Web Development

Introduction:

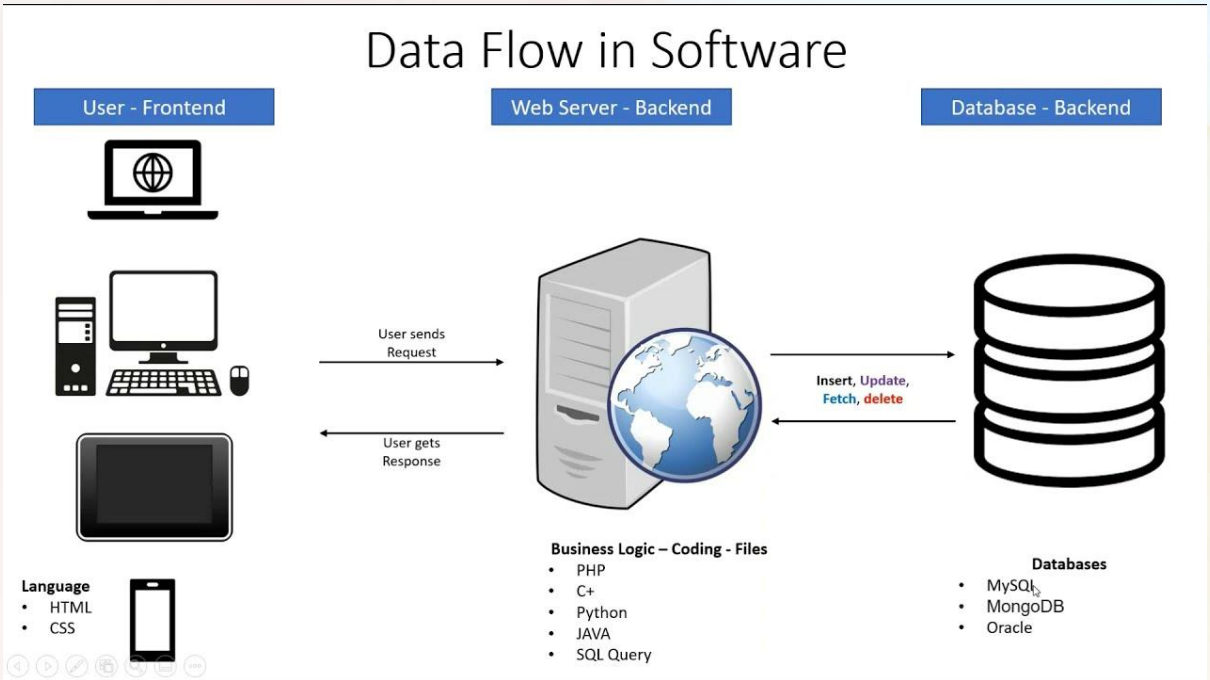
Web development is a broad area, but it refers to creating software that can be viewed online through a web browser. There are two parts to a web application: the front end and the back end.

Front-End:

The portion of a web application that controls what users can view, interact with, and how their interactions should appear is called the frontend. It is what users see when they go to a URL and access a web application on their web browser.

Backend:

The portion of a web application that handles its server-side components is called the backend. Back-end the components of a website that a user doesn't see or actively engage with are referred to as development. The back end manages user request processing, database interaction, application logic, and algorithms.



What is Flask?

Flask is an API of Python that allows us to build web applications. It was developed by Armin Ronacher.

There are many modules or frameworks which allow building your webpage using python like a bottle, Django, Flask, etc. But the real popular ones are Flask and Django. Django is easy to use as compared to Flask but Flask provides you with the versatility to program with.

To understand what Flask is you have to understand a few general terms:

1. **WSGI**
   Web Server Gateway Interface (WSGI) has been adopted as a standard for Python web application development. WSGI is a specification for a universal interface between the web server and the web applications.
2. **Werkzeug**
   It is a WSGI toolkit, which implements requests, response objects, and other utility functions. This enables building a web framework on top of it.
3. **jinja2**
   jinja2 is a popular templating engine for Python. A web templating system combines a template with a certain data source to render dynamic web pages.

