

# Documentation: Azure Cognitive Services OCR Integration Project

Mohammed Waseem Tabrez

801404522

## Introduction

This document provides a comprehensive overview of the **Azure Cognitive Services OCR Integration Project**. The project leverages Azure's powerful **Computer Vision API** to extract text from images using Optical Character Recognition (OCR). This API can analyze images to detect and extract printed or handwritten text, making it a valuable tool for text extraction from scanned documents, street signs, business cards, and more.

---

## Table of Contents

### 1. Overview

- Project Description
- Key Technologies Used

### 2. Azure Cognitive Services OCR

- What is OCR?
- Overview of Azure OCR

### 3. Technologies Used

- Node.js
- Express.js
- Axios
- Swagger

### 4. Project Setup

- Prerequisites
- Environment Setup
- Code Configuration

### 5. Execution of the Project

- Running the Server
- Testing with Postman
- Swagger Documentation Access

6. **Server Log Output**
  7. **Challenges and Solutions**
  8. **Conclusion**
  9. **Deployment of API on Render**
  10. **Screenshots and Demo**
- 

## 1. Overview

### Project Description

The **Azure Cognitive Services OCR Integration Project** connects a Node.js application to the **Azure Computer Vision API**, specifically the OCR functionality. This API extracts text from images by leveraging machine learning and computer vision techniques. The project demonstrates how to integrate the Azure API into a Node.js application, enabling the extraction of text from images hosted on remote servers.

### Key Technologies Used:

- **Node.js:** A JavaScript runtime used for building the server-side application.
  - **Express.js:** A web framework for Node.js used to handle API routes and server-side logic.
  - **Axios:** A promise-based HTTP client used for making requests to the Azure Cognitive Services API.
  - **Swagger:** A tool for documenting the API and making it easy for developers to interact with the endpoints.
- 

## 2. Azure Cognitive Services OCR

### What is OCR?

Optical Character Recognition (OCR) is a technology used to recognize text within images. It converts different types of documents, such as scanned paper documents, PDFs, or images taken by a digital camera, into editable and searchable data. OCR can detect both printed and handwritten text in images.

### Overview of Azure OCR

Azure's **Computer Vision API** provides OCR functionality, allowing users to extract text from images. It is part of the **Azure Cognitive Services** suite and is designed to be easy to use with minimal setup required.

The OCR API has the ability to analyze images asynchronously, providing both **printed text** (e.g., books, posters) and **handwritten text** recognition. Azure Cognitive Services offers multiple services, including **Image Analysis**, **Text Analytics**, and **Face Recognition**, all of which can be accessed via REST API endpoints.

- **Azure Cognitive Services OCR Features:**

- Detects printed text in images from URLs or local files.
- Supports multiple languages and character sets.
- Can process large and complex images, such as forms or documents with mixed content (images and text).

For more information on Azure OCR, refer to the [official documentation](#).

---

### 3. Technologies Used

#### Node.js

Node.js is a JavaScript runtime used to build scalable network applications. It is built on the V8 JavaScript engine, which is known for its high performance. Node.js allows us to handle asynchronous operations, which is ideal for making HTTP requests to Azure's OCR API.

#### Express.js

Express.js is a minimalist web framework for Node.js that simplifies routing, middleware integration, and server setup. In this project, Express handles HTTP requests and routes, enabling easy communication with the Azure OCR API.

#### Axios

Axios is a promise-based HTTP client that allows making asynchronous requests. In this project, Axios is used to send requests to the Azure API for OCR operations and to fetch results.

#### Swagger

Swagger is an open-source tool used for API documentation. It provides an interactive UI that allows developers to view and test API endpoints directly from the browser. This makes the process of understanding and interacting with the API much easier.

