Analytics Vidhya

Building APIs with Flask and FastAPI

Introduction to Creating RESTful APIs in Python

Instructor

Prashant Sahu
Manager Data Science, Analytics Vidhya

Introduction to Flask

A lightweight, micro web framework for Python

Ideal for building simple to medium-sized web application and APIs

Minimal set-up required to get started

Key Features: Flask

Minimalist core with extensibility through plugins

Simple routing for URL mapping

Built-in development server and debugger

Jinja 2 templating engine (optional for APIs)

Creating Endpoints with Flask

Steps:

1. Import Flask and create an app
2. Use decorators to define routes
3. Handle different HTTP methods (GET, POST, etc.), optional
4. Return responses (as string, JSON, HTML, etc)

flask_app.py > ...

```python
## Basic Flask Application Structure ##
from flask import Flask

app = Flask(__name__)

@app.route('/')
def home():
    return 'Hello, World!'

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

```
PS D:\Flask App\Coding Essentials Course> python flask_app.py
 * Serving Flask app 'flask_app'
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
```

Creating Endpoints with Flask

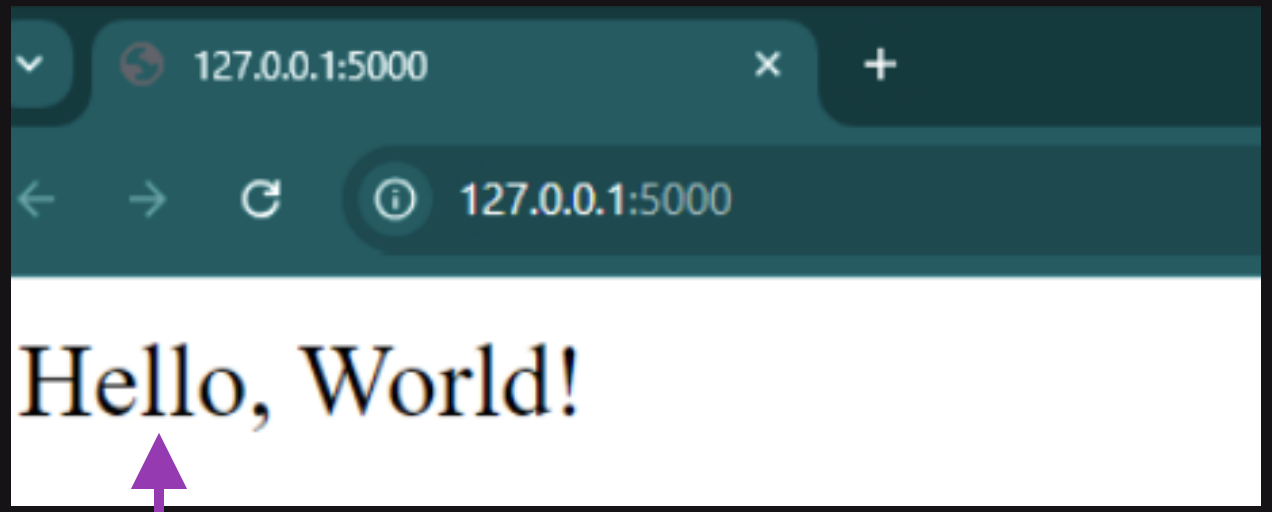
flask_app.py > ...

```python
## Basic Flask Application Structure ##
from flask import Flask

app = Flask(__name__)

@app.route('/')
def home():
    return 'Hello, World!'

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




```
PS D:\Flask App\Coding Essentials Course> python flask_app.py
 * Serving Flask app 'flask_app'
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
```

Creating Endpoints using the GET Method

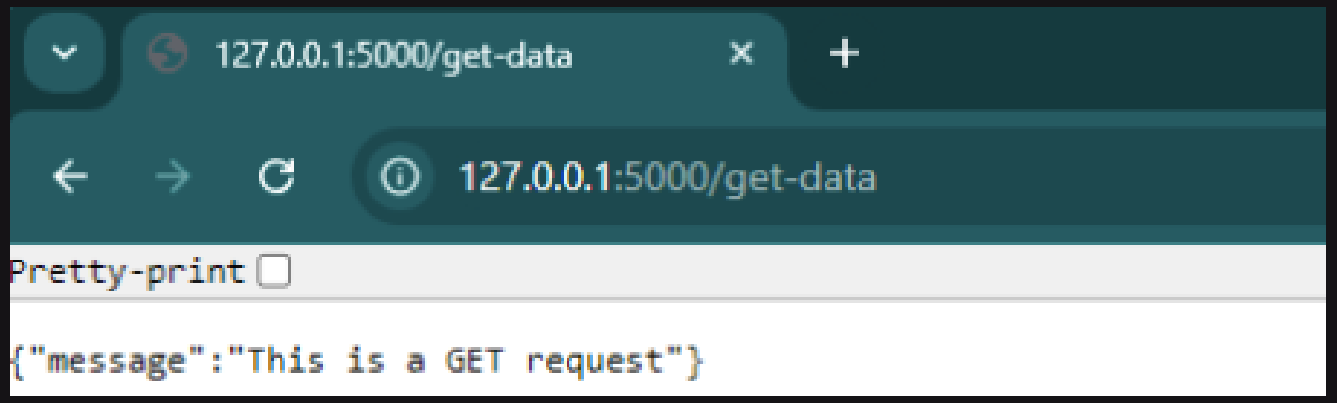
flask_app.py > ...

