Flask Web Framework Notes

Table of Contents

1. Introduction to Flask
2. Installation
3. Basic Application Structure
4. Routing
5. Templates
6. Forms
7. Database Integration with Flask-SQLAlchemy
8. User Authentication
9. Error Handling
10. Deployment

1. Introduction to Flask

Flask is a lightweight WSGI web application framework in Python. It is designed to make getting started quick and easy, with the ability to scale up to complex applications. Flask is classified as a micro-framework because it does not require particular tools or libraries.

Key Features:

* Simple and easy to use.
* Built-in development server and debugger.
* Integrated support for unit testing.
* RESTful request dispatching.
* Jinja2 templating engine.
* Support for secure cookies.

2. Installation

To install Flask, you need to have Python installed on your system. You can install Flask using pip:

bash

pip install Flask

Verify Installation

To verify that Flask is installed correctly, you can run the following command in your Python shell:

python

**import** flask

**print**(flask.\_\_version\_\_)

3. Basic Application Structure

A basic Flask application consists of a single Python file. Here is a simple structure:

text

/my\_flask\_app

├── app.py

├── templates/

│ └── index.html

└── static/

└── style.css

Example app.py

python

**from** flask **import** Flask, render\_template

app = Flask(\_\_name\_\_)

@app.route('/')

**def** home():

**return** render\_template('index.html')

**if** \_\_name\_\_ == '\_\_main\_\_':

app.run(debug=True)

4. Routing

Flask uses decorators to define routes. A route is a URL that is associated with a function.

Basic Route Example

python

@app.route('/')

**def** home():

**return** "Welcome to the Home Page!"

@app.route('/about')

**def** about():

**return** "This is the About Page."

Dynamic Routes

You can also create dynamic routes that accept parameters.

python

@app.route('/user/<username>')

**def** profile(username):

**return** f"Hello, {username}!"

5. Templates

Flask uses the Jinja2 templating engine to render HTML templates. Templates allow you to separate your application logic from the presentation layer.

Example Template (index.html)

xml

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Home Page</title>

</head>

<body>

<h1>Welcome to Flask!</h1>

</body>

</html>

Rendering Templates

You can render templates using the render\_template function.

python

@app.route('/')

**def** home():

**return** render\_template('index.html')

6. Forms

Flask provides a way to handle forms using the request object.

Example Form Handling

xml

<form action="/submit" method="POST">

<input type="text" name="username" placeholder="Enter Username">

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

</form>

Handling Form Submission

python

**from** flask **import** request

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

**def** submit():

username = request.form['username']

**return** f'Username: {username}'

7. Database Integration with Flask-SQLAlchemy

Flask-SQLAlchemy is an extension that simplifies database interactions.

Installation

bash

pip install Flask-SQLAlchemy

Example Configuration

python

**from** flask\_sqlalchemy **import** SQLAlchemy

app.config['SQLALCHEMY\_DATABASE\_URI'] = 'sqlite:///site.db'

db = SQLAlchemy(app)

**class** User(db.Model):

id = db.Column(db.Integer, primary\_key=True)

username = db.Column(db.String(150), nullable=False)

Creating the Database

python

**with** app.app\_context():

db.create\_all()

8. User Authentication

Flask can be used to create user authentication systems. Flask-Login is a popular extension for managing user sessions.

Installation

bash

pip install Flask-Login

Basic Authentication Setup

python

**from** flask\_login **import** LoginManager, UserMixin

login\_manager = LoginManager(app)

**class** User(UserMixin, db.Model):

*# User model definition*

@login\_manager.user\_loader

**def** load\_user(user\_id):

**return** User.query.get(int(user\_id))

Login Route Example

python

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

**def** login():

*# Handle login logic*

9. Error Handling

Flask provides a way to handle errors gracefully.

Custom Error Pages

python

@app.errorhandler(404)

**def** not\_found(error):

**return** render\_template('404.html'), 404

Handling Exceptions

You can also handle exceptions using try and except blocks.

10. Deployment

When you're ready to deploy your Flask application, you can use a WSGI server like Gunicorn.

Example Command

bash

gunicorn -w 4 app:app

Deployment Options

* Heroku
* AWS Elastic Beanstalk
* DigitalOcean

Conclusion

Flask is a powerful and flexible web framework that is perfect for building web applications quickly and efficiently. With its simple syntax and extensive documentation, it is an excellent choice for both beginners and experienced developers. Feel free to expand on each section with more examples, explanations, or illustrations as needed to reach your desired document length. You can also add code snippets, diagrams, or additional resources to enhance the content. If you need further customization or additional topics, let me know!