---

### 4. Project Setup

#### Prerequisites

Before getting started, make sure you have the following installed on your system:

- **Node.js:** [Download Node.js](#)
- **Postman:** [Download Postman](#)

Additionally, you need to have:

- **Azure Cognitive Services OCR Subscription Key and Endpoint URL.**

#### Environment Setup

1. Clone the repository:
2. `git clone https://github.com/Alitabrez786123/System-Integration-Final-Project.git`
3. `cd System-Integration-Final-Project`

4. Install dependencies:
  5. npm install
  6. Update the following values in your index.js file:
    - Replace YOUR\_SUBSCRIPTION\_KEY with your actual subscription key.
    - Replace YOUR\_REGION with the appropriate region (e.g., eastus).
  7. const region = 'eastus'; // Your region
  8. const subscriptionKey = 'YOUR\_SUBSCRIPTION\_KEY'; // Your subscription key
  9. const endpoint =  
'https://YOUR\_REGION.api.cognitive.microsoft.com/vision/v3.2/read/analyze';
- 

## 5. Execution of the Project

### Running the Server

1. Run the application using Node.js:
2. node index.js
3. The server will run on http://localhost:10000.

### Testing with Postman

To test the OCR functionality with **Postman**:

1. Open Postman and set the request type to **POST**.
2. Set the request URL to http://localhost:10000/ocr.
3. In the **Body** tab, select **raw** and choose **JSON** as the content type.
4. Provide the following sample request body:
5. {
6. "imageUrl": "https://your-image-url-here"
7. }
8. Send the request and you should receive the extracted text as a response.

### Swagger Documentation Access

Swagger provides an easy-to-use interface for testing the API. Once the server is running, access the Swagger documentation by visiting:

<http://localhost:10000/api-docs>

This page allows you to interact with the /ocr endpoint and test the OCR functionality.

---

## 6. Server Log Output

When the server is running and you make a successful POST request, you will see logs in the terminal indicating the extracted text. For example:

Extracted Text:

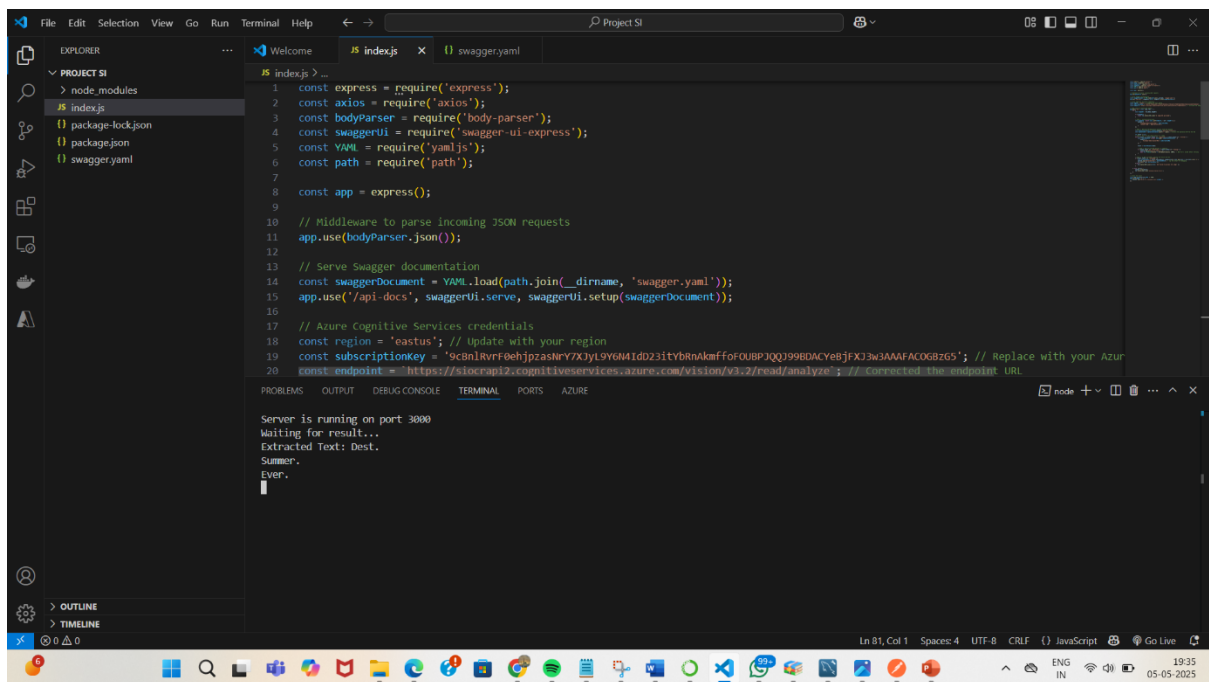
Pennsylvania

COMMERCIAL

DRIVER'S LICENSE

...

### Screenshot: Server Running Status



```
1 const express = require('express');
2 const axios = require('axios');
3 const bodyParser = require('body-parser');
4 const swaggerUi = require('swagger-ui-express');
5 const YAML = require('yamljs');
6 const path = require('path');
7
8 const app = express();
9
10 // Middleware to parse incoming JSON requests
11 app.use(bodyParser.json());
12
13 // Serve Swagger documentation
14 const swaggerDocument = YAML.load(path.join(__dirname, 'swagger.yaml'));
15 app.use('/api-docs', swaggerUi.serve, swaggerUi.setup(swaggerDocument));
16
17 // Azure Cognitive Services credentials
18 const region = 'eastus'; // Update with your region
19 const subscriptionKey = '9c8nlrvrF0ehjp2ashrY7XJYl9V6M4IdD23ityBrnAkMffof0UBP3QJ399BDACYeBjFXJ3w3AAAFACOGZ6S'; // Replace with your Azure
20 const endpoint = 'https://stocrapi2.cognitiveservices.azure.com/vision/v3.2/read/analyze'; // Corrected the endpoint URL
21
22 app.post('/ocr', async (req, res) => {
23   const { image_url } = req.body;
24   if (!image_url) {
25     res.status(400).json({ error: 'Image URL is required' });
26     return;
27   }
28   try {
29     const response = await axios.post(endpoint, { url: image_url }, { headers: { 'Ocp-Apim-Subscription-Key': subscriptionKey } });
30     const extractedText = response.data[0].text;
31     res.json({ text: extractedText });
32   } catch (error) {
33     res.status(500).json({ error: 'OCR processing failed' });
34   }
35 });
36
37 app.listen(3000, () => {
38   console.log('Server is running on port 3000');
39 });
```

Server is running on port 3000  
Waiting for result...  
Extracted Text: Dest.  
Summer.  
Ever.

## 7. Challenges and Solutions

### Challenge 1: Handling Asynchronous OCR Results

One challenge faced was the asynchronous nature of the OCR processing. Azure's OCR API processes images in the background, and we needed to poll for results. To address this, the solution included setting up a loop to check for the succeeded status of the OCR process until it completed.

### Challenge 2: Handling Invalid Responses

Another challenge was handling errors returned from the API, such as invalid or inaccessible URLs. The error handling mechanism was added to return appropriate error responses to the user.

## 8. Deployment of the API on Render

To deploy the OCR API on Render, we've chosen Render as the cloud platform for hosting our service. Render provides automatic scaling, efficient deployment, and a simple configuration process, making it an ideal choice for quickly hosting web services.

## 1. Linking the GitHub Repository

### Deployment and Accessing the API

- The deployment will be live at the URL:

<https://system-integration-final-project.onrender.com>

- **Important:** This URL will only respond to POST requests sent via **Postman** or another API client. It is not a simple GET request URL; to get OCR results, you must send a POST request with the image URL in the body.

