1. Introduction

This project demonstrates how to integrate Microsoft Azure's Speech Services into a Node.js/Express web application. By connecting to Azure's APIs, the application can list available voices for text-to-speech (TTS), convert both plain text and SSML (Speech Synthesis Markup Language) into spoken audio, and transcribe uploaded speech audio files into text. The core purpose is to simplify the interaction between client-facing interfaces and Azure's robust speech capabilities, enabling developers to create voice-enabled applications efficiently.

2. Prerequisites & Setup

To use this project, you will need:

- Node.js (Version 14 or later recommended) installed on your machine.
- NPM (installed automatically with Node.js).
- An Azure Account and a provisioned Speech resource. You must have a valid AZURE_SPEECH_API_KEY and AZURE_SPEECH_REGION, which are provided through the Azure Portal.
  After obtaining your credentials, you will place them in a .env file to ensure they remain secure and out of version control.

3. Project Structure

The project is organized as follows:

- **src/controllers/**: Contains controller files like speechController.js which handle request logic and responses.
- **src/services/**: Includes azureClient.js that communicates with Azure APIs.
- **src/routes/**: Defines Express routes mapping HTTP endpoints to controller actions.
- **src/utils/**: Contains utilities such as configuration handlers and logging.
- **docs/**: Stores OpenAPI (Swagger) files for API documentation.
- **.env**: Holds environment variables (not committed to Git).
- **package.json**: Lists project dependencies and scripts.

4. Installation & Configuration

1. Clone the repository from GitHub.
2. Inside the project directory, run:

npm install

to install all dependencies.

3. Create a .env file in the project's root directory and add your Azure Speech keys:

makefile

AZURE_SPEECH_API_KEY=your_speech_key_here

AZURE_SPEECH_REGION=your_region_here

PORT=3000

4. Make sure not to commit the .env file to source control.

5. Environment Variables

The .env file is crucial for configuring runtime settings without hardcoding sensitive information.

- AZURE_SPEECH_API_KEY: Your Azure Speech API key.
- AZURE_SPEECH_REGION: The region where your Speech resource is located (e.g., eastus).
- PORT: The port the Express server will listen on.
  This approach keeps secrets out of the repository's history and enables easily changing configurations between environments.

6. API Endpoints Documentation

The application provides the following primary endpoints:

- **GET /voices**: Retrieves a list of available TTS voices from Azure, including language, gender, and voice styles.
- **POST /synthesize/text**: Accepts plain text and converts it into a spoken audio file (WAV).
- **POST /synthesize/ssml**: Accepts SSML input, allowing you to control speech characteristics (e.g., breaks, pitch) for a more natural output.
- **POST /speech-to-text**: Accepts an audio file upload and returns the transcribed text.

For each endpoint, you'll find details on request bodies, example requests, and responses in the API documentation sections or by using the integrated Swagger UI.

7. Using the Application

Once your .env is set up, and dependencies are installed, run:

Sql

npm start

The server starts at http://localhost:3000 (or another port specified in .env).

- To test /voices, simply open http://localhost:3000/voices in a browser or use a tool like cURL or Postman.
- To test TTS or SSML, send a POST request with the required JSON payload.
- For speech-to-text, upload an audio file using multipart/form-data. Realistic request examples are provided in separate examples files or in this documentation's appendices.

8. Swagger Documentation
Swagger UI is integrated into the project, offering a browser-based interface to view and test endpoints.

- Access Swagger at http://localhost:3000/api-docs.
- You can interact with each endpoint, review parameters, and see sample responses directly in the browser without coding manual requests.

9. Deployment to Production
To host on a server (e.g., a DigitalOcean Droplet):

1. Install Node.js and clone the repository.
2. Configure .env with production keys.
3. Use pm2 or a similar process manager to ensure the app runs continuously.
4. Set up a reverse proxy (e.g., Nginx) for production-grade stability and SSL termination if desired.

10. Maintenance & Monitoring
In a production environment, you can:

- Monitor the app with pm2 logs to review runtime logs.
- Use PM2's restart and scaling features to handle increased load.
- Update environment variables or rotate keys as security best practices.

11. Troubleshooting & Common Issues

- If no voices are returned, ensure the AZURE_SPEECH_REGION and AZURE_SPEECH_API_KEY are correct.
- If speech synthesis fails, verify the key is valid and that the text or SSML is formatted correctly.
- For speech-to-text issues, confirm the uploaded audio file is in a supported format (WAV PCM 16kHz is recommended).

12. Additional Resources

- Azure Speech Documentation: https://docs.microsoft.com/azure/cognitive-services/speech-service
- OpenAPI/Swagger Documentation: https://swagger.io

Below is a detailed guide for using our APIs with Postman. This includes information on the input values, request body types, and various options you can experiment with, especially for the synthesize endpoints. You can use this to fine-tune your requests and understand the full flexibility of the system.

General Postman Usage

1. **Install and Open Postman:**
   If you haven't already, download Postman from https://www.postman.com/downloads/.
   Once installed, open Postman.
2. **Set the Base URL:**
   If your application runs locally on port 3000, your base URL is: http://localhost:3000
   If deployed remotely, use your server's domain or IP and port.
3. **Setting Up Requests in Postman:**
   - Use the **Method** dropdown to select GET or POST.
   - For POST requests dealing with JSON, set the "Body" tab to "raw" and "JSON" format.
   - For file uploads, select "form-data" in the Body tab and choose "File" as the type for the appropriate field.

GET /voices

Description: Retrieves a list of all available TTS voices.

