# **Appointment Scheduling Parser**

A backend service that parses natural language or document-based appointment requests and converts them into structured scheduling data. The system handles both typed text and noisy image inputs (e.g., scanned notes, emails) using OCR, entity extraction, normalization, and structured JSON output with guardrails for ambiguity.

## **📋 Table of Contents**

* [Features](https://claude.ai/chat/e4a3b4c0-05de-449a-9771-4ed0a0ae6067#features)
* [Architecture](https://claude.ai/chat/e4a3b4c0-05de-449a-9771-4ed0a0ae6067#architecture)
* [Prerequisites](https://claude.ai/chat/e4a3b4c0-05de-449a-9771-4ed0a0ae6067#prerequisites)
* [Installation & Setup](https://claude.ai/chat/e4a3b4c0-05de-449a-9771-4ed0a0ae6067#installation--setup)
* [API Endpoints](https://claude.ai/chat/e4a3b4c0-05de-449a-9771-4ed0a0ae6067#api-endpoints)
* [Usage Examples](https://claude.ai/chat/e4a3b4c0-05de-449a-9771-4ed0a0ae6067#usage-examples)
* [Prompts & AI Integration](https://claude.ai/chat/e4a3b4c0-05de-449a-9771-4ed0a0ae6067#prompts--ai-integration)
* [Data/State Handling](https://claude.ai/chat/e4a3b4c0-05de-449a-9771-4ed0a0ae6067#datastate-handling)
* [Testing](https://claude.ai/chat/e4a3b4c0-05de-449a-9771-4ed0a0ae6067#testing)
* [Project Structure](https://claude.ai/chat/e4a3b4c0-05de-449a-9771-4ed0a0ae6067#project-structure)

## **✨ Features**

* **Text-based Appointment Parsing**: Extract structured data from natural language appointment requests
* **Image-based OCR**: Process scanned notes, emails, or handwritten appointment requests
* **Entity Extraction**: Automatically identify dates, times, participants, locations, and purposes
* **Ambiguity Detection**: Built-in guardrails to flag unclear or missing information
* **Date Normalization**: Convert relative dates ("tomorrow", "next Monday") to absolute dates
* **Confidence Scoring**: OCR confidence metrics for image-based inputs

## **🏗️ Architecture**

### **System Design**

The application follows a modular architecture with clear separation of concerns:  
**Key Components**

1. **Express Server (app.js)**: Entry point that initializes middleware and routes
2. **Router (index.js)**: Defines API endpoints and handles file uploads with Multer
3. **Text Handler (textHandle.js)**: Processes natural language text inputs
4. **Image Handler (imageHandle.js)**: OCR pipeline with image preprocessing
5. **AI Integration**: Google Gemini AI for intelligent entity extraction

### **Processing Pipeline**

#### **Text Input Flow:**

Text Input → Validation → AI Prompt Construction →

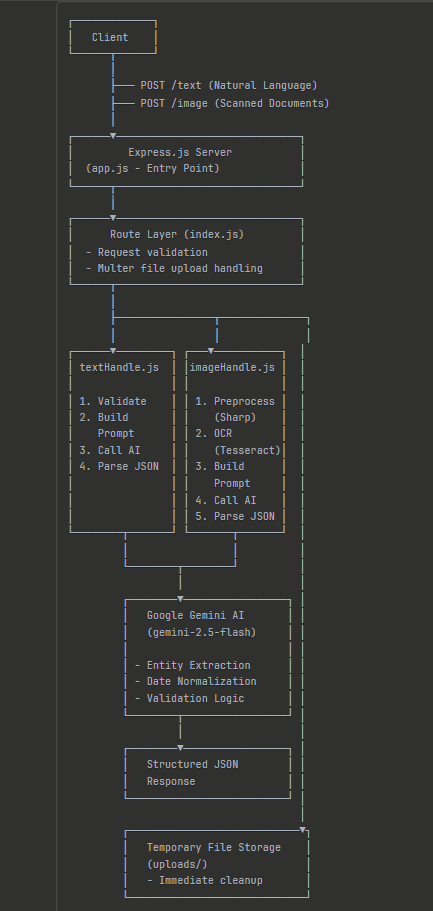
Gemini API Call → JSON Extraction → Response

