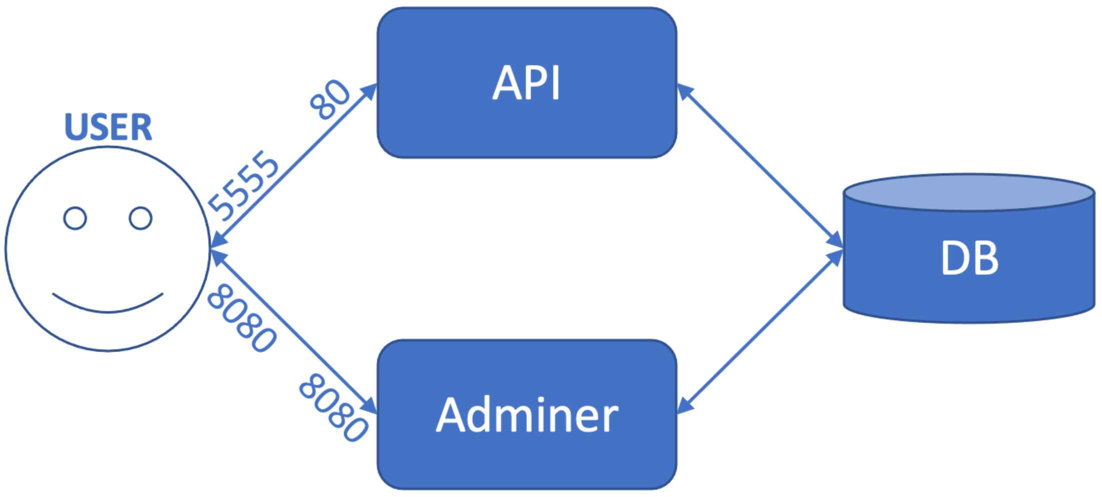
Task Requirement

00. Getting started

For the tasks in this lab, you will be using the files found in this archive, which contains a NodeJS application that creates an API for adding books in a library over a PostgreSQL database. The architecture of the application is shown in the image below. It is composed of a backend (or API), a database, and an administration component for the database.



In order to solve the tasks, you need to have Docker installed, or you need to use Play with Docker.

This is the overall requirement.

1. write the entire configuration described below in a Docker Compose file called ***docker-compose.yml***
2. deploy the configuration from the ***docker-compose.yml*** file using ***docker compose -f docker-compose.yml up -d***
3. use Postman and Adminer to verify that everything works just as before
4. stop all the containers using ***docker compose -f docker-compose.yml down***
5. remove the remaining volume

01. Compose a multi-container application with one single **docker-compose** file

Subtasks

1. create a bridge network called ***lab4-db-network***
2. create another bridge network called ***lab4-adminer-network***
3. create a volume named ***lab4-db-volume***
4. from the parent folder of ***backend***, launch a container ***in the background*** for a database with the following features:
    a. a bind mount will be attached that will map between the ***"$(pwd)"/database/init-db.sql*** file on the local machine (this will be the source to the bind mount flag and can be found in the lab archive) and the ***/docker-entrypoint-initdb.d/init-db.sql*** file in the container to be run (this will be the destination)
    b. attach the previously-created ***lab4-db-volume*** volume (as source) to the ***/var/lib/postgresql/data*** path in the running container (as destination)
    c. the container will be part the previously-created ***lab4-db-network*** network
    d. the following environment variables will be specified (in a ***docker run*** command, this is done as follows: ***docker run -e NAME=value***):
        I. variable ***POSTGRES_USER*** with value ***admin***
        II. variable ***POSTGRES_PASSWORD*** with value ***admin***
        III. variable ***POSTGRES_DB*** with value ***books***
    e. the container will be called ***lab4-db***
    f. the container will run the ***postgres*** image from the official register
5. add the ***lab4-db*** container to the ***lab4-adminer-network*** network
6. launch a container ***in the background*** for a database admin with the following features:
    a. the container will expose port 8080 and map it to 8080 on the local machine
    b. the container will be called ***lab4-adminer***
    c. the container will be part the previously-created ***lab4-adminer-network*** network
    d. the container will run the ***adminer*** image from the official register
7. start a container ***in the background*** based on the previously-created ***lab4-api-image*** image, with the following features:
    a. the container will be part of the previously-created ***lab4-db-network*** network

