**Invoice Management System Web Interface Design**

**Core Features**

**1. Invoice Upload and Information Extraction**

* **Functionality:**
  + Users upload an invoice image (JPEG, PNG, or PDF).
  + The system processes the uploaded file with the following flow:
    1. The uploaded image is passed to a YOLOv5 model.
    2. **YOLOv5** detects and provides boundary boxes for specific fields or sections of the invoice.
    3. These boundary boxes are cropped from the image.
    4. OCR (e.g., Tesseract) is applied to the cropped sections to extract text.
    5. Keys (field names) are derived from YOLO-detected labels, and values are extracted using OCR.
  + For now, the system uses pre-trained YOLOv5 weights to detect regions, but custom weights will be integrated later.
  + The extracted data is displayed in a form for user validation and editing.
  + Validation ensures all required fields are complete before submission.
* **UI Components:**
  + **File Upload Section:**
    1. Drag-and-drop or file picker functionality.
    2. Preview of the uploaded image.
  + **Form with Extracted Data:**
    1. **Fields to Capture:**
       - **Invoice Details:** Invoice Number, Date, Term of Sale.
       - **Supplier Information:** Name, Address, Sales Tax Registration No., NTN.
       - **Buyer Information:** Name, Address, Sales Tax Registration No., NTN.
       - **Itemized Details:** Serial Number, Description, Quantity, Rate, Amount Excluding S.T., Sales Tax, Amount Including S.T.
       - **Summary:** Total Quantity, Total Amount Excluding S.T., Sales Tax Total, Net Tax Inclusive Value.
    2. Error indicators for missing or invalid data.
    3. Submit button to save the validated data to the database.

**2. Query and Filter Records**

* **Functionality:**
  + Users can search and filter records using various criteria:
    - **Date Range:** Retrieve invoices within specific dates.
    - **Merchant Name (Supplier or Buyer):** Search for records by merchant.
    - **Item Details:** Query invoices containing specific products or services.
    - **Tax Range:** Filter records based on tax amounts.
  + Query results are displayed in a paginated, sortable table.
  + Option to export the results as CSV or Excel files.
* **UI Components:**
  + **Filter Section:**
    - Input fields for date range, merchant name, and item description.
    - Range sliders or numerical inputs for tax range.
  + **Results Table:**
    - Columns: Invoice Number, Supplier Name, Buyer Name, Date, Total Amount, Tax.
    - Expandable rows to show itemized details.
    - Export button for downloading query results.

**Additional Features**

* **Data Validation:**
  + Prevent invalid inputs, such as incorrect dates or empty fields.
  + Provide clear error messages for incomplete or incorrect data.
* **Responsive Design:**
  + Optimized layouts for desktops, tablets, and mobile devices.
* **Modern UI Design:**
  + Use Material-UI or TailwindCSS for a clean, user-friendly interface.
  + Consistent typography, colors, and button styles for a cohesive look.

**Technical Specifications**

**Frontend**

* **Framework:** React.js with reusable components.
* **Networking:** Axios for API communication.

**Backend**

* **Object Detection with YOLOv5:**
  + Use YOLOv5 with pre-trained weights (for now) to detect invoice fields.
  + Custom YOLO weights will be integrated once ready.
* **OCR Integration:**
  + Use libraries such as Tesseract to extract text from detected regions.
* **API Framework:** Flask or FastAPI for handling file uploads, YOLO processing, and OCR results.

**Database**

* **Platform:** Supabase (PostgreSQL).
* **Normalized Table Schema:**

sql

Copy code

CREATE TABLE suppliers (

id SERIAL PRIMARY KEY,

name VARCHAR(255) NOT NULL,

address TEXT NOT NULL,

sales\_tax\_reg\_no VARCHAR(50),

ntn VARCHAR(50)

);

CREATE TABLE buyers (

id SERIAL PRIMARY KEY,

name VARCHAR(255) NOT NULL,

address TEXT NOT NULL,

sales\_tax\_reg\_no VARCHAR(50),

ntn VARCHAR(50)

);

CREATE TABLE invoices (

id SERIAL PRIMARY KEY,

invoice\_number VARCHAR(50) NOT NULL,

invoice\_date DATE NOT NULL,

term\_of\_sale VARCHAR(50),

supplier\_id INT REFERENCES suppliers(id),

buyer\_id INT REFERENCES buyers(id),

total\_quantity INT NOT NULL,

total\_amount\_excluding NUMERIC(12, 2) NOT NULL,

sales\_tax\_total NUMERIC(12, 2) NOT NULL,

net\_tax\_inclusive NUMERIC(12, 2) NOT NULL,

uploaded\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

);

CREATE TABLE invoice\_items (

id SERIAL PRIMARY KEY,

invoice\_id INT REFERENCES invoices(id),

serial\_no INT NOT NULL,

description VARCHAR(255) NOT NULL,

quantity INT NOT NULL,

rate NUMERIC(12, 2) NOT NULL,

amount\_excluding NUMERIC(12, 2) NOT NULL,

sales\_tax NUMERIC(12, 2),

amount\_including NUMERIC(12, 2) NOT NULL

);

**Data Flow**

1. User uploads an invoice image.
2. The backend processes the image:
   * Passes the image and YOLOv5 weights to detect regions of interest.
   * Crops detected regions based on YOLOv5 bounding boxes.
   * Applies OCR on cropped regions to extract text.
   * Matches keys (labels from YOLO) with values (text from OCR).
3. Extracted data is sent to the frontend and displayed in a form.
4. User validates and submits the form.
5. Data is saved to the database.
6. User queries and filters records as needed.

