ALFRED

Personal Interactive Assistant for Independent Living and Active Ageing

WP3 – ALFRED Core

D3.5.2 – App Development Reference and Marketplace Integration

Deliverable Lead: ASC

Delivery Date: 03/2016

Dissemination Level: Public

Version 1.0

This deliverable provides a documentation of the first efforts being put into a well-defined app development reference for third party developers. It specifies the scope of this first version and the degree of fulfilment of the requirements to be covered by the documentation. Moreover, it specifies how developers will be able to develop for the Personal Assistant Service and how to consume the shared ALFRED APIs.

SEVENTH FRAMEWORK PROGRAMME

Document Status	
Deliverable Lead	Gerrit Klasen, ASC
Internal Reviewer 1	Robin Persson, TALK
Internal Reviewer 2	Tim Dutz, TUDA
Type	Deliverable
Work Package	WP3: ALFRED Core
ID	D3.5.2: App Development Reference and Marketplace Integration
Due Date	31.03.2015
Delivery Date	31.03.2015
Status	For Approval

Note

This deliverable is subject to final acceptance by the European Commission.

Disclaimer

Project Partners

Ascora GmbH, Germany	Atos Spain sau, Spain
Worldline, Spain	Charité - Universitätsmedizin Berlin - Department of Geriatrics, Germany
Asociacion de Investigacion de la Industria Textil, Spain	Technische Universität Darmstadt, Germany
National Foundation for the Elderly, The Netherlands	Talkamatic AB, Sweden
E-Seniors, France	TIE Nederland N.V., The Netherlands
IESE Business School, Spain	

Executive Summary

In the context of T3.5, the main element is the extendibility through new apps. Third party developers are welcome to contribute to the overall ALFRED eco system. Related deliverables 3.5.1 and 3.5.2 aim to support and advise them the procedure for the whole progress.

D3.5.2 however demonstrates the achievement of all formulated goals in D.3.5.1, which have not been reached so far. As the Context-Aware Dialogue Engine (CADE) and Knowledge and Information Storage (KIS) interfaces already were present, missing other API access points for the Personalization Manager (PM), Health Monitor (HM), Marketplace (MP) and Game Manager (GM) have been added to the Personal Assistant Shared library. A third party developer now has the promised, full functional access to the Personal Assistant API components. It offers inter-communication methods to every relevant module provided by the PA. As a result, third party apps are able to use the functionality of the Game Manager and others.

In addition, two third party solutions, “HealthMonitorDemo” and “CalendarAppDemo” are uploaded to the Github-organisation[1] as tangible examples. While “HealthMonitorDemo” will demonstrate the integration of CADE and the Health Monitor Client of the Personal Assistant, the “CalendarApp” will do the same for CADE and KIS. These samples are accompanied by the promised “how to”–file, explaining the integration process step by step. As shown there, third party apps connect to the Personal Assistant Service, and can also use the available APIs in the same way.

Beside the description of their integration in general, D3.5.2 additionally points out the special preparations needed for CADE, which allows the user to communicate with the ALFRED system via voice interaction. Apps can offer this service to the end user, but special preparations have to be performed to use this functionality in the app. It is necessary to provide a dialogue domain description (DDD), as described in deliverable D3.3.2, and to communicate with the CADE backend from the app itself. Other commands are referenced in the annexed “how to”-document.

Finally, the marketplace integration will allow the developer to distribute his or her app on Alfredo Marketplace Webserver. However, an end user will be able to install their apps with the Alfredo Marketplace Android app, which also is integrated into the Personal Assistant.

[1] https://github.com/ALFREDProject, as created in D.3.5.1

Table of Contents

List of Figures and Listings

List of Figures

List of Tables

Listings

1 Introduction

ALFRED – Personal Interactive Assistant for Independent Living and Active Ageing – is a project funded by the Seventh Framework Programme of the European Commission under Grant Agreement No. 611218. It will allow older people to live longer at their own homes with the possibility to act independently and to actively participate in society by providing the technological foundation for an ecosystem consisting of four pillars:

- **User-Driven Interaction Assistant** to allow older people to talk to ALFRED and to ask questions or define commands in order to solve day-to-day problems.
- **Personalized Social Inclusion** by suggesting social events to older people, taking into account their interests and their social environment.
- A more **Effective & Personalized Care** by allowing medical staff and caretakers to access the vital signs of older people monitored by (wearable) sensors.
- **Physical & Cognitive Impairments Prevention** by way of serious games that help the users to maintain and possibly even improve their physical and cognitive capabilities.

This deliverable provides a documentation of the first efforts being put into a well-defined app development reference for third party developers. It specifies how (third-party) developers will be able to develop for the Personal Assistant Service and how their applications can be enabled to consume the shared ALFRED APIs.

1.1 ALFRED Project Overview

One of the main problems of western societies is the increasing isolation of older people, who do not actively participate in society either because of missing social interactions or because of age-related impairments (physical or cognitive). The outcomes of the ALFRED project will help to overcome this problem with an interactive virtual butler (a smartphone application also called ALFRED) for older people, which is fully voice controlled.

The ALFRED project is wrapped around the following main objectives:

- To empower older people to live independently for longer by delivering a virtual butler with seamless support for tasks in and outside the home. This virtual butler (the ALFRED app) aims for a very high end-user acceptance by using a fully voice controlled and non-technical user interface.
- To prevent age-related physical and cognitive impairments with the help of personalized serious games.
- To foster active participation in society for the ageing population by suggesting and managing events and social contacts.
- And finally, to improve caring by offering direct access to vital signs for carers and other medical staff as well as alerting in case of emergencies. The data is collected by unobtrusive wearable sensors monitoring the vital signs of ALFRED's users.

To achieve its goals, the project ALFRED conducts original research from a user centred perspective and applies technologies from the fields of Ubiquitous Computing, Big Data, Serious Gaming, the Semantic Web, Cyber Physical Systems, the Internet of Things, the

Internet of Services, and Human-Computer Interaction. For more information, please refer to the project website at http://www.alfred.eu.

1.2 Deliverable Purpose, Scope and Context

The context of D.3.5.2 is based in T3.5. This task deals with the extendibility though new apps. Third party developers are welcome to contribute to the overall ALFRED ecosystem, so that the benefit of the whole ALFRED system increases.

For achieving these goals, the purpose of this deliverable is the demonstration of the fulfilment of the work in task 3.5, as first documented in D.3.5.1. To be concrete, the Personal Assistant Commons library will be demonstrated. It allows the communication between apps and the Personal Assistant (PA) and thus the external usage of all PA components. In addition, one integration use case will be shown with the Context-Aware Dialogue Engine (CADE), accompanied by a "how to"-document attached to this deliverable. To also provide practical examples, two app implementations named "HealthMonitorDemo" and "CalendarAppDemo" will be introduced. They concretize the usage for these guidelines.

1.3 Document Status and Target Audience

This document is listed as "public" in the Description-of-Work (DoW), as it provides general information about ALFRED's software extensions. While the document mainly aims at the project's contributing partners, this public deliverable can also be useful for the wider community.

1.4 Abbreviations and Glossary

A definition of common terms and roles related to the realization of the ALFRED project as well as a list of abbreviations is available in the supplementary document "Supplement: Abbreviations and Glossary", which is provided in addition to this deliverable. Further information can be found at http://www.alfred.eu.

1.5 Document Structure

This deliverable is broken down into the following sections:

- **Chapter 1** provides an introduction for this deliverable including a general overview of the project, and outlines the purpose, scope, context, status, and target audience of this deliverable.
- **Chapter 2** outlines the steps needed to work with the Personal Assistant Service.
- **Chapter 3** describes the steps needed to integrate the app into the ALFREDO marketplace.
- Finally, **Chapter 4** concludes the aforementioned content.

2 App Development Reference

Developers should consider getting their hands on the sample applications, as stated in chapter 2.5.3, to get the best in-depth information on how to handle the communication with the first prototypes. However, the following sections will give an overview on the overall API.

2.1 Context and Scope

The App Development Reference is targeted at developers for the ALFRED ecosystem. While offering practical and theoretical examples and documentation, new developers should be attracted for participating to the overall system.

2.2 State of the Prototype

In the overall architecture, the Personal Assistant and the Mobile Assistant Foundation form the central component of ALFRED. They connect all components, and allow an inter-component communication to the Game Manager, Health Monitor, CADE, Cloud Information Storage, and Personalization Manager.

Within this prototype, the Personal Assistant Commons library created in D.3.5.1 was completed to support the communication to the missing APIs. It can be found at Jenkins CI[2] and is not attached, because it is improved continuously, although its main functionalities are finished. The first prototype existing in D3.5.1 implemented first communication interfaces to CADE and KIS to bind the Personal Assistant Service to an app. The final implementation also added missing API bindings like Health Monitor, Personalization Manager, Marketplace and Game Manager. It also improved CADE bindings towards new frontend libraries. Finally, it provides a microphone button also usable for apps that use a GUI of their own.

For a better experience for third-party-developers, an organization has been created on github.com, a web-based Git repository hosting service, which should host all open, ALFRED-associated projects[3]. So, the developer is able to see all apps in only one sight.

“HealthMonitorDemo” and “CalendarAppDemo”-apps were developed and uploaded to github. Beside the improved commons-library and “how to”-documents, they form a reference for developers.

[2] http://alfred.eu:8081/job/Personal%20Assistant%20Commons/
[3] https://github.com/ALFREDProject.

Table 1: Implemented and Planned Features

Feature	Status D3.5.1	Status D3.5.2
Creation of Github organisation	√	√
Sample application for Personal Assistant usage	√	√
Description of third party app integration	In progress	√
Personal Assistant Commons (PAC) library	√	√
PAC CADE connection	Prototype	√
PAC KIS connection	√	√
PAS MP connection	X	√
PAS PM connection	X	√
PAS GM connection	X	√
PAS HM connection	X	√
Two other 3rd party solutions on Github	X	√
HowTo document for integration	X	√

2.3 Requirements and Preparations

This section defines the prerequisites needed to run the Personal Assistant (and the Mobile Assistant Foundation) on an Android device, and to guide the developer on how to write applications that can benefit from the Personal Assistant.

2.3.1 Personal Assistant

In order to allow communication between a third party app and Personal Assistant, developers need to have the latter installed on their device.

To work with the Personal Assistant, it is mandatory to get the Personal Assistant App on the development device, because all services that the ALFRED ecosystem offers are made available through the Personal Assistant Service.

The Personal Assistant App can be obtained through the ALFRED CI server[4]. Because of continuous improvement, this APK is not attached to this document.

