



# COMPRISE

Cost effective, Multilingual, Privacy-driven voice-enabled Services

[www.compriseh2020.eu](http://www.compriseh2020.eu)

Call: H2020-ICT-2018-2020

Topic: ICT-29-2018

Type of action: RIA

Grant agreement N°: 825081

**WP N°5: Cloud-based platform for multilingual voice interaction**

**Deliverable N°5.5: Final platform demonstrator and updated data protection and GDPR requirements**

**Lead partner: TILDE**

**Version N°: 1.0**

**Date: 31/05/2021**



<b>Document information</b>	
<b>Deliverable N° and title:</b>	<b>D5.5 - Final platform demonstrator and updated data protection and GDPR requirements</b>
<b>Version N°:</b>	<b>1.0</b>
<b>Lead beneficiary:</b>	<b>TILDE</b>
<b>Author(s):</b>	<b>Askars Salimbajevs (TILDE), Álvaro Moretón (ROOT)</b>
<b>Reviewers:</b>	<b>Thomas Kleinbauer (USAAR), Youssef Ridene (NETF)</b>
<b>Submission date:</b>	<b>31/05/2021</b>
<b>Due date:</b>	<b>31/05/2021</b>
<b>Type<sup>1</sup>:</b>	<b>DEM</b>
<b>Dissemination level<sup>2</sup>:</b>	<b>PU</b>

<b>Document history</b>			
<b>Date</b>	<b>Version</b>	<b>Author(s)</b>	<b>Comments</b>
<b>10/05/2021</b>	<b>0.1</b>	<b>Askars Salimbajevs, Álvaro Moretón</b>	<b>Initial draft version</b>
<b>23/05/2021</b>	<b>0.2</b>	<b>Askars Salimbajevs, Álvaro Moretón</b>	<b>Revised version following the reviewers' comments</b>
<b>31/05/2021</b>	<b>1.0</b>	<b>Akira Campbell, Emmanuel Vincent</b>	<b>Final version revised by the Project Manager and the Coordinator</b>

<sup>1</sup> **R:** Report, **DEC:** Websites, patent filling, videos; **DEM:** Demonstrator, pilot, prototype; **ORDP:** Open Research Data Pilot; **ETHICS:** Ethics requirement. **OTHER:** Software Tools

<sup>2</sup> **PU:** Public; **CO:** Confidential, only for members of the consortium (including the Commission Services)

## Document Summary

This deliverable describes the implementation of the final demonstrator of the COMPRISE Cloud Platform. The document contains an overview of the implemented data collection and curation features, a description of the unified Speech-to-Text (STT) model training recipe and a high-level documentation of the Cloud Platform architecture and the implemented application programming interface (API). Instructions on how to deploy and use the Cloud Platform are provided as well.

The described version of the COMPRISE Cloud Platform allows users to upload, store and manage data of two types: (1) speech and (2) text. For each uploaded audio or text segment, a label or annotation can be added. The current version of the COMPRISE Cloud Platform also implements downloading of trained STT and Spoken Language Understanding (SLU) models, as well as scheduling training of new models. However, only the STT model training recipe — a Unified Training Recipe for the six languages (English, French, German, Latvian, Lithuanian, and Portuguese) of the COMPRISE project — is currently implemented. In the future, support for other types of data and models may be added.

The main users of the COMPRISE Cloud Platform will be developers using the COMPRISE SDK (WP4), which will exchange data and models via a REST API. The COMPRISE Cloud Platform fills a gap in the current ecosystem: existing resource repositories are good for speech resource description, dissemination, sharing, and distribution, but to the best of our knowledge no platform currently facilitates speech data creation, labelling, and curation.

The COMPRISE Cloud Platform also includes a web-based User Interface (UI) which allows users to sign up, perform general operations and access the documentation.

The source code of the COMPRISE Cloud Platform is available on the COMPRISE Gitlab,<sup>3</sup> and it has been deployed as a demonstrator at <https://comprise-dev.tilde.com>.

Finally, this document contains a summary of the updated data protection and GDPR requirements which are made available as a separate document on the COMPRISE website.

---

<sup>3</sup> <https://gitlab.inria.fr/comprise/comprise-cloud-based-platform>

## Table of contents

1. Introduction	6
2. Final Platform Demonstrator	7
2.1. Overview	7
2.2. Data collection and curation features	8
2.3. Unified Training Recipe	8
2.4. COMPRISE Weakly Supervised STT training	10
2.5. Pretrained models and pre-collected data	10
3. COMPRISE Cloud Platform architecture	11
3.1. Overview	11
3.2. API service	13
3.2.1. Authentication and authorisation	13
3.2.2. Data API	14
3.2.3. Model API	17
3.2.4. Model component API	18
3.3. Model training	19
3.4. Web UI	20
4. Deployment	20
4.1. Prerequisites	20
4.2. Configuration	21
4.2.1. API service	21
4.2.2. Web UI	21
4.3. Building Docker images	21
4.3.1. Container registry	21
4.3.2. Building images and pushing to the registry	21
4.4. Deploying on Kubernetes	22
5. How to use	23
5.1. Signing up	23
5.2. Acquiring API keys	24
5.3. Data collection	25
5.4. Data annotation	25
5.5. Sharing data between applications	26
5.6. Model training	27
5.7. Language model adaptation using grammar	29
5.7.1. Preparing a grammar	29
5.7.2. Importing pre-trained models	30

---

5.7.3. Training language model from the grammar	31
6. Summary of updates to Deliverable D5.1	33
7. Conclusion	35

## 1. Introduction

The COMPRISE Cloud Platform is developed within the scope of Work Package 5 (WP5) “Cloud-based platform for multilingual voice interaction”. The objectives of this work package are to:

- bring together the results of WP2, WP3 and WP4 to develop a cloud-based platform to collect the users’ neutralised speech and text data, and curate them.
- provide access to the user-independent Speech-to-Text (STT) and Spoken Language Understanding (SLU) models trained on this data as a service via a web service application programming interface (API).

The current deliverable was developed in the scope of Task T5.4. “COMPRISE platform demonstrator”. The goal of this task is to implement a fully functional platform demonstrator that is populated with a few initial datasets and trained models and deploy it in a working environment for use in other WPs.

This Deliverable D5.5 is an extension to previous deliverables D5.3 “Data collection and curation features of the platform” (submitted to the European Commission on 31/08/2020) and Deliverable D5.4 “Initial Platform Demonstrator” (submitted to the European Commission on 30/11/2020). Deliverable D5.5 includes all content from previous deliverables and introduces new functionalities which have been developed since then, namely:

- COMPRISE Weakly Supervised STT model training (Sections 2.4 and 5.6);
- API for retraining separate parts of the models (Section 3.2.4);
- Language model training on a separate text corpus (Section 5.6);
- Language model adaptation with speech recognition grammars (Section 5.7);
- New pre-trained models and pre-collected data (Section 2.5).

In addition, this deliverable comes with an updated version of the contents of Deliverable D5.1 “Data protection and GDPR requirements” (submitted to the European Commission on 30/05/2019). For the sake of legibility, the updated contents are provided as a separate document on the COMPRISE website,<sup>4</sup> and only a summary is provided here.

The report consists of the following sections. Section 2 describes the final COMPRISE Cloud Platform demonstrator, lists data collection and curation features, pre-collected data and pre-trained models, and describes the Unified Training Recipe for STT model training. Section 3 describes the architecture of the final COMPRISE Cloud Platform demonstrator and provides a high-level description of authentication, authorisation, data collection, model training and platform APIs. Section 4 provides information on the configuration and deployment of the COMPRISE Cloud Platform. Section 5 describes how the COMPRISE Cloud Platform can be used from the developer’s point of view. Section 6 contains a brief summary of changes made to Deliverable D5.1, including updated data protection and GDPR requirements. Finally, Section 7 is devoted to conclusions.

---

<sup>4</sup> <https://www.compriseh2020.eu/deliverables/>

## 2. Final Platform Demonstrator

### 2.1. Overview

Developers of voice-enabled Apps (e.g., a personal assistant) naturally strive to provide the best possible user experience. This can be potentially solved by using domain-specific STT and/or SLU models for the particular usage domain of the App. However, this requires the collection of user data and specific knowledge on how to train STT and/or SLU models. The COMPRISE SDK and the COMPRISE Cloud Platform aim to help developers solve this problem.

The standard workflow is as follows. The developer signs up to the COMPRISE Cloud Platform and acquires an API key by registering their App into the COMPRISE Cloud Platform. Next, the developer embeds this API key into the App and the COMPRISE Client Library collects speech and text data from users of the App.

The domain-specific neutral data collected for each App and each language is grouped into separate corpora: speech data is appended to the App's speech corpus, and text data is appended to the App's text corpus.

The COMPRISE Cloud Platform is provided so that the developer can manage the collected data, process the collected data (e.g., apply machine translation) and train domain-specific user-independent STT and/or SLU models.

Since the collected data can be annotated, developers shall be able to add annotations to the collected corpora themselves or give data annotators employed by the developer's company access to the collected corpora. Data annotators use the COMPRISE Cloud Platform to label domain-specific neutral speech and text data. They are granted access to speech or text corpora by the developers via a specific URL that contains an authorisation key.

Each data annotator can access multiple corpora simultaneously. For speech corpora, data annotators provide a written transcription of each audio recording. For text corpora, data annotators label each user prompt in terms of intent or next dialogue state. The labelled data can then be used for training domain-specific models.

Two types of authentication mechanisms are implemented to provide multiple access levels for different types of users (tenants):

- OpenID Connect authentication for developers and administrators using an external authentication service.
- API key authentication for voice-enabled Apps and data annotators.

This allows the COMPRISE Cloud Platform to (1) improve security by separating frequently used functions from high-privilege access items, and (2) simplify authentication and authorisation for mobile Apps and data annotators.

The COMPRISE Cloud Platform also makes sure that developers and mobile Apps can manipulate only resources that they own. By default, resources created by other accounts are not visible nor accessible. However, there is a mechanism that allows developers to grant other developers access to their collected training data (see Section 5.5).

## 2.2. Data collection and curation features

The following data collection and curation features were developed in the COMPRISE Cloud Platform for developers using COMPRISE in their Apps:

- new developer account registration;
- authentication and authorisation;
- registration of new Apps and API key generation;
- changing or generating new API keys;
- upload and download of speech and text data;
- deleting speech and text data from the COMPRISE Cloud Platform;
- adding or editing annotations for each uploaded audio or text;
- creating a specific URL that allows data annotators to add or edit data annotations;
- invalidating all previous data annotator URLs and generating new ones;
- a Web User Interface (UI) that simplifies access to all of these features.

Data annotators do not have any accounts on the COMPRISE Cloud Platform. They access the COMPRISE Cloud Platform by using a special URL that is provided by the developers. Each such URL gives access to the data collected by each particular App. The URL contains a special “annotation key” of the App, that is generated in such a way that is impossible to guess or predict by a third-party. If compromised, old URLs can be easily replaced with a new one, by re-generating an annotation key. Also, developers can change the key-part of the URL manually.

The following data management features have been implemented for data annotators:

- retrieving speech and text data collected by a particular App;
- adding or editing annotations for each uploaded audio or text;
- a Web UI that simplifies access to all of these features.

