out.println("Delete Employee - Records successfully deleted");
384 System.out.println("-----");
385
386 return deleteResult;
387 }
388 }
389
390 }

```

The EmployeeDAO class contains a method for connecting to the database, and the CRUD methods for getting all employees, getting a single employee, inserting an employee, updating an employee and deleting an employee. The CRUD operations are actioned using a range of SQL statements. This class is crucial to the application, as its CRUD methods are called throughout the program.

Step 4 – 'DatabaseTester' Class:

```

16 public class DatabaseTester {
17
18
19 /**
20  * The main method within this class calls all methods from the EmployeeDAO class, and prints their results to the
21  * console screen. The parameters from the EmployeeDAO method are passed here, in order to test their
22  * functionality.
23  *
24  * @author Adam Martin
25  * @version 1.0
26  *
27  */
28
29 public static void main(String[] args) {
30
31     // Calls getAllEmployees method from EmployeeDAO and prints results to the console screen
32     EmployeeDAO dao = new EmployeeDAO();
33     ArrayList<Employee> allEmployees = new ArrayList<>();
34
35     try {
36         allEmployees = dao.getAllEmployees(); // Calls the method from the EmployeeDAO
37     } catch (SQLException e) {
38         e.printStackTrace();
39     }
40
41     // Iterates through all employees using toString() method
42     for (Employee e : allEmployees) {
43         System.out.println(e); // Prints the array list of employees to the console screen
44     }
45
46
47     // Calls getEmployee method from EmployeeDAO and prints results to the console screen
48     EmployeeDAO dao2 = new EmployeeDAO();
49     Employee tempEmp = new Employee(null, null, null, null, null, 0, null, null, 0, null); // Constructs a temporary employee to hold values
50     String id1 = "5"; // Tells the program which employee number record to retrieve
51
52     try {
53         tempEmp = dao2.getEmployee(id1); // Retrieves the employee with the 'id' (or 'Employee Number') set above
54     } catch (SQLException e) {
55         e.printStackTrace();
56     }
57
58 }
59
60 }

```

```

65 // Calls insertEmployee method from EmployeeDAO
66 EmployeeDAO dao3 = new EmployeeDAO();
67 boolean insertResult = false;
68
69 // Creates a new employee object to insert into the database
70 Employee e = new Employee("Sam James", "M", "14-04-1968", "Stoke", "S1 2EQ", 11, "Economics",
71     "01-09-2012", 49800, "sam@mail.com");
72
73 try {
74     insertResult = dao3.insertEmployee(e); // Inserts the above-created employee object into the database
75 } catch (SQLException ex) {
76     ex.printStackTrace();
77 }
78
79
80 // Calls updateEmployee method from EmployeeDAO
81 EmployeeDAO dao3 = new EmployeeDAO();
82 boolean updateResult = false;
83
84 // Creates a temporary employee record to hold values, then updates the values of the given employee number record
85 Employee David = new Employee("David Thompson", "M", "09-04-1998", "Sheffield", "S3 2QM", 7, "Media Studies", "23-07-2015", 35000, "david@mail.com");
86
87 try {
88     dao.updateEmployee(David); // Executes the update
89 } catch (SQLException ex) {
90     ex.printStackTrace();
91 }
92
93
94 // Calls deleteEmployee method from EmployeeDAO
95 EmployeeDAO dao4 = new EmployeeDAO();
96 boolean deleteResult = false;
97 String id = "8"; // Tells the program which employee number record to delete
98
99 try {
100     deleteResult = dao4.deleteEmployee(id); // Executes the deletion
101 } catch (SQLException ex) {
102     ex.printStackTrace();
103 }
104
105 }
106
107 }
108
109 }
110
111 }

```

The DatabaseTester class calls the CRUD methods from the EmployeeDAO class, by passing in test data. The results of the test are printed to the console screen.

DatabaseTester – Console Output:

```

Console X
<terminated> DatabaseTester [Java Application] C:\Program Files\Java\jre1.8.0_151\bin\javaw.exe (14 Dec 2017, 12:54:54)
Get All Employees - Database successfully opened
SQL Statement: SELECT * FROM employees;
Get All Employees - Records successfully located

-----
Get All Employees - All Employee Records:

Employee Name: James Smith Gender: M Date of Birth: 14-07-1970 Address: Manchester Postcode: M1 2FR Employee Number 1 Department Computing Start Date: 12-01-2000 Salary: 40000.0 Email: james@mail.com
Employee Name: Sarah Jones Gender: F Date of Birth: 14-07-1980 Address: Stockport Postcode: S8 7YT Employee Number 2 Department Biology Start Date: 25-03-2009 Salary: 35000.0 Email: sarah@mail.com
Employee Name: Helen Taylor Gender: F Date of Birth: 30-06-1950 Address: Oldham Postcode: O3 5NP Employee Number 3 Department English Start Date: 04-09-1980 Salary: 50000.0 Email: helen@mail.com
Employee Name: Adam Blake Gender: M Date of Birth: 28-03-1965 Address: Manchester Postcode: M3 6ES Employee Number 4 Department Nutrition Start Date: 08-10-1995 Salary: 38000.0 Email: adam@mail.com
Employee Name: Jane Nolan Gender: F Date of Birth: 01-07-1970 Address: Edale Postcode: E1 9QZ Employee Number 5 Department Engineering Start Date: 24-11-1998 Salary: 40000.0 Email: jane@mail.com
Employee Name: Geoff Oliver Gender: M Date of Birth: 09-04-1950 Address: Stockport Postcode: S1 5GV Employee Number 6 Department History Start Date: 23-07-1980 Salary: 45000.0 Email: geoff@mail.com
Employee Name: David Thompson Gender: M Date of Birth: 09-04-1990 Address: Manchester Postcode: M1 4JU Employee Number 7 Department Media Studies Start Date: 23-07-2015 Salary: 35000.0 Email: david@mail.com
Employee Name: Sally Bradshaw Gender: F Date of Birth: 30-12-1985 Address: Leeds Postcode: L5 4FQ Employee Number 8 Department Mathematics Start Date: 18-09-2010 Salary: 35000.0 Email: david@mail.com
Employee Name: Tim Stanley Gender: M Date of Birth: 30-12-1970 Address: Salford Postcode: S4 1RW Employee Number 9 Department Medicine Start Date: 03-11-1999 Salary: 45000.0 Email: tim@mail.com
Employee Name: Jenny Watson Gender: F Date of Birth: 19-04-1960 Address: Liverpool Postcode: L2 9JB Employee Number 10 Department Music Start Date: 18-09-1989 Salary: 43000.0 Email: jenny@mail.com

-----
Get Employee - Database successfully opened
SQL Statement: SELECT * FROM employees WHERE EmployeeNumber = '5'
Get Employee - Record successfully located

-----
Get Employee by ID - Record:

Employee Record: Employee Name: Jane Nolan Gender: F Date of Birth: 01-07-1970 Address: Edale Postcode: E1 9QZ Employee Number 5 Department Engineering Start Date: 24-11-1998 Salary: 40000.0 Email: jane@mail.com

-----
Insert Employee - Database successfully opened
SQL Statement: INSERT INTO employees (Name, Gender, DOB, Address, Postcode, EmployeeNumber, Department, StartDate, Salary, Email) VALUES ('Sam James', 'M', '14-04-1960', 'Stoke', 'S1 2EQ', '11', 'Economics', '01-09-2012', '49000.0', 'sam@mail.com');
Insert Employee - Records successfully created

-----
Update Employee - Database successfully opened
SQL Statement: UPDATE employees SET Name = 'David Thompson', Gender = 'M', DOB = '09-04-1990', Address = 'Sheffield', Postcode = 'S3 2QW', Department = 'Media Studies', StartDate = '23-07-2015', Salary = 35000.0 WHERE EmployeeNumber = '7';
Update Employee - Records successfully updated

