



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 N°: 825081

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

**Deliverable N°5.3: Data collection and curation
features of the platform**

Lead partner: TILDE

Version N°: 1.0

Date: 31/08/2020



Document information	
Deliverable N° and title:	D5.3 - Data collection and curation features of the platform
Version N°:	1.0
Lead beneficiary:	TILDE
Author(s):	Askars Salimbajevs, Raivis Skadiņš (TILDE)
Reviewers:	Irina Illina (INRIA), Álvaro Moretón (ROOT)
Submission date:	31/08/2020
Due date:	31/08/2020
Type ¹ :	OTHER
Dissemination level ² :	PU

Document history			
Date	Version	Author(s)	Comments
16/07/2020	0.1	Askars Salimbajevs, Raivis Skadiņš	Initial draft version
28/07/2020	0.2	Askars Salimbajevs, Raivis Skadiņš	Revised version following the reviewers' comments
31/08/2020	1.0	Emmanuel Vincent, Zaineb Chelly	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 first version of the COMPRISE Cloud Platform. The document contains an overview of the implemented data collection and curation features, and a high-level description of the Cloud Platform architecture and implemented application programming interface (API). Instructions on how to deploy and use the Cloud Platform are provided as well.

The described version of the 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 speech-to-text (STT) and spoken language understanding (SLU) models, as well as scheduling training of new models. However, the specific training recipes or scripts are still under development. In the future, support for other types of data and models might 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. This Cloud Platform will fill a gap in the current ecosystem: existing resource repositories are good for speech resource description, dissemination, sharing, and distribution, but according to our knowledge no platform would facilitate speech data creation, labelling, and curation.

The Cloud Platform also includes a web-based user interface (UI) which allows users to sign up to the COMPRISE Cloud Platform, perform general operations and access the documentation. An initial version of this Cloud Platform demonstrator has been set up: <https://comprisedev.tilde.com>.

Table of contents

1. Introduction	5
2. Data collection and curation features	5
2.1 Overview	5
2.2 Features for Developers	6
2.3 Features for Data Annotators	6
3. Cloud Platform architecture	6
3.1. Overview	6
3.2. API service	8
3.2.1. Authentication and authorisation	8
3.2.2. Data API	9
3.3. Web UI	13
4. Deployment	13
4.1. Prerequisites	13
4.2. Configuration	13
4.2.1. API service	13
4.2.2. Web UI	14
4.3. Building Docker images	14
4.3.1. Building the API service image and the job-wrapper image	14
4.3.2. Building the Web UI image	14
4.4. Deploying on Localhost	14
4.4.1. API service	14
4.4.2. Web UI	15
4.5. Deploying on Kubernetes	15
5. How to use	16
5.1. Signing up	16
5.2. Acquiring API keys	17
5.3. Data collection	18
5.4. Data annotation	19
5.5. Model training	20
6. Conclusion	21

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 these data as a service via a web service application programming interface (API).

The current report is developed in the scope of Task T5.3. “Data collection and curation features of the platform” and consists of the following sections. Section 2 informally defines data collection and curation features of the COMPRISE Cloud Platform, and details the features available to Developers and Data Annotators. Section 3 describes the updated architecture of the COMPRISE Cloud Platform and provides a high-level description of authentication, authorisation and data management API. Section 4 provides information on the configuration and deployment of the Cloud Platform. Section 5 describes how the Cloud Platform can be used from the Developer’s point of view. Finally, Section 6 is devoted to conclusions.

2. Data collection and curation features

2.1 Overview

Developers of voice-enabled applications (e.g., a personal assistant) have among their priorities providing 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 application. 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.

Developers sign up to the Cloud Platform and acquire the API keys by registering their application into the Cloud Platform. Next, Developers embed these API keys in the application and the COMPRISE Client Library collects speech and text data from users of the application.

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

Developers use the COMPRISE Cloud Platform to manage the collected data, process the collected data (e.g., apply machine translation) and train domain-specific user-independent STT and/or SLU models.

As the collected data can be annotated, Developers shall be able to add annotations to the collected corpora themselves or give access to the collected corpora to Data Annotators

employed by the Developer's company. 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 Developers via a specific URL.

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 is then used for training domain-specific models.

2.2 Features for Developers

To summarise, the following features were developed in the Cloud Platform for Developers using COMPRISE in their applications:

- 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;
- an API for scheduling STT and SLU model training;
- an API for downloading trained models;
- a Web user interface (UI) that simplifies access to all these features.

