Web Services Python

Introductieles: web API's gebruiken met Python

Kristof Michiels

Inhoud

- Web Services Python?
- Python en API's
- HTTP-requests maken met Python
- Inhoud labo 1
- Vrijblijvende labo-voorbereidende oefeningen

Web Services Python?

Web Services Python?

- Je ontwikkelt verschillende types web services en maakt hiervoor gebruik van de meest geschikte Python frameworks en technieken
- Je vertrekt van het consumeren van third-party API's en integreert ze in een eigen toepassing. Daarna maak je een eigen web service met het micro-framework Flask. Vervolgens exploreer je de mogelijkheden van het Django framework.
- Je ontsluit hierbij verschillende types achtergelegen databronnen (JSON, SQL en NoSQL databanken).

Python en API's

Python en API's

- Wat zijn API's? Belangrijke API-gerelateerde concepten
- Hoe een API gebruiken met behulp van Python
- Hoe Python gebruiken om beschikbare online API-data te lezen

We concentreren ons deze week op het *consumen* van API's. Met het binnentrekken en integreren van externe data kan je al heel wat doen. Bovendien is het een belangrijk facet van het bouwen van API-gedreven toepassingen.

Wat zijn API's?

- API = Application Programming Interface
- Een interface voor systemen
- Communicatielaag die interactie tussen verschillende systemen mogelijk maakt
- Systemen moeten enkel de API kunnen "gebruiken" om te kunnen samenwerken
- Verschillende types: web api's, hardware API's
- Werken meestal op dezelfde manier: een data-request wordt beantwoord door een data-response
- Kan interactie mogelijk maken tussen gescheiden systemen, maar binnen één en dezelfde toepassing kan de datalaag ook gelinkt zijn met de presentatielaag via een API

Soorten API's

- web API's bestaan al langere tijd, sinds ongeveer eind jaren 1990
- Twee grote design modellen: SOAP vs REST
- SOAP (Simple Object Access Protocol): meer gericht op de enterprise wereld, contract-gebaseerd gebruik, *actions* staan centraal
- REST (Representational State Transfer): dé manier bij uitstek waarmee publieke API's ontsloten zijn en de meeste data via het web kan worden opgevraagd. Veel dichter bij HTTP specs dan SOAP. Wij concentreren ons voor nu op REST
- GraphQL: zeer flexibele querytaal voor API's waarbij de client zeer fijnmazig de relevante data kan opvragen

De requests library

API's benaderen in Python kan relatief eenvoudig met behulp van de requests library. Volstaat voor alle acties noodzakelijk om API's te gebruiken. In de voorbeelden gebruik ik swapi.dev een API met informatie over Star Wars

```
import requests
antwoord = requests.get("https://swapi.dev/api/people/1")
print(antwoord)  # Response [200]
print(antwoord.text)
```

Endpoints en resources

- Elke API vertrekt van een base URL, hier https://swapi.dev/api. Vaak vind je er het woord API in terug. Andere voorbeelden: https://api.twitter.com, https://api.github.com
- Om gericht data (resources) te kunnen opvragen werken we met endpoints: ze vormen het gedeelte van de URL waarin je meegeeft welke data je wil terugkrijgen. Hoe vind je ze? Via de API reference als die er is
- Een goeie service om endpoints te testen is https://www.postman.com/. Ik raad jullie aan om een gratis account aan te maken. De komende maanden zullen we er goed gebruik van kunnen maken

Vraag en antwoord: request en response

Alle interacties tussen client en API zijn opgesplitst tussen een request en een response

- Request: bevatten alle relevante data over de API call: de base URL, het endpoint, de gebruikte method, de headers, ...
- Response: data teruggestuurd door de server zoals de content, de status code, de headers, ...

Het voorbeeld op de volgende pagina demonstreert de belangrijkste beschikbare attributen voor de request/response objecten.

