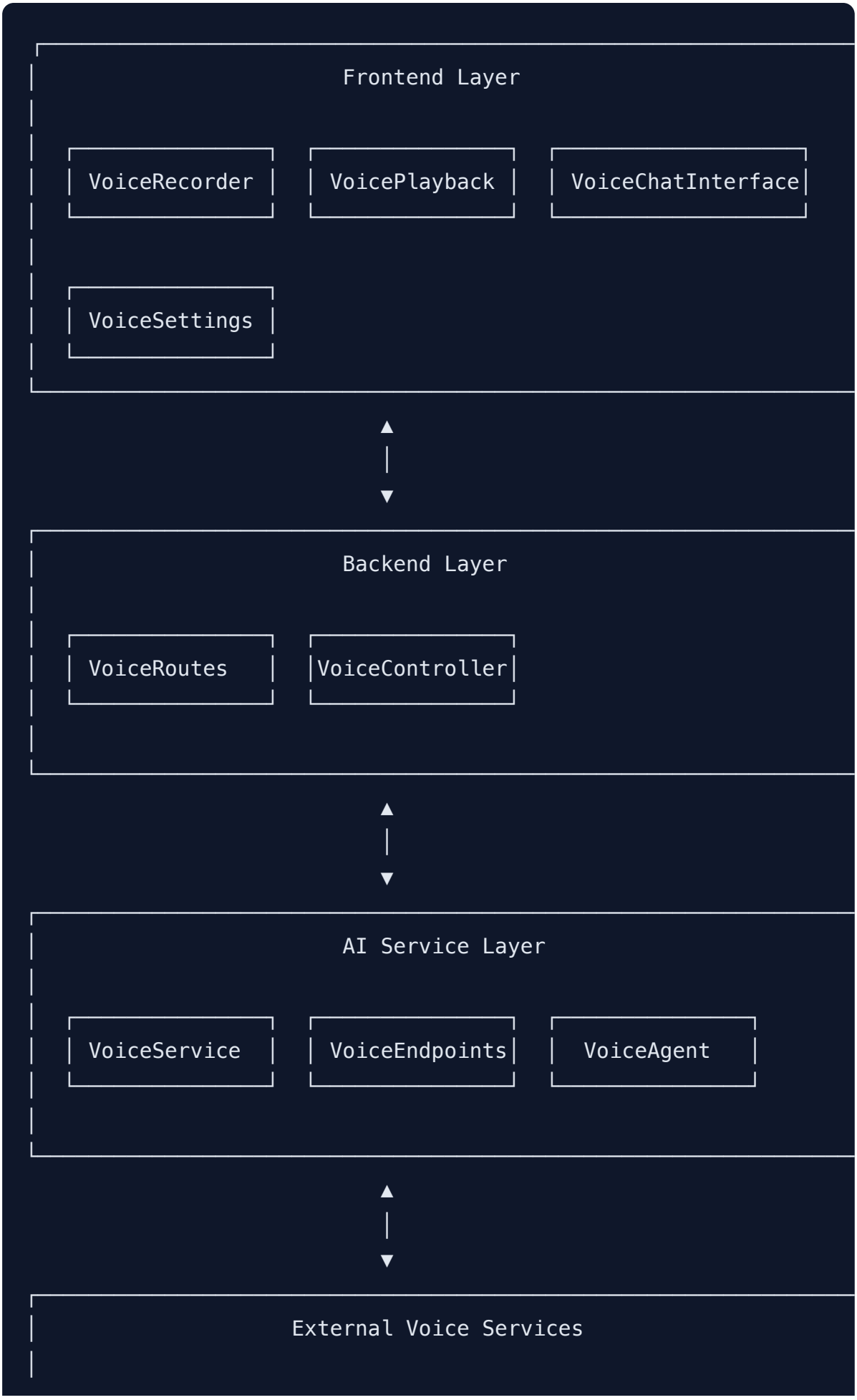
Voice API Architecture

This document provides a detailed overview of the architecture for the Voice API integration in the PSScript Manager platform.

