Azure AI Optical Character Recognition (OCR) API Project

Ruthvik Madavaram

University of North Carolina at Charlotte

IT IS 6177: System Integration

Dr. Fabio Nolasco

December 06, 2024

Table of Contents

1. Introduction

OCR (Optical Character Recognition) is a machine-learning-based technique for extracting text from various types of images, including in-the-wild and non-document images like product labels, user-generated images, screenshots, street signs, and posters. This technology leverages advanced AI models to identify and extract textual content from visual inputs, providing a foundation for numerous applications in content analysis, user experience enhancement, and automation.

The Azure AI Vision OCR service delivers a fast, synchronous API optimized for lightweight scenarios where images aren't text-heavy. This makes it suitable for embedding in near real-time user experiences to enrich content understanding and enable follow-up user actions with minimal delay. A custom API has been built using the Azure OCR API, allowing access to text analysis functionality by hitting the provided endpoint. This eliminates the need for users to subscribe to Microsoft Azure or manage their API keys, as the backend integration and key management are handled within the system.

2. Prerequisites

Before starting this project, ensure the following prerequisites are met:

- Node.js installed on your system.
- Azure account and subscription.
- Azure Vision API Key and Endpoint.
- Basic knowledge of JavaScript and Express.js.

3. Project Setup

1. Clone the repository or create a new Node.js project.

2. Install dependencies using npm:

```
npm install express body-parser multer dotenv @azure-rest/ai-vision-image-analysis@azure/core-auth
```

3. Set up a `.env` file with the following keys:

VISION_ENDPOINT=<Your Azure Vision API Endpoint>

VISION_KEY=<Your Azure Vision API Key>

---

4. Code Explanation

This section breaks down the major components of the code.

Middleware Setup

Middleware like body-parser is used to parse incoming JSON payloads. Multer is configured for file uploads with restrictions on file size and type.

```
app.use(bodyParser.json());

const upload = multer({
  dest: 'uploads/',
  fileFilter: (req, file, cb) => {
    const allowedTypes = ['image/jpeg', 'image/png'];
    if (!allowedTypes.includes(file.mimetype)) {
      return cb(new Error('Only JPEG and PNG images are allowed'), false);
    }
    cb(null, true);
  },
  limits: { fileSize: 5 * 1024 * 1024 },
});
```

Azure Vision Client

The Azure Vision client is created using the `@azure-rest/ai-vision-image-analysis` library. The API key is securely loaded from environment variables

```
const endpoint = process.env['VISION_ENDPOINT'];
const key = process.env['VISION_KEY'];
const credential = new AzureKeyCredential(key);
const client = createClient(endpoint, credential);
```

Endpoints

/api/analyze-url: Accepts an image URL in the request body and processes it using the Azure Vision API. (http://159.223.172.118:3000/api/analyze-url)

```
app.post('/api/analyze-url', async (req, res) => {
 const { imageUrl } = req.body;

 if (!imageUrl) {
  return res.status(400).json({ error: 'Image URL is required' });
 }

 try {
  const result = await client.path('/imageanalysis:analyze').post({
   body: { url: imageUrl },
   queryParameters: { features: features },
   contentType: 'application/json',
  });

  const iaResult = result.body;

  const caption = iaResult.captionResult ? iaResult.captionResult.text : null;
  const content = iaResult.readResult
   ? iaResult.readResult.blocks.flatMap(block =>
      block.lines.map(line => line.text)
    )
   : [];

  const response = {
   caption,
   content,
  };
```

```
  res.status(200).json(response);
 } catch (error) {
  console.error(error);
  res.status(500).json({ error: 'Failed to analyze image', details: error.message });
 }
});
```

