Flask Routes

Contents

Overview

Routes are very important in Flask. They provide the **URL endpoint** for your different pages as well as passing **variables** to your front-end HTML.

Routes

For a web app to navigate the user to the right page for each URL, it needs to be configured to do so. This is done using the `@app.route` decorator.

The route decorator allows us to assign our functions URLs easily.

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

This tells our application that when the domain is visited at 'www.app.com/home' to run the function associated with that route. In this instance, the function returns the phrase `Hello, World!` and will be displayed in the browser as a web page.

HTTP Requests in Routes

HTTP methods are utilised when accessing URLs for an application. There are four main methods to remember:

- `GET` requests are used to retrieve data from a server, such as a web page or information from an API.
- `POST` requests are used to create data on the server based on information that is sent with the request.
- `PUT` requests are used to update existing data on the server.
- `DELETE` requests are used to delete existing data on the server.

The `app.route()` decorator includes a `methods` parameter which can be used to allow different HTTP methods.

```
methods=['GET', 'POST', 'PUT', 'DELETE']
```

This parameter expects a `list` of methods. This allows you to provide a range of methods that are allowed to be received by this route.

If you do not specify any methods, the route will default to only allowing `GET` requests.

Flask parses through the request data and creates an object with various attributes called `request` (which must be imported from the Flask module).

```
from flask import request

@app.route('/home', methods=['GET', 'POST'])
def home():
    if request.method == 'POST':
        return 'This is a POST request'
    else:
        return 'This is a GET request'
```

Using conditional logic, we can use the `request` object to change the functionality of this route – and therefore what the page displays – depending on the method that was sent, as shown in the above example.

Dynamic URLs

In Flask routes you can enter variables into the URL like so:

```
@app.route('/home/<word>')
```

Any characters entered into the URL where `<word>` is specified can then be passed through as arguments to the route's function.

```
@app.route('/home/<word>')
def home(word):
    return word.upper()
```

We can then manipulate the value by passing it into the accompanying function, as above with `home(word)`. For example, if you were to navigate to the application with the URL `my-app.com/home/hello`, the web page will display `HELLO` as we use the `upper()` string method to make the value of `word` uppercase.

Dynamic URL variables are string types by default, but you can make it an integer with `int`:

```
@app.route('/home/<int:number>')
```

Here the variable name is **number** and it is cast as an integer data type.

Redirect Function

The `redirect()` function allows requests to be redirected to another URL. This may be done if they can't be fulfilled within the route, or when the user needs to be sent to a new page based on their interaction with the application.

`redirect()` needs to be imported from the `flask` module in order to be used.

```
from flask import redirect

@app.route('/home')
def home():
    if user_logged_in() == True:
        return redirect('http://my-app/account')
    else:
        return 'This is a home page'
```

This example uses a custom function to check if the user of the application is logged in. If they are, they will be redirected to their account page. If the user is not logged in they are taken to the home page.

url_for()

The `url_for()` function, when imported from the `flask` module, retrieves the associated URL endpoint for a given route function.

The parameter we pass to `url_for()` is the name of the **function**.

For example, if we have the following route for the user account page:

```
@app.route('/account')
def account():
    return render_template('account.html', title='My Account')
```

We can use `url_for()` in conjunction with `redirect()` to programmatically find the URL for the `account` route. This way, the redirection logic will continue to work even if the route's URL endpoint is changed.

```
from flask import redirect, url_for

@app.route('/home')
def home():
    if user_logged_in() == True:
        return redirect(url_for('account'))
    else:
        return 'This is a home page'
```

We can also use `url_for()` in conjunction with dynamic URLs to pass values to our routes. For example, if we had this route in our app:

```
@app.route('/home/<int:number>')
def home(number):
    return number + 1
```

The command `url_for('home', number=1)` would return the URL `/home/1`, passing through the value of `1` to the `home` route via a dynamic URL. The webpage would then display the number `2`.

Tutorial

This tutorial will walk you through how to make a basic web app that uses routes to provide a `/home` and an `/about` page.

Setup

This tutorial assumes you are working on an Ubuntu VM, at least version `18.04 LTS`.

First, install apt dependencies:

```
sudo apt update
sudo apt install python3 python3-venv python3-pip
```

Create a directory named `flask-routes` and make it your current working directory:

```
mkdir flask-routes && cd $_
```

We now need to create a Python virtual environment to install our pip requirements in. Create a new virtual environment named `venv` and activate it:

```
python3 -m venv venv
source venv/bin/activate
```

And install Flask with pip.

```
pip3 install flask
```

Build the application

Create a new file called `app.py`:

```
touch app.py
```

Open `app.py` with your text editor of choice and paste in the following code:

```
from flask import Flask

app = Flask(__name__)

@app.route('/')
@app.route('/home')
def home():
    return 'This is the home page'

@app.route('/about')
def about():
    return 'This is the about page'

if __name__ == "__main__":
    app.run(debug=True, host='0.0.0.0', port=5000)
```

There are three routes in this application: `/`, `/home` and `/about`. Each route is associated with a function, such that when the web app is accessed at any of those locations the function beneath it will be invoked.

In this example, two routes (`/` and `/home`) have been associated with the `home()` function, meaning that if the user went to either of these locations they would retrieve the information returned by the `home()` function.

Start the application

To start the app, run:

```
python app.py
```

You can then access the application in your browser either via your machine's public IP on port `5000`, or via `localhost` if you are running the app locally. You may need to allow network traffic on port `5000`.

Try accessing the application at each route: `/`, `/home` and `/about`. Check your terminal and take note of how Flask logs all incoming HTTP requests to the app while it's running.

Clean Up

To stop your Flask application running, navigate back to your terminal and press `Ctrl+C`. You should now have control over your terminal again.

To deactivate the virtual environment, run:

```
deactivate
```

If you wish to delete the virtual environment, run:

```
rm -rf venv
```

Exercises

Create an application that reads a number in the URL and returns that number squared.

▶ Hint