Advantages of Flask:

1. Flask is a **lightweight** backend framework with minimal dependencies.
2. Flask is **easy to learn** because it's simple and intuitive API makes it easy to learn and use for beginners.
3. Flask is a **flexible Framework** because it allows you to customize and extend the framework to suit your needs easily.
4. Flask can be used with **any database** like:- SQL and NoSQL and with **any Frontend Technology** such as React or Angular.
5. Flask is **great for small to medium projects** that do not require the complexity of a large framework.

Installation of Flask:

python -m pip install flask

Working with Flask:

Python3 is required for the installation of the Python Web Framework Flask. You can start by importing Flask from the Flask Python package on any Python IDE.

```
from flask import Flask
app = Flask(__name__)   # Flask constructor

# A decorator used to tell the application
# which URL is associated function
@app.route('/')

def hello():
    return 'HELLO'

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

The '/' URL is bound with hello() function. When the home page of the webserver is opened in the browser, the output of this function will be rendered accordingly.

The Flask Python application is started by calling the run() function.

The debug support is enabled so as to track any error.

```
app.run(debug=True)
```

Build Flask Routes in Python:

Nowadays, the web frameworks provide routing technique so that user can remember the URLs.

It is useful to access the web page directly without navigating from the Home page.

It is done through the following route() decorator, to bind the URL to a function.

```
@app.route(‘/hello’)

# binding to the function of route
def hello_world():
    return ‘hello world’
```

If a user visit http://localhost:5000/hello URL, the output of the hello_world() function will be rendered in the browser.

The add_url_rule() function of an application object can also be used to bind URL with the function as in above example.

```
def hello_world():
    return ‘hello world’

app.add_url_rule(‘/’, ‘hello’, hello_world)
```

Variables in Flask:

The Variables in the Python Web Framework flask is used to build a URL dynamically by adding the variable parts to the rule parameter. This variable part is marked as. It is passed as keyword argument.

```
from flask import Flask
app = Flask(__name__)

# routing the decorator function hello_name
@app.route('/hello/<name>')
def hello_name(name):
    return 'Hello %s!' % name

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

Open the browser and enter the URL –

http://localhost:5000/hello/NSUT_CS.

Output: Hello NSUT_CS!

The parameter of route() decorator contains the variable part attached to the URL '/hello' as an argument. Hence, if URL like http://localhost:5000/hello/NSUT_CS is entered then 'NSUT_CS' will be passed to the hello() function as an argument.

```
from flask import Flask
app = Flask(__name__)

@app.route('/blog/<postID>')
def show_blog(postID):
    return 'Blog Number %d' % postID

@app.route('/rev/<revNo>')
def revision(revNo):
    return 'Revision Number %f' % revNo

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

Input:

```
http://localhost:5000/blog/555
```

Output:

```
Blog Number 555
```

Input:

```
http://localhost:5000/rev/1.1
```

Output:

```
Revision Number: 1.100000
```

Build a URL in Flask:

Dynamic Building of the URL for a specific function is done using `url_for()` function. The function accepts the name of the function as first argument, and one or more keyword arguments.

```python
from flask import Flask, redirect, url_for
app = Flask(__name__)

@app.route('/admin')   # decorator for route(argument)
function
def hello_admin():   # binding to hello_admin call
     return 'Hello Admin'

@app.route('/guest/<guest>')
def hello_guest(guest):   # binding to hello_guest call
     return 'Hello %s as Guest' % guest

@app.route('/user/<name>')
def hello_user(name):
     if name == 'admin':   # dynamic binding of URL to
function
          return redirect(url_for('hello_admin'))
     else:
          return redirect(url_for('hello_guest',
guest=name))

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

Input: http://localhost:5000/user/admin

Output: Hello Admin

Input: http://localhost:5000/user/ABC

Output: Hello ABC as Guest

he above code has a function named user(name), accepts the value through input URL. It checks that the received argument matches the 'admin' argument or not. If it matches, then the function hello_admin() is called else the hello_guest() is called.

