Group 6

29/05/2016

Mikkel Koch Jensen - 201370747  
Mathias Schmidt Østergaard - 201371027

F16 – ITSMAP-01

AUH A30 App

Table of contents

[The project 2](#_Toc452167231)

[Requirements 2](#_Toc452167232)

[Short use case descriptions 3](#_Toc452167233)

[Requirements 3](#_Toc452167234)

[Activities and Fragments 3](#_Toc452167235)

[Activities 3](#_Toc452167236)

[Fragments 4](#_Toc452167237)

[Layout 4](#_Toc452167238)

[ListView – adapter and search 5](#_Toc452167239)

[Menu drawer 5](#_Toc452167240)

[Persisting data 6](#_Toc452167241)

[Firebase 6](#_Toc452167242)

[SharedPreferences 7](#_Toc452167243)

[Service and notifications 7](#_Toc452167244)

[Conclusion 8](#_Toc452167245)

[Work plan 9](#_Toc452167246)

# The project



Figure 1

Figure 1 shows the overall interaction with and in the system. More specific the following diagrams on Figure 2 and Figure 3 shows what use cases the apps should be apple to perform.

# 

Figure 2 Figure 3

## Short use case descriptions

Login: If the user is not logged in, he/she will be asked to enter an unique id.

Cooperation agreement: The cooperation agreement is for the parents and therefore the have the possibilities to create and edit these agreements.

Information: both apps should be able to see information about the baby. The difference in the use cases is how you access this information and what information there should be shown.

Create baby: Only the staff app will be able to create a new baby. This will make a new entry which can be subscribed to be the parents. This UC will have to be performed before the Login UC can be initiated.

Edit Baby: After the Create Baby UC has been performed the Edit Baby UC can be initiated. All the information about a specific baby should be edited here.

## Requirements

Sync data between Staff app and Parent app

Parent should be able to save Cooperation agreements

Notify parents about changes

The Parent app should only be logged into once.

# Activities and Fragments

## Activities[[1]](#footnote-1)

Activities are the main ui components of Android and an application usually consists of multiple activities. Each activity can start other activities to perform different actions. When an activity is started it is automatically pushed to the backstack, so the user can navigate back to the initial activity. When a new activity is started the previous activity is stopped, but kept alive. When an activity changes state it is notified through its lifecycle callback methods. Activities must be declared in the manifest.

## Fragments[[2]](#footnote-2)

Fragments are minor ui components typically representing a specific behaviour of the app. A fragment must always be embedded in an Activity and thus the lifecycle of the fragment is directly linked to that of the activity. Multiple fragments can be combined in a single activity in a modular fashion. While the activity is running the fragments can be manipulated independently, either by interacting with them, adding them or removing them from the activity. Fragments can also be added to a backstack managed by the activity, allowing navigation between fragments in that activity. On Figure 4 is an example, of how fragments can be utilized across devices. In the case of this project, the parent app consists of only two activities, but seven fragments, navigation between these are mainly handled through a drawer menu.

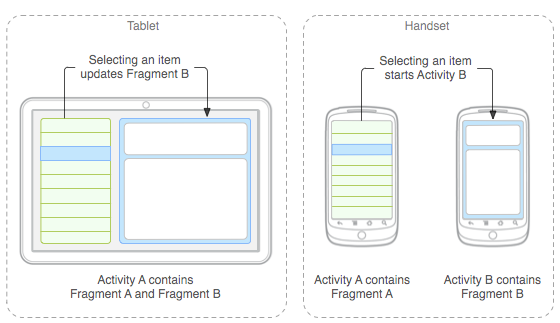


Figure 4

# Layout

Both app project was created over with Android Studio’s “navigation drawer activity”-template. This adds a good portion of scaffolded code to the projects and provides a good template. More specific the it creates a mainActivity and a navigation drawer, which is a menu that can be pulled from the left of the screen to the right to open it. The content has been modified to handle fragments and start activities.

One of the challenges with the layout has been the Datepicker object. In the Android API 21 and above, this object’s style has been changed a lot and appears very different on the different API’s. This was solved by making a separate layout where the Datepicker was used. This layout will only be shown on devices with API 21 and above. The style of the object is made to look like the old style for consistency in the app.

The app has been styled with Dark red colours and baby blue.

## ListView – adapter and search

For the listView there was need for a custom ListView Adapter. The adapter has two main functionalities. The first being the graphical presentation of the list and the second being the filter.

The graphical presentation is defining which values should be assigned where. This is achieved by overriding functions from the Android BaseAdapter class where the getView defines the value on each item on the list. The second functionality, the filter, is where the search happens. It gets the input from the EditText object every time it is changed and performs a check on the items to see if there are any matches. The search checks on strings from the baby model, which is: name, caretaker, birthday and ID.

The only true unique value to search on, is the ID. But in cases where the ID is either forgotten or unknown to the person who is performing the search, it is possible to search on other variables.

