Advanced Data Centric Web Applications

RESTful API (Server and Client)

1 Description

Write a Spring Boot MVC application that implements a RESTful API and an Angular application that accesses the API.

2 Marks

This project is worth 50% of the marks for the module as follows:

- Server Side 65%
- Client Side 25%
- Innovation 10% (extra functionality, exceeding the requirements listed in this document).

2.1.1 Plagiarism

Plagiarism will be dealt with in accordance with the university's Student Code.

NOTE: Students may be invited to an MS Teams meeting for a viva explanation of any or all parts of their submission.

3 Submission

A zipped file named GXXXXXXXX (which is your student number) containing 3 items:

- SGXXXXXXXX (where GXXXXXXXX is your student number) containing the Spring Boot application.
- CGXXXXXXXX (where GXXXXXXXX is your student number) containing the Angular application.
- A file called *Innovation.pdf* which contains a description of any innovation in your project (optional).

should be uploaded to Moodle before 5:00pm on Tuesday May 2nd 2023.

NOTE: Submissions will be tested on the modules's Virtual Machines (username = adcwa2022, password = ADCwa2022). So, if you are developing on your own machines, ensure that server and client both run on the VM.

4 Server-Side Application (API)

Download SGXXXXXXXX from Moodle, rename it to your student number e.g. SG12345678 and use it as the basis for the server-side application.

The Model for the application consists of three classes:

- Lecturer
- Module
- Student

These classes have been set up with the correct relationship mappings so that when the skeleton application is run the database will be setup and the tables populated according to *import.sql*.

4.1 Lecturer API endpoints

4.1.1 GET /lecturers

Returns an array of JSON objects containing the *id*, *lid*, *name*, *taxBand* and *salaryScale* of all Lecturers.

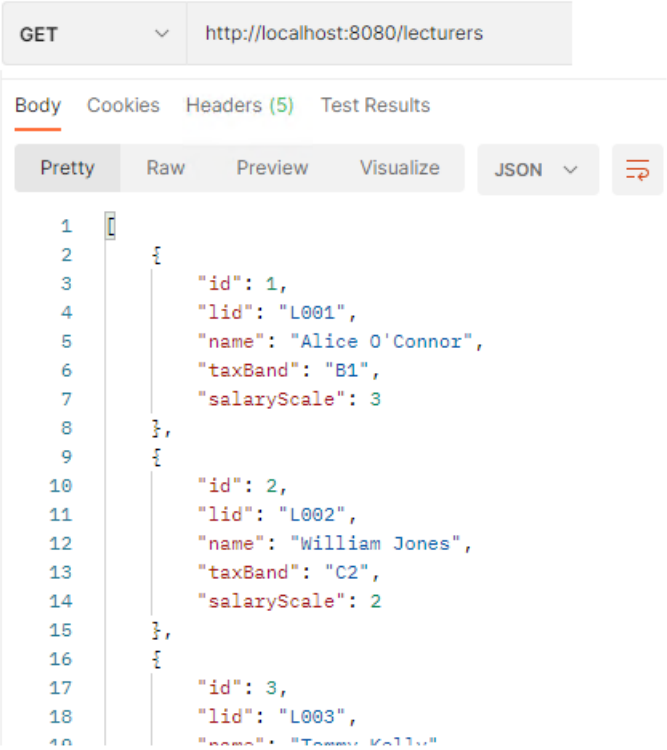


Figure 1 All Lecturers

4.1.2 POST /lecturer

Persists the Lecturer object specified in the request body to the database.

The Lecturer object must contain:

- *lid*
- *name*

The Lecturer object must not contain:

- *id*

The Lecturer object may contain:

- *taxBand*
- *salaryScale*

You can assume that the request body will never contain *modules*, so no error message is needed for this attribute.

