LeFoodie

Theme Project

SMAP

Group 20

20112407, Eduardo L. M. Knudsen - IKT

201305945, Christoffer D. Junge - IKT

201600057, Jonas J. Yde - E

201801505, Cuong C. Nguyen - Exchange

Aarhus University School of Engineering,
Inge Lehmanns Gade 10
DK-8000 Aarhus C

December 9, 2018

Contents

Vision | 1

1.1 | App Vision

The app vision is to create a tool which allows the user(s) to input and save all available cooking ingredients in their home, and to use these ingredients to help find a recipe for the night. To limit the scope of the project, the app will be initialized with a local database containing a list of normal everyday ingredients. The local database will hold ingredients with the following properties; the amount currently available as well as the expiration date of said ingredient.

When using the application the user will have the options to log-in or skip log-in. When logged in, the application will load information from an external database. When skipping the log-in, the app will still have all of its functionality, but all of the data will be saved locally.

When logged in, it will be possible to share/update ingredient lists between users. This functionality will be specified at targeting families as well as colleague students sharing a common cupboard/fridge.

Ingredients currently in the database will make a basis for a type ahead functionality, to enable the user to easily insert ingredients into the database without the need to fully write everything. Inserting an ingredient to the database which already exists will update it's amount.

A background service will run in the background which will periodically delete items from the database which have passed their expiration date. Deleted items wont be removed from the database, instead they will have their corresponding amount set to 0.

The front page will contain a list view of selectable ingredients, allowing the user to choose a maximum of 5. Selecting ingredients will enable a "Search" button which will allow the user to search for recipes containing the chosen ingredients

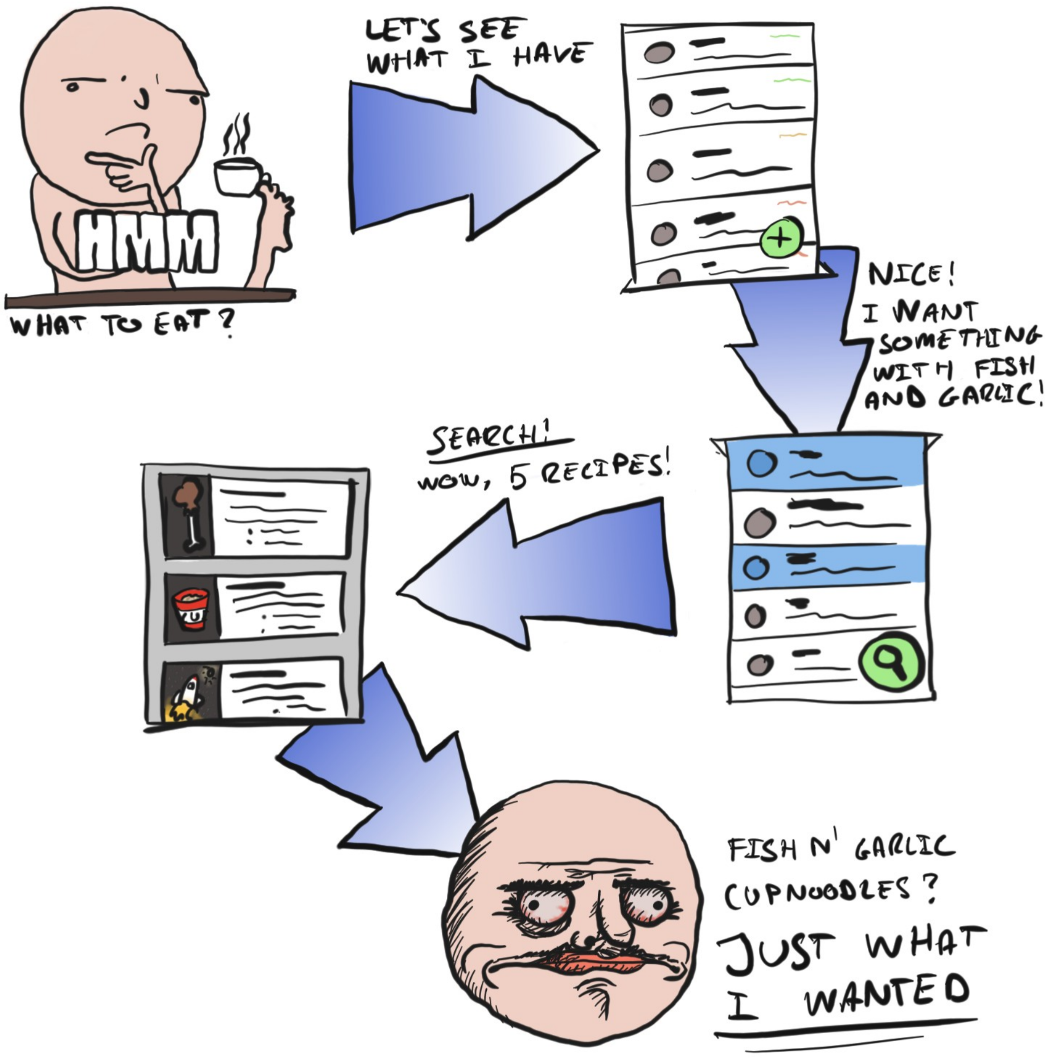
HMM
WHAT TO EAT?
LET'S SEE WHAT I HAVE
NICE! I WANT SOMETHING WITH FISH AND GARLIC!
SEARCH! WOW, 5 RECIPES!
FISH N' GARLIC CUPNOODLES? JUST WHAT I WANTED