## Menu drawer

For both applications the drawer menu is used. The reason for this, is that in this case, it grants the best navigation. A section of how the navigation is achieved can be seen in Code Snippet 1. The navigation is based purely on the identifiers of the different menu items, and a fragment is created based on which identifier was passed to the method. This method also handles whether to add the fragment or replace the currently shown one, this is mainly to handle when the activity is first created and no fragment is show, otherwise the fragment is always replaced. For the full implementation see the source code.

|  |
| --- |
| **public void** updateContentFromIdentifier(**int** identifier, **boolean** replace) {  clearBackstack();  Fragment fragment = **null**;   **switch** (identifier) {  **case** R.id.***nav\_dashboard***:  fragment = BabyDashboardFragment.*newInstance*();  …  }   **if** (fragment != **null**) {  FragmentTransaction transaction = getSupportFragmentManager().beginTransaction();  **if** (replace) {  transaction.setCustomAnimations(R.anim.***slide\_left\_in***, R.anim.***slide\_left\_out***);  transaction.replace(R.id.***content\_container***, fragment, fragment.getTag()).commit();  } **else** {  transaction.add(R.id.***content\_container***, fragment, fragment.getTag()).commit();  }  } } |

Code Snippet 1 A shortened version of the updateContentFromIdentifier() method.

# Persisting data

## Firebase

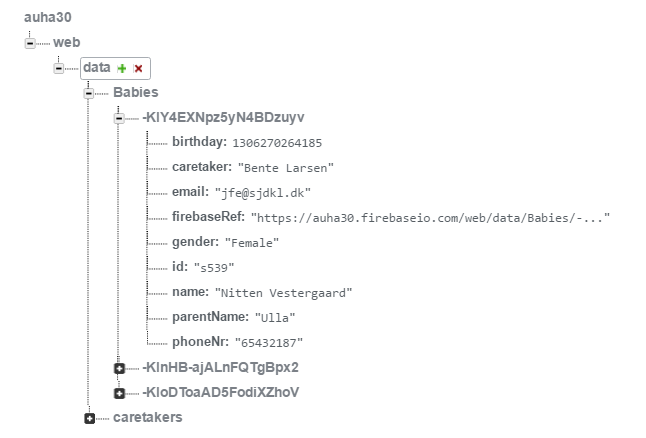
Firebase is the main technology the Staff app uses to persist data and what the Parent app uses to get information about a specific baby. It is a cloud based realtime database which provides an API for developers which is fairly easy to use. It is important to note when working with Firebase that the data persisted with Firebase is persisted in the Json format.[[3]](#footnote-3) The following will be a description on how the technology is used in this project to persist data with.

Figure 4

Firebase is first loaded into the Staff- and Parent-app projects with Gradle. After this the Firebase.setAndroidContext()-function must be called with an Android context before any Firebase related functions can be used. Now everything is set and Firebase can be used.

When creating a Firebase object you need to give it a reference (a Firebase url) to work on. This reference is where the data is stored and the structure is, as earlier mentioned, in a Json format. Because of this format it is displayed in the Firebase dashboard[[4]](#footnote-4) as a tree structure to give the developer an easy overview over the data.

The object with the reference is now ready to either receive or listen on data on the server. On Code Snippet 2 a Firebase object is created and a new baby object is pushed to the database with the push()-function. The push()-function is used to create a new entry in the database to insure that a previous entry is not overwritten. After the push is done. The setValue()-function can change values on whole objects of only childs on the object.

|  |
| --- |
| Firebase firebaseRef = **new** Firebase(**"https://auha30.firebaseio.com/web/data/Babies"**); Firebase newBabyRef = firebaseRef.push(); |

Code Snippet

The eventListeners one of the nicer features of Firebase. There different listeners designed to listen on different events, but the one seen in Code Snippet 3 is from the Parent app and triggers every time some data changes on the reference it got.

When the data has changed, a snapshot of the new data is returned to the function. These snapshot is where all the new data is in, and if the getValue()-function is used with a class description of the model, it can perform the mapping on its own.

|  |
| --- |
| **new** Firebase(extra.getString(***REFBABY***)).addValueEventListener(  **new** ValueEventListener() {  **@Override  public void** onDataChange(DataSnapshot snapshot) {  **thisBaby** = snapshot.getValue(Baby.**class**);  }   **@Override  public void** onCancelled(FirebaseError firebaseError) {  System.***out***.println(**"The read failed: "** + firebaseError.getMessage());  } }); |

Code Snippet 3

Firebase has other features and hosting, and received a major update after the Google I/O 2016, but this project has only been using some of the features prior to the update.

