**Flask API for an electricity bill**

**management system Documentation**

1. **My contact :**

* Full name : zouhir Khalil
* Email : [khalilz@hotmail.fr](mailto:khalilz@hotmail.fr)
* Linkedin : /khalil-zouhir-92382b236
* Github : /KhalilZouhir98
* Phone Number : 0642660196

1. **Introduction**

This documentation provides an overview of my Flask API that interacts with a MySQL database for an Electricity Bill Management System. The API supports various operations such as inserting records, fetching records, updating records, and deleting records related to users, meters, bills, and electricity readings.

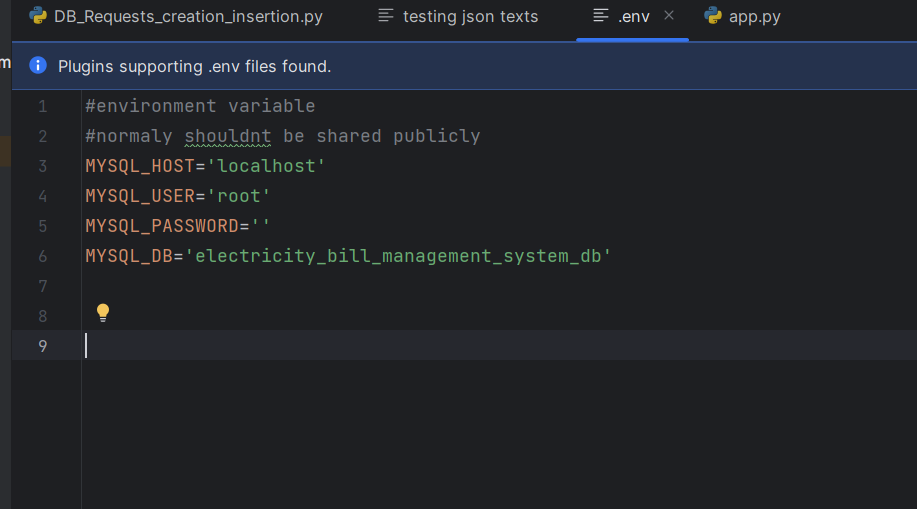
To gain a comprehensive understanding of the database structure, relationships, and other important details concerning the project please check my project documentation in the Git repo and thank you .

1. [**Environment Setup**](https://chat.openai.com/c/1ee6f4ea-8ec8-4817-8ec5-e7c2b63e2f2a#environment-setup)**:**
   1. **Prerequisites :**

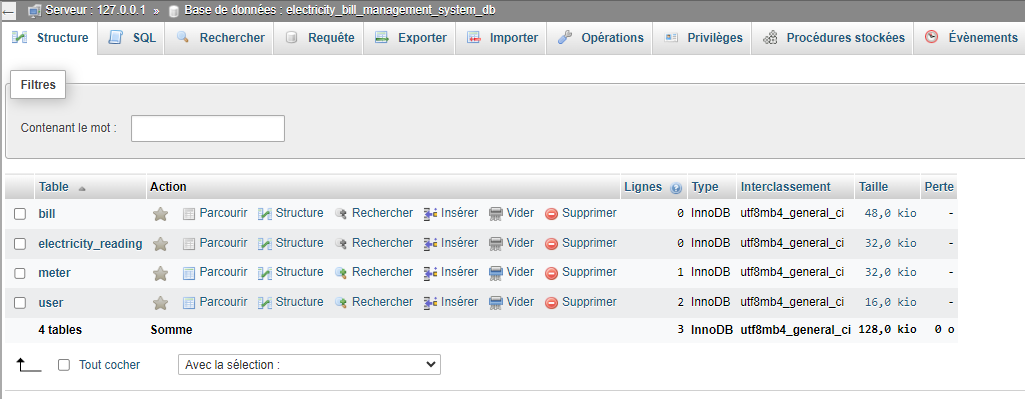
* The great news is that you don't need to install any packages all necessary packages are available in the virtual environment folder (venv).