Table of Contents

System Architecture

The Voice API integration follows a layered architecture with clear separation of concerns:


Frontend Layer
VoiceRecorder
VoicePlayback
VoiceChatInterface
VoiceSettings
Backend Layer
VoiceRoutes
VoiceController
AI Service Layer
VoiceService
VoiceEndpoints
VoiceAgent
External Voice Services

1. Frontend Layer

The frontend layer provides the user interface components for voice interaction:

- **VoiceRecorder**: Handles audio recording with visualization
- **VoicePlayback**: Manages audio playback with controls
- **VoiceChatInterface**: Integrates voice capabilities into the chat interface
- **VoiceSettings**: Provides user preferences for voice features

These components are implemented as React components with responsive design and accessibility features.

2. Backend Layer

The backend layer provides the API endpoints for voice functionality:

- **VoiceRoutes**: Defines the API routes for voice endpoints
- **VoiceController**: Implements the logic for handling voice requests

The backend layer is implemented using Express.js and communicates with the AI service layer for voice processing.

3. AI Service Layer

The AI service layer provides the core voice processing capabilities:

- **VoiceService**: Implements the core voice synthesis and recognition functionality
- **VoiceEndpoints**: Defines the FastAPI endpoints for the voice service
- **VoiceAgent**: Specialized agent for handling voice-related tasks

The AI service layer is implemented using Python with FastAPI and integrates with external voice service providers.

4. External Voice Services

The external voice services provide the actual voice processing capabilities:

- **Google Cloud**: Speech-to-Text and Text-to-Speech services
- **Amazon AWS**: Polly and Transcribe services
- **Microsoft Azure**: Cognitive Services for speech

Component Interactions

Voice Synthesis Flow

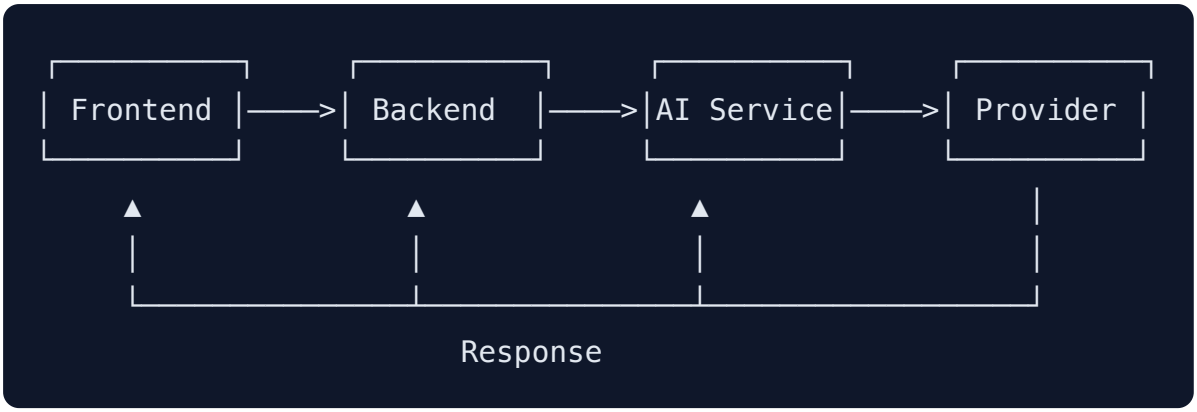
1. User requests voice synthesis through the frontend
2. Frontend sends a request to the backend API
3. Backend controller forwards the request to the AI service
4. AI service checks the cache for a matching request
5. If found in cache, returns the cached result
6. If not found, selects the appropriate voice service provider
7. Sends the request to the selected provider
8. Receives the synthesized speech from the provider
9. Caches the result for future use
10. Returns the result to the backend
11. Backend returns the result to the frontend
12. Frontend plays the synthesized speech