2.3 Features for Data Annotators

Data Annotators do not have any accounts on the Cloud Platform. They access the Cloud Platform by using special URLs provided by the Developers. Each such URL gives access to data collected by each particular Application. 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 these features.

3. 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 the Cloud Platform with data collection and curation functionality through an API.

- 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 applications, corpus annotation, triggering model training, etc.
- A queue service based on RabbitMQ³ and KEDA⁴. This service manages the training job queue and is responsible for starting training Docker containers when training is requested.

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 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:

- downloading training data and annotations from the cloud storage service;
- starting a `/job.sh` script which is to be overridden with actual training;
- creating data directories in Kaldi format if necessary;
- collecting trained models and uploading them to the cloud storage service.

For future maintainability, the implementation does not reinvent everything from scratch. Instead, it 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, we use 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 Cloud Platform is designed so that it does not rely on specific Azure Blob features. The Azure interface code is separated on its own submodule so that it can be easily replaced.

³ <https://www.rabbitmq.com/>

⁴ <https://github.com/kedacore/keda>

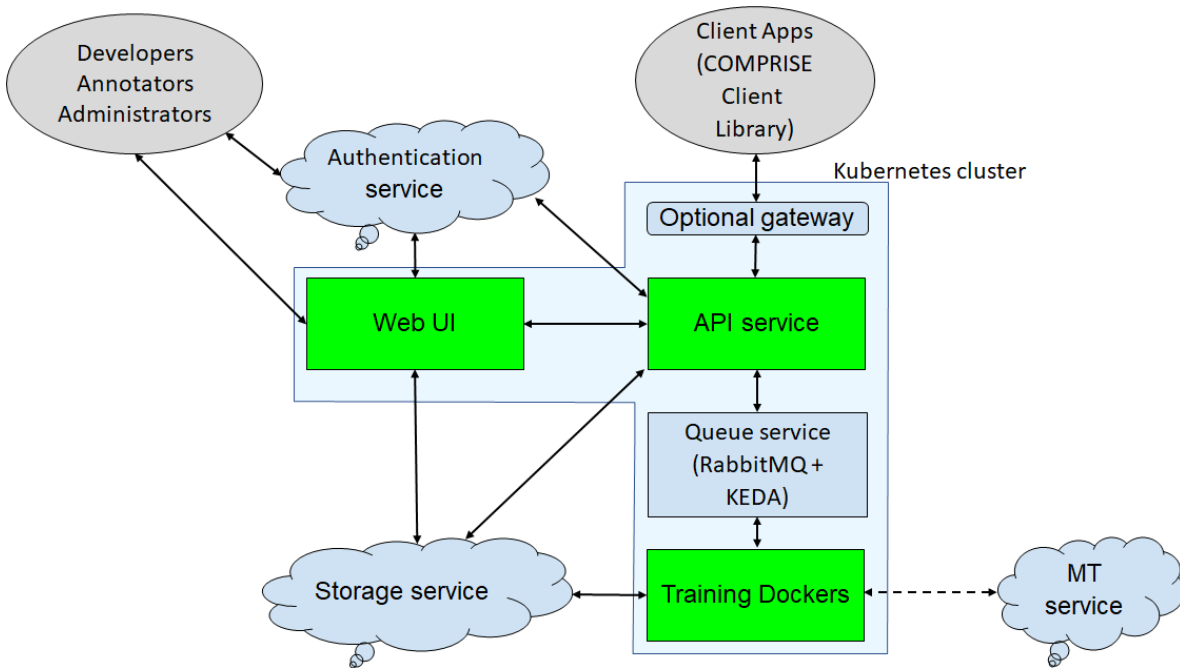


Figure 1: Cloud Platform architecture.

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 and 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 Cloud Platform is available on the COMPRISE GitLab⁵ 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.

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:

⁵ <https://gitlab.inria.fr/comprise/comprise-cloud-based-platform>

- 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 Cloud Platform owner can set up an OpenID Connect compatible authentication service either himself/herself or by using external service providers. Developers and Administrators must sign up through the Cloud Platform login page. Depending on the service provider and configuration, an approval from the Cloud Platform owner might be required. The current version of the 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 Cloud Platform.