* MYSQL
* The code was developed in PyCharm .
* Postman for testing
  1. **Installation  :**
* Clone the GitHub Repo
* Make changes on the .env file to set up your database



* Import the “electricity\_bill\_management\_system\_db” database in mysql (all in repo)

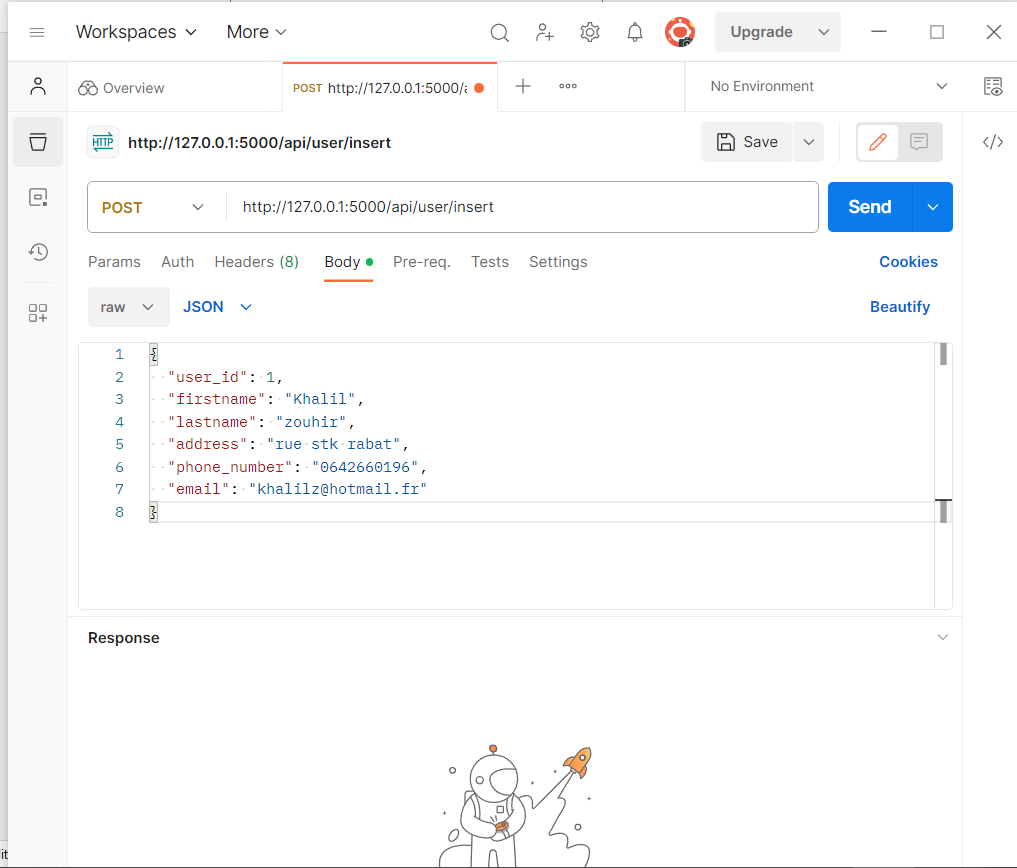


* Run the app

1. **API Endpoints :**

The code is runing in our local machine so our URI is : <http://127.0.0.1:5000/> u can change the port number obviously

* 1. **Insert operations :** 
     1. **Insert user :**
* Endpoint : ‘POST /api/user/insert’
* Request payload :



* Response :



Will create the table if it doesn’t exist and insert a record in the user table .

* + 1. **Insert meter :**
* Endpoint : ‘POST /api/meter/insert’
* Request payload :

{

    "meter\_id":9868,

    "meter\_location":"rue fal wouel omairagdal rabat morocco",

    "user\_id":1

}

* Response :

{

    "message": "meter with id : 9868 owned by the user with the id :1 inserted successfully"

}

Will create the table if it doesn’t exist and insert a record in the meter table

* + 1. **Insert bill :**
* Endpoint : ‘POST /api/bill/insert’
* Request payload :

{

    "meter\_id": 9868,

    "bill\_id": 74,

    "user\_id": 1,

    "bill\_date": "2023-12-05",

    "due\_date": "2023-12-06",

    "total\_amount": 98.00,

    "payment\_status": 1

}

* Response :

{

    "message": "bill with id : 74 of the meter with id :9868 owned by the user with the id :1 inserted successfully"}

Will create the table if it doesn’t exist and insert a record in the bill table

* + 1. **Insert Electricity Reading :**
* Endpoint : ‘POST /api/electricity\_reading/insert’
* Request payload :

{

    "meter\_id": 9868,

    "reading\_id": 7554,

    "reading\_value": 98456,

    "reading\_date": "2023-12-05"

