Mara Muda Pohan

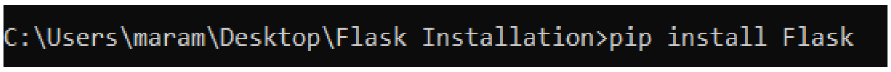
API Application User CRUD Test

Requirements:

- Python
- Flask
- Postman

Installation

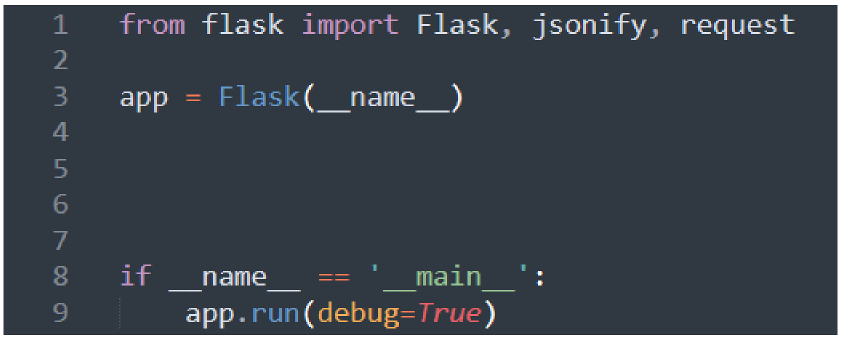
Assuming Python is installed, make a folder and create a .py file to code in and then open in cmd and change directory to the folder created, run the command "pip install Flask".



```
C:\Users\maram\Desktop\Flask Installation>pip install Flask
```

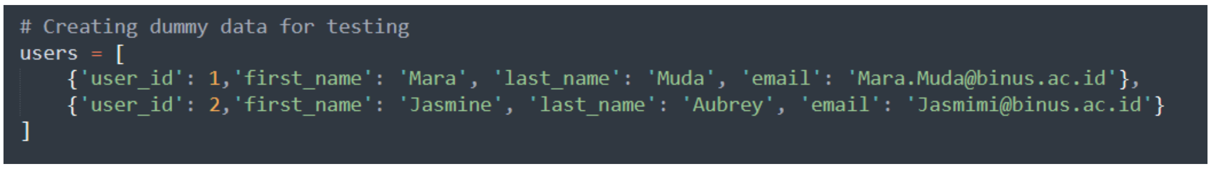
After Installation is done, we can now import Flask in our python code.

The following code is a basic template for a Flask application where I import Flask, jsonify, and request for basic requirements.



```python
from flask import Flask, jsonify, request

app = Flask(__name__)




if __name__ == '__main__':
    app.run(debug=True)
```

Since this is an API for managing user data, I created dummy data to test out basic functionalities such as GET user.

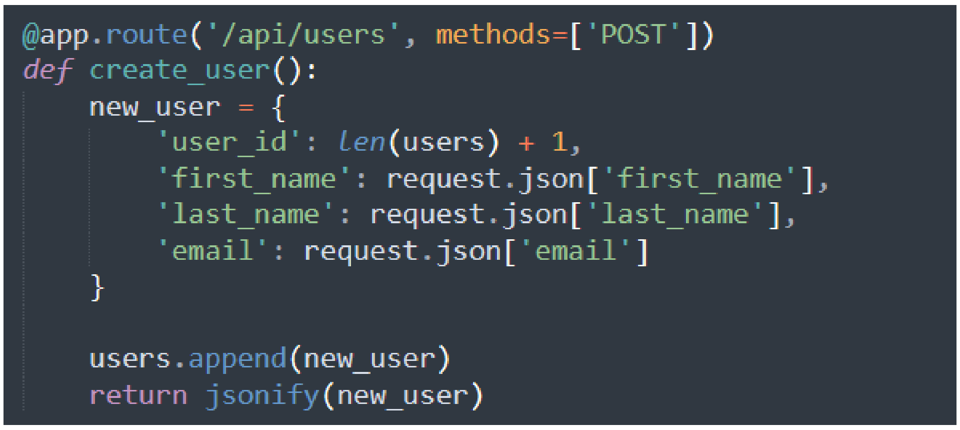


```python
# Creating dummy data for testing
users = [
    {'user_id': 1,'first_name': 'Mara', 'last_name': 'Muda', 'email': 'Mara.Muda@binus.ac.id'},
    {'user_id': 2,'first_name': 'Jasmine', 'last_name': 'Aubrey', 'email': 'Jasmimi@binus.ac.id'}
]
```

The next step is to create API routes for CRUD operations.

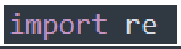
- **Create a new user.**
  Creating a new user requires us to use the "**POST**" method.
  We have to setup so the route goes to '/api/users' and make a function "create_user()"

Inside this function we will use the *request* function to request and receive data from the endpoint. The requested data will be in Json format.
Once we receive the information, simply append the new_user data to the list of the 'users' list from earlier.
We will return the new_user information to confirm to the endpoint that data has been received and saved.

