COMPRISE

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

www.compriseh2020.eu

Call: H2020-ICT-2018-2020
Topic: ICT-29-2018
Type of action: RIA
Grant agreement Nº: 825081

WP Nº5:	Cloud-based platform for multilingual voice interaction
Deliverable Nº5.4:	Initial Platform Demonstrator
Lead partner:	TILDE
Version Nº:	1.0
Date:	30/11/2020

European Commission

Document information	
Deliverable Nº and title:	D5.4 - Initial platform demonstrator
Version Nº:	1.0
Lead beneficiary:	TILDE
Author(s):	Askars Salimbajevs, Raivis Skadiņš (TILDE)
Reviewers:	Thomas Kleinbauer (USAAR), Youssef Ridene (NETF)
Submission date:	30/11/2020
Due date:	30/11/2020
Type[1]:	DEM
Dissemination level[2]:	PU

Document history

Date	Version	Author(s)	Comments
10/11/2020	0.1	Askars Salimbajevs, Raivis Skadiņš	Initial draft version
17/11/2020	0.2	Askars Salimbajevs, Raivis Skadiņš	Revised version following the reviewers' comments
30/11/2020	1.0	Emmanuel Vincent, Akira Campbell	Final version reviewed by the Coordinator and the project manager

[1] **R**: Report, **DEC:** Websites, patent filling, videos; **DEM:** Demonstrator, pilot, prototype; **ORDP:** Open Research Data Pilot; **ETHICS:** Ethics requirement. **OTHER:** Software Tools
[2] **PU:** Public**; CO:** Confidential, only for members of the consortium (including the Commission Services)

Document Summary

This deliverable describes the implementation of the initial 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 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,[3] and it has been deployed as an initial demonstrator at https://comprise-dev.tilde.com.

[3] https://gitlab.inria.fr/comprise/comprise-cloud-based-platform

Table of contents

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 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. “Initial 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.

For the sake of completeness, this reports briefly recalls Cloud Platform functionalities which had been presented in Deliverable D5.3 “Data collection and curation features of the platform” (submitted to the European Commission on 31/08/2020), and introduces in more detail new functionalities which have been developed since then.

The report consists of the following sections. Section 2 informally describes the initial COMPRISE Cloud Platform demonstrator, lists data collection and curation features, pre-collected data and pre-trained model, and describes the Unified training recipe for STT model training. Section 3 describes the architecture of the initial 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 Cloud Platform. Section 5 describes how the COMPRISE Cloud Platform can be used from the Developer’s point of view. Finally, Section 6 is devoted to conclusions.

2. Initial 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 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 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 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 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 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

Speech-to-Text (STT) model training in the initial platform demonstrator is implemented by a Unified training recipe. The recipe is based on Kaldi[4] 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.

The pronunciation lexicon for all words is generated using the grapheme-to-phoneme converter from the open-source ESpeak[5] 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,[6] 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

[4] Kaldi Speech Recognition Toolkit. https://kaldi-asr.org/
[5] eSpeak: Speech Synthesiser. http://espeak.sourceforge.net/
[6] https://gitlab.inria.fr/comprise/speech-to-text-weakly-supervised-learning/-/tree/master/egs/letsgo-15d

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.
- 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. Pretrained models and pre-collected data

The initial platform demonstrator is populated with pretrained speech recognition 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: Application management endpoint.

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

Pre-trained models can be accessed using the Model API (Section 3.2.3) and the credentials listed in Table 2.

Table 2: Application management endpoint.

Language	Application ID	AppKey
English	5faa8210b7540802f1f2832d	vosk-small-en
French	5faa8230b7540802f1f2832e	vosk-small-fr
German	5faa825ab7540802f1f2832f	vosk-small-de
Latvian	5faa9506b7540802f1f28331	latvian-small-stt
Lithuanian	5faa9539b7540802f1f28332	lithuanian-small-stt
Portuguese	5faa828db7540802f1f28330	vosk-small-pt

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.
- A Web UI which provides a simple UI for general Cloud Platform functionalities like registering Apps, corpus annotation, triggering model training, etc.

- A queue service based on RabbitMQ[7] and KEDA[8]. This service manages the training job queue and is responsible for starting training Docker containers when training is requested.

