1. Flask History and Introduction

Flask is a **micro** *web application development framework* written in Python. Flask was created by Armin Ronacher of Pocoo.

It is designed to make getting started quick and easy, with the ability to scale up to complex applications. It began as a simple wrapper around **Werkzeug** and **Jinja** and has become one of the most popular Python web application frameworks.

Flask offers suggestions, but doesn't enforce any dependencies or project layout. It is up to the developer to choose the tools and libraries they want to use. There are many extensions provided by the community that make adding new functionality easy.

2. Understanding 'Micro'

"Micro" does not mean that your whole web application has to fit into a single Python file (although it certainly can), nor does it mean that Flask is lacking in functionality. The "micro" in microframework means Flask aims to keep the core simple but extensible. Flask won't make many decisions for you, such as what database to use. Those decisions that it does make, such as what templating engine to use, are easy to change. Everything else is up to you, so that Flask can be everything you need and nothing you don't.

By default, Flask does not include a database abstraction layer, form validation or anything else where different libraries already exist that can handle that. Instead, Flask supports extensions to add such functionality to your application as if it was implemented in Flask itself. Numerous extensions provide database integration, form validation, upload handling, various open authentication technologies, and more. Flask may be "micro", but it's ready for production use on a variety of needs.

3. What is a web Framework?

A web framework is a collection of libraries which helps the web developers to create different web applications.

Flask internally uses two main components:

Werkzeug: Werkzeug is a comprehensive WSGI web application library. It began as a simple collection of various utilities for WSGI applications and has become one of the most advanced WSGI utility libraries.

Jinja: Jinja is a modern and designer-friendly templating language for Python. It is fast, widely used and secure with the optional sandboxed template execution environment.

WSGI: The Web Server Gateway Interface (WSGI, pronounced whiskey) is a simple calling convention for web servers to forward requests to web applications or frameworks written in the Python programming language. The current version of WSGI is version 1.0.1.

4. Flask Installation

Flask libraries can be easily installed using any the following commands:

- pip install flask
- python -m pip install flask
- conda install flask

Flask like any other web application development framework, utilizes the HTTP methods. Let's see the most used methods:

5. **HTTP Methods:**
   a) **GET:** The GET method means retrieve whatever information (in the form of an entity) is identified by the Request-URI. If the Request-URI refers to a data-producing process, it is the produced data which shall be returned as the response.For example, we can have an API called *getCustomerDetails* which returns the details of customers based on *customerID*.

   b) **POST**: The POST method is used to request that the origin server accept the entity enclosed in the request as a new subordinate of the resource identified by the Request-URI in the Request-Line. POST is designed to allow a uniform method to cover the following functions:

   - Posting a message to a bulletin board, newsgroup, mailing list, or similar group of articles;
   - Providing a block of data, such as the result of submitting a form, to a data-handling process;
   - Extending a database through an append operation.

   For example, we can have an *addCustomer* API which can add the new customers to the customer database.

   c) **PUT**: The PUT method requests that the enclosed entity be stored under the supplied Request-URI. If the Request-URI refers to an already existing resource, the enclosed entity SHOULD be considered as a modified version of the one residing on the origin server. If the Request-URI does not point to an existing resource, and that URI is capable of being defined as a new resource by the requesting user agent, the origin server can create the resource with that URI. If a new resource is created, the origin server MUST inform the user agent via the 201 (Created) response. If an existing resource is modified, either the 200 (OK). For Example, we can use an updateCustomer API which can update the details of existing customers based on customerID or if the customer is not present, it will create a new one.

   In the examples of HTTP methods, we have talked about APIs, let's see what an API is.

