Documentation for FLASK backend

Souldis

Arnór Heimir Sigurðsson

Katrín Lilja Pétursdóttir

Þór Breki Þorgrímsson

Table of contents

How to run

First head to the github page and clone into the project.
https://github.com/ArnorHeimir/Lokaverkefni-Souldis-API
To run our backend after cloning the following things must be setup first:

- Docker desktop
- Env file (found in resources)

First you need to head to the Docker Desktop official website and download docker desktop
Tutorial links for:

- Windows
- Linux
- Mac
-

After the Docker desktop has been set up we can move on to the Env file.
The Env is a file that holds all the data needed to connect to our database and our secret key that is used for hashing.

Env File setup

- Step 1 is create a new file within the “src” directory called “.env”
- Step 2 copy the data found here and put it into the .env file you just created
- Done!

If you're not going to be using our database the information can be changed to match your own database.

Now the setup should be complete and the backend can be started using the command “**docker compose up**” while inside the directory.

Disclaimer!
If the code is changed in any way you must run “**docker compose up -d --build”** this command rebuilds the image and implements the changes that have been made.

General information

JWT tokens: JWT tokens are required on all endpoints except login and register. When a user logs in a token is created and sent to the frontend so it can send it with the other requests
Password hashing: the backend hashes all user passwords so there is not a threat to the users password if there is a data leak.
Database: the database is being hosted temporarily on Elephantsql but can be set up with the database file that comes with the project everywhere that supports postgreSQL. If the database is changed the information in the Env file must be changed to match the new database.

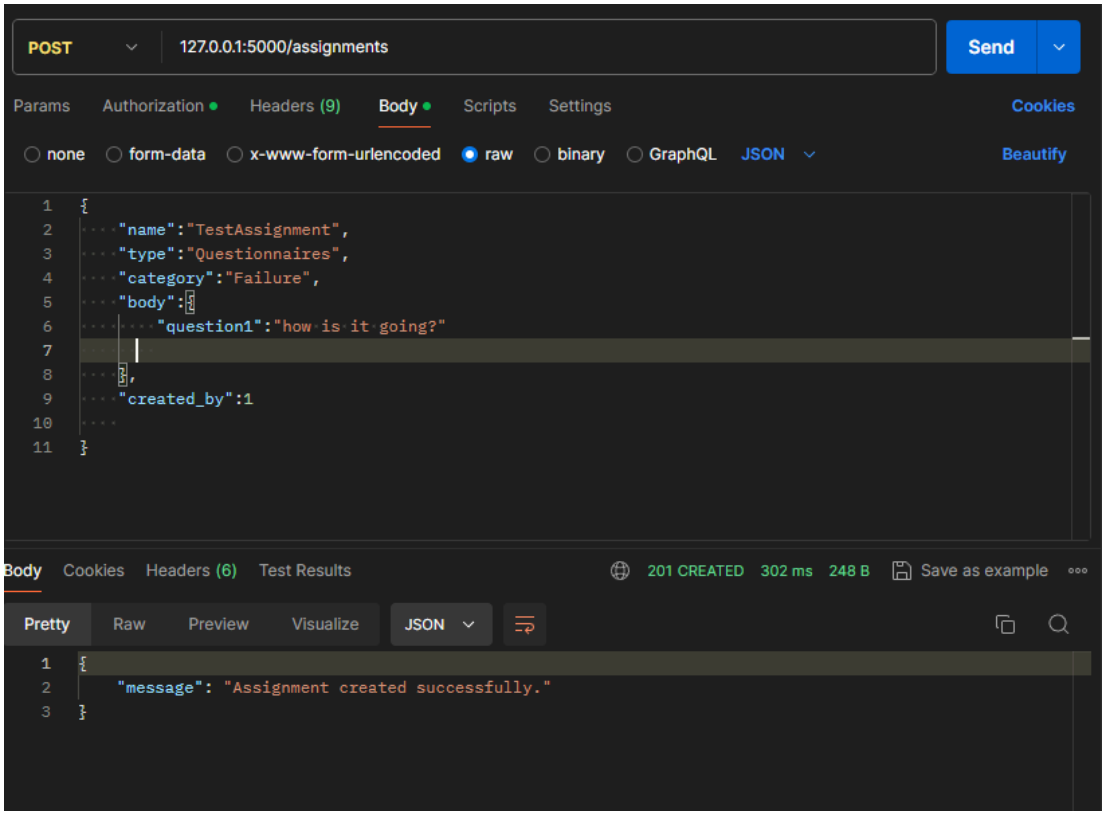
Secret key: the secret key that is found in the env file is the information that is used to create the token so it is unique to our project. This information can be changed to any string .
Authmiddleware: this file is to ensure that endpoints that require tokens are not accessible.
Database Middleware: this file is to create a connection to the database that is then used in all of the endpoints.

