

FINAL PROJECT
No 1



ENGETO

Author: Tereza Kadlecová
Date: 20.6.2024

TABLE OF CONTENTS

TABLE OF CONTENTS.....	2
ASSIGNMENT.....	3
TESTING SCENARIOS.....	4
1. HTTP method: GET.....	4
1.1 Positive scenario 1: Get data about an existing student.....	4
1.2 Negative Scenario 1: Get data about a non-existent student.....	5
1. 3 Negative scenario 2: Get data about a non-existent student based on wrong entry data format.....	5
2. HTTP method: POST.....	6
2.1 Positive scenario: a new student record successfully created.....	7
2.2 Negative scenario 1: Unsuccessful creation of a new student record with non-standard entries.....	8
2.3 Negative scenario 2: Unsuccessful creation of a new student record that does not meet the acceptance criteria.....	8
2. 4 Negative scenario 3: Unsuccessful creation of a new student record with duplicated email address.....	10
3. HTTP method: DELETE.....	11
3.1 Positive scenario.....	11
3.2 Negative scenario 1: Deletion of student with non-existent id.....	12
3.3 Negative scenario 2.....	12
TEST EXECUTION.....	13
1. HTTP method: GET.....	13
1.1 Positive scenario 1: Get data about an existing student.....	13
1.2 Negative Scenario 1: Get data about a non-existent student.....	15
1. 3 Negative scenario 2: Get data about a non-existing student based on wrong entry data format.....	16
2. HTTP method: POST.....	21
2.1 Positive scenario: a new student record successfully created.....	21
2.2 Positive scenario: New student record with non-standard entries.....	23
2.3 Negative scenario 2: Unsuccessful creation of a new student record that does not meet the acceptance criteria.....	29
2. 4 Negative scenario: Unsuccessful creation of a new student record with duplicated e-mail address.....	40
3. HTTP method: DELETE.....	43
3.1 Positive scenario: Successful deletion of all data about an existing student from the database.....	43
3.2 Negative scenario 1: Deletion of a non-existent student.....	47
3.3 Negative scenario 2: Deletion of a student with invalid ID.....	48
BUG REPORT.....	50

ASSIGNMENT

The goal of the final project is to test the functionality of an application designed for manipulating student data. The application has a REST-API interface that allows creating, deleting, and retrieving data.

Access details:

Database	database: qa_demo Host: aws.connect.psdb.cloud Username: xxxxxx Password: xxxxxx
REST-API	http://108.143.193.45:8080/api/v1/ students/

All test scenarios will be executed in environment Chrome Version 125.0.6422.141 (Official Build) (64-bit).

TESTING SCENARIOS

Based on the given test scenarios, I have verified the functionality of the application.

1. HTTP method: GET

1.1 Positive scenario 1: Get data about an existing student

Abstract: Get data about an existing student

Steps

1. MySQL Workbench: Test data preparation – verify existence of a student with selected **id**

a) Display the complete list of students

- `SELECT * FROM student`

b) then pick any student from the list, e.g. the first one

- `SELECT * FROM student`
- `where id=xxx;`

2. Postman

- a) Steps
 - Select HTTP method: GET
 - Insert URL with the verified id (see the step above)
 - Press Send button

b) Expected result

- Status code: 200 OK
- Response body in JSON format (corresponds to database row, correct data for the specific id)

```
{  
    id: xxx,  
    firstName: "xxx",  
    lastName: "xxx",  
    email: "xxx@xxx.xxx".  
    age: xxx  
}
```

3. Result verification in MySQL Workbench – Does response in Postman correspond to results in the database?

1.2 Negative Scenario 1: Get data about a non-existent student

Abstract: Get data about a non-existent student (student with non-existent id)

Steps

1. MySQL Workbench: Test data preparation: verify non-existence of a student with selected id

```
SELECT * FROM student
WHERE id=xxx (e.g. 1313 – to be verified);
```

2. Postman

a) Steps

- Select HTTP method: GET
- Insert URL with non-existent id (see the step above)
- Press Send button

b) Expected result

- Status code: 404 Not Found
- Response: no response expected

1. 3 Negative scenario 2: Get data about a non-existent student based on wrong entry data format

Abstract: Get data about a non-existent student and use incorrect format for entry data

Steps

1. MySQL Workbench: Test data preparation: Verify non-existence of a student with selected id

```
SELECT * FROM student
WHERE id=xxx;
```

2. Postman

a) Steps

- HTTP method: GET
- Insert URL with incorrect id format (text, very long number, space, zero, special sign - ?, !, @)
- Press Send button

b) Expected result

- Status code: 400 Bad request
- Response: no response expected

2. HTTP method: POST

Criteria for Successful Student Creation:

1. All attributes are mandatory – everything must be filled out

For successful student creation:

- All attributes are mandatory (firstName, lastName, email, age).
- If an attribute is missing -> 400 Bad Request (lastName and email are mandatory)

2. All attributes are stored in the database in lowercase format: firstName, lastName, email.

3. Validation criteria: age: 1-150 years:

- 400 Bad Request
- Error message: "age must be between 1-150"
- The student must not be created

4. Validation: firstName, lastName: 3-50 characters:

- 400 Bad Request
- Error message: "firstName must be between 3-50 characters"
- The student must not be created

5. Valid email format (at least one @ and one .): Email format: *@*.*

- 400 Bad Request
- Error message: "email must be in format *@*.*"
- The student must not be created

2.1 Positive scenario: a new student record successfully created

Abstract: Create a new student record in the database based on the acceptance criteria (above)

1. Postman

a) Steps

- Select HTTP method: POST
- Type URL `http://108.143.193.45:8080/api/v1/students/` with no student ID
- Click on Body
 - Select data type: raw
 - Select JSON format (not text)
- Insert all mandatory data for the new student record, i.e.

```
{
  "firstName": "",
  "lastName": "",
  "email": "",
  "age":
}
```
- Click send button

b) Expected result

- status code: 201 created
- response body:

```
{
  "id":,
  "firstName": "",
  "lastName": "",
  "email": "",
  "age":
}
```

- response body contains data listed when registering the new student
- response body contains *new automatically assigned id* for the new student

2. Workbench - Final check-up of the new record for student referring to the newly assigned id, check that all attributes are stored in the database in lowercase format.

2.2 Negative scenario 1: Unsuccessful creation of a new student record with non-standard entries

Common for the following four abstracts: Create a new student record in the database based on the acceptance criteria but with non-standard entries e.g. very long email, non-existing domain .xyz, name including special marks or numbers, age containing string)

2.2.1 Abstract:

Create a new student record in the database with email address:

[illegible]

2.2.2 Abstract:

Create a new student record in the database with email address:

hellokitty@gmail.xyz

2.2.3 Abstract:

Create a new student record in the database with firstName T3r3z4.

2.2.4 Abstract:

1. Postman

a) Steps

- Select HTTP method: POST
- Type URL with no student ID
- Click on Body
- Select data type: raw
- Select JSON format (not text)
- Insert data for the new student record
- Click send button

b) Expected result for the four above abstracts

- status code: 400 Bad Request
- response body: student not created

2.3 Negative scenario 2: Unsuccessful creation of a new student record that does not meet the acceptance criteria

Abstract 2.3.1: Create a new student record in the database with firstName Jo
(comment – name is bellow 3 mandatory signs)

Abstract 2.3.2: Create a new student record in the database with email: jo@.com (comment – email with incorrect format)

Abstract 2.3.3: Create a new student record in the database with age 149, 150,151 (comment – boundary testing, boundary for accepting 1-150)

Abstract 2.3.4: Create a new student record in the database with some mandatory data missing:

- ➔ Abstract: Create a new student record in the database with firstName missing
- ➔ Abstract: Create a new student record in the database with e-mail address missing

Steps:

1. Select HTTP method: POST
2. Type URL with no student ID
3. click on Body
4. Select data type: raw
5. Select JSON format (not text)
6. Insert data for the new student record
7. Click send button

Expected results

Abstract 2.3.1

- status code: *400 Bad Request*
- error message: *"firstName must be between 3-50 characters"*
- The student must not be created – to be checked in the MySQL Workbench

Abstract 2.3.2

- status code: *400 Bad Request*
- error message: *"Email must be in format *@*.*"*
- The student must not be created – to be checked in the MySQL Workbench

Abstract 2.3.3

a) for age 149 and 150

- status code: 201 created
- response body:

```
{
  "id":,
  "firstName": "",
  "lastName": "",
  "email": "",
  "age":
}
```
- response body contains data listed when registering the new student
- response body contains new automatically assigned id for the new student

- Workbench - Final check-up of the new record for student referring to the newly assigned id, check that all attributes are stored in the database in lowercase format.

b) for age 151

- status code: *400 Bad Request*
- error message: *"age must be between 1-150"*
- The student must not be created – to be checked in the MySQL Workbench

Abstract 2.3.4

- status code: *400 Bad Request*
- error message: *"firstName is mandatory"*
- error message: *"email is mandatory"*
- The student must not be created – to be checked in the MySQL Workbench

2. 4 Negative scenario 3: Unsuccessful creation of a new student record with duplicated email address

Postman

a) Steps:

- Select HTTP method: POST
- Type URL with no student ID
- click on Body
- Select data type: raw
- Select JSON format (not text)
- Insert data for the new student record
- Click send button

b) Expected result

- status code: *409 Conflict* (or *400 Bad Request*)
- error message: *"student with this email address already exists"*
- Result: The student must not be created – to be checked in the MySQL Workbench

Workbench: Final check-up that the new record for student with duplicated email address has not been created.

`SELECT * FROM student where email ="xxxx"`

3. HTTP method: DELETE

3.1 Positive scenario

Abstract: Deletion of all data about an existing student from the database

- a) Deletion of a student with an existing ID
- b) Verification that the deleted student is not available any more

Steps:

1. MySQL Workbench: Test data preparation: verify existence of a student with selected id

```
SELECT * FROM student where id=xxx;  
'id','age','email','firstName','lastName'
```

2. Postman

Steps

1. Verify existence of a student with the selected id: Select HTTP method: GET

Expected result: status code: **200 OK**

2. Select HTTP method: DELETE
3. Insert URL: `http://108.143.193.45:8080/api/v1/students/xxx`
4. Send button

Expected result:

- Status code: **200 OK / 204 No content**
- Message body: All data about particular student are REMOVED

5. Verify that no student with the id is available

- Select HTTP method: GET
- Insert URL: `http://108.143.193.45:8080/api/v1/students/xxx`
- Click Send button

Expected result: Status code **404 Not found**

3. MySQL Workbench: Check that the deleted student is not to be found in the database any more.

```
SELECT * FROM student where id=xxx;
```

3.2 Negative scenario 1: Deletion of student with non-existent id

Abstract: Deletion of a non-existent student (already deleted) – student with **non-existent ID**

1. MySQL Workbench: Test data preparation: verify non-existence of a student with selected **id**

```
SELECT * FROM student  
WHERE id=xxx;
```

2. Postman

- Select HTTP method: DELETE
- Insert URL: http://108.143.193.45:8080/api/v1/students/xxx
- Click Send button

Expected results

- status code:: 404 Not Found.

3.3 Negative scenario 2

Abstract: Deletion of a student with **invalid ID**

1. MySQL Workbench: Test data preparation: verify the **id** is in incorrect format

```
SELECT * FROM student  
WHERE id=xxx;
```

2. Postman

- Select HTTP method: DELETE
- Insert URL: http://108.143.193.45:8080/api/v1/students/xxx
- Click Send button

Expected results

- status code: 400 Bad Request

TEST EXECUTION

I have executed the test scenarios and attached the test results including screenshots.

1. HTTP method: GET

1.1 Positive scenario 1: Get data about an existing student

1. Test data preparation: MySQL Workbench

- I display the complete list of all students first: `SELECT * FROM student;`

The screenshot shows the MySQL Workbench interface. At the top, the 'Query 1' tab is active, displaying the SQL query: `select * FROM student;`. Below the query editor, the 'Result Grid' is visible, showing a table of student data. The table has columns: #, id, age, email, first_name, and last_name. The first row (id 191) is circled in red. Below the table, the 'Action Output' tab shows the execution details: `select * FROM student LIMIT 0, 1000` returned 661 row(s) in 0,022 sec / 0,0013 sec.