b. the following environment variables will be specified:
   I. variable ***PGUSER*** with value ***admin***
   II. variable ***PGPASSWORD*** with value ***admin***
   III. variable ***PGDATABASE*** with the value ***books***
   IV. variable ***PGHOST*** with value ***lab4-db***
   V. variable ***PGPORT*** with value ***5432***

c. the container will be called ***lab4-api***

d. the container will expose port 80 and map it to port 5555 on the local machine

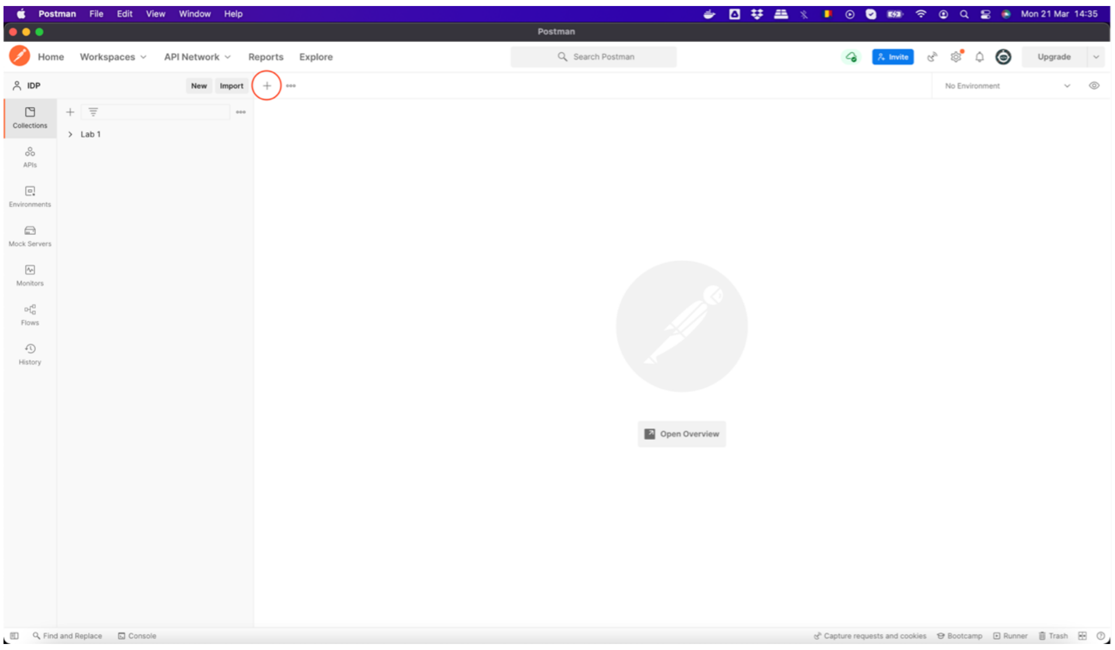
02. Testing a web application using Postman

The application started earlier is a web application that does not have a graphical interface (i.e., a frontend). For this reason, it can only be tested by sending HTTP requests using a tool such as Postman. We will now show how to use Postman to connect to a web application backend.
In the Postman application, we can create the two types of requests that we want to test:

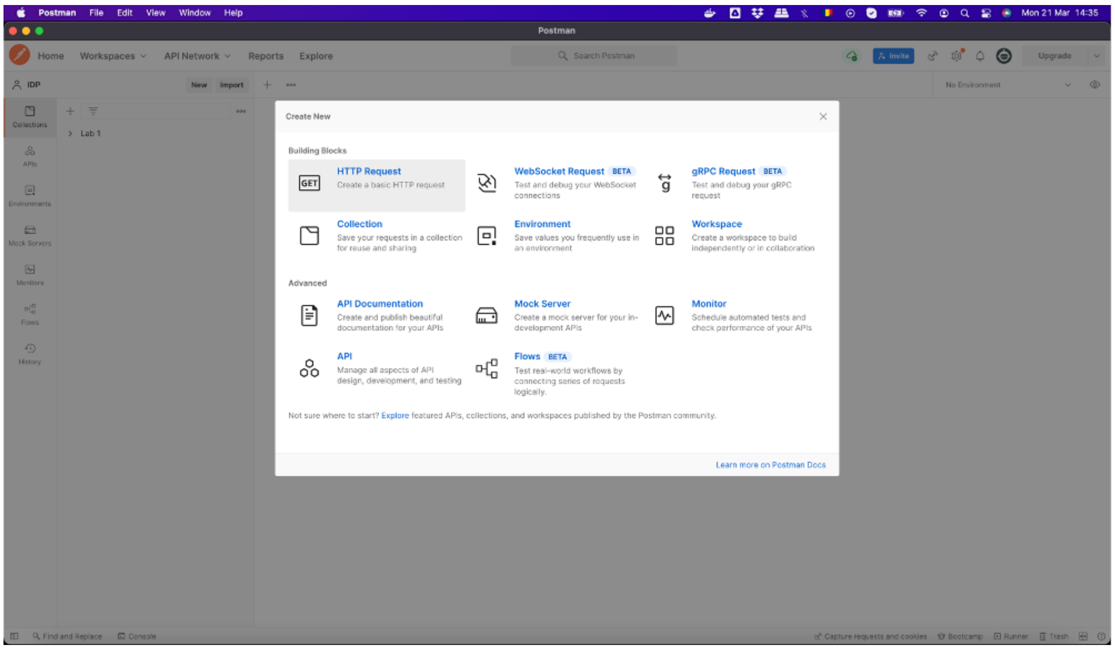
GET - requests information from the backend (in our case, the list of books in the library)

POST - sends information to the backend (in our case, adds a book to the library).

The steps to accomplish this are outlined below. Firstly, once we have opened the Postman application, we press the **+** button, as can be seen in the image below (the button is circled in red).



Next, we select the ***HTTP Request*** option, as shown below.


Create New

Building Blocks

HTTP Request
Create a basic HTTP request

WebSocket Request BETA
Test and debug your WebSocket connections

gRPC Request BETA
Test and debug your gRPC request

Collection
Save your requests in a collection for reuse and sharing

Environment
Save values you frequently use in an environment

Workspace
Create a workspace to build independently or in collaboration

Advanced

API Documentation
Create and publish beautiful documentation for your APIs

Mock Server
Create a mock server for your in-development APIs

Monitor
Schedule automated tests and check performance of your APIs

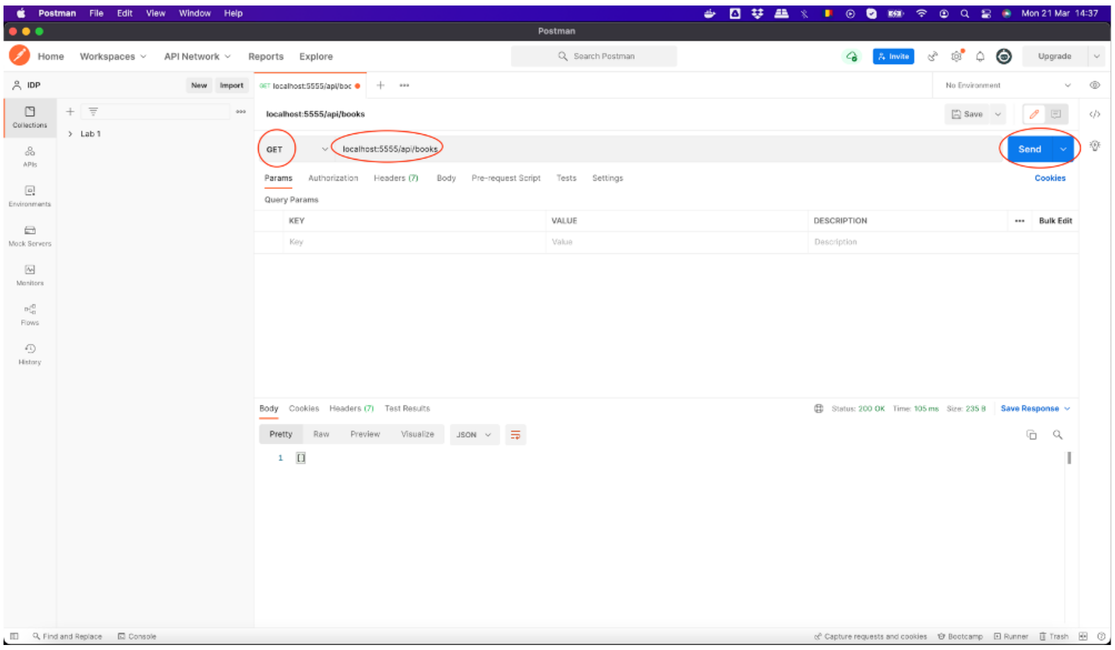
API
Manage all aspects of API design, development, and testing