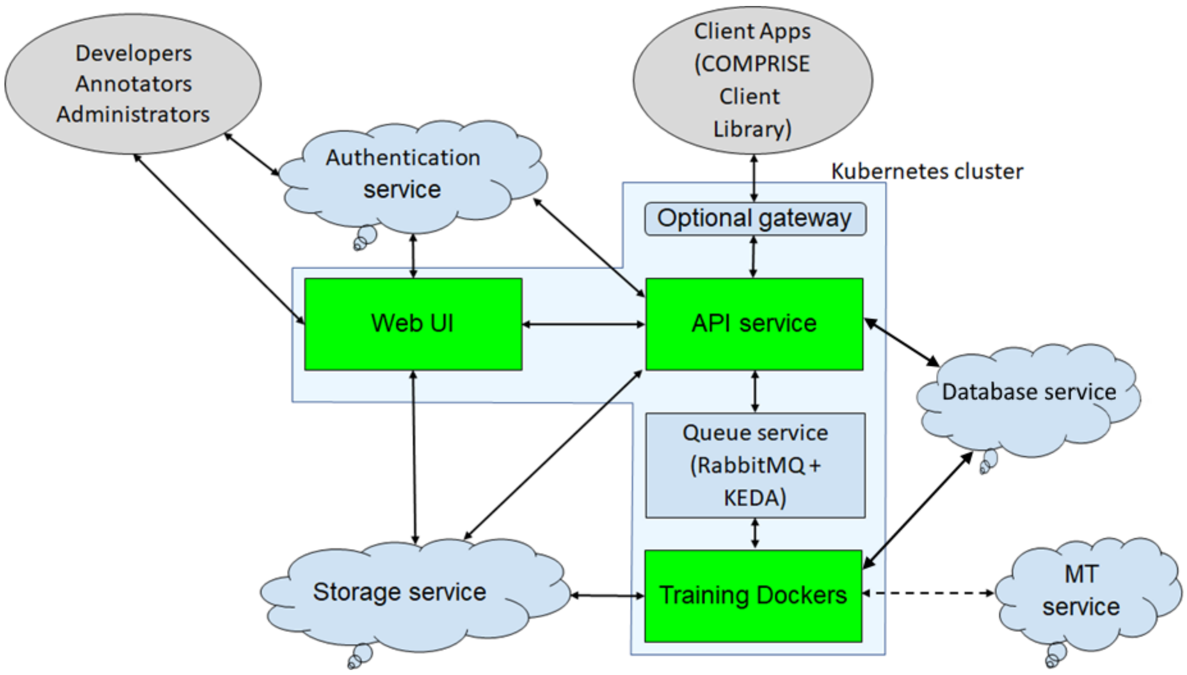


Figure 1: Cloud Platform architecture.

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

[7] https://www.rabbitmq.com/
[8] https://github.com/kedacore/keda

- 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 initial 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, 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[9] 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[10].

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 is provided (see Section 3.2.2 for details).

The COMPRISE Cloud Platform owner can set up an OpenID Connect compatible authentication service either by themselves or by using external service providers. 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. The current version of the

9 https://gitlab.inria.fr/comprise/comprise-cloud-based-platform
10 https://gitlab.inria.fr/comprise/comprise-stt-training-recipe

COMPRISE Cloud Platform has been tested to work with the Azure B2C authentication service.

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 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 initial 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 3).

Table 3: 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 up-
load",
  "text_upload_url" : "string, direct URL for speech segment upload",
  "owner_id" : "string, ID of application owner, automatically as-
signed",
  "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 4).

Table 4: 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/<utt_id>	Developer Administrator Annotator (AnnotatorKey)	Download speech segment <utt_id> 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 <utt_id> to delete. If no list is specified, the whole corpus is deleted.
DELETE /applications/<id>/speech/<utt_id>	Developer Administrator	Delete speech segment <utt_id>.

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 5).

Table 5: 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 using corpus "/applications/<id>/speech". 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 initial 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).

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 cluster configuration, new nodes can be dynamically added to the node pool by cluster autoscaler. The initial 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 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 initial 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 initial 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 initial 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 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 initial 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 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-plat-
form
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-plat-
form
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-rec-
ipe
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 https 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 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://comprise-dev.tilde.com and clicks on "Log In or Sign Up" (see Figure 2).

