ElderTech Voice Assistant – Full-Stack App Blueprint

Purpose

Create a warm, **voice-first** assistant that helps older adults master everyday technology through natural conversation and spoken answers—no animated avatar required.

---

1. High-Level User Flow

1. **Launch**: User opens the web app on a tablet/desktop. First-time visitors see an onboarding walkthrough with large visuals and voice narration.
2. **Conversation**: A friendly assistant icon gently pulses, prompting the user to tap a large **"Hold to Talk"** button (or use a wake-word).
3. **Transcription**: Microphone audio streams to the backend via WebRTC; Whisper API transcribes in real-time.
4. **NLP Processing**: Backend sends conversation context + transcript to GPT-4o with an empathy-focused system prompt tuned for seniors.
5. **Response Rendering**:
6. GPT returns plain text/markdown.
7. FastAPI calls **ElevenLabs TTS** → returns an MP3 of the spoken answer.
8. **Playback**: Front-end shows 24-pt subtitles and plays the audio. Controls: **Replay | Slower | Louder**.
9. **Logging & Learning**: Q&A pair saved to MongoDB → nightly batch clusters similar questions, grows *FAQs* and RAG index.

---

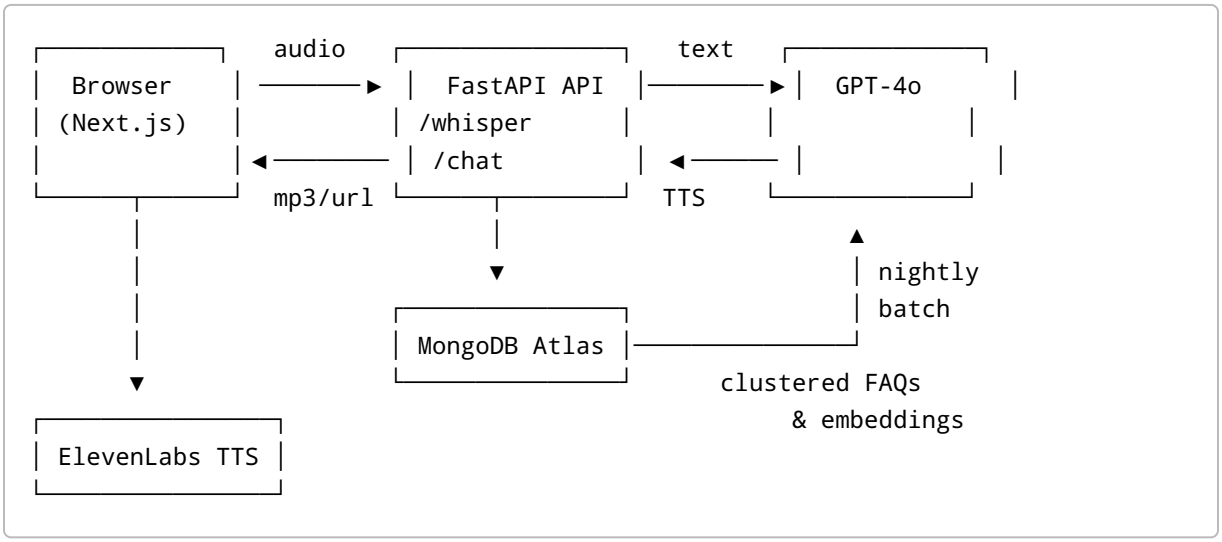
2. Tech Stack

Layer	Choice	Rationale
Frontend	Next.js 14 (React Server Components) | Tailwind CSS | Zustand	Fast, accessible UI with clear state management.
Real-time	WebRTC (audio only) | Socket.IO fallback	Low-latency mic streaming; works on restrictive networks.
Backend API	FastAPI + Pydantic v2	Async, type-safe, automatic OpenAPI docs.
Database	MongoDB Atlas (`users`, `sessions`, `messages`, `faqs`)	Flexible document model fits conversational data.

Layer	Choice	Rationale
AI Services	OpenAI Whisper & GPT-4o; **ElevenLabs TTS**	Best-in-class transcription, reasoning, and natural-sounding speech.
DevOps	Docker Compose (dev) → Render.com / Fly.io / Vercel Edge (prod)	Simple local spin-up; global edge functions for low latency.