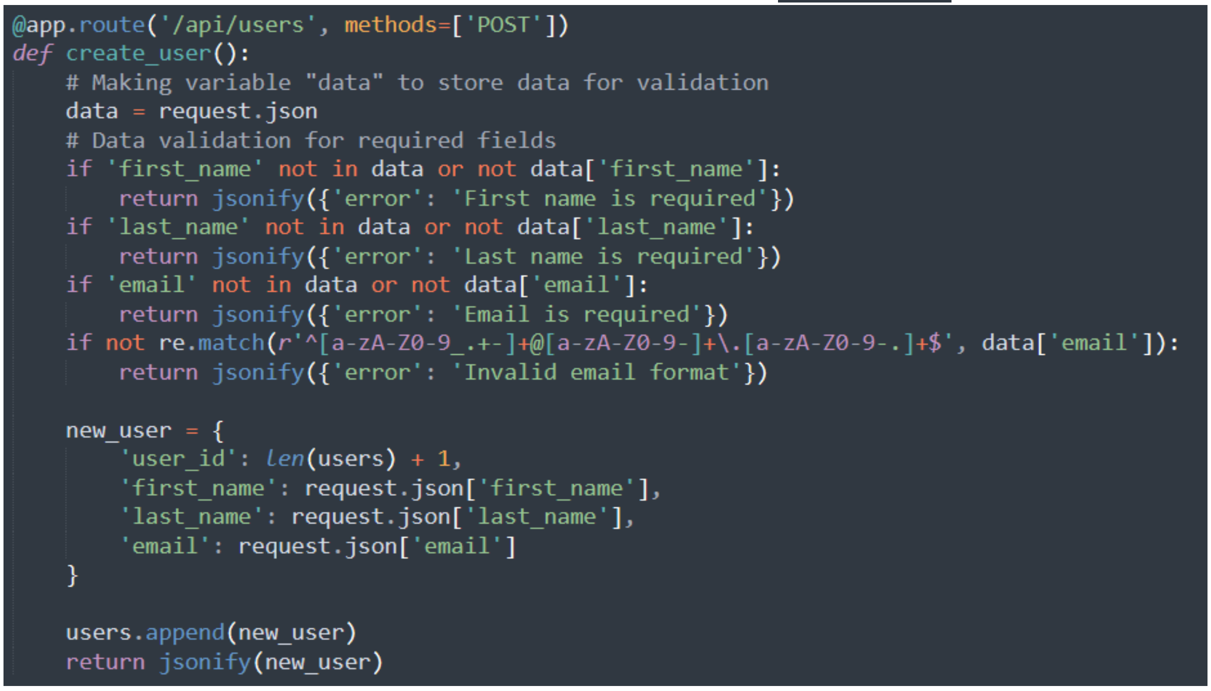


```python
@app.route('/api/users', methods=['POST'])
def create_user():
    new_user = {
        'user_id': len(users) + 1,
        'first_name': request.json['first_name'],
        'last_name': request.json['last_name'],
        'email': request.json['email']
    }

    users.append(new_user)
    return jsonify(new_user)
```

For data validation we will be using *ifs* statements where we will store the temporary data into the variable *data* and check if inside the Json message, does the required field exists or if it exists but has null/nothing inside the data.
To validate email, I will be using the *re (regular expression)* library from python. To set this up simply **import *re*** after importing flask libraries.

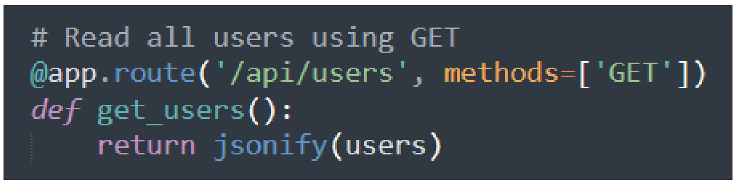


```python
import re
```



```python
@app.route('/api/users', methods=['POST'])
def create_user():
    # Making variable "data" to store data for validation
    data = request.json
    # Data validation for required fields
    if 'first_name' not in data or not data['first_name']:
        return jsonify({'error': 'First name is required'})
    if 'last_name' not in data or not data['last_name']:
        return jsonify({'error': 'Last name is required'})
    if 'email' not in data or not data['email']:
        return jsonify({'error': 'Email is required'})
    if not re.match(r'^[a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-.]+$', data['email']):
        return jsonify({'error': 'Invalid email format'})

    new_user = {
        'user_id': len(users) + 1,
        'first_name': request.json['first_name'],
        'last_name': request.json['last_name'],
        'email': request.json['email']
    }

    users.append(new_user)
    return jsonify(new_user)
```

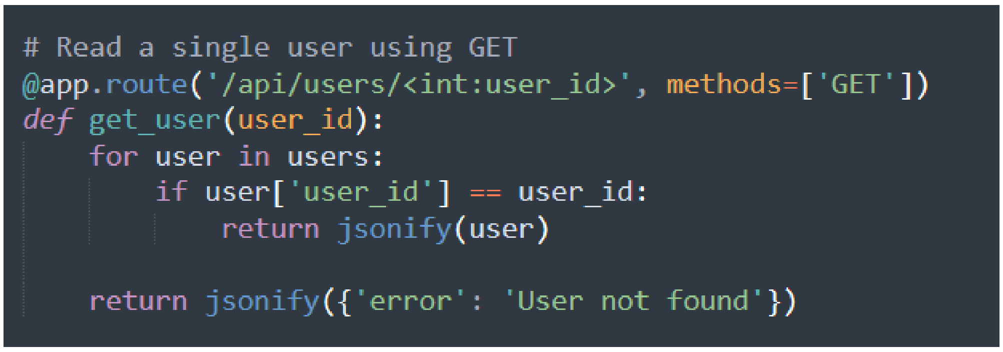
- **Read/Retrieve a list of all users.**
  In this step we will simply use the "**GET**" method and print all users in the list



```python
# Read all users using GET
@app.route('/api/users', methods=['GET'])
def get_users():
    return jsonify(users)
```