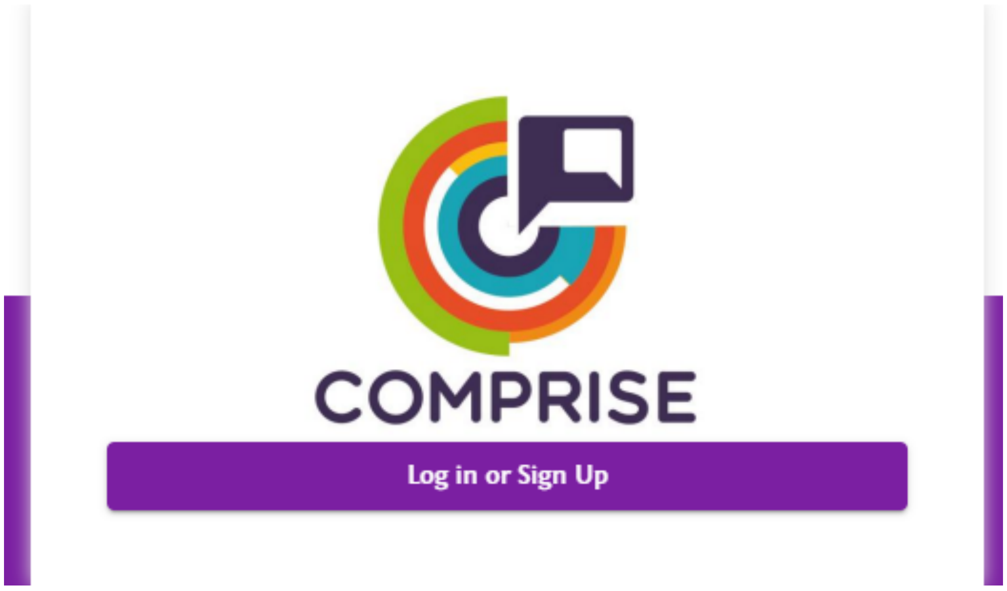


Figure 2: Login page of the Web UI.

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

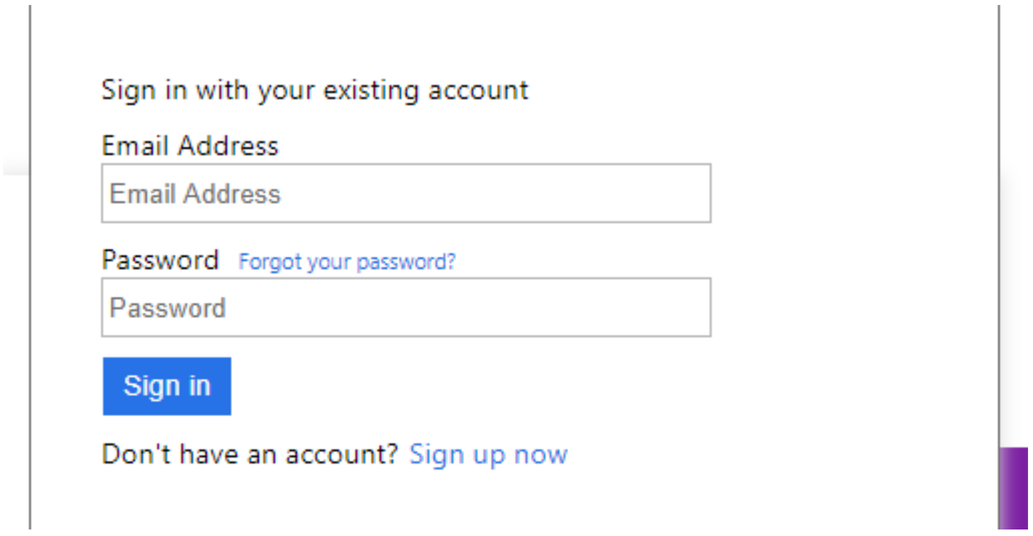


Figure 3: Login and sign up popup window.

5.2. Acquiring API keys

Once an account is created, and 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).

