# Voice API Integration

This document provides an overview of the Voice API integration for the PSScript Manager platform. The Voice API enables voice input and output capabilities, enhancing the user experience and accessibility of the platform.

## Table of Contents



## Overview

The Voice API integration adds voice capabilities to the PSScript Manager platform, allowing users to:

- Convert text to speech (voice synthesis)
- Convert speech to text (voice recognition)
- Interact with the platform using voice commands
- Receive voice responses to queries

The implementation supports multiple voice service providers (Google Cloud, Amazon AWS, Microsoft Azure) with fallback mechanisms and caching for improved performance and reliability.

## Architecture

The Voice API integration follows a layered architecture:

1. **AI Service Layer**:
2. `voice_service.py`: Core service with speech synthesis and recognition capabilities
3. `voice_endpoints.py`: FastAPI endpoints for voice processing
4. `voice_agent.py`: Specialized agent for voice tasks
5. **Backend Layer**:
6. `voiceController.js`: Express controller for voice functionality
7. `voiceRoutes.js`: API routes for voice endpoints
8. **Frontend Layer**:
9. `VoiceRecorder.jsx`: Audio recording with visualization
10. `VoicePlayback.jsx`: Audio playback with controls
11. `VoiceChatInterface.jsx`: Chat UI with voice integration
12. `VoiceSettings.jsx`: User preferences for voice features

## Features

- **Multi-provider Support**: Integration with Google Cloud, Amazon AWS, and Microsoft Azure voice services
- **Fallback Mechanism**: Automatic fallback to alternative providers if the primary provider fails
- **Caching**: Efficient caching of voice responses to reduce API calls and improve performance
- **Voice Settings**: User-configurable voice settings (voice selection, playback options)
- **Accessibility**: Enhanced accessibility through voice interaction
- **Visualization**: Audio waveform visualization during recording and playback
- **Error Handling**: Comprehensive error handling and recovery mechanisms

## Setup and Configuration

### Prerequisites

- Node.js 14+ for the backend
- Python 3.8+ for the AI service
- API keys for the voice service providers:
- Google Cloud Speech-to-Text and Text-to-Speech
- Amazon Polly and Transcribe
- Microsoft Azure Cognitive Services

### Installation

1. **AI Service Setup**: `bash cd src/ai pip install -r requirements.txt`
2. **Backend Setup**: `bash cd src/backend npm install`
3. **Environment Variables**:

Create a `.env` file in the `src/ai` directory with the following variables: ``` VOICE_API_KEY=your-api-key TTS_SERVICE=google # google, amazon, or microsoft STT_SERVICE=google # google, amazon, or microsoft TTS_CACHE_DIR=voice_cache TTS_CACHE_TTL=86400 # 24 hours in seconds

# Google Cloud credentials GOOGLE_APPLICATION_CREDENTIALS=path/to/google-credentials.json

# Amazon AWS credentials AWS_ACCESS_KEY_ID=your-aws-access-key AWS_SECRET_ACCESS_KEY=your-aws-secret-key AWS_REGION=us-east-1

# Microsoft Azure credentials AZURE_SPEECH_KEY=your-azure-speech-key AZURE_SPEECH_REGION=eastus ```

### Starting the Services

1. **Start the AI Service**: `bash cd src/ai uvicorn main:app --reload --port 8000`
2. **Start the Backend**: `bash cd src/backend npm run dev`

## API Reference

## Backend API Endpoints

### Voice Synthesis

```
POST /api/voice/synthesize
```

Request body:

```
{
  "text": "Text to synthesize",
  "voiceId": "en-US-Standard-A",
  "outputFormat": "mp3"
}
```

Response:

```
{
  "audio_data": "base64-encoded-audio-data",
  "format": "mp3",
  "duration": 2.5,
  "text": "Text to synthesize"
}
```

### Voice Recognition

```
POST /api/voice/recognize
```

Request body:

```
{
  "audioData": "base64-encoded-audio-data",
  "language": "en-US"
}
```

Response:

```
{
  "text": "Recognized text",
  "confidence": 0.92,
  "alternatives": [
    {
      "text": "Recognized text",
      "confidence": 0.92
    },
    {
      "text": "Alternative text",
      "confidence": 0.85
    }
  ]
}
```