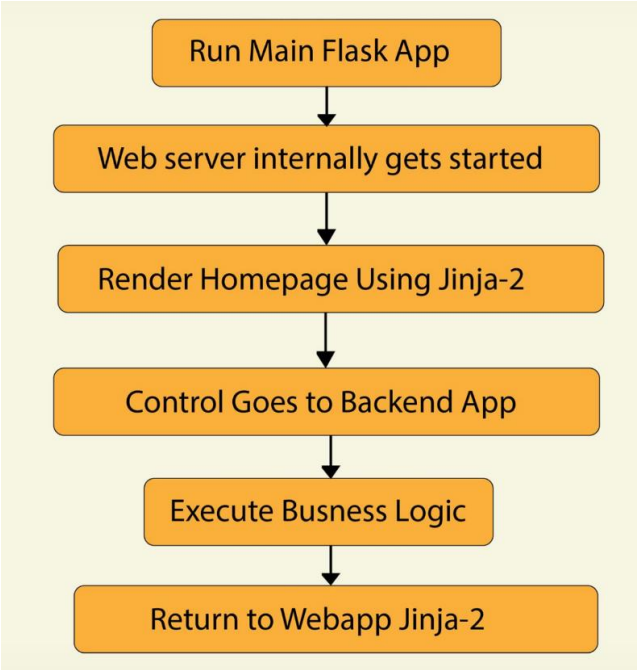
6. What is an API?

An application programming interface (API) is a computing interface exposed by a particular software program, library, operating system or internet service, to allow third parties to use the functionality of that software application.

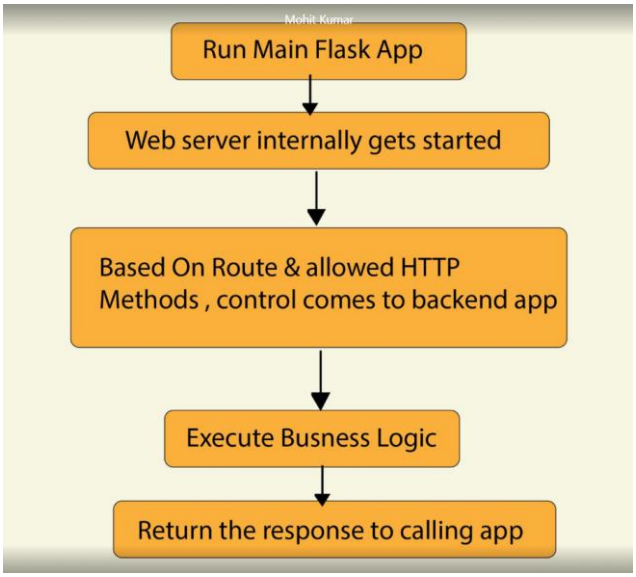
In the context of the internet, a Web API refers to the use of third party web services and sites , typically over HTTP, as part of new application development.

Majorly, there are two kinds of web applications that we can make using flask:

a) **Flask App with UI**: The flow for such application is:



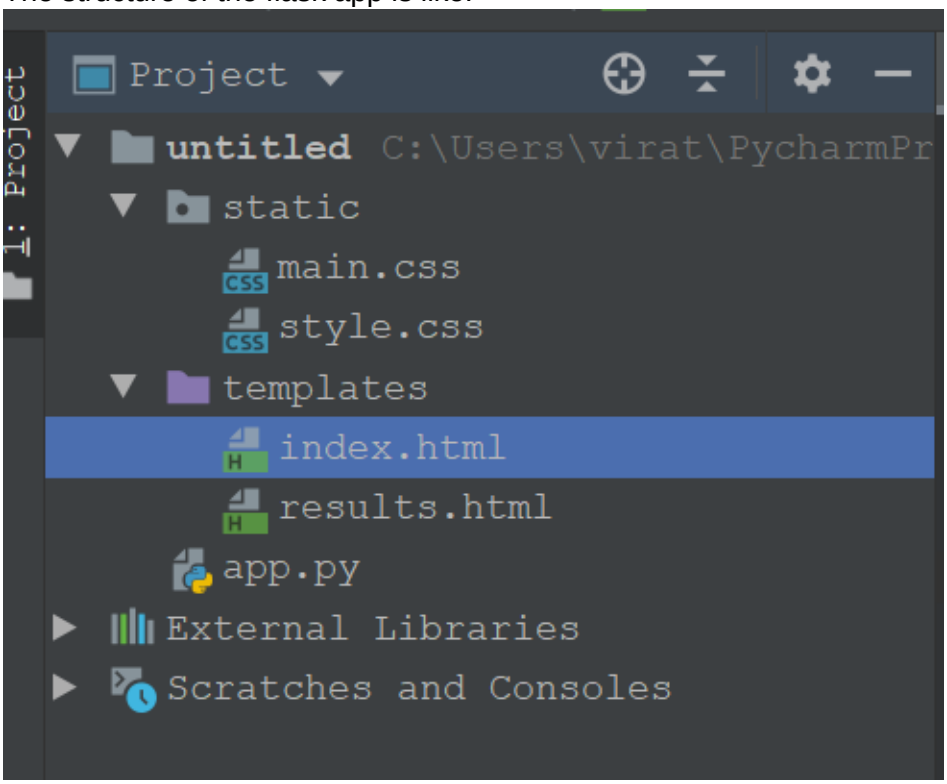
b) **Standalone Flask App without UI:** The flow for such application is:



Let's create a Flask app now.