Flows BETA
Test real-world workflows by connecting series of requests logically.

Not sure where to start? Explore featured APIs, collections, and workspaces published by the Postman community.

Learn more on Postman Docs

We add the first request, whose type should be ***GET***. In the address bar, we write ***localhost:5555/api/books/*** and then we press the ***Send*** button (shown in red in the image below).



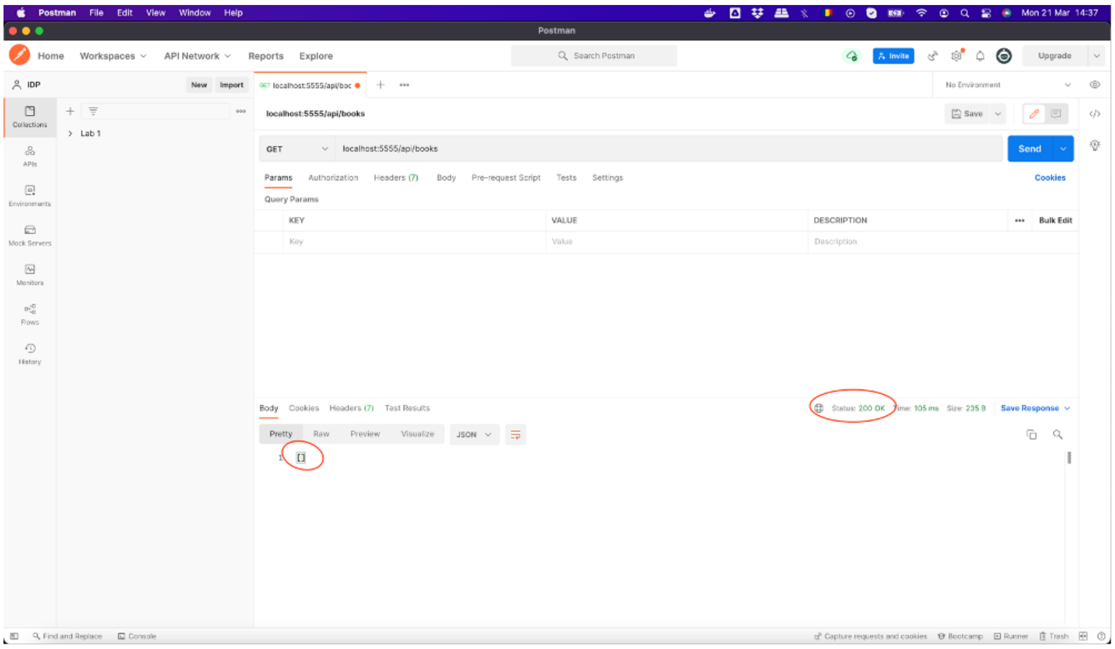
