17) HTTP requests & Cookies

Agenda

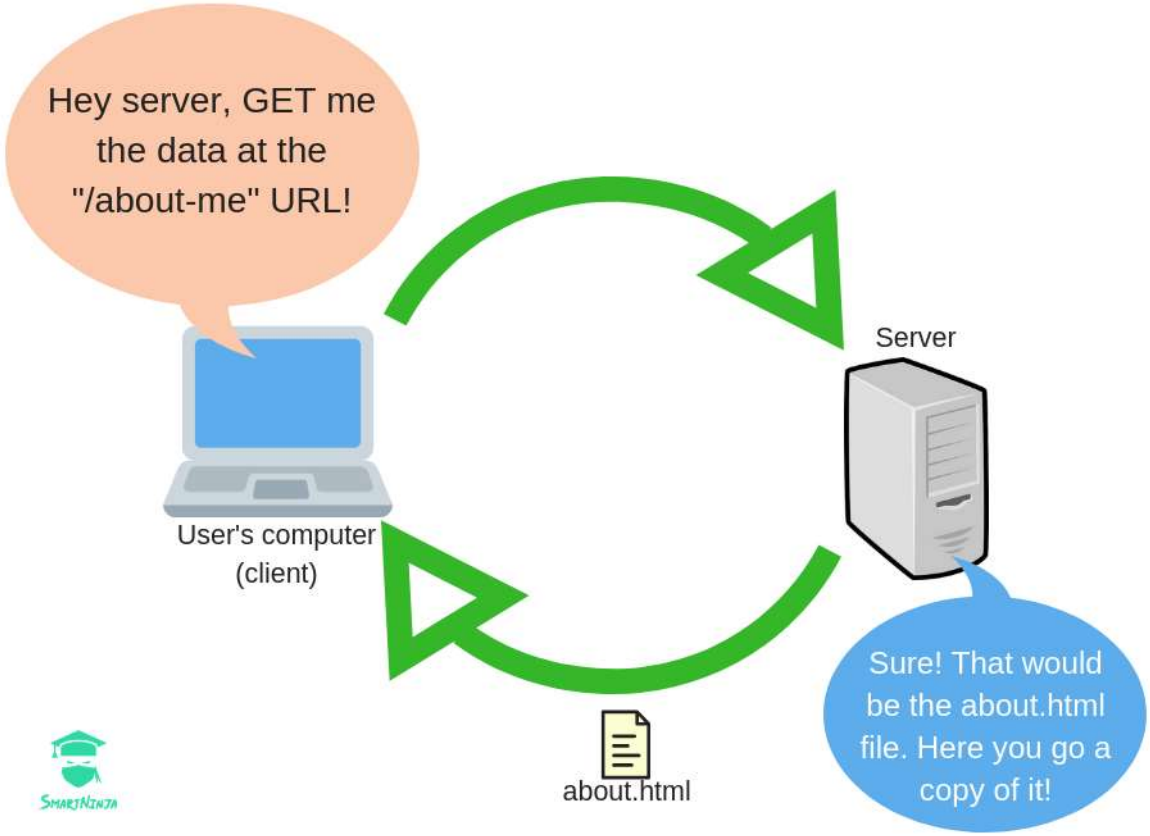
- GET request
- POST request
- GET & POST in Flask
- Cookies

A communication between the user's computer and a server is happening via "requests". The user's computer **requests** data from a server or it requests the server to accept data.

Most of requests are either **GET** or **POST**. Let's learn more about them.