It will run on Android Phones starting with Android targeting SDK version 19 (that is, Android 4.4). Tests showed however, that it can safely run as well under Android 5.1 . In

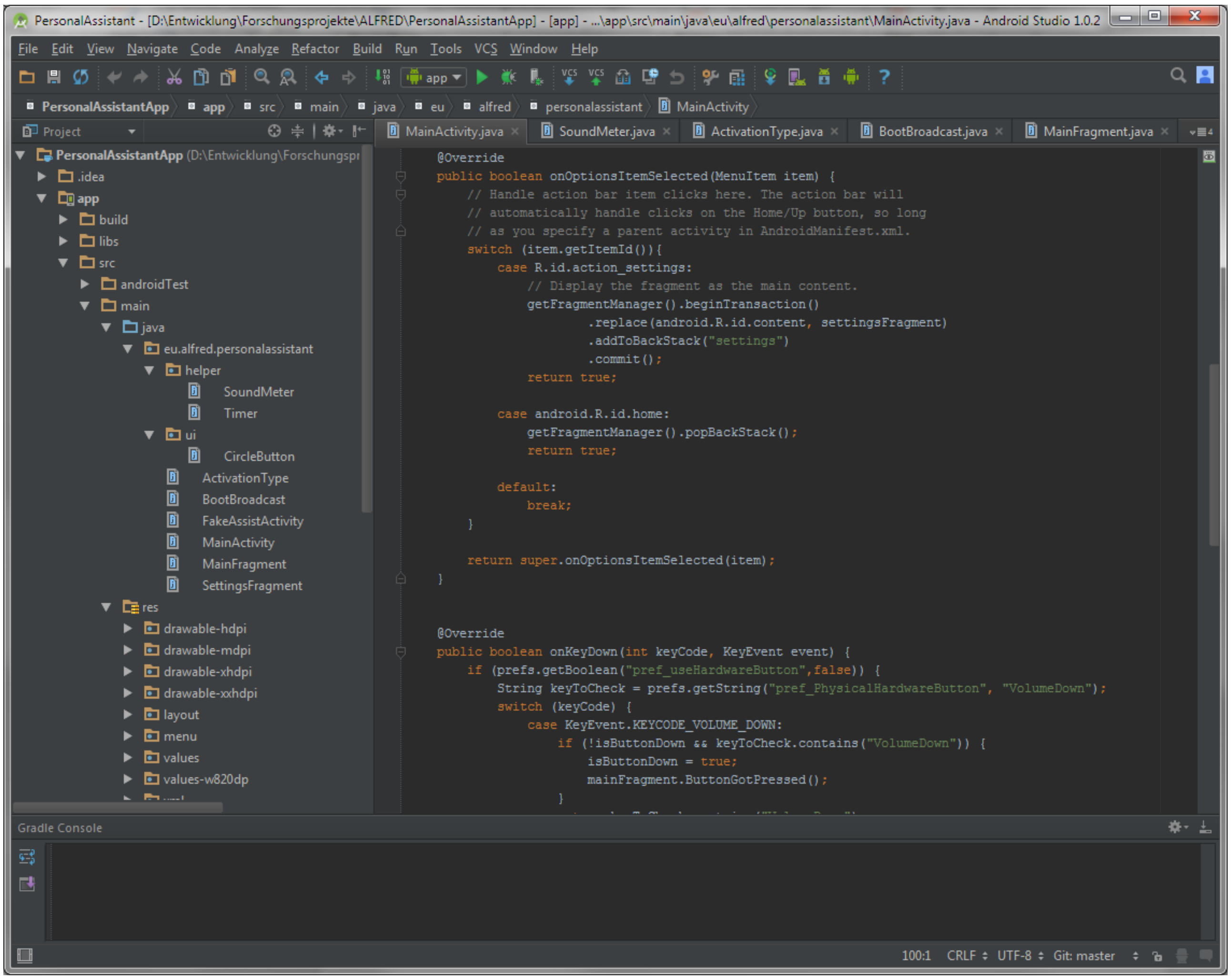
[4] http://alfred.eu:8081/job/Personal%20Assistant/

terms of devices, the Personal Assistant runs best on devices from the Google Nexus collection.

Sideloading of apps needs to be allowed for the Personal Assistant to work (See Figure 3). Besides of that, a working WiFi or mobile plan is required for the speech interaction to work.

2.3.2 Third Party Apps

For the best developing experience, it is recommended to utilize Android Studio (see Figure 1) because many of the source files used in the current prototype and in future versions will utilize features of Android Studio, e.g. so called "regions", which allow folding (= hiding) certain parts of the source code if they are not needed.


PersonalAssistant - [D:\Entwicklung\Forschungsprojekte\ALFRED\PersonalAssistantApp] - [app] - ...\app\src\main\java\eu\alfred\personalassistant\MainActivity.java - Android Studio 1.0.2
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
PersonalAssistantApp app src main java eu alfred personalassistant MainActivity
MainActivity.java SoundMeter.java ActivationType.java BootBroadcast.java MainFragment.java
Project
PersonalAssistantApp (D:\Entwicklung\Forschungspr
.idea
app
build
libs
src
androidTest
main
java
eu.alfred.personalassistant
helper
SoundMeter
Timer
ui
CircleButton
ActivationType
BootBroadcast
FakeAssistActivity
MainActivity
MainFragment
SettingsFragment
res
drawable-hdpi
drawable-mdpi
drawable-xhdpi
drawable-xxhdpi
layout
menu
values
values-w820dp
Gradle Console
100:1 CRLF UTF-8 Git: master

```java
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    // Handle action bar item clicks here. The action bar will
    // automatically handle clicks on the Home/Up button, so long
    // as you specify a parent activity in AndroidManifest.xml.
    switch (item.getItemId()){
        case R.id.action_settings:
            // Display the fragment as the main content.
            getFragmentManager().beginTransaction()
                    .replace(android.R.id.content, settingsFragment)
                    .addToBackStack("settings")
                    .commit();
            return true;

        case android.R.id.home:
            getFragmentManager().popBackStack();
            return true;

        default:
            break;
    }

    return super.onOptionsItemSelected(item);
}


@Override
public boolean onKeyDown(int keyCode, KeyEvent event) {
    if (prefs.getBoolean("pref_useHardwareButton",false)) {
        String keyToCheck = prefs.getString("pref_PhysicalHardwareButton", "VolumeDown");
        switch (keyCode) {
            case KeyEvent.KEYCODE_VOLUME_DOWN:
                if (!isButtonDown && keyToCheck.contains("VolumeDown")) {
                    isButtonDown = true;
                    mainFragment.ButtonGotPressed();
                }
```

Figure 1: Screenshot – Source Code Snippet of the PA in Android Studio

For third-party-developers, an organization has been created on github.com, a web-based Git repository hosting service, which should host all open, ALFRED-associated projects. These projects are supposed to offer a reference for the developer. During the ALFRED project, more and more exemplary application development projects will be added to the organization. As mentioned earlier, the ALFRED organization can be found at github.[5]

For D3.5.2, two 3rd party apps, HealthMonitorDemo and CalendarAppDemo, were developed and are now available on the github organization, showing different ways of interacting with different parts of the ALFRED system. While HealthMonitorDemo uses CADE and HMC APIs (responsible for speech interaction and health data screening), CalendarAppDemo uses CADE and KIS (also for speech, and storage of userdata). Third party developers will be able to use these apps as samples for their own projects. Separately, some “how to”-files will be published to explain procedures in a more conventional way. They are online at github and attached to this deliverable and describe the integration process in a more general way then what the concrete example of CADE does in section 2.5.2.

With the implementation of the “Personal Assistant Commons”-project started in D3.5.1, also a repository was added to github, which functions as a “blueprint” repository for third party developers. With this solution, they are not only able to use this template as a base for their own project, but for communicating with the Personal Assistant. It has several functions integrated like the call of the service, the initialization of CADE, and others. These are two key concepts, which can be easily applied on the other APIs available.

2.4 Deployment (Installation)

For installing the PA itself, installation of third party apps needs to be allowed on the Android Device.

On Android 4.4 (and Android 5), this can be achieved by going to the settings screen on the device, and selecting the “Security” item, as depicted in Figure 2.

[5] https://github.com/ALFREDProject.

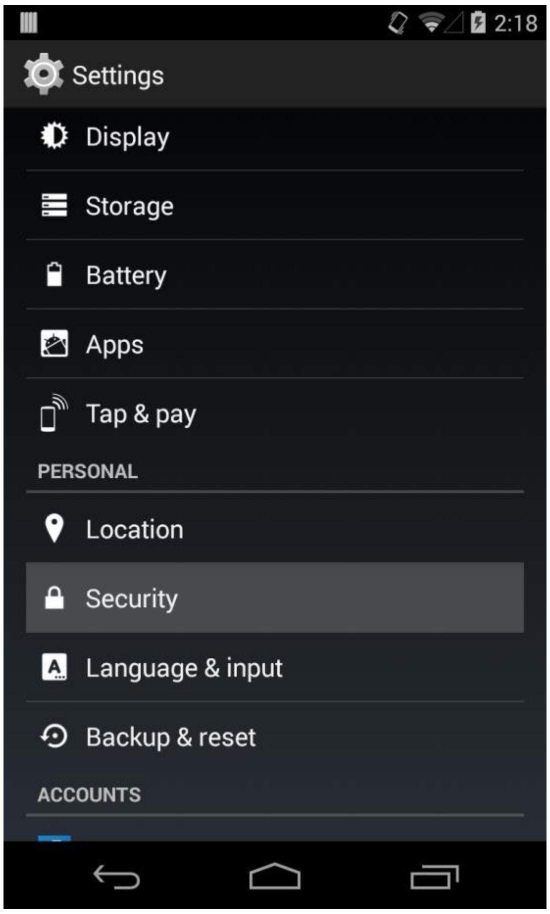
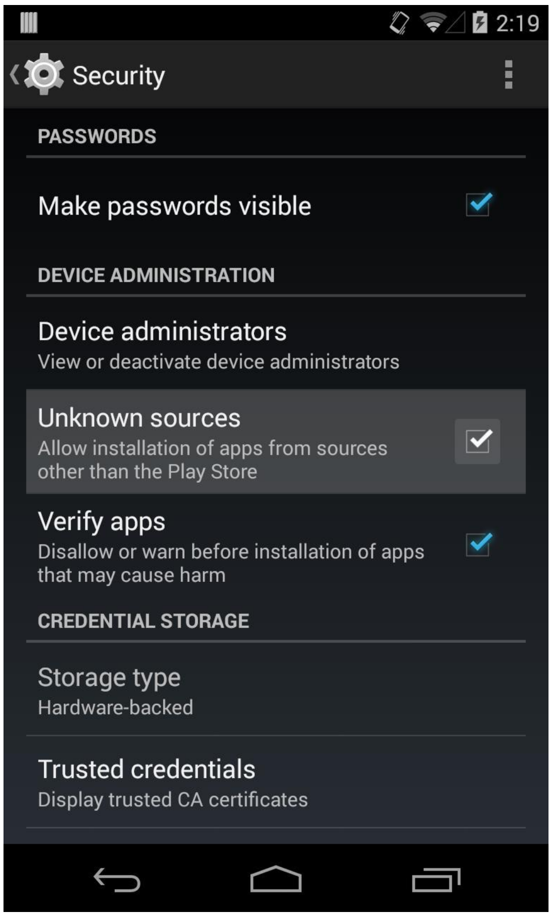


Figure 2: Screenshot Android Settings – Selecting the Security Settings



Security

PASSWORDS

Make passwords visible ✓

DEVICE ADMINISTRATION

Device administrators
View or deactivate device administrators

Unknown sources ✓
Allow installation of apps from sources other than the Play Store

Verify apps ✓
Disallow or warn before installation of apps that may cause harm

CREDENTIAL STORAGE

Storage type
Hardware-backed

Trusted credentials
Display trusted CA certificates

Figure 3: Screenshot Android Security Settings – Allowing Unknown Sources

After selecting the “Security” item, the corresponding “Security” screen will pop up and will allow you to tick the box next to “Unknown sources” (see Figure 3), which will allow the installation of the ALFRED Personal Assistant app.

