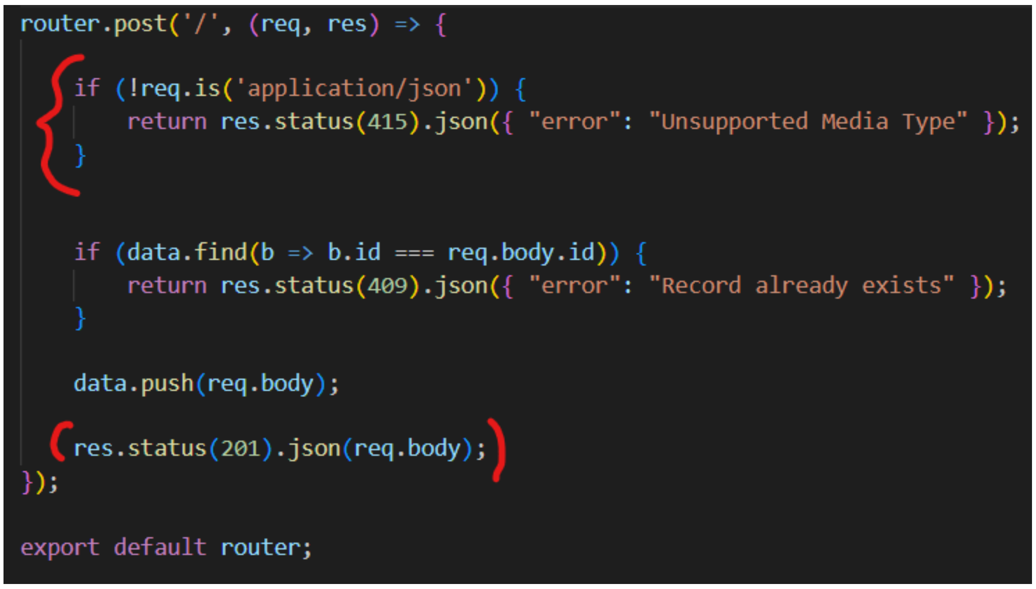
Marco Berrocal

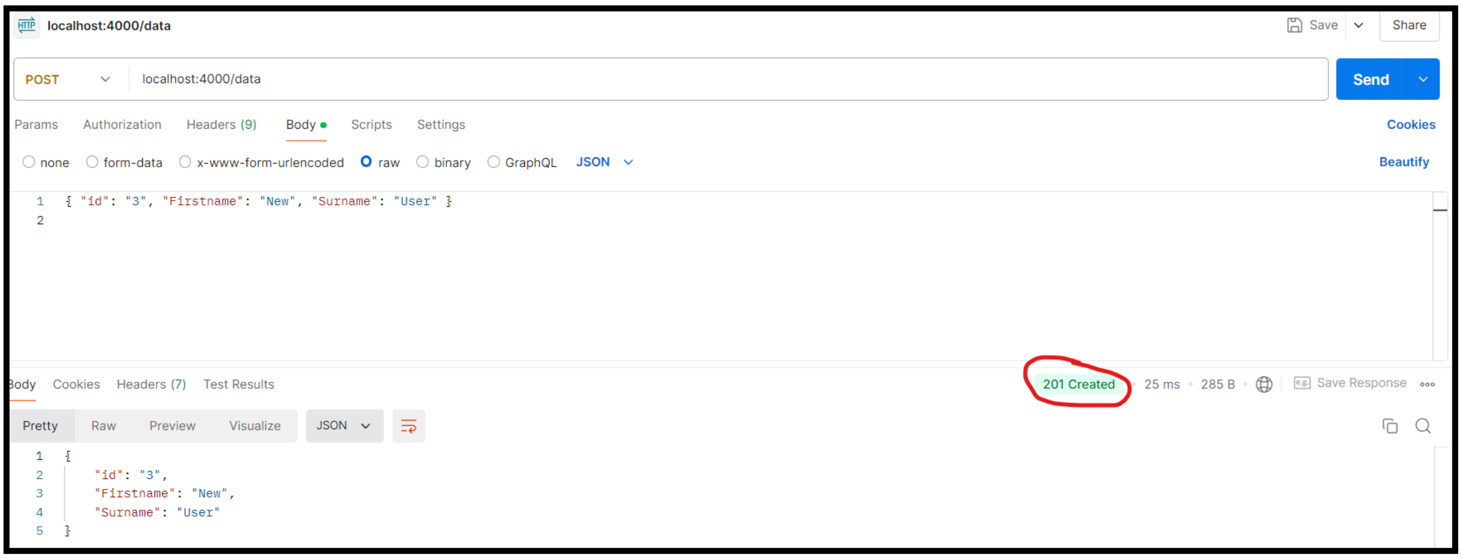
Exercise 5 - Adding PUT and DELETE. Automated testing.