GET request

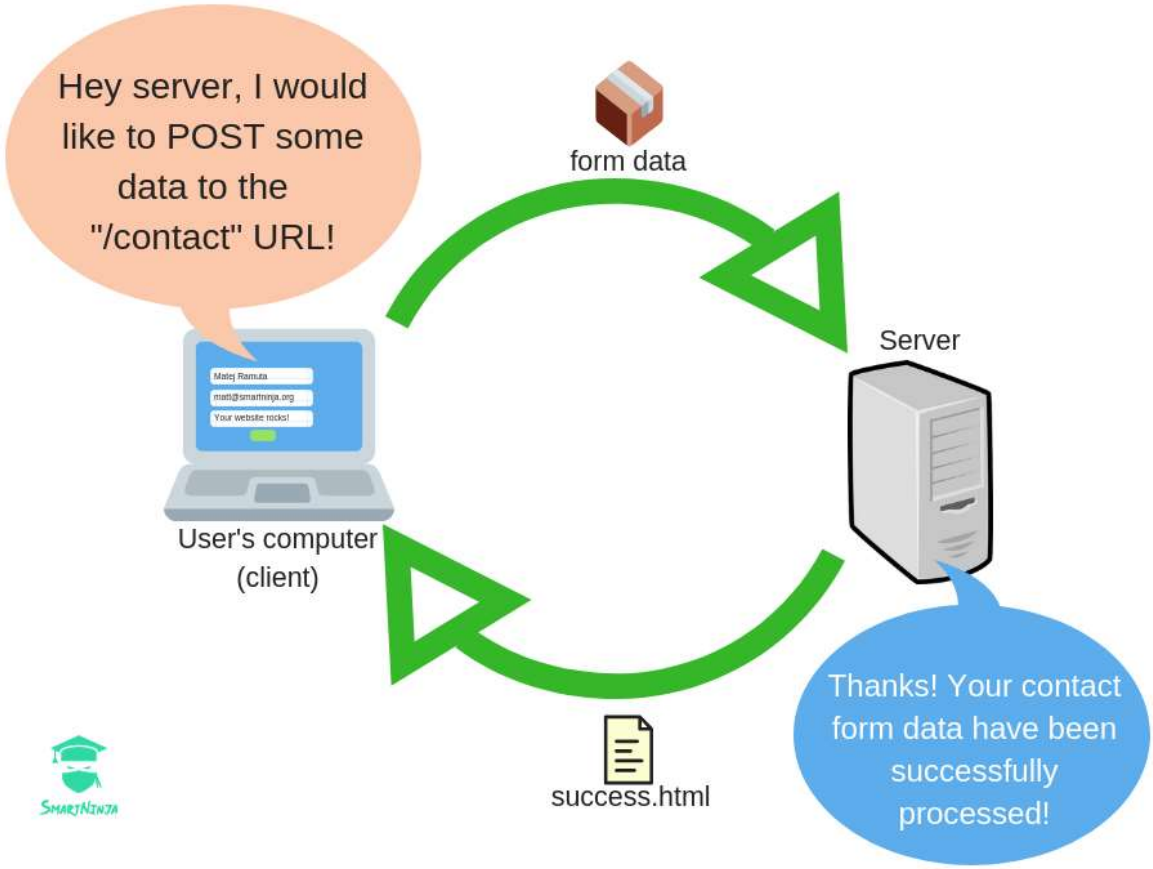
A GET request means that the browser wants to **get** some data from the server. This data is usually an HTML document.



POST request

So if the GET request is about **getting** data from the server, you can probably guess that the POST request is about sending data to server.

Imagine you have a contact form where user fills out their name, email address and a message they want to send to you. When the user clicks "Submit", a POST request is sent to the server.



The POST request carries the data from the form. When the server **receives** the data, it **process** it and sends back a **response**.

The server's **response** can either say:

- Your data was **successfully** accepted and processed.
- There was an **error**.

GET request in a Flask app

So far all our handlers were only able to process GET requests.

A client (browser) could only GET data (HTML template) from the server. It couldn't POST any data to the server.

Open the Jinja example from the previous lesson, to see how our typical Flask handlers looks like:

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

By default this handler accepts only GET requests. We can explicitly specify this in our `app.route()`:

```
@app.route("/about-me", methods=["GET"])
def about():
    return render_template("about.html")
```

Both of these two examples do exactly the same. Receive a GET request.

But how can we make a POST request?

POST request in Flask

Let's add a contact form into our `about.html` template:

```
<form method="post" action="/contact">
    <input type="text" name="contact-name" placeholder="Enter your name">
    <br>

    <input type="email" name="contact-email" placeholder="Enter email address">
    <br>

    <textarea name="contact-message" placeholder="Enter your message"></textarea>
    <br>

    <button type="submit">Submit</button>
</form>
```

And let's create then another HTML template that will tell the user that the message was successfully received. Let's call this template `success.html` :

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

{% block title %}Success{% endblock title %}

{% block content %}
<h3>Success!</h3>

<p>Your message was successfully received. Thank you!</p>

{% endblock content %}
```

And the final step is to create a new handler that will accept POST requests on the `/contact` URL.

First add the `request` to our imports from flask:

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

Next create the `contact()` handler:

```
@app.route("/contact", methods=["POST"])
def contact():
    contact_name = request.form.get("contact-name")
    contact_email = request.form.get("contact-email")
    contact_message = request.form.get("contact-message")

    print(contact_name)
    print(contact_email)
    print(contact_message)

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

As you can see, the `contact()` handler can receive POST requests to the `/contact` URL. You can see the received data printed in the Terminal window.

> *See the complete example here (https://github.com/smartninja/wd1-py3-exercises/tree/master/lesson-17/flask-post-request-1).*

Alternative

We've created a new handler (`contact()`) to receive a POST request.

But we could have used the `about()` handler to cover both the GET and the POST request.

