SCIENTIFIC SOFTWARE CENTER

UNIVERSITÄTS-RECHENZENTRUM

UNIVERSITÄT HEIDELBERG
ZUKUNFT
SEIT 1386

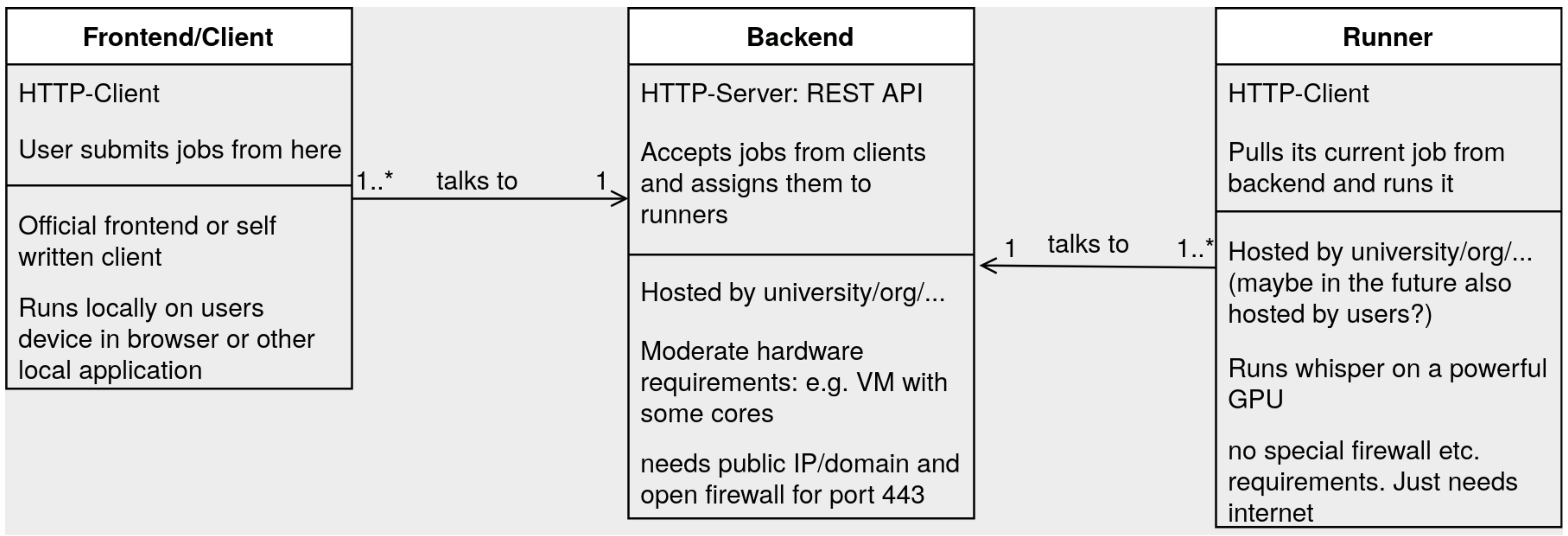
Introducing Project-W: A self-hostable platform for OpenAI's Whisper

Julian Partanen [1], Markus Everling [1], Dominic Kempf [1], Tim Knoth [2], Don Böhm [2], Nikolaus Kepper [2], Martin Baumann [2], Alexander Haller [2]

[1]Scientific Software Center, IWR, Heidelberg University [2]URZ, Heidelberg University

TLDR: What is Project-W?

- Platform for creating transcripts of audio files (speech-to-text) with OpenAIs whisper model (or different models in the future)
- Highest privacy standards
  - Self-hostable, on-premise
  - Suitable for sensitive data
  - GDPR compliance
- Easy to use for the end user
  - Just visit a website, user doesn't need to install anything
  - Simple workflow: Sign in and upload your file
  - Yet still adaptable to more complex use cases by directly interacting with the documented API with own script/client



Frontend/Client

HTTP-Client

User submits jobs from here

Official frontend or self written client

Runs locally on users device in browser or other local application

Backend

HTTP-Server: REST API

Accepts jobs from clients and assigns them to runners

Hosted by university/org/...