## 2.3. Unified Training Recipe

STT model training in the COMPRISE Cloud Platform demonstrator is implemented by a Unified Training Recipe. The recipe is based on Kaldi<sup>5</sup> and performs the following training steps:

- Preparation of pronunciation lexicon;
- Mel frequency cepstral coefficient (MFCC) feature extraction;
- N-gram language model training;
- Hidden Markov model – Gaussian mixture model (HMM-GMM) acoustic model training for the generation of alignments;
- I-vector extractor training and i-vector extraction;
- Data augmentation by speed-perturbation;
- Neural network acoustic model training;
- Compilation of traditional and lookahead decoding graphs.

The entry point of the recipe expects a single argument — a language code in ISO 639-1 format. The recipe is designed to train models for English, French, German, Latvian, Lithuanian and Portuguese languages.

---

<sup>5</sup> Kaldi Speech Recognition Toolkit. <https://kaldi-asr.org/>



The pronunciation lexicon for all words is generated using the grapheme-to-phoneme converter from the open-source ESpeak<sup>6</sup> synthesiser.

Currently, a tri-gram language model is trained on speech segments using the kaldi\_lm language modelling tools. The language model training scripts assumes that

- Spoken noise, if available in transcripts, is annotated in square brackets, e.g., [laughter], [cough], [noise], etc.
- Partially spoken or non-intelligible words, if available in transcripts, are annotated in triangular brackets, e.g., <compu...>.

The acoustic model training script is based on the COMPRISE recipe for the Let's Go dataset,<sup>7</sup> which follows the classic Kaldi nnet3 chain model training recipe. Before performing HMM-GMM training, the data is split into validation and training datasets. Then, the training dataset is split up further into smaller parts. The idea is that initial models do not need large amounts of data and can be trained on smaller datasets, therefore making the training faster. The final HMM-GMM model is a triphone model with Linear Discriminant Analysis – Maximum Likelihood Linear Transform (LDA+MLTT) and is trained on all available training data. Hyperparameters (senone count, number of Gaussians, etc.) are selected depending on the amount of data. Currently, there are two possible hyperparameter configurations:

- “Small” – configuration for datasets smaller than 80 hours.
- “Medium” – configuration for datasets between 80 and 190 hours.
- “Large” – configuration for datasets larger than 190 hours.

Next, the i-vector extractor is trained on a subset of the acoustic model training data. Then this i-vector extractor is used to prepare i-vectors for the whole training set.

The recipe trains a Kaldi chain factorised time delay neural network (TDNN-F) acoustic model. It assumes that at least one GPU is available on the training machine, otherwise the training will fail. There are two configuration files that describe the neural network architecture and training hyperparameters. Depending on the amount of training data, the recipe can choose one or another:

- “Small” – configuration for datasets smaller than 50 hours.
- “Large” – configuration for datasets larger than 50 hours.

Finally, several decoding graphs are created using the trained acoustic and language models. The recipe compiles a baseline Kaldi HCLG graph and also two graphs (HCLr+Gr and HCLr+Gr with composition from ARPA) for a lookahead composition during decoding (this reduces both the graph size and the decoding memory footprint, and thus enables recognition on less powerful hardware).

On average across all languages, the total training time is in the order of 5 hours on a modern compute node with two Nvidia 2080 Ti GPUs and 48 effective CPU threads. It can take up to 3 days on a less powerful platform.

The model is packaged into a single tar.gz file containing the following structure:

- model/tdnn – acoustic model.

<sup>6</sup> eSpeak: Speech Synthesiser. <http://espeak.sourceforge.net/>

<sup>7</sup> <https://gitlab.inria.fr/comprise/speech-to-text-weakly-supervised-learning/-/tree/master/egs/lets-go-15d>

- `model/tdnn_online/conf` – configuration files for Kaldi online decoding.
- `model/tdnn_online/ivector_extractor` – ivector extractor.
- `model/graph` – decoding graph (both for online and offline decoding).
- `model/graph_lookahead` – small decoding graph with a lookahead composition (currently offline decoding only).
- `model/graph_lookahead_arpa` – even smaller decoding graph with a lookahead composition with ARPA (currently offline decoding only).

## 2.4. COMPRISE Weakly Supervised STT training

Classical supervised learning requires speech data annotation which is a very expensive and time-consuming process. Weakly supervised learning techniques significantly reduce the necessary data annotation efforts when compared to classical supervised learning.

The COMPRISE Cloud Platform demonstrator includes an STT model training recipe that utilises the weakly supervised learning approach to train acoustic and language models from the collected unlabelled audio data. For this the “final weakly supervised learning library” is used, which was developed in WP4 of the COMPRISE project and described in Deliverable D4.4 “Final weakly supervised learning library” (Submitted to the European Commission on February 28, 2021).

This training script first performs classic STT supervised learning on annotated data with the Unified Training Recipe. Then an STT error detector is trained on a held-out dataset. This error detector is used together with the initial STT to create weak labels for the unannotated speech data. Final STT acoustic and language models are retrained on both supervised and weakly supervised data.

The COMPRISE Weakly Supervised Training recipe is fully compatible with the Unified Training Recipe and the resulting models are packaged into a tar.gz archive using exactly the same structure. It is also possible to skip the classic supervised learning step if the initial model is already trained using the Unified Training Recipe.

## 2.5. Pretrained models and pre-collected data

The final platform demonstrator is populated with pretrained STT models and pre-collected general speech training data for the 6 languages of the COMPRISE project, which developers can use for a quick start without any domain-specific data. Later, when enough in-domain data is collected, developers can train adapted in-domain models and replace the pre-trained models.

In order to access the pre-collected data, the developer should use the App data sharing feature (a description can be found in Section 3.2.2 and in Section 5.5). Table 1 lists the pre-collected datasets and access credentials.

**Table 1:** Pre-collected general speech training data.

Language	Data source	Application ID	Share key
English	CommonVoice	5ec05f9163883551daebe823	commonvoice-en
French	CommonVoice	5f7ee867ff13b782c70aea0e	commonvoice-fr
German	CommonVoice	5f7ee725ff13b782c70aea0d	commonvoice-de

Latvian	Tilde	5f802f5a4382d263452b58de	lsrc
Lithuanian	Tilde	5f85612bed2d8fee320b5e7d	liepa
Portuguese	CommonVoice	5f7713ed9406e54a21c754e7	commonvoice-pt

There are two types of pre-trained models available in the final COMPRISE Cloud Platform demonstrator: (1) models trained on pre-collected data and (2) pre-trained models from Vosk.<sup>8</sup> Both types of pre-trained models can be accessed using the Model API (Section 3.2.3) and the credentials listed in Table 1 and Table 2.

**Table 2:** Vosk STT models.

Language	Application ID	Share key	Note
English	5faa8210b7540802f1f2832d	vosk-small-en-share	
French	5faa8230b7540802f1f2832e	vosk-small-fr-share	Non-commercial
German	5faa825ab7540802f1f2832f	vosk-small-de-share	
Portuguese	5faa828db7540802f1f28330	vosk-small-pt-share	

While the final platform demonstrator does not provide Text-to-Speech (TTS) model training functionality, we also include pretrained TTS models. This allows us to make COMPRISE SDK packaging smaller and download models only when needed. Similar to STT models, these TTS models can be accessed using the Model API (Section 3.2.3) and the credentials listed in Table 3.

**Table 3:** Pretrained TTS models.

Language	Application ID	Share key	API key
German	609a38a3ddfe91187cf745e5	609a3a26ddfe91187cf745e8	tts-de
French	609a38c1ddfe91187cf745e6	609a3c07ddfe91187cf745e9	tts-fr
English	609a38f9ddfe91187cf745e7	609a3c5addfe91187cf745ea	tts-en

## 3. COMPRISE Cloud Platform architecture

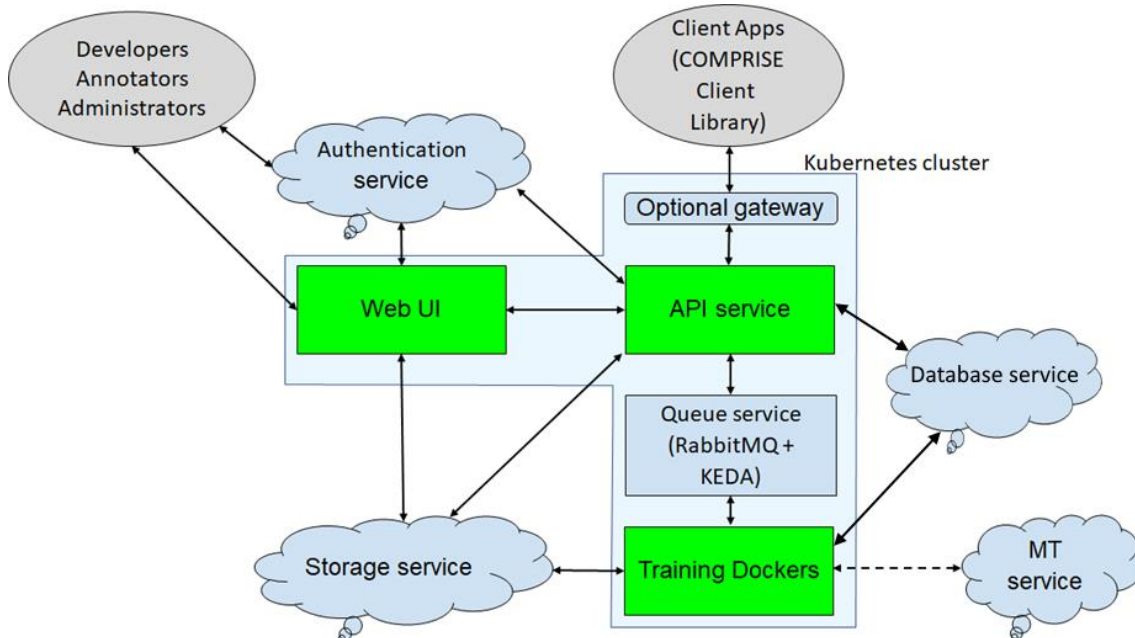
### 3.1. Overview

The COMPRISE Cloud Platform is designed to work in a cloud environment as a collection of web services. As seen in Figure 1, the COMPRISE Cloud Platform consists of four main services:

- An API service which provides an external API for all features of the platform (data collection, model training, etc.)
- Training Docker containers which provide STT and SLU model training functionality. For Machine Translation (MT) of the training data, an external machine translation service will be used: Tilde MT.

<sup>8</sup> Vosk speech recognition toolkit. <https://alphacephei.com/vosk/>

- A Web UI which provides a simple UI for general COMPRISE Cloud Platform functionalities like registering Apps, corpus annotation, triggering model training, etc.
- A queue service based on RabbitMQ<sup>9</sup> and KEDA<sup>10</sup>. This service manages the training job queue and is responsible for starting training jobs on the platform.



**Figure 1:** COMPRISE Cloud Platform architecture.

These services are designed to run as Docker containers in a Kubernetes cluster. Multiple replicas of the API, the training and Web UI services can be run simultaneously. RabbitMQ can also be configured for redundancy and clustering. Such a solution enables COMPRISE Cloud Platform operators to set up a highly available and scalable infrastructure.

There is a Docker image for each training recipe. All training images are based on a “Job Wrapper” image which provides the following functionalities:

- download training data and annotations from the cloud storage service;
- start a `/job.sh` script which is to be overridden with actual training;
- create data directories in Kaldi format if necessary;
- collect trained models and upload them to the cloud storage service.