Let's see how this would look like:

```
@app.route("/about-me", methods=["GET", "POST"])
def about():
    if request.method == "GET":
        return render_template("about.html")
    elif request.method == "POST":
        contact_name = request.form.get("contact-name")
        contact_email = request.form.get("contact-email")
        contact_message = request.form.get("contact-message")

        print(contact_name)
        print(contact_email)
        print(contact_message)

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

As you can see, you would need an **if statement** to differentiate between the GET and the POST request that your handler might be receiving.

Also, you would need to change the `action` attribute in your `about.html` template:

```
<form method="post" action="/about-me">
```

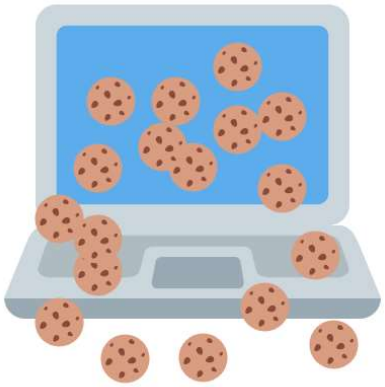
Previously the action attribute pointed to the `/contact` URL, but if you use the `about()` handler to handle your POST request, you need to change the `action` attribute to `/about-me`.

Both of the mentioned approaches (GET & POST in the same handler, or in separate handlers) produce **the same result**. It's up to you which one to choose in a given situation.

> *See the complete alternative example here (https://github.com/smartninja/wd1-py3-exercises/tree/master/lesson-17/flask-post-request-2).*

Cookies

The last thing we'll learn about today are cookies.



Cookies are a form of **data storage** inside a browser. A single cookie can store up to 4KB of data, which equals to a **string** with max. 1024-4096 characters.

So it's not a big data storage, but it's enough for certain usecases.

Storing user's name into a cookie

What we'll use a cookie for is to store a user's name in it.

Whenever a user will send us a message through a contact form, we will store his/her name into a cookie. And the next time s/he comes back to the About page, the message saying `Hello {{name}}` will show up.

This will need some adjustments to our handlers. Let's first focus on our POST method:

```
print(contact_name)
print(contact_email)
print(contact_message)

response = make_response(render_template("success.html"))
response.set_cookie("user_name", contact_name)

return response
```

Just below the print statements we've entered the code, which prepares the server's response. This response will hold the cookie that the server will send to the user's browser.

> *Make sure you've added `make_response` to the import statement at the top of `main.py` :*
>
> ```
> from flask import Flask, render_template, request, make_response
> ```

Now let's change the GET part of our handler:

```
if request.method == "GET":
    user_name = request.cookies.get("user_name")
    return render_template("about.html", name=user_name)
```

Here we take a look in the user's cookies and try to find the one with the user's name in it (the one that we set in the POST request).

We store the cookie into a variable and send it into the `about.html` template.

So let's add this piece of code into the `about.html` template, just above the contact form:

```
{% if name %}
<p>Hello {{name}}!</p>
{% endif %}
```

This means that in case we found the cookie with the user's name data in it, say `Hello {{name}}!` to the user.

Let's try it out! Cool, right? :)

> *See the complete example here (https://github.com/smartninja/wd1-py3-exercises/tree/master/lesson-17/cookie-example).*

Deleting a cookie

For deleting a cookie you have two options.

You can set it's value to an empty string:

```
response.set_cookie("user_name", "")
```

But be aware that this won't actually delete the cookie. It will just make it hold an empty value `""` (which, to be honest, is good enough).

To truly delete a cookie, you must set it's expiration date to `0`:

```
response.set_cookie("user_name", expires=0)
```

That's it! Your cookie has been successfully deleted.

Q&A

Any question?

Homework 17.1: Our favorite game

> *If there's enough time, students can start working on this exercise already at the session. If some students want to work in pairs, they are free to do it.*

Let's put our favorite game online! You know which one we're talking about... It's the **Guess the secret number** game! :)

Create a new Flask web app that will have a GET and a POST request handler. They can be both in one handler, or each in a separate handler (doesn't matter).

In the GET handler you'll do the following things:

- Check if there's a cookie with the name "secret_number".
- If there isn't any such cookie, create a new cookie with this name.
- The new cookie should store a random number between 1 and 30.

The POST handler should do the following:

- Get the `secret_number` cookie.
- Get the user's `guess` from the HTML form.

- Compare the `guess` and the `secret_number`
- If the guess is the same as the secret number, set a new random number in the cookie and return the user a `result.html` page with a congrats message.
- If not, return the user the `result.html` page with a message saying whether the guess was too big or too small.

> *Solution (https://github.com/smartninja/wd1-py3-exercises/tree/master/lesson-17/guess-secret-number)*