Moderate hardware requirements: e.g. VM with some cores

needs public IP/domain and open firewall for port 443

Runner

HTTP-Client

Pulls its current job from backend and runs it

Hosted by university/org/... (maybe in the future also hosted by users?)

Runs whisper on a powerful GPU

no special firewall etc. requirements. Just needs internet

Figure 1. The overall architecture of Project-W

Why not just use OpenAIs own service?

- Some research fields deal with sensitive data that cannot be shared with third parties
- i.e. recordings containing private interviews or medical information
- Uploading these audio files to e.g. OpenAIs servers would, for example, violate data protection requirements

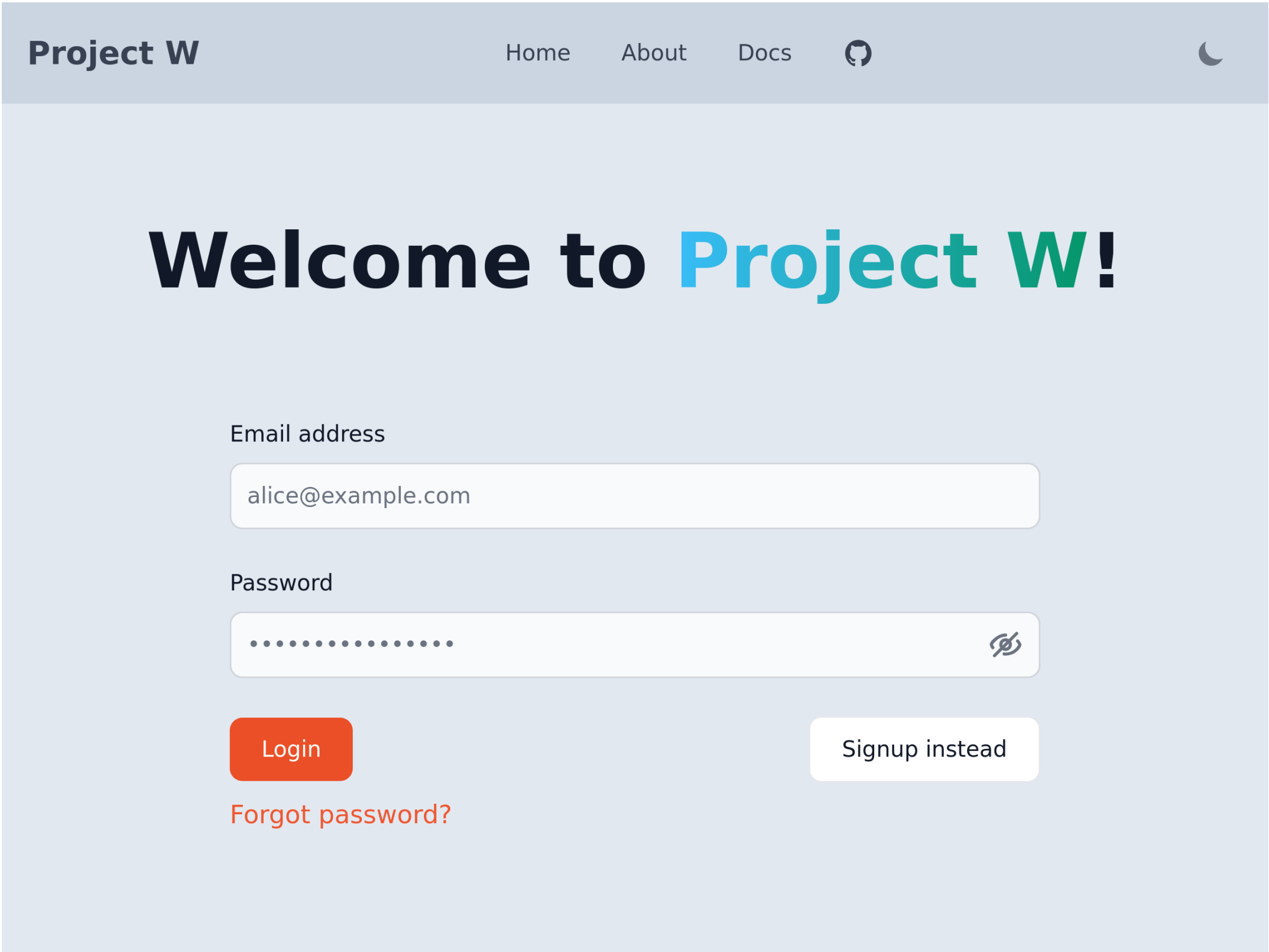
Why a server-client architecture?

