voquill

Voquill Logo

Discord 18 online

Your keyboard is holding you back.

Make voice your new keyboard. Type four times faster by using your voice.

Visit our website →

Voquill is an open-source, cross-platform speech-to-text workspace that lets you dictate into any desktop application, clean the transcript with AI, and keep your personal glossary in sync. The repo bundles the production desktop app, marketing site, Firebase backend, and all shared packages in a single Turborepo.

Highlights

- Voice input everywhere: overlay, hotkeys, and system integrations work across macOS, Windows, and Linux.

- Choose your engine: run Whisper locally (with optional GPU acceleration) or point to Groq's hosted Whisper via your own API key.
- AI text cleanup: remove filler words and false starts automatically with the post-processing pipeline in `@repo/voice-ai`.
- Personal dictionary: create glossary terms and replacement rules so recurring names and phrases stay accurate.
- Batteries included: Tauri auto-updates, Firebase functions for billing and demos, and shared utilities/types

Monorepo Layout

Path	Description
`apps/desktop`	Tauri desktop app (Vite + React + Zustand) controlling UI, state, and business logic.
`apps/desktop/src-tauri`	Rust API layer invoked from TypeScript for native capabilities, SQLite storage, and Whisper inference.
`apps/web`	Astro-powered marketing site hosted at voquill.com.
`apps/firebase`	Firebase Functions project handling authentication, billing, demos, and Groq-powered server transcription.
`packages/voice-ai`	Audio chunking + Groq client used for transcription and transcript cleanup.
`packages/types`	Shared domain models (users, transcriptions, dictionary terms, etc.).
`packages/functions`, `packages/firemix`, `packages/utilities`, `packages/pricing`, `packages/ui`, `packages/typescript-config`, `packages/eslint-config`	Reusable utilities, UI primitives, configuration, and Firemix helpers consumed by every app.
`docs`	Architecture notes, release guides, and reference material.
`scripts`	Automation and helper scripts (for example Linux dependency setup).

Architecture Overview

The desktop app follows a TypeScript-first design: Zustand maintains a single global store, while pure utility functions in `apps/desktop/src/utils` read and mutate state. Actions compose those utilities and may call out to repositories. Repos abstract whether persistence happens locally (SQLite through Tauri commands) or remotely (Firebase/Groq).

```
User input / system events
        ↓
React + Zustand state (TypeScript)
        ↓
Repos choose local vs. remote storage
        ↓
Tauri commands (Rust API bridge)
        ↓
SQLite, Whisper models, or external services
```

Rust stays focused on native integrations—audio capture, keyboard injection, updater, encryption, GPU enumeration, filesystem paths. TypeScript owns business logic, routing, and UI. The same shared packages are imported by the desktop app, Firebase functions, and the marketing site, which keeps the product vocabulary in sync.

See `docs/desktop-architecture.md` for the full tour.

Getting Started

Prerequisites

- Node.js 18+ (desktop/web) and npm 10+.
- Rust toolchain with `cargo`, `rustup`, and the Tauri CLI (`cargo install tauri-cli`).
- Platform dependencies for Tauri (GTK/WebKit/AppIndicator/etc.). On Linux you can run `apps/desktop/scripts/setup-linux.sh`. On Windows use `powershell -ExecutionPolicy Bypass -File apps/desktop/scripts/setup-windows.ps1` (add `-EnableGpu` to pull the Vulkan SDK for GPU builds).
- A Groq API key if you plan to use hosted transcription or transcript cleanup (`GROQ_API_KEY`).
- Firebase CLI (`npm install -g firebase-tools`) when working on the functions project.

Install dependencies

```
npm install
```

Build everything

Use the prepared `turbo` task instead of manually invoking `turbo dev`.

```
npm run build
```

Run the desktop app

Pick the platform-specific script; avoid `turbo dev` since the desktop app manages its own watcher.

```
# Mac
npm run dev:mac --workspace apps/desktop

# Windows
npm run dev:windows --workspace apps/desktop

# Linux (CPU)
npm run dev:linux --workspace apps/desktop

# Linux with Vulkan GPU acceleration
npm run dev:linux:gpu --workspace apps/desktop
```

During local development you can override platform detection by exporting `VOQUILL_DESKTOP_PLATFORM` (`darwin`, `win32`, or `linux`). The desktop dev journey now defaults to the `emulators` flavor (`apps/desktop/.env.emulators`) so that `turbo dev` and the workspace dev scripts point at the Firebase emulator suite; pass `FLAVOR=dev` or `VITE_FLAVOR=dev` when you explicitly want the hosted dev project. Set `VITE_USE_EMULATORS=true` to point at Firebase emulators (this is already true in the emulator flavor).

Running in windows

```
# 1) Make sure MSVC is initialized
# (Skip this if you're already in "Developer PowerShell for VS 2022")
& "C:\Program Files\Microsoft Visual
Studio\2022\Community\VC\Auxiliary\Build\vcvars64.bat"

# 2) Required for whisper.cpp / ggml Vulkan builds
$env:WHISPER_CMAKE_ARGS = '-DGGML_VULKAN=ON -DCMAKE_INSTALL_PREFIX=C:/w'

# (Optional but recommended) shorten build paths on Windows
$env:CARGO_TARGET_DIR = 'C:\cargo'
```

```
# 3) Build
npm run dev
```

Run the marketing site

```
npm run dev --workspace apps/web
```

Firebase Functions and emulators

The Firebase workspace targets Node 20. Install its dependencies (`npm install` inside `apps/firebase/functions`) and start emulators with:

```
npm run dev --workspace apps/firebase/functions
```

Secrets such as `GROQ_API_KEY` can be managed through `.secret.local` when using the emulator suite.

Quality checks

```
npm run lint
npm run check-types
npm run test
```

Individual workspaces expose the same commands if you need a narrower scope.

Run in prod mode

1. Comment out devUrl in `apps/desktop/src-tauri/tauri.conf.json`.
2. cd /apps/desktop
3. npm run build
4. VITE_FLAVOR=prod npx tauri dev --no-dev-server

Environment Reference

Variable	Purpose
`VOQUILL_API_KEY_SECRET`	Secret used by the desktop app to encrypt API keys stored on disk (`apps/desktop/src-tauri/src/system/crypto.rs`).

Variable	Purpose
`VOQUILL_WHISPER_MODEL_URL` / `VOQUILL_WHISPER_MODEL_URL_<SIZE>`	Override download locations for Whisper models when running locally.
`VOQUILL_WHISPER_DISABLE_GPU`	Force the desktop app to avoid GPU inference, useful for debugging.
`VITE_USE_EMULATORS`	When set to `true`, the desktop app points to Firebase emulators instead of production services.
`GROQ_API_KEY`	Enables Groq-backed transcription/cleanup in Firebase functions and in the desktop API transcription mode.

Releases & CI

- Desktop builds are produced by `.github/workflows/release-desktop.yml`. The workflow bumps a channel tag, builds all three platforms, and publishes assets plus `latest.json` manifests. Promo runs handle dev→prod promotion. See `docs/desktop-release.md` for step-by-step instructions.
- Turbo caching is configured in `turbo.json`; CI jobs call `npm run build`, `npm run lint`, and other workspace-scoped commands.

Documentation

- Desktop architecture: `docs/desktop-architecture.md`
- Release playbook: `docs/desktop-release.md`
- Additional resources and inspiration: `docs/resources.md`
- Contributor conventions and workspace notes: `AGENTS.md`

License

Unless otherwise noted, Voquill is released under the AGPLv3. See `LICENCE` for the complete terms and third-party attributions.