UNIVERSITÀ DI TRENTO

REST Basics

Service Design and Engineering - 11/05/2025

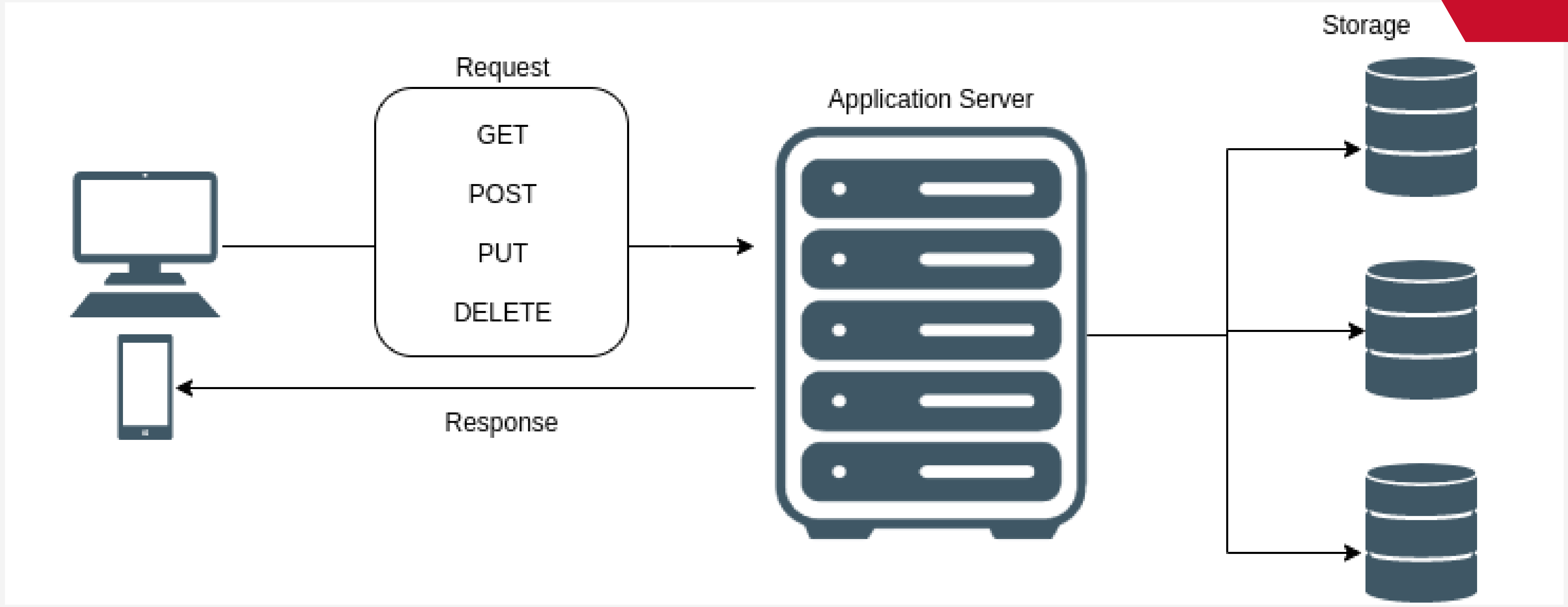
Marco Lasagna
Denise Comincioli
Mehrab Fajar
Lewis Ndambiri
Luca Dematté

Theory

REST principles

- ***client/server decoupling***: both sides of the application need to be completely independent;
- ***uniform interface***: all API requests for the same resource must look the same, independently of where they are coming from;
- ***statelessness***: REST APIs are *stateless* and are not allowed to store any data coming from a request;
- ***cacheability*** (to improve performance on the client's side and scalability on the server's side);
- ***layered architecture***: neither the client nor the server must be able to tell who they are communicating with except for the layer they are directly interacting with;
- ***code on-demand*** (optional): APIs can download code to simplify client implementations.

REST architecture

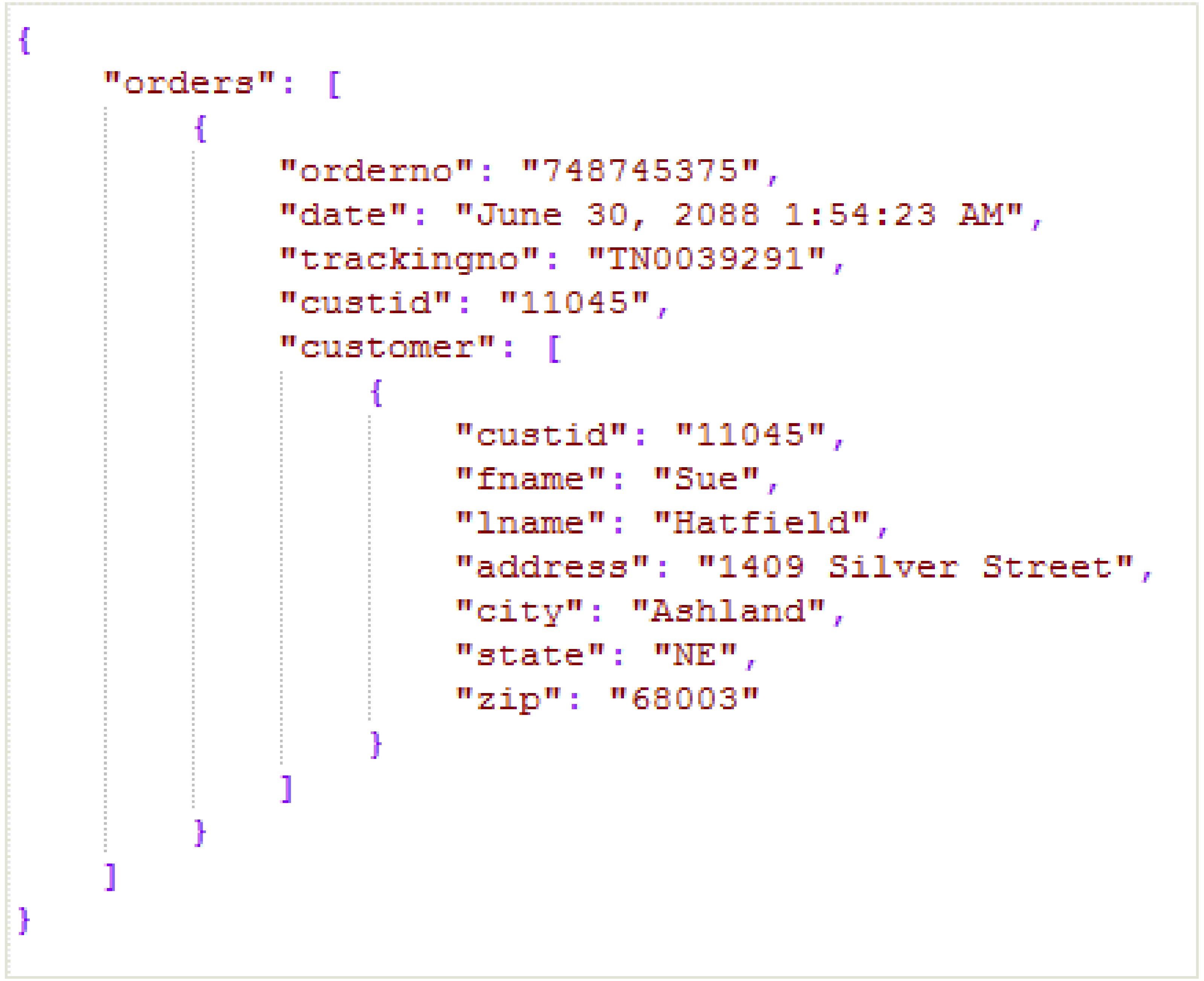


Picture from: https://lo-victoria.com/a-deep-look-into-restful-apis

CRUD vs HTTP

HTTP methods	Matching CRUD methods
GET	Retrieve
POST	Create
PUT	Update
PATCH	Update
DELETE	Delete

JSON

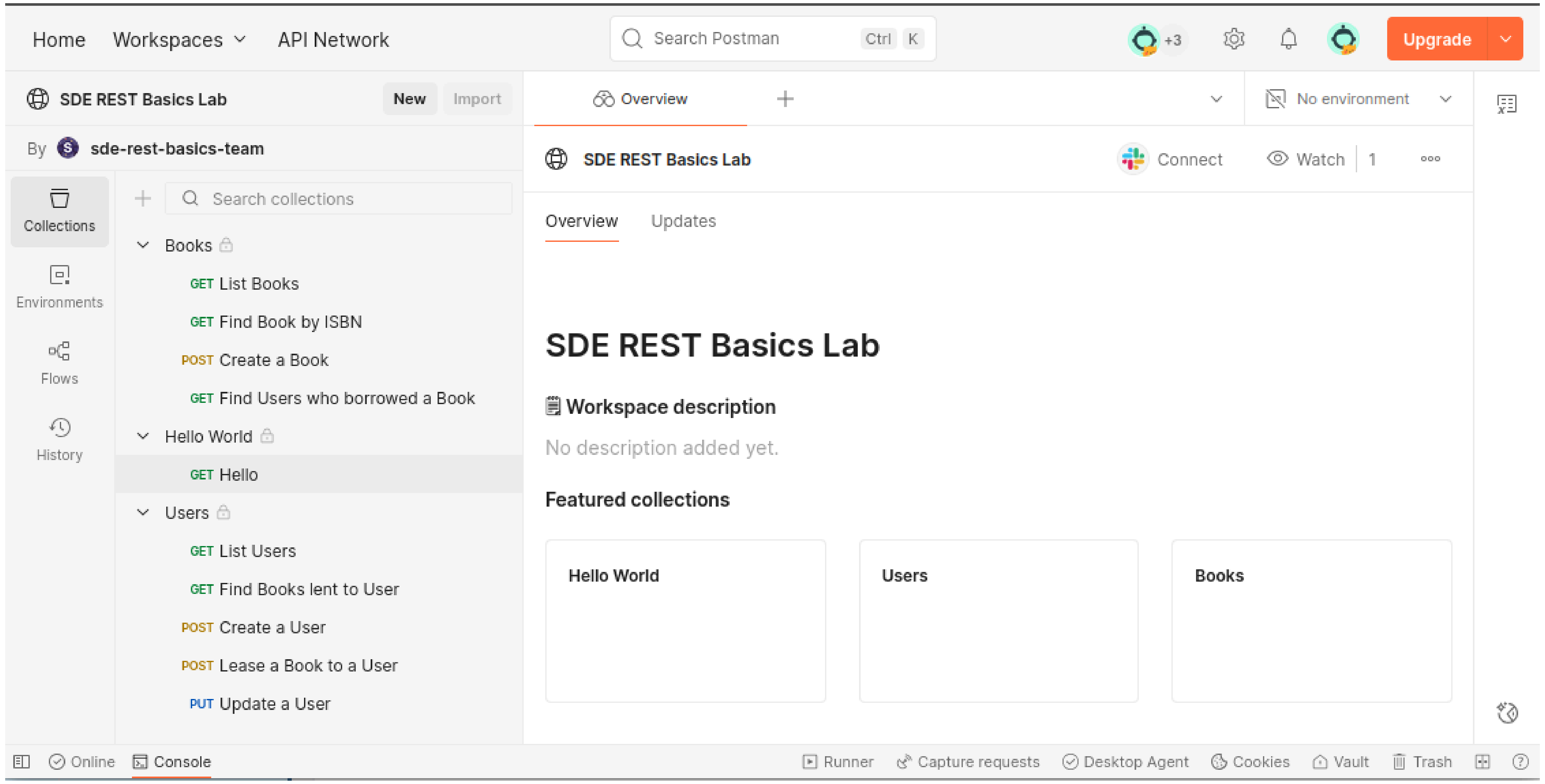


```
{
    "orders": [
        {
            "orderno": "748745375",
            "date": "June 30, 2088 1:54:23 AM",
            "trackingno": "TN0039291",
            "custid": "11045",
            "customer": [
                {
                    "custid": "11045",
                    "fname": "Sue",
                    "lname": "Hatfield",
                    "address": "1409 Silver Street",
                    "city": "Ashland",
                    "state": "NE",
                    "zip": "68003"
                }
            ]
        }
    ]
}
```

