ElderTech Voice Assistant – Frontend Blueprint

> **Scope**: Client-side code delivered via **Next.js 14** with React Server Components (RSC) + Tailwind CSS. State handled by **Zustand**, streaming handled with **WebRTC / MediaRecorder**. Testing via Playwright + Vitest. This blueprint is organised around the Scrum storyboard epics.

---

0. Core Principles

- **Accessibility-first**: WCAG 2.2 AA, high contrast, large tap/keyboard targets, captions, landmarks.
- **Voice-centric UX**: Minimal visual clutter; every action speakable & audible.
- **Component Isolation**: Each user story maps to an atomic component or hook; facilitates parallel work in Scrum.

---

1. File / Folder Layout (Next.js `app/`)

```
/app
  ├─ layout.tsx             # Providers, global styles, <Toast />
  ├─ page.tsx               # Landing – voice chat canvas
  ├─ (portal)/              # Family portal route group ➜ /portal
  │   ├─ layout.tsx         # Auth boundary
  │   └─ page.tsx           # Dashboard
  ├─ components/
  │   ├─ ui/                # Re-export shadcn/ui primitives
  │   ├─ MicButton.tsx      # Story 1.1
  │   ├─ AudioPlayer.tsx    # Story 1.3
  │   ├─ Waveform.tsx       # Live mic viz (Epic 1)
  │   ├─ Captions.tsx       # SRT-sync overlay
  │   ├─ Onboarding.tsx     # Story 3.1
  │   ├─ HelpCenter.tsx     # Story 3.2
  │   └─ FaqCard.tsx
  ├─ hooks/
  │   ├─ useMic.ts          # WebRTC + MediaRecorder
  │   ├─ useWhisperWS.ts    # WS wrapper – partial & final transcripts
  │   └─ useChat.ts         # GPT + TTS pipeline orchestrator
  ├─ stores/
  │   ├─ chatStore.ts       # Zustand: messages, status, error
  │   └─ uiStore.ts         # Modals, settings, contrast mode
  ├─ lib/
  │   ├─ fetcher.ts         # SWR fetch wrapper (auth tokens)
  │   ├─ validators.ts      # zod schemas shared with backend
  │   └─ utils.ts           # sentence-split, debounce, etc.
```

```
└─ styles/
    └─ globals.css          # Tailwind base + custom utilities
```

---

2. Epic-to-Component Mapping

Scrum Epic	User Story ID	Component / Hook	Sprint
Epic 1 – Core Voice Interaction	1.1	`MicButton`, `useMic`	1
	1.2	`useWhisperWS`, `Waveform`	1
	1.3	`AudioPlayer`, `Captions`, `useChat`	1
Epic 2 – Knowledge & Learning	2.1	`HelpCenter` (instant FAQ display)	2
	2.2	Log UI channel (no component)	2
Epic 3 – Onboarding & Help Center	3.1	`Onboarding`, `uiStore` flags	1
	3.2	`HelpCenter`, `FaqCard`	2
Epic 4 – Family Portal	4.1	`(portal)/` pages, `FaqForm`	3
Epic 5 – DevOps & Quality	5.1	Playwright tests in `e2e/`	1+

---

3. Key Components (Code-level Notes)

3.1 `MicButton.tsx`

```
export default function MicButton() {
  const { recording, start, stop } = useMic();
  return (
    <button
      aria-label={recording ? 'Release to stop recording' : 'Hold to talk'}
      className={cn(
        'rounded-full bg-primary text-white p-6 shadow-lg',
        recording && 'bg-red-600',
      )}
      onPointerDown={start}
      onPointerUp={stop}
    >
      <MicIcon className="w-8 h-8" />
```

```
    </button>
  );
}
```

Technical Tip: Use `navigator.mediaDevices.getUserMedia({audio:true})` once, then reuse the stream across `useMic` & `useWhisperWS` to avoid multiple permission prompts.

3.2 `useWhisperWS.ts`

- Opens WS to `/whisper`.
- Sends `audio/webm` blobs every 800 ms.
- Emits `partial` and `final` events.
- Closes gracefully on component unmount.

3.3 `AudioPlayer.tsx`

- Accepts `audioUrl` + `captions` (array of `{start,end,text}`)
- Uses `<audio>` tag with preload="auto"; on `timeupdate` syncs captions.
- Provides buttons: Replay, Speed ↓ (0.75×), Speed ↑ (1.25×), Volume slider.

3.4 `Onboarding.tsx`

- Framer Motion carousel (`AnimatePresence`).
- Each slide has SVG illustration + voice-over (MP3) auto-play.
- LocalStorage `eldertech_onboarded` flag.

3.5 `HelpCenter.tsx`

- Fetches `/faqs` (static gen + ISR).
- Filters by search query; groups by `tags`.
- Renders `FaqCard` expandable accordion with answer & “Play explanation” (TTS).