The APKs are available on the ALFRED CI Server, which utilizes Jenkins[6]. A user account on this build server is required before one can download the APKs.

If a third party app should be installed, one may proceed the same way in case of testing. The official way as an end user will be to do this with the ALFREDO marketplace. How this is done is described in chapter 3: Marketplace Integration.

[6] http://alfred.eu:8081/job/Personal%20Assistant/

2.5 Execution and Usage

After the Personal Assistant and the commons-libaray were set up, the following chapter now shows the process a developer has to follow in case of his own app integration.

2.5.1 Integration of the Personal Assistant

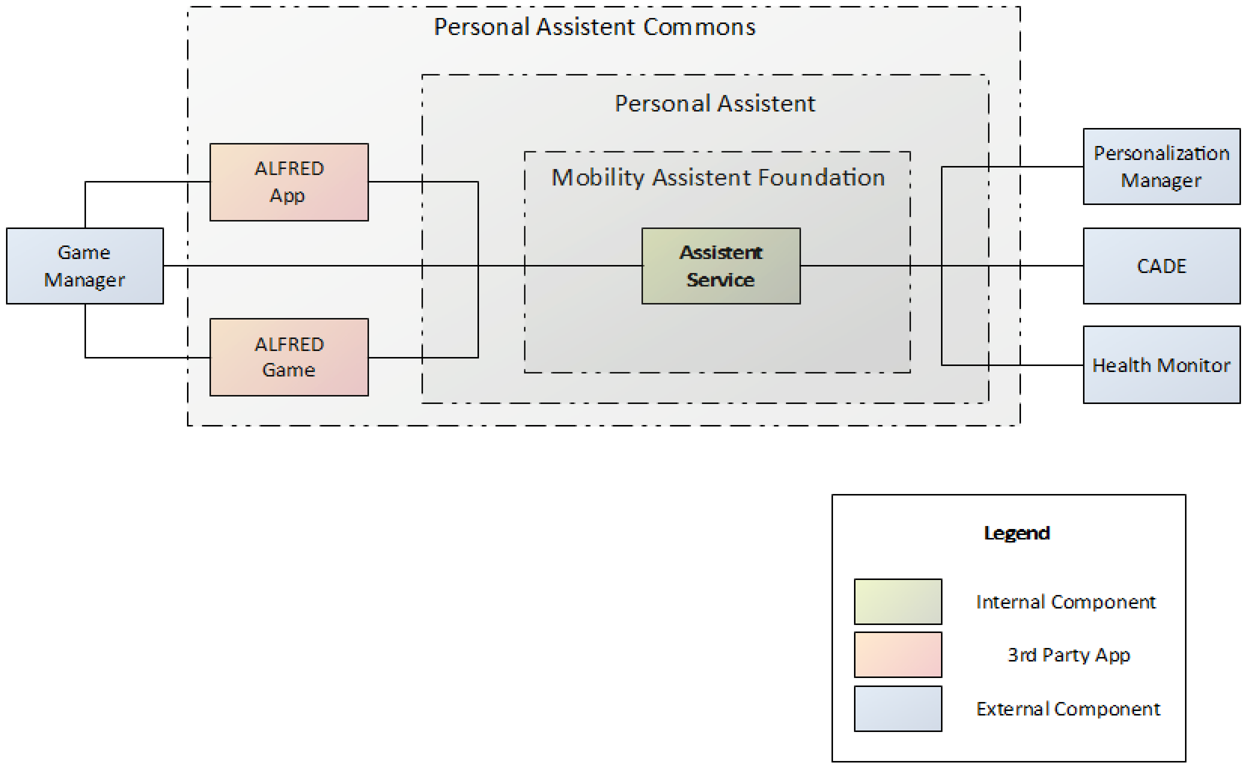


Figure 4: Overview of the Personal Assistant Architecture

An app that consumes the services of the ALFRED ecosystem needs to include the `personalassistantshared-debug.aar` which is part of the binary package you get with this deliverable. Alternatively, it can be found in the Jenkins CI already mentioned for the "PersonalAssistantCommons". This Commons-library provides templates the Personal Assistant itself and especially every third party app uses. Already in previous versions, very little code was required to actually use and connect to the Personal Assistant Service.

The `PersonalAssistant` class needs a context provided in the constructor. The only event available is the `setOnPersonalAssistantConnectionListener`, which offers the two methods `OnConnected` and `OnDisconnected`.

`OnConnected` is called after the library has established the connection to the Personal Assistant Service. In the `OnConnected` method, it is possible to initialize several APIs like the

GameManager or CADE. The `OnDisconnected` method is used likewise for cleaning up the used instances.

This is the overall initialization workflow for the Personal Assistant Service.

Finally, the `Init` method has to be called. It will do the binding to the service itself.

This procedure can also be seen in Listing 1:

Listing 1: Old way of connecting to the Personal Assistant Service

```
personalAssistant = new PersonalAssistant(this);

personalAssistant.setOnPersonalAssistantConnectionListener(new
PersonalAssistantConnection() {
    @Override
    public void OnConnected() {
        // Do some stuff
    }

    @Override
    public void OnDisconnected() {
        // Do some cleanup stuff
    }
});

personalAssistant.Init();
```

In this case, we will react on the `OnConnected` event, and will create a new instance of the Context-Aware Dialogue Engine (CADE). This object can then be used throughout the application lifetime.

In general, every wrapper library will need an injected dependency, an instance of the `Messenger` class, which gets created by the `PersonalAssistant` object.

The version in D.3.5.1 was improved a second time. The way of connecting is still valid, but is adopted by a base activity called `AppActivity`. It is part of commons library and takes the Personal Assistant registration inclusive all wrappers as seen above. So, one has no longer to use this snippet in his own in-app-code. The app only has to extend from `AppActivity`.

In addition, a user should be able to interact with the PA, although another third party app may be open. Therefore, another, smaller microphone button should be provided for every app having a GUI. `AppActivity` also does this, including registering broadcast receivers for button press actions.

So, a developer has to do nothing more than extending this class and setting the predefined listener to the microphone button in onCreate(), as seen in Listing 2: New way of connecting to the Personal Assistant Service.

Listing 2: New way of connecting to the Personal Assistant Service

```
public class MainActivity extends AppActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        circleButton = (CircleButton) findViewById(R.id.voiceControlBtn);
        circleButton.setOnTouchListener(new CircleTouchListener());
    }
}
```

2.5.2 Using the implementations with CADE

Having the Personal Assistant bound to an app, a developer may want to use some components, which are provided by the assistant, for example CADE.

If the integration of CADE has been considered for an app, some additional tasks for the integration of the overall voice command infrastructure have to be performed.

First, every developer needs to provide a Dialogue Domain Description (DDD), as described in D4.1.2. This description will then be installed into the CADE Session Manager and CADE Backend, which is running on an ALFRED server instance.

Additionally, the developer will need to integrate an additional IPC channel, which is based on Intents. By extending `AppActivity`, this is already done. The Personal Assistant Service will send a special Intent to the calling application. Important: The appname described in the DDD has to be the same as the actual Android application's name. Note the case sensitivity. This intent has the plaintext command as payload (also the same in DDD and app) and some additional parameters belonging to the command to call. Figure 5 depicts a complete run of the speech interaction.

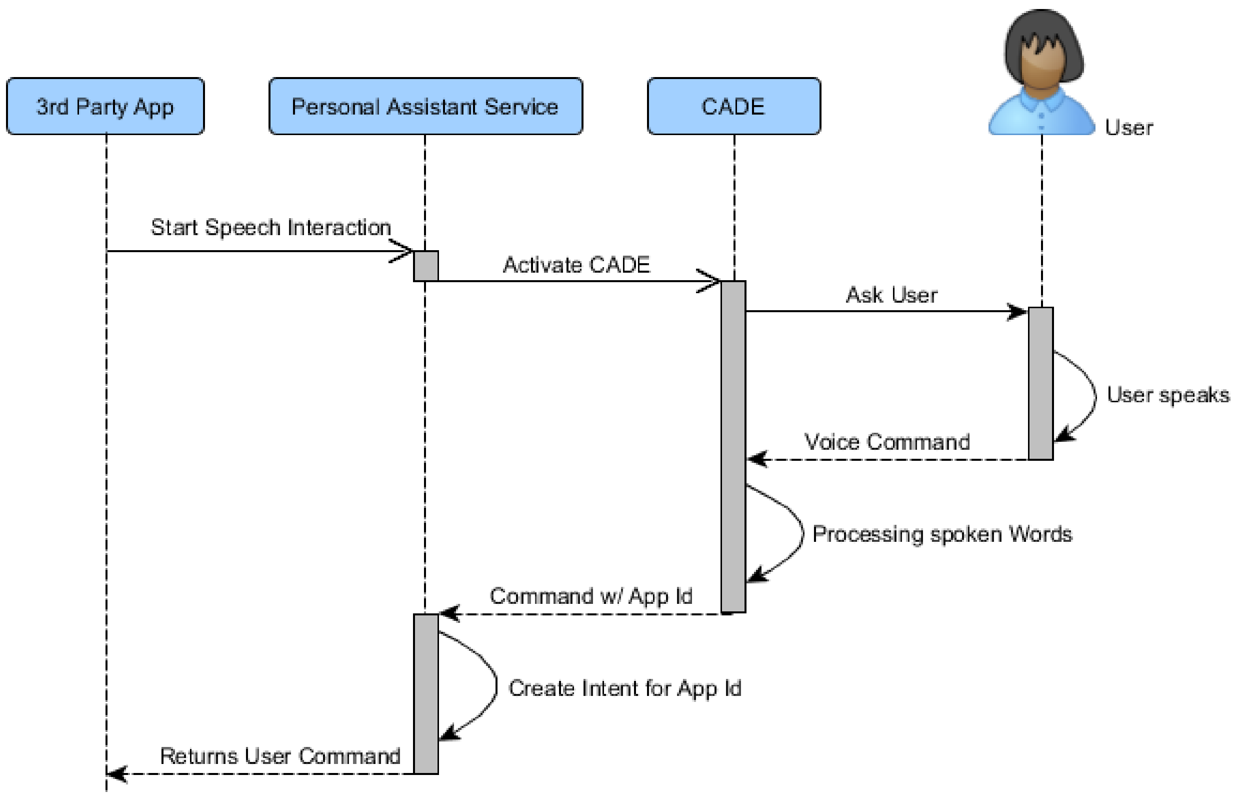


Figure 5: Sequence Diagram of the CADE Interaction for Third Party Apps

Thanks to the Personal Assistant Commons library, the app is able to handle that intent accordingly. The library provides a mocked Cade class with the method needed. The receiving app simply needs to implement an interface of the external component (CADE).