Gedetailleerde informatie over HTTP messages vind je in de MDN documentatie

Request en response: een voorbeeld

```
import requests
headers = {"X-Request-Test-id": "<mijn-request-test-id>"}
antwoord = requests.get("https://swapi.dev/api/planets", headers=headers)
print(antwoord)
print(antwoord.request)
print(antwoord.request.url)
print(antwoord.request.path_url)
print(antwoord.request.method)
print(antwoord.request.headers)
print(antwoord.text)
print(antwoord.status_code)
print(antwoord.reason)
print(antwoord.headers)
```

Status codes

- Elke API response komt met een status code. Deze code vertelt je of je request succesvol was of (waarom) niet. Dit zijn de meest voorkomende status codes:
  - 200 OK: request was succesvol
  - 201 Created: request geaccepteerd en resources aangemaakt (nuttig voor vervolg)
  - 400 Bad Request: request fout of informatie ontbreekt
  - 401 Unauthorized: permissies ontbreken
  - 404 Not Found: resource bestaat niet
  - 405 Method Not Allowed: gebruikte method niet toegestaan
  - 500 Internal Server Error: fout gebeurd langs de kant van de server

Status codes

- Gedetailleerde informatie over status codes vind je in de MDN documentatie

HTTP Headers

- Worden gebruikt om een aantal parameters voor verzoeken en antwoorden te definiëren:
  - Accept: type content die de client accepteert
  - Content-Type: type content waarmee de server zal antwoorden
  - User-Agent: software gebruikt door de client om te communiceren met de server
  - Server: software gebruikt door de server om te communiceren met de client
  - Authentication: wie gebruikt de API en met welke credentials
- Gedetailleerde informatie over alle types headers vind je in de MDN documentatie

Custom headers

- Het gebeurt dat je bij het aanroepen van API's verwacht wordt custom headers te gebruiken
- Hiermee kan bijkomende extra informatie worden meegestuurd vanuit de client die dan door de server kan worden geïnterpreteerd en gebruikt
- Gebruikelijk: voorafgegaan door "X-". Je kan een dictionary gebruiken om custom headers toe te voegen

```
import requests
headers = {"X-Request-Test-id": "<mijn-request-test-id>"}
antwoord = requests.get("https://swapi.dev/api/planets", headers=headers)
```

Content types: JSON domineert

- Het overgrote merendeel van de API's gebruiken vandaag JSON (application/json) als default content type
- Hier en daar kom je andere content types tegen: afbeeldingen, video, XML,...
- Hier vind je een oplijsting van zowat alle mogelijke content types

```
import requests
antwoord = requests.get("https://via.placeholder.com/150")
print(antwoord)
print(antwoord.headers.get("Content-Type"))
if (antwoord.headers.get("Content-Type") == "image/png"):
    bestand = open("afb.png", "wb")
    bestand.write(antwoord.content)
    bestand.close()
```

Response content

- De requests library biedt verschillende Response attributen waarmee je de response data kunt aanpassen:
  - .text: response content in unicode. Zie voorgaande voorbeeld.
  - .content: response content in bytes. Bij voorkeur te gebruiken voor bepaalde types niet-textuele data zoals bvb afbeeldingen
  - .json(): specifieke method voor json-responses. Zet antwoord automatisch om in een Python data-structuur

Response content: voorbeeld

```
import requests
antwoord = requests.get("https://swapi.dev/api/starships/")
print(antwoord)  # Response [200]
print(antwoord.text)
print(antwoord.content)
print(antwoord.json())
print(antwoord.json()["results"])
for starship in antwoord.json()["results"]:
    print(f'{starship["name"]} (crew: {starship["crew"]})')
```

HTTP Methods

- Een API kan worden aangeroepen met verschillende methods
- Deze methods, ook REST *verbs* genoemd bepalen welke actie zal worden uitgevoerd:
  - POST: requests.post() - nieuwe resource aanmaken
  - GET: requests.get() - bestaande resource opvragen
  - PUT: requests.put() - bestaande resource updaten
  - DELETE: requests.delete() - bestaande resource verwijderen
- We spreken hier vaak over CRUD-handelingen (create, read, update, delete). We maken deze week exclusief gebruik van read (get). De komende weken zullen we de andere ook gebruiken
- Gedetailleerde informatie over alle HTTP methods vind je in de MDN documentatie