#	id	age	email	first_name	last_name
1	191	34	ijohn34@yourmail.com	JOSHUA	
2	205	3400	1gmail.com	ne	SH
3	206	0	1gmail.com	peter	SMITH
4	208	25	ana.larson@xxmail.com	Anabela	LARSON
5	216	34	548	John	JOSHUA
6	219	34	ijohn34@yourmail.com	John	JOSHUA
7	220	99	zkouska@yourmail.com	John	JOSHUA
8	227	25	ana.larson@xxmail.com	Anabela	LARSON
9	228	34	ijohn34@yourmail.com	John	JOSHUA
10	233	34	548	John	JOSHUA
11	238	34	peter21@gmail.com	peter	SMITH
12	242	34	peter21@gmail.com	peter	SMITH
13	243	34	peter21@gmail.com	peter	SMITH
14	254	344	abc@abc.com	Lucia	LESNA
15	260	21	az@gmail.com	adriana	NEZO
16	264	-35	ijohn34@yourmail.com	John	JOSHUA
17	266	34	ijohn34@yourmail.com	John	JOSHUA
18	267	34	ijohn34@yourmail.com	John	JOSHUA
19	268	34	ijohn34@yourmail.com	John	JOSHUA
20	271	34	548	John	JOSHUA

- Then I pick any student, e.g. the first one in the list with **id 191**
- For detail:

`SELECT * FROM student`

`WHERE id=191;`

Query 1 ✖

1 • **SELECT** * **FROM** student
2 **WHERE** id=191;
3

Result Grid

#	id	age	email	first_name	last_name
1	191	34	jjohn34@yourmail.com	JOSHUA	

* NULL NULL NULL NULL NULL

- Result: '191','34','jjohn34@yourmail.com','','JOSHUA'

2. Postman

I select HTTP method GET, insert URL

<http://108.143.193.45:8080/api/v1/students/191> and press Send button

The screenshot shows the Postman interface. The top section displays the HTTP method 'GET' and the URL 'http://108.143.193.45:8080/api/v1/students/191'. Below this, the 'Query Params' section is empty. The bottom section shows the response body in JSON format, which is a single object with the following fields: 'id' (191), 'firstName' (empty string), 'lastName' ('JOSHUA'), 'email' ('jjohn34@yourmail.com'), and 'age' (34). The status bar at the bottom right indicates a '200 OK' response with a response time of 158 ms and a body size of 249 B.

b) Expected result

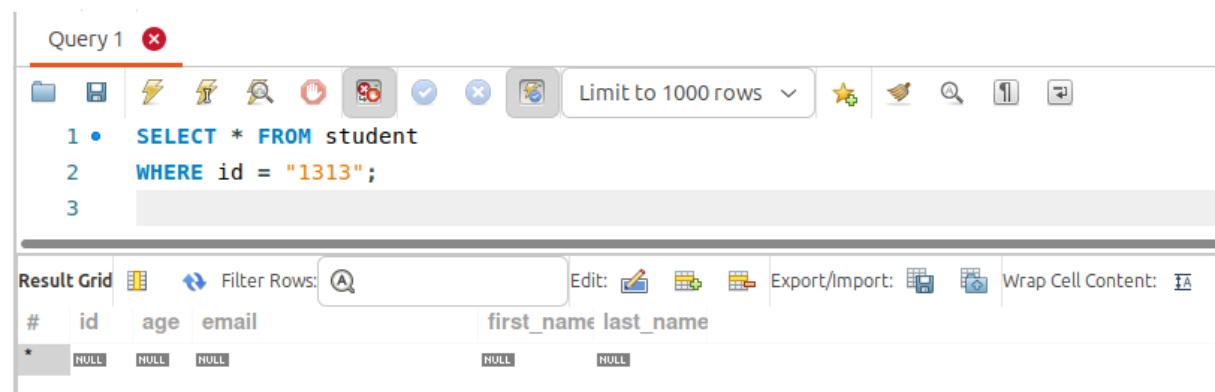
- Status code: 200 OK
- Response body in JSON format (corresponds to database row, correct data for the specific id)

3. Does the response in Postman correlate to results in the database? → **YES**, see the green arrow

1.2 Negative Scenario 1: Get data about a non-existent student

1. **MySQL Workbench:** Test data preparation: verify non-existence of a student with id e.g. 1313

- **SQL Request:** SELECT * FROM student
WHERE id=1313;



The screenshot shows the 'student 2' window in MySQL Workbench. The 'Action Output' tab is selected, displaying a table with the following columns: #, Time, Action, Message, and Duration / Fetch. The table contains two rows of data.

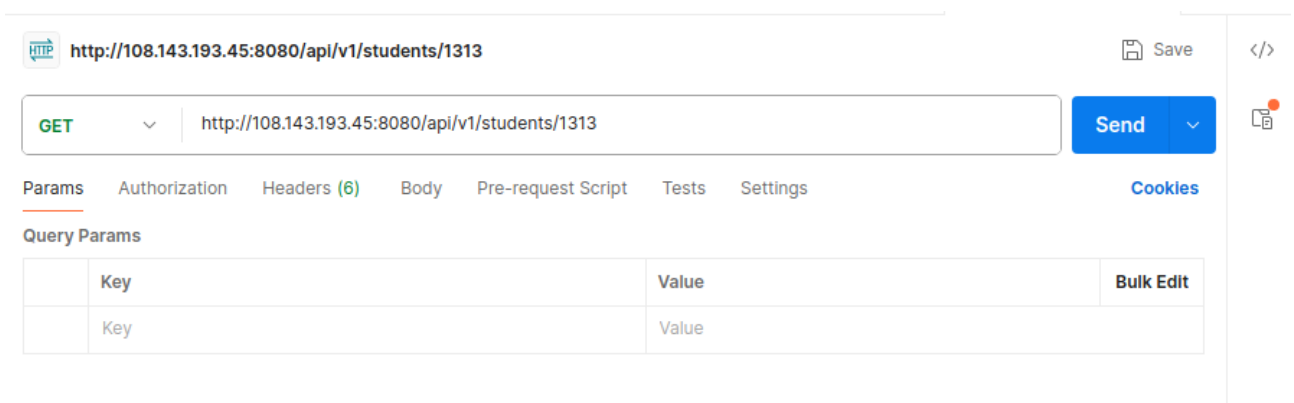
#	Time	Action	Message	Duration / Fetch
1	09:12:50	SELECT * FROM student WHERE id=191 LIMIT 0, 1000	1 row(s) returned	0,019 sec / 0,00003...
2	09:23:28	SELECT * FROM student WHERE id=1313 LIMIT 0, 1000	0 row(s) returned	0,022 sec / 0,00001...

- **Result:** no student with id 1313 exists

2. Postman

I select HTTP method GET, insert the URL

<http://108.143.193.45:8080/api/v1/students/1313> and click Send button.

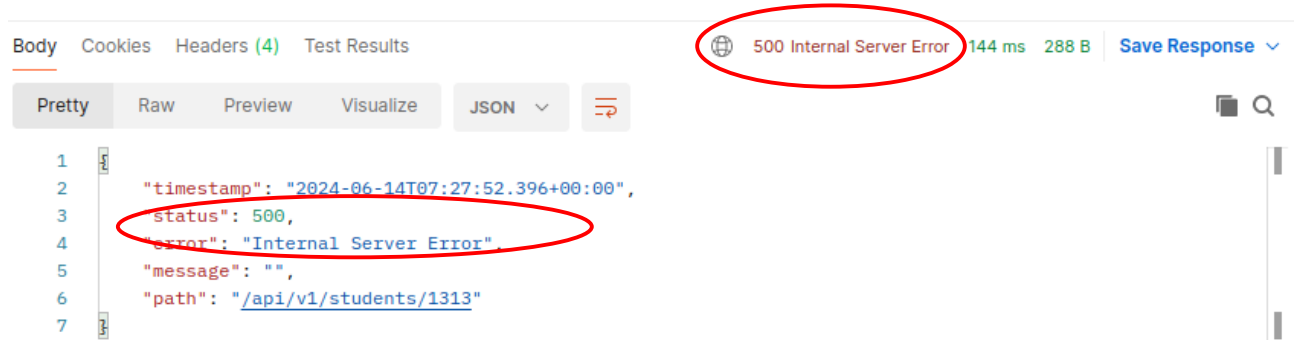


Expected result:

- Status code: 404
- Error: Not found

Actual result:

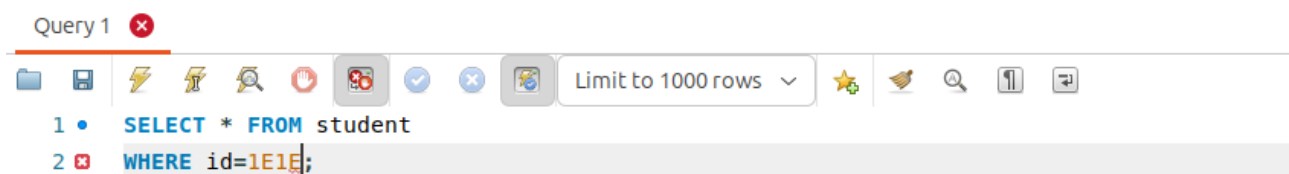
- Status code: **500**,
- error: **Internal Server Error** -meaning I have found **a bug**.



1. 3 Negative scenario 2: Get data about a non-existing student based on wrong entry data format

1. **MySQL Workbench:** Test data preparation: Verify non-existence of a student with selected **id**

a) SQL Request: `SELECT * FROM student WHERE id=1E1E;`



b) SQL Response: Syntax Error, Code 1105

Action Output				
#	Time	Action	Message	Duration / Fetch
1	09:12:50	SELECT * FROM student WHERE id=191 LIMIT 0, 1000	1 row(s) returned	0,019 sec / 0,00003...
2	09:23:28	SELECT * FROM student WHERE id=1313 LIMIT 0, 1000	0 row(s) returned	0,022 sec / 0,00001...
3	09:45:59	SELECT * FROM student WHERE id=1E1E	Error Code: 1105. syntax error at position 35 near...	0,022 sec

2. **Postman:**

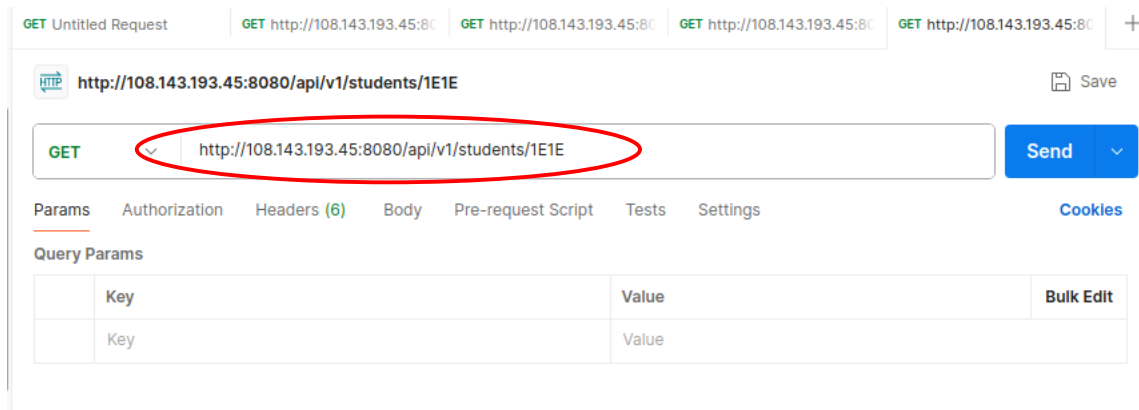
a) **id = 1E1E**

I select HTTP method GET, insert URL

<http://108.143.193.45:8080/api/v1/students/1E1E> and click Send button.

Expected result for cases a) - e)

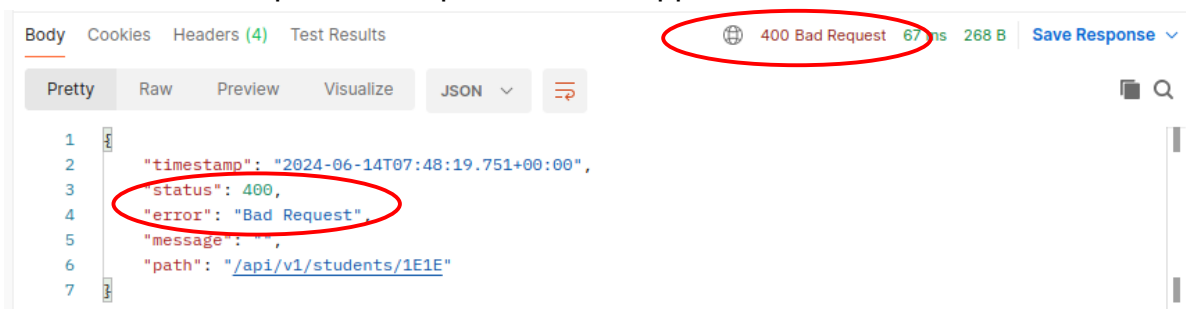
- Status code: 400
- Error: Bad Request



Actual result:

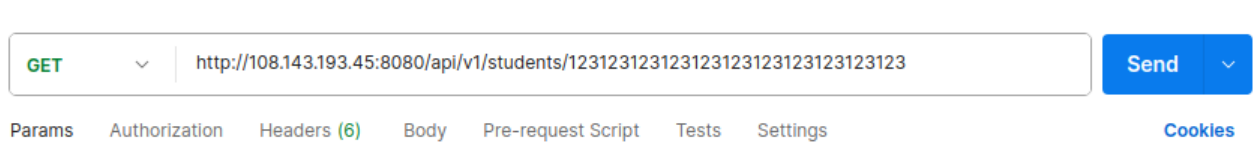
- Status code: 400
- Error: Bad Request

Actual result corresponds to expected result, application has worked **OK**.