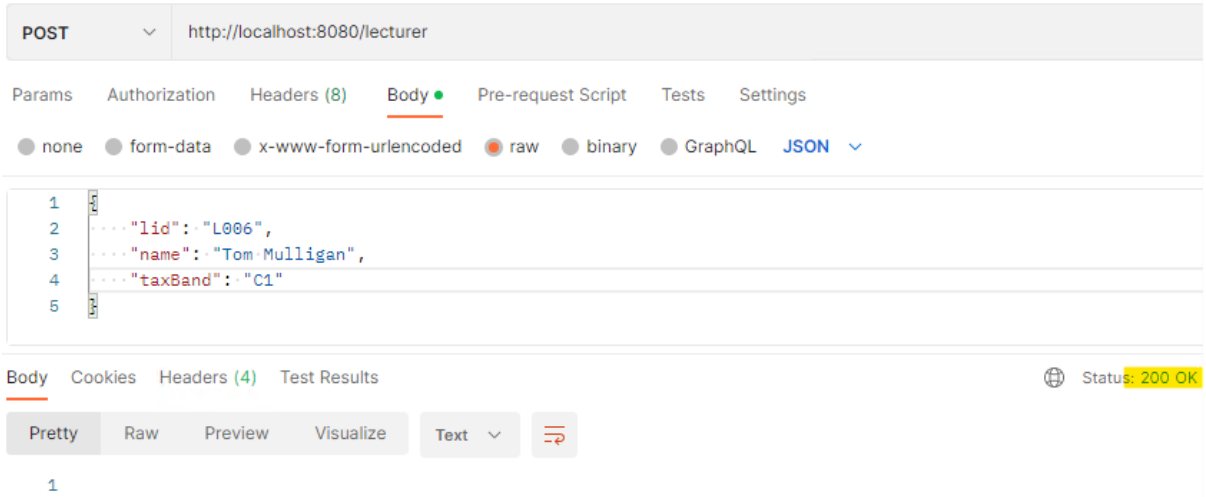
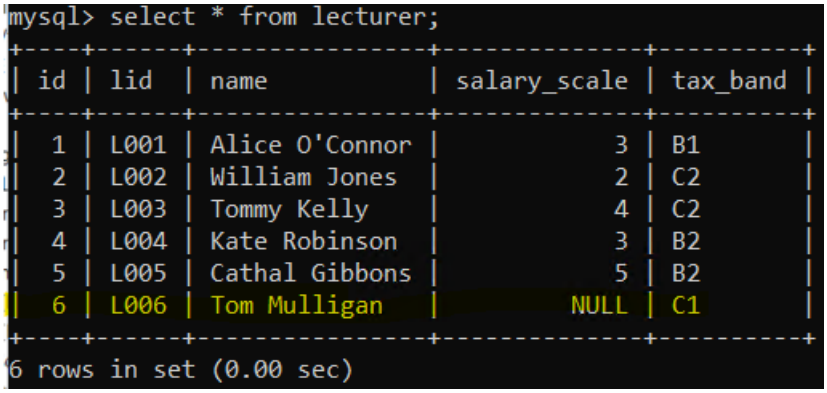


Figure 2 Valid Lecturer object for POST



```
mysql> select * from lecturer;
+----+------+----------------+--------------+----------+
| id | lid  | name           | salary_scale | tax_band |
+----+------+----------------+--------------+----------+
|  1 | L001 | Alice O'Connor |            3 | B1       |
|  2 | L002 | William Jones  |            2 | C2       |
|  3 | L003 | Tommy Kelly    |            4 | C2       |
|  4 | L004 | Kate Robinson  |            3 | B2       |
|  5 | L005 | Cathal Gibbons |            5 | B2       |
|  6 | L006 | Tom Mulligan   |         NULL | C1       |
+----+------+----------------+--------------+----------+
6 rows in set (0.00 sec)
```

Figure 3 Lecturer object persisted to database

4.1.2.1 Error Conditions

If any attributes that should not be present in the body of the POST, an appropriate error message, or error messages, are returned with at HTTP status of 500.