a) We make sure that the media type is application/json with this code, and we send the proper messages in case it's not the case.



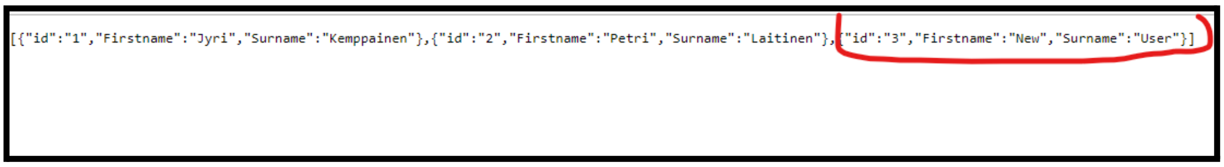
```
router.post('/', (req, res) => {
    if (!req.is('application/json')) {
        return res.status(415).json({ "error": "Unsupported Media Type" });
    }

    if (data.find(b => b.id === req.body.id)) {
        return res.status(409).json({ "error": "Record already exists" });
    }

    data.push(req.body);

    res.status(201).json(req.body);
});

export default router;
```

TEST: Correct sending (Postman)



```
POST localhost:4000/data
Body (raw, JSON):
{ "id": "3", "Firstname": "New", "Surname": "User" }

201 Created  25 ms  285 B
{
    "id": "3",
    "Firstname": "New",
    "Surname": "User"
}
```

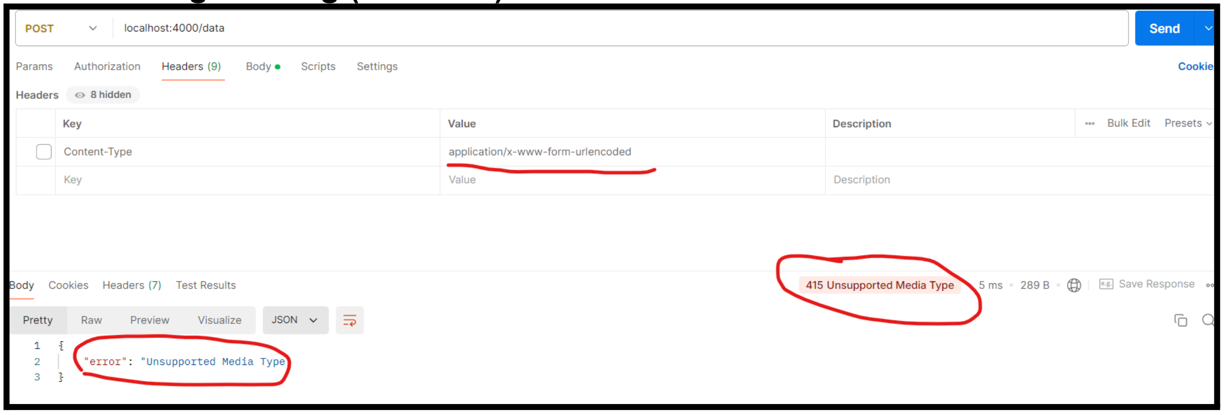
CHECKING localhost:4000/data



```
[{"id":"1","Firstname":"Jyri","Surname":"Kemppainen"},{"id":"2","Firstname":"Petri","Surname":"Laitinen"},{"id":"3","Firstname":"New","Surname":"User"}]
```

Marco Berrocal

TEST: Wrong sending (Postman)


