

Advanced Data Centric Web Applications

RESTful API (Server and Client)

1	Description	3
2	Marks	3
2.1.1	Plagiarism	3
3	Submission	3
4	Server-Side Application (API)	4
4.1	Lecturer API endpoints.....	5
4.1.1	GET /lecturers	5
4.1.2	POST /lecturer	6
4.1.2.1	Error Conditions	7
4.1.3	PUT /lecturer/{lid}.....	9
4.1.4	GET /lecturer/search.....	10
4.2	Student API endpoints	11
4.2.1	GET /students.....	11
4.2.2	DELETE /students/{sid}.....	12
4.2.2.1	Error Conditions	13
5	Client-Side Application	14
5.1	Main Screen	14
5.2	Lecturers	15
5.3	Update Lecturer	16
5.3.1.1	Error Conditions	17
5.4	Students	18
5.4.1	Delete Student	19
5.4.1.1	Error Conditions	19

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 GXXXXXXX (which is your student number) containing 3 items:

- SGXXXXXXX (where GXXXXXXX is your student number) containing the Spring Boot application.
- CGXXXXXXX (where GXXXXXXX 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:

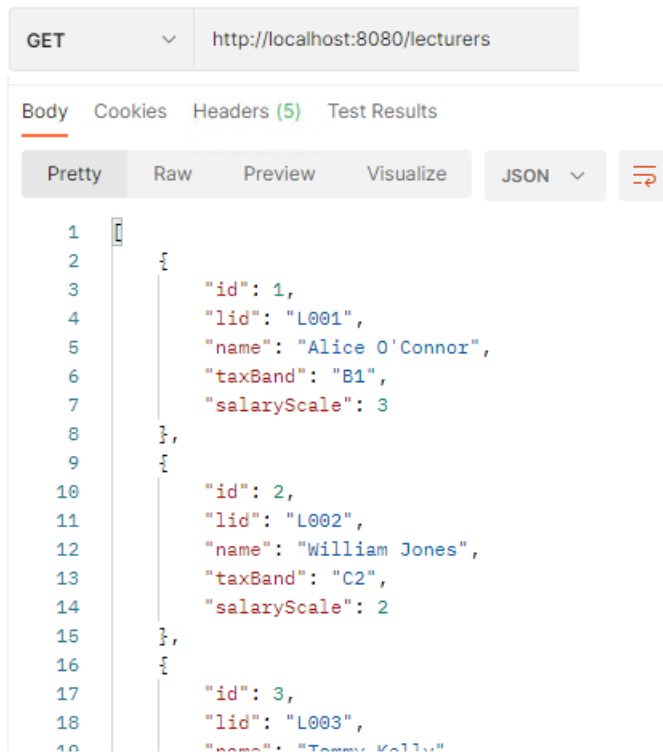
- Lecturer
- Module
- Student

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.



```
GET http://localhost:8080/lecturers

Body
Pretty Raw Preview Visualize JSON
1  {
2    "id": 1,
3    "lid": "L001",
4    "name": "Alice O'Connor",
5    "taxBand": "B1",
6    "salaryScale": 3
7  },
8  {
9    "id": 2,
10   "lid": "L002",
11   "name": "William Jones",
12   "taxBand": "C2",
13   "salaryScale": 2
14 },
15 {
16   "id": 3,
17   "lid": "L003",
18   "name": "Tommy Kelly",
19   "taxBand": "B1",
20   "salaryScale": 3
21 }
```

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.