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

app = Flask(__name__)

# GET endpoint
@app.route('/get-data', methods=['GET'])
def get_data():
    data = {'message': 'This is a GET request'}
    return jsonify(data)

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



1. Import Modules: request for accessing request data, jsonify for JSON responses.
2. Define Routes: Use methods parameter to specify HTTP methods.
3. Access Request Data: request.get_json() retrieves JSON payload.
4. Return Responses: Use jsonify() to return JSON data.

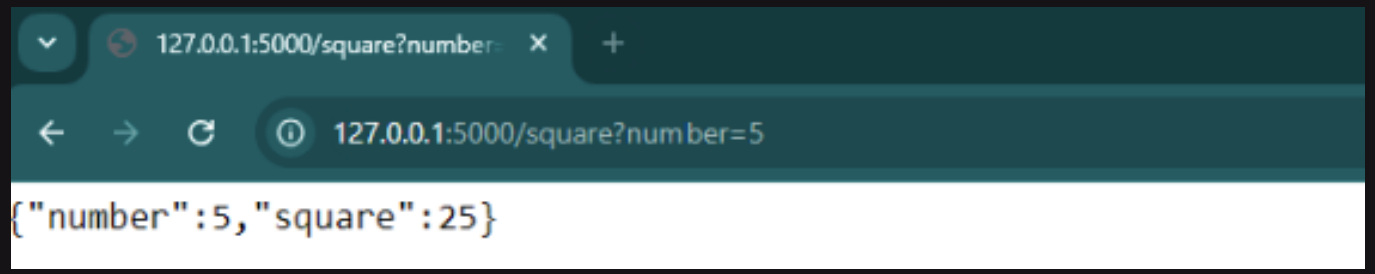
GET Method with User Inputs

```python
from flask import Flask, request

app = Flask(__name__)

@app.route('/square', methods=['GET'])
def square():
    number = request.args.get('number', default=0, type=int)
    result = number ** 2
    return {'number': number, 'square': result}

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



1. **The request.args.get() method takes three arguments:**
   - The name of the parameter ('number')
   - A default value (0) in case the parameter is not provided
   - The type to convert the input to (int)

2. **To use this endpoint, you would make a GET request to a URL like:** **http://127.0.0.1:5000/square?number=5**

Demonstrating GET and POST Methods

```python
## Handling GET and POST Requests ##
from flask import Flask, request, jsonify

app = Flask(__name__)

# GET endpoint
@app.route('/get-data', methods=['GET'])
def get_data():
    data = {'message': 'This is a GET request'}
    return jsonify(data)

# POST endpoint
@app.route('/post-data', methods=['GET', 'POST'])
def post_data():
    input_data = request.get_json()
    response = {'message': 'Data received', 'data': input_data}
    return jsonify(response)

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

1. Import Modules: **request** for accessing request data, **jsonify** for JSON responses.
2. Define Routes: Use **methods** parameter to specify HTTP methods.
3. Access Request Data: **request.get_json()** retrieves JSON payload.
4. Return Responses: Use **jsonify()** to return JSON data.

Accessing Endpoints

Options

- **Browser address bar: This only works for GET requests.**
- **Command-line tools:**

  curl: A command-line tool for transferring data.

  Example: curl http://localhost:5000/hello

  Another command-line tool.

  Example: wget -qO- http://localhost:5000/hello

- API testing tools like Postman

Accessing Endpoints via Browser and Command Line

Accessing GET Endpoint

Using Browser

Navigate to http://localhost:5000/get-data

Browser displays JSON response.

Using curl Endpoint

curl http://localhost:5000/get-data

Accessing Endpoints via Browser and Command Line

Accessing POST Endpoint

POST endpoints cannot be accessed via browser.