Administrators have access to all API methods for all resources on the Cloud Platform, while Developers have access to all API methods only for resources (registered Apps, speech and text) that they have created. Resources created by other accounts are not visible nor accessible.

The current Cloud Platform version has two alternative authorisation implementations. The first implementation uses a text file which stores the mapping between user IDs and roles (Administrator, Developer, none). The Cloud Platform owner grants access to the 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 Cloud Platform owner grants access to the 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 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, like: <http://<host>/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://<host>/v1/<endpoint>> or <http://<host>/v2/<endpoint>>.

Application registration and management are provided by [/v1alpha/applications endpoint](#) (see Table 1).

Table 1: 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",
  "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"
}
```

Speech and text data upload is 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 speech or text data upload 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](#). As these endpoints are symmetrical, here we describe only the former (see Table 2).

Table 2: 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:

```
{
  "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 a JSON:

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

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

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

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. Currently, the API allows training of models only on data collected by a particular App. In the future, additional API methods will be implemented to enable data sharing between different Apps.

As STT and SLU training endpoints are symmetrical, here we only describe the former (see Table 3).

Table 3: 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 JSON contain the following 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 should be posted to the STT or SLU endpoint:

```
{
  "recipe": "string, name of the training recipe"
}
```

Available recipe names are not predefined and depend on the particular configuration of the Cloud Platform. The API service creates a training request in the job queue corresponding to the given recipe. Then, if such a recipe handler is configured, KEDA will start a Kubernetes job which will perform the training.

3.3. 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 Cloud Platform features, like 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 COMPRISE Cloud Platform depends on several 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;
- a Kubernetes cluster (or at least Docker).

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

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

4.2. Configuration

4.2.1. API service

The API service uses a file named `config.yaml` for reading and storing configuration. 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. Building the API service image and the job-wrapper image

You can build the API service Docker image by running the following commands:

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

4.3.2. Building the Web UI image

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 docker build -t comprisedev.azurecr.io/comprise-web:latest .
```

4.4. Deploying on Localhost

The COMPRISE Cloud Platform API service and the Web UI can be run on a local computer.

4.4.1. API service

To deploy the COMPRISE Cloud Platform API service on a local computer (localhost):

- Prepare the `config.yaml` configuration file in the current directory (use `config.yaml.localhost` as a template).
- Prepare `user_roles.yaml` (see `user_roles.yaml.example`).
- Run MongoDB from the Docker image as follows:

```
docker run -d -p 27017:27017 \
  -e MONGO_INITDB_ROOT_USERNAME=mongo \
  -e MONGO_INITDB_ROOT_PASSWORD=mongo \
  mongo
```

- Run RabbitMQ from the Docker image as follows:

```
docker run -d -p 5672:5672 \
  -e RABBITMQ_DEFAULT_USER=guest \
```

```
-e RABBITMQ_DEFAULT_PASS=guest \  
rabbitmq:3
```

- Run the the API service container using `config.yaml` from the current working directory:

```
docker run \  
-v "$(pwd)":/etc/config \  
-p 5000:80 \  
comprisedev.azurecr.io/comprise-api-service:latest \  
--conf=/etc/config/config.yaml
```

- You can now use the API service running on `localhost:5000`

4.4.2. Web UI

To deploy the COMPRISE Cloud Platform Web UI on localhost:

- Configure the Web UI to use the API from localhost.
- Build the Docker image as described above.
- Run the Web UI Docker container:

```
docker run \  
-p 8080:80 \  
comprisedev.azurecr.io/comprise-web:latest
```

- Open <http://localhost:8080/> in a browser to use the Web UI.

4.5. Deploying on Kubernetes

In practice, the Cloud Platform owner will often want to run it 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.

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 Cloud Platform can work behind https endpoints;
- an ingress controller (nginx), that will provide routing to 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 configuration that sets up an API service and a Web UI, a fake STT training job and enables https access to the cluster.

To start this example configuration, one should first edit the settings in `kubernetes/overlays/example/config.yaml` and provide credentials to Azure Blob storage and services. It is also possible to configure the service to use a file instead of Azure AD.

Then the COMPRISE Cloud Platform can be deployed as follows:

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

5. How to use

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

Consider the Developer of a voice-enabled mobile application 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 COMPRISE.

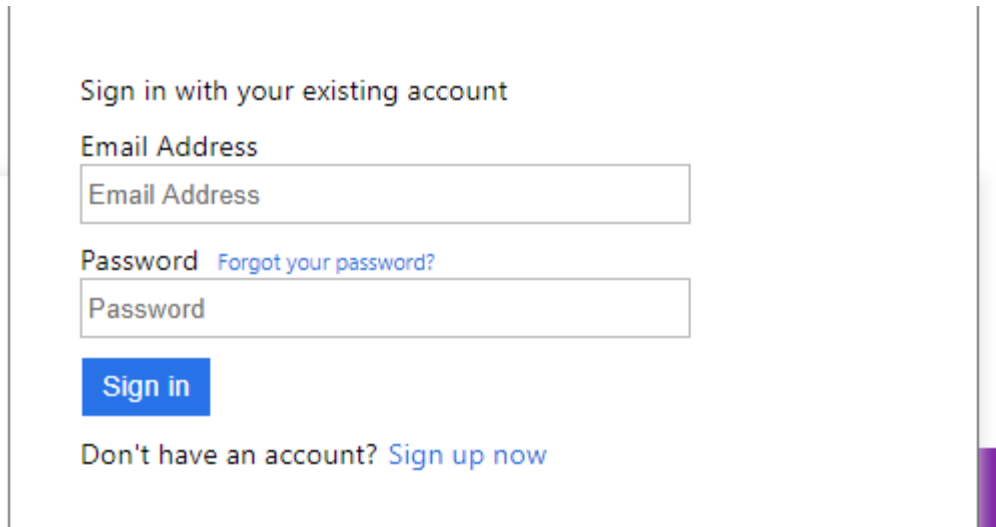
5.1. Signing up

The Developer goes to <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 can create a new account or login into an existing one.



Sign in with your existing account

Email Address

Password [Forgot your password?](#)

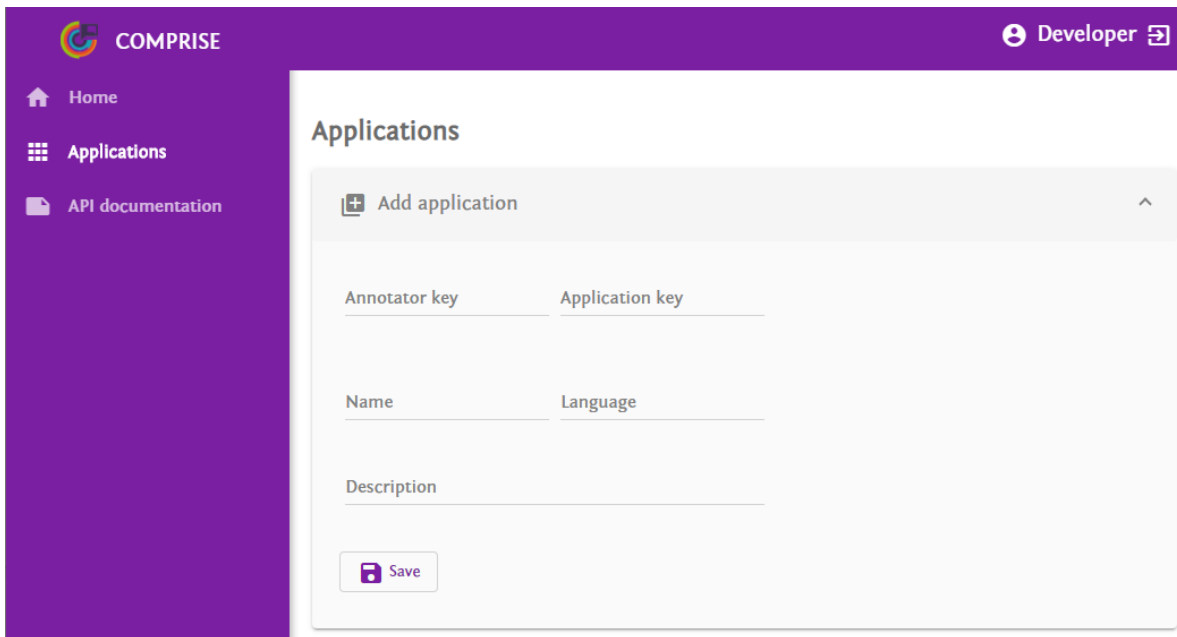
[Sign in](#)

Don't have an account? [Sign up now](#)

Figure 3: Login and sign up popup window.

5.2. Acquiring API keys

After creating an account and logging into the Web UI of the Cloud Platform, the Developer can get API keys for his/her mobile App. Firstly, the Developer needs to register the mobile App on the Cloud Platform. This can be done by clicking on “Applications” and then on “Add application” (see Figure 4).



COMPRISE Developer

- Home
- Applications**
- API documentation

Applications

[+ Add application](#)

Annotator key Application key

Name Language

Description

[Save](#)

Figure 4: Registering a new App.