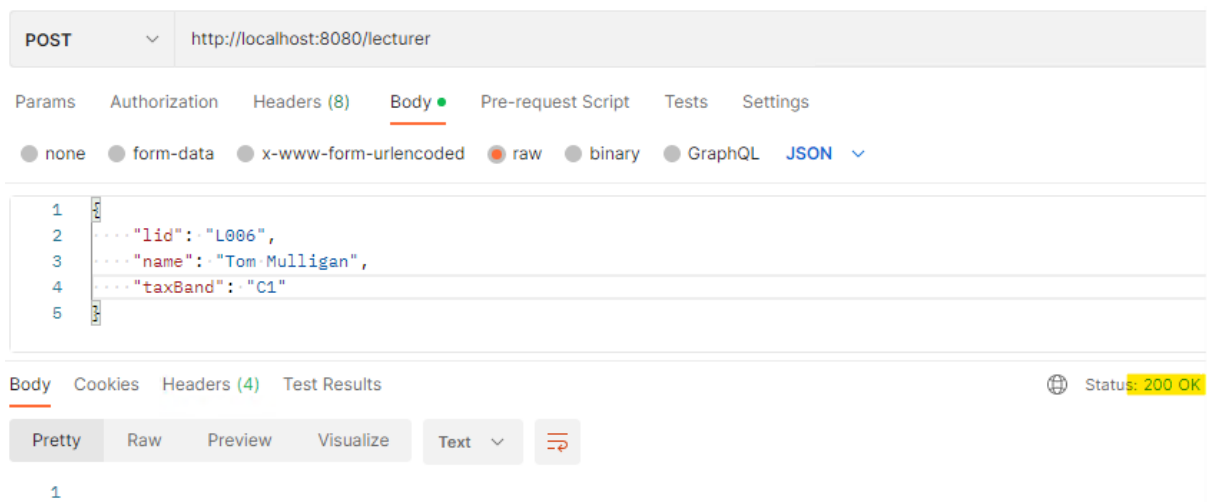


Figure 2 Valid Lecturer object for POST

```
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.

The screenshot displays a REST client interface for a POST request to `http://localhost:8080/lecturer`. The request body is a JSON object with two attributes: `"id": 6` and `"lid": "L006"`. The response status is `500 Internal Server Error`. The response body, shown in the 'Pretty' view, is a JSON object containing the following fields:

```
1 {
2   "timestamp": "2023-03-30T20:04:31.991+00:00",
3   "status": 500,
4   "error": "Internal Server Error",
5   "message": "addLecturer.1.id: id not allowed, addLecturer.1.name: name mandatory",
6   "path": "/lecturer"
7 }
```

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.

POST http://localhost:8080/lecturer

Params Authorization Headers (9) Body Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```
1 {
2   "lid": "L001",
3   "name": "Jimmy McNulty",
4   "taxBand": "A1",
5   "salaryScale": 4
6 }
```

Body Cookies Headers (5) Test Results Status: 403 Forbidden

Pretty Raw Preview Visualize JSON

```
1 {
2   "timestamp": "2023-03-31T16:34:06.072+00:00",
3   "status": 403,
4   "error": "Forbidden",
5   "message": "Lecturer: L001 already exists",
6   "path": "/lecturer"
7 }
```

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.