}

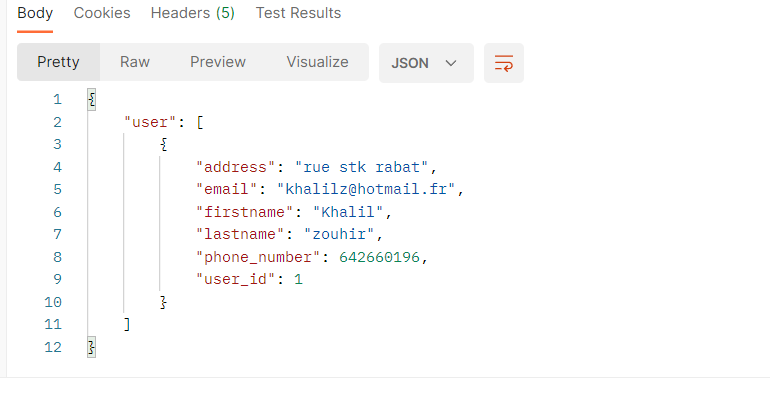
* Response :

{

    "message": "electricity reading with id : 7554 of the meter with id :9868 inserted successfully"

}

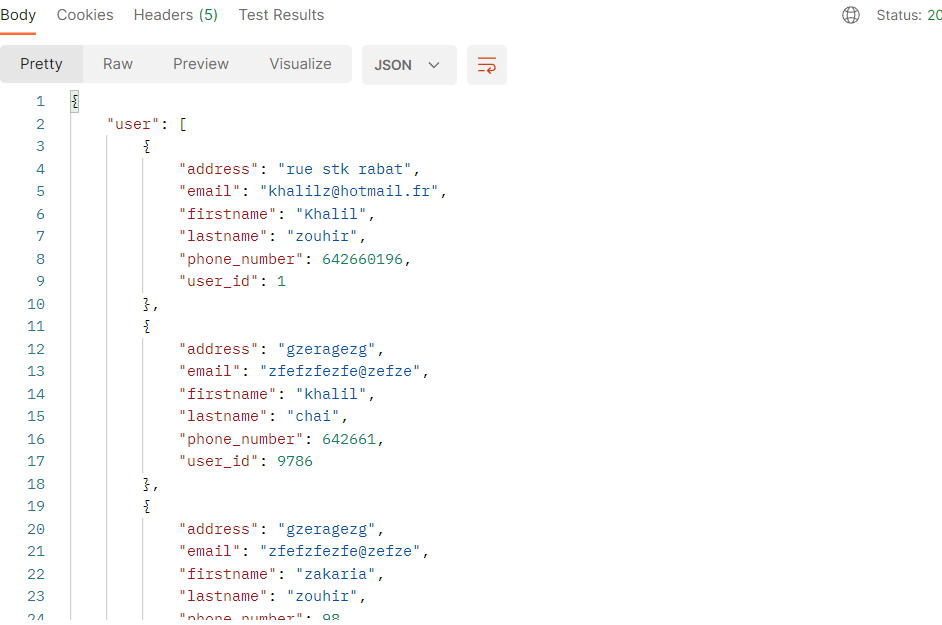
* 1. **Select operations :**
     1. **Get user by ID :**
* Endpoint : ‘GET /api/user/<int:user\_id>
* Response :



Select user with the specified id.

