**Real-Time Conversational AI Assistant Web App**

**🧠 Project Goal**

To build a **real-time conversational AI assistant web app** using **WebRTC**, **Gemini AI**, **OpenAI Whisper**, **Google TTS**, and more — mimicking a human-like assistant capable of understanding, speaking, and interacting in real time.

**🧱 Architecture Overview**

+-------------+ WebRTC +-------------+ Whisper +--------------+

| Frontend | <-----------------> | Janus GW | <-----------------> | FastAPI |

| (Web App) | Audio + DataChan | | RTP/RTSP/WebSock | Backend |

+-------------+ +-------------+ +---------------+

| | |

|<---> Google TTS (streamed MP3) |<--> Whisper ASR (real-time) |

|<---> Gemini LLM via API |<--> VAD/Noise Supp |

| | |

**✅ Functional Components**

| **Feature** | **Tool/Technology** | **Purpose** |
| --- | --- | --- |
| Voice Input | WebRTC + Janus + VAD + NoiseSupp | Real-time microphone capture and audio transmission |
| Speech-to-Text (ASR) | OpenAI Whisper | Convert user speech to text |
| Language Understanding | Gemini Pro 1.5 / Gemini API | Natural language processing |
| Voice Output (TTS) | Google TTS | Convert AI response to audio |
| Text/Audio Sync | WebRTC DataChannel/WebSockets | Control signaling, commands, and text output |
| Backend Framework | FastAPI (Python) | Handle APIs, business logic, model calls |
| Media Gateway | Janus WebRTC Gateway / Freeswitch | Manage RTP, RTSP, WebRTC media streams |
| Real-time Communication | WebRTC + WebSockets fallback | Robust audio/video/data handling |
| Audio Pipeline | Web Audio API + WebRTC + Processing | Gain control, VAD, noise suppression, echo cancellation |
| Sample Rating | WebSockets + Frontend UI | Capture feedback to improve models |

**📅 Project Plan – Phased Approach**

**Phase 1 – MVP (Voice-to-Text with LLM Response)**

**Goal:** Set up a working pipeline: mic → Whisper → Gemini → text output

**✅ Tasks:**

* Set up FastAPI backend
  + POST /transcribe: send audio and return Whisper text
  + POST /chat: send text to Gemini and return response
* Frontend: React/Vue + Mic capture
* Record short audio → send to backend → get text
* Text → Gemini → response
* Display conversation flow in chat-style interface

**Technologies:**

* Web Audio API (record mic)
* Whisper (server or OpenAI hosted)
* Gemini API
* FastAPI

**Phase 2 – Real-time Audio Streaming & Transcription**

**Goal:** Real-time voice input and Whisper transcription via WebRTC

**✅ Tasks:**

* Set up Janus or Freeswitch for WebRTC streaming
* Use VAD to stream only when voice is detected
* Add noise suppression client-side (WebRTC-supported)
* Integrate WebRTC DataChannel for control messages

**Tech Stack:**

* Janus Gateway or Freeswitch
* WebRTC, RTP, RTSP
* VAD, NoiseSuppression, AudioContext

**Phase 3 – Full Duplex Streaming Assistant**

**Goal:** Mic input → real-time streaming → Whisper → Gemini → Google TTS → back to audio stream

**✅ Tasks:**

* Enable continuous streaming (chunks)
* Streaming Whisper or fast VAD-splitting
* Gemini integration in real-time
* Google TTS with streamed MP3 → frontend
* Auto-play audio response via AudioBuffer/WebRTC

**Considerations:**

* Handle simultaneous input/output (voice interruptions)
* Play response over audio sink (speaker)

**Phase 4 – UI/UX Polish & Ratings**

**✅ Tasks:**

* Chat bubble interface with both text and audio
* Allow user to rate assistant response (1–5 stars)
* Store rating + transcript + metadata (PostgreSQL or Firestore)
* Admin dashboard to review conversations

**Phase 5 – Productionization**

* Add user authentication (Firebase Auth or Auth0)
* Rate limiting and token quota (for LLM)
* Logging and monitoring (Sentry, Prometheus)
* Host backend (Render, Railway, GCP, Fly.io)
* CDN for static files (Cloudflare, Netlify)

**🛠️ Additional Technologies to Consider**

| **Category** | **Tool/Service** | **Why it helps** |
| --- | --- | --- |
| **Auth** | Firebase Auth / Auth0 | Login, session handling |
| **Database** | PostgreSQL / Firestore / Redis | Store conversation, feedback |
| **Vector Store** | Pinecone / Weaviate / FAISS | (Optional) Memory & retrieval for long conversations |
| **Noise Cancel/VAD** | RNNoise, WebRTC built-in | Improve quality before ASR |
| **LLM Caching** | Redis / SQLite | Cache Gemini replies to reduce API calls |
| **Streaming ASR** | Faster Whisper, Nvidia Riva, Vosk | If you want on-prem speed |
| **Monitoring** | Prometheus + Grafana / Sentry | Logs and metrics |
| **DevOps** | Docker + CI/CD | Fast deployment pipeline |

**🔐 Privacy & Ethics Consideration**

* Warn users when recording.
* Mask or redact PII if storing transcripts.
* Clearly communicate model limitations.

**🧪 Testing Tips**

* Unit test VAD and streaming logic
* Test Whisper speed and compare local vs API
* A/B test Gemini vs Claude or Mistral
* Emulate low bandwidth for edge cases

**🧠 Bonus Ideas**

* Add **contextual memory** (RAG + Vector Store)
* Add **wake word detection** (Snowboy / Porcupine)
* Use **character voices** (Google Cloud TTS prosody)
* Add **multi-language support** (Whisper multilingual + Gemini)

**✅ Summary: High-Level Stack**

| **Layer** | **Tool/Tech** |
| --- | --- |
| Frontend | React + WebRTC + Web Audio API |
| Media Gateway | Janus / Freeswitch |
| Backend | FastAPI + Whisper + Gemini + TTS |
| LLM | Gemini API |
| ASR | Whisper (OpenAI or local) |
| TTS | Google Cloud Text-to-Speech |
| Audio Streaming | RTP, RTSP, DataChannel |
| Infra | Docker + CI/CD + Cloud Hosting |

Would you like me to generate a full code scaffold, or perhaps a system diagram to help you visualize how the components interact?