Build a Simple Flask Web Project

Create a Virtual Environment

```
$ mkdir rp_flask_board
$ cd rp_flask_board
…..
$ python -m venv venv
$ source venv/bin/activate
(venv) $
```

Add Dependencies

```
(venv) $ python -m pip install Flask
```

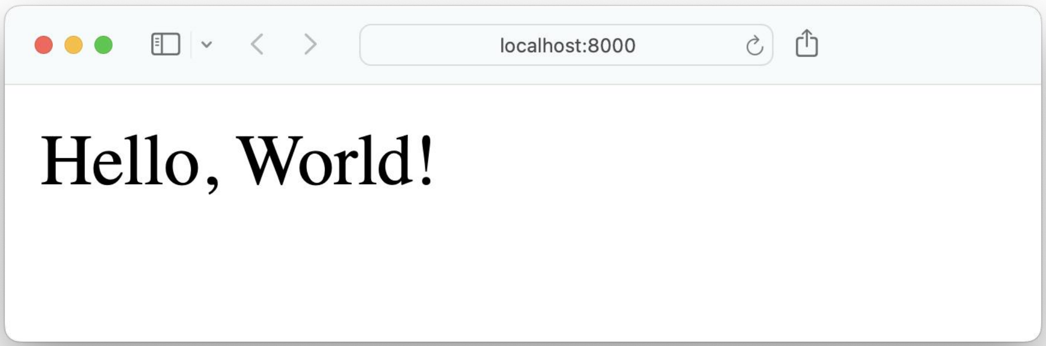
Run the Flask Development Server

creating app.py in rp_flask_board/ and adding the following content:

```
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)
```

```
(venv) $ python app.py
```

When you run app.py, a web server will start on port 8000. If you open a browser and navigate to http://localhost:8000, then you should see Hello, World! displayed:



Hello, World!

Transform Your Project Into a Package

```
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)
```

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

```
(venv) $ mkdir board
(venv) $ mv app.py board/__init__.py
```

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

```
rp_flask_board/
 |
 └──── board/
    └──── __init__.py
```

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

```
(venv) $ python -m flask --app board run --port 8000 --debug
```

Work With an Application Factory

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:

```
board/__init__.py
```

```
from flask import Flask

def create_app():
    app = Flask(__name__)

    return app
```

Leverage Blueprints

Blueprints are modules that contain related views that you can conveniently import in __init__.py. For example, you'll have a blueprint that stores the main pages of your project. Create a file named pages.py in the board/ folder and add the content below:

board/pages.py

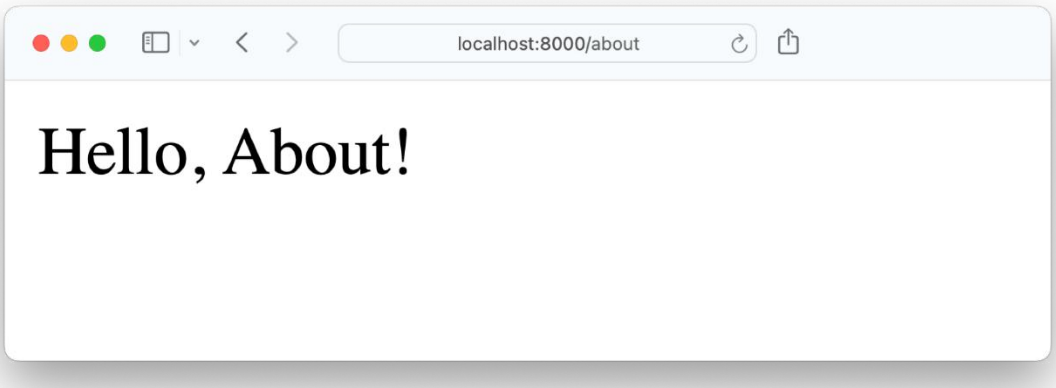
```
from flask import Blueprint

bp = Blueprint("pages", __name__)

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

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

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

board/__init__.py

```
from flask import Flask

from board import pages

def create_app():
    app = Flask(__name__)

    app.register_blueprint(pages.bp)
    return app
```

Once you've imported and registered the pages blueprint in your application factory, you can visit the routes that you defined in pages.py in your browser. When you visit http://localhost:8000/about, then you can see the Hello, About! string:


localhost:8000/about

Hello, About!

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, which you already installed as a dependency when you installed Flask.

Build a Base Template

Create a new template named base.html in a templates/ directory inside 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 messages.</p>{% endblock content %}
 </main>
</section>
</body>
</html>
```

Add Your Child Templates :

