Cognitive Services And OCR Analytics

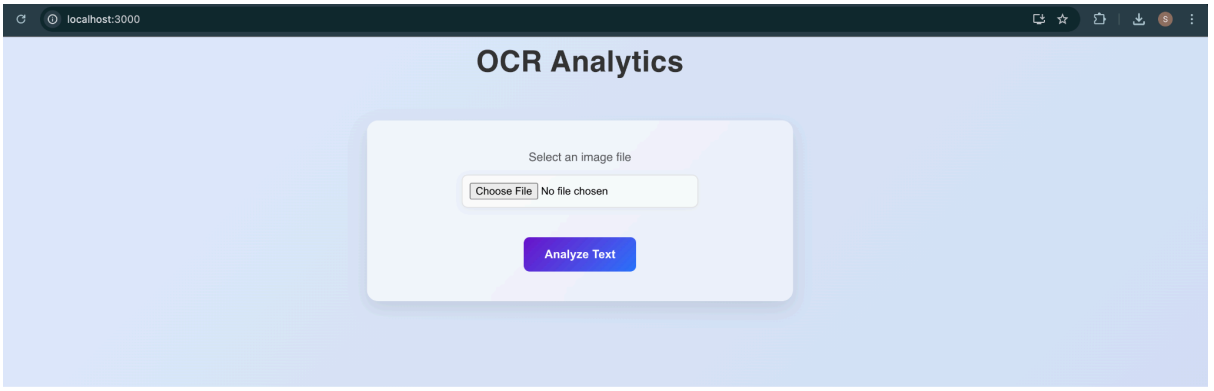
1. Introduction

This report provides a detailed overview of the OCR Analytics Application, a tool designed to allow users to extract and view text from uploaded images using Optical Character Recognition (OCR). The application leverages React for the frontend interface, Node.js and Express for the backend server, and integrates Azure's Computer Vision API to perform OCR. This document covers the core functionality, technical components, and the integration of Azure's API.

2. Core Application Features

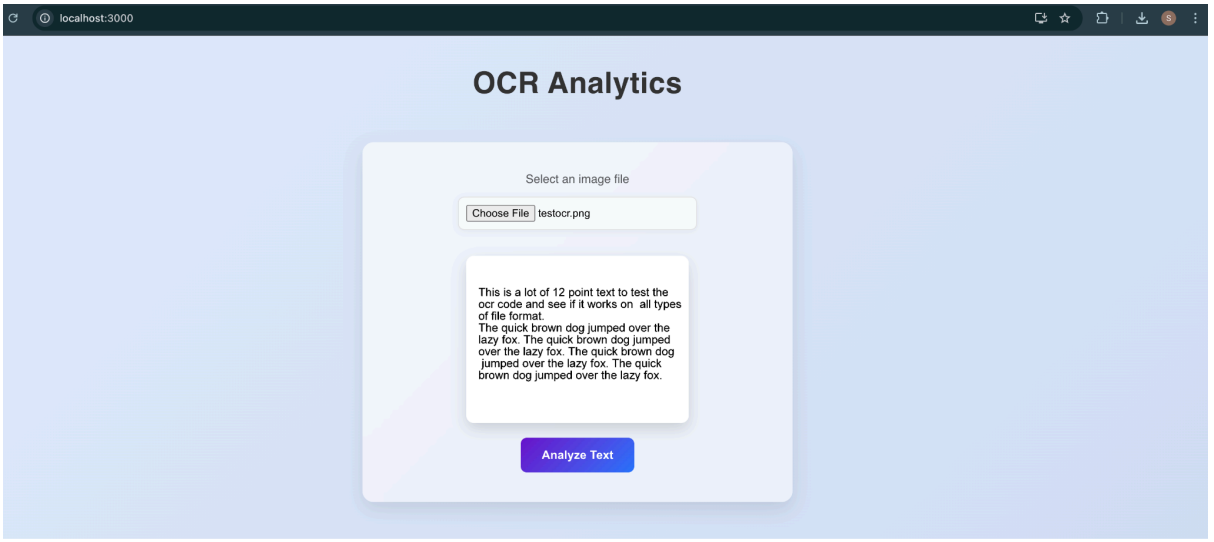
2.1 Image Upload and Preview

- **Description**: Users can select an image file from their device to upload.
- **Components**:
    - **File Input**: Allows the user to choose an image.
    - **Image Preview**: Displays the selected image to confirm before processing.