For future maintainability, the implementation relies on open standards, mainstream technologies, existing cloud services and components, such as Docker, Kubernetes, RabbitMQ and KEDA. The COMPRISE Cloud Platform also relies on the following external services:

- An external authentication service, which authenticates users using the standard OpenID Connect protocol. This enables us to use high-quality existing

<sup>9</sup> <https://www.rabbitmq.com/>

<sup>10</sup> <https://github.com/kedacore/keda>

authentication solutions and cloud providers. Currently, the system is using the Azure B2C service.

- An external storage service, which provides scalable and reliable object storage. Currently, the Azure Blob storage is being used. However, the COMPRISE Cloud Platform is designed so that it does not rely on specific Azure Blob features. The Azure interface code is separated into its own submodule so that it can be easily replaced.
- An external database service, which provides a database for storing App and model metadata. In the platform demonstrator, the Azure Cosmos DB is being used, but the platform can use any other MongoDB database.

To efficiently balance the load between services and avoid unnecessary resource consumption, COMPRISE Cloud Platform API clients upload/download data to/from the storage service directly without the API service acting as an intermediary. This is achieved by using special signed URLs that grant upload/download access for a particular amount of time. This feature is not unique to the Azure Blob storage service. It can also be found in Amazon S3 and other cloud storage services.

The source code of the COMPRISE Cloud Platform is available on the COMPRISE GitLab<sup>11</sup> and is structured as follows:

- `examples/fakejob` – example of a Docker image based on a job wrapper;
- `kubernetes` – example Kubernetes configuration files;
- `src` – API service and training job wrapper source code;
- `web-ui` – Web UI source code.

The source code of the Unified Training Recipe is published in a separate repository in the COMPRISE GitLab.<sup>12</sup>

## 3.2. API service

### 3.2.1. Authentication and authorisation

Authentication is implemented as described in Deliverable D5.2 “Platform hardware and software architecture”.

The API service implements multiple levels of access for the API. These access levels are implemented by having two types of authentication mechanisms:

- OpenID Connect authentication for developers and administrators using an external authentication service;
- API key authentication for mobile Apps and data annotators.

Depending on the authentication method, access to different API methods are provided (see Section 3.2.2 for details). For example, API keys provide mostly read-only access to the COMPRISE Cloud Platform and cannot access data from different mobile Apps and mobile App users. Therefore, if compromised, they cannot be used to modify the Apps or to listen to other user voice queries. Moreover, these keys can be revoked at any time, disabling the access.

<sup>11</sup> <https://gitlab.inria.fr/comprise/comprise-cloud-based-platform>

<sup>12</sup> <https://gitlab.inria.fr/comprise/comprise-stt-training-recipe>

OpenID Connect provides increased security of the COMPRISE Cloud Platform by putting the responsibility for user identity verification in the hands of the dedicated software solutions and most expert service providers.

The COMPRISE Cloud Platform owner can set up an OpenID Connect compatible authentication service either by themselves or by using external service providers. The current version of the COMPRISE Cloud Platform has been tested to work with the Azure B2C authentication service.

However, developers and administrators must sign up through the COMPRISE Cloud Platform login page. Depending on the service provider and configuration, an approval from the COMPRISE Cloud Platform owner might be required.

After successful authentication via OpenID Connect, developers and administrators acquire a JSON Web Token that is added to each API call, and grants access to the COMPRISE Cloud Platform.

Administrators have access to all API methods for all resources on the COMPRISE Cloud Platform, while developers only have access to API methods for resources (registered Apps, speech and text) that they have created. Resources created by other accounts are neither visible nor accessible.

The current COMPRISE Cloud Platform version implements two alternative authorisation methods. The first method uses a text file which stores the mapping between user IDs and roles (administrator, developer, none). The COMPRISE Cloud Platform owner grants access to the COMPRISE Cloud Platform by adding user IDs to this file. An alternative implementation uses the Azure Active Directory service and maps Active Directory user groups for the developer and administrator roles. The COMPRISE Cloud Platform owner grants access to the COMPRISE Cloud Platform by moving users between groups.

Mobile Apps (created by developers) and data annotators authenticate via API keys:

- mobile Apps use an AppKey to access limited COMPRISE Cloud Platform functionality;
- data annotators use an AnnotatorKey to label collected data.

The AppKey or the AnnotatorKey should be supplied to the API calls by appending a query `?api_key=<AppKey or AnnotatorKey>` to the API endpoint URL.

### 3.2.2. Data API

All data collection and curation features can be accessed via a REST API. This API is documented using the OpenAPI specification standard to simplify the adoption of the COMPRISE Cloud Platform by partners and future users.

The API documentation is dynamically generated by the API service from the source code and available through a special URL, e.g., <http://comprise-dev.tilde.com/v1alpha/.well-known/service-desc>. In this section, we provide a high-level API description.

For API versioning each endpoint is prefixed with an API version, e.g., <http://comprise-dev.tilde.com/v1/<endpoint>> or <http://comprise-dev.tilde.com/v2/<endpoint>>. At the moment of writing, the platform demonstrator provides v1alpha API, which is described in the sections below.

Application registration and management are provided by the [/v1alpha/applications](#) endpoint (see Table 4).

**Table 4:** Application management endpoint.

Method and URL	Access	Description
GET /applications	Developer Administrator	Retrieve the list of Apps to which the user has access. Returns: JSON, list of IDs and names.
GET /applications/<id>	Developer Administrator Mobile App (AppKey)	Get metadata for App <id>. Returns: application JSON (see below).
POST /applications	Developer Administrator	Create a new App and specify its metadata. Input: application JSON.
PUT /applications/<id>	Developer Administrator	Update metadata for App <id>. Input: application JSON.
DELETE /applications/<id>	Developer Administrator	Remove App <id>.

Each App is represented by a JSON object, which contains the following metadata:

```
{
  "name" : "string, name of the application",
  "description" : "string, optional description",
  "language" : "string, language of the application (ISO 639-1)",
  "app_key" : "string, API access key for mobile app",
  "annotator_key" : "string, access key for Data Annotators",
  "share_key" : "string, access key for sharing training data ",
  "speech_upload_url" : "string, direct URL for speech segment
upload",
  "text_upload_url" : "string, direct URL for speech segment upload",
  "owner_id" : "string, ID of application owner, automatically
assigned",
  "id" : "string, automatically generated ID of application"
}
```

The uploading of speech and text data are performed via unique upload URLs from the JSON App. These URLs are generated automatically when a JSON App is requested and provide direct access for uploading speech or text data to the storage service. They are valid for a single upload only and should be updated before uploading the next data segment.

The uploaded speech and text segments can be managed via endpoints [/v1alpha/applications/<id>/speech](#) and [/v1alpha/applications/<id>/text](#). Since these endpoints are symmetrical, here we describe only the former (see Table 5).

**Table 5:** Speech data management endpoint.

Method and URL	Access	Description
GET /applications/<id>/speech	Developer Administrator Annotator (AnnotatorKey)	Retrieve list of speech segments in the speech corpus of App <id>. Returns: segment JSON (see below).
GET /applications/<id>/speech/<u >	Developer Administrator Annotator (AnnotatorKey)	Download speech segment <u> or annotations (specified by the query parameter). Returns: segment JSON.
DELETE /applications/<id>/speech	Developer Administrator	Delete utterances from the speech corpus of App <id>. Optional input: array of <u> to delete. If no list is specified, the whole corpus is deleted.
DELETE /applications/<id>/speech/<u>	Developer Administrator	Delete speech segment <u>.

Each speech segment is represented by the following JSON declaration:

```
{
  "annotation_url": "string, direct URL to annotation file",
  "audio_url": "string, direct URL to audio file",
  "id": "string, ID of the speech segment (utt_id)"
}
```

Both URLs are generated automatically upon request and provide direct access to the storage service so that the API service does not act as a proxy during audio downloads. The generated URLs have an expiration time so that only authorised users can access the collected data.

Annotations (transcriptions, SLU labels, etc.) can be added by calling PATCH [/v1alpha/applications/<id>/speech/<utt\\_id>](#) for speech segments or PATCH [/v1alpha/applications/<id>/text/<utt\\_id>](#) for text segments.

For speech segments, the request body is expected to be in JSON format:

```
{
  "text": "string, speech segment transcription"
}
```

For text segments, the request body is expected to be in JSON format as well:



```
{
  "text": "string, intent or other label"
}
```

### 3.2.3. Model API

The collected data can be used to train STT models via API requests to the [/v1alpha/applications/<id>/models/ASR](#) endpoint and SLU models via requests to the [/v1alpha/applications/<id>/models/NLU](#) endpoint. The API allows training of models on data collected by a particular App and also using collected data from other Apps.

Since STT and SLU training endpoints are symmetrical, here we only describe the former (see Table 6).

**Table 6:** STT model training endpoint.

Method and URL	Access	Description
GET /applications/<id>/models/ASR	Developer Administrator Mobile App (AppKey)	Retrieve list of STT models for App <id>. Returns: JSON containing STT model metadata (see below).
GET /applications/<id>/models/ASR /<m>	Developer Administrator Mobile App (AppKey)	Download STT model <m>. Returns: binary model file.
POST /applications/<id>/models/ASR	Developer Administrator	Train new STT model. Input: training recipe JSON
DELETE /applications/<id>/models/ASR /<m>	Developer Administrator	Delete STT model <m>.

Both STT and SLU model descriptions contain the following JSON-encoded metadata:

```
{
  "created": "date, model training request date",
  "id": "string, automatically assigned ID of the model",
  "is_mt": "bool, is this model trained on MT data, automatically assigned",
  "latest": "bool, automatically assigned if model is the latest for the given app",
  "recipe": "string, name of the training recipe used to train a model",
  "status": "string, model status",
  "trained": "date, model training date"
}
```

To request model training, the following JSON-encoded requests should be posted to the STT or SLU endpoint:

```
{
  "additional_corpora": [
    {
      "app_id": "string, ID of the application",
      "share_key": "string, share key"
    }
  ],
  "recipe": "string, name of the training recipe"
}
```

Additional training data from other Apps can be requested via the “additional\_corpora” list, where you can list Apps from which you would like to use the data. For each App in the list, developers should provide the correct “share\_key” which grants access to the data.

Available recipe names are not predefined and depend on each specific configuration of the COMPRISE Cloud Platform. The API service creates a training request in the job queue corresponding to the given recipe. When such a recipe handler is configured, KEDA will start a Kubernetes job which will perform the training. The platform demonstrator is configured with two STT training recipes:

- `fake` – a dummy recipe that does not do any training, used for debugging;
- `base` – performs STT model training using the Unified Training Recipe (see Section 2.3).
- `weaksup` – performs STT model training using the COMPRISE Weakly Supervised Training recipe (see Section 2.4).
- `copy` – copies pre-trained model to your application (see Section 2.5 for the list of pre-trained models).

### 3.2.4. Model component API

By default, models are trained and stored as packages (tar.gz files) containing multiple components. In many cases, the developer would like to download or update components individually without retraining the whole model. For example, models trained with the Unified Training Recipe will have the following components:

- `tdnn` – TDNN acoustic model;
- `tdnn_online` – TDNN acoustic model configured for online recognition;
- `graph_lookahead` – language model prepared for usage in STT. HCLG graph for lookahead decoding. Trained on audio transcripts.
- Etc.