-----
Delete Employee - Database successfully opened
SQL Statement: DELETE FROM employees WHERE EmployeeNumber = '9'
Delete Employee - Records successfully deleted

```

DatabaseTester – Updated ‘Employees’ Database Table:

TABLE	employees	Search	Show All						Add	Duplicate	Edit
rowid	Name	Gender	DOB	Address	Postcode	EmployeeN...	Department	StartDate	Salary	Email	
1	James Smith	M	14-07-1970	Manchester	M1 2FR	1	Computing	12-01-2000	40000	james@mail.c...	
2	Sarah Jones	F	14-07-1980	Stockport	S8 7YT	2	Biology	25-03-2009	35000	sarah@mail.com	
3	Helen Taylor	F	30-06-1950	Oldham	O3 5NP	3	English	04-09-1980	50000	helen@mail.co...	
4	Adam Blake	M	28-03-1965	Manchester	M3 6ES	4	Nutrition	08-10-1995	38000	adam@mail.co...	
5	Jane Nolan	F	01-07-1970	Edale	E1 9QZ	5	Engineering	24-11-1998	40000	jane@mail.com	
6	Geoff Oliver	M	09-04-1950	Stockport	S1 5GV	6	History	23-07-1980	45000	geoff@mail.com	
7	David Tho...	M	09-04-1990	Sheffield	S3 2QW	7	Media Studi...	23-07-2015	35000	david@mail.co...	
8	Sally Bradsh...	F	30-12-1985	Leeds	L5 4FQ	8	Mathematics	18-09-2010	35000	david@mail.co...	
10	Jenny Watson	F	19-04-1960	Liverpool	L2 9IB	10	Music	18-09-1989	43000	jenny@mail.co...	
11	Sam James	M	14-04-1960	Stoke	S1 2EQ	11	Economics	01-09-2012	49000	sam@mail.com	

This screenshot depicts the ‘employees’ SQLite database table, after the DatabaseTester class was run in Eclipse. As you can see, all of the operations worked successfully. All employee records were printed to the console, and a single employee, with an employee number of ‘5’ was also printed to the console. As was stated in the test data, a new employee, ‘Sam James’, with an employee number of ‘11’ was inserted into the database. Also, the employee with an employee number of ‘7’, David Thompson, had his address, postcode and salary information updated (all other attributes can also be updated). As you can see in the table, the employee with an employee number of ‘9’ (Tim Stanley) is no longer present, as his record was successfully deleted from the database.

Step 5 – 'ControllerHttpServer' Class:

```
ControllerHttpServer.java X
16 public class ControllerHttpServer {
17
18
19 /**
20  * The main method within this class contains code for the creation of URL 'handlers' for each server task class.
21  * The class also contains code for starting the local host server, running on port 8014. When the server is running
22  * successfully, a confirmation message is outputted to the console screen. Otherwise, an IOException error
23  * message displays.
24  *
25  * @author Adam Martin
26  * @version 1.0
27  */
28
29
30 public static void main(String[] args) {
31
32     try {
33         // Create the server on port 8000
34         HttpServer server = HttpServer.create(new InetSocketAddress(8014), 0);
35
36         // Creates instances of each handler class, and maps them to RESTful URLs
37         // Handlers allow user to:
38         // Get all employees in text and JSON format
39         // Post an Employee object to the database in JSON format
40         // Retrieve a single Employee record based on a given Employee Number
41         // Update an Employee record with a given Employee Number
42         // Delete an Employee record with a given Employee Number
43         server.createContext("/", new HomeHandler());
44         server.createContext("/get-employees", new GetEmployeesTextHandler());
45         server.createContext("/get-json", new GetEmployeesJSONHandler());
46         server.createContext("/process_post", new ProcessJSONPostHandler());
47         server.createContext("/process_search", new ProcessEmployeeRetrieveHandler());
48         server.createContext("/process_update", new ProcessEmployeeUpdateHandler());
49         server.createContext("/process_delete", new ProcessEmployeeDeleteHandler());
50
51         // Start the server on port 8014, notify user of this at the console
52         System.out.println("Server running on port 8014");
53         server.start();
54         System.out.println();
55
56     }
57
58     // Catch any exceptions and display an error message in case of a server problem
59     catch (IOException ioe) {
60         System.err.println("IOException: " + ioe.getMessage() + " " + ioe.getStackTrace());
61     }
62 }
```

The ControllerHttpServer class was the first step of the 'server-side programming' element of the assignment. The first purpose of this class was to create a local host Http web server, running on port 8014. The server runs from this class, and displays console confirmation of this to the user. The second purpose of this class was to create server contexts for each function of the RESTful web service, by mapping the URLs to instances of each 'handler' class (handler classes are described below).

Console Output – ControllerHttpServer:

```
Console X
ControllerHttpServer (1) [Java Application] C:\Program Files\Java\jre1.8.0_151\bin\javaw.exe (14 Dec 2017, 13:24:14)
Server running on port 8014
```


Step 6 – ‘Home Handler’ Class:

```
18
19 public class HomeController implements Handler {
20
21
22 /**
23  * The handle method uses HttpExchange to output specified content to the browser, on the specified server
24  * context ("/"). In this case, a home page for the RESTful web service is being outputted using HTML.
25  *
26  * @param HttpExchange
27  * @throws IOException
28  *
29  * @author Adam Martin
30  * @version 1.0
31  *
32  */
33
34 // Handle method for browser output
35 public void handle(HttpExchange http) throws IOException {
36
37     // Buffered writer for outputting content to the browser
38     BufferedWriter out = new BufferedWriter(new OutputStreamWriter(http.getResponseBody()));
39
40     // Sends '200 okay' HTTP response header to display the content
41     http.sendResponseHeaders(200, 0);
42
43     // Uses Buffered writer to writes instructions for using the RESTful web service to the browser
44     out.write("<html><head></head><body><h1> Welcome to the Employee database service, running on server port 8014! </h1>");
45     out.write("\n");
46     out.write("<h3> Instructions for using the online Employee Database System (Use the 'POST' method in the RESTClient for all requests): </h3>");
47     out.write("<ul> <li> Go to 'http://localhost:8014/get-employees' to open a list of all Employees in the database in text format. </li>");
48     out.write("<li> Go to 'http://localhost:8014/get-json' to open a list of all Employees in the database in JSON format. </li>");
49     out.write("<li> To post an Employee object to the database in JSON format, write a JSON string in the RESTClient Body. (Use 'http://localhost:8014/process_post' as the URL). </li>");
50     out.write("<li> To retrieve a single Employee object from the database, write an existing Employee Number in the RESTClient Body. (Use 'http://localhost:8014/process_search' as the URL).");
51     out.write("<li> To update an Employee record based on its Employee Number, place the full record in the RESTClient Body in JSON format, and alter the attributes. (Use 'http://localhost:8014/process_update' as the URL).");
52     out.write("<li> To delete a single Employee record from the database, write its Employee Number in the RESTClient Body. (Use 'http://localhost:8014/process_delete' as the URL). </li>");
53
54     out.close();
55 }
56 }
```

The HomeController class includes code for outputting a system homepage/ instruction page to the user, with instructions displayed using embedded HTML code. The ControllerHttpServer class allows this content to display by creating an instance of the HomeController class, and placing it at the “/” server context.

Browser Output – HomeController:



Welcome to the Employee database service, running on server port 8014!

Instructions for using the online Employee Database System (Use the 'POST' method in the RESTClient for all requests):

- Go to 'http://localhost:8014/get-employees' to open a list of all Employees in the database in text format.
- Go to 'http://localhost:8014/get-json' to open a list of all Employees in the database in JSON format.
- To post an Employee object to the database in JSON format, write a JSON string in the RESTClient Body. (Use 'http://localhost:8014/process_post' as the URL).
- To retrieve a single Employee object from the database, write an existing Employee Number in the RESTClient Body. (Use 'http://localhost:8014/process_search' as the URL).
- To update an Employee record based on its Employee Number, place the full record in the RESTClient Body in JSON format, and alter the attributes. (Use 'http://localhost:8014/process_update' as the URL).
- To delete a single Employee record from the database, write its Employee Number in the RESTClient Body. (Use 'http://localhost:8014/process_delete' as the URL).

Step 7 – 'GetEmployeesTextHandler' Class:

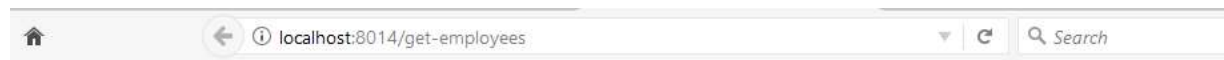
```

