Flask 101: Crafting Your First Python Web Application

Kollin R. Napier, Ph.D.

Agenda

- Introduction
  - What is Flask?
  - What does Flask do?
  - Why use Flask?
  - Who uses Flask?
  - Examples of using Flask
- Install and Use Flask
- Hello World
- Routing
  - Dynamic
- Templates and Static Files
  - Bootstrap
- Databases
- Conclusion

What is Flask?

- "*A simple framework for building complex web applications*."
- Micro web framework written in Python
- Lightweight and Flexible
- Dependencies
  - Werkzeug Toolkit (WSGI: Web Server Gateway Interface)
  - Jinja2 Template Engine
  - Click CLI Toolkit
  - MarkupSafe, ItsDangerous, Blinker
- Version 3.0.0 (September 30, 2023)

Flask

web development,
one drop at a time

Flask

What Does Flask Do?

- Routing
  - Maps URLs to Python functions, making it easy to define how your web app responds to client requests
    - Dynamic
- Templates
  - Supports rendering templates to create dynamic web pages with HTML and Python-like expressions
    - Bootstrap
- Development Server
  - Comes with a built-in development server and debugger for easy testing
    - Real-time testing
- Extensions
  - Offers a wide range of extensions for tasks like form validation, authentication, and database integration
    - Databases, Authentication, APIs, etc.

Why Use Flask?

- Simplicity and Flexibility
  - Designed to be simple and easy to use, providing the flexibility to choose how you want to implement things
- Rapid Development
  - Quick to set up and start developing, making it ideal for small to medium-sized web applications and for developers who want to hit the ground running
- Great Documentation
  - Well-documented with a large community and plenty of resources for learning and troubleshooting
- Scalability
  - While simple, Flask applications can be scaled to handle large amounts of traffic and complex functionality
- Alternatives
  - Django, Tornado, Bottle

Who Uses Flask?

- Startups and Small Companies
  - Due to its simplicity and rapid development capabilities
- Large Companies
  - Some parts of larger applications or for internal tools
    - Pinterest, Zillow, Patreon, Samsung, Netflix, Uber, Reddit, etc.
- Individual Developers
  - For personal projects, prototypes, or learning web development
- Educational Institutions
  - Often used in computer science and web development courses

Examples of Using Flask

- Web Applications
  - From simple web pages to complex web-based applications
- APIs
  - Creating RESTful APIs for mobile or front-end applications
- Blogs and Content Management Systems
  - Though there are more specialized tools for these, Flask provides the flexibility to build such systems from scratch
- Prototyping and Learning
  - An ideal choice for developing prototypes and for educational purposes due to its simplicity and ease of use

Install and Use Flask

- Check Python version
  - Windows
    - Command Prompt
    - python --version
    - python3 --version
  - macOS (OS X)
    - Terminal
    - python --version
    - python3 --version
  - Example:
    - Python 3.12.0
- Install / Check Flask
  - pip install Flask
  - pip show Flask
  - pip3 install Flask
    - Flask, Werkzeug, Jinja2, itsdangerous, click, blinker, MarkupSafe
  - pip3 show Flask
    - Version 3.0.0
- Create directory for development
  - Create folder “my_flask”
  - Navigate to folder

https://github.com/krn65/flask_example

Hello World!

- Create a simple Flask web application that outputs “Hello World!” to the user
  - Import Flask class
  - Create Flask class instance
  - Use route() decorator to bind a function to URL
  - Run development server only when app is executed

```python
from flask import Flask

app = Flask(__name__)

@app.route('/')
def hello_world():
    return 'Hello, World!'

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

```python
# Importing the Flask class from the 'flask' package.
# Flask class is a WSGI application and acts as the central object of your application.
from flask import Flask

# Creating an instance of the Flask class. The first argument is the name of the
# application's module or package, which is used by Flask to find resources,
# templates, static files, instance folder, etc. '__name__' is a built-in
# variable which evaluates to the name of the current module. This is needed so
# that Flask knows where to look for templates, static files, and so on.
app = Flask(__name__)

# The route() decorator is used to bind a function to a URL. Here, we are defining
# the 'route' for the root URL, which is '/'. This means that when a web browser
# requests the root URL, Flask will invoke the 'hello_world' function below.
@app.route('/')
def hello_world():
    # This function is called when the root URL is accessed. It returns a string,
    # which will be displayed on the client's web browser. The return value can
    # also be more complex, such as an HTML template.
    return 'Hello, World!'

# This conditional is used to ensure the server is only run when the script is
# executed directly and not imported as a module. If we import this script as a
# module in another script, we may not want to start a web server. Hence, this
# check is crucial.
if __name__ == '__main__':
    # Runs the Flask application on the local development server.
    # The 'debug=True' parameter enables debug mode. This mode allows us to see
    # the interactive debugger and reloader, which makes it easier to debug the
    # application. The server will reload itself on code changes, and it will
    # also provide detailed error pages on exceptions.
    app.run(debug=True)
```

Hello World!

