2nd Mandatory Assignment

Travel Card App

Casper Riboe

Table Of Contents

Design Choices

User Interface

The user interface consists of two main views: The *Setup Page* and the *ViewPager* "Page" consisting of the *Travel Page* and the *Profile Page*. This is illustrated below by *Figure 1*.

Firstly, the *Travel Page* consists of limited elements and functionality, in order to keep focus on the key functionality of view and no distract the user with less important features. The *Travel Page* provides a quick overview of what point is nearest (e.g. where is the user located) and can he/she check in/out.
The *Profile Page* contains other relevant info, such as the information on the user and past trips. From here, more functionality is added such as adding a new window to add balance to the user's current balance and to access a detailed window with information regarding a past trip.
Besides the core functionality, a setup window exists outside *ViewPager* and it's *Travel-* & *Profile Page.* The *Setup Page*'s use is to setup the information needed by the app and return to main flow through the *ViewPager*, once done.

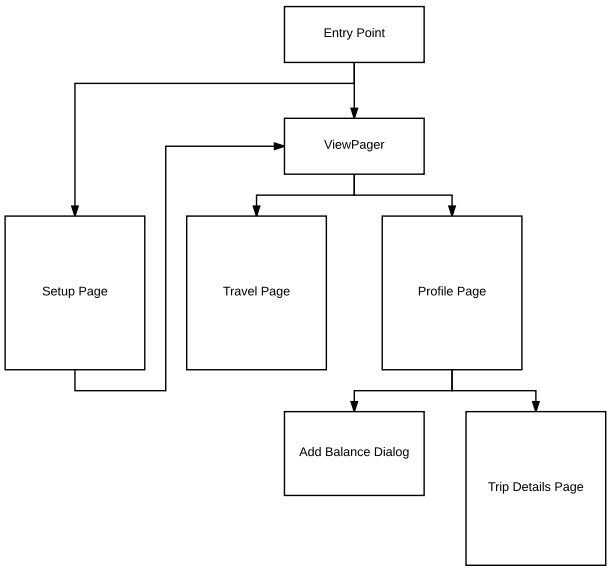


Figure 1 – A general tree structure is used in order to separate user flow and remain focus on relevant features & functionality.

To use a tabbed layout was quite straightforward given that having only two views would fit this structure perfectly. Having the views tugged away in a navigation drawer would seem irrelevant given that only two views exist and a drawer's primary use is to hide a lot of functionality and entry points.

As seen on *figure 1* a simple tree structure is used to hide away functionality like the navigation drawer and keep the other user flow separated. A navigation drawer could perhaps be used to display the few functionality of the *Profile Page* such as displaying the name, current balance and a few buttons to navigate to the *Travel Page* and a *Trips Page*. Having this layout seems more user friendly to me so the user will not overlook the information displayed in the navigation drawer, given the limited data.

Database Structure

The data is stored locally using Realm which makes it incredibly convenient to pack and unpack data without having to explicitly write code for ever change in the models' structures.

To save and retrieve data a Manager is used, called *RealmManger*. All data used by *RealmManager* exists in the *Models* package that consists of the models *User, Beacon* & *Trip*. The Realm database therefore has 3 tables a *User-, Beacon-* & *Trip* table.

Given that only one user exists in the app, this The *RealmManager* takes care of retrieving the correct user object and model it accordingly to methods called throughout the app.
Each *User* has a list of *Trips* and each *Trip* has a list of *Beacons* as illustrated by *Figure 2*. That means all referencing is done automatically by Realm and no explicit primary keys are needed.

