API-DRIVEN OCR BANK STATEMENTS

v0.1 2024-05-19

The API application is at a robust and stable point in development and is available for integration now.

Conversion of ABSA, Capitec, FNB, Nedbank and Standard Bank statements are done accurately. Other banks can be added by creating templates. The API converts the PDF and afterwards provides the information via certain API calls. It relies on back-end server functionality exclusively as specified in the design.

FUNCTIONALITY OVERVIEW

The OCR API starts functioning when a PDF bank statement is found in the upload folder of the server and the **'job'** call is fired.

The process will start running through several sub-processes. Some of these are intensive and may result in excessive duration or CPU lag, therefore the main process may be canceled via the **'cancel'** call.

The **'progress'** call may be run in a loop on the frontend to indicate progress.

The API will produce the result, containing the following:

1. Header: Static information from the first page, e.g. customer details.

2. Anomalies: Auto-corrected cell values, invalid dates, amount, decimals, etc.

3. Template details.

4. Transactions in row/column format.

5. Recon: Possible problems that should be investigated, but was also auto-corrected.

API CALLS

After the main OCR process has completed, the information will be available for querying without running the time-consuming OCR again, all the above information will be instantly available via the API:

Examples:

v1/statement/$id Full statement details

v1/statement/header-field/$field/$id Specific field value

v1/statement/grid-row/$page\_row/$id Specific row in grid

It is straight-forward to add more queries to the API.

// TODO: Build API functionality to update corrected information. This will rely on a front-end edit screen, where errors can be manually corrected.

TECHNICAL

1. The application runs on an (Linux) Apache web server with PHP enabled.

2. It relies on command-line functionality and several applications to run, mainly Tesseract for OCR conversion and Poppler for PDF conversion. These applications are CPU-intensive.

//TODO: Ensure open-source license compliancy.

3. API Authorization is handled via HTTP headers and API-keys.

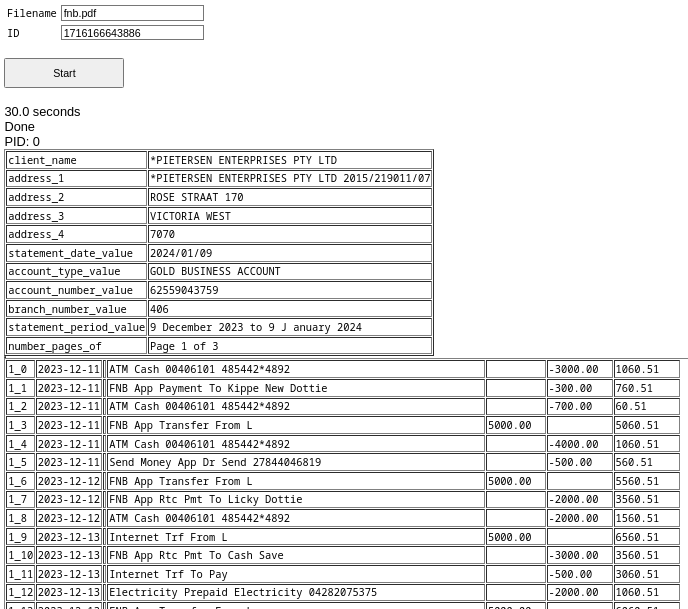
4. Error handling provides traceable error numbers and messages.

FRONT-END

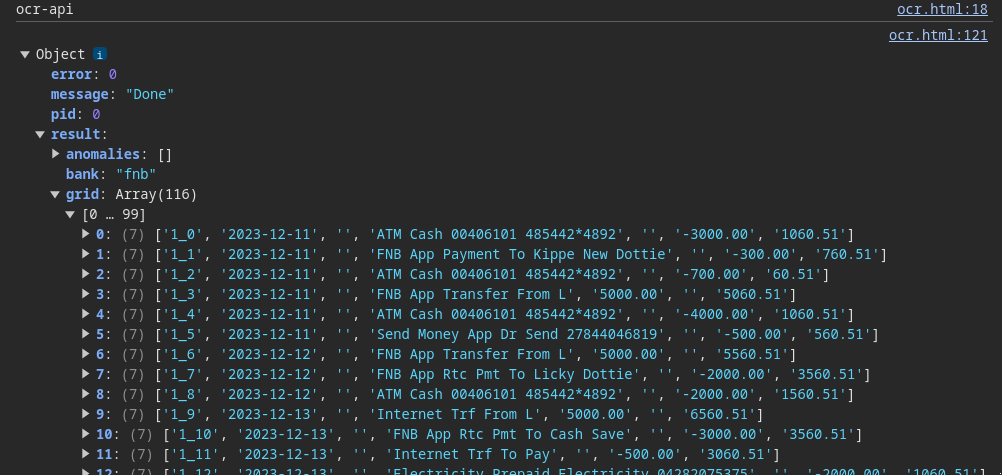
The API-driven approach cannot provide front-end functionality. However, a simple Javascript frontend is provided as an example, which shows how to use modern async **fetch** and **await** methods. Any other technologies beyond Javascript would be easy to implement.

Displayed are example outputs, captured from a browser.

Front-end output:



JSON output:



SUB-PROCESSES

1. Split the PDF pages into separate PNG files.

2. Convert the first page PNG to OCR.

3. Search for various words in specific positions to determine the correct bank template, utilizing this first page OCR.

4. Extract the header information from the first page OCR after finding the template.

5. Split the PNGs of each page into columns and OCR each column separately.

6. Loop through the pages and columns OCR and collect the information per words in specific positions on the page.

7. Sanitize and find anomalies.

8. Recon according to the detected running balance, etc.

9. Return result as a downloadable JSON file or via API call.

.oooOOOooo.