The same procedure was used for testing other wrong data entries for id b) – e)

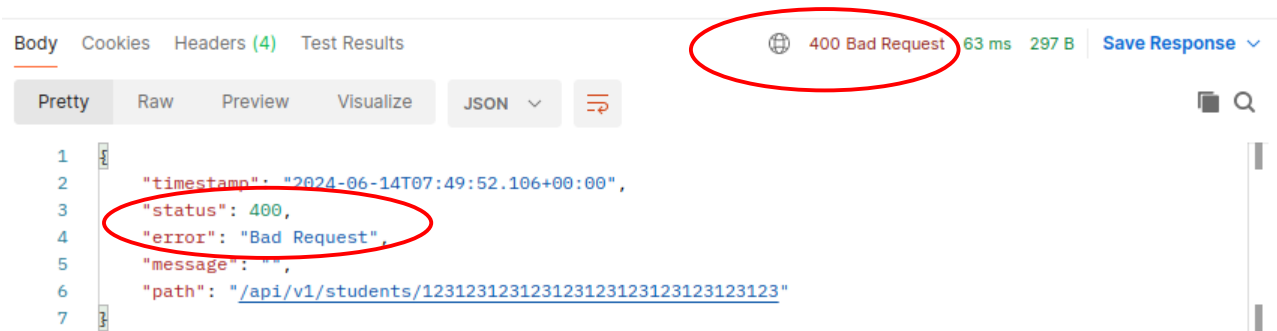
b) id = 123123123123123123123123123123123



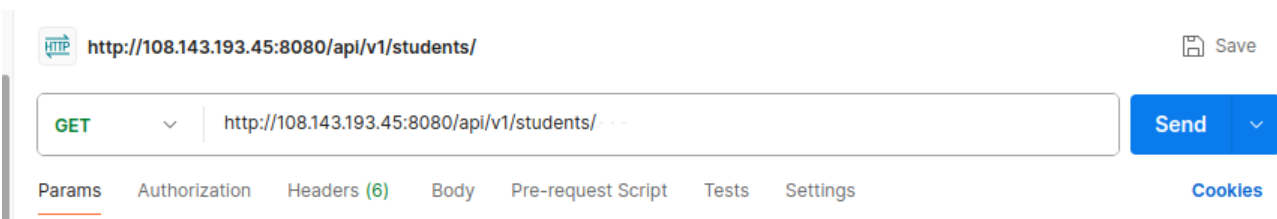
Actual result:

- Status code: 400
- Error: Bad Request

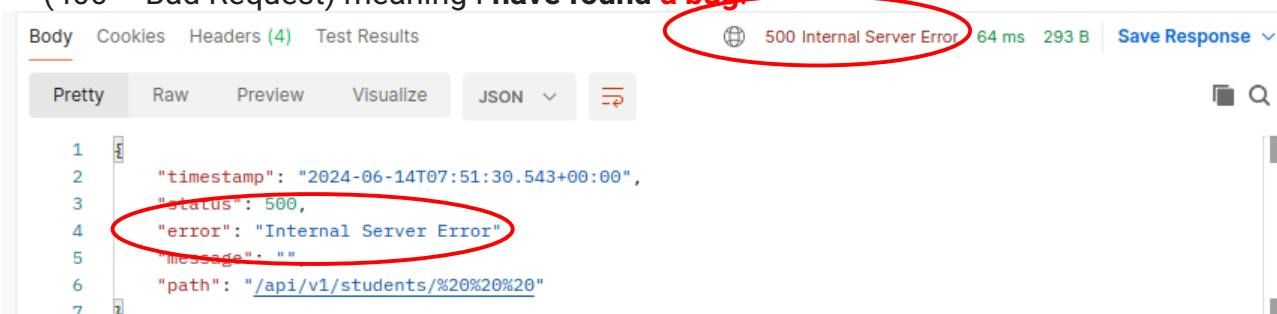
Actual result corresponds to expected result, application has worked **OK**.



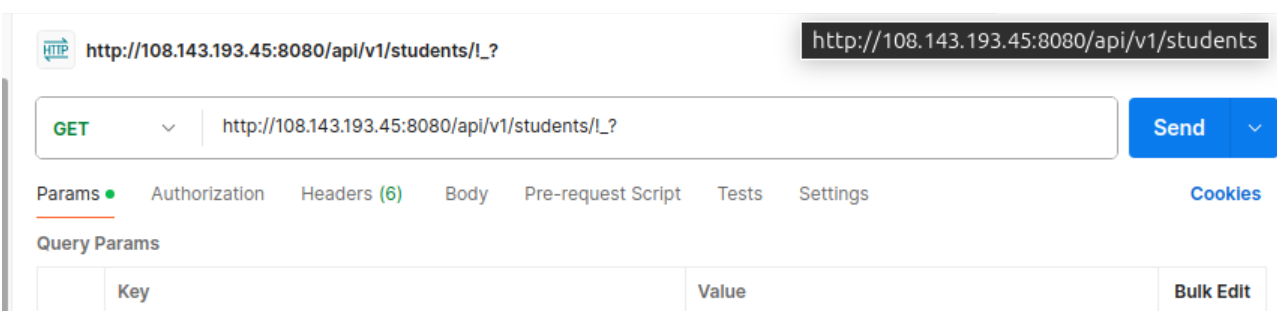
c) id = space (3x pressed spacebar)



However the result I have got (**500 – Internal Server Error**) was not what I expected (400 – Bad Request) meaning I **have found a bug**.



d) id = !_?



Actual result:

- Status code: 400
- Error: Bad Request

Actual result corresponds to expected result, application has worked **OK**.

Body Cookies Headers (4) Test Results

400 Bad Request 70 ms 266 B Save Response

Pretty Raw Preview Visualize JSON

```

1 {
2   "timestamp": "2024-06-14T07:52:44.031+00:00",
3   "status": 400,
4   "error": "Bad Request",
5   "message": "",
6   "path": "/api/v1/students/!"
7 }

```

e) id = ?

http://108.143.193.45:8080/api/v1/students/?

GET http://108.143.193.45:8080/api/v1/students/? Send

Params Authorization Headers (6) Body Pre-request Script Tests Settings Cookies

Actual result:

- Status code: 200 - OK (not 400 – Bad Request, as expected) meaning I have found **a bug**.
- In response body I got a long list of results in JSON showing data for students starting with id 206 down to 1249. I assume I have recieved the complete list of all students.

http://108.143.193.45:8080/api/v1/students/?

GET http://108.143.193.45:8080/api/v1/students/? Send

Params Authorization Headers (6) Body Pre-request Script Tests Settings Cookies

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

This request does not have a body

Body Cookies Headers (5) Test Results

200 OK 206 ms 68.77 KB Save Response

Pretty Raw Preview Visualize JSON

```

1 {
2   {
3     "id": 206,
4     "firstName": "peter",
5     "lastName": "SMITH",
6     "email": "lgmail.com",
7     "age": 0
8   },
9   {
10    "id": 219,
11    "firstName": "",
12    "lastName": "JOSHUA",
13    "email": "jjohn34@yourmail.com",
14    "age": 34
15  },
16  {
17    "id": 220,

```

```

4907         "age": 48
4908     },
4909     {
4910         "id": 1248,
4911         "firstName": "Tereza",
4912         "lastName": "NOVÁKOVÁ",
4913         "email": "tn@mail.com",
4914         "age": 21
4915     },
4916     {
4917         "id": 1249,
4918         "firstName": "Bronislava",
4919         "lastName": "SLIVOVA",
4920         "email": "",
4921         "age": 67
4922     }
4923 ]

```

By

Cookies

Headers (5)

Test Results

Status: 200 OK

Time: 206 ms

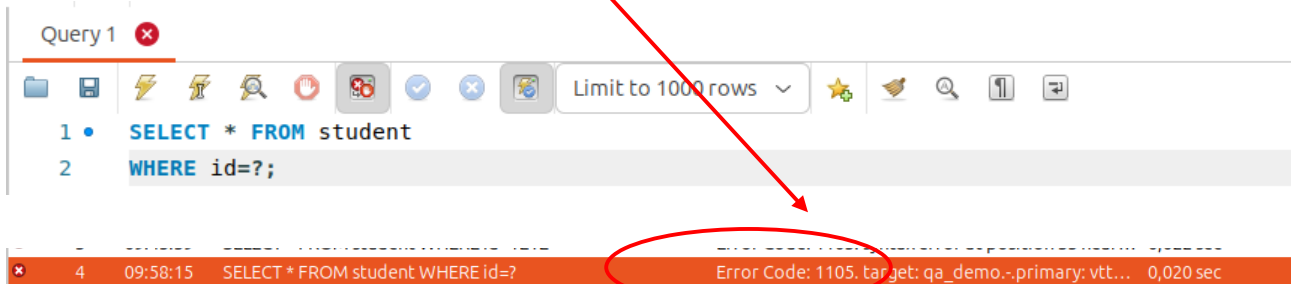
Size: 68.77 KB

Save Response >

PrettyRawPreviewVisualize

[{"id":206,"firstName":"peter","lastName":"SMITH","email":["gmail.com"],"age":0}, {"id":219,"firstName":"","lastName":"JOSHUA","email":["john34@yourmail.com"],"age":34}, {"id":220,"firstName":"John","lastName":"","email":["zkouska@yourmail.com"],"age":99}, {"id":227,"firstName":"Anabela","lastName":"LARSON","email":["ana.larson@xmail.com"],"age":25}, {"id":228,"firstName":"","lastName":"JOSHUA","email":["john34@yourmail.com"],"age":34}, {"id":233,"firstName":"John","lastName":"JOSHUA","email":["548@..."], {"id":238,"firstName":"Marie","lastName":"NOVOTNÁ","email":["m.novotna"],"age":34}, {"id":242,"firstName":"Martin","lastName":"KOMÁŘ","email":["komar@seznam.cz"],"age":34}, {"id":243,"firstName":"","lastName":"SMITH","email":["pete21@gmail.com"],"age":34}, {"id":254,"firstName":"Lucia","lastName":"LESNA","email":["abc@abc.com"],"age":34}, {"id":260,"firstName":"adrian","lastName":"NEZO","email":["az@gmail.com"],"age":21}, {"id":264,"firstName":"John","lastName":"JOSHUA","email":["john34@yourmail.com"],"age":35}, {"id":266,"firstName":"","lastName":"JOSHUA","email":["john34@yourmail.com"],"age":34}, {"id":267,"firstName":"John","lastName":"","email":["john34@yourmail.com"],"age":34}, {"id":268,"firstName":"John","lastName":"JOSHUA","email":["john34@yourmail.com"],"age":34}, {"id":271,"firstName":"John","lastName":"JOSHUA","email":["548@..."], {"id":278,"firstName":"Test","lastName":"TESTSDP","email":["john34@yourmail.com"],"age":34}, {"id":283,"firstName":"PoPoLUska","lastName":"ČRĚVIČKOVÁ","email":["polusa@gmail.com"],"age":170}, {"id":284,"firstName":"Jen","lastName":"","email":["jbais3@freewebs.com"],"age":19}, {"id":285,"firstName":"Krung ThapMahanakhonAmonRattanakosin","lastName":"NOVÁK","email":["pnov@sda.com"],"age":105}, {"id":287,"firstName":"Marek","lastName":"TESTVOANINENIZADNARAKE TOVAĐASTEFANA","email":["skupina@jedna.cz"],"age":67}, {"id":288,"firstName":"LADISLAV","lastName":"PODMOSTEK","email":["skupina@jednazc"],"age":67}, {"id":289,"firstName":"AI","lastName":"NEVĚŘÍČI","email":["skupina@jedna.cz"],"age":999}, {"id":290,"firstName":"A","lastName":"MESNARD","email":["amesnard@lundi.de"],"age":93}, {"id":291,"firstName":"Adel","lastName":"SEVIDRON","email":["adelsvidron@mail.com"],"age":11}, {"id":292,"firstName":"TOMAS","lastName":"NEVĚŘÍČI","email":["skupina@jedna.cz"],"age":7}, {"id":293,"firstName":"AI","lastName":"NEVĚŘÍČI","email":["skupina@jedna.cz"],"age":100},

However, in MySQL Workbench, result for query **SELECT * FROM student WHERE id=?;** is shown as error with code 1105 (see below).



2. HTTP method: POST

2.1 Positive scenario: a new student record successfully created

Abstract: Create a new student record in the database based on the acceptance criteria.