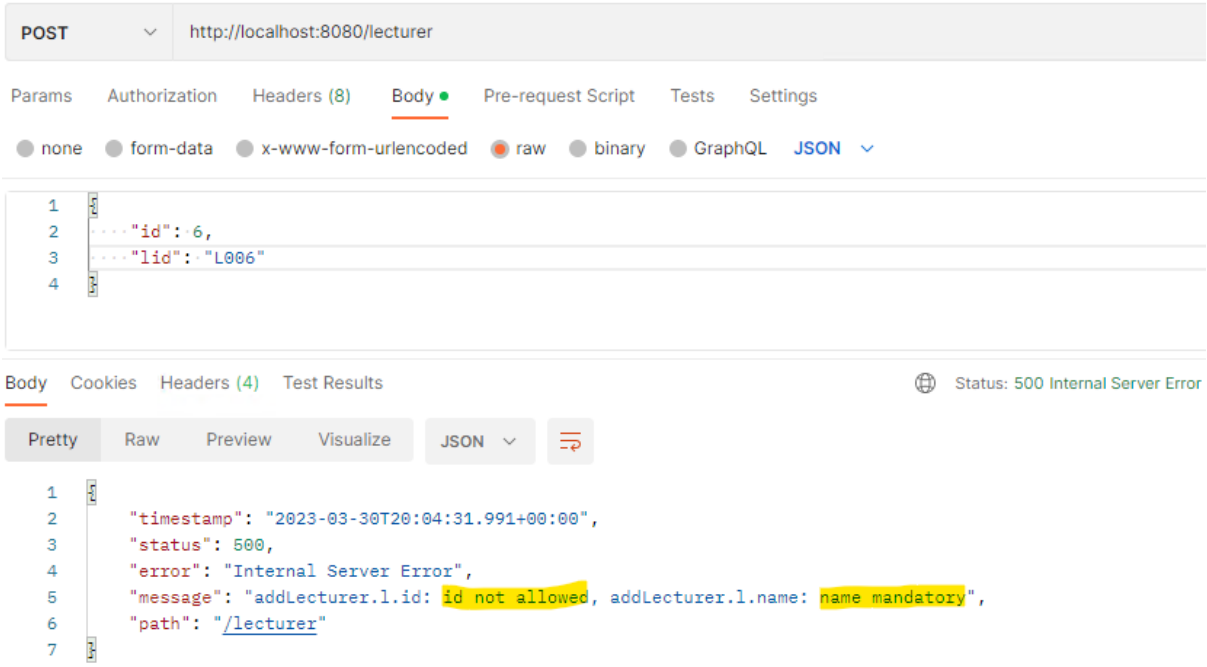


Figure 4 Invalid POST attributes for Lecturer

- If an employee with the specified *lid* already exists, a HTTP 403 response should be returned with the following error message: "*Lecturer: {lid} already exists*" where {lid} is the lid in the request body.

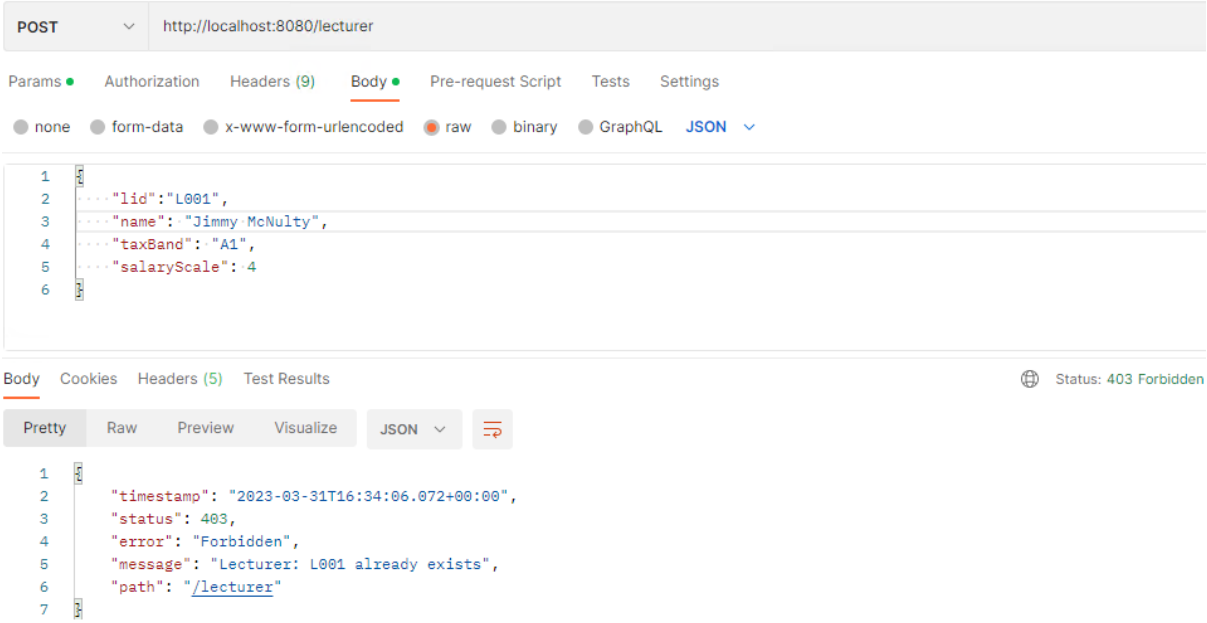


Figure 5 Lecturer with specified lid already exists

4.1.3 PUT /lecturer/{lid}

Updates an existing Lecturer object in the database.

The Lecturer object must contain:

- *name*

The Lecturer object must not contain:

- *id*
- *lid*

The Lecturer object may contain:

- *taxBand*
- *salaryScale*

You can assume that the request body will never contain *modules*, so no error message is needed for this attribute.