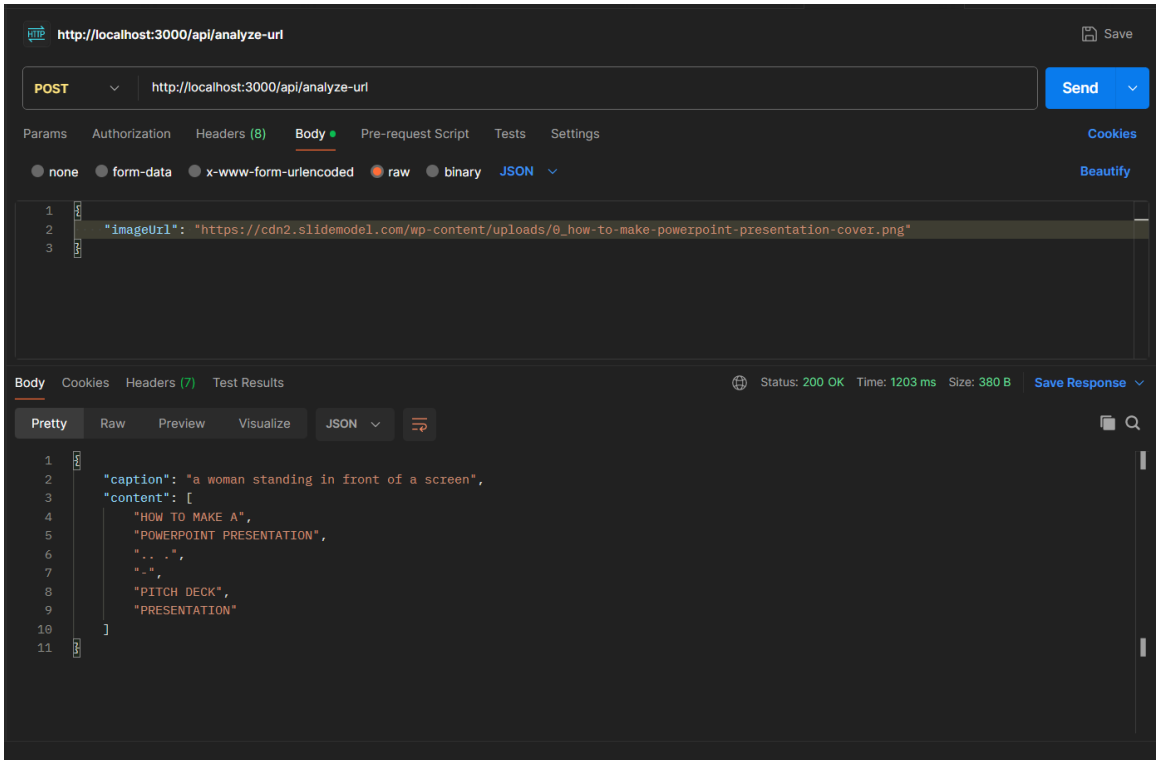
/api/analyze-upload: Accepts an image file via a multipart request and processes it. (http://159.223.172.118:3000/api/analyze-upload)

```
app.post('/api/analyze-upload', upload.single('image'), async (req, res) => {
 if (!req.file) {
  return res.status(400).json({ error: 'Image file is required' });
 }

 const filePath = path.join(__dirname, req.file.path);

 try {
  const imageData = fs.readFileSync(filePath);

  const result = await client.path('/imageanalysis:analyze').post({
   body: imageData,
   queryParameters: { features: features },
   headers: {
    'Content-Type': 'application/octet-stream',
   },
  });

  const iaResult = result.body;

  const caption = iaResult.captionResult ? iaResult.captionResult.text : null;
  const content = iaResult.readResult
   ? iaResult.readResult.blocks.flatMap(block =>
     block.lines.map(line => line.text)
    )
   : [];

  const response = {
   caption,
   content,
  };
```

```
  res.status(200).json(response);
 } catch (error) {
  console.error(error);
  res.status(500).json({ error: 'Failed to analyze image', details: error.message });
 } finally {
  // Clean up uploaded file
  fs.unlinkSync(filePath);
 }
});
```

