POSEiDON

PersOnalized Smart Environments to increase Inclusion of people with DOwn's syndrome

Deliverable D4.5

HCI user and developer manuals

Call:	FP7-ICT-2013-10
Objective:	ICT-2013.5.3 ICT for smart and personalised inclusion
Contractual delivery date:	30.04.2015 (M18)
Actual delivery date:	01.05.2015
Version:	v1
Editor:	Silvia Rus (FhG)
Contributors:	Riitta Hellman, Karde Erlend Øverby. Karde Lars Thomay Boye, Tellu Alexandra Covaci, MU Fengling Han, Karde Martin Majewski, FhG Steeven Zeiss, FhG
Reviewers:	Terje Grimstad, Karde Dean Kraemer, MU
Dissemination level:	Public
Number of pages:	35

Content

1 HCI components and the rationale of UI in POSEIDON

The four HCI components of the POSEIDON system are:

- POSEIDON web – a web application that offers basic calendar and reminder capabilities, but also extends these by allowing to add additional information needed for the special calendar appointment, ensuring an extended functionality.
- POSEIDON app – a mobile application which offers diverse functionality like a calendar and navigation services, being strongly interconnected with the POSEIDON web and the stationary navigation training
- Navigation training system using Virtual Reality (VR)- offers the functionality to prepare and enrich routes with different media like photos and special instructions, allowing for the primary user to train in a safe environment
- Moneyhandlin App with the interactive table (CapTap)– offers training for recognizing coins and bills and paying products as well. Real money can be moved on the surface of table. The table recognizes which coin has been selected.

Appendix 1, POSEIDON guidelines for developing accessible user interfaces, explains the concept of the POSEIDON family design and the influencing factors which lead to the current design.

In the following the HCI interface for each of these components will be described.

2 Poseidon Web

2.1 Introduction

Poseidon web is the personalized environment for attentive users, which are called the carers in POSEIDON project context. The target user of Poseidon web is the carers of people with Down's syndrome, who are attentive of people with Down's syndrome and helps to plan their daily activities. The main component of the Poseidon web is the personalized calendar which helps to schedule the daily events of people with Down's syndrome with specialized IT support. The Poseidon web is cooperated with the Poseidon app running on a mobile system, typically a smart phone, to coordinate the attentive users and the primary users.

In the following section, we will first introduce the Poseidon system composition and functions in Section 3.2 and then give a screen shot based detailed illustration of the usage of the system in Section 3.3.

2.2 Preparation and system introduction

In order to run the Poseidon web, the carers have to have a running operating system which is connected to the internet. The system has to install an up-to-date web browsers, e.g. Internet explorer (IE), Chrome, or Firefox browser on windows operating system or safari on iOS operating system. There is no specific system performance requirement.

1. Google Calendar account. We choose Google as the public calendar server to provide the service. In order to access the Google calendar service, the carer needs to apply for a Google account for each primary user. The document for helping apply for a Google account can be obtained from https://accounts.google.com/SignUp.
2. POSEIDON web. Poseidon web is a web based system that can be accessible via any browsers. The main functionality contains three parts: First, a personalized event scheduling calendar;

monitoring, showing information reported from the mobile app such as position, battery and possibly others. The third function is personalization: setting user preferences.

3. Personalization. The Poseidon web is a customized and personalized event scheduling system with additional IT supports. E.g. the primary user who is holding the mobile version of Poseidon app can be displayed on the map of Poseidon web.

2.3 Use of Poseidon web

In the section, we will describe the usage of Poseidon web in three basic scenarios.

1. Login and change system setting.
   Use the distributed primary user name and password to login to the Poseidon web, as shown in Figure 1. The Web can be opened with the following link:
   http://ri.smarttracker.no/poseidon.

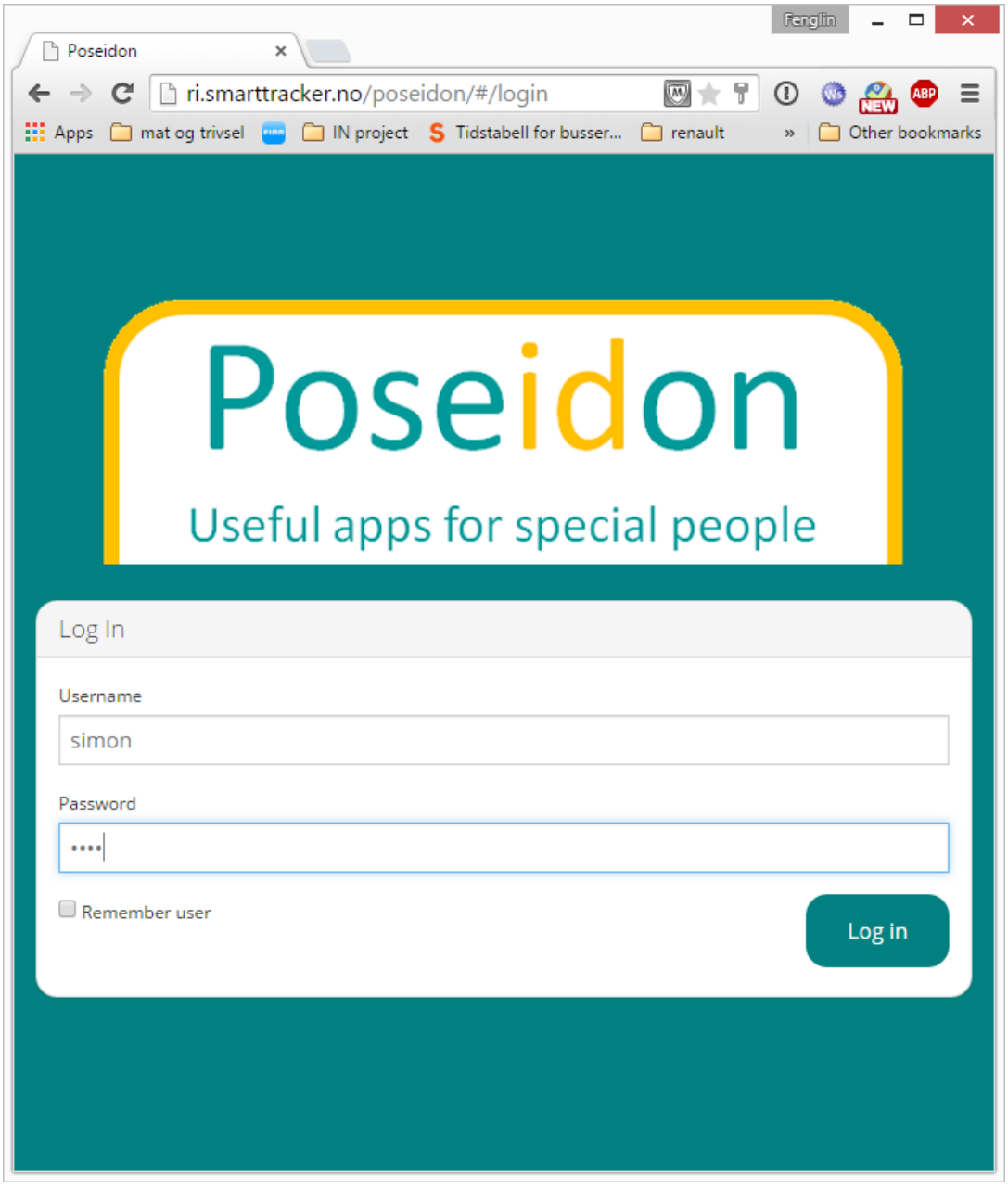


Figure 1 Login Interface (click the Remember user to store the user information)

After logged into the system, you can click the gear icon (on the top-right corner) to go to system setting to change the system language.

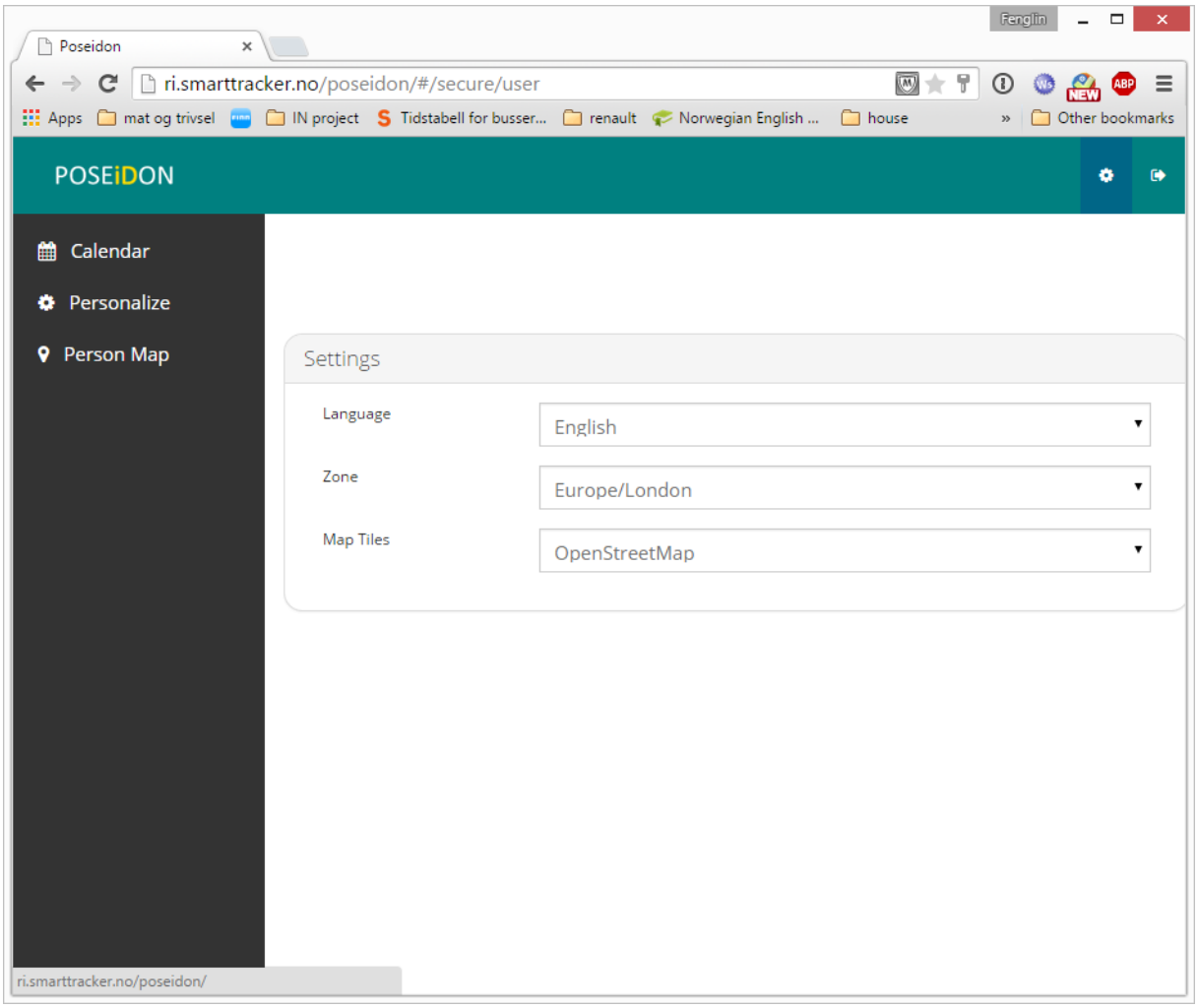


Figure 2 Change language setting

2. Connect to Google Account
    - On the left side of the web, you can see a POSEIDON logo on the top followed by a navigation bar containing calendar, Personalize and Monitor. Click the calendar icon or text to see the calendar.
    - Click on *Connect Google calendar* button (Figure 3) to connect to Google calendar service. You might have other google account, if so click on *add account* to change to the Google account you are going to connect to published event scheduling.
    - Some browser might prevent popup page like in Figure 4, you need to manually enable the popup of authorization page.