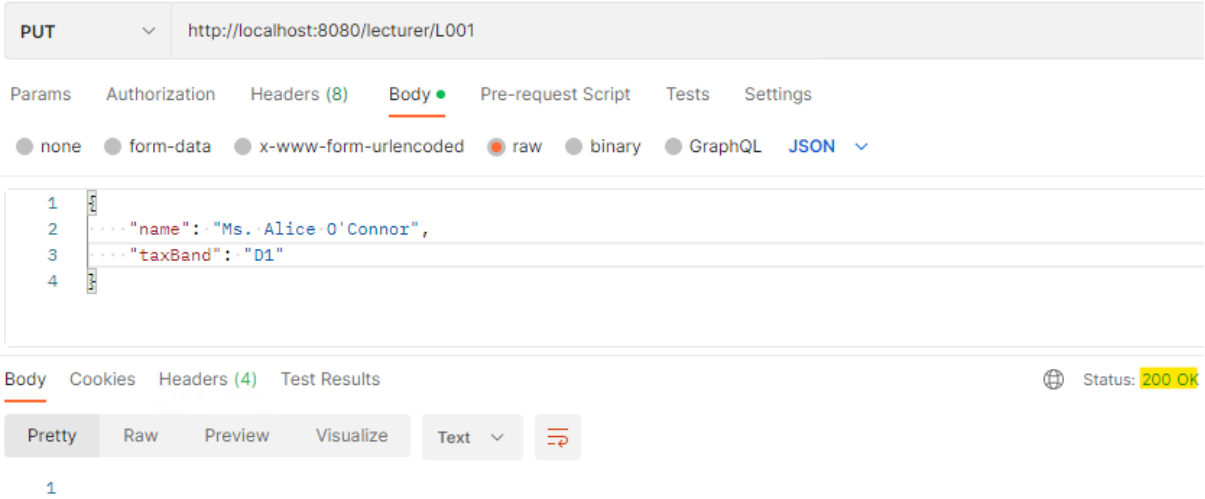


Figure 6 Figure 2 Valid Lecturer object for PUT

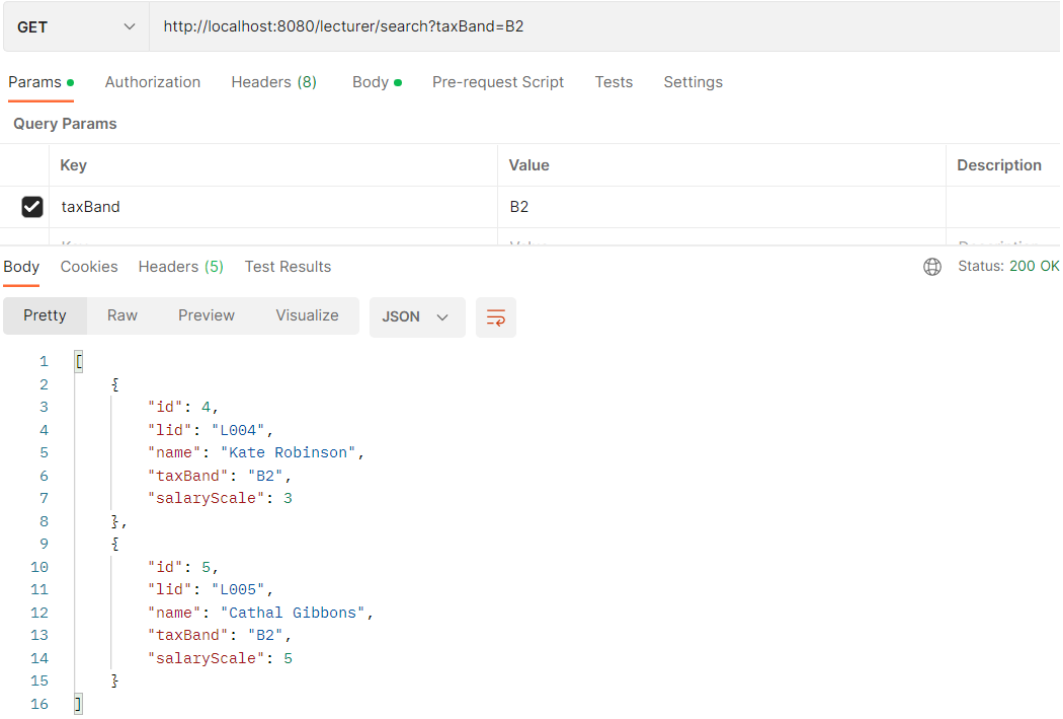
```
mysql> select * from lecturer;
+----+------+--------------------+--------------+----------+
| id | lid  | name               | salary_scale | tax_band |
+----+------+--------------------+--------------+----------+
|  1 | L001 | Ms. Alice O'Connor |            3 | D1       |
|  2 | L002 | William Jones      |            2 | C2       |
|  3 | L003 | Tommy Kelly        |            4 | C2       |
|  4 | L004 | Kate Robinson      |            3 | B2       |
|  5 | L005 | Cathal Gibbons     |            5 | B2       |
|  6 | L006 | Tom Mulligan       |         NULL | C1       |
+----+------+--------------------+--------------+----------+
6 rows in set (0.00 sec)
```

Figure 7 Lecturer updated in database

4.1.4 GET /lecturer/search

Allows the user to search for Lecturers based on the following parameters. This requirement must be implemented using a Native Query.

