Form Handling in Flask:

- Forms are an essential part of web development, allowing users to input data that your application can process. Flask provides a simple and flexible way to handle forms using the `request` object.
- To send the data using the HTML form page, generally we use GET or POST methods of the HTTP protocol.

HTTP Methods: GET and POST

What is GET?

- **GET** is used to request data from a server.
- Form data is appended to the URL as query parameters (e.g., `?name=John&age=25`).
- To retrieve form data sent using GET, use the `request.args.get()` method.
- **Characteristics**:
    - Data is **visible in the URL** (not secure for sensitive information).
    - Suitable for **retrieving non-sensitive data** like search queries or filters.
    - Limited data size due to URL length restrictions.
    - Allows bookmarking or sharing of URLs with query parameters.
    - It is relatively faster than the POST method.

What is POST?

- **POST** is used to send data to the server for processing or storing.
- Form data is sent in the **request body,** not visible in the URL.
- To retrieve form data sent using POST, use the `request.form.get()` method or `request.form["key"]`
- **Characteristics**:
    - Data is **hidden from the URL** (suitable for sensitive information).
    - No size limitations on data (useful for large payloads like file uploads).

- Commonly used for **submitting sensitive or modifying server-side data** (e.g., creating accounts, uploading files).

Key Differences Between GET and POST:

Aspect	GET	POST
Data Location	Appended to the URL	Sent in the request body
Visibility	Visible in the URL	Hidden from the URL
Use Case	Fetching data	Sending/modifying data
Bookmarkable	Yes	No
Security	Less secure for sensitive data	More secure

When to Use GET vs. POST

- **Use GET**:
  - When the action is **idempotent** (does not change data on the server).
  - For search forms, filters, or non-sensitive data submissions.
- **Use POST**:
  - For actions that **modify server-side data**.
  - For **sensitive data submissions** like passwords or large payloads.

How to Handle Forms in Flask

Step 1: Set Up the Flask Application

- Import the required libraries and set up the Flask app.

Step 2: Create an HTML Form

- Design a form in HTML using `<form>` tags. Define the action (`url_for` endpoint) and the HTTP method (`GET` or `POST`).

Step 3: Handle the Request in Flask

- Use Flask's `@app.route` decorator to handle routes and specify allowed methods (`GET`, `POST`, or both).
- Use Flask's `request` object to capture form data.

Example1. Displaying the the User information:

Folder Structure:

FlaskFormApp
|
|-- app.py
|
|--templates
|
|--user_form.html
|--result.html

user_form.html:

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

<head>
    <meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>

</head>

<body bgcolor="cyan">

    <h1 style="text-align: center;">Welcome to Chitkara</h1>

    <form action="{{url_for('add')}}">

        <label for="num1">Enter Number1:</label>
        <input type="number" id="num1" name="n1" required>
```

```
        <br><br>

        <label for="num2">Enter Number2:</label>
        <input type="number" id="num2" name="n2" required>

        <br><br>

        <input type="submit" value="GetResult">

    </form>

</body>

</html>
```

app.py:

```
from flask import Flask, render_template, request
app = Flask(__name__)


@app.route('/')
def home():
    return render_template('user_form.html')


@app.route("/add", methods=["GET"])
def add():
    n1 = request.args.get("n1")
    n2 = request.args.get("n2")
    result = float(n1) + float(n2)
    return render_template("result.html", result=result)


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

result.html:

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

<head>
    <meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>

<body bgcolor="yellow">

    <h1>The Result is: {{result}}</h1>

</body>

</html>
```

Note: Since form data is sent here without mentioning the method, it is using the GET method by default. and the data is visible to the URL also.

In order to send the data using the POST method from the html form page and handle the form POST data inside the view function do the following modification in the above example:

1. inside the user_form the html:

```
<form action="{{url_for('add')}}", method='POST'>
```

2. inside the app.py file:

```
@app.route("/add", methods=["POST"])
def add():

