Unit: 4 – Flask Framework

Prepared By: Prince Kumar *Assistant Professor, CSE*

This unit introduces the fundamentals of web development using the Flask framework. Students will learn how to build dynamic web applications using Python, including routing, templates, database connectivity, authentication, and deployment.

Topics Covered

- Introduction to Flask and Web Development
- Installation and Virtual Environment Setup
- Routing and Application Configuration
- URL Building
- HTTP Methods
- Templates (Jinja2)
- Static and Media Files
- Form Handling
- Database Connectivity (SQLite3 and MySQL)
- Exception and Error Handling
- Flash Messages
- Email Integration
- Authentication and Authorization (Flask-Login)
- Deployment of Flask Application

*****

1. Introduction to Flask and Web Development

Q. What is Web Development?

Ans. Web development means creating websites and web applications that run on the internet. It involves designing, building, and maintaining websites.

There are mainly two parts of web development:

A. Frontend (Client-Side)

B. Backend (Server Side)

A. Frontend (Client-Side)

Frontend is the part of a website that users can see and interact with directly in their web browser (like Chrome, Edge, etc.).

It focuses on:

- Design
- Layout
- Colors and fonts
- Buttons and forms
- User experience

Simple words: Frontend = What users see and click.

Technologies Used in Frontend

HTML (Hypertext Markup Language)

- Creates the **structure** of the webpage.
- Used for headings, paragraphs, images, forms, buttons.

Example:

- Login form
- Registration form
- Headings and text

CSS (Cascading Style Sheets)

- Adds **style and design** to the webpage.
- Controls colors, fonts, spacing, layout.

Example:

- Background color
- Button color
- Font size

JavaScript

- Adds **interactivity** to the webpage.
- Makes the page dynamic.

Example:

- Button click actions
- Form validation
- Showing/hiding content
- Dropdown menus

B.Backend (Server-Side)

Backend is the part of a website that works behind the scenes. Users cannot see it, but it is very important for the website to function properly.

Simple words: Backend = The brain of the website.

It handles the main logic, processes user requests, and manages data.

Main Functions of Backend

- Handles Application Logic
- Stores and Retrieves Data
- Manages User Authentication
- Connects Frontend with Database

Technologies Used in Backend

1. **Python:** Used to write backend logic.

   Frameworks:

   - Flask
   - Django

2. **Databases:** Used to store data permanently.

   Examples:

   - MySQL
   - MongoDB

Q. What is Flask?

Ans. Flask is a web framework written in Python that helps developers build web applications easily.

It allows us to connect:

- 🌐 Web pages (HTML)
- Python code
- 🗄 Databases

So basically, Flask helps create the backend of a website.

Q. Why is Flask called Lightweight?

Ans. Flask is called a lightweight (micro) framework because:

- It is simple
- It has fewer built-in features
- It gives developers freedom to add only what they need

Key Points Explained

1. Written in Python

Flask is built using the Python programming language, so if you know Python, you can easily learn Flask.

2. Simple and Beginner-Friendly

- Easy to understand
- Clean and short code
- Perfect for students and new developers

3. Used for Backend Development

Backend means the server-side part of a website.

Backend handles:

- Login verification
- Saving data
- Connecting to database
- Processing forms

Flask helps us do all this.

4. Connects Web Pages with Python Code

When a user clicks a button or submits a form:

- Flask receives the request
- Python processes the data
- Flask sends a response back

👉 This makes the website interactive.

Examples of Flask Usage

Flask is used to build:

- 🔐 Login systems (username & password verification)
- 🎓 Student portals
- 📝 Online forms
- 🔄 REST APIs (used by mobile apps & websites to share data)

Q. What is a Framework?

Ans. A framework is a ready-made structure that helps developers build applications quickly and easily.

Instead of writing everything from the beginning, a framework provides:

- Pre-written code
- Built-in tools
- Proper structure
- Rules to follow

👉 This saves time, effort, and reduces errors.

Real-Life Example

- 🏗️ **Building a house from scratch**
  - You design everything yourself
  - Takes more time
  - More difficult

- 🏠 **Using a ready-made design**
  - Structure is already planned
  - Faster construction
  - Easier work

👉 A framework is like a ready house design for software development.

2.Installation and Virtual Environment Setup

1. Install Python

Before using Flask, you must install Python.
Steps:

1. Go to the official Python website.
2. Download the latest version.
3. Install it.
4. During installation, ✓ check "Add Python to PATH".

To verify installation, open Command Prompt / Terminal and type: python –version
If Python is installed correctly, it will show the version number.

2. What is a Virtual Environment?

A virtual environment is a separate workspace for your project.
It allows you to:

- Install project-specific libraries
- Avoid conflicts between different projects
- Keep your system clean

👉 Each project can have its own packages and versions.

