Aufgaben für 'Einführung in Web-APIs'

Einleitung

In diesem Dokument findet sich eine Sammlung an kleinen Ideen zum Anwenden des eigenen Wissens zu WebAPIs. Diese Liste soll nicht zum abhaken dienen sondern im Idealfall bei einigen Einträgen das eigene Interesse wecken.
Im Allgemeinen sind die Aufgaben unterschiedlich schwer und reichen von einer Umsetzung im Besuchen einer Webseite bis zum Implementieren von Python-Scripts.

Zusatzinformationen:

- Einige Einträge haben eine "Weiterführende Aufgabe" in der die Daten meistens in einem Python-Script aufgearbeitet (und visualisiert) werden. Hierfür benötigt ihr idealerweise eine bestehende Python3-Installation. Zusätzlich benötigt ist immer die Requests Library, abhängig von der Aufgabe kommen auch noch weitere Dependencies dazu.

  ```
  Installation Requests über pip:
  pip3 install requests
  ```

- Um die Bandbreitenauslastung (für andere Workshops) möglichst niedrig zu halten bitte ich euch darum die Anzahl eurer Requests gering zu halten (idealerweise <10\Minute)
- Falls ihr einen Response-Code bekommt den ihr nicht vollständig versteht ist diese Webresource sehr hilfreich!

Viel Spaß!

1. Shared Mobility GBFS (CallABike)

Die Shard Mobility GBFS API der DB erlaubt Abfragen zum Status von Leihservices (hier: CallABike) der Deutschen Bahn

Aufgabe:

Findet heraus wo aktuell freie Fahrräder von CallABike stehen.
Für diese Aufgabe ist Postman empfehlenswert, da ARC nicht gut mit den großen Rückgaben umgeht.
Weiterhin ist die weiterführende Aufgabe sehr zu empfehlen.

Vorgehensweise:

1. Finden des korrekten Endpunkt für Informationen über freie Fahrräder
2. Abfragen (und Verstehen) der Daten

Relevante Daten:

- DB-Client-ID: ec0b224441a9443450c41a514bbbb38b
- DB-Api-Key: 43d6309a2538cdca051d8986d4f21c43
- Documentation

Weiterführende Aufgabe:

Visualisiert mit Python die Positionen der freistehenden Fahrräder auf einer Deutschlandkarte.
Klont euch hierfür das Beispielrepository und implementiert die TODO in `Bahn/GBFS/GBFSMap.py`.
Der Script stellt euch schon das gesamte Framework für die Visualisierung bereit, ihr müsst also nur die Positionsdaten abrufen und zurückgeben. Das Script generiert eine `germany_map.html` Datei die ihr euch im Browser anzeigen könnt.
Zusätzlich zu Requests wird `pip3 install folium` benötigt.

2. Bahn RIS::Stations

Die RIS::Stations API bietet Informationen zu Bahnhöfen der Deutschen Bahn an

Aufgabe:

Findet heraus welche Services der Darmstadt Hauptbahnhof anbietet und ob öffentliches WLAN dabei ist.

Vorgehensweise:

Die Schwierigkeit dieser Aufgabe liegt in der Komplexität der angebotenen Endpunktes. Es gilt herauszufinden welche Endpunkte tatsächlich notwendig sind.

1. Finden der "stationId" des Darmstädter Hbfs
2. Finden des relevanten Endpunkt für "Services" von Bahnhöfen
3. Einstellen der korrekten Parameter für Abfrage *entweder* nach Darmstadt Hbf oder WIFI

Relevante Daten:

- DB-Client-ID: ec0b224441a9443450c41a514bbbb38b
- DB-Api-Key: 43d6309a2538cdca051d8986d4f21c43
- Documentation

3. Bahn FaSta (Facility Status)

Die FaSta API bietet u.a. Daten zum aktuellen Status von Aufzügen und Rolltreppen in Bahnhöfen der Deutschen Bahn an

Aufgabe:

Findet heraus welcher Prozentsatz aller Aufzüge in Bahnhöfen der Deutschen Bahn defekt sind.
Diese Aufgabe lässt sich auch direkt in Python implementieren (siehe Beispielrepository).

Vorgehensweise:

1. Finden des korrekten Endpunkt für Informationen über Aufzüge
2. Abfragen der Gesamtzahl von Aufzügen
3. Abfragen der Gesamtzahl von kaputten Aufzügen

Relevante Daten:

- DB-Client-ID: ec0b224441a9443450c41a514bbbb38b
- DB-Api-Key: 43d6309a2538cdca051d8986d4f21c43
- Documentation

4. NASA NeoWs (Near Earth Object Web Service)

Die NeoWs Schnittstelle der NASA gibt Asteroiden-Daten und die Daten zu Erdannäherungen zurück.

Aufgabe:

Findet heraus welche Asteroiden der Erde an eurem Geburtstag am nähesten gekommen sind.

Vorgehensweise:

1. Finden des korrekten Endpunkt für die Informationen
2. Abfragen der Daten mit den korrekten Parametern

Relevante Daten:

- NASA-Api-Key: RUO4ABnSnWSZ0JulyEqzIN1inFD55AxnuzOf8EBY
- Documentation (Unterkategorie “Asteroids NeoWs”)

Weiterführende Aufgabe:

Schreibt einen Python-Script welcher für ein gegebenes Datum den Asteroiden zurückgibt, der der Erde am nähesten gekommen ist. Siehe hierfür die Vorlage im Beispielrepository

5. NASA Earth

Die NASA Earth API bietet Satellitenaufnahemen des LANDSAT 8 (seit 2013 im Betrieb) für jeden Ort der Erde an.

