Aarhus universitet

Smartphone Applikationer

6. Semester

Fridge-i-nator

Gruppemedlemmer:	*Studienr:*
Daniel Tøttrup (Elektro)	201509520
Stinus Skovgaard (Elektro)	201401682
Mathias Friis (Elektro)	201505665
Philip Schmidt (Elektro)	201506381

AARHUS
UNIVERSITY
SCHOOL OF ENGINEERING

16. maj 2018

Indhold

Figurer

1 App vision

Fridge-i-nator is an app that works together with you and your fridge. It will make it easier for you to make a shopping list and keep track of the items in your fridge especially if your share fridge with others. You can create an essential item list, which is items you never want to run out of. That way if the item comes under a minimum quantity Fridge-i-nator will automatically add that item to your shopping list. Fridge-i-nator will minimize the risk of you forgetting to buy items you need, and make sure you never have to check your fridge before going to the supermarket.

2 Personal visions

2.1 Mathias Friss

While building the app, i wish to become better at designing system architecture, with well defined segregation of layers. I wish to create an app with clear seperation between UI and back-end. Furthermore i wish to learn more about using fragments in android.

2.2 Stinus Skovgaard

I wish to become a firebase god!

2.3 Philip Schmidt

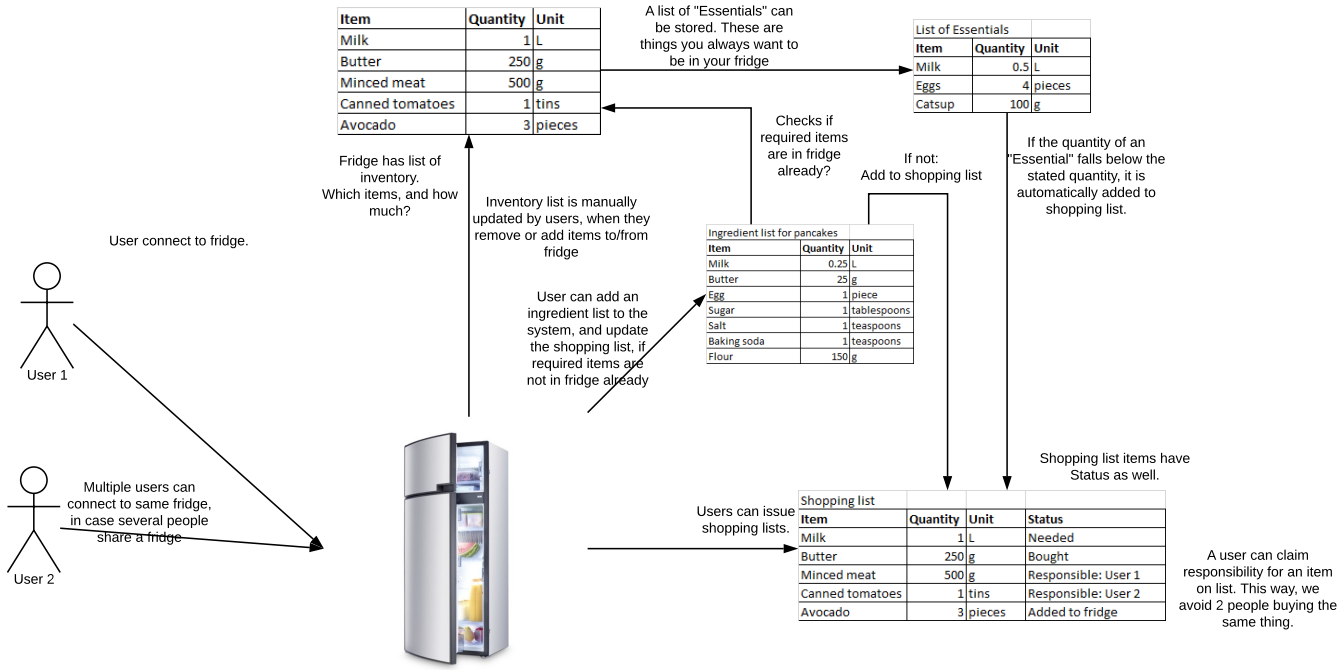
In contrary to no former Android Development and Java experience , i'm striving to become a more experienced developer within the Android scene. This includes a goal of being able to create a great UI experience to the users, managing API calls, maintenance and implementation of databases amongst a lot of other possibilities with the Android Studio tool.