POST localhost:4000/data

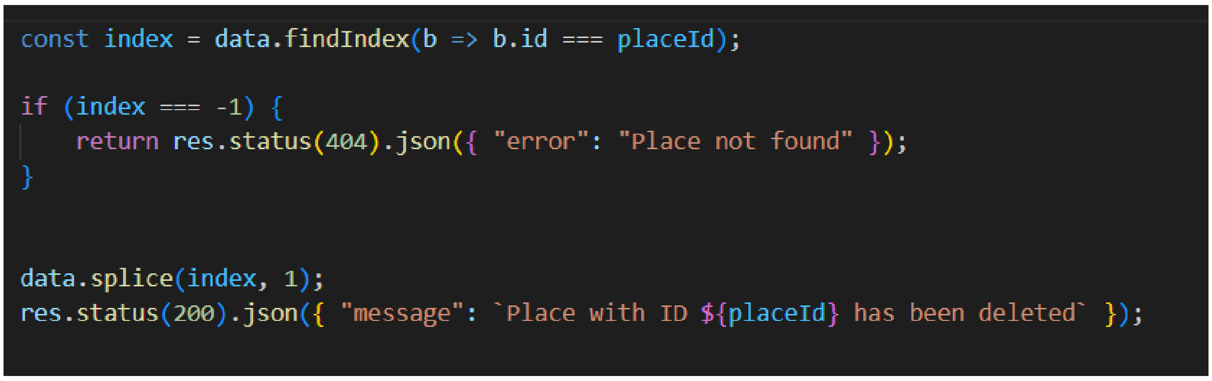
Headers (9)

Key	Value	Description
Content-Type	application/x-www-form-urlencoded	

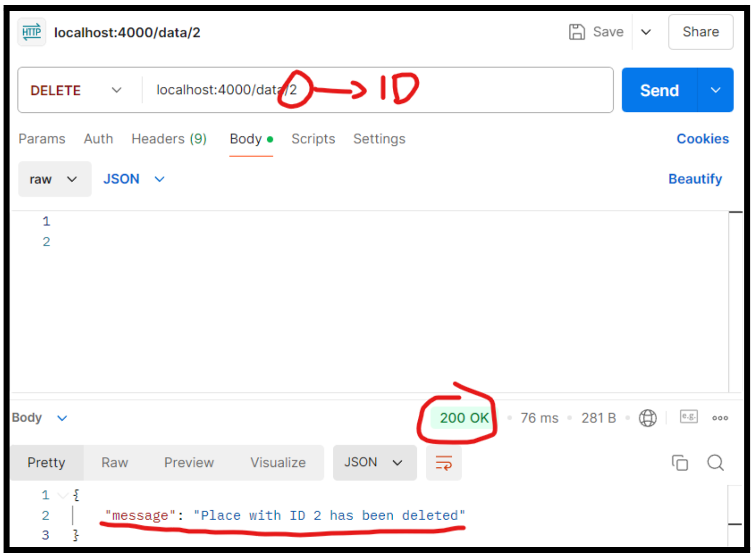
415 Unsupported Media Type · 5 ms · 289 B

```
{
  "error": "Unsupported Media Type"
}
```

c) In this case the code added should look like this, making sure that in case the ID doesn't exist (-1), the error 404 will pop. And on the other hand, if the ID exists (-1), it has to be deleted.


```
const index = data.findIndex(b => b.id === placeId);

if (index === -1) {
    return res.status(404).json({ "error": "Place not found" });
}


data.splice(index, 1);
res.status(200).json({ "message": `Place with ID ${placeId} has been deleted` });
```

Then we will try to delete the id number 2. In this case the id number 2 exists, so it should be able to delete it.

