DOCUMENT CAPTURE PROBLEM STATEMENT FOR GOVERNMENT- ISSUED ID(PASSPORT)

OBJECTIVE :

1. To accurately extract and record essential information from a government-issued passport.
2. The data will be used for identification verification or record-keeping.
3. This involves capturing specific fields that uniquely identify an individual as listed on their passport.

SCOPE:

1.This document capture task focuses on extracting key details from a sample passport.

2.For the purpose of demonstration,we'll use publicly available sample data.

DETAILS:

First Name	: Sirigireddy
Middle Name	: Sidda
Last Name	: Reddy
PASSPORT NUMBER	: A12410037
NATIONALITY	: Indian
DATE OF BIRTH	: 13 November 1985
PLACE OF BIRTH	: Kadapa dist, Andhra Pradesh
GENDER	: M(Male)
DATE OF ISSUE	: 01 July 2020
DATE OF EXPIRATION	: 01 July 2030
ISSUING AUTHORITY	: A.P. Department of State
PASSPORT TYPE	: P(Standard passport)
Additional Information	: Personal ID Number : 987654321(if present)

SYSTEM DESIGN

To Capture , process , store , and verify document data , I propose the following system design:
`   1. Document Upload: A frontend interface where the user uploads a scanned image or takes a picture of the document.

2.OCR (Optical Character Recognition) Processing: A backend service that processes the image and extracts text using an OCR library such as Tesseract.

1. Data Validation: After extraction, the data should be validated to ensure it follows a specific format (e.g., expiration date is not in the past).
2. Storage: Store the extracted information in a database (SQL or NoSQL).
3. Verification: Include an option to verify extracted data against a government database or perform manual verification.
4. User Interface (UI): A simple web-based interface for the user to interact with the system, view captured data, and validate it.

For a high-level diagram, you can depict:

1. Frontend: ReactJS UI that sends the document image to the backend.
2. Backend: Node.js or Python Flask/Django server that processes the image, extracts the data, validates it, and stores it.
3. OCR Engine: Tesseract or a cloud OCR API to handle image-to-text conversion.
4. Database: Use MySQL, PostgreSQL, or MongoDB to store extracted data.
5. Verification/Approval: An optional module for validation by either a third-party API or manual review.

Here's a flowchart that outlines the high-level system design for capturing, processing, storing, and verifying document data. This flowchart visualises the different steps involved in the document capture process.

Flowchart Diagram

```
+------------------+
|  User Uploads    |
|  Document        |
+------------------+
        |
        v
+------------------+
| Backend Receives |
| Document         |
+------------------+
        |
        v
+----------------------+
|  OCR Processing      |
|  (Tesseract.js/Pytess)|
+----------------------+
        |
        v
+--------------------------+
| Extract Text from Image  |
| (OCR Result)             |
+--------------------------+
        |
        v
+----------------------------+
| Text Parsing & Data        |
| Extraction (Regex)         |
+----------------------------+
        |
        v
+--------------------------+
| Basic Data Validation    |
| (Expiry Date Check, etc.)|
+--------------------------+
        |
        v
+-----------------------------+
| Store Data in Database      |
| (MongoDB/PostgreSQL)        |
+-----------------------------+
        |
        v
+---------------------------+
| Return Data to Frontend   |
| (JSON Response)           |
+---------------------------+
        |
        v
+------------------------------+
| User Review/Verification (Optional) |
+------------------------------+
        |
        v
+----------------------------+
|  Completed Process         |
+----------------------------+
```

Explanation of Steps:

1. User Uploads Document: The user initiates the process by uploading a scanned image or taking a photo of the document.

2.Backend Receives Document: The backend receives the image and prepares it for OCR processing.
3.OCR Processing: The backend uses Tesseract.js (or Pytesseract) to process the document and extract raw text.
4.Extract Text from Image: The raw text is generated through OCR, which may include noise or incorrect recognition.
5.Text Parsing & Data Extraction: The backend parses the raw OCR text to extract fields like name, document number, expiration date, and other relevant details.
6.Basic Data Validation: A validation step ensures the extracted data makes sense (e.g., no empty fields, expiration date is not in the past).
7.Store Data in Database: Once validated, the extracted data is stored securely in the database for future access.
8.Return Data to Frontend: The validated data is sent back to the frontend as a response, typically in JSON format, so it can be displayed to the user.
9.User Review/Verification (Optional): Depending on the use case, the user may be asked to verify or correct the extracted data.

Completed Process: The process is complete, and the data is ready for use.

This flowchart provides a high-level view of the process and interactions between the different components in the system. It helps visualize the sequence of steps from document upload to data extraction, validation, and storage.