Figure 1.1: Conceptual sketch of the core use of the app

1.2 Personal Vision

Our personal vision with the app is to expand our knowledge on how to build practical apps, and how to implement the elements learned during the course. Especially how to work together to create an app. The end-goal is of course also to have at tool we can actually use in everyday life: To have an app to use, when we just don't know what to make for dinner and to help us keep stock on our kitchen.

Requirements 2

The following user stories go through what we believe is the core functions of the app.

FEATURE: Log in

1. As a user
   I would like to log in with my google account
   so that I can sync my ingredients list between devices.

2. As a user
   I would like to avoid logging in
   so that i can keep my ingredients list locally.

3. As a user
   I would like to be able to logout
   so that I can use another account

FEATURE: List of ingredients

1. As a user
   I would like to add items to a list of ingredients
   so I always know what I have.

2. As a user
   I would like to delete items on my list
   so That I can manage items that have expired.

3. As a user
   I would like to have auto completion when inserting a new ingredient
   so I know it is spelled correctly and is more consistent.

4. As a user
   I would like to have access to multiple ingredient lists.
   So i can manage stuff at different places.

5. As a user
   I would like to be able to join a couple of lists,
   so I can search for a recipe with ingredients across lists.

FEATURE: Recipes

1. As a user
   I would like to search for recipes using what ingredients I want and have

so that I can get inspiration for dinner.

FEATURE: Shared lists
1. As a user
   I would like to share my list with other people
   so everyone can add or remove ingredients from the list.

2. As a user
   I would like to join my dorms shared list
   so that I can use all of the shared ingredients

3. As a user
   I would like to leave a shared list
   so that I am not notified when my old roommate buys tomatoes

4. As the system
   I would like to delete shared lists with no people
   To reduce disk usage

FEATURE: Expiration
1. As a user
   I would like to receive a notification when an ingredient
   expired and the system deleted it
   so I always am able to purchase a new one.

2. As a user
   I would like to be notified when someone restocks an expired item
   so that it is not restocked twice.

FEATURE: Running low
1. As a user
   I would like to flag ingredients as running out.
   So everybody concerned with the ingredients are notified.

2. As a user
   I would like to be notified when an ingredient is flagged as running out,
   so I can purchase it when convenient

3. As a user,
   I would like to flag ingredients as restocked,
   so its only restocked by me

4. As a user,
   I would like to be notified when an ingredient is restocked
   so I do not accidentally purchase it as well

5. As a user,
   I would like to be only notified once when multiple ingredients are restocked
   so I am not overflown with notifications.

FEATURE: Ease of shopping

1. As a user,
   I would like to get a list of all items expired/low
   so I know what to purchase

2. As a user,
   I would like to be able to delete an element for the shopping list
   so that it is not cluttered when I will not purchase an item again

FEATURE: Barcodes

1. As a user
   I would like to scan ingredients using their barcode,
   so that I do not strain my fingers typing in my whole fridge

2. As a user
   I would like to add information to a scanned item
   so that I can track the expiration date

3. I would appreciate to be prompted the extra needed information
   In a simple and user-friendly way
   So that it is easy to add the needed information.

FEATURE: Gestures

1. As a user
   I would like to be able to shake my app
   so that the phone searches for a random recipe with my ingredients

2.1 Scope

The user stories marked in red have not been implemented, and therefor left out of scope. They represent what the group would've wanted in a full release of the app. But they have been left out of this assignments scope, due to limitations such as technical knowledge and time pressure.