- *taxBand* – Returns all Lecturers whose *taxBand* is equal to this parameter.



- *taxBand* – Returns all Lecturers whose *taxBand* is equal to this parameter *salaryScale – And* who are on a *salaryScale* greater than this parameter.

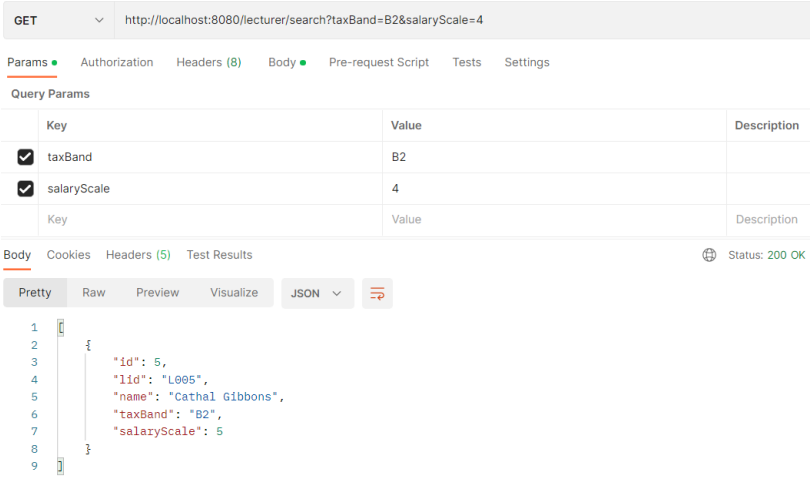


Figure 8 Employee not deleted – working on a project.

4.2 Student API endpoints

4.2.1 GET /students

Returns an array of JSON objects containing the *id*, *sid*, *name*, and *modules* of all Students.

The *modules* should show *id, mid, name, credits, level* and *lecturer.*

The *lecturer* should show *id, name, taxBand* and *salaryScale*.

GET http://localhost:8080/students

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize JSON


```
1  [
2      {
3          "id": 1,
4          "sid": "G001",
5          "name": "Tom Higgins",
6          "modules": [
7              {
8                  "id": 1,
9                  "mid": "JAV",
10                 "name": "Java Programming",
11                 "credits": 10,
12                 "level": 7,
13                 "lecturer": {
14                     "id": 2,
15                     "lid": "L002",
16                     "name": "William Jones",
17                     "taxBand": "C2",
18                     "salaryScale": 2
19                 }
20             },
21             {
22                 "id": 2,
23                 "mid": "DB",
24                 "name": "Databases",
25                 "credits": 5,
26                 "level": 8,
27                 "lecturer": {
28                     "id": 2,
29                     "lid": "L002",
30                     "name": "William Jones",
31                     "taxBand": "C2",
32                     "salaryScale": 2
```


Figure 9 All Students

4.2.2 DELETE /students/{sid}

Deletes the Student whose *sid* is equal to the *sid* in the URL and returns a HTTP 200 response.

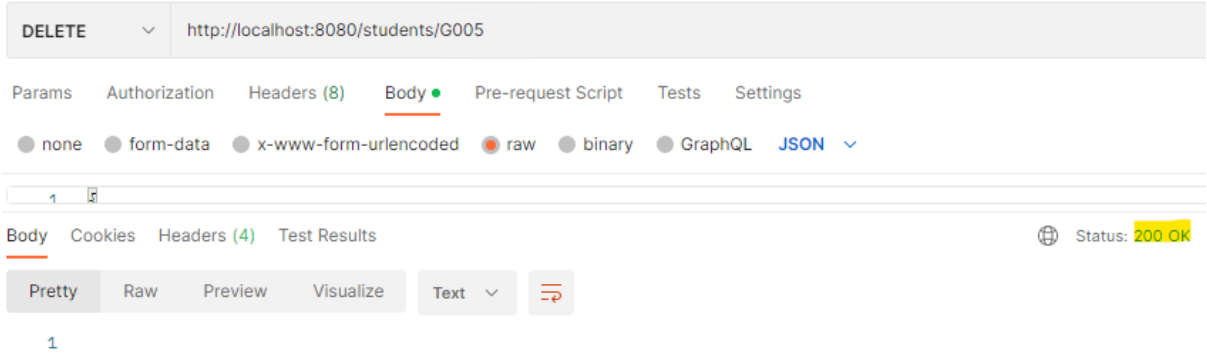


Figure 10 Successfully deleted Student

```
mysql> select * from student;
+----+-----------------+------+
| id | name            | sid  |
+----+-----------------+------+
|  1 | Tom Higgins     | G001 |
|  2 | Mary Murphy     | G002 |
|  3 | Alan Fitzgibbon | G003 |
|  4 | Bart O'Malley   | G004 |
+----+-----------------+------+
4 rows in set (0.00 sec)
```

Figure 11 Student G005 deleted from database

4.2.2.1 Error Conditions

- If the Student to be deleted has associated Modules, he/she should not be deleted. Instead, a HTTP status 500 response should be returned with a message indicating the reason the delete could not be performed.