3. Create a Virtual Environment

Step 1: Open terminal / command prompt
Go to your project folder.
Step 2: Create virtual environment
python -m venv venv
Here:

- venv = name of the virtual environment folder

4. Activate the Virtual Environment

- On Windows: venv\Scripts\activate
- On Mac/Linux: source venv/bin/activate
- After activation, you will see: (venv) in front of your terminal line. That means the virtual environment is active.

5. Install Flask

After activating the virtual environment, install Flask: pip install flask

To check installation: pip list
You will see Flask in the list.

6. Why Use Virtual Environment?

❖ **Without virtual environment:**

- Packages may conflict
- Different projects may break
- System becomes messy

❖ **With virtual environment:**

- Clean project
- No conflicts
- Easy management

3.Routing and Application Configuration

Q. What is Routing?

Ans. Routing means connecting a URL (web address) to a function in the program.
In simple words:

👉 When a user types a URL, Flask decides which function should run.

Example of Routing:

```
from flask import Flask

app = Flask(__name__)

@app.route("/")
def home():
    return "Welcome to Home Page"

@app.route("/about")
def about():
    return "This is About Page"

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

Explanation:

- / → Calls the home() function
- /about → Calls the about() function
- @app.route() → Defines the URL

Q. What is Application Configuration?

Application configuration means **setting rules and settings** for the Flask application.
It controls:

- Debug mode
- Security key
- Database settings
- Application behavior

Example of Configuration

```
app.config["DEBUG"] = True
app.config["SECRET_KEY"] = "mysecretkey"
```

4.URL Building, HTTP Methods, Templates & Static Files

Q. What is URL Building?

Ans. URL building means generating URLs dynamically using the url_for() function. Instead of writing URL paths manually, Flask creates them automatically.

Q. Why use url_for()?

- Avoid hardcoding URLs
- If route changes, no need to update everywhere
- Makes code clean and maintainable

Syntax:

url_for('function_name')

Example:

```
from flask import Flask, url_for

app = Flask(__name__)

@app.route('/')
def home():
    return "Home Page"

@app.route('/about')
def about():
    return url_for('home')
```

Q. What are HTTP Methods?

Ans. HTTP methods define what action the client wants to perform.

Common HTTP Methods:

Method	Purpose
GET	To get data from server
POST	To send data to server
PUT	To update data
DELETE	To delete data

Example (GET & POST):

```
@app.route('/login', methods=['GET', 'POST'])
def login():
    return "Login Page"
```

Q. What are Templates?

Ans. Templates allow us to write HTML with dynamic data.
Flask uses Jinja2 template engine.

Q. Why Templates?

- Separate Python code from HTML.
- Display dynamic data.
- Example: app.py

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

@app.route('/')
def home():
    return render_template('index.html', name="Prince")
```

- **index.html (inside templates folder)**

```
<h1>Hello {{ name }}</h1>
```

Important Jinja2 Syntax:

Syntax	Use
{{ variable }}	Print variable
{% if %}	Conditional
{% for %}	Loop

Q. What are Static Files?

Static files are files that do not change:

- CSS
- JavaScript
- Images

5. Form Handling, Database Connectivity (SQLite3 & MySQL), Exception and Error Handling, Flash Messages, and Email Integration.

Q. What is Form Handling?

Form handling means collecting data from users (like login form, registration form) and processing it in the backend.

Steps in Form Handling

1. Create HTML Form
2. Use POST method
3. Get data using request.form
4. Process data in Flask route

Example

HTML Form (login.html)

```
<form method="POST">
        Name: <input type="text" name="username">
         <input type="submit">
</form>
```

Flask Code

```
from flask import Flask, render_template, request

app = Flask(__name__)

@app.route('/login', methods=['GET', 'POST'])
def login():
  if request.method == 'POST':
    name = request.form['username']
        return "Welcome " + name
        return render_template('login.html')

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

Points

- request.form → Used to get form data
- POST → Used to send data securely
- Always specify methods=['GET','POST']

2. Database Connectivity in Flask

Flask supports many databases like:

- SQLite3 (Lightweight, built-in)
- MySQL (Large applications)

A) SQLite3 in Flask

What is SQLite?

- Small database
- No separate server required
- Stored in a file

Example (SQLite)

```
import sqlite3

conn = sqlite3.connect('student.db')
cursor = conn.cursor()

cursor.execute("CREATE TABLE IF NOT EXISTS students(id INTEGER
PRIMARY KEY, name TEXT)")
cursor.execute("INSERT INTO students(name) VALUES('Prince')")
conn.commit()
conn.close()
```

B) MySQL in Flask

To connect MySQL:

Install: pip install mysql-connector-python

Example:

```
import mysql.connector

db = mysql.connector.connect(
host="localhost",
user="root",
password="1234",
database="testdb"
)

cursor = db.cursor()
cursor.execute("SELECT * FROM students")
```

Exam Difference Table

Feature	SQLite	MySQL
Server Required	No	Yes
Best For	Small Apps	Large Apps
Speed	Fast (small data)	Better for big data

Exception and Error Handling in Flask

Q.What is Exception Handling?

Ans.Handling errors properly so that the application does not crash.

Python Exception Example

