APIs

Allgemeines

API steht für *Application Programming Interface*, also für Programmierschnittstelle.

Damit kann eine Anwendung bestimmte Funktionalitäten und Programmteile für andere, externe Anwendungen zur Verfügung stellen. Eine API besitzt also Ähnlichkeiten zur Einbindung und Verwendung von externen Bibliotheken, tatsächlich wird eine Verwendung von solchen Bibliotheksfunktionen im weiteren Sinne oft als API-Aufruf bezeichnet.

Eine API ist eine Menge von klar definierten Methoden, die in den unterschiedlichsten Bereichen eingesetzt werden: Auf Betriebssystemebene finden sich bspw. Schnittstellen zwischen den Hardware-Controllern und dem Betriebssystem bzw. Schnittstellen zwischen den Anwendungen und dem Betriebssystem. Auch Frameworks und Bibliotheken bieten wie beschrieben Schnittstellen, während die API selbst das erwartete Verhalten spezifiziert, während die Bibliothek selbst die eigentliche Implementierung dieser Regeln darstellt. Ein sehr weit verbreiteter Gebrauch von APIs findet sich auch im Web-Umfeld (siehe unten).

Vorteile von APIs

APIs bieten unterschiedliche Vorteile, die einen Einsatz oft unverzichtbar machen. Einige dieser Vorteile sind im Folgenden aufgeführt:

- *Abstraktion/Entkopplung*: APIs bieten eine starke Entkopplung zwischen Spezifizierung und eigentlicher Implementierung der Funktionalität. Selbst wenn sich die Implementierung eines Services grundlegend ändert, können über APIs trotzdem die gleichen Funktionalitäten bereitgestellt werden. Die Gegenseite benötigt kein Verständnis über die Implementierungsdetails.
- *Attraktivität des Anbieters*: Indem ein Anbieter Schnittstellen zu seinem System bereitstellt, können dafür zusätzliche Funktionalitäten durch externe Entwickler entwickelt werden, was sich positiv auf die Wahrnehmung des Unternehmens auswirken und einen Wettbewerbsvorteil darstellen kann.
- *Zugriffssteuerung*: Durch die Abstraktion bzw. Entkopplung zwischen Implementierung und Spezifizierung der Funktionalität lassen sich Zugriffe feiner steuern

Web Services / Web APIs

Eine der am häufigsten verbreiteten Arten von Programmierschnittstellen sind die Web APIs. Diese werden nicht wie Bibliotheken direkt eingebunden oder sind wie Betriebssystemschnittstellen bereits auf dem System vorhanden, sondern diese werden über Webaufrufe an den entfernten Anwendungsserver (Remotesystem) ausgeführt. Hieraus ergibt sich eine triviale Grundvoraussetzung: Beide Systeme (das aufrufende sowie das bereitstellende System) müssen mit dem Internet verbunden sein, damit die Schnittstellen genutzt werden können (was bei Embedded Systemen oftmals nicht möglich oder gewünscht ist).

Der Anwendungsserver stellt die Funktionalität dann über ein bestimmtes Protokoll, meist HTTP, zur Verfügung. Außerdem bedarf es zur Nutzung eine Spezifikation, da der Konsument / Nutzer der Schnittstelle das zugrundeliegende System nicht kennt und daher eine Beschreibung benötigt, welche Funktionalität mit welchen Aufrufen und Parametern ausgeführt werden kann.

Die Beschreibung umfasst für jede Schnittstelle meist mehrere Bestandteile, die der Konsument für eine reibungslose Nutzung benötigt:

- *Protokoll*: Über welches Protokoll kann die Schnittstelle aufgerufen werden (z.B. HTTP, SOAP, RPC, …)
- *Endpoint*: Unter welcher Basis-URL und welcher Route kann die Funktionalität aufgerufen werden (z.B. Basis-

URL `api.example.com` und Route `users/`)

- *Methode*: Falls HTTP-Protokoll: Mit welcher HTTP-Methode kann die Schnittstelle aufgerufen werden (z.B. GET bei Datenausgabe, POST bei Dateneingabe, …)
- *Inputdaten*: Welche Inputparameter und Eingabedaten werden benötigt (z.B. spezifische Nutzer-ID, Datumseinschränkungen, …)
- *Inputkodierung*: Wie werden die Eingabedaten übergeben (z.B. bei der GET-Methode meist über Query-Parameter bzw. als URL-Bestandteil, bei POST-Methode meist im Body als JSON-, XML- oder einfaches Textdokument)
- *Outputdaten*: Welche Daten werden von der Schnittstelle zurückgegeben
- *Outputkodierung*: Wie werden die Daten zurückgegeben (z.B. JSON- oder XML-kodiert, einfache Textausgabe, …)

