


# – Task 27 –

REST API Development and Apache Configuration

Name: Basel Amr Barakat  
Email: [baselamr52@gmail.com](mailto:baselamr52@gmail.com)

---


## Project Overview

- Build a REST API using Flask with GET and Post methods
  - Handle JSON responses and errors gracefully
  - Configure Apache Virtual Hosts to serve the API
- 

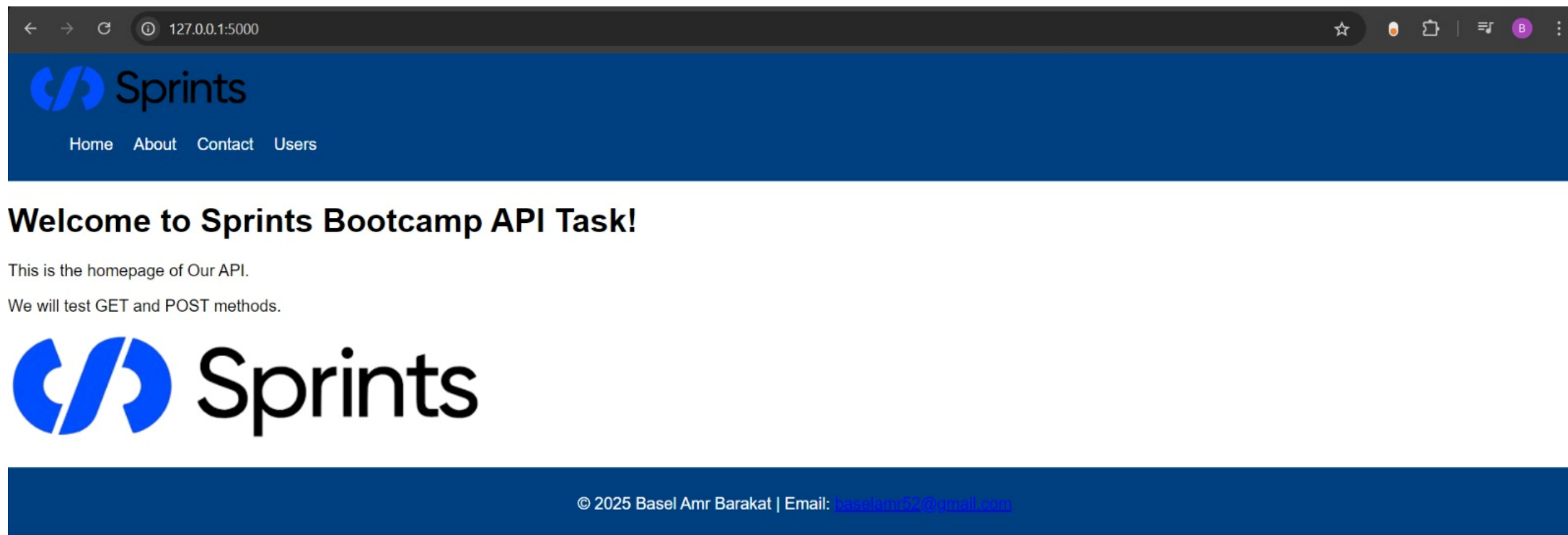
## API Design

- **/users (GET)** : List all users
- **/users (POST)** : Add a new user
- **/users/<id> (DELETE)** : Delete a user
- **/users (Search)** : Search for a user by name or id

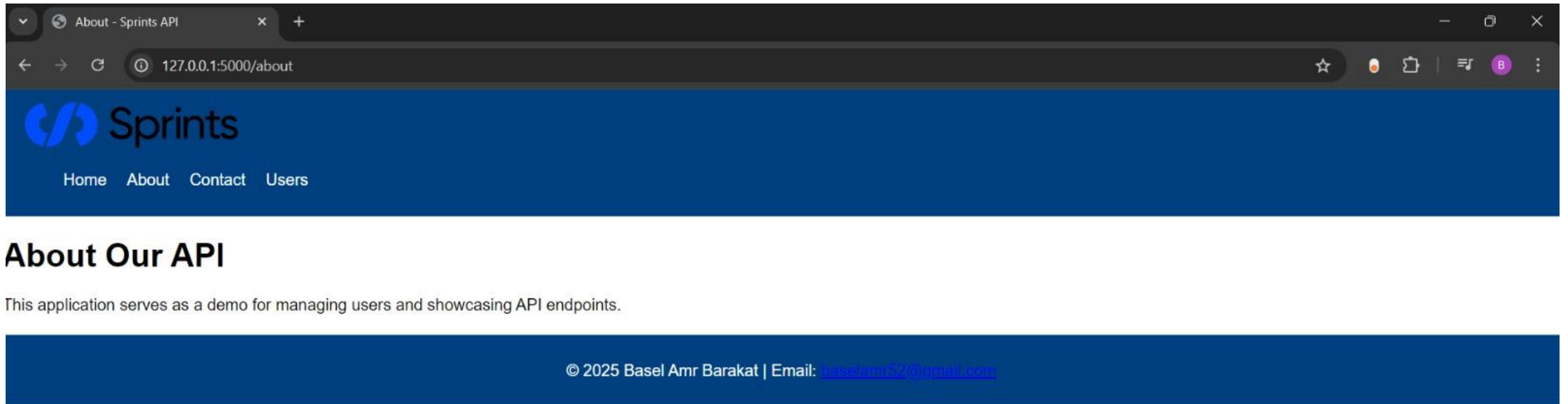
## API Features

- SQLite database integration
  - Input validation for user data (ID, name, email)
  - JSON and HTML response handling
  - Error handling with try-except blocks
- 

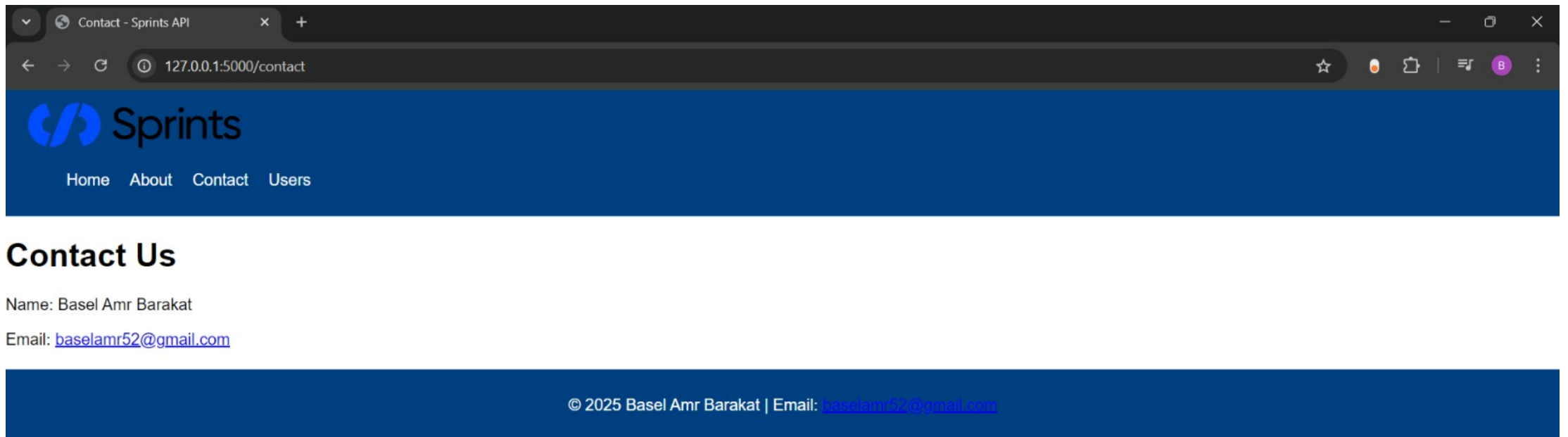
# Home Page



# About Page



# Contact Page



# User List Page

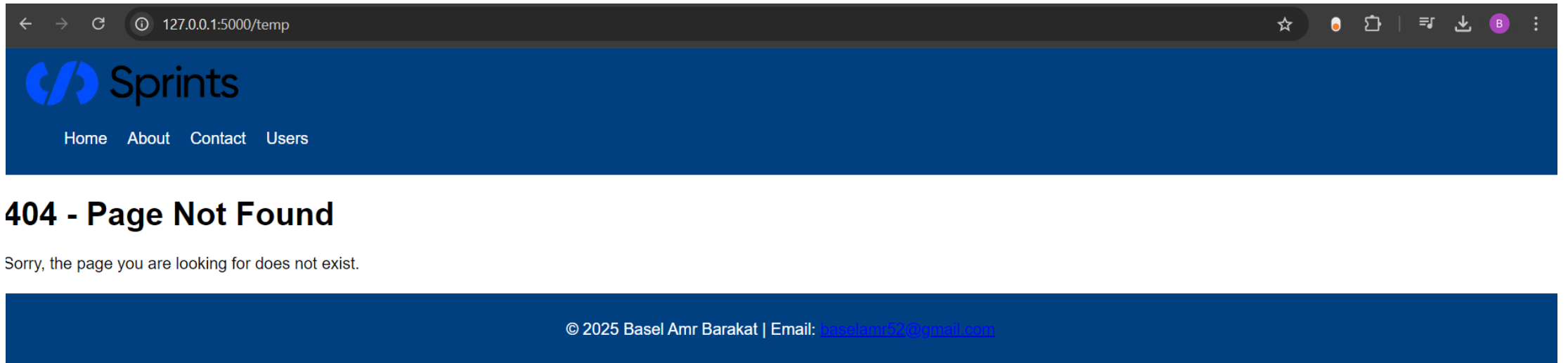


## Users List

ID	Name	Email	Age
1	Basel Amr Barakat	baselamr52@gmail.com	26
2	Aya Amr Barakat	ayaamr@gmail.com	26
3	Mostafa Amr Barakat	mostafa@gmail.com	28
4	Mohamed Khaled	MohamedKhaled@gmail.com	30
12	Omar Ahmed	OmarAhmed@gmail.com	25