Available endpoints

Endpoints are sorted into 5 different files under the “routes” directory. In this section we will dive into what endpoints are available as well as how to access them. Examples will be shown of POST and PATCH requests but not the GET requests as they are more straightforward and mostly do the same thing.

Assignments.py

URL: http://localhost:5000/assignments
Methods available: GET,POST
Token required in Authorization Header: Yes
GET request: returns all assignments within the database
POST request: Expects the fields in body: name,type,category,created_by,body. Tries creating an assignment and returns a message if it succeeds or fails.
Example POST request:



URL: http://localhost:5000/assignments/user/<user_id>

Methods available: GET
Token required in Authorization Header: Yes
GET request: returns all assignments that are created by the user_id found in the URL

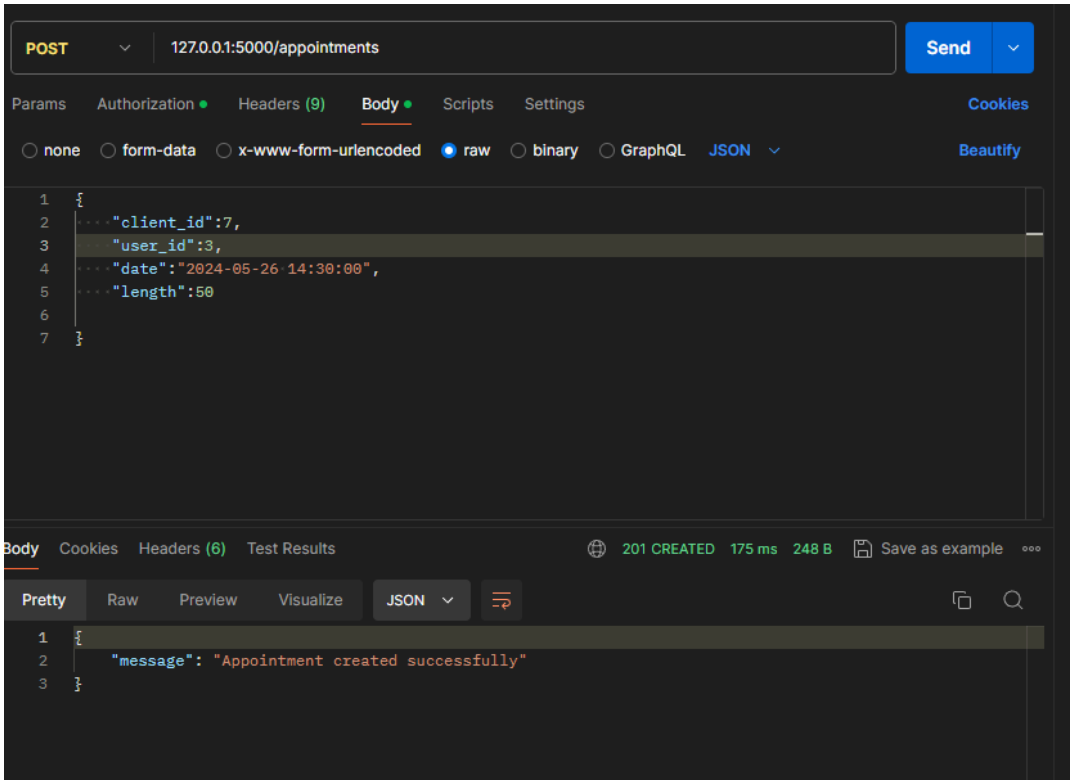
URL: http://localhost:5000/assignments/client/<client_id>
Methods available: GET
Token required in Authorization Header: Yes
GET request: returns all assignments that have been assigned to the client with the client_id found in the URL.

