



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°4: Cost-effective multilingual
voice interaction**

**Deliverable N°4.3: Initial COMPRISE SDK
prototype**

Lead partner: ASCO

Version N°: 1.0

Date: 31/08/2020



Document information	
Deliverable N° and title:	D4.3 – Initial COMPRISE SDK prototype
Version N°:	1.0
Lead beneficiary:	ASCO
Author(s):	Gerrit Klasen (ASCO)
Reviewers:	Denis Jouvét (INRIA), Youssef Ridene (NETF)
Submission date:	31/08/2020
Due date:	31/08/2020
Type ¹ :	DEM
Dissemination level ² :	PU

Document history			
Date	Version	Author(s)	Comments
10/07/2020	0.1	Gerrit Klasen	First draft of the deliverable
20/07/2020	0.2	Gerrit Klasen	Revised version following the reviewer comments
17/08/2020	0.3	Gerrit Klasen	Revised version following the project manager 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 (D4.3) is the third deliverable of WP4 which is about cost-effective multilingual voice interaction. It describes the initial version of the COMPRISE software development kit (SDK) prototype, which is aligned to the goals of Task T4.2.

The COMPRISE SDK integrates and interfaces multiple algorithms, application programming interfaces (APIs) and tools developed within COMPRISE. It facilitates the development of multilingual, voice-enabled applications by providing developers with a language abstraction for voice interaction and with interfaces to easily access the desired functionalities. Besides, the SDK allows developers to compile the output across multiple platforms and to create executable applications both for Android and iOS.

This document explains the current technical status of the SDK. We also discuss the necessary adjustments compared to Deliverable D4.1, where the initial architecture was defined. These include the introduction of an additional component, the COMPRISE Personal Server, as well as the corresponding changes in the architecture. A slight modification in the terminology is introduced to ensure the focus of the COMPRISE SDK.

The COMPRISE SDK consists of three components: the COMPRISE Personal Server, the COMPRISE App Wizard, and the COMPRISE Client Library. Instructions are given for application developers on how to install and use them and which degree of functionality they can include in their own applications.

Table of contents

1. Introduction	5
1.1 COMPRISE SDK as initially defined	5
1.2 COMPRISE SDK adjustments	5
1.3 Contents	6
2. COMPRISE SDK prototype	6
2.1 COMPRISE Personal Server	6
2.2 COMPRISE App Wizard	10
2.3 COMPRISE Client Library	13
2.3.1 Operating Branch	14
2.3.2 Training Branch	22
2.3.3 Cloud Platform API	25
3. Conclusion	26

1. Introduction

1.1 *COMPRISE SDK as initially defined*

Deliverable D4.1 “SDK Software Architecture” (submitted to the European Commission on 29/11/2019) defined the purpose of the SDK, its initial architecture, and an implementation plan. We briefly summarise these items below. For more details and terminology, reading Deliverable D4.1 prior to the current document is strongly recommended.

The COMPRISE SDK was initially planned to consist of two components:

- the **COMPRISE SDK Developer UI**, a user interface which helps developers attach and configure all COMPRISE functionalities in their App;
- the **COMPRISE SDK Client Library**, which implements all the voice interaction functionalities. The library provides a set of methods and functionalities to be accessed by developers within their preferred integrated development environment (IDE) during the development phase. After implementation, all of the value of this library shall run on client devices during runtime.

Deliverable D4.1 explained why the COMPRISE SDK is needed in addition to other toolkits for the development of voice-enabled applications currently available on the market. Specifically, the COMPRISE SDK provides additional benefits in terms of privacy, cost-effectiveness, and inclusiveness.

Deliverable D4.1 also gave an overview of the architecture of the SDK in terms of the architecture of all subcomponents, the workflow between the subcomponents, and how it fits into the whole COMPRISE software architecture and environment. The SDK is used by developers to design and deploy Apps, and it interfaces with the COMPRISE Cloud Platform to train domain-specific models. Further insights were given about the usage of the COMPRISE Cloud Platform API and the COMPRISE Client Library components running within the App. The deliverable differentiates the components which belong to the speech and language processing toolchain (Operating Branch) from those which aim to collect large-scale in-domain speech and language data for multiple languages (Training Branch).

The implementation plan provided a timeline for the integration and the documentation of the results of WP2, WP3, T4.1 and WP5, with the goal of releasing an initial prototype at M21 and a final one at M30. After the submission of Deliverable D4.1, we followed this implementation plan in order to translate all the concepts mentioned in the plan into a concrete technical implementation.

1.2 *COMPRISE SDK adjustments*

The first version of the COMPRISE SDK released together with this document involves two adjustments compared to Deliverable D4.1, which turned out to be necessary in the course of development.

Personal Server

Feedback from various stakeholders about the initial COMPRISE SDK architecture has raised the need for a third SDK component: the **COMPRISE Personal Server**. Section 2.1 will explain the need for this component in more detail.

Change of Terminology

Feedback has also shown that the COMPRISE SDK Developer UI was often misunderstood as an IDE, which could be used to develop Apps without relying on any additional IDEs and/or frameworks. To avoid this misunderstanding, we renamed the COMPRISE SDK Developer UI into **COMPRISE App Wizard**. This new name underlines the purpose of the tool: rather than a classical Developer UI which is used to write source code, the COMPRISE App Wizard helps developers attach COMPRISE Client Library functionality to their App and configure it.

Furthermore, we identified the need to clarify the usage of the term “SDK”. Classically, an SDK is a collection of helpful methods, functionalities, libraries and tools, which together ease developers’ work. The names of the underlying libraries typically do not include “SDK”. Based on this fact, we also renamed the COMPRISE SDK Client Library into **COMPRISE Client Library**.

To sum up, the **COMPRISE SDK** is an SDK in the classical sense, that is a collection of three components: the COMPRISE App Wizard that helps setup a future App with COMPRISE functionality, the COMPRISE Client Library that implements the functionality itself, and the COMPRISE Personal Server.