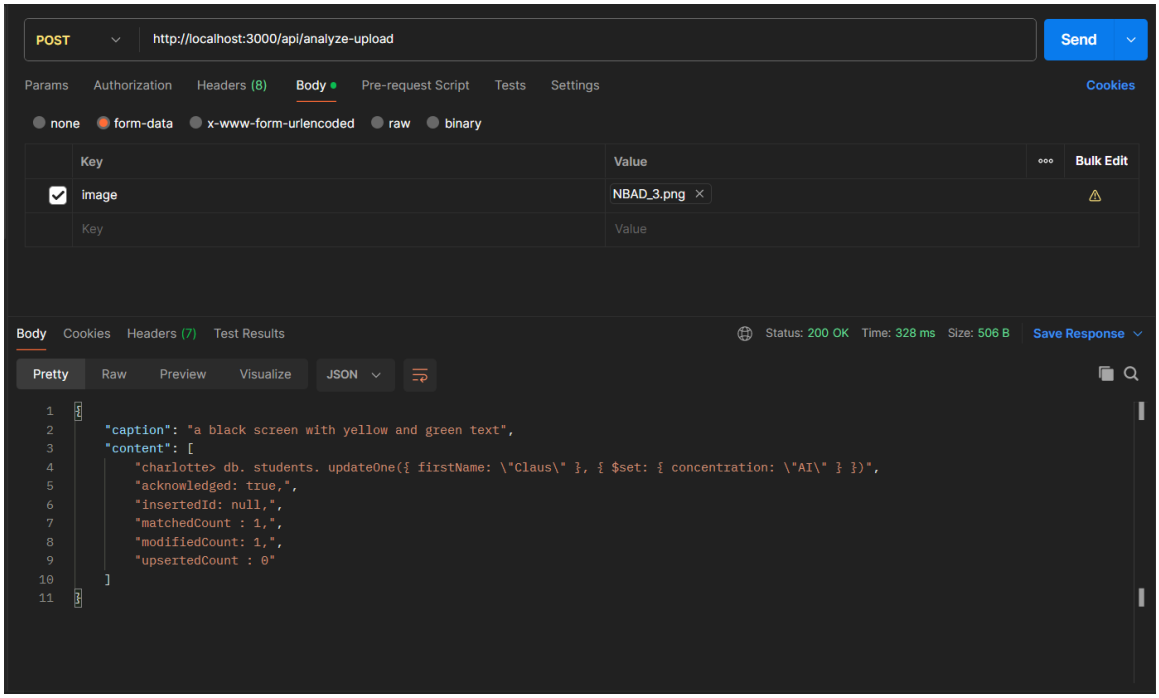
5. Features

1. **Analyze Images via URL**: Upload an image URL for analysis.

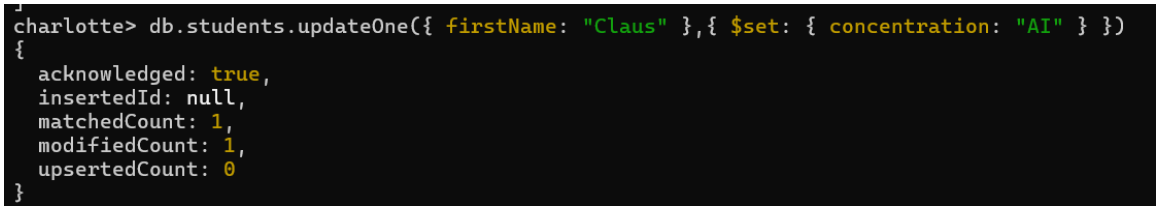


The above figure shows the response of the text extracted from the [image]

2. **Analyze Uploaded Files**: Upload a file directly for OCR.