    # n1 = request.form["n1"]
    n1 = request.form.get("n1")
    n2 = request.form.get("n2")
    result = float(n1) + float(n2)
    return render_template("result.html", result=result)
```

Example: Modification of the above program to display the result to the same page.Using POST method.

- Delete the result.html file from the above application.

user_form.html:

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

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>

</head>

<body bgcolor="cyan">

    <h1 style="text-align: center;">Welcome to Chitkara</h1>

    <form action="{{url_for('home')}}" , method="POST">

        <label for="num1">Enter Number1:</label>
        <input type="number" id="num1" name="n1" required>

        <br><br>

        <label for="num2">Enter Number2:</label>
        <input type="number" id="num2" name="n2" required>

        <br><br>

        <input type="submit" value="GetResult">
    </form>

    <hr>
```

```
    <!-- {% if result %} -->

  {% if result%}
    <h2>The Result is {{result}}</h2>
  {% endif %}




</body>

</html>
```

app.py:

```
from flask import Flask, render_template, request
app = Flask(__name__)


@app.route('/', methods=["GET", "POST"])
def home():


    if request.method == "POST":
        # n1 = request.form["n1"]
        n1 = request.form.get("n1")
        n2 = request.form.get("n2")
        sum= float(n1) + float(n2)

    return render_template('user_form.html', result=sum)


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

Student Task: Flask Application for Login Validation

- Create a Flask application that accepts a username and password from the user via a form. Based on the entered credentials, perform the following:
    1. If the username is `"admin"` and the password is `"123456"`, display a Home Page with a welcome message that includes the username.
    2. If the credentials are incorrect, display an error message ("Invalid username or password") on the same login page.

Solution:

FlaskLoginApp
|
|-- app.py
|
|--templates
|
|--user_form.html
|--profile.html

app.py

```
from flask import Flask, render_template, request, redirect, url_for,
session


app = Flask(__name__)
app.secret_key = "My Secret key"


@app.route('/')  # 127.0.0.1:5000/
def home():
    return render_template("user_form.html")


@app.route("/login", methods=["GET", "POST"])
```

```python
def login():
    message = "Please Login First"
    if request.method == "POST":
        name = request.form.get("name")
        uname = request.form.get("username")
        pwd = request.form.get("password")

        if uname == "admin" and pwd == "12345":
            session["name"] = name
            return redirect(url_for("profile"))
        else:
            message = "Invalid Username or password"

    return render_template("user_form.html", result=message)



@app.route("/profile")
def profile():
    name = session.get("name")
    if name:
        return render_template("profile.html", name=name)
    else:
        return redirect(url_for("home"))



@app.route("/logout")
def logout():
    session.pop("name", None)  # Remove 'name' from session
    return redirect(url_for("home"))


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

user_form.html:

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

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>

<body bgcolor="cyan">

    <h1 style="text-align: center;">Welcome to Chitkara</h1>


    <form action="{{url_for('login')}}" method="POST">

        <label for="name">Enter Name:</label>
        <input type="text" id="name" name="name" required>

        <br><br>

        <label for="username">Enter Username:</label>
        <input type="text" id="username" name="username" required>

        <br><br>
        <label for="password">Enter Password:</label>
        <input type="password" id="password" name="password" required>

        <br><br>

        <input type="submit" value="SignIn">




    </form>

    {% if result %}
```

```
    <h2 style="color:red">{{result}}</h2>

    {% endif %}

</body>

</html>
```

profile.html:

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

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>

<body bgcolor="yellow">

    <h1>Welcome to the Dashboard</h1>

    <h2>Welcome: {{name}} </h2>

    <a href="{{url_for('logout')}}">Logout</a>

</body>

</html>
```

Session & Secret Key Explanation

What is a session in Flask?

- session is a way to store user-specific data across multiple requests.
- It is stored on the server but uses a signed cookie to track the session data.
- The session allows the application to remember logged-in users until they close the browser or logout.

How does a session work in your app?

- When a user logs in successfully, their name is stored in session["name"].
- The profile route checks if "name" exists in session. If it does, the user is allowed to view the profile page.
- If the session does not contain "name", the user is redirected to the login page.

