22 Flask Web app

Flask is a powerful and flexible micro web framework for Python, ideal for both small and large web projects.

Get Started

Create a Virtual Environment

```
mkdir flask_board
cd flask_board

python -m venv venv

# On Windows
.\venv\Scripts\activate

# On Linux or Mac
source venv/bin/activate
```

Add Dependencies

```
python -m pip install Flask
```

Initiate Your Flask Project

Initiate your project by start writing in `app.py`

Run the Flask Development Server

A Flask project can be as basic as one Python file, which is commonly named `app.py`

```
# app.py
from flask import Flask

app = Flask(__name__)

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

```
if __name__ == "__main__":
    app.run(host="0.0.0.0", port=8000, debug=True)
```

Functions like these are commonly referred to as **views** in Flask.
In this case, the `home` view returns a `"Hello, World!"` string that the browser can display.

Transform Your Project Into a Package

Since you're just at the start of the project, your to-do list to create the package has only three tasks:

1. Creating a package folder named `board/`
2. Moving `app.py` into `board/`
3. Renaming `app.py` to `__init__.py`

```
mkdir board
mv app.py board/__init__.py
```

With the commands above, you create the `board/` sub-folder. Then, you move `app.py` into `board/` and rename it to `__init__.py`.

By now, your Flask project structure should look like this:

```
flask_board/
│
└── board/
    └── __init__.py
```

With the new package structure in place, make sure that you stay in the `flask_board/` folder and run your Flask project with this command:

```
python -m flask --app board run --port 8000 --debug
```

With the command above, you're telling Flask to look for an app named `board` and serve it in debug mode on port `8000`.

When you visit `http://localhost:8000`, you see the same *Hello, World!* displayed as before.

Work With an Application Factory

An **application factory** in Flask is a design pattern that's highly beneficial for scaling Flask projects. With an application factory, you can create and configure your Flask project flexibly and efficiently, making it easier to develop, test, and maintain as it grows and evolves.

Instead of having your application's code at the root level of your `__init__.py` file, you'll work with a function that returns your application. To do so, replace the content of `__init__.py` with the content below:

```
from flask import Flask

def create_app():
    app = Flask(__name__)

    return app
```

Leverage Blueprints

Flask Blueprints encapsulate **functionality**, such as views, templates, and other resources. Each Flask Blueprint is an **object** that works very similarly to a Flask application. They both can have resources, such as static files, templates, and views that are associated with routes.

However, a Flask Blueprint is not actually an application. It needs to be registered in an application before you can run it.

Blueprints contain related views that you can conveniently import in `__init__.py`

Create a file named `pages.py` in the `board/` folder and add the content below:

```
# pages.py
# flask_board/board/pages.py
from flask import Blueprint
bp = Blueprint("pages", __name__)

@bp.route("/")
def home():
    return "Hello, Home!"

@bp.route("/about")
def about():
    return "Hello, About!"
```

You define two routes, one as the `home` view and the other as the `about` view. Each of them returns a string to indicate on which page you are on.

Before you can visit the pages, you need to connect the `pages` blueprint with your Flask project:

```
# __init__.py
# flask_board/board/__init__.py
from flask import Flask

from board import pages

def create_app():
    app = Flask(__name__)

    app.register_blueprint(page.pb)
    return app
```

When you visit `http://localhost:8000/about`, then you can see the *Hello, About!* string:

Introduce Templates

Templates are HTML files with the capability of rendering dynamic content sent over from your Flask views. For this, Flask uses the popular Jinja **template engine**.

Jinja is a template engine commonly used for web templates that receive dynamic content from the back end and render it as a static page in the front end. But you can use Jinja without a web framework running in the background.

With a template engine like Jinja, you can embed Python-like expressions into your templates to create dynamic web content. You can use loops, conditionals, and variables directly in your templates.

Jinja also supports **template inheritance**, allowing the creation of a **base template** that you can extend in **child templates**, promoting code reuse and a consistent layout across different pages of your project.

Build a Base Template

Create a new template named `base.html` in a `templates/` directory inside `board/`:

```
<!-- base.html -->
<!-- flask_board/board/templates/base.html -->
<!DOCTYPE html>
<html lang="en">
    <head>
        <title>Message Board - {% block title %}{% endblock title %}</title>
```

```
    </head>
    <body>
        <h1>Message Board</h1>
        <section>
            <header>
                {% block header %}{% endblock header %}
            </header>
            <main>
                {% block content %}<p>No message.</p>{% endblock content %}
            </main>
        </section>
    </body>
</html>
```

Commonly, the Jinja base template is named `base.html`, and it contains the main HTML structure of your web pages.

Add Your Child Templates

Start with the template for your `home` view by creating a `home.html` file in a `pages/` subfolder of your `templates/` folder:

```
<!-- home.html -->
<!-- flask_board/board/templates/pages/home.html -->
{% extends 'base.html' %}

{% block header %}
    <h2>{% block title %}Home{% endblock title %}</h2>
{% endblock header %}

{% block content %}
    <p>
        Learn more about this project by visiting the <a href="{{
url_for('pages.about')}}"></a>
    </p>
{% endblock content %}
```

Child templates also contain `{% block %}` tags. By providing the block's name as an argument, you're connecting the blocks from the child template with the blocks from the parent template.

When you use `url_for()`, Flask creates the full URL to the given view for you. This means that if you change the route in your Python code, then the URL in your templates updates automatically.

Continue by creating a file named `about.html` in the same `pages/` directory:

```
<!-- about.html -->
<!-- flask_board/board/templates/pages/about.html -->
{% extends 'base.html' %}

{% block header %}
    <h2>{% block title %}About{% endblock title %}</h2>
{% endblock header %}

{% block content %}
    <p>This is a message board for friendly messages.</p>
{% endblock content %}
```

With both child templates in place, you need to adjust your views to return the templates instead of the plain strings.

```
# pages.py
# flask_board/board/pages.py
from flask import Blueprint, render_template

bp = Blueprint("pages", __name__)

@bp.route("/")
def home():
    return render_template("pages/home.html")

@bp.route("/about")
def about():
    return render_template("pages/about.html")
```

By default, Flask expects your templates to be in a `templates/` directory. Therefore, you don't need to include `templates/` in the path of the templates.

Improve the User Experience

A good user experience is crucial in any web application. It ensures that users find the application not only convenient to use, but also enjoyable.

Include a Navigation Menu

The navigation of a website is usually displayed on every page. With your base and child template structure, it's best to add the code for the navigation menu into `base.html`.

Instead of adding the navigation menu code directly into `base.html`, you can leverage the `{% include %}` tag. By referencing another template with `{% include %}`, you're loading the whole template into that position.

Included templates are partials that contain a fraction of the full HTML code. To indicate that a template is meant to be included, you can prefix its name with an underscore (`_`).

Follow the prefix-based naming scheme and create a new template named `_navigation.html` in your `templates/` folder:

```
<!-- _navigation.html -->
<!-- flask_board/board/templates/_navigation.html -->
<nav>
    <ul>
        <li><a href="{{ url_for('pages.home') }}">Home</a></li>
        <li><a href="{{ url_for('pages.about') }}">About</a></li>
    </ul>
</nav>
```

Include `_navigation.html` in `base.html` to display your navigation menu on all of your pages:

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <title>Message Board - {% block title %}{% endblock title %}</title>
    </head>
    <body>
        <h1>Message Board</h1>
        {% include("_navigation.html") %}
        <section>
          <header>
            {% block header %}{% endblock header %}
          </header>
          <main>
            {% block content %}<p>No messages.</p>{% endblock content %}
          </main>
        </section>
    </body>
</html>
```

Since both `home.html` and `about.html` extend `base.html`, you didn't need to make any changes in these templates to make the navigation menu appear.

Make Your Project Look Nice

The templates in your project provide the logic and structure for the **front end** of your project. With the help of **Cascading Style Sheets (CSS)**, you can style the content.

In Flask projects, you commonly save CSS files in a `static/` directory.

Create a new directory named `static/` inside of `board/` and place a file named `styles.css` in it with the **CSS declarations** below:

```
/* styles.css */
/* flask_board/board/static/styles.css */
* {
  box-sizing: border-box;
}

body {
  font-family: sans-serif;
  font-size: 20px;
  margin: 0 auto;
  text-align: center;
}

a,
a:visited {
  color: #007bff;
}

a:hover {
  color: #0056b3;
}

nav ul {
  list-style-type: none;
  padding: 0;
}

nav ul li {
  display: inline;
  margin: 0 5px;
}

main {
  width: 80%;
```

```
  margin: 0 auto;
}
```

With the style sheet in place, you need to add a link to `styles.css` inside `<head>` of `base.html`:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Message Board - {% block title %}{% endblock title %}</title>
  <link rel="stylesheet" href="{{ url_for('static', filename='styles.css')
}}">
</head>
<body>
<!-- ... -->
</body>
</html>
```

By adding the CSS file reference to `base.html`, you're again taking advantage of the inheritance mechanism that Jinja templates provide. You only need to add `styles.css` in your base template to make it present in your child templates.