Model components can also provide additional extended functionality. E.g., for the Unified Training Recipe the following extended components are available:

- `lm` – language model trained on separately collected text data.
- `gram` – language model adapted using a special grammar provided by the developer.

Both are prepared as HCLG graphs for lookahead decoding and are ready for direct usage in STT.

All of the above can be achieved using Model component endpoints. Since STT and SLU training endpoints are symmetrical, here we only describe the former (see Table 7).

**Table 7:** STT model component endpoint.

Method and URL	Access	Description
GET /applications/<id>/models /ASR/<m>/<c>	Developer Administrator Mobile App (AppKey)	Download component <c> STT model <m>. Returns: binary model component file.
POST /applications/<id>/models /ASR/<m>/<c>	Developer Administrator	Train component <c> of the STT model <m> Input: optional attachment (multipart/form-data)

Available component names and attachment formats are not predefined and depend on each specific training recipe.

### 3.3. Model training

When a model training request is submitted, a model metadata object is created in the database and the training job is added to the RabbitMQ queue of the specified recipe. For example, if an STT training request with “base” recipe is submitted, it will be scheduled to the queue named “ASR.base”, and an SLU training request with recipe “advanced” will be scheduled to the queue “SLU.advanced”.

It is assumed that there is a Docker image for each recipe, which implements the training. The KEDA autoscaler is configured to monitor specific queues and create Kubernetes jobs with corresponding Docker images to process jobs in these queues. For instance, it will try to create a Kubernetes job “asr-base-abcd” for each item in the monitored queue “ASR.base”. If a training request is submitted to an unconfigured queue, nothing will be executed.

Kubernetes jobs created by autoscaler are scheduled for execution on cluster nodes. Depending on the cluster configuration, new nodes can be dynamically added to the node pool by cluster autoscaler. The COMPRISE Cloud Platform demonstrator is configured to work with fixed node pools.

When a job is successfully allocated to a given node, a pod or container is created from the Docker image of the training recipe and the training process is started. All training images are based on a “comprise-job-wrapper” image which provides the following basic functionality:

- retrieve a training job from the queue;
- download training data and annotations from the cloud storage service;
- start the actual training;
- create data directories in Kaldi format if necessary;
- collect trained models and upload them to the cloud storage service;
- update model metadata.

Additionally, there is “comprise-kaldi-job-wrapper” to ease the development of Kaldi training recipes.

When developing new training recipes based on a COMPRISE job wrapper image, developers should take note of the following assumptions:

- The speech data collected by Apps can be found in `/data/speech`.
- The text data collected by Apps can be found in `/data/text`.
- The speech data shared from other Apps can be found in `/data/add/speech`.
- The text data shared from other Apps can be found in `/data/add/text`.
- There is a convenience script (`/api-server/utils/data_prep.py`) that parses `/data/speech` and creates Kaldi data directories.
  - `/data/kaldi-data-transcribed` – only utterances with transcriptions.
  - `/data/kaldi-data-untranscribed` – only utterances without transcriptions.
  - `/data/kaldi-data-all` – all utterances.
- The entry point of the training script should be `/job.sh`.
- The trained model should be packaged into a single file named `/model.mdl`.
- For future diagnostics, a training recipe can save logs to `/logs.tgz` which will be uploaded to the cloud storage.

### 3.4. Web UI

A Web UI was implemented for the COMPRISE Cloud Platform using the Angular 9 framework and the Angular Material UI component library. The source code also contains a Docker image definition, that creates a Docker container with a NodeJS server that can run the Web UI in any environment (e.g., a Kubernetes cluster).

The UI uses the same API and provides simple access to all COMPRISE Cloud Platform features, such as the registration of mobile Apps, collected data annotation, model training, etc. The Web UI also provides documentation for the COMPRISE Cloud Platform API with easy-to-use try-out forms.

## 4. Deployment

### 4.1. Prerequisites

The platform demonstrator of the COMPRISE Cloud Platform depends on external services that should be set up beforehand:

- an OpenID Connect or OAuth2 authentication service (e.g., Azure B2C);
- a cloud storage service (currently only support for Azure Blob storage is implemented);
- a MongoDB database service;
- the RabbitMQ message broker;
- container registry for Docker containers;
- a Kubernetes cluster.

The configuration of these services is out of the scope of this document.

For a quick start, you can skip the installation process and just use the COMPRISE Cloud Platform demonstrator hosted at <https://comprise-dev.tilde.com/>. Currently, the approval of new accounts is manual, so you will need to contact Tilde to get your account approved.

## 4.2. Configuration

The platform demonstrator is designed to run in a Kubernetes cluster rather than a local computer. Since setting up a Kubernetes cluster is a non-trivial procedure specific to a particular infrastructure and it is out of the scope of COMPRISE, we include only example scripts and configuration files to install the COMPRISE Cloud Platform inside an existing Kubernetes cluster. These examples can be found in the `kubernetes/` directory of the COMPRISE Cloud Platform and COMPRISE STT training recipe repositories.

### 4.2.1. API service

In the example Kubernetes files, the API service is configured by `kubernetes/overlays/example/config.yaml`. This file contains credentials for Azure Blob storage, the RabbitMQ queue, the Mongo database and other settings. It is possible to configure the API service to use files for authorisation and authentication instead of Azure Active Directory. Please see the `config.yaml.example` file which describes all configuration parameters.

### 4.2.2. Web UI

The Web UI configuration is stored in `web-ui/src/environments/environment.prod.ts`.

The contents of `environment.prod.ts` are:

- `production: {true or false}`
- `clientId: {OAuth2 client ID}`
- `redirectUri: {Web UI host URI}`
- `tenantId: {omit, not used}`
- `authority: {URL for OAuth2 login page}`
- `webApiUrl: {URL for API service host}`

## 4.3. Building Docker images

### 4.3.1. Container registry

A registry is a repository for storing container images. After a Docker image is built, it should be pushed into the registry so Kubernetes can download it from the registry server. Set up a container registry service and configure it to be accessible by the Kubernetes cluster. A detailed configuration of the container registry server is out of the scope of this document.

The platform demonstrator consists of multiple Docker images. For each image we provide a Makefile script that performs image building and also pushes to the repository. By default, the script will try to push the image to `comprisedev.azurecr.io`. The Makefiles should be edited to replace this with your own container registry. Also make sure that Docker has the necessary credentials to push images to the specified repository.

### 4.3.2. Building images and pushing to the registry

You can build the API service and job-wrapper Docker images by running the following commands:

```
git clone https://gitlab.inria.fr/comprise/comprise-cloud-based-  
platform  
cd comprise-cloud-based-platform  
sudo make push
```

The Web UI Docker image is built similarly:

```
git clone https://gitlab.inria.fr/comprise/comprise-cloud-based-  
platform  
cd comprise-cloud-based-platform/web-ui  
sudo make push
```

The same also applies to the Docker image for the Unified Training Recipe:

```
git clone https://gitlab.inria.fr/comprise/comprise-stt-training-  
recipe  
cd comprise-stt-training-recipe  
sudo make push
```

#### 4.4. Deploying on Kubernetes

First, make sure you have Helm and kubectl installed and working, then run the following commands:

```
cd kubernetes  
./install_cert_manager.sh  
./install_ingress.sh  
./install_keda.sh
```

This will install:

- a certificate manager, so that the COMPRISE Cloud Platform can work behind HTTPS endpoints;
- an ingress controller (nginx), that will provide routing to the COMPRISE Cloud Platform services;
- KEDA and RabbitMQ, a Kubernetes-based event-driven autoscaler and a message broker for training job queueing.

Directory `kubernetes/base` contains baseline configuration files that should be overridden by files in one of the overlays in `kubernetes/overlays`.

In `kubernetes/overlays/example` you can find an example overlay configuration that sets up an API service and a Web UI, a fake STT training job and enables SSL access to the cluster. Then the COMPRISE Cloud Platform can be deployed as follows:

```
kubectl apply -k kubernetes/overlays/example
```

The example configuration file `kubernetes/train-job.yaml` for running STT training jobs in the Kubernetes cluster, can be found in the COMPRISE STT training recipe repository (<https://gitlab.inria.fr/comprise/comprise-stt-training-recipe>).

This configuration can be applied as follows:

```
kubectl apply -f kubernetes/train-job.yaml
```

## 5. How to use

In the following, we provide step-by-step guidelines on how developers can start using the COMPRISE Cloud Platform.

Consider a developer of a voice-enabled mobile App that wishes to achieve the best possible user experience but lacks expertise in STT or SLU model training. The developer wants to overcome this obstacle using the COMPRISE solution.

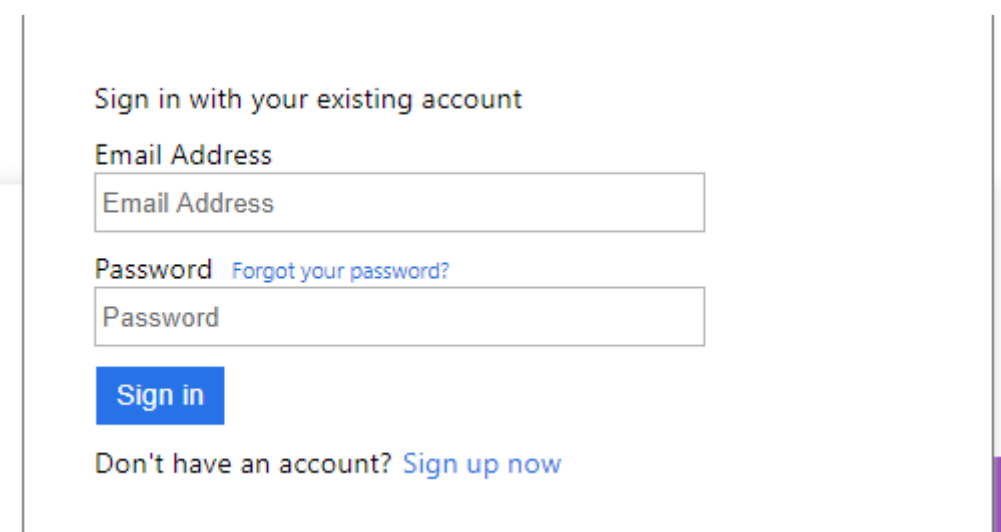
### 5.1. Signing up

To sign up to the COMPRISE Cloud Platform, the developer visits <https://comprisedev.tilde.com> and clicks on “Log In or Sign Up” (see Figure 2).