HTTP method is provided by Flask:

Python Web Framework Flask support various HTTP protocols for data retrieval from the specified URL, these can be defined as:-

Method	Description
GET	This is used to send the data in a without encryption of the form to the server.
HEAD	provides response body to the form
POST	Sends the form data to server. Data received by POST method is not cached by server.
PUT	Replaces current representation of target resource with URL.
DELETE	Deletes the target resource of a given URL

GET Method

The GET method is used to request data from a specified resource. It should not be used to modify data on the server. It's like asking for information without making any changes.

Example:

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

@app.route('/greet', methods=['GET'])
def greet():
      name = request.args.get('name', 'Guest')

      return f"Hello, {name}!"


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

In this example, when you access the URL `/greet?name=Mohan`, the server retrieves the name parameter from the URL and returns `"Hello, Mohan!"`. If the name parameter is not provided, it defaults to `"Guest"`.

POST Method

The POST method is used to send data to a server to create/update a resource. It's like submitting a form where data needs to be processed or stored on the server.

Example:

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

@app.route('/submit', methods=['POST'])
def submit():
    data = request.form.get('data')
    return f"Data received: {data}"

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

In this example, when data is submitted to the URL `/submit` via a POST request, the server processes the data sent in the form. For instance, if you send a POST request with the form data `data=Hello`, it will return "`Data received: Hello`".

Request and Response Objects in Flask:

In Flask, the request and response objects are essential for handling incoming HTTP requests and sending back HTTP responses. Here's a breakdown of their usage:

Request Object:

- **Import:** from flask import request
- **Purpose:** The request object provides access to various details about the incoming request, such as:
    - **Form data:** request.form
    - **Query parameters:** request.args
    - **JSON data:** request.get_json()
    - **Headers:** request.headers
    - **Method:** request.method
    - **URL:** request.url
    - **Files:** request.files

Example:

```
from flask import Flask, request

app = Flask(__name__)

@app.route('/hello', methods=['GET', 'POST'])

def hello():
    if request.method == 'POST':
        name = request.form.get('name')
        return f'Hello, {name}!'
    else:
        return 'Hello, World!'

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

Response Object:

- **Import:** from flask import make_response
- **Purpose:** The response object represents the HTTP response that Flask sends back to the client. It can be created using:
  - **Direct return:** Flask automatically converts the return value of a view function into a response object (e.g., a string, a dictionary, a tuple).
  - make_response function: For more control over the response, you can use from flask import make_response.

Example:

```
from flask import Flask, make_response

app = Flask(__name__)

@app.route('/json')
def json_response():
    data = {'message': 'Hello, from Flask!'}
    response = make_response(data)
    response.headers['Content-Type'] = 'application/json'
    return response

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

FLASK
(CREATING FIRST SIMPLE APPLICATION)

Installation:

We will require two packages to set up your environment. ***virtualenv*** for a user to create multiple Python environments side-by-side. Thereby, it can avoid compatibility issues between the different versions of the libraries and the next will be ***Flask*** itself.

virtualenv

```
pip install virtualenv
```

Create Python virtual environment

```
virtualenv venv
```

Activate virtual environment

```
windows > venv\Scripts\activate
linux > source ./venv/bin/activate
```

For windows, if this is your first time running the script, you might get an error like below:

venv\Scripts\activate: File C:\flask_project\venv\Scripts\Activate.ps1 cannot be loaded because running scripts is disabled on this system.

For more information, see about_Execution_Policies at https:/go.microsoft.com/fwlink/?LinkID=135170.

At line:1 char: 1
+venv\Scripts\activate
+FullyQualifiedErrorId: UnauthorizedAccess

This means that you don't have access to execute the scripts.

To solve this error, run the powershell as admin, when you right click on powershell icon, choose the option 'Run as adminstrator'. Now, the powershell will open in the admin mode.

Type the following command in Shell

```
set-executionpolicy remotesigned
```

Now, you will be prompted to change the execution policy. Please type A. This means Yes to all.