The above figure shows the response of a screenshot taken from a command prompt shown below



Screenshot taken from command prompt

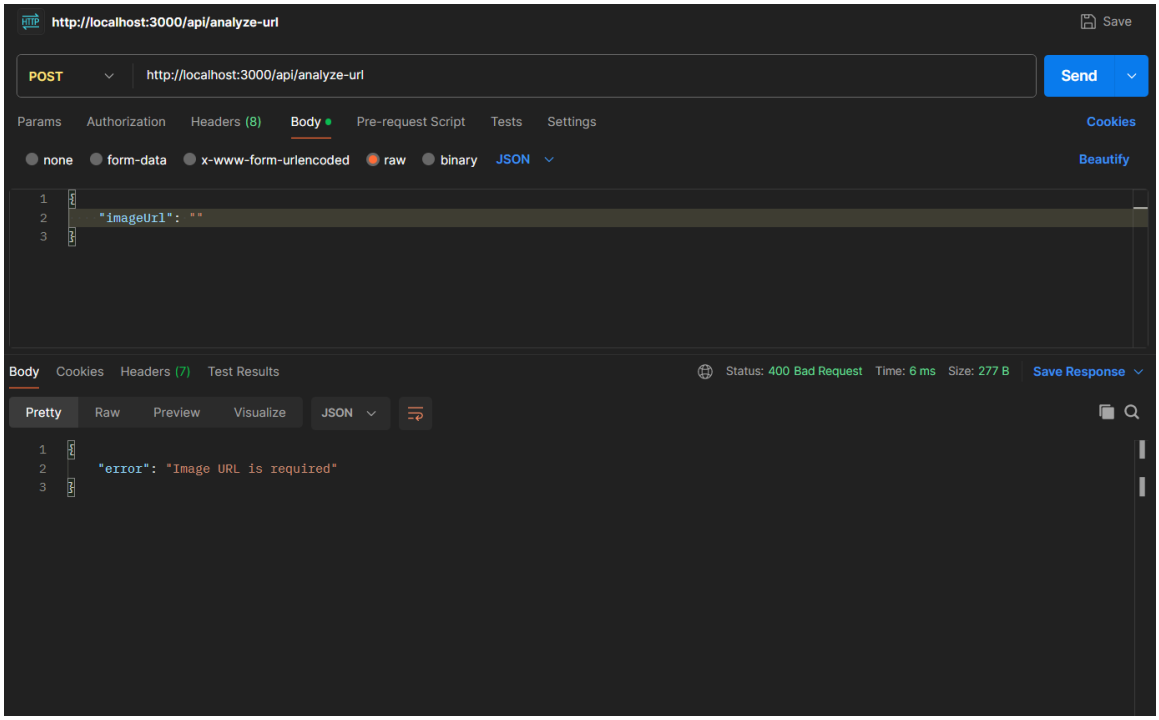
3. **Secure API Key Management**: API keys are stored securely.

6. Error Handling

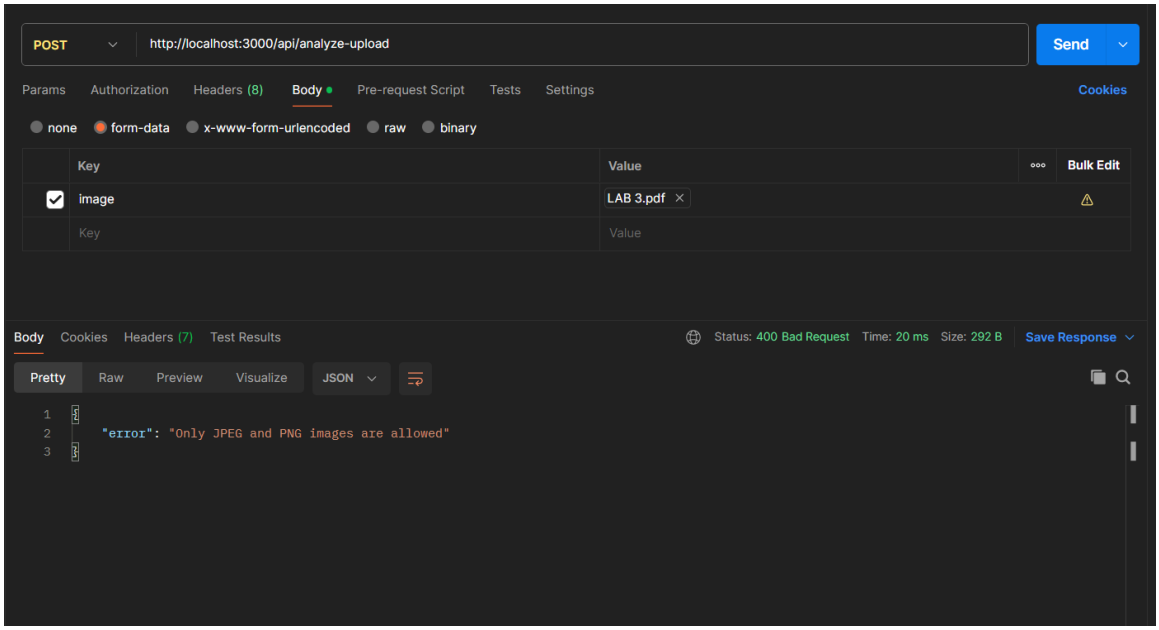
Errors are handled gracefully with appropriate status codes and detailed messages.