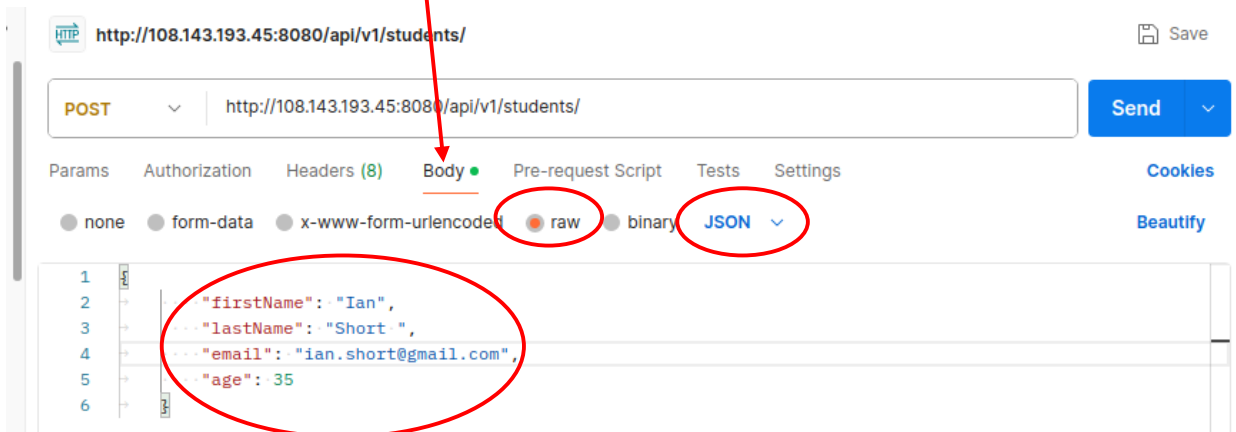
1. Postman

As displayed in the figure below, I selected HTTP method POST and inserted URL <http://108.143.193.45:8080/api/v1/students/> with no student ID.

Then I completed the Body part - data type: raw and JSON format. Finally I inserted all mandatory data for the new student record, i.e.

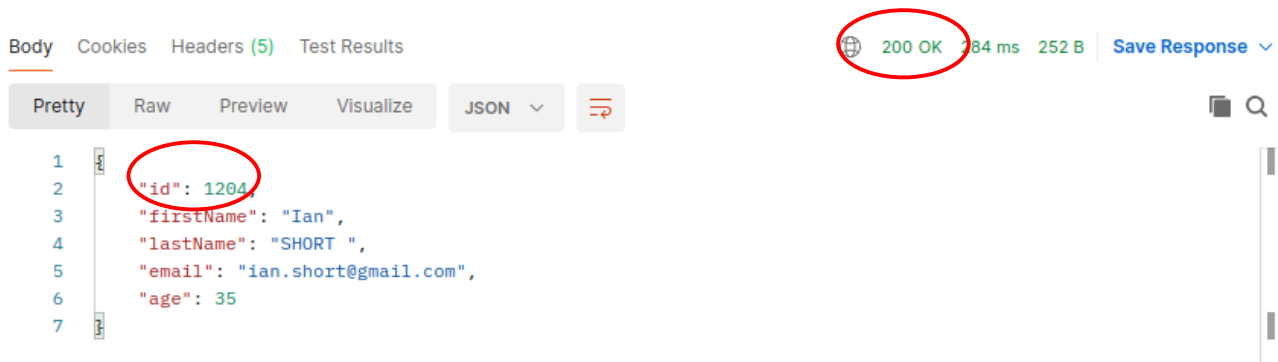
```
{
  "firstName": "Ian",
  "lastName": "Short",
  "email": "ian.short@gmail.com",
  "age": 35
}
```

And clicked Send button.



Result

- Status code: **200 – OK** (not **201 – Created**, as expected) meaning I **have found a bug**.
- The response body in JSON contains data listed when registering the new student.
- Also, new **id 1204** has been assigned to the new student.

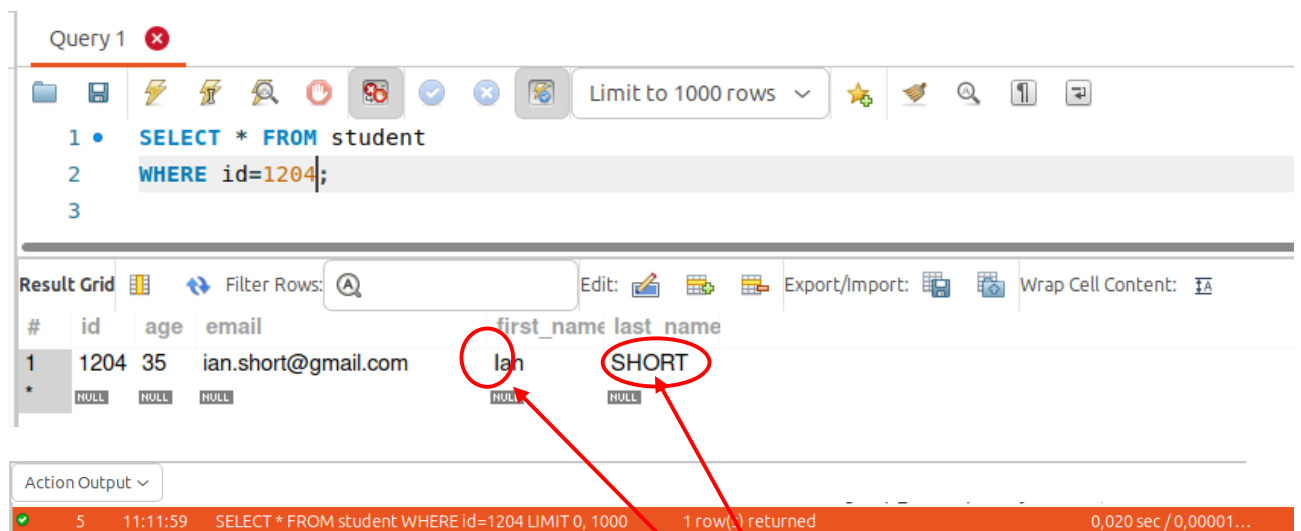


The last step is to check whether the newly created student has also been stored in the database in the required format.

2. My SQL Workbench

Final check-up of the new record for student referring to the newly assigned id, verification that all attributes are stored in the database in lowercase format.

a) SQL Request: `SELECT * FROM student where id=1204;`



b) Result

- all attributes are stored in the database – **Yes: OK**
- all attributes are stored in lowercase format – **No (Ian instead of ian and SHORT instead of short): Bug**

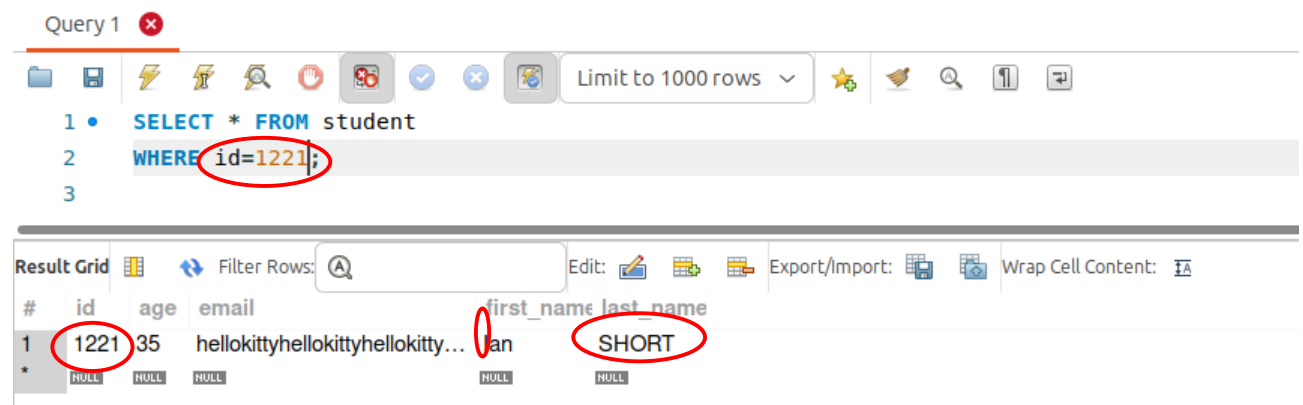
2. My SQL Workbench

New student with **id 1221** has been stored alright in the database as checked in MySQL Workbench.

```
SELECT * FROM student
WHERE id=1221;
```

- all attributes are stored in the database – **Yes: OK**

See the figure below.



Similarly, as in the former case, all attributes have not been stored in the database in the requested lowercase format. (**ian instead of ian and SHORT instead of short**):
bug

2.2.2 non-existent domain used

1. Postman

I used the same registering data for the new student as in 2.1 scenario but for an email address I have chosen a non-existent domain **.xyz**.

hellokitty@gmail.xyz

I expect this should not be enabled and the status code should be **400 Bad request**.

HTTP <http://108.143.193.45:8080/api/v1/students/> Save

POST <http://108.143.193.45:8080/api/v1/students/> Send

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

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

```
1 {
2   "firstName": "Ian",
3   "lastName": "Short",
4   "email": "hellokitty@gmail.xyz",
5   "age": 35
6 }
```

Body Cookies Headers (5) Test Results 200 OK 274 ms 253 B Save Response

Pretty Raw Preview Visualize JSON

```
1 {
2   "id": 1222,
3   "firstName": "Ian",
4   "lastName": "SHORT",
5   "email": "hellokitty@gmail.xyz",
6   "age": 35
7 }
```

Result

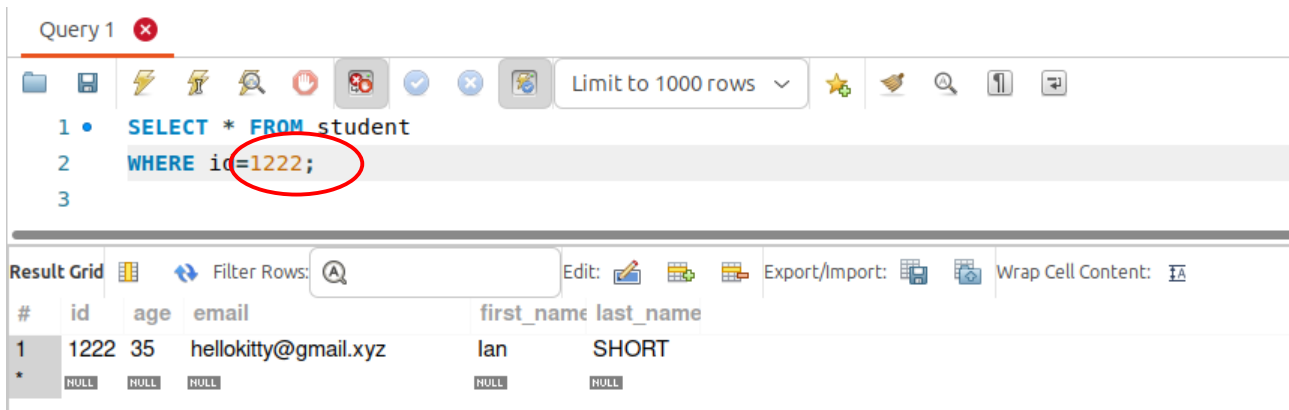
However, the new student with wrong (non-existent) domain was successfully registered, new id 1222 has been assigned to him.

The only persisting bug is the status code one. So instead of **400 Bad Request** (or at least 201 created), I am getting **200 OK**.

2. My SQL Workbench

For check-up in the database, I used the newly assigned id 1222.

```
SELECT * FROM student
WHERE id=1222;
```



4 09:06:21 SELECT * FROM student WHERE id=1222 LIMIT 0, 1000 1 row(s) returned 0,030 sec / 0,00001...

The new student with the non-existent domain in email address has also been successfully stored in the database, as we can see from the pic above. And again all attributes are not stored in the database in lowercase format as requested. (**lan** instead of **ian** and **SHORT** instead of **short**): **Bug**

2.2.3 Abstract: Create a new student record in the database with firstName T3r3z4

1. Postman

I used the same registering data for the new student as in 2.1 scenario but for the first name I tried combination of letters and numbers: **T3r3z4**

I expect this should not be enabled and the status code should be **400 Bad request**.

HTTP <http://108.143.193.45:8080/api/v1/students/> Save

POST <http://108.143.193.45:8080/api/v1/students/> Send

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

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

```
1 {
2   "firstName": "T3r3z4",
3   "lastName": "Short ",
4   "email": "ian.short@gmail.com",
5   "age": 35
6 }
```

Body Cookies Headers (5) Test Results 200 OK 258 ms 255 B Save Response

Pretty Raw Preview Visualize JSON

```
1 {
2   "id": 1223,
3   "firstName": "T3r3z4",
4   "lastName": "SHORT ",
5   "email": "ian.short@gmail.com",
6   "age": 35
7 }
```

Result

The new student with incorrect firstName format was successfully registered, new id 1223 has been assigned to him. The only persisting bug is the status code one. So instead of 400 Bad Request (or at least 201 created), I am getting 200 OK.

2. My SQL Workbench

For check-up in the database, I used the newly assigned id 1223.