**Figure 2:** Login page of the Web UI.

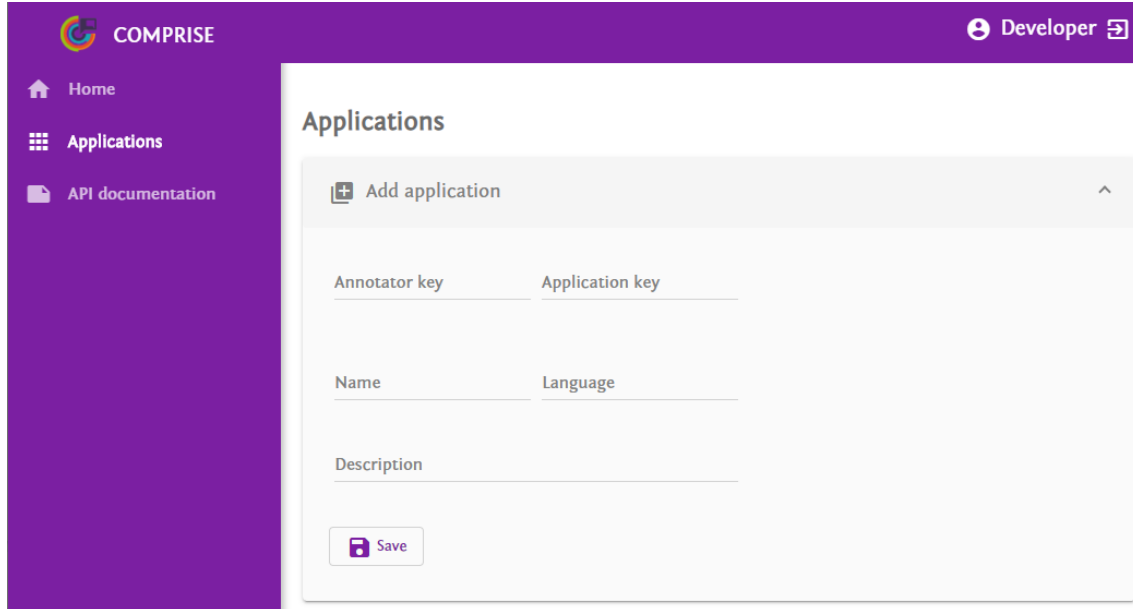
A popup window (see Figure 3) is displayed where the developer creates a new account or logs in using existing account details.



**Figure 3:** Login and sign up popup window.

## 5.2. Acquiring API keys

Once an account is created and the developer has logged into the Web UI of the COMPRISE Cloud Platform, the developer can create an API key for their mobile App. To do so, first the developer needs to register the mobile App in the COMPRISE Cloud Platform. This is done by clicking on “Applications” and then “Add application” (see Figure 4).

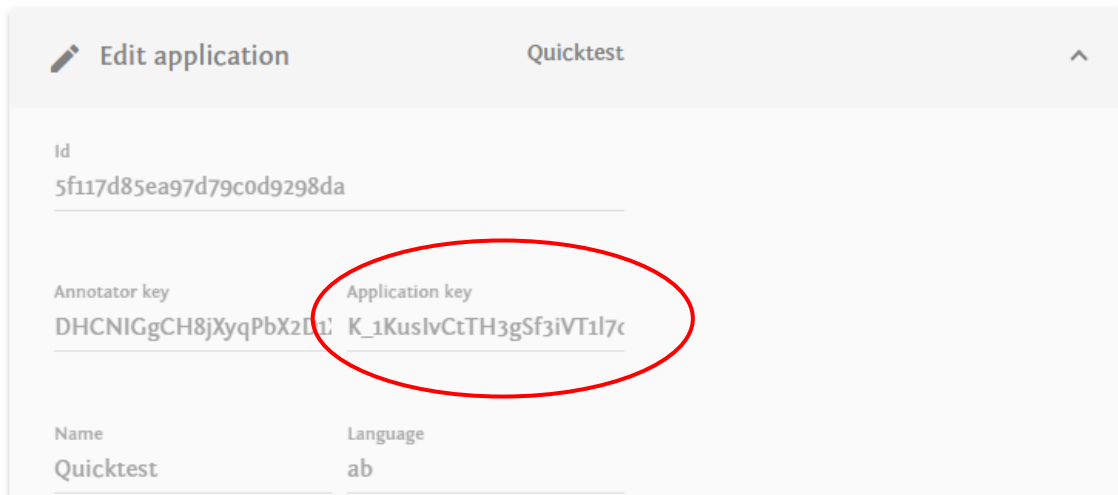


The screenshot shows the COMPRISE web interface. On the left is a purple sidebar with navigation links: Home, Applications, and API documentation. The main content area is titled 'Applications' and contains a form to 'Add application'. The form has several input fields: 'Annotator key', 'Application key', 'Name', 'Language', and 'Description'. A 'Save' button is located at the bottom of the form. The top right of the interface shows the user is logged in as 'Developer'.

**Figure 4:** Registering a new App.

To register an App, the fields “Name” and “Language” must be filled in and the API access keys will be created automatically. Developers can override the automatic generation by manually filling the corresponding fields in the form.

After clicking on “Save”, the new App will be registered. The generated API keys can be viewed by selecting the newly registered App in the list and then clicking “Edit application” (see Figure 5).



The screenshot shows the 'Edit application' form for an application named 'Quicktest'. The form displays several fields with their values: 'Id' (5f117d85ea97d79cod9298da), 'Annotator key' (DHCNIGgCH8jXyqPbX2D1), 'Application key' (K\_1KuslvCtTH3gSf3iVT1l7c), 'Name' (Quicktest), and 'Language' (ab). The 'Application key' field is circled in red, highlighting the automatically generated key.

**Figure 5:** Automatically generated API keys.



### 5.3. Data collection

With the API access keys generated, the developer embeds the generated API key into their mobile App, which uses the COMPRISE Client Library for STT, SLU, and other technologies.

The COMPRISE Client Library automatically uploads privacy-transformed speech and text data to the COMPRISE Cloud Platform using the provided API key.

If the developer does not want to use the COMPRISE SDK for their App, then they must call the COMPRISE Cloud Platform API directly. All necessary information, as well as try-out forms, can be found in the Web UI under “API documentation” (see Figure 6).

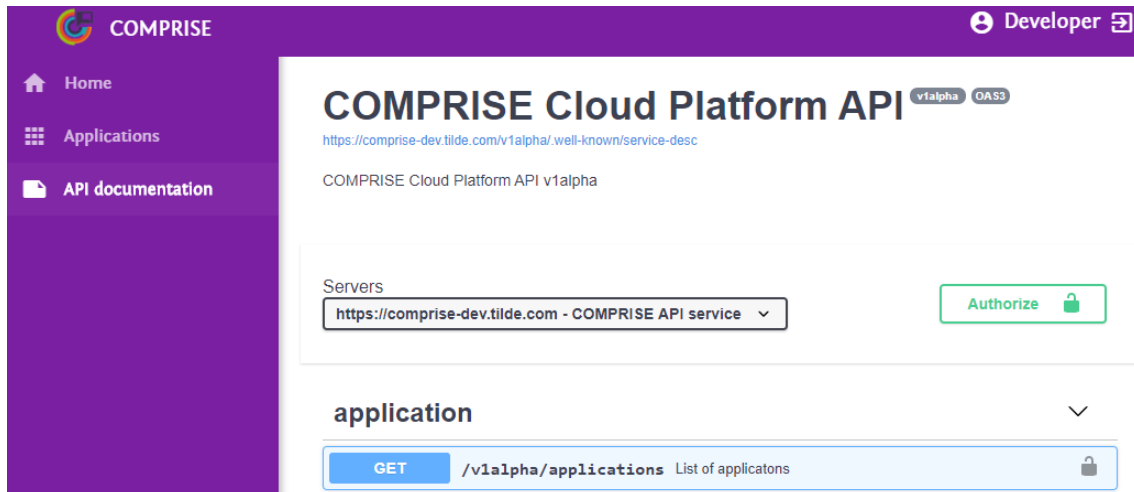


Figure 6: Interactive API documentation.

### 5.4. Data annotation

Once the data collection has started, the developer can use the Web UI and the COMPRISE Cloud Platform to curate the collected data. The list of collected audio samples can be found by selecting a particular App from the list and clicking on the “Speech data” tab (see Figure 7). From this tab, the developer can listen to the collected audio samples, add a transcription or delete samples that contain only noise.

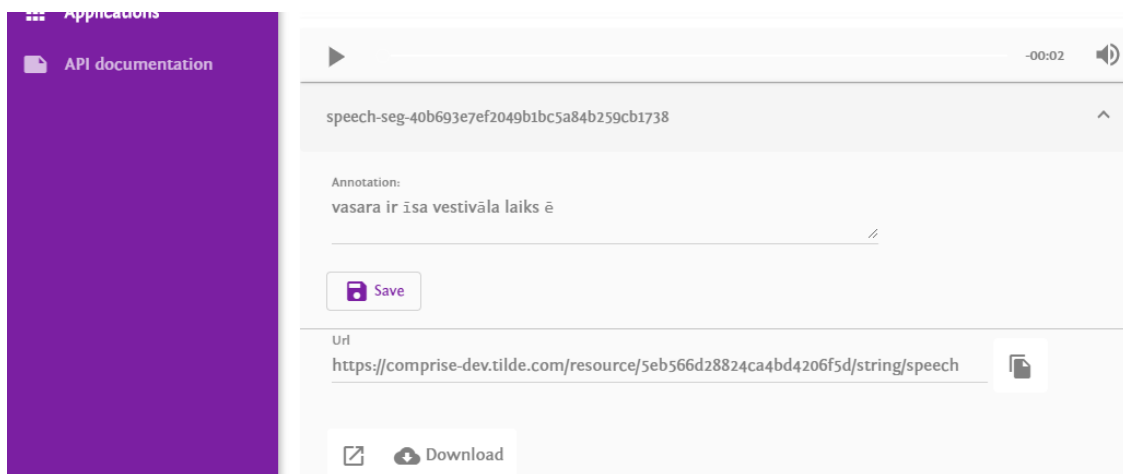
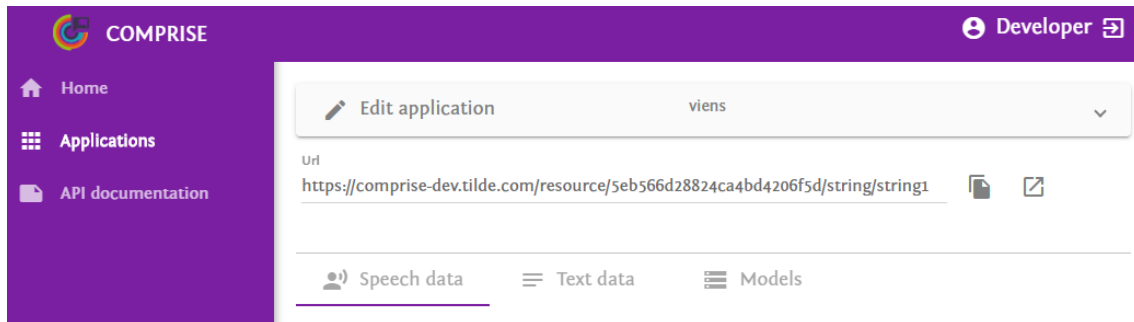


Figure 7: Speech data tab.