**Voice Settings**

```
GET /api/voice/settings
```

Response:

```
{
  "voiceId": "en-US-Standard-A",
  "autoPlay": true,
  "volume": 0.8,
  "speed": 1.0
}
```

```
PUT /api/voice/settings
```

Request body:

```
{
  "voiceId": "en-US-Standard-A",
  "autoPlay": true,
```

```
  "volume": 0.8,
  "speed": 1.0
}
```

#### Available Voices

```
GET /api/voice/voices
```

Response:

```
{
  "voices": [
    {
      "id": "en-US-Standard-A",
      "name": "English US (Female)",
      "language": "en-US",
      "gender": "female"
    },
    {
      "id": "en-US-Standard-B",
      "name": "English US (Male)",
      "language": "en-US",
      "gender": "male"
    }
  ]
}
```

### AI Service Endpoints

#### Voice Synthesis

```
POST /voice/synthesize
```

Request body:

```
{
  "text": "Text to synthesize",
  "voice_id": "en-US-Standard-A",
  "output_format": "mp3"
}
```

### Voice Recognition

```
POST /voice/recognize
```

Request body:

```
{
  "audio_data": "base64-encoded-audio-data",
  "language": "en-US"
}
```

## Usage Examples

### Frontend Integration

```
 import React, { useState } from 'react';
import VoiceRecorder from '../components/VoiceRecorder';
import VoicePlayback from '../components/VoicePlayback';

const VoiceExample = () => {
  const [recognizedText, setRecognizedText] = useState('');
  const [audioData, setAudioData] = useState(null);

  const handleVoiceInput = (text, audio) => {
    setRecognizedText(text);
    setAudioData(audio);
  };

  return (
```

```
    <div className="voice-example">
      <h2>Voice Example</h2>

      <VoiceRecorder onAudioCaptured={handleVoiceInput} />

      {recognizedText && (
        <div className="recognized-text">
          <h3>Recognized Text:</h3>
          <p>{recognizedText}</p>
        </div>
      )}

      {audioData && (
        <div className="playback">
          <h3>Playback:</h3>
          <VoicePlayback audioData={audioData} autoPlay={true} />
        </div>
      )}
    </div>
  );
};
```

### Backend Integration

```
const voiceController = require('./controllers/voiceController');

// Synthesize speech
app.post('/api/custom/speak', async (req, res) => {
  try {
    const { text } = req.body;

    // Call the voice controller
    const result = await voiceController.synthesizeSpeech({
      body: {
        text,
        voiceId: 'en-US-Standard-A',
        outputFormat: 'mp3'
      }
```

```
    }, res);

    // The result is already sent by the controller
  } catch (error) {
    console.error('Error in custom speak endpoint:', error);
    res.status(500).json({ error: 'Internal server error' });
  }
});
```

## Testing

The Voice API integration includes a comprehensive test script (`test-voice-api.sh`) that tests all aspects of the integration:

```
# Make the script executable
chmod +x test-voice-api.sh

# Run the tests
./test-voice-api.sh
```

The test script performs the following tests:

1. Get available voices
2. Get voice settings
3. Update voice settings
4. Synthesize speech
5. Recognize speech
6. End-to-end test (synthesize then recognize)
7. Performance test with multiple languages
8. Stress test with long text
9. Direct AI service test
10. Cache test

## Troubleshooting

## Common Issues

1. **Authentication Errors**:
2. Ensure that the API keys for the voice service providers are correctly set in the environment variables.
3. Check that the authentication middleware is correctly configured.
4. **Voice Service Errors**:
5. Verify that the selected voice service provider is available and properly configured.
6. Check the logs for specific error messages from the voice service provider.
7. **Audio Format Issues**:
8. Ensure that the audio format specified in the request is supported by the voice service provider.
9. Check that the audio data is correctly encoded in base64 format.
10. **Performance Issues**:
11. Verify that the caching mechanism is working correctly.
12. Check the network latency between the backend and the voice service provider.

## Logs and Debugging

- AI Service logs: Check the console output of the AI service for error messages.
- Backend logs: Check the backend logs for API request/response details.
- Browser console: Check the browser console for frontend errors.

## Support

For additional support, please contact the development team or refer to the documentation of the specific voice service provider being used.