## 2. Test the API using Postman

- Open **Postman** and send a POST request to <https://system-integration-final-project.onrender.com/ocr> with a JSON body containing the imageUrl of the image you want to process. For example:

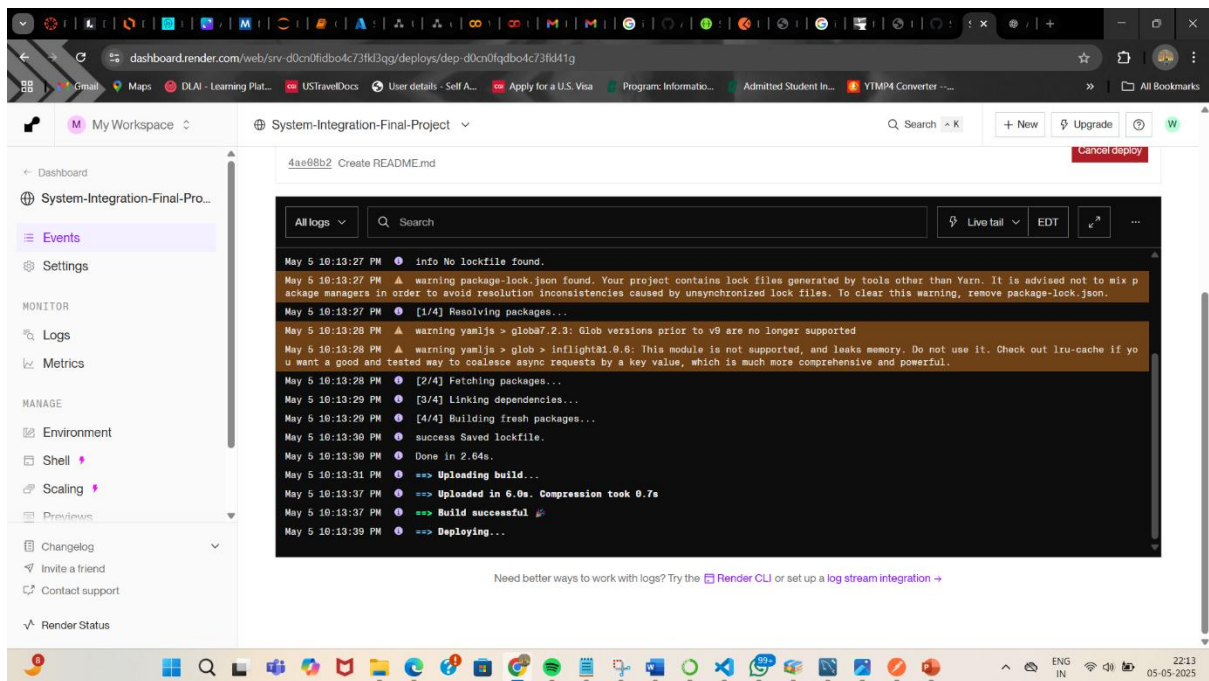
```
{  
  "imageUrl": "https://images.unsplash.com/photo-1494871262121-  
49703fd34e2b?fm=jpg&q=60&w=3000&ixlib=rb-  
4.0.3&ixid=M3wxMjA3fDB8MHxzZWZyY2h8Nnx8dGV4dHxlbmwHwwfHx8MA%3D%3D"  
}
```

- When the request is sent, the API will process the image and return the text extracted from the image, if successful.

## 3. Monitor and Manage the Service

- Render provides live logs and monitoring tools where you can see the real-time status of your service, monitor request errors, and check deployment logs.
-

## Render Deployment Screenshot:



## 9. Conclusion

This project successfully integrates Azure's Cognitive Services OCR API with a Node.js backend application, allowing users to extract text from images using a simple API. The use of **Axios** for making requests and **Swagger** for API documentation improves the overall user experience for interacting with the OCR service.

