Lab Instruction: Building Your Own Python-Based API Gateway

Objective:

The goal of this lab is to design and implement a basic Python-based API Gateway using Flask, which will interact with multiple microservices. You will implement essential features such as routing, rate limiting, logging, authentication, error handling, caching, and load balancing. By the end of this lab, you will understand how an API Gateway functions and how to customize it for real-world use cases.

Prerequisites:

1. Basic understanding of Python and Flask.
2. Python installed on your system.
3. Basic understanding of HTTP methods (GET, POST).
4. Familiarity with RESTful APIs.

Lab Setup:

1. Install Flask and Required Packages
Open your terminal/command prompt and install Flask, requests, and other necessary dependencies using pip:

```
pip install Flask requests
```

2. Create Project Structure
Create a directory for your lab project, e.g., python-api-gateway-lab and inside it, create subfolders for your gateway and two microservices:

```
python-api-gateway-lab/
├── gateway/
├── service1/
└── service2/
```

3. Create Microservices

- Service 1 (service1/app.py)
- Service 2 (service2/app.py)

4. Create the API Gateway
The API Gateway will route requests to different services and implement key functionalities like rate-limiting, logging, error handling, and more.

Step-by-Step Lab Instructions:

1. **Start the Microservices**
- Open two terminal windows.
- In one terminal, navigate to `service1/` and run the service:
```bash
python app.py
```
- In the other terminal, navigate to `service2/` and run the service:
```bash
python app.py
```
- The services will run on ports 5001 and 5002 respectively.

2. **Start the API Gateway**
- In another terminal, navigate to `gateway/` and run the API Gateway:
```bash
python app.py
```
This will start the gateway on port 5000.

3. **Test the API Gateway**
- Open your browser or use **Postman** or **curl** to test the following endpoints:

1. Test Service 1:

```bash
curl http://localhost:5000/service1
```
Expected response:
```json
{
    "message": "Data from Service 1"
}
```

2. **Test Service 2**:
```bash
curl http://localhost:5000/service2
```
Expected response:
```json
{
    "message": "Data from Service 2"
}
```

3. **Rate Limiting**:
If you try to hit the same endpoint multiple times in quick succession, you'll get a 'Too many requests' error:
```bash
curl http://localhost:5000/service1
curl http://localhost:5000/service1
```
Expected response for the second request:
```bash
Too many requests, please try again later.
```

4. **Error Handling:**
Stop `service1` or `service2` and try accessing them again to see the error response:
```bash
curl http://localhost:5000/service1
```

Optional Extensions:

1. **Authentication**: Implement authentication using JWT tokens to secure your routes.