# 404 Page



# Some Code Screen Shoots

# 01\_Get\_All\_Users

```
# GET Route: Retrieve all users
@app.route('/users', methods=['GET'])
def get_all_users():
    """
    Users Page Route
    Description: Displays all users in an HTML page or returns JSON data based on the request.
    Query Parameters:
        format (str): 'json' to return JSON response, otherwise returns HTML.
    Returns:
        HTML: User list rendered in a template.
        JSON: User list in JSON format.
    """
    try:
        conn = sqlite3.connect('users.db')
        cursor = conn.cursor()
        cursor.execute("SELECT * FROM users")
        users = cursor.fetchall()
        conn.close()

        # Convert users list to a dictionary format
        user_list = [{"id": user[0], "name": user[1], "email": user[2], "age": user[3]} for user in users]

        # Check if 'format=json' is in the query string
        if request.args.get('format') == 'json':
            return jsonify(user_list), 200
        else:
            return render_template('users.html', users=users)

    except Exception as e:
        return jsonify({"error": str(e)}), 500
```

## 02\_Add\_User

```
# POST Route: Add a new user
@app.route('/users', methods=['POST'])
def add_user():
    """
    Add a new user to the SQLite database.
    Args:
        JSON: A JSON payload containing "id", "name", and "email".
    Description:
        - Validates required fields.
        - Checks for unique 'id' and 'email'.
        - Validates 'name' contains only alphabetic characters.
        - Validates 'email' format.
    Returns:
        JSON: A JSON response containing the status of the operation.
    """
    try:
        new_user = request.json # Get JSON data from the POST request

        # Validate required fields
        if not new_user.get('id') or not new_user.get('name') or not new_user.get('email') or not new_user.get('age'):
            missing_fields = [field for field in ['id', 'name', 'email', 'age'] if field not in new_user]
            return jsonify({"error": f"Missing required field(s): {' '.join(missing_fields)}"}), 400 # 400: Bad Request

        # Validate name format (only letters and spaces)
        if not re.match("^[A-Za-z ]+$", new_user['name']):
            return jsonify({"error": "The 'name' field must only contain alphabetic characters and spaces."}), 400

        # Validate email format
        if not re.match(r"^[^@]+@[^@]+\.[^@]+$", new_user['email']):
            return jsonify({"error": "Invalid email format."}), 400
```

```
# Validate email format
if not re.match(r"^[^@]+@[^@]+\.[^@]+$", new_user['email']):
    return jsonify({"error": "Invalid email format."}), 400

# Check if ID is unique
conn = sqlite3.connect('users.db')
cursor = conn.cursor()
cursor.execute("SELECT id FROM users WHERE id=?", (new_user['id'],))
if cursor.fetchone():
    conn.close()
    return jsonify({"error": "A user with this 'id' already exists."}), 400

# Check if email is unique
cursor.execute("SELECT email FROM users WHERE email=?", (new_user['email'],))
if cursor.fetchone():
    conn.close()
    return jsonify({"error": "A user with this 'email' already exists."}), 400

# Add the new user to the database
cursor.execute("INSERT INTO users (id, name, email, age) VALUES (?, ?, ?, ?)",
              (new_user['id'], new_user['name'], new_user['email'], new_user['age']))
conn.commit()
conn.close()

return jsonify({"message": "User added successfully"}), 201 # 201: Created
except Exception as e:
    return jsonify({"error": str(e)}), 500 # 500: Internal Server Error
```

## 03\_Delete\_User

```
@app.route('/users/<int:id>', methods=[DELETE])
def delete_user(id):
    """
    Delete a user by ID.
    Args:
        id (int): The ID of the user to delete.
    Description:
        This endpoint deletes a user with the specified ID from the database.
    Returns:
        JSON: A JSON response with a success or error message.
    """
    try:
        # Connect to the database
        conn = sqlite3.connect('users.db')
        cursor = conn.cursor()

        # Check if the user exists
        cursor.execute("SELECT * FROM users WHERE id=?", (id,))
        user = cursor.fetchone()
        if not user:
            conn.close()
            return jsonify({"error": f"User with ID {id} not found."}), 404 # 404: Not Found

        # Delete the user
        cursor.execute("DELETE FROM users WHERE id=?", (id,))
        conn.commit()
        conn.close()

        return jsonify({"message": f"User with ID {id} deleted successfully."}), 200 # 200: OK
    except sqlite3.Error as db_error:
        return jsonify({"error": f"Database error: {str(db_error)}"}), 500 # 500: Internal Server Error
    except Exception as e:
        return jsonify({"error": f"Unexpected error: {str(e)}"}), 500 # 500: Internal Server Error
```

## 04\_SearchUser

```
# Search Route: Search for users by name or email
@app.route('/search', methods=['GET'])
def search_users():
    """
    Search for users by name or email.
    Args:
        query (str): The search query parameter.
    Description:
        This endpoint searches for users by name or email using a case-insensitive search.
    Returns:
        JSON: A list of matching users.
    """
    try:
        search_query = request.args.get('query')
        if not search_query:
            return jsonify({"error": "Query parameter is required"}), 400 # 400: Bad Request

        conn = sqlite3.connect('users.db')
        cursor = conn.cursor()
        cursor.execute("SELECT * FROM users WHERE name LIKE ? OR email LIKE ?",
                        ('%' + search_query + '%', '%' + search_query + '%'))
        users = cursor.fetchall()
        conn.close()

        if not users:
            return jsonify({"error": "No users found matching the search criteria"}), 404 # 404: Not Found

        # Convert users list to a dictionary format
        user_list = [{"id": user[0], "name": user[1], "email": user[2], "age": user[3]} for user in users]
        return jsonify(user_list), 200
    except Exception as e:
        return jsonify({"error": str(e)}), 500 # 500: Internal Server Error
```

# Pages Codes

# Error and User Pages

```
# Custom 404 Page
@app.errorhandler(404)
def page_not_found(e):
    """
    404 Error Page Route
    Description: Custom page for 404 errors.
    Returns:
        HTML: 404 error page content.
    """
    return render_template('404.html'), 404

# Users Page
@app.route('/users')
def list_users():
    """
    Users Page Route
    Description: Displays all users in a table format.
    Returns:
        HTML: List of users.
    """
    conn = sqlite3.connect('users.db')
    cursor = conn.cursor()
    cursor.execute("SELECT * FROM users")
    users = cursor.fetchall()
    conn.close()
    return render_template('users.html', users=users)
```



# About and Settings Pages

```
# About Page
@app.route('/about')
def about():
    """
    About Page Route
    Description: Provides information about the application.
    Returns:
    |   HTML: About page content.
    """
    return render_template('about.html')

# Settings Page
@app.route('/settings')
def settings():
    """
    Settings Page Route
    Description: Allows users to adjust application preferences
    Returns:
    |   HTML: Settings page.
    """
    return render_template('settings.html')
```

# Home and Docs Pages

```
# Home Page
@app.route('/')
def home():
    """
    Home Page Route
    Description: Displays a welcome message with an image.
    Returns:
        HTML: Welcome page with an image.
    """
    return render_template('index.html')

# API Documentation Page - /docs
@app.route('/docs')
def docs():
    return render_template('docs.html')
```