Figure 4 and 5 show the interface that Google requires your login information to permit Poseidon web to publish events.

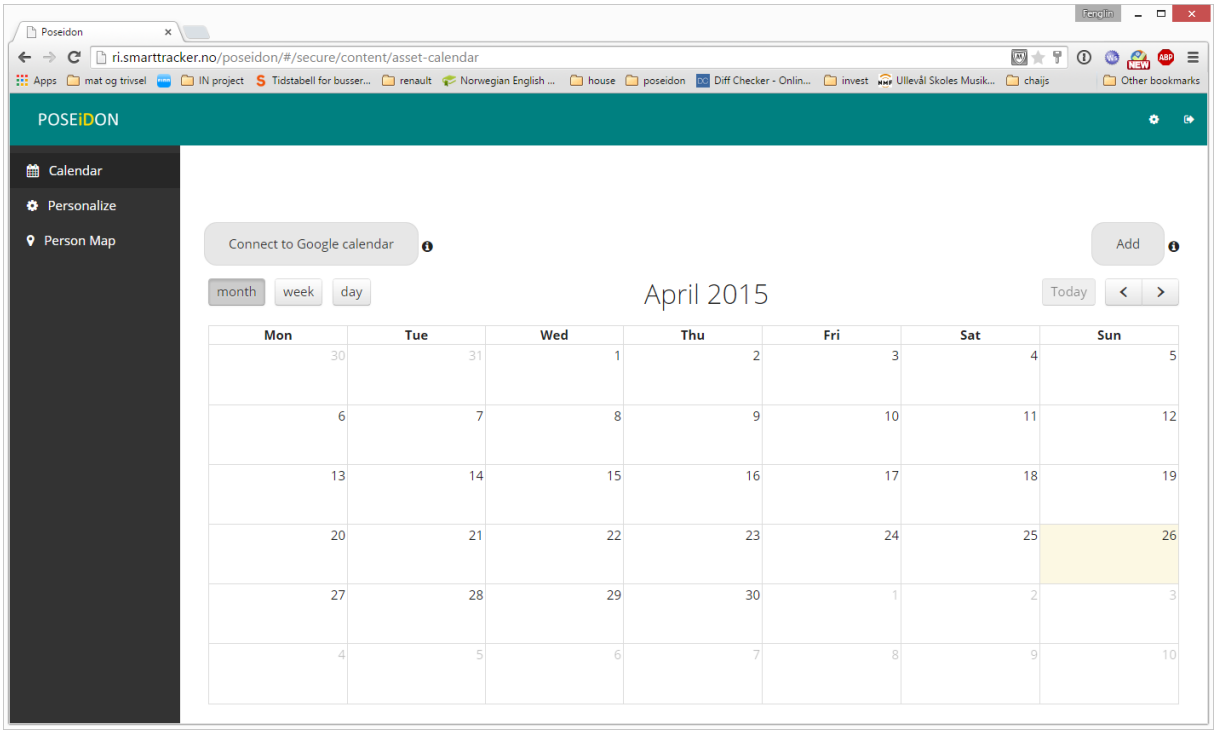


Figure 3 Calendar view

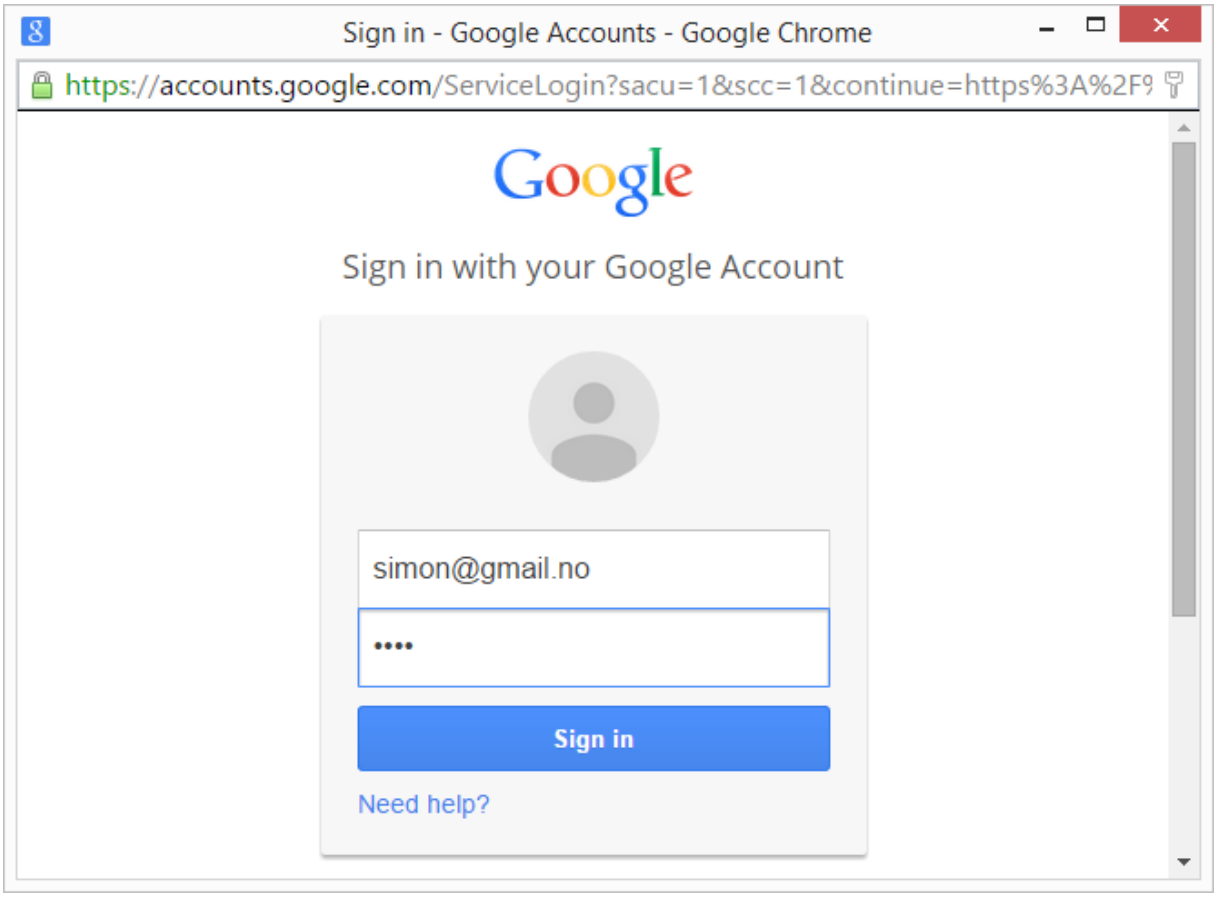


Figure 4 Interface shows the interface that Google requires your login information to permit Poseidon web to publish events.

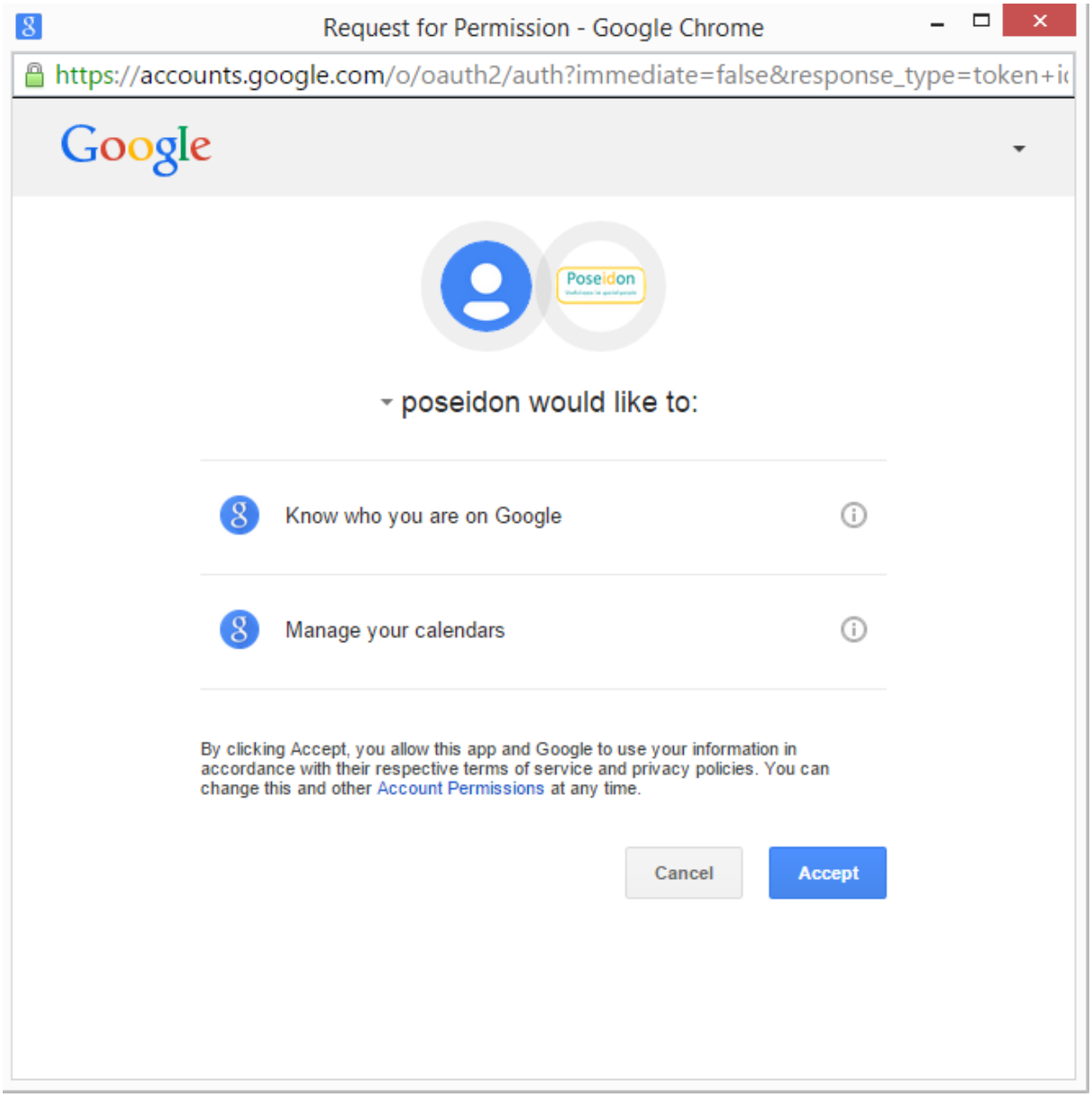


Figure 5 Click accept to give Poseidon web this ability to publish event

3. Working with Calendar

The basic calendar use contains adding, updating and deleting an event.

- Add event. Click a blank area of a calendar day or click the *add* button can add a calendar event.
- In the popup page as shown in Figure 6, you can choose the event starting date, hour and minutes and ending date, hour and minutes. This has to be done sequentially.