- The alternative would be to run whisper locally on every users machine
- Viable, there are many great graphical programs for whisper out there that run purely locally
- However you might encounter some problems with that approach:
  - High hardware requirements (GPUs), especially for larger models and longer audio recordings
  - Difficult installation: CUDA driver, download of models, ...
  - Installation on a fleet of machines across many operating systems. What about smartpones, iPads, ...?
  - Possibly still more difficult to use: Many programs tend to be a bit more technical

-> A website that every user can just visit from any device with a deployment of runners on proper GPU-backed hardware might be the better approach.

User flow (alpha software, subject to change)

1. Login (or Signup) (Fig. 2)
2. Create a new job (Fig. 3)
3. Fill in details and select audio file, submit job (Fig. 4)
4. Wait for job to complete (Fig. 5)
5. Download transcript as a text file (Fig. 6)



Welcome to Project W!

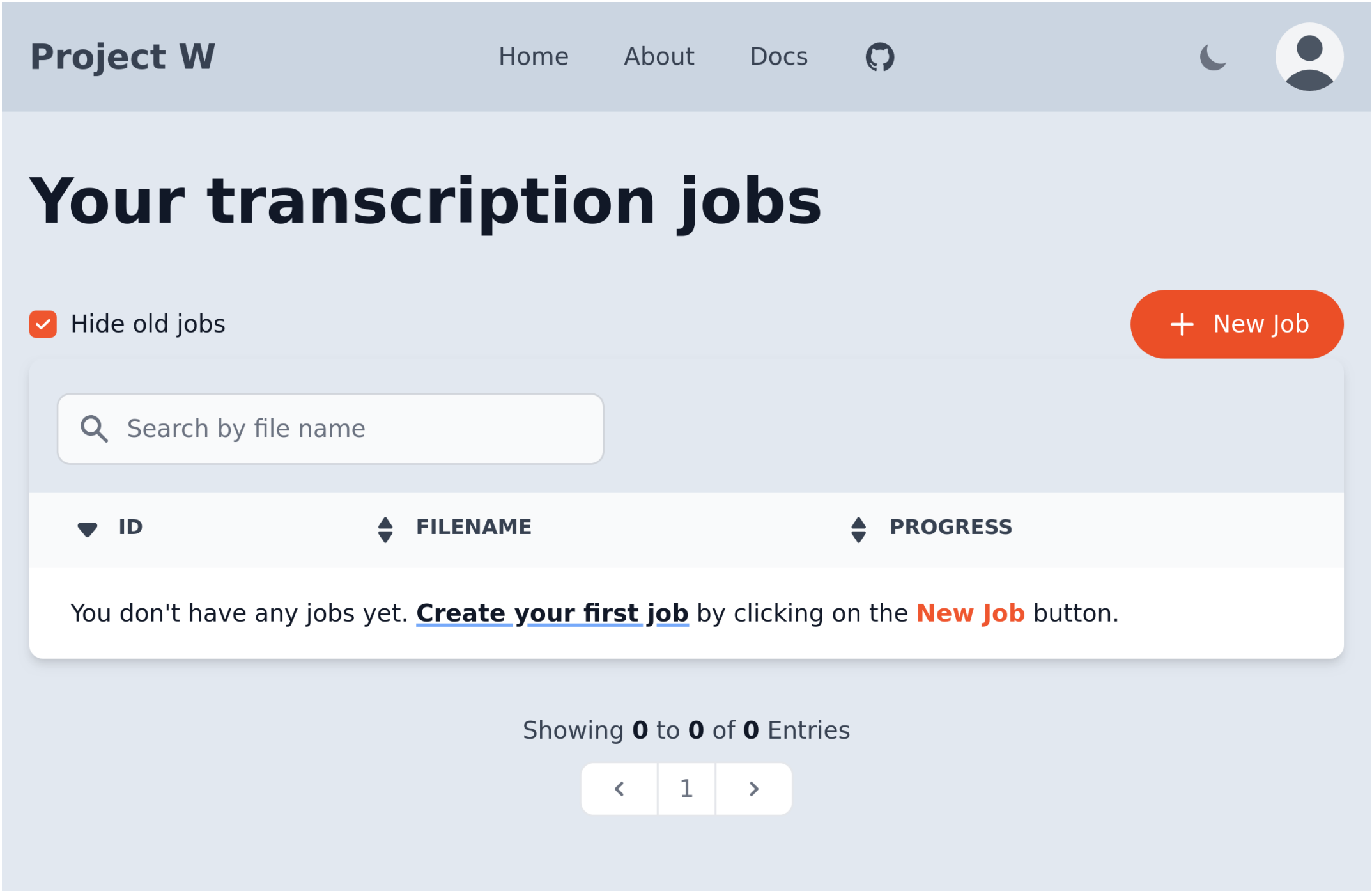
Email address

alice@example.com

Password

••••••••••••••••

Figure 2. Login mask



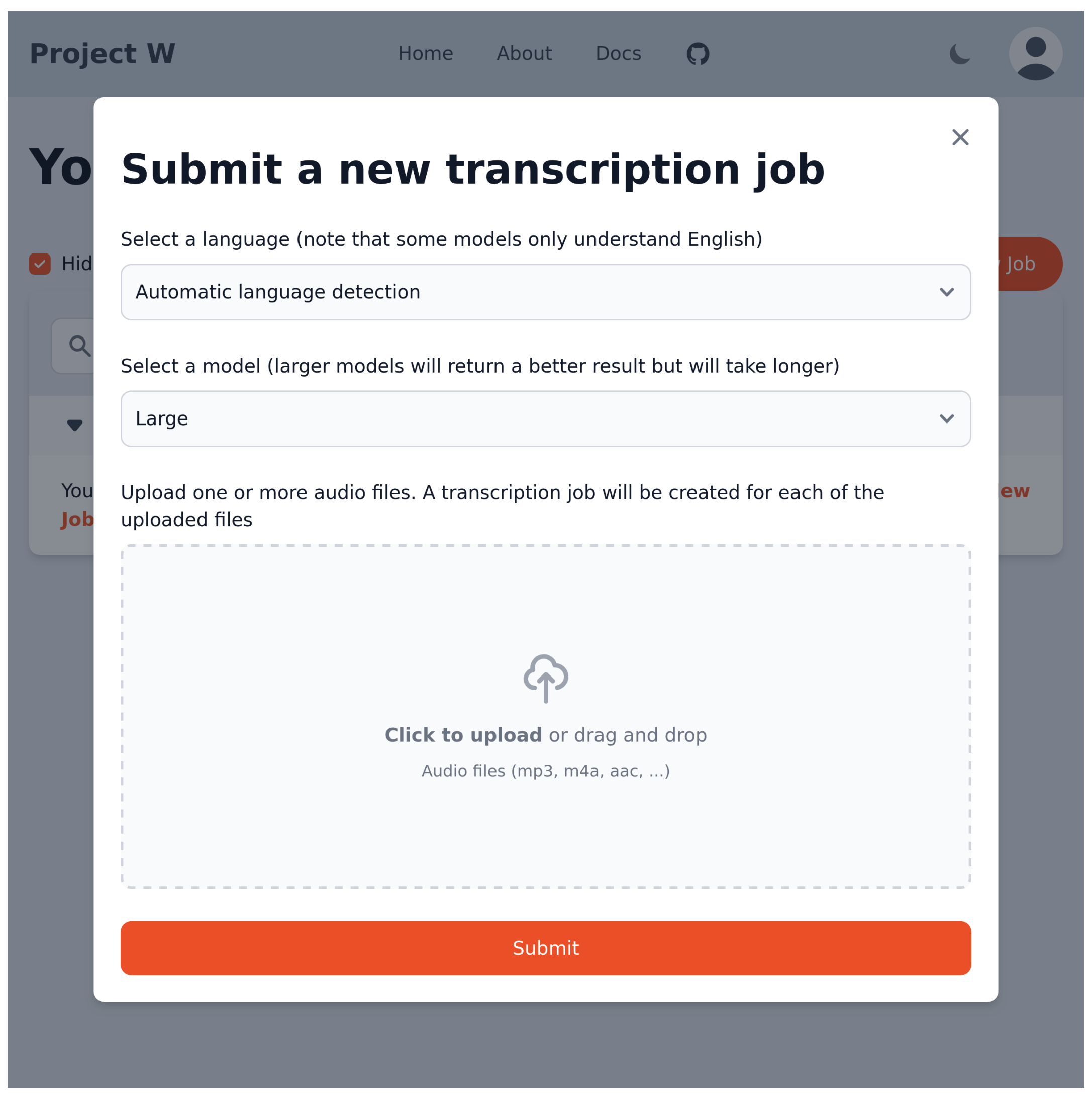
Your transcription jobs

☑ Hide old jobs

ID	FILENAME	PROGRESS
You don't have any jobs yet. **Create your first job** by clicking on the **New Job** button.		

Showing **0** to **0** of **0** Entries

Figure 3. Main screen without any submitted jobs



Submit a new transcription job

Select a language (note that some models only understand English)

Automatic language detection

Select a model (larger models will return a better result but will take longer)