# Invalid Pages

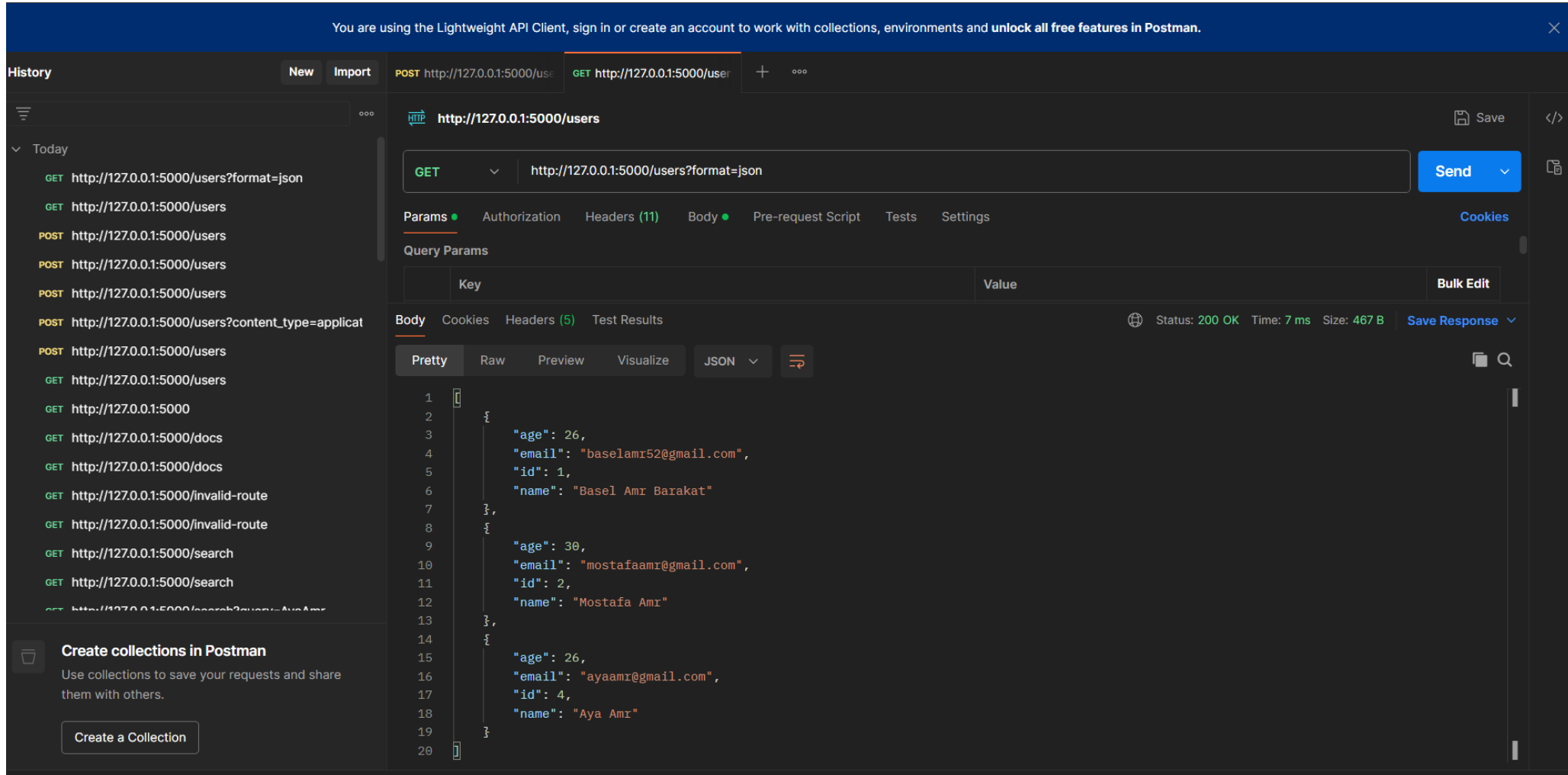
```
# Handle invalid routes (404 error)
@app.errorhandler(404)
def page_not_found(e):
    """
    Handle 404 errors for undefined routes.
    Provides a custom error message and lists valid routes.

    Args:
        e: The exception object for the 404 error.

    Returns:
        JSON: List of valid routes.
    """
    valid_routes = [
        {"method": "GET", "route": "/"},
        {"method": "GET", "route": "/users"},
        {"method": "POST", "route": "/users"},
        {"method": "DELETE", "route": "/users/<id>"},
        {"method": "GET", "route": "/search"}
    ]
    return jsonify({
        "error": "Page not found",
        "message": "The route you entered does not exist.",
        "valid_routes": valid_routes
    }), 404
```

# Testing GET Method

# 1.1 Successfully Getting All Users Postman



You are using the Lightweight API Client, sign in or create an account to work with collections, environments and unlock all free features in Postman.

**History** New Import POST http://127.0.0.1:5000/users GET http://127.0.0.1:5000/users + ...

**Today**

- GET http://127.0.0.1:5000/users?format=json
- GET http://127.0.0.1:5000/users
- POST http://127.0.0.1:5000/users
- POST http://127.0.0.1:5000/users
- POST http://127.0.0.1:5000/users
- POST http://127.0.0.1:5000/users?content\_type=applicat
- POST http://127.0.0.1:5000/users
- GET http://127.0.0.1:5000/users
- GET http://127.0.0.1:5000
- GET http://127.0.0.1:5000/docs
- GET http://127.0.0.1:5000/docs
- GET http://127.0.0.1:5000/invalid-route
- GET http://127.0.0.1:5000/invalid-route
- GET http://127.0.0.1:5000/search
- GET http://127.0.0.1:5000/search
- GET http://127.0.0.1:5000/search?query=Aya Amr

**Create collections in Postman**  
Use collections to save your requests and share them with others.  
[Create a Collection](#)

**GET** http://127.0.0.1:5000/users?format=json Send Save </>

**Params** **Authorization** **Headers (11)** **Body** **Pre-request Script** **Tests** **Settings** **Cookies**

**Query Params**

Key	Value
-----	-------

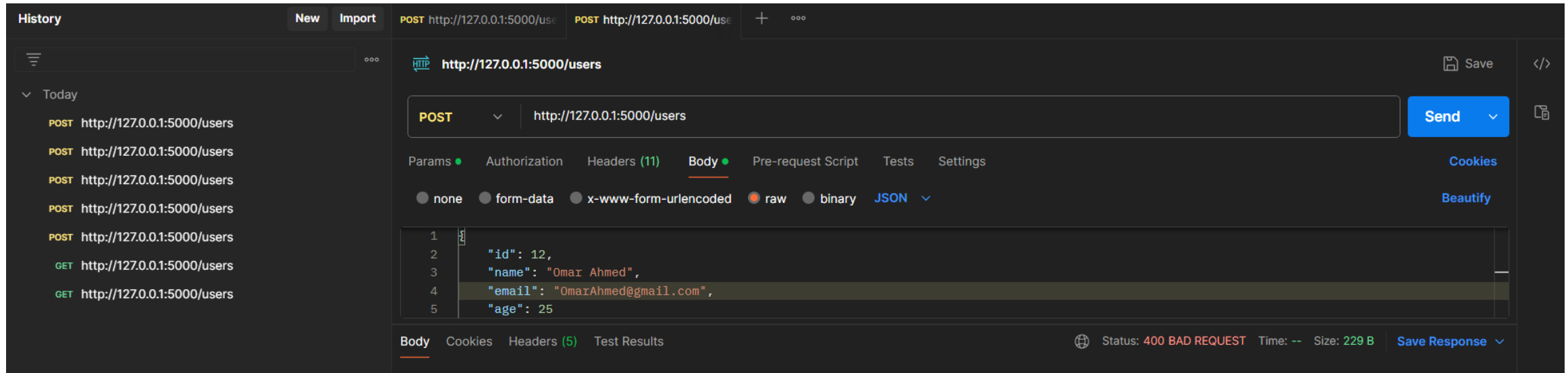
**Body** **Cookies** **Headers (5)** **Test Results** Status: 200 OK Time: 7 ms Size: 467 B Save Response

**Pretty** **Raw** **Preview** **Visualize** **JSON** ↕

```
1 {
2   {
3     "age": 26,
4     "email": "baselamr52@gmail.com",
5     "id": 1,
6     "name": "Basel Amr Barakat"
7   },
8   {
9     "age": 30,
10    "email": "mostafaamr@gmail.com",
11    "id": 2,
12    "name": "Mostafa Amr"
13  },
14  {
15    "age": 26,
16    "email": "ayaamr@gmail.com",
17    "id": 4,
18    "name": "Aya Amr"
19  }
20 }
```