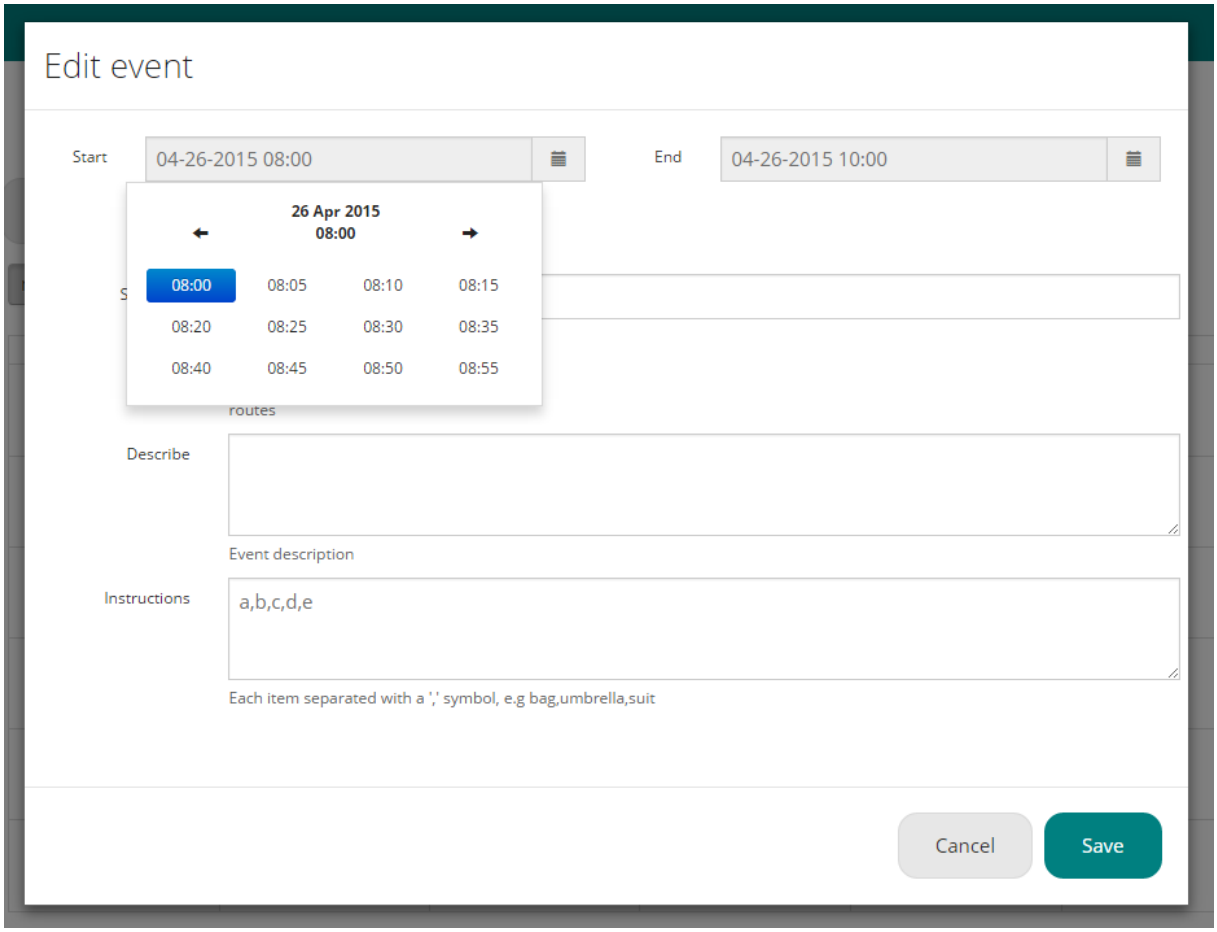


Figure 6 Popup page for editing event

- Figure 7 shows an edited event "*Go to school*", the fields include starting and ending time, summary, routes, description and instructions, description are optional. The routes are predefined in Poseidon system. You can add an alarm for the event by clicking the *alarm* check box and edit the minute's number prior to the event.
- Click *save* to save the event on Poseidon web and publish it to the calendar server such that the Poseidon app on the mobile device can receive it. From now on you can discover the interaction between the Poseidon Web and the mobile app for the primary user.

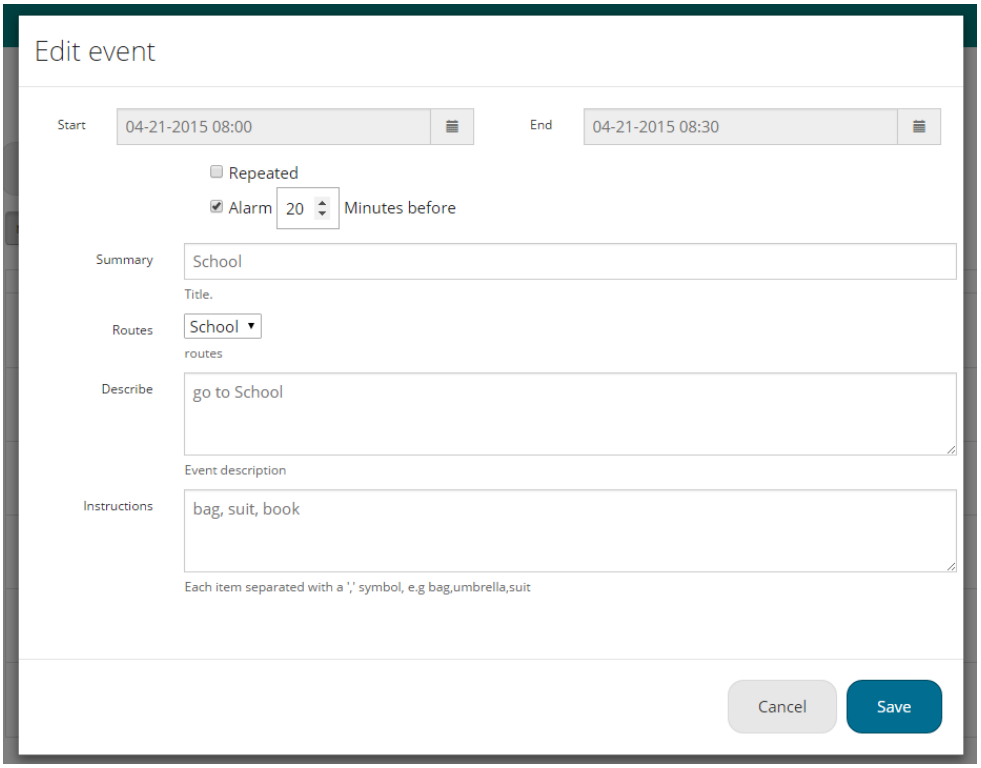


Figure 7 Go to school event

- After adding an event, you can click on the event icon on the calendar to delete or update it.
- The event editing are personalized to Poseidon use, and passed to the mobile app. Figure 8 shows a possible outlook of the mobile app for a calendar for how to use the mobile app, please refer to screen shots of the mobile Poseidon app.

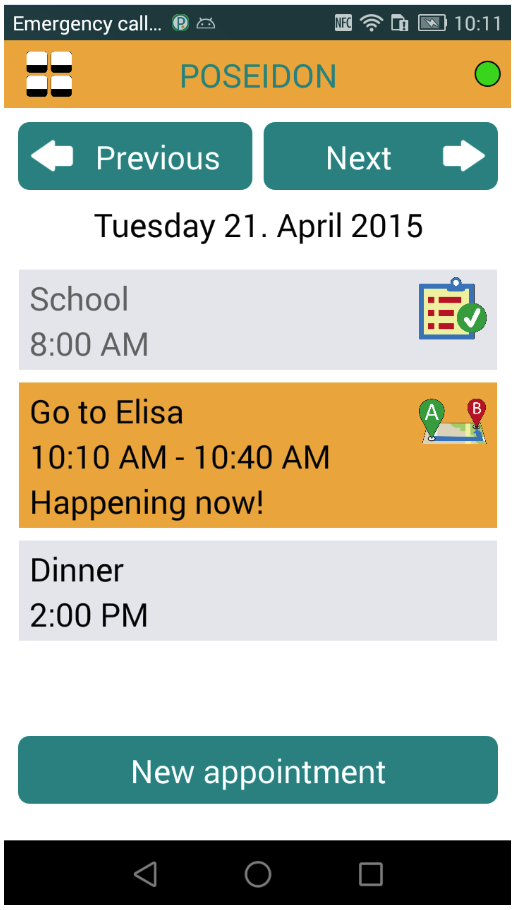


Figure 8 events passed to the mobile Poseidon app

2.4 Conclusions

In summary, the Poseidon web is easy to use browser application that can edit personalized, extensible events. Notifications of events and instruction can be added and automatically shown on the mobile app.

3 Poseidon App - Android User Interface Implementation

Android was selected as the primary platform for mobile (tablet and phone) applications in the project, and the platform for the first pilot, previously discussed in D5.1. When designing a mobile application, there are a number of aspects to keep in mind if the result is to be user friendly. One needs to know the possibilities and limitations of the hardware and the operating system of the target devices. One must also consider the type of functionality which will be included and the type of user we are targeting. Finally, it is important to not only consider the wanted functionality, but also non-functional requirements and how to handle all the errors and exceptions which may occur. As the focus of this deliverable is on user interfaces, we will not cover all these aspects here. We will mainly examine the possibilities and challenges for user interface implementation on Android, with a special focus on usability and personalisation/adaptivity as these are key aspects for the POSEIDON solution.

However, working with Android, as with any of the mobile platforms, it is important to understand that we do not have complete freedom as application developers. We work within a context which places restrictions on what we are allowed to do, and in addition there are user interface principles followed by most apps which we should follow unless we have very good reasons for doing otherwise. Android is an app platform rather than a general purpose operating system, designed with a specific

type of use in mind, to keep the user in control of the device and minimize the possibility for making an app that violates the user's wishes or drains the battery. If we don't understand our target platform we risk planning features which cannot be implemented as well as making an app which does not behave the way users expect. Therefore, we look at some important topics of Android user interaction.

When reading this chapter, also keep in mind that Android has evolved a lot during the years it has been one of the two big app platforms. Aspects of user interaction have changed quite a bit. This is a usability problem, especially since Android devices typically only get a few updates before they are stranded on an outdated version, meaning not all devices have the same Android version. In addition, different manufacturers add their own features to the system, and while these modifications do not usually go very deep, they affect the look and feel of the user interface.

The Android developer web page[1] provides very comprehensive documentation for all aspects of Android development, and is the main source for this chapter, along with Tellu's experience developing both commercial apps and research prototypes on Android.

3.1 Android Device Interface

We start with a quick look at the fixed parts of the interfaces of Android phones and tablets. Just about all devices have the same three physical buttons along the sides: a power button and volume up/down buttons. Mobile devices are rarely turned completely off, but the power button is used to turn on and off the screen. All devices also have some navigation buttons (usually touch-sensitive areas rather than physical buttons) along the bottom of the front. These were originally below the screen, and there were four of them. Their order and icons varied between phone manufacturers and models, which was bad for usability. From Android version 4.0, the first version to fit both phones and tablets, navigation buttons could be shown on screen by the Android system. Google prefers this approach but does not enforce it on manufactures, so some (notably Samsung) still have external buttons of their own design.

Any app design, including POSEIDON, must take these navigation buttons into consideration. They are the intended way to navigate between functions in the device, and some of them are completely outside the control of the app, so we cannot prevent the user from using them. We may incorporate some of their functions into our own interfaces for usability's sake, but we should keep in mind that screen space is limited, and that those who have Android experience expect the buttons to have their standard functions.