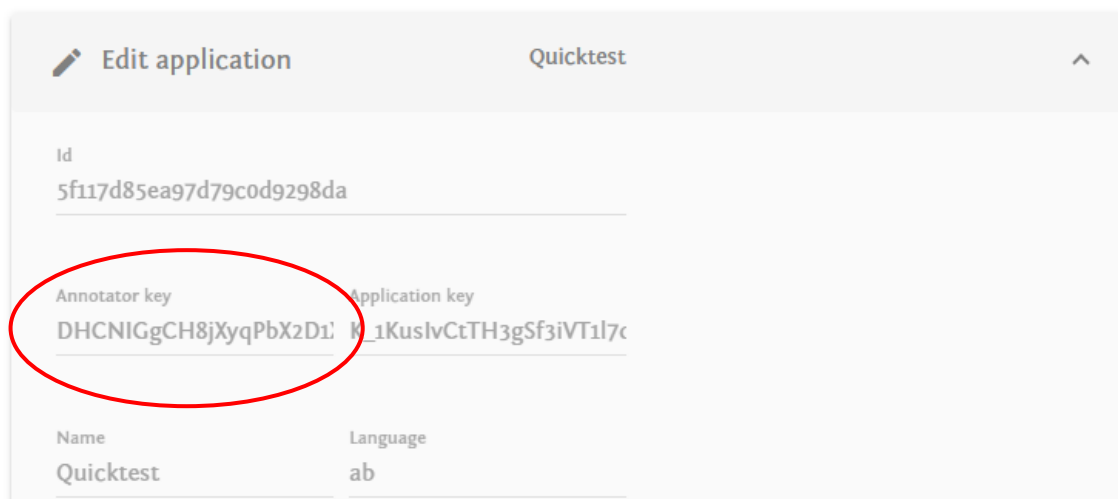
In normal practice, data annotation is not performed by the developer themselves, but by data annotators employed by the developer’s company.

To provide access for the data annotators, the developer grants access to the collected speech and text data by sharing a specific URL with data annotators. This URL is displayed on the App details page (see Figure 8).



**Figure 8:** Shareable URL for Data Annotators.

When required, the URL can be invalidated by changing the AnnotatorKey of the App (see Figure 9). It is not possible to create an empty AnnotatorKey, since leaving this field empty will trigger an automatic key generation.

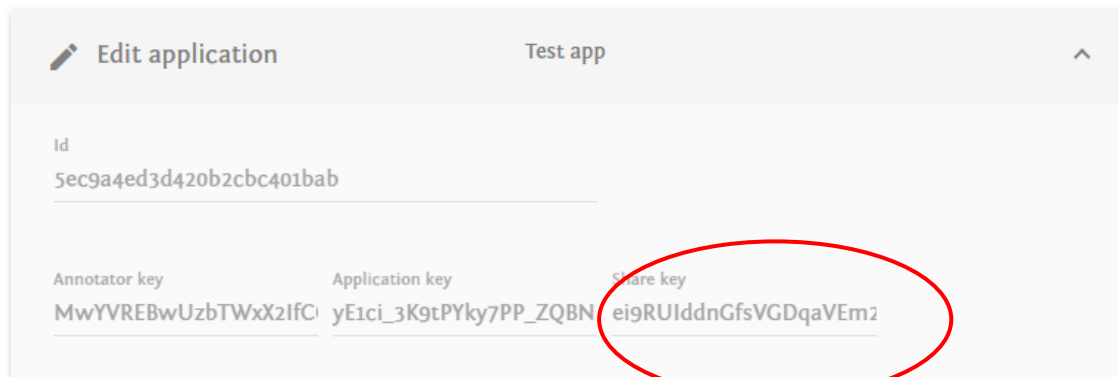


**Figure 9:** Changing annotator key to invalidate shared URLs.

### 5.5. Sharing data between applications

To share the data collected by an App, the developer must set a share key for the App. Providing the share key together with the ID of the App will enable other developers to use the data collected by this App for training other models.

The share key can be set when registering a new App or by editing the existing App (see Figure 10).

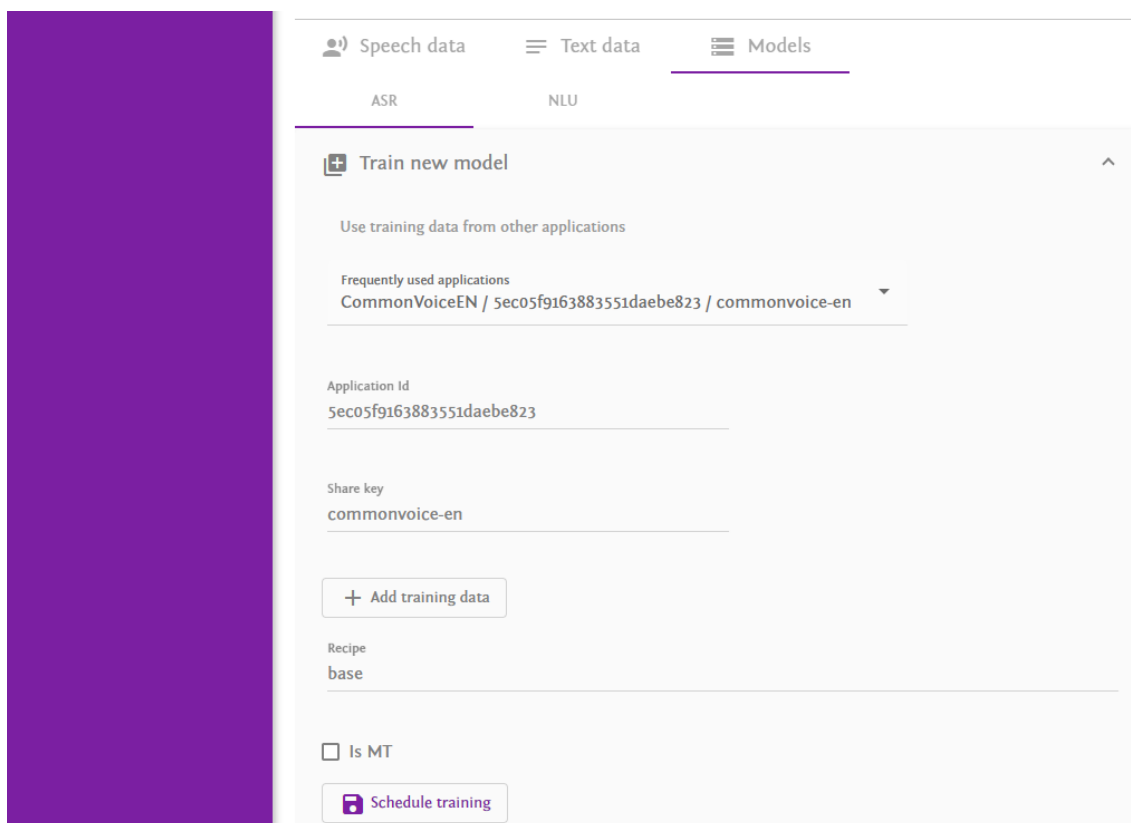


**Figure 10:** Setting the share key.

## 5.6. Model training

Once sufficient data has been collected and annotated, the developer can schedule the training of a new, improved model. To train a new STT model, the developer must select a particular App from the list, click on the “Models” tab and then on “ASR” and then on “Train new model”.

In the model training tab (see Figure 11), the developer can specify additional training data from other Apps to be used for training of the model. This is done by entering the ID and share key of the other App and clicking on “Add training data”.

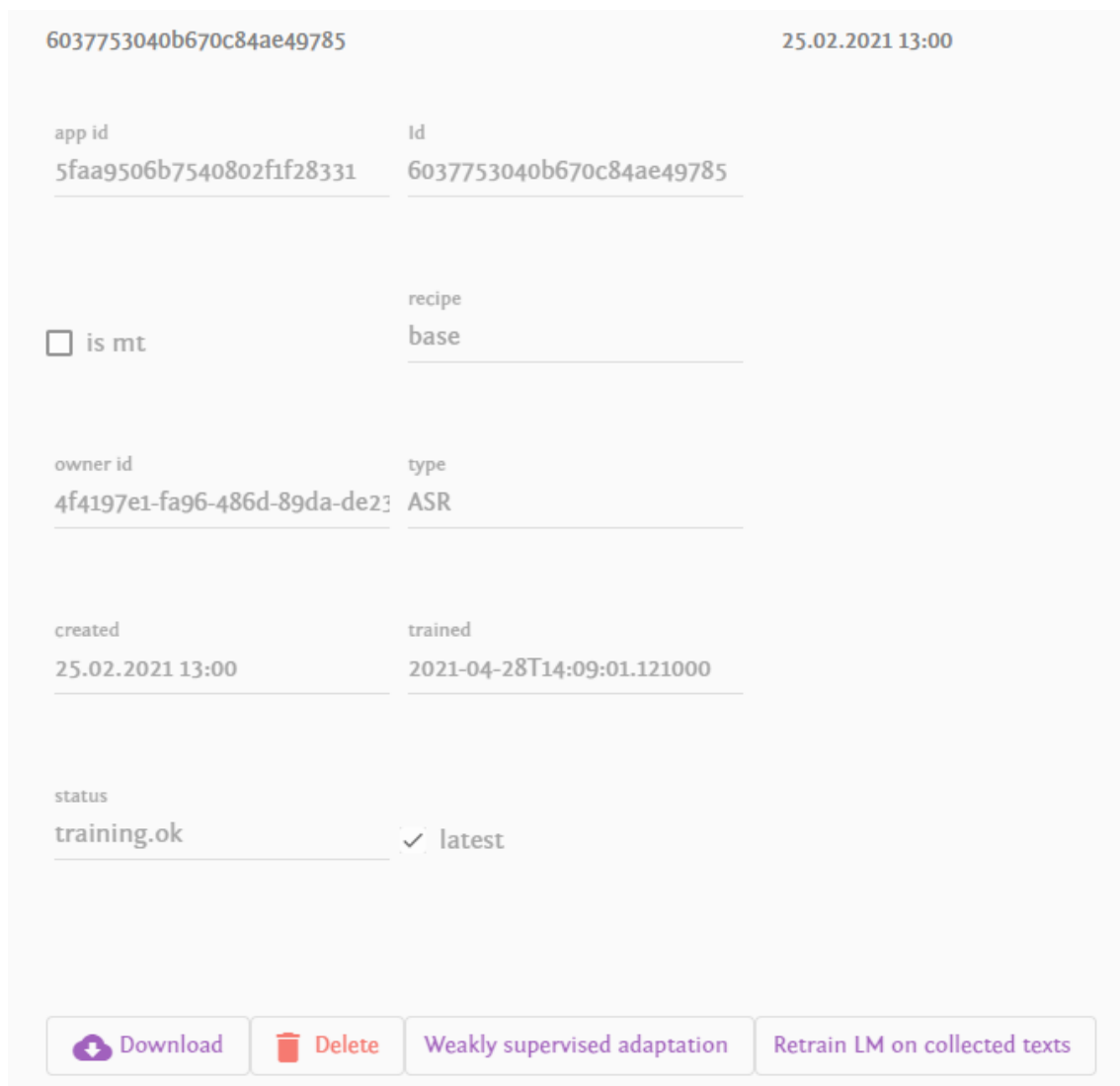


**Figure 11:** Model training tab.

For convenience, the Web UI provides a dropdown list of “Frequently used applications”, which makes it easier to add pre-collected training data (see Section 2.4) to the model training.

To initiate the training, the developer enters the name of the training recipe (the platform demonstrator uses name “base” for the Unified STT model training recipe) and clicks on “Schedule training”. The new training job will be added to the queue.

The progress of the model training can be monitored on the “Models” tab (see Figure 12). When the training is completed, the model can be downloaded directly by the mobile App using the COMPRISE Client Library functions or by calling the COMPRISE Cloud Platform API. It is also possible to download a trained model using the Web UI for manual integration.