URL: http://localhost:5000/assignments/<id>
Methods available: GET,DELETE,PATCH
Token required in Authorization Header: Yes
GET request: returns the assignment with the id found in the URL.
DELETE request: Tries to remove the assignment with the id found in the URL. returns a message if it succeeds or fails.
PATCH request: checks for the fields in body: name,type,category,created_by,body. Tries to update the assignment with the id found in the URL. returns a message if it succeeds or fails.

URL: http://localhost:5000/assignments/type/<type>
Methods available: GET
Token required in Authorization Header: Yes
GET request: returns all assignments that have the type found in the URL.

Appointments.py

URL: http://localhost:5000/appointments
Methods available: GET,POST
Token required in Authorization Header: Yes
GET request: returns all appointments within the database
POST request: Expects the fields in body: client_id,user_id,date,length. Tries creating an appointments and returns a message if it succeeds or fails
Example POST request:

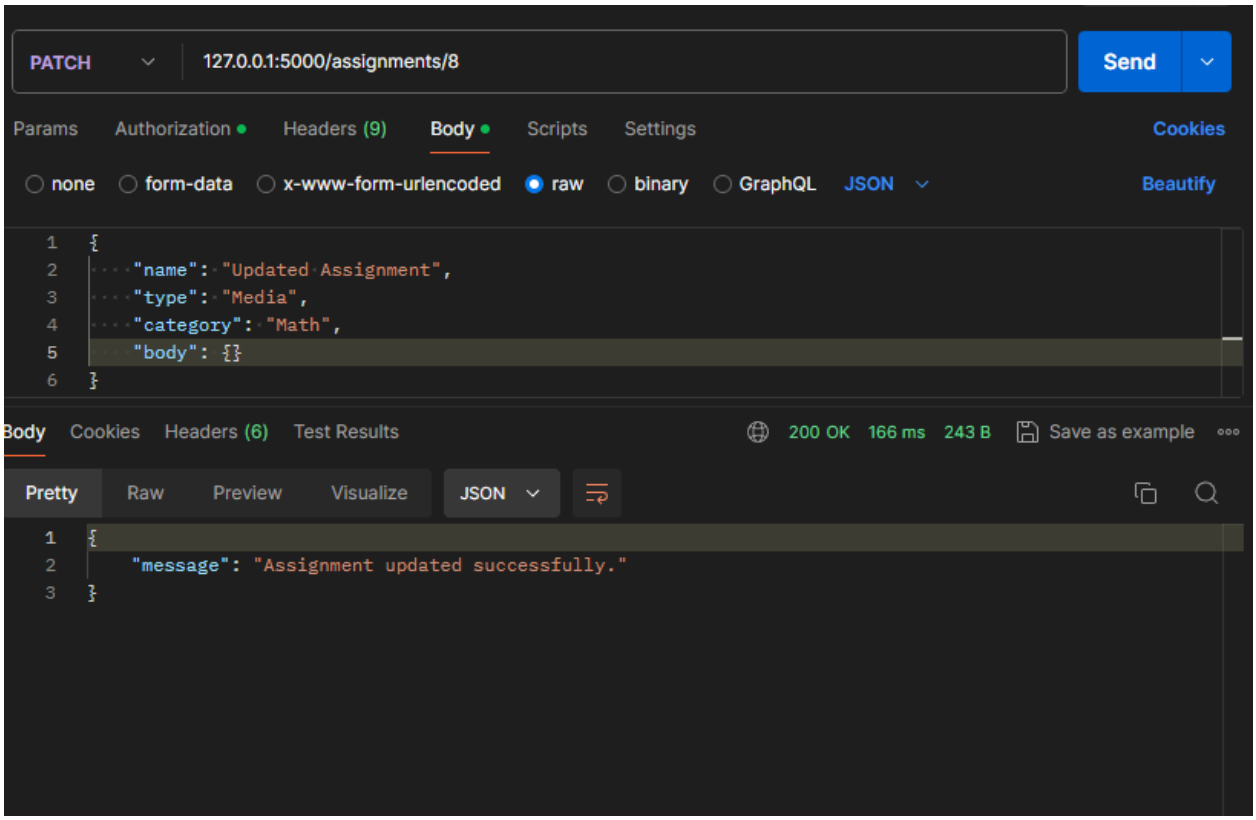


URL: http://localhost:5000/appointments/user/<user_id>
Methods available: GET
Token required in Authorization Header: Yes
GET request: returns all appointments within the database that has the user_id as the user_id found in the URL.

URL: http://localhost:5000/appointments/client/<client_id>
Methods available: GET
Token required in Authorization Header: Yes
GET request: returns all appointments within the database that has the client_id as the client_id found in the URL.

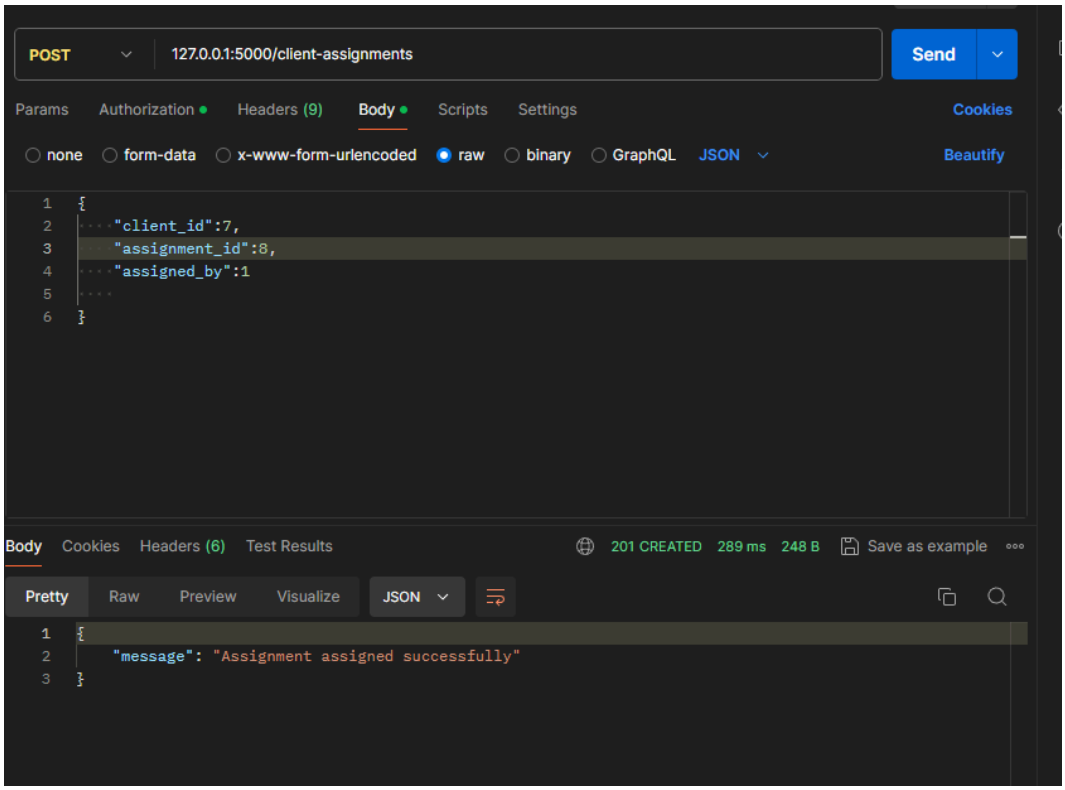
URL: http://localhost:5000/appointments/date/<date>/<user_id>
Methods available: GET
Token required in Authorization Header: Yes
GET request: returns all appointments within the database that has the user_id as the user_id found in the URL and is on the same date as the date found in the URL.