Two buttons are found in all interface variations: the home and back buttons. The home button (usually a house icon) hides any app currently shown, to bring up the home screen. This button can't be disabled by the app, and means the user can always leave our app. If we really want it to be persistent, we can detect every time it is removed and bring it back again, but this is generally a bad practice as it takes control away from the user and makes the device unusable for anything else. Note that the home screen is also just an app, a so-called launcher designed to start other apps, and it is possible to make our app a launcher and use it as the home screen, although it is always possible for the user to revert to the default launcher that comes with the device.

[1] http://developer.android.com

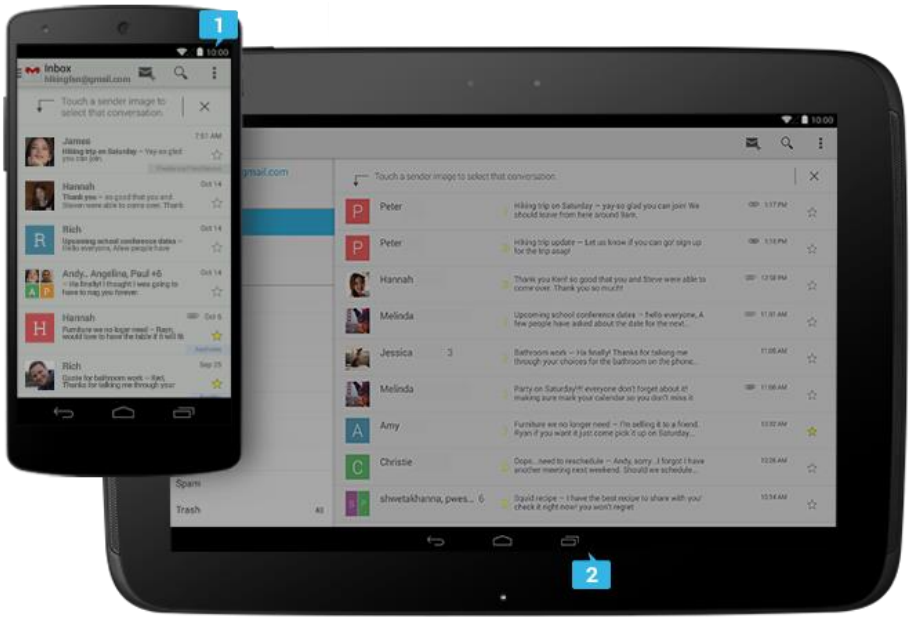


Figure 9 Reference phone and tablet from Google. Note status lines along the top (1) and navigation lines along the bottom (2)

The back button (with arrow icon) is meant to work much like in a web browser, going back to the previous screen. This button is under app control. It is generally a good idea to let the end-user backtrack in our app, as he would expect if he has previous Android experience, and to leave the app when backtracking beyond the first view (it can still be active in the background).

Of the four buttons on older phones, the other two were menu and search. The menu button brings up a pop-up menu, if one is defined by the app. A modern Android design guideline is to put the menu items in an action bar along the top of the screen instead, but the popup menu is still much used, and Samsung devices still has the menu button. The search button is not usually found on modern devices. It may be overridden by the app, otherwise it will bring up the built-in search function of the phone, hiding whatever the user was previously doing.

Finally, navigation includes a list of previously used apps, to allow the user to quickly switch between tasks. This function has its own button in the modern on-screen interface, but other designs do not. When there is no dedicated button, keeping the home button pressed for some time brings up this list. As with the home button, this function cannot be prevented by the app.

In addition to considering the use of these buttons in relation to the app being designed, one should also keep their location in mind when designing a screen layout. Putting a row of buttons along the bottom of the screen within the app interface makes it easy to hit the navigation buttons by mistake, especially for end-users with some impaired motor control. It may also be somewhat visually confusing, although their styles will be clearly distinct, with the device's buttons usually being quite small white symbols on pure black.

If the device has on-screen navigation, the navigation bar is a constant presence along the bottom of the screen. All devices have a status bar along the top. This shows vital status such as the time and indicators for battery and networks, and it can show notifications – messages from apps or the system. The status bar can be pulled down to show the notifications in full, and pressing a notification can bring up the app in question. It is possible to have our app hide the status bar, though we need good reasons to do so, as it is something users need if they are to use the device for more than our app. The details

of these bars have actually changed quite a bit with different Android versions. On tablets they used to be combined in a single bar, and it is now possible to temporarily hide both bars, though the user can always drag the navigation bar back on screen.

The last interface feature which must be considered is the keyboard. Just about all Android devices today rely on “soft” keyboards shown on the screen. A soft keyboard will automatically show up when an input field is selected, and the app may also request it specifically. Note that soft keyboards are part of the Android system, not the app, and that device manufactures and individual users may install various keyboard types and switch between them. It need not be a keyboard in the strict sense either; it can be based on voice or handwriting input. This is a point of user customization of the device, but outside the scope of an app.

So we have no control over this when designing the app, we just need to keep in mind that the keyboard will probably show up when needed, taking over some of the screen space. In the vertical orientation, the keyboard will typically take up the lower half of the screen. The app interface can either shrink, effectively halving in size, or let the keyboard cover one half. The effect of the keyboard must be considered when designing the interface. On phones, the keyboard with an input field will generally take over the whole screen in the horizontal orientation. Regarding keyboards, it is also important to keep in mind that these are usually quite small and cumbersome to use, so it’s always a good idea to design to minimize the need for keyboard input.

3.2 Android System and Application Model

As was discussed in the framework deliverable (D5.1), mobile apps can either be built “natively” for an app platform such as Android or iOS, or by using web technology so that it can (theoretically) run on all major mobile platforms. We chose the first approach for the main POSEIDON mobile prototype, partly because we have more of an existing code base to work from, but most importantly because we have access to much more device functionality in this way, which is needed for such things as sensor tracking in the background. For developing an assistive app we typically want a high degree of control of the device. But while an Android app has more control of the underlying system than a web application, and also somewhat more than an iPhone app, there are still clear restrictions. An app is not like a native piece of software for a traditional desktop computer, which can have pretty much full access to the system. Instead, it should be thought of as a plug-in to the app platform, where it must co-exist with many other apps as well as a system that control it all and doesn’t give full system access to any app.

There are some central principles behind the Android application model. One is that the user should always be in control – the user can always close an app or change system-wide settings which affects apps. Another is that of modularity and reuse – the user interfaces of an app should be able to mix with other components in the system, and use other components which provides the necessary functionality rather than reinventing the wheel. To give an example of both of these principles, let’s say we want our app to send out an email. The Android way of doing this, is to send what is called an *intent* to the system, with the email details such as address and text. The system will then ask the user which of the email applications installed on the device should be used (the user can set a default to avoid this question), and then bring up the email app with the email filled in, so that the user can choose to send it or not. While these principles are good in many cases, they are problematic for us. Our app gives up control, and the user may be confused by the various user interaction presented. We cannot provide instructions for this, as it depends on what email apps are installed as well as Android version and settings. It is technically possible for our app to send the email itself, but some things, such as making a phone call, cannot be done without invoking system components and giving up control.

Another important principle stems from the fact that Android is made for mobile systems, particularly phones – an app can be closed by the system at any time, such as when battery is low or when a phone call is coming in. In fact, the Android execution model is quite different from that of traditional desktop applications, and it is important to understand it because of its implications for user interaction design. The end-user is not supposed to turn apps "on" and "off"; they should just navigate between the views they want, much like how you use the web. Some processes may be active in the background, based on an activity started from a view, or a configuration, but it is left to the operating system to destroy and recreate views and background processes based on available resources and what else is going on. When you navigate back to a previously used app or bring it back from the list of last used apps, you should not need to know if it was kept "behind the scenes" since you last used it, or if you get a freshly created instance. This is an execution model adopted to the mobile use – the phone could run out of battery at any time, or a phone call could come in forcing whatever else was going on to be removed. You should still be able to return to previous activities and resume what you were doing.

One consequence of this execution model is very important for interface design. In the Android convention, any input/changes the user has provided in a view should be preserved when the view is removed, whether it's because the end-user presses back, another view is taking over, or the phone is about to shut down due to low battery. Therefore, a save button is usually not needed. This differs from a desktop application, where you would expect to lose unsaved changes if you simply close the window. Of course, it is still possible to include an explicit save button, or a cancel to not save changes, but the convention of saving by default when removing a view should be followed for the most part, both because smartphone users will expect it and because of the inherent instability of the mobile device. If the end-user was editing something and then pressed the back button, consider showing a little message to verify that the data was saved. Without going into technical details here, we will just note that this "save on remove" behaviour is not something that comes for free. It requires great care from the app developer to make sure the user always comes back to the app in a consistent state independently of the rather arbitrary choice of the system to keep it in memory or create a new instance.

For the POSEIDON app, we will use a simplified model, where we say that the app can be running in the foreground or running in the background. No one but the app developer needs to know that the distinction between these two and between background and not running at all are not so clear in the Android execution model. We will say it is in the foreground when it is on screen and in focus, meaning it is what the end-user sees and interacts with when using the phone. It is in the background when it is running but not in the foreground. It is in the background when the end-user is not using the phone, turning off the screen to save battery, and when another app is in the foreground. We need some background functionality, such as keeping some sensors active and communicating with servers occasionally. If we need to notify the user when running in the background, the Android convention is to place a notification in the notification bar (this can have sound and/or vibration). Placing an icon in the notification bar to show that the app is running in the background can also be considered. It can indicate app status, for instance with colour. However, we should strive to keep user interaction to a necessary minimum, and showing status might not be necessary.

3.3 Widgets and Styles – The "look-and-feel"

Turning now to the app user interface itself, we will consider options for its style. Widgets are the elements of an interactive GUI (Graphical User Interface), such as buttons, input boxes, check boxes, menus and scrollbars. Android has an extensive GUI framework, with widget types and visual styles for these. One fundamental question is whether to use the native Android look as much as possible, making the POSEIDON prototype look and feel much like other modern Android apps. There are several

advantages to this. For users with prior Android experience, it provides familiarity, while a custom user interface may be harder to use simply for not following the learned conventions. And if we develop a simple, cleaner UI to make the app easier to use, the simplicity can make the app feel both less appealing and stigmatizing to some users. Using the default look also saves us the work of designing an alternative.

On the other hand, there are some usability issues with the default Android look-and-feel. One is that this is not the same on all devices. It has changed with different Android versions, and also manufacturers add style modifications. This makes it difficult to provide detailed instructions that fit all devices. However, the look-and-feel has become more stable and uniform from version 4.0 and above, with the Android theme called Holographic, although this could change again in future. The Holographic theme can also be criticized for its usability. It is a fashionable and somewhat minimalistic design, but has occasional problems of low contrast and abstract widgets that are hard to interpret.

The alternative is to make something with better usability by emphasising simplicity, large elements that are easy to see and interact with and good contrast. In fact we do not have to choose one or the other. We can provide both options and make this an area of personalisation.

3.4 Android User Interface Framework

As we at least want the possibility to use Android's default look, we want to use the Android GUI framework. There is really no reason not to use it, at least as a starting point, because it is a flexible and powerful system, with all the functionality needed to build a user interface. It is built up like most such frameworks, with user interface elements (widgets), layouts for organising the elements on screen and hooks for app code to handle interface events. It cleanly separates the rendering of the widgets (how they look) from their logic, allowing us to change their style. We can both develop new or modified elements, or put a different look on the existing ones. We will examine Android's GUI framework from the adaptability perspective, where there are strengths and weaknesses.