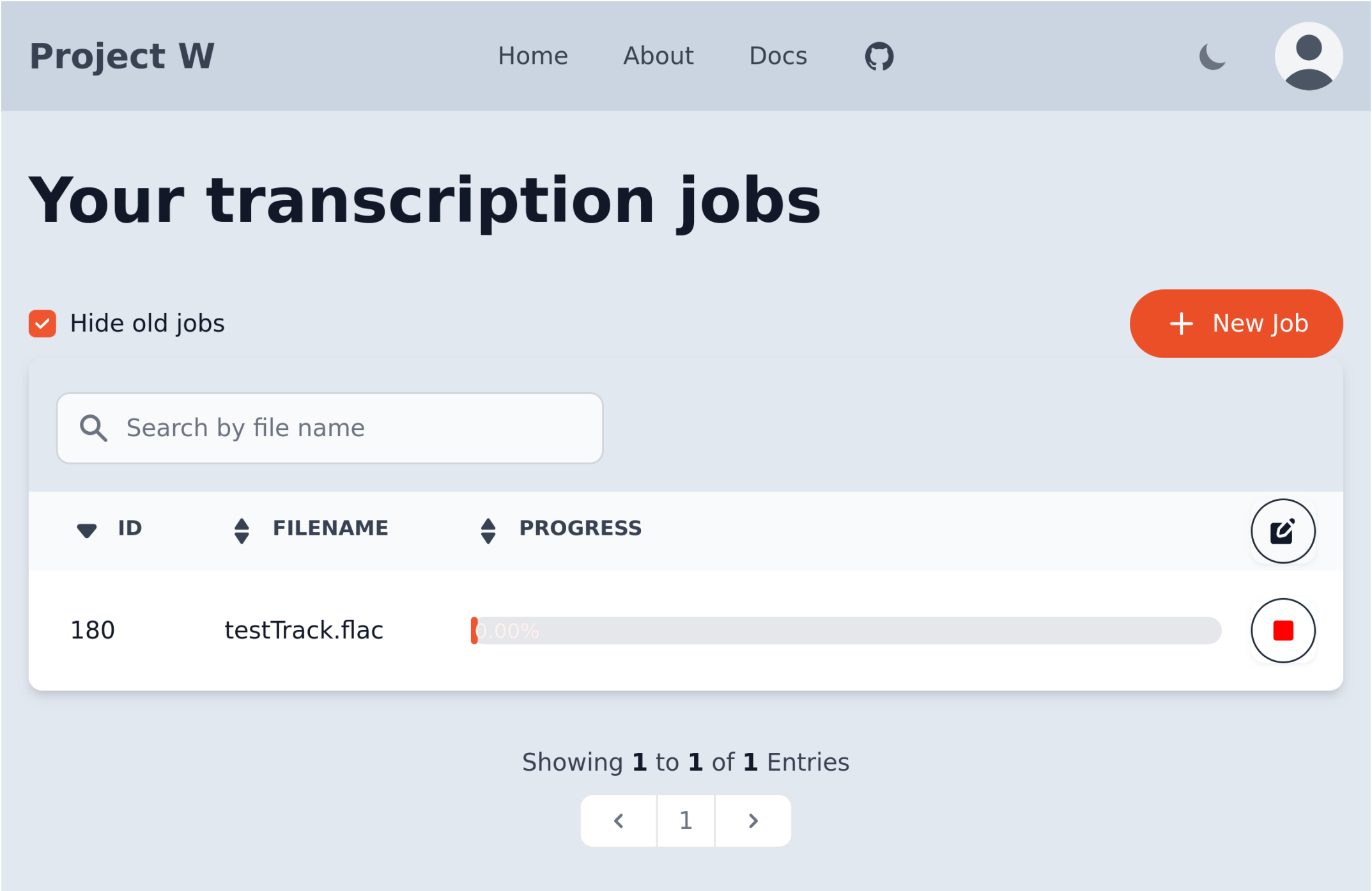
Large

Upload one or more audio files. A transcription job will be created for each of the uploaded files

Click to upload or drag and drop

Audio files (mp3, m4a, aac, ...)