1.3 Contents

The rest of this document is structured as follows. Section 2.1 justifies the need for the COMPRISE Personal Server and explains how users can install it. Section 2.2 introduces the COMPRISE App Wizard and describes the workflow for application developers. Section 2.3 details the functionalities currently implemented in the COMPRISE Client Library. These methods are documented for developers to use within the IDE of their choice. Section 3 summarises future steps towards the final version of the COMPRISE SDK due at M30.

2. COMPRISE SDK prototype

As stated above, the COMPRISE SDK now consists of three components, namely:

- 1) the COMPRISE Personal Server
- 2) the COMPRISE App Wizard
- 3) the COMPRISE Client Library

To attach COMPRISE functionality to individual projects, developers shall use these components in the above order.

2.1 COMPRISE Personal Server

The COMPRISE Personal Server answers two concerns which arose during the development of the COMPRISE Client Library. First, some Client Library components currently rely on Python code. The integration of these components on mobile devices relying on Java, Swift, or Angular requires porting this code into a supported language and/or compiling it using Web Assembly. The amount of work needed is large and hard to estimate precisely. Second, as mentioned in Deliverable D4.1, some components need high computational power and/or storage capacity, which cannot be achieved by older or low-end mobile devices.

It was therefore decided to offer users and developers the possibility to run these (and possibly all) Client Library components on a Personal Server, which facilitates their integration and provides greater computational power and storage capacity. Once installed on a user’s personal computer at home, the Personal Server communicates with his/her Android/iOS front-

end device via a secure (SSL) channel. The user device sends the input data for a given component to the Personal Server, processing takes place on the Personal Server, and the output is sent back to the user device. Thanks to SSL, this solution protects privacy almost³ as well as if the components were running on the user device itself. Also, the Personal Server is expected to offer similar user experience in terms of latency and usability as a public cloud. For instance, it can handle large vocabulary applications and the user may communicate with it from virtually anywhere (e.g., while in public transport).

The Personal Server is based on the Docker⁴ framework, which helps to build containerised services and applications. The use of Docker makes it possible to serve all of the functionalities needed, package them within a container, and ship it multiple times within images. This allows a large number of users to use the same type of server for their individual needs.

The overall COMPRISE architecture including the COMPRISE Personal Server is depicted in Figure 1.

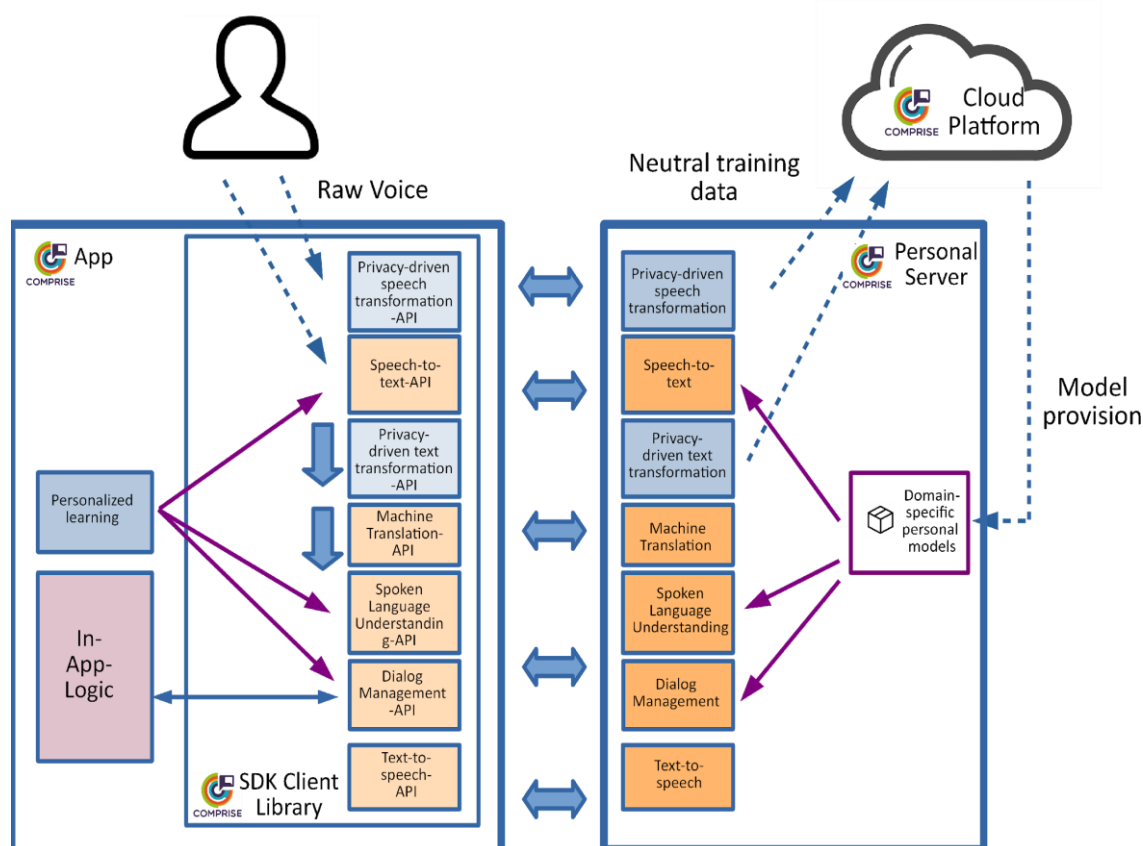


Figure 1: New COMPRISE architecture with Personal Server.

Once all Client Library components have been integrated on the Personal Server, we will use the remaining time (if any) to integrate as many components as possible in Android/iOS, so as to offer developers the alternative possibility to run them on the user device.

In the following, we provide instructions for a developer to install a local instance of the Personal Server on his/her desktop for the purpose of testing the App being developed. In the

³ The risk that an attacker manages to break encryption and listen to the data sent from the user's device to the Personal Server (a so-called man-in-the-middle attack) is very low and this requires huge effort. The risk that an attacker manages to access user data stored in a public cloud is much higher.