GUI adaptability on Android is closely tied to its system for application resources. We can say that these resources are application content which is not programmed in the source code. The classic example is media, such as images and sound we want to bundle with the app. But, just about everything you see on screen can be defined by resources. All text to be seen by the user should be defined in a text file put in the resources. And the user interfaces themselves – the structures of layouts and elements – can be defined as XML files in the resources. While the creation and organisation of a user interface can also be coded in the Java source code, it is easier to define this structure in an XML file. But the important point of the resource system is that any resource file can have different versions, and there are many different qualifiers which can be used to select between versions. One qualifier is language and region, which is used for text. As long as all text is defined in a resource file, providing support for another language is as easy as providing a translation for the text in the other language. Other than language, the most important quantifiers are those dealing with screen properties, such as size, resolution and orientation. The resource system is important for making the app fit different screens. We will look at this in detail in the next section.

There is a whole system of resource files we can use to define a user interface, because we can define an entity using references to other resource files for some of its properties. This serves two important purposes. One is that it makes it easy to change a property for the developer, having the whole app affected by changing it one place instead of having to search through all the layout files. The other is that the referenced resource file can be qualified separately from the one which references it, so that different GUI properties can adapt to different types of context changes. We will describe a strategy for fulfilling this potential.

Firstly, a layout file defines the layout – the elements and how they are laid out in relation to each other – but with many of the properties defined in other resource files. All text is of course defined in its own resource, so that the app can be adapted to several languages. Sizes, such as margins between elements, should be defined in its own file and referenced from the layout file, both so that it is easy to change for the whole app in one place and so that we can provide variations based on screen properties if we want. For many GUI elements, we should define styles. A style is a collection of properties. Styles are especially important for text elements, where it typically combines font size, colour and other such properties. The font size of a style should in turn reference the size file already mentioned, while colour properties should reference colour variables defined by a theme. A theme is basically a style to be used for the app as a whole. In this strategy we mainly use it to give values to colours and other drawables (a solid colour and an image can often be interchanged). It is possible to have multiple themes and switch between them, thereby changing the look of the whole app with a single property. This system of resources is complex, and there can be difficulty in getting the full picture with the definitions spread throughout a number of files. However, it does make it possible to change certain properties while keeping the rest constant without redundant definitions that would damage application maintainability.

A resource in itself is static, so we should not completely define all user interfaces this way. If we need to adapt a user interface to content and context at runtime, this must be done by code. A simple example is a list of elements – we cannot fully define the interface statically because the number of elements and what they are will vary. Instead we can define the frame interface holding the list as one resource, and the list element as another resource. So for dynamic interfaces we break it down into pieces we can define as resources and then instantiate in code as they are needed to populate the interface.

Layout files are processed by the compilation process, and source code is generated which gives each resource element an integer identifier which can be used in the code. The Android system automatically selects the correct resources based on device configuration. For instance, language is an Android setting which the user can change through the setting interface. Whenever a configuration changes, including the orientation of the screen, the Android system destroys any currently visible apps and launches them again with the correct resources.

This brings us to the limitations of the resource system. One negative is that we as app developers have no control over what resources are selected; this is controlled by the system. We also cannot define our own qualifiers to use the system to do our own adaptations. Another drawback is that the whole resource set is “hard-coded” in the app by the compilation process. It is not possible to introduce resources at run-time. We cannot even provide reference to resources directly from external sources such as a server or a file. If we have two themes, called “theme_a” and “theme_b” in the resource file, these will have integers assigned to them by the compilation process, let’s say 1 and 2. We can provide the value 1 from the external source to select the first theme, and it works, but if later we update the app and compile a new version we have no guarantee that the numbers are assigned the same way, so 1 may now refer to something else.

One of the most requested features for personalisation is text size. This however simple to implement can cause issues in the UI. One problem is inherent in the concept: it becomes harder to create good layouts if the text size can vary. What used to fit on a single screen does not any more when the size increases. Another problem is the implementation. When we define sizes in resources, we do so in a pixel-independent unit which is automatically scaled to different resolutions. We can also use a unit which in addition can be scaled based on a text size property. Although this unit type was there and documented from the very beginning, it took a very long time before it was actually implemented in

Android. Now it is implemented, and it depends on a system setting the user can change in the setting interface. This typically has three possible values: normal, larger and largest. So Android has a solution, but it is device-specific rather than app-specific, and an app is not supposed to tamper with it. We can rely on it, and give the user instructions for how to change this setting (a bit complicated since the setting interface is different on different devices). Alternatively we could make our own solution by putting size definitions into themes, making it a choice of theme. This could be a feasible approach, although theme is usually a question of colours and images, with text size possibly varying with screen size and the system setting.

The resource system is good for what it's made for, but whenever we need other forms of adaptability we must create it ourselves.

3.5 Screen Adaptivity

One thing Android has very good mechanisms for, is adapting the app to different screen configurations. It has been a requirement for Android, since it was designed to run on many different devices. Two key screen properties are pixel density and size. Let's say two screens A and B have the same number of pixels, such as 1280x800. In a sense they are the same from the low-level software perspective – the frame buffer is the same size and we can draw the same amount of pixels on them. But A could be a phone while B is a tablet. B is much larger, while A has a much higher pixel density (the pixels are smaller). In this case we want to fit more content on B instead of just making everything look very big, so we want to use less pixels than on A when drawing the same element. Screen C could be the same size as A but with half the pixel density. Since they are phones of the same size we want to fit the same content, which means C must also use fewer pixels per element.

In an Android device the screen has a pixel density qualifier. We can define all dimension values, such as text sizes and layout margins, in a pixel-independent unit which is scaled automatically based on pixel density. Graphical resources will also be scaled automatically. This way Android adapts an app to different screens with minimal effort from the developer. It is also possible to provide different resources for different screen densities, mainly for images as providing them in the right resolution may give better image quality than relying on the automatic scaling.

Supporting different screen sizes requires a bit more care to do well. If we design the layouts for phones and don't provide alternatives, it will run on tablets but it will typically not utilize the extra space well, thereby looking badly designed. Here we can use the resource qualifiers to provide different layouts for different screen sizes. However, we should strive to minimize the duplication of layout definitions, only providing different versions of what really needs to be different. It's important to make clever layouts that can fit on as wide a range of screen sizes as possible. It's very easy to make a layout that fits exactly to the screen of your phone, and then when the app runs on other phones with slightly different sizes or a different aspect ratio it's all wrong. We risk this problem with layouts designed to fit all its elements on the screen at once, while layouts based on scrolling are generally not a problem. So making a non-scrolling layout we need to take great care, using relative positioning of elements and dynamic gaps between them. We must also keep in mind the possible effects of text size adjustment and multiple language support (the translation may have quite a different number of characters).

Pixel density and screen size are static properties of a device, but keep in mind that the size available to the app can be halved when the keyboard shows up. There is one other important screen property which can change at any time – the orientation. A mobile device can have its screen in the vertical portrait orientation, or in the horizontal landscape orientation. The app needs a strategy for how it will handle this. One option is to only design for one of the orientations, and lock the app to this orientation

so that what's on screen doesn't change if the phone is turned around. If we do let the orientation vary we can choose to provide separate versions of some of the layouts for the two orientations. Typically at least some of the layouts need this to look good and utilize screen space in both orientations, so it is more work to support both orientations. The choice of strategy depends on the type of devices we want the app to run on and the content we want it to display. For phones and smaller tablets, the portrait orientation tends to be the natural way to hold the device, except for when looking at film or photos, while a larger tablet is seldom held this way.

Figure 10 An Android phone in portrait and landscape orientations

For the POSEIDON pilots, the target device is a large phone (5-6 inches screen size). This means that we don't have to have multiple versions of layouts. We can focus on this limited size range and portrait orientation, at least initially, and later provide alternatives if there is a need.

4 Moneyhandling App using the interactive table (CapTap)

The interactive table is placed on a stable surface in front of a monitor. For example it can be placed on a regular coffee table. The Moneyhandling App is running on the local PC, while the interactive table is connected to the local network. How to set up the interactive table is described more in detail in D5.6 - Integrated POSEIDON technology technical documentation.

The UI with the Moneyhandling App comprises of two parts, the UI of the application itself and the interactive table with its overlay. These components will be presented in the following.

4.1 The overlay

The Moneyhandling App features localized CapTap overlays that visualize the spatial distribution of interactive elements of the CapTap. For the Pilot 1 there are three overlays versions, one with Norwegian Krones, English Pound and the Euro.

Figure 11 Overlay for the CapTap – here the Euro version

The overlay for the CapTap is divided in three regions. The upper region represents the buttons that can also be found in the application. The functions connected to the buttons are displayed at the screen and change according to the content and context of the current screen. The lower left region is the money area. Each border limits the sensitive area of the CapTap for that specific printed coin or note. The orange rectangle is the money placement area (the tray).

4.2 Using the Application

Start the application as described in D5.6.

The main menu shown in Figure 12 gives you three options:

Quit: quit the application and return to the Desktop.

Settings: options to change the language and the position of the button bar (Pilot 1)

Scenarios: choose a scenario (Pilot 1 has currently one single exercise scenario)

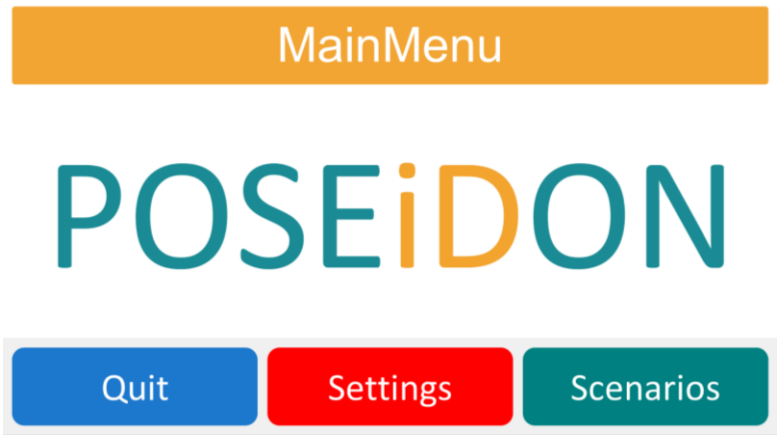


Figure 12 Start screen of the application

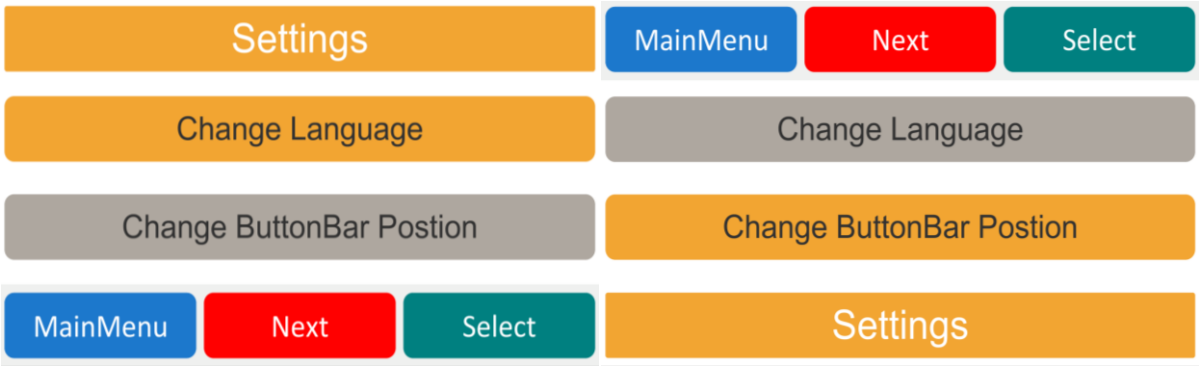


Figure 13 Setting menu: Positioning the buttons - at the bottom (l.) or at the top (r.).

