Intelligent Document Extraction & Verification System

\*A full-stack platform to automate data extraction from scanned forms using AI

# 1. Project Objective

The goal of this project is to develop a full-stack intelligent platform that allows users to upload scanned documents (PDFs or images), extract relevant text data using Optical Character Recognition (OCR), structure it into a machine-readable format using AI, and store it in a PostgreSQL database for review, editing, and verification.  
  
This system automates data entry from semi-structured forms (e.g., bank applications, registration forms) and makes the information searchable, editable, and verifiable through a user-friendly interface.

# 2. Functionality Workflow

### File Upload

* Users upload image or PDF documents via the React frontend.
* Supported formats: jpg, png, pdf

### Backend Processing

* Express server receives files via /upload-routes.
* Uses “multer” to store files temporarily in the uploads /directory.
* If the file is a PDF, each page is converted to an image using pdf2pic.

### OCR + AI Structuring

* Each image (from upload or PDF conversion) is sent to Azure Cognitive Services OCR.
* Extracted raw text from all images/pages is concatenated.
* The combined text is sent to the Groq API for structuring into labeled key-value JSON.

### Data Insertion

* The structured JSON is passed to a Python script “insert\_to\_pg.py”.
* The script dynamically creates or updates a PostgreSQL table and inserts the data.

### Frontend Visualization

* The React frontend fetches all data from the backend.
* Nested JSON structures are visualized in a dynamic, editable table.
* Verified documents are moved to a separate "after verification" database (`Permanent\_Database`).

### Optional Editing & Verification

* Users can edit fields inline in the frontend table.
* Verified entries are saved into the “Permanent\_Database” database.

# 3. Algorithm / Logic

## OCR + Structuring Pipeline

if file.type == 'PDF':  
 # Convert each page to image using pdf2pic  
 images = pdf2pic.convert(pdf)  
else:  
 images = [file]  
all\_text = ''  
for image in images:  
 text = azure\_ocr(image)  
 all\_text += text + '\n'  
  
structured\_json = groq\_api(all\_text)

## DB Insert Logic (`insert\_to\_pg.py`)

for key in json\_data:  
 if not column\_exists(table, key):  
 alter\_table\_add\_column(table, key)  
insert\_row(table, json\_data)

## 🖥️ Frontend Table Display Logic

const renderCell = (value) => {  
 if (typeof value === 'object' && value !== null) {  
 return <SubTable data={value} />;  
 } else {  
 return <TextField value={value} />;  
 }  
};

# 4. Input and Output

## Input

Uploaded Document: Image (`.jpg`, `.png`) or Scanned PDF (`.pdf`)  
Via: React frontend upload form

## Output Format: Structured JSON

{  
 "Full Name": "Ramesh Kumar",  
 "Father's Name": "Suresh Kumar",  
 "Date of Birth": "1998-10-12",  
 "Account Type": "Savings",  
 "Phone Number": "9876543210"  
 // ... more fields  
}

## 🧾 Output

• Stored in PostgreSQL under `before\_verify2` (raw) and `after\_verify2` (verified)  
• Displayed in a dynamic table (React) with support for:  
 - Inline editing  
 - Viewing nested JSON fields as sub-tables  
 - (Optional) Exporting data

# 5. Tech Stack

|  |  |
| --- | --- |
| Layer | Technology |
| Frontend | React, Material UI |
| Backend | Node.js (Express), Multer |
| AI & OCR | Azure Cognitive Services, Groq API |
| PDF Tools | pdf2pic |
| DB Layer | PostgreSQL |
| Script Bridge | Python (psycopg2) |

# 6. Summary of Flow

* User uploads a document (image or PDF) via the React frontend.
* Backend receives the file, processes it:

- Converts PDF to images if needed.

- Runs OCR on each image.

- Sends extracted text to Groq for structuring.

- Stores structured data in PostgreSQL.

* Frontend fetches and displays all documents in a table, allowing edits and verification.
* Verified documents are moved to the “Permanent\_Database” database.