⁴ <https://www.docker.com/>

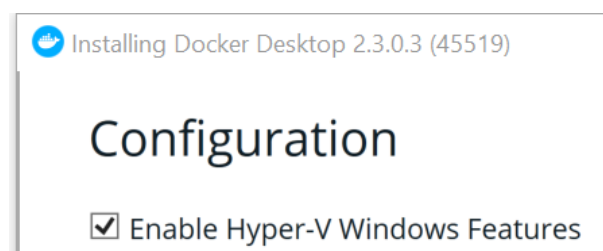
future, any user shall also be able to easily install an instance of the Personal Server on his/her personal computer in order to run the functionalities behind multiple Apps. The installation steps for users are out of the scope of this deliverable, and will be detailed in Deliverable D4.5 which is planned for M30.

Step 1 – Install Docker Engine on your desktop

To run the Personal Server, you first need to install Docker Engine on your desktop. To do so, follow the instructions for MacOS⁵ or Windows⁶ to download and install a Docker Desktop environment or the instructions for Linux⁷ to download and install a Docker Server.

The instructions below describe the subsequent steps for Docker Desktop. Similar steps can be followed for Docker Server.

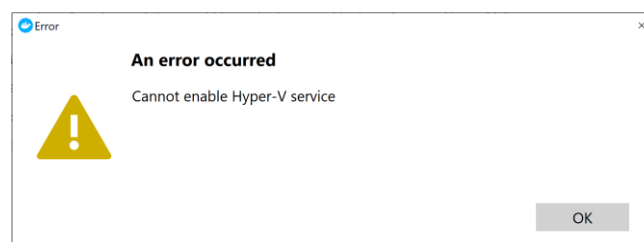
During installation, you will be asked whether you want to enable HyperV-Features. Agree on that (see Windows example below).



Step 2 – Start Docker Engine

Start Docker Engine. Sign-in is potentially needed. You may use "comprise-dev@web.de" and "comprise_h2020" as credentials.

If you get the Alert below, access the BIOS to activate HyperV-Features. On Windows, if the problem persists, it is often helpful to search for "Activate / Deactivate Windows Features" and turn "HyperV" off and on again, both after a re-start of the system.



Step 3 – Install the COMPRISE Personal Server

Once Docker Engine is running, open a shell/console (with admin rights, if needed) and type:

```
docker pull compriseh2020/personalized_server
```

⁵ <https://docs.docker.com/docker-for-mac/install/>

⁶ <https://docs.docker.com/docker-for-windows/install/>

⁷ <https://docs.docker.com/engine/install/>

This will cause Docker to download a container with the COMPRISE Personal Server on your desktop. The Docker container is hosted at DockerHub⁸ and will soon be pushed to the COMPRISE GitLab. You will see multiple layers of the Docker image being downloaded and extracted in parallel.

```

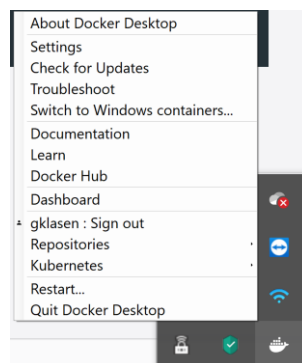
C:\Users\Gerrit>docker run -dp 8443:8443 -dp 4040:4040 compriseh2020/personalized_server
Microsoft Windows [Version 10.0.18362.900]
(c) 2019 Microsoft Corporation. Alle Rechte vorbehalten.

C:\Users\Gerrit>docker run -dp 8443:8443 -dp 4040:4040 compriseh2020/personalized_server
Unable to find image 'compriseh2020/personalized_server:latest' locally
latest: Pulling from compriseh2020/personalized_server
5bed26d33875: Downloading [====>] 2.489MB/26.69MB
f11b29a9c730: Download complete
930bda195c84: Download complete
78bf9a5ad49e: Download complete
d4cbd40905a0: Downloading [>] 2.145MB/363.3MB
bd3c052989f9: Downloading [=====] 752.1kB/4.103MB
b008b9899ddf: Waiting
008ca0c6eff5: Waiting
4d66f82c6feb: Waiting
3ec4f9e9f28c: Waiting
8d4fe97689c0: Waiting
90a3ab955b05: Waiting
4bf4e2c43fc0: Waiting
c96248a2e347: Waiting
286de4981561: Waiting
7db72aa3ae4a: Waiting
f61dac3a664b: Waiting
714eaf6fd094: Waiting
6951216a95de: Waiting
dd312201df9b: Waiting
a7419e4a1ed9: Waiting
931a6bc9d29a: Waiting
4f6412ec01ae: Waiting
6041bc4a397a: Waiting
007027a3085a: Waiting
c4669f1d761b: Waiting
eba1c4d2e20d: Waiting
0908773680d7: Waiting
8576d7a40e3a: Waiting
338d5b22a207: Waiting
ff9e7f13e9ec: Waiting
a532fc1893a3: Waiting
6ef620b74f76: Waiting
  
```

Repeat this regularly to check on updates. After all layers have been downloaded, start running the container:

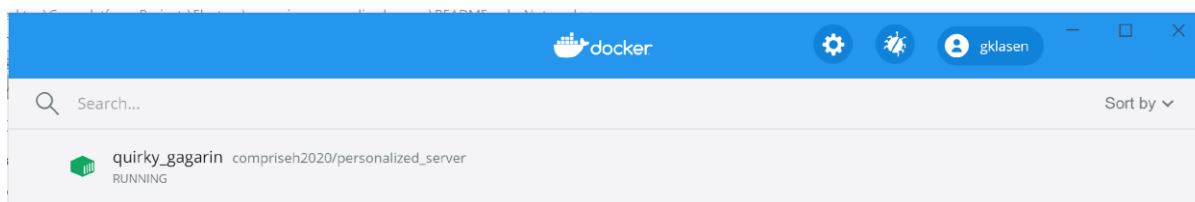
```
docker run -dp 8443:8443 -dp 4040:4040 compriseh2020/personalized_server
```

To check whether everything went well, on MacOS or Windows, open Docker Desktop (e.g., via the notification bar) and select "Dashboard".