Figure 4. Job submission mask



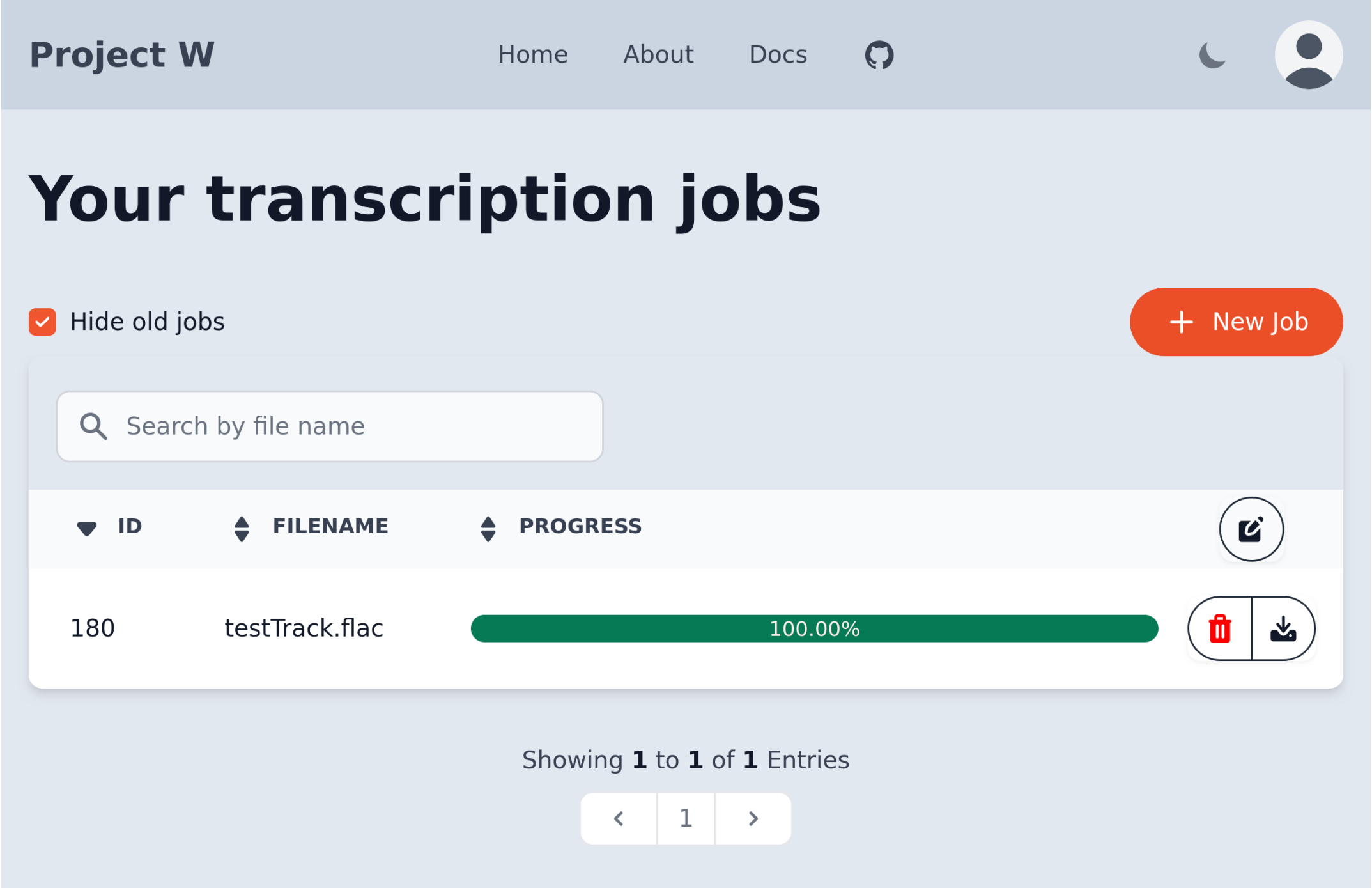
Your transcription jobs

☑ Hide old jobs

ID	FILENAME	PROGRESS
180	testTrack.flac	

Showing **1** to **1** of **1** Entries

Figure 5. Main screen with one job listed that is currently running



Your transcription jobs

☑ Hide old jobs

ID	FILENAME	PROGRESS
180	testTrack.flac	100.00%

Showing **1** to **1** of **1** Entries

Figure 6. Main screen with one job listed that has finished

Give it a try!

(a) `https://project-w.urz.uni-heidelberg.de`

(b) `https://limesurvey.urz.uni-heidelberg.de/index.php/461259`

(c) `https://github.com/JulianFP/project-W`

Figure 7. Links to our current test instance, a feedback survey for the project and the main Github repository

Architecture

Project-W consists of three components (Fig. 1):

- Clients
  - Served to the user's browser over a simple webserver (e.g. nginx)
  - Runs entirely in the browser after that (no nodejs or similar required on the server, just static files)
  - Communicates with the Backend over its REST-API
- Backend
  - REST-API with database and application state
  - HTTP-server, accessible over internet/intranet
  - Manages jobs, runners and authenticates users
  - Users submit their jobs to the Backend, backend then assigns them to runners
- Runners
  - Communicate over HTTP with the Backend as well
  - They download the jobs that the backend assigned to them and execute them
  - Runners execute whisper and do the actual compute
  - HTTP-client only, can stay behind firewall and don't need to be accessible
  - Recommended to have multiple of them (each can only process one job at a time) running on GPU-servers

Technology stack

- Backend (after rewrite)
  - FastAPI web-framework in Python
  - Asynchronous using asyncio
  - PostgreSQL as a central database