```
try:
  x = 10 / 0
except ZeroDivisionError:
  print("Cannot divide by zero")
```

Flask Error Handling

Custom Error Page:

```
@app.errorhandler(404)
def page_not_found(error):
  return "Page Not Found", 404
```

Common Errors

- 404 → Page Not Found
- 500 → Internal Server Error

Flash Messages in Flask

Q. What is Flash Message?

Flash messages show temporary messages to users.

Example:

- Login successful
- Invalid password
- Registration completed

Steps to Use Flash

1. Set secret key
2. Use flash()
3. Display in template

Example

```
from flask import Flask, flash, render_template

app = Flask(__name__)
app.secret_key = "secret"

@app.route('/')
def home():
  flash("Login Successful!")
  return render_template('home.html')
```

In HTML

```
{% with messages = get_flashed_messages() %}
    {% for message in messages %}
       <p>{{ message }}</p>
    {% endfor %}
{% endwith %}
```

Important for Exam

- Flash requires secret_key
- Used for one-time messages

Email Integration in Flask

Flask can send emails using SMTP.

Install Flask-Mail

pip install flask-mail

Example

```
from flask import Flask
from flask_mail import Mail, Message

app = Flask(__name__)

app.config['MAIL_SERVER'] = 'smtp.gmail.com'
app.config['MAIL_PORT'] = 587
app.config['MAIL_USE_TLS'] = True
app.config['MAIL_USERNAME'] = 'your_email@gmail.com'
app.config['MAIL_PASSWORD'] = 'your_password'

mail = Mail(app)
```

```
@app.route('/send')
 def send_mail():
        msg = Message("Hello",
        sender="your_email@gmail.com",
        recipients=["student@gmail.com"])
        msg.body = "This is a test email"
        mail.send(msg)
        return "Email Sent!"
```

Q. What is Authentication?

Ans. Authentication means verifying the identity of a user.

Example:

- Login with username & password
- OTP verification

If credentials are correct → User is authenticated.

Q. What is Authorization?

Ans. Authorization means checking what the user is allowed to access.

Example:

- Admin can delete data
- Student can only view data

Q. Difference Between Authentication & Authorization

Authentication	Authorization
Who are you?	What can you access?
Login process	Permission control
Happens first	Happens after login

Q. What is Flask-Login?

Flask-Login is an extension that helps manage:

- User login
- User logout
- User session
- Remember logged-in users

It handles authentication easily.

Install Flask-Login
pip install flask-login

Basic Steps in Flask-Login

1. Import Flask-Login
2. Create LoginManager
3. Create User class
4. Define user_loader
5. Use login_user()
6. Use logout_user()
7. Protect routes using @login_required

Example

```
from flask import Flask, render_template
from flask_login import LoginManager, UserMixin, login_user, login_required, logout_user

app = Flask(__name__)
app.secret_key = "secretkey"

login_manager = LoginManager()
login_manager.init_app(app)

# Dummy User Class
class User(UserMixin):
    def __init__(self, id):
        self.id = id

@login_manager.user_loader
def load_user(user_id):
    return User(user_id)

@app.route('/login')
def login():
    user = User(1)
    login_user(user)
    return "User Logged In"

@app.route('/dashboard')
@login_required
def dashboard():
```

```
    return "Welcome to Dashboard"
@app.route('/logout')
def logout():
    logout_user()
    return "Logged Out"

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

Important Functions

Function	Purpose
login_user(user)	Logs in user
logout_user()	Logs out user
@login_required	Protects route
current_user	Gets current logged-in user

Points

- Flask-Login manages sessions
- Requires secret key
- Used for secure web applications
- Helps implement role-based access control

Q. What is Deployment?

Ans. Deployment means making your Flask application available online so users can access it from anywhere.

Development vs Deployment

Development	Deployment
Runs on local machine	Runs on server
Debug mode ON	Debug mode OFF
Not secure	Secure

Steps for Deployment

1. Turn off debug mode
2. Use Production Server (Gunicorn / uWSGI)
3. Choose hosting platform
4. Upload project
5. Configure environment variables

Common Deployment Platforms

- PythonAnywhere
- Heroku
- AWS
- Render
- Railway

Example: Running in Production

Instead of:

app.run(debug=True)

Use:

```
app.run()
```

Or use Gunicorn:

```
pip install gunicorn
gunicorn app:app
```

Files Required for Deployment

- app.py
- requirements.txt
- Procfile (for some platforms)
- templates folder
- static folder

Important Environment Variables

- SECRET_KEY
- DATABASE_URL
- MAIL_USERNAME
- MAIL_PASSWORD

Points

- Deployment makes app public
- Debug mode should be OFF
- Production server is required
- Hosting platforms are used

*****