## 10. Screenshots and Demo

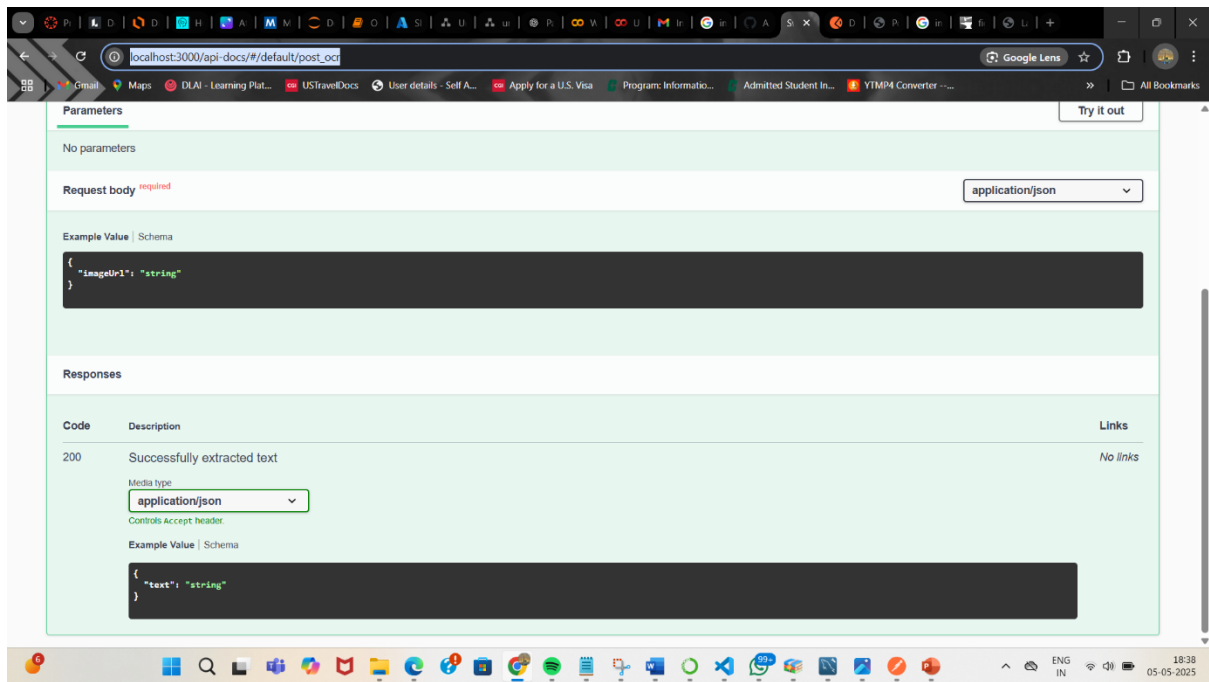
## Azure OCR Computer Vision Dashboard

The screenshot shows the Microsoft Azure portal interface. The browser address bar displays the URL: `portal.azure.com/#@waseemtabrez786outlook.onmicrosoft.com/resource/subscriptions/6865014b-c01e-437f-b108-aea48bdb24c7/resourceGroups/MySIOCRAPI/providers/Microsoft...`. The page title is "SIOCRAPI2 Computer vision". The left sidebar contains a navigation menu with options: Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Resource visualizer, Resource Management, Security, Monitoring, Automation, and Help. The main content area is titled "Essentials" and includes a "Delete" button, a search bar, and a "JSON View" link. Below this, there are details for the resource group (move), status (Active), location (East US), subscription (move), subscription ID, tags (edit), and a "See more" link. A "Get Started" section is also present, with a "Monitoring" tab and a "Get started with your resource in Vision Studio" link. A "Keys and endpoint" section is visible at the bottom, with a "Go to Vision Studio" button. The bottom of the screen shows a Windows taskbar with various application icons and a system tray with the date and time (19:28, 05-05-2025).