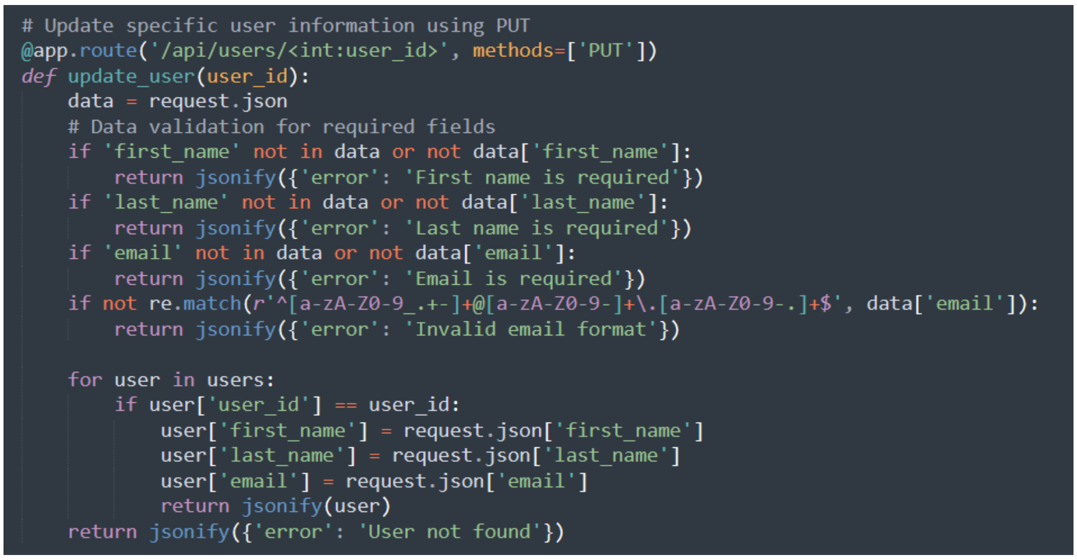
This will return the user list in a Json format.

- **Read/Retrieve a specific user.**
  In this function, we will ask for a user_id input from endpoint with '**GET**' method where the route format is '/api/users/<int:user_id>' and this code will search for user in users and see if any of the user_id input matches any of the existing users in the list and print the information, but if it doesn't exist then simply return an error message saying user is not found.



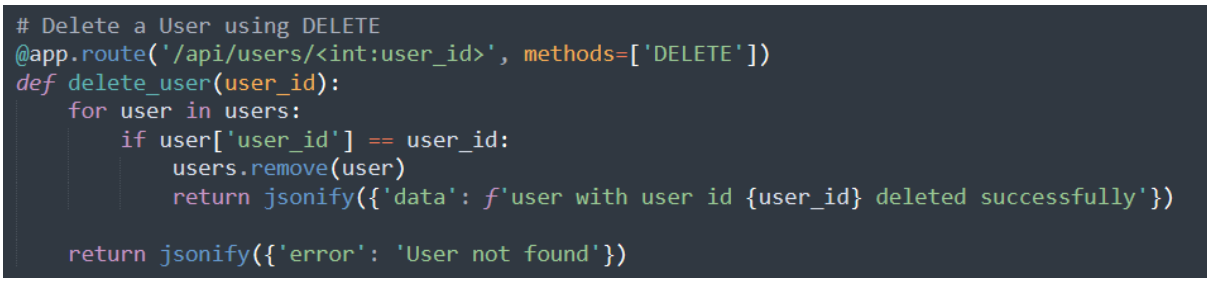
```python
# Read a single user using GET
@app.route('/api/users/<int:user_id>', methods=['GET'])
def get_user(user_id):
    for user in users:
        if user['user_id'] == user_id:
            return jsonify(user)

    return jsonify({'error': 'User not found'})
```

- **Update an existing user by user_id.**
  Here we will use '**PUT**' method for updating information, with the route being '/api/users/<int:user_id>'. We will go through data validation with the assumption that every field is changed and updated, if user_id is not found, it will send an error message.
  The code will ask for the new information on each variable starting from first_name, last_name, and email except for user_id. user_id will remain the same.



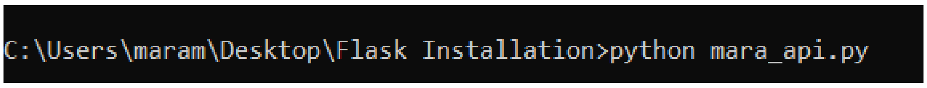
```python
# Update specific user information using PUT
@app.route('/api/users/<int:user_id>', methods=['PUT'])
def update_user(user_id):
    data = request.json
    # Data validation for required fields
    if 'first_name' not in data or not data['first_name']:
        return jsonify({'error': 'First name is required'})
    if 'last_name' not in data or not data['last_name']:
        return jsonify({'error': 'Last name is required'})
    if 'email' not in data or not data['email']:
        return jsonify({'error': 'Email is required'})
    if not re.match(r'^[a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-.]+$', data['email']):
        return jsonify({'error': 'Invalid email format'})

    for user in users:
        if user['user_id'] == user_id:
            user['first_name'] = request.json['first_name']
            user['last_name'] = request.json['last_name']
            user['email'] = request.json['email']
            return jsonify(user)
    return jsonify({'error': 'User not found'})
```

- **Deleting a user**
  In this function, we will user the '**DELETE**' method with '/api/users/<int:user_id>' route. This code will search for the matching user_id in users in the *for loop* and if a match is found, then use the 'users.remove(user)' to remove data. Upon successful task it will return a message saying which user_id has been deleted successfully, if there is no match then it will send an error message saying user not found.