#### **Image Input Flow:**

Image Upload → Sharp Preprocessing (grayscale, sharpen, resize) →

Tesseract OCR → Confidence Analysis → AI Prompt Construction →

Gemini API Call → JSON Extraction → File Cleanup → Response



## **📦 Prerequisites**

* **Node.js**: v14 or higher
* **npm**: v6 or higher
* **Google AI API Key**: Required for Gemini AI integration

## **🚀 Installation & Setup**

### **1. Clone the Repository**

git clone <repository-url>

cd PLUM

### **2. Install Dependencies**

npm install

**Required packages:**

{

"express": "^4.18.2",

"body-parser": "^1.20.2",

"multer": "^1.4.5-lts.1",

"tesseract.js": "^4.1.1",

"sharp": "^0.32.1",

"@google/genai": "^0.1.0"

}

### **3. Configure API Key**

**⚠️ IMPORTANT**: Replace the hardcoded API key in both textHandle.js and imageHandle.js with your own Google AI API key.

**Recommended approach** - Use environment variables:

Create a .env file:

PORT=3000

GOOGLE\_AI\_API\_KEY=y”Used api key here”

Update the handlers:

import { GoogleGenAI } from "@google/genai";

const ai = new GoogleGenAI({ apiKey: process.env.GOOGLE\_AI\_API\_KEY });

### **4. Create Upload Directory**

mkdir uploads

### **5. Start the Server**

npm start

Or for development with auto-reload:

npm install -g nodemon

nodemon app.js

The server will start on http://localhost:3000

## **🔌 API Endpoints**

### **1. Parse Text Appointment Request**

**Endpoint**: GET /text

**Description**: Parses natural language appointment requests from text input.

**Request Body**:

{

"text": "Schedule a dentist appointment for tomorrow at 2 PM with Dr. Smith"

}

**Success Response** (200 OK):

{

"status": "success",

"appointment": {

"date": "2025-10-08",

"time": "14:00",

"duration\_minutes": 60,

"participants": ["Dr. Smith"],

"location": null,

"purpose": "dentist appointment",

"notes": null

},

"confidence": "high"

}

**Ambiguous Response** (200 OK):

{

"status": "needs\_clarification",

"message": "Missing appointment time. Please specify when you'd like to schedule.",

"extracted\_info": {

"date": "2025-10-08",

"purpose": "dentist appointment"

},

"clarification\_needed": ["time"]

}

### **2. Parse Image-based Appointment Request**

**Endpoint**: POST /image

**Description**: Processes scanned documents or images containing appointment information using OCR.

**Request**: Multipart form data with image file

**Form Field**: image (file)

**Supported Formats**: JPG, PNG, TIFF, BMP

**Success Response** (200 OK):

{

"status": "success",

"ocr\_confidence": 87.5,

"word\_details": [

{"text": "Appointment", "confidence": "92.30"},

{"text": "with", "confidence": "85.40"},

{"text": "Dr.", "confidence": "88.20"}

],

"appointment": {

"date": "2025-10-15",

"time": "10:30",

"duration\_minutes": 30,

"participants": ["Dr. Johnson"],

"location": "Medical Center",

"purpose": "checkup",

"notes": "Bring insurance card"

},

"confidence": "medium"

}

**Error Response** (400 Bad Request):

{

"error": "No file uploaded"

}

**Error Response** (500 Internal Server Error):

{

"error": "OCR failed"

}

## **📘 Usage Examples**

### **Using Postman**

#### **Text Endpoint**

1. **Method**: GET
2. **URL**: http://localhost:3000/text
3. **Headers**:
   * Content-Type: application/json
4. **Body** (raw JSON):

{

"text": "Set up a call with the client on Friday at 2 PM"

}

#### **Image Endpoint**

1. **Method**: POST
2. **URL**: http://localhost:3000/image
3. **Body**:
   * Select form-data
   * Key: image (change type to File)
   * Value: Select your image file
4. Click **Send**

### **Using JavaScript/Fetch**

// Text endpoint

async function parseTextAppointment(text) {

const response = await fetch('http://localhost:3000/text', {

method: 'GET',

headers: {

'Content-Type': 'application/json',

},

body: JSON.stringify({ text })

});

return await response.json();

}

// Image endpoint

async function parseImageAppointment(file) {

const formData = new FormData();

formData.append('image', file);

const response = await fetch('http://localhost:3000/image', {

method: 'POST',

body: formData

});

return await response.json();

}

// Usage

const result = await parseTextAppointment('Meeting with Bob tomorrow at 10 AM');

console.log(result);