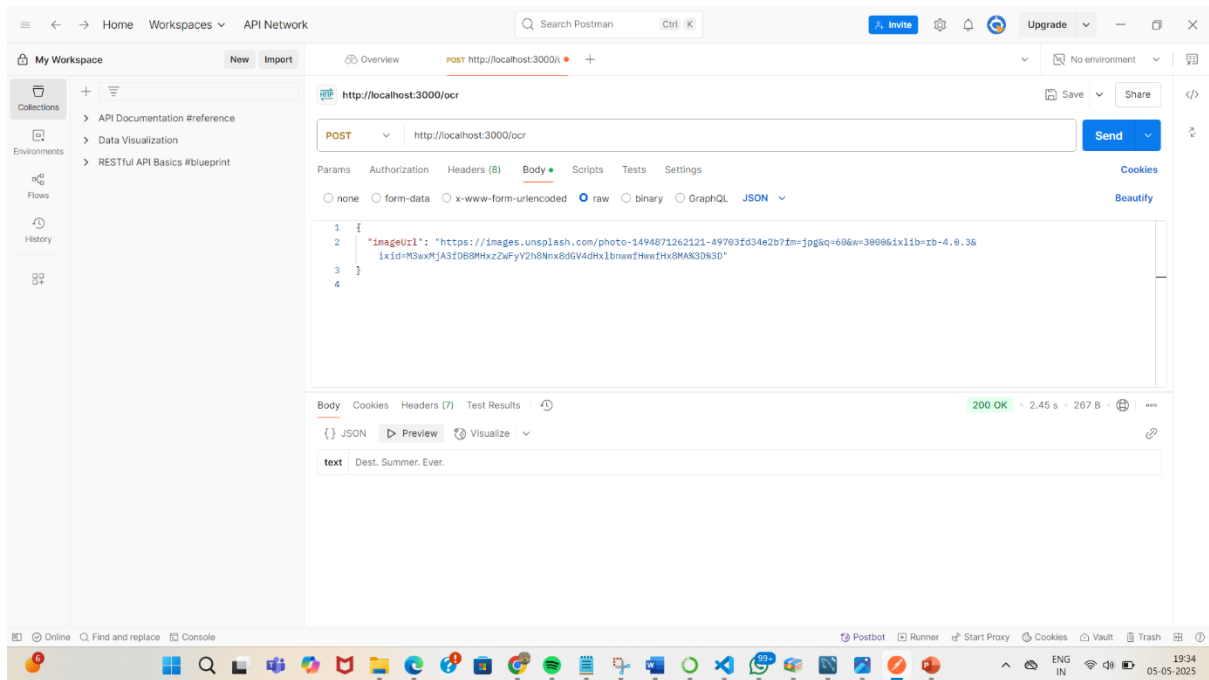
## Swagger Documentation

The screenshot shows the Swagger documentation for the Azure OCR API. The browser address bar displays the URL: `localhost:3000/api-docs/#/default/post_ocr`. The page title is "Azure OCR API" with version "1.0.0" and "OAS 3.0" labels. The description is "API for extracting text using Azure OCR". The "default" section is expanded, showing a "POST /ocr" endpoint with the description "Extract text from an image using Azure OCR". The "Parameters" section is empty, with a "Try it out" button. The "Request body" section is marked as "required" and has a dropdown menu set to "application/json". An "Example Value" is shown as a JSON object: `{ "image1": "string" }`. The bottom of the screen shows a Windows taskbar with various application icons and a system tray with the date and time (18:38, 05-05-2025).