Voice Recognition Flow

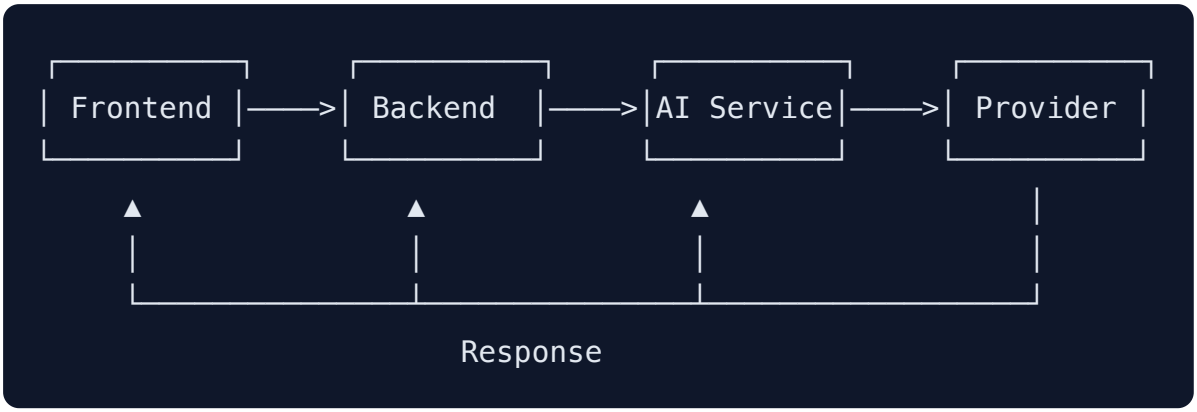
1. User records audio through the frontend
2. Frontend sends the audio data to the backend API
3. Backend controller forwards the request to the AI service
4. AI service selects the appropriate voice service provider
5. Sends the audio data to the selected provider
6. Receives the recognized text from the provider
7. Returns the result to the backend
8. Backend returns the result to the frontend
9. Frontend displays the recognized text

Data Flow

Voice Synthesis Data Flow



Voice Recognition Data Flow



Security Considerations

Authentication and Authorization

- **API Authentication**: All API endpoints require authentication using JWT tokens
- **Service Authentication**: Communication with external voice service providers uses API keys
- **Authorization**: Access to voice features is controlled by user permissions

Data Protection

- **Encryption**: Voice data is encrypted in transit using HTTPS
- **Storage**: Cached voice data is stored securely with appropriate access controls
- **Retention**: Voice data is retained only for the necessary duration

Privacy

- **User Consent**: Users must provide consent for voice recording
- **Data Minimization**: Only necessary data is collected and processed
- **Transparency**: Users are informed about how their voice data is used

Performance Optimizations

Caching

The Voice API implements a multi-level caching strategy:

1. **Memory Cache**: Fast in-memory cache for frequently accessed voice data
2. **Disk Cache**: Persistent cache for larger voice data with longer TTL
3. **Cache Invalidation**: Time-based and LRU-based cache invalidation

Asynchronous Processing

- Voice processing operations are performed asynchronously to avoid blocking
- Long-running operations use background processing

Resource Pooling

- Connection pooling for external service providers
- Resource reuse for improved performance

Error Handling and Fallbacks

Error Detection

- Comprehensive error detection for all voice operations
- Detailed error logging for troubleshooting

Fallback Mechanisms

- Provider Fallback: If the primary provider fails, fallback to alternative providers
- Graceful Degradation: If voice features are unavailable, fallback to text-based interaction

Retry Logic

- Automatic retry for transient errors
- Exponential backoff for repeated failures

Extensibility

Adding New Providers

The Voice API is designed to be easily extended with new voice service providers:

1. Implement the provider-specific methods in the VoiceService class
2. Add the provider to the provider selection logic
3. Update the configuration to include the new provider

Adding New Features

The modular architecture allows for easy addition of new features:

1. Add new endpoints to the VoiceEndpoints class
2. Implement the corresponding methods in the VoiceService class
3. Add new routes to the VoiceRoutes module
4. Implement the corresponding methods in the VoiceController class
5. Add new components to the frontend as needed

Customization

The Voice API supports customization through:

- User preferences for voice selection
- Configuration options for voice processing
- Extensible component architecture

Generated 2026-01-16 21:23 UTC