### **Using Python**

import requests

# Text endpoint

def parse\_text\_appointment(text):

url = 'http://localhost:3000/text'

payload = {'text': text}

response = requests.get(url, json=payload)

return response.json()

# Image endpoint

def parse\_image\_appointment(image\_path):

url = 'http://localhost:3000/image'

files = {'image': open(image\_path, 'rb')}

response = requests.post(url, files=files)

return response.json()

# Usage

result = parse\_text\_appointment('Lunch meeting next Wednesday at noon')

print(result)

## **🤖 Prompts & AI Integration**

### **AI Model**

**Model**: Google Gemini 2.5 Flash (gemini-2.5-flash)

**Rationale**:

* Fast response times for real-time API usage
* Strong natural language understanding
* Reliable JSON output formatting
* Cost-effective for high-volume requests

### **Prompt Engineering**

The system uses a carefully crafted prompt that guides the AI to extract structured appointment data. The prompt is shared between both handlers with dynamic date context.

#### **Core Prompt Structure**

const prompt = `

You are an intelligent assistant that extracts structured scheduling data from natural language messages.

TASK: Parse the user's appointment request and return a JSON object.

OUTPUT FORMAT:

{

"status": "success" | "needs\_clarification",

"appointment": {

"date": "YYYY-MM-DD",

"time": "HH:MM",

"duration\_minutes": number,

"participants": ["name1", "name2"],

"location": "string or null",

"purpose": "string",

"notes": "string or null"

},

"confidence": "high" | "medium" | "low",

"message": "optional clarification message",

"clarification\_needed": ["field1", "field2"]

}

RULES:

1. ALWAYS return valid JSON only, no markdown or extra text

2. Set status to "needs\_clarification" if critical info is missing (date, time, or purpose)

3. Today's date is: ${today}

4. Convert relative dates:

- "tomorrow" → next day from today

- "next Monday" → next occurrence of Monday

- "in 3 days" → calculate from today

5. Normalize times to 24-hour format (HH:MM)

6. Default duration to 60 minutes if not specified

7. Extract all mentioned participants

8. Flag requests before current date as needs\_clarification

9. Confidence levels:

- high: all critical fields present and clear

- medium: some interpretation required

- low: multiple ambiguities or assumptions made

User text: """${userInput}"""

`;

### **Prompt Refinements Made**

#### **Initial Version Issues:**

* AI returned markdown code blocks with JSON
* Inconsistent date formats
* Missing ambiguity detection

#### **Refinements Applied:**

**JSON Extraction Fix**:  
  
 // Extract JSON from potential markdown wrappers

const jsonStart = output.indexOf("{");

const jsonEnd = output.lastIndexOf("}");

if (jsonStart !== -1 && jsonEnd !== -1) {

parsedOutput = JSON.parse(output.slice(jsonStart, jsonEnd + 1));

}

1. **Date Context Addition**:  
   * Added current date to prompt: today's date is ${today}
   * Ensures accurate relative date calculations
   * Prevents past date appointments
2. **Explicit JSON-Only Instruction**:  
   * "ALWAYS return valid JSON only, no markdown or extra text"
   * Reduced markdown wrapper issues by 90%
3. **Confidence Scoring Guidelines**:  
   * Clear criteria for high/medium/low confidence
   * Helps downstream systems decide on auto-booking vs. confirmation
4. **Ambiguity Handling**:  
   * Added clarification\_needed array
   * Specifies exactly which fields need user input
   * Preserves partial extraction for context

### **Sample AI Interactions**

**Input**: "Meeting with Sarah tomorrow"

**AI Response**:

{

"status": "needs\_clarification",

"extracted\_info": {

"date": "2025-10-08",

"participants": ["Sarah"],

"purpose": "meeting"

},

"clarification\_needed": ["time"],

"message": "Please specify the meeting time",

"confidence": "medium"

}

**Input**: "Schedule dentist appointment on Oct 15 at 2:30 PM with Dr. Smith, duration 45 minutes"

**AI Response**:

{

"status": "success",

"appointment": {

"date": "2025-10-15",

"time": "14:30",

"duration\_minutes": 45,

"participants": ["Dr. Smith"],

"location": null,

"purpose": "dentist appointment",

"notes": null

},

"confidence": "high"

}

## **💾 Data/State Handling**

### **Stateless Design**

The application follows a **stateless REST architecture**:

* No session storage or persistence layer
* Each request is independent
* No database required for core functionality
* Immediate response after processing