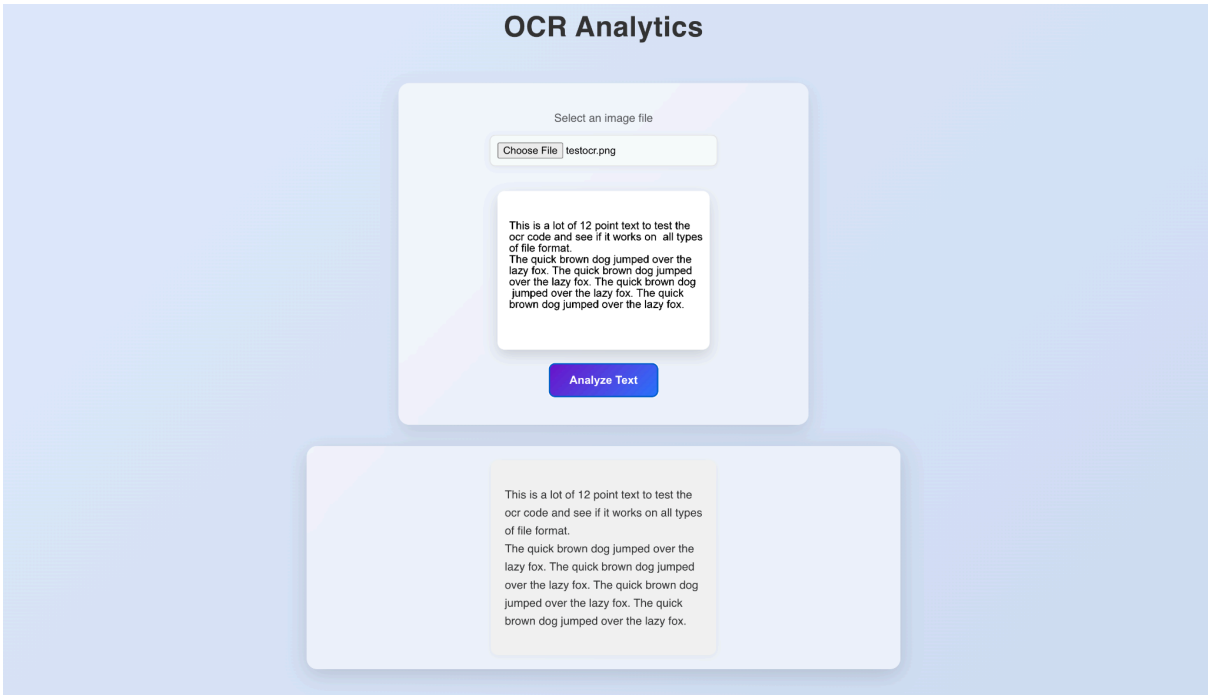
2.2 Text Extraction with OCR

- **Description**: After uploading an image, users can trigger the OCR process. The application sends the image to a backend server, which forwards it to Azure's Computer Vision API.
- **Process**:
    - **Analyze Button**: Triggers OCR processing.
    - **Loading Indicator**: Provides visual feedback to indicate ongoing processing.
- **Result Handling**: Once processed, the backend sends extracted text back to the frontend, where it is displayed in a readable format.



2.3 Display of OCR Results

- **Description**: The extracted text is displayed in a formatted view for easy reading.
- **Components**:
    - **Text Output**: Shows OCR results with appropriate line breaks for readability.
- **Error Handling**: Displays an error message if the OCR process encounters any issues.



3. System Components

3.1 Frontend Components (React)

- **Image Upload**: Handles file input, image preview, and preparation for processing.
- **Analyze Button**: Provides an intuitive control to start OCR processing, ensuring that an image is selected.
- **Loading Spinner**: Indicates the status of the OCR process.
- **Text Display**: Formats and displays the extracted text for the user.
- **Styled Components**: Provides consistent styling across the application, creating a clean, modern interface.

3.2 Backend Components (Node.js and Express)

- **File Upload Handler**: Manages image file uploads using `multer`.
- **API Communication**: Handles secure communication with Azure's Computer Vision API.
- **Text Extraction and Formatting**: Retrieves and formats text data from Azure's OCR response, preparing it for display on the frontend.

4. Azure API Integration

4.1 Overview of Azure Computer Vision API

- **Functionality**: Azure's OCR API processes images to identify and extract text, returning structured text data with line-by-line segmentation.
- **Endpoint Used**: `/vision/v3.2/read/analyze`, which supports various image formats and languages.

4.2 Backend Integration Steps

- **Step 1: Initial API Call**:
  - The backend sends the uploaded image to Azure's endpoint using an HTTP POST request. The request header includes the `Content-Type` set to `application/octet-stream` and the subscription key for authentication.
  - Azure responds with a `operation-location` URL, which is used to check the status of the OCR process.
- **Step 2: Polling for Results**:
  - The backend polls the `operation-location` URL every few seconds to check if the OCR process is complete.
  - Once the status is `succeeded`, the backend retrieves the result. If the OCR process fails or times out, an error is returned to the frontend.
- **Step 3: Text Extraction and Formatting**:
  - The backend extracts the recognized text from Azure's response, joining lines and pages for readability.
  - The text is sent to the frontend in a JSON response.

4.3 Security Measures

- The API subscription key and endpoint are stored securely in environment variables on the backend, ensuring sensitive data is not exposed in the frontend.
- Only the backend communicates directly with Azure, minimising the risk of unauthorised access.