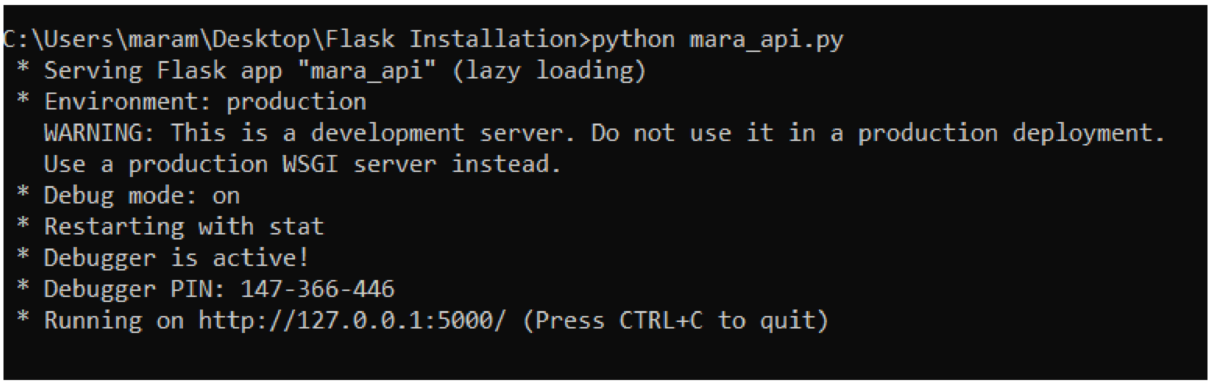
```python
# Delete a User using DELETE
@app.route('/api/users/<int:user_id>', methods=['DELETE'])
def delete_user(user_id):
    for user in users:
        if user['user_id'] == user_id:
            users.remove(user)
            return jsonify({'data': f'user with user id {user_id} deleted successfully'})

    return jsonify({'error': 'User not found'})
```

Testing the code.
To run and test the code, open cmd and change the directory to the folder where the code is located in. Then run the code.



```
C:\Users\maram\Desktop\Flask Installation>python mara_api.py
```

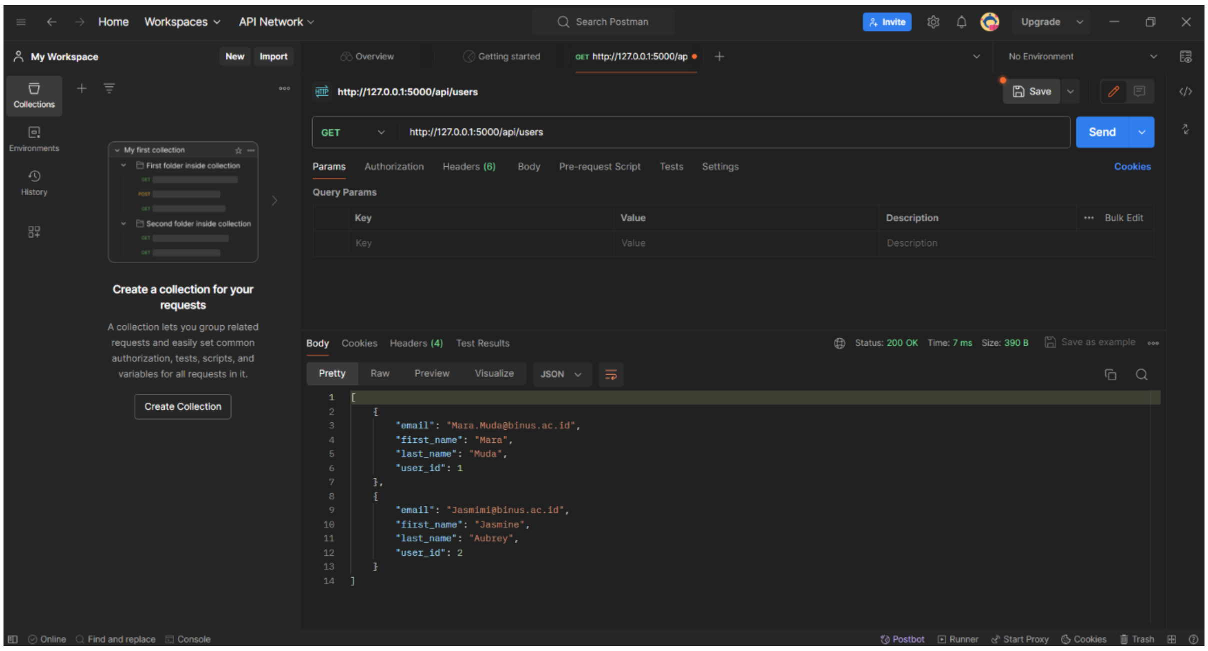
If there are no problems, this message will appear and tell us where the API is running.
In this case, it is running on http://127.0.0.1:5000/ or localhost.



```
C:\Users\maram\Desktop\Flask Installation>python mara_api.py
 * Serving Flask app "mara_api" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: on
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 147-366-446
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

- **Retrieving all user.**

Once it is running open postman and input the route we want to run. Here we will try the read all user function which uses GET and uses the '/api/users' route. It will show on the body the list of users that are stored.



http://127.0.0.1:5000/api/users

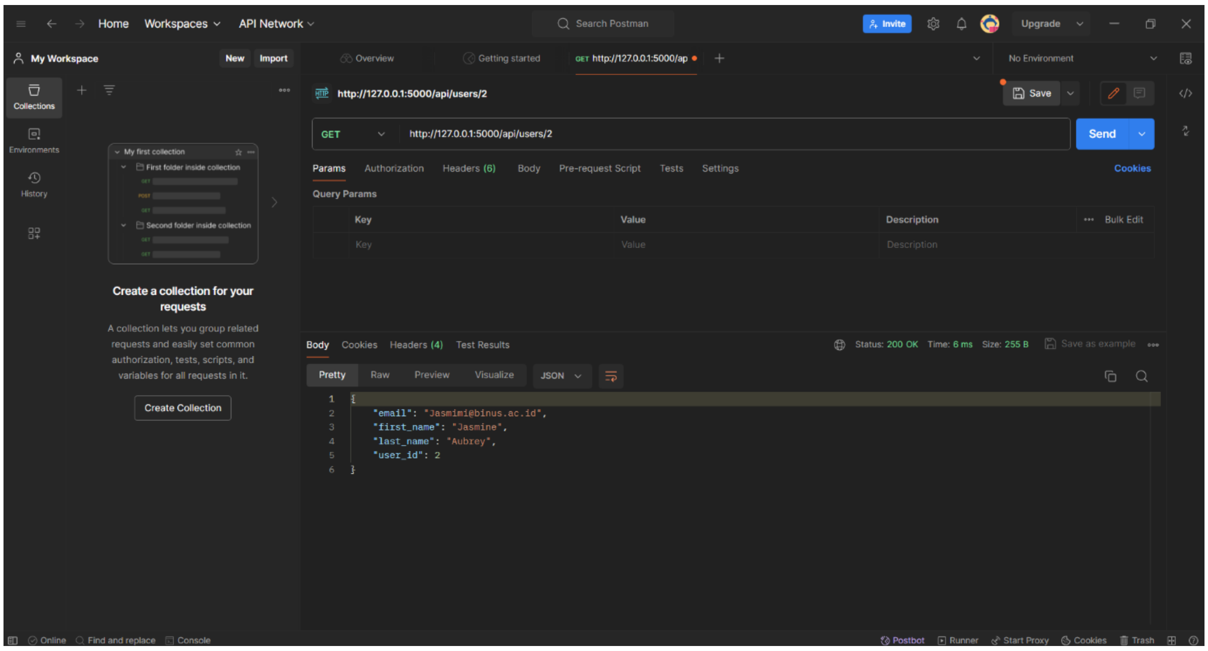
GET http://127.0.0.1:5000/api/users

Status: 200 OK Time: 7 ms Size: 390 B

