XtractIQ – Built to Decode Your Documents

1. Introduction

1.1 Purpose

This High-Level Design (HLD) report is meant to illustrate the structure, main components, and data flow of XtractIQ. XtractIQ is a complete platform utilizing AI to autonomously extract and validate data from scanned documents. This report fills the gap between the requirements to the detailed design stage so that everyone can see and understand the system architecture at a high level.

It sets the overall structure and significant design decisions that will lead the development process. It ensures that all components, technologies, and workflows enable the project goals.

1.2 Scope

This HLD document describes the key components and how they interact with each other in the XtractIQ system. These include:

- General system design and part diagram
- Key modules such as file upload, OCR processing, AI field extraction, and verification
- Backend APIs and front-end interaction
- Basic organization of a database
- Communication with external libraries or services (e.g., AI models and OCR services)
- Overview of deployment and technology tools

This document does not contain implementation decisions, low-level structure of code, or algorithms at module level. Those are to be addressed in the Low-Level Design (LLD) documents.

2. System Overview

2.1 System Description

XtractIQ is a document processing platform based on AI. It is utilized to assist in automatically retrieving and verifying structured information from PDF, images, and scanned forms. Documents can be imported through a web portal.

Optical Character Recognition (OCR) is utilized by the system to extract text. It also employs machine learning models to identify and retrieve crucial information such as names, dates of birth, phone numbers, and more.

After being pulled out, the data is validated and classified manually and saved in a database for access or examination in the future. The system aims to minimize the need for manual data input, minimize errors, and automate document-related processes in the finance, insurance, and onboarding sectors.

Key features are:

- Secure document upload (image, PDF, Word)
- OCR-based text extraction
- Field classification using AI (e.g., Full Name, Date of Birth, etc.)
- Post-processing and verification of the extracted fields
- Structured data presentation and graphical representation
- PostgreSQL database storage for verification and analysis

2.2 System Context

XtractIQ is a stand-alone platform but one that is designed to integrate into broader enterprise workflows and document-based processes. It is a smart middleware layer that cleanses dirty, unstructured documents and transforms them into clean, structured, and verifiable data — ready for storage, display, or further processing.

Technologically, the system is built into a modern web-based cloud system. It utilizes:

- A React.js-based user interface for user interaction and file uploading
- A Node.js (Express) server to process API requests, route data streams, and manage processing
- Azure Cognitive Services (OCR API) for extracting text from uploaded documents
- A PostgreSQL database to store structured and validated document data

Instead of executing in-house OCR engines or Python microservices, XtractIQ leverages cloud-based AI in the form of Azure's OCR API. This is cloud-scalable, has low maintenance overhead, and enjoys best-in-class OCR quality with low infrastructure complexity.

In a business setup, XtractIQ improves operational effectiveness, improves data accuracy, and reduces man-hours of form-processing efforts substantially — which makes it very applicable to sectors such as banking, insurance, HR, and KYC-compliant environments.