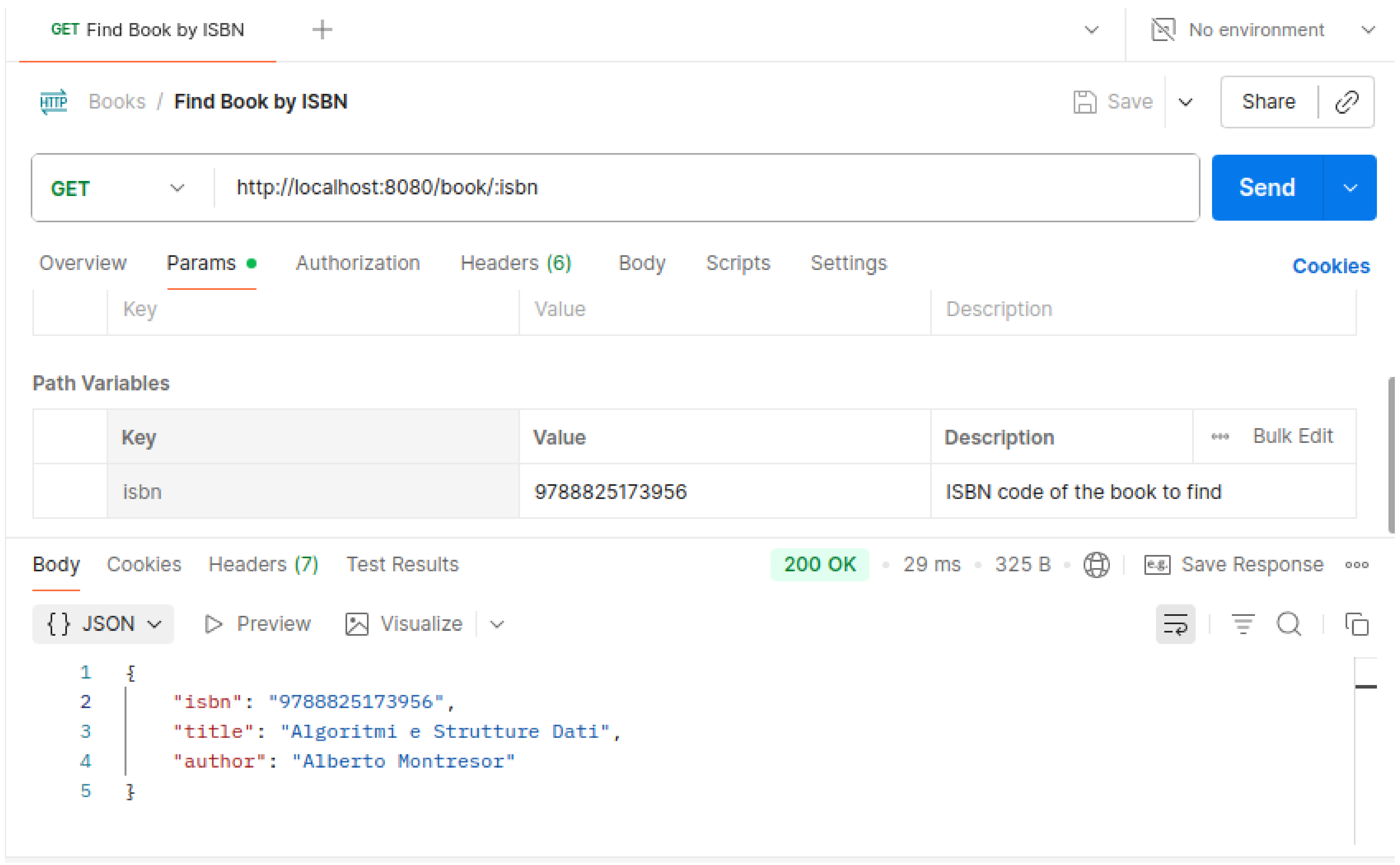
Picture from: https://www.goanywhere.com/resources/training/how-to-read-json-and-insert-into-database

Postman

A tool to test and document web APIs



API testing interface:



GET http://localhost:8080/book/:isbn

Key	Value	Description

Path Variables

Key	Value	Description
isbn	9788825173956	ISBN code of the book to find

200 OK • 29 ms • 325 B

```
{
    "isbn": "9788825173956",
    "title": "Algoritmi e Strutture Dati",
    "author": "Alberto Montresor"
}
```

Node.js

- JavaScript framework to run JavaScript code on servers
- Uses callbacks and ***Promises*** for asynchronous operations
- Very useful to create Web applications and services through further frameworks or internal modules
- Packages to complement framework features (such as the ***pg*** package to integrate *PostgreSQL*) can be downloaded from the ***npm*** package manager

ExpressJS

- Node.js framework to make the creation of Web applications and services easier.
- Uses ***routes*** to represent APIs.

Each route must:

- declare an HTTP operation and a ***route path***;
- implement a callback function to handle the request.