  - Redis for caching and keeping the application state (i.e. information about runner)
  - Connects to SMTP server to send emails to users (account confirmation, notifications)
  - Deployable as a docker container using docker compose
  - Kubernetes helm chart planned
  - Configurable over YAML config file
- Client
  - Svelte JS-framework in Typescript
  - Compiles to HTML, CSS and JS files that can be served by any web server as static files
  - Svelte-spa-router for hash-based routing
  - flowbite-svelte and tailwindcss for UI components and styling
- Runner
  - Written in Python with asyncio
  - Executes Whisper transcription
  - Deployable as a docker container, preferable on multiple GPU machines

Hosting requirements

- Backend
  - Low hardware requirment, e.g. VM with some cores
  - Accessible over intranet/internet, open on ports 80/443
  - Domain that points to its IP, some way to get a valid SSL certificate for that domain
- Runners
  - Need to be able to access backend as clients, but no ports need to be opened
  - At least two runners running on GPUs are recommended (although CPU is also possible)

Planned future work

- The backend is currently being rewritten from the ground up
  - Switch from Flask to FastAPI framework
  - Multiple authentication backends with LDAP and OIDC
  - Separate application state from web server into Redis
  - Everything is now asynchronous
  - More efficient, better performance, better code quality
  - Kubernetes for High-availability
- More features
  - Speaker diarization (so that the transcript contains information about who talked when)
  - More advanced transcription settings (temperature, prompt, timestamps, customizable output format)
  - Translation abilities
  - Permission and tag system for who can access which runners,
- Await results of current evaluation and implement feedback
- Stabilize project, get out of alpha stage