CS50 Lecture 9: Flask

Jordan Mandel

2021/12/06

Web Programming

As a reminder, we can use `http-server` to test a web page. An `HTTP` request might look like

```
GET / HTTP/1.1
...
```

or can pass a parameter:

```
GET /search?q=cats HTTP/1/1
```

Where `/` means get `index.html`

Other programs return pages dynamically.

Flask

has

```
application.py
requirements.txt
static/
templates/
```

`MVC` design pattern.

Minimal flask web app:

```
from flask import Flask
app = Flask(__name__)
@app.route("/")
def index():
    return "hello, world"
```

The `@` is a decorator.

Run with `flask run`

Make it return HTML

Basic Program

```
from flask import Flask, render_template
app = Flask(__name__)
@app.route("/") # use the following function for this route
def index():
    return render_template("index.html", name=request.args.get("first_name","world"))
```

Then we put index.html into the templates directory.

`index.html` might look like

```
<!DOCTYPE html>

<html lang="en">
    <head>
        <title>hello</title>
    </head>
    <body>
        hello world, {{first_name}}
    </body>
</html>
```

Added `/?name=David`. The application can then get from the URL arguments. The route defines where the base file.

Forms

Can go:

Move the previous html into `greet.html` and make a new `index.html` file.

```
<!DOCTYPE html>

<html lang="en">
    <head>
        <title>hello</title>
    </head>
    <body>
        <form action="/greet" method="get">
            <input, autocomplete="off" autofocus name="first_name" type="text" placeholder="name">
            <input type="submit">
        </form>
    </body>
</html>
```

The action is `/greet`. That takes us to the next page. The form submits the input via `GET`.

And change our python code:

```
@app.route("/")
def index():
    return render_template("index.html")


@app.route("/greet")
def greet(): #second argument below is the default
    return render_template("greet.html", first_name=request.args.get("first_name", "world"))
```

Here the greet page gets the `name` arg from the form on the `index` page.

Post

The above form is not good because it puts user info in the URL. Should change to:

```
<form action="/greet" method="post">
```

And then change the controller to have `[POST]` as the method and `request.form` rather than `request.args`:

```
@app.route("/")
def index():
    return render_template("index.html")

@app.route("/greet", methods=["POST"])
def greet(): # it is form.get here
    return render_template("greet.html", name=request.form.get("name", "world"))
```

Layouts/Templates

Can make a reusable html template, `layout.html`:

```
<!DOCTYPE html>

<html lang="en">
    <head>
        <title>hello</title>
    </head>
    <body>
        {% block body %}{% endblock %}
    </body>
</html>
```

Now we can use `layout.html` to extend `index.html`

```
{% extends "layout.html" %}

{% block body %}

    <form action="/greet" method="post">
        <input autocomplete="off" autofocus name="name" placeholder="Name" type="text">
        <input type="submit">
    </form>

{% endblock %}
```

Can also change the `greet.html`:

```
{% extends "layout.html" %}

{% block body %}

    hello, {{ name }}

{% endblock %}
```

Post (again)

can then condense the controller python code: put thme into one route:

```
@app.route("/", methods=["GET", "POST"])
def index():
    if request.method == "POST":
        return render_template("greet.html", name=request.form.get("name", "world"))
```

```
    return render_template("index.html")
```

Note that index has the default method, `GET`. Keep in mind `GET` is what you use by default when you visit a `URL`.

Before the form's action was `/greet` but now it is just `/` because we are conditionaing on the request which can direct us to the correct page.

```
{% extends "layout.html" %}

{% block body %}

    <form action="/" method="post">
        <input autocomplete="off" autofocus name="name" placeholder="Name" type="text">
        <input type="submit">
    </form>

{% endblock %}
```

Can see the app in `src9/hello3`[

Frosh IMs

Viewport is just what we see in the web browser. Everything else, make the viewport scale with the web page.