Examples include missing image URLs, invalid file types, and internal server errors.

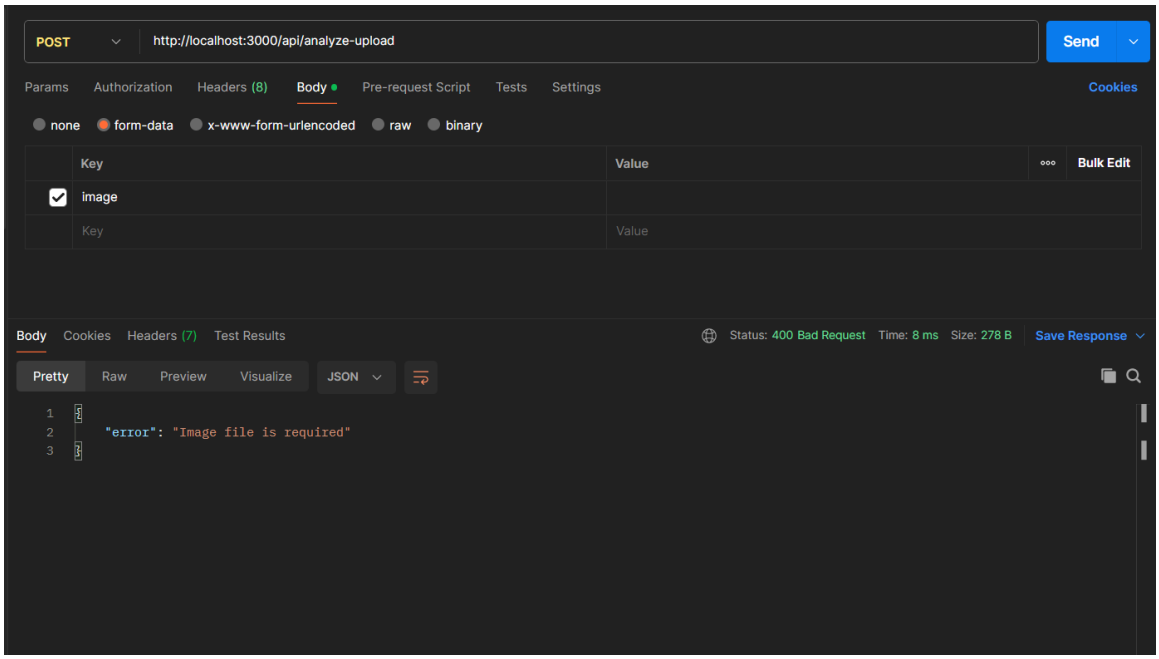
```
app.use((err, req, res, next) => {
  if (err instanceof multer.MulterError) {
    res.status(400).json({ error: `Multer Error: ${err.message}` });
  } else if (err.message === 'Only JPEG and PNG images are allowed') {
    res.status(400).json({ error: err.message });
  } else {
    res.status(500).json({ error: 'An unexpected error occurred', details:
err.message });
  }
});
```



*The above figure shows error message if **imageUrl** is not provided*



The above figure shows the error if incorrect format of image is provided



The above figure shows the error if image is not uploaded

7. Security Considerations

1. API keys are stored in environment variables to prevent exposure.
2. Uploaded files are validated for type and size before processing.

