Technical Documentation: Medical Speech Translator

1. Overview

The Medical Speech Translator, a web-based application designed to facilitate real-time transcription, enhancement, and translation of medical speech. It leverages the Web Speech API for speech recognition, the Gemini AI API for medical term correction and multilingual translation, and a React.js front-end with Material-UI for a responsive, modern user interface.

2. System Architecture

The application is built as a single-page React application with a modular, component-based architecture. Below is an overview of its key components and their responsibilities:

2.1 Core Components

- **App.jsx**: The root component orchestrating state management, API interactions, and rendering of child components. It handles global state (e.g., dark mode, language selections, transcripts) and coordinates speech recognition and AI processing.
- **SpeechRecognitionService (App.jsx)**: A custom JavaScript class encapsulating the Web Speech API for real-time speech recognition. It supports continuous transcription with interim results and audio level calculations.
- **Header.jsx**: Renders the application title and a theme toggle button for switching between light and dark modes.
- **LanguageSelector.jsx**: Provides dropdown menus for selecting input and output languages, with a swap button for quick language reversal.
- **Controls.jsx**: Manages speech recognition controls (start, stop, pause) and provides visual feedback via audio level bars and status chips.
- **TranscriptPanel.jsx**: Displays original and translated transcripts in a tabbed interface, with options to copy, speak, or reset text.
- **theme.js**: Defines Material-UI themes (light and dark) for consistent styling across the application.

2.2 External Integrations

- **Web Speech API**: Used for both speech recognition (transcription) and speech synthesis (text-to-speech).
- **Gemini AI API**: Powers two critical functions:
    - **Medical Term Enhancement**: Corrects and refines medical terminology in transcripts.
    - **Translation**: Translates enhanced transcripts into the target language with context-aware accuracy.

2.3 File Structure

- **App.jsx**: Main application logic and state management.
- **components/**: Houses reusable UI components (Header, LanguageSelector, Controls, TranscriptPanel).
- **themes/theme.js**: Material-UI theme configurations.
- **Environment Variables**: API keys are stored in VITE_OPENAI_API_KEY for secure access to the Gemini API.

3. Key Features

The Medical Speech Translator offers a robust set of features tailored for healthcare communication:

- **Real-Time Speech Recognition**: Continuously transcribes spoken input with interim and final results, supporting multiple languages.
- **Medical Term Enhancement**: Uses Gemini AI to correct misheard or misspelled medical terms (e.g., drugs, conditions, procedures) while preserving non-medical content.
- **AI-Powered Translation**: Translates enhanced transcripts into one of 12 supported languages, maintaining medical accuracy and context.
- **Multilingual Support**: Includes languages like English, Spanish, French, German, Chinese, and more, with flag-based UI for intuitive selection.
- **Responsive UI**: Features a Material-UI-based interface with light/dark themes, glassmorphism effects, and mobile-friendly layouts.
- **Interactive Controls**: Allows users to start/stop/pause recognition, speak translated text, copy transcripts, and reset sessions.
- **Error Handling**: Gracefully manages microphone permissions, API failures, and recognition errors with user-friendly alerts.

4. Technical Implementation

4.1 Data Flow

The application's data flow is orchestrated in App.jsx and follows these steps:

1. **Language Selection**:
    - Users select input and output languages via LanguageSelector.jsx.
    - State updates (inputLanguage, outputLanguage) trigger transcript resets or re-translations if necessary.
2. **Speech Recognition**:
    - Initiated via the startRecognition function, which configures SpeechRecognitionService with the input language.
    - The service processes audio input, generating interim and final transcript segments.
    - Audio levels are calculated based on result confidence (scaled to 0–100).
3. ***Medical Enhancement***:
    - Final transcript segments are sent to the enhanceMedicalTerms function, which constructs a prompt for the Gemini API.
    - The API corrects medical terms and returns the enhanced text, appended to originalTranscript.
4. **Translation**:
    - The entire updated originalTranscript is passed to translateTextWithGemini, which constructs a translation prompt for the Gemini API.
    - The translated text updates translatedTranscript for display.
5. **UI Rendering**:
    - TranscriptPanel.jsx renders both transcripts in a tabbed view, with loading states for enhancement/translation.
    - Controls.jsx reflects recognition status and audio levels, enabling user interaction.