# Testing POST Method

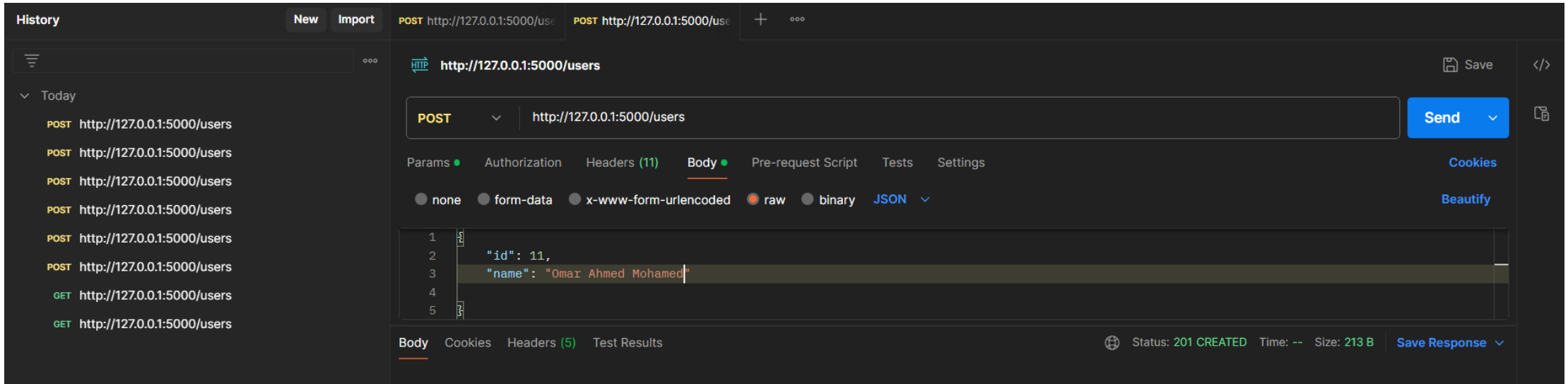
## 2.1 Successful User Creation



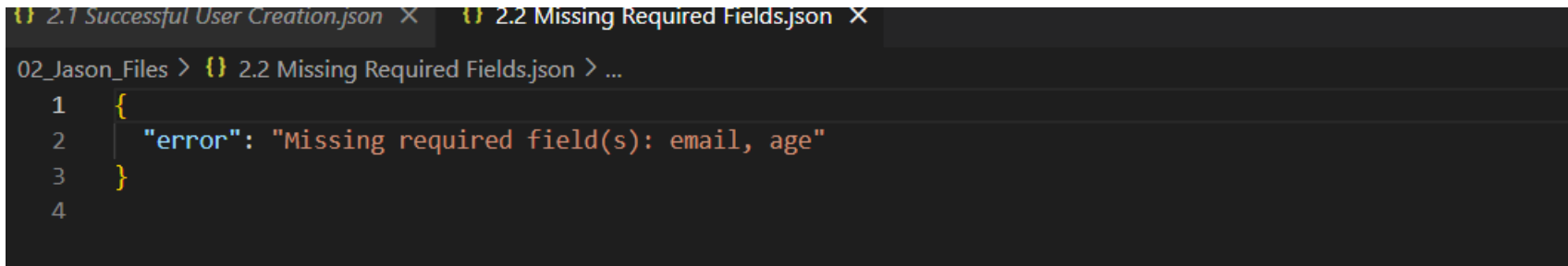
The screenshot shows a REST client interface with a history panel on the left and a main editor on the right. The history panel lists several POST requests to `http://127.0.0.1:5000/users` and one GET request. The main editor shows a POST request to the same URL. The request body is a JSON object: `{ "id": 12, "name": "Omar Ahmed", "email": "OmarAhmed@gmail.com", "age": 25 }`. The status bar at the bottom indicates a `400 BAD REQUEST` response.

```
02_Jason_Files > {} 2.1 Successful User Creation.json > ...
1  {
2    "message": "User added successfully"
3  }
4
```

## 2.2 Missing Required Fields



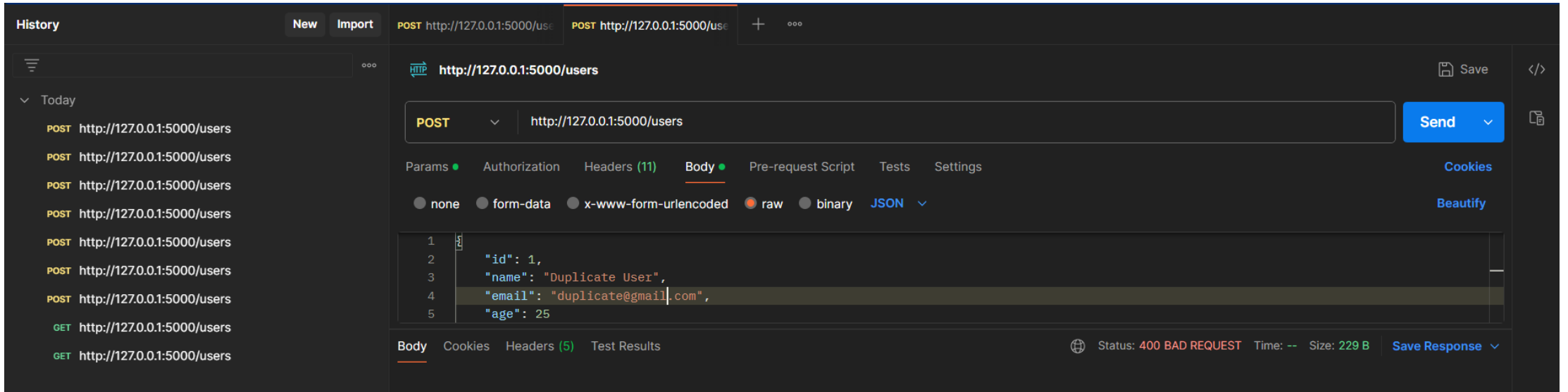
The screenshot shows a REST client interface with a history panel on the left and a main editor on the right. The history panel lists several POST requests to `http://127.0.0.1:5000/users` and one GET request. The main editor shows a POST request to the same URL with a JSON body: `{ "id": 11, "name": "Omar Ahmed Mohamed" }`. The status bar at the bottom indicates the request was successful with a 201 CREATED status.



```
{ "error": "Missing required field(s): email, age" }
```



## 2.3 Duplicate ID



The screenshot shows a REST client interface with a history panel on the left and a main editor on the right. The history panel lists several POST requests to `http://127.0.0.1:5000/users`. The main editor shows a POST request to the same URL. The request body is a JSON object:

```
1 {  
2   "id": 1,  
3   "name": "Duplicate User",  
4   "email": "duplicate@gmail.com",  
5   "age": 25  
}
```

The response status is `400 BAD REQUEST`. The response body is not visible.

02\_Jason\_Files > {} 2.3 Duplicate ID.json > ...

```
1 {  
2   "error": "A user with this 'id' already exists."  
3 }  
4
```

## 2.4 Duplicate Email

You are using the Lightweight API Client, sign in or create an account to work with collections, environments and unlock all free features in Postman.

History New Import POST http://127.0.0.1:5000/users POST http://127.0.0.1:5000/users + ...

HTTP http://127.0.0.1:5000/users Save </>

POST http://127.0.0.1:5000/users Send ⌵ ⌵

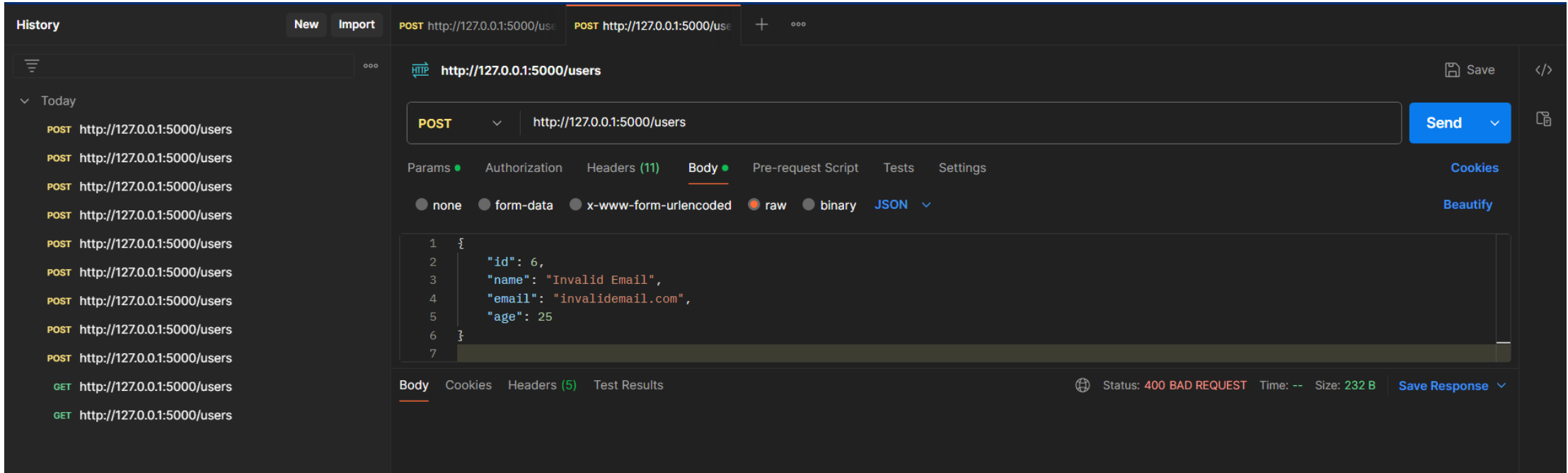
Params Authorization Headers (11) Body Pre-request Script Tests Settings Cookies Beautify

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

```
1 {
2   "id": 52,
3   "name": "Duplicate Email",
4   "email": "baselamr52@gmail.com",
5   "age": 25
6 }
7
```

```
02_Jason_Files > {} 2.4 Duplicate Email.json > ...
1 {
2   "error": "A user with this 'email' already exists."
3 }
4
```

## 2.5 Invalid Email Format



The screenshot shows a REST client interface with a history panel on the left and a main editor on the right. The history panel lists several POST requests to `http://127.0.0.1:5000/users`. The main editor shows a POST request to the same URL with a JSON body. The status bar at the bottom indicates a **400 BAD REQUEST** error.