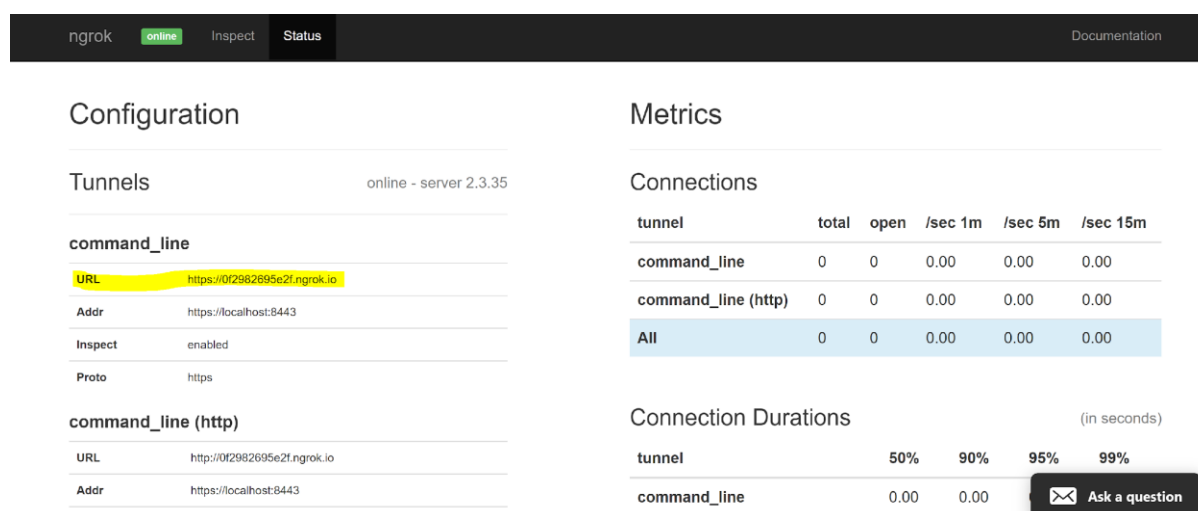
You will see your Personal Server running.

⁸ https://hub.docker.com/repository/docker/compriseh2020/personalized_server



Step 4 – Connect the COMPRISE Personal Server with the application⁹

Open your web browser and visit <http://localhost:4040/status>. The resulting webpage includes a unique, secure URL (highlighted in yellow below) which forwards all incoming traffic to the local server instance. This URL is needed because, by default, neither the localhost URL nor the URL of the local network are accessible to mobile phones from the outside. This is why a kind of proxy is needed to forward incoming traffic from the mobile phone to the server.



The screenshot shows the ngrok Status page. The 'Configuration' section displays details for a tunnel named 'command_line' which is 'online - server 2.3.35'. The 'URL' is highlighted in yellow: `https://0f2982695e2f.ngrok.io`. Other details include 'Addr: https://localhost:8443', 'Inspect: enabled', and 'Proto: https'. The 'Metrics' section shows a table of connections and connection durations.

Connections					
tunnel	total	open	/sec 1m	/sec 5m	/sec 15m
command_line	0	0	0.00	0.00	0.00
command_line (http)	0	0	0.00	0.00	0.00
All	0	0	0.00	0.00	0.00

Connection Durations (in seconds)					
tunnel	50%	90%	95%	99%	
command_line	0.00	0.00			

Copy this URL and continue with the instructions for the COMPRISE App Wizard.

2.2 COMPRISE App Wizard

The COMPRISE App Wizard is a helper tool which allows software developers to equip their solutions with COMPRISE functionalities and to enable their software to have voice-enabled, multilingual, privacy-aware behaviour.

The App Wizard currently runs with Angular v9.1.4, Electron v8.2.5, and Electron Builder v22.6.0. To install it, first install the latest LTS versions of NodeJS¹⁰ and Git¹¹, following documentation at the corresponding URLs. Clone the App Wizard repository locally:

```
git clone https://gitlab.inria.fr/comprise/comprise_app_wizard.git
```

and install its dependencies on npm (an online repository for open-source NodeJS projects) and @angular/cli in the npm global context:

```
npm install -g @angular/cli && npm install
```

⁹ This step relies on <https://ngrok.com/> as a solution to forward incoming traffic from the mobile phone to the local server instance. The instructions are specific to this solution. An alternative solution such as <https://github.com/localtunnel> will be offered in the final version of the Personal Server.

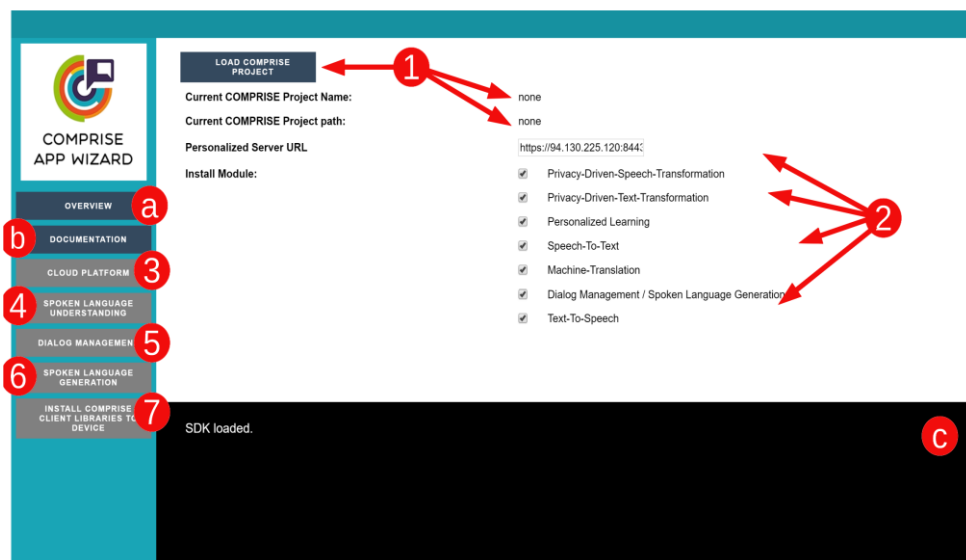
¹⁰ <https://nodejs.org/en/download/>

¹¹ <https://git-scm.com/book/en/v2/Getting-Started-Installing-Git>

Afterwards, the App Wizard can be executed with:

```
npm start
```

Once the COMPRISE App Wizard has been started, you will see the screen below, starting with the **Overview (a)** frame. If you accidentally closed the documentation, **Documentation (b)** will get it back. Sometimes, in case of installation processes or errors, the **Console (c)** can be checked for more information.