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

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

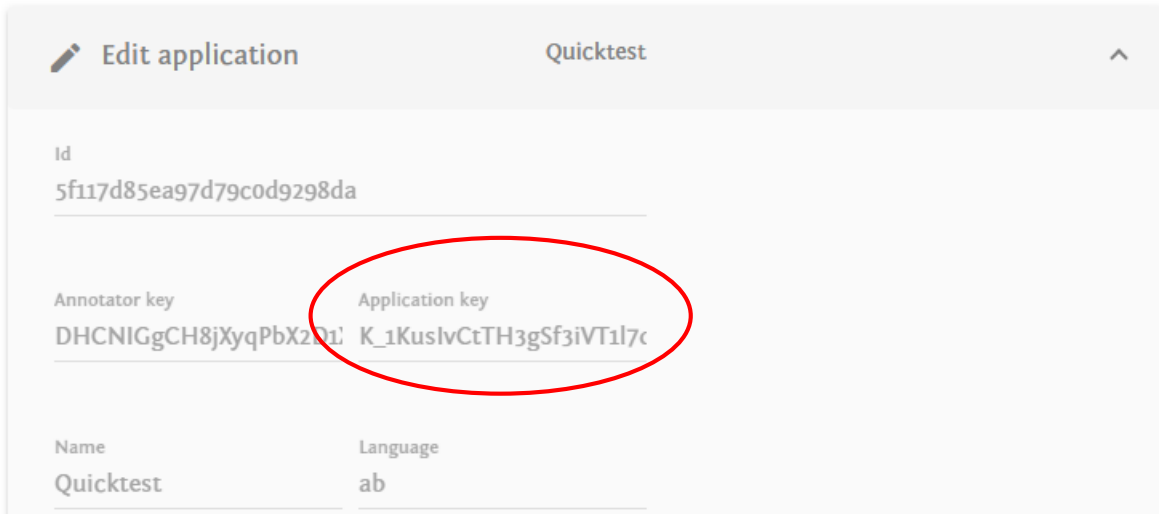


Figure 5: Automatically generated API keys.

5.3. Data collection

Next, the Developer embeds the generated API key into his/her 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 his/her App, then he 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).

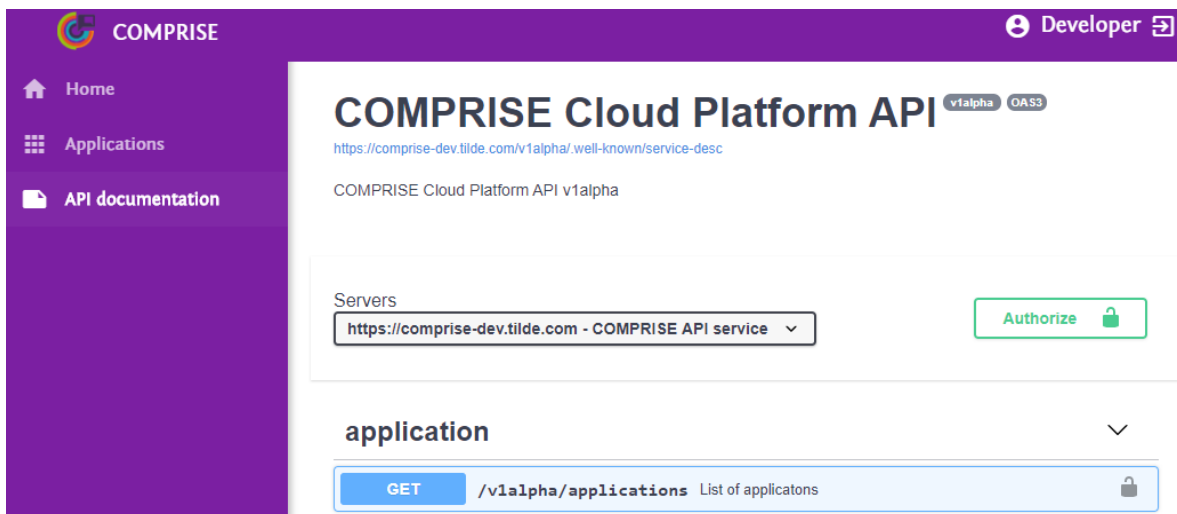


Figure 6: Interactive API documentation.

5.4. Data annotation

After data collection has started, the Developer can use the Web UI and the Cloud Platform to curate the data collected. 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). On this tab the Developer can listen to the collected audio samples, add a transcription and delete samples that contain only noise.

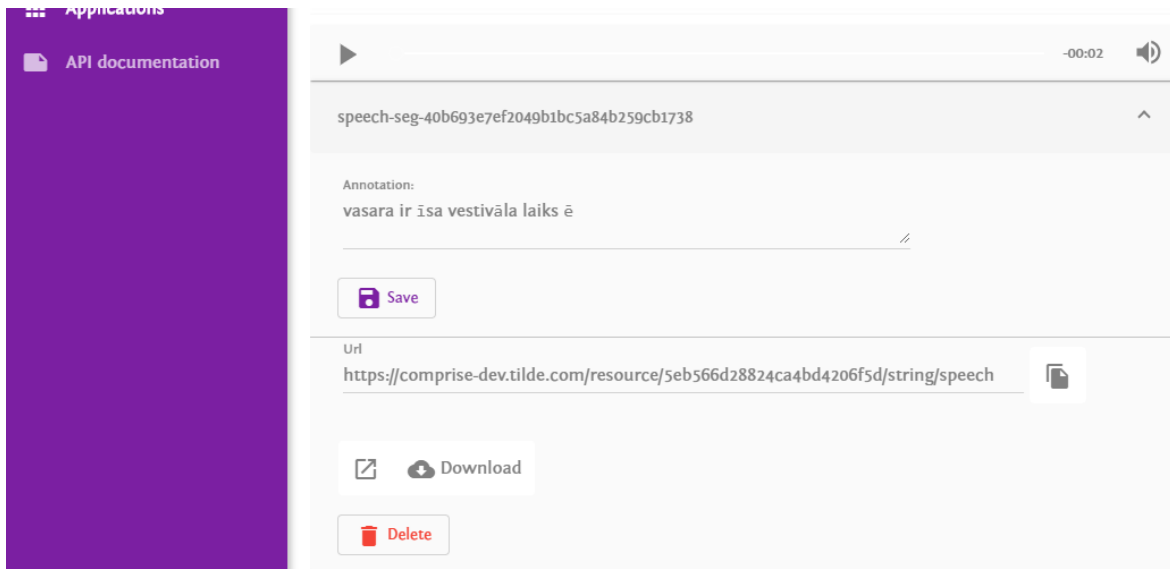


Figure 7: Speech data tab.

Usually, data annotation is not performed by the Developer himself/herself, but by Data Annotators employed by the Developer’s company.

