Automated Invoice & Receipt Scanner SaaS

Comprehensive Developer Documentation

Table of Contents

---

1. Project Overview

Problem: Manual invoice data entry is time-consuming and error-prone.
Solution: A SaaS platform that:

- Automatically extracts structured data (vendor, amount, date) from invoices/receipts using AI/OCR.
- Provides a user-friendly interface to review, edit, and export data to tools like QuickBooks or Excel.
- Scales with Redis caching, Apache Kafka for bulk processing, and Docker for deployment.

Key Features:

- Drag-and-drop file upload (PDF/PNG/JPG).
- Real-time processing status.
- Multi-user authentication (JWT).
- Export to CSV/Excel.
- Caching for frequent vendor names.

---

2. Tech Stack

Category	Technology	Purpose
Frontend	React + TypeScript, Redux Toolkit	State management, UI rendering
Styling	TailwindCSS	Responsive design
Backend	Node.js + Express	REST API, authentication
AI/OCR Service	Python + FastAPI + gRPC	Image preprocessing and text extraction
Database	MongoDB (Atlas)	Store invoices and user data
Caching	Redis	Real-time status updates
Event Streaming	Apache Kafka	Bulk invoice processing
Testing	Jest (frontend/backend) + Pytest (OCR)	Unit and integration tests
DevOps	Docker, AWS EC2/S3	Containerization, file storage
Auth	JWT	Secure user authentication

---

3. System Architecture

```
flowchart LR
    A[Frontend] -->|HTTP| B[Node.js]
    B -->|gRPC| C[Python OCR]
    B --> D[MongoDB]
    B --> E[Redis]
    C -->|Kafka| B
    A --> F[AWS S3]
    B --> G[GraphQL]
```

Data Flow:

1. User uploads file via React frontend.
2. Node.js backend stores file in AWS S3 and queues processing via Kafka (optional).
3. Python OCR service processes the file via gRPC, extracts data, and saves to MongoDB.
4. Redis caches recent results for quick retrieval.
5. GraphQL fetches filtered invoice data for the UI.

---

4. Setup & Installation

Prerequisites

- Node.js v18+, Python 3.9+, Docker, MongoDB Atlas, Redis.
- AWS account (for S3/EC2) *[optional]*.

Step 1: Clone Repositories

```
git clone https://github.com/your-repo/frontend
git clone https://github.com/your-repo/backend
git clone https://github.com/your-repo/ocr-service
```

Step 2: Backend Setup

1. Install dependencies:

   ```
   cd backend
   npm install
   ```

2. Configure `.env`:

   ```
   MONGODB_URI=your_mongodb_uri
   JWT_SECRET=your_jwt_secret
   AWS_ACCESS_KEY=your_key
   AWS_SECRET_KEY=your_secret
   ```

3. Start the server:

   ```
   npm run dev
   ```

Step 3: Python OCR Service

1. Install dependencies:

   ```
   cd ocr-service
   pip install -r requirements.txt  # Includes Tesseract, OpenCV, grpcio
   ```

2. Start the gRPC server:

   ```
   python server.py
   ```

Step 4: Frontend Setup

1. Install dependencies:

```
cd frontend
npm install
```

2. Configure API endpoints in `src/config.ts`:

```
export const API_URL = "http://localhost:5000";
```

3. Start the app:

```
npm start
```

---

5. API Documentation

REST Endpoints

Endpoint	Method	Description
`/api/upload`	POST	Upload invoice (PDF/PNG/JPG)
`/api/invoices`	GET	List invoices (paginated)

GraphQL Query

```
query GetInvoices($userId: ID!) {
  invoices(userId: $userId) {
    id
    vendor
    amount
    date
    status
  }
}
```

gRPC Service Definition

```
service InvoiceExtractor {
  rpc Extract (InvoiceImage) returns (ExtractedData);
}

message InvoiceImage {
  bytes image_data = 1;
}

message ExtractedData {
  string vendor = 1;
  float amount = 2;
  string date = 3;
}
```

---

6. Frontend Guide

Redux State Management

- **Slices**:

```
// uploadsSlice.ts
const uploadsSlice = createSlice({
  name: "uploads",
  initialState: { progress: 0 },
  reducers: {
    setProgress: (state, action) => { state.progress = action.payload; },
  },
});
```

- **Components**:

```
// FileUpload.tsx
import { useDropzone } from "react-dropzone";
const FileUpload = () => {
  const onDrop = (files: File[]) => {
    dispatch(uploadInvoice(files[0]));
  };
  return <div {...useDropzone({ onDrop })}>Drop files here</div>;
};
```

---

7. Backend Services

File Upload Flow

1. User uploads file → Node.js validates and stores it in AWS S3.
2. Node.js sends a gRPC request to the Python OCR service.
3. Python preprocesses the image (OpenCV), extracts text (Tesseract), and returns structured data.
4. Results are cached in Redis and saved to MongoDB.

Authentication

- JWT tokens issued on login.
- Middleware to protect routes:

```
const authMiddleware = (req, res, next) => {
  const token = req.headers.authorization?.split(" ")[1];
  if (!token) return res.status(401).json({ error: "Unauthorized" });
  jwt.verify(token, process.env.JWT_SECRET, (err, user) => {
    if (err) return res.status(403).json({ error: "Invalid token" });
    req.user = user;
    next();
  });
};
```

---

8. AI/OCR Processing

Steps

1. **Preprocessing**:
   - Deskew and denoise images using OpenCV.
2. **Text Extraction**:
   - Use Tesseract OCR to extract raw text.
3. **Data Parsing**:
   - Regex patterns to identify dates (e.g., `\d{2}-\d{2}-\d{4}`) and amounts (e.g., `\$?\d+\.\d{2}`).

Optimizations

- Cache common vendor names in Redis for autocomplete suggestions.

---

9. Testing

Backend (Jest)

```
test("Upload invoice", async () => {
  const res = await request(app)
    .post("/api/upload")
    .attach("file", "test_invoice.png");
  expect(res.statusCode).toBe(200);
  expect(res.body).toHaveProperty("vendor");
});
```

Python OCR (Pytest)

```
def test_extract_amount():
    data = extract_from_image("test_invoice.png")
    assert data["amount"] == 99.99
```

---

10. Deployment

Docker Setup

```
# Node.js Dockerfile
FROM node:18
WORKDIR /app
COPY package*.json ./
RUN npm install
COPY . .
CMD ["npm", "start"]
```

AWS Deployment

1. **EC2**: Host Node.js and Python services.
2. **S3**: Store uploaded files.
3. **ElastiCache**: Configure Redis for caching.

---

11. Future Enhancements

- **Multi-language Support**: Integrate Google Cloud Vision API.
- **AI Validation**: Use GPT-4 to verify extracted data.
- **Payment Integration**: Add Stripe for subscriptions.

---

12. License & Support

- **License**: MIT.
- **Contribute**: Submit issues/pull requests to the GitHub repo.
- **Contact**: For support, email `support@invoice-scanner.com`.