localhost:5555/api/books

GET localhost:5555/api/books

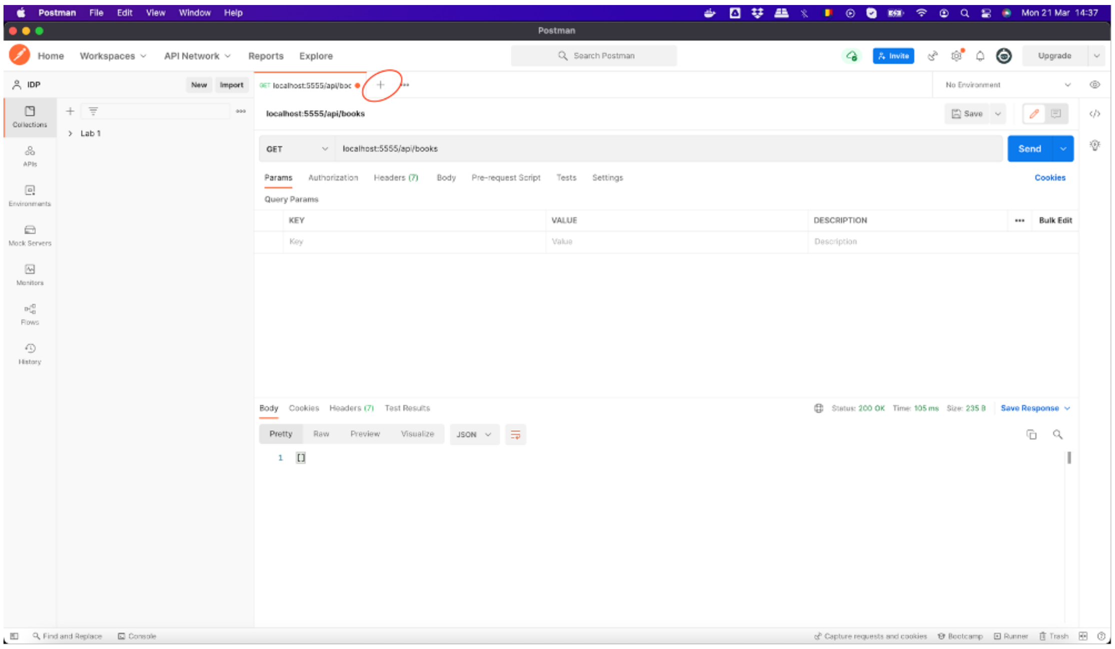
Status: 200 OK Time: 105 ms Size: 235 B

```
[]
```

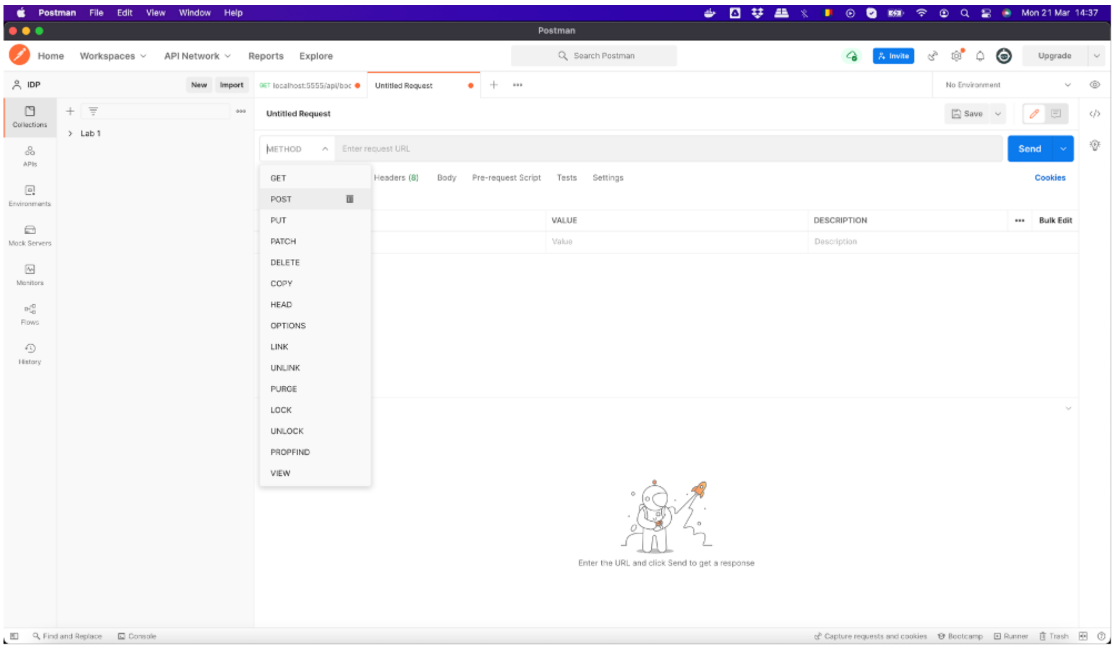
Once the request is sent and everything is in order, we should receive a reply with the status ***200 OK*** (circled in the image below), which contains an empty string (because there are currently no books in the library), as shown below.