2.4 Daniel Tøttrup

Through designing and implementing the fridge-i-nator in this app-project, I want to improve my app design skill. I also want to get familiar with firebase, which we intend to use in this project, and even more familiar with Android studio and the opportunities that comes with this tool. All in all, I want to be a better app developer.

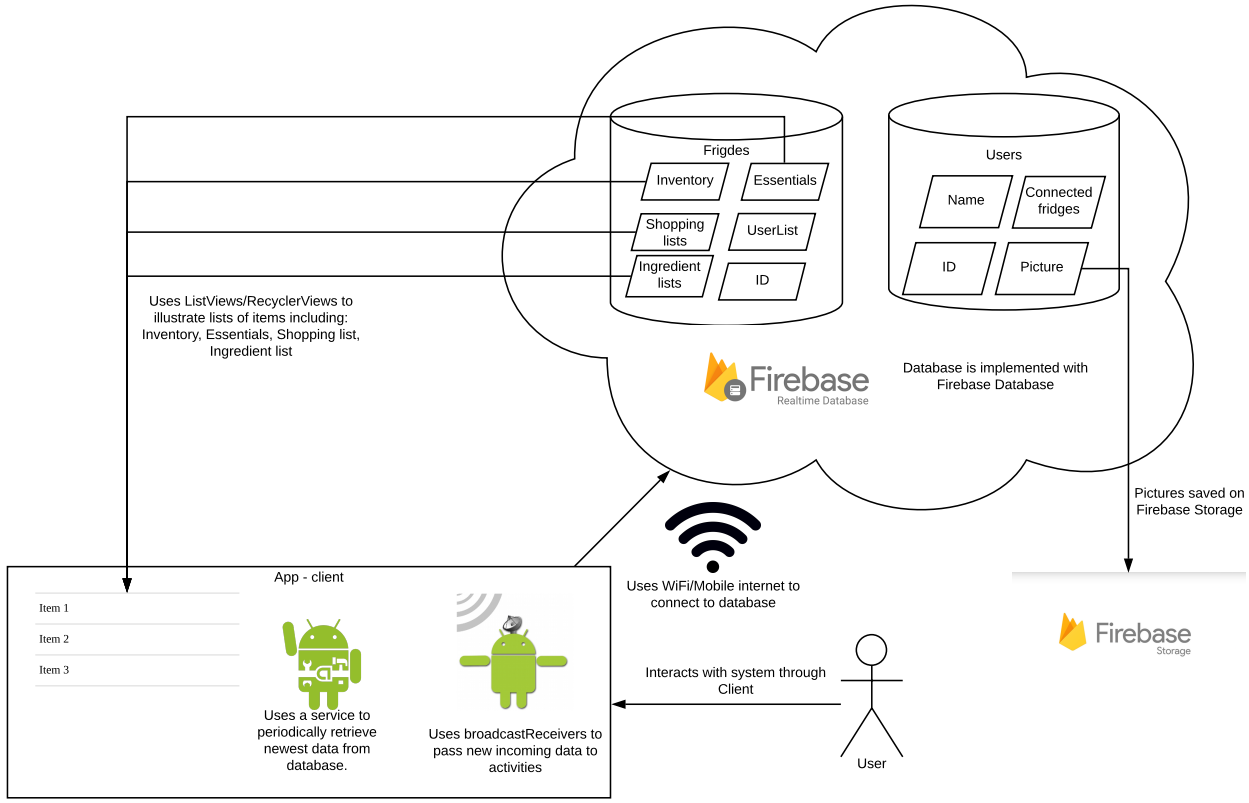
3 Rich picture

Below is a rich picture which shows how the Fridge-i-nator app works.



Figur 1: Rich picture of Fridge-i-nator

4 Early design



Figur 2: Early design diagram of Fridge-i-nator

5 User stories

The below user stories

CONNECTION

As a user, I can connect to a create a new fridge, so that i can acces connect to it.

As a user, I can connect to a fridge, so that I can access information about it.

