16) Web servers, cloud and Jinja

Agenda

- Intro to web servers and the "cloud"
- Deployment on Heroku, Google App Engine and Azure App Service
- Jinja templating engine

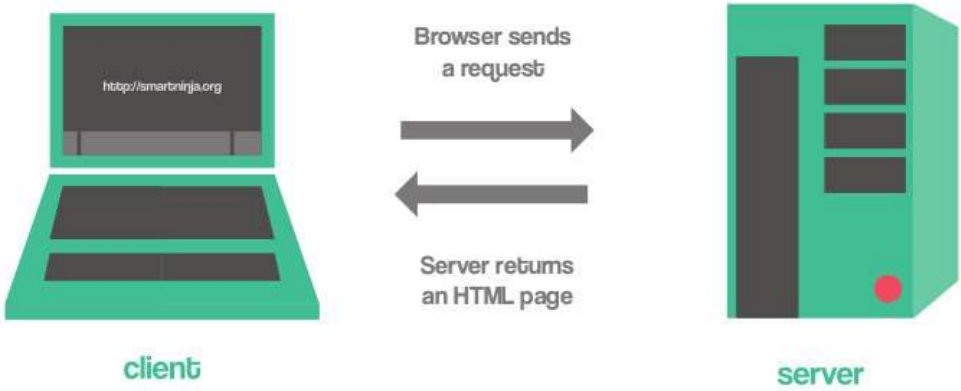
In the previous lesson you created your **first web app**. Your web has both **front-end** (HTML&CSS) and **back-end** (Python code).

But currently your web app runs **only on your computer**. When you run main.py, you can access your website in your browser via `http://127.0.0.1:5000/`. Other people cannot access your web app.

127.0.0.1 - What is this???

127.0.0.1 is an IP address of a local device. To put it simply: it means that your web app runs on your computer.

Remember this image from lesson 1:



It shows how your browser (the client) can access a website that's running on a server. It sends a request to a server and receives a response back.

Currently both the browser and the server for our web app are on **the same computer** - our own. Flask created a server on our computer that can be accessed by our browser.

We call this server a "local server" or a **localhost**. You can in fact change the link `http://127.0.0.1:5000/` into `http://localhost:5000/` and you would still be able to access your web app.

What is a server anyway?

Server is a computer where your web app is stored. If another computer would like to access the web app, it needs to send a request to the server (a request that says: please give me a copy of this website HTML).

Ideally, we would want our server to be accessible 24/7 (all the time). The best option for this is to have a server in the **cloud**.

What is the "cloud"?

The cloud is just a **fancy** name for **big hosting providers**, such as Amazon Web Services (AWS), Microsoft Azure, Google Cloud, IBM Cloud etc.

It basically means that these companies have **thousands of servers** on different locations **all across the world** and you can **rent** their servers to **host** your web app.

And because these companies are in a **pricing war**, you can get some pretty sweet deals. On **many** of them you can host your web app for **free**, if you don't have too much traffic to your website. And the free quota is very generous (meaning: you get a lot for free).

Platform-as-a-Service (PaaS)

In this lesson you'll learn how to **deploy and host** your web app on the three biggest cloud providers: AWS, MS Azure and Google Cloud.

More precisely, you'll be using a **Platform-as-a-Service** (PaaS) hosting on each of these clouds. PaaS hosting means that you **don't have to deal with configuring the servers**. Instead, you just push your code to PaaS hosting and everything is configured (and scaled) **automatically**. This saves you a **lot** of time.

The PaaS services that we'll use on each of these cloud providers are:

- Heroku (AWS)
- App Engine (Google Cloud)
- App Service (Azure)

Enough theory, let's go into action!

The instructor will show a deployment on **one** of these three services (you'll try the others at home):

- Heroku (instructions (https://github.com/smartninja/wd1-py3-exercises/tree/master/lesson-16/deployment-tutorials/heroku)) - the fastest and the easiest
- Google App Engine (instructions (https://github.com/smartninja/wd1-py3-exercises/tree/master/lesson-16/deployment-tutorials/google-app-engine))
- Azure App Services (instructions (https://github.com/smartninja/wd1-py3-exercises/tree/master/lesson-16/deployment-tutorials/azure-app-service))

All the instructions are on SmartNinja GitHub (https://github.com/smartninja/wd1-py3-exercises/tree/master/lesson-16/deployment-tutorials).

Deploy the web app that you created in the previous lesson (Homepage). Paste the links to your deployed websites on the SmartNinja forum.

Jinja

Jinja is a very useful tool that helps us organize HTML templates better and work with data inserted in these templates. It comes as a part of the Flask pip installation.

Let's learn how to use Jinja by doing an **example** web app.