CODING PROTOTYPE

For the coding prototype, we will implement the following:

Frontend: A simple ReactJS application to upload a document, process it, and display the results.
Backend: A Node.js/Express server that processes the uploaded image using Tesseract.js for OCR.
Frontend: ReactJS
First, you need to install the necessary dependencies:

```
npx create-react-app
document-capture
cd document-capture
npm install axios
```

App.js: The frontend interface for uploading the document.

```
import React, { useState } from 'react';
import axios from 'axios';

function App() {
  const [file, setFile] = useState(null);
  const [documentData, setDocumentData] = useState(null);

  const handleFileChange = (event) => {
    setFile(event.target.files[0]);
  };

  const handleUpload = async () => {
    const formData = new FormData();
    formData.append('document', file);

    try {
      const response = await axios.post('http://localhost:5000/upload', formData, {
        headers: { 'Content-Type': 'multipart/form-data' }
      });
      setDocumentData(response.data);
    } catch (error) {
      console.error("Error uploading document", error);
    }
  };

  return (
    <div className="App">
      <h1>Document Capture</h1>
      <input type="file" onChange={handleFileChange} />
      <button onClick={handleUpload}>Upload Document</button>
      {documentData && (
        <div>
          <h2>Extracted Data</h2>
          <pre>{JSON.stringify(documentData, null, 2)}</pre>
      </div>
  )}
    </div>
  );
}

export default App;
```

Backend: Node.js with Tesseract.js
Set up the Node.js backend:

```
mkdir server
cd server
npm init -y
npm install express multer tesseract.js
```

server.js: The backend code to process the image and extract text using Tesseract.js.

```
const express = require('express');
const multer = require('multer');
const Tesseract = require('tesseract.js');

const app = express();
const upload = multer({ dest: 'uploads/' });

app.post('/upload', upload.single('document'), (req, res) => {
  const imagePath = req.file.path;

  Tesseract.recognize(
    imagePath,
    'eng',
    {
      logger: (m) => console.log(m),
    }
  ).then(({ data: { text } }) => {
    const extractedData = extractDataFromText(text);
    res.json(extractedData);
  }).catch((error) => {
    res.status(500).send('Error processing document');
  });
});

function extractDataFromText(text) {
  // Simple regex to extract information from the OCR output
  const name = text.match(/Name:\s*(.*)/);
  const docNumber = text.match(/Document Number:\s*(.*)/);
  const dob = text.match(/Date of Birth:\s*(.*)/);
  const issueDate = text.match(/Issue Date:\s*(.*)/);
  const expiryDate = text.match(/Expiration Date:\s*(.*)/);

  return {
    name: name ? name[1] : 'N/A',
    documentNumber: docNumber ? docNumber[1] : 'N/A',
    dateOfBirth: dob ? dob[1] : 'N/A',
    issueDate: issueDate ? issueDate[1] : 'N/A',
    expiryDate: expiryDate ? expiryDate[1] : 'N/A'
  };
}

app.listen(5000, () => console.log('Server running on port 5000'));
```
Explanation:

The frontend allows the user to upload a document.
The backend receives the image, uses Tesseract.js to extract text from the document, and parses it using simple regular expressions to extract relevant fields.
The extracted data is then sent back to the frontend for display.
Running the system:
Start the backend:
```
node server.js
```
Start the frontend:
```
npm start
```
You can now visit the React app in your browser, upload a sample driver's license image, and view the extracted data.

Working Version

To deploy your working version, you can use platforms like Heroku or Render for the backend and Vercel or Netlify for the frontend.

- Backend Deployment: Deploy the Node.js server on a platform like Heroku : Deploying Node.js app to Heroku.
- Frontend Deployment: Deploy the React app on Vercel or Netlify : Deploying React app on Netlify. Once deployed, provide a live demo link to the recruitment team.

Feedback and Opportunities for Iteration

What I Enjoyed: I enjoyed building the end-to-end solution, from setting up the frontend to implementing OCR and data extraction.

Challenges Encountered: One challenge was ensuring that the OCR accuracy is high enough for real-world documents, as text recognition can be affected by the quality of the image.
Improvements:
Scalability: To handle a large number of document uploads, the system could be optimised using cloud storage (e.g., AWS S3) and queues (e.g., RabbitMQ) for background processing.
Security: For production use, ensure encryption of sensitive data both in transit (HTTPS) and at rest (database encryption).
User Experience: A progress bar or spinner while the OCR is processing the document would improve UX.
This solution provides a full-stack implementation that extracts data from government-issued documents and displays it via a ReactJS interface. The backend uses Tesseract.js for OCR text extraction and validates the information.