SELECT * FROM student

WHERE id=1223;

Query 1

Limit to 1000 rows

```
1 • SELECT * FROM student
2 WHERE id=1223;
3
```

Result Grid Filter Rows: Edit: Export/Import: Wrap Cell Content:

#	id	age	email	first_name	last_name
1	1223	35	ian.short@gmail.com	T3r3z4	SHORT
*	NULL	NULL	NULL	NULL	NULL

5 09:13:48 SELECT * FROM student WHERE id=1223 LIMIT 0, 1000 1 row(s) returned 0,023 sec / 0,00000...

New student record with the incorrect firstName format has also been successfully stored in the database, as we can see from the pic above. And again all attributes are not stored in the database in lowercase format as requested (**T3r3z4** instead of **t3r3z4** and **SHORT** instead of **short**): **Bug**

2.2.4 Abstract: Create a new student record in the database with age 1?!4

I tried incorrect age format containing not only numbers but also special signs (I also tried putting the "1?!4" age in quotes as for string but with the same result)

The screenshot shows a REST client interface with the following details:

- URL:** `http://108.143.193.45:8080/api/v1/students/`
- Method:** `POST`
- Body:**

```
{  "firstName": "Ian",  "lastName": "Short",  "email": "ian.short@gmail.com",  "age": "1?!4"}
```
- Response:** `400 Bad Request` (circled in red), 62 ms, 264 B. The response body is:

```
{  "timestamp": "2024-06-17T07:16:11.784+00:00",  "status": 400,  "error": "Bad Request",  "message": "",  "path": "/api/v1/students/"}
```

Result

This entry was not accepted by the application and the status code was correctly **400 Bad Request**, which is the correct response. So **OK**.

2.3 Negative scenario 2: Unsuccessful creation of a new student record that does not meet the acceptance criteria

Steps:

1. Select HTTP method: POST
2. Type URL with no student ID
3. click on Body
4. Select data type: raw
5. Select JSON format (not text)
6. Insert data for the new student record
7. Click send button

Abstract 2.3.1: Create a new student record in the database with firstName Jo
(comment – name is bellow 3 mandatory signs)

1. Postman

As displayed in the figure below, I selected HTTP method POST and inserted URL <http://108.143.193.45:8080/api/v1/students/> with no student ID.

Than I completed the Body part - data type: raw and JSON format. Finally, I inserted all mandatory data for the new student record, but for the firstName I intentionally tried an entry with only 2 signs

```
{
  "firstName": "Jo",
  "lastName": "Short",
  "email": "jo.short@gmail.com",
  "age": 35
}
```

And clicked Send button.

Results

Expected results:

- status code: *400 Bad Request*
- error message: *"firstName must be between 3-50 characters"*
- The student must not be created – to be checked in the MySQL Workbench

The screenshot shows a REST client interface. At the top, a POST request is configured to `http://108.143.193.45:8080/api/v1/students/`. The request body is a JSON object: `{ "firstName": "Jo", "lastName": "Short", "email": "jo.short@gmail.com", "age": 35 }`. The response is a 200 OK status with a response time of 250 ms and a body size of 250 B. The response body is a JSON object: `{ "id": 1231, "firstName": "Jo", "lastName": "SHORT ", "email": "jo.short@gmail.com", "age": 35 }`.

Actual results:

- status code: **200 OK - bug**
- New student with assigned **id 1231** created - **bug**

2. MySQL Workbench

For check-up in the database, I used the newly assigned id 1231.

SELECT * FROM student

WHERE id=1231;

The screenshot shows the MySQL Workbench interface. A query is executed: `SELECT * FROM student WHERE id=1231;`. The result grid shows one row with the following data:

#	id	age	email	first_name	last_name
1	1231	35	jo.short@gmail.com	Jo	SHORT

The student with id 1231 has really been created. - **bug**

Abstract 2.3.2: Create a new student record in the database with email: jo@.com (comment – email with incorrect format)

1. Postman

The procedure in Postman was the same as in 2.3.1 scenario, except the firstName was correct but the email address was in incorrect format: jo@.com

Expected results

- status code: *400 Bad Request*
- error message: *"Email must be in format *@*.*"*
- The student must not be created – to be checked in the MySQL Workbench

Actual result

- status code: **200 OK** – bug
- new student with newly assigned **id 1238** created - **bug**

The screenshot shows a Postman interface for a POST request to `http://108.143.193.45:8080/api/v1/students/`. The request body is a JSON object:

```
1 {
2   "firstName": "Ian",
3   "lastName": "Short",
4   "email": "jo@.com",
5   "age": 40
6 }
7
```

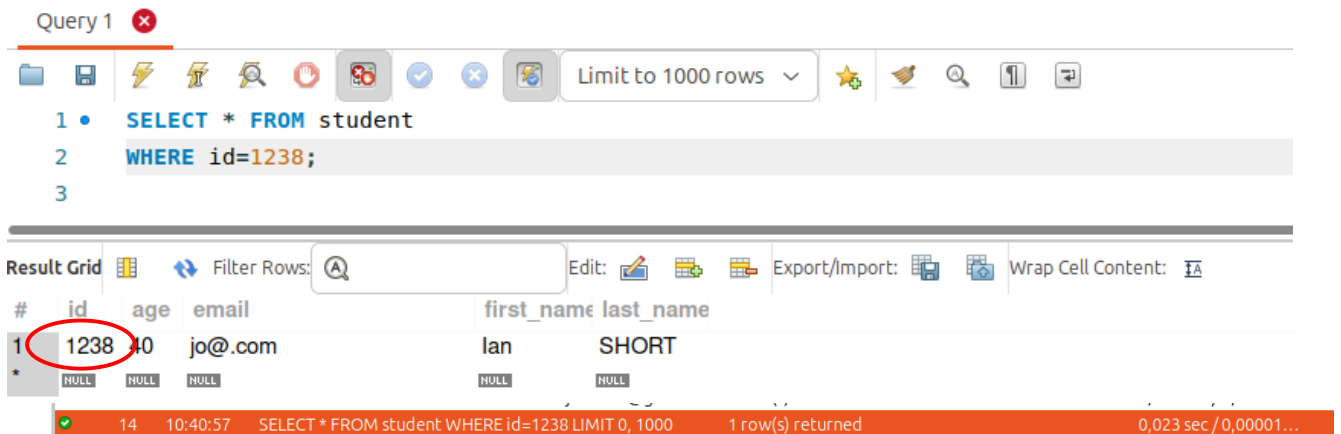
The response status is **200 OK**, with a response time of 249 ms and a size of 239 B. The response body is a JSON object:

```
1 {
2   "id": 1238,
3   "firstName": "Ian",
4   "lastName": "SHORT",
5   "email": "jo@.com",
6   "age": 40
7 }
```

2. MySQL Workbench

For check-up in the database, I used the newly assigned id 1238.

```
SELECT * FROM student
WHERE id=1238;
```



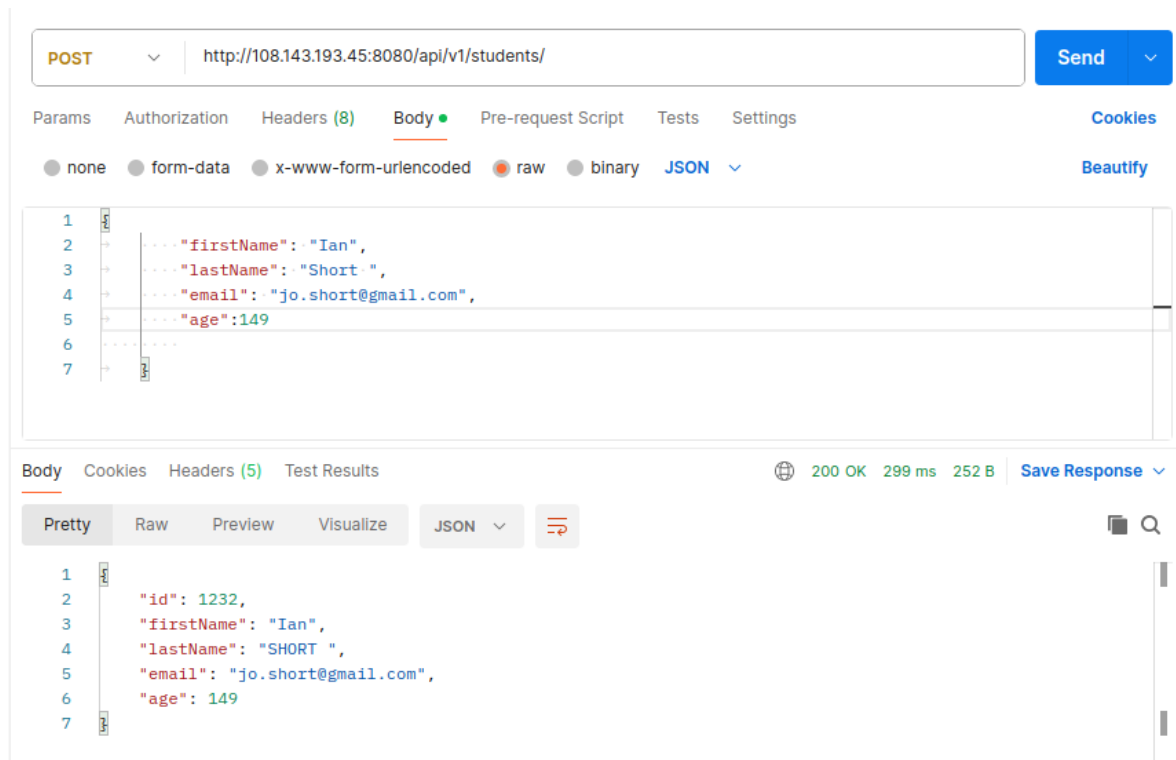
The new student record with the id 1238 has really been stored in database - **bug**

Abstract 2.3.3: Create a new student record in the database with age 149, 150, 151
(comment – boundary testing, boundary for accepting 1-150)

a) for age 149 and 150

1. Postman

I kept the same data (firstName, lastName, email) in all three cases, I only changed age. So for age 149 and 150 – within the allowed age boundary.



Expected result:

- status code: **201 created**
- new student record with new id assigned

Actual result:

- status code: **200 OK – bug**
- new student with id 1232 created
- the response body in JSON contains data listed when registering the new student.

2. MySQL Workbench

For check-up in the database, I used the newly assigned id 1232.

SELECT * FROM student

WHERE id=1232;

Query 1

1 * SELECT * FROM student
2 WHERE id=1232
3

Result Grid

#	id	age	email	first_name	last_name
1	1232	149	jo.short@gmail.com	Ian	SHORT
*	NULL	NULL	NULL	NULL	NULL

7 10:10:14 SELECT * FROM student WHERE id=1232 LIMIT 0, 1000 1 row(s) returned 0,022 sec / 0,00001...

The new student record with the id 1232 has really been stored. **OK**
But the data are not stored in lowercase as requested. **bug**

The same procedure and results apply to age 150, as shown in the below pics.

1. Postman

http://108.143.193.45:8080/api/v1/students/

POST http://108.143.193.45:8080/api/v1/students/

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

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

```

1 {
2   "firstName": "Ian",
3   "lastName": "Short ",
4   "email": "jo.short@gmail.com",
5   "age": 150
6 }
7

```

Body Cookies Headers (5) Test Results 200 OK 243 ms 252 B Save Response

Pretty Raw Preview Visualize JSON

```

1 {
2   "id": 1233,
3   "firstName": "Ian",
4   "lastName": "SHORT ",
5   "email": "jo.short@gmail.com",
6   "age": 150
7 }

```

Expected result:

- status code: **201 created**
- new student record with new assigned id

Actual result:

- status code: **200 OK – bug**
- new student with id 1233 created
- the response body in JSON contains data listed when registering the new student.

2. MySQL Workbench

For check-up in the database, I used the newly assigned id 1233.

SELECT * FROM student

WHERE id=1233;

The screenshot shows the MySQL Workbench interface. At the top, a query editor window titled 'Query 1' contains the following SQL query:

```
1 • SELECT * FROM student
2 WHERE id=1233;
3
```

Below the query editor, the 'Result Grid' tab is active, displaying the results of the query. The results are shown in a table with the following columns: #, id, age, email, first_name, and last_name. The first row is highlighted, showing the student with id 1233, age 150, email jo.short@gmail.com, first_name lan, and last_name SHORT. The second row shows NULL values for all columns.