What is app.secret_key?

- The secret_key is used to sign the session cookies to prevent tampering.
- Without setting a secret_key, Flask will not be able to securely store session data.
- The key should be complex and secret (avoid hardcoding it in production).

Scenario:

Imagine you are hosting a **private party** 🎉, and you want to allow only people with an **invitation card** to enter. The invitation card has a **special seal** that only you can create, so nobody can **fake an invitation**.

In Flask, the `SECRET_KEY` acts like that **special seal**. It helps Flask verify that data (like cookies or session data) **has not been modified or faked** by someone else.

- The session data is **signed** using `SECRET_KEY` so that no one can tamper with it.
- If a hacker tries to modify the session data **without the correct `SECRET_KEY`**, Flask will reject it.

Using the Flash messages: (Message flashing)

- Flash messages in Flask are a mechanism for sending one-time messages from the server to the client, typically to notify users about the result of an action (e.g., success, error, or warning). They are a great way to provide feedback to users after performing operations like logging in, signing up, or submitting forms.
- Message Flashing means when we click on any button(any website) and the website sends immediately any message that can be an error, alert, or any type of message, it is called the **Message Flashing** and when that happens in a flask or we are creating any website which shows an alert, error type message than its called as **Flask-Message Flashing**.

How Flash Messages Work:

1. **Temporary Storage**: Flash messages are stored in the session, which means they persist only for a single request.
2. `flash()` **Function**: Used to add a message to the session's flash storage.
3. `get_flashed_messages()` **Function**: Used to retrieve and display the flash messages in templates(inside the HTML).

Flash Message Components:

1. `flash()` **Function**

   Syntax:

   ```
   flash(message, category)
   ```

   - `message` (required): The message you want to display.
   - `category` (optional): The type of message (e.g., `success`, `error`, `info`, etc.)

   Example:

   ```
   flash("This is an informational message.", "info")
   flash("Form submitted successfully!", "success")
   flash("An error occurred. Please try again.", "error")
   ```

2. `get_flashed_messages()` **Function**

    - It is used in templates to retrieve and display flash messages.

    Syntax:

    ```
    get_flashed_messages(with_categories=False, category_filter=[])
    ```

- `with_categories` **(boolean): If** `True`**, returns a list of tuples** `(category, message)`**. Default is** `False`**.**
- `category_filter` **(list): Filters messages by the given categories. Default is an empty list (no filtering).**

Example:

Without category:

```
{% for messages in get_flashed_messages() %}
  <h3 style="color:red">{{messages}}</h3>
{% endfor %}
```

With category:

```
{% for category, messages in get_flashed_messages(with_categories=True) %}
  <h3 style="color:red">{{category}} ==== {{messages}}</h3>
{% endfor %}
```

With Bootstrap:

- Make sure that bootstrap related JavaScript also included.

```
{% for category,messages in get_flashed_messages(with_categories=True) %}

<div class="alert alert-{{category}} alert-dismissible fade show"
role="alert">
            {{messages}}
<button type="button" class="btn-close" data-bs-dismiss="alert"
aria-label="Close"></button>
</div>

{% endfor %}
```

Key Features

1. **Categorization**: Flash messages can have categories (`success`, `info`, `error`, etc.), which are useful for styling.
2. **Bootstrap Integration**: Works seamlessly with Bootstrap's alert classes for better styling.
3. **One-Time Messages**: Flash messages are automatically removed after being displayed once.
4. **Custom Filters**: Use `category_filter` in `get_flashed_messages()` to retrieve specific message categories.

Common Use Cases

1. **User Authentication**:
    - Success message after login: "Welcome, John!"
    - Error message for invalid credentials: "Incorrect username or password."
2. **Form Validation**:
    - Success: "Your form has been submitted successfully."
    - Error: "Please fill in all required fields."
3. **Action Feedback**:
    - Success: "Item added to your cart."
    - Warning: "This action is irreversible."