Sadly, one of the main features of the app has been left out, namely `shared lists`. However, the feature is faked by letting all online users share all lists. This is not ideal in the real world, of course, but the solution would need some server-side logic to sort the lists by users, and the group decided that there wasn't enough time to implement this.

Some of these topics are discussed in further detail in chapter 5.

Design Considerations | 3

The following section will showcase design considerations in form of a flowdiagram of the UI and a componentdiagram of the inner parts.

The following, figure 3.1, shows the final UI of the app, and is meant to provide enough information for you to get a sense of the look and feel of the application.

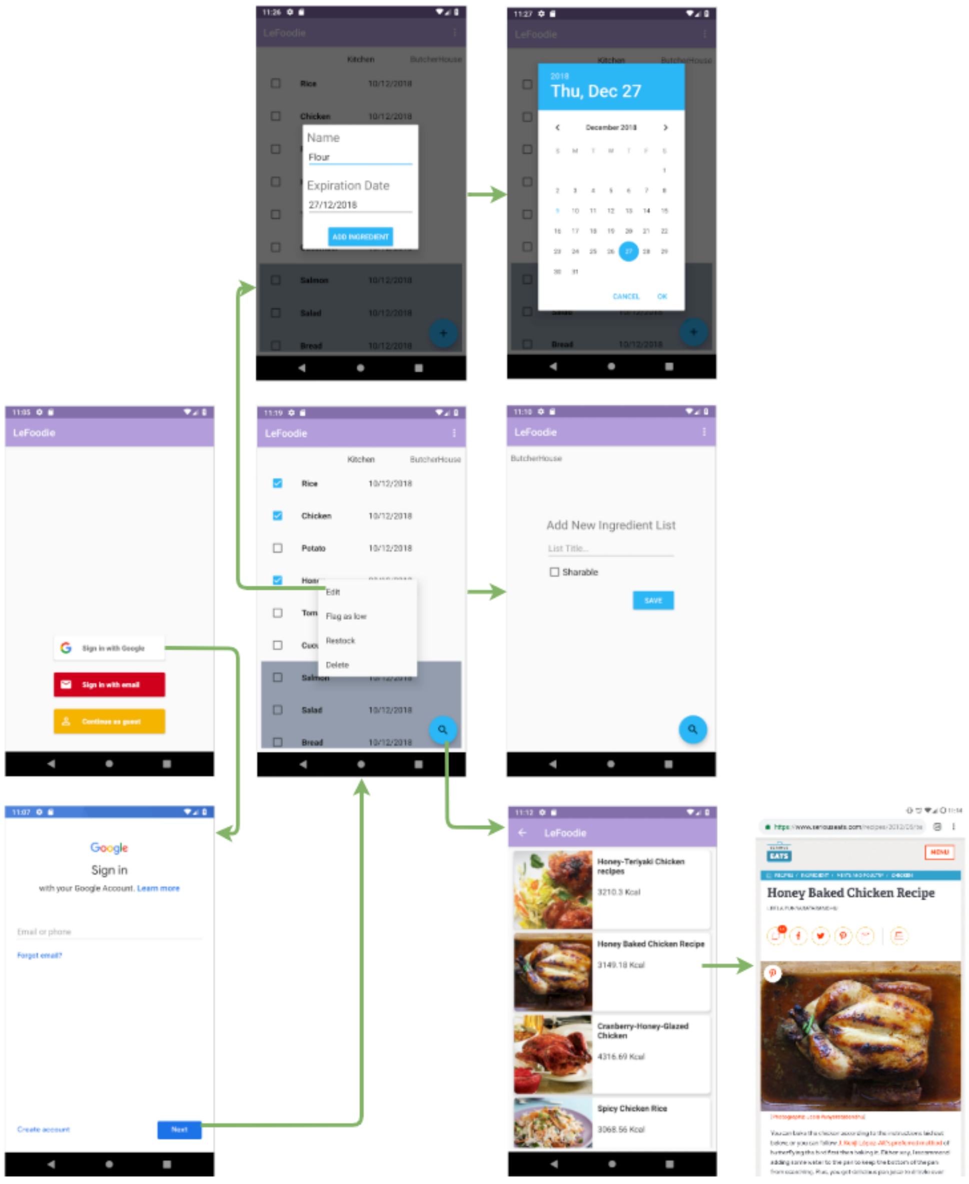

Figure 3.1: Updated flowdiagram

3.1 | Componentdiagram

The following figure 3.2, crude UML diagram, is visualizing how the app is build on an architectural level. The component diagram, true to its name, shows which components are used in the app and how they interconnect.