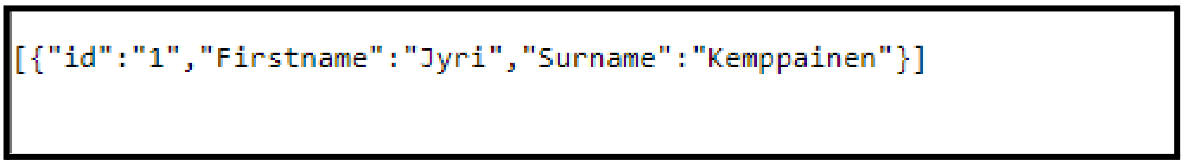

DELETE localhost:4000/data/2

$\xrightarrow{}$ ID

200 OK · 76 ms · 281 B

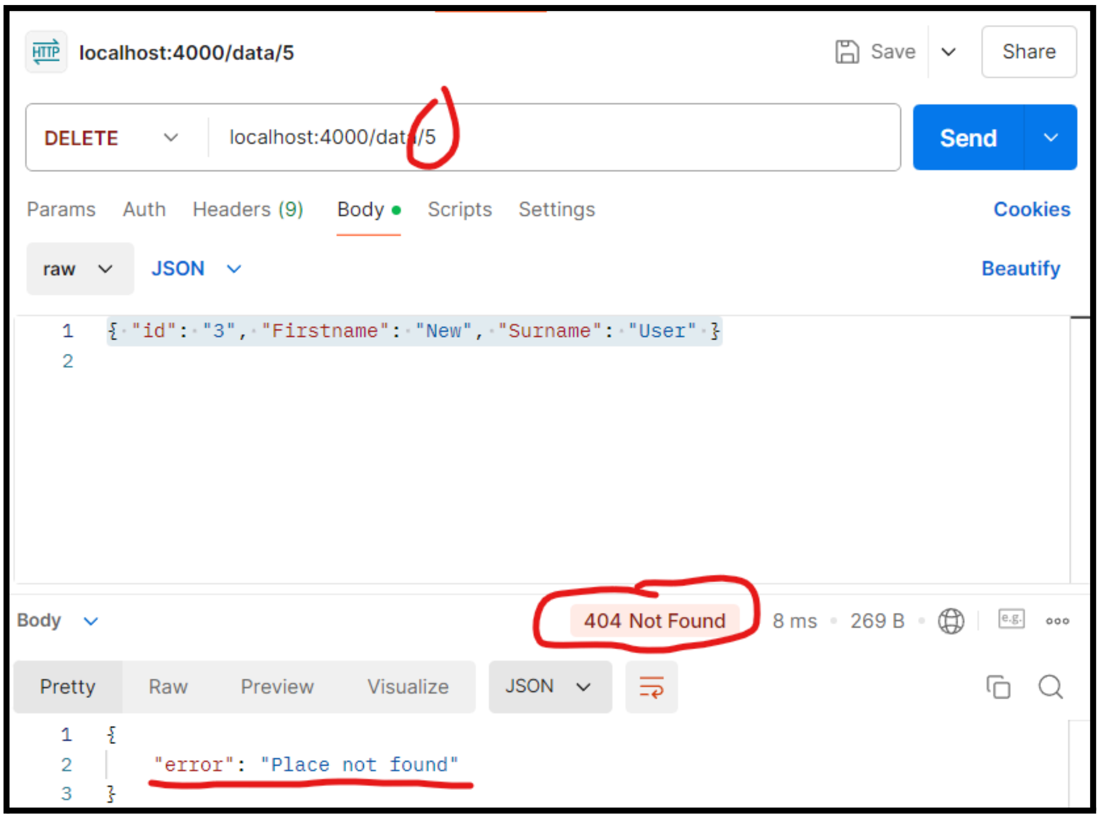
```
{
  "message": "Place with ID 2 has been deleted"
}
```

As we check it in Localhost, there's no element with id=2 anymore, only the id=1, so it has been deleted successfully.



```
[{"id":"1","Firstname":"Jyri","Surname":"Kemppainen"}]
```

But what if we use a different id? In this case the id=5 is not existing, so it will retrieve the information about not being able to find this place.