Optionally, routes can also call ***middleware functions*****.**

Guided Examples

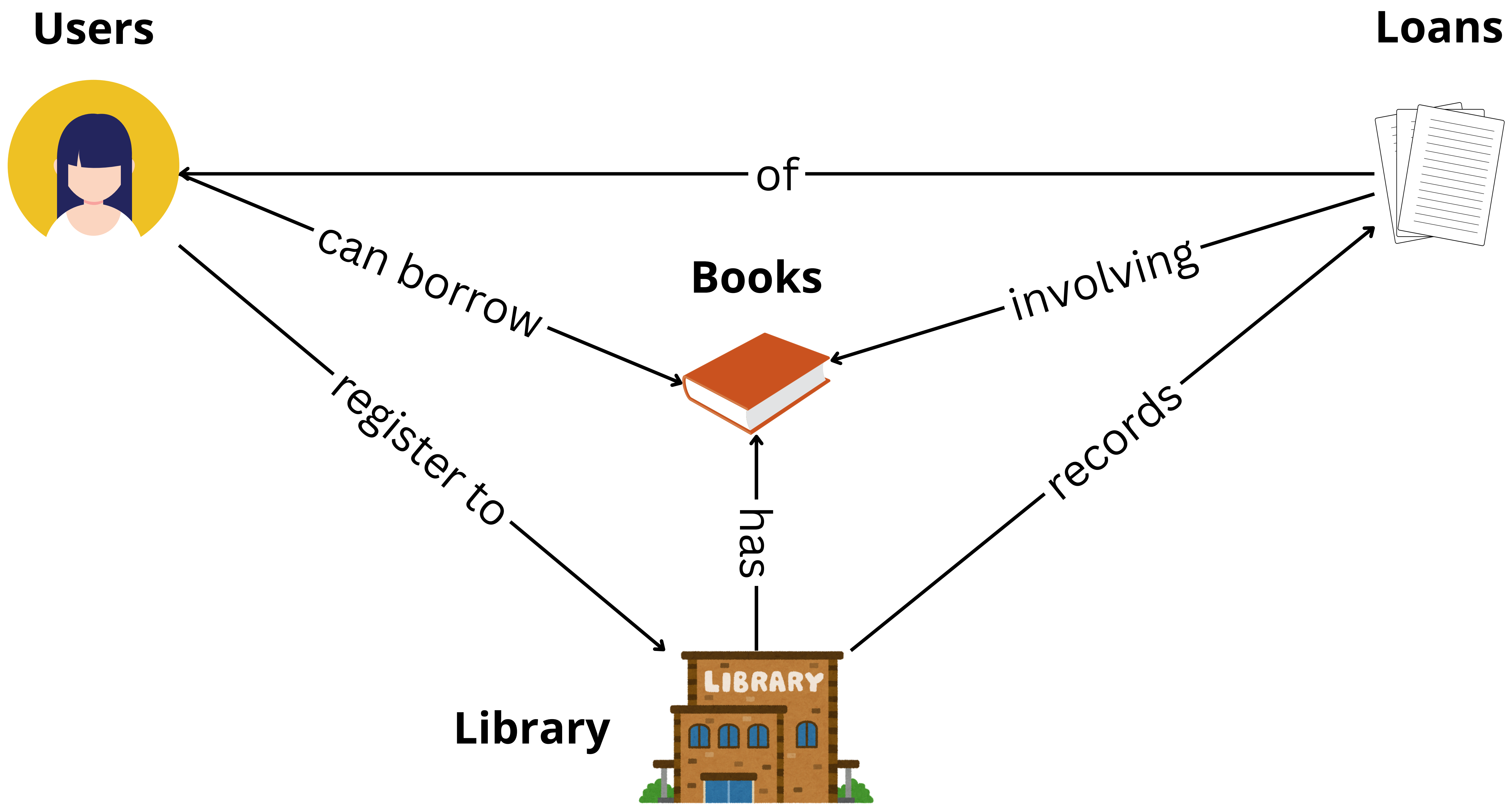
Start the VM

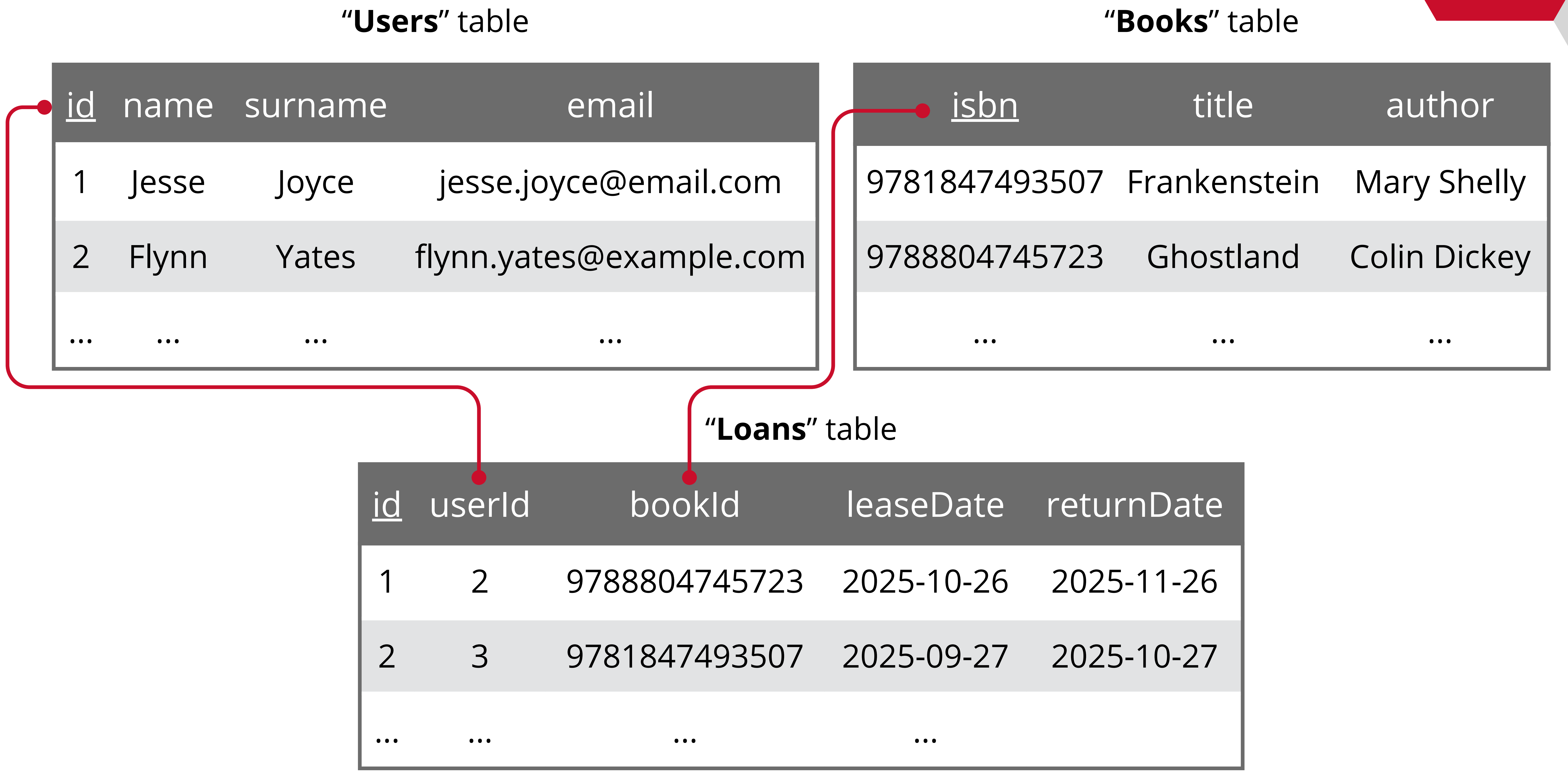
1. Start the Lab 2 VM
2. Log in as either ***user*** (if using the Apple Silicon VM) or ***vboxuser*** (otherwise) with password ***pass***
3. Open the project with an IDE
4. In a terminal run **sudo systemctl restart postgresql.service**
5. Open the workspace in Postman using one of the two options:
   - Open Postman in the web browser and run the Postman Desktop Agent (in ***Desktop/Postman setup/Postman Agent***)
   - Open the Postman Desktop App

Project Structure:

the Database

Library System





"**Users**" table

id	name	surname	email
1	Jesse	Joyce	jesse.joyce@email.com
2	Flynn	Yates	flynn.yates@example.com
…	…	…	…

"**Books**" table

isbn	title	author
9781847493507	Frankenstein	Mary Shelly
9788804745723	Ghostland	Colin Dickey
…	…	…

"**Loans**" table

id	userId	bookId	leaseDate	returnDate
1	2	9788804745723	2025-10-26	2025-11-26
2	3	9781847493507	2025-09-27	2025-10-27
…	…	…	…	

Example 1

Hello endpoint

Set up server

1. Go to ***line 2*** of the ***app.js*** file and import *express*

```
const express = require("express");
```

2. Set up Express application (***line 5***)

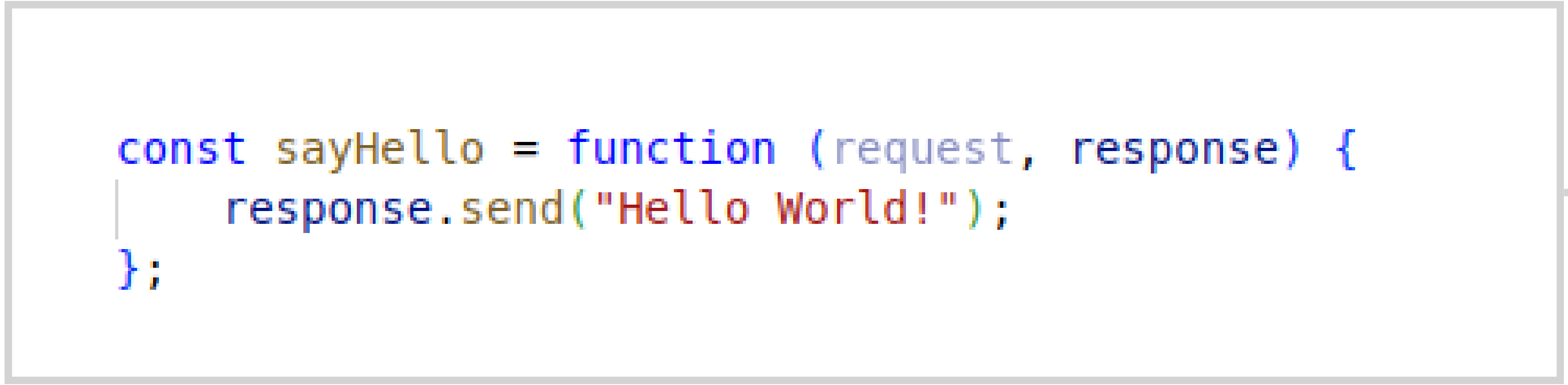
```
const server = express();
```

3. Set up port for server using the ***listen*** function (***line 33***)

```
server.listen(8080, function () {
    console.log("Server listening on port 8080");
});
```

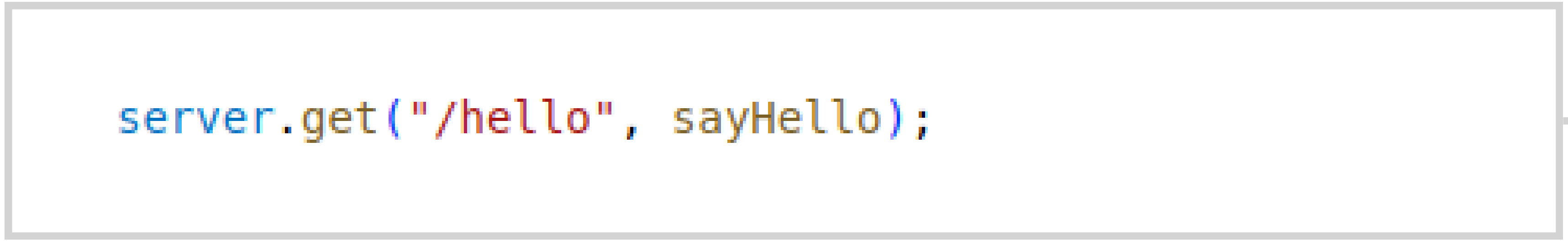
4. Start the server from terminal by going to the root directory of the project and typing ***npm start***

Define route



```
const sayHello = function (request, response) {
    response.send("Hello World!");
};
```

