Project-W — Self-hosted Speech-to-Text mit OpenAI Whisper

Julian Partanen, Markus Everling

2024-04-11

Was ist Project-W? Wozu brauchen wir es?

Project-W ist ein self-hosted Transkriptionsservice basierend auf dem OpenAI Whisper Modell.

In der Forschung müssen oft Sprachaufnahmen (z.B. Interviews) transkribiert werden. Inzwischen sind AI-Models wie OpenAI Whisper darin besser (und schneller und billiger) als Menschen.

Es gibt bereits kommerzielle Services für solche Transkription (z.B. die OpenAI API). Probleme damit:

- Datenschutzrechtlich schwierig.
- Meist keine einfache Nutzeroberfläche verfügbar.

Project-W löst diese Probleme.

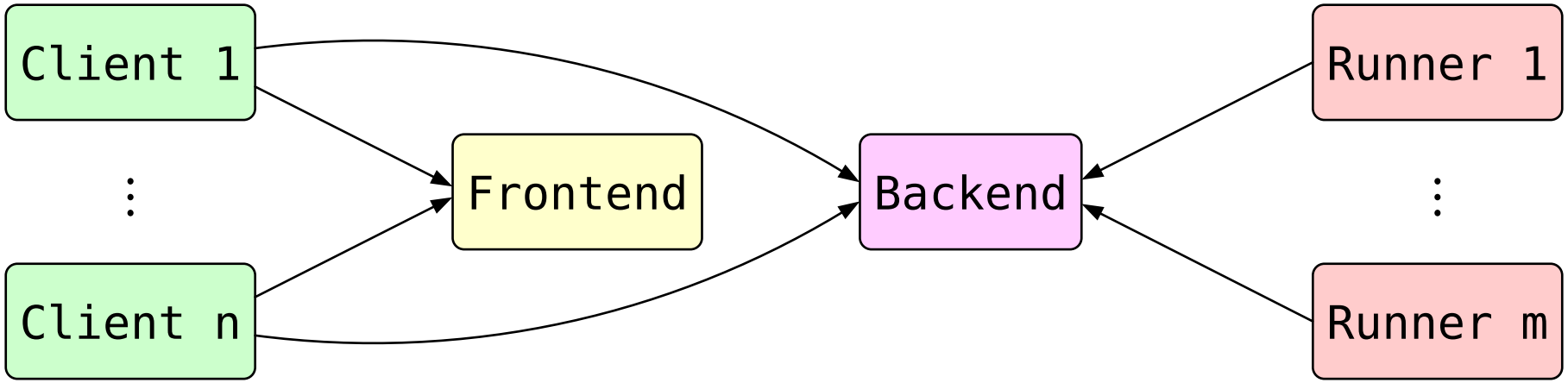
- Komplett self-hosted, keine vertraulichen Daten verlassen das Uni-Netzwerk.
- Einfache Nutzeroberfläche verfügbar.

Grundlegendes Software-Design

Project-W besteht aus drei Teilen:

- dem Frontend, das die schöne Nutzeroberfläche bereitstellt.
- den Runnern, die die Transkriptionen erstellen.
- dem Backend, das die anderen Komponenten verknüpft und verwaltet.

Konzeptuell interagieren die drei Teile folgendermaßen:



- Die Clients (Browser) kommunizieren sowohl dem Frontend als auch mit dem Backend. Es besteht keine direkte Kommunikation zwischen Frontend und Backend.
- Die Kommunikation zwischen Runnern und Backend ist einseitig, das Backend macht nie Requests an die Runner.

(Erklärungen folgen…)

Das Backend

Das Backend ist der zentrale Server von Project-W und stellt eine REST-API bereit. Es ist das Herzstück von Project-W, und ist für die Verwaltung aller Komponenten verantwortlich.

Die verfügbaren Routen lassen sich in drei Teile unterteilen:

1. `/api/users/*`: User-Management. Sachen wie Signup, Login, Account Activation, etc.
2. `/api/jobs/*`: Job-Management. Ein Job entspricht dabei einer zu transkribierenden Audiodatei. User können Jobs hochladen, und das Transkript herunterladen sobald der Job fertig bearbeitet ist.
3. `/api/runners/*`: Kommunikation zwischen Runnern und Backend. Diese Routen werden fast ausschließlich von Runnern abgerufen und erlauben den Runnern z.B. ihnen zugeteilte Jobs herunterzuladen und die Transkripte an das Backend zu senden.

Das Backend: Implementierungsdetails

Das Backend ist in Python geschrieben und verwendet das Flask-Framework für die REST-API.

Informationen über User, Jobs und Runner werden in einer SQLite-Datenbank gespeichert, die mit Hilfe von der SQLAlchemy-Bibliothek verwaltet wird. Zudem speichert sich das Backend zur Laufzeit Informationen darüber, welche Runner gerade online sind und welchem Runner welcher Job zugeordnet ist.

