MANCHESTER METROPOLITAN UNIVERSITY
School of Computing, Mathematics & Digital Technology

ASSIGNMENT COVER SHEET

Unit:	6G5Z1001 Advanced Programming
Assignment set by:	Dr Alan Crispin/Dr Mohammed Kaleem
Verified by:	Mr Nick Whittaker
Moderated by:	Mr Nick Whittaker
Assignment number:	1CWK50
Assignment title:	Student RESTful Web Service Coursework
Type: (GROUP/INDIVIDUAL)	Individual
Hand-in format and mechanism:	Submission is online, via Moodle. More information is available in the attached coursework specification.
Deadline:	As indicated on Moodle.

Unit Learning Outcomes being Assessed:

1) Design and implement object oriented applications and understand principles relating to interactive user interface development

2) Use tools and techniques and utilise existing classes and libraries in a systematic way to develop a complex software application

It is your responsibility to ensure that your work is complete and available for assessment by the date given on Moodle. If submitting via Moodle, you are advised to check your work after upload; and that all content is accessible. Do not alter after the deadline. You should make at least one full backup copy of your work.
Penalties for late hand-in: see Regulations for Undergraduate Programmes of Study:

http://www.mmu.ac.uk/academic/casqe/regulations/assessment-regulations.php

The timeliness of submissions is strictly monitored and enforced.

Plagiarism: Plagiarism is the unacknowledged representation of another person's work, or use of their ideas, as one's own. MMU takes care to detect plagiarism, employs plagiarism detection software, and imposes severe penalties, as outlined in the Student Handbook
http://www.mmu.ac.uk/academic/casqe/regulations/docs/policies_regulations.pdf

and Regulations for Undergraduate Programmes
http://www.mmu.ac.uk/academic/casqe/regulations/assessment.php

Exceptional Factors affecting your performance: see Regulations for Undergraduate Programmes of Study :
http://www.mmu.ac.uk/academic/casqe/regulations/assessment/docs/ug-regs.pdf

Assessment Criteria:	Indicated in the attached assignment specification.
Formative Feedback:	Formative feedback will be provided in laboratory sessions with an assignment check point
Summative Feedback	Written feedback in the form of a commented mark grid will be provided to each student and marks made available via Moodle
Weighting:	This Assignment is weighted at 50% of the total unit assessment.

Introduction

The unit is 100% coursework based, and has two components:

1. A Java programming assignment covering learning outcomes 1 and 2 (1CWK50) worth **50%** of the overall unit mark, and

2. An Android assignment covering learning outcomes 3 and 4 (2CWK50) worth **50%** of the overall unit mark

Aim

This unit covers concepts relating to advanced object-oriented program design, the use of framework libraries, web server and mobile application development.

The aim of this assignment is to develop a RESTful web service in Java which allows requests to be made to a back-end SQLite database storing fictitious student data. The JavaScript Object Notation (JSON) format should be used for data exchange. This assignment will give you practical experience of Java development using several technologies that would be required to carry out the role of a Java developer and so will enhance your employability skills for future job applications.

The Brief

You are required to implement your own RESTful web service in Java using HTTP Server which allows a user to make CRUD (i.e. Create, Retrieve, Update and Delete) database operations on a back-end SQLite database storing fictitious student data. The default URL should be http://localhost:8005/. The JavaScript Object Notation (JSON) format should be used for data exchange. CRUD operations should be accessible through structured URLs. For example, http://localhost:8005/get-json should return all student information stored in the database in JSON format which should resemble that below.

[{"studentNumber":14,"courseTitle":"Software
Engineering","startDate":"01-10-
2016","bursary":500.0,"email":"alan@mail.com","name":"Alan
Crispin","gender":"M","dob":"14-06-
1969","address":"Manchester","postcode":"M1 5GD"},

{"studentNumber":15,"courseTitle":"Computer
Science","startDate":"01-10-
2016","bursary":500.0,"email":"kaleem@mail.com","name":"Mohammed
Kaleem","gender":"M","dob":"14-06-
2000","address":"Manchester","postcode":"M1 5GD"},

{"studentNumber":16,"courseTitle":"Computer
Science","startDate":"01-10-
2016","bursary":500.0,"email":"mary@mail.com","name":"Mary
Poppins","gender":"F","dob":"14-06-
1949","address":"London","postcode":"SW1 5MP"}]

All operations should be fully tested using a Java web service tester class and REST client such as Firefox RESTClient (used for the screenshots below) or Google Postman.