The Developer grants access to the collected speech and text data by sharing a special URL with Data Annotators. This URL is displayed on the App details page (see Figure 8).

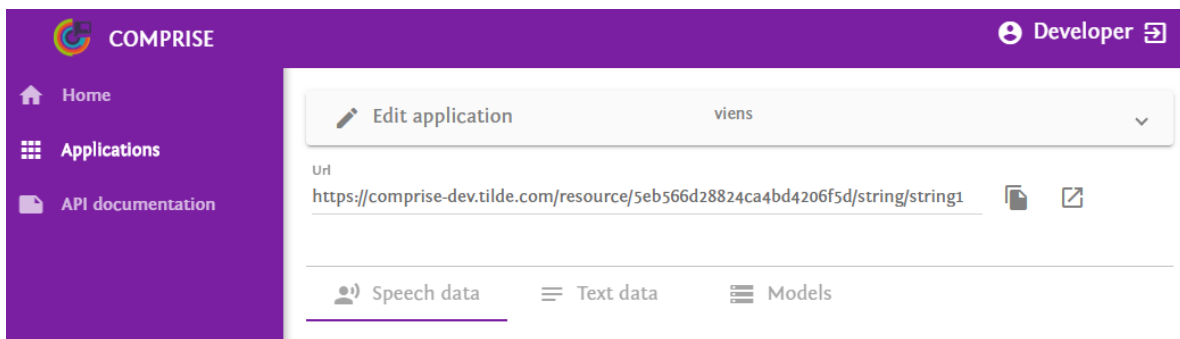


Figure 8: Shareable URL for Data Annotators.

If needed, this URL can be invalidated by changing the AnnotatorKey of the application (see Figure 9). Trying to set an empty AnnotatorKey will trigger automatic key generation.

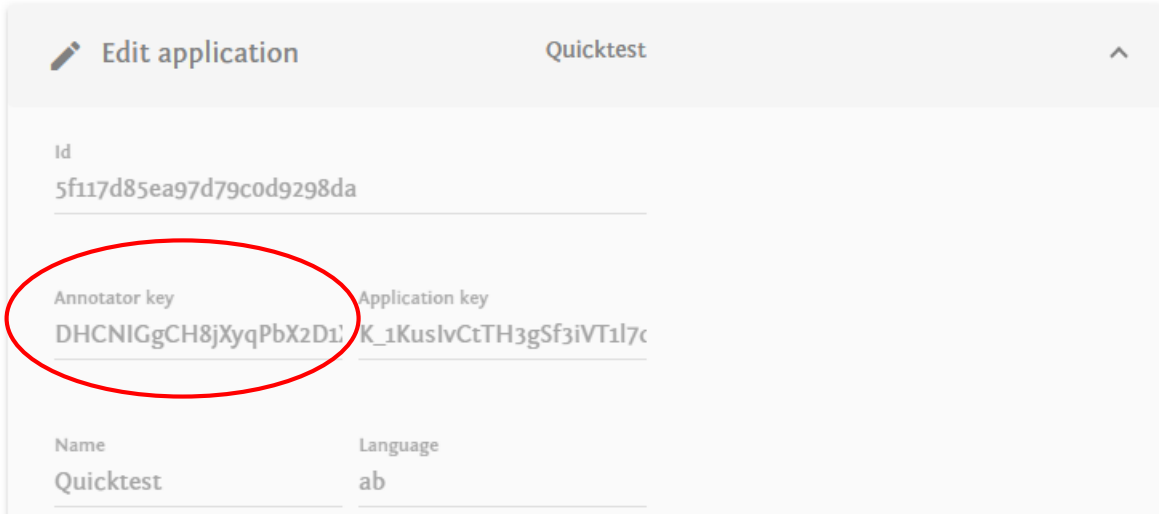


Figure 9: Changing annotator key to invalidate shared URLs

5.5. Model training

After enough data have been collected and annotated, the Developer can schedule the training of a new, improved model. For example, for training a new STT model, the Developer must select a particular App from the list, click on the “Models” tab and then on “ASR” and “Train new model”.

After choosing the training recipe, the Developer clicks on “Schedule training” and a training job is added to the queue (see Figure 10).

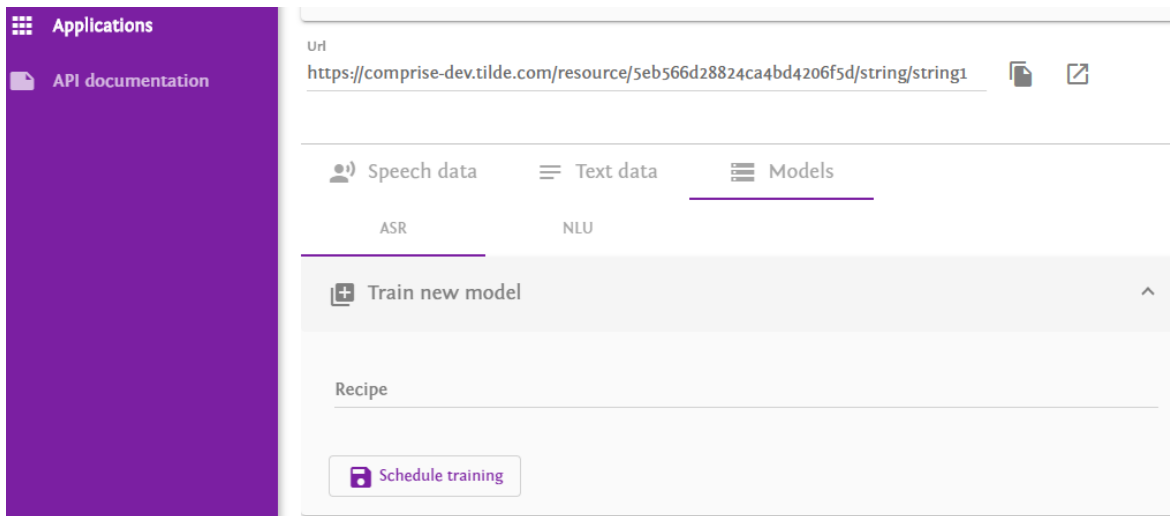


Figure 10: Model training tab.