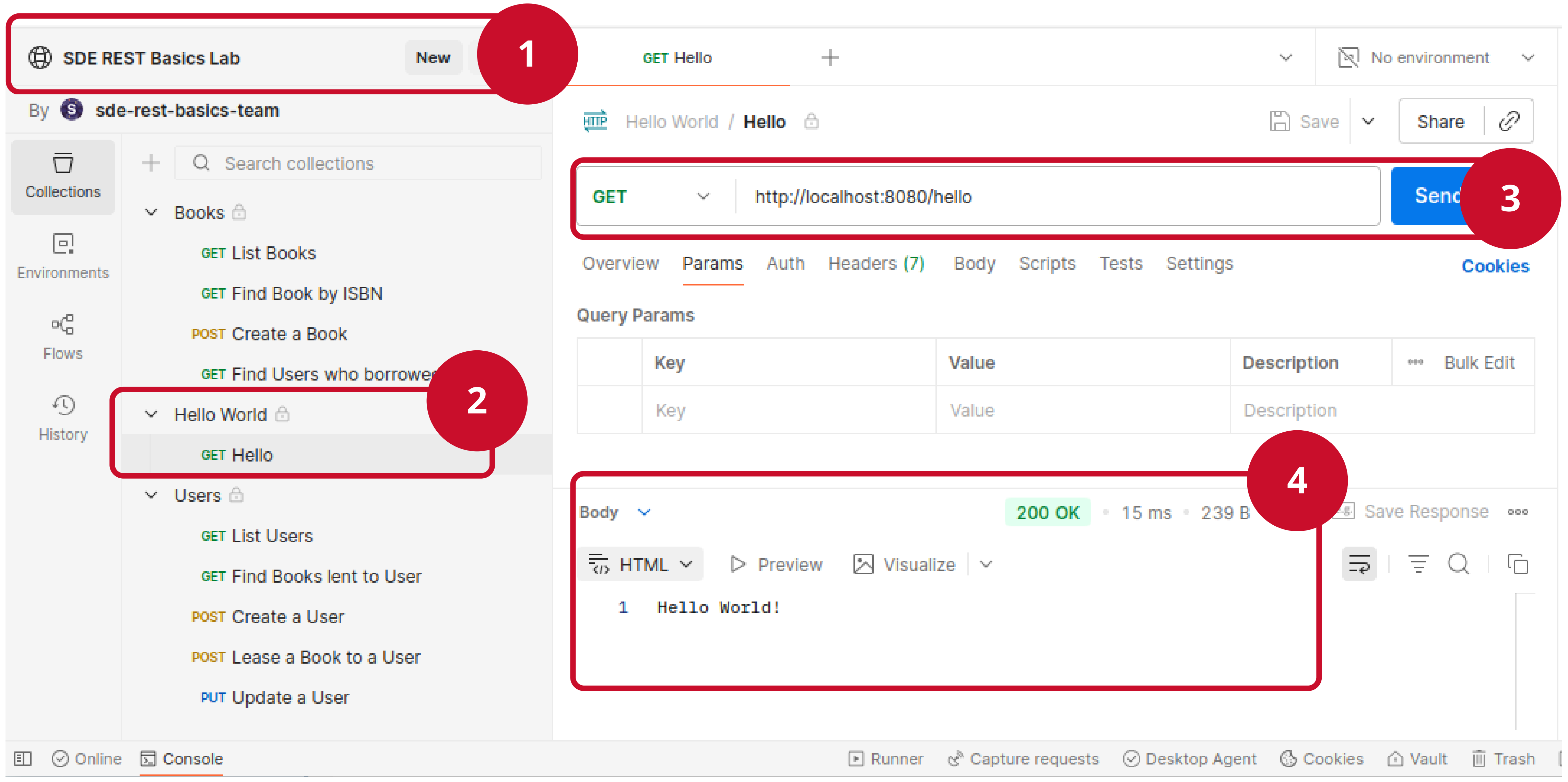
1 In ***app.js: line 12*** define ***sayHello*** route handler



```
server.get("/hello", sayHello);
```

2 Define the ***/hello*** endpoint for the ***sayHello*** function (***line 16***)

Testing with Postman



```
Hello World!
```

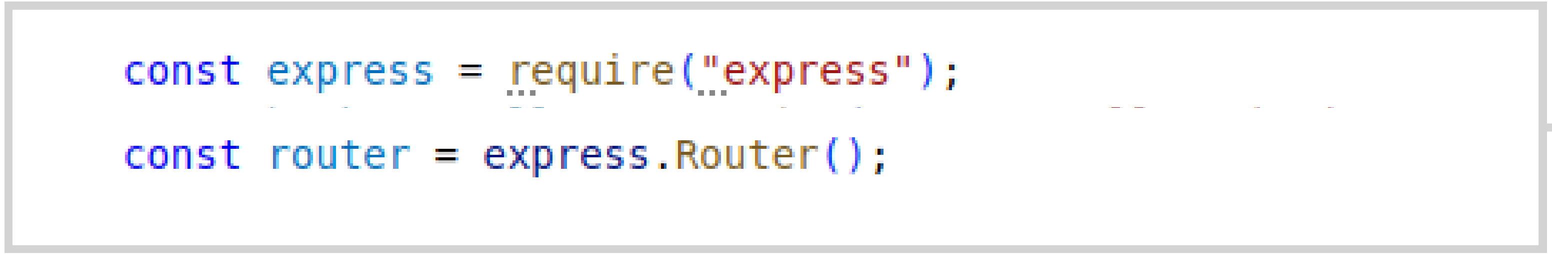
1. Make sure that the **workspace** to test this project is open
2. Select ***Hello*** request under the ***Hello World*** collection
3. Press ***Send***
4. Check server response

Example 2

getBookByIsbn endpoint

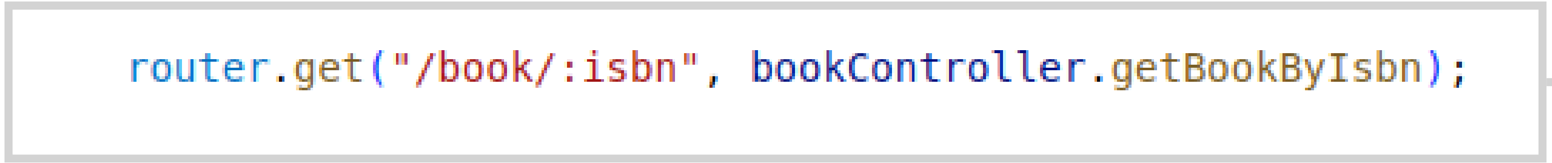
Book route

In ***routes/bookRoutes.js*** look at how the route path for the function ***bookController.getBookByIsbn*** is defined:



```
const express = require("express");
const router = express.Router();
```

1 ***router*** object setup (***lines 2 and 9***)



```
router.get("/book/:isbn", bookController.getBookByIsbn);
```

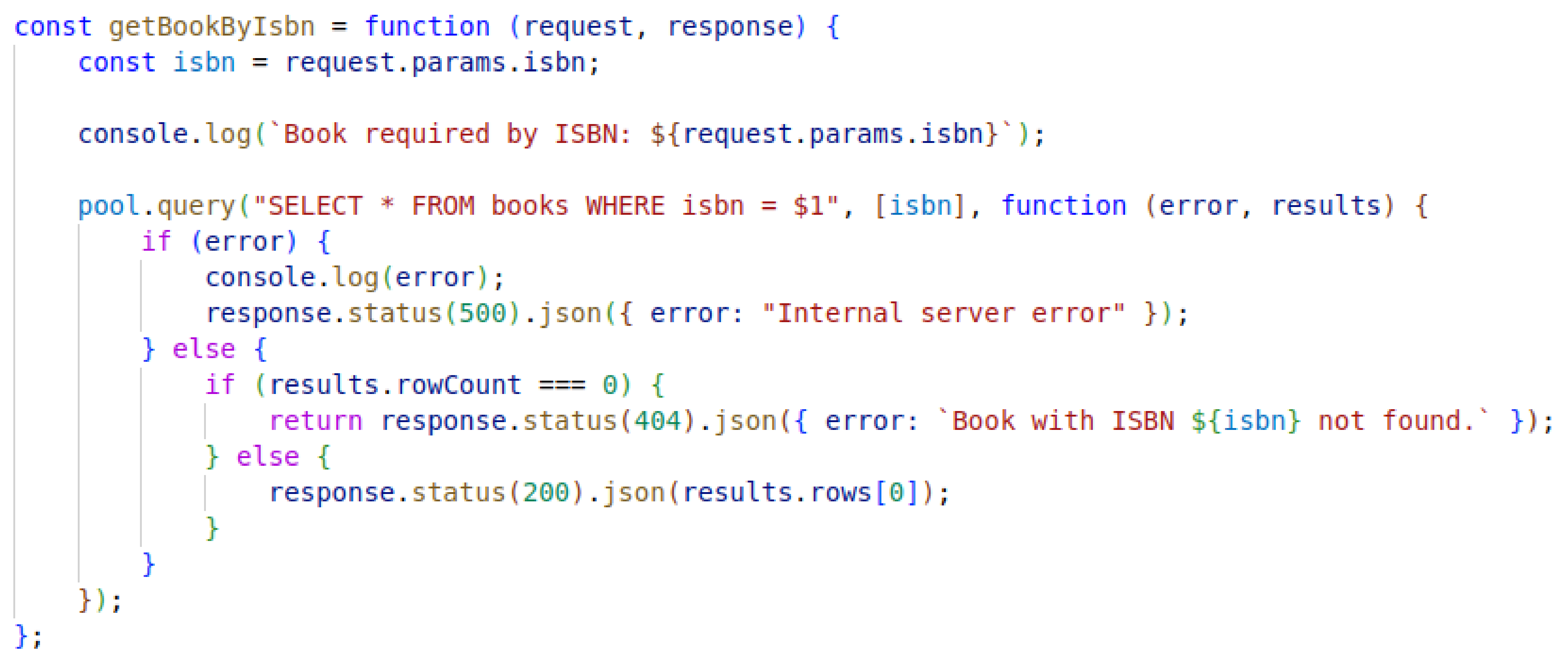
2 ***router.get()*** defines the HTTP request method it responds to and ***/book/:isbn*** defines the route path through which ***getBookByIsbn*** can be called (***line 13***)