Bear in mind that this diagram is a display of how the current architecture is build. This means that some of the parts are not perfectly structured yet, but reflects a prototype-like structure.

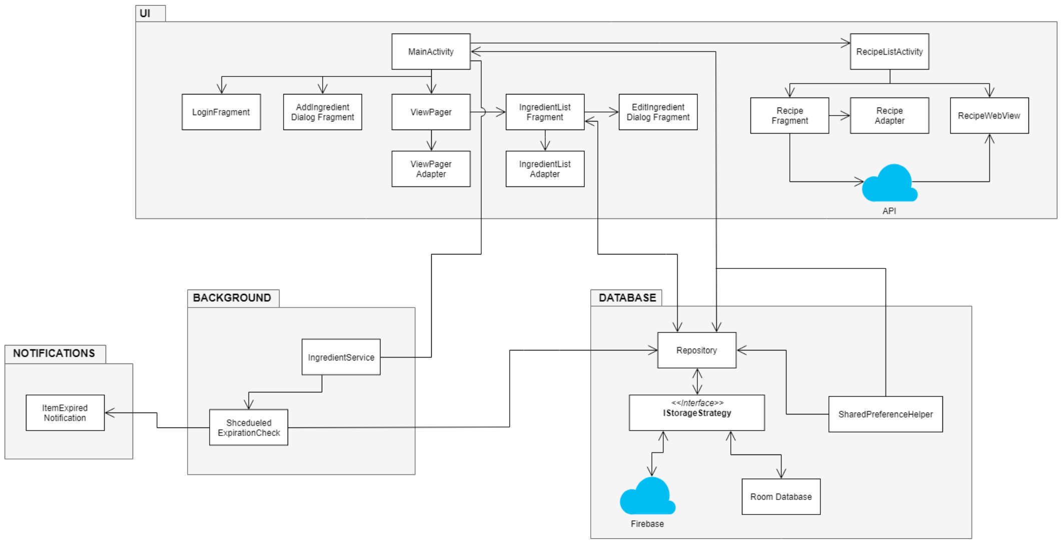

Figure 3.2: Componentdiagram

3.1.1 UI - Main

The UI package consist of two parts, the repice part and the main part, this section will cover the main part. The main activity is responsible for handling multiple fragments. We decided to use fragments instead of activities because the nature of the app required us to handle and generate a lot of UI components at the same time, which would be nearly impossible to implement with activities.

The first fragment the user will see is the login fragment, which is further described in the login section (3.1.3). When you have chosen to login or stay offline the user will be shown an ingredient list. The ingredient list is made as a recycler view, which is basically a scrollable list containing the different ingredients. If the user swipe to the far right, he/she will be able to add a new ingredient list to their inventory. Each list is created as a fragment that is controlled by a view pager that will give the user the ability to swipe back and forth through the different ingredient lists. This proved quite difficult to implement since it added another layer to the app. Initially we faced a lot of issues having the view pager work together with the livedata from the database. The ingredient list simply did not update. After spending a lot of hours trying to come up with a solution, we finally figured out what was wrong. Apparently the view pager adapter had to override the function "getItemPosition()" and return "POSITION NONE", in order for it to work properly with livedata.

The solution was found here (3rd answer, 17 likes): [2]

The ingredient list and the view pager each have their own adapter which is responsible for setting up the UI correctly. If the user does a long press on an ingredient a context menu will pop up over the corresponding ingredient. The menu will contain four options: Edit, flag low, restock, and delete.

CONTEXT MENU

The context menu is responsible for handling the above mentioned options. The context menu is handled inside the current ingredient list fragment. Each option has executes some functionality and then tell the repository what to do. The context menu proved more difficult to implement than originally expected, since there had to be a connection from the context menu to the edit ingredient dialog fragment. Initially when you did a change through the menu on the 2nd or 3rd page in the view pager, it would always apply the change to the first page. This was solved by checking on androids inbuilt function getUserVisibleHint(). This created a connection to the correct ingredient list, and not just the first.

ADD INGREDIENT FRAGMENT

While not having any ingredients marked the user can add additional ingredients to the current list, by hitting the floating action button. We decided to have this as a separate button in the main activity instead of having it in the individual fragment, since it had to work across multiple fragments. The add ingredient itself is a dialog fragment which is due to the user experience.