localhost:4000/data/5

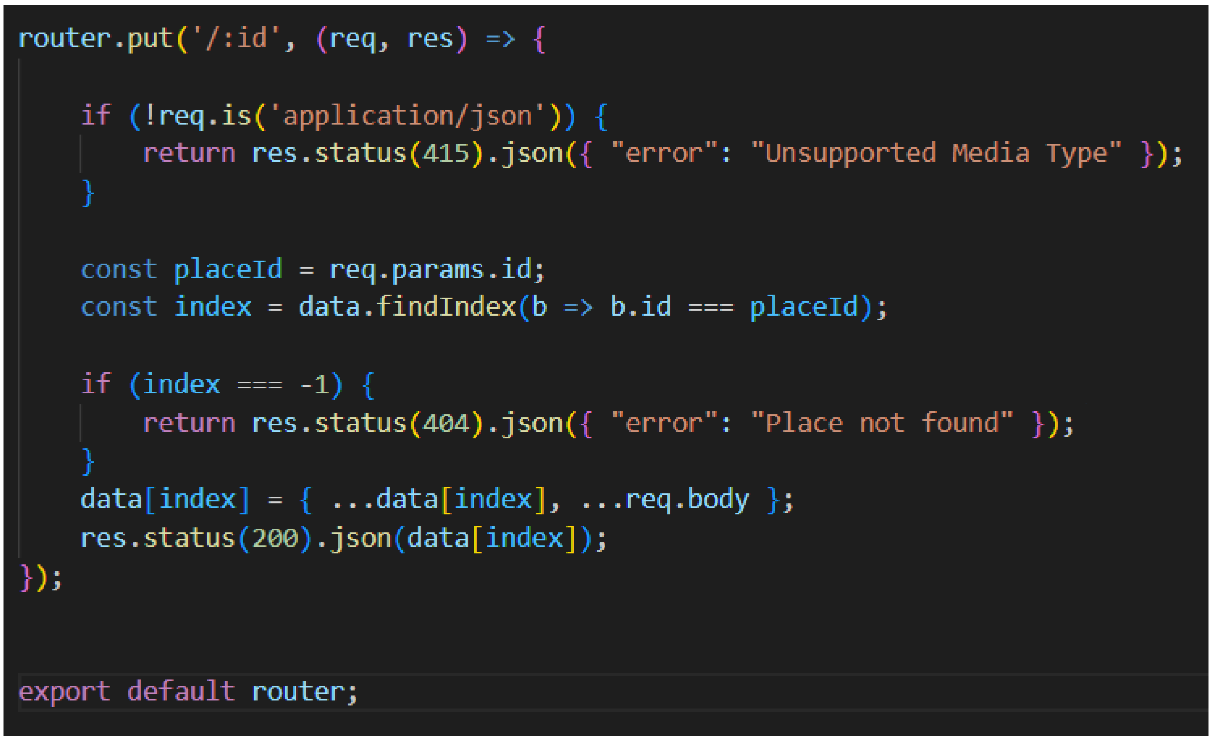
DELETE localhost:4000/data/5

```
{ "id": "3", "Firstname": "New", "Surname": "User" }
```

404 Not Found 8 ms 269 B

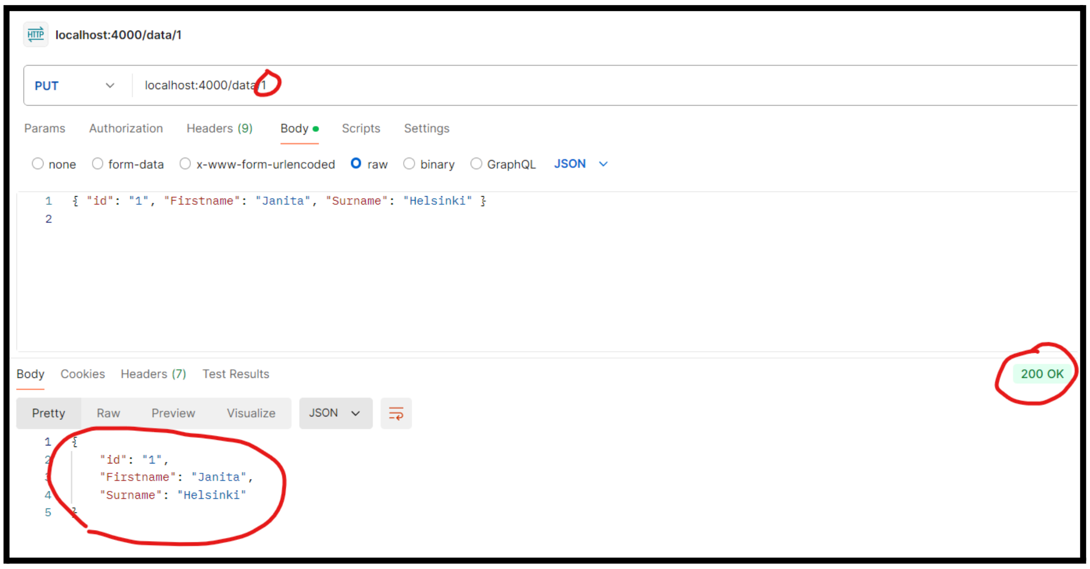
```
{
    "error": "Place not found"
}
```