Außerdem werden bei manchen Systemen noch weitere Informationen mit angegeben:

- *Rate-Limit*: Um Missbrauch zu vermeiden (DOS-Attacke), stehen dem Nutzer oft nur eine limitierte Anzahl an Aufrufen innerhalb einer bestimmten Zeit zur Verfügung. In der Dokumentation kann spezifiziert werden, wie hoch diese Rate ist.
- *Kosten*: Bei den meisten kostenpflichtigen APIs werden die Kosten je nach Anzahl der Aufrufe ermittelt, teilweise unterscheiden sich die Kosten pro Endpoint (z.B. kann ein POST-Request teurer sein als ein GET-Request)
- *Nutzungsbedingungen/Lizenz*: Oftmals beinhalten die Spezifikationen auch eine Lizenz, in der festgelegt wird, inwiefern die Daten weitergenutzt werden dürfen, z.B. Ausschluss kommerzieller Nutzung.

Diese Informationen reichen aus, um Funktionalitäten eines externen Systems nutzen zu können und in das eigene Programm einzubauen. Dabei können die APIs als "Bausteine" genutzt werden, um die Funktionalitäten aus verschiedenen Systemen zu kombinieren und daraus wiederum eine eigene Anwendung zu kreieren.

REST

Um Schnittstellen einheitlich zu gestalten, damit sich Entwickler nicht bei jeder neu zu implementierenden API erneut in eine komplett unterschiedliche Schnittstellenarchitektur einarbeiten müssen, wurde im Jahr 2000 von Roy Fielding das Prinzip des sogenannten Representational State Transfer (REST) geschaffen.

Unter diesem Begriff sind verschiedene "Regeln" zusammengefasst, die zur Entwicklung und der Bereitstellung von APIs gelten sollen. Web Services, die diese Regeln berücksichtigen, werden als *RESTful Web Services* bezeichnet.

> Ziel ist es, eine einfache Möglichkeit zur Abfrage und Manipulation textbasierter Repräsentationen von Webressourcen zu schaffen, indem eine einheitliche und vordefinierte Menge an zustandslosen Operationen genutzt wird.

REST liegen 6 Regeln zugrunde, die erfüllt sein müssen, um einen Webservice als RESTful bezeichnen zu können:

- *Client-Server Architektur*: *Separation of concerns*
- *Zustandslosigkeit*: Zwischen den einzelnen Requests wird auf dem Server kein clientbezogener Zustand / Kontext gespeichert. Dieser wird allein vom Client verwaltet und muss falls nötig bei jedem Request mit übermittelt werden
- *Caching*: Requests und Responses können bzw. sollen gecached werden. Responses müssen daher eine Indikation besitzen, ob sie vom Client gecached werden können
- *Mehrschichtige Systeme*: Systeme sollen mehrschichtig aufgebaut werden (Zwischen API und Datenbank können weitere Systeme wie Load Balancer, Caching System, … liegen), dem Nutzer soll diese Architektur aber verborgen bleiben
- *Einheitliche Schnittstelle*:
  - *Adressierbarkeit von Ressourcen*: Individuelle Ressourcen werden z.B. als Teil der URL in den Requests

identifiziert.
    - *Repräsentation der Manipulation von Ressourcen*: Falls ein Client eine Repräsentation einer Ressource besitzt, hat er ausreichend Informationen, diese Ressource zu modifizieren oder zu löschen
    - *Selbstbeschreibende Nachrichten*: Jede Nachricht enthält genug Informationen, um zu beschreiben, wie diese Nachricht verarbeitet wird.
    - *Hypermedia as the engine of application state (HATEOAS)*: Sobald die initiale URL der REST Anwendung besucht wurde, soll der REST Client in der Lage sein, durch den Server bereitgestellte Links zu nutzen, um dynamisch alle verfügbaren Aktionen und Ressourcen zu entdecken.
- *Code on demand* (Optional): Server kann im Bedarfsfall ausführbaren Code an den Client übermitteln

JSON-Format

Für Web APIs gibt es kein universelles Format, in dem die Daten zwischen Client und Server ausgetauscht werden. Vielmehr gibt es eine Vielzahl an Möglichkeiten, dazu zählen z.B. das XML-Format, das CSV-Format, einfache Textform, usw. Jedoch hat sich in den letzten ein Format besonders etabliert, das inzwischen für den Datenaustausch bei den meisten modernen Schnittstellen genutzt wird: Das JSON-Format.