```
[
    {
        "email": "Mara.Muda@binus.ac.id",
        "first_name": "Mara",
        "last_name": "Muda",
        "user_id": 1
    },
    {
        "email": "Jasmimi@binus.ac.id",
        "first_name": "Jasmine",
        "last_name": "Aubrey",
        "user_id": 2
    }
]
```

- **Retrieving a specific user.**

Using **GET** and specifying the user id in the route '/api/users/<int:user_id>', we can read specific user information. In the screenshot below, we input the user_id as "2" and click send, it will only show data of a specific user_id in the body.



http://127.0.0.1:5000/api/users/2

GET http://127.0.0.1:5000/api/users/2

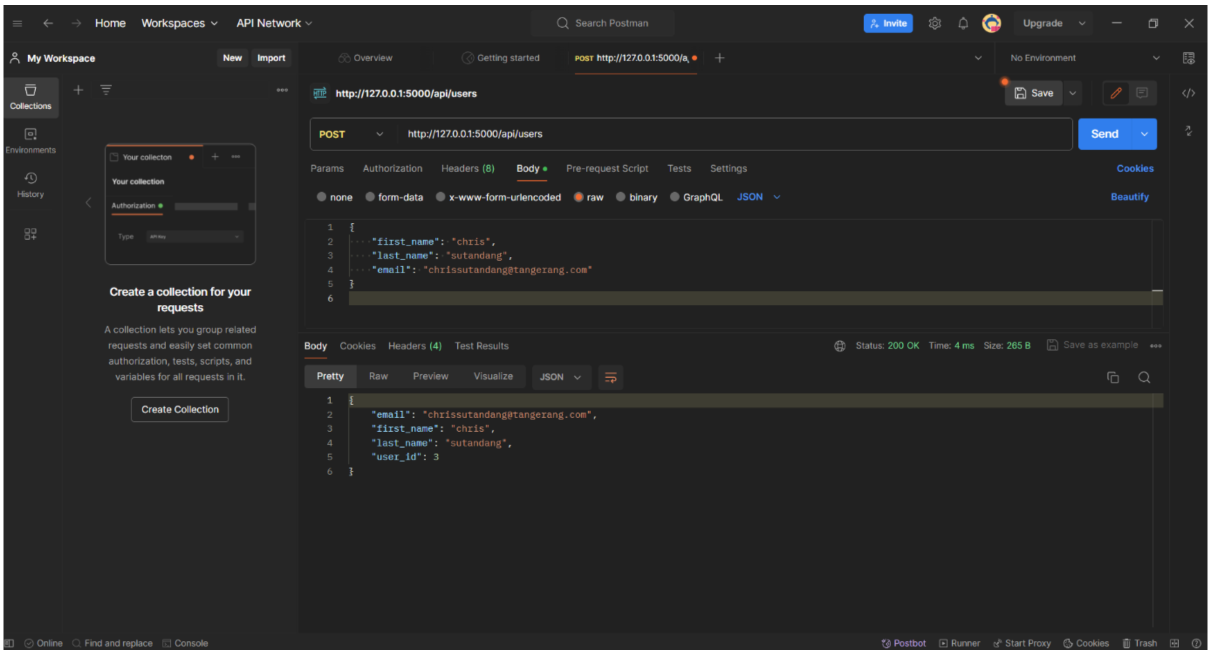
Status: 200 OK Time: 6 ms Size: 255 B