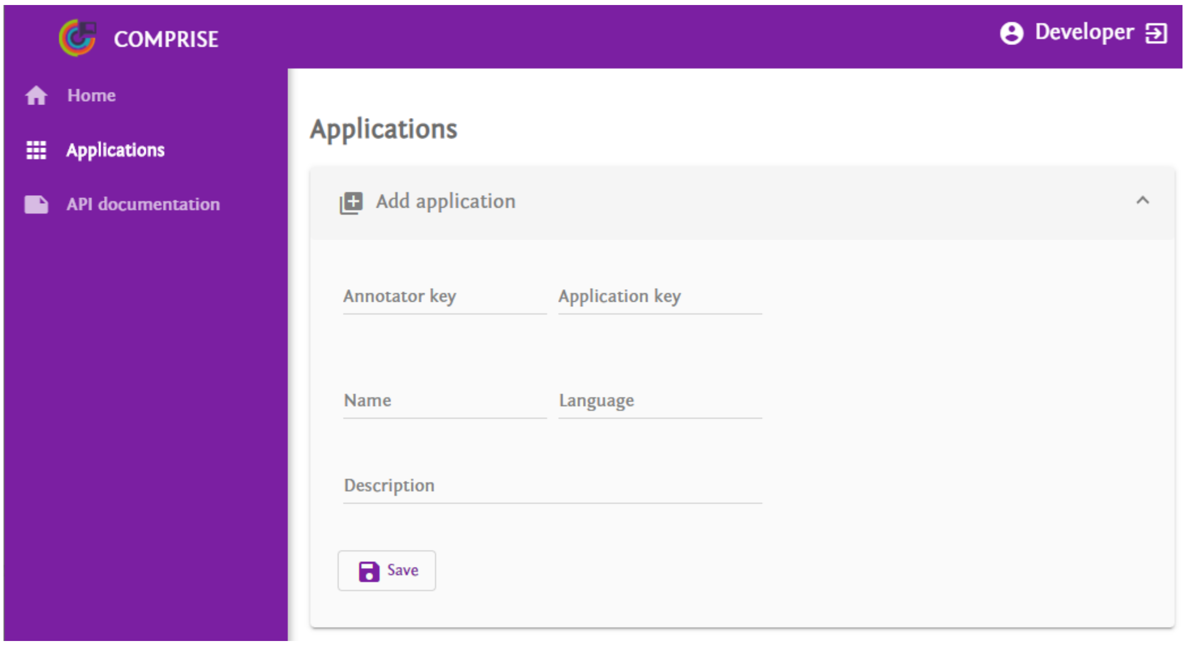
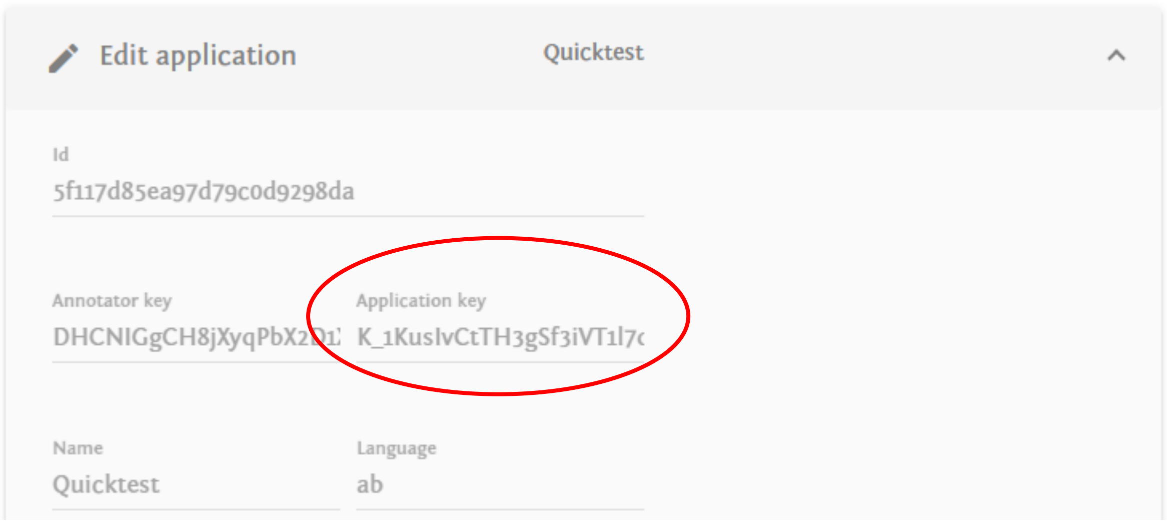


Figure 4: Registering a new App.

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).


Edit application — Quicktest

Id: 5f117d85ea97d79c0d9298da

Annotator key: DHCNIGgCH8jXyqPbX2D1

Application key: K_1KusIvCtTH3gSf3iVT1l7

Name: Quicktest

Language: ab

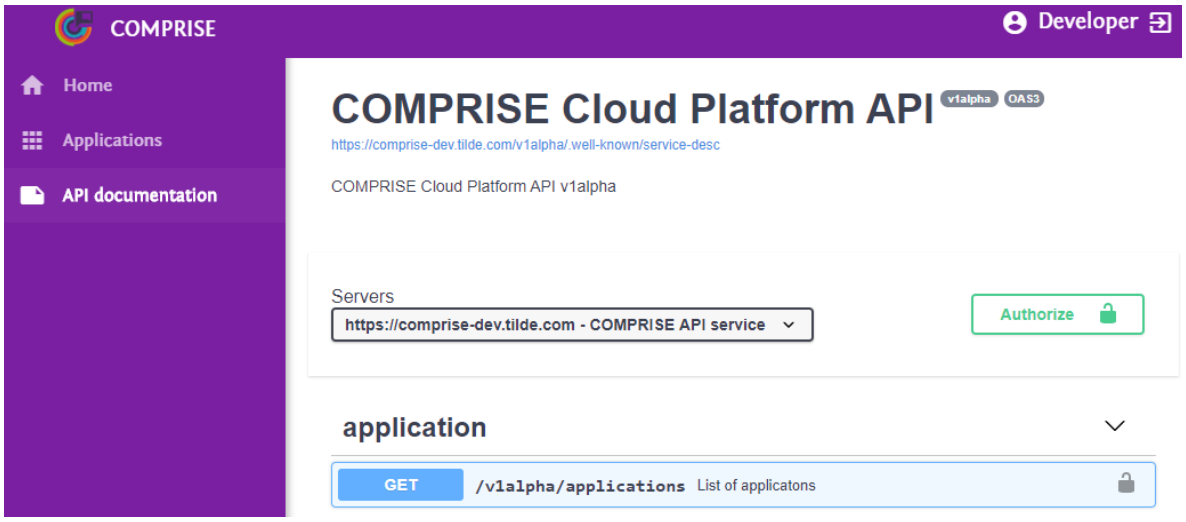
Figure 5: Automatically generated API keys.