Inside the menus you can navigate either by using the mouse and clicking on the buttons, or by knocking on the corresponding places on the overlay on top of the CapTap. The three buttons (blue, red and green) have analogous meaning on both, the screen and the overlay.

With that in mind you can toggle the settings options for changing the buttons position an entering the language menu by clicking or knocking the red button, which is labelled “next” in this particular menu screen.

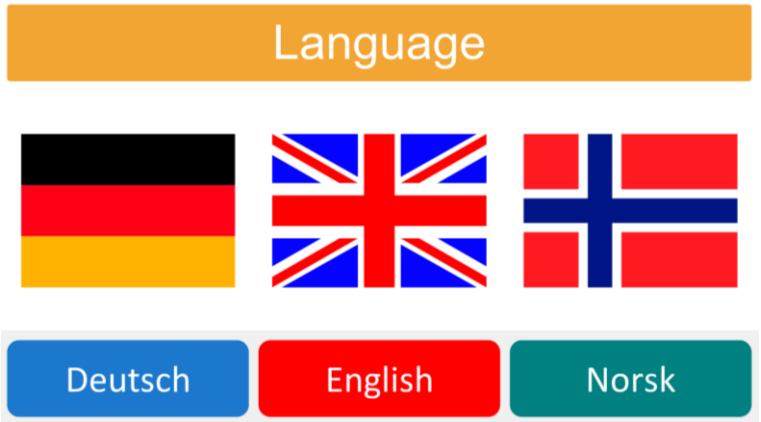


Figure 14 Localization options

From the MainMenu the scenario menu can be reached. Pilot 1 has only one scenario that can be started by clicking or knocking the green button that is labelled “play”. After selection the exercise begins. On the upper left a product is shown. Right next to it the price to pay is displayed. (Compare

Figure 14). Choose the notes and coins you need to pay the product by knocking on the note or coin and then knock on the orange tray field.

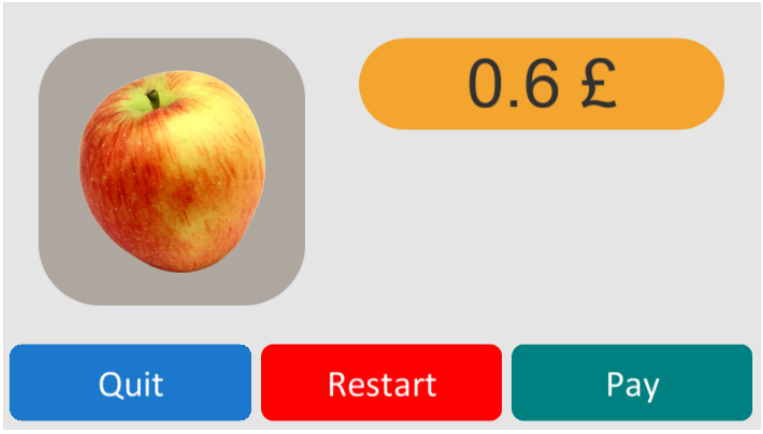


Figure 15 Example screen of a product

If you think you have picked the needed money knock or click on the green button that is labelled "pay". Now you get a visual feedback.

Figure 16 Visual feedback for the chosen amount of money.
(left) exact choice, (middle) overpaid, (right) not enough paid.

4.3 CapTap as static display

The Moneyhandling App is build with Unity3D for Windows x86 / x64 and Mac OS X 10.10 (Yosemite) and uses the CapTap as input (HCI) device on which it heavily relies – in fact the CapTap is mandatory for using the Moneyhandling App because it "extends" the user interface from the virtual representation on the computer screen into the physical world of the user. Without the CapTap a wide area of the applications (virtual) screen-surface is not available.

Figure 17 Developer's view of scene "ScnScenarioOne" with the actual screen on top and the virtual area on the bottom.

The technical point of view is that the CapTap represents a static display for dynamic interaction. Static display means that the content visible on the CapTap is fixed for the timespan of the usage of the application. This static display is visualized by the overlays described in 4.1.1. It is static but customizable: the user can switch between different overlay sheets available for the CapTap and the Moneyhandling App to e.g. customize the localization of the imprints (currency and language). Being a static display with capacitive touch and audio knock (tap) functionalities the content shown on the CapTaps overlays corresponds directly to the Moneyhandling App's logic and mechanics. The interaction process is described in section 4.1.2. The mentioned overlays are currently not recognized by the CapTap per se thus no automatic calibration or customization of the app's logic or settings is performed. They are a pure visual guidance for the user to recognize the spatial distribution of the different logical sensing locations. The customization has to be performed through the Moneyhandling App's settings menu, also described in section 4.1.2.

4.4 Unity3D application

For extending the Moneyhandling App the content of the software projects archive or repository has to be made available on the developer's local machine. Further the most current version of Unity3D (not lower than 5.0.1f1) is necessary to avoid problems with compatibility and feature availability. Pilot 1 has been developed with the professional version of Unity3D for that a purchased license is needed. Nevertheless the first pilot is not using any pro-features by now, so the free versions should work as well until further notice.

Please refer to the Unity3D's user manual (http://docs.unity3d.com/Manual/index.html) for further information on how to work with Unity3D per se.

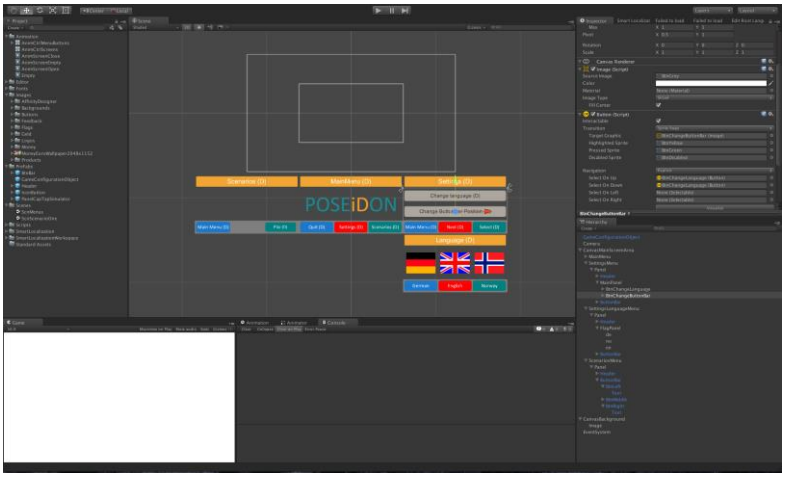


Figure 18 Unit3D project view. In this editor's configuration the project overview pane is on the left hand side.

To begin extending the Moneyhandling App the root folder of the project has to be opened with Unity3D. Unity3D scans the content of the entire folder structure and creates all necessary supplementary data. For repository and data handling convenience this generated data should be dismissed or ignored when versioning any changes because it can affect the work of another developer respectively on another machine. After importing the project into Unity3D the Scenes folder can be found on the projects overview pane. Inside this folder the so-called Scenes are located, which represents a logical entity including e.g. all menus, interactive objects and visible assets. Pilot 1 consists of two scenes that are *ScnMenus* and *ScnScenarioOne*.

- **ScnMenus** contains the entry point for the application where the user can navigate into the settings of the Moneyhandling App or start a new scenario / exercise.
- **ScnScenarioOne** contains the Pilot 1 single exercise and can be started through the Scenarios menu inside the ScnMenus scenario screen.

Either the ScnMenus' as well as the ScnSenarioOne' buttons and screens can receive input from the CapTap's *ButtonBar* shown in the figures above or the computer mouse as commonly known from WIMP systems. The main difference is that the mouse interaction is only meant for assistance purposes and cannot "solve" exercises because they rely on the interaction with the (imprinted) money and the tray on the CapTap. Nevertheless an assisting person can cancel, restart or confirm an action via a click on one of the three context related buttons on the screen's ButtonBar.

All logic interactions and scripts are written in the C# (C Sharp) language but Unity3D also supports JavaScript.

The main further development focus of the Moneyhandling App should be on the layout, the feedback provision and the perceptibility of the shown information. It is a important but not trivial to achieve a relatedness of the content and interaction to real life challenges but having enough abstraction to make a learning app available on a computer. Pilot 1 will gather feedback and experience for further knowledge especially in combination with the CapTap.

4.5 UI rational and guidelines implementation

The Moneyhandling App implements the overall design of the Poseidon UI guidelines. The buttons, the overlays and the App uses the colours specified in the POSEIDON interface design document V1. Some examples are given in Figure 11, Figure 12, and Figure 15 The used font throughout the application is Arial. The flat design paradigm was forced, so no outlines or 3D effects where used. According to the guidelines no decorative elements are used and no disturbing animations are implemented in the

application that disturbs or irritates the user. Only the shift of the screens is visualized with a swift animation to indicate that the screen's content has changed.

Figure 12 shows the screen design, which is divided into three parts: the button area (ButtonBar), the title area and the content area. The position of the button area and title area can be swapped in the preferences. The title area is held in the POSEIDON orange, the buttons are coloured in red, blue and POSEIDON turquoise (green), the product area is in grey and the price area in POSEIDON orange. The visual feedback icons shown in Figure 15 are coloured in the POSEIDON colours scheme as well.

5 Navigation training system using Virtual Reality (VR)

The navigation training system is addressed to two personas: caregiver and person with Down Syndrome. Each user can interact with the system through dedicated interfaces that allow them to accomplish a set of actions.

5.1 Language and role selection

The users are asked to select their language and their role, and based on this selection, the specific screens and functionalities are displayed (Figure 1).

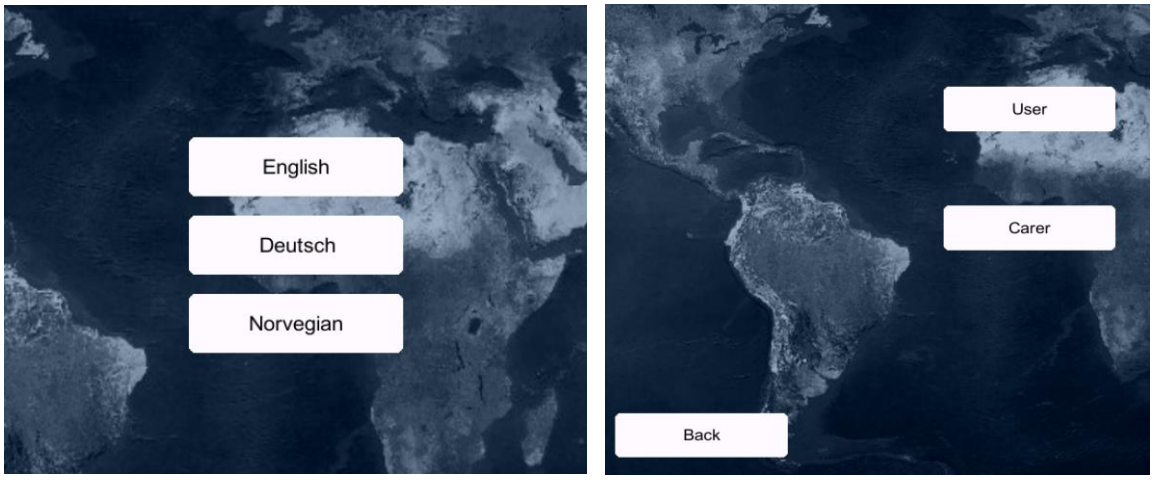


Figure1 Language and Role selection

5.2 Caregiver side

After language and role selection, the caregiver is asked to configure details for routes: points of interest, start and end addresses, customised information about every step (Figure 2).