URL: http://localhost:5000/appointments/<id>
Methods available: GET,DELETE,PATCH
Token required in Authorization Header: Yes
GET request: returns all appointments within the database that has the id as the id found in the URL.
DELETE request: Tries to delete the appointment with the id found in the URL. returns a message if it succeeds or fails.
PATCH request: checks for fields in the body: client_id,user_id,date,length. Tries to update the appointment with the id found in the URL in the database with the fields found in the body.
Example PATCH request:



clientAssignments.py

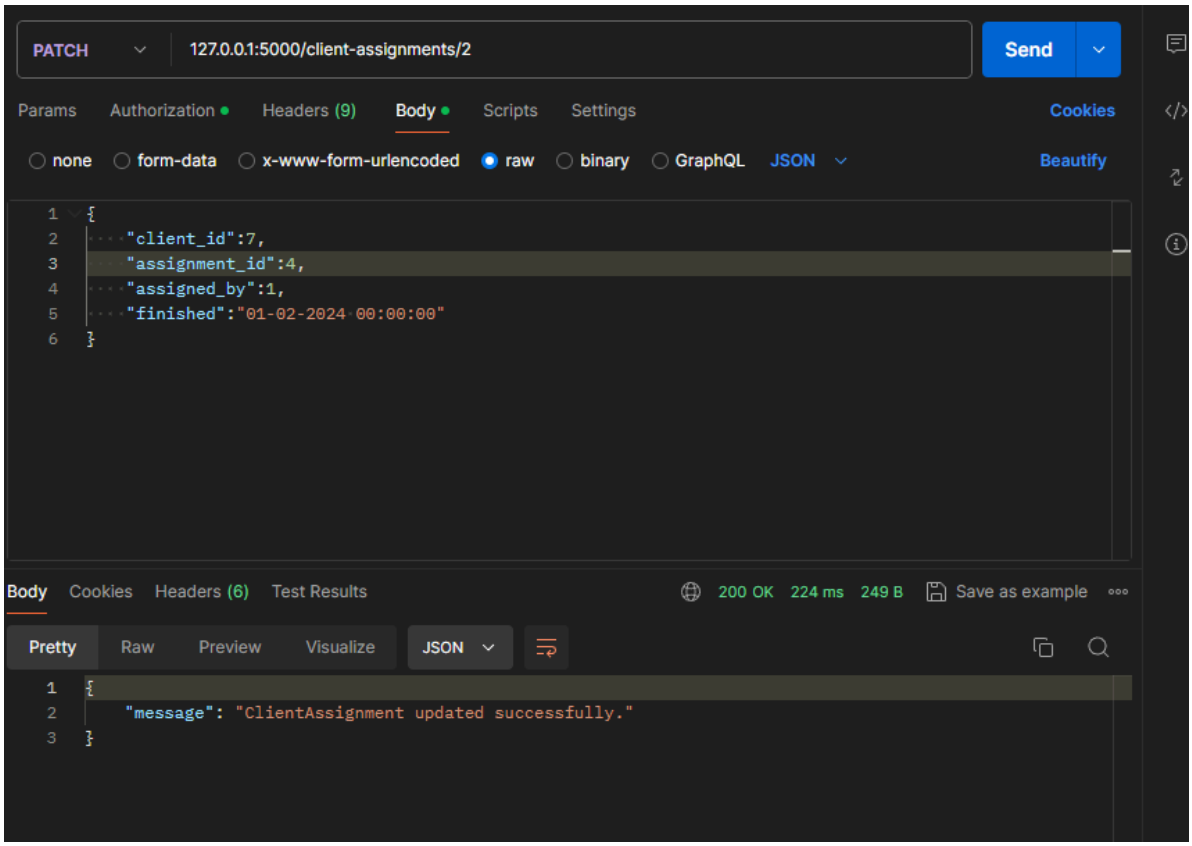
URL: http://localhost:5000/client-assignments
Methods available: POST
Token required in Authorization Header: Yes
POST request: Expects the fields in body: **client_id, assignment_id, assigned_by**. Tries assigning an assignment to a client and returns a message if it succeeds or fails.
Example POST request:



URL: http://localhost:5000/client-assignments/user/<user_id>
Methods available: GET
Token required in Authorization Header: Yes
GET request: returns all client-assignments that have been assigned by the user_id found in the URL.