Make basic layout:

```
<!DOCTYPE  html>
<html lang="en">
    <head>
        <meta name="viewport" content="initial-scale=1, width=device-width">
        <title>froshims</title>
    </head>
    <body>
    {% block body %}{% endblock %}
    </body>
</html>
```

Make basic flask file:

```
from flask import Flask, render_template, request

app = Flask(__name__)

@app.route("/")
def index():
    return render_template("index.html")
```

Here is index.html:

```
{% extends "layout.html" %}

{% block body %}
<h1>Register</h1>
<form action="/register" method="post">
     <input autocomplete="off" autofocus name="name" placeholder="Name" type="text">
     <select name="sport">
        <option disabled selected value="">Sport</option>
```

```
        <option value="Dodgeball">Dodgeball</option>
        <option value="Flag Football">Flag Football</option>
        <option value="Soccer">Soccer</option>
        <option value="Volleyball">Volleyball</option>
        <option value="Ultimate Frisbee">Ultimate Frisbee</option>
    </select>

    <input type="submit" value="Register">
</form>
{%endblock}
```

now we add registration content; have to include `POST`:

```
from flask import Flask, render_template, request

app = Flask(__name__)

@app.route("/")
def index():
    return render_template("index.html")

@app.route("/register", methods=["POST"]) #make sure you specify the methods.
def register():
    if not request.form.get("name") or not request.form.get("sport"):
        return render_template("failure.html")
    return render_template("success.html")
```

Now we make `success.html`:

```
{%extends "layout.html" %}
{% block body %}
You are registered!
{% endblock %}
```

and `failure.html`

```
{%extends "layout.html" %}
{% block body %}
You are not registered!
{% endblock %}
```

Jinja For Loop

Problem here is we can hack the website and send an unsupported sport.

So first we get rid of the hardcoded sports:

```
{% extends "layout.html" %}

{% block body %}
<h1>Register</h1>
<form action="register" method="post">
    <input autocomplete="off" autofocus name="name" placeholder="Name" type="text">
    <select name="sport">
        <option disabled  selected value="">Sport</option>
        {% for sport in sports %}
            <option value="{{ sport }}">{{sport}}</option>
        {% endfor %}
```

```
     </select>

     <input type="submit" value="Register">
</form>
{%endblock}
```

Then update the python (checking if the sport in the sport form is not in the global variable sports):

```
from flask import Flask, render_template, request

app = Flask(__name__)

SPORTS = [
    "Dodgeball",
    "Flag Football",
    "Soccer",
    "Volleyball"
    "Ultimate Frisbee"
]
@app.route("/")
def index():
    return render_template("index.html, sports=SPORTS") #have to pass a name

@app.route("/register", methods=["POST"])
def register():
    if not request.form.get("name") or request.form.get("sport") not in SPORTS:
        return render_template("failure.html")
    return render_template("success.html")
```

look above; we added a `for` loop with Jinja.

Can delete the `<select>` tags above and use ***radio*** buttons:

```
{% extends "layout.html" %}

{% block body %}
<h1>Register</h1>
<form action="register" method="post">
     <input autocomplete="off" autofocus name="name" placeholder="Name" type="text">
     {% for sport in sports %}
<input name="sport" type="radio" value="{{ sport }}"> {{sport}}
     {% endfor %}

     <input type="submit" value="Register">
</form>
{%endblock}
```

Actually Registering

With error.html we have:

```
{% extends "layout.html" %}
{% block body %}
{{message}}
{% endblock %}
```

make a `registrants` array and other changes, including uding redirect.

```
from flask import Flask, render_template, request, redirect

app = Flask(__name__)

REGISTRANTS = {}

SPORTS = [
    "Dodgeball",
    "Flag Football",
    "Soccer",
    "Volleyball"
    "Ultimate Frisbee"
]
@app.route("/")
def index():
    return render_template("index.html, sports=SPORTS")

@app.route("/register", methods=["POST"])
def register():
    name = request.form.get("name")
    if not name:
        return render_template("error.html", message="Missing name")
    sport = request.form.get("sport")
    if not sport:
        return render_teplate("error.html", message="Missing sport")
    if sport not in SPORTS:
        return render_template("error.html", message="Invalid sport")
    REGISTRANTS[name] = sport
    print(REGISTRANTS)
    return redirect("/registrants")

@app.route("/registrants")
def registrants():
    return render_template("registrants.html", registrants=REGISTRANTS)
```