7. The Flask App:

The structure of the flask app is like:



- The **static** folder contains the files which do not change like the **css** files. These will be used to control to appearance of the UI.
- The **templates** folder contains the html pages which will be used to create the UI.
- The **app.py** file contains the python code that can redirect to execute the custom business logic.
- **Note:** static and template folders are mostly required to build UI and not for a standalone API.

7.1 Code Snippets:

Index.html:

```
{% block head %}

<title>Calculator</title>
<link rel="stylesheet" href="{{ url_for('static',
filename='css/style.css') }}">
{% endblock %}

{% block body %}
<div class="content">
    <h1 style="text-align: center">Calculator</h1>

    <div class="form">
        <form action="/math" method="POST">
            <label for="operation">Choose a Mathematical
```

```
Operation</label>

<select id="operation" name="operation">
  <option value="add">add</option>
  <option value="subtract">subtract</option>
  <option value="multiply">multiply</option>
  <option value="divide">divide</option>
</select>
            <input type="text" name="num1" id="num1">
            <input type="text" name="num2" id="num2">
            <input type="submit" value="Calculate">
        </form>
    </div>
</div>
{% endblock %}
```

app.py:

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

app = Flask(__name__)


@app.route('/', methods=['GET', 'POST']) # To render Homepage
def home_page():
    return render_template('index.html')

@app.route('/math', methods=['POST'])  # This will be called from
UI
def math_operation():
    if (request.method=='POST'):
        operation=request.form['operation']
        num1=int(request.form['num1'])
        num2 = int(request.form['num2'])
        if(operation=='add'):
            r=num1+num2
            result= 'the sum of '+str(num1)+' and '+str(num2) +'
is '+str(r)
        if (operation == 'subtract'):
            r = num1 - num2
            result = 'the difference of ' + str(num1) + ' and ' +
str(num2) + ' is ' + str(r)
        if (operation == 'multiply'):
            r = num1 * num2
            result = 'the product of ' + str(num1) + ' and ' +
str(num2) + ' is ' + str(r)
        if (operation == 'divide'):
            r = num1 / num2
            result = 'the quotient when ' + str(num1) + ' is
divided by ' + str(num2) + ' is ' + str(r)
        return render_template('results.html',result=result)

@app.route('/via_postman', methods=['POST']) # for calling the
API from Postman/SOAPUI
def math_operation_via_postman():
```

```
    if (request.method=='POST'):
        operation=request.json['operation']
        num1=int(request.json['num1'])
        num2 = int(request.json['num2'])
        if(operation=='add'):
            r=num1+num2
            result= 'the sum of '+str(num1)+' and '+str(num2) +'
is '+str(r)
        if (operation == 'subtract'):
            r = num1 - num2
            result = 'the difference of ' + str(num1) + ' and ' +
str(num2) + ' is ' + str(r)
        if (operation == 'multiply'):
            r = num1 * num2
            result = 'the product of ' + str(num1) + ' and ' +
str(num2) + ' is ' + str(r)
        if (operation == 'divide'):
            r = num1 / num2
            result = 'the quotient when ' + str(num1) + ' is
divided by ' + str(num2) + ' is ' + str(r)
        return jsonify(result)


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

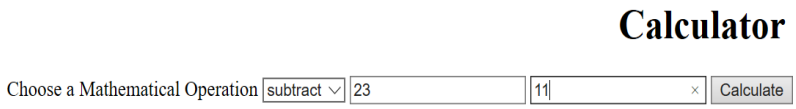
results.html

```
<!DOCTYPE html>
<html lang="en" >

<head>
  <meta charset="UTF-8">
  <title>Review Page</title>
    <link rel="stylesheet"
href="https://cdnjs.cloudflare.com/ajax/libs/normalize/5.0.0/norm
alize.min.css">
      <link rel="stylesheet" href="./style.css">
    <link rel="stylesheet" href="{{ url_for('static',
filename='css/style.css') }}">
</head>
<body>
  <div class="table-users">
   <div class="header">Calculation Result</div>

      {{result}}
</div>
</body>
</html>
```