You are required to add additional functionality to the web service (for extra marks) as follows:-

1) Provide an additional RESTful route to retrieve a single student record based on a given student number

2) Add back-end server code to validate all fields (name, gender, email, postcode, start date, course title, bursary etc.) before inserting a new record into the database to ensure that data being entered into the database is free from errors

3) Add an access token security feature to allow only users with a valid token to access the web service and use its features (for this feature you can hard code the token key into the database)

4) Provide an additional route to allow a user to login and retrieve a token

In addition to the above your code should be fully documented using Javadoc.

The following provides a step by step guide to the assignment but you can tackle it in your own way if you so wish.

Step 1. Create the SQLite database

Using the SQLite Manager Mozilla Add-on tool (or similar tool) create a basic database called “studentdb.sqlite” with a single table called “students” to contain the essential data needed for a student record system as shown Fig.1. For further guidance on how to do this please refer to the lecture notes and the screen cast showing how to create an SQLite database and use Java to connect to it.

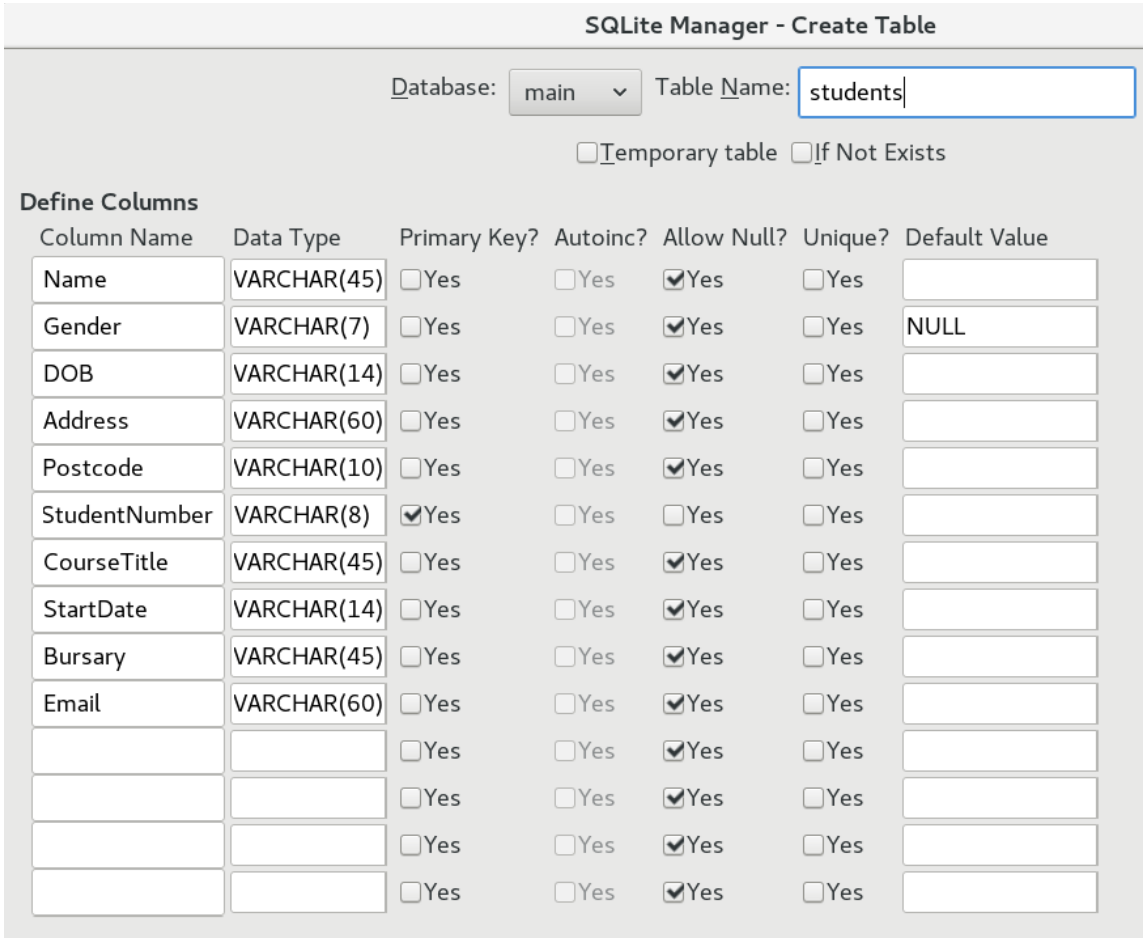

Fig.1. Creating the table called “students” in the SQLite database

The database stores the student name, gender, date of birth (DOB), address, postcode, student number, course title, start date, bursary and email. Enter some fictitious records for testing purposes as shown in Fig 2.

