Flask

---

1. Overview

Flask is a lightweight and flexible **Python micro web framework** designed for simplicity and easy extensibility.

- Created by Armin Ronacher and first released in 2010.
- Minimalist core, with no built-in database or form validation.
- Allows developers to pick libraries/tools as needed.
- Ideal for small to medium-sized web applications and APIs.
- Uses **Werkzeug** as the underlying WSGI toolkit and **Jinja2** as the template engine.

---

2. Architecture

- Based on **WSGI** (Web Server Gateway Interface), the Python standard for web applications.
- Modular design:
    - Core handles routing, request/response.
    - Extensions provide ORM, authentication, forms, etc.
- Supports synchronous request handling (async supported experimentally in newer versions).

---

3. Core Concepts

Application Instance

- Central object: Flask class instance.
- Configures URL routes, error handlers, middleware, extensions.

Example:

```
from flask import Flask

app = Flask(__name__)
```

Routing

- Maps URLs to Python functions called **view functions**.
- Uses **decorators** for route definitions.

```
@app.route('/')
def home():
  return "Hello, Flask!"
```

Request & Response

- request object represents incoming HTTP requests.
- Access data like form, query parameters, headers.
- Response objects represent outgoing HTTP responses.
- Supports JSON responses with jsonify().

Templates

- Uses **Jinja2** templating engine.
- Enables embedding Python-like expressions in HTML.
- Supports template inheritance and filters.

Example:

```
<!DOCTYPE html>
<html>
<head><title>{{ title }}</title></head>
<body>
 <h1>Welcome, {{ user }}</h1>
</body>
</html>
```

Static Files

- Served from the /static folder by default.

- Includes CSS, JavaScript, images.

---

4. HTTP Methods

- Supports all HTTP verbs: GET, POST, PUT, DELETE, PATCH.
- Methods specified in route decorator.

Example:

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

---

5. Extensions

- Flask is lightweight by design, so features like ORM, forms, authentication are provided by extensions.
- Popular extensions:
    - **Flask-SQLAlchemy**: ORM integration.
    - **Flask-WTF**: Forms and CSRF protection.
    - **Flask-Login**: User authentication.
    - **Flask-Migrate**: Database migrations.
    - **Flask-RESTful**: Building REST APIs.

---

6. Database Integration

- No built-in ORM; commonly paired with SQLAlchemy via Flask-SQLAlchemy.
- Supports relational databases (PostgreSQL, MySQL, SQLite) and NoSQL via third-party libs.

---

7. Security

- CSRF protection via Flask-WTF.
- Support for session management.

- HTTPS recommended for deployment.
- Developers responsible for input validation and secure coding.

---

8. Blueprint

- Flask supports modular apps via **Blueprints**.
- Enables breaking an app into reusable components.

Example:

```
from flask import Blueprint


auth = Blueprint('auth', __name__)


@auth.route('/login')
def login():
  return "Login Page"
```

---

9. Debugging & Development

- Built-in debugger and development server.
- Supports **debug mode** with auto-reload on code changes.
- Logging support.

---

10. Deployment

- Flask apps run on WSGI servers like **Gunicorn** or **uWSGI**.
- Often deployed behind Nginx or Apache reverse proxy.
- Can be containerized with Docker.
- Supports cloud deployment platforms: Heroku, AWS, GCP, Azure.

---

11. Advantages

- Simple and flexible; easy to learn.

- Great for prototyping and small projects.
- Modular design with lots of third-party extensions.
- Fine-grained control over components.
- Large community and good documentation.