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') }}">About page</a>.
  </p>
{% endblock content %}
```

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

board/templates/pages/about.html

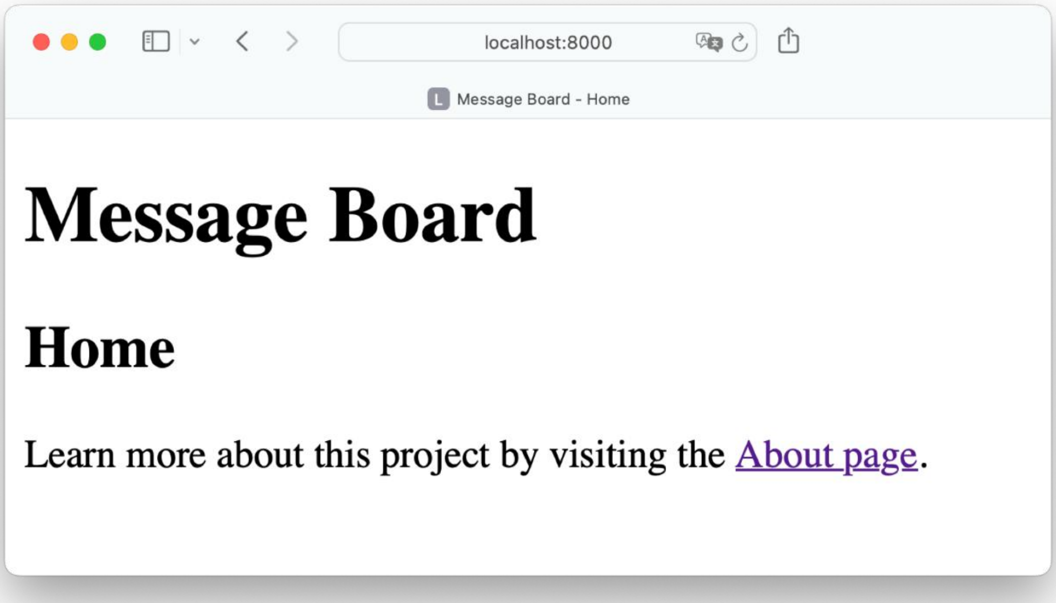
```
{% extends 'base.html' %}

{% 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. Hop over to pages.py and adjust home() and about():

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")
```

To verify that your templates work, restart your Flask development server and visit http://localhost:8000:



Message Board

Home

Learn more about this project by visiting the About page.

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

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:

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:

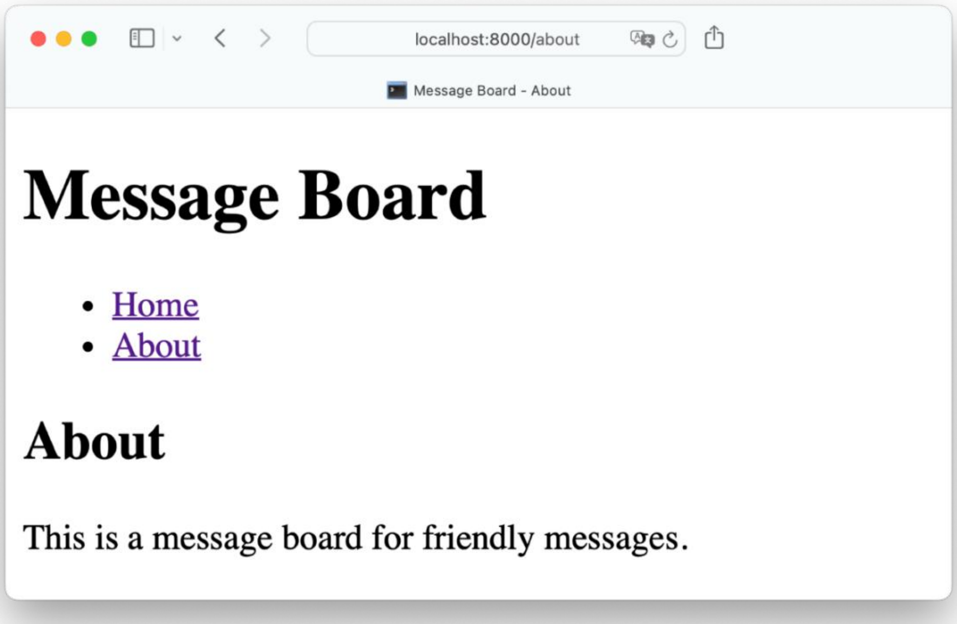
templates/base.html

```
<!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>
```

Instead of adding the navigation menu code directly into base.html, you include _navigation.html in your website's header.

Visit http://localhost:8000/about to check out your new navigation menu:



Message Board

- Home
- About

About

This is a message board for friendly messages.

Make Your Project Look Nice:

board/static/styles.css