JSON steht für *JavaScript Object Notation* und bietet mehrere Vorteile: Es besitzt eine sehr einfache Struktur, es wird in Textform ausgetauscht (im Gegensatz zu Binärformaten, die für den Menschen nicht lesbar sind), und es ist sehr übersichtlich, was zur schnellen Auswertung und Überprüfung hilfreich ist.

Zudem besteht im Kontext der Nutzung von Python ein weiterer großer Vorteil bei der Nutzung von JSON: Python Dictionaries (`dict()` bzw. `{}`) sind größtenteils kompatibel zum JSON-Standard, sodass eine aufwändige und komplexe Konvertierung nicht nötig ist.

JSON besteht wie Python Dicts auch aus Key-Value Paaren, wobei die Values wiederum Dictionaries (bei JSON *"Objekte"* genannt) sowie Listen (bei JSON *"Arrays"* genannt) enthalten können. Unterschied: Bei Python Dicts können fast alle Objekte als Key genutzt werden, während sich die Keys bei JSON auf Strings beschränken.

Die Ähnlichkeit wird durch folgendes Beispiel offensichtlich:

```
import json

json_string = '{
        "status_code": 200,
        "success": true,
        "content": [
                {
                        "title": "Sein größter Irrtum",
                        "teaser": "WikiLeaks-Gründer Julian Assange hoffte auf Donald
Trump und verhalf ihm zur Macht. Jetzt lässt die US-Regierung ihn jagen.",
                        "url": "https://www.zeit.de/2019/17/julian-assange-donald-
trump-usa-wikileaks"
                },{
                        "title": "Aretha Franklin posthum mit Pulitzersonderpreis
ausgezeichnet",
                        "teaser": "Die verstorbene Soulsängerin wird als erste Frau
mit einem Sonderpreis geehrt. Sie habe fünf Jahrzehnte lang unauslöschliche Beiträge
zur Kultur geleistet, hieß es.",
                        "url": "https://www.zeit.de/kultur/musik/2019-04/pulitzer-
preis-aretha-franklin-posthum-parkland"
                }
        ]
}'


python_obj = json.loads(json_string)
```

```
print(python_obj)

>> {
        'status_code': 200,
        'success': True,
        'content': [
                {
                        'title': 'Sein größter Irrtum',
                        'teaser': 'WikiLeaks-Gründer Julian Assange hoffte auf Donald
Trump und verhalf ihm zur Macht. Jetzt lässt die US-Regierung ihn jagen.',
                        'url': 'https://www.zeit.de/2019/17/julian-assange-donald-
trump-usa-wikileaks'
                }, {
                        'title': 'Aretha Franklin posthum mit Pulitzersonderpreis
ausgezeichnet',
                        'teaser': 'Die verstorbene Soulsängerin wird als erste Frau
mit einem Sonderpreis geehrt. Sie habe fünf Jahrzehnte lang unauslöschliche Beiträge
zur Kultur geleistet, hieß es.',
                        'url': 'https://www.zeit.de/kultur/musik/2019-04/pulitzer-
preis-aretha-franklin-posthum-parkland'
                }
        ]
}
```

Wie man erkennen kann, sind die beiden Darstellung fast identisch. Es muss lediglich darauf geachtet werden, dass die Strings im JSON-Format jeweils mit `"` umschlossen werden und nicht mit `'`. Außerdem werden die boolschen Werte im JSON-Format kleingeschrieben (`true` und `false`), während sie in Python großgeschrieben werden (`True` und `False`). Der Rest ist identisch, somit eignet sich der Datenaustausch im JSON-Format sehr gut in der Kommunikation zu APIs in Python.