Once the addresses composing the route are described, the carer configures the details for each step.

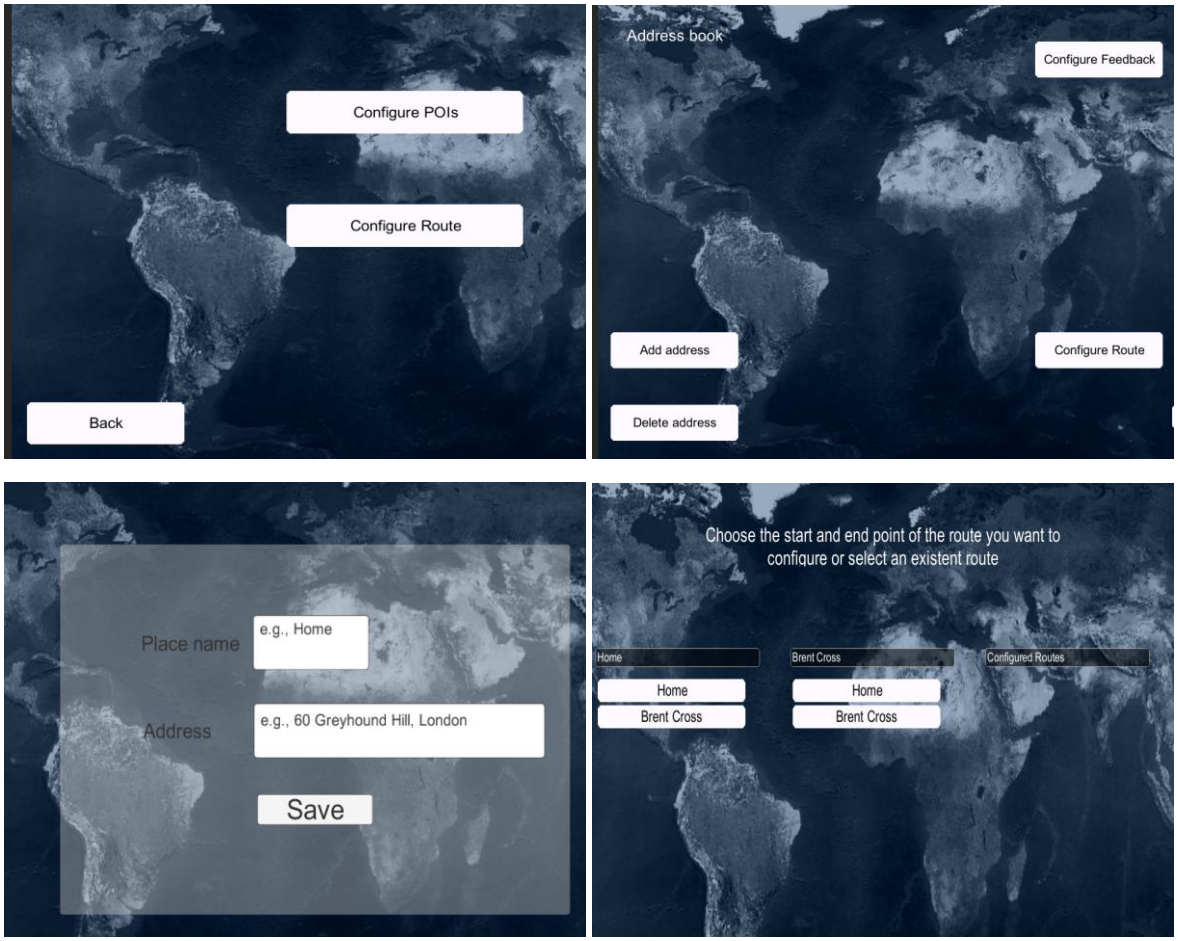


Figure 2 Addresses of interest and Routes configuration

Caregivers can customise routes step by step by orientating the heading in a certain direction, give written indications, choose representative photos and enable or disable types of automatic feedback (maps, audio, Google directions).

Images added as visual feedback can be taken from sources like Google Places or Panoramio, or uploaded from the hard disk (Figure 3).

Figure 2 Addresses of interest and Routes configuration

Appendix 1:
POSEIDON guidelines for developing accessible user interfaces

Edited by Riitta Hellman and Erlend Øverby – Karde AS (13.3.2015)

The purpose of this document is to provide guidance on usability and accessibility for the designers, developers and the ones responsible for testing and evaluating the POSEIDON system and its services. This chapter provides a collection of practical advice of how to design the interaction so that it meets the requirements and capabilities of the target group of POSEIDON. The goal has been to try to keep these guidelines as simple as possible.

This document is based on a number of existing guidelines. The most important of these are:

- Principles of universal design[2]
- Information for all: European standards for making information easy to read and understand[3] (follows this document as separate attachment)
- Guidelines for the development of accessible mobile interfaces by Funka[4] (follows this document as separate attachment)
- Cognitive Accessibility User Research of W3C[5] (Chapter 3.5 follows this document as separate attachment)

1. Principles of universal design

Universal design is more than accessibility. With a focus on universal design we can have a more holistic approach to how we develop the POSEIDON system and services.

The concept of universal design is based on the design of products and environments to be usable by all people. Some of the principles are difficult to apply to ICT-based products and services. A subset of the principles for universal design will, however, apply for the POSEIDON project. In addition, we propose some implications these will have for the POSEIDON system:

Principle 1: Equitable use

The design is useful and marketable to people with diverse abilities.

Principle 2: Flexibility in use

The design accommodates a wide range of individual preferences and abilities.

---

2 http://www.ncsu.edu/project/design-projects/udi/center-for-universal-design/the-principles-of-universal-design/

3 http://www.inclusion-europe.org/images/stories/documents/Project_Pathways1/Information_for_all.pdf

4 http://www.funkanu.com/PageFiles/22075/Guidelines_for_the_development_of_accessible_mobile_interfaces.pdf

5 https://w3c.github.io/coga/user-research/#down-syndrome

POSEIDON recommendations

- All users should be allowed to adjust their preferences for how the system should communicate with them (e.g., size of fonts, contrasts, colours, etc.). Enabling assistive technology such as synthetic speech (i.e., multimodality) is important.
- If a user has tremor, or problems with fine motor skills, and tapping a tablet/smartphone with her fingers could be problematic, the system should be set up so that it is more tolerant for user errors, timing for accepting that a button is pressed etc.

Principle 3: Simple and intuitive use

Use of the design is easy to understand, regardless of the user's experience, knowledge, language skills, or current concentration level.

POSEIDON recommendations

- Only relevant actions should be displayed in the user interface.
- All (most) action buttons should be located at the bottom of the screen, and they should always be visible.
- Help/information should always be located at the bottom-right corner of the screen, or to the right from the specific location where help or additional information is available.
- Search (if relevant) should always be located at the top-right corner of the screen.
- Action buttons should be a combination of icons and text. Exceptions may be made for the user interface, such as lists of choices created by the end-user and where suitable icons are not available, or when uploading of an icon is not provided.
- Focus points of a video service should always be at the centre of the screen (important for video content).

Principle 4: Perceptible information

The design communicates necessary information effectively to the user, regardless of ambient conditions or the user's sensory abilities.

POSEIDON recommendations

- Regardless of context of use, all information on the screen should be easily readable and understandable.
- Textual information presented to the end-user must be meaningful, and take into account the end-users capabilities and cultural frame of understanding.

Principle 5: Tolerance for error

The design minimises hazards and the adverse consequences of accidental or unintended actions.

POSEIDON recommendations

- When the end-user enters wrong data or is using the system in a "non-predicted" way, the system should be "forgiving" and provide guidance to the end-user, and help to recover the error.
- If there are technical problems (disturbances in data communication, internet connection failure etc.), the system should still try to provide meaningful information to the end-users.

Principle 6: Low physical effort

The design can be used efficiently and comfortably and with a minimum of fatigue.

This principle is important when handling tools, opening doors etc. The POSEIDON system that will be operated on tablet PCs and/or smartphones will require very low physical effort.

Principle 7: Size and space for approach and use

Appropriate size and space is provided for approach, reach, manipulation, and use regardless of user's body size, posture, or mobility.

POSEIDON recommendations

- All action buttons both on the web and on tablet PCs and smartphones must be large and easy to click/tap.

2. Design methodology for mock-up development (and beyond)

To achieve the goal of a highly usable system, it is important that we start the interaction design by developing high-level mock-ups (paper prototypes). These could be pictures of the web-design of the POSEIDON family tools, or tablet PC functionality. This will help the design team to figure out how the POSEIDON system should work, what input does it require, and what kinds of output are expected. When the users perform an action "using" the mock-up, they do have some expectations for what will happen.

It is important that the POSEIDON system meets these expectations. The initial design cycle should work to identify the end-users expectations, and at the same time identify what information is needed from a developer's perspective to make the system work, and to provide the expected result.

For the mock-ups (high-fidelity paper prototypes), we have identified a set of design principles that should be followed. These are presented in the next section. Later, these principles should be applied to the development of the HCI of the final POSEIDON system.

When developing user interfaces there are some very central design principles to follow[6]:

Principle 1: Learnability

The user interface should be easy to use from the first time a user interacts with it. There should be no need to learn new functionality or new ways of user interaction. The system should be based on recognition rather than the need to recall previous experiences.

POSEIDON recommendations

- All action buttons should be located at the bottom of the screen, and they should always be visible, common tasks should be located at the same place all the time, such as help, search, information etc.
- Based on the experience of the system, the first time a user uses the application some help and guidance should be provided, and after some times of use the help and guidance should be less intrusive.

[6] Adapted from:
http://www.slideshare.net/OpenRoad/mobile-ui-design-user-centered-design-and-ui-best-practices,
http://www.google.com/about/company/philosophy/,
http://developer.apple.com/library/ios/#documentation/userexperience/conceptual/mobilehig/UEBestPractices/UEBestPractices.html,
http://designfestival.com/5-principles-of-user-centered-interface-design/,
http://www.netmagazine.com/features/10-principles-mobile-interface-design,
http://uxdesign.smashingmagazine.com/2011/07/18/seven-guidelines-for-designing-high-performance-mobile-user-experiences/

Principle 2: Efficiency

The number of steps a user takes to complete a task should be as few as possible. The need for horizontal and vertical scrolling should be kept to a minimum. Wizards should be used to simplify complex interactions. Real world metaphors should be used where applicable. Less is more: most likely we need to leave stuff out.

POSEIDON recommendations

- Main device-orientation is horizontal. (This applies to tablets, but not a carer app for smartphones, where the vertical orientation is preferable for most tasks.)
- All scrolling is vertical – no horizontal scrolling.
- All action buttons are located at the bottom of the screen, and are always visible. Exceptions to this may be buttons to handle the video content (e.g. Start-button in the middle of the screen and the like).
- Only relevant functionality should be visible.
- All information and help texts should be context sensitive, the information and help text should be relevant and to the point.

Principle 3: Error recovery

The system should be designed so that it is hard or even impossible for a user to make mistakes. However, when a user mistake occurs, this should be clearly communicated with information on which actions to take to continue the use of the system.