```
* {
 box-sizing: border-box;
}

body {
 font-family: sans-serif;
 font-size: 20px;
 margin: 0 auto;
```

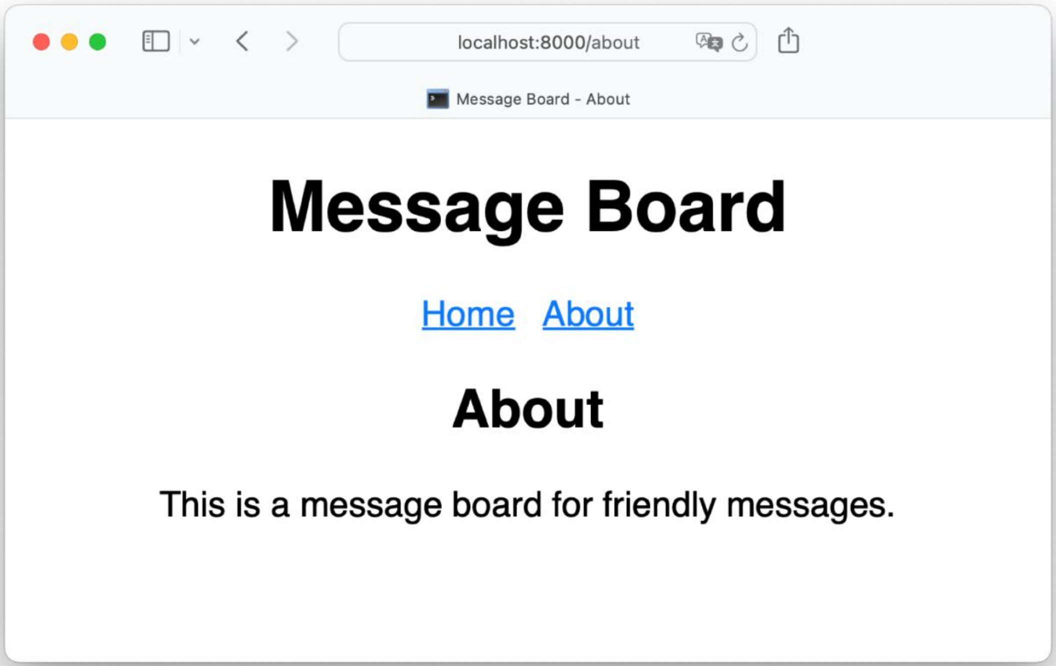
```
  text-align: center;
}

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;
}
```

The color scheme for the links provides a visual cue to users about clickable elements. The font choice and size contribute to readability. The overall layout and alignment of the web elements give the website a clean and modern look. With the style sheet in place, you need to add a link to styles.css inside <head> of base.html:

templates/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. You may need to restart your Flask development server again to see your styling in action:



Message Board

Home About

About

This is a message board for friendly messages.

Tugas

1. Buatlah akun github/gitlab.
2. Upload percobaan diatas ke repositori anda pada branch **main**, anda bebas memberikan nama repositori terkait project anda, anda juga bisa menggunakan nama sesaui dengan folder/project yang anda kerjakan saat ini.
3. Buatlan branch **dev** pada repositori anda.
4. Setelah anda mencoba membuat branch, buat kode program berdasarkan refrensi yang ada di sini. Pada poin ini anda bisa mengerjakan secara ber tim (2 orang per tim)(jangan lupa clone repositori teman anda), anda dapat menentukan repositori siapa yang akan digunakan. Simpan setiap perubahan yang anda lakukan dengan melakukan commit.
5. Setelah anda mengerjakan poin 4, upload hasil pekerjaan anda pada repositori.
6. Pemilik repositori melakukan merge branch **dev** ke branc **main**
7. Buat file "Dockerfile", kemudian tambahkan :

Dockerfile

```
# syntax=docker/dockerfile:1

FROM python:3.10-alpine
WORKDIR /flask
RUN apk add --no-cache gcc musl-dev linux-headers
COPY requirement.txt requirement.txt
RUN pip install -r requirement.txt
RUN pwd
RUN ls -a
EXPOSE 5000
COPY . .
RUN python -m flask --app board init-db
CMD [‘python‘, ‘-m‘, ‘flask‘, ‘--app‘, ‘board‘, ‘run‘, ‘--host=0.0.0.0‘, ‘--debug‘]
```

8. Jalankan :

```
$ docker build -t board-app:1.0 .
$ docker run -p 8080:5000 board-app:1.0
```

9.

Reference :

https://realpython.com/flask-project/

https://realpython.com/flask-database/