Create a **new** Flask project. The project should have **two handlers**:

- `index()` (URL: `/`)
- `about()` (URL: `/about-me`)

And **three** HTML templates:

- `base.html`
- `index.html`
- `about.html`

The handlers should look like this:

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


@app.route("/about-me")
def about():
    return render_template("about.html")
```

Jinja extends

Jinja has a really nice feature called **"extends"**, which helps us **reduce** the amount of code we need to write in our HTML templates. It follows the **DRY** approach ("don't repeat yourself"). Let's see how it works!

Your `base.html` file should include the following code:

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="UTF-8">
        <title>SmartNinja with Jinja</title>

        <link rel="stylesheet" href="/static/css/style.css">
    </head>

    <body>
        <h1>Let's try out Jinja!</h1>
    </body>
</html>
```

Your `index.html` and `about.html` should be completely **empty**!

Now run your program. What do you see?

Nothing. Because your `index.html` file is empty.

Now add this line of code into `index.html` :

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

Reload your web app. Now you can see the code from `base.html` in your browser.

What happened?

The `index.html` file **inherited** (or extended) the code from **base.html**, the same way as Python classes can inherit code from another class.

But the true power of the `extends` feature comes with another Jinja feature: **blocks**.

Jinja blocks

Add the `block` code into **base.html**:

```
<body>
    <h1>Let's try out Jinja!</h1>

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

Now add blocks also in the other two HTML files, but each block will contain different code in it.

index.html:

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

{% block content %}
    <p>Hello, index.html here!</p>
{% endblock content %}
```

about.html:

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

{% block content %}
    <p>More about me :)</p>
{% endblock content %}
```

Now go to both URLs, `/` and `/about-me` and observe what happens.

As you can see, both of these pages inherit code from `base.html` and add some of their own in (the `block` part).

You can also add this code into `base.html` to easily switch between HTML templates:

```
<p>
    <a href="/">Home</a>,
    <a href="/about-me">About me</a>
</p>
```

Pretty cool, right?

We can have as many types of blocks in our HTML as we want. For example, we can create a block for the `<title>` tag in the `<head>` section:

```
{% block title %}{% endblock title %}
```

So that each HTML template can have a different head title.

But Jinja can do **much more** than just organize HTML documents. It can also **work with variables** inside HTML documents.

Jinja variables

Jinja enables us to send variables from a handler to an HTML template:

```
import datetime

@app.route("/")
def index():
    some_text = "Message from the handler."
    current_year = datetime.datetime.now().year

    return render_template("index.html", some_text=some_text, current_year=current_year)
```

As you can see, we are sending two variables to the `index.html` template.

Let's show these variables in our HTML document:

```
{% block content %}
    <p>Hello, index.html here!</p>

    <p>{{some_text}}</p>
    <p>Current year: {{current_year}}</p>
{% endblock content %}
```

As you can see, Jinja shows a variable inside the HTML template using double curly barckets:

```
{{some_text}}

{{current_year}}
```

Jinja for loop

You can even use a for loop in Jinja.

Let's send a list from our handler to our HTML document:

```
@app.route("/")
def index():
    some_text = "Message from the handler."
    current_year = datetime.datetime.now().year

    cities = ["Boston", "Vienna", "Paris", "Berlin"]

    return render_template("index.html", some_text=some_text, current_year=current_year, cities
```

And now use a for loop to print cities one-by-one in `index.html` :

```
{% for city in cities %}
    <p>{{city}}</p>
{% endfor %}
```

Jinja if statement

```
{% if cities %}
    <p>The cities variable exists.</p>
{% else %}
    <p>There are no cities :(</p>
{% endif %}
```

> *Notice that we used `{% %}` instead of `{{ }}` for the for loop and the if statement. Double curly brackets `{{ }}` are **only** used for variables.*

The complete example code

See the code for this Jinja example here (https://github.com/smartninja/wd1-py3-exercises/tree/master/lesson-16/working-with-jinja).

Q&A

Any question regarding servers, deployment, Jinja or similar topics?

If there's enough time after Q&A, students can start working on their homework.

Homework 16.1: Try deployments to all three clouds

> *If there's enough time, you can start doing this exercise already at the session (after Q&A).*

So far you tried the deployment to only one of the three PaaS hosting providers.

Now try to deploy your web app also on the other two. If you'll have any difficulties, ask a question on the SmartNinja forum.

> ***Which PaaS service to use for real projects?***
>
> *Each of them has pros and cons:*
>
> - *Heroku: easy deployment; nice dashboard user interface (UI)*
> - *Google App Engine: the best free quota (you get more for free than on Heroku or Azure); deployment is a bit more complex (but you can automate it via CI); the best logging system*
> - *Azure: easy deployment; dashboard UI is not as nice as Heroku's or Google's; free trial expires very fast; log monitoring is terrible*

Homework 16.2: Use Jinja in your Homepage web app

Use Jinja to improve the HTML templates in your Homepage web app.