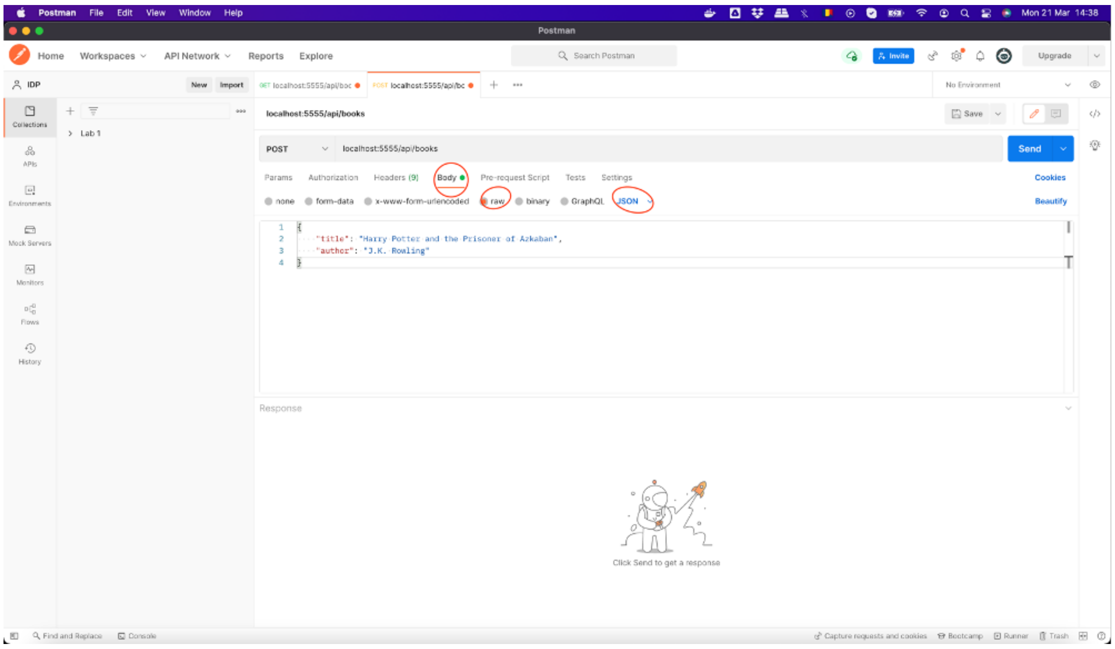
Next, we can create a ***POST*** request. We start by pressing the circled **+** button in the image below.



Next, we set the request type to ***POST*** and then put the same address as before.