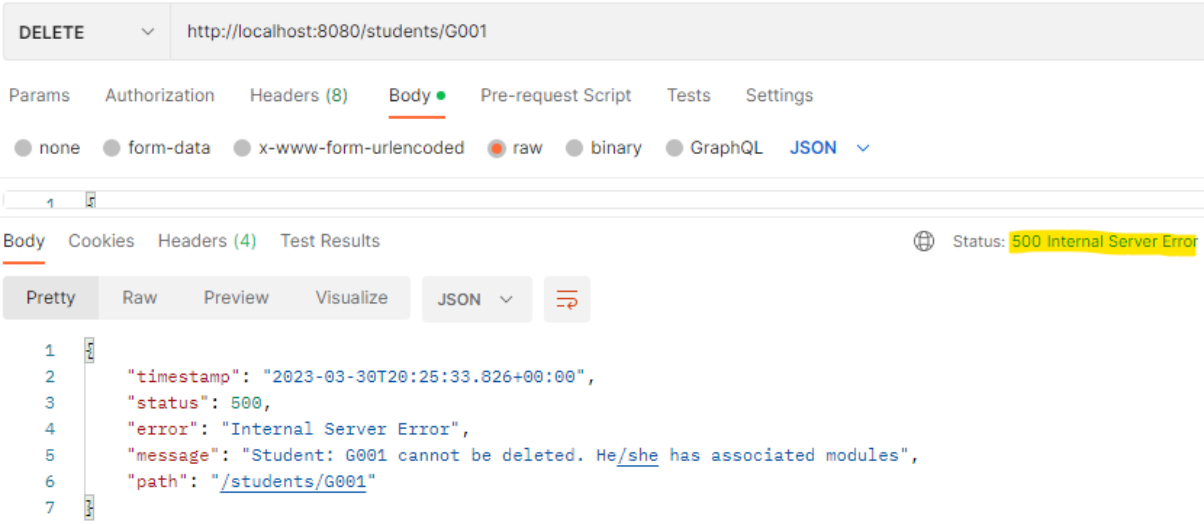


Figure 12 Cannot delete Student. He/she has associated modules.

- If the Student to be deleted does not exist, a HTTP 500 response with an appropriate error message is returned.

5 Client-Side Application

Create an Angular application using the following command:

```
ng new CGXXXXXXXX --routing=true
```

(**REMINDER**: As mentioned in Submission section, your application will be tested on the VM. If you have different versions of NPM/Angular on your own machine, this could cause errors when the application is tested on the VM. It is your responsibility to ensure your application works on the VM).

5.1 Main Screen

The main screen should have two links:

- Lecturers
- Students

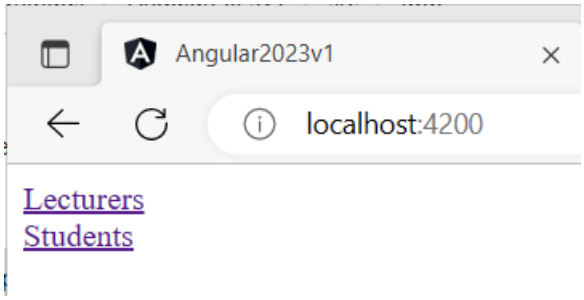


Figure 13 Main Screen

5.2 Lecturers

When the *Lecturers* route is selected, the *lid*, *name, taxBand* and *salaryScale* of all Lecturers are fetched via the API and shown along with an *Update* button.