6037753040b670c84ae49785		25.02.2021 13:00	
app id	id		
5faa9506b7540802f1f28331	6037753040b670c84ae49785		
<input type="checkbox"/> is mt	recipe		
	base		
owner id	type		
4f4197e1-fa96-486d-89da-de23	ASR		
created	trained		
25.02.2021 13:00	2021-04-28T14:09:01.121000		
status			
training.ok	✓ latest		

Download
Delete
Weakly supervised adaptation
Retrain LM on collected texts

**Figure 12:** Detailed information on a trained model.

The Web UI also provides two ways of adapting an already trained model: (1) language model adaptation on collected text data and (2) weakly supervised adaptation.

The first option trains an n-gram language model on the collected text data (as opposed to using speech data transcripts in the Unified Training Recipe) and prepares the HCLG decoding graph. The adapted decoding graph can be downloaded using the model component API (section 3.2.4).

The second option is to perform weakly supervised adaptation (Section 2.4) of both acoustic and language models using unannotated speech data. After the model adaptation is finished, the updated model can be downloaded using the “Download” button or via the API.

## 5.7. Language model adaptation using grammar

When creating a new mobile App, a developer might not have any collected data at all. For this case the final COMPRISE Cloud Platform demonstrator provides the ability to adapt the STT language model using a speech recognition grammar. The following subsections show how this can be done.

### 5.7.1. Preparing a grammar

A speech recognition grammar contains a set of rules which construct different possible queries that can be recognised by an STT system. JSGF and SRGS are examples of standard speech recognition grammar formats.

A grammar can be composed of a simple set of sentences or a set of constructs that follow a particular syntax and generate multiple sentences. The format for listing simple sentences and more complex constructs are detailed below.

Simple sentences can be listed in the grammar file, one per line, as follows:

```
order a pizza for two
buy two coupons for dinner
reserve a table for four
```

Words which are optional in a sentence can be marked within square brackets []. For example:

```
order [a] pizza for two
buy [two] coupons for dinner
reserve a table [for four]
```

Different alternatives for a particular word can be listed with the | symbol within parentheses (). For example:

```
order a pizza for (two | three | four) people
buy (two | three) coupons for (lunch | dinner)
```

Sentences or grammar constructs in the grammar file can be grouped into separate sections as follows:

```
[Reserve]
reserve a table [for four people]
buy (two | three) coupons for (lunch | dinner)

[Order]
order [a] pizza for (two | three | four) [people]
order a (burger | sandwich) [menu]
```

The section names [Reserve] and [Order] in the above examples are simply placeholders that improve readability and are not used by the STT system.

The sentences and constructs discussed above can be defined as a named construct within a section. Creating a named construct helps to simplify complex constructs and to reuse them within the same section as well as in another section.

A named construct is defined using the = sign and referred using the <> brackets. For example:

```
[Reserve]
place = (table | room)
guests = (two | three | four) [people]
reserve [a] <place> for <guests>

[Order]
order [a] pizza for <Reserve.guests>
```

The sentences and constructs listed in the grammar file will result into a closed set of queries. STT based only on such a closed grammar will fail to recognise words and queries which are not part of the grammar. To handle this, the grammar is mixed with a generic language model (LM). The header of the grammar file specifies the weight given to the fallback LM.

It must be noted that the fallback LM itself is a closed vocabulary model and it may not be able to recognise very specific words like proper names, product names, acronyms, etc.

The first line of the grammar file is expected to be a header defining the weight assigned to the fallback LM. The header line begins with the # symbol and specifies the weight as follows:

```
#mix_coeff=0.1
```

This implies that the fallback LM has a weight of 0.1 and the grammar has a weight of 0.9.

The description of the speech recognition grammar supported by the COMPRISE Cloud Platform and a few grammar examples can be found in the GitLab repository of the Unified Training Recipe.<sup>13</sup>

### 5.7.2. Importing pre-trained models

Normally, you need to train a baseline STT model using general domain data before performing any adaptation. The COMPRISE Cloud Platform demonstrator contains pre-collected data for 6 COMPRISE languages (see Section 2.5). For example, for English the baseline STT models are trained on CommonVoice English data.

However, if every developer on the COMPRISE Cloud Platform would train the same baseline STT models on the same pre-collected data that would be a waste of resources and time. Therefore, a special “copy recipe” is added to the COMPRISE Cloud Platform that copies pre-trained models instead of training from scratch. Copied models can then be adapted as described previously.

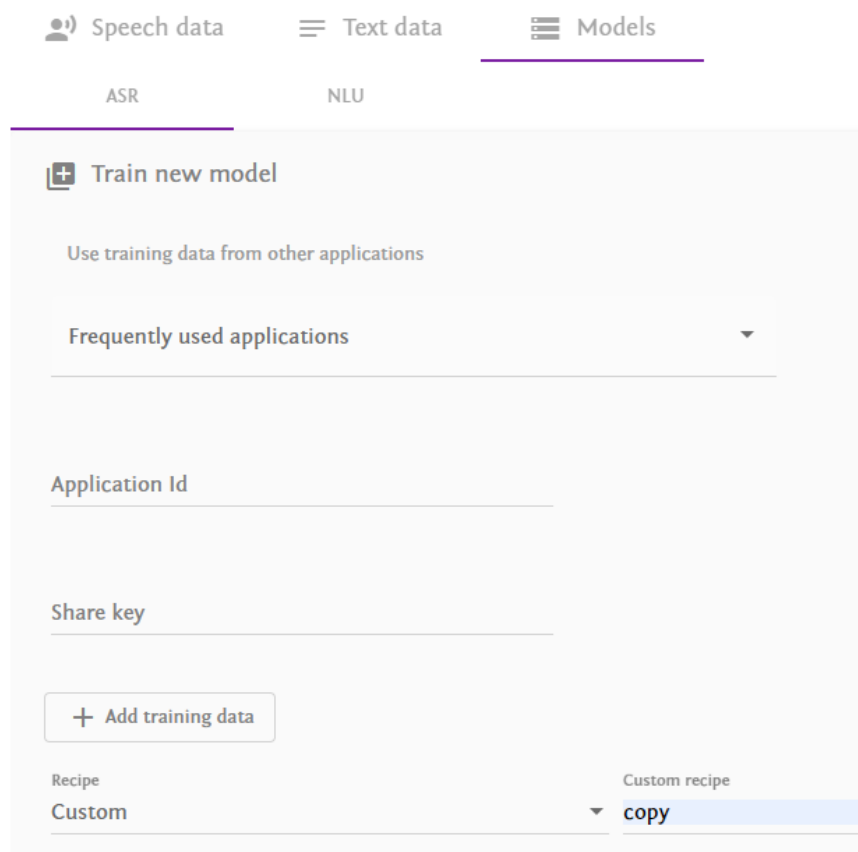
<sup>13</sup> COMPRISE GitLab repository: [https://gitlab.inria.fr/comprise/comprise-stt-training-recipe/-/blob/master/docs/grammar\\_doc.md](https://gitlab.inria.fr/comprise/comprise-stt-training-recipe/-/blob/master/docs/grammar_doc.md)

To copy a pre-trained model to the application, the developer should go to the Models tab of the application and click on “Train new model”. The Model training tab will be displayed (see Figure 13).

To get a model trained on CommonVoice English, the developer should select “CommonVoiceEN” from the “Frequently used applications” list and click “Add training data”. Other pre-trained models can be selected in a similar way. Note that only one dataset can be added, otherwise copying will produce unusable results.

Finally, the developer should choose “Custom” for “Recipe”, and enter “copy” in the “Custom recipe” box and click on “Schedule training”.

After a few minutes, a copy of the pre-trained model will be added to the application.



**Figure 13:** Copying a pre-trained model to a new application.

### 5.7.3. Training language model from the grammar

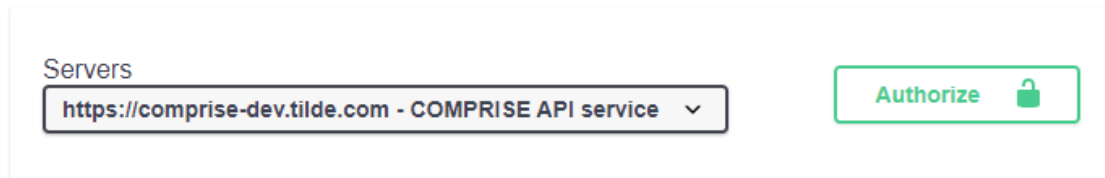
Currently, Web UI does not provide convenient access to training of language models using grammars. However, the developer can make any API call on the API documentation page of the Web UI.

First, the developer should open the API documentation page (<https://comprise-dev.tilde.com/dashboard/api-documentation>) and click on “Authorize” (Figure 14).

# COMPRISE Cloud Platform API v1alpha OAS3

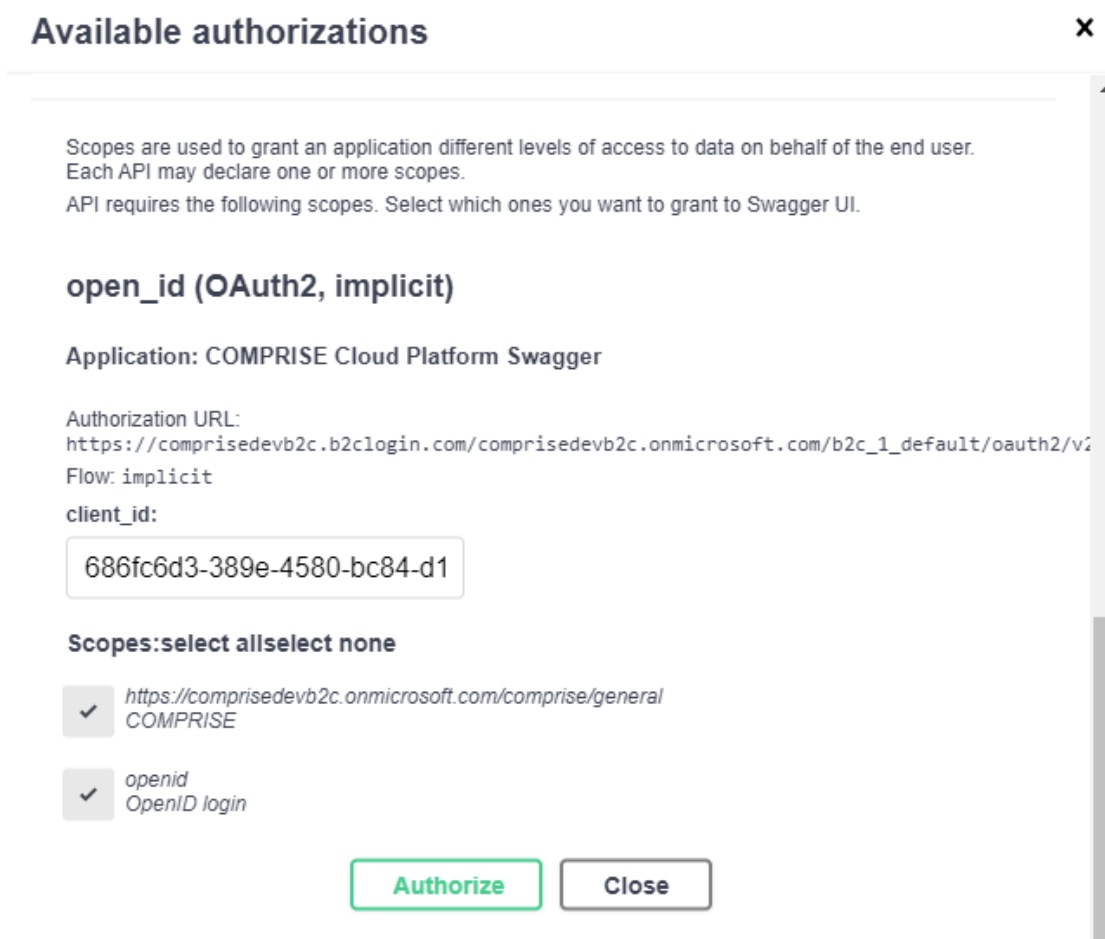
<https://comprise-dev.tilde.com/v1alpha/well-known/service-desc>

COMPRISE Cloud Platform API v1alpha



**Figure 14:** COMPRISE Cloud Platform API swagger.

The language model adaptation requires OpenID authorisation. In the “Available authorization” windows (Figure 15), the developer scrolls down to “open\_id (OAuth2, implicit)”, checks all scopes and presses “Authorize” again. The Web UI will ask to enter the developer’s COMPRISE Cloud Platform login and password to finish the authorisation.