If there is a system error, this should also be communicated in a clear way, with simple and understandable information to the end-user. All error messages should be useful. The system should provide guidance on how the user should recover from the error.

POSEIDON recommendations

There are three different types of errors that need to be addressed and have a consistent error recovery methodology.

1. User error
2. Device application error
3. Server application error (including error on services invoked from the POSEIDON system)

It should not be necessary to re-enter any data when an error situation appears.

When restarting the app or service, it should launch at the same state as it was when the error occurred.

- The system should be "forgiving" on user errors and provide mechanism for graceful degradation of functionality when an error situation occurs.
- E.g., if the video service is not responding, the system should continue to work (just not display the video), and the "space" used for the video should not be left blank.

Principle 4: Simplicity

Tasks frequently performed should be easy to do, and less common tasks should be possible to do. Unnecessary functionality should be avoided. The layout and design should be uncluttered.

The navigation should be ***narrow and shallow***, providing only necessary functionality. For this, we need to understand profoundly the context of when and where our users will use the system.

POSEIDON recommendations

- There should not be more than maximum three levels of navigation, ideally there should only be one level of navigation in the POSEIDON system.
- The design and design elements should be clean and simple.
- Where applicable, well-known principles and best practice should be applied.

Principle 5: Mapping

What the user expects to happen is what should happen. There should be a mapping between the conceptual model the user has of the system, and how the system actually works.

POSEIDON recommendations

- The conceptual model behind the POSEIDON system should be well documented and described.

Principle 6: Visibility

The most important information should be most visible, and less important information should be less visible. When using a touch interface, no button should be smaller than the user's fingertips plus "necessary margin" for users with tremor or problems with fine motor skills.

POSEIDON recommendations

- Only relevant actions should be displayed.
- All buttons should be of an easily press able size, with clear boundaries.
- Buttons should not be smaller than the size of the thumb.

Principle 7: Feedback

The user should be in control of the interface and not the other way around. The system should provide quick responses. If the response will take some time a progress bar or some other useful information should be provided. Speed and responsiveness are crucial for the user experience.

In today's computing environment one second is an "eternity" to wait for response from the system or application. If a system does not respond within a reasonable time frame, the users will assume there is an error and try again, or press other buttons that will null-out existing action causing confusion and a bad user experience.

POSEIDON recommendations

- If an action takes more than one second, a progress bar or other relevant information should be displayed to the end-user.

Principle 8: Consistency

Identical items, and identical functionality should always be displayed and behave the same way across the entire system/application.

POSEIDON recommendations

- All agreed common user actions should be tested and documented.
- All confirmation, information, help and error "pages" should have the same look-and-feel throughout the system.
- The system should be as predictable as possible.

Principle 9: Satisfaction

The users should enjoy using the POSEIDON system/software. The software should perform its expected tasks well and nothing more. If she would like to perform another task, she would most likely use another application (or system).

POSEIDON recommendations

- Only core-functionality should be provided by the POSEIDON system.
- For secondary functionality we should defer the end-users to other applications that solve those tasks well.

Principle 10: Predictability

When a system follows the principle of predictability, the user would know what to expect from the system: The behaviour is consistent throughout the application/system/service. With a consistent user interface the user will not experience surprises. When a user presses a button or invokes a service, it should be evident for the user what to expect, and it should also be evident how the results will be presented.

To ensure a predictable user experience, it is important to understand the targeted users' expectations and the conceptual models[7] they have for the system they are using. If we design a system based on a different conceptual model than the one of the end-users', the user interaction and how they use the system will never match the anticipations of the developers, and the system will score low on usability and expectations of the users. If the system is designed following the conceptual model of the end-users, we will get a high score on usability, because the behaviour of the system is what the end-users predict. The system and the user interaction follow the users' expectations.

Ideally there should not be any surprises for the end-user when using the system. If something unexpected happens, the methods for solving the unexpected should be predictable and well known by all users.

3. Information for all

All information in the apps and other software products of the POSEIDON project should follow easy-to-read guidelines of Inclusion Europe[8], where appropriate, and possible to implement (i.e. Electronic information and Videos).

4. Funka's guidelines for the development of accessible mobile interfaces

Funka's guideliens include easy-to-understand advice in following categories:

- Choice of solution
- Design
- Layout and design
- Interaction
- Content
- User settings

[7] In this context, a conceptual model is the mental model an end-user has of how the systems works, and the end-users understanding of how different services and functions provided by the system works. An end-user will use the system based on the mental model she has on how the system works.

[8] http://www.inclusion-europe.org/images/stories/documents/Project_Pathways1/Information_for_all.pdf

5. Additional guidelines

This information has been provided by Gill Whitney at Middlesex University.

Mobile Accessibility Standards

A good starting point is the information on Mobile Accessibility[9] available from W3C. There is a lot of content on this page and linked to this page, so it would be advisable to scan to see what is of use.

The Draft BBC Mobile Accessibility Guidelines[10] (and[11]) are possibly slightly easier to read and provide some good (although possibly basic) information on issues such as fonts and layout.

A basic information sheet describing the main issues with respect to mobile accessibility is available from Tim Shelton of AccessibleTech, Inc at[12] There are a number of similar papers available, but this one benefits from being short and easy to read.

Standards to enable access for people with cognitive impairments

W3C are currently involved in working on the area of standards for accessibility for people with cognitive and learning disabilities, more information can be found at[13] This task force has not yet published anything but you can browse their mail archives at[14].

WebAim is a non-profit organisation based at the Center for Persons with Disabilities at Utah State University. They provide web accessibility expertise internationally. Their information on designing for people with cognitive impairments can be found at[15]. The information describes both what the users need and how to meet those needs.

WebAim also provides WAVE, a web accessibility evaluation tool, the section Evaluating Cognitive Web Accessibility can be found at[16]. Their Cognitive Web Accessibility Checklist is probably useful for any system.

ETSI (the European Telecommunications Standards Institute) has a Human Factors Section, they are currently trying to get funding for future research on the needs of people with Cognitive Impairments to support their standardisation work. If they get funding for this work it is likely to result in draft standardisation information and/or a report such as their report on Access to ICT by Young People available from[17].

More information on ETSIs human factors work can be found at[18].

[9] http://www.w3.org/WAI/mobile/
[10] http://www.bbc.co.uk/blogs/internet/posts/Accessibility-Mobile-Apps
[11] http://www.bbc.co.uk/guidelines/futuremedia/accessibility/mobile_access.shtml
[12] http://www.accessibletech.com/dw/resources/mobile_web_access.html
[13] http://www.w3.org/WAI/PF/cognitive-a11y-tf/
[14] http://lists.w3.org/Archives/Public/public-cognitive-a11y-tf/
[15] http://webaim.org/articles/cognitive/
[16] http://wave.webaim.org/cognitive
[17] http://docbox.etsi.org/EC_Files/EC_Files/tr_102133v010101p.pdf
[18] http://portal.etsi.org/TBSiteMap/HF/Summary.aspx

Other Sources of Informal Standards Information

In addition to the formal standards a number of companies provide in-house information on how their ICT products meet the needs of people with cognitive impairments. These include Microsoft, see[19].

7. Terminology and symbols in POSEIDON

When developing an application that shall be used across several countries and regions it is important that the terminology used in the user interfaces are meaningful and accurate with regard to what the end-user expects. It is also important that the terminology used in the user interface is easy to translate, and that the terms used are meaningful also in other languages. The terminology used should not only be meaningful in all languages, we should also thrive for language equivalency, and not merely a translation of terms.

When using symbols (icons) in the user interface to communicate different user actions it is important that the symbols used convey the same meaning across cultural borders – as far as possible. The symbols should mean the same for all people using the system. If some symbols could be interpreted differently in different cultures/languages an alternative symbol should be used.

It is also important that the symbols and terminology are used consistently throughout the applications and systems in POSEIDON.

All texts and icons/symbols should be tested and evaluated by relevant user groups in the partner countries of POSEIDON to ensure that they communicate well and convey the same meaning and expectations across culture and language.

In our user interfaces we will also provide help texts and information texts that will be invoked in different parts of the system. It is important that the help and information texts are contextualised and only provide relevant information for that specific part of the system. In addition all texts should be in all POSEIDON partner languages, and the text provided in all languages should be semantically equivalent.

8. Visual appearance, icons and colour palettes in POSEIDON

The POSEIDON apps and other software products (interactive table, carer's web) must be designed according to best possible practice of accessibility, and with special attention given to the requirements of persons with Down Syndrome.

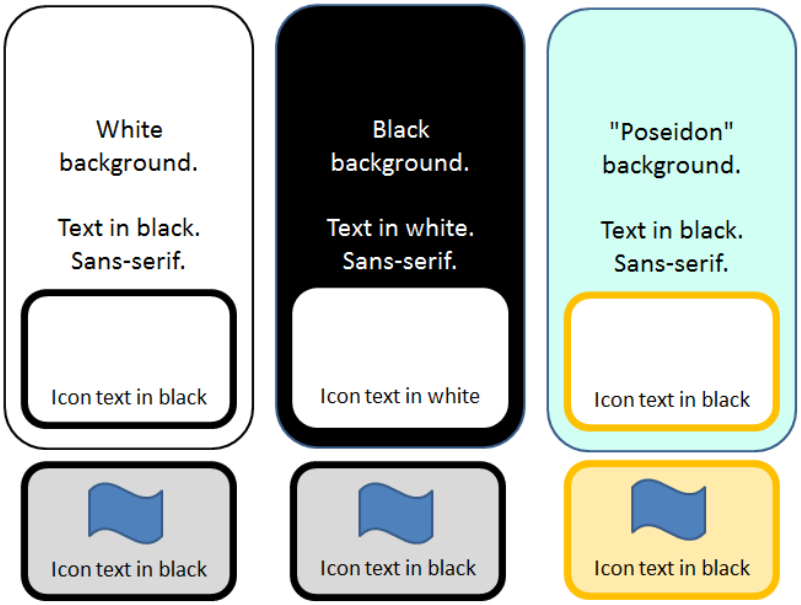
The visual design must be very clear, without decorative elements, disturbing animations, or other design elements which may make the interaction difficult. Simple and clear icons, good contrasts everywhere, etc., are basic requirements. Faded edges should be avoided.

The visual appearance must also follow a "family resemblance" so that the user always understands that she in inside the POSEIDON system. Important elements here are choice of colour palettes and the idea behind the icon design. Icons should when possible be combined with text.

How users interpret icons is a difficult matter to handle, though, and ambiguities are certainly permanent to a certain extent. In order to avoid misunderstandings, it is recommended to choose commonly used icons, such as following examples (not necessarily relevant for POSEIDON as such):

[19] http://www.microsoft.com/enable/guides/language.aspx

For visually impaired users, the POSEIDON system must provide high contrast alternatives, such as black-and white palettes, in addition to the "branded" POSEIDON palette. This is very important for the readability of the text elements.



9. Basic accessibility requirements for web based systems

All web based services and information produced for the end-users of the POSEIDON project should ideally meet the following recommendations towards accessibility and usability:

- Separate content from presentation
- Use HTML5
- Use CSS3 for presentation
- Follow the W3C/WCAG2 guidelines[20]

10. Branding

Apart from the accessibility aspects, it may be important to brand teh POSEIDOM with a logo. This may be useful for end users who need to recognise the apps and other software products. One sketch is drafted below:

Poseidon
Useful apps for special people

[20] http://www.w3.org/TR/WCAG20/#guidelines