---

3. System Architecture



---

4. Frontend Breakdown (Next.js `/app` directory)

```
/app
  ├─ layout.tsx            # Global providers (Zustand, Theme)
  ├─ page.tsx              # Voice-chat landing
  ├─ components/
  │    ├─ AudioPlayer.tsx  # Streams/plays TTS MP3 + captions
  │    ├─ MicButton.tsx    # Press-&-hold recording, large accessible
  │    ├─ Onboarding.tsx   # Carousel tutorial
  │    └─ HelpCenter.tsx   # Themed FAQ explorer
  └─ lib/
       ├─ webrtc.ts         # Mic capture + mediaRecorder
       ├─ api.ts           # fetcher (SWR) for /chat, /faqs
       └─ validators.ts    # zod schemas
```

Key UI Patterns

- **Large Typography**: 18-24 pt base font; 44 px touch targets.
- **Voice Everywhere**: `aria-label` + optional spoken hints on all controls.
- **Assistive Controls**: *Repeat*, *Slower*, *Examples* buttons always visible.

---

5. Backend Breakdown (FastAPI)

```
backend/
  ├─ main.py            # FastAPI app factory
  ├─ routers/
  │    ├─ chat.py       # /chat POST {transcript}
  │    ├─ whisper.py    # /whisper WS stream
  │    ├─ faqs.py       # CRUD on FAQs
  │    └─ auth.py       # JWT /family login
  ├─ services/
  │    ├─ openai.py     # GPT-4 & Whisper calls
  │    ├─ tts.py        # ElevenLabs wrapper
  │    └─ db.py         # Motor (async Mongo) client
  └─ scripts/
       └─ nightly_faq_clustering.py
```

Sample `/chat` Endpoint

```python
@router.post("/chat", response_model=ChatResponse)
async def chat(req: ChatRequest, user: User = Depends(require_user)):
    # 1. Generate answer
    system = "You are a patient tech coach..."
    msgs = [
        {"role": "system", "content": system},
        *req.history,
        {"role": "user", "content": req.transcript},
    ]
    gpt_resp = await openai.chat(messages=msgs, model="gpt-4o")
    answer = gpt_resp.choices[0].message.content

    # 2. Generate speech
    mp3_url = await tts.speak(answer)

    # 3. Persist
    await db.messages.insert_one({
        "user_id": user.id,
        "q": req.transcript,
        "a": answer,
```

```
        "audio": mp3_url,
        "ts": datetime.utcnow()
    })

    return {"answer": answer, "audio": mp3_url}
```

---

6. Database Schema (Mongo)

```
users: {
  _id, name, role: "elder" | "family", language, createdAt
}
sessions: {
  _id, user_id, startedAt, endedAt
}
messages: {
  _id, session_id, q, a, audio, ts
}
faqs: {
  _id, question, answer_md, tags: ["email", "scams"], updatedAt
}
```

---

7. Accessibility & UX Principles

- **WCAG 2.2 AA** compliant colors, text sizes, captions.
- **Keyboard & switch-control** navigation (focus rings, `tabindex`).
- **Latency budget**: <400 ms TTFB, <2 s TTS playback (cached path).

---

8. Onboarding Experience

1. Welcome screen with assistant voice: “Hi! I’m here to help you with tech.”
2. Guided demo: User taps mic, asks “What’s an app?” → hears sample reply.
3. Brief quiz: User tries asking; gets celebratory sound + confetti.
4. Option to browse “common topics” grid or start chatting.

---

9. Family Portal (Optional)

- Auth0-protected dashboard.
- Pre-load FAQs, view transcripts & audio, flag incorrect answers.
- Email digests of weekly usage.

---

10. Deployment & Ops

- **Dev**: `docker compose up` spins Next.js + FastAPI + Mongo.
- **Staging**: Vercel (frontend) → Render (backend) → MongoDB Atlas.
- **Observability**: Sentry (front & back), Prometheus + Grafana metrics.
- **CI/CD**: GitHub Actions → lint, type-check, Playwright e2e.

---

11. Next Steps

1. Scaffold repo with `pnpm create next-app@latest` + `fastapi-project-generator`.
2. Implement Whisper streaming endpoint.
3. Build **AudioPlayer** component and connect to `/chat`.
4. Conduct hallway tests with 2–3 seniors; refine controls & font sizes.
5. Seed FAQ JSON; connect nightly clustering job later.

–– End of Blueprint ––