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 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 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).



COMPRISE Cloud Platform API v1alpha OAS3

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

COMPRISE Cloud Platform API v1alpha

Servers

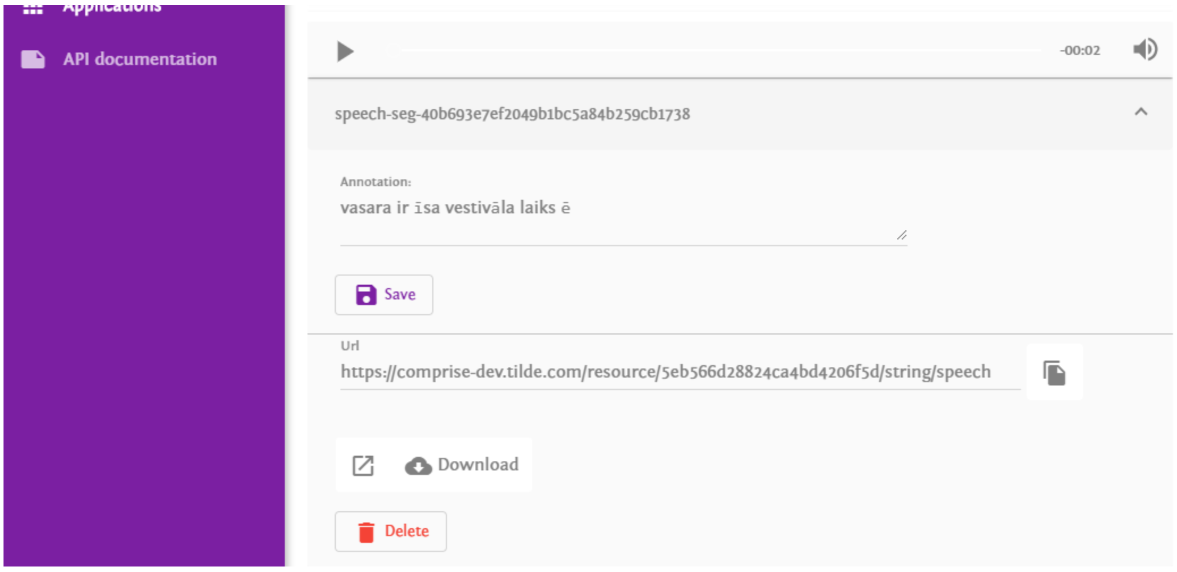
https://comprise-dev.tilde.com - COMPRISE API service

GET /v1alpha/applications List of applicatons

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.



speech-seg-40b693e7ef2049b1bc5a84b259cb1738

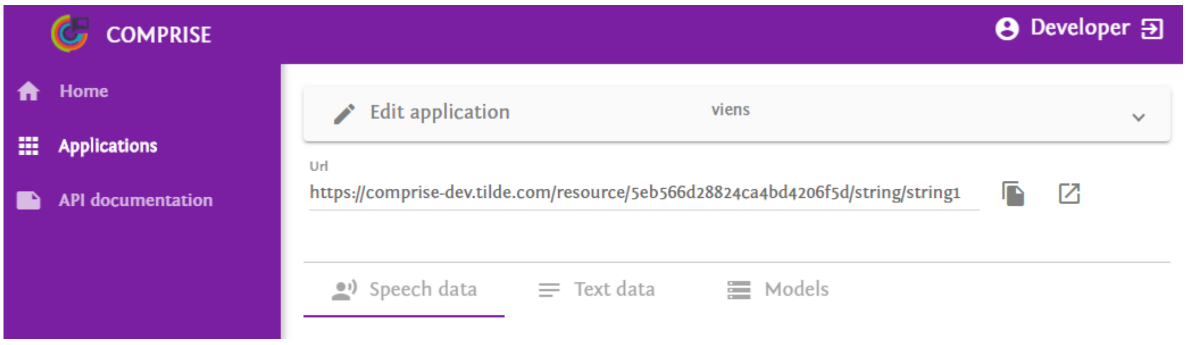
Annotation:
vasara ir īsa vestivāla laiks ē

Url
https://comprise-dev.tilde.com/resource/5eb566d28824ca4bd4206f5d/string/speech

Figure 7: Speech data tab.