**History:**

- POST `http://127.0.0.1:5000/users`
- POST `http://127.0.0.1:5000/users`
- POST `http://127.0.0.1:5000/users`
- POST `http://127.0.0.1:5000/users`
- POST `http://127.0.0.1:5000/users`
- POST `http://127.0.0.1:5000/users`
- POST `http://127.0.0.1:5000/users`
- GET `http://127.0.0.1:5000/users`
- GET `http://127.0.0.1:5000/users`

**Request Details:**

- Method: POST
- URL: `http://127.0.0.1:5000/users`
- Body (JSON):

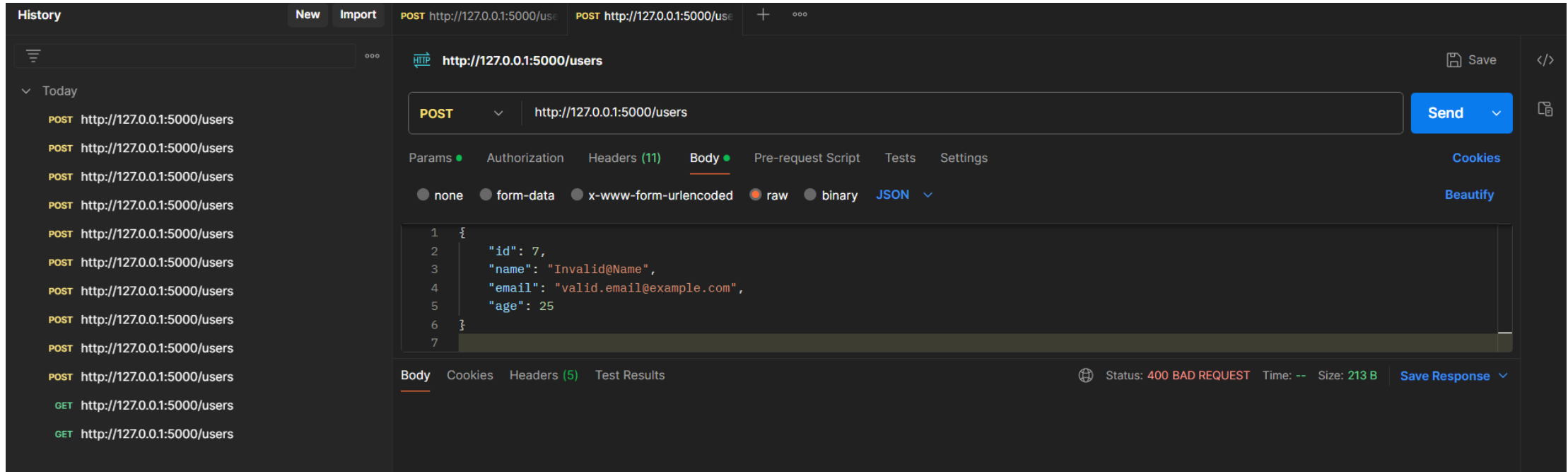
```
1 {
2   "id": 6,
3   "name": "Invalid Email",
4   "email": "invalidemail.com",
5   "age": 25
6 }
```

**Response:**

- Status: 400 BAD REQUEST
- Time: --
- Size: 232 B
- Save Response

```
02_Jason_Files > {} 2.4 Duplicate Email.json > ...
1 {
2   "error": "A user with this 'email' already exists."
3 }
4
```

## 2.6 Invalid Name Format



The screenshot shows a REST client interface with a history panel on the left and a main editor on the right. The history panel lists several POST requests to `http://127.0.0.1:5000/users` and two GET requests. The main editor shows a POST request to the same URL. The request body is a JSON object:

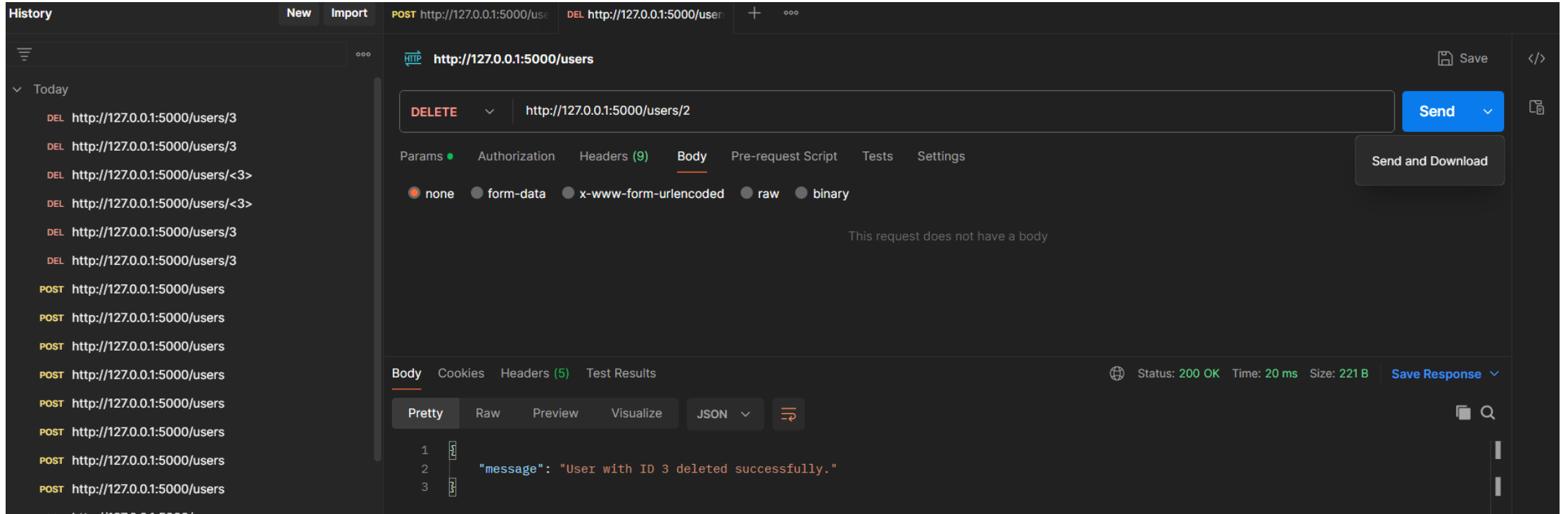
```
{
  "id": 7,
  "name": "Invalid@Name",
  "email": "valid.email@example.com",
  "age": 25
}
```

The request is set to the `JSON` format. The response status is `400 BAD REQUEST`, and the response size is `213 B`. The response body is not visible.

```
02_Jason_Files > {} 2.6 Invalid Name Format.json > ...
1  {
2    "error": "The 'name' field must only contain alphabetic characters and spaces."
3  }
4
```

# Testing DELETE Method

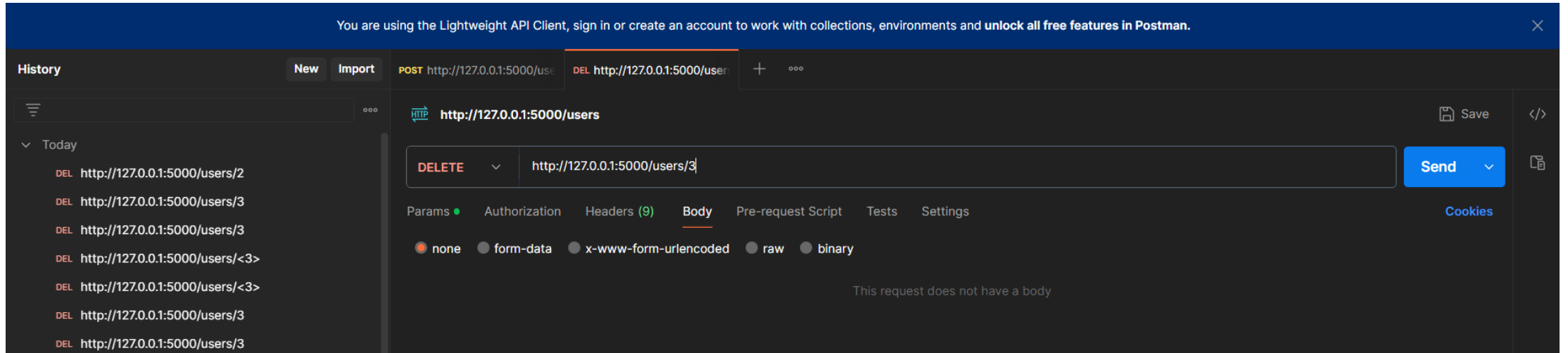
## 3.1 Successful Deletion



The screenshot shows the Postman interface with a DELETE request to `http://127.0.0.1:5000/users/2`. The request is configured with the method `DELETE` and the URL `http://127.0.0.1:5000/users/2`. The response is displayed in the Body tab, showing a JSON message: `"message": "User with ID 3 deleted successfully."`. The status is `200 OK`, the time is `20 ms`, and the size is `221 B`. The left sidebar shows a history of requests, including several DELETE requests to `http://127.0.0.1:5000/users/3` and POST requests to `http://127.0.0.1:5000/users`.

```
02_Jason_Files > {} 3.1 Successful Deletion.json > ...
1  {
2    "message": "User with ID 2 deleted successfully."
3  }
4
```