Model training progress can be monitored on the “Models” tab (see Figure 11). When training is completed, the model can be downloaded directly by the mobile App using COMPRISE Client Library functions or directly by calling the Cloud Platform API. It is also possible to download a trained model using the Web UI for manual integration.

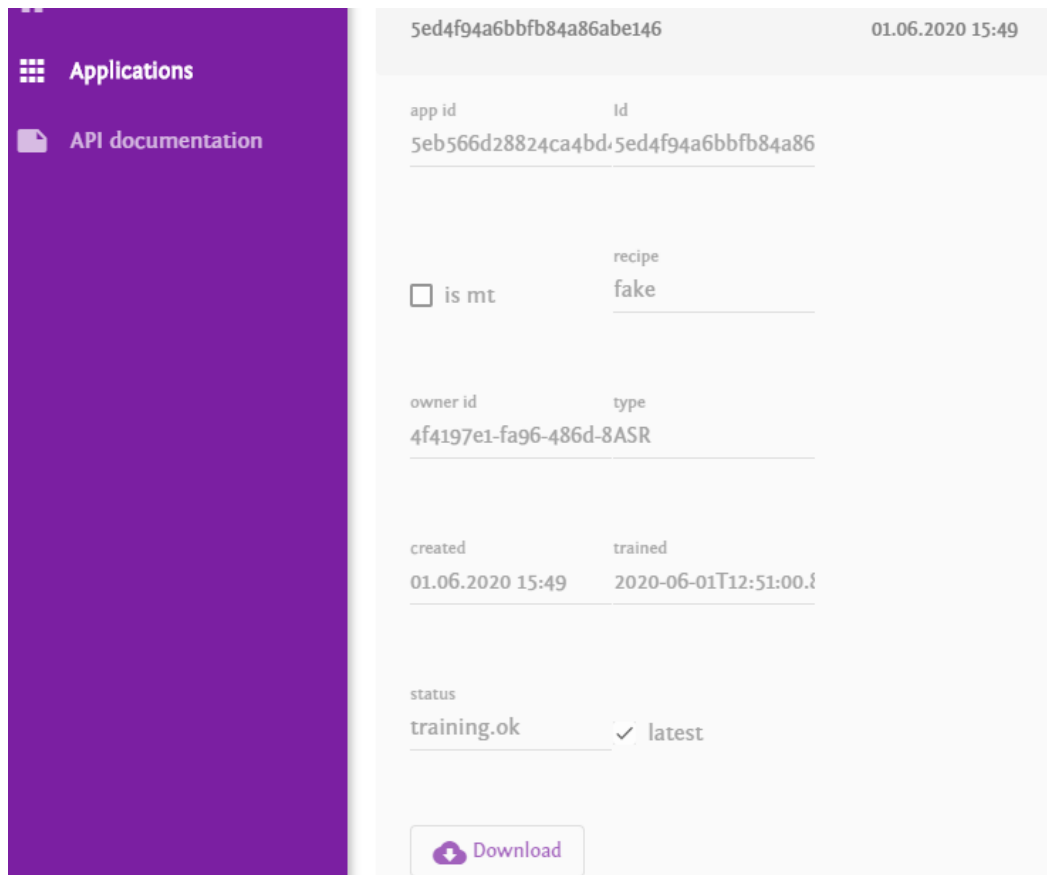


Figure 11: Detailed information on a trained model.

6. Conclusion

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

This first version provides data collection and curation features, model storage, model training scheduling and a web UI, and it is deployed online: <https://comprisedev.tilde.com>.

The source code of the COMPRISE Cloud Platform is freely available on the COMPRISE GitLab⁶. The repository also contains deployment documentation and example Kubernetes configuration files.

In the next steps, we plan to extend the Cloud Platform API with the ability to share data between mobile apps from different Developers and populate the Cloud Platform with real speech and text data, as well as implement real STT and SLU model training recipes.

⁶ <https://gitlab.inria.fr/comprise/comprise-cloud-based-platform>