#	id	age	email	first_name	last_name
1	1233	150	jo.short@gmail.com	lan	SHORT
*	NULL	NULL	NULL	NULL	NULL

At the bottom of the interface, a status bar shows the following information: 8, 10:12:02, SELECT * FROM student WHERE id=1233 LIMIT 0, 1000, 1 row(s) returned, 0,021 sec / 0,00001...

b) for age 151

1. Postman

Expected results

- status code: 400 - Bad Request
- error message: *"age must be between 1-150"*
- student must not be created

HTTP <http://108.143.193.45:8080/api/v1/students/> Save

POST <http://108.143.193.45:8080/api/v1/students/> Send

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

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary **JSON** Beautify

```
1 {
2   "firstName": "Ian",
3   "lastName": "Short ",
4   "email": "jo.short@gmail.com",
5   "age": 151
6 }
7
```

Body Cookies Headers (5) Test Results 200 OK 204 ms 252 B Save Response

Pretty Raw Preview Visualize **JSON** ⌵ 🔍

```
1 {
2   "id": 1234,
3   "firstName": "Ian",
4   "lastName": "SHORT ",
5   "email": "jo.short@gmail.com",
6   "age": 151
7 }
```

Actual results

- status code **200 OK - bug**
- newly assigned id 1234 – **bug** (student must not be created)
- the response body in JSON contains data listed when registering the new student.

2. MySQL Workbench

Expected result: Student must not be created.

SELECT * FROM student

WHERE id=1234;

Query 1 ✖

Limit to 1000 rows ▼

```

1 • SELECT * FROM student
2 WHERE id=1234;
3

```

Result Grid ⌵ Filter Rows: Edit: ✎ Export/Import: 📄 Wrap Cell Content: ⌵

#	id	age	email	first_name	last_name
1	1234	151	jo.short@gmail.com	Ian	SHORT
*	NULL	NULL	NULL	NULL	NULL

9 10:12:58 SELECT * FROM student WHERE id=1234 LIMIT 0, 1000 1 row(s) returned 0,023 sec / 0,00001...

The student with id 1234 has really been created. - **bug**

Abstract 2.3.4: Create a new student record in the database with some mandatory data missing:

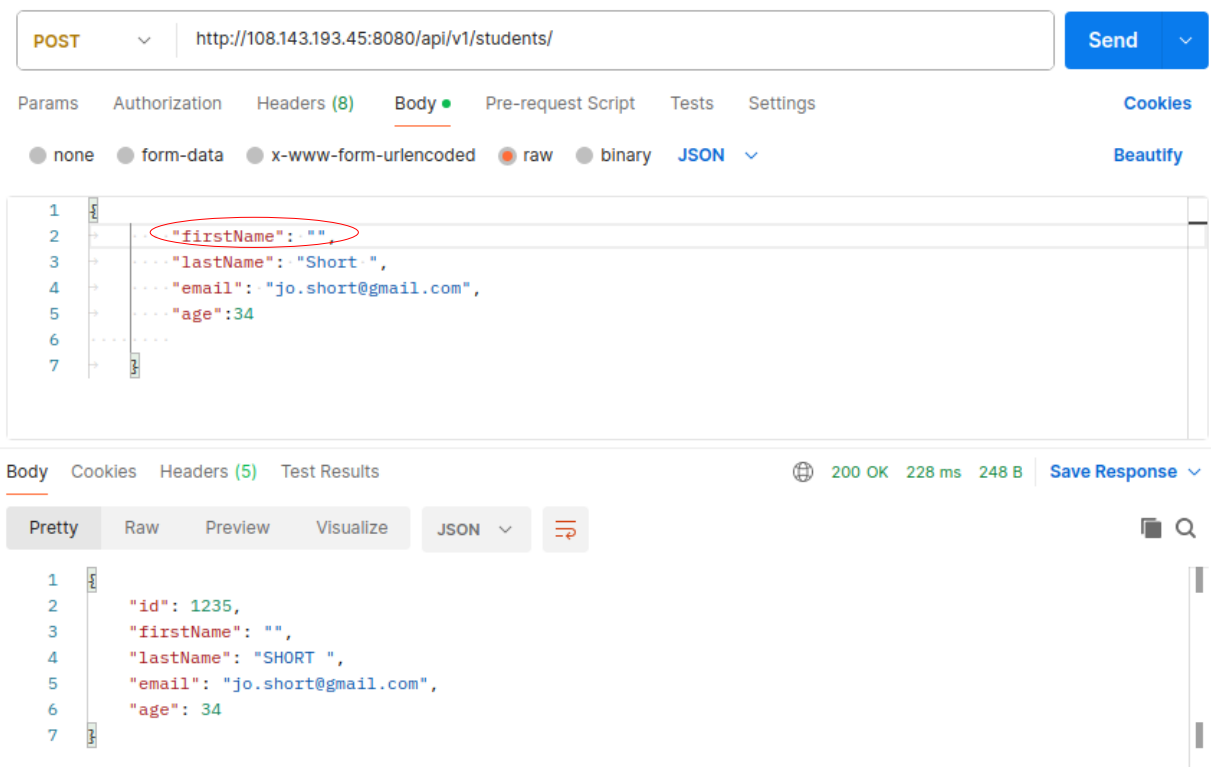
- ➔ firstName missing
- ➔ e-mail address missing

Expected results

- status code: *400 Bad Request*
- a) error message: *"firstName is mandatory"*
- b) error message: *"email is mandatory"*
- The student must not be created – to be checked in the MySQL Workbench

a) firstName missing

1. Postman



Actual result:

- **200 OK** – this is **bug**
- new id 1235 has been assigned – **bug** (student must not be created)
- the response body in JSON contains data listed when registering the new student.

2. MySQL Workbench

Expected result: Student must not be created.

```
SELECT * FROM student
```

```
WHERE id=1235;
```

Actual result: The student with id 1235 has really been created. - **bug**

Query 1 ✖

Limit to 1000 rows

```

1 • SELECT * FROM student
2 WHERE id=1235;
3

```

Result Grid

#	id	age	email	first_name	last_name
1	1235	34	jo.short@gmail.com		SHORT
*	NULL	NULL	NULL	NULL	NULL

10 10:14:50 SELECT * FROM student WHERE id=1235 LIMIT 0, 1000 1 row(s) returned 0,021 sec / 0,00001...

b) email missing

1. Postman

POST ▼ http://108.143.193.45:8080/api/v1/students/ Send ▼

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

none form-data x-www-form-urlencoded raw binary **JSON** ▼ Beautify

```

1 {
2   "firstName": "Ian",
3   "lastName": "Short",
4   "email": "",
5   "age": 34
6 }
7

```

Body Cookies Headers (5) Test Results 200 OK 236 ms 233 B Save Response ▼

Pretty Raw Preview Visualize **JSON** ▼ ≡

```

1 {
2   "id": 1236,
3   "firstName": "Ian",
4   "lastName": "SHORT",
5   "email": "",
6   "age": 34
7 }

```

Actual result:

- **200 OK** – this is **bug**
- new id 1236 has been assigned – **bug**
- the response body in JSON contains data listed when registering the new student.

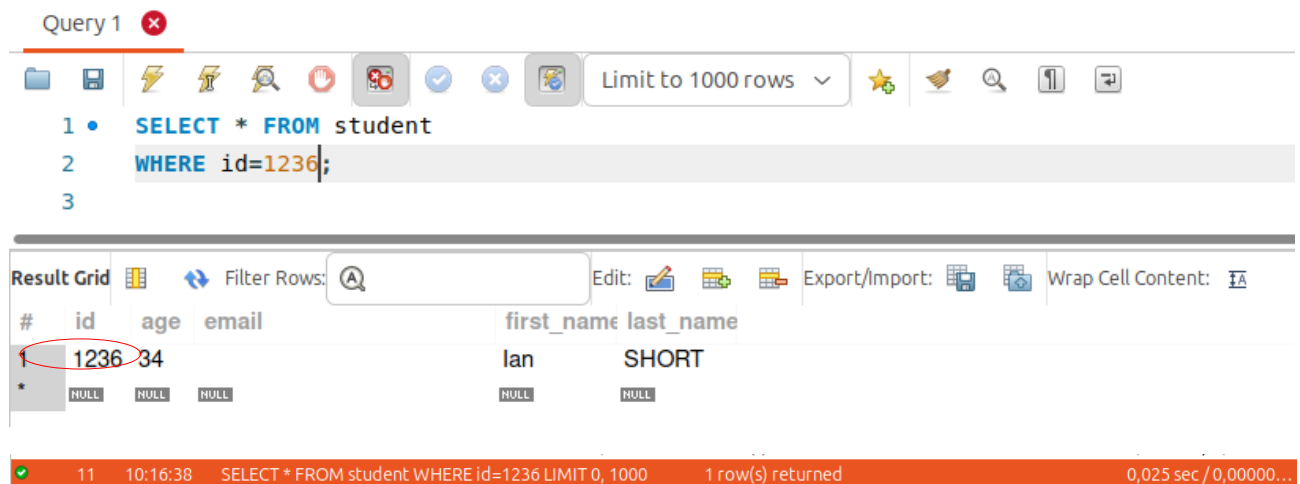
2. MySQL Workbench

Expected result: Student must not be created.

Query

```
SELECT * FROM student  
WHERE id=1236;
```

Actual result: The student with id 1236 has really been created. - **bug**



The screenshot shows the MySQL Workbench interface. At the top, a query editor window titled 'Query 1' contains the following SQL query:

```
1 • SELECT * FROM student  
2 WHERE id=1236;  
3
```

Below the query editor, the 'Result Grid' tab is active, displaying the results of the query. The results are shown in a table with the following columns: #, id, age, email, first_name, and last_name. The first row of data is highlighted, showing a student with id 1236, age 34, and first_name 'lan'. The last_name is 'SHORT'. The email field is empty, and the first_name and last_name fields are also empty. The status bar at the bottom indicates that the query was executed successfully, returning 1 row(s) in 0.025 seconds.

#	id	age	email	first_name	last_name
1	1236	34		lan	SHORT

2. 4 Negative scenario: Unsuccessful creation of a new student record with duplicated e-mail address

1. Postman

Similarly to former POST scenarios, I selected HTTP method POST, typed URL <http://108.143.193.45:8080/api/v1/students/> with no student ID.

Then I completed the Body part - data type: raw and JSON format. Finally I inserted different brand new data except for the email address, which I used from the last record (jo.short@gmail.com).

```
{  
  "firstName": "Peter",  
  "lastName": "Keeping",  
  "email": "jo.short@gmail.com",  
  "age": 40  
}
```


Expected results

- status code: 409 Conflict (or 400 Bad Request)
- error message: "student with this email address already exists"
- Result: The student must not be created

Actual results

- status code: **200 OK - bug**
- new id 1237 assigned and new student created – **bug**
- the response body in JSON contains data listed when registering the new student.

The screenshot shows a REST client interface with the following details:

- URL:** `http://108.143.193.45:8080/api/v1/students/`
- Method:** `POST`
- Body (Request):** A JSON object with the following fields:

```
{  "firstName": "Peter",  "lastName": "Keeping",  "email": "jo.short@gmail.com",  "age": 40}
```

The email field is circled in red.
- Status:** `200 OK` (circled in red), 293 ms, 254 B
- Body (Response):** A JSON object with the following fields:

```
{  "id": 1237,  "firstName": "Peter",  "lastName": "KEEPING",  "email": "jo.short@gmail.com",  "age": 40}
```

The email field is circled in red.

2. Workbench

Final check-up that the new record for student with duplicated email address has not been created.

`SELECT * FROM student WHERE id=1237;`

Query 1 ✖

Limit to 1000 rows

```

1 • SELECT * FROM student
2   WHERE id=1237;
3

```

Result Grid

#	id	age	email	first_name	last_name
1	1237	40	jo.short@gmail.com	Peter	KEEPING
*	NULL	NULL	NULL	NULL	NULL

12 10:19:28 SELECT * FROM student WHERE id=1237 LIMIT 0, 1000 1 row(s) returned 0,038 sec / 0,00001...

The student with id 1237 has really been created. – **bug**

Final step:

Now, to properly prove the bug, I filter the database and use a simple query to check whether the email address jo.short@gmail.com is unique.

```

SELECT * FROM student
WHERE email = "jo.short@gmail.com";

```

Query 1 ✖

Limit to 1000 rows

```

1 • SELECT * FROM student
2   WHERE email="jo.short@gmail.com";
3

```

Result Grid

#	id	age	email	first_name	last_name
1	1231	35	jo.short@gmail.com	Jo	SHORT
2	1232	149	jo.short@gmail.com	Ian	SHORT
3	1233	150	jo.short@gmail.com	Ian	SHORT
4	1234	151	jo.short@gmail.com	Ian	SHORT
5	1235	34	jo.short@gmail.com		SHORT
6	1237	40	jo.short@gmail.com	Peter	KEEPING
*	NULL	NULL	NULL	NULL	NULL

Actual result:

There are 6 occurrences of the same email address in the database. This means that the database is not designed to prevent duplicates in the email address. This, in my view, is a **bug**.