As a user, I can add other people to a fridge I am connected to, so that they can access information about it.

As a user, I can leave a fridge, so that I cannot acces information about it anymore.

INVENTORY As a user, I can acces an inventory list of items which the fridges contain, so all users connected to the fridge can keep track of what is in the fridge.

As a user, I can add items to the inventory list, so that all connected users may know that the item is in the fridge.

SHOPPING LIST As a user, I can acces a shopping list of items, so all users connected to the fridge can keep track of what needs to be bought for the fridge.

As a user, I can add items to the shopping list, so that all connected users may know that the item needs to be bought.

ESSENTIALS LIST As a user, I can acces a list of essential items with a minimum quantity of each item, so when the quantity of at specific item comes below the minimum quantity fridge-i-nator automatically adds the item to the shopping list.

As a user, I can add items to the lsit of essential items, so that all the system automatically will add the item to the shopping list, if the quantity of the item falls below a specifiec quantity.

INGREDIENT LIST As a user, I can add an ingredient list of items, so that the system will add missing items to the shopping list.

USING THE SHOPPIN/INVENTORY LIST As a user, I can claim responsibility of an item on the shopping list, so that other useres will be notified about who has taken responsibility of the given item to avoid double shopping.

As a user, I can move an item from the shopping list to the inventory list, so that all users connected to the fridge will know that the item has been bought.

5.1 Reflection on the user stories

If we look at all the user stories we made before we started the implementation of the app. These requirements are almost all met, except a couple of them. One of them was: "As a user, I can claim responsibility of an item on the shopping list, so that other useres will be notified about who has taken responsibility of the given item to avoid double shopping." This is no longer true. Instead we decided it would be a better user experience to claim responsibility of a whole shopping list. The reason for this is that the user would waist a lot of time, claiming responsibility of each individual items he/she wants to have responsibility of buying. If the user didn't buy all the items on the shopping list, he/she could just move the bought items to the inventory and another user could overtake the responsibility of that shopping list. Another feature we have implemented to improve the user experience, is if the user has bought all items in a shopping list it's now possible to move all items to the inventory at once, instead of one at the time.

Another thing we didn't think about before we started to implement the app, was whether we wanted the app to be able to rotate. But after we had implemented the landscape mode and tried it out, it just felt like a really bad user experience. Because it gave nothing extra to the user on the contrary it just made the experience worse. It's because our UI manly consist of listviews and when you then turn your phone to landscape mode, its only possible to see three items in the listview at the time. We therefore decided to lock rotation

to improve the user experience.

6 Requirements

If we look at all the user stories we made before we started the implementation of the app. These requirements are almost all met, except a couple of them. One of them was: "As a user, I can claim responsibility of an item on the shopping list, so that other useres will be notified about who has taken responsibility of the given item to avoid double shopping." This is no longer true. Instead we decided it would be a better user experience to claim responsibility of a whole shopping list. The reason for this is that the user would waist a lot of time, claiming responsibility of each individual items he/she wants to have responsibility of buying. If the user didn't buy all the items on the shopping list, he/she could just move the bought items to the inventory and another user could overtake the responsibility of that shopping list. Another feature we have implemented to improve the user experience, is if the user has bought all items in a shopping list it's now possible to move all items to the inventory at once, instead of one at the time.

Another thing we didn't think about before we started to implement the app, was whether we wanted the app to be able to rotate. But after we had implemented the landscape mode and tried it out, it just felt like a really bad user experience. Because it gave nothing extra to the user on the contrary it just made the experience worse. It's because our UI manly consist of listviews and when you then turn your phone to landscape mode, its only possible to see three items in the listview at the time. We therefore decided to lock rotation to improve the user experience.

7 Assumptions for and explanations to the chosen design

GUI
Throughout the start of the designing process of the app, there were several opinions on how the GUI could be designed, in order to achieve the best user experience. In accordance to achieving the best user experience, you would want the user to feel that the app feeling and usage is intuitive, and therefore shouldn't raise questions about how to use it, after it being installed on the user android device.