22 public class GetEmployeesTextHandler implements Handler {
23
24     // Creates an instance of the EmployeeDAO class for calling DAO CRUD methods
25     final EmployeeDAO DAO = new EmployeeDAO();
26
27
28     /**
29     * The handle public method uses HttpExchange to output specified content to the browser, on the specified server
30     * context ("/get-employees"). This is achieved by calling the DAO 'getAllEmployees' method. All records contained
31     * within the 'employees' database table are outputted in a HTML table.
32     *
33     * @param he
34     * @throws IOException
35     *
36     * @author Adam Martin
37     * @version 1.0
38     *
39     */
40
41     // Handle method for browser output
42     public void handle(HttpExchange he) throws IOException {
43
44         // Console output - tells users when the employee list has been printed to the browser
45         System.out.println("Print Operation - All employee records printed to browser");
46         System.out.println();
47
48         // HTML table for easily-readable browser display of the 'employees' database table
49         final String head = "<html><head></head><body><h1>Employee Database - All Records: </h1><table><tr>"
50             + "<th>Employee Number</th>"
51             + "<th>Name</th>"
52             + "<th>Gender</th>"
53             + "<th>DOB</th>"
54             + "<th>Address</th>"
55             + "<th>Postcode</th>"
56             + "<th>Department</th>"
57             + "<th>Start Date</th>"
58             + "<th>Salary</th>"
59             + "<th>Email</th>"
60             + "</tr>";
61
62         // Buffered writer for outputting content to the browser
63         BufferedWriter out = new BufferedWriter(new OutputStreamWriter(he.getResponseBody()));
64
65         // Executes the getAllEmployees DAO method for printing records to the browser
66         ArrayList<Employee> employees = null;
67         try {
68
69             employees = DAO.getAllEmployees();
70         } catch (SQLException e) {
71             e.printStackTrace();
72         }
73
74         // Sends '200 okay' HTTP response header to display the content
75         he.sendResponseHeaders(200, 0);
76
77         // Outputs the 'head' HTML table to the browser
78         out.write(head);
79         for (Iterator<Employee> iterator = employees.iterator(); iterator.hasNext(); ) {
80             Employee employee = (Employee) iterator.next();
81
82             // Outputs the list of employees to the browser (inside the 'head' HTML table)
83             out.write(
84                 "<tr><td>"
85                 + employee.getEmployeeNumber() + "</td><td>"
86                 + employee.getName() + "</td><td>"
87                 + employee.getGender() + "</td><td>"
88                 + employee.getDOB() + "</td><td>"
89                 + employee.getAddress() + "</td><td>"
90                 + employee.getPostcode() + "</td><td>"
91                 + employee.getDepartment() + "</td><td>"
92                 + employee.getStartdate() + "</td><td>"
93                 + employee.getSalary() + "</td><td>"
94                 + employee.getEmail() +
95                 "</td></tr>");
96
97             // Prints the records to the browser simultaneously for user convenience
98             System.out.println(employee.getName()
99                 + " " + employee.getGender()
100                 + " " + employee.getDOB()
101                 + " " + employee.getAddress()
102                 + " " + employee.getPostcode()
103                 + " " + employee.getEmployeeNumber()
104                 + " " + employee.getDepartment()
105                 + " " + employee.getStartdate()
106                 + " " + employee.getSalary()
107                 + " " + employee.getEmail());
108         }
109         out.close();
110     }
111 }
112
113
114
115

```

The GetEmployeesTextHandler class includes code for outputting HTML table of all employees to the user. This code involved calling the GetAllEmployees method from the EmployeeDAO class. The ControllerHttpServer class allows this content to display by creating an instance of the GetEmployeesTextHandler class, and placing it at the "/get-employees" server context.

Browser Output – GetEmployeesTextHandler:



Employee Database - All Records:

Employee Number	Name	Gender	DOB	Address	Postcode	Department	Start Date	Salary	Email
1	James Smith	M	14-07-1970	Manchester	M1 2FR	Computing	12-01-2000	40000.0	james@mail.com
2	Sarah Jones	F	14-07-1980	Stockport	S8 7YT	Biology	25-03-2009	35000.0	sarah@mail.com
3	Helen Taylor	F	30-06-1950	Oldham	O3 5NP	English	04-09-1980	50000.0	helen@mail.com
4	Adam Blake	M	28-03-1965	Manchester	M3 6ES	Nutrition	08-10-1995	38000.0	adam@mail.com
5	Jane Nolan	F	01-07-1970	Edale	E1 9QZ	Engineering	24-11-1998	40000.0	jane@mail.com
6	Geoff Oliver	M	09-04-1950	Stockport	S1 5GV	History	23-07-1980	45000.0	geoff@mail.com
7	David Thompson	M	09-04-1990	Sheffield	S3 2QW	Media Studies	23-07-2015	35000.0	david@mail.com
8	Sally Bradshaw	F	30-12-1985	Leeds	L5 4FQ	Mathematics	18-09-2010	35000.0	david@mail.com
10	Jenny Watson	F	19-04-1960	Liverpool	L2 9IB	Music	18-09-1989	43000.0	jenny@mail.com
11	Sam James	M	14-04-1960	Stoke	S1 2EQ	Economics	01-09-2012	49000.0	sam@mail.com

The above screenshot depicts browser output at the “/get-employees” server context. As you can see, the records are updated to reflect the results of the CRUD tests conducted using the DatabaseTester class.

Step 8 – ‘GetEmployeesJSONHandler’ Class:

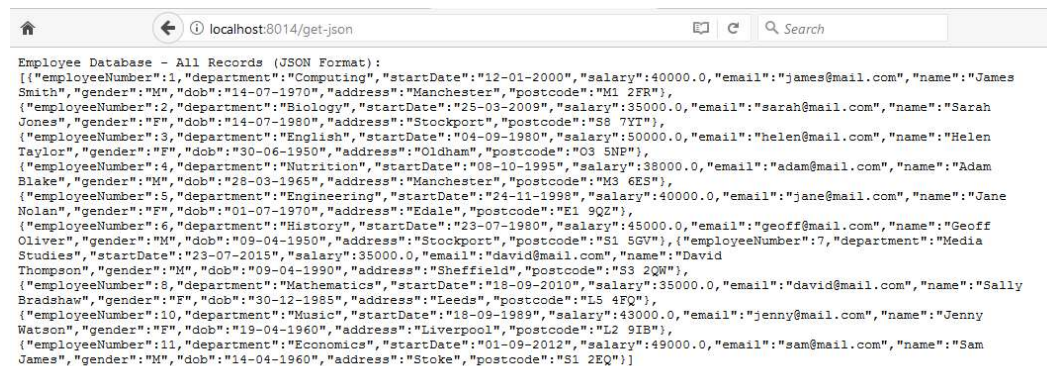
```

1 GetEmployeesJSONHandler.java
24
25 public class GetEmployeesJSONHandler implements HttpHandler {
26
27     // Creates an instance of the EmployeeDAO class for calling DAO CRUD methods
28     final EmployeeDAO dao = new EmployeeDAO();
29
30     /**
31     * The handle public method uses HttpExchange to output specified content to the browser, on the specified server
32     * context ("/get-json"). This is achieved by calling the DAO 'getAllEmployees' method. All records contained within
33     * the 'employees' database table are outputted in JSON format.
34     *
35     * @param he
36     * @throws IOException
37     * @author Adam Martin
38     * @version 1.0
39     */
40
41     // Handle method for browser output
42     public void handle(HttpExchange he) throws IOException {
43
44         // Console output - tells users when the employee list has been printed to the browser in JSON format
45         System.out.println("JSON Operation - All employee records printed to browser in JSON format using getAllEmployees method");
46         System.out.println();
47
48         // Sets a 'head' title for the web page
49         final String head = "Employee Database - All Records (JSON Format)";
50
51         // Buffered writer for outputting content to the browser
52         BufferedWriter out = new BufferedWriter(new OutputStreamWriter(he.getResponseBody()));
53
54         // Executes the getAllEmployees DAO method for printing records to the browser
55         ArrayList<Employee> employees = null;
56         try {
57             employees = dao.getAllEmployees();
58         } catch (IOException e) {
59             e.printStackTrace();
60         }
61
62         // Uses GSON to convert the employee ArrayList to a JSON string and output to browser
63         Gson gson = new Gson();
64         he.sendResponseHeaders(200, 0); // Sends '200 okay' HTTP response header to display the content
65
66         String employeesJson = gson.toJson(employees); // Puts the employees in a JSON string
67
68         // Outputs the 'head' web page title to the browser
69         out.write(head);
70
71         // Prints the JSON string of employees to the browser
72         out.write(employeesJson);
73         out.close();
74         System.out.println(employeesJson); // Prints the JSON string of employees to the console for user convenience
75         System.out.println();
76     }
77 }
78
79
80

```

The GetEmployeesJSONHandler class includes code for outputting all employee records to the user in JSON (JavaScript Object Notation) format. This code involved calling the GetAllEmployees method from the EmployeeDAO class, converting a String of these employees 'toJson', and outputting it to the user. This functionality can be tested in a web browser, at <http://localhost:8014/get-json>, or using Firefox RESTClient (see below screenshots). The ControllerHttpServer class allows this content to display by creating an instance of the GetEmployeesJSONHandler class, and placing it at the "/get-json" server context.

Browser Output – GetEmployeesJSONHandler:



```

Employee Database - All Records (JSON Format):
[{"employeeNumber":1,"department":"Computing","startDate":"12-01-2000","salary":40000.0,"email":"james@mail.com","name":"James Smith","gender":"M","dob":"14-07-1970","address":"Manchester","postcode":"M1 2FR"}, {"employeeNumber":2,"department":"Biology","startDate":"25-03-2009","salary":35000.0,"email":"sarah@mail.com","name":"Sarah Jones","gender":"F","dob":"14-07-1980","address":"Stockport","postcode":"S8 7YT"}, {"employeeNumber":3,"department":"English","startDate":"04-09-1980","salary":50000.0,"email":"helen@mail.com","name":"Helen Taylor","gender":"F","dob":"30-06-1950","address":"Oldham","postcode":"O3 5NP"}, {"employeeNumber":4,"department":"Nutrition","startDate":"08-10-1995","salary":38000.0,"email":"adam@mail.com","name":"Adam Blake","gender":"M","dob":"28-03-1965","address":"Manchester","postcode":"M3 6ES"}, {"employeeNumber":5,"department":"Engineering","startDate":"24-11-1998","salary":40000.0,"email":"jane@mail.com","name":"Jane Nolan","gender":"F","dob":"01-07-1970","address":"Edale","postcode":"E1 9QZ"}, {"employeeNumber":6,"department":"History","startDate":"23-07-1980","salary":45000.0,"email":"geoff@mail.com","name":"Geoff Oliver","gender":"M","dob":"09-04-1950","address":"Stockport","postcode":"S1 5GV"}, {"employeeNumber":7,"department":"Media Studies","startDate":"23-07-2015","salary":35000.0,"email":"david@mail.com","name":"David Thompson","gender":"M","dob":"09-04-1990","address":"Sheffield","postcode":"S3 2QW"}, {"employeeNumber":8,"department":"Mathematics","startDate":"18-09-2010","salary":35000.0,"email":"david@mail.com","name":"Sally Bradshaw","gender":"F","dob":"30-12-1985","address":"Leeds","postcode":"L5 4FQ"}, {"employeeNumber":9,"department":"Music","startDate":"18-09-1988","salary":43000.0,"email":"jenny@mail.com","name":"Jenny Watson","gender":"F","dob":"19-04-1960","address":"Liverpool","postcode":"L2 9IB"}, {"employeeNumber":10,"department":"Economics","startDate":"01-09-2012","salary":49000.0,"email":"sam@mail.com","name":"Sam James","gender":"M","dob":"14-04-1960","address":"Stoke","postcode":"S1 2EQ"}]

```


Testing of 'GetEmployeesJSON, Using the Firefox RESTclient:

[-] Request

Method POST URL http://localhost:8014/get-json

SEND

[-] Response

Headers Response

1. Status Code : 200 OK
2. Date : Thu, 14 Dec 2017 16:07:18 GMT
3. Transfer-encoding : chunked

[-] Response

Headers Response

1 Employee Database - All Records (JSON Format):
[{"employeeNumber":1,"department":"Computing","startDate":"12-01-2000","salary":45000.0,"email":"james2@mail.com","name":"James Smith","gender":"M","dob":"14-07-1970","address":"Altrincham","postcode":"A4 5WV"},
{"employeeNumber":2,"department":"Biology","startDate":"25-03-2009","salary":35000.0,"email":"sarah@mail.com","name":"Sarah Jones","gender":"F","dob":"14-07-1980","address":"Stockport","postcode":"S8 7YT"},
{"employeeNumber":3,"department":"English","startDate":"04-09-1980","salary":50000.0,"email":"helen@mail.com","name":"Helen Taylor","gender":"F","dob":"30-06-1950","address":"Oldham","postcode":"O3 SNP"},
{"employeeNumber":5,"department":"Engineering","startDate":"24-11-1998","salary":40000.0,"email":"jane@mail.com","name":"Jane Nolan","gender":"F","dob":"01-07-1970","address":"Edale","postcode":"E1 9QZ"},
{"employeeNumber":6,"department":"History","startDate":"23-07-1980","salary":45000.0,"email":"geoff@mail.com","name":"Geoff Oliver","gender":"M","dob":"09-04-1950","address":"Stockport","postcode":"S1 5GV"}, {"employeeNumber":7,"department":"Media Studies","startDate":"23-07-2015","salary":35000.0,"email":"david@mail.com","name":"David Thompson","gender":"M","dob":"09-04-1990","address":"Sheffield","postcode":"S3 2QW"},
{"employeeNumber":8,"department":"Mathematics","startDate":"18-09-2010","salary":35000.0,"email":"david@mail.com","name":"Sally Bradshaw","gender":"F","dob":"30-12-1985","address":"Leeds","postcode":"LS 4FQ"},
{"employeeNumber":10,"department":"Music","startDate":"18-09-1989","salary":43000.0,"email":"jenny@mail.com","name":"Jenny Watson","gender":"F","dob":"19-04-1960","address":"Liverpool","postcode":"L2 9IB"},
{"employeeNumber":11,"department":"Economics","startDate":"01-09-2012","salary":49000.0,"email":"sam@mail.com","name":"Sam James","gender":"M","dob":"14-04-1960","address":"Stoke","postcode":"S1 2EQ"},
{"employeeNumber":12,"department":"Chemistry","startDate":"18-02-1992","salary":48000.0,"email":"amy@mail.com","name":"Amy Radcliffe","gender":"F","dob":"14-07-1952","address":"Trafford","postcode":"T3 4YC"}]

These screenshots depict the successful testing of the retrieval of all employees in JSON format. As you can see, the JSON employee records were successfully received, and a positive '200 OK' response header was returned.

Step 9 – 'ProcessJSONPostHandler' Class:

```

24 public class ProcessJSONPostHandler implements Handler {
25
26
27     // Creates an instance of the EmployeeDAO class for calling DAO CRUD methods
28     final EmployeeDAO dao = new EmployeeDAO();
29
30
31     /**
32      * The handle public method uses HttpExchange to process a request on the specified server context ("/process_post").
33      * In this case, an input request, which takes the form of an employee object (in JSON format) is taken. The JSON string
34      * is then converted to an employee object, and posted to the database, using the DAO 'insertEmployee' method.
35      * This functionality can be tested using the Firefox RESTClient.
36      *
37      * @param he
38      * @throws IOException
39      *
40      * @author Adam Martin
41      * @version 1.0
42      *
43      */
44
45     // Handle method for browser output
46     public void handle(HttpExchange he) throws IOException {
47
48         // Sets a line and request to be read in
49         String line = "";
50         String request = "";
51
52         // Read in a request while the request line is not null
53         BufferedReader in = new BufferedReader(new InputStreamReader(he.getRequestBody()));
54         while ((line = in.readLine()) != null) {
55             request = request + line;
56             System.out.println(request);
57         }
58
59         // Buffered writer for outputting content to the browser
60         BufferedWriter out = new BufferedWriter(new OutputStreamWriter(he.getResponseBody()));
61
62         // Converts the user's JSON Employee String 'from JSON' to a regular Employee object, so it can be added to the database
63         Gson gson = new Gson();
64         Employee JSONemp = gson.fromJson(request, Employee.class);
65
66         out.write(request);
67
68         try {
69             he.sendResponseHeaders(200, 0); // Sends '200 okay' HTTP response header to display the content
70
71             // calls the DAO 'insertEmployee' method to insert the Employee object into the database
72             dao.insertEmployee(JSONemp);
73             out.write("Employee successfully posted in JSON format");
74         }
75
76         catch (SQLException se) {
77
78             // Sends a HTTP 500 (Internal Server Error) error to the user if there is an error during the process
79             he.sendResponseHeaders(500, 0);
80             out.write("Error posting Employee in JSON format");
81         }
82
83         finally {
84             out.close();
85         }
86     }
87
88 }

```

The ProcessJSONPostHandler class includes code for allowing the user to post an Employee record to the database, in JSON format. This code takes in a request, which takes the form of a JSON Employee string. The JSON string request is then converted to a regular employee object, and added to the database via the calling of the EmployeeDAO insertEmployee method. This functionality can be tested using the Firefox RESTClient (see below screenshots). The ControllerHttpServer class sets up this handler by creating an instance of the ProcessJSONPostHandler class, and placing it at the "/process_post" server context.

Testing of JSON Post, Using the Firefox RESTclient:

The screenshot shows the Firefox REST Client interface. The top section is labeled '[-] Request' and shows a POST method to the URL 'http://localhost:8014/process_post'. The body contains a JSON object for a new employee: `{ "employeeNumber": 12, "department": "Chemistry", "startDate": "18-02-1992", "salary": 48000.0, "email": "amy@mail.com", "name": "Amy Radcliffe", "gender": "F", "dob": "14-07-1952", "address": "Trafford", "postcode": "T3 4YC" }`. The bottom section is labeled '[-] Response' and shows a '200 OK' status code, the date 'Thu, 14 Dec 2017 14:56:34 GMT', and 'chunked' transfer encoding. A second '[-] Response' section below shows the full JSON response: `{ "employeeNumber": 12, "department": "Chemistry", "startDate": "18-02-1992", "salary": 48000.0, "email": "amy@mail.com", "name": "Amy Radcliffe", "gender": "F", "dob": "14-07-1952", "address": "Trafford", "postcode": "T3 4YC" } Employee successfully posted in JSON format`.

These screenshots depict the successful testing of the JSON post. As you can see, the employee was successfully posted, and a positive '200 OK' response header was returned.

Evidence of the New Employee Record (12, Amy Radcliffe) in the Database:



Employee Database - All Records:

Employee Number	Name	Gender	DOB	Address	Postcode	Department	Start Date	Salary	Email
1	James Smith	M	14-07-1970	Manchester	M1 2FR	Computing	12-01-2000	40000.0	james@mail.com
2	Sarah Jones	F	14-07-1980	Stockport	S8 7YT	Biology	25-03-2009	35000.0	sarah@mail.com
3	Helen Taylor	F	30-06-1950	Oldham	O3 5NP	English	04-09-1980	50000.0	helen@mail.com
4	Adam Blake	M	28-03-1965	Manchester	M3 6ES	Nutrition	08-10-1995	38000.0	adam@mail.com
5	Jane Nolan	F	01-07-1970	Edale	E1 9QZ	Engineering	24-11-1998	40000.0	jane@mail.com
6	Geoff Oliver	M	09-04-1950	Stockport	S1 5GV	History	23-07-1980	45000.0	geoff@mail.com
7	David Thompson	M	09-04-1990	Sheffield	S3 2QW	Media Studies	23-07-2015	35000.0	david@mail.com
8	Sally Bradshaw	F	30-12-1985	Leeds	L5 4FQ	Mathematics	18-09-2010	35000.0	david@mail.com
10	Jenny Watson	F	19-04-1960	Liverpool	L2 9IB	Music	18-09-1989	43000.0	jenny@mail.com
11	Sam James	M	14-04-1960	Stoke	S1 2EQ	Economics	01-09-2012	49000.0	sam@mail.com
12	Amy Radcliffe	F	14-07-1952	Trafford	T3 4YC	Chemistry	18-02-1992	48000.0	amy@mail.com



Step 10 – 'ProcessEmployeeUpdateHandler' Class:

```

1 ProcessEmployeeUpdateHandler.java X
25 public class ProcessEmployeeUpdateHandler implements Handler {
26
27
28 // Creates an instance of the EmployeeDAO class for calling DAO CRUD methods
29 final EmployeeDAO dao = new EmployeeDAO();
30
31
32 /**
33  * The handle public method uses HttpExchange to process a request on the specified server context ("/process_update").
34  * In this case, an input request, which takes the form of an employee object (in JSON format) containing updated
35  * attributes, is taken. The JSON string is then converted to an employee object, with the updated attributes attached to the
36  * specified database record, using the DAO 'updateEmployee' method. This functionality can be tested using the Firefox RESTClient.
37  *
38  * @param he
39  * @throws IOException
40  *
41  * @author Adam Martin
42  * @version 1.0
43  */
44
45 // Handle method for browser output
46 public void handle(HttpExchange he) throws IOException {
47
48 // Sets a line and request to be read in
49 String line = "";
50 String request = "";
51
52 // Read in a request while the request line is not null
53 BufferedReader in = new BufferedReader(new InputStreamReader(he.getRequestBody()));
54 while ((line = in.readLine()) != null) {
55 request = request + line;
56 System.out.println(request);
57 }
58
59 // Buffered writer for outputting content to the browser
60 BufferedWriter out = new BufferedWriter(new OutputStreamWriter(he.getResponseBody()));
61
62 // Converts the user's updated JSON Employee String 'from JSON' to a regular Employee object
63 Gson gson = new Gson();
64 Employee updatedEmp = gson.fromJson(request, Employee.class);
65
66 try {
67     he.sendResponseHeaders(200, 0); // Sends '200 okay' HTTP response header to display the content
68
69
70 // Calls the DAO 'updateEmployee' method to update the attributes of the given Employee object
71 dao.updateEmployee(updatedEmp);
72 out.write("Employee successfully updated");
73 }
74
75 catch (SQLException se) {
76
77 // Sends a HTTP 500 (Internal Server Error) error to the user if there is an error during the process
78 he.sendResponseHeaders(500, 0);
79 out.write("Error updating Employee");
80 }
81
82 finally {
83     out.close();
84 }
85
86 }
87
88

```

The ProcessEmployeeUpdateHandler class includes code for allowing the user to update all of the attributes of an employee record, with a specified employee number. This code takes in a request, which takes the form of an existing employee record, in JSON format. The attributes of the employee can then be edited, and posted to the database, via the calling of the EmployeeDAO updateEmployee method. This functionality can be tested using the Firefox RESTClient (see below screenshots). The ControllerHttpServer class sets up this handler by creating an instance of the ProcessEmployeeUpdateHandler class, and placing it at the "/process_update" server context.

Testing of the Server-Side Update, Using the Firefox RESTclient:

The image contains two screenshots from the Firefox REST Client. The top screenshot shows a 'Request' tab with a POST method to the URL 'http://localhost:8014/process_update'. The request body is a JSON object:

```
{ "employeeNumber": 1, "department": "Computing", "startDate": "12-01-2000", "salary": 45000.0, "email": "james2@mail.com", "name": "James Smith", "gender": "M", "dob": "14-07-1970", "address": "Altrincham", "postcode": "A4 5WV" }
```

. The bottom screenshot shows a 'Response' tab with a status code of 200 OK, a date of Thu, 14 Dec 2017 15:23:16 GMT, and a transfer encoding of chunked. The response body is '1 Employee successfully updated'.

Request

Method: POST URL: http://localhost:8014/process_update

Body:

```
{ "employeeNumber": 1, "department": "Computing", "startDate": "12-01-2000", "salary": 45000.0, "email": "james2@mail.com", "name": "James Smith", "gender": "M", "dob": "14-07-1970", "address": "Altrincham", "postcode": "A4 5WV" }
```

Response

1. Status Code : 200 OK
2. Date : Thu, 14 Dec 2017 15:23:16 GMT
3. Transfer-encoding : chunked

1 Employee successfully updated

These screenshots depict the successful testing of the employee update. A request to update James Smith's (Employee 1) salary, email address, address and postcode was sent (it is possible for the user to update other attributes if they wish). As you can see, the employee was successfully updated, and a positive '200 OK' response header was returned.

Evidence of the Updated Employee Record (1, James Smith) in the Database:

Existing:

Employee Number	Name	Gender	DOB	Address	Postcode	Department	Start Date	Salary	Email
1	James Smith	M	14-07-1970	Manchester M1 2FR	Computing	12-01-2000	40000.0	james@mail.com	

New (updated salary, email address, address and postcode):

Employee Number	Name	Gender	DOB	Address	Postcode	Department	Start Date	Salary	Email
1	James Smith	M	14-07-1970	Altrincham	A4 5WV	Computing	12-01-2000	45000.0	james2@mail.com

Step 11 – 'ProcessEmployeeDeleteHandler' Class:

```

ProcessEmployeeDeleteHandler.java
24 public class ProcessEmployeeDeleteHandler implements Handler {
25
26     // Creates an instance of the EmployeeDAO class for calling DAO CRUD methods
27     final EmployeeDAO dao = new EmployeeDAO();
28
29
30     /**
31      * The handle public method uses HttpExchange to process a request on the specified server context ("/process_delete").
32      * In this case, an input request, which takes the form of an employee number, is taken. When this request is processed,
33      * the DAO 'deleteEmployee' method is called to delete the employee record attached to the employee number from the database.
34      * This functionality can be tested using the Firefox RESTClient.
35      *
36      * @param he
37      * @throws IOException
38      *
39      * @author Adam Martin
40      * @version 1.0
41      */
42
43     // Handle method for browser output
44     public void handle(HttpExchange he) throws IOException {
45
46         // Sets a line and request to be read in
47         String line = "";
48         String request = "";
49
50         // Buffered reader for reading in data requests
51         BufferedReader in = new BufferedReader(new InputStreamReader(he.getRequestBody()));
52
53         // Read in a request while the request line is not null
54         while ((line = in.readLine()) != null) {
55             request = request + line;
56         }
57
58         // Outputs the data request to the console for user convenience
59         System.out.println(request);
60     }
61
62     // Buffered writer for outputting content to the browser
63     BufferedWriter out = new BufferedWriter(new OutputStreamWriter(he.getResponseBody()));
64
65     try {
66         he.sendResponseHeaders(200, 0); // Sends '200 okay' HTTP response header to display the content
67
68         // Calls the DAO 'deleteEmployee' method to delete an employee based on the request (an employee number)
69         dao.deleteEmployee(request);
70
71         // Confirms the deletion to the user
72         out.write("Employee successfully deleted" + " ");
73         out.write("Employee Number of deleted employee: " + request + " ");
74     }
75     catch (SQLException se) {
76
77         // Sends a HTTP 500 (Internal Server Error) error to the user if there is an error during the process
78         he.sendResponseHeaders(500, 0);
79         out.write("Error deleting Employee");
80     }
81     finally {
82         out.close();
83     }
84 }

```

The ProcessEmployeeDeleteHandler class includes code for allowing the user to delete an employee record with a specified employee number. This code takes in a request, which takes the form of an existing employee number. The employee record attached to that employee number is then deleted, via the calling of the EmployeeDAO deleteEmployee method. This functionality can be tested using the Firefox RESTClient (see below screenshots). The ControllerHttpServer class sets up this handler by creating an instance of the ProcessEmployeeDeleteHandler class, and placing it at the "/process_delete" server context.

Testing of the Server-Side Delete, Using the Firefox RESTclient:

The screenshot shows the Firefox REST client interface. The top section is labeled '[-] Request' and contains a 'Method' dropdown set to 'POST' and a 'URL' field with the value 'http://localhost:8014/process_delete'. A red 'SEND' button is visible. Below this is a 'Body' section with a text input field containing the number '4'. The bottom section is labeled '[-] Response' and has two tabs: 'Headers' and 'Response'. The 'Response' tab is selected, showing a list of response details: '1. Status Code : 200 OK', '2. Date : Thu, 14 Dec 2017 15:43:04 GMT', and '3. Transfer-encoding : chunked'. Below this, another '[-] Response' section shows a single line of text: '1 Employee successfully deleted Employee Number of deleted employee: 4'.

These screenshots depict the successful testing of the employee delete. A request to delete the employee record with an employee number of '4' was sent. As you can see, the employee was successfully deleted, and a positive '200 OK' response header was returned.

Evidence of the Deletion of Employee Record (4) from the Database:

localhost:8014/get-employees

Search

Employee Database - All Records:

Employee Number	Name	Gender	DOB	Address	Postcode	Department	Start Date	Salary	Email
1	James Smith	M	14-07-1970	Altrincham A4 5WV	Computing	12-01-2000	45000.0	james2@mail.com	
2	Sarah Jones	F	14-07-1980	Stockport S8 7YT	Biology	25-03-2009	35000.0	sarah@mail.com	
3	Helen Taylor	F	30-06-1950	Oldham O3 5NP	English	04-09-1980	50000.0	helen@mail.com	
5	Jane Nolan	F	01-07-1970	Edale E1 9QZ	Engineering	24-11-1998	40000.0	jane@mail.com	
6	Geoff Oliver	M	09-04-1950	Stockport S1 5GV	History	23-07-1980	45000.0	geoff@mail.com	
7	David Thompson	M	09-04-1990	Sheffield S3 2QW	Media Studies	23-07-2015	35000.0	david@mail.com	
8	Sally Bradshaw	F	30-12-1985	Leeds L5 4FQ	Mathematics	18-09-2010	35000.0	david@mail.com	
10	Jenny Watson	F	19-04-1960	Liverpool L2 9IB	Music	18-09-1989	43000.0	jenny@mail.com	
11	Sam James	M	14-04-1960	Stoke S1 2EQ	Economics	01-09-2012	49000.0	sam@mail.com	
12	Amy Radcliffe	F	14-07-1952	Trafford T3 4YC	Chemistry	18-02-1992	48000.0	amy@mail.com	

As you can see, an employee record with an employee number of '4' no longer exists in the database.

Step 12 – 'ProcessEmployeeRetrieveHandler' Class:

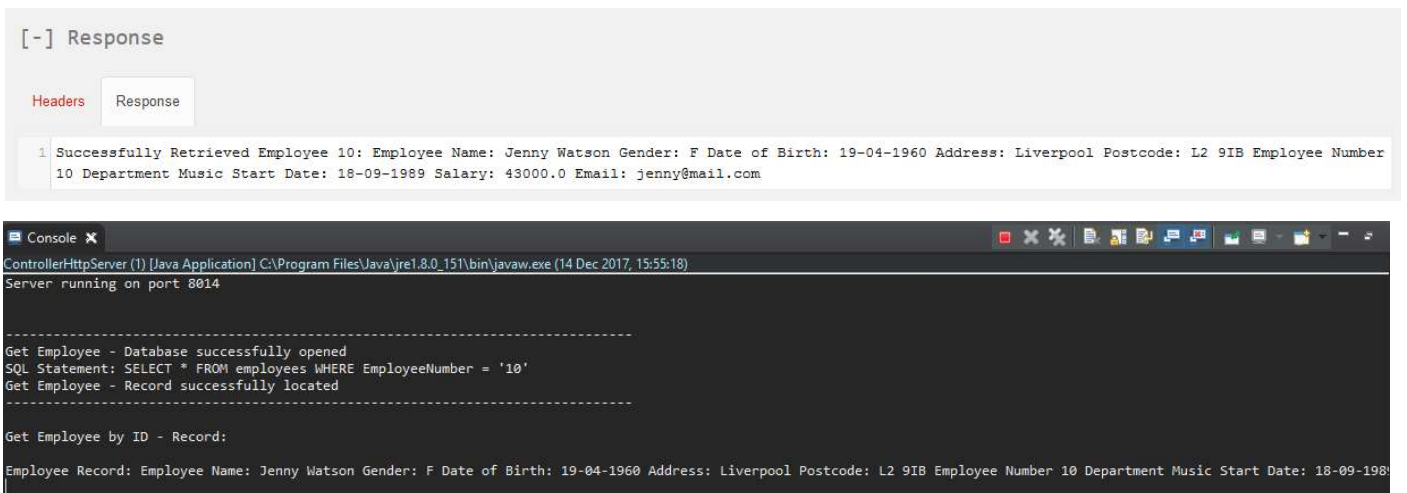
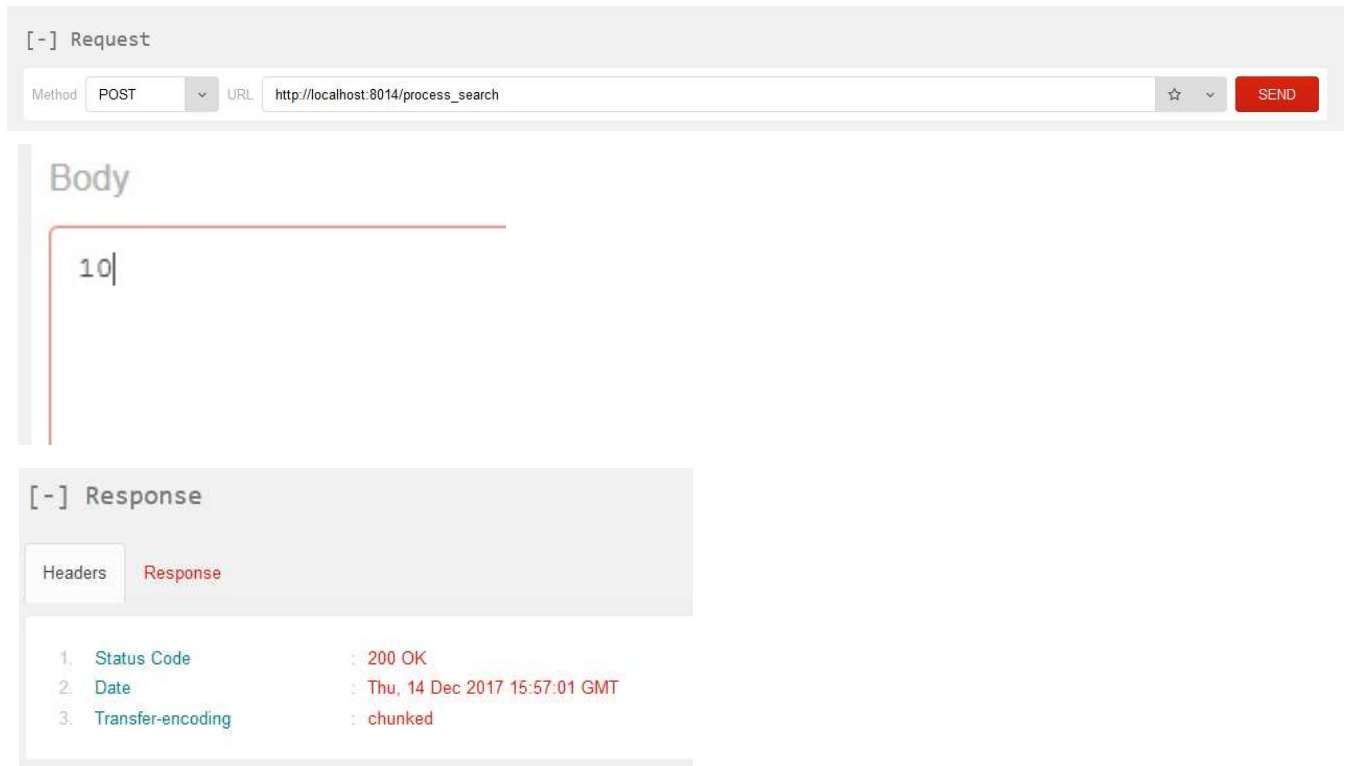
```