**Wireframe Overview**

**Page 1: Invoice Upload**

* **Header:**
  + Navigation links: Upload, Query, Help.
* **Main Section:**
  + Left: File upload area with drag-and-drop functionality.
  + Right: Image preview.
  + Below: Form with extracted fields.
  + Submit button (disabled until all required fields are validated).

**Page 2: Query Records**

* **Header:**
  + Navigation links: Upload, Query, Help.
* **Main Section:**
  + Top: Filters for date range, merchant, tax range, and item search.
  + Middle: Query execution button.
  + Bottom: Paginated table displaying results with an export option.

Here's a step-by-step guide to building the **Invoice Management System Web Interface** from scratch. This guide covers project setup, development, and deployment, with detailed instructions.

**Step 1: Project Initialization**

**Frontend: React Setup**

1. **Create React Project:**
   * Open your terminal and run:

bash

Copy code

npx create-react-app invoice-management-system

cd invoice-management-system

1. **Install Required Libraries:**
   * Material-UI or TailwindCSS for styling:

bash

Copy code

npm install @mui/material @emotion/react @emotion/styled

OR, for TailwindCSS:

bash

Copy code

npm install -D tailwindcss postcss autoprefixer

npx tailwindcss init

Add TailwindCSS configuration in tailwind.config.js:

js

Copy code

module.exports = {

content: ["./src/\*\*/\*.{js,jsx,ts,tsx}"],

theme: {

extend: {},

},

plugins: [],

};

Add Tailwind imports to src/index.css:

css

Copy code

@tailwind base;

@tailwind components;

@tailwind utilities;

* + Axios for API calls:

bash

Copy code

npm install axios

* + React Router (optional, for page navigation):

bash

Copy code

npm install react-router-dom

**Backend: Flask Setup**

1. **Create a Virtual Environment:**

bash

Copy code

python -m venv env

source env/bin/activate # On Windows: env\Scripts\activate

1. **Install Required Packages:**

bash

Copy code

pip install flask flask-cors torch torchvision tesseract pytesseract pillow

1. **Set Up Project Structure:** Create the following structure:

Copy code

backend/

├── app.py

├── models/

│ └── yolov5\_model.pt

└── uploads/

**Step 2: Develop Backend**

**1. Basic Flask App (app.py):**

python

Copy code

from flask import Flask, request, jsonify

from flask\_cors import CORS

from PIL import Image

import pytesseract

import torch

import os

app = Flask(\_\_name\_\_)

CORS(app)

UPLOAD\_FOLDER = "uploads"

os.makedirs(UPLOAD\_FOLDER, exist\_ok=True)

@app.route('/upload', methods=['POST'])

def upload\_file():

if 'file' not in request.files:

return jsonify({"error": "No file provided"}), 400

file = request.files['file']

file\_path = os.path.join(UPLOAD\_FOLDER, file.filename)

file.save(file\_path)

# Dummy YOLO and OCR integration

detected\_data = dummy\_yolo\_ocr(file\_path)

return jsonify(detected\_data)

def dummy\_yolo\_ocr(file\_path):

# Replace this with actual YOLO and OCR logic

return {

"Invoice Number": "12345",

"Date": "2024-12-01",

"Supplier": "ABC Supplies",

"Total Amount": "5000"

}

if \_\_name\_\_ == '\_\_main\_\_':

app.run(debug=True)

**2. Integrate YOLOv5:**

* Add YOLO detection code. Place the yolov5\_model.pt file in the models/ directory.
* Install YOLOv5 dependencies:

bash

Copy code

pip install torchvision

* Replace dummy\_yolo\_ocr with YOLO inference logic.

**3. Configure OCR (Tesseract):**

* Install Tesseract:

bash

Copy code

sudo apt install tesseract-ocr # For Linux

brew install tesseract # For Mac

* Use Tesseract for OCR in your app:

python

Copy code

def dummy\_yolo\_ocr(file\_path):

image = Image.open(file\_path)

text = pytesseract.image\_to\_string(image)

return {"Detected Text": text}

**Step 3: Develop Frontend**

**1. File Upload Component:**

Create a FileUpload component in src/components/FileUpload.js:

jsx

Copy code

import React, { useState } from "react";

import axios from "axios";

const FileUpload = () => {

const [file, setFile] = useState(null);

const [data, setData] = useState(null);

const handleFileChange = (e) => {

setFile(e.target.files[0]);

};

const handleUpload = async () => {

if (!file) return alert("Please select a file.");

const formData = new FormData();

formData.append("file", file);

try {

const response = await axios.post("http://127.0.0.1:5000/upload", formData);

setData(response.data);

} catch (error) {

console.error("Error uploading file:", error);

}

};

return (

<div className="upload-section">

<input type="file" onChange={handleFileChange} />

<button onClick={handleUpload}>Upload</button>

{data && <pre>{JSON.stringify(data, null, 2)}</pre>}

</div>

);

};

export default FileUpload;

**2. Add Query Feature:**

* Create a QueryRecords component for filters and table.
* Use React Table for the paginated table:

bash

Copy code

npm install react-table

**3. Add Navigation:**

Use React Router to navigate between the upload and query pages.

**Step 4: Database Setup**

1. Install PostgreSQL:
   * Install locally or use Supabase for a cloud database.
2. Create Tables:
   * Execute the SQL schema provided earlier.

**Step 5: Testing and Deployment**

**1. Test Locally**

* Start the backend:

bash

Copy code

python app.py

* Start the frontend:

bash

Copy code

npm start

* Visit http://localhost:3000 in your browser.

**2. Deployment**

* Deploy Frontend:
  + Use Vercel or Netlify for React.
* Deploy Backend:
  + Use Heroku, AWS, or Render for Flask.
  + Ensure the database is connected.