That concluded in us knowing that in order to achieve a great user experience, we needed to make the app as simple as possible in terms of a GUI. In conclusion to the decision on focusing on a user friendly app experience, the group decided to make a GUI mockup, in order to agree on the most efficient way to build the app in order for it to make sense for the end-users. The GUI-mockup can be seen in appendix (REFERER TIL BILAG HER).

At first time opening the app, the user will be presented with a login screen. Here the user will be asked to enter the email address. Afterwards the app will check in the Firestore

database, whether the email address has been used before or this is the first time. If the email address is used for the first time, the app asks the user to select a password, which will be used in the future to log into the users account. If used before, the app simply asks for the password, unless you have setup Google Play to remember password – then it automatically log-in for you.

Then the user lands on the OverviewActivity, where the user will be presented a listview of fridges he/her previously has been subscribed to. Obviously that list will be empty, in case this is the first login. The user has the possibility to either create a new fridge, subscribe to an existing fridge, or to either delete or share a subscribed fridge in the listview. In addition, the user can choose to press a fridge on the listview to go to the fragmentview, DetailsActivity, to see details for the choosen fridge.

The fragment view concludes 4 different tabs, where each of the holds separate information presented in a listview. There is an inventory list, which concludes the things that the user currently has on shelves/in fridge. The essentials-list, where the user can define which items they never want to go below a given threshold. If that threshold is reached, it will automatically be added to the shoppinglist. The user can access each different shoppinglist and see what items and quantity they hold. Ingredientlists is the last tab, and it holds a list of known recipes, which the user can press and see what is needed to make the specific recipe. The possibility to add further items to the recipe and add the total list of items needed for the recipe to the shoppinglist. As stated, the above, has been the outline of the GUI designed and thereafter been carried out within the development-process of the app.

Fragments

Due to the group wanting a great user experience troughout the app, there was a need for us to display the relevant data to the user, in the best possible way. We concluded that one of the best solutions would be in a fragmented view, which listed 4 tabs: Inventory, Essentials, Shoppinglists and IngredientsList. Thereby the user can quickly access the data and switch beetween the list, depending on what the user intend to do, whether he/she wants to go to shoppinglists to add all items to inventory, or seemleesly check an ingredientslist.

The fragmented views normally comes as one single class. Since the group felt there was an slight chance there could become to much chaos in the loop of creating and switching from tab to tab inside one single class (DetailsActivity), we therefor decided to split up each tab into their own single class.

Then we implemented the functionality in the detailsActivity so they would display the correct tabs and refer to the tabs individual class. In each class it extends Fragment and a onCreateView class, which inflates the correct tab .xml file, in order to show the correct UI information to the user.

The outline and outcome of this decision have maked the development of the fragtment

activities easier than normal, without compromising the users user interface experience.

Data Access Layer As database for this project, we are using firestore by firebase. This was chosen because it offers an easy-to-use database solution which is highly integrable with Android. It provides the capability to add "EventListeners" to documents and collections in the database, which will trigger events in the code. This provides us with a sort of observer pattern, where the individual devices subscribed to the database can receive new data, when it is changed in the database.

The Data Access Layer is split into 2 parts: **fireStoreCommunicator**: handles the lowest level of communication with the database. This includes adding new lists, adding items to list, adding users to database, subscribing to fridges, receiving data, making lists from this data, etc...

ServiceUpdater: this serves as the Programming Interface throughout the app. The service also holds a local copy of all fridges the user has subscribed to. All actions done in the activities/fragments goes through the service. For instance, when adding an item to a list in an activity, the service checks if an item with matching name is already on the list. If so, it increments the quantity of that item in the database, instead of adding a duplicate item. From the service, we send out a broadcast whenever the service has retrieved new data from the database. This way, the activities/fragments can retrieve the newest data from the service, and thus stay in sync with the database. The reason for choosing the approach where the service holds a local list of the data in the database is because we had experience with this approach from the Assignment 2.

One could argue that it is bad programming practice to have such a huge amount of logic in these 2 classes, and that we instead should have divided the functionality into several lesser classes. Had we had more time, a rework of the architecture would be a topic of high priority.

Authentication For user authentication we used firebaseUI, which provides an easy to use solution for authentification with a variety of different social media integration as well. We chose to keep it simple and use authentification via email and password.

8 Bugs and known problems