Project Documentation: PDF Data Extraction and Consolidation Web Application Prepared for Nissifulfillment

Saad Hanif Taj

May 29, 2025

Prepared by Saad Hanif Taj for Nissifulfillment

Contents

1 Overview

This document, prepared by Saad Hanif Taj for Nissifulfillment, details a custom web-based solution developed to extract and consolidate structured product data from PDF files. The core deliverable, the FastAPI-based web application (`app.py`), leverages a tailored Azure Document Intelligence model, `customscan`, to process PDFs containing product details such as BARCODE, ARTICLE, PRICE, QUANTITY, and Carton ID. The extracted data is compiled into a single, sorted Excel file, accessible via an intuitive web interface hosted on the Azure Web App `shtWebApp`. The solution includes a local testing script for validation, the main application script, and deployment instructions for Azure App Service, delivering a robust tool for Nissifulfillment.

2 Azure Document Intelligence Custom Model: `customscan`

The `customscan` model is a bespoke Azure Document Intelligence model, designed for Nissifulfillment's PDF layout to extract structured data with high accuracy. Developed using Azure Document Intelligence Studio, it targets key fields: Carton ID and table-based data (BARCODE, ARTICLE, PRICE, QUANTITY).

2.1 Model Training and Layout Customization

To meet Nissifulfillment's requirements, the `customscan` model was trained on client-provided sample PDFs. These PDFs were annotated to define the layout, including:

- **Carton ID**: A unique identifier located in a consistent header or footer section.
- **Table Data**: Structured tables with columns for BARCODE, ARTICLE, PRICE, and QUANTITY, arranged in a predictable format.

The training process involved labeling these fields across multiple documents to ensure accurate recognition, accommodating minor variations like font changes or positional shifts. This customization ensures precise data extraction tailored to Nissifulfillment's PDFs.

2.2 Carton ID Mapping

The Carton ID links each PDF's table data to its source document, critical for data consolidation. The mapping process, implemented for Nissifulfillment, operates as follows:

1. The `customscan` model identifies the Carton ID based on its trained understanding of the PDF layout.
2. Table data is extracted row by row, with each row (BARCODE, ARTICLE, PRICE, QUANTITY) associated with the document's Carton ID.

3. The extracted data is stored in a structured format, ensuring all rows are tagged with their Carton ID.

This mapping enables the Excel output to be sorted by Carton ID, providing Nissifulfillment with an organized view of product data.

2.3 Requirements

To utilize the `customscan` model, Nissifulfillment must ensure:

- An Azure Document Intelligence resource is provisioned.
- The `customscan` model is trained and deployed via Azure Document Intelligence Studio.
- The Azure endpoint, API key, and model ID (`customscan`) are accessible.

3 Local Processing Script (`local.py`)

The `local.py` script, developed for Nissifulfillment, supports validation of the `customscan` model and extraction logic before deployment to `shtWebApp`.

3.1 Features

- Reads PDFs from a local directory.
- Connects to Azure Document Intelligence using `customscan`.
- Extracts Carton ID and table data (BARCODE, ARTICLE, PRICE, QUANTITY).
- Consolidates and sorts data into `consolidatedfile.xlsx`.

3.2 Local Test Instructions

Nissifulfillment can run the script with:

```
python local.py
```

A `.env` file must include:

```
FORM_RECOGNIZER_ENDPOINT="YOUR_AZURE_ENDPOINT"
FORM_RECOGNIZER_KEY="YOUR_AZURE_KEY"
```

4 Web Application (`app.py`)

The main deliverable, `app.py`, is a FastAPI-based web application deployed on `shtWebApp`, enabling Nissifulfillment users to upload PDFs via a browser and download a consolidated Excel file.

4.1 Features

- Supports simultaneous upload of multiple PDFs.
- Handles temporary file storage securely during processing.
- Employs the `customscan` model for data extraction.
- Delivers a downloadable Excel file with sorted data.

4.2 Endpoints

- `GET /`: Displays the HTML upload form.
- `POST /process_pdfs/`: Processes uploaded PDFs and returns the Excel file.

5 Web Interface (`templates/index.html`)

The web interface, hosted on `shtWebApp`, provides a seamless experience for Nissifulfillment users.

5.1 Features

- Drag-and-drop file upload functionality.
- Progress and status indicators during processing.
- Automatic download of the consolidated Excel file.

5.2 Workflow

1. Users select PDFs via the web interface on `shtWebApp`.
2. Files are uploaded to the FastAPI server.
3. The server processes PDFs using the `customscan` model.
4. A download link for the Excel file is provided.

6 Dependencies (`requirements.txt`)

The application requires:

```
fastapi
uvicorn
gunicorn
pandas
openpyxl
azure-ai-formrecognizer
python-dotenv
jinja2
```

`python-multipart`

Nissifulfillment should ensure unused packages (e.g., Flask) are excluded.

7 Environment Variables

Sensitive credentials are managed via environment variables:

- $\texttt{FORM}_R ECOGNIZER_E NDPOINT : Azure Document Intelligence endpoint URL. \texttt{FORM}_R ECOGNIZ$

7.1 Local Development

Store variables in a `.env` file.

7.2 Azure Deployment

Set as Application Settings in the Azure Portal for `shtWebApp`.

8 Azure App Service Deployment

The application is deployed to the Azure Web App `shtWebApp` for scalability.

8.1 Deployment Checklist

- Upload: `app.py`, `requirements.txt`, `templates/index.html`.
- Set runtime to Python 3.9+ in the Azure Portal for `shtWebApp`.
- Configure Application Settings for Azure credentials in `shtWebApp`.

8.2 Startup Command

Use in `shtWebApp`:

```
gunicorn -w 4 -k uvicorn.workers.UvicornWorker app:app --bind=0.0.0.0 --t
```

8.3 Deployment Options

- Zip Deploy via Azure CLI to `shtWebApp`.
- GitHub Actions or Azure DevOps for CI/CD.
- Local Git deployment to `shtWebApp`.

9 Workflow Summary

1. Nissifulfillment users access `shtWebApp`.
2. Multiple PDFs are uploaded via the browser.

3. The `customscan` model extracts data.
4. Data is consolidated and sorted by Carton ID.
5. A downloadable Excel file is returned.

10 Troubleshooting Tips

- **Error**: "Form data requires 'python-multipart'" — **Solution**: Add `python-multipart` to `requirements.txt`.
- **Error**: "FastAPI.__call__() missing 1 argument" — **Solution**: Use `uvicorn.workers.Uvi`
- **Error**: "No module named 'uvicorn'" — **Solution**: Add `uvicorn` to `requirements.txt`.

11 Summary

This solution, developed by Saad Hanif Taj for Nissifulfillment, leverages Azure Document Intelligence's `customscan` model to automate data extraction from PDFs tailored to the client's layout. Deployed on the Azure Web App `shtWebApp`, the core deliverable, `app.py`, powers a scalable FastAPI web application. The solution ensures precise extraction of Carton ID and table data, with robust mapping for data consolidation. The deliverables include:

- The main FastAPI application script (`app.py`) for `shtWebApp`.
- A user-friendly web interface for file uploads and downloads.
- Deployment instructions for `shtWebApp`.
- A local testing script (`local.py`) for validation.

This application provides Nissifulfillment with an efficient, reliable tool for processing product data into sorted Excel files, ready for operational use.