Example: Using Flash message inside the above application:

app.py:

```
from flask import Flask, render_template, request, redirect, url_for,
session, flash


app = Flask(__name__)
app.secret_key = "My Secret key"


@app.route('/')  # 127.0.0.1:5000/
def home():
    return render_template("user_form.html")


@app.route("/login", methods=["GET", "POST"])
def login():
    if request.method == "POST":
        name = request.form.get("name")
        uname = request.form.get("username")
        pwd = request.form.get("password")

        if uname == "admin" and pwd == "12345":
            session["name"] = name
            return redirect(url_for("profile"))
        else:
            flash("Invalid Username or password", "danger")
            return render_template("user_form.html")

     # If user tries to access /login directly via GET request
    flash("Please login first", "warning")
    return render_template("user_form.html")


@app.route("/profile")
def profile():
    name = session.get("name")
```

```
    if name:
        return render_template("profile.html", name=name)
    else:
        flash("Please login first", "warning")
        return redirect(url_for("home"))


@app.route("/logout")
def logout():
    session.pop("name", None)  # Remove 'name' from session
    flash("You have been logged out.", "info")
    return redirect(url_for("home"))


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

user_form.html:

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

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>

    <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.
css" rel="stylesheet"

integrity="sha384-QWTKZyjpPEjISv5WaRU9OFeRpok6YctnYmDr5pNlyT2bRjXh0JMhjY6h
W+ALEwIH" crossorigin="anonymous">

</head>

<body bgcolor="cyan">
```

```
    <h1 style="text-align: center;">Welcome to Chitkara</h1>


    <form action="{{url_for('login')}}" method="POST">

        <label for="name">Enter Name:</label>
        <input type="text" id="name" name="name" required>

        <br><br>

        <label for="username">Enter Username:</label>
        <input type="text" id="username" name="username" required>

        <br><br>
        <label for="password">Enter Password:</label>
        <input type="password" id="password" name="password" required>

        <br><br>

        <input type="submit" value="SignIn">




    </form>




    {% for category,messages in get_flashed_messages(with_categories=True)
%}


    <div class="alert alert-{{category}} alert-dismissible fade show"
role="alert">
        {{messages}}
        <button type="button" class="btn-close" data-bs-dismiss="alert"
aria-label="Close"></button>
    </div>
```

```
    {% endfor %}



    <script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/js/bootstrap.bundle
.min.js"

integrity="sha384-YvpcrYf0tY3lHB60NNkmXc5s9fDVZLESaAA55NDzOxhy9GkcIdslK1eN
7N6jIeHz"
        crossorigin="anonymous"></script>

</body>

</html>
```

profile.html:

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

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>

<body bgcolor="yellow">

    <h1>Welcome to the Dashboard</h1>

    <h2>Welcome: {{name}} </h2>

    <a href="{{url_for('logout')}}">Logout</a>


</body>
```

```
</html>
```

Example: Creating a Form to capture the Student information(roll, name, address, marks) and showing the student information below the form page inside a table.

Folder Structure:

FlaskStudentApp2
|
|-- app.py
|
|--templates
|
|--base.html
|--index.html
|--student.htmll

base.html

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

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>{% block title_block %} {% endblock %}</title>

    <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.
css" rel="stylesheet"
```

```
integrity="sha384-QWTKZyjpPEjISv5WaRU9OFeRpok6YctnYmDr5pNlyT2bRjXh0JMhjY6h
W+ALEwIH" crossorigin="anonymous">

    <style>
        body {
            margin: 0;
            padding: 0;
        }

        header {
            background-color: aqua;
            height: 10vh;
        }

        main {
            background-color: antiquewhite;
            height: 80vh;
        }

        footer {
            background-color: aquamarine;
            height: 10vh;
        }
    </style>

</head>

<body bgcolor="Cyan">

    <header>
        <h1 class="text-center">Welcome to Chitkara</h1>
    </header>

    <main>

        {% block main_block %}

        {% endblock %}
```

```
    </main>


    <footer>
        <p class="text-center">&copy; 2025 Student Information System</p>
    </footer>



</body>

</html>
```