Nutzerauthentifizierung verwendet den von OWASP empfohlenen Argon2-Hasher für Passwort-Hashing. Nach der Authentifizierung erhalten Nutzer ein JWT-Token, das bei darauffolgenden API-Aufrufen zur Authentifikation genutzt wird.

Für die Config verwendet das Backend pyaml-env und jsonschema. Damit kann man Umgebungsvariablen in die Config interpolieren, und durch jsonschema werden fehlende Config-Werte rechtzeitig gemeldet.

Die Runner

Die Runner sind für die Transkriptionen der Jobs zu erstellen. Da sie vom Backend separiert sind, kann man den Durchsatz von Project-W einfach erhöhen, indem man mehr Runner erstellt.

Die Kommunikation zwischen Runner und Backend verläuft einseitig, d.h. nur die Runner machen Anfragen an das Backend, nie andersherum. Sämtliche Informationen vom Backend an die Runner werden in Responses der Requests übermittelt. Das hat zwei große Vorteile gegenüber beidseitiger Kommunikation:

- Die Runner brauchen keine öffentlich erreichbare IP-Adresse, sondern können auf jedem beliebigen PC laufen.
- Solange das Backend HTTPS verwendet, ist alle Kommunikation zwischen Runner und Backend automatisch verschlüsselt.

Jeder Runner schickt dem Backend periodisch einen Heartbeat der signalisiert, dass der Runner noch online ist. Wenn das Backend dem Runner einen Job zugeteilt hat, teilt es ihm das in der Heartbeat-Response mit. Der Runner lädt sich dann den Job vom Backend herunter und bearbeitet ihn. Anschließend schickt der Runner das Transkript zurück an das Backend.

Die Runner authentifizieren sich gegenüber dem Backend mit Hilfe eines *Runner Tokens*. Dieses kann vom Backend durch Abrufen der `/api/runners/create` Route erstellt werden. Jeder Runner braucht ein eigenes Token, und das Backend verwendet die Tokens z.B. um den Runnern die richtigen Job-Daten zu schicken.

Die Runner: Implementierungsdetails

Intern verwendet der Runner das OpenAI Whisper Modell für die Transkription. Dieses ist Open-Source, und kann leicht über die openai-whisper Bibliothek aufgerufen werden. Whisper hat verschiedene mögliche Modelle (z.B . tiny, medium, large), die verschieden viel Rechenleistung verbrauchen und zu verschiedener Qualität des Transkripts führen. Man kann über die Job-API kontrollieren, welches Modell ein Job verwenden soll.

Zusätzlich verwendet der Runner ffmpeg, um die Audiodateien in ein Format zu dekodieren das Whisper versteht.

Für die Kommunikation mit dem Backend verwendet der Runner die asyncio und aiohttp Bibliotheken.

Für Konfiguration verwendet der Runner das gleiche System wie das Backend, basierend auf pyaml_env und jsonschema.

Das Frontend: Implementierungsdetails

Technology-Stack: Svelte + Typescript + Vite + svelte-spa-router + flowbite-svelte + tailwindcss

Svelte: Modernes Javascript-Framework, deklarativ, kompakt, schreibt sich zu großen Teil wie eine template-Engine für HTML, compiled (ohne Laufzeit-Umgebung)!

Typescript: Explizite Types, type safety!

Vite: Development environment mit Compiler und development-Server

svelte-spa-router: SvelteKit nicht nötig für Projekt, nur unnötige serverseitige dependencies -> hash-based routing (alles clientseitig)

flowbite-svelte: UI-component-library, ich bin faul und bin kein Designer

tailwindcss: CSS framework, stellt fertige CSS-Klassen bereit

Tests und Dokumentation

Für Unit-Tests verwendet wir die pytest Bibliothek. Flask stellt für diese Tests einen sehr einfachen Mock-Client zur Verfügung. Zusätzlich verwenden wir pytest-mock, z.B. um einen SMTP-Server zu mocken.

Für die Dokumentation verwenden wir sphinx mit einigen Plugins, die es z.B. erlauben, das jsonschema der Config direkt in die Dokumentation einzubinden. Die Dokumentation, inklusive Installationsguide und API Reference, ist abrufbar auf https://project-w.readthedocs.io.

Installation/Setup

Für ein einfaches Setup stellt Project-W Dockerfiles für Frontend, Backend und Runner bereit. Damit kann man mit nur wenigen Befehlen eine komplette Project-W Instanz installieren und starten. Ausführliche Installationsguides gibt es auf https://project-w.readthedocs.io/en/latest/installation.html.

Wenn man das mitgelieferte Frontend verwenden will, bieten wir dafür eine docker-compose config an, die Backend und Frontend zusammen auf einem Server hinter einer nginx Proxy hostet, und die sich automatisch um SSL-Zertifikate kümmert. Für andere Setups (z.B. selbstgeschriebenes Frontend) kann man auch das Backend alleine installieren. Die Runner können auf ähnliche Weise mit nur wenigen Befehlen installiert und gestartet werden.

Zusätzlich stellen wir NixOS Flakes als alternative Installationsmethode für NixOS bereit.

Demo