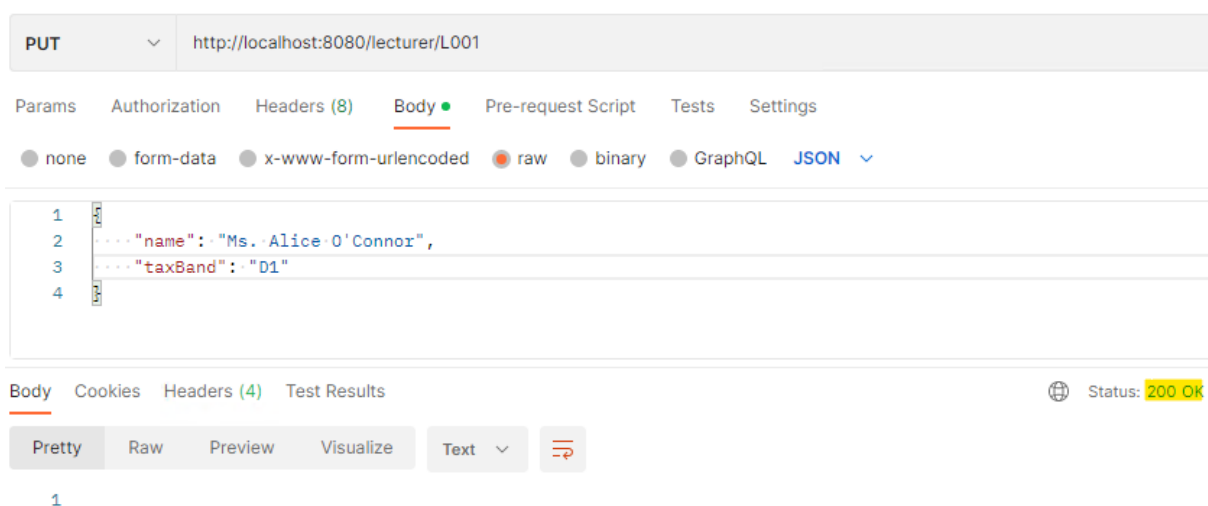


Figure 6 Figure 2 Valid Lecturer object for PUT

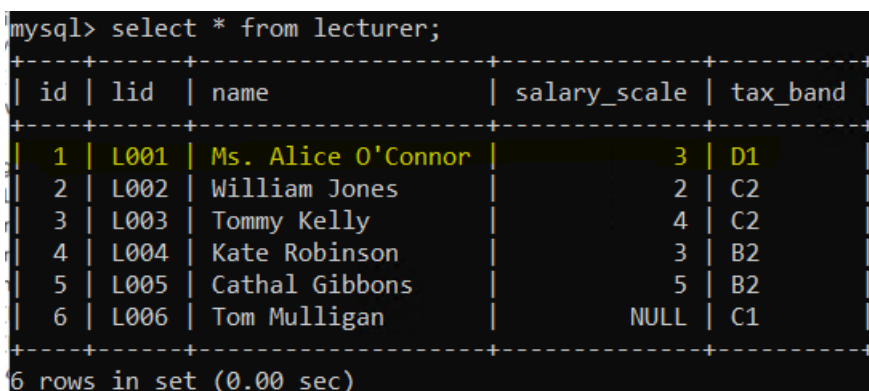


Figure 7 Lecturer updated in database

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.

GET <http://localhost:8080/lecturer/search?taxBand=B2>

Params **Authorization** Headers (8) Body **Pre-request Script** Tests Settings

Query Params

Key	Value	Description
<input checked="" type="checkbox"/> taxBand	B2	

Body Cookies Headers (5) Test Results **Status: 200 OK**

Pretty Raw Preview Visualize JSON **≡**

```
1  [
2    {
3      "id": 4,
4      "lid": "L004",
5      "name": "Kate Robinson",
6      "taxBand": "B2",
7      "salaryScale": 3
8    },
9    {
10     "id": 5,
11     "lid": "L005",
12     "name": "Cathal Gibbons",
13     "taxBand": "B2",
14     "salaryScale": 5
15   }
16 ]
```

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

GET <http://localhost:8080/lecturer/search?taxBand=B2&salaryScale=4>

Params **Authorization** Headers (8) Body **Pre-request Script** Tests Settings

Query Params

Key	Value	Description
<input checked="" type="checkbox"/> taxBand	B2	
<input checked="" type="checkbox"/> salaryScale	4	

Body Cookies Headers (5) Test Results **Status: 200 OK**

Pretty Raw Preview Visualize JSON **≡**

```
1  [
2    {
3      "id": 5,
4      "lid": "L005",
5      "name": "Cathal Gibbons",
6      "taxBand": "B2",
7      "salaryScale": 5
8    }
9  ]
```