```
{
    "email": "Jasmimi@binus.ac.id",
    "first_name": "Jasmine",
    "last_name": "Aubrey",
    "user_id": 2
}
```

Mara Muda Pohan

- **CREATE a new user.**

To create a new user, we have the change the method to **POST**. After that click on Body, select raw, and choose Json format. To send data we have to use the Json format and click send.



POST http://127.0.0.1:5000/api/users

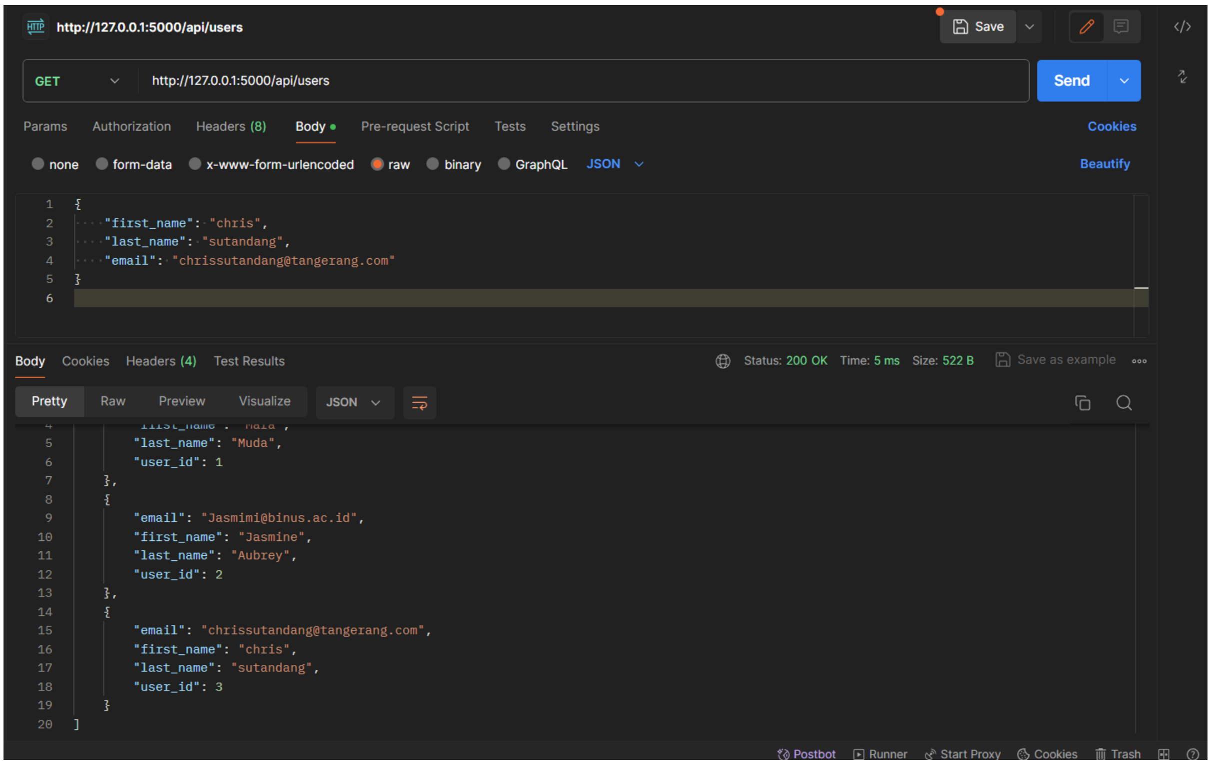
Body: raw, JSON

```
{
    "first_name": "chris",
    "last_name": "sutandang",
    "email": "chrissutandang@tangerang.com"
}
```

Status: 200 OK Time: 4 ms Size: 265 B