d) In this case, we add a PUT endpoint where we want to substitute a specific ID with other information. In the code below we see how it takes care of sending the proper information if the media type is not supported, and also in case the ID we want to substitute is not found.



```
router.put('/:id', (req, res) => {

    if (!req.is('application/json')) {
        return res.status(415).json({ "error": "Unsupported Media Type" });
    }

    const placeId = req.params.id;
    const index = data.findIndex(b => b.id === placeId);

    if (index === -1) {
        return res.status(404).json({ "error": "Place not found" });
    }
    data[index] = { ...data[index], ...req.body };
    res.status(200).json(data[index]);
});


export default router;
```

Now we test it in Postman, checking that the code works properly.



PUT localhost:4000/data/1

```
{ "id": "1", "Firstname": "Janita", "Surname": "Helsinki" }
```

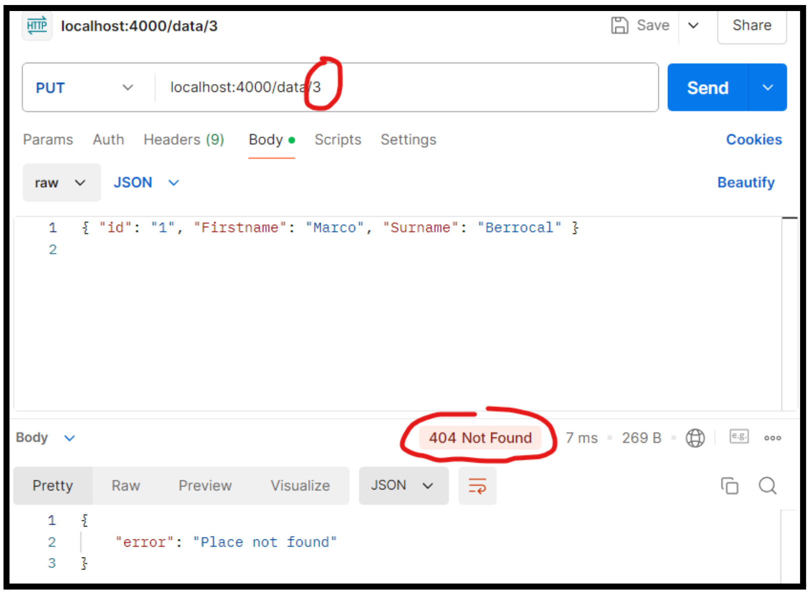
200 OK

```
{
    "id": "1",
    "Firstname": "Janita",
    "Surname": "Helsinki"
}
```

```
[{"id":"1","Firstname":"Janita","Surname":"Helsinki"},{"id":"1","Firstname":"Marco","Surname":"Berrocal"}]
```

We will try the fact of asking for an non-existing id (404 error) and a non-supported media type:

Marco Berrocal



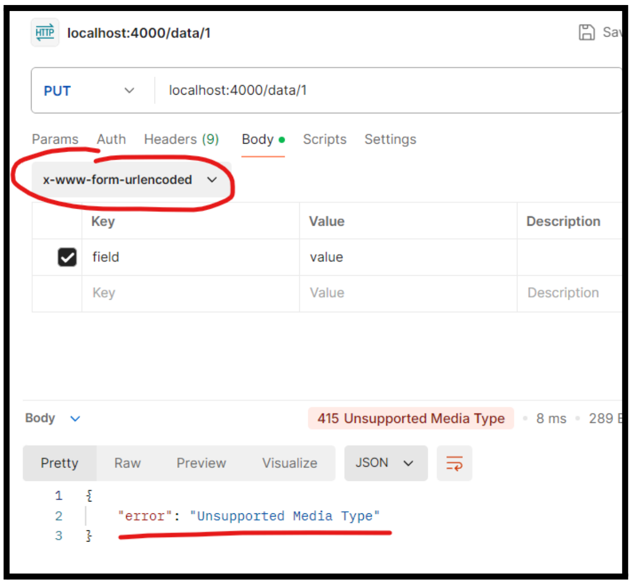
localhost:4000/data/3

PUT localhost:4000/data/3

```
{ "id": "1", "Firstname": "Marco", "Surname": "Berrocal" }
```