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   {
23     "id": 2,
24     "mid": "DB",
25     "name": "Databases",
26     "credits": 5,
27     "level": 8,
28     "lecturer": {
29       "id": 2,
30       "lid": "L002",
31       "name": "William Jones",
32       "taxBand": "C2",
33       "salaryScale": 2
34     }
35   }
36 ]
```

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.

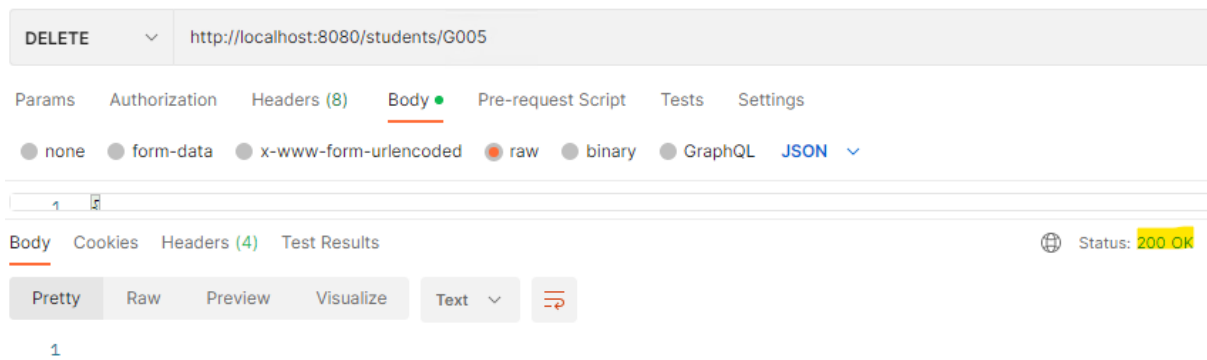


Figure 10 Successfully deleted Student

```
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.

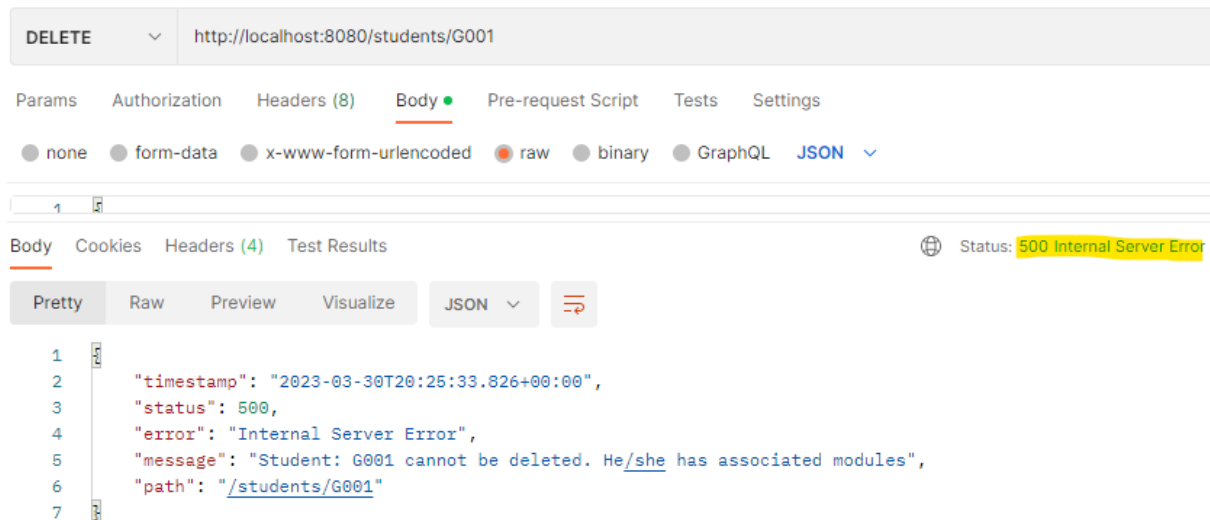
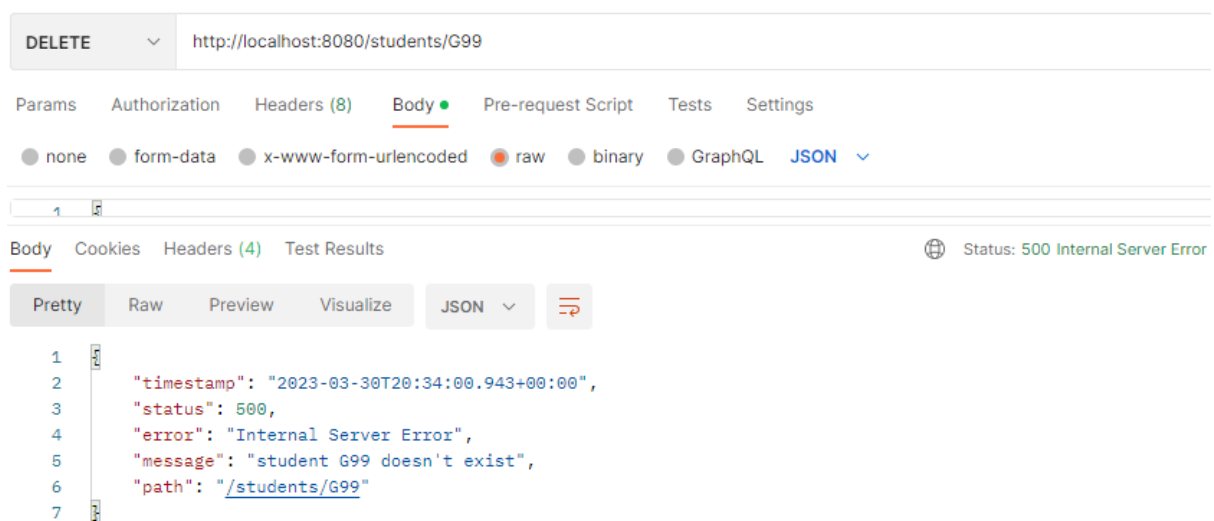