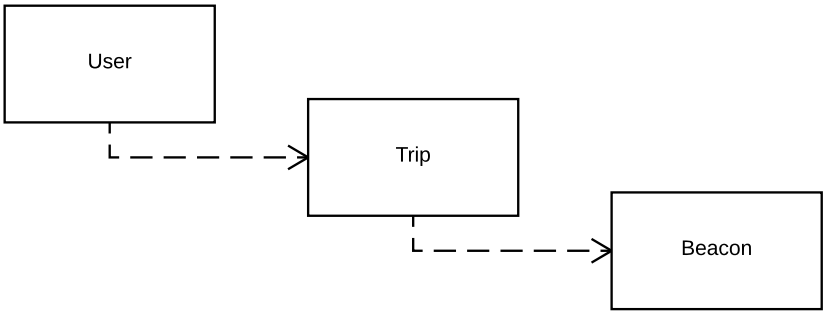
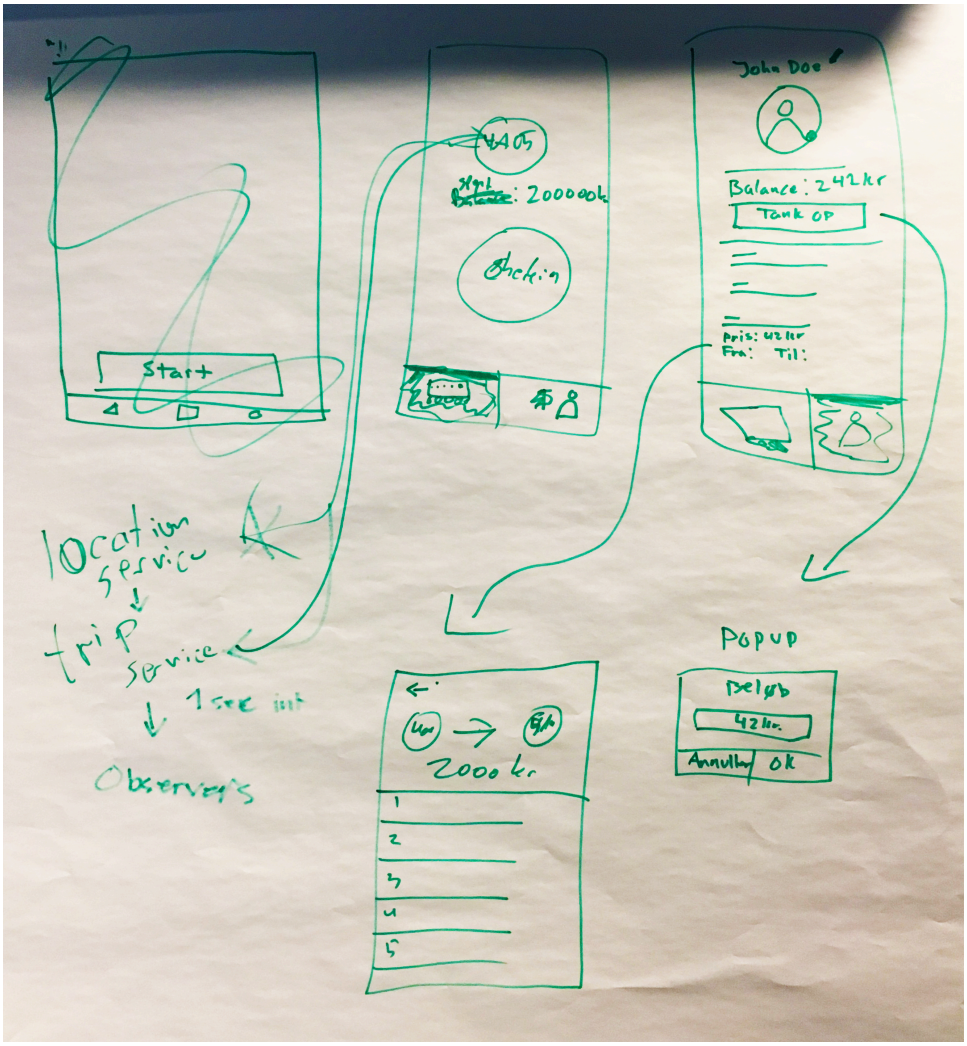


Figure 2 – Each User consists of Trips which consists of Beacons.

Program Sketch

At the start of the project a minor sketch was laid out to propose the layout and interface flow. This is illustrated in *figure 3.*
Minor design choices were reconsidered and changed to similar approaches tweaked in little ways. For example, the tab bar were first portrayed in the bottom but seemed more convenient to put at the top, due to guidelines for interfaces on Android.

A fixed price for a trip was chosen over a calculated price in order to bring down power consumption from constantly checking for new beacons in the background, adding them to the list of "passed by" beacons. This was also partially due to possibilities of errors n detecting the correct nearest beacon. For instance a beacon on the 1st floor could be detected on the 5th floor, where the user could still be on the 5th, but the app wouldn't know whether or not the user's position had actually changed the 1st floor. This design choice is also seen on the view in the lower left region, illustrating a list of passed beacons.



Key Functions & Data Flow

The app has 2 key main functions, which is to check in/-out at a point (*Beacon*) and add money to the current user's balance.
Besides checking in/out and adding money to the user's balance, the user can see past trips and monitor nearby beacons.