To get the data of book with ISBN 9788825173956 we will send a GET request to:

http://localhost:8080/book/9788825173956

Book controller

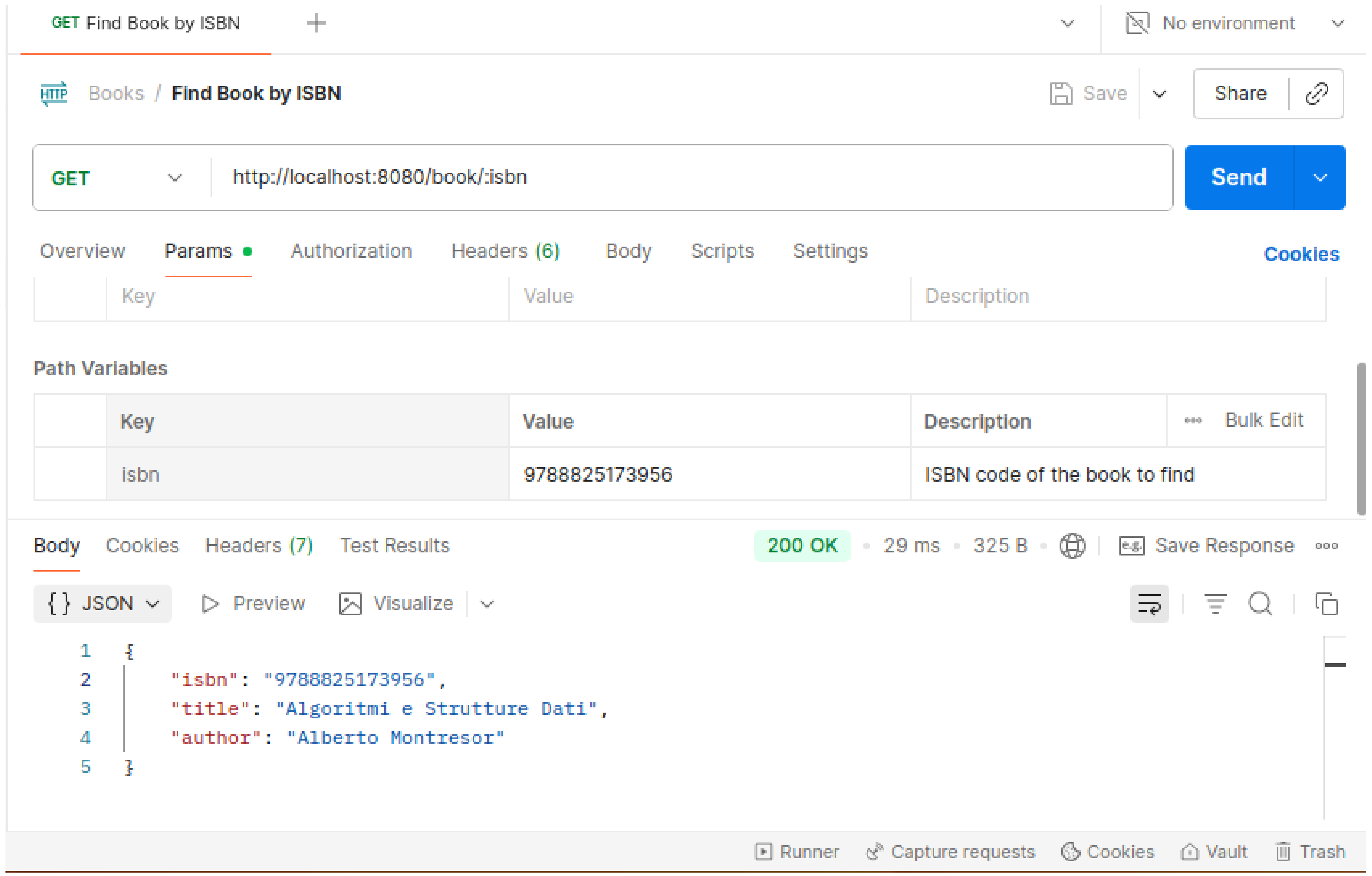
In ***controllers/bookController.js*** look at how the the function ***getBookByIsbn*** is defined (***line 33***)



```
const getBookByIsbn = function (request, response) {
    const isbn = request.params.isbn;

    console.log(`Book required by ISBN: ${request.params.isbn}`);

    pool.query("SELECT * FROM books WHERE isbn = $1", [isbn], function (error, results) {
        if (error) {
            console.log(error);
            response.status(500).json({ error: "Internal server error" });
        } else {
            if (results.rowCount === 0) {
                return response.status(404).json({ error: `Book with ISBN ${isbn} not found.` });
            } else {
                response.status(200).json(results.rows[0]);
            }
        }
    });
};
```

1. get request parameter
2. send SQL query to database
3. handle errors
4. if response is empty, send **404 Not Found** response to client
5. if not empty, send **200 OK** response to client

Postman testing



GET http://localhost:8080/book/:isbn

Key	Value	Description

Path Variables

Key	Value	Description
isbn	9788825173956	ISBN code of the book to find

200 OK · 29 ms · 325 B

```
{
    "isbn": "9788825173956",
    "title": "Algoritmi e Strutture Dati",
    "author": "Alberto Montresor"
}
```

1. Open Postman and select the ***Find Book by ISBN*** query under the ***Books*** collection
2. Fill the ***isbn*** field with value “**9788825173956**”
3. Send the request
4. Check the response

API security: input validation

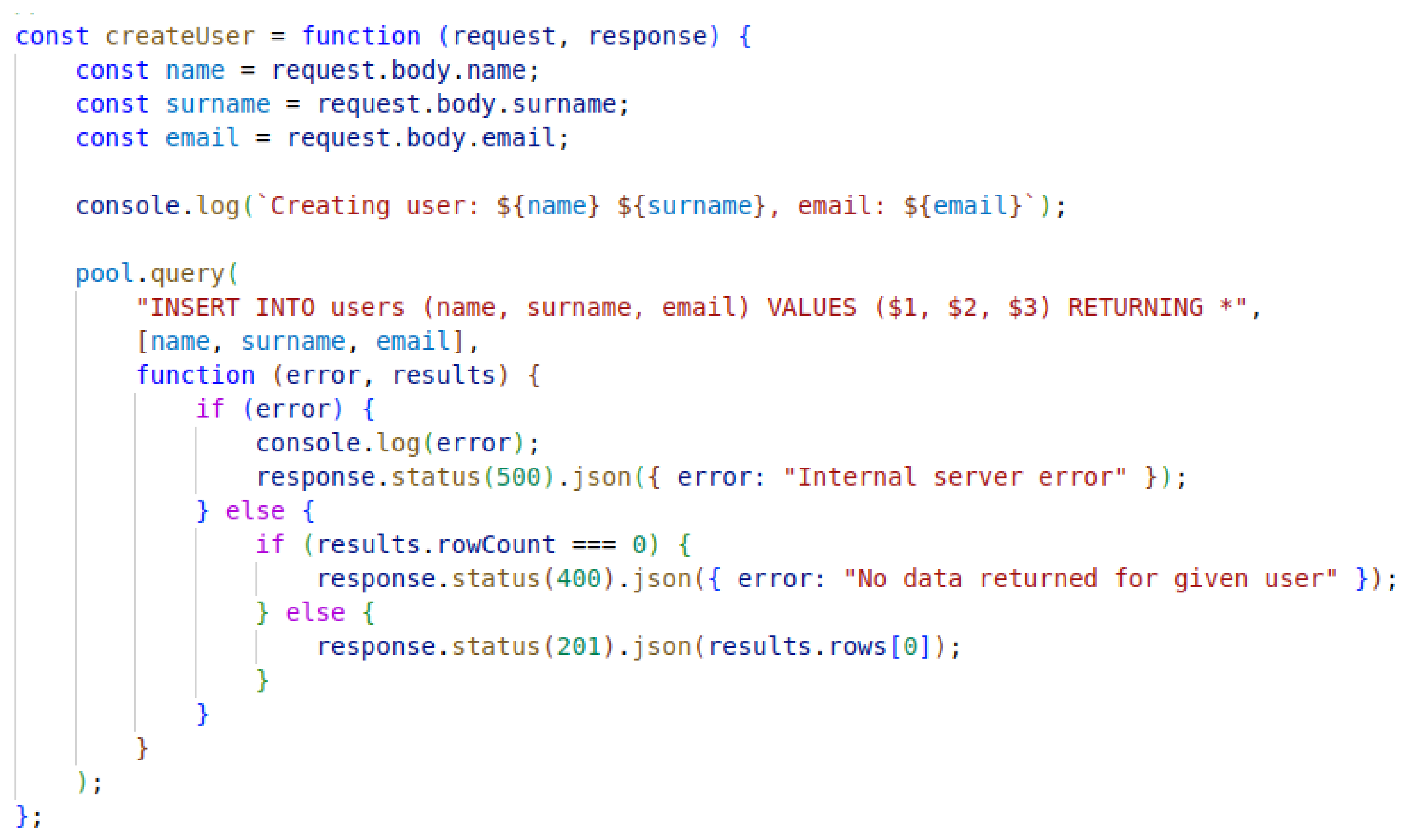
- Crucial to security of Web applications and services
- Ensures inputs are properly structured and constrained within validity limits
- Helps against a variety of attacks (*SQL Injection*, *XSS*, etc.)
- NPM already provides packages to achieve this (*node-input-validator*, *express-validator*)
- We will use ***node-input-validator*** (documentation link)

Example 3

createUser endpoint

Endpoint

In ***controllers/userController.js*** look at how the the function ***createUser*** is defined (***line 50***).
NB: *To correctly parse json responses we put in the **app.js** file **server.use(express.json())***



```
const createUser = function (request, response) {
    const name = request.body.name;
    const surname = request.body.surname;
    const email = request.body.email;

    console.log(`Creating user: ${name} ${surname}, email: ${email}`);

    pool.query(
        "INSERT INTO users (name, surname, email) VALUES ($1, $2, $3) RETURNING *",
        [name, surname, email],
        function (error, results) {
            if (error) {
                console.log(error);
                response.status(500).json({ error: "Internal server error" });
            } else {
                if (results.rowCount === 0) {
                    response.status(400).json({ error: "No data returned for given user" });
                } else {
                    response.status(201).json(results.rows[0]);
                }
            }
        }
    );
};
```