Figure 12 Cannot delete Student. He/she has associated modules.

- 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

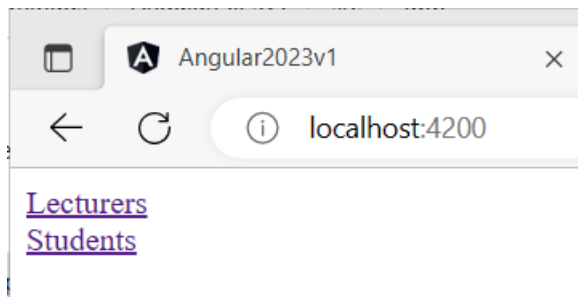
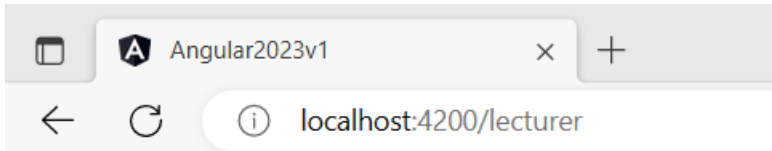


Figure 13 Main Screen

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.



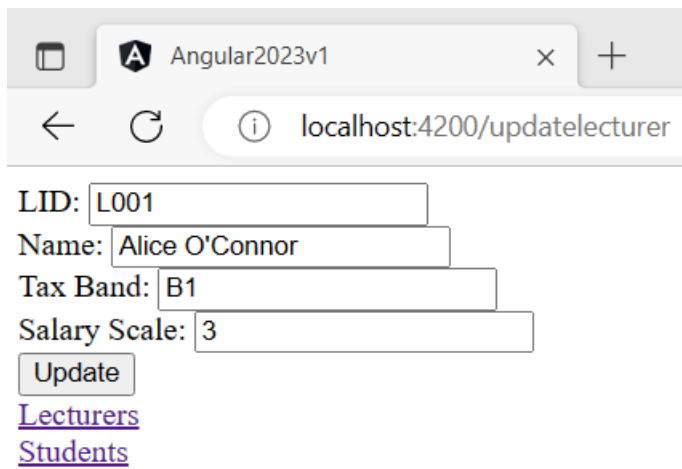
LID	Name	Tax Band	Salary Scale	Action
L001	Alice O'Connor	B1	3	<button>Update</button>
L002	William Jones	C2	2	<button>Update</button>
L003	Tommy Kelly	C2	4	<button>Update</button>
L004	Kate Robinson	B2	3	<button>Update</button>
L005	Cathal Gibbons	B2	5	<button>Update</button>