Aufgabe:

Findet das Bild des LANDSAT 8 welches er zu Beginn der Coronapandemie (wir gehen vom 15.01.2020 aus) von Darmstadt aufgenommen hat. Hier lässt sich die "Weiterführende Aufgabe" sehr empfehlen.

Vorgehensweise:

1. Finden des korrekten Endpunkt für die Informationen
2. Abfragen der Daten mit dem korrektem Datum

Tipps:

1. Beachtet die Formatierung des Datums
2. Es gibt zwei Endpunkte. Sucht den, der das Bild ausgibt welches am nähesten am Datum liegt
3. Der Endpunkt gibt euch nicht das Bild sondern eine URL des Bildes zurück
4. Über den Parameter `dim` könnt ihr den "Zoom" des Bildes bestimmen. Es empfiehlt sich ein Wert über 0.1

Relevante Daten:

- NASA-Api-Key: RUO4ABnSnWSZ0JulyEqzIN1inFD55AxnuzOf8EBY
- Documentation (Unterkategorie "Earth")

Weiterführende Aufgabe:

Visualisiert den Wandel des Wetters und der Stadt über die Zeit hinweg. Hierfür wollen wir ein Programm schreiben was aus einer Zeitspanne die Bilder des LANDSAT 8 holt und diese chronologisch anzeigt. Implementiert hierfür die TODOs in der Vorlage der Klasse `NASA/Earth/darmstadt.py` im Beispielrepository. *Beachtet dass diese Aufgabe extrem viele Anfragen benötigt. Um unser Netzwerk zu schonen würde ich euch bitten das Script nur einmal auszuführen.*
Zusätzlich zu wird `pip3 install pillow` und `pip3 install aiohttp` benötigt.

6. HTTP Cats

*HTTP Cats ist ein Verzeichnis von Katzenbildern für alle HTTP-Status-Codes. Diese Aufgabe dient mehr zum Verständnis der HTTP-Status-Codes, da der Endpunkt der API **sehr** einfach ist.*

Vorraussetzung:
Das Anzeigen von Bildern scheint mit ARC nicht zu funktionieren. Benutzt für die Web-Requests entweder Postman oder einen Browser eurer Wahl (das Besuchen einer bestimmten URL im Browser ist nichts anderes als ein HTTP-GET an diese URL).

Aufgabe:

Findet die URL des Katzenbildes welches den HTTP-Status-Code 418 darstellt.

Relevante Daten:

- Webseite

Weiterführende Aufgabe:
Implementiert den sehr sinnvollen Wrapper `Cats/HTTPCats/cat_wrapper.py` im Beispielrepository. Dieser soll die beiden Methoden `requests.get` und `requests.post` ausführen und zusätzlich das passende Katzenbild zu dem zurückgegebenen Statuscode öffnen.
Ihr findet in `Cats/HTTPCats/cat_facts.py` ein Beispiel welches ihr ausführen könnt um eure Implementierung zu testen. Beachtet hierfür dass jede Ausführung die Bilder neu downloaded.
Zusätzlich zu Requests wird `pip3 install pillow` benötigt.

7. BeatSaver Tag Search

BeatSaver ist eine Plattform zum Teilen von BeatSaber-Maps, die auch eine API anbietet.

Aufgabe:

Finde die Tags (Musik-Genres) der Map des Songs “Seeing Red - Architects”.

Vorgehensweise:

1. Finden des korrekten Endpunktes in der Dokumentation
2. Abfragen der Daten zum spezifischen Song
3. Suchen der Tags in den Antwortdaten

Tipps:

1. Die API-URL ist https://api.beatsaver.com/
2. Die Dokumentation von BeatSaver hat ein gutes Try-Out Feature, deshalb funktioniert diese Aufgabe auch ohne ARC oder Postman
3. Beachtet bei einer Suche die sortOrder der Ergebnisse. Nicht alle Sortierungen werden die Map ganz oben anzeigen!

Relevante Daten:

- Documentation

8. Climatiq C02e Estimate

Climatiq ist eine Plattform die C02-equivalente (C02e) für unterschiedliche "Emission Activities" liefert. Diese Aufgabe ist eine der komplexeren, insbesondere erfolgt die Datenabfrage hier über einen POST

Aufgabe:

Vergleiche den abgeschätzten C02e für einen Inlandsflug von 500km mit dem einer (high-speed) Zugfahrt von 500km für eine Person.

Vorgehensweise:
Bei dieser Aufgabe habe ich schon die konkreten Endpunkte angegeben, da es extrem viele unterschiedliche Datenquellen gibt. Beachte hier auch, dass dieses Beispiel durchaus konstruiert ist, je nach Datenquelle unterscheiden sich die Ergebnisse erheblich!

1. Abfragen der aktuellen Datenversion
2. Konstruieren und Abfragen der C02e für die Zugfahrt/den Inlandsflug

Tipps:
1. Die Dokumentation stellt die Requests als curl da, wenn ihr euch dieses ein wenig anschaut solltet ihr trotzdem verstehen wie der Request aussehen muss
2. Achtet insbesondere bei den C02e Abfragen auf den Request-Typen (POST) und darauf, dass die Daten hier als JSON-Body übergeben werden (bei curl ist dies das 'data' Feld). Das "parameters" in diesem ist also ein JSON-Feld und kein URL-encoded-Parameter!

Relevante Daten:
- Documentation Data Versions
- Documentation Train
- Documentation Plane
- Authentication: Bearer Z8G2QXCKTP4BR4JVWST97SSFCM0Q