24 public class ProcessEmployeeRetrieveHandler implements Handler {
25
26     // Creates an instance of the EmployeeDAO class for calling DAO CRUD methods.
27     final EmployeeDAO dao = new EmployeeDAO();
28
29
30     /**
31      * The handle public method uses HttpExchange to process a request on the specified server context ("/process_search").
32      * In this case, an input request, which takes the form of an employee number, is taken. When this request is processed,
33      * when this request is processed, the DAO 'getEmployee' method is called to retrieve the employee record attached to the
34      * employee number from the database. This employee record is then sent to the console. This functionality can be tested
35      * using the Firefox RESTClient.
36      *
37      * @param he
38      * @throws IOException
39      *
40      * @author Adam Martin
41      * @version 1.0
42      *
43      */
44
45     // Handle method for browser output
46     public void handle(HttpExchange he) throws IOException {
47
48         // Sets a line and request to be read in
49         String line = "";
50         String request = "";
51
52         // Read in a request while the request line is not null
53         BufferedReader in = new BufferedReader(new InputStreamReader(he.getRequestBody()));
54         while ((line = in.readLine()) != null) {
55             request = request + line;
56         }
57
58         // Buffered writer for outputting content to the browser
59         BufferedWriter out = new BufferedWriter(new OutputStreamWriter(he.getResponseBody()));
60
61         try {
62             he.sendResponseHeaders(200, 0); // Sends '200 okay' HTTP response header to display the content
63
64             // Calls the DAO 'getEmployee' method to retrieve an employee based on the request (an employee number)
65             Employee emp = dao.getEmployee(request);
66
67             // Converts the employee object to a String using the 'toString' method for readable output
68             String employee = emp.toString();
69
70             // Outputs the requested employee number and employee to the user
71             out.write("Successfully Retrieved Employee " + request + ": " + employee);
72         }
73         catch (SQLException se) {
74
75             // Sends a HTTP 500 (Internal Server Error) error to the user if there is an error during the process
76             he.sendResponseHeaders(500, 0);
77             out.write("Error retrieving Employee");
78         }
79         finally {
80             out.close();
81         }
82     }
83
84 }
85
86
87
88

```

The ProcessEmployeeRetrieveHandler class includes code for allowing the user to search the database for a single employee record with a specified employee number. This code takes in a request, which takes the form of an existing employee number. The employee record attached to that employee number is then located, and posted in the RESTClient body and the Eclipse console, via the calling of the EmployeeDAO getEmployee method. This functionality can be tested using the Firefox RESTClient (see below screenshots). The ControllerHttpServer class sets up this handler by creating an instance of the ProcessEmployeeRetrieveHandler class, and placing it at the "/process_search" server context.

Testing of the Server-Side Single Employee Retrieve, Using the Firefox RESTclient:



These screenshots depict the successful testing of the single employee retrieve. A request to delete the employee record with an employee number of '4' was sent. As you can see, the employee was successfully retrieved, and a positive '200 OK' response header was returned. The employee record displays in the 'response' section of the RESTclient, and in the Eclipse console.

Step 13 – 'WebServiceTester' Class:

```

24 public class WebserviceTester {
25     |
26     /**
27     * The main method within this class calls the private static methods of the 'WebServiceTester' class, in order
28     * to test the server side CRUD operations, and print the results to the console screen. The methods are tested
29     * using test data and their relevant URL, if appropriate.
30     *
31     * @author Adam Martin
32     * @version 1.0
33     */
34     //
35
36     public static void main(String[] args) throws IOException {
37         // Console title
38         System.out.println("----Console Testing of RESTful URL Output:---");
39         System.out.println();
40         // Console RESTful testing outputs
41         // Testing of the output of the 'get-JSON' URL by printing its contents to the console
42         System.out.println("-----Output of 'Employees in JSON format' page:-----");
43         System.out.println(getAllEmployees());
44         System.out.println();
45         // Testing of the RESTful route to post a JSON Employee object to the database
46         System.out.println("-----Test JSON Employee Sent to Database:-----");
47         System.out.println(postJSONEmp("{\"employeeNumber\":\"13\",\"department\":\"Design\",\"startDate\":\"08-03-2008\",\"salary\":32000.0,\"email\":\"jack@mail.com\",\"name\":\"Jack Morris\",\"gender\":\"http://localhost:8014/process_post\"}"));
48         System.out.println();
49         // Testing of the RESTful route to retrieve an Employee record based on a given Employee id/ Number
50         System.out.println("-----Test Retrieving of Single Employee Record:-----");
51         System.out.println(getEmp("6", "http://localhost:8014/process_search"));
52         System.out.println();
53         // Testing of the RESTful route to update the attributes of an employee record with a particular Employee Number
54         System.out.println(updateEmp("{\"employeeNumber\":\"5\",\"department\":\"Physics\",\"startDate\":\"24-11-1998\",\"salary\":47000.0,\"email\":\"jane@mail.com\",\"name\":\"Jane Nolan\",\"gender\":\"http://localhost:8014/process_update\"}"));
55         System.out.println();
56     }
57 }

```

```

70 // Testing of the RESTful route to delete an Employee record based on a given Employee id/ Number
71 System.out.println(deleteEmp("8", "http://localhost:8014/process_delete"));
72
73 }
74
75 // Static methods for printing the contents of the RESTful URL pages to the console
76
77 // Creates a StringBuffer for storing and returning the response for testing (the list of employees in JSON format)
78 // 'getAllEmployees' StringBuffer is called in the main method for testing
79 private static StringBuffer getAllEmployees() {
80     StringBuffer response = new StringBuffer();
81
82     try {
83         // Reads in the 'get-JSON' URL for testing
84         URL url = new URL("http://localhost:8014/get-json");
85         BufferedReader reader = new BufferedReader(
86             new InputStreamReader(url.openStream()));
87
88         String output;
89
90         // Adds/ appends to the response line String when the line is not null
91         while ((output = reader.readLine()) != null) {
92             response.append(output);
93         }
94         reader.close();
95     } catch (Exception e) {
96         System.out.println(e.getMessage());
97     }
98
99 // Returns the response using the StringBuffer
100 return response;
101 }
102
103 }
104
105 }
106
107 }

```

```

110 // Creates a StringBuffer for storing and returning the response for testing (the posting of an employee object in JSON format)
111 // 'postJSONEmp' StringBuffer is called in the main method using test data, for testing
112 private static StringBuffer postJSONEmp(String request, String url) {
113     StringBuffer response = new StringBuffer();
114
115     try {
116         // Uses HttpURLConnection to connect to HTTP Server for testing the post request
117         URL newURL = new URL(url);
118         HttpURLConnection connection = (HttpURLConnection) newURL.openConnection();
119         connection.setDoInput(true);
120         connection.setDoOutput(true);
121
122         // Output request is sent to the server, and a response is obtained
123         DataOutputStream output = new DataOutputStream(connection.getOutputStream());
124         output.writeBytes(request); // Writes out the request string
125         InputStream input = connection.getInputStream(); // Allows for an input stream
126
127         // Allows for the reading in of a request using BufferedReader
128         BufferedReader in = new BufferedReader(new InputStreamReader(input));
129
130         // Adds/ appends to the response line String when the line is not null
131         String line;
132         while ((line = in.readLine()) != null) {
133             response.append(line);
134         }
135     } catch (Exception e) {
136         System.out.println(e.getMessage());
137     }
138
139 // Returns the response using the StringBuffer
140 return response;
141 }
142
143 }
144
145 }
146
147 }

```

```

149 // Creates a StringBuffer for storing and returning the response for testing (the retrieving of an Employee with a given Employee Number)
150 // 'postJSONEmp' StringBuffer is called in the main method using an Employee Number, for testing
151 private static StringBuffer getEmp(String request, String url) {
152     StringBuffer response = new StringBuffer();
153
154     try {
155         // Uses HttpURLConnection to connect to json Server for testing the request
156         URL newURL = new URL(url);
157         HttpURLConnection connection = (HttpURLConnection) newURL.openConnection();
158         connection.setDoInput(true);
159         connection.setDoOutput(true);
160
161         // Output request is sent to the server, and a response is obtained
162         DataOutputStream output = new DataOutputStream(connection.getOutputStream());
163         output.writeBytes(request);
164         InputStream input = connection.getInputStream();
165
166         // Allows for the reading in of a request using BufferedReader
167         BufferedReader in = new BufferedReader(new InputStreamReader(input));
168
169         // Adds/ appends to the response line String when the line is not null
170         String line;
171         while ((line = in.readLine()) != null) {
172             response.append(line);
173         }
174     } catch (Exception e) {
175         System.out.println(e.getMessage());
176     }
177
178     // Returns the response using the StringBuffer
179     return response;
180 }
181
182
183
184
185

```

```

186
187 // Creates a StringBuffer for storing and returning the response for testing (the updating of an Employee's attributes based on Employee Number)
188 // 'updateEmp' StringBuffer is called in the main method using test data, for testing
189 private static StringBuffer updateEmp(String request, String url) {
190     StringBuffer response = new StringBuffer();
191
192     try {
193         // Uses HttpURLConnection to connect to json Server for testing the request
194         URL newURL = new URL(url);
195         HttpURLConnection connection = (HttpURLConnection) newURL.openConnection();
196         connection.setDoInput(true);
197         connection.setDoOutput(true);
198
199         // Output request is sent to the server, and a response is obtained
200         DataOutputStream output = new DataOutputStream(connection.getOutputStream());
201         output.writeBytes(request);
202         InputStream input = connection.getInputStream();
203
204         // Allows for the reading in of a request using BufferedReader
205         BufferedReader in = new BufferedReader(new InputStreamReader(input));
206
207         // Adds/ appends to the response line String when the line is not null
208         String line;
209         while ((line = in.readLine()) != null) {
210             response.append(line);
211         }
212     } catch (Exception e) {
213         System.out.println(e.getMessage());
214     }
215
216     // Returns the response using the StringBuffer
217     return response;
218 }
219
220
221
222
223
224

```