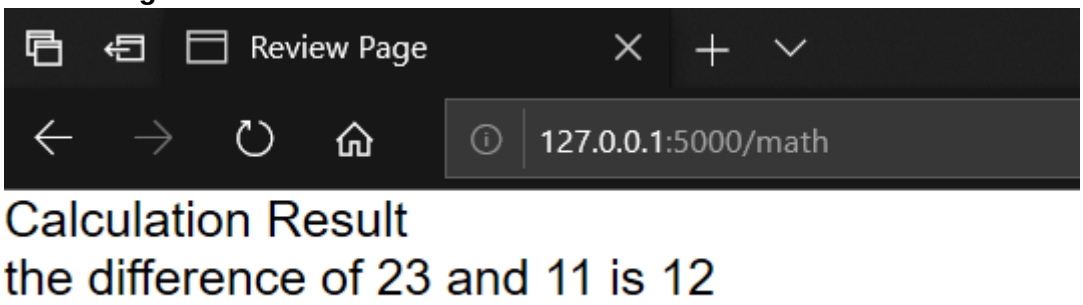
8. Results after execution:

8.1 From UI

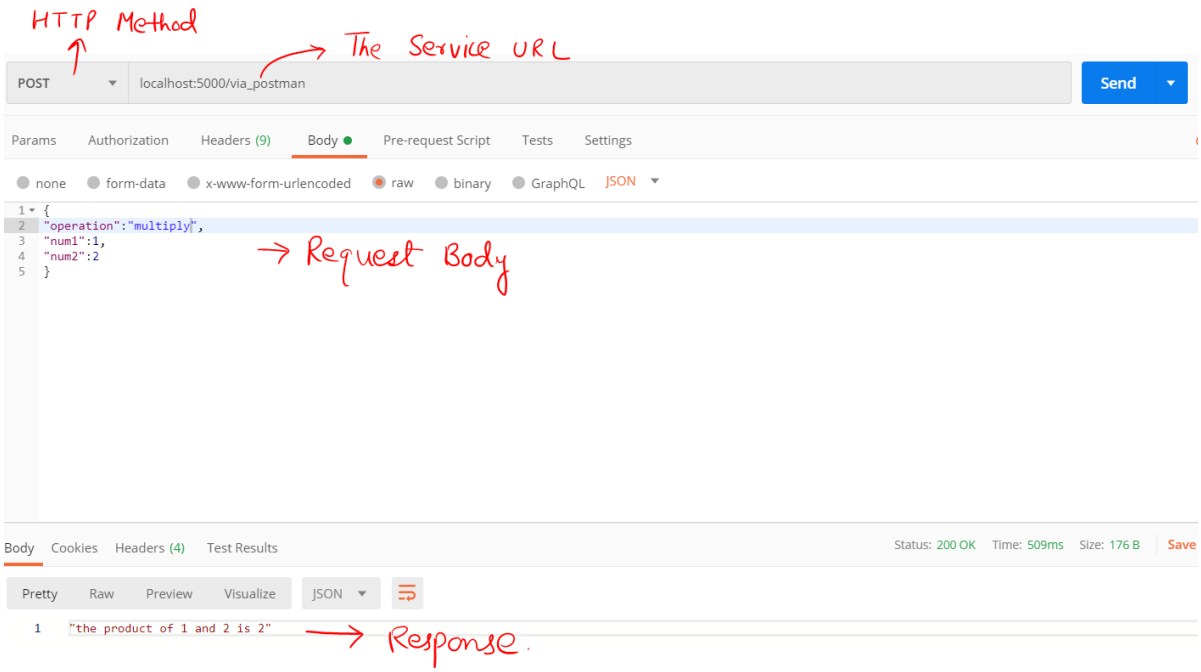
Homepage:



Result Page:



8.2 From Postman



References:

- The official Flask Documentation
- Wikipedia