Passport Data Extraction Web Application – Documentation

Table of Contents

1. Overview

This project is a web application designed for extracting key information from passport images. The app uses a Flask backend for data processing and a React frontend for user interaction. Users can upload passport images, and the system extracts fields such as name, passport number, and expiration date using Optical Character Recognition (OCR).

2. System Analysis and Requirements

A. Objectives

The primary goal of this project is to automate data extraction from passport images using OCR. The application reduces manual data entry by letting users upload a passport image and retrieve information with a click.

B. Functional Requirements

- **Image Upload**: Interface for users to upload passport images.
- **Data Extraction**: Extraction of essential information, such as name, passport_number, and expiration_date.
- **Result Display**: Display the extracted data to the user.

C. Non-Functional Requirements

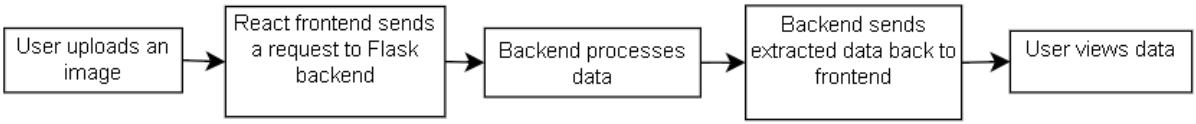
- **Accuracy**: High precision in extracting and parsing information.

- **Security**: Handle images securely without storing them permanently.
- **User-Friendly Interface**: Provide a simple, intuitive UI.

D. User Flow Diagram

- **Description**
  The User Flow Diagram shows how users interact with the system, from uploading an image to viewing extracted data.
- **Elements:**
  - **User**: The starting point, representing the user.
  - **Frontend (React)**: Interacts with the user, displaying the UI and sending image data to the backend.
  - **Backend (Flask)**: Processes the image for data extraction.
  - **Output**: Returns extracted data to the frontend for display.

E. Diagram:



3. System Design

A. Architecture Overview

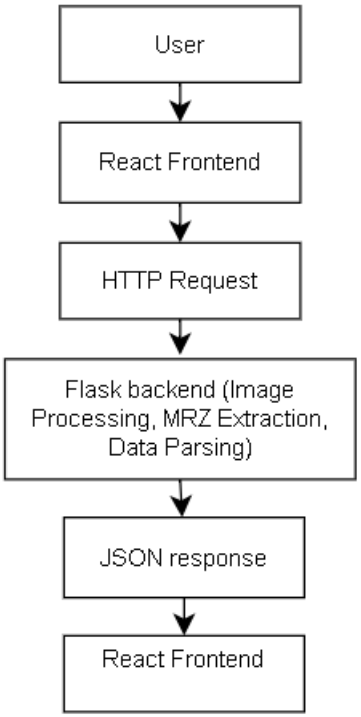
The system follows a client-server architecture:

- **Frontend**: Built with React for a responsive UI.
- **Backend**: Developed in Flask to process image data with OCR libraries.

B. Data Flow Diagram

- **Description**
  The Data Flow Diagram provides an overview of data movement across the system, illustrating how the image and data are handled.
- **Elements:**
  - **User**: Initiates the process by uploading a passport image.
  - **React Frontend**: Sends the image to the Flask backend.
  - **Flask Backend**: Processes the image with MRZ extraction and OCR.
  - **Return Data**: JSON response with extracted fields is returned to the frontend.
  - **Display on Frontend**: The extracted data is shown to the user.

- **Diagram:**


User
React Frontend
HTTP Request
Flask backend (Image Processing, MRZ Extraction, Data Parsing)
JSON response
React Frontend

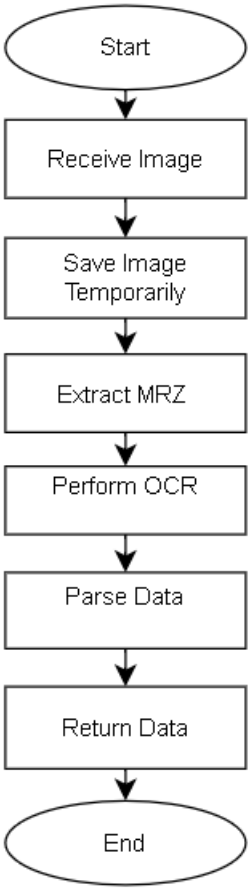
C. Data Processing Flow

- **Description**

  The Data Processing Flow Diagram details the steps in the backend to process and extract MRZ data from a passport image.

- **Elements:**
  - **Receive Image**: The backend receives and temporarily stores the uploaded image.
  - **Extract MRZ Region**: MRZ area is isolated for text extraction.
  - **OCR Processing**: OCR is applied to detect and read text from the MRZ.
  - **Data Parsing**: Fields (name, passport number, expiration date) are parsed.
  - **Return Data**: A JSON response containing extracted fields is sent to the frontend.

- **Diagram:**


Start
Receive Image
Save Image Temporarily
Extract MRZ
Perform OCR
Parse Data
Return Data
End

D. System Design Diagram

- **Description**

The System Design Diagram provides a comprehensive look at the architecture, showing the interaction between components.

- **Elements:**
  - **User Interface (React)**: Accepts input from the user and displays extracted data.
  - **Image Upload & Data Display**: Visual representation of these components in the UI.
  - **Flask API**: Handles requests from the frontend and interacts with OCR libraries.
  - **OCR & Image Processing**: Modules like EasyOCR and PassportEye involved in text extraction.
  - **Data Output**: JSON data returned to the frontend.

- **Diagram:**

4. Code Explanation

The backend uses Flask, and the frontend is built in React. Here's a detailed look at key functions.

A. Flask App Structure

- **app.py:** This file contains the main application.
    - **upload_file():** Handles POST requests for image uploads and data extraction.
    - **get_data_from_mrz():** Processes the image, extracts MRZ data, and parses fields.

B. Key Functions

- **upload_file()**
    - **Description:** Manages image uploads, calls get_data_from_mrz() for processing, and returns JSON data.
- **get_data_from_mrz(img_path)**
    - **Description:** Extracts text from the MRZ using PassportEye and EasyOCR, then parses fields (name, passport_number, expiration_date).

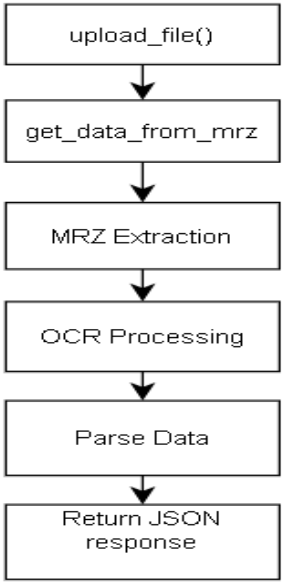
C. Function Flow Diagram

- **Description**

The Function Flow Diagram illustrates the sequence of function calls and actions in the backend, from receiving an image to returning extracted data.

- **Elements to Include:**
  - **upload_file():** Entry point for image upload.
  - **get_data_from_mrz():** Processes the MRZ data.
  - **Data Extraction:** Steps to extract and parse fields.
  - **Return Response:** JSON response with parsed data.

- **Diagram:**



5. Reflection

A. Reflection on the Process

This project successfully automates MRZ data extraction from passports, offering a smooth user experience. Key challenges included handling low-quality images and ensuring accurate MRZ parsing. Adopting a modular code structure improved code readability and debugging.