## SharedPreferences

We use Android’s SharedPreferences to persist data that should be within the app. It is only the Parent app which makes use of this feature. The Parent app need to persist a couple of things, such as if the app is logged in to and the cooperation agreement. When the app is started it runs a check on a Boolean saved in the SharedPreferences. Depending on what the variable value is, it will open start the loginActivity or continue to the main dashboard. This value is saved in the SharedPreferences with the PRIVATE\_MODE attribute because the app is the only one who needs the information. Furthermore, is it a practical way of persisting local data in an app where the information persisted always needs to be saved whether or not the app is running.

Another thing that need persisting, as earlier mentioned, is the cooperation agreements. We chose to save this here since it does not need to be synchronized with the Staff app. The cooperation agreement is for the user of the Parent app to see, and should therefore only be saved locally. If it should be able to synchronize with another app, then Firebase would, in this project, be the way to go since it already has a Firebase reference to a baby it could be hooked on to.

# Service and notifications

One of the requirements for the app, was the ability to notify the Parent app about changes to the baby it subscribes on. The notification should be a push-message which will be showed on the Android Status Bar. Since the notifications would mostly be relevant when the app isn’t running, it was decided to make this functionality a service. It is only the Parent app which implements this service, since they are the only once the project aims to notify.

The service is called BabyService and extends the implementations from the Android class Service. Furthermore, it needed a trigger for when it should notify the user. For this the Firebase addChildEventListener was used. This function only needs a reference to a point in the Firebase database to function. Once it is set up there are various of different functions that can be overridden, that each triggers when a different event occurs. In Code Snippet 4 we see two of the functions in the childEventListener. One of them is the onChildAdded, which has an empty implementation since we do not need this trigger, and the other is onChildChanged. The last function will trigger when a child of the Firebase reference is changed, and then call the showNotification()-function.

|  |
| --- |
| firebase.addChildEventListener(**new** ChildEventListener() {  **@Override  public void** onChildAdded(DataSnapshot dataSnapshot, String s) {   }   **@Override  public void** onChildChanged(DataSnapshot dataSnapshot, String s) {  showNotification(getString(R.string.***serviceUpdate***));  }  […] |

Code Snippet 4

To show a notification the NotificationCompat.Builder was used. This builder needs an Intent to know what it should start when it is clicked, and furthermore the message to be displayed is built with and icon and text. This can be seen in Code Snippet 5

|  |
| --- |
| NotificationCompat.Builder builder = **new** NotificationCompat.Builder(**this**); builder.setSmallIcon(R.mipmap.***ic\_launcher***);  Intent intent = **new** Intent(**this**, MainActivity.**class**);  PendingIntent pendingIntent = PendingIntent.*getActivity*(**this**, 0, intent, 0); builder.setContentIntent(pendingIntent); builder.setLargeIcon(BitmapFactory.*decodeResource*(getResources(), R.mipmap.***ic\_launcher***)); builder.setContentTitle(**"AUH A30 Parent App"**); builder.setContentText(msg); builder.setAutoCancel(**true**); NotificationManager notificationManager = (NotificationManager) getSystemService(***NOTIFICATION\_SERVICE***); notificationManager.notify(1, builder.build()); |

Code Snippet 5

Lifecycle of the service

The service is set to start when the app calls the overridden onStop()-function from the mainActivitity. If we take a couple of steps back into the lifecycle of an activity, there is no reason why the service should be running here. This is because the service should not notify the user about changes when the app is running and the user can the current information. For example, it could be considered unnecessary to get information about your baby, while you were looking at the information about the baby.

The service is set stop when the when the mainActivity calls the overridden onResume()-function. This is again to stop notification when the app is active.

# Conclusion

The project uses:

activities, fragments and intents to control the layout and functions like opening the phone app if a phone number is clicked from the contact section in the Parent app. It Persists data with sharedpreferences locally on the device and externally with Firebase. Firebase has had a big impact on the project with data synchronization. It has a background service in the Parent app to notify the user of changes in data. Resource externalization through the app. Action bar and a menu drawers are modified to fit this project. And much more has used in this project. Overall this project is making use of a lot of what Android has to offer.

Sadly, the project has had little to no interaction from the nurses who pitched the idea. This mean that it hasn’t been evaluated by anyone who would be using the app.

# Work plan

While we have tried to distribute the work in a manner that allowed both of us to work with all aspects of the application, like firebase, fragments and the likes. Some of the smaller tasks however we have split between us, and as such we have worked on a few things individually, which are as follows:

* **Mikkel**
  + Animations
  + Drawer menu navigation in the parent application
  + Edit and discharge baby functionality
  + Styling
* **Mathias**
  + Background service
  + Shared preference persisting
  + Login
  + Displaying data from firebase

1. https://developer.android.com/guide/components/activities.html [↑](#footnote-ref-1)
2. https://developer.android.com/guide/components/fragments.html [↑](#footnote-ref-2)
3. https://firebase.google.com/docs/android/setup#add\_firebase\_to\_your\_app [↑](#footnote-ref-3)
4. The Firebase dashboard is where the database is administrated. [↑](#footnote-ref-4)