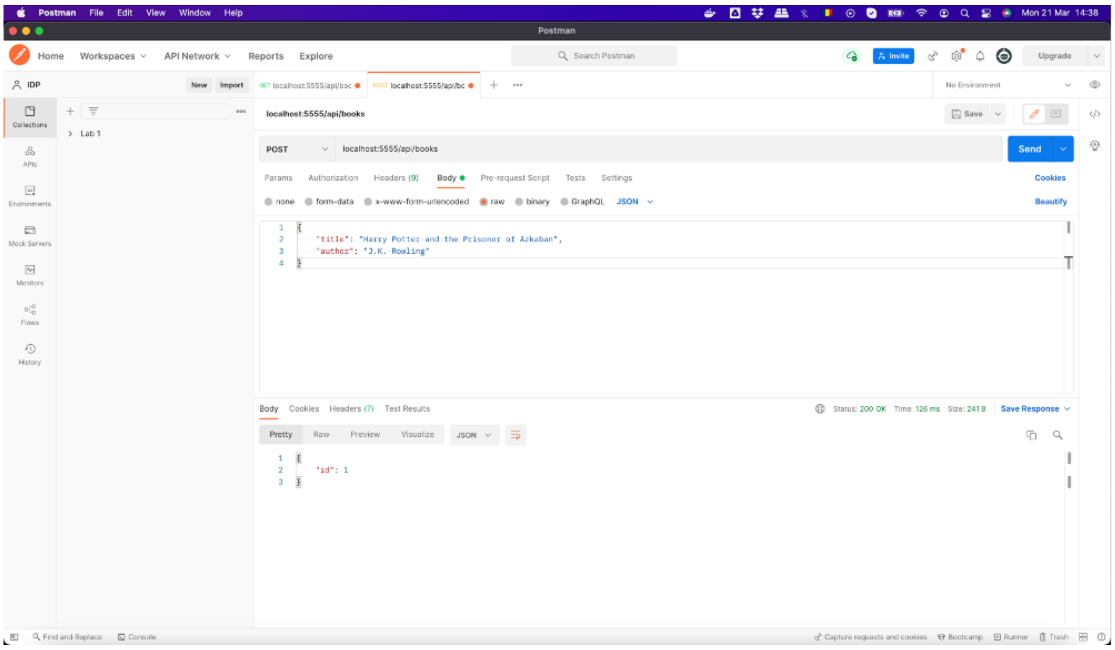


Because we are now adding a book, we need to place the book information in the ***POST*** request we are making. This is done by selecting the ***Body*** component of the request (circled in red in the image below), marking the data as ***Raw*** and ***JSON*** (also in red in the image) and adding the book information in JSON format (see below an example for "Harry Potter and the Prisoner of Azkaban"):



```
{
    "title": "Harry Potter and the Prisoner of Azkaban",
    "author": "J.K. Rowling"
}
```

When we press the ***Send*** button, if everything is fine, we will receive a reply with the status ***200 OK*** and the ID of the newly-added book, as can be seen in the image.


Postman File Edit View Window Help
Mon 21 Mar 14:38
Home Workspaces API Network Reports Explore
Search Postman
Invite Upgrade
IDP New Import
Collections APIs Environments Mock Servers Monitors Flows History
Lab 1
No Environment
Save
Send
Params Authorization Headers (9) Body Pre-request Script Tests Settings
Cookies Beautify
none form-data x-www-form-urlencoded raw binary GraphQL JSON
Body Cookies Headers (7) Test Results
Save Response
Pretty Raw Preview Visualize JSON
Find and Replace Console
Capture requests and cookies Bootcamp Runner Trash