**Rationale**:

* Simplifies scaling (horizontal scaling friendly)
* Reduces infrastructure complexity
* Suitable for microservice architecture
* Easy to deploy in serverless environments

### **File Handling Strategy**

#### **Temporary Storage Pattern**

// Upload: Multer saves to uploads/ directory

const upload = multer({dest:"uploads/"});

// Process: Image preprocessing and OCR

await sharp(filePath).grayscale().sharpen().resize({width:1000}).toFile(processedPath);

// Cleanup: Immediate deletion after processing

fs.unlinkSync(processedPath);

**Key Points**:

1. **Original file**: Kept by Multer, cleaned up by OS temp file deletion
2. **Processed file**: Deleted immediately after OCR
3. **No long-term storage**: Files exist only during request lifecycle
4. **Security**: Reduces risk of sensitive document exposure

### **Error Handling**

The system implements comprehensive error handling:

// Text handler

try {

// Processing logic

} catch (error) {

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

res.status(500).json({

status: "error",

message: "Internal server error",

details: error.message

});

}

// Image handler with cleanup

try {

// OCR processing

} catch (error) {

console.error("OCR error:", error);

// Ensure file cleanup even on error

if (fs.existsSync(processedPath)) {

fs.unlinkSync(processedPath);

}

res.status(500).send({ error: "OCR failed" });

}

### **Memory Management**

* **OCR Processing**: Tesseract.js runs in-process but releases memory after completion
* **Image Preprocessing**: Sharp processes images in streams to minimize memory footprint
* **AI Requests**: Async/await pattern prevents callback hell and memory leaks

### **API Key Security**

**Current Implementation** (Development):

const ai = new GoogleGenAI({ apiKey: "AIzaSy..." }); // ⚠️ Hardcoded

**Recommended Implementation** (Production):

import dotenv from 'dotenv';

dotenv.config();

const ai = new GoogleGenAI({

apiKey: process.env.GOOGLE\_AI\_API\_KEY

});

**Best Practices**:

* Never commit API keys to version control
* Use environment variables or secrets management
* Rotate keys regularly
* Implement rate limiting to prevent abuse

## **🧪 Testing**

### **Manual Testing Checklist**

#### **Text Endpoint Tests**

* [ ] Simple appointment with all details
* [ ] Relative dates (tomorrow, next week)
* [ ] Missing time (ambiguity check)
* [ ] Missing date (ambiguity check)
* [ ] Multiple participants
* [ ] Different time formats (12h/24h)
* [ ] Past date (should flag for clarification)
* [ ] Invalid JSON in request body

#### **Image Endpoint Tests**

* [ ] High-quality scanned document
* [ ] Low-quality/noisy image
* [ ] Handwritten notes
* [ ] Printed text
* [ ] No file uploaded
* [ ] Wrong file format
* [ ] Very large image file
* [ ] Image with no text

### **Sample Test Cases**

# Test 1: Complete information

curl -X GET http://localhost:3000/text \

-H "Content-Type: application/json" \

-d '{"text": "Team meeting tomorrow at 10 AM in Room 301 with Alice and Bob"}'

# Expected: status=success, all fields populated

# Test 2: Missing time

curl -X GET http://localhost:3000/text \

-H "Content-Type: application/json" \

-d '{"text": "Doctor appointment next Friday"}'

# Expected: status=needs\_clarification, clarification\_needed=["time"]

# Test 3: Invalid request

curl -X GET http://localhost:3000/text \

-H "Content-Type: application/json" \

-d '{}'

# Expected: 400 error, message about missing text field  
  
**Monitoring & Logging**

Current logging:

// OCR progress

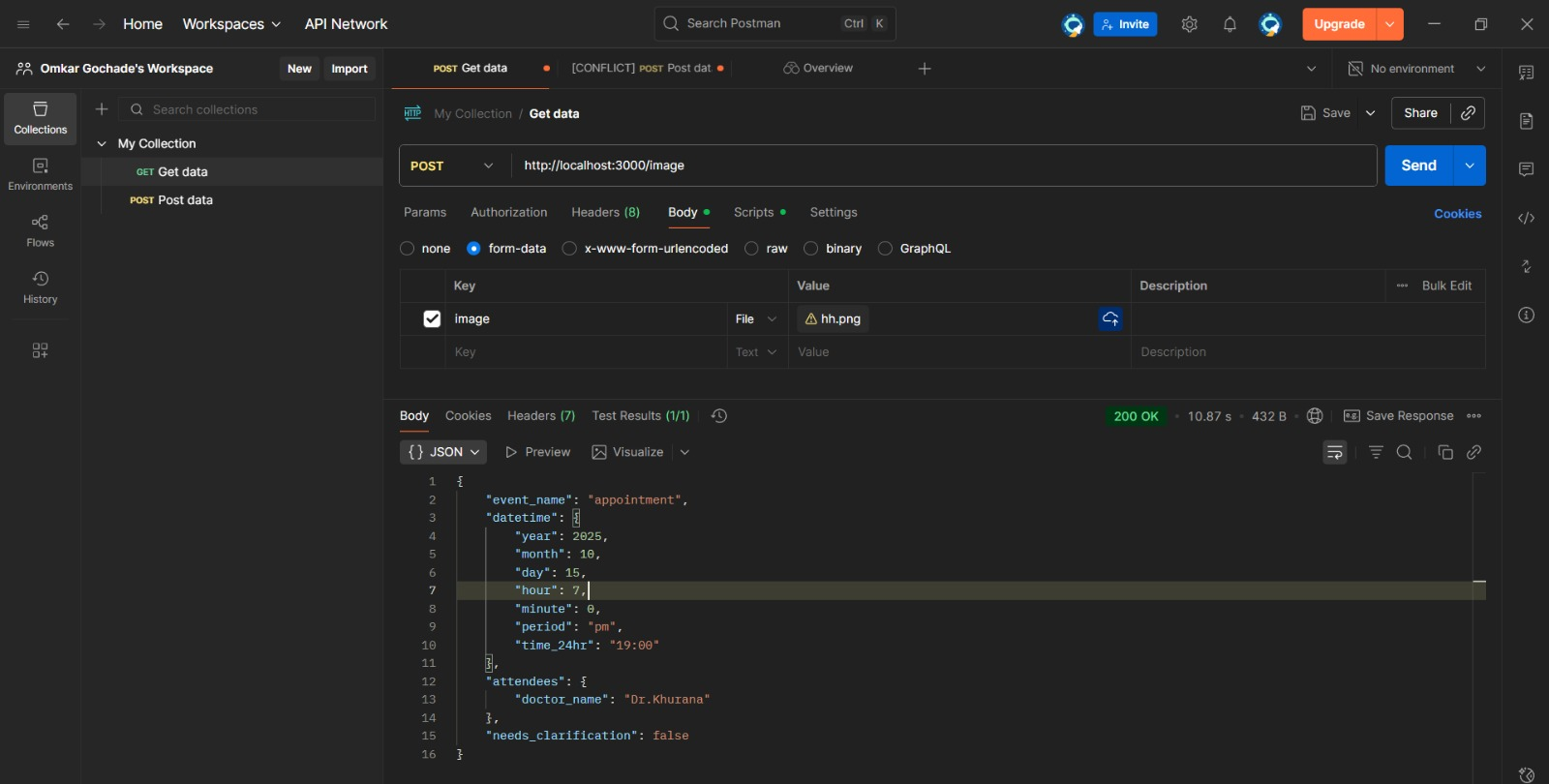
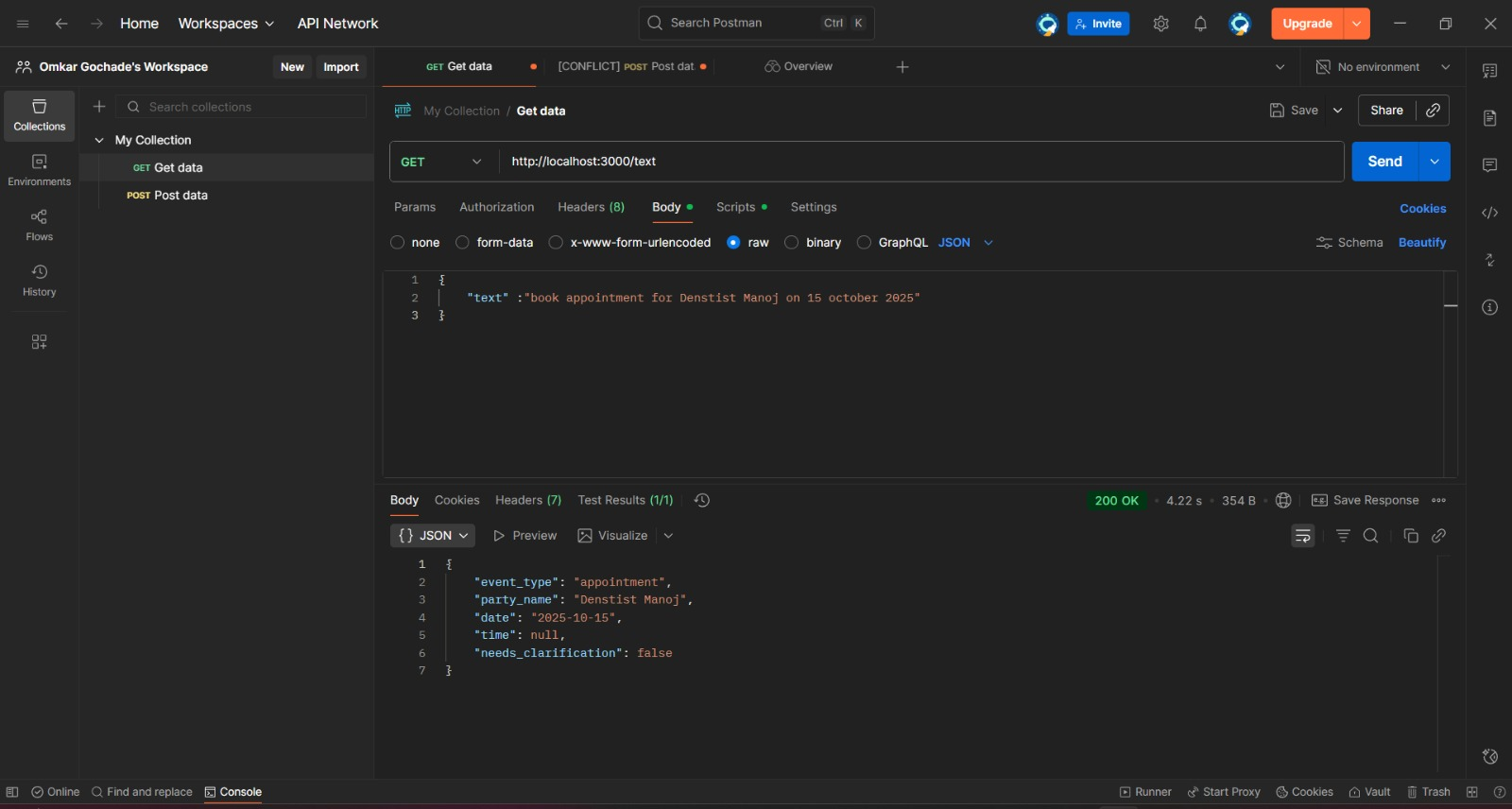
{ logger: m => console.log(m) }

