Flask

- Flask is a lightweight and flexible web framework for Python.
- It's designed to make getting started with web development quick and easy, while also providing the tools necessary for building robust and scalable web applications.

Key features of Flask

Minimalistic

Flask is intentionally kept small and simple. It doesn't come bundled with a lot of features out of the box, but instead provides a solid foundation for building web applications.

Routing

Routing in Flask is achieved using Python decorators. You can define routes for different URLs and map them to specific Python functions, known as view functions, which handle the requests and return responses.

Template Engine

Flask uses Jinja2 as its template engine. Templates allow you to build HTML pages dynamically by embedding Python code within HTML markup.

HTTP Methods

Flask supports all standard HTTP methods such as GET, POST, PUT, DELETE, etc. You can define routes that respond to specific HTTP methods.

Web Development Made Easy

Flask simplifies web development by providing built-in development server, integrated unit testing support, and a simple yet powerful API.

Extensions

Flask's functionality can be extended through various extensions. These extensions cover a wide range of features including form validation, database integration (e.g., SQLAlchemy), authentication (e.g., Flask-Login), and more.

Werkzeug and Jinja2

Flask is built on top of the Werkzeug WSGI toolkit and the Jinja2 template engine. Werkzeug provides a set of utilities for handling HTTP requests and responses, while Jinja2 enables powerful and flexible template rendering.

RESTful APIs

Flask is commonly used for building RESTful APIs due to its simplicity and flexibility. With Flask, you can easily create endpoints that respond to HTTP requests with JSON data.

Scalability

While Flask is often considered a micro-framework, it's highly scalable. You can use Flask to build small projects as well as large, complex applications.

Create Flask Application and add some routes

In [ ]:
```
from flask import Flask,request
from flask_cors import CORS # for apis

server = Flask(__name__)
CORS(server)

@server.route('/',methods=['GET'])  # This is a route decorator
def index():
    return 'Hello, World! This is index page'

@server.route('/about') # default method is GET
def about():
    return 'This is the about page'


@server.route('/login', methods=['POST'])
def login():
    username = request.form['username']
    password = request.form['password']
    # Check username and password
    return 'Login successful'

@server.route('/update', methods=['PUT'])
def update():
    # Update resource
    return 'Resource updated'

@server.route('/partial-update', methods=['PATCH'])
def partial_update():
    # Apply partial modifications to the resource
    return 'Resource partially updated'

@server.route('/delete', methods=['DELETE'])
def delete():
    # Delete resource
    return 'Resource deleted'
```

```
if __name__ == "__main__":
    server.run(host='0.0.0.0',port=5050)
```

Flask Variables and Rules

In [ ]:
```
from flask import Flask

app = Flask(__name__)

@app.route('/user/<username>')
def show_user_profile(username):
    return 'User %s' % username

@app.route('/post/<int:post_id>')
def show_post(post_id):
    return 'Post %d' % post_id

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

- In the example above, two routes are defined with variable rules: /user/ and /post/int:post_id.
- In the first route, is a variable rule that captures the username from the URL and passes it as a parameter to the show_user_profile() function.
- Similarly, in the second route, int:post_id is a variable rule that captures an integer value from the URL and passes it as a parameter to the show_post() function.
- Variable rules are defined by enclosing the variable name within < and > angle brackets in the route URL. You can also specify converters like int, float, or path to define the type of data expected for the variable.

Static files and Templates in Flask

- Dynamic web applications also need static files. That's usually where the CSS and JavaScript files are coming from. Just create a folder called static in our application and it will be available at /static on the application.
- To generate URLs for static files, use the special 'static' endpoint name: url_for('static', filename='css/style.css')
- In Flask, rendering templates allows developers to generate dynamic HTML pages by combining static content with dynamic data.
- Flask utilizes the Jinja2 templating engine for this purpose.
- Templates in Flask are HTML files containing placeholders, which are replaced with actual values when the template is rendered.

Example

In [ ]:
```
# server.py
from flask import Flask,render_template

server = Flask(__name__)
```

```
@server.route('/',methods=['GET']) # define route and methods
def index():
    # render index.html and pass name to it
    return render_template('index.html',name='Tek Raj')


if __name__ == "__main__":
   server.run(host='0.0.0.0',port=4000)