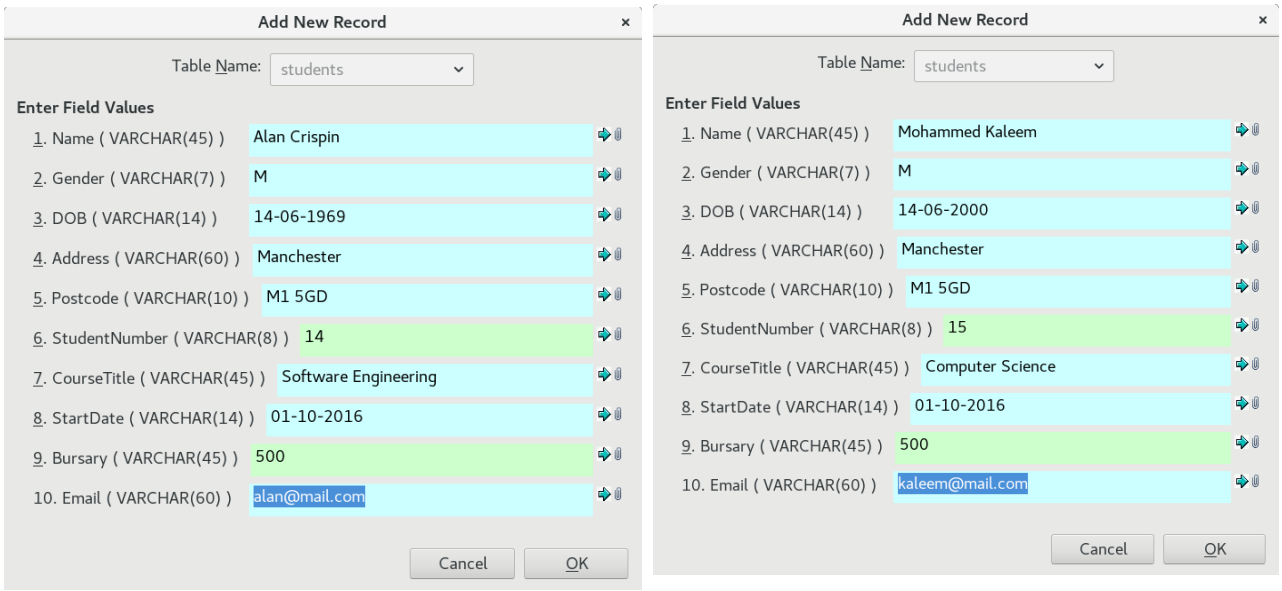

Fig.2. Adding fictitious records to the students table

Step 2. Base classes

The assignment requires that a set of base classes (Person, Student, StudentDAO) are developed to underpin the student record system. The minimum specification for base classes used for the server implementation is illustrated in Fig 2.

Fig.2. Base classes that underpin the student record system

The Person class defines a set of attributes for a Person such as their name, gender, date of birth (DOB) etc. The Student class extends the Person class and should define attributes associated with a student e.g. student number, course title being studied, start date when student enrolled, any bursary awarded etc. The StudentDAO class is the data access object (DAO) that provides an interface to the SQLite database and should contain the create, read, update and delete (CRUD) methods.

Step 3. Testing CRUD operations at the console

This stage of the assignment requires that you write a class called “DatabaseTester” to test each of the CRUD (i.e. Create, Retrieve, Update and Delete) database operations developed for the StudentDAO class to check their functionality. To implement the CRUD operations you should use standard SQL statements (e.g. SELECT, INSERT, DELETE).