**Figure 15:** API authorisation in the Web UI.

After authorisation the model component API (see Figure 16) can be used to submit a speech recognition grammar for language model adaptation. The developer should enter

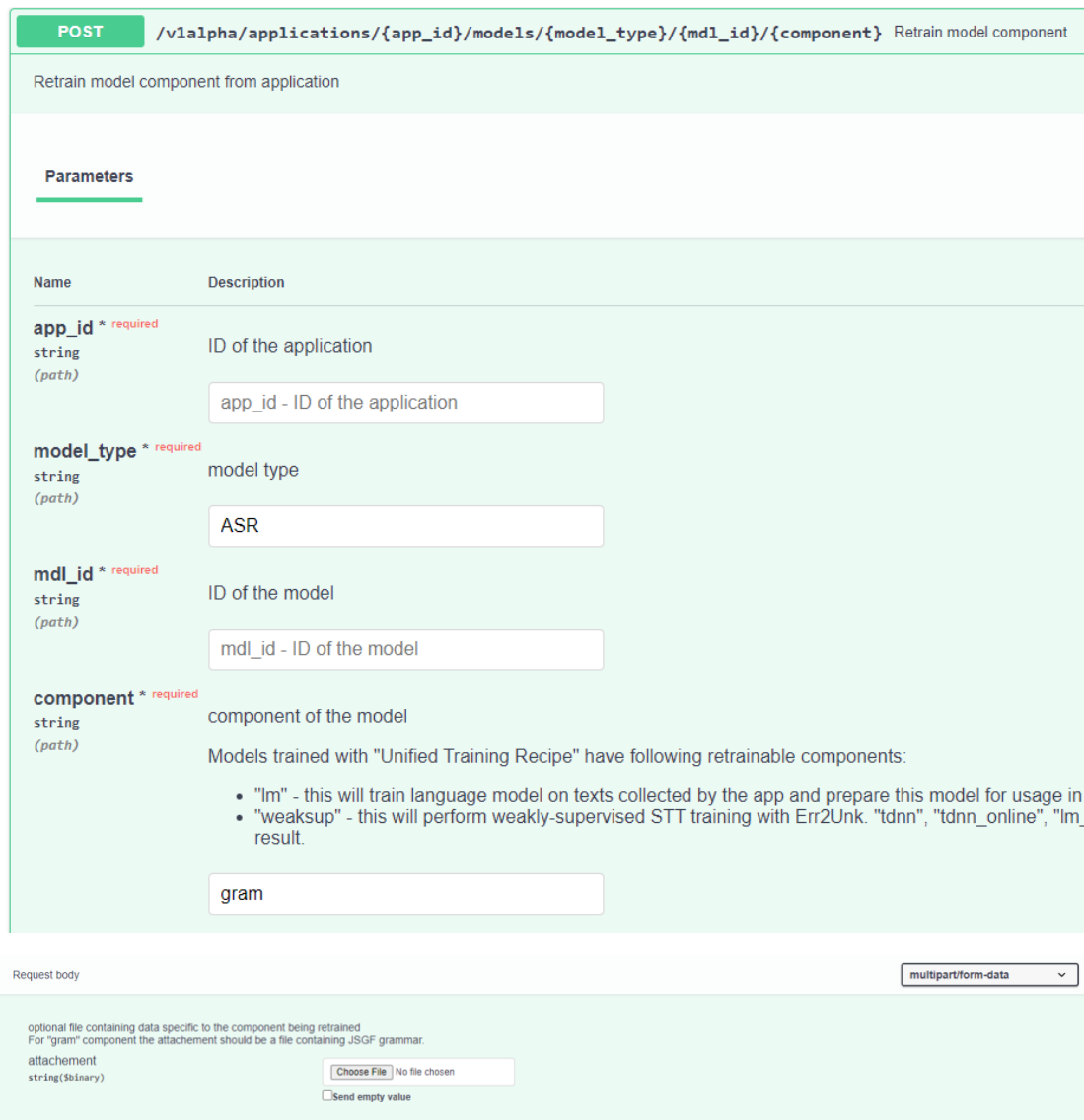


the ID of the application, the ID of the STT model that should be adapted (IDs can be seen on model details tab, see Figure 12), the name of the component to train (“gram”) and add the prepared grammar file to the request body via the “Choose file” button.

Finally, after pressing “Execute” a LM training job will be scheduled on the platform. After the training is finished, the language model can be downloaded using the following API call:

```
/v1alpha/applications/<app_id>/models/ASR/<m>/gram?api_key=<api_key>
```

The result will be a tar.gz archive containing the decoding graph (HCLG.fst) which contains the trained LM and is compatible with the acoustic model <m>. Section 5.2. describes how to obtain the API key for the application.



**POST** /v1alpha/applications/{app\_id}/models/{model\_type}/{mdl\_id}/{component} Retrain model component

Retrain model component from application

**Parameters**

Name	Description
<b>app_id</b> * required string (path)	ID of the application
<b>model_type</b> * required string (path)	model type
<b>mdl_id</b> * required string (path)	ID of the model
<b>component</b> * required string (path)	component of the model Models trained with "Unified Training Recipe" have following retrainable components: <ul style="list-style-type: none"> <li>"lm" - this will train language model on texts collected by the app and prepare this model for usage in s</li> <li>"weaksup" - this will perform weakly-supervised STT training with Err2Unk. "tdnn", "tdnn_online", "lm_</li> </ul> result.

Request body multipart/form-data

optional file containing data specific to the component being retrained  
For "gram" component the attachment should be a file containing JSGF grammar.

attachment  
string(binary) Choose File No file chosen  
 Send empty value

Figure 16: Submitting a model adaptation request.

## 6. Summary of updates to Deliverable D5.1

The updated contents of Deliverable D5.1. “Data protection and GDPR requirements” (submitted to the European Commission on 30/05/2019) are provided as a separate

document on the COMPRISE website.<sup>14</sup> This document provides an in-depth analysis of good practices to successfully implement General Data Protection Regulation (GDPR) principles, based on recent guidelines published by Supervisory Authorities, research work and good practices adopted by voice technology companies.

The GDPR principles and their application to voice technologies are enriched with use cases. In this sense, some scenarios provided by the French Commission Nationale de l'Informatique et des Libertés (CNIL) and the European Data Protection Board (EDPB) where personal data processing operations through voice technologies and the legal bases for each of them are analysed. The document also reviews scenarios where voice-enabled systems record individuals without their consent or any other lawful legal basis.

Readers will find helpful the list of the categories of data that voice assistants and voice Apps may collect, the information that could be extracted from each of them, and the recommendations provided on how to inform the data subject on this matter properly. Common purposes for the processing of personal data through voice assistants or voice Apps are included in the document as well.

The importance of training voice-enabled systems with diverse datasets to avoid biases and comply with the fairness principle and good practices implemented by voice technologies companies to minimise personal data processing through voice systems (e.g., wake word detection, guest modes, etc.) are also explored.

Readers will also encounter some functionalities that voice assistants provide to facilitate the exercise of data subject's rights, such as deleting their recordings or revoking permissions to third parties. The identification of the data controller and data processor in a voice-enabled systems' ecosystem where parties usually fulfil more than one role is also analysed through multiple use cases presented by the CNIL.

Based on this document, it can be concluded that several practices of voice technology companies aren't compliant with the GDPR. However, over the past years, good practices and technical solutions to improve this flaw have emerged. The document presents mechanisms and recommendation to comply with the GDPR principles when personal data is processed through voice technologies.

The document also deepens on the concepts of privacy by design and by default, adopting a more practical approach than its predecessor. It presented recommendations for engineers on how to comply with privacy by design and using anonymisation as a privacy by design technique. Regarding the latter, stress was placed on the importance of performing a re-identification risk assessment to determine whether the risk of re-identification of the data subject is tolerable or not and, consequently, decide on how data should be treated, as personal data or non-personal data. The particular scenario of COMPRISE, where anonymisation is automatic, was also studied, and a series of recommendations to decrease the risk of re-identification (e.g., expanding the number of identifiers and quasi-identifiers the solution could remove or substitute) are presented.

The document analyses changes in the international transfer of data caused by the elimination of the Privacy Shield, which affects voice technology companies in Europe, and ends with a list of cybersecurity threats, including attacks specifically designed to affect voice technologies, and how users, manufacturers and developers could reduce the risks to be affected by them.

---

<sup>14</sup> <https://www.compriseh2020.eu/deliverables/>

It also presents several concerns revolving ethics in the use of voice-enabled systems, which employs technologies such as Artificial Intelligence and Machine Learning that, improperly used, could result in severe consequence to the data subject's privacy (e.g., oversharing of personal information). Issues such as the ownership of the data collected through voice-based systems/apps, etc. or the existence of biases leading to discriminatory treatments, as well as the European initiatives to address ethics are studied.

Lastly, several threats associated with voice-enabled systems (e.g., deepfakes, Surfing Attacks) are analysed together with measures different stakeholders (e.g., device manufacturers) and users can implement to reduce the risks of being targeted by attackers.

## 7. Conclusion

This deliverable presents the final demonstrator of the COMPRISE Cloud Platform. The COMPRISE Cloud Platform is designed as multiple services running in a Kubernetes cluster. For authentication and storage, the COMPRISE Cloud Platform relies on external cloud services.

The COMPRISE Cloud Platform demonstrator provides data collection and curation features, model storage, STT model training for six languages (English, French, German, Latvian, Lithuanian and Portuguese) using a Unified Training Recipe and a Web UI. The demonstrator is deployed online at <https://comprise-dev.tilde.com>.

The source code of the COMPRISE Cloud Platform is freely available on the COMPRISE GitLab.<sup>15</sup> The source code of the Unified Training Recipe is published in a separate repository on the COMPRISE GitLab<sup>16</sup>. Both repositories also contain deployment documentation and example Kubernetes configuration files.

---

<sup>15</sup> <https://gitlab.inria.fr/comprise/comprise-cloud-based-platform>

<sup>16</sup> <https://gitlab.inria.fr/comprise/comprise-stt-training-recipe>