Thurgau

Tag 11
Webserver

05. Mai 2023

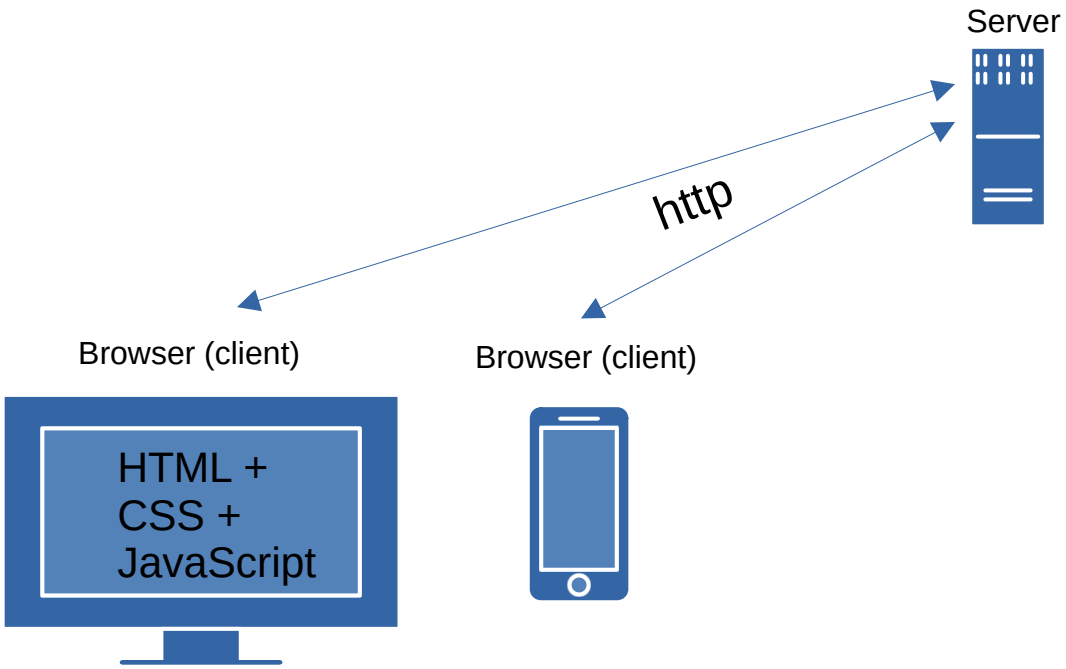
Ablauf

- Prüfung Besprechen
- Rückblick zehnter Tag
- Überblick Web
- Einstieg Flask (Python Webserver)

