Proof of Concept: Rapid Prototype for AIVoiceTranslator

Objective:

Develop a fully functional voice translation prototype for 1 teacher and 3 students in a live classroom environment. This PoC should demonstrate real-time translation from speech-to-speech in multiple languages with minimal latency and maximum clarity.

Target Setup

- Participants: 1 teacher (microphone input), 3 students (each receiving different language audio)
- Languages: English (input), Spanish, German, French (outputs)

Tech Stack (Sophisticated Tools for Rapid Prototyping)

Component	Tool	Reason
Audio Ingestion	WebRTC / MediaRecorder API	For low-latency browser-based audio capture
Speech-to-Text (STT)	Deepgram (API)	Ultra-low latency, high accuracy
Fallback STT	OpenAI Whisper (API)	Superior multilingual support, robust in noisy settings
Translation	OpenAI GPT-4 Turbo API	High quality, real-time language translation
Text-to-Speech (TTS)	ElevenLabs / OpenAI TTS	Natural voice output, multiple languages
Streaming Server	Node.js + WebSocket	Real-time audio distribution to clients
Frontend	React (Next.js or Vite)	Fast, modular UI for language selection and playback
Deployment	Render / Vercel (Frontend), Railway / Fly.io (Backend)	Rapid CI/CD, low overhead, free/ch

Implementation Steps

Step 1: Audio Capture from Teacher

- Use WebRTC or MediaRecorder API in a React frontend.
- Stream audio in 12 sec chunks via WebSocket to backend server.

Step 2: STT Integration (Primary)

- Integrate Deepgram Real-time Streaming API on backend (Node.js).
- Configure Deepgram's Nova-3 model with "enhanced" tier for best accuracy.
- WebSocket stream from frontend Deepgram transcript.

Step 3: STT Fallback

- If Deepgram fails or latency spikes, fallback to OpenAI Whisper API.
- Use audio buffering and Whisper API in async mode.
- Select model: whisper-large for accuracy.

Step 4: Translation Pipeline

- Send transcript to OpenAI GPT-4 Turbo API.
- Use system prompt to ensure translation accuracy & tone.
- Cache repeated phrases to reduce token cost.

Step 5: TTS Generation

- Use ElevenLabs or OpenAI TTS API.

- Generate voice for each target language.

- Save as small .mp3 files or stream directly (preferred).

Step 6: Audio Streaming to Students

- Implement WebSocket-based audio push per language.

- React client subscribes to desired language channel.

- Add play/pause buttons and fallback download link.

Deployment & Execution

Backend:

- Use Railway.app or Fly.io to deploy the Node.js backend.

- Secure environment variables (API keys for Deepgram, OpenAI, TTS).

- Auto-scale enabled for 35 concurrent users.

Frontend:

- Deploy React frontend to Vercel or Render.

- Enable HTTPS for secure audio streams.

- Minimal UI with:

 - Mic toggle (teacher)

 - Language select (students)

 - Live status & audio playback

Metrics to Validate

- Latency: Speech to playback < 2 seconds
- Accuracy: > 90% transcript + translation match
- Clarity: TTS quality rated > 4.5/5
- Throughput: System handles 3 concurrent streams with no delay

Key Notes

- Ensure caching is implemented at both translation & TTS layers.
- Log token usage and errors for feedback loop.
- Prioritize latency over full transcript integrity.
- PoC will inform architecture for larger 25+ student pilot.

Document Version: April 2025

Project: AIVoiceTranslator

Author: GPT, Technical Co-Developer