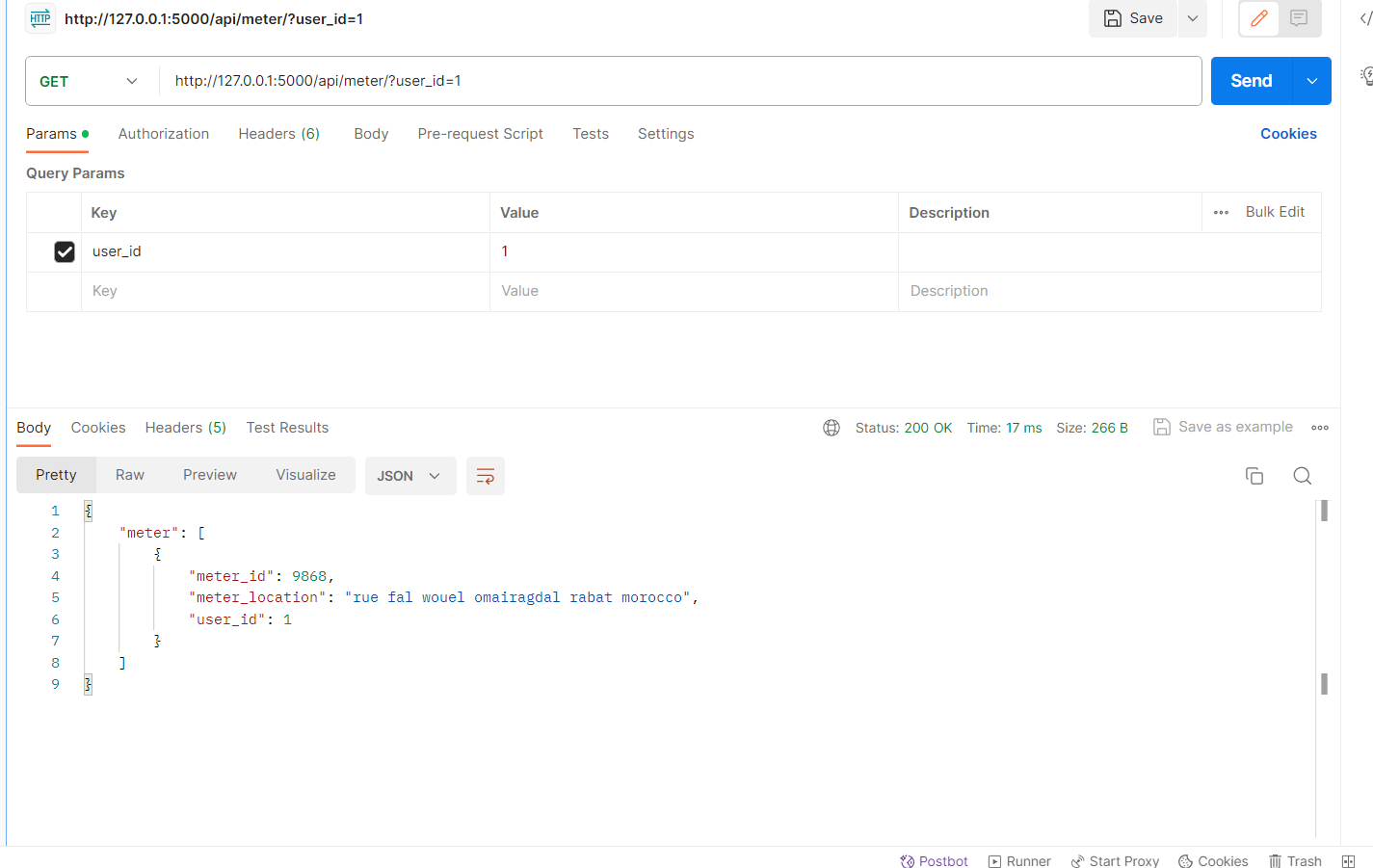
* + 1. **Get user :**
* Endpoint : ‘GET /api/user/’
* Query Parameters :
  1. Firstname
  2. Lastname
  3. Address
  4. phone\_number
  5. email
  6. user\_id
* Response :

Without any parameters will display all users in database :



