Flask Einführung

Client - Server - Prinzip

Wir wollen uns an die Standardprozedur des Internets halten:

- Auf dem Raspberry PI läuft ein Webserver
- Der Benutzer schickt mit einem Webbrowser (Der erst mal auch auf dem RasPi läuft, später eventuell auch auf anderen Geräten) Anfragen an diesen Server
- Der Server generiert die Webseite. Mit einem Python-Web-Framework kann hierbei beliebiger Pythoncode ausgeführt werden, somit können Sensoren, Schalter etc. genutzt werden
- Der Server schickt Webseite als Antwort an den Clienten, der die Webseite entsprechend darstellt und dem Nutzer damit Interaktion ermöglicht

Vorteilhaft ist dabei: Das GUI ist sehr einfach mit sehr wenig Code generierbar, besonders mit der Benutzung von Front-End-Frameworks lassen sich moderne, gutaussehende Interfaces mit wenig Aufwand generieren. Die "Anwendung" ist dann portabel auf allen Systemen nutzbar, die einen Webbrowser haben. Die Trennung in Client und Server führt dazu, dass das System sehr einfach von mehreren Clients benutzbar ist. Man könnte bspw in einer Wohnung mehrere Bedienpanels haben)

Grundkonzept

Das GUI soll ähnlich der Kacheln sein, die aus Windows 10 bekannt sind. Diese sind dann antippbar und führen dann eine Aktion aus, bspw. "Licht einschalten". Die erste Webanfrage eines Clients führt also zu einer generellen Übersicht, die Antwort ist eine Webseite mit mehreren Kacheln, darunter eine mit der Aufschrift "Licht einschalten" Der Nutzer tippt eine dieser Kacheln an. Der Webbrowser schickt dann eine Anfrage an den Server mit der Information, dass die Kachel angetippt wurde. Der Webserver führt die entsprechende Aktion (also hier: "Licht einschalten") aus und gibt wieder die Seite aus, diesmal steht auf der Kachel "Licht ausschalten" und ein Antippen bewirkt schaltet das Licht wieder aus.

Flask Allgemein

Flask ist ein (Micro-)Webframework für Python. Es "kümmert" sich um die grundlegende Serverfunktionalität (also: öffnet einen Socket, wartet auf TCP Verbindungen, nimmt HTTP-Requests an, analysiert die Anfrage, ruft dann von uns definierte Funktionen auf, die das Antwort-HTML generieren und schickt eine

entsprechende HTTP-Antwort) Es ist dabei sehr klein und leichtgewichtig, dafür verzicht es auf bestimmte Funktionen, die aber nachgerüstet werden können.

Alles, was wir ergänzen müssen, sind Funktionen, die für eine bestimmte URL HTML-Code zurückgeben. Wir werden dabei unterstützt durch eine "Template Engine". Diese ermöglicht es uns, ein Gerüst für eine Webseite zu bauen, in das wir dann mit Python bestimmte Werte einsetzen. Die Idee ist also, dass dieses Gerüst die grundlegende Kacheldarstellung ermöglicht und wir lediglich Informationen wie Beschriftung und Aktion beim Anklicken an die Template-Engine übergeben müssen, die daraus eine entsprechende Webseite zusammenstellt. Für die grundlegende Funktionalität nutzen wir das Frontend-Framework Bootstrap, da dies bereits viele Funktionalitäten enthält, die wir dann nutzen können.

Weiterhin sind in vielen Webframeworks Datenbankanbindungen integriert. Flask ist sehr leichtgewichtig und verzichtet daher von Hause aus auf diese Funktionalität, sie lässt sich aber bei Bedarf leicht nachrüsten und nutzen.

