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](#_Toc452152437)

[Requirements 2](#_Toc452152438)

[Activities and Fragments 2](#_Toc452152439)

[Layout 2](#_Toc452152440)

[ListView – adapter and search 3](#_Toc452152441)

[Menu drawer 3](#_Toc452152442)

[Persisting data 3](#_Toc452152443)

[Firebase 3](#_Toc452152444)

[SharedPreferences 3](#_Toc452152445)

[Service and notifications 4](#_Toc452152446)

[Conclusion 5](#_Toc452152447)

# The project

## 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

## 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 1 is an example, of how fragments can be utilized across devices.

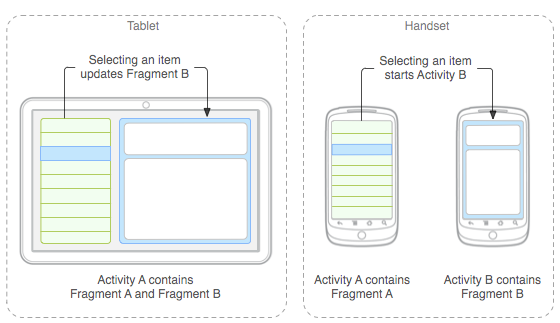


Figure 1

# Layout

asd

## ListView – adapter and search

## 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

Brief firebase description

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.

Firebase is first loaded into the Staff- and Parent-app projects with Gradle. After this the Firebase.setAndroidContext must be called before any Firebase related functions can be used. And

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

EventListener

|  |
| --- |
| **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());  } }); |

## 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 1 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 2

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 build with and icon and text. This can be seen in Code Snippet 2

|  |
| --- |
| 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 3

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

# 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)