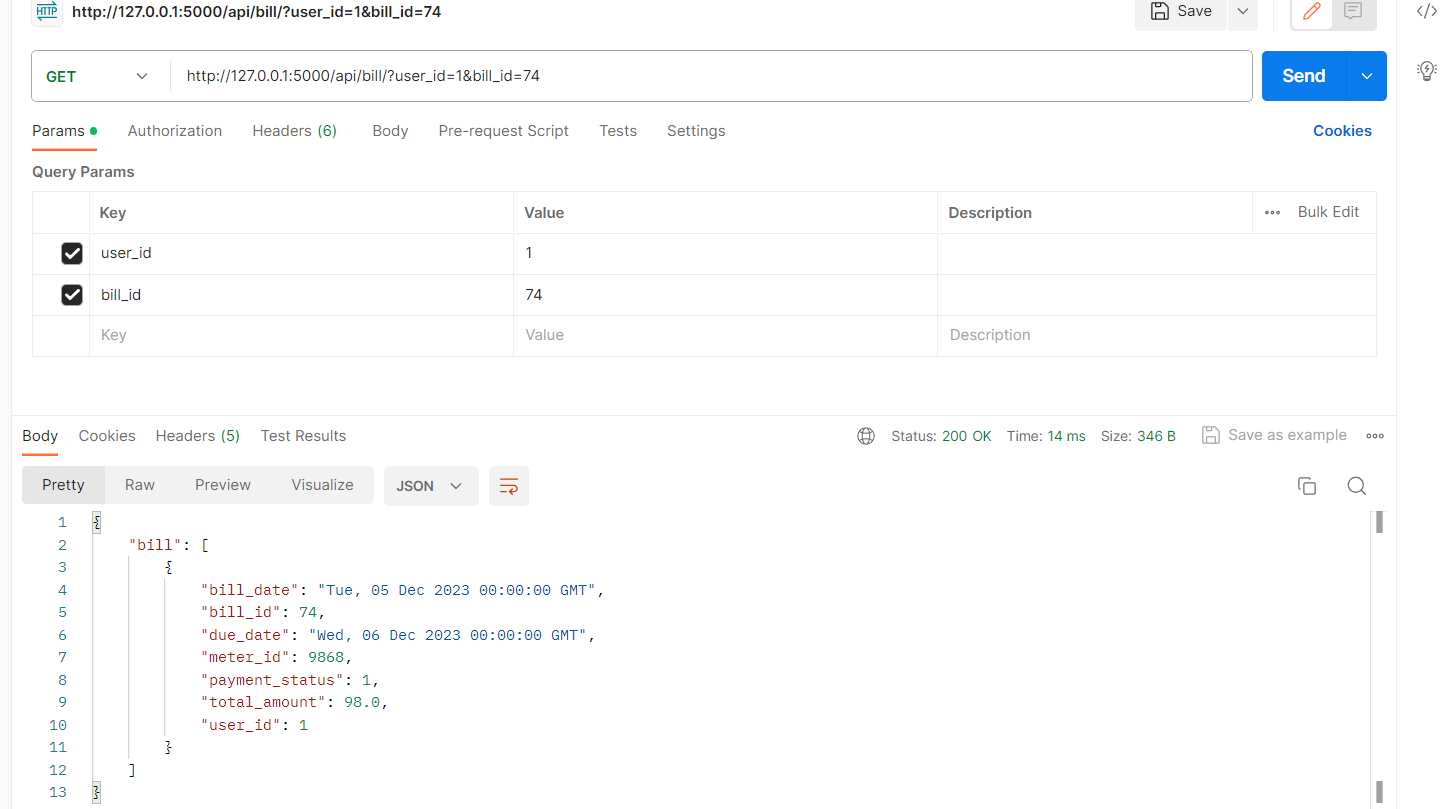
* + 1. **Get meter**
* Endpoint : ‘GET /api/meter/’
* Query Parameters :
  1. Meter\_id
  2. Meter\_location
  3. meter\_id
* Response :

Will return all records of the table or return specifique records if we use parameters exemple :



* + 1. **Get bill :**
* Endpoint : ‘GET /api/bill/’
* Query Parameters :
  1. bill\_id
  2. bill\_date
  3. due\_date
  4. total\_amount
  5. payment\_status
* Response :

Will return all records of the table or return specifique records if we use parameters exemple of using multiple parameters :



* + 1. **Get electricity reading :**
* Endpoint : ‘GET /api/electricity\_reading/’
* Query Parameters :
  1. Reading\_id
  2. Reading\_date
  3. Reading\_value
  4. Meter\_id
* Response :

Will return all records of the table or return specifique records if we use parameters exemple :

{

    "electricity\_reading": [

        {

            "meter\_id": 9868,

            "reading\_date": "Tue, 05 Dec 2023 00:00:00 GMT",

            "reading\_id": 7554,

            "reading\_value": 98456

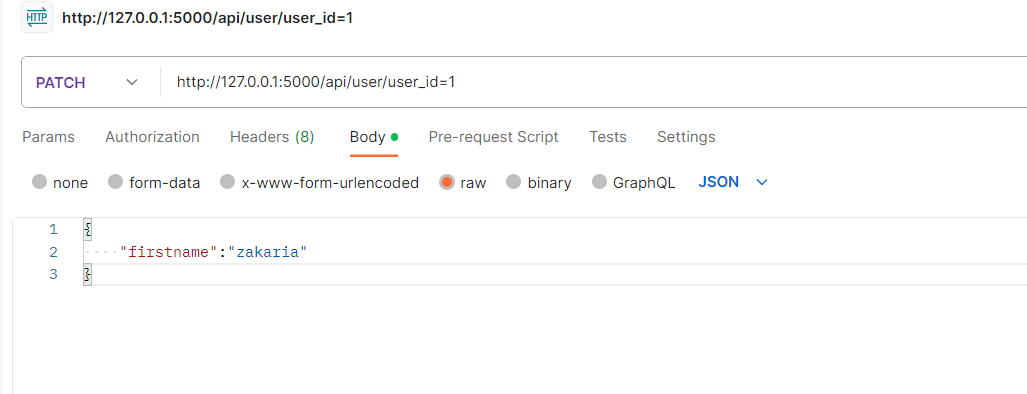
        }

    ]