Listing 3: Handling incoming CADE command

```
public class MainActivity extends AppActivity {

    final static String SHOW_CALENDAR_ACTION = "ShowCalendarAction";

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        ...
    }

    @Override
    public void performAction(String calledAction, Map<String, String> map) {
        switch (calledAction) {
            case SHOW_CALENDAR_ACTION:
                ShowCalendarAction sca = new ShowCalendarAction(this, cade);
                sca.performAction(calledAction, map);
                break;
            default:
                break;
        }
    }

    @Override
    public void performWhQuery(String s, Map<String, String> map) {}

    @Override
    public void performValidity(String s, Map<String, String> map) {}

    @Override
    public void performEntityRecognizer(String s, Map<String, String> map) {}
}
```

CADE has four methods, which could be called in the IPC channel of the Personal Assistant (perform actions and queries, validate speech commands and recognize queried entitites). The `NewIntent()` method of `AppActivity` ensures, that the needed perform-method in the third party app will be called instantly.

For instance, an action named **"ShowCalendarAction"** was called. To encapsulate the code properly, the developer may search for the called action in a switch-environment and launch an action class of their own, carrying the context and mocked cade-instance as parameters.

Listing 4: Executing CADE Action

```
public class ShowCalendarAction implements ICadeCommand {

    MainActivity main;
    Cade cade;
    CalendarView mView;

    public ShowCalendarAction(MainActivity main, Cade cade) {
        this.main = main;
        this.cade = cade;
    }

    @Override
    public void performAction(String s, Map<String, String> map) {
        mView = (CalendarView) main.findViewById(R.id.calendar);
        mView.initCalendarView(false);
        cade.sendActionResult(true);
    }

    @Override
    public void performWhQuery(String s, Map<String, String> map) {
    }

    @Override
    public void performValidity(String s, Map<String, String> map) {
    }

    @Override
    public void performEntityRecognizer(String s, Map<String, String> map) {
    }
}
```

Having done some work in her action-class, the developer needs to send a command back to the Personal Assistant, that she has taken care of the response from the Personal Assistant Service. With the help of the mocked cade class from shared-libary, the call of `cade.sendActionResult(true);` informs the PA that the action was finished successfully.

The latter again informs CADE server about the suCcess. This behaviour of CADE could be transferred to other external components needed, so the developer has access to all resources.

2.5.3 CalendarAppDemo / HealthMonitorDemo examples

As promised earlier, two example apps were implemented to give a third party developer references, how an app can be integrated into the Personal Assistant. Both solutions extend from the commons library and are now shortly explained.

The first app is called "HealthMonitorDemo" and is responsible for showing health data to the user, including: Steps already made, body temperature, heart beat rate and respiration rate. While the first two values always are shown, the user can choose between displaying heart or respiration rate. He can do it with in-app-clickhandlers (which are not interesting for integration) or by using DDD-commands. For the latter, the app integrates CADE from the PA to be able to receive the commands. In addition, HealthMonitorDemo implements a SAF/HMC facade. Like the CADE-façade, HMC-wrapper connects to the PA to get cached measurement data. It is done similar to Listing 4, even though it is the developer's decision when to use it: Directly when the app was called (MainActivity) or after a special speech command was given (Action-Class).

The second solution called "CalendarApp" is responsible for letting users insert meetings via speech and showing a simple calendar. This app also uses DDDs, which enable the decision of which calendar month to display right now, or which meeting to insert into the calendar. For the latter, CalendarApp uses KIS for information storage, saving the meeting to insert and the corresponding date. When older users start the program with already inserted meetings, the app automatically will contact the PA over the KIS-wrapper to request all occurrences for the selected month.

2.6 Test Plan

In order to fulfil the requirements of a proper testing plan, a testing framework will be utilized. Because every app will most likely be written using Android Studio / Eclipse / similar in the Java programming language, it makes sense to use the most wide-spread testing framework for Java, JUnit, which has seamless integration into the aforementioned IDEs. Every 3rd party app, which will officially be added to the Marketplace, should have been tested properly.

The Personal Assistant (and each of its subcomponents, including the Mobile Assistant Foundation) and every example app being uploaded to Github are written based on the principles of "test-driven development". For each of the functionalities a unit test is written in first place, which keeps failing until the desired functionality is accomplished.

In terms of the new, small microphone view being present for every GUI-app, the same principles apply as for the other components. However, for testing the user interface, the Application Exerciser Monkey is used. This tool is part of the Android testing tools, and allows testing the user interface in terms of robustness.

2.7 Target Performance

The target performance for the Personal Assistant is still valid from D3.2.2. Therefore, it should be ensured that these goals are still reached, when 3rd party developer apps are used in addition to the PA performance itself.

2.8 Limitations and Further Developments

In this chapter, the current state of the deliverable will be compared to limitations and further development.

2.8.1 Limitations

The current prototype of the Personal Assistant has implemented the access to KIS, CADE, the Game Manager, the ALFREDO Marketplace and the Health Monitor. Missing APIs not having been implemented in D3.5.1 were caught up in this version. So, app developers now have full access to all functionality needed.

2.8.2 Further Development

As the current implementation phase ends in project month 30, the required API functionalities are present, as well as the 3rd party developer integration guidance. This blueprint is embodied with the PA commons library, example apps and a how to document. From project month 31 onwards, there will be continuous improvement of all components, based on the user studies feedback.

2.9 Summary

As the Personal Assistant and the Mobile Assistant Foundation were already developed, Personal Assistant Commons library in an early stage provided access to CADE and KIS. Implementing the missing APIs (MP, HM, GM, PM), the library has been finished. Nevertheless, based on the user studies, there will be continuous improvements.

The test plan and the target performance from D3.2.2 are still valid. They referenced to the Personal Assistant, whose performance should stay the same, because these plans already took 3rd party app integration into account.

The Personal Assistant Commons library was written to support third party developers. It delivers templates and wrappers for how to implement an app correctly, so that it may communicate fore- and backwards with the PA.

With the ALFRED project at Github, a repository was created for developers to have a better experience for app creation. There, one can find the two apps HealthMonitorDemo and CalendarAppDemo as a reference on how external solutions may interact with the Personal Assistant. In addition, “how to”- documents offer another attempt to explain proper development.

3 Marketplace Integration

In addition the Personal Assistant described in earlier sections, the Alfredo Marketplace is the component of the ALFRED platform that supports the location of ALFRED apps and eases their deployment.

The Alfredo Marketplace is useful for both older end users, as well as for the partners in charge of the development of ALFRED apps and 3rd developers.

3.1 Context and Scope

To be part of the ALFRED ecosystem, a mobile app has to include some integration with several components in order to accomplish its final goal, to take advantage of the services and the infrastructure of the ALFRED platform. In this section will be explained which components have to be integrated and how it has to be done.

On the other hand for locating an app in the Alfredo Marketplace, a specific workflow and testing have been passed in order to guarantee the quality of the ALFRED apps before publishing it. Each application has to pass a set of tests in order to be ready for publication.

3.2 State of the Prototype

In the first version of this deliverable, the integration between the ALFREDO Marketplace and the rest of the ALFRED components had not been implemented. Current version includes the Marketplace Android Client integration with PA and PM, and offers the wrapper for integrating the ALFREDO marketplace with the rest of the apps. Marketplace Web prototype includes the integration with CADE and the centralized OAuth.

Table 2: ALFREDO Marketplace Implemented Features

Features of ALFREDO Marketplace Client P2	D3.5.1	D3.5.2
Martetplace Android Client – Integration PA	x	√
Martetplace Android Client – Integration PM	x	ongoing
Marketplace Web integration CADE	x	√
Marketplace Web integration OAuth	x	√

Legend	
Planned Feature for P2	Blue
Feature not planned for P1	Orange

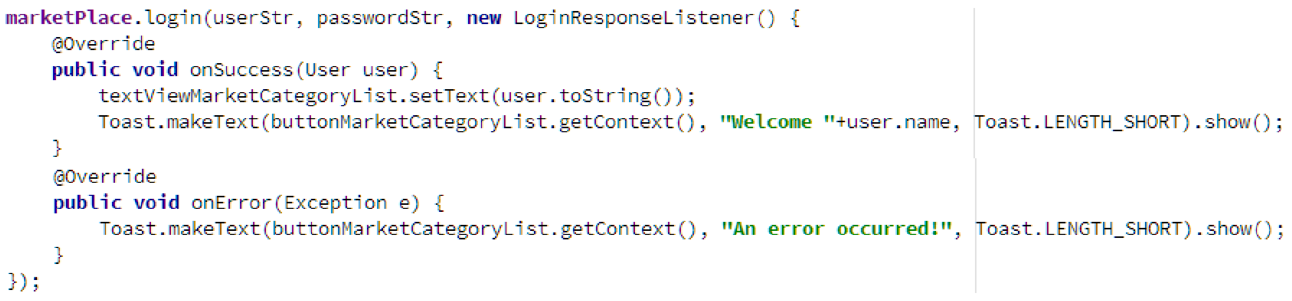
3.2.1 App Marketplace Android

The only way to allow the communication between the Marketplace services and any application is enabling the connection using the Personal Assistant Commons to the Mobile Assistant Foundation.

For example, a third party application can enable communication through the Personal Assistant Commons by sending a Message to the PA with the corresponding linking code to the service selected.

The marketplace application now uses the Personal Assistant to access the marketplace services. These services are hosted on a remote server that will enable a variety of market information. The image below shows a little example on how to use the Personal Assistant in the call login app to market.

Listing 5. Integration with Personal Assistant. Login access.



```
marketPlace.login(userStr, passwordStr, new LoginResponseListener() {
    @Override
    public void onSuccess(User user) {
        textViewMarketCategoryList.setText(user.toString());
        Toast.makeText(buttonMarketCategoryList.getContext(), "Welcome "+user.name, Toast.LENGTH_SHORT).show();
    }
    @Override
    public void onError(Exception e) {
        Toast.makeText(buttonMarketCategoryList.getContext(), "An error occurred!", Toast.LENGTH_SHORT).show();
    }
});
```

The login service shown above is one of many services that the Personal Assistant enables to the developers to use.