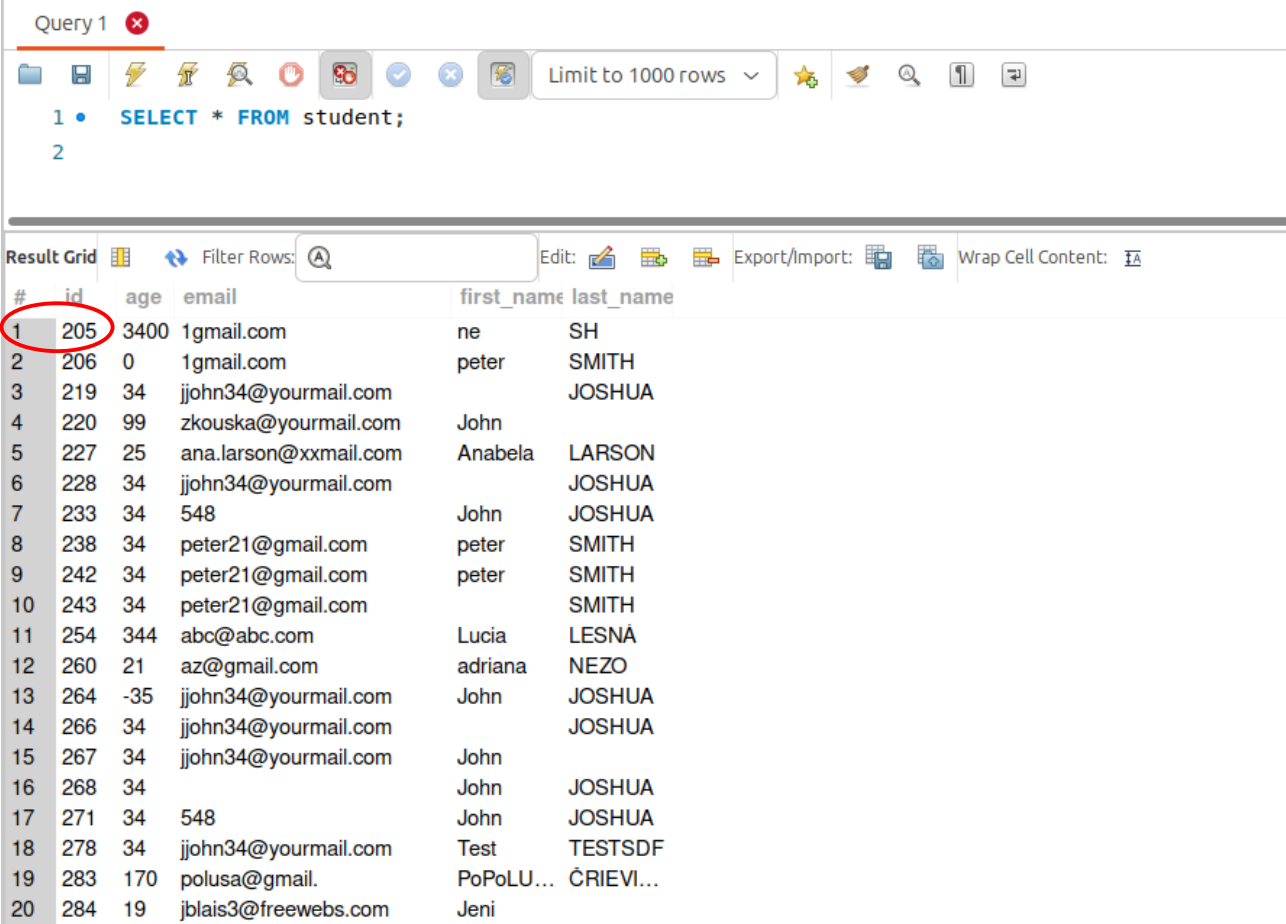
3. HTTP method: DELETE

3.1 Positive scenario: Successful deletion of all data about an existing student from the database

1. MySQL Workbench: Test data preparation:

Firstly, I have to verify existence of a student with selected id. With a simple query I display a list of all students and pick the first one with **id 205**.

```
SELECT * FROM student;
```



Query 1

```
1 • SELECT * FROM student;
```

Limit to 1000 rows

#	id	age	email	first_name	last_name
1	205	3400	1gmail.com	ne	SH
2	206	0	1gmail.com	peter	SMITH
3	219	34	jjohn34@yourmail.com		JOSHUA
4	220	99	zkouska@yourmail.com	John	
5	227	25	ana.larson@xxmail.com	Anabela	LARSON
6	228	34	jjohn34@yourmail.com		JOSHUA
7	233	34	548	John	JOSHUA
8	238	34	peter21@gmail.com	peter	SMITH
9	242	34	peter21@gmail.com	peter	SMITH
10	243	34	peter21@gmail.com		SMITH
11	254	344	abc@abc.com	Lucia	LESNA
12	260	21	az@gmail.com	adriana	NEZO
13	264	-35	jjohn34@yourmail.com	John	JOSHUA
14	266	34	jjohn34@yourmail.com		JOSHUA
15	267	34	jjohn34@yourmail.com	John	
16	268	34		John	JOSHUA
17	271	34	548	John	JOSHUA
18	278	34	jjohn34@yourmail.com	Test	TESTSDF
19	283	170	polusa@gmail.	PoPoLU...	CRIEVI...
20	284	19	jblais3@freewebs.com	Jeni	

```
SELECT * FROM student
```

```
WHERE id = 205;
```

Query 1 ✕

1 • **SELECT** * **FROM** student
 2 **WHERE** id = 205;
 3

Result Grid Filter Rows: A Edit: Export/Import: Wrap Cell Content: IA

#	id	age	email	first_name	last_name
1	205	3400	1gmail.com	ne	SH
*	NULL	NULL	NULL	NULL	NULL

I get the result '205','3400','1gmail.com','ne','SH' , which, by the way, is an excellent example of a student record breaking all stated conditions (age beyond the stated limit, email in incorrect format, first and last name with insufficient length, and surname not stored in lowercase)

2. Postman

a) I now display the student record in Postman with GET method, insert URL <http://108.143.193.45:8080/api/v1/students/205> to read data about the existing student with id 205 and click Send button.

HTTP <http://108.143.193.45:8080/api/v1/students/205> Save

GET v <http://108.143.193.45:8080/api/v1/students/205> Send v

Params Authorization Headers (6) Body Pre-request Script Tests Settings Cookies

Query Params

	Key	Value	Bulk Edit
	Key	Value	

Body Cookies Headers (5) Test Results 200 OK 234 ms 239 B Save Response v

Pretty Raw Preview Visualize JSON v ≡

```

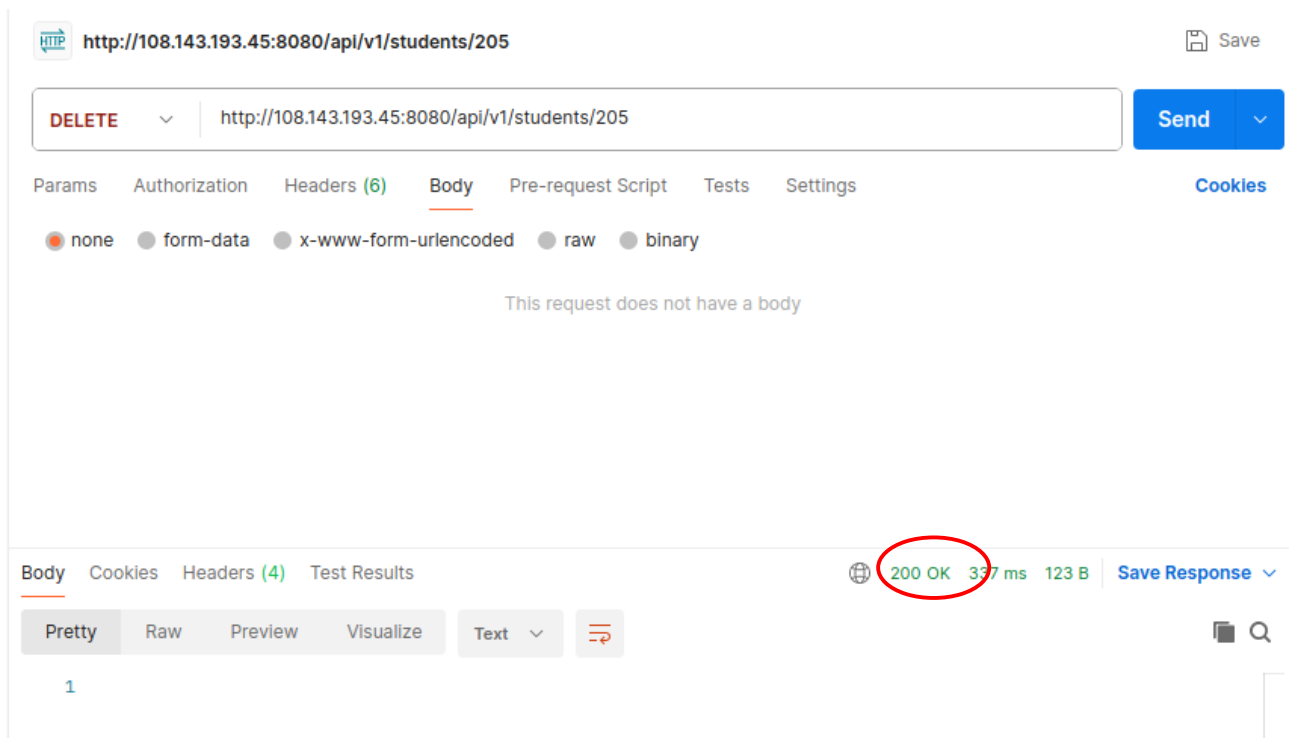
1 {
2   "id": 205,
3   "firstName": "ne",
4   "lastName": "SH",
5   "email": "1gmail.com",
6   "age": 3400
7 }
  
```

I have got result as expected :

- status code: **200 OK**
- data about student in JSON format corresponding to the record in MySQL Workbench
- the response body in JSON contains data listed when registering the new student.

b) Now with existing student verified, I delete this student. I select HTTP method DELETE, insert URL <http://108.143.193.45:8080/api/v1/students/205> and click the Send button.

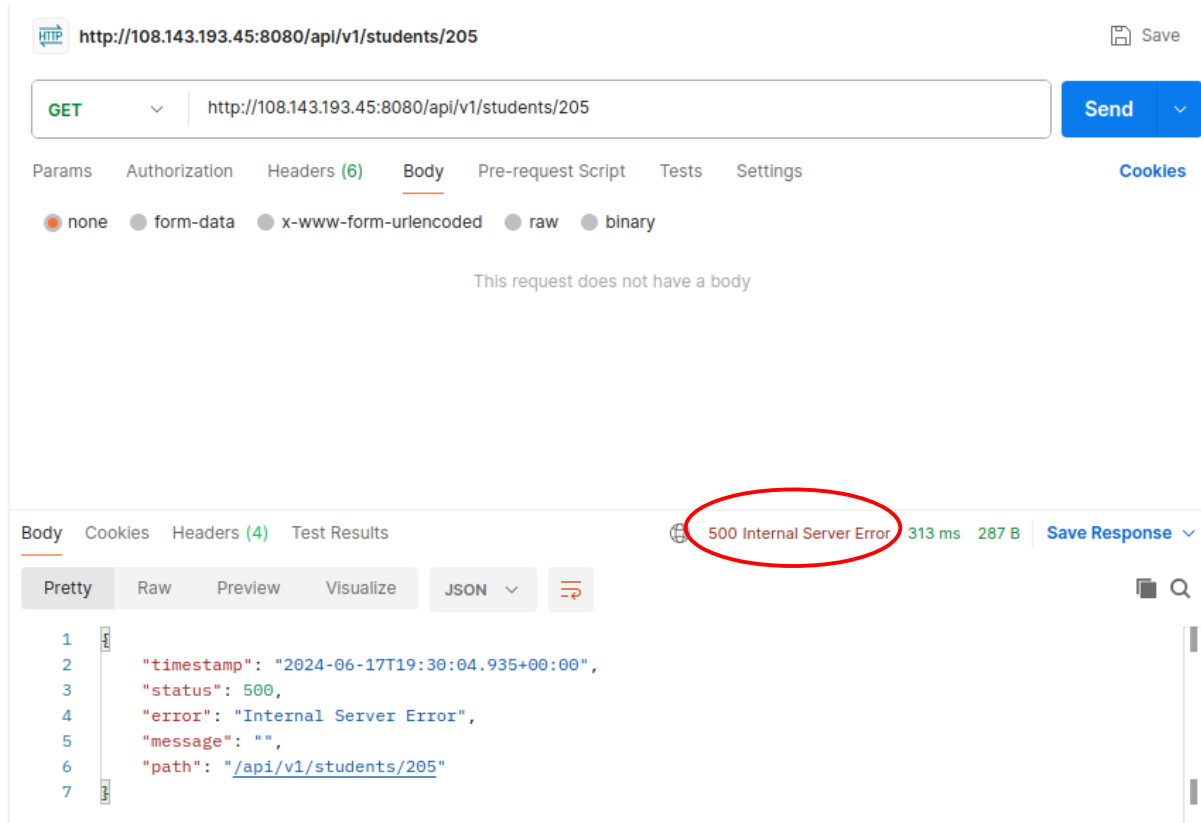
Actual result corresponds to the expected one: Status code: **200 OK** (or **204 No content**).



c) As a final step, I need to verify that the deleted student is no longer stored in the database.