1 get **POST** request parameter

2 if the request was successful, send **201 Created** response to client

Input validation

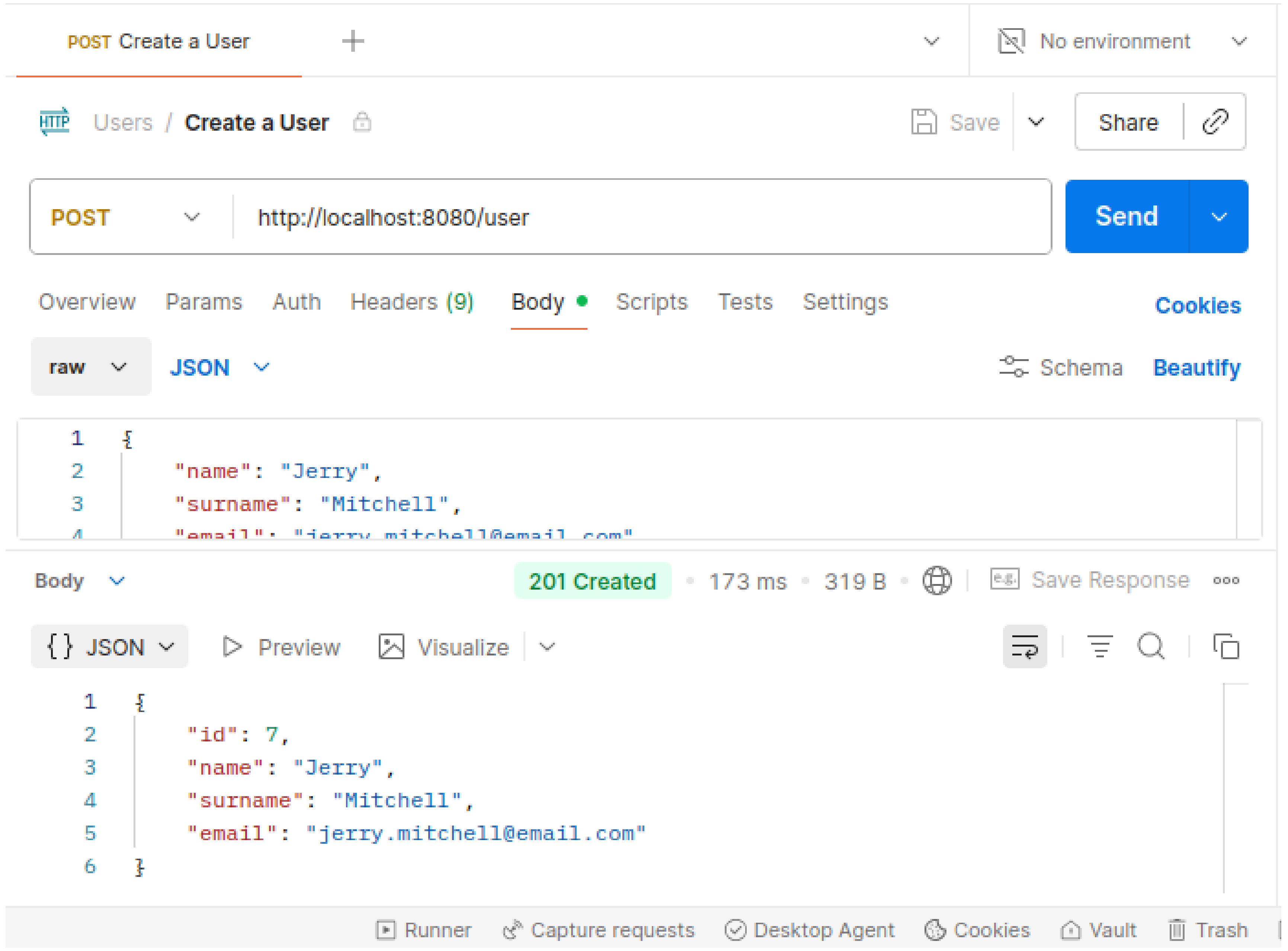
1 In ***routes/userRoutes.js*** import the ***validate*** middleware from the ***middleware/validation.js*** file (***line 6***)

```
const validate = require("../middleware/validation");
```

2 Call it from the ***/user*** route handler (***line 18***) as follows

```
router.post(
    "/user",
    validate({
        name: "required|string|minLength:1",
        surname: "required|string|minLength:1",
        email: "required|email",
    }),
    userController.createUser
);
```

Postman testing



POST http://localhost:8080/user

```
{
    "name": "Jerry",
    "surname": "Mitchell",
    "email": "jerry.mitchell@email.com"
```

201 Created • 173 ms • 319 B

```
{
    "id": 7,
    "name": "Jerry",
    "surname": "Mitchell",
    "email": "jerry.mitchell@email.com"
}
```

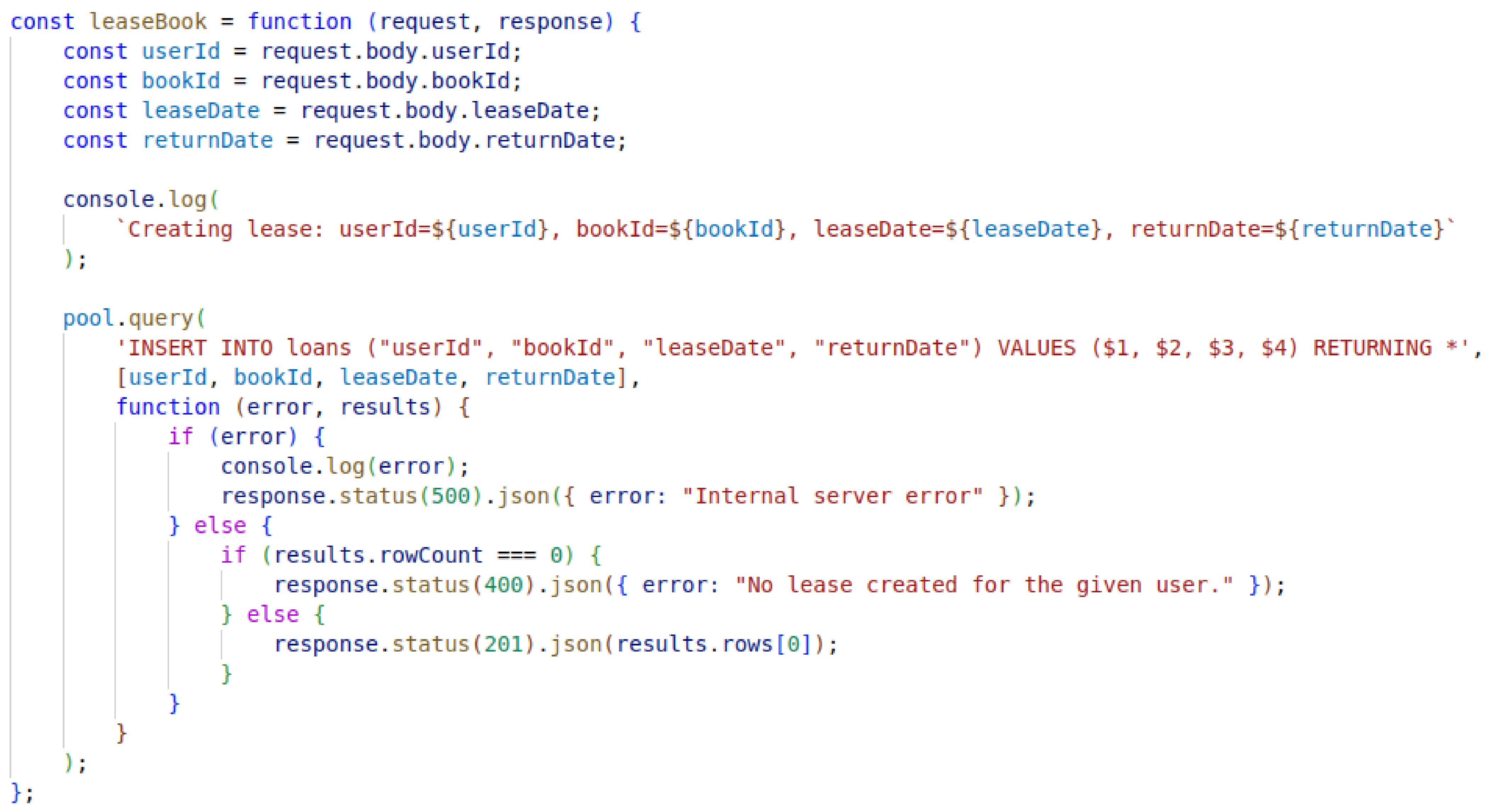
1. Open Postman and select the ***Create a User*** query under the ***Users*** collection
2. Fill the ***name***, ***surname*** and ***email*** field
3. Send the request
4. Check the response

Example 4

leaseBook endpoint

Endpoint

In ***controllers/userController.js*** look at how the the function ***leaseBook*** is defined (***line 81***) :



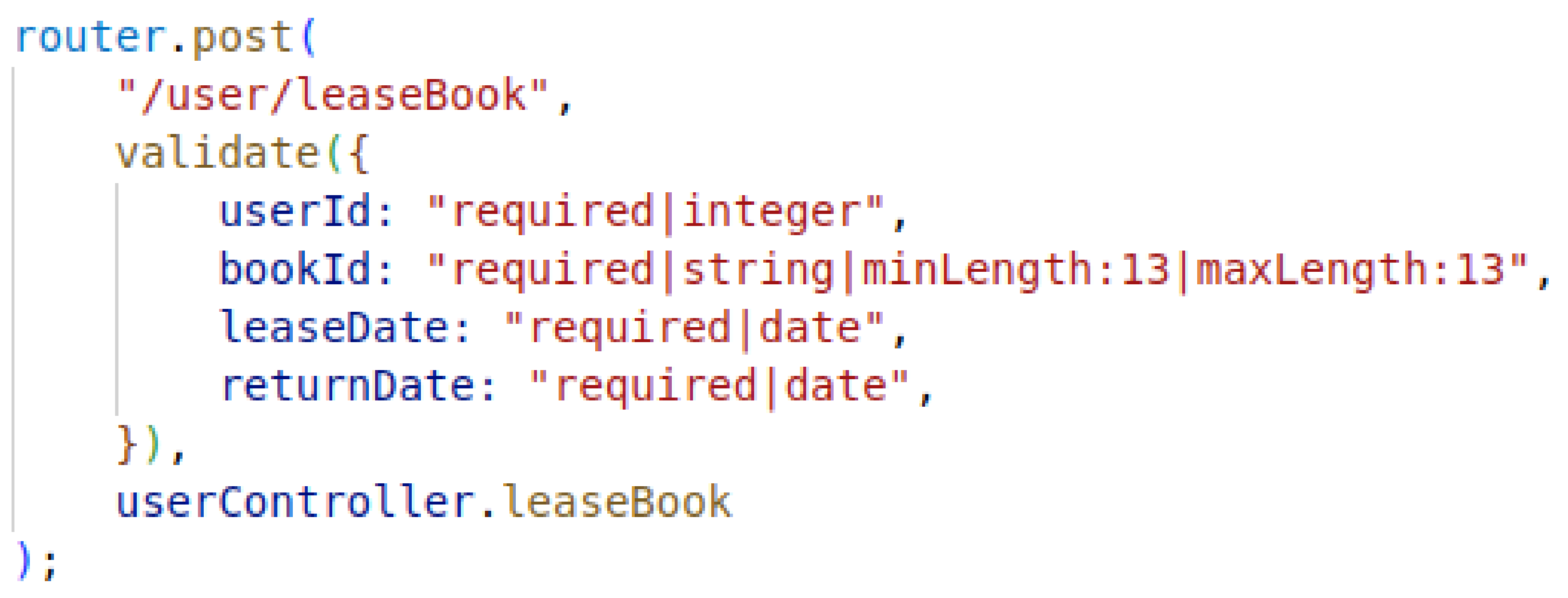
```
const leaseBook = function (request, response) {
    const userId = request.body.userId;
    const bookId = request.body.bookId;
    const leaseDate = request.body.leaseDate;
    const returnDate = request.body.returnDate;

    console.log(
        `Creating lease: userId=${userId}, bookId=${bookId}, leaseDate=${leaseDate}, returnDate=${returnDate}`
    );

    pool.query(
        'INSERT INTO loans ("userId", "bookId", "leaseDate", "returnDate") VALUES ($1, $2, $3, $4) RETURNING *',
        [userId, bookId, leaseDate, returnDate],
        function (error, results) {
            if (error) {
                console.log(error);
                response.status(500).json({ error: "Internal server error" });
            } else {
                if (results.rowCount === 0) {
                    response.status(400).json({ error: "No lease created for the given user." });
                } else {
                    response.status(201).json(results.rows[0]);
                }
            }
        }
    );
};
```