3.2.2 App Marketplace Web

The Marketplace Web is the web app for ALFREDO and runs independently from the ALFREDO Web Portal. The Marketplace Web allows developers to create, test and publish applications for the PAS. It has a version control for updates and a testing system to verify that the functionalities required work as expected.

In order to allow those functionalities, a set of roles has been defined. Depending on the role, a user will be able to do different actions and therefore each role has its own screens.

The Marketplace web provides the services that allow end users to browse apps published on the marketplace and install them on their devices through the ALFREDO Android Client. It does the installation through a Push Service. It also provides a search functionality that allows users to search for apps matching specific filter parameters and provides the means to read the app description. The user will be able to also see the comments and feedback provided by other users of the apps to help make the decision.

In order to integrate with the CADE, a push message is sent to the CADE when an application is published.

The Admin user can create new users, these users are integrated with the centralized OAuth with the following code:

Listing 6. Register user into the ALFRED OAuth

```
OkHttpClient client = new OkHttpClient();

MediaType mediaType = MediaType.parse("application/json;charset=UTF-8");
RequestBody body = RequestBody.create(mediaType, "{\n    \"name\": \"" +
user_name + "\",\n    \"email\": \"" + user_mail + "\",\n    \"" + user_mail
+ "\": \"\"" + user_password + "\",\n    \"roles\": [\"" + user_role +
"\"]\n}");
Request request = new Request.Builder()
  .url("http://alfred.eu:9000/auth/register")
  .post(body)
  .addHeader("accept", "application/json")
  .addHeader("content-type", "application/json;charset=UTF-8")
  .addHeader("cache-control", "no-cache")
  .build();

Response response = client.newCall(request).execute();
```

Listing 7. Login with new user using the ALFRED OAuth

```
OkHttpClient client = new OkHttpClient();

MediaType mediaType = MediaType.parse("application/json;charset=UTF-8");
RequestBody body = RequestBody.create(mediaType, "{\n    \"email\": \"" +
user_mail + "\",\n    \"password\": \"" + user_password + "\"\n}");
Request request = new Request.Builder()
  .url("http://alfred.eu:9000/auth/login")
  .post(body)
  .addHeader("accept", "application/json")
  .addHeader("content-type", "application/json;charset=UTF-8")
  .addHeader("cache-control", "no-cache")
  .build();

Response response = client.newCall(request).execute();
```

3.3 Requirements and Preparations

This section provides information about what potential users, administrators (people that will install the platform), and software developers (people that will develop mobile applications and publish them on the platform), need to do in order to use the functionalities, described in the previous section, of the delivered prototype.

3.3.1 App Marketplace Android

The main requirement is to install the Personal Assistant application, because it has the main functionality to enable the Alfred services to all the third party apps.

To create a third party application, Android Studio is chosen to integrate the Personal Assistant Commons library and create the application. Because it is the main Android development framework to create Android applications, it is recommended to install Android studio.

3.3.2 App Marketplace Web

To install the required software to make the App Marketplace Web-UI work, the next steps will have to be followed.

1. Before installing the needed packages it is recommended to update the package index of the Advanced Packing Tool (APT). This is done via the command:

```
sudo apt-get update
```

2. To have a fully working environment available, it is needed to run the following command with a user that is eligible to use “sudo” to install Tomcat server that will be responsible for deploying and presenting the App Marketplace Web-UI:

```
sudo apt-get install tomcat7
```

3. After the installation of tomcat7 a change to port 80 may be needed. (Further configurations in http://tomcat.apache.org/tomcat-7.0-doc/config/). The file /etc/tomcat7/server.xml has to be edited. Open the file with a file editor (e.g., vi)

```
sudo vi /etc/tomcat7/server.xml
```

4. In lines 72 to 75 of the standard config, the port attribute from the connector tag has to be changed from “8080” to “80”:

```
<Connector port=”8080” protocol=”HTTP/1.1”
        connectionTimeout=”20000”
        URIEncodign=”UTF-8”
        redirectPort=”8443” />
```

5. Additionally line 47 (the last line) in /etc/default/tomcat7 has to be changed to match the following:

```
AUTHBIND=yes
```

6. Now the following three commands need to be run:

```
sudo touch /etc/authbind/byport/80
sudo chmod 500 /etc/authbind/byport/80
sudo chown tomcat7 /etc/authbind/byport/80
```

7. Tomcat can now be restarted to make changes take effect:

```
sudo service tomcat7 restart
```

8. A MySQL Server database is needed to provide the persistence, to install it the next command has to be executed:

```
sudo apt-get install mysql-server-5.6
```

9. Once all the steps above are worked through, the preparations for the App Marketplace Web-UI are done.

3.4 Deployment (Installation)

This section provides guidelines on how to install and deploy the first prototype of the Media Data Streams on a Debian Linux or a Debian based derivate machine, i.e., Ubuntu or Linux Mint.

3.4.1 App Marketplace Android

The installation instructions are same as described in section 2.4.

3.4.2 App Marketplace Web-UI

To deploy the App Marketplace Web-UI in the prepared infrastructure, these steps have to be followed.

1. The following command has to be executed to create the database and populate it with the basic information needed.

```
mysql < create_database_script.sql
```

2. If the database is installed on another server, a new ddbb.properties file must be set. Set the corresponding ddbb.properties, following the one provided and execute the following command.

```
./setdatabase.sh AppMarketplaceWebUI-1.0.war ddbb.properties
```

3. After that, the application file needs to be moved to the deployment folder of Apache Tomcat.

```
sudo mv .AppMarketplaceWebUI-1.0.war /var/lib/tomcat7/webapps
```

4. Once all the steps above are done, the App Marketplace Web-UI is ready to be used.

3.5 Execution and Usage

This section describes how to use the different subcomponents of the prototype. To access the App Marketplace Web, it will be available in the deliverable repository.

App Marketplace Android

To enable communication with the Marketplace services, a call to the `mobileassitantfoundations` has to be made via the `personalassistantcommons`. For example, here are the calls from the third party app with the `personalassistantcommons` integrated library:

The call from the `personalassistantcommons` using the Messenger class to connect to the main service:

Listing 8: Message transmission trough PersonalAssistentCommons

```
Message msg = getMessage(code);
Bundle data = new Bundle();
data.putString("callerName", callerName);
msg.setData(data);
try {
    messenger.send(msg);
} catch (RemoteException e) {
    e.printStackTrace();
}
```

When the call reaches the `mobileassistantfoundations`:

Listing 9: Message handling in MobileAssistentFoundation

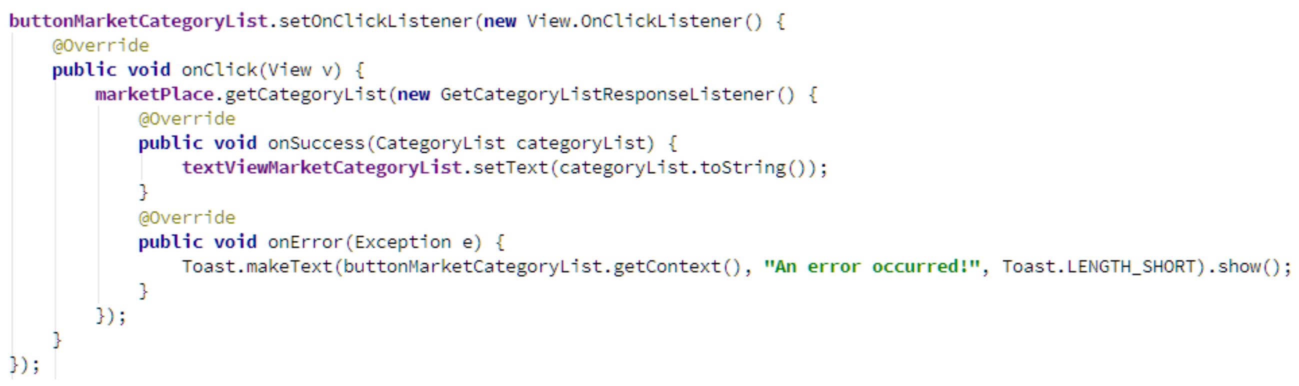
```
case MarketPlaceConstants.GET_APP_LIST:
    try {
        MarketMessageClass marketMessageClass = new MarketMessageClass();
        marketMessageClass.replyTo = msg.replyTo;
        marketMessageClass.data = msg.getData();
        marketMessageClass.json =
msg.getData().getString(eu.alfred.personalassistant.service.alfredomarket.Mar
ketPlaceConstants.EXTRAS_JSON);
        MarketCRUDTaskFactory.getInstance(getApplicationContext(),
                msg.what).execute(marketMessageClass);
    } catch (IllegalStateException e) {
        e.printStackTrace();
    }
    break;
```

Then it calls the ALFRED servers to retrieve information about the user. After that, the required information is returned back to the `personalassistantcommons` library and so on to the third party application.

The current state of the module enables applications to access any service of the marketplace. Because of the marketplace SDK migration to the Personal Assistant, any UI can be used because of its not direct dependency.

Any application included the Marketplace application use the Personal Assistant facade to access the services. For example in the listing below, there is an example that applications can use and marketplace actually does:

Listing 10: Message to obtain the current apps by category



```
buttonMarketCategoryList.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        marketPlace.getCategoryList(new GetCategoryListResponseListener() {
            @Override
            public void onSuccess(CategoryList categoryList) {
                textViewMarketCategoryList.setText(categoryList.toString());
            }
            @Override
            public void onError(Exception e) {
                Toast.makeText(buttonMarketCategoryList.getContext(), "An error occurred!", Toast.LENGTH_SHORT).show();
            }
        });
    }
});
```

3.5.1 App Marketplace Web

Depending on the role, a user will be able to do different actions and therefore each role has its own screens. The roles as well as the available screens are listed below.

- Developer
    - Allowed to: Create apps, submit media, submit information, publish the app and/or update it (Last two only after reaching the status “approved for publication”)
    - Screens: My Apps, Add App
- Tester
    - Allowed to: Go through the tests prepared for the app, answer the questions verifying that the app works properly
    - Screens: My Tests
- Approver
    - Allowed to: Create/delete/update tests, approve an app for testing and allow the developer to publish an application or reject it for some reason (the last two only after the testing is done)
    - Screens: Approve for testing, Approve for publication, Tests Management
- Admin
    - Allowed to: Every action
    - Screens: Every screen

Figure 6 shows the My Apps screen that lists all the apps created by a developer. The figure shows different lists depending on the status.