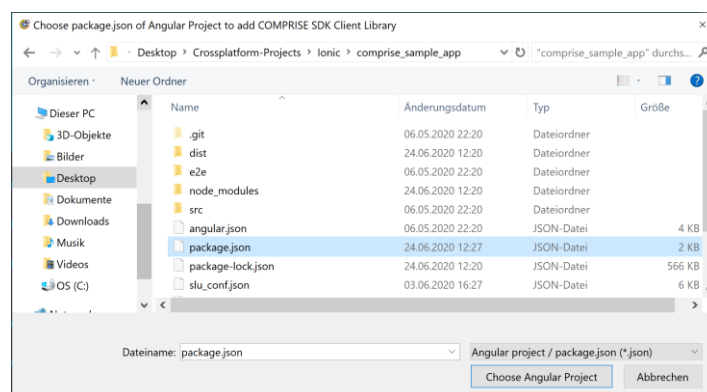


The following steps numbered 1 to 7 each correspond to a specific frame or area on the screen above.

Step 1 – COMPRISE project initialisation

The first step is to choose an existing project which needs to be enriched with COMPRISE functionality. Currently, for mobile applications, we support Angular projects within Ionic applications.

Click on "Load COMPRISE project". A file chooser will open and let you choose the project to which you want to attach the COMPRISE Client Library. Each Angular project has a `package.json` file. Enter the folder of the chosen project (here, we choose `comprise_sample_app` as an example) and select that file.



Step 2 – COMPRISE Client Library and Personal Server configuration

Still on the Overview frame, fill the Personal Server URL field with the URL generated in Section 2.2. It will be assigned to the `package.json` file of your project and be used by the Client Library to communicate with the Personal Server.

The COMPRISE Client Library consists of multiple subcomponents listed in Section 2.3 below (Operating Branch, Training Branch, Cloud Platform API). In this step, you can also choose which subcomponents of the COMPRISE Operating Branch shall be part of your mobile application. By default, every subcomponent is enabled. Subcomponents of the COMPRISE Training Branch and communication with the Cloud Platform API are expected to be always required.

In the final version of the COMPRISE App Wizard, dropdown menus will be added to let you select the desired Speech-to-Text models to be downloaded from the COMPRISE Cloud Platform. In the current version, a single pretrained Speech-to-Text model is assumed to be used for every available language.

Step 3 – Cloud Platform

When loading your project in **Step 1**, an associated identifier and storage area was automatically created within the COMPRISE Cloud Platform. The Cloud Platform is meant for uploading, storing and managing speech or text data and labels, training large-scale user independent models on these data, and making them available to COMPRISE Apps. Its functionalities include secure cloud-based data and model storage, scalable and dynamic cloud-based high-performance computing, APIs for continuous data upload and occasional model download, and general platform features (user interface, authentication, usage analytics, etc.) and procedures for data labelling and curation. For more information about the Cloud Platform UI, see Deliverable D5.3¹² “Data collection and curation features of the platform” (submitted to the European Commission on 31/08/2020).

Step 4 – Spoken Language Understanding

In this step, you can provide example sentences and corresponding intents and train the initial model for Spoken Language Understanding. Following that, you can also see how your input is interpreted by the Spoken Language Understanding subcomponent and which intent is identified. Find all documentation needed within the Tilde.AI training PDF¹³ or video¹⁴.

Step 5 – Dialogue Management

After this, you need to define how the system will react to each detected intent. The way to proceed is also documented in the Tilde.AI training PDF and video.

Step 6 – Spoken Language Generation

Based on the result of the Dialogue Management, you will need to tell the system which answer shall be generated. The way to proceed is also documented in the Tilde.AI training PDF and video.

¹² <https://www.compriseh2020.eu/files/2020/08/D5.3.pdf>

¹³ https://gitlab.inria.fr/comprise/comprise_app_wizard/-/blob/master/doc/Tilde.AI%20Training.pdf

¹⁴ <https://www.youtube.com/watch?v=fVciLmr1VDI&feature=youtu.be>

Step 7 – Attach/update the COMPRISE Client Library

By clicking **Install COMPRISE Client Library to Device**, all of the components you chose in **Step 2** will be part of the COMPRISE Client Library and installed as part of the dependencies of your application. In your console, you will see the ongoing progress of installation of the COMPRISE Client Library components and a test build, to ensure everything went well.

It should look like below. You also can use the App Wizard for updating packages already present. Then, the output might be a little bigger.

```
chunk {polyfills} polyfills-es2015.js, polyfills-es2015.js.map (polyfills) 140 kB [initial] [rendered]
chunk {polyfills-es5} polyfills-es5.js, polyfills-es5.js.map (polyfills-es5) 742 kB [initial] [rendered]
chunk {main} main-es2015.js, main-es2015.js.map (main) 42 kB [initial] [rendered]
chunk {main} main-es5.js, main-es5.js.map (main) 47.6 kB [initial] [rendered]
chunk {runtime} runtime-es2015.js, runtime-es2015.js.map (runtime) 6.16 kB [entry] [rendered]
chunk {runtime} runtime-es5.js, runtime-es5.js.map (runtime) 6.16 kB [entry] [rendered]
chunk {styles} styles-es2015.js, styles-es2015.js.map (styles) 9.76 kB [initial] [rendered]
chunk {styles} styles-es5.js, styles-es5.js.map (styles) 11 kB [initial] [rendered]
chunk {vendor} vendor-es2015.js, vendor-es2015.js.map (vendor) 3.05 MB [initial] [rendered]
chunk {vendor} vendor-es5.js, vendor-es5.js.map (vendor) 3.6 MB [initial] [rendered] Date: 2020-06-24T09:
```