Using curl Command:

```
curl -X POST -H "Content-Type: application/json" \-
d '{"name": "Alice", "age": 30}'
\http://localhost:5000/post-data
```

or use Python code to make POST requests

Explanation:

-X POST: Specifies the POST method.

-H: Sets the request header.

-d: Sends the data payload.

Testing APIs Using Postman

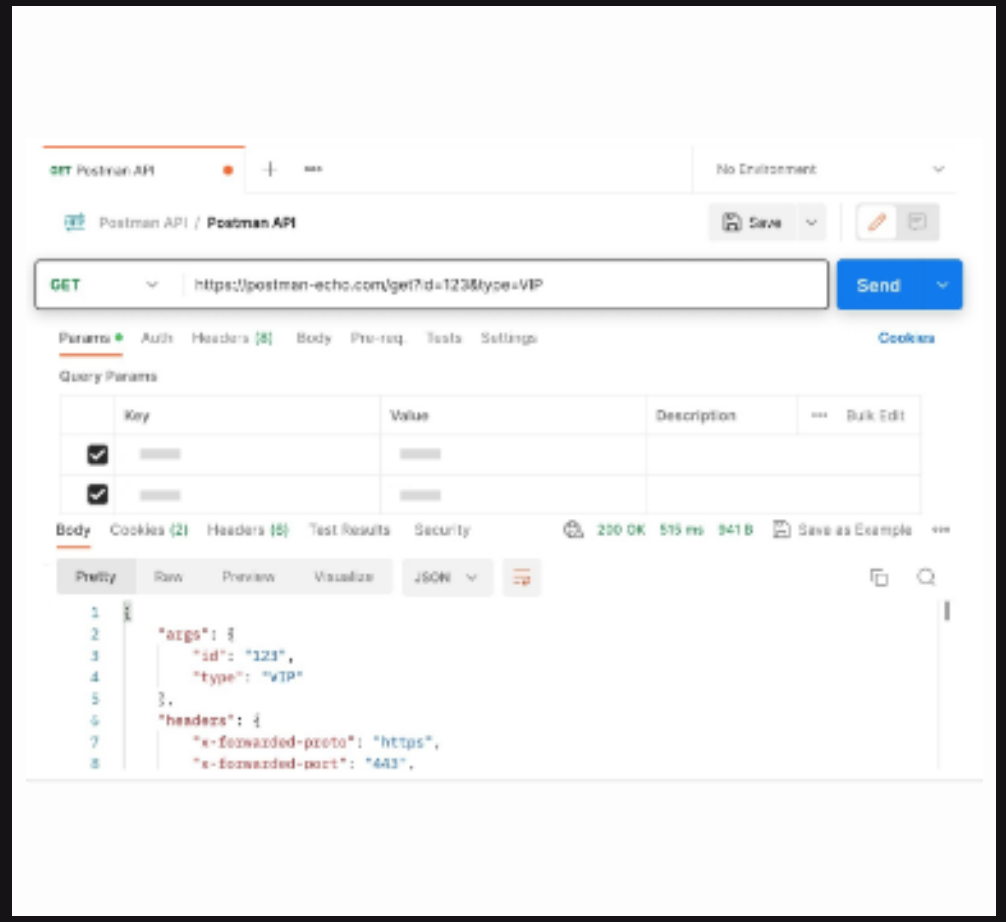
What is Postman?

- A powerful API development and testing tool.
- Provides a user-friendly interface to spend requests and view responses
- Supports all HTTP methods and custom headers
- Download Postman Free from here: https://www.postman.com/downloads/

Testing APIs Using Postman

Using Postman to Test Endpoints

1. Open Postman and create a new request.
2. Enter the endpoint URL (http://localhost:5000/post-data).
3. Select the HTTP method (GET or POST).
4. For POST requests, go to the "Body" tab:
   1. Select "raw" and choose "JSON" from the dropdown.
   2. Enter JSON data (e.g., {"name": "Alice", "age": 30}).
5. Step 5: Click "Send" to submit the request.
6. Step 6: View the response in the lower pane, including details like status code, response time, and response body.



Analytics Vidhya

Introduction to FastAPI

What is FastAPI?

- **A modern, fast (high-performance) web framework for building APIs with Python 3.6+.**
- **Key Features:**
  - Built on top of Starlette for web requests and Pydantic for data validation.
  - Automatic interactive API documentation (Swagger UI).
  - High performance comparable to Node.js and Go.
  - Supports asynchronous programming for better scalability.

FastAPI

Comparison: Flask and FastAPI

Flask

- Simple and flexible framework.
- Ideal for small to medium projects.
- Requires additional setup for advanced features.

FastAPI

- High-performance and modern.
- Automatic documentation and data validation.
- Better for larger projects requiring scalability.