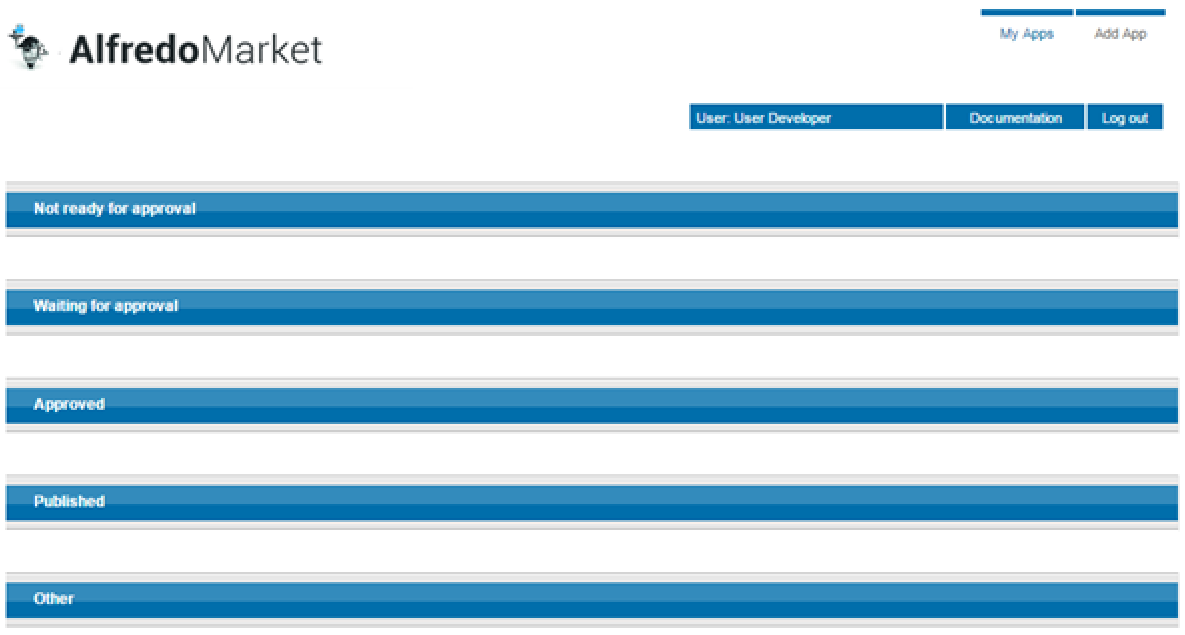
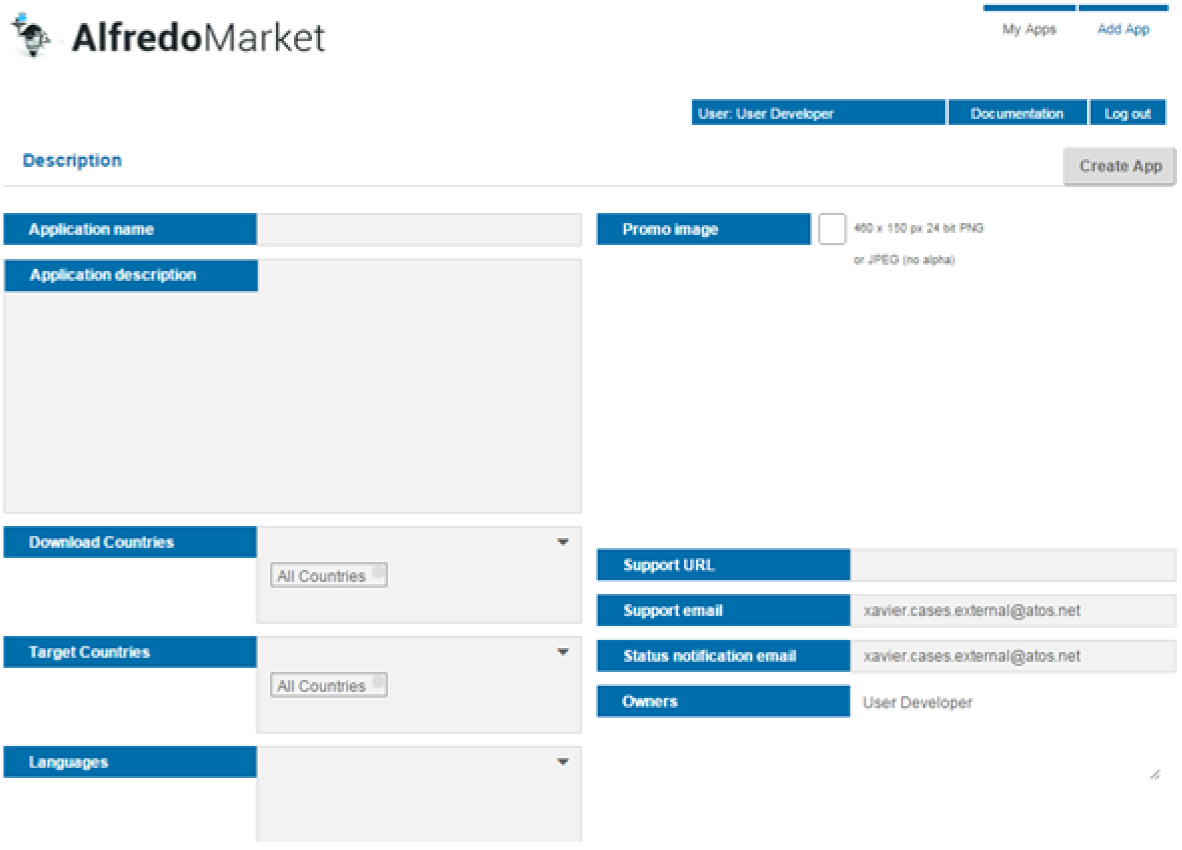


Figure 6: View of My Apps from the Developer Role's Perspective

In

Figure 7 the "Add App" screen is shown. This screen allows the developer to create a new app providing the name of the app, a description and other information to enrich it.



Application name	
Application description	
Download Countries	All Countries
Target Countries	All Countries
Languages	
Promo image	460 x 150 px 24 bit PNG or JPEG (no alpha)
Support URL	
Support email	xavier.cases.external@atos.net
Status notification email	xavier.cases.external@atos.net
Owners	User Developer

Figure 7: View of Add App from the Developer Role's Perspective

Figure 8 shows the “My Tests” screen. On this screen a tester will see the apps that have tests in progress or are already finished. By clicking on the listed apps, the tester will get into the detail-view and will be able to see questions and provide an answer to them.

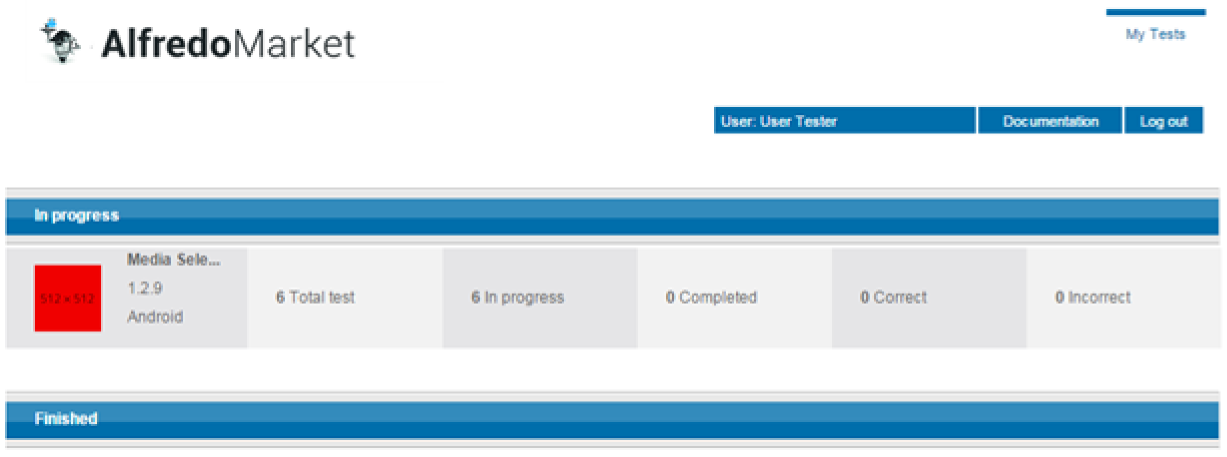


Figure 8: View of My Tests from the Tester Role’s Perspective

Figure 9 shows the “Approve for Testing” screen. On this screen, an approver can assign a group of tests to a user.

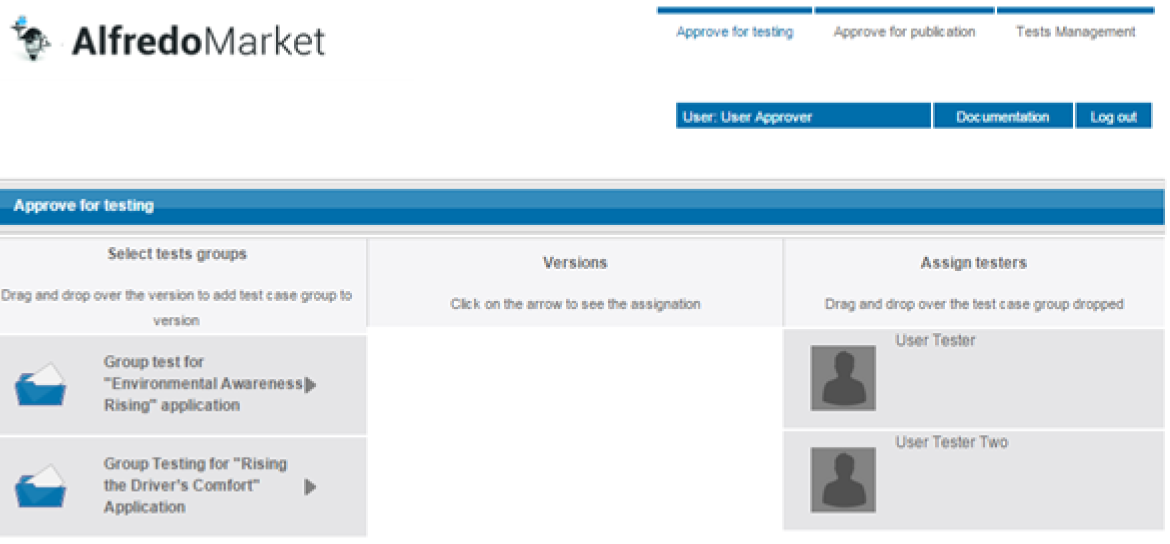
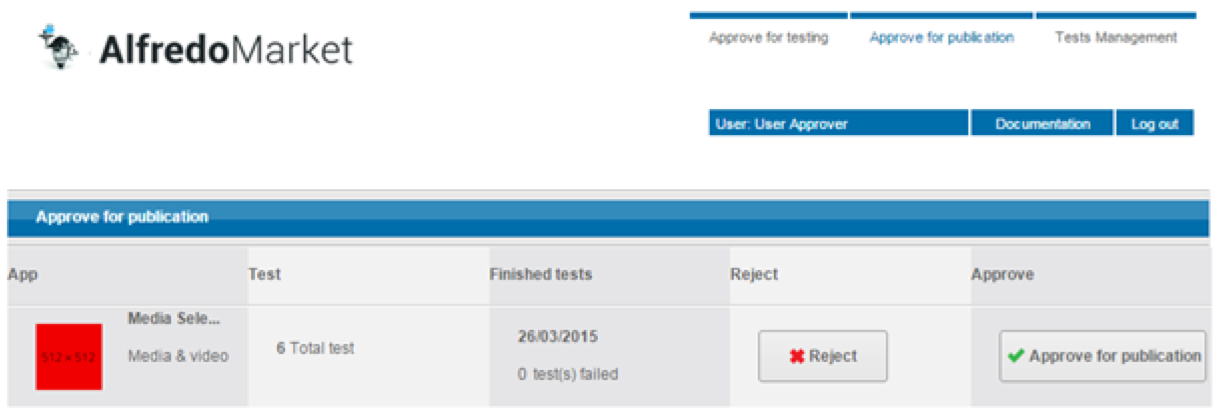