URL: http://localhost:5000/client-assignments/client/<client_id>
Methods available: GET
Token required in Authorization Header: Yes
GET request: returns all client-assignments that have been assigned to the client_id found in the URL.

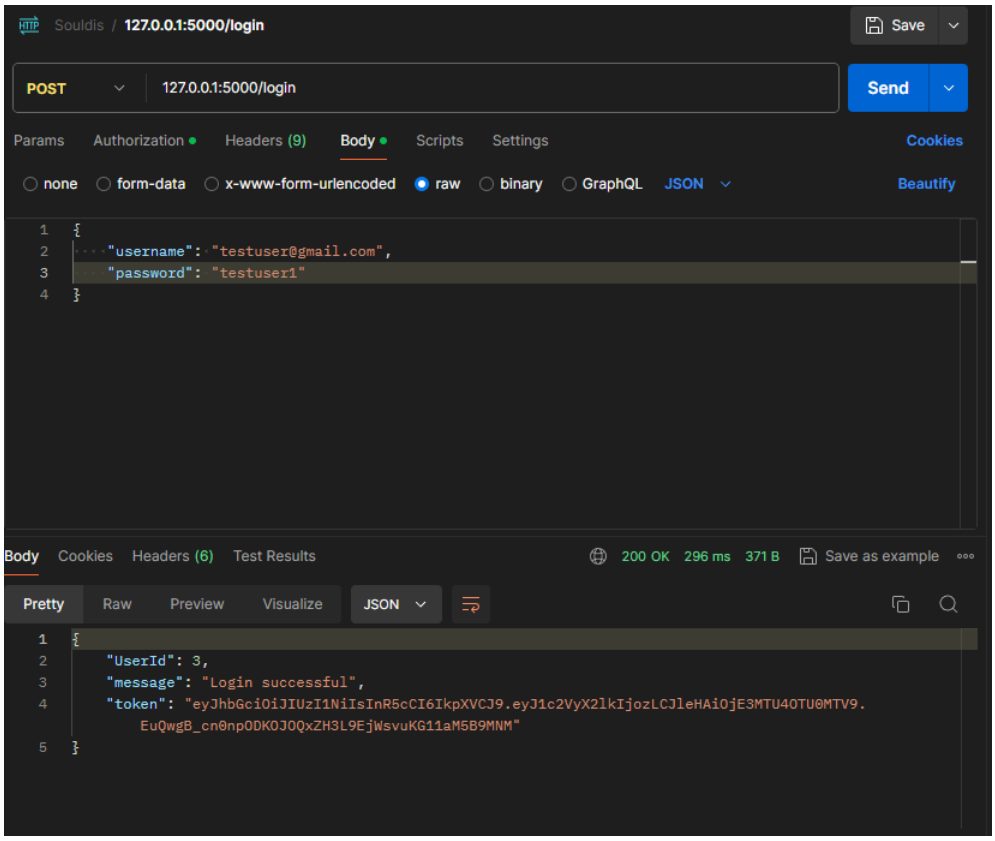
URL: http://localhost:5000/client-assignments/<id>
Methods available: GET,PATCH
Token required in Authorization Header: Yes
GET request: returns the client-assignment that has the id found in the URL.
PATCH request: checks for the fields in body:
client_id,assignment_id,assigned_by,finished,body. Tries to update the client-assignment that has the id found in the URL with the fields found in the body. Returns a message if it succeeds or fails.
Example PATCH request:



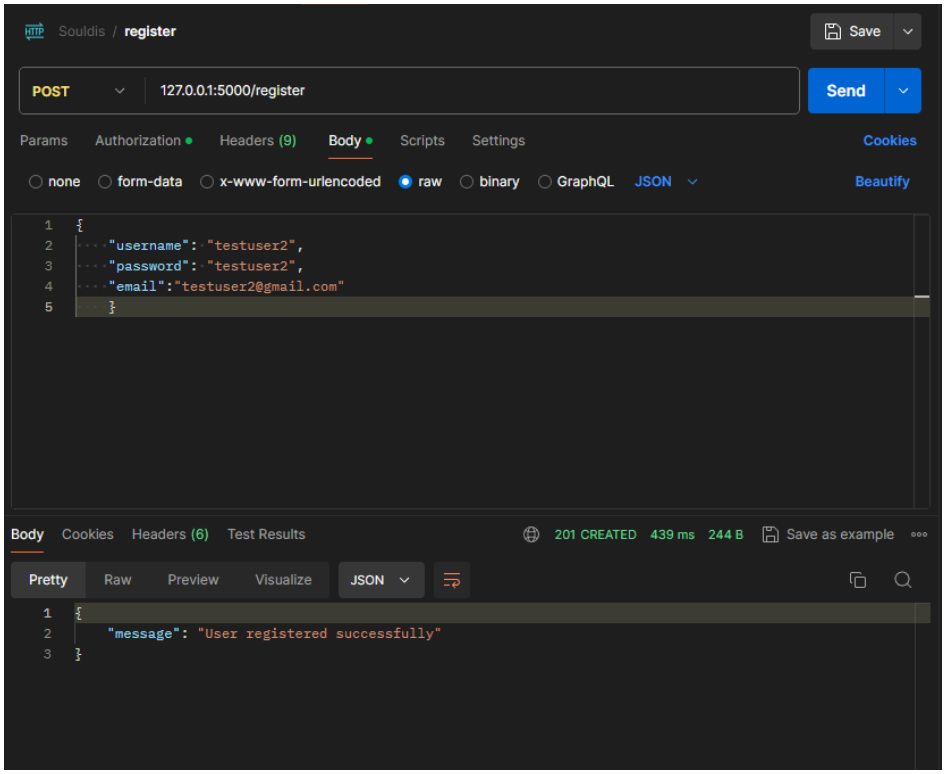
User.py

URL: http://localhost:5000/login
Methods available: POST
Token required in Authorization Header: No
POST request: Expects the fields in body: **username,password.** Checks the database if this matches to a user that exists. Returns the user_id and a valid JWT token to access other endpoints.

Extra info: login can be both performed with the email of the user as well as their username.
Example POST request:



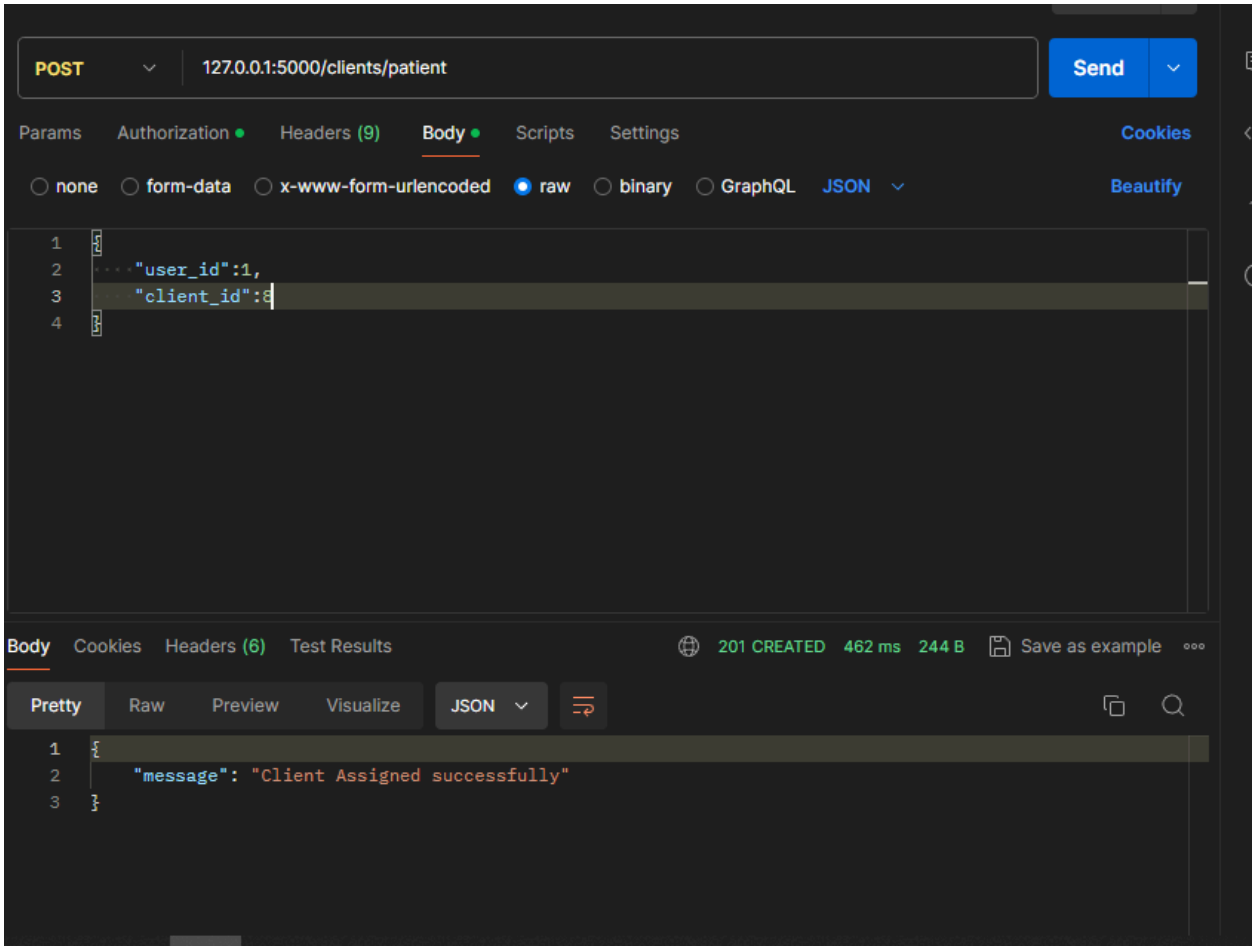
URL: http://localhost:5000/register
Methods available: POST
Token required in Authorization Header: No
POST request: Expects the fields in body: **username, password, email.** Tries to create the user in the database. Returns a message if it succeeds or fails.
Example POST request:



Clients.py

URL: http://localhost:5000/clients
Methods available: GET,POST
Token required in Authorization Header: Yes
GET request: returns all clients in the database.
POST request: Expects the fields in body: **full_name, birth_day, image.** Tries to create the client in the database. Returns a message if it succeeds or fails.

URL: http://localhost:5000/clients/patient
Methods available: POST
Token required in Authorization Header: Yes
POST request: Expects the fields in body: **client_id, user_id.** Tries to create a connection in the UserClients table making the client a patient for the user. Returns a message if it succeeds or fails.
Example POST request:



URL: http://localhost:5000/clients/patient/<user_id>
Methods available: GET
Token required in Authorization Header: Yes
GET request: returns all the clients that are patients of the user_id found in the URL.

URL: http://localhost:5000/clients/client_id
Methods available: GET
Token required in Authorization Header: Yes
GET request: returns the client with the client_id that is found in the URL.

Resources

Contents of the ENV file.

```
DB_NAME=hcbljknd
DB_USER=hcbljknd
DB_PASSWORD=c1DFftTx8JAtRX95wYQJAcMQgZ4jYij8
DB_HOST=flora.db.elephantsql.com
DB_PORT=5432
SECRET_KEY=souldis_api_ahs
```