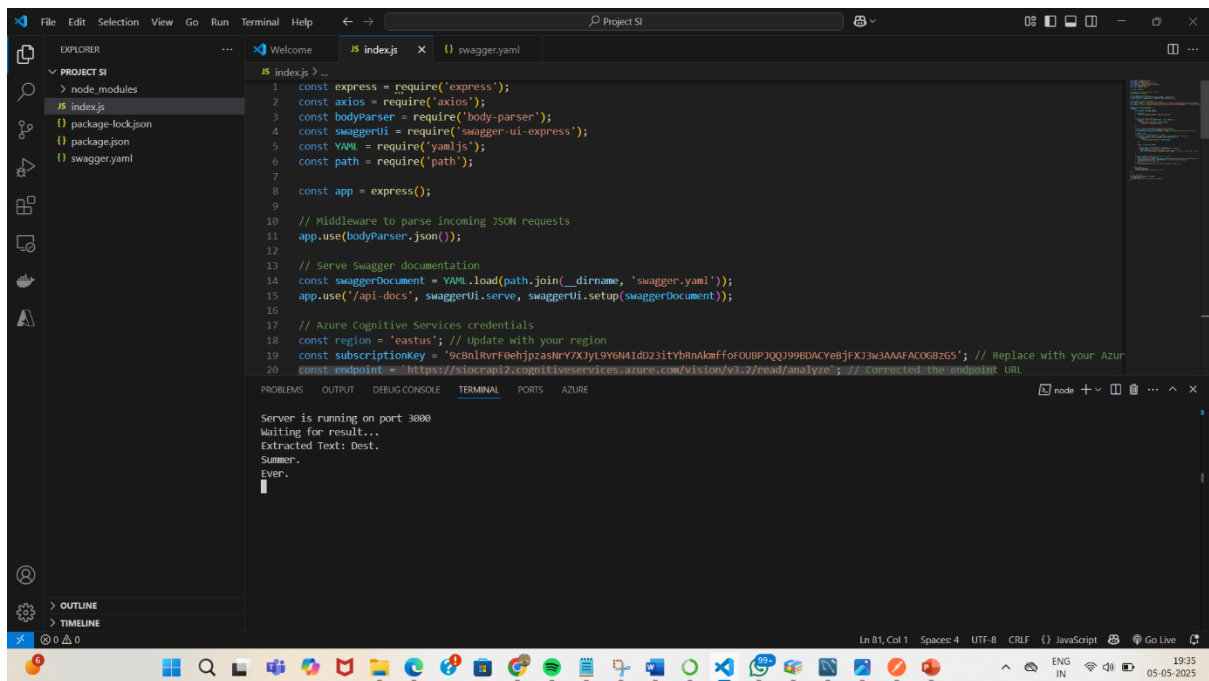




## Postman Execution



## VSCode Server log output



The screenshot displays the Visual Studio Code (VS Code) interface. The Explorer panel on the left shows the project structure with files: `index.js`, `package-lock.json`, `package.json`, and `swagger.yaml`. The main editor area shows the content of `index.js`, which is a Node.js script using Express.js to serve Swagger documentation and make an API call to Azure Cognitive Services. The script includes comments for updating the region and subscription key, and a note about correcting the endpoint URL. The terminal at the bottom shows the output of the script, indicating it is running on port 3000 and has successfully extracted text from an image.

```
1 const express = require('express');
2 const axios = require('axios');
3 const bodyParser = require('body-parser');
4 const swaggerUi = require('swagger-ui-express');
5 const YAML = require('yamljs');
6 const path = require('path');
7
8 const app = express();
9
10 // Middleware to parse incoming JSON requests
11 app.use(bodyParser.json());
12
13 // Serve Swagger documentation
14 const swaggerDocument = YAML.load(path.join(__dirname, 'swagger.yaml'));
15 app.use('/api-docs', swaggerUi.serve, swaggerUi.setup(swaggerDocument));
16
17 // Azure Cognitive Services credentials
18 const region = 'eastus'; // Update with your region
19 const subscriptionKey = '9c8nleVrF0ehjpasrY7XjYl9Y6M1dD23iTYBrnAkffofOUBP3QQJ99BDACYe8JfXJ3w3AAAFACOG8Z65'; // Replace with your Azure
20 const endpoint = 'https://stocrapi2.cognitiveservices.azure.com/vision/v3.2/read/analyze'; // Corrected the endpoint URL
```

Server is running on port 3000  
Waiting for result...  
Extracted Text: Dest.  
Summer.  
Ever.