AlfredoMarket
Approve for testing
Approve for publication
Tests Management
User: User Approver
Documentation
Log out
Approve for testing
Select tests groups
Drag and drop over the version to add test case group to version
Versions
Click on the arrow to see the assignation
Assign testers
Drag and drop over the test case group dropped
Group test for "Environmental Awareness Rising" application
Group Testing for "Rising the Driver's Comfort" Application
User Tester
User Tester Two

Figure 9: View of Approve for Testing from the Approver Role’s Perspective

In Figure 10 the “Approve for publication” screen can be seen. On this screen the approver can see a list of apps and their status. From this screen the approver can “Approve” or “Reject” an app.



Approve for publication

App		Test	Finished tests	Reject	Approve
512 x 512	Media Sele... Media & video	6 Total test	26/03/2015 0 test(s) failed	✖ Reject	✔ Approve for publication

Figure 10: View of Approve for Publication from the Approver Role’s Perspective

When an application is approved for publicising, a notification is automatically sent to the CADE in order to inform it that it has to load the DDDs for this application.

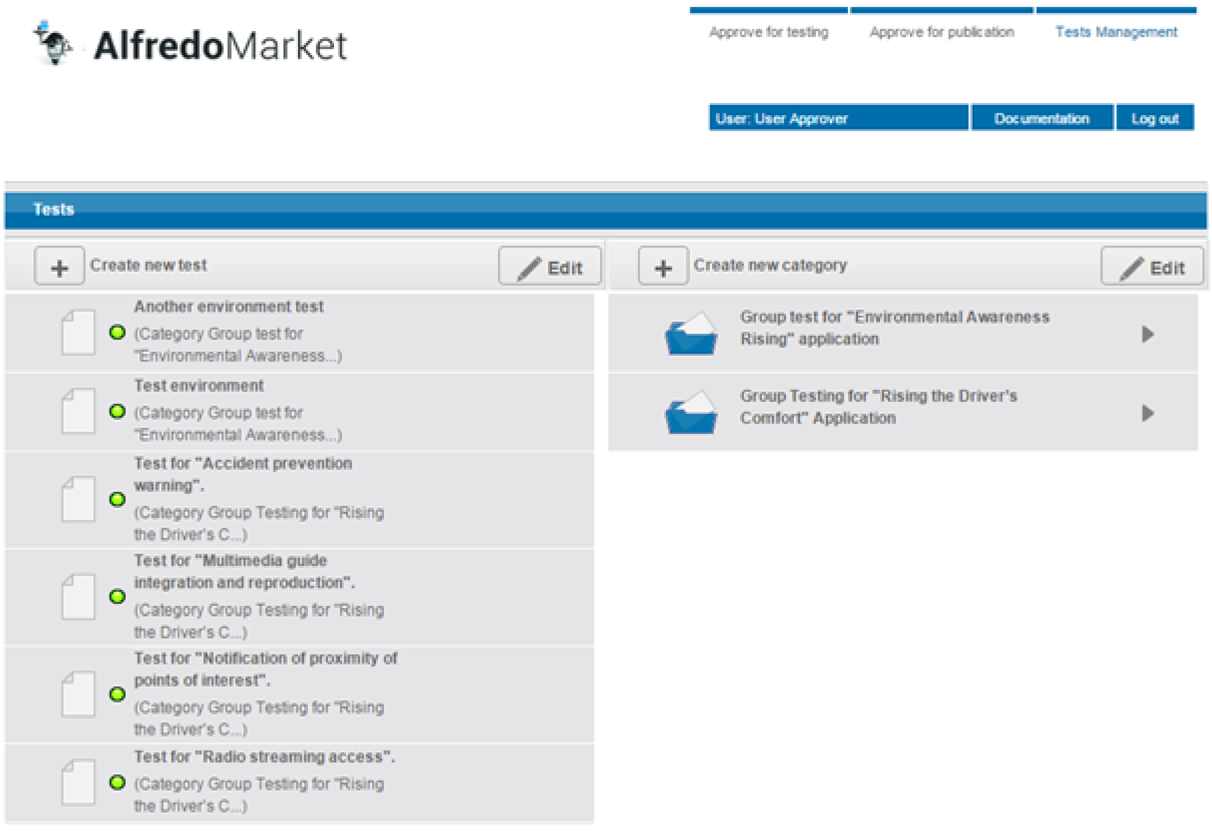
Listing 11: Message sent to the CADE.

```
OkHttpClient client = new OkHttpClient();

Request request = new Request.Builder()
  .url("http://alfred.url:9090/users/" + user_id + "/ddds/" +
application_name)
  .put(null)
  .addHeader("accept", "application/json")
  .addHeader("content-type", "application/x-www-form-urlencoded")
  .addHeader("cache-control", "no-cache")
  .build();

Response response = client.newCall(request).execute();
```

Figure 11 depicts the “Test Management” screen. On this screen an approver is able to create new tests, and he/she can group them to categories and assign specific groups of tests to an app.



Another environment test
(Category Group test for "Environmental Awareness...)

Test environment
(Category Group test for "Environmental Awareness...)

Test for "Accident prevention warning".
(Category Group Testing for "Rising the Driver's C...)

Test for "Multimedia guide integration and reproduction".
(Category Group Testing for "Rising the Driver's C...)

Test for "Notification of proximity of points of interest".
(Category Group Testing for "Rising the Driver's C...)

Test for "Radio streaming access".
(Category Group Testing for "Rising the Driver's C...)

Group test for "Environmental Awareness Rising" application

Group Testing for "Rising the Driver's Comfort" Application

Figure 11: View of Tests Management from the Approver Role’s Perspective

3.6 Test Plan

3.6.1 Test Plan Marketplace Android Client

The test plan for the Marketplace Android Client is still valid form D3.2.2. Therefore, it should be ensured that these goals are still reached which 3rd party developer apps are used.

3.6.2 Test Plan Marketplace Web

Due to the fact that the Marketplace Web is fully focused on the UI, it cannot be properly tested with unit tests or automated checks. Instead, testing will be performed based on real users who will log in to the system and test its functionality. For this purpose, different web browsers, system languages and user profiles will be tested and feedback will be fed directly to the development team, so that it can be considered for integration into the final prototpe delivered at the end of the project.

3.7 Target Performance

Table 3 list the defined key performance indicators (KPI) for this component:

Table 3: Key Performance Indicators

Topic	Description	Target KPI
Ease of Use	Ease of use is an important topic and performance indicator for this component. Each component owner should be able to use this component without greater knowledge of the connected databases but only with knowledge of the provided interfaces.	Based on a short feedback questionnaire the overall component owner contentment we want to achieve is **90%**. The questionnaire will consider the configuration complexity, integration and usage of the component and its API.
Availability	The Marketplace Web component should provide a high availability and allow users to request the event data at any time.	After the provision of the second prototype version the availability shall be **95%** or higher.
Privacy	Privacy is a rather important topic for this component, as sensitive information is displayed in the web UI.	After the second prototype, the number of sustained complaints regarding breaches of data privacy should be **0**.
Authentication Server Usage	Using the authentication server within ALFRED dramatically improves the security of the system as a whole.	The second prototype should be used by all sensitive components, dealing with user data.

3.8 Limitations and Further Developments

Although the current version of the Marketplace web prototype includes integration with the CADE server, this integration is in an early stage, and some manual actions have to be done. In further development iterations the integration with the CADE can be enhanced by sending the DDDs instead of only informing the CADE to load them from a static path.

The Marketplace Android Client is ready to be integrated with the Personalization Manager; this integration will be completed when the PM component was available.

3.9 Summary

The current deliverable includes and describes the upgraded versions of the ALFREDO prototypes (client and web).

The current prototype of the Marketplace Web has already implemented the access to the CADE and the centralized user authenticator.

The Marketplace Android Client has been integrated with the PA, and it is ready to be integrated with the PM. This integration will be done when the PM component was available.

The Marketplace provides a centralized location to place the developed applications and gives the possibility to go through a testing flow in order to guarantee the quality of the approved applications. It has simplified and integrated access to other components in order to make it easy to develop tests and integrate these applications.

The Marketplaces prototypes contain all functionalities that are needed to ease the life of users, app developers, as well as service developers. The Marketplace Web of the App Marketplace contains all means to provide users of ALFRED with a good selection of apps and offer a reliable and straight-forward UI to extend the ALFRED app ecosystem.

4 Conclusion

This deliverable has presented and described the overall approach established to get developers to work with the ALFRED ecosystem. It is mandatory for developers to understand the need for a unified approach, as the ALFRED system presents it.

Furthermore, the public visibility from a developer's perspective by choosing Github as a main platform for publicly available projects was explained. For further support, a "how to" file and two example projects, HealthMonitorDemo and CalendarAppDemo, have been uploaded. They guide developers through the implementation process. The Personal Assistant Shared library was completed, functioning as an access point to the Personal Assistant modules like CADE, GM and others. Overall, it is shown how to integrate the Personal Assistant Service itself into the app, and how to integrate the ALFRED APIs.

As a result, third party developers will be able to create their ALFRED related apps much easier, spending less time into the implementation and producing more valuable outputs for elderly people.

Form the point of view of the Alfredo Marketplace integration, the current prototype has already implemented access to CADE and the centralized user authenticator. The functionalities of Marketplace Web UI have been presented as well as the Android Marketplace app, so that developers, testers and users may interact with their corresponding components.

To extend the functionality of the PAS, The Marketplace Web-UI of the App Marketplace contains all means to provide users of ALFRED with a good selection of apps and offer a reliable and straight-forward experience.

The Application Marketplaces prototypes contain all functionalities that are needed to ease the life of users, app developers, as well as service developers.