API-Nutzung in Python

Für das hier umzusetzende Projekt können APIs auf zwei Arten eingebunden werden: Eine Möglichkeit besteht darin, dem Nutzer die Seite auszugeben, nachdem diese geladen wurde werden die APIs clientseitig, also im Browser des Nutzers aufgerufen. Die Daten, die von den APIs zurückgegeben werden werden dann durch eine clientseitige Webprogrammiersprache, also JavaScript, dynamisch in die geladene Seite eingefügt. Eine weitere Möglichkeit ist es, die APIs bereits nach dem Request des Nutzers vom Server aus aufzurufen und die erhaltenen Daten anschließend über Templates in die Website einzufügen und diese letztendlich an den Nutzer zurückzugeben.

Wir bedienen uns hier der zweiten Möglichkeit. Dafür wird die Python-Library `requests` genutzt, die, wie der Name bereits vermuten lässt, zum Senden und Empfangen von Web Requests entwickelt wurde.

Diese Bibliothek ist speziell zur Nutzung von Web-APIs konzipiert und äußerst einfach in der Bedienung.

Beispiel:

Abfrage von Breaking-News Headlines:

```
import requests
url = ('https://newsapi.org/v2/top-headlines?'
       'country=de&'
       'apiKey=6e04c7acdd4147a8be769b550ab5c311')

response = requests.get(url)
if response.ok:
        print response.json()
```

```
{
        "status": "ok",
        "totalResults": 34,
        "articles": [
                {
                        "source": {
                                "id": null,
                                "name": "T-online.de"
                        },
                        "author": "rtr",
                        "title": "Dieselskandal: Staatsanwaltschaft erhebt Anklage
gegen Ex-VW-Chef Martin Winterkorn - t-online.de",
                        "description": "Der Dieselskandal holt Martin Winterkorn ein:
Die Staatsanwaltschaft Braunschweig hat den ehemaligen VW-Chef angeklagt. Der Vorwurf
lautet auf schweren Betrug.",
                        "url": "https://www.t-
online.de/nachrichten/id_85588052/dieselskandal-staatsanwaltschaft-erhebt-anklage-
gegen-ex-vw-chef-martin-winterkorn.html",
                        "urlToImage": "https://bilder.t-
online.de/b/85/58/80/88/id_85588088/tid_da/martin-winterkorn-der-ehemalige-vw-
manager-wird-angeklagt-.jpg",
                        "publishedAt": "2019-04-15T10:57:00Z",
                        "content": "Der Dieselskandal holt Martin Winterkorn ein: Die
Staatsanwaltschaft Braunschweig hat den ehemaligen VW-Chef angeklagt. Der Vorwurf
lautet auf schweren Betrug. Winterkorn, der nach Bekanntwerden der millionenfachen
Dieselmanipulation im September 2015 zurückg… [+1092 chars]"
                },
                ...
        ]
}
```


Zunächst wird `requests` importiert. Anschließend wird die URL definiert, von der die Nachrichten abgefragt werden sollen. Die Base-URL hierbei ist `https://newsapi.org/`, der Endpoint ist `v2/top-headlines`. Es handelt sich also um die 2. Version der API, außerdem erreichen wir unter diesem Endpoint die *Top Headlines*. Die Adresse bzw. Ressource ist selbstbeschreibend und entspricht damit einem Kernprinzip der RESTful Webservices. Außerdem müssen der URL 2 weitere Parameter angehängt werden: Zum einen der Code des Landes, für das die Nachrichten abgefragt werden und außerdem der API-Key, der zunächst erstellt werden muss. Mit diesem kann z.B. bei Missbrauch der API die Nutzung unterbunden werden. Da es sich hierbei um einen GET-Request handelt besitzt der Request keinen Body, daher werden die Parameter direkt über Query-Strings an die URL angehängt. Mehrere Parameter zu einem Query-String zu kombinieren kann schnell unübersichtlich werden und daher zu Fehler führen. Aus diesem Grund bietet `requests` eine einfachere Möglichkeit, die Parameter der URL anzuhängen:

```
url = 'https://newsapi.org/v2/top-headlines'
payload = {'country': 'de', 'apiKey': '6e04c7acdd4147a8be769b550ab5c311'}

response = requests.get(url, params=payload)
print(response.url)
```

ergibt die gleiche URL:

```
'https://newsapi.org/v2/top-headlines?
country=de&apiKey=6e04c7acdd4147a8be769b550ab5c311'
```