Firstly in Postman, I select HTTP method GET, insert URL: <http://108.143.193.45:8080/api/v1/students/205> and click the Send button

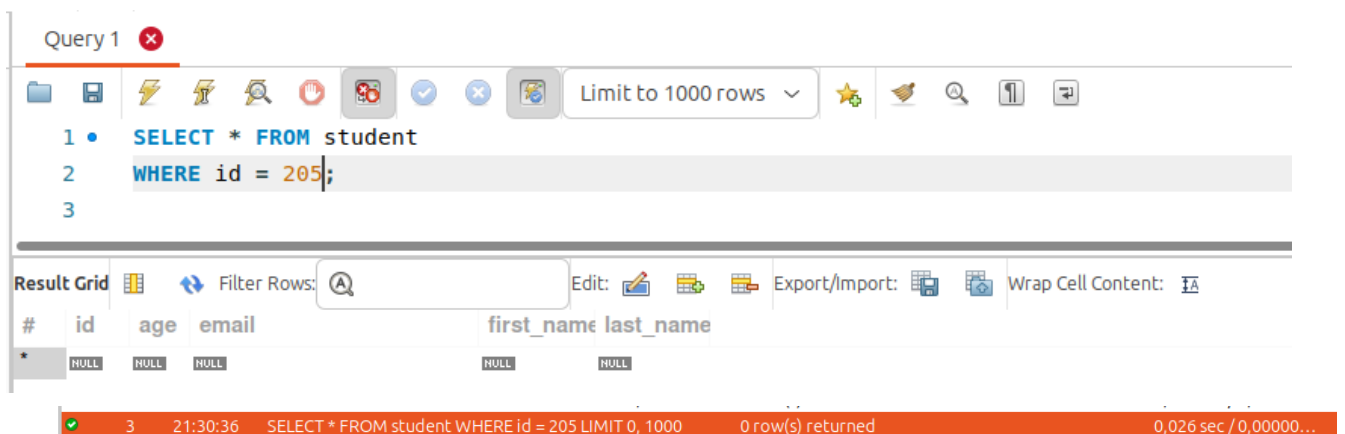
Expected result: Status code **404 Not found**



Actual result: **500 Internal Server Error - bug**

For a final check to ensure that the deleted student is no longer in the database, I write a simple query in MySQL Workbench:

```
SELECT * FROM student  
WHERE id=205;
```



I receive back 0 rows meaning there is no student with id 205. The student record has been really deleted from the database.

3.2 Negative scenario 1: Deletion of a non-existent student

I now try to delete the same student I deleted in 3.1 scenario, i.e. with id 205

1. MySQL Workbench: I have verified the non-existence of a student with id 205 above on page 48.

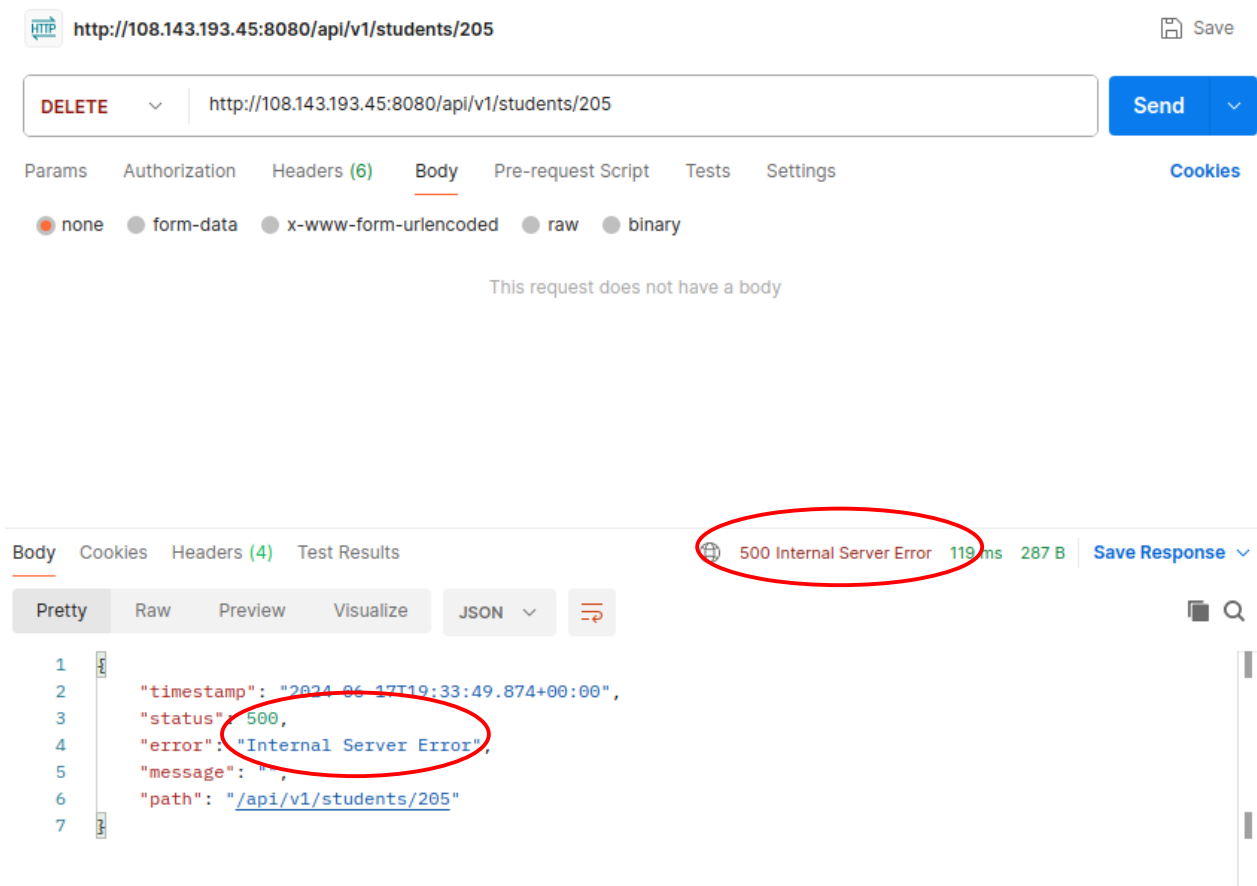
```
SELECT * FROM student
```

```
WHERE id=205;
```

2. Postman

Now, I delete this non-existing student in Postman. I select HTTP method DELETE, insert URL: <http://108.143.193.45:8080/api/v1/students/205> and click Send button

Expected results: status code: 404 Not Found.

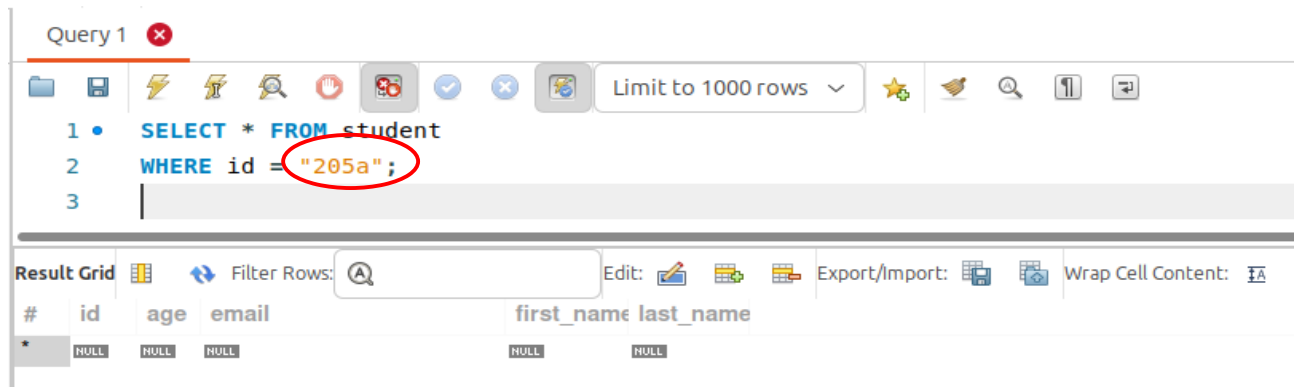


The actual result: status code **500 Internal Server Error** – I have found **a bug**

3.3 Negative scenario 2: Deletion of a student with invalid ID

1. **MySQL Workbench:** Test data preparation: verify the **id** is in incorrect format

SELECT * FROM student where id=205a;



4 21:35:14 SELECT * FROM student WHERE id = "205a" LIMIT 0, 1000 0 row(s) returned 0,028 sec / 0,00001...

2. Postman

I select HTTP method DELETE, insert URL:

<http://108.143.193.45:8080/api/v1/students/205a> and click the Send button.

Expected result:

- status code: 400,
- Error: Bad Request

Actual result:

- status code: 400
- Error: Bad Request – **OK**

Actual result corresponds to the expected one: Status code: **400 Bad Request**

HTTP <http://108.143.193.45:8080/api/v1/students/205a>

Save

DELETE <http://108.143.193.45:8080/api/v1/students/205a>

Send

Params Authorization Headers (6) Body Pre-request Script Tests Settings

Cookies

☒ none ☐ form-data ☐ x-www-form-urlencoded ☐ raw ☐ binary

This request does not have a body

Body Cookies Headers (4) Test Results

400 Bad Request 66 ms 268 B Save Response

Pretty Raw Preview Visualize JSON

```
1 {
2   "timestamp": "2024-06-17T19:36:24.843+00:00",
3   "status": 400,
4   "error": "Bad Request",
5   "message": "",
6   "path": "/api/v1/students/205a"
7 }
```

BUG REPORT

Based on the executed scenarios, I have identified the following application errors.

Test scenario	Abstract	Expected Result	Bug / Actual Result
1.2	Getting data about a non-existent student (student with non-existing id)	404 – Not found	500 – Internal server error
1.3	Getting data about a non-existent student using wrong id format - spacing	400 – Bad Request	500 – Internal server error
1.3	Getting data about a non-existent student using wrong id format – question mark ?	400 – Bad Request	200 – OK Postman: Response body returns data for all students in the database
2.1	New student record in the database successfully created	201 - Created	200 – OK MySQL Workbench: The criterion ' <i>All attributes are stored in the database in lowercase format</i> ' has not been met.
2.2.1	New student record with non-standard entries – very long email address	201 - Created	200 – OK MySQL Workbench: The criterion ' <i>All attributes are stored in the database in lowercase format</i> ' has not been met.
2.2.2	New student record with non-existent domain in the email address	400 – Bad Request	200 – OK MySQL Workbench: The criterion ' <i>All attributes are stored in the database in lowercase format</i> ' has not been met.
2.2.3	New student record	400 – Bad Request	200 – OK

	with incorrect firstName format combining letters and numbers		MySQL Workbench: The criterion ' <i>All attributes are stored in the database in lowercase format</i> ' has not been met.
2.3.1	Unsuccessful creation of a new student record with firstName below 3 signs	400 – Bad Request Error message: "firstName must be between 3-50 characters" The student must not be created.	200 – OK student with new id created. MySQL Workbench: The criterion ' <i>All attributes are stored in the database in lowercase format</i> ' has not been met.
2.3.2	Unsuccessful creation of a new student record with email in incorrect format	400 – Bad Request Error message: "Email must be in format *@*.*" The student must not be created.	200 – OK student with new id created. MySQL Workbench: The criterion ' <i>All attributes are stored in the database in lowercase format</i> ' has not been met.
2.3.3 a)	Successful creation of a new student record for age 149 and 150	status code: 201 created new student record with new assigned id 1232 and 1233	200 - OK student with new id created. MySQL Workbench: The criterion ' <i>All attributes are stored in the database in lowercase format</i> ' has not been met.
2.3.3 b)	Unsuccessful creation of a new student record for age 151	status code: 400 Bad Request error message: "age must be between 1-150"	200 - OK newly assigned id 1234 The student with id 1234 has really been created. MySQL Workbench: The criterion ' <i>All attributes are stored in the database in lowercase format</i> ' has not been met.
2.3.4 a)	Unsuccessful creation of a new student record with first name missing	400 - Bad Request error message: "firstName is	200 - OK, new id has been assigned

		mandatory" The student must not be created.	The student with id 1235 has really been created. MySQL Workbench: The criterion ' <i>All attributes are stored in the database in lowercase format</i> ' has not been met.
2.3.4 b)	Unsuccessful creation of a new student record with email address missing	400 - Bad Request error message: "email is mandatory" The student must not be created.	200 - OK, new id has been assigned The student with id 1236 has really been created. MySQL Workbench: The criterion ' <i>All attributes are stored in the database in lowercase format</i> ' has not been met.
3.1	Successful deletion of all data about an existing student from the database, step verification that the deleted student is no longer stored in the database	404 - Not found	500 - Internal Server Error
3.2	Unsuccessful deletion of a non-existent student	404 - Not found	500 - Internal Server Error