Step 4. RESTful route for retrieving all student records

This stage of the assignment requires that you write server code (using the HttpServer class) which will accept client GET requests and return student data in JSON format. The server will connect to an SQLite database that contains student information and return that data to the requesting clients. The default URL (http://localhost:8005/get-json) should output all student records stored in the database. You should test that the server code providing the web service is working correctly by using a REST client to perform a GET request as shown in Fig 3.

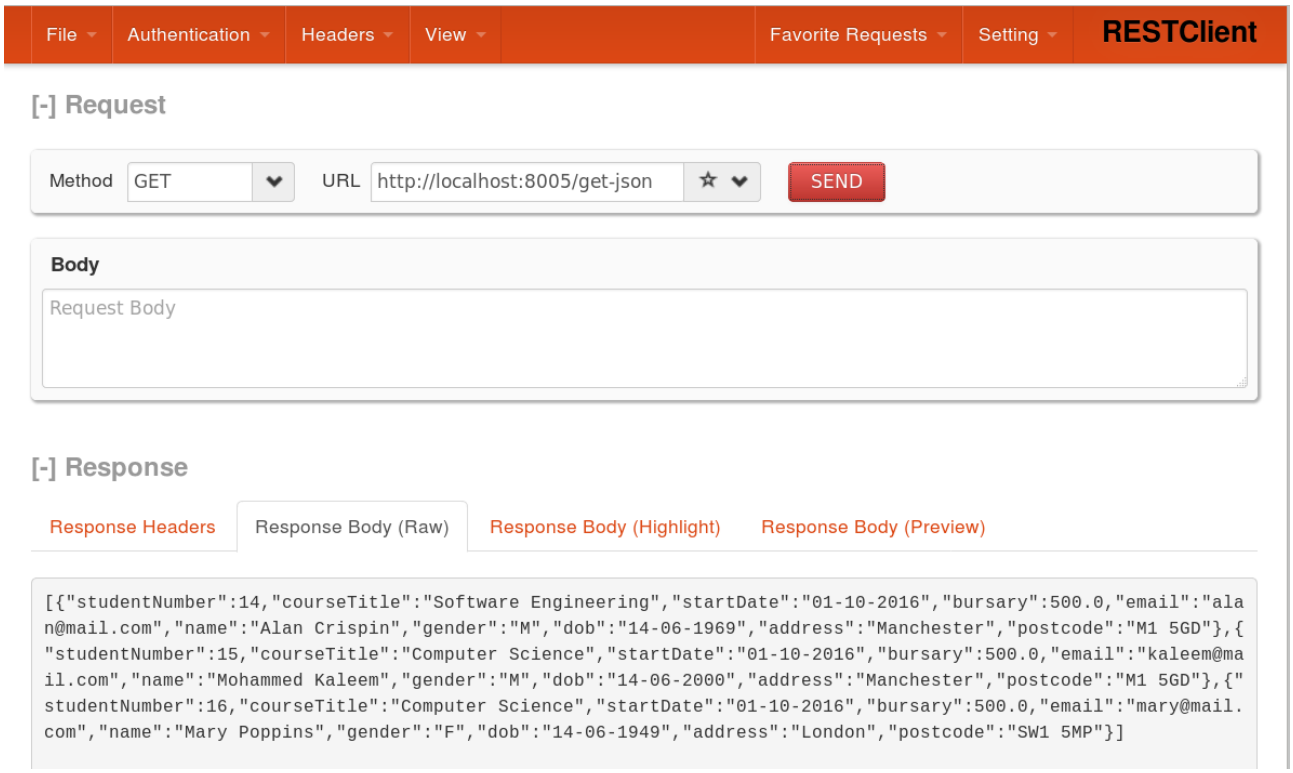

Fig.3. Firefox RESTClient used for testing the HTTP GET request

The response header status code should be 200 OK.

Step 5. Web service tester class

Write a Java web service tester class called “WebServiceTester” to test the GET operation at the console as shown below.

```
public class WebServiceTester {

public static void main(String[] args) {
      System.out.println("Students = " + getStudents());
}

private static StringBuffer getStudents() {
StringBuffer response = new StringBuffer();
try {
```

```
      URL url = new URL("http://localhost:8005/get-json");
      BufferedReader reader = new BufferedReader(new
InputStreamReader(url.openStream()));
      String output;
      while ((output = reader.readLine()) != null) {
            response.append(output);
      }
      reader.close();

} catch (Exception e) {
System.out.println(e.getMessage());
}
return response;
}
```

Step 6. Restful route to create a new student record from a JSON object

Now write HTTP server code for a POST request. A JSON object containing student information should be posted to the server as shown in Fig 4.