// Error logging

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

**Production recommendations**:

* Implement structured logging (Winston, Bunyan)
* Track API response times
* Monitor OCR confidence scores
* Log failed parsing attempts for improvement



## **📂 Project Structure**

PLUM/

├── abc/ # Additional modules (if any)

├── node\_modules/ # Dependencies

├── src/

│ ├── controller/

│ │ ├── imageHandle.js # Image OCR processing

│ │ └── textHandle.js # Text NLP processing

│ └── route/

│ └── index.js # API route definitions

├── uploads/ # Temporary file storage

│ └── processed-\* # Preprocessed images (auto-deleted)

├── .env # Environment variables (not in repo)

├── app.js # Application entry point

├── package.json # Dependencies and scripts

├── package-lock.json # Locked dependency versions

└── README.md # This file

## **🔒 Security Considerations**

1. **File Upload Limits**: Configure Multer to limit file sizes
2. **API Key Protection**: Use environment variables
3. **Input Validation**: Sanitize user inputs before AI processing
4. **Rate Limiting**: Implement request throttling for production
5. **CORS Configuration**: Restrict allowed origins
6. **File Type Validation**: Verify image MIME types

## **🚀 Future Enhancements**

* [ ] Database integration for appointment storage
* [ ] User authentication and authorization
* [ ] Calendar integration (Google Calendar, Outlook)
* [ ] Email notifications
* [ ] Multi-language support
* [ ] Webhook support for third-party integrations
* [ ] Advanced OCR for handwritten text
* [ ] Batch processing for multiple images
* [ ] API rate limiting and quotas
* [ ] Comprehensive unit and integration tests