3. Architecture Design

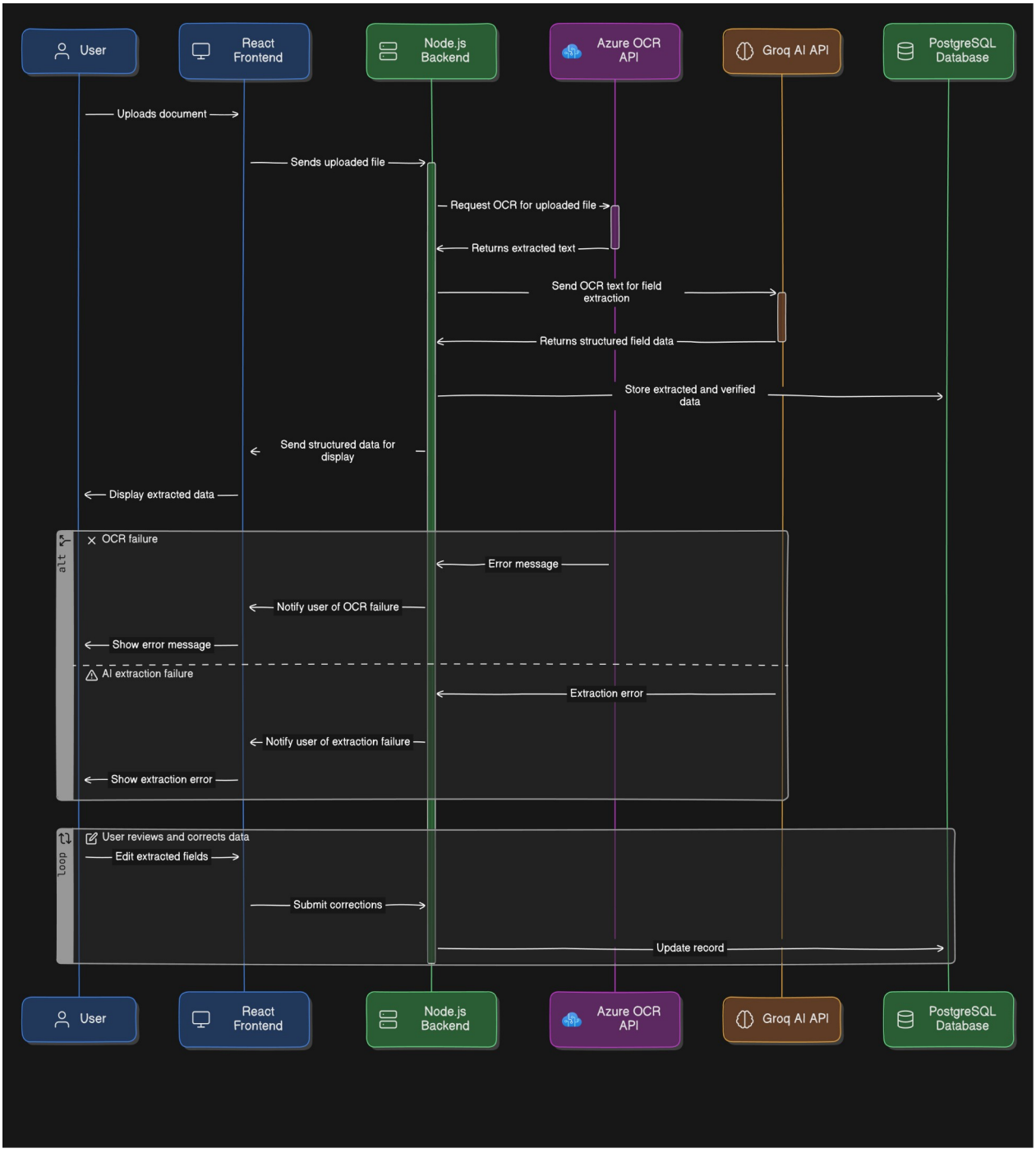
3.1 Architectural Overview

XtractIQ is a cloud-enabled, modular document processing system built on a modern web architecture. It follows a three-tier architecture consisting of:

- Frontend (Client Layer): Built with React.js, the frontend handles file uploads, form previews, and displays extracted data in a user-friendly interface. It communicates with the backend via HTTP APIs.
- Backend (Application Layer): A Node.js (Express) backend processes incoming requests, routes document data to external services (Azure OCR and Groq AI), and manages database interactions. It also handles error logging, file storage paths, and response formatting.
- External Services:
    - Azure OCR API: Extracts raw text from uploaded documents.
    - Groq AI (LLM API): Processes OCR output to extract relevant fields such as name, date of birth, phone number, etc.
- Database (Storage Layer): PostgreSQL is used to store original documents, extracted field data (before and after verification), and metadata related to user activity and system processing.

This design ensures modularity, scalability, and ease of integration with other enterprise systems in the future.

3.2 Component Diagram



4. Module Design

4.1 Module Overview

The XtractIQ platform is structured into the following primary modules, each with a specific aspect of the document extraction and verification pipeline:

1. Frontend (React.js) Responsibility: Handles user interface for uploading files, viewing extracted data, and submitting for verification.
   Features:
   - Drag-and-drop or browse file upload
   - Shows raw and structured extracted data
   - Enables users to edit or verify extracted fields
   - Sends data to the backend through HTTP APIs

2. Backend Server (Node.js + Express) Role: Coordinates the process by managing file uploads, routing requests to OCR/AI services, and handling responses.