8. API Documentation

Endpoint: /api/analyze-url

- Method: POST
- Input: JSON body with an `imageUrl` field.
- Output: Caption and extracted text content.

Endpoint: /api/analyze-upload

- Method: POST
- Input: Multipart file upload with an image.
- Output: Caption and extracted text content.

9. Conclusion

This project demonstrates the integration of the Azure AI OCR API with a Node.js backend. By following security and coding best practices, the project serves as a robust foundation for further development and real-world applications.

Detailed Code Explanation

This section provides a detailed explanation of each line of code used in the Azure AI OCR API project. Each block of code is described to help developers understand its purpose and implementation.

Code Setup

```
const express = require('express');
```

The `express` module is imported to set up the web server. Express simplifies the process of creating server-side applications with Node.js.

```
const bodyParser = require('body-parser');
```

The `body-parser` middleware is used to parse incoming JSON requests and make the data available in the `req.body` object.

```
const { ImageAnalysisClient } = require('@azure-rest/ai-vision-image-analysis');
const createClient = require('@azure-rest/ai-vision-image-analysis').default;
const { AzureKeyCredential } = require('@azure/core-auth');
```

These lines import the Azure AI Vision API client library, including tools for creating the client and managing API credentials securely.

Environment Variables

```
require('dotenv').config();
```

This line loads environment variables from a `.env` file, ensuring sensitive data like the API key and endpoint are not hardcoded.

Middleware Configuration

```
app.use(bodyParser.json());
```

The `bodyParser.json()` middleware ensures incoming JSON requests are parsed into JavaScript objects.

Azure Vision Client Initialization

```
const endpoint = process.env['VISION_ENDPOINT'];
const key = process.env['VISION_KEY'];
const credential = new AzureKeyCredential(key);
const client = createClient(endpoint, credential);
```

These lines set up the Azure Vision client using the endpoint and key stored in environment variables. The `AzureKeyCredential` securely manages the API key, and the `createClient` function initializes the client with the specified endpoint.

Endpoint: Analyze Image via URL

```
app.post('/api/analyze-url', async (req, res) => {
  const { imageUrl } = req.body;
  if (!imageUrl) {
    return res.status(400).json({ error: 'Image URL is required' });
  }
  ...
});
```

This endpoint accepts an image URL via a POST request. If the `imageUrl` is missing, it returns a `400 Bad Request` error.

```
const result = await client.path('/imageanalysis:analyze').post({
  body: { url: imageUrl },
  queryParameters: { features: features },
  contentType: 'application/json',
});
```

The Azure Vision API is called with the image URL. The `features` parameter specifies the desired analysis tasks (e.g., Caption, Read).

```
const iaResult = result.body;
const caption = iaResult.captionResult ? iaResult.captionResult.text : null;
const content = iaResult.readResult
  ? iaResult.readResult.blocks.flatMap(block => block.lines.map(line => line.text))
  : [];
const response = { caption, content };
res.status(200).json(response);
```

The API response is processed to extract the caption and text content. The results are returned as a JSON object.

```
fs.unlinkSync(filePath);
```

After the analysis is complete, the temporary file is deleted to conserve storage and enhance security.

NOTE:

THE ABOVE SCREENSHOTS PROVIDED ARE RUN AND TESTED ON **LOCALHOST.**
THE CODE IS HOSTED ON DIGITAL OCEAN SERVER AND THE APIS CAN BE TESTED BY USING THE BELOW ENDPOINTS

1. http://159.223.172.118:3000/api/analyze-url

2. http://159.223.172.118:3000/api/analyze-upload

REFERENCES

Quick Start Image Analysis

(https://learn.microsoft.com/en-us/azure/ai-services/computer-vision/quickstarts-sdk/image-analysis-client-library-40?tabs=visual-studio%2Cwindows&pivots=programming-language-javascript#prerequisites)

Azure AI VISION Studio

(https://portal.vision.cognitive.azure.com/demo/extract-text-from-images)