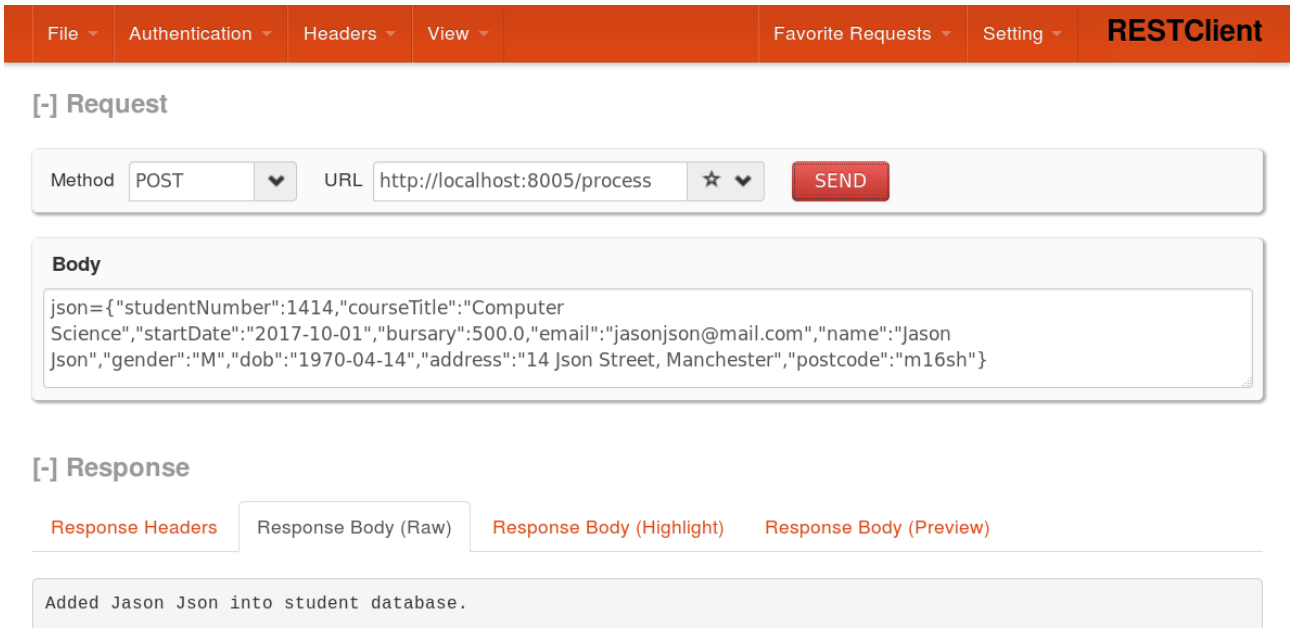

Fig.4. Firefox RESTClient testing HTTP POST with JSON

The response header status code should be 200 OK.

Extend the “WebServiceTester” class to include code for testing the POST client request.

Step 7. RESTful routes for updating and deleting student records

The HTTP server code should be extended to include other routes for updating a student record and deleting a student record. Extend the “WebServiceTester” class to include code for testing these requests.

Step 8. Add an additional route to allow a single student record to be selected and viewed based on student number. Extend the "WebServiceTester" class to include code for testing the search request.

Step 9. Add validation error checking. Error checking can be performed by using regular expressions to ensure that data being entered into the database (via the web service) is valid. For example, the name field should only contain characters and spaces and no numbers, special characters or punctuation marks etc. Use regular expressions (regex) to validate an email address and to match valid UK postcodes.

Step 10. In a real system a user would be issued with an API key (token) to grant them access to the web service. Incorporate a security access token for all RESTful routes. This is done by passing an access token (key) as a parameter in the URL string which is used to restrict access to the web service to only users with a valid token. An incorrect access key should prevent a user accessing student information and respond with a message stating that the key is not valid as shown in Fig 5.