- Projektarbeit

Rückblick GUI

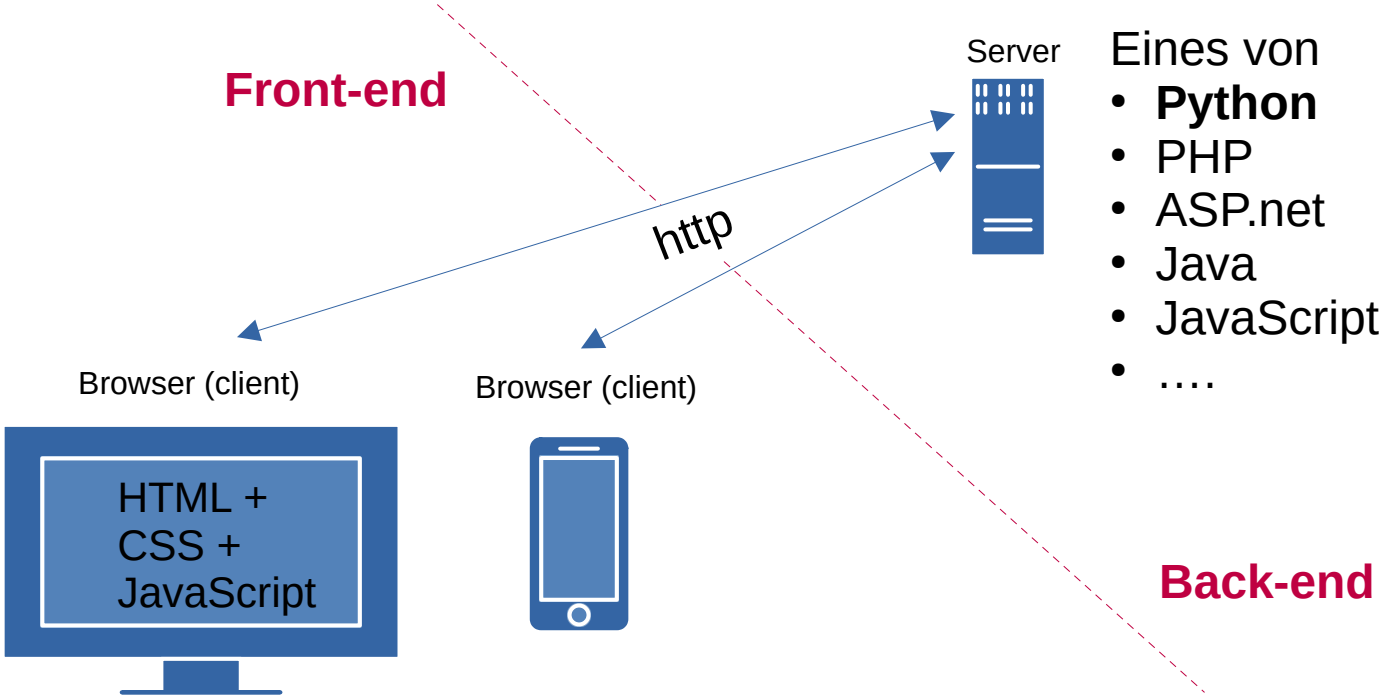
Überblick Web



Eines von

- **Python**
- PHP
- ASP.net
- Java
- JavaScript
- ….

Überblick Web



Eines von

- **Python**
- PHP
- ASP.net
- Java
- JavaScript
- ….

Front-end (im Browser)

- **HTML Hypertext Markup Language**
  - *‹Textauszeichnungssprache›*
  - Text/Inhalt einer Seite
- **CSS Cascading Style Sheets**
  - *‹Wasserfallartige Stilbereiche›*
  - Formatierung einer Seite
- **JavaScript**
  - Programmiersprache für Dynamik im Frontend

HTTP

- **Hypertext Transport Protokoll**
  - Zustandsloses Protokoll zur Übertragung von (Text-)dateien
    - HTML, CSS und JavaScript sind Textdateien
  - GET und POST am wichtigsten
  - Client fragt Server (request), Server antwortet darauf (response)

Back-end (Server)

1) Wartet auf Request
2) Bearbeitet Request
3) Sendet Response

- Asynchron → mehrere Clients können gleichzeitig bedient werden
  … gut ist HTTP ohne Status

- Status (z. B. An-& Abmelden, Warenkorb, etc.) werden obendrauf gemacht

HTML

```
<html>
    <head>
        <title>Hello World with HTML and Name</title>
    </head>

    <body>
        <h1>Hallo World</h1>
        Guten Tag <b>Harry</b>
    </body>
</html>
```

- Markup (in spitzigen Kammern) und Text kombiniert
- Jedes gültige Element (Markup) hat eine Bedeutung und wird von allen Browsern verstanden.

CSS

```css
h1 {
    background-color: navy;
    color: white;
}

body {
    background-color: deepskyblue
}
```

- Darstellungsregeln für Elemente
- Regeln sind für Typen (h1), Klassen (.class) oder spezifisch pro Element (#id) möglich.
- Kaskadierend/Wasserfallartig → mehrere Regeln können zutreffen, das Resultat ist die Vereinigung aller Anweisungen

Lets do it!

- Übung Frontend

  Gute Ressource: SelfHTML HTML CSS

Flask Python webserver

Leichtgewichter, ‹einfacher› Webserver (web application framework) in und für Python.

Nur sehr wenige Features per Default:

- **Werkzeug** (WSGI Bibliothek, Kern von Flask)
- **Jinja** (Templating Bibliothek)

- Tutorial: https://www.tutorialspoint.com/flask/flask_quick_guide.htm

Hello World

Installation: *pip install flask*

```
from flask import Flask

app = Flask(__name__)

@app.route('/')
def hello_world():
    return "Hello, World!"

app.run()
```

Öffne die angegebene Adresse im Browser
(vermutlich http://127.0.0.1:5000 oder localhost:5000)

Routen

Routen werden verwendet, um URLs Aufrufen (Python Funktionen) zuzuordnen.

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

@app.route('/demo2')
@app.route('/demo2/')
@app.route('/demo2/<name>')
@app.route('/demo2/<name>/')
def hello_world_css(name: str=''):
    return …
```

Zweites Beispiel mit Variable (hier vom Typ String)

Hands on

- Zeit vom Server zurückgeben
- Bild einbinden und ausliefern
- Wirkt das CSS noch wie erwartet?
- Was muss bei welchen Änderungen neu gestartet werden?
- Weshalb hat demo2 vier Routings? Welche Route hat welche Auswirkung?
- .. das Aussehen und das Layout darf natürlich auch verbessert werden.

Vorschau: Templating

<html><head><title>Hello World with HTML and CSS</title><link rel="stylesheet" href="/static/stylesheet.css" /></head><body><h1>Hallo World</h1>⭐Seasons greetings {name}🌲</body></html>

- HTML von */demo2* ist jetzt noch nicht eine ganze Seite, aber schon maximal unübersichtlich.
- Gibt es das auch in hübsch und lesbar?

Ja, fast: jinja templates

template.html.jinja:

```
<html>
  <head>
    <title>{{ variable|escape }}</title>
  </head>
  <body>
  {% for item in item_list %}
    {{ item }}{% if not loop.last %},{% endif %}
  {% endfor %}
  </body>
</html>
```

*.py:

```
from flask import render_template
…
def handle_request():
        some_argument = 'irgendwas'
        return render_template('template.html.jinja', some_argument = some_argument)
```