Now again, **Activate virtual environment**

```
windows > venv\Scripts\activate
```

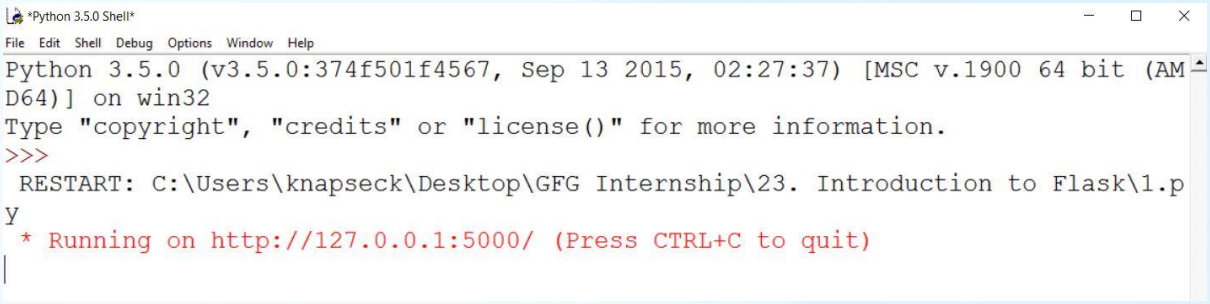
Install Flask

```
pip install Flask
```

After completing the installation of the package, let's get our hands on the code.

```
# Importing flask module in the project is mandatory
# An object of Flask class is our WSGI application.
from flask import Flask

# Flask constructor takes the name of
# current module (__name__) as argument.
app = Flask(__name__)

# The route() function of the Flask class is a decorator,
# which tells the application which URL should call
# the associated function.
@app.route('/')
# '/' URL is bound with hello_world() function.
def hello_world():
    return 'Hello World'

# main driver function
if __name__ == '__main__':

    # run() method of Flask class runs the application
    # on the local development server.
    app.run()
```

Save it in a file and then run the script we will be getting an output like this.



Then go to the URL given there you will see your first webpage displaying hello world there on your local server.

Digging further into the context, the **route()** decorator in Flask is used to bind a URL to a function. Now to extend this functionality our small web app is also equipped with another method **add_url_rule()** which is a function of an application object that is also available to bind a URL with a function as in the above example, route() is used.

Example:

```
def gfg():
   return ‘Hi NSUT’
app.add_url_rule(‘/’, ‘g2g’, gfg)
```

Output:

```
Hi NSUT
```

You can also add variables in your web app, well you might be thinking about how it'll help you; it'll help you to build a URL dynamically. So, let's figure it out with an example.

```
from flask import Flask
app = Flask(__name__)

@app.route('/hello/<name>')
def hello_name(name):
    return 'Hello %s!' % name

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

And go to the URL *http://127.0.0.1:5000/hello/geeksforgeeks* it'll give you the following output.

We can also use HTTP methods in Flask let's see how to do that The HTTP protocol is the foundation of data communication on the world wide web. Different methods of data retrieval from specified URL are defined in this protocol. The methods are described down below.

GET : *Sends data in simple or unencrypted form to the server.*

HEAD : *Sends data in simple or unencrypted form to the server without body.*

PUT : *Replaces target resource with the updated content.*

DELETE : *Deletes target resource provided as URL.*

By default, the Flask route responds to the GET requests. However, this preference can be altered by providing methods argument to route() decorator. In order to demonstrate the use of the POST method in URL routing, first, let us create an HTML form and use the POST method to send form data to a URL.

Now let's create an HTML login page:

```
<html>
   <body>
      <form action = "http://localhost:5000/login" method = "post">
         <p>Enter Name:</p>
         <p><input type = "text" name = "nm" /></p>
         <p><input type = "submit" value = "submit" /></p>
      </form>
   </body>
</html>
```

Now save this file HTML and try this python script to create the server.

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


@app.route('/success/<name>')
def success(name):
      return 'welcome %s' % name

@app.route('/login', methods=['POST', 'GET'])
def login():
      if request.method == 'POST':
            user = request.form['nm']
            return redirect(url_for('success', name=user))
      else:
            user = request.args.get('nm')
            return redirect(url_for('success', name=user))


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

After the development server starts running, open login.html in the browser, enter your name in the text field and click *submit* button. The output would be the following:

The result will be something like this