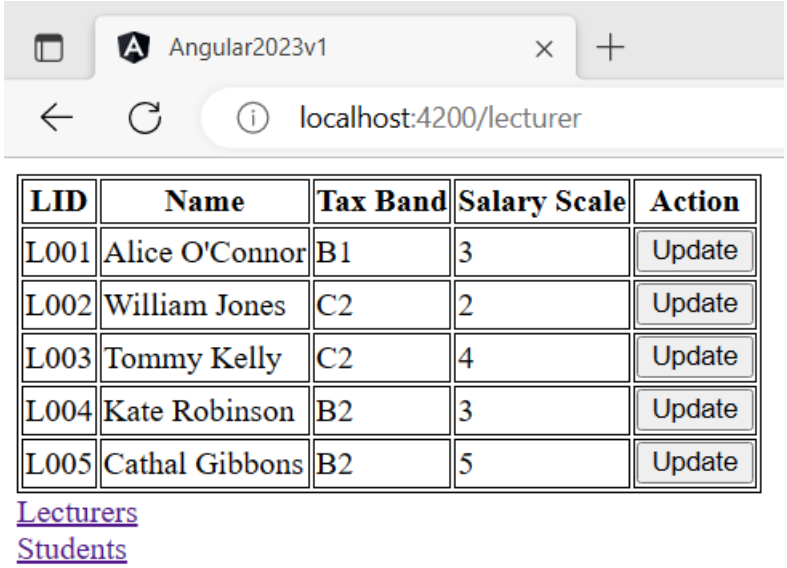


Figure 14 Lecturers

5.3 Update Lecturer

When the *Update* button is pressed beside a Lecturer his/her details on a new page.

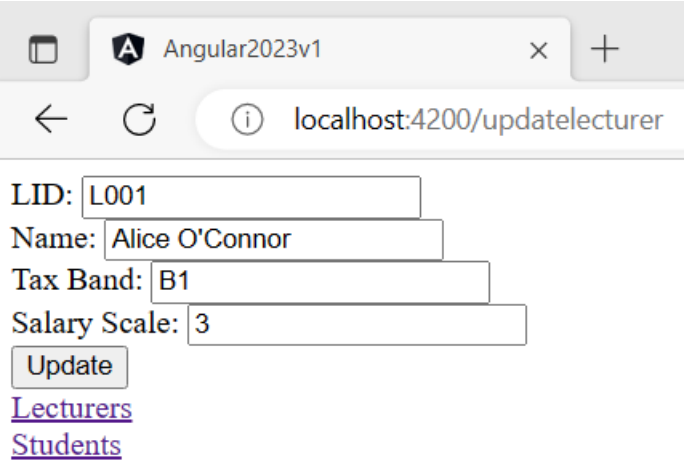


Figure 15 Lecturer details shown.

All these details except *lid* can be edited.

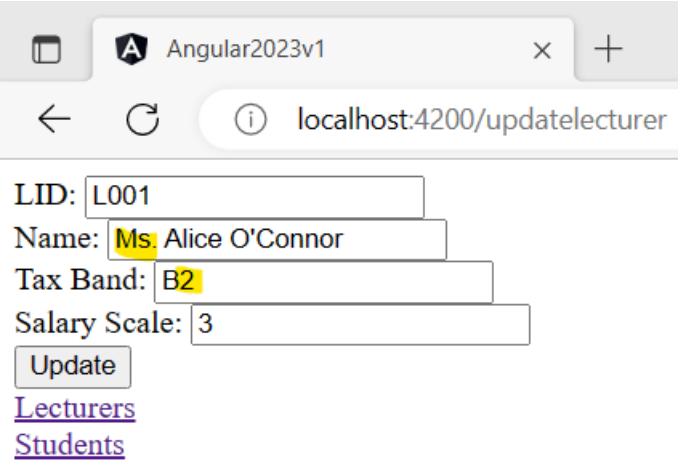


Figure 16 Updated Lecturer details

When *Update* is pressed a PUT request with the details is sent to the server. If everything is OK, the user is redirected to the Lecturers route, where the list of lecturers is fetched from the database and shown.

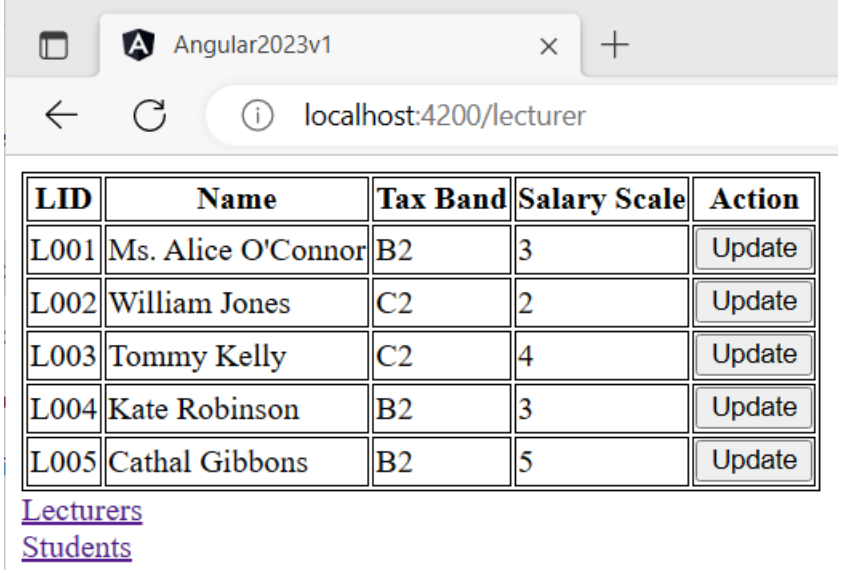


Figure 17 Updated Lecturers List

5.3.1.1 Error Conditions

After a Lecturer's details have been displayed, but before the *Update* button is pressed, the Lecturer record may have been deleted from database. In this case an error message should be displayed.