# templates/index.html
'''
<!doctype html>
<title>Hello from Flask</title>
# name passed from flask
{% if name %}
  <h1>Hello {{ name }}!</h1>
{% else %}
  <h1>Hello, World!</h1>
{% endif %}
'''
```

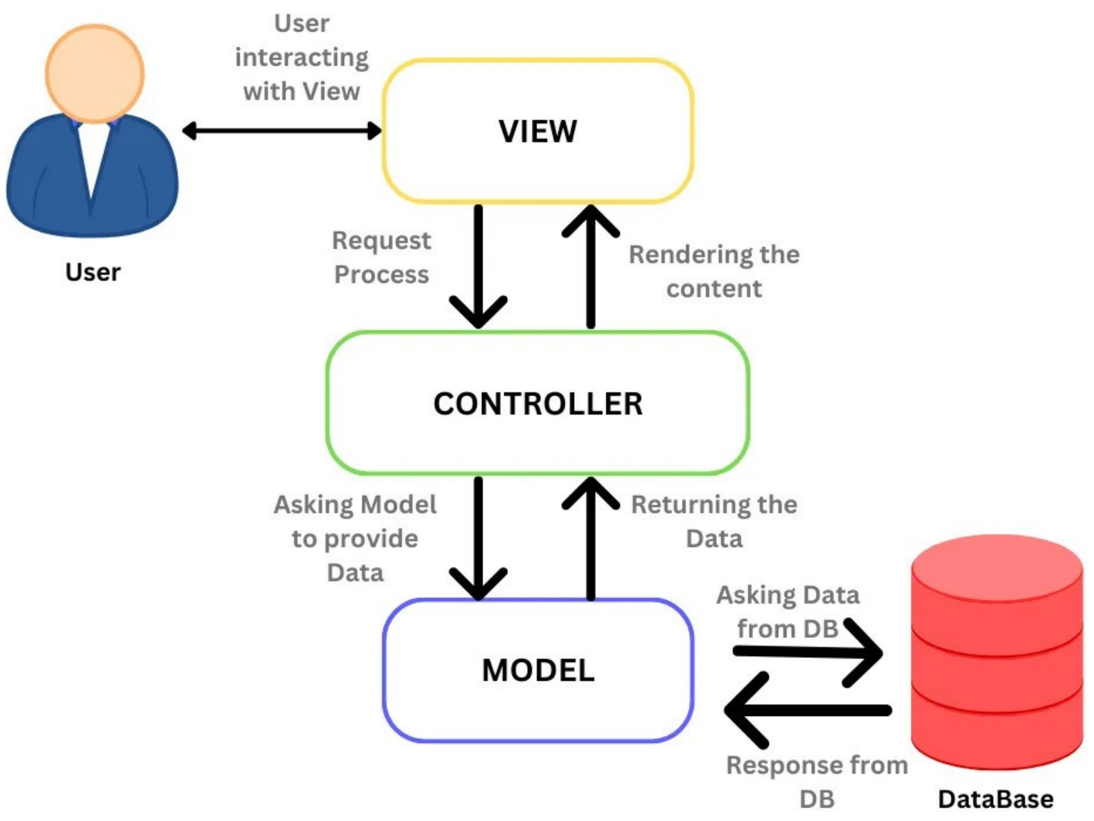
Flask Request Object

- Accessing Form Data
- Accessing Query Parameters (http://localhost:4000/index?page=2)
- Here in this url page is query params
- Accessing Request Headers (Request headers can be accessed using the request.headers attribute.)
- Accessing JSON Data
- Accessing Cookies
- Accessing Uploaded File

In [ ]:
```
from flask import Flask,request

server = Flask(__name__)


@server.route('/')
def index():
    # access request headers
    auth_token = request.headers.get('Authorization')
    # access cookies
    username = request.cookies.get('username')

# accessing form data
@server.route('/login', methods=['POST'])
def login():
    username = request.form['username']
    password = request.form['password']

# Accessing Query Parameters:
# (http://localhost:4000/index?page=2) here page is query prams
@app.route('/search')
def search():
    page = request.args.get('page')
```

```
# get json data
@app.route('/json', methods=['POST'])
def process_json():
    data = request.json
    # Process the JSON data

@app.route('/upload', methods=['POST'])
def upload_file():
    file = request.files['file']
    # Save the uploaded file to the server
```

Flask MVC Pattern



Example

Model

In [ ]:
```
from flask_sqlalchemy import SQLAlchemy

db = SQLAlchemy()

class User(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(50), unique=True, nullable=False)
    email = db.Column(db.String(120), unique=True, nullable=False)
    def __repr__(self):
        return f'<User {self.username}>'
```

View (user.html)

In [ ]:
```
''' <!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>User Profile</title>
</head>
<body>
    <h1>User Profile</h1>
    <p>Username: {{ user.username }}</p>
    <p>Email: {{ user.email }}</p>
</body>
</html> '''
```

Controller

In [ ]:
```
from models.user import User
class UserController:
    def __init__(self):
        self.user =  User()

    def get_users(self):
        users = User.getUsers();
        return users

    def create(self,user):
        new_user = self.user.create(user)
        return new_user

    def update(self,id,update_data):
        update = self.user.update(update_data).where(id,id)
        return update

    def delete(self,id):
        self.user.delete(id)
        return True
```

Routes

In [ ]:
```
from flask import Flask,render_template
from controllers.user import User
server = Flask(__name__)

user = User()

@server.route('/')
def index():
    users = user.get_users()
    return render_template('user.html',users)
```