The project you chose in **Step 1** now includes all of the COMPRISE functionalities you wanted to attach.

```

"dependencies": {
  "@angular/animations": "~9.0.1",
  "@angular/common": "~9.0.1",
  "@angular/compiler": "~9.0.1",
  "@angular/core": "~9.0.1",
  "@angular/forms": "~9.0.1",
  "@angular/platform-browser": "~9.0.1",
  "@angular/platform-browser-dynamic": "~9.0.1",
  "@angular/router": "~9.0.1",
  "rxjs": "~6.5.4",
  "tslib": "^1.10.0",
  "zone.js": "~0.10.2"
},
  "dependencies": {
    "@angular/animations": "~9.0.1",
    "@angular/common": "~9.0.1",
    "@angular/compiler": "~9.0.1",
    "@angular/core": "~9.0.1",
    "@angular/forms": "~9.0.1",
    "@angular/platform-browser": "~9.0.1",
    "@angular/platform-browser-dynamic": "~9.0.1",
    "@angular/router": "~9.0.1",
    "comprise_machine_translation": "latest",
    "comprise_natural_language_generation": "latest",
    "comprise_natural_language_understanding": "latest",
    "comprise_personalized_learning": "latest",
    "comprise_privacy_driven_speech_transformation": "latest",
    "comprise_privacy_driven_text_transformation": "latest",
    "comprise_speech_to_text": "latest",
    "comprise_text_to_speech": "latest",
    "rxjs": "~6.5.4",
    "tslib": "^1.10.0",
    "zone.js": "~0.10.2"
  },
  "devDependencies": {
    "@angular-devkit/build-angular": "~0.900.1",
    "typescript": "~3.5.3"
  }
}

```

Step 8 – Start with the COMPRISE Client Library!

After the setup, you can open the project in your preferred IDE, look at the COMPRISE Client Library functionalities to be used within the app, and start using them.

2.3 COMPRISE Client Library

The resulting app will be running with the help of the COMPRISE Client Library. This library integrates state-of-the-art voice technologies and new functionalities developed within WP2, WP3 and T4.1 of COMPRISE. These technologies are implemented via the following subcomponents:

- Operating Branch:
 - Speech-To-Text (including Acoustic Model Personalisation)
 - Machine Translation
 - Spoken Language Understanding
 - Dialogue Management / Spoken Language Generation
 - Text-To-Speech
- Training Branch:

- Privacy-Driven Speech Transformation (based on the COMPRISE Speech and Text Transformers)
- Privacy-Driven Text Transformation (based on the COMPRISE Text Transformer)
- Speech-To-Text Language Model Personalisation
- Spoken Language Understanding Model Personalisation
- COMPRISE Cloud Platform API.

The following subsections list the methods available in the current version of the SDK, i.e., for all subcomponents except Speech-To-Text Language Model Personalisation and Spoken Language Understanding Model Personalisation which are due at M27. For each method, we provide a short description, list the corresponding parameters and callbacks, and provide an example.

These functionalities have also been integrated as an additional example in a COMPRISE Sample App¹⁵ within a one-pager example¹⁶. This will act as a reference on how developers can integrate all the source code in a proper way.

2.3.1 Operating Branch

Component Name	Speech-To-Text	
Component Description	Transforms user's vocal input into native textual representation.	
Method Name	startRecording	
Method Description	Currently uses RecordRTC ¹⁷ to start recording the user's voice as 16 kHz, audio/wav, mono.	
Parameters	None	
Callback	Success	boolean
	Success of started operation.	
Example	<pre>import { startRecording } from 'com- prise_speech_to_text' import { HTTP } from '@ionic-native/http/ngx'; ... constructor(public http: HTTP) { ... startRecording().then(() => { console.log("Start talking"); }, (onerror) => { console.log("Following error happened :"+onerror);</pre>	

¹⁵ https://gitlab.inria.fr/comprise/comprise_sample_app

¹⁶ https://gitlab.inria.fr/comprise/comprise_sample_app/-/blob/master/src/app/home/home.page.ts

¹⁷ <https://www.npmjs.com/package/recordrtc>

	<pre> }); } </pre>
--	--------------------------------

Component Name	Speech-To-Text	
Component Description	See above	
Method Name	stopRecording	
Method Description	<p>Terminates the recording, sends the recorded audio file to the Personal Server to be processed, and gets the resulting textual answer.</p> <p>Currently uses the language parameter to pick the model to be used by the Personal Server, which includes i-vector based personalisation. Access to multiple models per language and app-specific models will be enabled via the App Wizard. X-vector based personalisation and simultaneous recording and transcription will also be provided in the final version.</p>	
Parameters	Language	String
	The language key, format "de", "en", ...	
	http	Any
	The native Plugin from Android or iOS ¹⁸	
Callback	Text	String
	Textual representation of the voice input.	
Example	<pre> import { stopRecording } from 'com- prise_speech_to_text' import { HTTP } from '@ionic-native/http/ngx'; ... constructor(public http: HTTP) { ... stopRecording(this.privateStorage.langShort, http).then((text) => { console.log("Start talking to me until I stop"); }, (onerror) => { console.log("Following error happened: "+onerror); }); } </pre>	

¹⁸ <https://github.com/silkimen/cordova-plugin-advanced-http> — This plugin is required for HTTP(S) functionality. Ionic also provides HTTP(S) functionality, but the system treats it as if the source of any call is "localhost" instead of the actual IP of the device. Using the native plugin, calls work as expected.