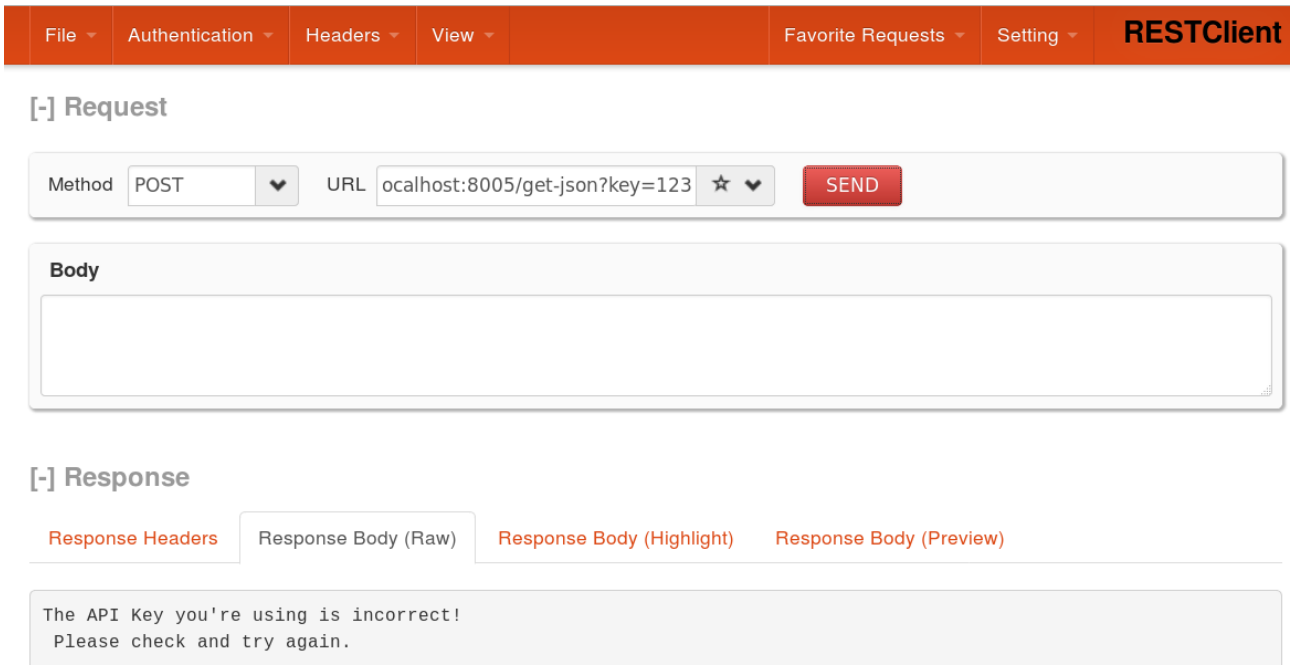

Fig.5. Users with an invalid access token key are unable to access data

A user with a valid token is able to access the web services as shown in Fig. 6. To undertake this component of the assignment you can assume that the API key (token) is hard coded into the database. Create a new table called users in the SQLite database with fields id, username, password and token. Provide an additional route to allow a user to login and retrieve a token. Comment in your report on the implementation of your security feature identifying strengths and weaknesses.