Annex A: How to integrate 3rd party apps into Personal Assistent (PA)

Getting the connection library for the Personal Assistent

1. First step is to inherit the „Personal Assistent Shared“-library into your app.

   This libary enables the developer to access functionalities from
   - CADE (Context-Aware Dialogue Engine)
   - KIS (Knowledge and Information Storage)
   - GM (Game Manager)
   - HM (Health Monitor)
   - PM (Personalization Manager)

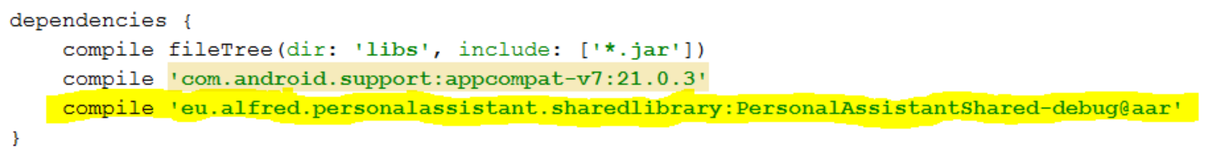
   … which are all part of the Personal Assistent.

2. Therefore, clone git@alfred.eu:dgilbert/personalassistantcommons.git.

3. The cloned project has following path:
   \personalassistantcommons\PersonalAssistantShared\build\outputs\aar
   You can find PersonalAssistentShared-debug.aar there. Copy it.
4. Paste the file into **yourproject\app\libs**

5. Run Android Studio. There, put into your build.gradle (module):

   ```
   'eu.alfred.personalassistant.sharedlibrary:PersonalAssistantShared-
   debug@aar'
   ```



```
dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    compile 'com.android.support:appcompat-v7:21.0.3'
    compile 'eu.alfred.personalassistant.sharedlibrary:PersonalAssistantShared-debug@aar'
}
```

6. Build the project.

Using the library in your app

Now, the Personal Assistent Shared-library is in your project. To use it correctly and enable it for accessing PA modules / being accessed

1. Now, your MainActivity can extend from „AppActivity“.

What AppActivity does:

- It already integrates the every Personal Assistent Module.
- In addition, it registers the „CircleButton“, the PA also uses.
- Registers corresponding recievers (for start / stop listening).

So, your on create looks like:

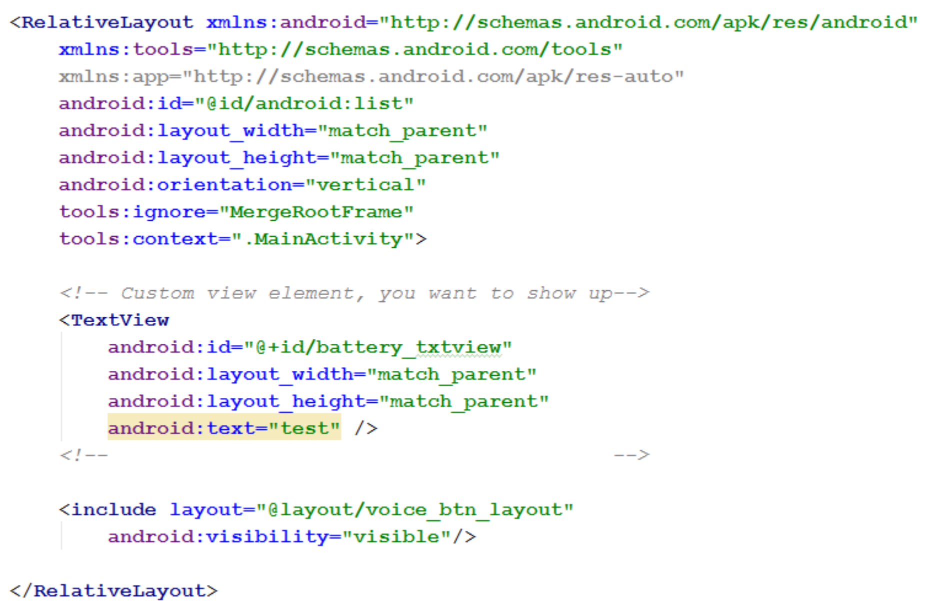
```
public class MainActivity extends AppActivity
{

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        circleButton = (CircleButton) findViewById(R.id.voiceControlBtn);
        circleButton.setOnTouchListener(new CircleTouchListener());
    }
}
```

IMPORTANT: Dont implement your own TouchListener, otherwise it will not work.

2. As you can see, you are referencing to a „CircleButton“. It is an overlay to your app GUI, so that you also will be able to continue speaking, although you are not in the Personal Assistent App itself. Define the Button also in your activity_main.xml-file:



```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:id="@id/android:list"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:ignore="MergeRootFrame"
    tools:context=".MainActivity">

    <!-- Custom view element, you want to show up-->
    <TextView
        android:id="@+id/battery_txtview"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:text="test" />
    <!--                                          -->

    <include layout="@layout/voice_btn_layout"
        android:visibility="visible"/>

</RelativeLayout>
```

3. Till now, the app has a microphone button. Of course, you want to use the GameManager, HealthMonitor etc. as well. In addition, some methods only should be called if they are accessed via speech. Thats why you AppActivity forces you to override the following four methods:

```
public class MainActivity extends AppActivity {

    final static String HELP_TO_POSTURE_ACTION = "HowToPostureAction";

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        circleButton = (CircleButton) findViewById(R.id.voiceControlBtn);
        circleButton.setOnTouchListener(new CircleTouchListener());
    }

    @Override
    public void onNewIntent(Intent intent) {
        super.onNewIntent(intent);
    }

    @Override
    public void performAction(final String calledAction, final Map<String,
String> map) {
        switch (calledAction) {
            case HELP_TO_POSTURE_ACTION:
                HelpToPostureAction htpa = new HelpToPostureAction(this, cade);
                htpa.performAction(calledAction, map);
                break;
            default:
                break;
        }
    }
```

- PerformAction is called every time, when your DDD sends an action command to your device (for instance, call my buddy John)
- PerformWhQuery is called every time, when your DDD sends a query to your device (for instance, return the phone number for John)
- PerformValidity is called every time, when your DDD sends a validity request to your device (for instance, is there a phone number for my parameter „John“?)
- PerformEntityRecognizer is called every time, when your DDD asks your device, if is has entries for a request (for instance, are there phone numbers for my parameter „John“?)

- Every of these methods have firstly the name of the action as a parameter, secondly a map of other arguments your action to call might use. Here, actionname is „HowToPostureAction“, which is figured out in the switch-construct.

⇨ In this example, we want to call an action, which shows an image with body posture instructions. Because we may launch different kinds of actions / query, it is recommended to use own action /

query classes for this, like „HelpToPostureAction“. You could give the context and wrappers, you want to use as parameters.

4. Lets see the action we just called.

Important here: Implement ICadeCommand as an interface, so that „performAction(…)“ and others are callable in this class as well.

Here, we use the arguments, our called action has beside it (Map<String,String>). For this example, the map has one key-value-pair. The key is called „selected_posture“ and is self explaining. The value however could be another String called „lie“, „sit“ or „stand“. Dependent on what the user said, the app sets an ImageView to its screen, telling how to sit, stand or lie correctly.

```
public class HelpToPostureAction implements ICadeCommand {

    final static String LIE = "lie";
    final static String SIT = "sit";
    final static String STAND = "stand";

    MainActivity main;
    Cade cade;

    public HelpToPostureAction(MainActivity main, Cade cade) {
        this.main = main;
        this.cade = cade;
    }

    @Override
    public void performAction(String s, Map<String, String> map) {

        ImageView howToImage = (ImageView) main.findViewById(R.id.image_howto);
        String posture = map.get("selected_posture");

        if(posture.equals(STAND)) {
            howToImage.setImageResource(R.drawable.howto_stand);
        } else if(posture.equals(SIT)) {
            howToImage.setImageResource(R.drawable.howto_sit);
        } else {
            howToImage.setImageResource(R.drawable.howto_lie);
        }

        cade.sendActionResult(true);
    }

    @Override
    public void performWhQuery(String s, Map<String, String> map) {}

    @Override
    public void performValidity(String s, Map<String, String> map) {}

    @Override
    public void performEntityRecognizer(String s, Map<String, String> map) {}
}
```

5. So, we have launched our app with speech, and did an action with it. Finally, we have to inform CADE, that our speech ended, to CADE my say something like a results report.
   Therefore, we use „cade.sendActionResult(true)“. The same procedure has to be followed for queries, validity checks, and entity recognizers.

6. Of course, as said in the beginning, you are able to use other APIs like KIS. Just use them in your action / query just before you call „cade.sendActionResult(true)“. Here an example for KIS, reading from a bucket created before, searching for entries for a specific date.
   Except for CADE, most of other API methods have responses. So, continue coding when OnSuccess was called.

```
JSONObject obj = new JSONObject();
try {
   obj.put("date", day+"_"+(month+1)+"_"+year);
} catch (Exception e) {}


cloudStorage.readJsonArray("GarysCalendarBucket", obj, new
BucketResponse() {
    @Override
    public void OnSuccess(JSONObject jsonObject) {

    }

    @Override
    public void OnSuccess(JSONArray jsonArray) {
        try {
            for(int i = 0; i < jsonArray.length(); i++) {
                TextView tv = new TextView(getApplicationContext());
                tv.setText(jsonArray.getJSONObject(i).toString());
                eventView.addView(tv);
            }
        } catch (JSONException e) {
            e.printStackTrace();
        }
    }

    @Override
    public void OnSuccess(byte[] bytes) {

    }

    @Override
    public void OnError(Exception e) {

    }
});
```

To get an overview of all commands you have (Inclusive their responses), see

CADE:
http://alfred.eu:9292/dgilbert/personalassistantcommons/blob/master/PersonalAssistantShared/src/main/java/eu/alfred/api/speech/Cade.java
Note: Don't use StartListening / StopListening. It is already done by „AppActivity".

GameManager:
http://alfred.eu:9292/dgilbert/personalassistantcommons/blob/master/PersonalAssistantShared/src/main/java/eu/alfred/api/gamemanager/GameManager.java
MarketPlace:
http://alfred.eu:9292/dgilbert/personalassistantcommons/blob/master/PersonalAssistantShared/src/main/java/eu/alfred/api/market/MarketPlace.java
Note: Should only be used from the Marketplace App.

PersonalizationManager:

http://alfred.eu:9292/dgilbert/personalassistantcommons/blob/master/PersonalAssistantShared/src/main/java/eu/alfred/api/personalization/webservice/PersonalizationManager.java
HealthMonitor:
http://alfred.eu:9292/dgilbert/personalassistantcommons/blob/master/PersonalAssistantShared/src/main/java/eu/alfred/api/sensors/SAFDataFacade.java
Knowledge and Information Storage:
http://alfred.eu:9292/dgilbert/personalassistantcommons/blob/master/PersonalAssistantShared/src/main/java/eu/alfred/api/storage/CloudStorage.java

7. No problems so far? Congratulations, you just developed an app interacting with the Personal Assistent.