ElderTech Voice Assistant – API Integration Sheet

> **Purpose**: Serve as the authoritative contract between the **frontend** (Next.js) and the **backend** (FastAPI). It lists every public route, payload shape, transport protocol, and failure-handling tactic so designers, engineers, and QA stay in sync.

---

1. Endpoint Matrix

Capability	Front-end Trigger (Hook / Component)	Backend Route	**Backend File**	Verb / Transport	Request Payload (key fields)
Live transcription (Whisper)	`useWhisperWS` (from `MicButton`, `Waveform`)	`/whisper`	`routers/whisper.py`	**WebSocket**	`audio/webm` binary *chunks* every 800 ms
Chat → TTS	`useChat.ask()`	`/chat`	`routers/chat.py`	**POST JSON**	`{transcript:string, history:Message[], locale?:string}`
FAQs (bulk)	`HelpCenter` initial load	`/faqs`	`routers/faqs.py`	**GET**	`?limit=100&lang=en`
FAQ search	`HelpCenter` search bar	`/faqs/search`	`routers/faqs.py`	**GET**	`?q=dark mode&lang=en`
FAQ CRUD (Family Portal)	`FaqForm`	`/faqs/:id`	`routers/faqs.py`	**POST / PUT / DELETE**	JSON schema
Auth token	`fetcher.ts` interceptor	Auth0 `/oauth/token`	*(external)*	**POST**	`client_id`, `refresh_token`
Metrics ping	`useChat.ask()` success	`/metrics/usage`	`routers/metrics.py`	**POST**	`{user_id,msg_len,latency_ms}`

---

2. Sequence – *Ask a Question*

1. **Press & hold** `MicButton` → `useMic` streams mic blobs.
2. `useWhisperWS` opens `/whisper`; streams every 800 ms.
3. Server returns **partial** text; captions update in real time.
4. On **final** event, `useChat.ask()` POSTs `/chat` with transcript & history.
5. Backend → GPT-4o → ElevenLabs; stores in Mongo; responds with MP3 + captions.
6. FE pushes message to `chatStore`, renders `AudioPlayer`.
7. MP3 streams; captions sync; user hears answer.
8. FE sends `/metrics/usage` (non-blocking).

---

3. Resilience & Observability Hooks

- **Global Error Boundary** shows voice-narrated modal on network loss.
- **/healthz** polled every 60 s; disables mic if unhealthy.
- **Sentry** captures fetch/WS errors with breadcrumbs.

---

4. Shared Schemas (zod ⇄ pydantic)

```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*: Import these schemas on the backend with **pydantic-zod** to generate identical Pydantic models and avoid drift.

–– End API Integration Sheet ––