## 3.2 User Not Found

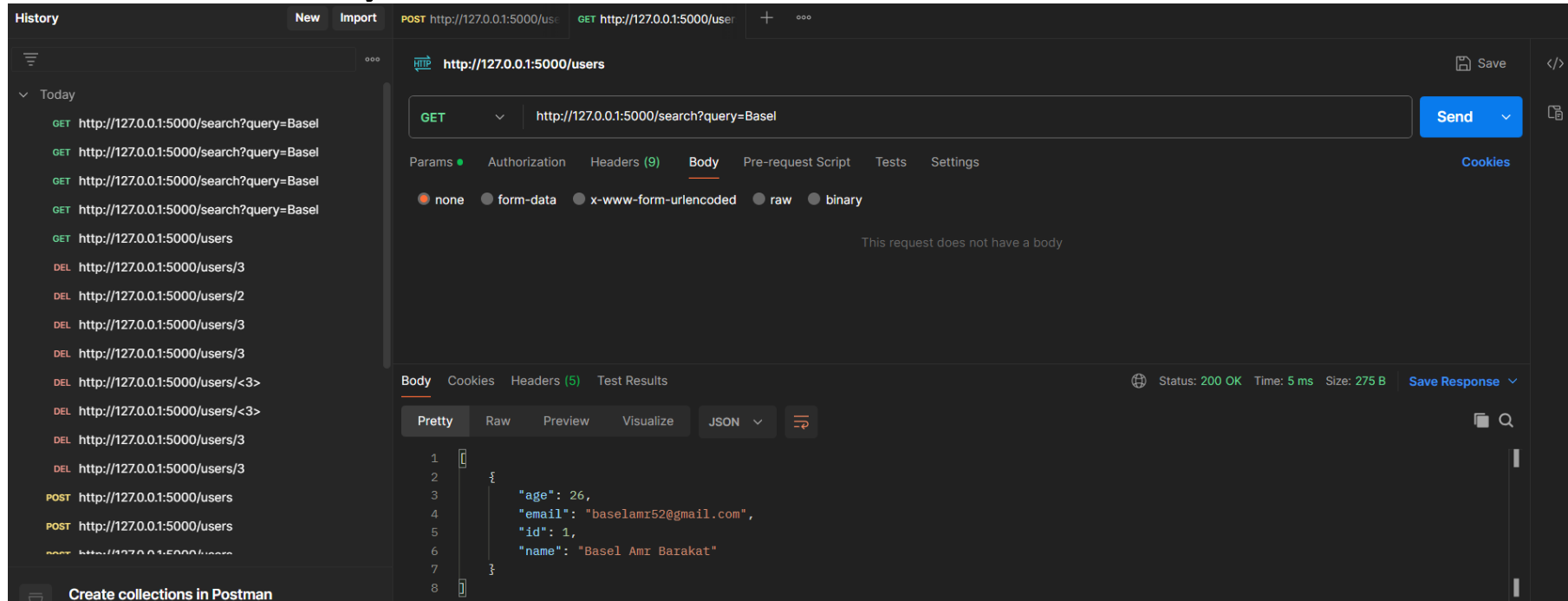


```
02_Jason_Files > {} 3.2 User Not Found.json > ...
1  {
2    "error": "User with ID 3 not found."
3  }
4
```

# Testing SEARCH Method



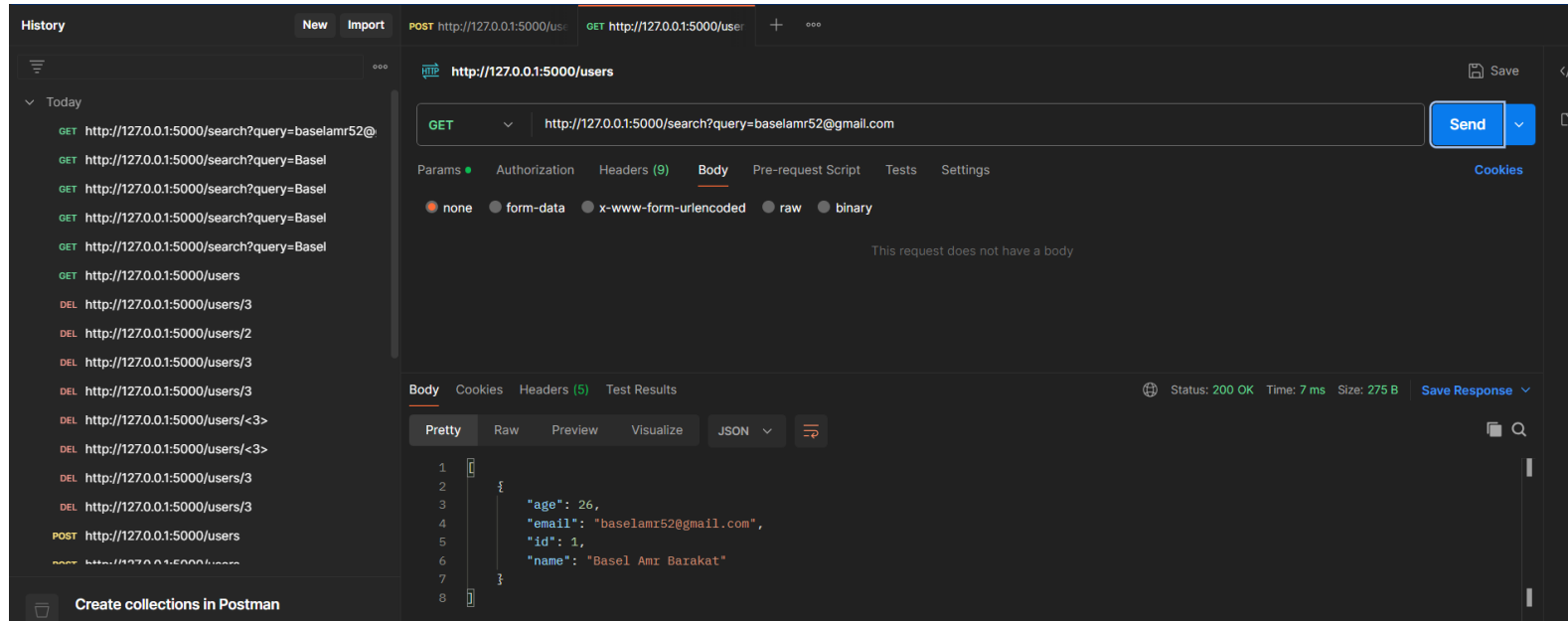
## 4.1 Search by Name



02\_Jason\_Files > {} 4.1 Search by Name.json > ...

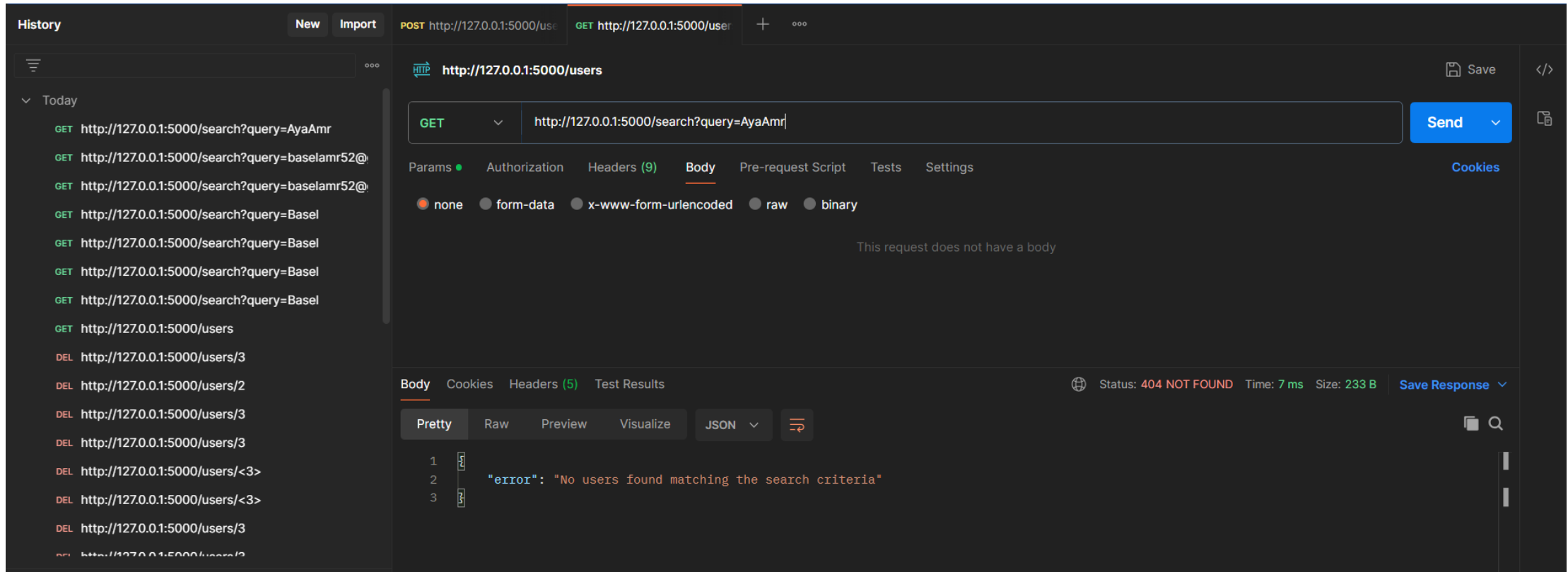
```
1  [
2    {
3      "age": 26,
4      "email": "baselamr52@gmail.com",
5      "id": 1,
6      "name": "Basel Amr Barakat"
7    }
8  ]
9
```

## 4.2 Search by Email



```
02_Jason_Files > {} 4.2 Search by Email.json > ...  
1  [  
2    {  
3      "age": 26,  
4      "email": "baselamr52@gmail.com",  
5      "id": 1,  
6      "name": "Basel Amr Barakat"  
7    }  
8  ]  
9
```