In normal practice, data annotation is not performed by the Developer themself, 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).

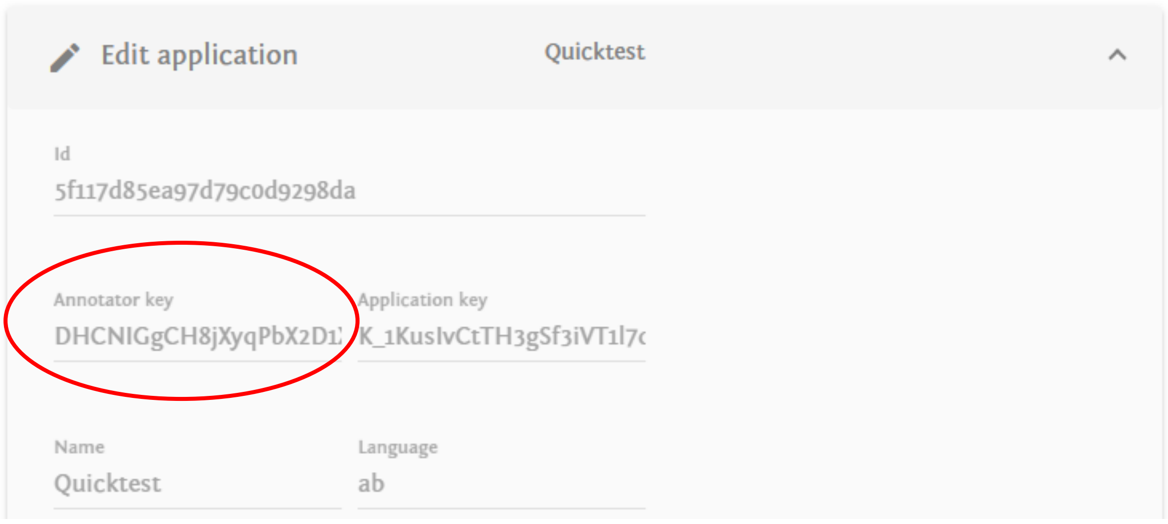


Edit application viens

Url
https://comprise-dev.tilde.com/resource/5eb566d28824ca4bd4206f5d/string/string1

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.