```
{
    "email": "chrissutandang@tangerang.com",
    "first_name": "chris",
    "last_name": "sutandang",
    "user_id": 3
}
```

To test if it works, we will use the GET all user function to see if the data is inputted correctly.



GET http://127.0.0.1:5000/api/users

Body: raw, JSON

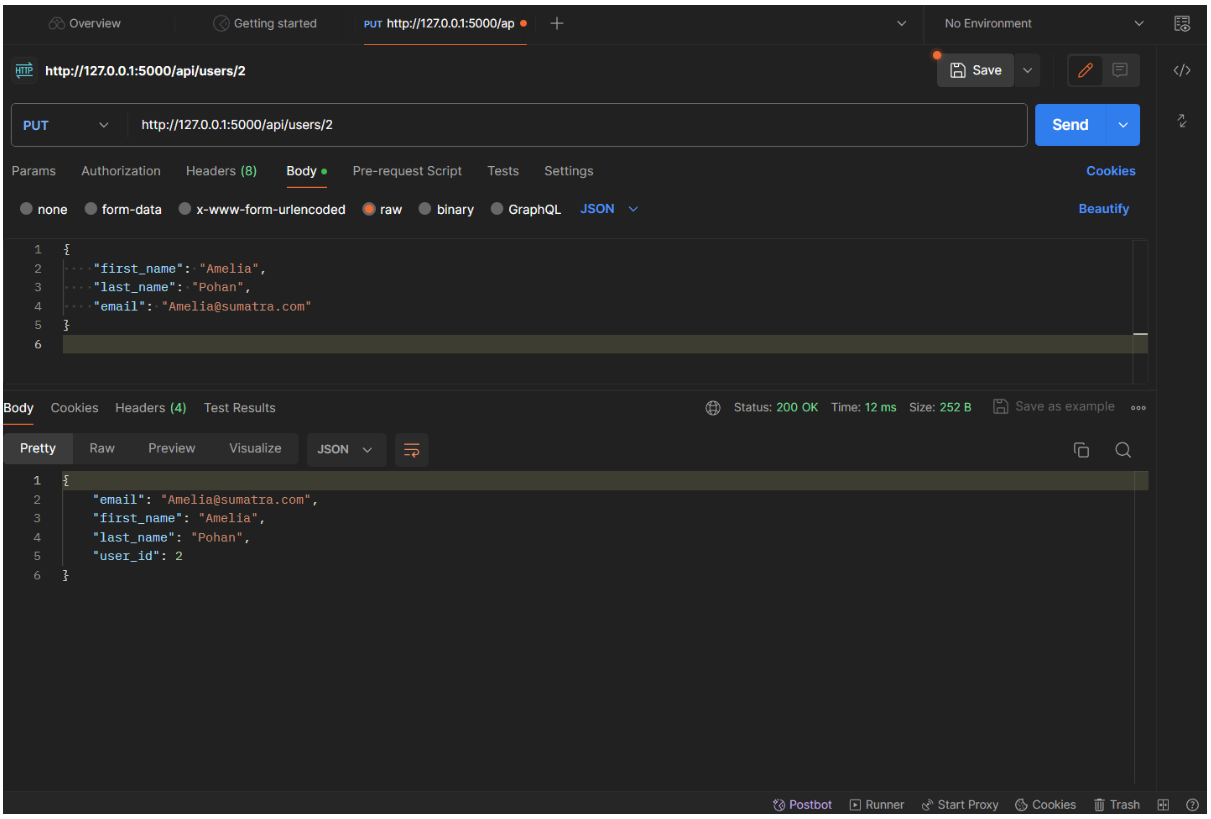
```
{
    "first_name": "chris",
    "last_name": "sutandang",
    "email": "chrissutandang@tangerang.com"
}
```

Status: 200 OK Time: 5 ms Size: 522 B

```
        "last_name": "Muda",
        "user_id": 1
    },
    {
        "email": "Jasmimi@binus.ac.id",
        "first_name": "Jasmine",
        "last_name": "Aubrey",
        "user_id": 2
    },
    {
        "email": "chrissutandang@tangerang.com",
        "first_name": "chris",
        "last_name": "sutandang",
        "user_id": 3
    }
]
```

- **Updating a user.**

To update a user, choose **PUT** method and include the user_id in the route. Then provide the new information and send.



http://127.0.0.1:5000/api/users/2

PUT http://127.0.0.1:5000/api/users/2

none form-data x-www-form-urlencoded raw binary GraphQL JSON

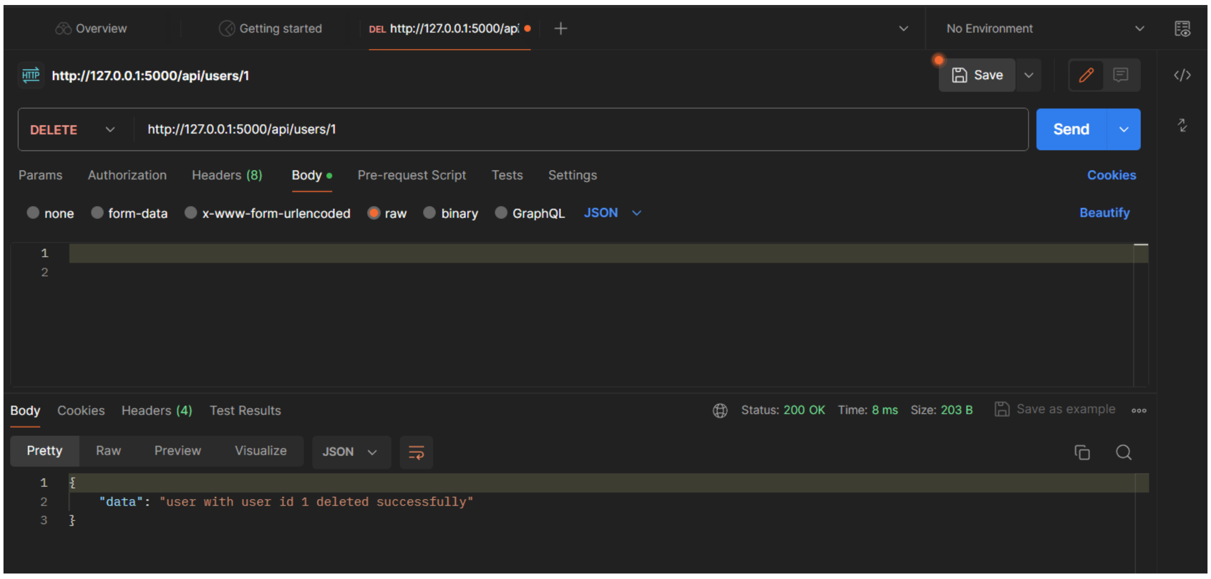
```
{
    "first_name": "Amelia",
    "last_name": "Pohan",
    "email": "Amelia@sumatra.com"
}
```

Body Cookies Headers (4) Test Results   Status: 200 OK Time: 12 ms Size: 252 B