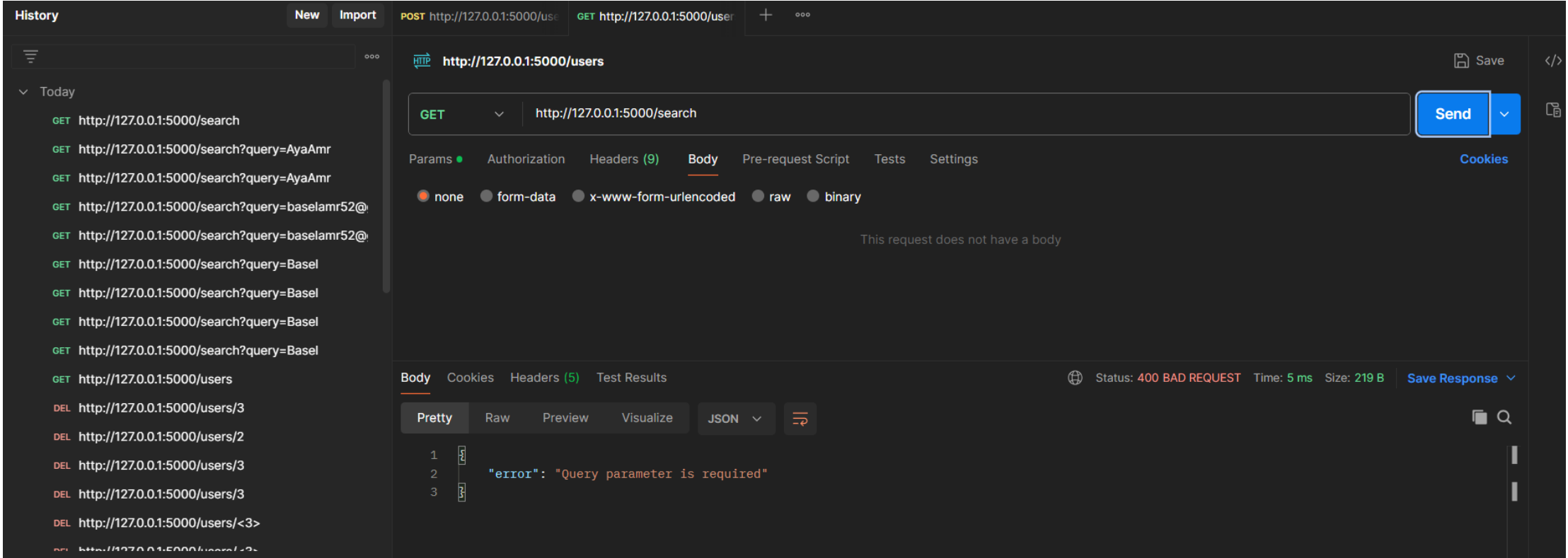
## 4.3 No Results Found



The screenshot shows a REST client interface with a dark theme. On the left is a 'History' panel with a list of requests, including several GET requests to a search endpoint and several DELETE requests to a users endpoint. The main panel shows a GET request to `http://127.0.0.1:5000/search?query=AyaAmr`. The 'Body' tab is selected, showing a message: 'This request does not have a body'. Below this, the 'Body' tab is also selected, showing a JSON response in 'Pretty' format: `{ "error": "No users found matching the search criteria" }`. The status bar at the bottom indicates 'Status: 404 NOT FOUND', 'Time: 7 ms', and 'Size: 233 B'.

```
02_Jason_Files > {} 4.3 No Results Found.json > ...
1  {
2    "error": "No users found matching the search criteria"
3  }
4
```

## 4.4 Missing Query Parameter



The screenshot shows a REST client interface with a history panel on the left and a main workspace. The history panel lists several GET requests to `http://127.0.0.1:5000/search` with various query parameters, followed by several DELETE requests to `http://127.0.0.1:5000/users/3`. The main workspace shows a GET request to `http://127.0.0.1:5000/search` without any query parameters. The response status is `400 BAD REQUEST` with a message: `"error": "Query parameter is required"`.

History:

- GET `http://127.0.0.1:5000/search`
- GET `http://127.0.0.1:5000/search?query=AyaAmr`
- GET `http://127.0.0.1:5000/search?query=AyaAmr`
- GET `http://127.0.0.1:5000/search?query=baselamr52@`
- GET `http://127.0.0.1:5000/search?query=baselamr52@`
- GET `http://127.0.0.1:5000/search?query=Basel`
- GET `http://127.0.0.1:5000/search?query=Basel`
- GET `http://127.0.0.1:5000/search?query=Basel`
- GET `http://127.0.0.1:5000/search?query=Basel`
- GET `http://127.0.0.1:5000/users`
- DEL `http://127.0.0.1:5000/users/3`
- DEL `http://127.0.0.1:5000/users/2`
- DEL `http://127.0.0.1:5000/users/3`
- DEL `http://127.0.0.1:5000/users/3`
- DEL `http://127.0.0.1:5000/users/<3>`
- DEL `http://127.0.0.1:5000/users/<3>`

Current Request:

GET `http://127.0.0.1:5000/search`

Params: none (selected), form-data, x-www-form-urlencoded, raw, binary

Body: This request does not have a body

Response:

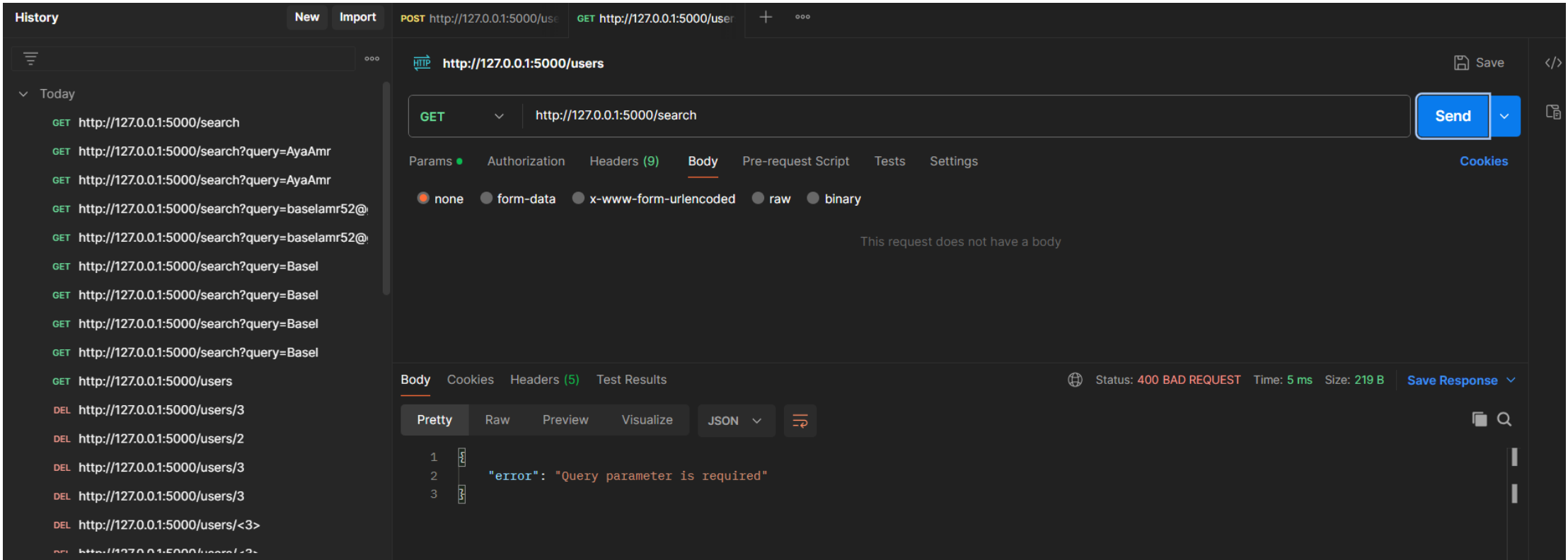
Status: `400 BAD REQUEST` Time: 5 ms Size: 219 B

Body (Pretty):

```
1 {
2   "error": "Query parameter is required"
3 }
```

```
02_Jason_Files > {} 4.4 Missing Query Parameter.json > ...
1 {
2   "error": "Query parameter is required"
3 }
4
```