```

225
226 // Create a StringBuffer for storing and returning the response for testing (the deleting of an Employee with a given Employee Number)
227 // 'deleteEmp' StringBuffer is called in the main method using an Employee Number, for testing
228 private static StringBuffer deleteEmp(String request, String url) {
229     StringBuffer response = new StringBuffer();
230
231     try {
232         // Uses HttpURLConnection to connect to json Server for testing the request
233         URL newURL = new URL(url);
234         HttpURLConnection connection = (HttpURLConnection) newURL.openConnection();
235         connection.setDoInput(true);
236         connection.setDoOutput(true);
237
238         // Output request is sent to the server, and a response is obtained
239         DataOutputStream output = new DataOutputStream(connection.getOutputStream());
240         output.writeBytes(request);
241         InputStream input = connection.getInputStream();
242
243         // Allows for the reading in of a request using BufferedReader
244         BufferedReader in = new BufferedReader(new InputStreamReader(input));
245
246         // Adds/ appends to the response line String when the line is not null
247         String line;
248         while ((line = in.readLine()) != null) {
249             response.append(line);
250         }
251     } catch (Exception e) {
252         System.out.println(e.getMessage());
253     }
254
255     // Returns the response using the StringBuffer
256     return response;
257 }
258
259
260
261
262
263
264
265

```

The `WebServiceTester` class includes code for testing the functionality of the server-side handler CRUD methods at the console. The class tests methods for the getting of all employees in JSON format, the posting of a JSON employee object, the retrieval of a single employee object, the updating of an employee record, and the deletion of an employee record. The class consists of private static methods for each operation. These methods use `HttpURLConnection` to connect to the server, and `String Buffers` for storing and returning the

responses. These methods are called in a main method, using test data. Confirmation of the methods' successful testing is printed to the console screen. To test the JSON post, a new JSON employee record with an employee number of '13' is posted. Employee number 6 is retrieved, employee number 5's department, salary, email address, address and postcode is updated, and employee number 8 is deleted (see below screenshots for evidence that these operations were successful).

Console Output – WebServiceTester (CRUD operations actioned, and employee 6 retrieved)

```

Console
terminated> WebServiceTester (1) [Java Application] C:\Program Files\Java\jre1.8.0_151\bin\javaw.exe (14 Dec 2017, 17:01:57)
---Console Testing of RESTful URL Output:---
-----Output of 'Employees in JSON format' page:-----
Employee Database - All Records (JSON Format):[{"employeeNumber":1,"department":"Computing","startDate":"12-01-2000","salary":45000.0,"email":"james2@mail.com","name":"James :
-----Test JSON Employee Sent to Database:-----
{"employeeNumber":13,"department":"Design","startDate":"08-03-2008","salary":32000.0,"email":"jack@mail.com","name":"Jack Morris","gender":"M","dob":"08-06-1979","address":"S
-----Test Retrieving of Single Employee Record:-----
Successfully Retrieved Employee 6: Employee Name: Geoff Oliver Gender: M Date of Birth: 09-04-1950 Address: Stockport Postcode: S1 5GV Employee Number 6 Department History St
Employee successfully updated
Employee successfully deleted Employee Number of deleted employee: 8
    
```

Evidence of the New Employee Record (13, Jack Morris) in the Database:



Employee Database - All Records:

Employee Number	Name	Gender	DOB	Address	Postcode	Department	Start Date	Salary	Email
1	James Smith	M	14-07-1970	Altrincham A4 5WV	Computing	12-01-2000	45000.0	james2@mail.com	
2	Sarah Jones	F	14-07-1980	Stockport S8 7YT	Biology	25-03-2009	35000.0	sarah@mail.com	
3	Helen Taylor	F	30-06-1950	Oldham O3 5NP	English	04-09-1980	50000.0	helen@mail.com	
5	Jane Nolan	F	01-07-1970	High Lane H2 6HS	Physics	24-11-1998	47000.0	jane2@mail.com	
6	Geoff Oliver	M	09-04-1950	Stockport S1 5GV	History	23-07-1980	45000.0	geoff@mail.com	
7	David Thompson	M	09-04-1990	Sheffield S3 2QW	Media Studies	23-07-2015	35000.0	david@mail.com	
10	Jenny Watson	F	19-04-1960	Liverpool L2 9IB	Music	18-09-1989	43000.0	jenny@mail.com	
11	Sam James	M	14-04-1960	Stoke S1 2EQ	Economics	01-09-2012	49000.0	sam@mail.com	
12	Amy Radcliffe	F	14-07-1952	Trafford T3 4YC	Chemistry	18-02-1992	48000.0	amy@mail.com	
13	Jack Morris	M	08-06-1979	Stockport S2 9JA	Design	08-03-2008	32000.0	jack@mail.com	



Evidence of the Updated Employee Record (5, Jane Nolan) in the Database:

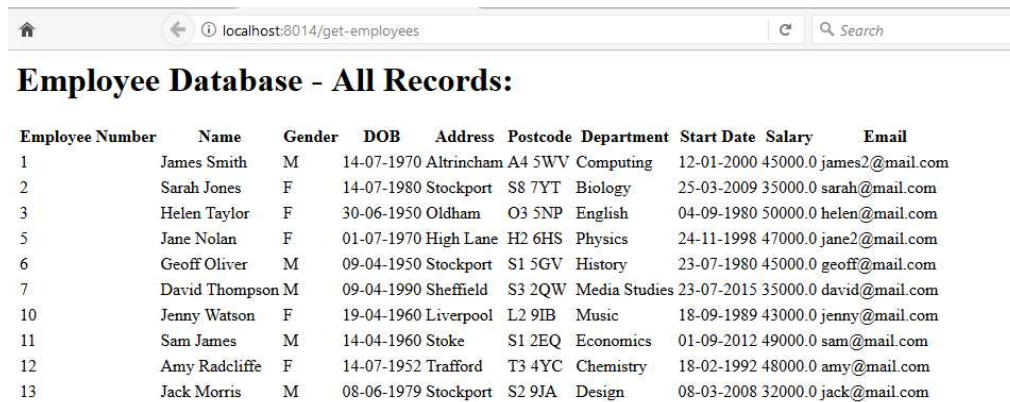
Existing:

5	Jane Nolan	F	01-07-1970	Edale E1 9QZ	Engineering	24-11-1998	40000.0	jane@mail.com
---	------------	---	------------	--------------	-------------	------------	---------	---------------

New (updated department, salary, email address, address and postcode):

5	Jane Nolan	F	01-07-1970	High Lane H2 6HS	Physics	24-11-1998	47000.0	jane2@mail.com
---	------------	---	------------	------------------	---------	------------	---------	----------------

Evidence of the Deletion of Employee Record (8) from the Database:



Employee Number	Name	Gender	DOB	Address	Postcode	Department	Start Date	Salary	Email
1	James Smith	M	14-07-1970	Altrincham	A4 5WV	Computing	12-01-2000	45000.0	james2@mail.com
2	Sarah Jones	F	14-07-1980	Stockport	S8 7YT	Biology	25-03-2009	35000.0	sarah@mail.com
3	Helen Taylor	F	30-06-1950	Oldham	O3 5NP	English	04-09-1980	50000.0	helen@mail.com
5	Jane Nolan	F	01-07-1970	High Lane	H2 6HS	Physics	24-11-1998	47000.0	jane2@mail.com
6	Geoff Oliver	M	09-04-1950	Stockport	S1 5GV	History	23-07-1980	45000.0	geoff@mail.com
7	David Thompson	M	09-04-1990	Sheffield	S3 2QW	Media Studies	23-07-2015	35000.0	david@mail.com
10	Jenny Watson	F	19-04-1960	Liverpool	L2 9IB	Music	18-09-1989	43000.0	jenny@mail.com
11	Sam James	M	14-04-1960	Stoke	S1 2EQ	Economics	01-09-2012	49000.0	sam@mail.com
12	Amy Radcliffe	F	14-07-1952	Trafford	T3 4YC	Chemistry	18-02-1992	48000.0	amy@mail.com
13	Jack Morris	M	08-06-1979	Stockport	S2 9JA	Design	08-03-2008	32000.0	jack@mail.com

As you can see, an employee record with an employee number of '8' no longer exists in the database.

How could this system be improved?

The above screenshots and accompanied source code illustrate the successful creation of a Java RESTful web service, which can manipulate database data in both offline and online server-side contexts. However, it is important to note that there is room for improvement. Security is a hugely important issue in web application development. However, due to time constraints, this was not implemented. Also, data validation, which would have assured that all data entered was error-free, was also not added. Taking time to implement the advanced features of the assignment, which include these features, will address these issues.