---

4. State & Data Flow

1. **Recording**: `MicButton` → `useMic` streams blobs.
2. **Transcription**: `useWhisperWS` → dispatches `chatStore.addTranscript()`.
3. **Chat**: On `final` transcript, `useChat.ask()`:
4. Adds *pending* assistant message.
5. Calls backend `/chat` (GPT + ElevenLabs).
6. Updates message with `audioUrl`, `text`, `captions`.
7. **Playback**: `AudioPlayer` auto-plays; `Captions` overlay subscribes to the same audio element time.

---

5. Styling & Theming

- **Tailwind** config: `fontSize: { base: '20px', lg: '24px' }`.
- **Custom plugin**: `tap-target` → `@apply w-14 h-14`.
- **Dark / Contrast Mode**: Stored in `uiStore`; toggled in settings.

---

6. Testing Strategy

Layer	Tool	Scope
Unit	Vitest + React Testing Library	Component props / hooks logic
End-to-End	Playwright	Happy path: onboarding → ask question → hear answer
Accessibility	@axe-core/playwright	Each page ci-check for WCAG violations

---

7. Performance & Observability

- **Code splitting**: RSC for data-heavy components; client bundles kept under 200 kB gz.
- **Lazy audio**: MP3 streamed via `Range` requests; avoids blocking.
- **Sentry**: `@sentry/nextjs` + replay for session recordings (mask PII).

---

8. Sprint 1 Checklist (Frontend-only)

-

---

9. Future Enhancements (Mapping to Follow-Up Steps)

1. **Edge-socket fallback**: Migrate WS to WebTransport when stable.
2. **PWA Installability**: Add manifest + service worker for offline onboarding.
3. **Internationalisation**: `next-intl` integration; dynamic TTS voices.
4. **Avatar Feature Flag**: If video avatar reintroduced, extend `AudioPlayer` into `MediaPlayer` with `<video>`.
5. **Switch-Control UX**: Research Spatial Navigation API to support TV remotes.

10. Frontend ↔ Backend Endpoint Integration Design Sheet

This section maps **client hooks / components** to their corresponding **backend routes**, with payload shapes, transport protocols, and resilience tactics. Use it as a contract between FE and BE teams.

10.1 Endpoint Matrix

Capability	Front-end Trigger (Hook / Component)	Backend Route	Verb / Transport	Request Payload (key fields)	Success Respon
Live transcription (Whisper)	`useWhisperWS` (called by `MicButton`, `Waveform`)	`/whisper`	**WebSocket**	`audio/webm` binary *chunks* every 800 ms	`{type:'partial'
Chat → TTS	`useChat.ask()`	`/chat`	**POST JSON**	`{transcript:string, history:Message[], locale?:string}`	`{answer:str audio:string (MP3 URL), captions:SRT`
FAQs (bulk)	`HelpCenter` *initial fetch*	`/faqs`	**GET**	`?limit=100&lang=en`	`Faq[]`
FAQ search	`HelpCenter` search bar	`/faqs/search`	**GET**	`?q=dark mode&lang=en`	`Faq[]`
FAQ CRUD (Family Portal)	`FaqForm`	`/faqs/:id`	**POST / PUT / DELETE**	JSON schema-validated	200 OK with up doc
Auth token	`fetcher.ts` interceptor	`Auth0` domain	**GET /oauth/token**	client_id, refresh_token	`{id_token}`
Metrics ping	`useChat.ask()` success	`/metrics/usage`	**POST**	`{user_id,msg_len,latency_ms}`	204 No Conten

10.2 Sequence: Ask a Question

1. **User presses & holds** `MicButton` → `useMic` starts stream.

2. `useWhisperWS` opens WS `/whisper` and begins sending chunks.
3. Backend returns **partial** transcripts which appear live as captions.
4. On **final** transcript event, FE triggers `useChat.ask()`.
5. `useChat` POSTs to `/chat`; displays pending spinner.
6. BE queries GPT-4o → ElevenLabs TTS → persists; returns payload.
7. FE pushes message to `chatStore`, instantiates `AudioPlayer` with `audioUrl`.
8. `AudioPlayer` streams MP3, captions sync; user hears answer.
9. FE fires `/metrics/usage` ping (non-blocking).

10.3 Resilience & Observability Hooks

- **Global Error Boundary** displays modal with voice narration on network loss.
- **Sentry** middleware captures unhandled rejections from any fetch/WS.
- **Ping endpoint** `/healthz` polled every 60 s; disables mic if backend unhealthy.

10.4 Mock Contracts (zod schemas, shared via `validators.ts`)

```
export const WhisperPartial = z.object({
  type: z.literal('partial'),
  text: z.string(),
  ts: z.number(),
});
export const ChatRequest = z.object({
  transcript: z.string().min(1),
  history: z.array(MessageSchema),
  locale: z.string().optional(),
});
export const ChatResponse = z.object({
  answer: z.string(),
  audio: z.string().url(),
  captions: z.string(),
});
```

> **Tip**: Use these same schemas in FastAPI via `pydantic-zod` to avoid drift.

–– End Frontend Blueprint ––