1. get **POST** request parameter
2. if the request is not well-formed, send **400 Bad Request** response to client
3. if the request is successful, send **201 Created** response to client

Input validation

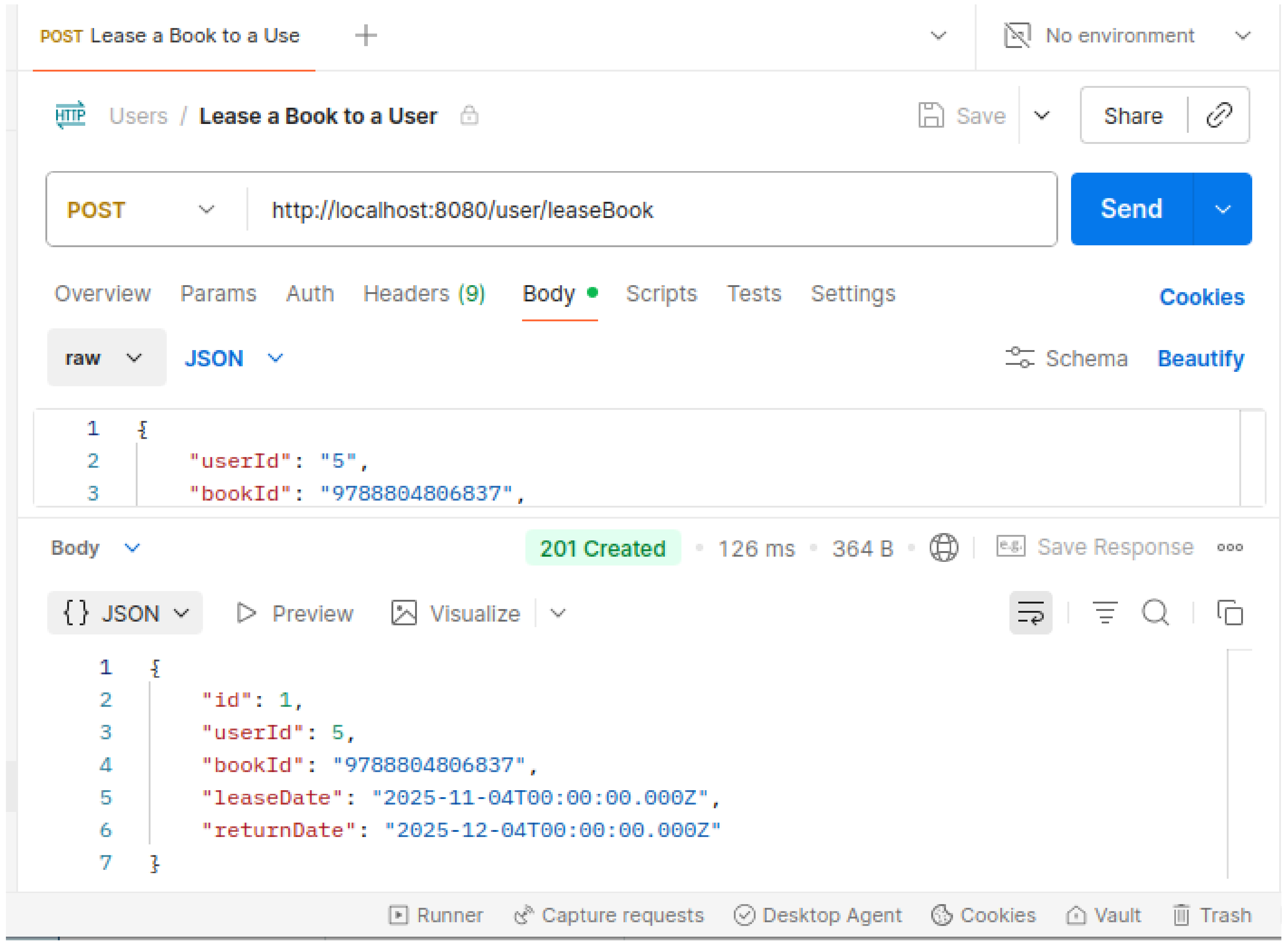
1. Import the ***validate*** middleware as before

2. Call it from the ***/user/leaseBook*** endpoint as follows (***routes/userRoutes.js*** at ***line 28***):



```
router.post(
    "/user/leaseBook",
    validate({
        userId: "required|integer",
        bookId: "required|string|minLength:13|maxLength:13",
        leaseDate: "required|date",
        returnDate: "required|date",
    }),
    userController.leaseBook
);
```

Postman testing



http://localhost:8080/user/leaseBook

```
{
    "userId": "5",
    "bookId": "9788804806837",
```

201 Created • 126 ms • 364 B

```
{
    "id": 1,
    "userId": 5,
    "bookId": "9788804806837",
    "leaseDate": "2025-11-04T00:00:00.000Z",
    "returnDate": "2025-12-04T00:00:00.000Z"
}
```

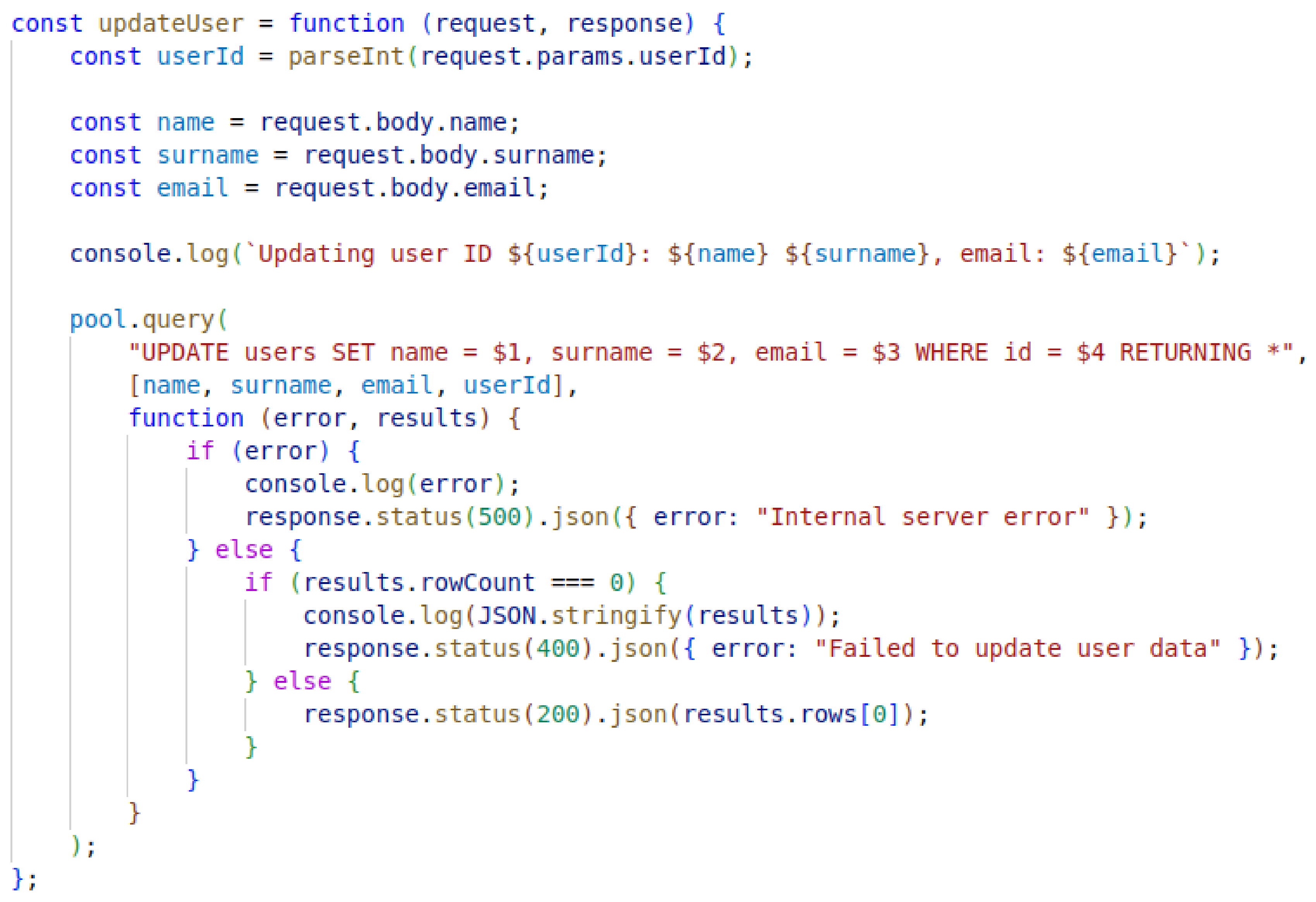
1. Open Postman and select the ***Lease a Book to a User*** query under the ***Users*** collection
2. Look at the field values
3. Send the request
4. Check the response