Letztendlich wird der Request durch die `get()`-Methode ausgeführt und das Ergebnis in der `response`-Variable gespeichert. Nun wird überprüft, ob der Request erfolgreich war (`ok`-Property des Response-Objekts). Dies ist der Fall, wenn ein positiver Statuscode (z.B. 200) zurückgegeben wurde. Falls der Requests erfolgreich war, können die Nutzdaten mit der `json`-Methode ausgelesen werden, da die Ergebnisse laut API-Dokumentation im JSON-Format

zurückgegeben werden. Da JSON-Objekte einfach durch Python Dictionaries dargestellt werden können, ist eine Konvertierung nicht nötig.

Analog zur `get()`-Methode ist auch die `post()`-Methode verfügbar, die genutzt werden kann, um Daten an eine Schnittstelle zu senden. Hierbei muss jeweils der Inhalt des Bodys übergeben werden, z.B.

```
response = requests.post('https://example.com/api/post', data = {'key1': 'value1',
'key2': 'value2'})
```

JSON-Objekte müssen dabei serialisiert, also als Strings übergeben werden. Auch hierfür gibt es von `requests` eine Hilfsmethode, bei der die Serialisierung automatisch erfolgt:

```
response = requests.post('https://example.com/api/post', json = {'key1': 'value1',
'key2': 'value2'})
```

Hier wird das Python Dict direkt als Objekt übergeben, jedoch nicht als `data`-Parameter, sondern als `json`-Parameter. Somit weiß `requests`, dass das Objekt serialisiert werden muss.

Aufgabe

1. News API Vorbereitung: Erzeuge einen API-Key auf https://newsapi.org/. Dieser wird für die nachfolgenden Aufgaben benötig.
    1. Erweitere die Funktionalität der `/`-Route so, dass beim Aufruf die Titel der ersten 10 Top-Headlines aus einem beliebigen Land deiner Wahl angezeigt werden. Diese sollen jeweils als Paragraph-Element im HTML-Code ausgegeben werden.
    2. Erweitere die vorherige Aufgabe so, dass der Titel jeder Headline jeweils auf den Originalartikel verlinkt. Wenn der Nutzer auf den Titel klickt, soll er zur Website des Originalartikels geleitet werden.
    3. Erweitere vorherige Aufgabe so, dass bei einer Auswahl einer Nachrichten-Kategorie lediglich die Top-10 Nachrichtentitel dieser Kategorie angezeigt werden.
2. Wikipedia API
    1. Suche mithilfe der Wikipedia-API nach Wikipedia-Einträgen zu einem bestimmten Stichwort (z.B. "Stuttgart") und gebe die Titel dieser Einträge sowie deren Snippets analog zu Aufgabe 1 in einer Liste aus.

    Tip zu den Snippets: Jinja bietet verschiedene Filter zur Verarbeitung von Inhalten, darunter der safe-Filter zum HTML-Escaping

    2. Erweitere vorherige Aufgabe so, dass der Nutzer in der Eingabezeile ein beliebiges Stichwort eingeben kann und dann die Ergebnisse entsprechend diesen Stichworts angezeigt werden.
3. Pastebin API Vorbereitung: Erstelle einen Account auf Pastebin. Hierdurch erhält jeder Nutzer automatisch einen API-Key. Dieser wird für die nachfolgenden Aufgaben benötig (Nach Registrierung / Login wird dieser auf https://pastebin.com/api#1 angezeigt)
    1. Der Text, der in der entsprechenden Textarea eingegeben wird, soll über einen API-Request in Pastebin als neuer *Paste* gespeichert werden. Nach dem Anlegen soll die URL des zuletzt erzeugten Pastes als Link unter der Textarea angezeigt werden.
4. Mercedes-Benz API Vorbereitung: Erstelle einen Account auf Mercedes Benz /developers. Subscribe die *Vehicle Image API*, hierdurch erhältst du eine App-ID sowie einen API-Key, die für die folgenden Aufgaben benötigt werden.
    1. Anhand der Dropdown-Liste sollen dem Nutzer die Bilder zu den entsprechenden Modellen angezeigt werden.

5. Binde weitere APIs ein, z.B. Instagram, Wetter, Deutsche Bahn, SpaceX, IBM Text to Speech, NASA, RandomCat, …