Starting a trip by checking in sets a state traveling to active. A new *Trip* is created and the starting point is added to that trip. The user can then travel as he/she pleases and end the trip by checking out at another nearby point. This point is then added to the on-going *Trip*, which is passed on to be archived to the user through the *RealmManager.* The manager then begins a new transaction to the local database and adds the trip to the current *User* object. When done, the manager commits the changes, which is then permitted in the database.

Because the way Realm operates, no fields are set for the *RealmManager*. Realm provides a copy-free access to the data it stores in which the data you retrieve are actually pointers rather than a new instance of the data. This also means that it might be difficult for classes to work together with a data type set by Realm. Therefore view explicitly updates data on the screen when a change is done or when their view becomes active.

When data needs to change, it is passed through the *RealmManager*. Because only one user exists, other classes don't need to know about the *User* object used for this user. This means that objects can simply pass data- and call methods to change data through the manager. Tapping "Add Balance" on the *Profile Page* and entering an amount in the presented dialog will pass on the entered amount the the *RealmManager* which then retrieves the current balance, add the amount passed to the method and saves the new balance to the current user. In this way no other classes will ever need to know about the current user.

Given that the *RealmManager* is a singleton it *could* have fields with pointers to the data in the database that other classes could observe on. In hindsight, this would probably be the best approach, but the current one was chosen due to other design choices set beforehand that needed to change.

User Guide

Opening The App For The First Time

When opening the app for the first time, you will be presented with a quick setup. Here you will have to enter your name in order to begin using the app. Once a name is entered, hit the “Enter” button below the text field and you will be sent to the main app, ready to begin travelling.

Start A Travel

On the *Travel Page* you will find the nearest check in/-out point to you. Provided that you have sufficient balance added to your profile, you will be able to start travelling, by tapping the ”Check In” button. Once a travel is started you can start to reach your end point. Given a point is near you, you can checkout. Tapping the same button as you tapped when starting your travel that now read “Check Out” does this. Once tapped, the trip will be recorded and money will be withdrawn from your balance.

See Your Past Travels

In the top of your screen is a tab bar. Here is a tab called ”Travel” which is the *Travel Page* and ”Profile” which is your *Profile Page.* Tap on the ”Profile” tab and you will be brought to the *Profile Page*. Here you can see the current user’s name right underneath the tab bar. Under the name are your current balance and a button to add more balance. At the bottom is a list of past travels, provided you have had any. Tapping on a travel will bring you to a new window where the trip details are displayed. You can get back by tapping on the back button in the lower left corner.

Add Balance To Your Profile

When you navigate to your *Profile Page* you can see your current balance in the upper region of the screen. Right underneath is a button that reads “Add Balance”. Tapping on the button brings up a dialog where you enter the desired amount. Once entered, tap on the “Accept” button to add the amount to your balance. If you would like to cancel, just hit the “Cancel” button.

Testing

Subject	Tests	Result
1. Welcome Screen (ProfileSetupActivity)	1.1 Opens the first time the app is opened 1.2 Opens if a user is missing 1.3 Opens if a users name is invalid 1.4 Proceed only if the name is ”valid” (There is something)	• Worked • Worked • Worked • Worked
2. Travel Page (TravelFragment)	2.1 Displays the nearest Beacon if its valid to ITUs naming conventions 2.2 Start a travel if a beacon is near and considered valid 2.3 End a travel if a beacon is near and considered valid 2.4 When ending a travel money if withdrawn 2.5 Checkin/-out button changes title based on travel status 2.6 Display an error if there are not enough balance 2.8 Display and error if trying to checkin/-out if no valid beacon is near	• Worked • Worked • Worked • Worked • Worked • Worked • Worked
3. Profile Page (ProfileFragment)	3.1 Displays the users name 3.2 Displays the current balance 3.3 Updates balance when value is changed by a trip 3.4 Updates list of trips when the list changes 3.5 Pressing the ”Add Balance” button displays a dialog to add money 3.6 Tapping on a trip displays an activity containing the trip information	• Worked • Worked • Worked • Worked • Worked • Worked, displays the same information
4. Trip Details (TripDetailsFragment)	4.1 Display the location of the start beacon from the selected trip 4.2 Display the location of the end beacon from the selected trip 4.3 Display the price from the selected trip 4.4 Return to the previous activity if something goes wrong when loading	• Worked • Worked • Worked • Worked
5. Add Balance Dialog (From ProfileFragment)	1.5 Correctly add the entered amount to the current balance	• Worked