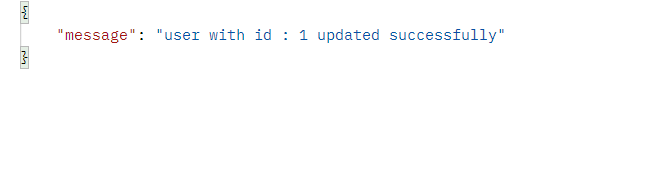
}

* 1. **Update operations :**
     1. **Update user :**
* Endpoint : ‘patch /api/user/user\_id=<int:user\_id>’
* Request payload :

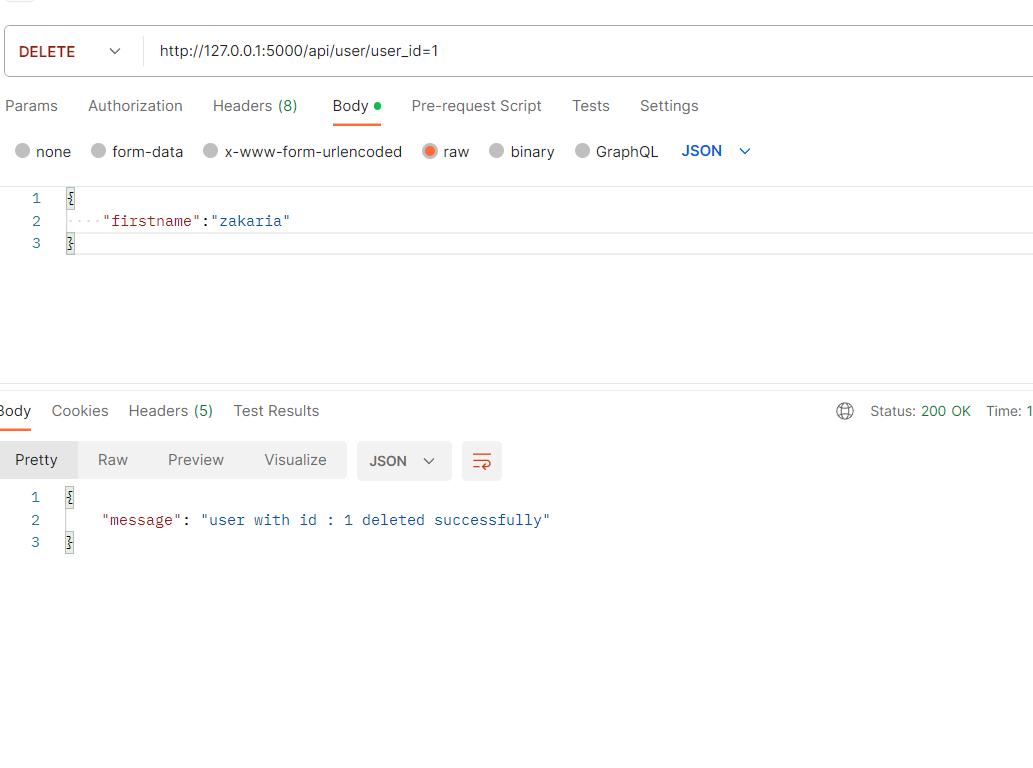
exemple updating lastname :



* Response :



* + 1. **Update meter :**
* Endpoint : ‘patch /api/meter/meter\_id=<int:meter\_id>’
  + 1. **Update Bill :**
* Endpoint : ‘patch /api/bill/bill\_id=<int:bill\_id>’
  + 1. **Update Electricity Reading :**
* Endpoint : ‘patch /api/electricity\_reading/reading \_id=<int:reading \_id>’ .
  1. **Delete Operations :** 
     1. **Delete user :**
* Endpoint : ‘delete /api/user/user\_id=<int:user\_id>'
* Response :



* + 1. **Delete meter :**
* Endpoint : ‘delete /api/meter/meter\_id=<int: meter\_id>'
  + 1. **Delete bill :**
* Endpoint : ‘delete /api/bill/bill\_id=<int: bill\_id>'
  + 1. **Delete electricity reading :**
* Endpoint : ‘delete /api/electricity\_reading/reading\_id=<int:reading\_id>