404 Not Found · 7 ms · 269 B

```
{
    "error": "Place not found"
}
```



localhost:4000/data/1

PUT localhost:4000/data/1

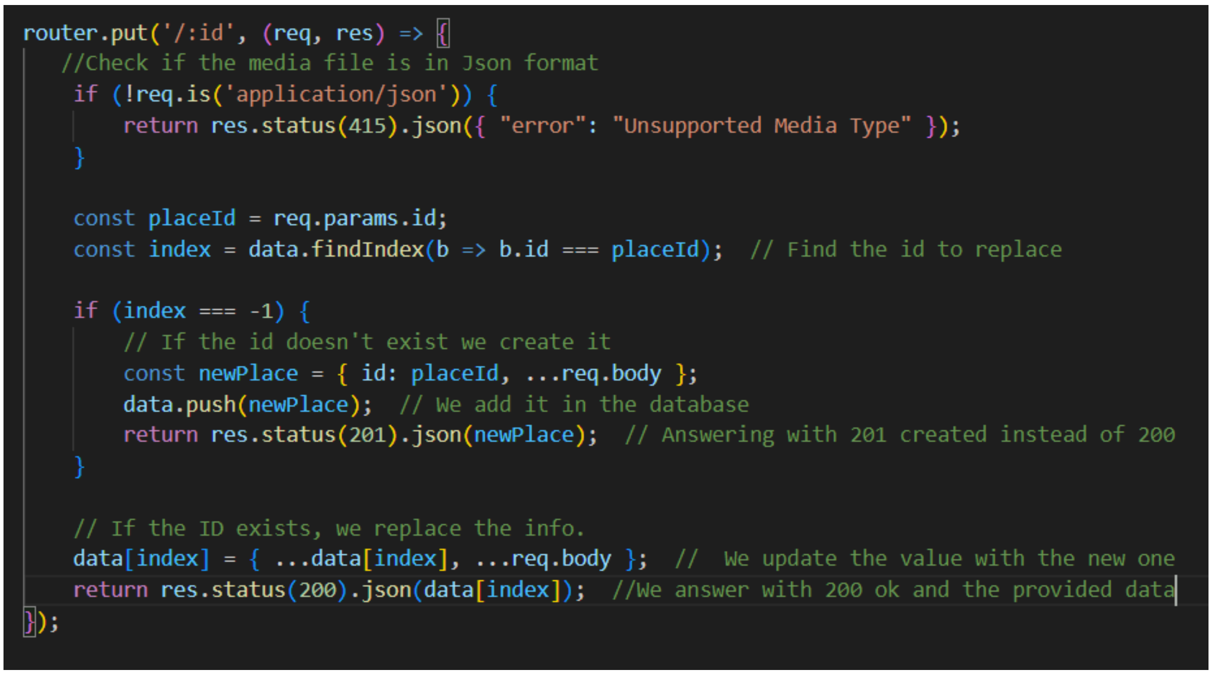
Body: x-www-form-urlencoded

Key	Value	Description
field	value	

415 Unsupported Media Type · 8 ms

```
{
    "error": "Unsupported Media Type"
}
```

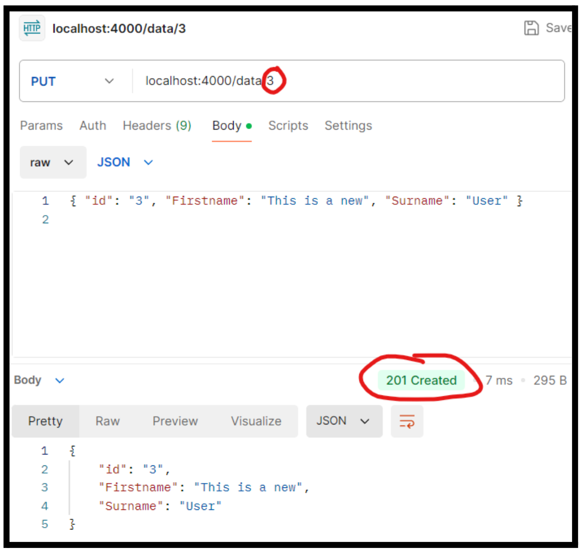
e) Is your PUT method idempotent? If not, then modify it. The status code has to be 201 (Created) if a new record had been created. Otherwise 200 OK.



```
router.put('/:id', (req, res) => {
   //Check if the media file is in Json format
    if (!req.is('application/json')) {
        return res.status(415).json({ "error": "Unsupported Media Type" });
    }

    const placeId = req.params.id;
    const index = data.findIndex(b => b.id === placeId);  // Find the id to replace

    if (index === -1) {
        // If the id doesn't exist we create it
        const newPlace = { id: placeId, ...req.body };
        data.push(newPlace);  // We add it in the database
        return res.status(201).json(newPlace);  // Answering with 201 created instead of 200
    }

    // If the ID exists, we replace the info.
    data[index] = { ...data[index], ...req.body };  //  We update the value with the new one
    return res.status(200).json(data[index]);  //We answer with 200 ok and the provided data
});
```

Marco Berrocal

NEW ID (WITH “PUT”)



localhost:4000/data/3

PUT localhost:4000/data/3

Params Auth Headers (9) Body Scripts Settings

raw JSON

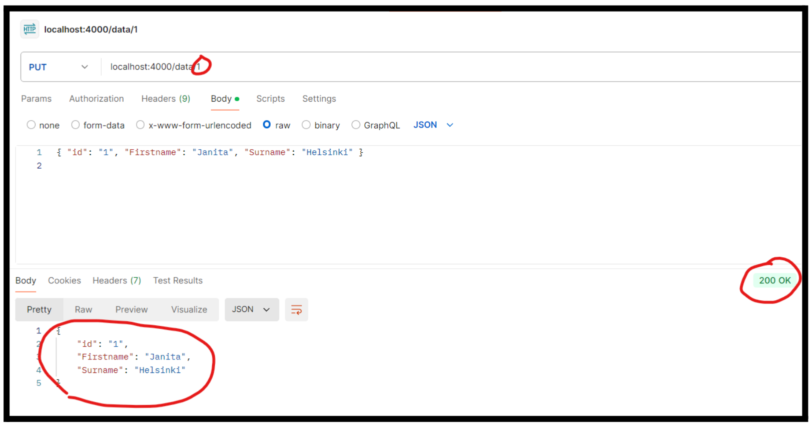
```
{ "id": "3", "Firstname": "This is a new", "Surname": "User" }
```

Body — 201 Created — 7 ms · 295 B

Pretty Raw Preview Visualize JSON

```
{
    "id": "3",
    "Firstname": "This is a new",
    "Surname": "User"
}
```

ALREADY EXISTING ID



localhost:4000/data/1

PUT localhost:4000/data/1

Params Authorization Headers (9) Body Scripts Settings

none form-data x-www-form-urlencoded [✓ raw] binary GraphQL JSON

```
{ "id": "1", "Firstname": "Janita", "Surname": "Helsinki" }
```

Body Cookies Headers (7) Test Results — 200 OK

Pretty Raw Preview Visualize JSON

```
{
    "id": "1",
    "Firstname": "Janita",
    "Surname": "Helsinki"
}
```

f) Select an API endpoint to exercise automated unit testing. Implement a test suite for that endpoint.

Already tested and implemented in code.