Query Parameters

- Met query parameters kan je (indien de API dit ondersteunt) je datazoektocht meer precies maken
- Bij goed ontworpen API's kan de URL-structuur in principe query parameters vervangen: documentatie maakt dit duidelijk
- voorafgegaan door ? voor de eerste query parameter
- Meerdere parameters worden gescheiden door een ampersand (&)

Query parameters

```
# https://metmuseum.github.io/
import requests
antwoord = requests.get("https://collectionapi.metmuseum.org/public/" \
    "collection/v1/search?hasImages=true&q=Peter Paul Rubens")
print(antwoord.json())
query_params = {"hasImages": "true", "q": "Peter Paul Rubens"}
endpoint = "https://collectionapi.metmuseum.org/public/collection/v1/search"
antwoord2 = requests.get(endpoint, params=query_params).json()
print(antwoord2)
```

Authenticatie

- Sommige API's vereisen geen authenticatie en zijn volledig publiek
- Voor anderen heb je authenticatie nodig
- Hier vind je een goed overzicht: https://github.com/public-apis. Bvb de GitHub en Twitter API vereisen authentificatie
- Je hebt basisauthenticatie met een API key of meer geavanceerde zoals OAuth

Authenticatie met een API Key

- Soms dien je een API key aan te vragen bij de service om jezelf bij gebruik van de API te kunnen identificeren en je gebruik te kunnen monitoren
- API keys worden dan meegegeven aan de request als een request header of via een query parameter

Authenticatie met een API Key: voorbeeld

```
import requests
endpoint = "https://api.nasa.gov/mars-photos/api/v1/rovers/curiosity/photos"
api_key = "API Key komt hier"
query_params = {"api_key": api_key, "earth_date": "2020-07-01"}
response = requests.get(endpoint, params=query_params)
print(response)
photos = response.json()["photos"]
print(f"Found {len(photos)} photos")
print(photos[4]["img_src"])
```

Open Authentication of OAuth

- Vaak gebruikte vorm van authenticatie, veiliger dan een API-key
- Je dient een applicatie aan te maken die een ID zal hebben (app_id of client_id) en een secret (app_secret of client_secret)
- Je hebt een redirect URL nodig die door de API zal gebruikt worden om je informatie toe te sturen (redirect_uri)
- Je krijgt een code terug als resultaat van de authenticatie die je dient in te ruilen voor een access token

Paginering

- Bij grote(re) datasets gaan API's soms gebruik maken van paginering. Je kan dit ook zien in de lesvoorbeelden die swapi.dev gebruiken
- De data wordt hierbij opgesplits in kleinere onderdelen
- itereren naar de volgende (of vorige) pagina gebeurt meestal door gebruik van een query parameter (gebruikelijk page of size)

Snelheidsbeperking

- Om misbruik tegen te gaan gaan API's vaak beperken hoe vaak ze kunnen aangesproken worden binnen een bepaalde tijd (rate limiting)
- Bij misbruik kan de API key geblokkeerd worden

Labo 1

Morgen in Labo 1

In het labo morgen zien we:

- Hoe je een Python script kunt deployen naar een cloud service
- Hoe je een eenvoudige Flask app creëert
- Je content haalt uit een Headless CMS. Het concept wordt ook hier uitgelegd

Labo-voorbereidende oefeningen

Oefening 1

Je zoekt in de lijst van publieke API's en probeert er eentje (of meerdere) uit: https://github.com/public-apis. Kijk ook eens in Vlaanderen. Experimenteer ermee door data binnen te trekken en onderzoek wat de mogelijkheden zijn.

Oefening 2

Onderzoek Storyblok en de Storyblok API op de mogelijkheden als Headless CMS. Maak een community account aan, creëer wat content en probeer ze via de API te ontsluiten. Heb je een beter alternatief? Onderzoek dat want dat kan ook.