Example 5

updateUser endpoint

Endpoint

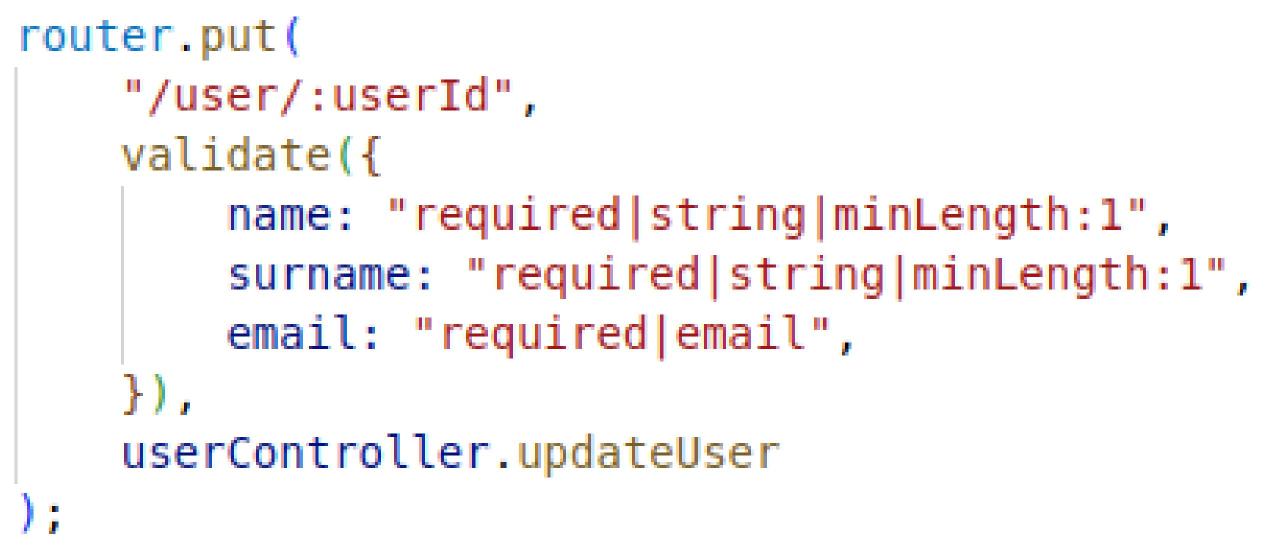
In ***controllers/userControllers.js*** starting at ***line 115***:



```
const updateUser = function (request, response) {
    const userId = parseInt(request.params.userId);

    const name = request.body.name;
    const surname = request.body.surname;
    const email = request.body.email;

    console.log(`Updating user ID ${userId}: ${name} ${surname}, email: ${email}`);

    pool.query(
        "UPDATE users SET name = $1, surname = $2, email = $3 WHERE id = $4 RETURNING *",
        [name, surname, email, userId],
        function (error, results) {
            if (error) {
                console.log(error);
                response.status(500).json({ error: "Internal server error" });
            } else {
                if (results.rowCount === 0) {
                    console.log(JSON.stringify(results));
                    response.status(400).json({ error: "Failed to update user data" });
                } else {
                    response.status(200).json(results.rows[0]);
                }
            }
        }
    );
};
```

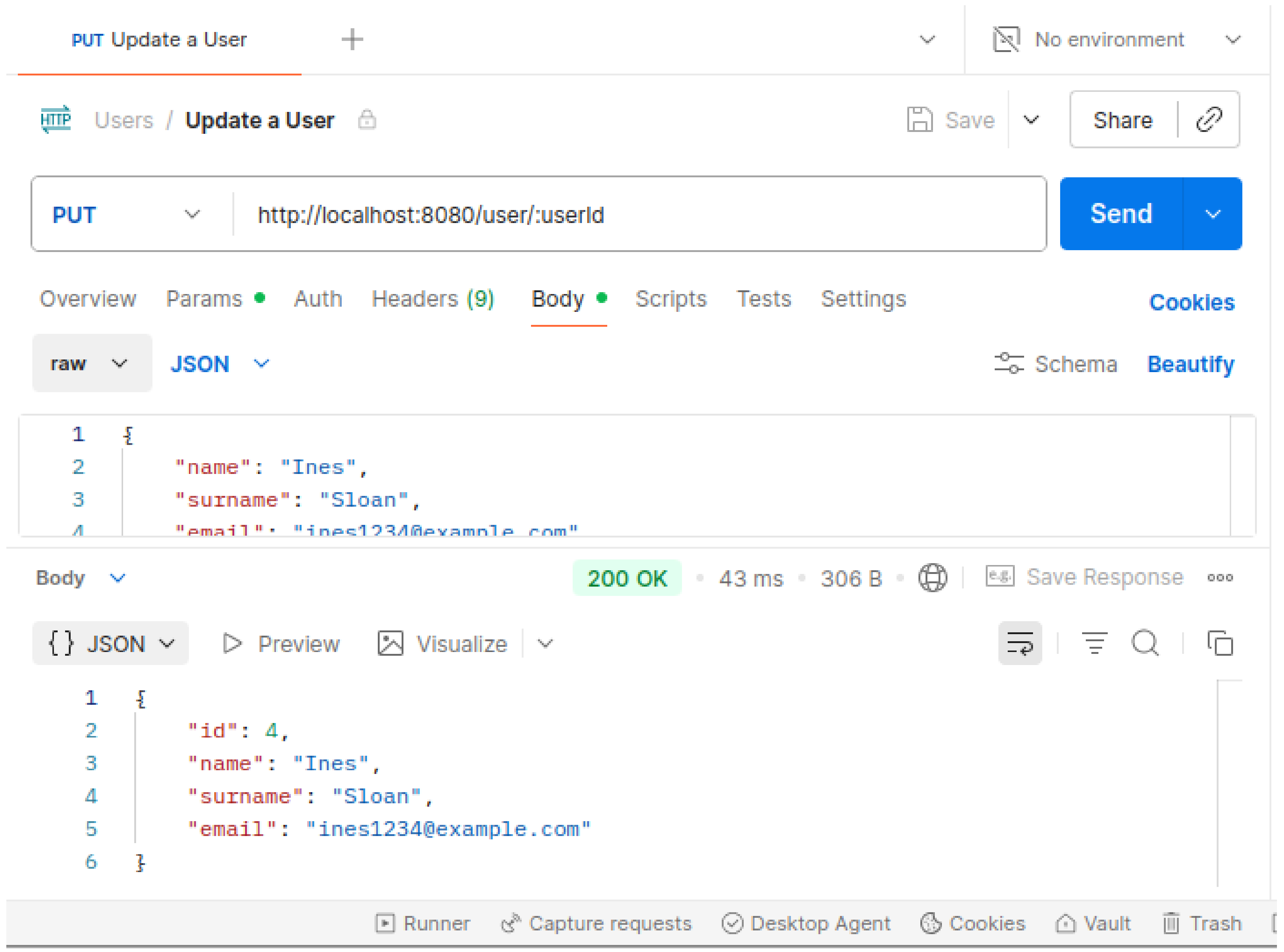
Input validation

1. Import the ***validate*** middleware as before
2. Call it from the ***/user/:userId*** endpoint as follows (***routes/userRoutes.js*** at ***line 39***):



```
router.put(
    "/user/:userId",
    validate({
        name: "required|string|minLength:1",
        surname: "required|string|minLength:1",
        email: "required|email",
    }),
    userController.updateUser
);
```

Postman testing


PUT Update a User
No environment
Users / Update a User
Save
Share
PUT http://localhost:8080/user/:userId
Send
Overview Params Auth Headers (9) Body Scripts Tests Settings Cookies
raw JSON Schema Beautify

```
{
    "name": "Ines",
    "surname": "Sloan",
    "email": "ines1234@example.com"
```

Body 200 OK 43 ms 306 B Save Response
{} JSON Preview Visualize

```
{
    "id": 4,
    "name": "Ines",
    "surname": "Sloan",
    "email": "ines1234@example.com"
}
```

Runner Capture requests Desktop Agent Cookies Vault Trash

1. Open Postman and select the ***Update a User*** query under the ***Users*** collection
2. Set the userId in the Params tab
3. Set the new information about the user in the Body tab
4. Send the request
5. Check the response

SNACK BREAK

Exercises

Before we begin…

Instructions and hints

- You will find ***TODO*** comments in the project to guide you step by step through the exercises
- If you are struggling to complete an exercise, you can look at the **hints** on the slides

HINT: *This is a hint!*

- Use Postman to test the endpoints
- If all fails … don’t hesitate to **ask for help!**

Exercise 1

Write an endpoint to get the list of all users in the database

1. Go to ***userRoutes.js: line 14*** and add a ***/user/list*** route which accepts **GET** requests and handles them with the ***userController.getUsers*** function

??? ***HINT:*** *Look at line 12 of* ***bookRoutes.js***

2. Go to ***userControllers.js: line 14*** and add the instruction to send a **500 Internal Server Error** response with a JSON error message (if an error occurs)

3. Remaining in ***userControllers.js: line 18***, add the instruction to send a **200 OK** response with the list of users as JSON data

??? ***HINT:*** *Look at lines 20-26 of* ***bookControllers.js***

4. Test your endpoint with Postman using the ***List Users*** query in the ***Users*** collection.

Exercise 2

Write an endpoint to get the list of all books lent to a user

1. Go to ***userControllers.js: line 28*** and add a variable ***userId*** that stores the request parameter

2. Add a **200 OK** response with the list of books as JSON data at ***line 42***

3. Test your endpoint with Postman using the ***Find books lent to User*** query in the ***Users*** collection.

*??? **HINT:** Take a look at **Example 2** in **bookControllers.js**: lines 33-54*

Exercise 3

Write an endpoint to add new books to the database

1. Go to ***bookRoutes.js: line 19*** and add a ***/book*** route which accepts **POST** requests and handles them with the ***bookController.createBook*** function

2. Go to ***bookControllers.js: line 61*** and extract ***isbn***, ***title*** and ***author*** from the request body

3. At ***line 79***, add a **201 Created** response with the created book as JSON data

4. Test your endpoint with Postman using the ***Create Book*** query in the ***Books*** collection.

HINT: *Take a look at* ***Example 3*** *in* ***userControllers.js****: lines 50-77*

Final assignment

Write an endpoint that returns the list of users who took a specific book from the library:

1. The route path is ***/book/:isbn/borrowers*** in ***bookRoutes.js***. The route accepts **GET** requests
2. Go to ***bookControllers.js: line 91*** and extract the ***isbn*** from the request
3. At ***line 106***, add the instruction to send a **200 OK** response with the list of users as JSON data
4. Test your endpoint in Postman with the ***Find Users who borrowed a Book*** query in the ***Books*** collection

HINT : *Take a look at* ***Exercise 2*** *in* ***userControllers.js****: lines 25-46*

Thank you for your attention

Marco Lasagna

Denise Comincioli

Mehrab Fajar

Lewis Ndambiri

Luca Dematté