[Lecturers](#)
[Students](#)

Figure 14 Lecturers

5.3 Update Lecturer

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



LID:

Name:

Tax Band:

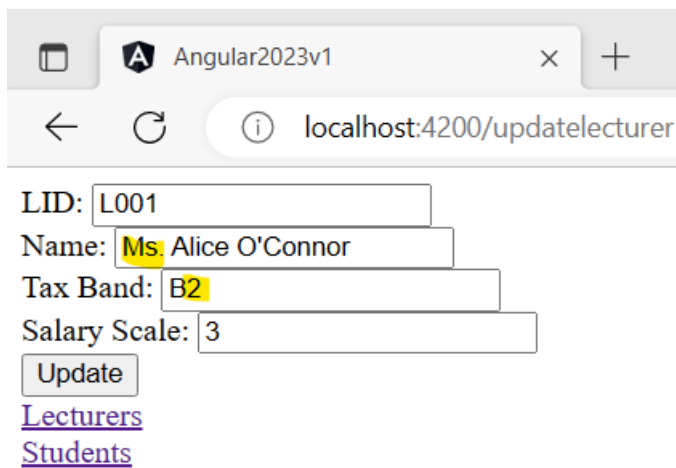
Salary Scale:

[Lecturers](#)

[Students](#)

Figure 15 Lecturer details shown.

All these details except *lid* can be edited.



LID:

Name:

Tax Band:

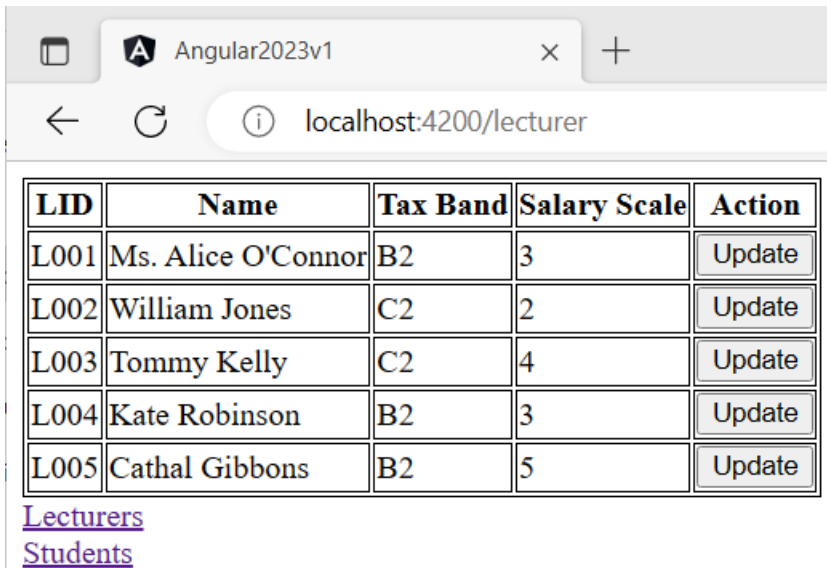
Salary Scale:

[Lecturers](#)

[Students](#)

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.



The screenshot shows a web browser window with the title 'Angular2023v1' and the address bar displaying 'localhost:4200/lecturer'. Below the browser window is a table with five rows of lecturer data. Each row has columns for LID, Name, Tax Band, Salary Scale, and an Action button labeled 'Update'. The lecturers listed are Ms. Alice O'Connor, William Jones, Tommy Kelly, Kate Robinson, and Cathal Gibbons. Below the table are two links: 'Lecturers' and 'Students'.