How to Test in Postman:

- **Method:** GET
- **URL:** http://localhost:3000/voices
- **Headers:** None required.

Steps:

1. Click on “+” to create a new request in Postman.
2. Set method to GET.
3. Enter the URL http://localhost:3000/voices.
4. Click “Send”.

Expected Response:
A JSON object containing an array of voices, e.g.:

json


```
{
  "voices": [
    {
      "Name": "en-US-AriaNeural",
      "ShortName": "en-US-AriaNeural",
      "Locale": "en-US",
      "Gender": "Female",
      "VoiceType": "Neural",
      "StyleList": ["chat", "customerservice"]
    },
    ...
  ]
}
```

POST /synthesize/text

Description: Converts plain text into spoken audio (WAV).

Key Parameters:

- **text (string)**: The text you want to convert into speech.
- **voiceName (string, optional)**: Choose from the voices listed in /voices. If omitted, a default voice is used.
- **language (string, optional)**: The language/locale code (e.g., en-US, fr-FR). Matches the chosen voice's locale.

Content Type: application/json

How to Test in Postman:

1. Create a new request tab.
2. Set method to POST.
3. URL: http://localhost:3000/synthesize/text
4. Under "Headers", ensure Content-Type is application/json.
5. Go to the "Body" tab, select "raw" and then choose "JSON" from the dropdown.
6. Enter a JSON body like this:

json

```
{
  "text": "Hello, this is a test.",
  "voiceName": "en-US-AriaNeural",
  "language": "en-US"
}
```

7. Click "Send".

Response:
You'll receive a binary WAV file. Postman may show this as binary data. Use "Send and Download" or "Save Response" to store it locally. Then, open it in an audio player to listen.

Alternative Inputs:

- **Different Languages:** If you want French, for instance:

json

```
{
  "text": "Bonjour, ceci est un test en français.",
  "voiceName": "fr-FR-DeniseNeural",
  "language": "fr-FR"
}
```

- **Longer Text:**
  Provide a longer paragraph to test more natural speech:

json

```
{
  "text": "Welcome to our advanced testing scenario. We hope you enjoy the richness of neural voices and their ability to convey emotion and emphasis.",
  "voiceName": "en-US-AriaNeural",
  "language": "en-US"
}
```

Experimentation:

- Try omitting voiceName and language to see if it defaults properly.
- Use voices from /voices that have style variations. Some voices support additional tags or styles (e.g., “chat”, “newscast”). Although styles are more relevant in SSML, you can still pick different voices here and observe differences.

POST /synthesize/ssml

Description: Uses SSML input to produce speech audio. SSML lets you modify pitch, speed, breaks, emphasis, etc.

Key Parameters:

- **ssml (string)**: An SSML string that defines how the speech is pronounced.
- **voiceName (string, optional)**: Specific voice to use.
- **language (string, optional)**: Locale for the SSML content.

Content Type: application/json

How to Test in Postman:

1. Create a new POST request.
2. URL: http://localhost:3000/synthesize/ssml
3. Headers: Content-Type: application/json
4. Body: “raw” + “JSON”
5. Example body:

json

```
{
  "ssml": "<speak version='1.0' xml:lang='en-US'><voice name='en-US-AriaNeural'>Hello, <break time='1s'/> how are you today?</voice></speak>",
  "voiceName": "en-US-AriaNeural",
  "language": "en-US"
}
```

6. Click “Send” and download the resulting audio.

Alternative SSML Inputs:

- **Adjusting Rate and Pitch:**

json

{

"ssml": "<speak version='1.0' xml:lang='en-US'><voice name='en-US-AriaNeural'><prosody rate='slow' pitch='+2st'>This is slower speech with a higher pitch.</prosody></voice></speak>",

"voiceName": "en-US-AriaNeural",

"language": "en-US"

}

- **Emphasis and Pauses:**

Json

{

"ssml": "<speak version='1.0' xml:lang='en-US'><voice name='en-US-AriaNeural'>We can add <emphasis level='moderate'>emphasis</emphasis> on words and include <break time='500ms'/>short pauses.</voice></speak>",

"voiceName": "en-US-AriaNeural",

"language": "en-US"

}

- **International Languages:**

Json

{

"ssml": "<speak version='1.0' xml:lang='fr-FR'><voice name='fr-FR-DeniseNeural'><prosody rate='fast'>Bonjour, ceci est un test rapide en français.</prosody></voice></speak>",

"voiceName": "fr-FR-DeniseNeural",

"language": "fr-FR"

}

Experimentation:

- Modify <break time='Xms'/> values to introduce different pause lengths.
- Use <prosody> tags to alter pitch, rate, or volume.
- Add <emphasis> tags around key words.
- Combine multiple SSML elements to create highly customized speech patterns.

POST /speech-to-text

Description: Transcribes uploaded audio (WAV recommended) into text.

Key Parameters:

- Form-data upload with a field named audioFile.

How to Test in Postman:

1. Create a new POST request.
2. URL: http://localhost:3000/speech-to-text
3. In “Body”, select “form-data”.
4. For Key, type audioFile. Change “Text” to “File” on the right.
5. Click “Select File” and choose a WAV audio file from your computer.
6. Click “Send”.

Expected Response: A JSON object containing the transcript, for example:

Json

```
{
  "transcript": "Hello world, this is a transcribed audio."
}
```

Alternative Inputs:

- Try different audio clips with varying accents, languages, or clarity.

- Test audio recorded at different sampling rates (16kHz PCM WAV is recommended for best results).
- Experiment with shorter vs. longer audio files. Short, clear speech should produce more accurate results.