Edit application

Quicktest

Id
5f117d85ea97d79c0d9298da

Annotator key
DHCNIGgCH8jXyqPbX2D1

Application key
K_1KusIvCtTH3gSf3iVT1l7

Name
Quicktest

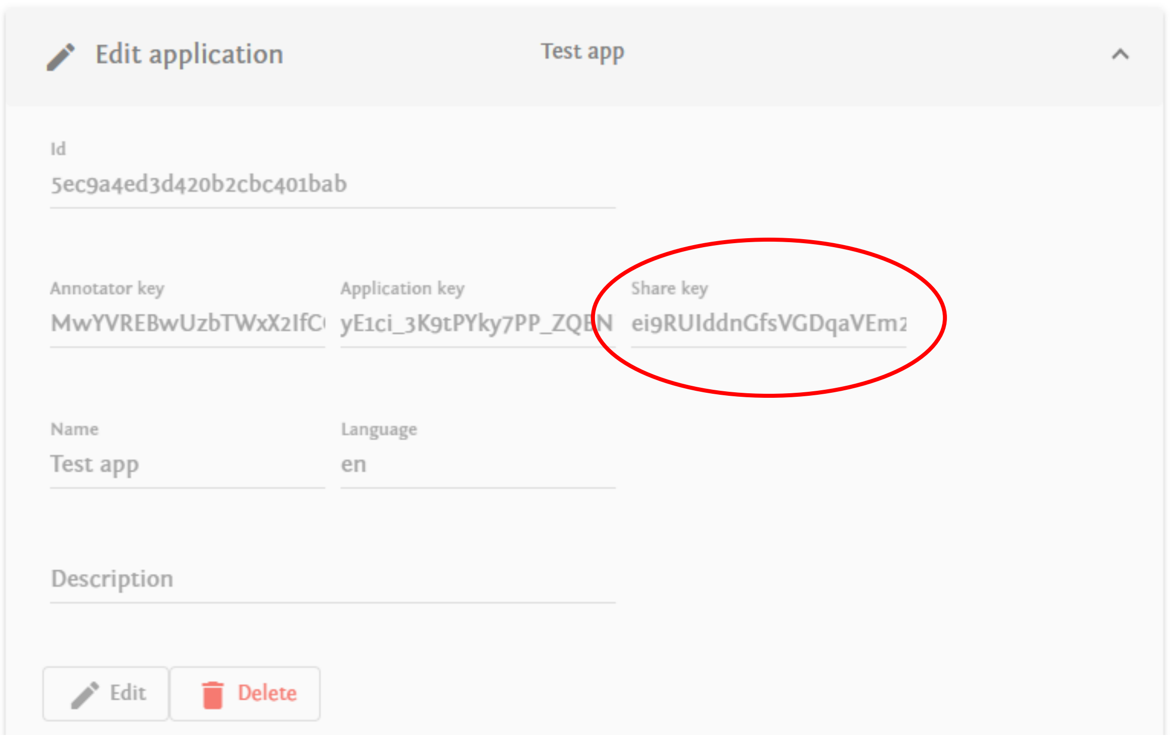
Language
ab

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).


Edit application

Test app

Id
5ec9a4ed3d420b2cbc401bab

Annotator key
MwYVREBwUzbTWxX2IfC

Application key
yE1ci_3K9tPYky7PP_ZQEN

Share key
ei9RUIddnGfsVGDqaVEm

Name
Test app

Language
en

Description

Figure 10: Setting the share key.

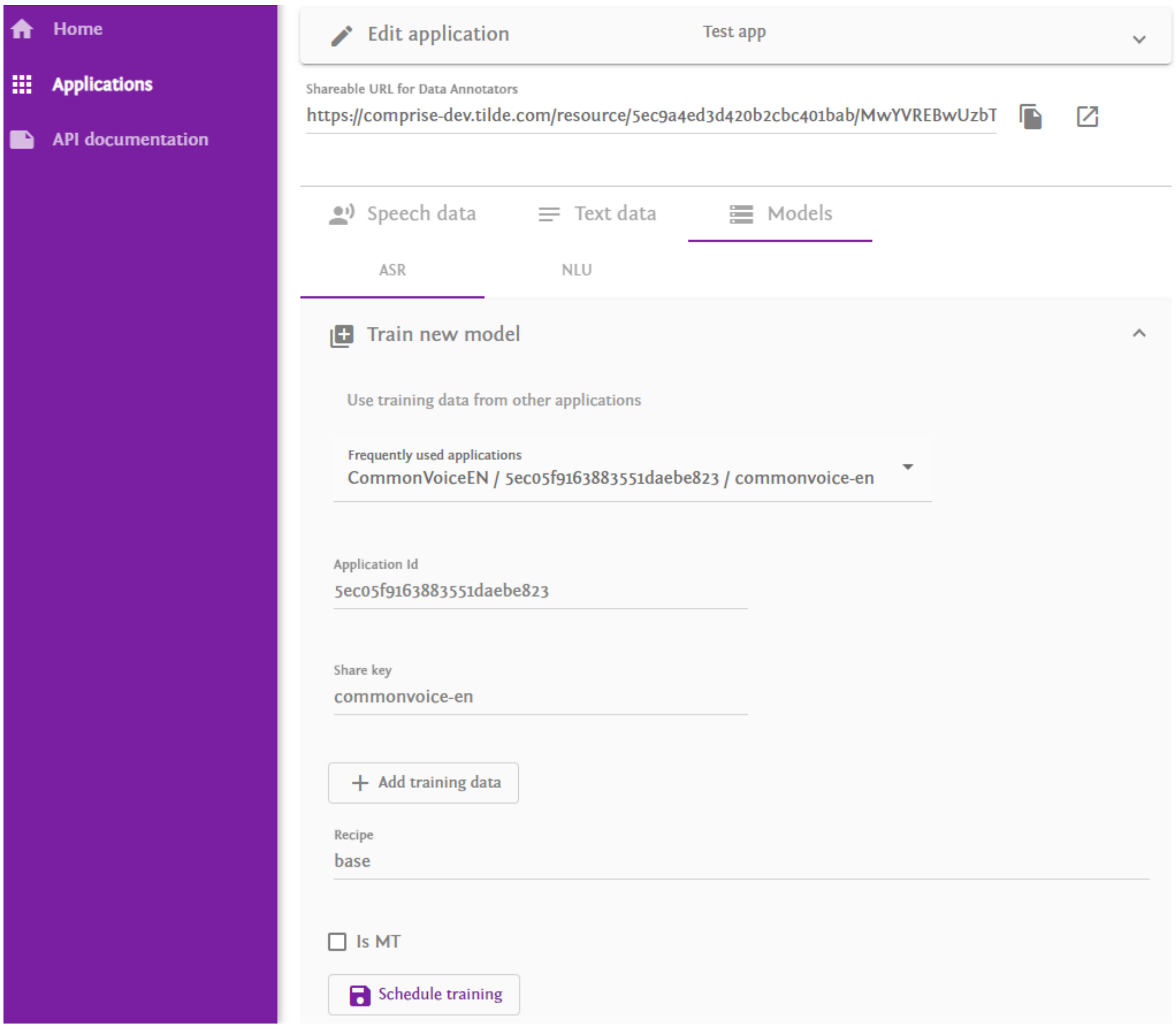
5.6. Model training

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”.