4.2 State Management

- **React Hooks**: useState, useEffect, useCallback, and useRef manage state and side effects.
    - darkMode: Persists theme preference in localStorage.
    - inputLanguage/outputLanguage: Tracks selected languages.
    - originalTranscript/translatedTranscript/interimTranscript: Store transcription data.
    - isRecognizing/isPaused/isEnhancing/isTranslating: Control UI states and button availability.
    - speechRecognitionRef: Maintains a singleton instance of SpeechRecognitionService.
- **Optimization**: useCallback ensures stable function references, preventing unnecessary re-renders.

4.3 Gemini API Integration

- **Endpoint**: https://generativelanguage.googleapis.com/v1beta/models/gemini-2.0-flash:generateContent.
- **Configuration**:
    - temperature: 0.3 for balanced creativity and accuracy.
    - maxOutputTokens: 2048 to handle longer transcripts.
    - Safety settings block harmful content (e.g., harassment, hate speech).
- **Prompt Design**:
    - **Enhancement**: Instructs Gemini to correct medical terms while preserving non-medical text.
    - **Translation**: Specifies source and target languages, emphasizing medical accuracy.
- **Error Handling**: Checks for HTTP errors, blocked prompts, and missing candidates, falling back to original text if necessary.

4.4 Web Speech API Usage

- **Recognition**:
    - Supports continuous mode with continuous: true and interimResults: true.
    - Handles errors like no-speech, audio-capture, and not-allowed.
- **Synthesis**:
    - Uses SpeechSynthesisUtterance for text-to-speech, selecting voices based on the output language.
    - Falls back to default voices if language-specific ones are unavailable.

4.5 UI and Styling

- **Material-UI**: Provides responsive components (e.g., Grid, Paper, Tabs) and themes.
- **Themes** (theme.js):
    - **Light**: Primary color #072c52, default background.
    - **Dark**: Primary color #6fbd7f, background #121212, paper #1e1e1e.
    - Typography: Montserrat, Roboto, sans-serif.
- **Custom Styling**:
    - Glassmorphism effects in Controls.jsx and LanguageSelector.jsx using backdropFilter: blur(10px).
    - Scrollable transcript areas with custom scrollbars in TranscriptPanel.jsx.
- **Responsive Design**: Uses useMediaQuery for mobile-friendly layouts (e.g., full-width tabs on small screens).

4.6 Error Handling

- **Microphone Permissions**: Checks navigator.mediaDevices.getUserMedia and displays alerts for denials.
- **Speech Recognition Errors**: Handles network, no-speech, audio-capture, not-allowed, and aborted errors with descriptive messages.
- **Gemini API Errors**:
    - Validates API key presence.
    - Catches HTTP errors, blocked prompts, and missing candidates.
    - Logs detailed errors to the console for debugging.
- **UI Feedback**: Alert components display errors, while Snackbar notifications confirm actions.

5. Performance Considerations

- **Asynchronous Processing**: Uses async/await for API calls and speech recognition to prevent UI blocking.
- **Latency**:
    - Speech recognition is near-instantaneous, but Gemini API calls may introduce 1–3 seconds of delay.
    - Interim transcripts provide immediate feedback during processing.
- **Optimization**:
    - Memoized functions (useCallback) reduce re-renders.
    - Efficient state updates minimize DOM operations.

- **Browser Compatibility**: Tested on Chrome and Firefox; may require polyfills for older browsers.
- **Resource Usage**: Memory-intensive for long sessions due to continuous recognition and transcript storage.

6. Security

- **API Key Security**: Stored in VITE_OPENAI_API_KEY environment variable, not hardcoded.
- **Data Privacy**:
    - Transcripts are processed in memory and not persisted.
    - No local file I/O or network storage is used.
- **Gemini Safety**: Configured to block harmful content, ensuring compliance with ethical standards.
- **Browser Security**: Relies on browser sandboxing for microphone access and API calls.

7. Deployment

- **Build**: Use Vite to bundle the React application (npm run build).
- **Hosting**: Deploy on static hosting services (e.g.Vercel) with environment variable configuration.
- **Environment Variables**: Ensure VITE_OPENAI_API_KEY is set in the hosting environment.
- **CORS**: Gemini API calls are made directly from the client, requiring no server-side proxy.

10. Limitations

- **API Dependency**: Requires a valid Gemini API key and internet connectivity.
- **Latency**: Real-time AI processing may introduce slight delays for long or complex transcripts.
- **Browser Support**: Limited to browsers supporting the Web Speech API (e.g., Chrome, Firefox).
- **Language Coverage**: Currently supports 12 languages, which may not cover all healthcare needs, but this feature is easily extendible.