```
{
    "email": "Amelia@sumatra.com",
    "first_name": "Amelia",
    "last_name": "Pohan",
    "user_id": 2
}
```

- **Deleting a user.**

To delete a user, specify the user_id in the route and choose the DELETE method and send.



http://127.0.0.1:5000/api/users/1

DELETE http://127.0.0.1:5000/api/users/1

none form-data x-www-form-urlencoded raw binary GraphQL JSON

Body Cookies Headers (4) Test Results   Status: 200 OK Time: 8 ms Size: 203 B

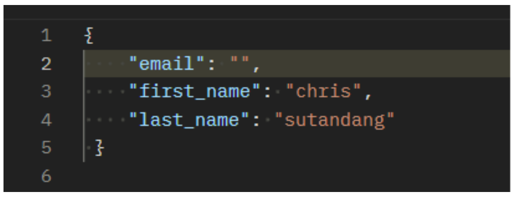
```
{
    "data": "user with user id 1 deleted successfully"
}
```

Mara Muda Pohan

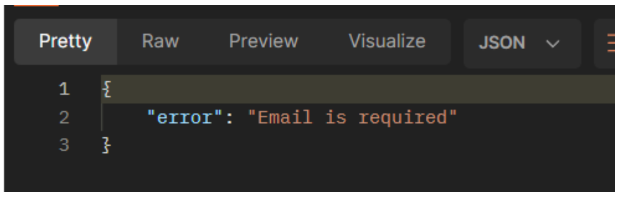
ERROR HANDLING

Here are some examples of data validation and the message that will pop up.

- **No email**

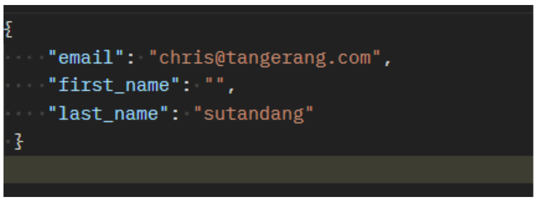


```
{
    "email": "",
    "first_name": "chris",
    "last_name": "sutandang"
}
```

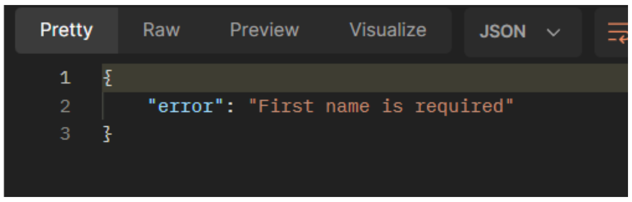


```
{
    "error": "Email is required"
}
```

- **No first_name**

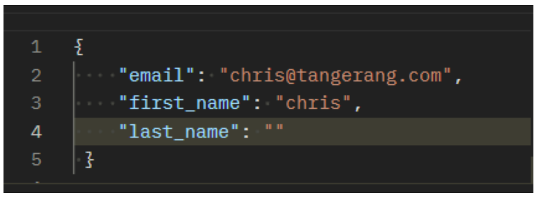


```
{
    "email": "chris@tangerang.com",
    "first_name": "",
    "last_name": "sutandang"
}
```

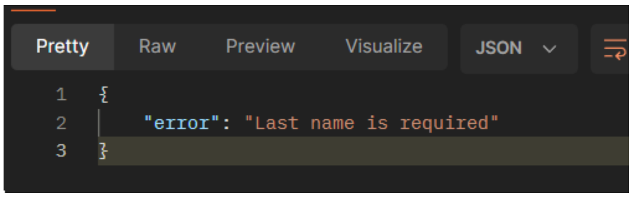


```
{
    "error": "First name is required"
}
```

- **No last_name**

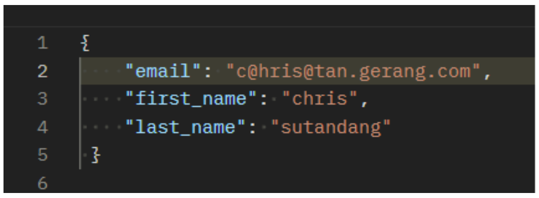


```
{
    "email": "chris@tangerang.com",
    "first_name": "chris",
    "last_name": ""
}
```

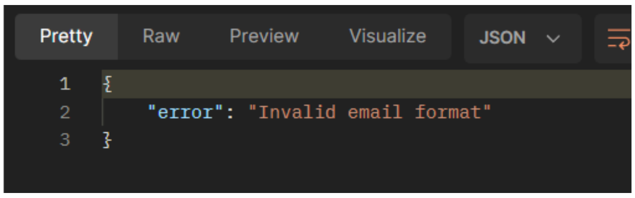


```
{
    "error": "Last name is required"
}
```

- **Wrong email format**



```
{
    "email": "c@hris@tan.gerang.com",
    "first_name": "chris",
    "last_name": "sutandang"
}
```



```
{
    "error": "Invalid email format"
}
```