Templating

Allgemeines

Da HTML lediglich eine Auszeichnungssprache und keine Programmiersprache ist, können in HTML keine Funktionalitäten umgesetzt werden. Sobald Websites aber dynamischen und nicht nur statischen Inhalt ausgeben sollen, reicht HTML allein dafür nicht mehr aus.

Daher muss der dynamische Inhalt zuerst auf dem Server erzeugt werden, anschließend in eine passende Form gebracht werden (z.B. in HTML) und dieses dynamisch generierte Dokument schließlich an den Client gesendet werden.

Jedoch leidet sowohl die Übersichtlichkeit und Lesbarkeit als auch die Wiederverwendbarkeit stark darunter, wenn der HTML-Code wie folgendermaßen ausgegeben wird:

```
return "<html><head><title>Testdokument</title></head><body><h1>Eine Überschrift</h1>
<div><p>Ein Textabschnitt</p></div></body></html>"
```

Daher werden sogenannte *"Templates"* genutzt, um dynamisch generierte Seiten einfach zu erzeugen und wiederverwenden zu können. Hierdurch wird eine Trennung zwischen den statischen Inhalten und den dynamischen Inhalten erreicht, die dennoch auf einfache Art und Weise verknüpft werden können.

Es gibt unzählige Templating-Systeme /-Engines, nicht nur im Bereich der Webanwendungen. Flask setzt standardmäßig auf die Jinja2 Template-Engine, die wie Flask selbst ebenfalls von Armin Ronacher entwickelt wurde und u.a. von Instagram, Mozilla u.v.m. eingesetzt wird.

Der Nachteil von Template-Engines liegt darin, dass häufig noch eine weitere Syntax erlernt werden muss, die jedoch meist sehr einfach gehalten ist.

Jinja2 Syntax

Die Syntax von *Jinja2* ist sehr einfach gehalten: Steueranweisung wie *if*, *for*, *while*, … werden in `{% ... %}`-Blöcken dargestellt. Die Ausgabe von Variablenwerten erfolgt dagegen durch `{{ ... }}` Blöcke, sie stellen sozusagen Platzhalter für die eigentlichen Werte dar.

Diese Blöcke können in herkömmlichen HTML-Code eingebettet werden, sie werden dann von der Templating-Engine ausgewertet und in regulären HTML- Code umgewandelt.

Da HTML-Code wie beschrieben unabhängig von Einrückungen ausgewertet wird, müssen die Anweisungsblöcke beim Templating durch entsprechende Befehle abgeschlossen werden, z.B. `endif`, `endfor`, …

```
{% if user.active %}
        <p>The user is active</p>
{% elif %}
        <p>The user in inactive</p>
{% endif %}
```

Beispiel eines Templates:

```
<!DOCTYPE html>
<html>
        <head>
```

```
                <title>
                        {{title}}
                </title>
        </head>
        <body>
                <h1>{{title}}</h1>
                {% for i in range(10) %}
                        <p>{{i}}. Textabschnitt</p>
                {% endfor %}
        </body>
</html>
```

Die Templates können nun vom Server ausgegeben werden, indem die `render_template()`-Methode importiert wird. Mit dieser kann das Template ausgewertet und in HTML-Code umgewandelt werden:

```
@app.route('/')
def index():
        return render_template('template.html', title="Titel der Seite")
```

Beachte, dass der `render_template()`-Methode alle Parameter übergeben werden, die innerhalb des Templates verwendet werden, ansonsten wird ein Fehler erzeugt.

Standardmäßig sucht Flask im Ordner `templates/` nach den Templates.

Aufgabe

1. Erstelle ein Template, das bei Aufruf der Route `/random` jeweils 10 zufällig generierte Zufallszahlen zwischen 1 und 10 ausgibt.
2. Erweitere das Beispiel aus dem Kapitel *Requests* so, dass nach dem Absenden des Formulars die eingegebenen Daten unter dem Formular ausgegeben werden.