   Characteristics:
   - Exposes /api/upload and other URLs
   - Uploads documents to Azure OCR to extract text
   - Uploads OCR result to Groq AI to classify fields
   - Passes structured data to the Python script for inserting into a database
   - Manages response formatting and error validation

3. Processing Module Responsibility: Prepares documents for OCR by converting them to compatible formats and invoking external services.
   Components:
   - pdf2pic: PDFs are converted into image files
   - Azure OCR API: Raw text is extracted from images
   - Groq AI (LLM API): Raw OCR text is processed and key-value pairs in a structured format are returned for pre-defined fields (Full Name, Date of Birth, etc.)

4. Python Bridge Responsibility: It is a light microservice responsible for injecting processed data into the PostgreSQL database.
   - Inserts structured JSON into before_verify or after_verify tables based on status

5. Database (PostgreSQL) Responsibility: Retains unverified and verified extracted data from documents.
   Structure:
   - before_verify: Retains raw extracted fields for user review
   - after_verify: Retains finalized and verified document data

4.2 Module Interaction

Modules in XtractIQ communicate in a pipeline fashion, with explicit roles and communication standards:

- User uploads a file via the React frontend.
- File is posted via /api/upload to the Node.js backend.
- Backend verifies file type and passes it on to the Processing Module
- PDF files are translated into images using pdf2pic
- Images are passed to Azure OCR for text extraction
- The OCR text output is passed to the Groq AI API for field classification.
- The structured data (key-value fields) is passed back to the backend, then to the Python Bridge.
- The Python script puts the unverified data into the before_verify table.
- The frontend presents the structured data to the user for inspection and editing.
- On submit, the backend calls the Python script again to put verified fields into the after_verify table.
- Raw and validated data are both stored in PostgreSQL so that tracking and audit trails are possible.

5. Data Design

5.1 Data Model

Data Model The data model of XtractIQ is based on two principal entities:

- Document
  Represents one uploaded file (PDF, image). Each document is distinct and associated with its extracted data.
  Attributes: filename file_type
- Extracted Field
  Represents a key-value pair that has been extracted from a document by AI (e.g., "Full Name: John Doe").
  There are two logical states for field data:

- Unverified: Raw AI output stored in before verification database
- Verified: User-verified or confirmed fields saved in after verification database
  Attributes: field_label (i.e., Full Name, DOB) field_value

5.2 Data Flow

The CRU flows for data are linear, controlled ones:

- Create
  Document upload initiates OCR + LLM-based field extraction.
  Extracted fields are inserted into the before_verify table through the Python script.

- Read
  Unverified data from before_verify is retrieved and displayed in the frontend.
  Users can see, edit, and verify the extracted fields.

- Update
  After verification by the user, edited fields are sent back to the backend.
  These are added as new records to the after_verify table.

6. Interface Design

6.1 User Interface

The XtractIQ user interface is made simple, clear, and usable. It is constructed with React.js and has a clean, modular structure to enable intuitive navigation and seamless user interaction. At a high level, the UI is segmented into the following major screens:

1. Home / Upload Screen
   Initial point of interaction of the user to upload scanned documents (PDF or images).
   Key Features:
   - Drag-and-drop upload or file picker.
   - Indicators for supported file formats.
   - Progress indicator.
   - Upload Button to initiate extraction and processing.

2. Extracted Data Review Screen
   Shows the extracted key-value data fields produced by Azure OCR and Groq AI.
   Key Features:
   - Table or card-based layout to display extracted fields
   - Labels such as "Full Name," "Date of Birth," etc.
   - Inline editing for user correction
   - "Submit for Verification" button

3. Verification / Final Submission Screen
   Enables users to verify edited data prior to final submission.
   Key Features:
   - Read-only display of all fields after edit
   - Confirmation alert for final submission
   - Submit button invokes data insertion into after_verify