index.html:

```
{% extends "base.html" %}

{% block title_block %} Home Page {% endblock %}


{% block main_block %}


<a href="{{url_for('student')}}">Register A Student</a>

{% endblock %}
```

app.py:

```
from flask import Flask, render_template, request
app = Flask(__name__)

students = []


@app.route("/")
def index():
    return render_template("index.html")
```

```python
@app.route("/student", methods=["GET", "POST"])
def student():
    if request.method == "POST":
        # Fetch the form data
        roll = request.form.get("roll")
        name = request.form.get("name")
        address = request.form.get("address")
        marks = request.form.get("marks")

        # compose the dictionary object
        student = {"roll": int(roll), "name": name,
                   "address": address, "marks": int(marks)}
        # appending to the global list
        students.append(student)

    return render_template("student.html", students=students)


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

student.html:

```
{% extends "base.html" %}

{% block title_block %} Student Page {% endblock %}


{% block main_block %}

<h2 class="text-center">Student Registration Screen</h2>


<a href="{{url_for('index')}}">Back</a>

<form action="{{url_for('student')}}" method="POST">
    <div class="form-group">
        <label for="roll">Roll Number</label>
```

```
        <input type="number" class="form-control" id="roll"
placeholder="Enter Roll Number" name="roll" required>

    </div>
    <div class="form-group">
        <label for="name">Name</label>
        <input type="text" class="form-control" id="name"
placeholder="Enter Name" name="name" required>
    </div>


    <div class="form-group">
        <label for="address">Address</label>
        <input type="text" class="form-control" id="address"
placeholder="Enter Address" name="address" required>
    </div>



    <div class="form-group">
        <label for="marks">Marks</label>
        <input type="number" class="form-control" id="marks"
placeholder="Enter Marks" name="marks" required>
    </div>


    <button type="submit" class="btn btn-primary">Register</button>
</form>


{% if students %}

<hr>

<table class="table table-primary">

    <thead>
        <tr>
            <th>Roll Number</th>
            <th>Student Name</th>
```

```
            <th>Student Address</th>
            <th>Student Marks</th>
        </tr>

    </thead>


    <tbody>

        {% for student in students %}

        <tr>
            <td>{{student.roll}}</td>
            <td>{{student.name}}</td>
            <td>{{student.address}}</td>
            <td>{{student.marks}}</td>
        </tr>

        {% endfor %}
    </tbody>

</table>

{% endif %}

{% endblock %}
```

Note: The students list is currently stored globally, which works fine for now but may lead to issues with concurrency in larger-scale applications. In a production environment, it's recommended to store it permanently in a database or inside a file for better scalability and data persistence.

Storing inside a file:

- If you want data to persist even after the server restarts, you can store the student information in a file (e.g., a JSON or CSV file). This is still a simple solution, but it can be more reliable than using a global variable for storing persistent data.

- Modify the app.py file as follows:

app.py:

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

app = Flask(__name__)


def load_students():
    try:
        with open("students.json", "r") as f:
            students = json.load(f)
            return students
    except Exception as e:
        print(f"An Error occurred {e}")
        return []




def save_students(students):
    with open("students.json", "w") as f:
        json.dump(students, f)


@app.route("/")
def index():
    return render_template("index.html")


@app.route("/student", methods=["GET", "POST"])
def student():
    students = load_students()  # loads the student data from the file

    if request.method == "POST":
        # Fetch the form data
        roll = request.form.get("roll")
```

```
        name = request.form.get("name")
        address = request.form.get("address")
        marks = request.form.get("marks")

        # compose the dictionary object
        student = {"roll": int(roll), "name": name,
                   "address": address, "marks": int(marks)}
        # appending to the global list
        students.append(student)

        save_students(students)  # Save the updated list of students

    return render_template("student.html", students=students)


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

- **With the above implementation, our student data is stored persistently in a `students.json` file. Even after restarting the server, the data will not be lost. Additionally, this approach avoids the overhead of maintaining a global `students` list in memory on the server, making it more scalable in small applications.**