Then we make `registrants.html`.

```
{% extends "layout.html" %}
{% block body %}
<h1> Registrants </h1>
<table>
    <thead>
        <tr>
            <th>Name</th>
            <th>Sport</th>
        </tr>
    </thead>
    <tbody>
    {% for name in registrants %}
        <tr>
            <td> {{name}} </td>
            <td> {{registrants[name]}} </td>
        </tr>
    {% endfor %}
    </tbody>
```

```
</table>
{% endblock %}
```

Adding a Database

We add to the above a registrants route:

```
from cs50 import SQL
from flask import Flask, render_template, request, redirect

app = Flask(__name__)

db = SQL("sqlite:///froshims.db")

SPORTS = [
    "Dodgeball",
    "Flag Football",
    "Soccer",
    "Volleyball"
    "Ultimate Frisbee"
]
@app.route("/")
def index():
    return render_template("index.html, sports=SPORTS")

@app.route("/register", methods=["POST"])
def register():
    name = request.form.get("name")
    if not name:
        return render_template("error.html", message="Missing name")
    sport = request.form.get("sport")
    if not sport:
        return render_teplate("error.html", message="Missing sport")
    if sport not in SPORTS:
        return render_template("error.html", message="Invalid sport")

    db.execute("INSERT INTO registrants (name, sport) VALUES(?, ?)", name, sport)
    return redirect("/registrants")

@app.route("/registrants")
def registrants():
    registrants = db.execute("SELECT * FROM registrants;")
    return render_template("registrants.html", rows=rows)
```

Deleted `registrants` variable and `print` statement. Imported redirect that does an `HTTP 310` to go somewhere else. This gets rid of some redundant code.

made an sqlite table in advance: `CREATE TABLE registrants (id INTEGER, name TEXT NOT NULL, sport TEXT NOT NULL, PRIMARY KEY(id));`

We then have to update the registrants html:

```
{% extends "layout.html" %}
{% block body %}
<h1> Registrants </h1>
<table>
```

```
    <thead>
        <tr>
            <th>Name</th>
            <th>Sport</th>
        </tr>
    </thead>
    <tbody>
    {% for row in rows %}
        <tr>
            <td> {{row.name}} </td>
            <td> {{row.sport}} </td>
        </tr>
    {% endfor %}
    </tbody>
</table>
{% endblock %}
```

Adding Email (getting rid of some SQL)

```
import os
from flask import Flask, render_template, request, redirect
from flasx_mail iport Mail, Message

app = Flask(__name__)
app.config["MAIL_DEFAULT_SENDER"] = os.getenv("MAIL_DEFAULT_SENDER")
app.config["MAIL_PASSWORD"] = os.getenv("MAIL_PASSWORD")
app.config["MAIL_PORT"] = 587
app.config["MAIL_SERVER"] = "smtp.gmail.com"
app.config["MAIL_USE_TLS"] = True
app.config["MAIL_USERNAME"] = os.getenv("MAIL_USERNAME")
mail = Mail(app)

REGISTRANTS = {}

SPORTS = [
    "Dodgeball",
    "Flag Football",
    "Soccer",
    "Volleyball"
    "Ultimate Frisbee"
]
@app.route("/")
def index():
    return render_template("index.html, sports=SPORTS")

@app.route("/register", methods=["POST"])
def register():
    email = request.form.get("email")
    if not email:
        return render_template("error.html", message="Missing email")
    sport = request.form.get("sport")
    if not sport:
        return render_teplate("error.html", message="Missing sport")
```

```
    if sport not in SPORTS:
        return render_template("error.html", message="Invalid sport")



    message = Message("You are registered", recipients=[email])
    mail.send(message)

    return render_template("success.html")
```

Update the HTML to reflect the changes:

```
{% extends "layout.html" %}

{% block body %}
<h1>Register</h1>
<form action="register" method="post">
     <input autocomplete="off" autofocus name="email" placeholder="Email" type="email">
     {% for sport in sports %}
<input name="sport" type="radio" value="{{ sport }}">
{{sport}}
     {% endfor %}
     <input type="submit" value="Register">
</form>
{%endblock}
```

Additionally have to add `Flask-Mail` to `requirements.txt` # Sessions First visit to website:

```
GET / HTTP/1.1
Host: gmail.com
```

Followed by:

```
HTTP/1.1 200 OK
Content-Type: text/html
Set-Cookie: session=value
```

The cookie might be a unique identifieer. The browsers send the cookie back.

```
GET / HTTP/1.1
Host: gmail.com
Cookie: session=value
```

Incognito mode throws away your cookies.