## 4.4 Missing Query Parameter



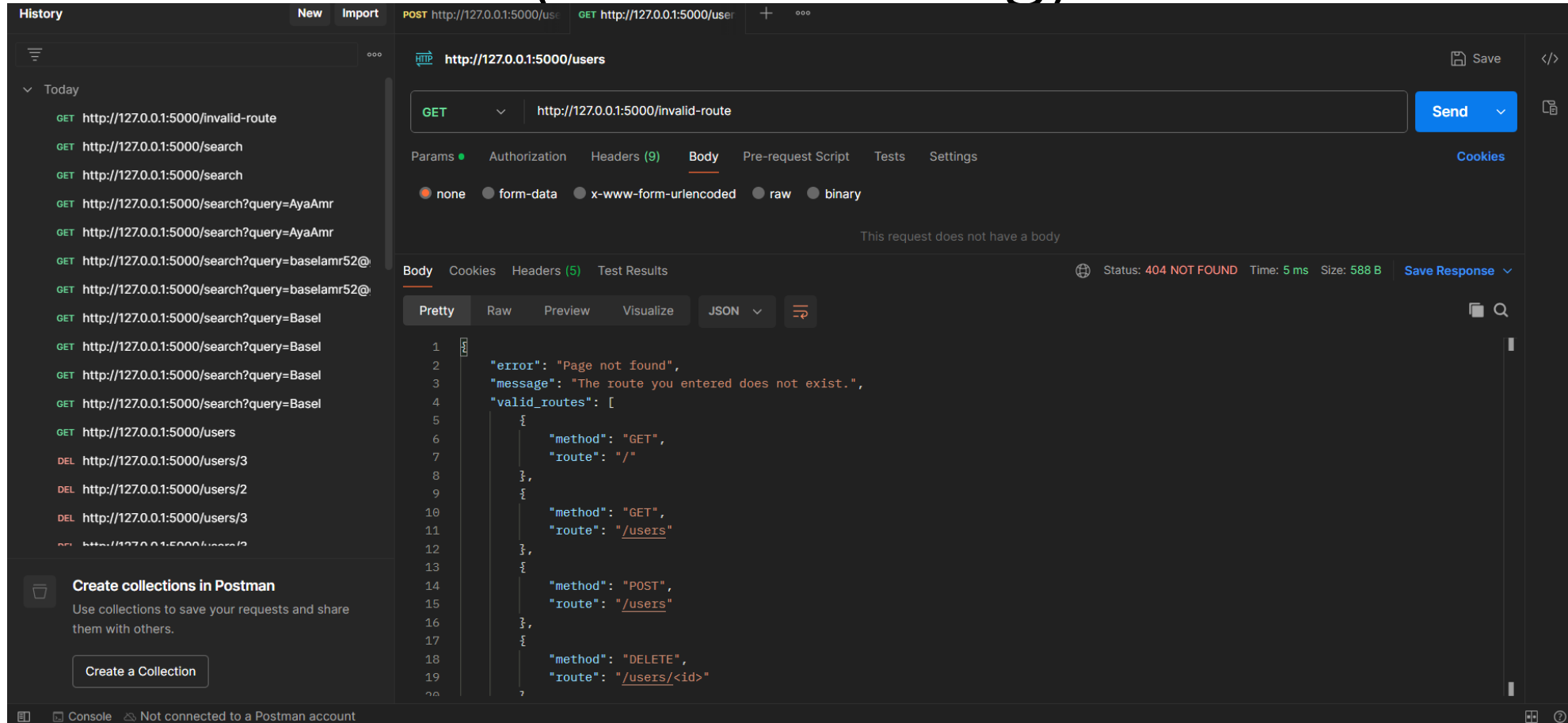
The screenshot shows the Postman interface. On the left, the 'History' tab lists several requests. The main panel shows a GET request to `http://127.0.0.1:5000/search`. The 'Body' tab is selected, and it displays the message 'This request does not have a body'. The status bar at the bottom indicates a **400 BAD REQUEST** with a response time of 5 ms and a size of 219 B. The response body is shown in JSON format:

```
{
  "error": "Query parameter is required"
}
```

```
02_Jason_Files > {} 4.4 Missing Query Parameter.json > ...
1  {
2    "error": "Query parameter is required"
3  }
4
```

Invalid Route

## 5. Invalid Route (404 Handling)



The image shows the Postman application interface. On the left, the 'History' panel lists several requests, including a GET request to `http://127.0.0.1:5000/invalid-route`. The main panel displays the details of the selected request, which is a GET request to `http://127.0.0.1:5000/invalid-route`. The 'Body' tab is selected, showing a JSON response with the following structure:

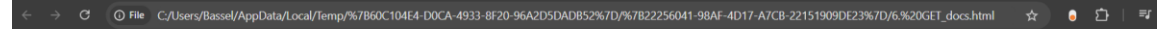
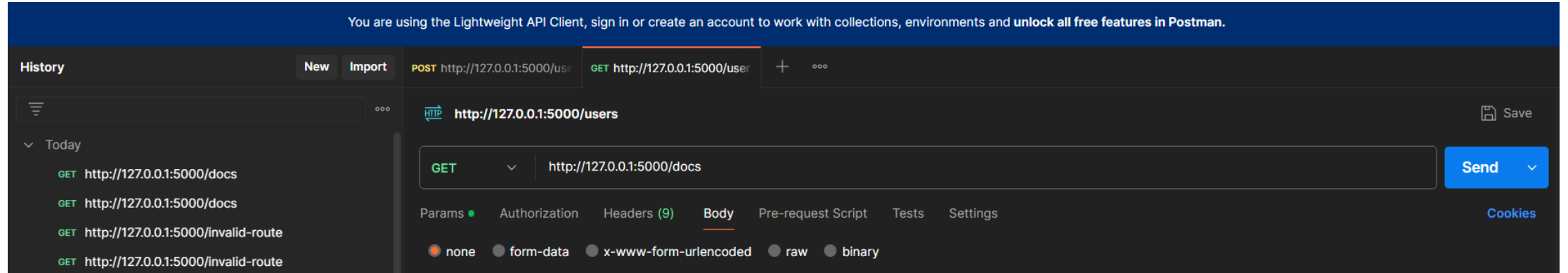
```
1 {
2   "error": "Page not found",
3   "message": "The route you entered does not exist.",
4   "valid_routes": [
5     {
6       "method": "GET",
7       "route": "/"
8     },
9     {
10      "method": "GET",
11      "route": "/users"
12    },
13    {
14      "method": "POST",
15      "route": "/users"
16    },
17    {
18      "method": "DELETE",
19      "route": "/users/<id>"
20    }
21  ]
22 }
```

The status bar at the bottom indicates a 404 NOT FOUND response with a time of 5 ms and a size of 588 B. The bottom status bar also shows 'Console' and 'Not connected to a Postman account'.

# API Documentation



# 6. GET\_docs



6. GET\_docs.html

## API Documentation

### GET /users

Retrieve all users in the system.

```
Response:
[
  {
    "id": 1, "name": "Basel Amr Barakat", "email": "baselamr52@gmail.com", "age": 26,
    "id": 2, "name": "Aya Amr Barakat", "email": "ayaamr@gmail.com", "age": 26,
    "id": 3, "name": "Mostafa Amr Barakat", "email": "mostafa@gmail.com", "age": 28,
    "id": 4, "name": "Mohamed Amr Barakat", "email": "mohamed@gmail.com", "age": 32,
  }
]
```

### POST /users

Add a new user to the system. Send JSON with user data.

```
Request:
{
  "id": 5,
  "name": "Amr Barakat",
  "email": "amrbarakat@gmail.com"
}
Response:
{
  "message": "User added successfully"
}
```

# Setting up Flask with Apache using WGSII

# 1. Prerequisites for WSGI Deployment

- Install Apache for windows
- Install `mod\_wsgi` for Apache
- Verify installation by checking mod\_wsgi version

```
C:\Windows\System32>mod_wsgi-express module-config
LoadFile "C:/Users/Bassel/AppData/Local/Programs/Python/Python313/python313.dll"
LoadModule wsgi_module "C:/Users/Bassel/AppData/Local/Programs/Python/Python313/Lib/site-packages/mod_wsgi/server/mod_wsgi.cp313-win_amd64.pyd"
WSGIPythonHome "C:/Users/Bassel/AppData/Local/Programs/Python/Python313"
```

2. Open the http.conf file located in the conf directory of my Apache installation and add the following lines

```
#1. Add Server Name
ServerName localhost:80
```

```
#2. Load configuration details
LoadFile "C:/Users/Bassel/AppData/Local/Programs/Python/Python313/python313.dll"
LoadModule wsgi_module "C:/Users/Bassel/AppData/Local/Programs/Python/Python313/Lib/site-packages/mod_wsgi/server/mod_wsgi.cp313-win_amd64.pyd"
WSGIPythonHome "C:/Users/Bassel/AppData/Local/Programs/Python/Python313"
```

### 3. Create a WSGI File

- 1. Create a file named flaskapp.wsgi in the same directory as our api.py file
- 2. Add the following content

```
1 import sys
2 import os
3 from 27_BaselAmr_Task3_API import app as application
4
5 # Add the project directory to the Python path
6 sys.path.insert(0, os.path.dirname(__file__))
7
```

## 4. Setting Up Apache Virtual Host

- 1. Open the httpd-vhost-conf file located in the conf/extra directory of our Apache installation
- 2. Add the following content

```
<VirtualHost *:80>
    ServerName firstflaskapp.localhost
    DocumentRoot "D:/Education/Courses/Sprints/AL_and_ML_Sprints_BootCamp/04-Tasks/05_FromDatatoAI/27_BaselAmr_Task3_API"
    WSGIScriptAlias / "D:/Education/Courses/Sprints/AL_and_ML_Sprints_BootCamp/04-Tasks/05_FromDatatoAI/27_BaselAmr_Task3_API/api.wsgi"

    <Directory "D:/Education/Courses/Sprints/AL_and_ML_Sprints_BootCamp/04-Tasks/05_FromDatatoAI/27_BaselAmr_Task3_API">
        Require all granted
    </Directory>

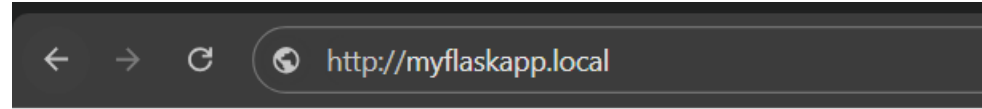
    ErrorLog "logs/flaskapp-error.log"
    CustomLog "logs/flaskapp-access.log" common
</VirtualHost>
```

## 5. Updating the host file

- 1. Open the hosts file with administrative privileges
- 2. Add a line to map our hostname to myflask.local

```
# localhost name resolution is handled within DNS itself.  
127.0.0.1 myflaskapp.local|
```

6. Test the hostname by opening the browser and navigation to `http://myflaskapp.local`



**It works!**

7. Testing Users



## Users List

ID	Name	Email	Age
1	Basel Amr Barakat	baselamr52@gmail.com	26
2	Aya Amr Barakat	ayaamr@gmail.com	26
3	Mostafa Amr Barakat	mostafa@gmail.com	28
4	Mohamed Khaled	MohamedKhaled@gmail.com	30
12	Omar Ahmed	OmarAhmed@gmail.com	25