Frameworks

Allgemeines

Ein Framework stellt einen Rahmen / ein Gerüst dar, das als Basis eines komplexeren Projekts dient. Es stellt bestimmte Grundfunktionalitäten bereit, die die Entwicklung beschleunigen, da diese nicht von Grund auf neu entwickelt werden müssen, sondern einfach wiederverwendet werden können.

Dabei können einzelne Bestandteile / Module des Frameworks erweitert, abgeändert, entfernt oder in sonstiger Art und Weise modifiziert werden, um den Anforderungen des Projekts und des Entwicklers zu entsprechen.

Der Unterschied zwischen Frameworks und Bibliotheken liegt darin, dass Bibliotheken fertige abgeschlossene Funktionalitäten enthalten, die einfach in den Code des Entwicklers eingebunden und genutzt werden können. Frameworks dagegen sind diesbezüglich viel offener, sie stellen lediglich eine Grundlage dar, auf die der Nutzer aufbauen kann und diese nach seinen eigenen Wünschen und Anforderungen erweitern kann.

Der Zweck eines Frameworks liegt in der Wiederverwendbarkeit: Dadurch, dass die meisten Funktionen eher abstrakt gehalten sind und nicht für eine spezifische Anwendung gestaltet sind, können sie von vielen Nutzern verwendet werden, ohne den Kern des Frameworks anpassen zu müssen.

Ein Framework allein ist zwar in vielen Fällen ausführbar und lauffähig, für einen sinnvollen projektbezogenen Einsatz muss es jedoch erweitert werden.

Das bedeutet, dass der Anwender Funktionalitäten entwickelt, die vom Framework gesteuert und ausgeführt werden. Auch hier zeigt sich der Unterschied zu einer Bibliothek: Bei einer Bibliothek wird auf die Funktionen aus dieser Funktionalität zugegriffen und diese aktiv vom Nutzer ausgeführt, während er bei einem Framework dagegen die Funktionalität definiert, welche vom Framework kontrolliert bzw. ausgeführt wird. Dieses *Pattern* wird als *Inversion of Control* bezeichnet.

Arten von Frameworks

Es gibt die unterschiedlichsten Arten von Frameworks, da diese wie beschrieben äußerst vielseitig und universell eingesetzt werden können. Dazu gehören etwa *Anwendungsframeworks*, die beispielsweise Grundfunktionalitäten für eine generische Anwendung bereitstellen, z.B. eine Desktopanwendung. Auch für Softwaretests gibt es eine große Menge an Frameworks, diese werden als *Testframeworks* bezeichnet. Die wohl verbreitetste Art von Frameworks sind aber die *Webframeworks*, die genutzt werden, um größere / komplexere Webanwendungen zu entwickeln. Sie stellen unterschiedlichste Funktionalitäten wie beispielsweise Datenbankanbindungen, Templating, Login-Systeme, Routing etc. bereit.

Flask Framework

Das Flask-Framework ist ein Webframework, das in Python geschrieben ist und zur Entwicklung von Webanwendungen in Python genutzt wird. Es wurde erstmals 2010 von Armin Ronacher veröffentlicht. Ursprünglich handelte es sich dabei um einen Aprilscherz des Entwicklers, stieß im Anschluss jedoch auf große Akzeptanz der Python-Community, sodass es sich zu einem der meistgenutztesten Python-Frameworks entwickelte. (http://mitsuhiko.pocoo.org/flask-pycon-2011.pdf)

Es wird als *"Microframework"* bezeichnet, da es auf den Kern reduziert ist und im Gegensatz zu anderen Frameworks wie etwas das ebenfalls weitverbreitete *Django*-Framework auf viele Komponenten verzichtet, wodurch die Codebasis schlank gehalten wird und dem Entwickler die Möglichkeit gegeben wird, es nach seinen Wünschen zu erweitern.

Flask zeichnet sich auch durch eine umfassende und ausführliche Dokumentation aus, die auf der Website

(http://flask.pocoo.org/) zu finden ist.

Flask beinhaltet auch einen eigenen kleinen Webserver, der jedoch nur zur Entwicklung genutzt werden sollte, im Livebetrieb sollte auf einen sichereren und stabileren Webserver zurückgegriffen werden.

Mithilfe von Flask können Webanwendungen in nur wenigen Zeilen erstellt werden, im Folgenden ist ein Minimalbeispiel dargestellt:

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

@app.route("/")
def hello():
    return "Hallo Welt"

if __name__ == "__main__":
    app.run()
```

Dazu muss das Flask-Paket zunächst installiert werden:

`pip install Flask`

Nun kann die Flask-Klasse importiert werden. Anschließend wird eine Instanz dieser Klasse erzeugt und in der `app`-Variable gespeichert. Als Argument wird hier die spezielle Variable `__name__` übergeben. In dieser wird automatisch der Name des jeweils aktuellen Moduls gespeichert, falls das Skript direkt ausgeführt wird nimmt `__name__` den Wert `__main__` an.

Anschließend wird definiert, welche Funktion ausgeführt werden soll, wenn eine bestimmte Route (URL) aufgerufen wird. Die Base-URL ist immer *"/"*, also beispielsweise `http://localhost:8080/`. Wenn also im Beispiel die Base-URL aufgerufen wird, wird die Funktion `hello()` ausgeführt, die einfach nur den String *"Hallo Welt"* an den Client zurückgibt. Die Route ist hier durch einen sogenannten *Dekorator* festgelegt.

Zuletzt wird die Webanwendung gestartet, indem `app.run()` ausgeführt wird. Hierdurch wird der Development-Webserver gestartet (z.B. unter `http://127.0.0.1:5000/`) und wartet nun auf eingehende Requests.

Aufgabe:

1. Erweitere das oben gezeigte Minimalbeispiel so, dass beim Aufruf von `/info` der Name des jeweiligen Moduls an den Client zurückgegeben wird.
2. Erweitere das Beispiel so, dass das aktuelle Datum und die aktuelle Uhrzeit an den Client zurückgegeben wird (Format: "08.05.2019 10:15:30") .
3. Erweitere das Beispiel so, dass bei allen anderen URLs, die nicht explizit definiert sind (404-Fehler), automatisch eine Weiterleitung zur "/"-Route erfolgt.
4. Erweitere das Beispiel so, dass bei Aufruf der Route `/image` ein Bild deiner Wahl zurückgegeben wird.