```
 * Serving Flask app '1-hello_world'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 292-598-648
```

Routing

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

@app.route('/')
def hello_world():
    return 'Hello, World!'

@app.route('/about')
def about():
    return 'About Page'

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

Routing (Dynamic)

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

@app.route('/')
def hello_world():
    return 'Hello, World!'

@app.route('/about')
def about():
    return 'About Page'

@app.route('/user/<username>')
def show_user_profile(username):
    return f'Hello User: {username}!'

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

Hello User: bob!

Dynamic Routing Usage

- User Profiles
- Blog Posts or Articles
  - /post/<post_id>
- Product Pages
  - /product/<product_id>
- Navigation Based on Categories or Tags
  - /category/<category_name>
- API Endpoints
  - /api/users/<user_id>
- Date-Based Filtering
  - /archive/<year>/<month>/<day>
- Location-Based Services
  - /location/<city>
- Content Management Systems (CMS)
  - For creating a CMS where editors can add pages with unique URLs, dynamic routing allows these pages to be accessed without the need to create a specific route for each one.

Templates and Static Files

- Templates
  - Contain layout and structure
  - Allow for dynamic content
    - Placeholders
- Pros
  - Separation of concerns
  - Reusability
  - Dynamic content
  - Template Inheritance
  - Easy to learn and use
- Cons
  - Performance
  - Complexity
  - Security Risks

- Static Files:
  - Do not change based on app state or user action
  - Served “as-is”
    - Images, CSS, JS, etc.
- Pros
  - Performance
  - Caching
  - Ease of use
- Cons
  - Lack of dynamics
  - Version control
  - Separate management

Templates

- Create folder “templates”
  - Flask expects to follow a “/templates/file” path by default
- Create HTML file “index.html”
  - Basis for your template
- Establish parameter in template
  - Pass value for parameter

```python
from flask import Flask, render_template

app = Flask(__name__)

@app.route('/')
def home():
    # Pass the name "Visitor" to the template
    return render_template('index.html', name='Visitor')

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

```html
<html>
<body>
    <h1>Welcome to the Home Page</h1>
    <p>Hello, {{ name }}!</p>
</body>
</html>
```

Static Files

- Create folder “static”
  - Flask expects to follow a “/static/file” path by default
- Create HTML file “style.css”
- Update index.html with path for static styling
- Apply, save, refresh

```css
body {
    color: white;
    background-color: black;
}
```


```html
<html>
<body>
    <h1>Welcome to the Home Page</h1>
    <p>Hello, {{ name }}!</p>
    <link rel="stylesheet" href="{{ url_for('static', filename='style.css') }}">
</body>
</html>
```

Databases

- SQLAlchemy
  - Database Toolkit for Python
- pip install SQLAlchemy
  - Successfully installed greenlet-3.0.1 sqlalchemy-2.0.23 typing-extensions-4.8.0
- pip show SQLAlchemy
  - Version 2.0.23

SQLA

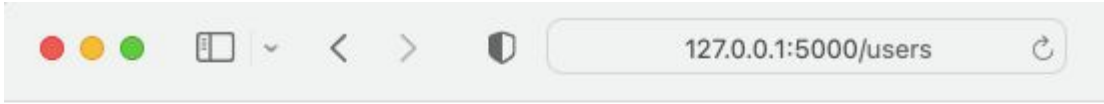
SQLAlchemy

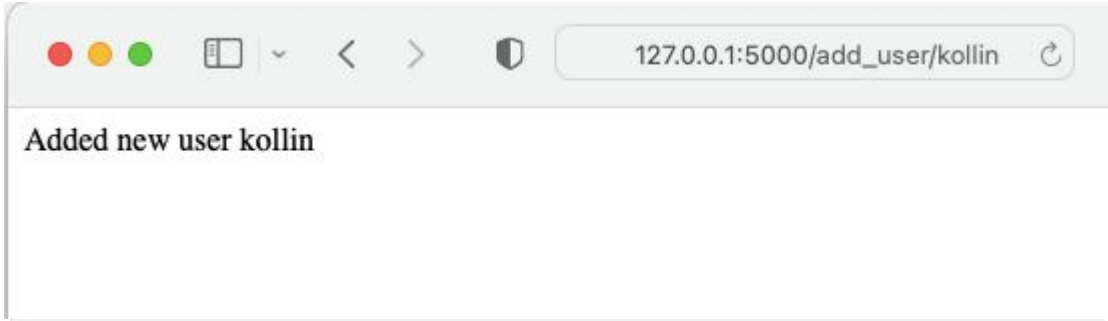
- Flask-SQLAlchemy
  - Extension for Flask that adds support for SQLAlchemy to your application
- pip install Flask-SQLAlchemy
  - Successfully installed Flask-SQLAlchemy-3.1.1
- pip show Flask-SQLAlchemy
  - Version 3.1.1

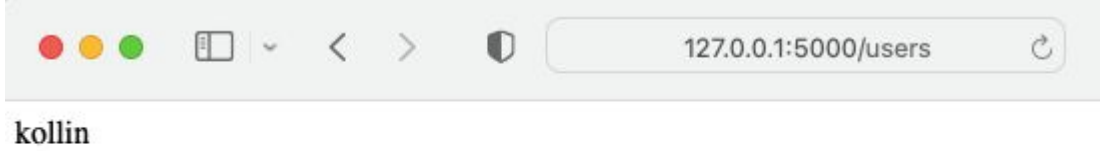
Flask SQLAlchemy

Using Databases

- Create a database of users
- Initialize empty database
- View current users
  - /users
- Add new user
  - /add_user/{USER}
  - Example: /add_user/kollin
- View current users again
  - /users
  - Should see “kollin” as a user now
  - Record was created







Conclusion

- Key Takeaways
  - Flask is a lightweight and flexible microframework.
  - Quick setup and easy to learn for beginners.
  - Core concepts: routing, templates, and static files.
  - Extensible: can add more functionalities as needed.
  - Great for both small projects and scalable for larger applications.

- Resources
  - Official documentation
  - https://flask.palletsprojects.com/en/3.0.x/
  - Tutorials / Guides
    - Real Python: realpython.com
    - freeCodeCamp: freecodecamp.org
    - Full Stack Python: fullstackpython.com
  - Book
    - "Flask Web Development" by Miguel Grinberg
  - GitHub
    - Explore and contribute to Flask projects

Flask 101: Crafting Your First Python Web Application

Kollin R. Napier, Ph.D.