Login

to `requirements.txt` add

```
Flask
Flask-Session
```

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="utf-8">
        <meta name="viewport" content="initial-scale=1, width=device-width">
        <title>login</title>
    </head>
```

```
    <body>
        {% block body %}{% endblock %}
    </body>
</html>
```

then make a flask app with login:

```
from flask inport Falask, redirect, render_template, request, session
from flask_session import Session

app = Flask(__name__)
app.config["SESSION_PERMANENT"] = False
app.config["SESSION_TYPE"] = "fiflesystem:
Session(app)

@app.route("/")
def index():
    if not session.get("name"):
        return redirect("/login")
    return render_template("login.html")

@app.route("/login", methods=[GET, POST])
def login(:
    if request.method==["POST"]
        # Remember that user logged in
        session["name"] = request.form.get("name")
        return redirect("/")
    return render_template("login.html")

@app.route("/logout")
def logout:
    session["name"] = None
    return redirect("/")
```

session is global and unique to each user.

```
{% extends "layout.html" %}

{% block body %}

    <form action="/login" method="post">
        <input autocomplete="off" autofocus name="name" placeholder="Name" type="text">
        <input type=submit>
    <form>
{% endblock %}
```

Login failure, intex.html:

```
{% extends "layout.html" %}

{% block body %}
    {% if session.name %}
        You are logged in as {{ session.name }}. <a href="/logout">Log out</a>.
    {% else %}
        You are not logged in. <a href="/login">Log in</a>.
    {% endif %}
{% endblock %}
```

There are different kinds of cookies (different info, expiration times).

Store

Look st source code

Flask Short

```
from flask import Flask
from datetime import datetime
from pytz import timezone

app = Flask(__name__)

@app.route("/")
def time():
    now = datetime.now(timezone('America/New_York'))
    return "The current date and time in Cambridge is {}".format(now)
```

Another example:

```
@app.route("/")
def index():
    return "You are at the index page!"

@app.route("/sample")
def sample():
    return "You are at the sample page!"
```

They recommend that you do:

```
export FLASK_APP=application.py
export FLASK_DEBUG=1
flask run
```

Pass data via `GET`

Can do it via URL:

```
@app.route("/show/<number>")
def show(number):
    return "You passed in {}".format(number)
```

`POST` takes data via HTML forms.

Here is an example:

```
@app.route("/login", methods=['GET', 'POST'])
def login():
    if not request.form.get("username")
        return apology("must provide username")
```

Vary based on Request type

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

```
if request.method == POST:
# do a thing
else:
# do a different thing
```

- Things I should be aware of: `url_for()`, `redirect()`, `session()`, `render_teplate()`.
- Flask quickstart: `http://flask.pocoo.org/docs/0.12/quickstart`
- Jinja quickstart: `http://jinja.pocoo.org`

Ajax Short

Making things happen on the server There is a Javascript object called `XMLHttpRequest`. Make it like:

```
var xhttp = new XMLHttpRequest();
```

Then we have to make an `onreadystatechange` behavior. The steps that happen when we visit a page. The `readyState` property goes from `0` to `4`. `0` is not yet initialized, `4` is request finished. Want `readyState` to be `4`, `status` to be `200`.

`JS` function that does an Ajax request

```
function ajax_request(argument)
{
    var aj = new XMLHttpRequest();
    an.onreadystatechange = function() {
        if (aj.readyState == 4 && aj.status == 200)
        };
        aj.open("GET", /* url */, true);
        aj.send();
}
```

This will usually be written in `jquery`: `http://api.jquery.com/jquery.ajax` to learn how