Minimales funkionierendes Beispiel für Flask-Server

Flask Code für ein einfaches "Hello World":

```
from flask import Flask     # Flask-Modul importieren
app = Flask(__name__)       # eine Flask-Application erstellen (Controller-Einheit)

@app.route('/')             # Funktionsdekorator, teilt Flask den zugehörigen Pfad mit
def hello_world():          # Funktionsname frei wählbar
    return 'Hello, World!'  # HTML-Code, der dann gerendert wird (Hier eigentlich kein
                            # valides HTML
```

Die Route kann auch Variablen enthalten, um so etwas wie: www.domain.com/forum/page/27/ zu ermöglichen. Eine Funktion für alle Forenseiten, kriegt die Seitenzahl übergeben und kann die entsprechende Seite generieren

Erweiterungen mit komplexerem Seitenaufbau durch Nutzung von Templates

Flask Code:

```
from flask import Flask
app = Flask(__name__)

@app.route('/')
def hello_world():
    return render_template("template.html", variable1="Hallo", variable2="Wert")
```

Mögliches Template:

```
<!doctype html>

<html>

<head>
  <title>Meine Webseite</title>
</head>

<body>
  <h1>Überschrift</h1>
  Variable 1 hatte den Wert {{ variable1 }}. Variable 2 hatte den Wert {{ variable2 }}.
</body>

</html>
```

Die Template-Engine generiert daraus eine entsprechende HTML-Seite. Es sind Kontrollstrukturen wie Schleifen und Bedingungen möglich. Außerdem können auch komplexe Objekte übergeben und auf diesen beispielsweise Methoden aufgerufen werden.

Statische Dateien, die im HTML referenziert werden

Die Einbettung von CSS- oder JavaScript-Dateien in HTML Code führt dazu, dass der Browser die entsprechendes Daten nachlädt und interpretiert.

- JS: Code, der vom Webbrowser aufgeführt wird. Hierbei kann die Webseite beim Client modifiziert werden, zum Beispiel wenn etwas angeklickt oder das Fenster vergrößert / verkleinert wird. Theoretisch sind dabei auch kompliziertere Berechnungen möglich. Mögliche Einbettung am Ende des HTML-Body:

  ```
  <body>
    ...
    <script src = "common.js>
  </body>
  ```

- CSS: Style Sheets: Geben an, wie bestimmte Elemente dargestellt werden sollen (Farbe, Schrift, Position). Mögliche Einbettung im `head` Teil der Seite:

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

Vorlage für den Embedded Smart Home 2017 Kurs

Wir haben eine Vorlage für einen Server erstellt und im GitHub hochgeladen. Mit folgendem Befehl kann das Git-Repository geklont werden:

```
git clone https://github.com/openHPI/Embedded-Smart-Home-2017.git
```

Das Hauptprogramm liegt hier in der Datei `smarthome.py`, die auch schon eine beispielhafte Implementierung enthält. Wir müssen lediglich Objekte der Klasse "SimpleTile" erstellen, deren Text und Farbe setzen und sie einem Objekt der Klasse "TileManager" hinzufügen. Der Manager kümmert sich um Anordnung der Kacheln, das Template um die Darstellung. Dafür muss im render_template Aufruf das TileManager Objekt übergeben werden. Außerdem muss ein Kontext-Objekt übergeben werden. Dies ist notwendig für Navigation am oberen Rand der Seite.

Wie geht es weiter?

Sie müssen nun die Vorlage nach Ihren Wünschen anpassen. Dafür müssen eventuell neue Unterseiten hinzugefügt werden. Denkbar wäre die Auswertung von Sensoren oder Ansteuerung von Schaltern. Der Kreativität ist hier keine Grenze gesetzt.

Benutzung von anderen Computern:

Der Flask Webserver hört standardmäßig nur auf Anfragen von dem Computer, auf dem der Server läuft. Dies liegt daran, dass Debug Modus, der standardmäßig aktiviert ist, Remote Code Execution möglich ist, wodurch es gefährlich wäre, den Server offen im Netzwerk zu hosten. Falls es notwendig sein sollte, den Server zu öffnen, sollten Sie die entsprechende Dokumentation von Flask lesen, verstehen, warum diese Entscheidung von den Entwicklern getroffen wurde und die entsprechend beschriebenen Änderungen vornehmen. Dann ist der Server im ganzen Netzwerk beziehungsweise nach der Einrichtung einer Portweiterleitung im Router auch aus dem Internet erreichbar.