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 initial 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.



Test app

Shareable URL for Data Annotators
https://comprise-dev.tilde.com/resource/5ec9a4ed3d420b2cbc401bab/MwYVREBwUzbT

Use training data from other applications

Frequently used applications
CommonVoiceEN / 5ec05f9163883551daebe823 / commonvoice-en

Application Id
5ec05f9163883551daebe823

Share key
commonvoice-en

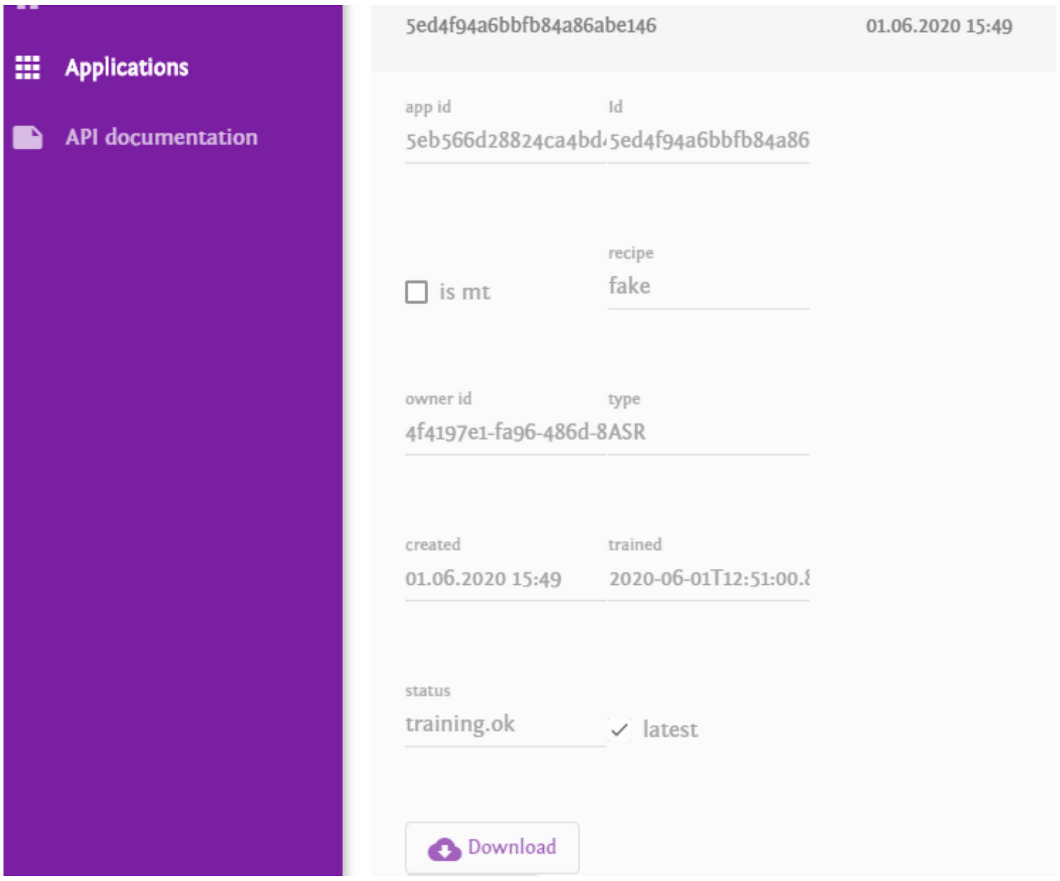
Recipe
base

☐ Is MT

Figure 11: Model training tab.

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.



5ed4f94a6bbfb84a86abe146 01.06.2020 15:49

app id: 5eb566d28824ca4bd
Id: 5ed4f94a6bbfb84a86

☐ is mt
recipe: fake

owner id: 4f4197e1-fa96-486d-8
type: ASR

created: 01.06.2020 15:49
trained: 2020-06-01T12:51:00.

status: training.ok
✓ latest

Figure 12: Detailed information on a trained model.

6. Conclusion

This deliverable presents the initial 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 Cloud Platform relies on external cloud services.

The 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.[11] The source code of the Unified training recipe is published in a separate repository on the COMPRISE GitLab[12]. Both repositories also contain deployment documentation and example Kubernetes configuration files.

[11] https://gitlab.inria.fr/comprise/comprise-cloud-based-platform
[12] https://gitlab.inria.fr/comprise/comprise-stt-training-recipe