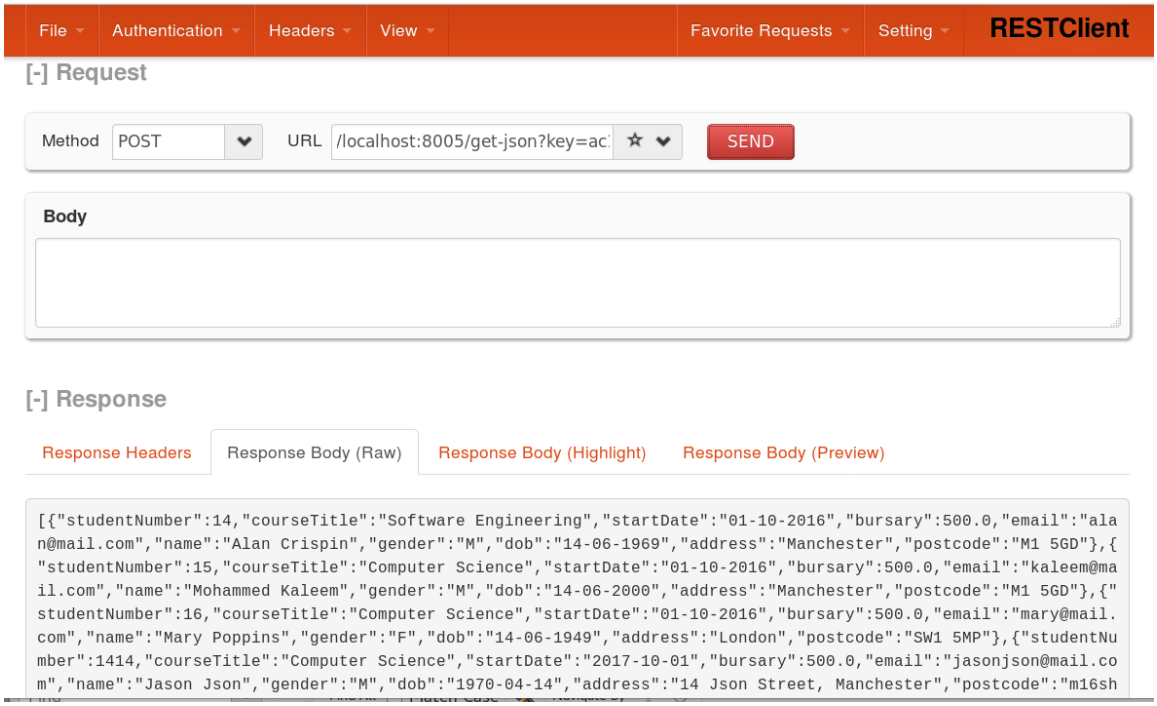

Fig.6. A user with a valid access token key is able to access data

Hand In:

You are required to up-load your project files and a short report containing screen shots of your completed application with a narrative explaining which features have been implemented successfully. Also provide an annotated marking grid showing which features you have completed. You should submit a complete zipped copy of your source code (i.e. the bases classes, DAO class, database tester class, server code, web service tester class, SQLite database, jar files etc.) and your report with the format

surname_studentnumber.zip

Please ensure that all code submitted compiles and runs at the command line and your name and student number are included as comments in your source code files.

Screencasts: to support this assignment include those on Java syntax, classes and objects, inheritance and polymorphism, Java Database Connectivity (JDBC), Java web programming and HTTPServer.

Feedback: Marks will be made available via Moodle with a turnaround time in accordance with University guidelines. Written feedback in the form of a commented mark grid will be provided to each student by tutors.

Dr Alan Crispin (19-10-17)

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 StudentDAO class	0-4 marks	No/non-functioning attempt at a server	0-3 marks	No/little attempt at advanced features	0-3 marks	Code is **poorly** or **moderately** presented, with **some** Javadoc documentation, and **occasional** comments.	0-3 marks
Evidence of an attempt at base class Person and tested using a class called Tester	5-9 marks	CRUD methods partially implemented	5-9 marks	Server connects to the SQLite database and RESTful route for reading all student information (GET) and output in JSON format	4-8 marks	Validation (using regular expressions) of all fields before inserting new records into server database (name, gender, email, postcode, start date, course title, bursary)	4-10 marks	Code is **well** presented, with **good** Javadoc documentation and **sensible** comments	4-8 marks
Base classes Person, Student and Tester partially implemented	10-14 marks	CRUD methods fully implemented	10-14 marks	RESTful route to create a student record by posting a JSON object	9-13 marks	Access token code (hard coded into database) fully implemented on all routes	11-14 marks	**Report** containing screen shots of completed application	9-15 marks
				RESTful routes to update and delete student records	14-16 marks				
				RESTful route to retrieve a student record based on student ID	17-20 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"	21-25 marks	Route for login where user can register and retrieve an API key	15-20 marks		

For example, a student earning 20 marks in *Base Classes*, 20 marks in *Database Connectivity*, 10 marks each for their *RESTful Web Service (Server)* and *Advanced Features* and 5 marks for *Code Quality/Report* would achieve an overall mark of 65% (20 + 20 + 10 + 10 + 5). Note that in each area you must work upwards in terms of features implemented to achieve the maximum mark for that area. If you need any help in understanding this assignment please talk to the tutor who takes you for your laboratory sessions or arrange to see either Dr Alan Crispin or Dr Mohammed Kaleem during their office hours.