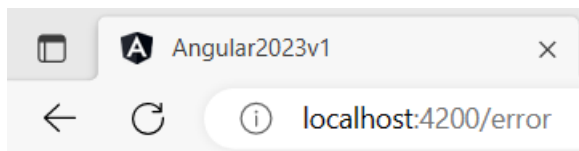
LID	Name	Tax Band	Salary Scale	Action
L001	Ms. Alice O'Connor	B2	3	<button>Update</button>
L002	William Jones	C2	2	<button>Update</button>
L003	Tommy Kelly	C2	4	<button>Update</button>
L004	Kate Robinson	B2	3	<button>Update</button>
L005	Cathal Gibbons	B2	5	<button>Update</button>

[Lecturers](#)
[Students](#)

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.



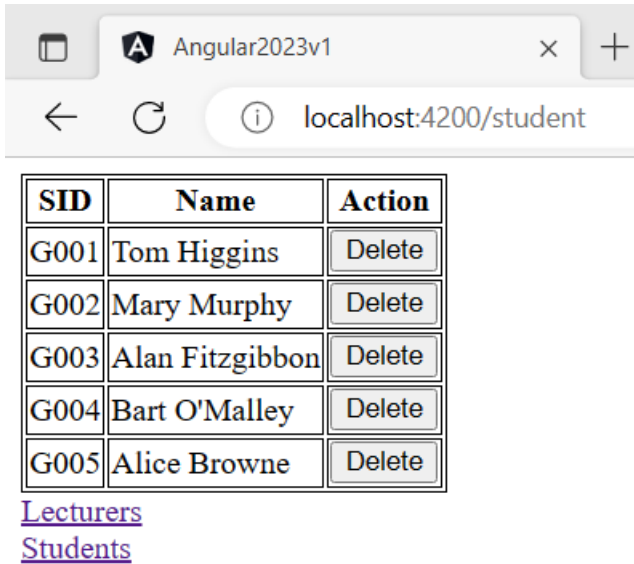
Lecturer: L005 doesn't exist

[Lecturers](#)
[Students](#)

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.



The screenshot shows a web browser window with the title 'Angular2023v1' and the address bar displaying 'localhost:4200/student'. Below the browser window is a table with three columns: 'SID', 'Name', and 'Action'. The table contains five rows of student data, each with a 'Delete' button in the 'Action' column. Below the table are two underlined links: 'Lecturers' and 'Students'.

SID	Name	Action
G001	Tom Higgins	Delete
G002	Mary Murphy	Delete
G003	Alan Fitzgibbon	Delete
G004	Bart O'Malley	Delete
G005	Alice Browne	Delete

[Lecturers](#)
[Students](#)

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.

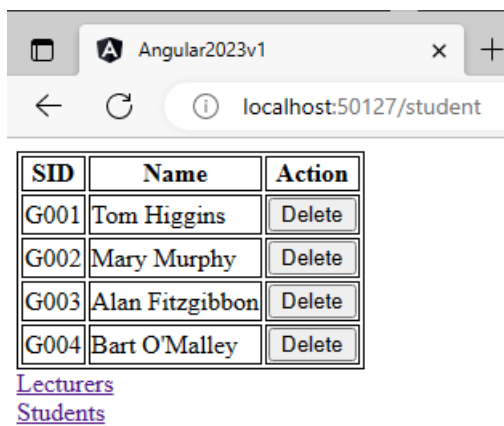


Figure 20 Student G005 deleted

5.4.1.1 Error Conditions

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

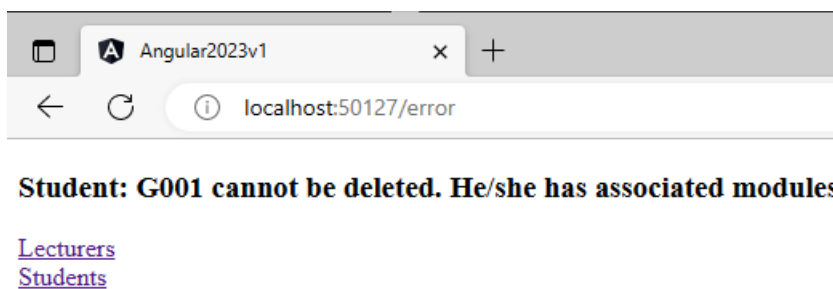


Figure 21 Cannot delete Student G001