Health Transcribe: Application Report

Overview

Health Transcribe is a real-time multilingual translation application designed for healthcare settings. It allows healthcare providers and patients to overcome language barriers by providing immediate transcription, enhancement of medical terminology, and translation between languages. The application features a modern, responsive user interface with support for mobile devices.

Architecture

The application follows a client-server architecture with:

1. **Frontend**: A Next.js web application built with React and TypeScript
2. **Backend**: A FastAPI Python service that integrates with various AI and cloud services
3. **Deployment**: Containerized using Docker and deployed on Google Cloud Run

Core Features

1. **Real-time Speech Recognition**: Records and transcribes speech using browser-based Web Speech API and Google Cloud Speech-to-Text.
2. **Medical Text Enhancement**: Improves transcription accuracy for medical terminology using Google's Gemini AI model.
3. **Multilingual Translation**: Translates between multiple languages using Google Cloud Translation API and/or Gemini AI.
4. **Text-to-Speech**: Provides audio playback of original and translated text using browser-based Web Speech Synthesis.
5. **Conversation History**: Stores conversation history for authenticated users.
6. **Responsive Design**: Works on both desktop and mobile devices with touch-friendly controls.
7. **Caching**: Implements client-side and server-side caching to improve performance and reduce API calls.

Technologies Used

Frontend

- **Framework**: Next.js (React framework for server-side rendering)
- **Language**: TypeScript
- **Styling**: Tailwind CSS with custom theming
- **State Management**: React hooks and context
- **API Communication**: Axios HTTP client
- **Audio Processing**:
- Web Speech API for speech recognition
- Web Speech Synthesis API for text-to-speech
- MediaRecorder API for audio recording
- **UI Components**:
- React Icons for iconography
- Custom components for language selection, transcription display, etc.
- **Browser Compatibility**: Bowser for feature detection

Backend

- **Framework**: FastAPI (Python)
- **AI Services**:
- Google Cloud Speech-to-Text for audio transcription
- Google Cloud Translation API for language translation
- Google Gemini AI for medical text enhancement and fallback translation
- **Authentication**: JWT-based authentication system
- **Caching**: In-memory caching for API responses
- **Environment Management**: dotenv for configuration
- **Logging**: Python's logging module
- **CORS Handling**: FastAPI's built-in CORS middleware
- **Error Handling**: Global exception handler for graceful error recovery

DevOps & Deployment

- **Containerization**: Docker for building portable containers

- **Orchestration**: Docker Compose for local development
- **Cloud Platform**: Google Cloud Run for serverless deployment
- **CI/CD**: Manual deployment through deployment scripts
- **Version Control**: Git

Data Flow

1. **Audio Recording**:

- User speaks into their device's microphone
- Audio is captured by the MediaRecorder API
- If browser supports it, Web Speech API provides real-time transcription

2. **Transcription Processing**:

- Real-time transcript updates are sent to the backend
- The backend enhances medical terminology using Gemini AI
- Enhanced text is translated to the target language

3. **Response Handling**:

- Enhanced and translated text are displayed to the user
- User can use text-to-speech to hear the translation
- If authenticated, conversations can be saved

Technical Implementation Details

Speech Recognition

The system implements a dual approach to speech recognition:

- Primary: Browser-based Web Speech API for immediate feedback
- Backup: Google Cloud Speech-to-Text for higher accuracy medical transcription

Text Enhancement

Medical text enhancement is performed using the Gemini AI model with prompts specifically designed for medical context. The system includes fallback mechanisms to handle API rate limits by returning the original text if enhancement fails.

Translation

The application uses a tiered approach to translation:

1. Google Cloud Translation API (preferred for accuracy and speed)
2. Gemini AI as a fallback option

Users can select their preferred provider through the UI.

Caching System

The application implements a multi-level caching strategy:

- Client-side cache for API responses with a 30-minute expiration
- Server-side cache for transcriptions, translations, and enhancements
- LocalStorage for user preferences

Responsive Design

The UI adapts to different screen sizes through:

- Tailwind CSS responsive breakpoints
- Mobile-specific touch interactions
- Dynamic layout adjustments based on viewport size

Error Handling

The system has robust error handling:

- Frontend: Try-catch blocks with fallbacks for critical functionality
- Backend: Global exception handler that returns graceful error responses
- User Feedback: Clear error messages for user-facing issues

Performance Optimizations

1. **API Caching**: To reduce redundant API calls
2. **Debounced Processing**: To limit real-time processing frequency
3. **Lazy Loading**: Components and resources loaded only when needed
4. **Content Throttling**: Processing transcript only when significant changes occur
5. **Browser Feature Detection**: To use optimal APIs based on device capabilities

Security Considerations

1. **Data Privacy**: Audio data is not stored permanently
2. **Authentication**: JWT-based authentication for user sessions

3. **CORS Policy**: Configured to allow only specific origins
4. **API Key Management**: Secure handling of Google Cloud credentials