	<pre> } } }</pre>
--	-------------------------------

Component Name	Machine Translation	
Component Description	Translates textual input in the user's native language into a pivot language and backwards. Currently uses LetsMT ¹⁹ for LV-EN, EN-LV, and EN-DE translation. Other language pairs are (temporarily) handled by Yandex ²⁰ . Using the same language as both native and pivot language will simply return the text without changes.	
Method Name	Translate	
Method Description	Translates one language into another	
Parameters	Sentence	String
	The text to be translated	
	from	string
	The language key of the current language, format "de", "en", ...	
	to	string
	The language key of the language to translate to, format "de", "en", ...	
	letsMTID	string
	The client ID of LetsMT, in case it can be used.	
	yandexID	string
	The client ID of Yandex, in case it can be used.	
	http	any
	The native Plugin from Android or iOS	
Callback	Translation	string
	Translated Text.	
Example	<pre> import { translate } from 'comprise_machine_translation' import { HTTP } from '@ionic-native/http/ngx';</pre>	

¹⁹ <https://www.letsmt.eu/>

²⁰ <https://www.yandex.com/>

	<pre> ... constructor(public http: HTTP) { ... translate("Hey, translate me please", "en", "de", "XXXXXXXXXX", "XXXXXXXXXX", http).then((translation) => { console.log("My translation is: "+translation); }, (onerror) => { console.log("Following error happened: "+onerror); }); } </pre>
--	---

Component Name	Spoken Language Understanding	
Component Description	<p>Interprets the intent of the user in textual input.</p> <p>Currently uses the Tilde.AI²¹ API for determining possible intents. For training the corresponding model, use the COMPRISE App Wizard.</p> <p>Directly accessed without Personal Server, as Tilde.AI is running directly at partner Tilde. Planned to be moved to the Personal Server soon.</p>	
Method Name	detectIntent	
Method Description	Interprets the intent of the user.	
Parameters	Text	String
	The text to be interpreted for the intent	
	applID	String
	ID/Name of the application.	
	langID	String
	The language key of the language to translate to, format "de-de", "en-en", ...	
	apiKey	string
	The API Key to access Tilde.AI.	

²¹ <https://botdashboard.tilde.ai/Intents>

	http	any
	The native Plugin from Android or iOS	
Callback	Intent	string
	The detected intent.	
Example	<pre>import { detectIntent } from 'comprise_natural_language_understanding' import { HTTP } from '@ionic-native/http/ngx'; ... constructor(public http: HTTP) { ... detectIntent(text, "myCOMPRISEApp", "en-en", "XXXXXXXXXX", http).then((intent) => { console.log("Detected intent is: "+intent) }, (onerror) => { console.log("Following error happened: "+onerror); }); }</pre>	

Component Name	Dialogue Management / Spoken Language Generation
Component Description	<p>Based on the detected user intent, Dialogue Management decides which route to go within the current communication scenario and Spoken Language Generation creates a suitable textual answer. This module combines both components.</p> <p>Currently uses the Tilde.AI API for both Dialogue Management²² and Spoken Language Generation²³. For training of dialogue scenarios and related answers use the COMPRISE App Wizard.</p> <p>Directly accessed without Personal Server, as Tilde.AI is running directly at partner Tilde. Planned to be moved to the Personal Server soon.</p>
Method Name	openConversation
Method Description	Opens a new conversation between the user and a chatbot, including a fresh communication scenario.

²² <https://botdashboard.tilde.ai/Model>

²³ <https://botdashboard.tilde.ai/Lang>

Parameters	conversationID	string
	The ID of the conversation.	
	apiKey	string
	The API Key to access Tilde.AI.	
	http	any
	The native Plugin from Android or iOS	
Callback	conversation	object
	A "conversation" object with attributes below	
	conversation.token	string
	Token of chosen communication.	
	conversation.conversationId	string
	ID of chosen communication.	
Example	<pre>import { openConversation } from 'com- prise_natural_language_generation' import { HTTP } from '@ionic-native/http/ngx'; ... constructor(public http: HTTP) { ... openConversation("XXXXXXXXXXXX", http).then((conversation) => { console.log("Conversation ob- ject is: " + conversation); console.log("Conversation token is: " + conversation.token + " and the ID: " + conversation.conversationId); }, (onerror) => { console.log("Following error happened: "+onerror); }); }</pre>	

Component Name	Dialogue Management / Spoken Language Generation
Component Description	See above
Method Name	retrieveConversation

Method Description	Retrieves a conversation already started in the past between the user and a chatbot, including the current communication scenario at that time.	
Parameters	conversationID	string
	The ID of the conversation to be retrieved.	
	apiKey	string
	The API Key to access Tilde.AI.	
	http	any
	The native Plugin from Android or iOS	
Callback	Conversation	object
	A "conversation" object with attributes below	
	conversation.token	string
	Token of chosen communication.	
	conversation.conversationId	string
	ID of chosen communication.	
Example	<pre>import { retrieveConversation } from 'com- prise_natural_language_generation' import { HTTP } from '@ionic-native/http/ngx'; ... constructor(public http: HTTP) { ... retrieveConversation("123abc", "XXXXXXXXXXXX", http).then((conversation) => { console.log("Conversation ob- ject is: " + conversation); console.log("Conversation token is: " + conversation.token + " and the ID: " + conversation.conversationId); }, (onerror) => { console.log("Following error happened: "+onerror); }); }</pre>	

Component Name	Dialogue Management / Spoken Language Generation
----------------	---

Component Description	See above	
Method Name	generateAnswerToMessage	
Method Description	Sends an intent / a message to the chatbot of the chosen communication and get the suitable answer back.	
Parameters	conversationID	string
	The ID of the chosen communication.	
	Intent	string
	The message / the intent to communicate.	
	conversationToken	String
	The token of the chosen communication	
	http	Any
	The native Plugin from Android or iOS	
Callback	Answer	Text
	The answer to the sent message	
Example	<pre>import { generateAnswerToMessage } from 'com- prise_natural_language_generation' import { HTTP } from '@ionic-native/http/ngx'; ... constructor(public http: HTTP) { ... generateAnswerToMessage("123abc", "hungry", "XXXXXXXXXXXX", http).then((answer) => { console.log("Your answer is: "+answer); }, (onerror) => { console.log("Following error happened: "+onerror); }); }</pre>	

Component Name	Text-To-Speech
Component Description	Transforms text into voice, which will be spoken out.