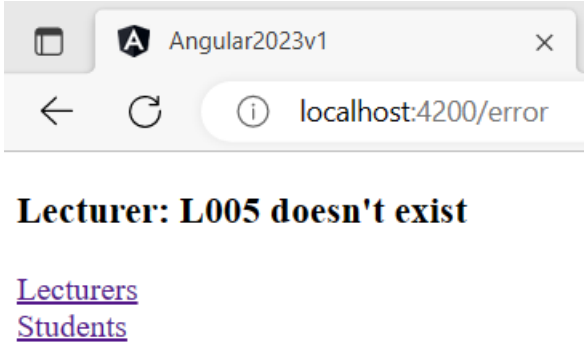


Figure 18 Lecturer doesn't exist

5.4 Students

When the *Students* route is selected, the *sid*, and *name* of all Students are fetched via the API and shown along with a *Delete* button.

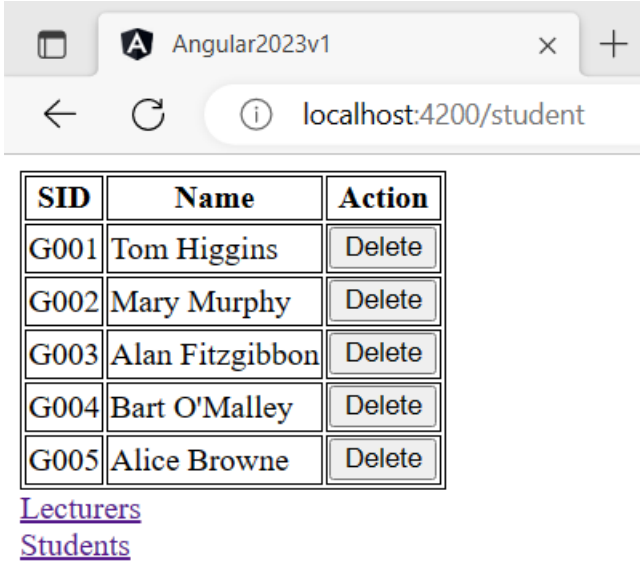


Figure 19 Departments

5.4.1 Delete Student

When the *Delete* button is pressed, a DELETE HTTP request is sent to the server. If the request was successful the appropriate Student will have been deleted from the database.

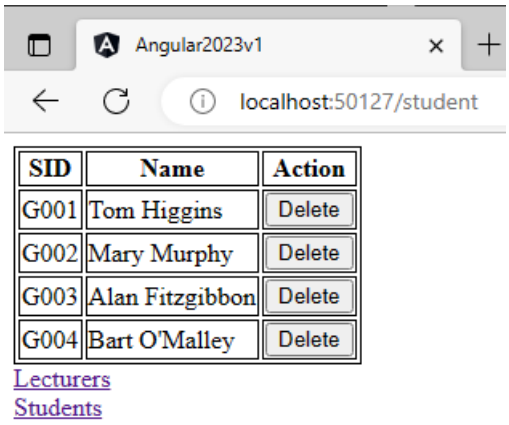


Figure 20 Student G005 deleted

5.4.1.1 Error Conditions

If the server cannot delete a Student the error message returned from the server should be shown.

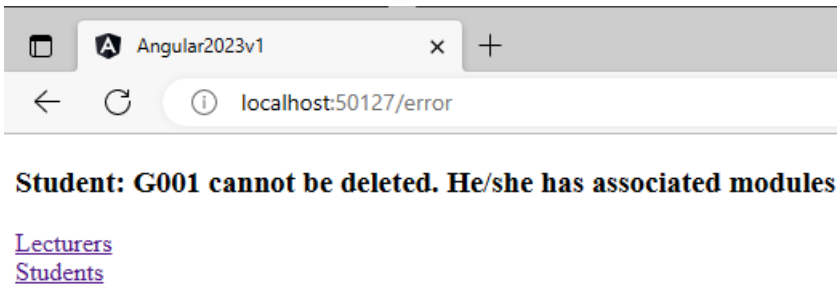


Figure 21 Cannot delete Student G001