Handleiding PDF-API

Peter Vantomme

Inhoud

Installatie

Alle bestanden die nodig zijn, zijn normaalgezien te vinden in de git repo (https://github.com/PeterVantomme/pdf_processor). Eenmaal gecloned kan de docker-compose command worden uitgevoerd in de directory van de docker-compose file. Deze zal een container opstarten met alle benodigde installaties alsook een Nginx server die werkt als reverse-proxy.

In de repo is er ook een Insomnia environment terug te vinden, deze maakt het makkelijk om de API uit te testen omdat hier alle mogelijke requests staan opgelijst. Deze kan je als volgt installeren in Insomnia:

1. Ga naar Dashboard-screen
2. Klik op de Create-knop aan de rechterkant
3. Selecteer Git Clone
4. Vul de hierboven vermelde repo in
5. Klik Import
6. De omgeving zou aangemaakt moeten zijn met alle nodige environment variables

Gebruik

Zie Insomnia omgeving en github readme.

Nieuwe extractors toevoegen

Als het nodig is om een extractor toe te voegen, zijn de .ipynb-files handig. Dit zijn Jupyter Notebooks die de prototypes bevatten van alle extractors. Ze zijn een goede bron om uit te starten omdat ze alle nodige code bevatten om een nieuwe extractor aan te maken en je stap voor stap kan zien hoe de code de tekst zal verwerken.

Het finetunen van de extractor om de juiste informatie op de juiste manier te extraheren, is hoofdzakelijk trial and error. De beste toenadering bij het probleem is om eerst alle rijen per type te gaan samennemen en hier dan transformaties op uitvoeren en niet omgekeerd. (~ETL principe)

Het beste is om eerst aan de hand van de Camelot library het bestand uit te lezen en de correcte tabel-zone (of zone waar je informatie uit wilt) te selecteren door de values in “table-area” aan te passen. Deze komen overeen met de coördinaten van x1y1 en x2y2 punten (x1y1 is linksboven, x2y2 is rechtsonder). Met de camelot.plot() methode is het mogelijk om hier een visuele representatie van te zien.

Hierna is het dan nodig om de juiste rijen uit het DataFrame (tabel) te selecteren met behulp van pandas.loc en pandas.iloc methodes. Vervolgens kan je dan met re.search() of pandas.Series.str.contains() gaan zoeken in de rijen naar de nodige waarden om deze ten slotte in een JSON dictionary op te bergen.

De Akte Extractor werkt helemaal anders, voornamelijk omdat hier de informatie in annotaties en de tekst zelf zit waardoor we dus geen tabulaire data hebben. In dit geval gebruiken we de PyMuPDF library om het PDF-document in te lezen en de annotaties eruit te halen. Als dit niet werkt gaan we de selecteerbare tekst van het document doorzoeken en ten slotte maken we gebruik van de tesseract-ocr library om OCR toe te passen om niet-selecteerbare tekst te doorzoeken. Documenten waarin de belangrijke gegevens geschreven zijn (met potlood/balpen) zullen niet per se correct uitgelezen worden en hebben een controle nodig, voornamelijk omdat het voor standaard OCR niet makkelijk is om handschrift te lezen wanneer deze hier niet op is getraind.

QR-reader aanpassen

De QR-reader bestaat uit verschillende onderdelen. Als documenten niet meer correct kunnen ingelezen worden is het belangrijk eerst de pyzbar (zie QR-Interpreter module) implementatie te controleren. Vervolgens kan het nodig zijn om de opencv (zie QR-Transformer momdule) implementatie te bekijken, deze staat ook als prototype in een Jupyter notebook en kan een handige bron zijn om te onderzoeken welk effect de transformaties hebben.

Toekomst

Uitbreidingen toevoegen is een evident gegeven:

1. Maak een gepaste View aan voor de nieuwe functionaliteit (zie Django-REST documentatie of baseer je op de bestaande klassen)
2. Maak een helper aan als het een volledig nieuwe functionaliteit is, ander is het nodig om de huidige helpers aan te passen zodat ze de nieuwe functie kunnen aanroepen, deze helper koppel je ook aan de view (dit gebeurt in de view zelf door de helper te importeren en aan te roepen).
3. Maak een nieuwe module aan voor je functie of voeg deze toe aan een bestaande module, zorg ervoor dat de klasse/methode een JSON of file teruggeeft. Deze module zal dan de verwerkte bestanden doorgeven aan de helper die dan alles doorgeeft aan de view.
4. Koppel view aan URL door deze toe te voegen aan de urls.py module. Urls.py is verantwoordelijk voor de routing, views bouwen hier verder op. Aan de hand van decorators (@action[“detail”=True/False, “methods”=”GET”/”POST”/”PUT”/”PATCH”/”DELETE”]).

   Detail verwijst eigenlijk naar een eventuele parameter in de URI.
   Als de URI bijvoorbeeld https://api/gegevens is, dan moet details=False omdat hier geen parameters zijn meegegeven. Als de URI https://api/gegevens/123903 zou zijn, kunnen we zien dat 123903 een id is en dus een parameter is van de URI. In dat geval moet detail=True.

Extra hulp

Op mijn e-mail adres peter.vantomme3@gmail.com kan je een mail sturen als er verder hulp nodig is.

Architectuur