	Currently uses the Cordova plugin as a solution to run native functionality on device, connected to Angular / Web-Code on the Personal Server.	
Method Name	Speak	
Method Description	Transforms text into voice, which will be spoken out.	
Parameters	tts_options	Object
	Configuration object with parameters below.	
	tts_options.text	String
	Text to be spoken out.	
	tts_options.locale	String
	The language used, format "de-DE", "en-EN", ...	
Callback	success	boolean
	Success of started operation.	
Example	<pre>declare var cordova: any; ... constructor() { ... const tts_options = {"text": "Please talk to me!", "locale": "en-EN"}; cordova.plugins.COM- PRISE_TextToSpeech.speak(tts_options, () => { console.log('Success') }, (onerror) => { console.log("Following error happened :"+onerror); }); }</pre>	

2.3.2 Training Branch

Component Name	Privacy-Driven Speech Transformation
Component Description	Removes personal information from speech by means of the COMPRISE Speech and Text Transformers. Sends speech to the Personal Server, and gets the resulting privacy-transformed speech. Based on the COMPRISE WP2

	pipeline example ²⁴ including VTLN-based voice transformation ²⁵ followed by word masking ²⁶ .	
Method Name	transformPrivateSpeech	
Method Description	Transforms speech to a version which is free of personal information. For more information, see D2.1 ²⁷ .	
Parameters	http	Any
	The native Plugin from Android or iOS	
Callback	transformedAudio	String
	Base64 string of the transformed audio in WAV	
Example	<pre>import { transformPrivateSpeech } from 'com- prise_privacy_driven_speech_transformation' import { HTTP } from '@ionic-native/http/ngx'; ... constructor(public http: HTTP) { ... transformPri- vateSpeech(http).then((transformedAudio) => { const audio = new Audio(); audio.src = "data:au- dio/wav;base64,"+transformedAudio; audio.play(); console.log("I just played your transformed Audio!"); }, (onerror) => { console.log("Following error happened :"+onerror); }); }</pre>	

Component Name	Privacy-Driven Text Transformation
Component Description	Removes personal information from text by means of the COMPRISE Text Transformer ²⁸ . Sends text to the Personal Server, and gets the resulting privacy-transformed text.
Method Name	transformPrivateText

²⁴ https://gitlab.inria.fr/comprise/development/wp2_pipeline_example

²⁵ https://gitlab.inria.fr/comprise/development/voice_transformation

²⁶ <https://gitlab.inria.fr/comprise/development/word-masking-tool>

²⁷ <https://www.compriseh2020.eu/files/2019/08/D2.1.pdf>

²⁸ https://gitlab.inria.fr/comprise/text_transformer

Method Description	<p>Transforms text to a version which is free of personal, private information, using one translation strategy among the following:</p> <ul style="list-style-type: none"> • REDACT_NE_REPLACEMENT = Deface all private entities, e.g. "Donald Trump" becomes = "XXXXXX XXXXX" • WORD_BY_WORD_REPLACEMENT = Replace private entities word by word, e.g. "Donald" as is mapped to "John", "Trump" is mapped to "Doe" • FULL_ENTITY_REPLACEMENT = Replace private entities as complete entity, e.g. "Donald Trump" as a whole is mapped to "John Doe" <p>Find more information about the differences in D2.1²⁹.</p>	
Parameters	text	string
	The text to be transformed	
	strategy	string
	The translation strategy to apply, see options above.	
	http	any
	The native Plugin from Android or iOS	
Callback	transformedText	string
	Text without sensitive information.	
Example	<pre>import { transformPrivateText, FULL_ENTITY_REPLACEMENT } from 'comprise_privacy_driven_text_transformation' import { HTTP } from '@ionic-native/http/ngx'; ... constructor(public http: HTTP) { ... transformPrivateText(text, FULL_ENTITY_REPLACEMENT, http).then((transformedText) => { console.log("I am free of private content: "+transformedText); }, (onerror) => { console.log("Following error happened :"+onerror); }); }</pre>	

²⁹ <https://www.compriseh2020.eu/files/2019/08/D2.1.pdf>

2.3.3 Cloud Platform API

Component Name	Cloud Platform
Component Description	Used to upload the register storage buckets for client applications, as well as for upload of speech and text data, and model downloads. The API is documented in the Cloud Platform portal ³⁰ and mapped to the methods listed below. Note that some of the methods will have restricted access rights, depending on whether they are used within the App Wizard or a user application.
Method Names	registerNewApplication() listApplications() editApplicationData() getApplicationData() deleteApplication() <hr/> uploadSpeechSegment() listSpeechSegments() deleteSpeechSegments() getSpeechSegment() submitSpeechTranscription() deleteSpeechSegment() <hr/> uploadTextSegment() listTextSegments() deleteTextSegments() getTextSegment() submitTextTranscription() deleteSpeechSegment() <hr/> listModelTypes() trainNewModel() getListOfModels() downloadModel() deleteModel()
Example	<pre>import { listTextSegments } from 'com- prise_cloud_platform_api' import { HTTP } from '@ionic-native/http/ngx'; ...</pre>

³⁰ <https://comprise-dev.tilde.com/dashboard/api-documentation>

	<pre> constructor(public http: HTTP) { ... listTextSegments(api_id, http).then((segments) => { console.log("These are the transformed texts!"); }, (onerror) => { console.log("Following error happened :"+onerror); }); } </pre>
--	---

3. Conclusion

This deliverable introduces the first prototype of the COMPRISE SDK, consisting of the COMPRISE Personal Server, the COMPRISE Client Library, and the COMPRISE App Wizard.

The main change compared to the architecture initially presented in Deliverable D4.1 is the introduction of the COMPRISE Personal Server. Following this change, the COMPRISE Client Library does not run on the user device at this stage, but on a Personal Server controlled by the user.

According to the implementation plan in Deliverable D4.1, the COMPRISE SDK development schedule is on time, regardless that the first version did not plan to include a Personal Server solution.

Until Deliverable D4.5, which is planned for M30, the consortium will test and evaluate all the solutions provided as part of the COMPRISE SDK to figure out necessary changes and adjustments. In addition, Speech-To-Text Language Model Personalisation, Spoken Language Understanding Model Personalisation and component updates conducted in the context of WP2, WP3, and T4.1 will be integrated within the SDK.

The results of the tests will be used to improve all components of the COMPRISE SDK, namely the App Wizard, the Client Library and the Personal Server. This will be done to continuously improve the experience of both application developers and users.