Unit – 4

Development with Flask

Some Basic Info :

1. **Framework :**
   A framework is like a structure that provides a base for the application development process. With the help of a framework, you can avoid writing everything from scratch. Frameworks provide a set of tools and elements that help in the speedy development process. It acts like a template that can be used and even modified to meet the project requirements.Frameworks are based on programming languages.

2. **Web Application :**
   A **web application** is a software program that runs on a web server and can be accessed by users through a web browser. Unlike a traditional desktop application that runs locally on your computer, a web application is hosted on the internet and is usually accessible from anywhere with a network connection.

- **Runs in a browser**: Users don't need to install the application on their device. They just open a browser (like Chrome, Firefox, etc.) and visit the app's URL.
- **Interactivity**: Web applications allow users to interact with the application, like submitting forms, sending messages, or performing tasks online (e.g., buying products, posting on social media).
- **Server-based**: The logic, processing, and data are managed on a server, not on the user's device.
- **Languages and frameworks**: Web apps are typically built using various programming languages and frameworks. For Python, common frameworks include **Flask** and **Django**.

Examples:

- Gmail
- Facebook
- Online banking systems

3. **Database Integration :**

Database integration in simple words means connecting an application (like a web app) to a database so they can work together. The database stores important information (like user data, messages, or product details), and the application can **retrieve** (get), **add**, **update**, or **delete** that information.

Key Points:

- **Database**: A place to store data in an organized way (like a big storage room for information).
- **Integration**: Connecting the application to the database so they can communicate.
- **Purpose**: To store and manage data (e.g., user accounts, orders) that the app needs.

4.1 Introduction to Flask Framework

Flask is a lightweight, flexible, and modular web framework written in Python. It allows developers to build web applications quickly and efficiently. Here's an overview:

Key Features:

1. Microframework: Flask has a small codebase, making it easy to learn and use.

2. Modular: Flask supports extensions and libraries for added functionality.

3. Flexible: Flask allows developers to choose their own database, template engine, and other components.

4. RESTful: Flask supports RESTful API development.

5. Unit testing: Flask provides built-in support for unit testing.

Core Components:

1. Routes: Define URL endpoints for handling requests.

2. Views: Functions that handle requests and return responses.

3. Templates: Render HTML templates using Jinja2 templating engine.

4. Request and Response objects: Handle HTTP requests and responses.

5. Application instance: The central configuration and initialization point.

Advantages:

1. Easy to learn and use

2. Fast development and prototyping

3. Lightweight and scalable

4. Extensive community support

5. Integrates well with other libraries and frameworks

Use cases:

1. Web applications (e.g., blogs, portfolios)

2. RESTful APIs

3. Microservices architecture

4. Web services (e.g., authentication, payment gateways)

5. IoT applications

Basic Example:

```
from flask import Flask, render_template

app = Flask(__name__)

@app.route("/")

def index():

    return render_template("index.html")

if __name__ == "__main__":

    app.run()
```

This example creates a simple web application with a single route ("/") that renders an index.html template.

4.2 Creating web application using flask

1. Install Flask

First, you need to install Flask on your system. Open your terminal or command prompt and run:

pip install flask

2. Create a Python File

Next, you need to create a Python file for your Flask app. Let's call it app.py.

3. Set Up Flask Basic Structure

In your app.py, write the following code to set up the basic Flask app:

```
from flask import Flask

# Create the Flask app

app = Flask(__name__)

# Define a route

@app.route('/')

def home():

    return "Welcome to my first Flask web app!"

# Run the app

if __name__ == '__main__':

    app.run(debug=True)
```

Breaking Down the Code:

- **Import Flask**: This line imports the Flask module.
- **Create the App**: app = Flask(__name__) creates the web app.
- **Define a Route**: @app.route('/') specifies that when someone visits the root URL (e.g., http://127.0.0.1:5000/), the home function will run and display "Welcome to my first Flask web app!".
- **Run the App**: app.run(debug=True) starts the server. The debug=True allows you to see any errors and reloads automatically when you make changes.

3. Run the Flask App

Now, go to your terminal, navigate to the folder where app.py is saved, and run this command: python app.py

4. You should see something like this in the terminal: Running on http://127.0.0.1:5000/
5. Open your browser and visit http://127.0.0.1:5000/. You'll see:

   Welcome to my first Flask web app!

4.3 Flask extension for database integration

Flask is a lightweight Python web framework that provides useful tools and features for creating web applications in the Python Language. SQLAlchemy is an SQL toolkit that provides efficient and high-performing database access for relational databases. It provides ways to interact with several database engines such as SQLite, MySQL, and PostgreSQL. It gives you access to the database's SQL functionalities. It also gives you an Object Relational Mapper (ORM), which allows you to make queries and handle data using simple Python objects and methods. Flask-SQLAlchemy is a Flask extension that makes using SQLAlchemy with Flask easier, providing you tools and methods to interact with your database in your Flask applications through SQLAlchemy.

Flask extensions are Python packages that provide Flask applications with extra functionality. Although these extensions are not a part of the Flask core, they were made to function well with Flask. Adding features like user authentication,

database integration, caching, email support, and many more are possible using Flask extensions.
Adding Flask extensions to your application is a simple process that involves a few steps. Here are the steps you need to follow:

1) **Install the extension using pip:** The first step is to install the extension using pip. This can be done from the command line using the following command:

```
pip install <extension_name>
```

Replace <extension_name> with the name of the extension you want to install.
2) Import the extension: Once the extension is installed, you can import it into your Flask application using the "import" statement.
3) Initialize the extension: After importing the extension, you need to initialize it with the Flask application object. This is typically done in the main application file (e.g., app.py). For example, to use the Flask-Login extension, you would initialize it like this:

```
from flask_login import LoginManager


login_manager = LoginManager(app)
```

In this example, the "app" variable represents the Flask application object.

Configure the extension: Finally, you may need to configure the extension to work with your application. This can typically be done using configuration variables in your Flask application's configuration file (e.g., config.py).