Requests

Get Requests

Die Routes, die wir bisher mit Flask erzeugt haben, werden lediglich bei `GET`-Requests ausgeführt, da ohne Angabe einer spezifischen HTTP-Methode standardmäßig die GET-Methode angenommen wird. Daher sind diese Routen auch einfach durch die URL über den Browser aufrufbar.

In manchen Fällen ist es jedoch nötig, näher zu beschreiben, welche Ressource genau angefragt wird. Beispielsweise sind unter der Route `/users` alle Nutzerinformationen hinterlegt. Nun sollen jedoch nicht alle Nutzer abgerufen werden, sondern lediglich die Daten zu einem spezifischen Nutzer. Dies kann erfolgen, indem zusätzlich ein Parameter mit der ID des angefragten Users übergeben wird. Hierfür gibt es bei GET-Methoden zwei unterschiedliche Möglichkeiten:

- Direkt über die URL: Die ID wird als Bestandteil der URL bzw. der Route übermittelt, z.B. `/users/123`. Hierdurch wird aus der URL sofort ersichtlich, dass ein bestimmter User mit der ID 123 angefragt wird. Hier stellt sich jedoch die Frage, wie diese "dynamische" URL in Flask abgebildet werden kann. Für jeden verfügbaren Nutzer eine eigene Route mit `@app.route("/users/123")`, `@app.route("/users/124")`, `@app.route("/users/125")`, … anzulegen wäre deutlich zu viel Aufwand. Daher bietet Flask die Möglichkeit für variable Routen. So kann eine Route erzeugt werden mit:

  ```
  @app.route("/users/<user_id>")
  def get_user(user_id):
          # ...
  ```

  Hierdurch wird in der Route ein Platzhalter (`<user_id>`) definiert, gekennzeichnet durch die spitzen Klammern, der dann der jeweiligen Funktion als Parameter übergeben wird.

- Durch einen Query-String: Die oben gezeigte Methode, bei der Parameter direkt als Teil der URL definiert werden, bietet sich lediglich bei URLs an, bei denen durch den Parameter auf eine bestimmte, eindeutige Ressource verwiesen wird. `/users/123` verweist also auf eine eindeutige Ressource, da hier durch die ID 123 auf einen bestimmten Nutzer zugegriffen wird. Werden dagegen Parameter übergeben, die auf keine Ressource verweisen, sollten Query-Strings verwendet werden. Wird beispielsweise ein Filter angewendet (z.B. Alle Nutzer, die 25 Jahre alt sind), sieht die URL etwa folgendermaßen aus: `/users?age=25`. Der Teil `?age=25` stellt dabei den Query-String dar. Query-Strings werden immer mit einem `?` an die normale URL angehängt, dann folgen Key-Value-Paare (also `age=25`). Es können auch mehrere Queries in einem Query-String enthalten sein, die durch ein `&` konkateniert werden, z.B. `/users?age=25&city=stuttgart`. Hier wird also nach Nutzern gefiltert, die sowohl 25 Jahre alt sind als auch in Stuttgart wohnen. Merke: Ein Query-String beginnt immer mit einem `?`, das direkt an die normale URL angehängt wird. Anschließend folgen ein oder mehrere Key-Value-Paare, die mit einem `=` verbunden werden. Mehrere Queries können mit einem `&` verbunden werden. In Flask können diese Parameter mit dem `args`-Keyword des `request`-Objekts ausgelesen werden:

  ```
  from flask import request

  @app.route("/users")
  def  get_user():
          age = request.args.get('age')
          city = request.args.get('city')
          # ...
  ```

  Im `request`-Objekt sind alle Informationen zum jeweiligen Request enthalten. Dieses muss jedoch zunächst importiert werden, bevor es genutzt werden kann.

Post Requests

Nun wollen wir aber nicht nur Informationen anfordern, sondern auch Daten an den Server schicken. Dies kann wie zuvor beschrieben über die POST-Methode erreicht werden. Hierfür nutzen wir das Formular, das in der Aufgabe im Kapitel HTML erstellt wurde.

Dieses Formular ist über die Route `/form` erreichbar. Wenn diese Route durch die GET-Methode aufgerufen wird soll wie bisher das Formular erscheinen. Wenn jedoch durch das Absenden des Formulars ein POST-Request an `/post` gesendet wird, sollen die übergebenen Daten vom Webserver ausgelesen werden und durch einen String ausgegeben werden.

Dem Dekorator der jeweiligen Funktion muss dafür zunächst mitgeteilt werden, dass es sich hierbei nicht um einen GET-Request handelt, sondern um einen POST-Request. Die HTTP-Methoden müssen als Liste angegeben werden, da eine Route sowohl GET- als auch POST-Requests (sowie die weiteren HTTP-Methoden) verarbeiten kann.

```
@app.route("/post", methods=['POST'])
``\

Nun können die Daten, die an den Server gesendet wurden, innerhalb der entsprechenden
Funktion ausgelesen werden. Die einzelnen Werte werden bei Flask im `request`-Objekt
gespeichert.

Anschließend kann innerhalb der Funktion auf das `request`-Objekt zugegriffen werden.
Alle Werte des Formulars befinden sich wiederum im `form`-Objekt innerhalb des
`request`-Objekts. Darin sind die unterschiedlichen Werte des Formulars enthalten. Um
die Werte unterscheiden und richtig zuordnen zu können sind sie entsprechend des
`name`-Attributs, das im HTML-Code zugewiesen wurde, zugreifbar.

Wenn im HTML-Code also folgendes Formular enthalten ist:

```html
<form action="/post" method="POST">
        <input type="text" name="first_name">
        <input type="submit">
</form>
```

ist der Inhalt des *"first_name"*-Inputfelds auf dem Server in der Variablen `request.form.first_name` enthalten.

Aufgabe

1. Erstelle eine Route, die aus `/users/` + einem beliebigen Namen besteht. Bei Aufruf dieser Route soll der User mit dem entsprechenden Namen gegrüßt werden. Also z.B. bei `/users/Jonas` wird "Hallo, Jonas" ausgegeben.
2. Erstelle eine weitere Route `/users`. Diese kann beliebige Parameter empfangen. Lese alle übergebenen Parameter aus und gebe diese in Textform an den Client zurück.
3. Erweitere das Beispiel aus dem Kapitel *HTML* so, dass unter der Route *"/form"* über einen GET-Request ausgegeben wird, die Inhalte des Formulars beim Absenden an die *"/post"*-Route gesendet werden und dort wieder als Text an den Client ausgegeben werden.
```