localhost:5555/api/books

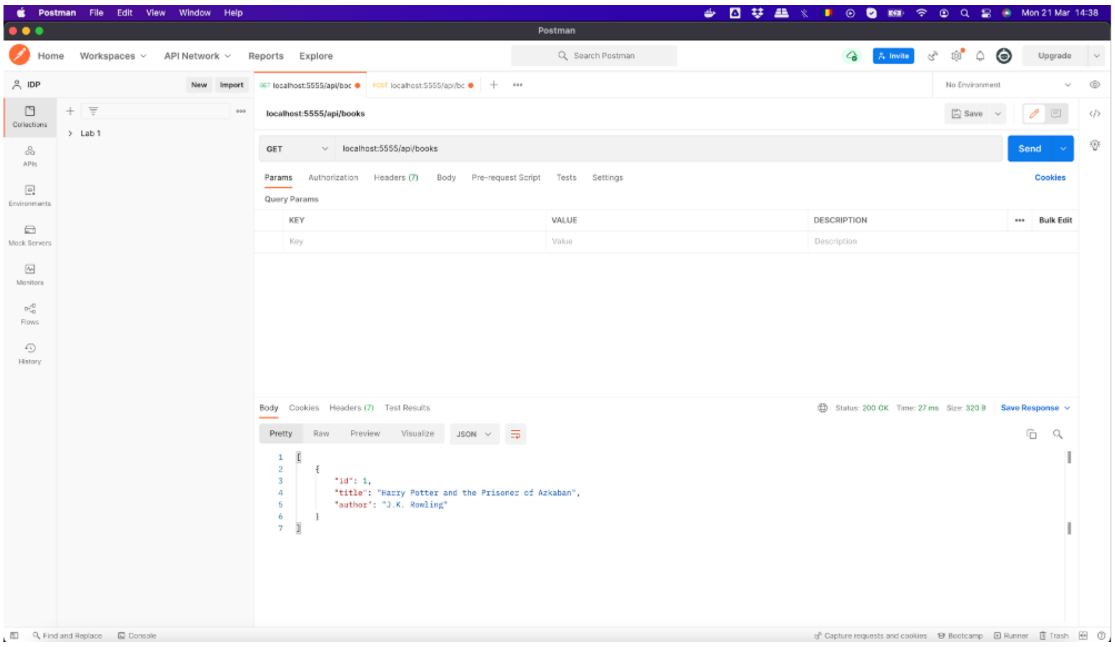
POST localhost:5555/api/books

```
{
    "title": "Harry Potter and the Prisoner of Azkaban",
    "author": "J.K. Rowling"
}
```

Status: 200 OK Time: 126 ms Size: 241 B

```
{
    "id": 1
}
```

Finally, if we run the ***GET*** request again, we can check the existence of the book we added in the library.


Postman File Edit View Window Help
Mon 21 Mar 14:38
Home Workspaces API Network Reports Explore
Search Postman
Invite Upgrade
IDP New Import
Collections APIs Environments Mock Servers Monitors Flows History
Lab 1
No Environment
Save
Send
Params Authorization Headers (7) Body Pre-request Script Tests Settings
Cookies
Query Params
Bulk Edit
Body Cookies Headers (7) Test Results
Save Response
Pretty Raw Preview Visualize JSON
Find and Replace Console
Capture requests and cookies Bootcamp Runner Trash

localhost:5555/api/books

GET localhost:5555/api/books

KEY	VALUE	DESCRIPTION
Key	Value	Description

Status: 200 OK Time: 27 ms Size: 320 B

```
[
    {
        "id": 1,
        "title": "Harry Potter and the Prisoner of Azkaban",
        "author": "J.K. Rowling"
    }
]
```

If you are using Play with Docker instead of running on your own computer, you need to replace the ***localhost:5555*** part of the address with the VM's actual address (see lab 3 for a reminder on how to do this).

Subtasks

1. install Postman
2. send a ***GET*** request to your application based on the steps above and check that it returns ***200 OK*** and an empty list of books

3. add a book of your choosing in the library using a ***POST*** request based on the steps above and check that it returns ***200 OK***
4. send a ***GET*** request again and check that you can see the book you just added

03. Testing for data persistence

Subtasks

1. stop and delete the three containers started at exercise 1
2. start them again using the same commands given at exercise 1
3. send a ***POST*** request using Postman and verify that the book you added at exercise 2 is still in the library

04. Administrating a database

Subtasks

1. go to the Adminer interface on http://localhost:8080 (or see lab 3 if you are working on Play with Docker)
2. log into the database using the following details:
    1. System: PostgreSQL
    2. Server: lab4-db
    3. Username: admin
    4. Password: admin
    5. Database: books
3. check that you can observe a table called ***books*** and click on it
4. click the ***Select data*** button to see the books already in the database (you should see the book you previously added)
5. click the ***New item*** button and add a new book (you should only fill in the ***title*** and ***author*** text boxes and then press ***Save***)
6. using Postman, send a ***GET*** request and check that the backend also returns the book you added in Adminer

05. Cleaning up

Subtasks

1. stop and remove the three containers created previously

2. remove the two networks created previously
3. remove the volume created previously

If you use docker compose, the actions above will be automatically done with docker compose down.