EDIT INGREDIENT FRAGMENT

The edit dialog fragment is very much like add ingredient fragment. The main difference is that the edit ingredient is called from within the context menu. Also the main challenge with this one, was that you had parse the original ingredient to the repository, in order for it to update it properly.

3.1.2 UI - RECIPE

The Recipe is handled by the Recipe activity. Upon receiving an intent with the desired ingredients to search for, the activity will perform a query to the Edamam recipe Api, which returns a list of 10 recipes which each contain the selected ingredients. Upon receiving the data as JSON format, a fragment together with its adapter will activate which are responsible for rendering the recipe cards with an Image, Title and calorie value. When tabbing on a recipe card, another fragment is activated which only contains a web view with the corresponding recipe.

3.1.3 LOGIN

`Firebase` is used for all authentication, with google and email as the main authentication methods. Since the offline functionality is required, it's also possible to select a guest/anonymous user.

The `AuthStateListener` detects if the authentication state changes. This is where the UI for logging in is generated, as well as the logic for what storage strategy to use.

`Firebase-UI` is used to generate all UI considering logging in. It usually have a bug that makes it impossible to go back and out of the app, if a user wouldn't like to log in. This bug is fixed by finishing the `MainActivity` in the `onActivityResult` function.

3.1.4 Background

The background service we had checks the storage everyday on 0h22m30s to see if any ingredient is expired. If it finds anything, it will fire a notification to the user. It is a background service so it runs even when the app is closed.

3.1.5 Notifications

We made a notification to prompt the users once the ingredients are expired and deleted from the user interface. The users will know which ingredients are deleted so they know which to restock for their kitchen if they need it. This is a way to keep track of the amount of ingredient. Moreover, the user can get rid of the expired stuff in the notification otherwise it will still stay there.

3.1.6 Database

The database package will handle all activity regarding data persistence. In the nature of the app, where it is desired to switch between being online or offline, a strategy pattern is utilized to handle both the `Firebase`- and `Room` connection. A `Repository` was used to abstract the different databases away, as well as handle all the asynchronous code. Using `LiveData` here ensured that the app will always be up to date when changes come in. Regardless of being online or offline.

Lastsly, a helper class kept track of all string keys needed for the app to use `Shared Preferences`.

Difference between Room and Firebase

`Room` and `Firebase` are of course fundamentally different, being a SQL- and document database respectively. This created some issues, when using the same model-class, handling IDs, and supporting the same interface.

Instead of letting `Firebase` decide the database key, it is manually set to be a integer ID. It will never be difficult to find it this way, and it will make it easier to transition from `Room`, and vice versa. However, this means it is the programmers responsibility to manage the IDs when using `Firebase.`. This is done in the simplest way possible: Finding the ID of the last entry, and adding one. This means, that the order of the lists may never be changed.

Returning `LiveData` also produced some difficulties. `Room` supports it directly, and returns it from the `IngredientListDAO`. `Firebase` does not support the live data directly. The `Firebase ChildEventListener` will maintain a in-memory list of `IngredientLists`, which is then used to replace the value of the `LiveData` used throughout the app. This solution was a product of a discussion with another group facing the same problem.

One to many

The list in the `IngredientList` model class created some difficulty. `Room` doesn't handle this one-to-many relationship itself. Many hours were spent to implement the `Relationship` tag in a wrapper class, but turned out to present some shaky behaviour. Instead, the strategy is responsible for also getting `Ingredients` and manually append them to the requested list. But this solution also presented some difficulty when the `LiveData` was added. Following a guide [1] fixed this problem, by using `MediatorLiveData`.

SHARED PREFERENCE HELPER

The helper class keeps track of all keys used to access the shared preference values, making the user-storage easy for any part of the app. `Shared Preferences` is used to store the current user, as well as needed variables to detect whether we should switch storage strategy or not.

WORK PLAN | 4

The following work plan, table 4.1, show how the group divided and conquered.

Even with an initial commit, base-lining most of the files and interfaces, there were a lot of merge conflicts. The person noted as responsible for it in table 4.1 is the one taking the bulk of the conflicts.

	Eduardo	**Jonas**	**Cuong**	**Christoffer**
Ingredient List	■			
View pager setup		■		
Adding ingredient			■	
Edit ingredient			■	
Deletion of ingredient		■		
Context menu		■		
Login			■	
Firebase setup				■
Room database				■
Repository and database strategies				■
Expiration task w/ notifications	■	■		
Ingredient flagging		■		
Recipe search	■			
UI design	■			■
Merge handling	■			

Table 4.1: Work distribution between the different tasks

Discussion | 5

Overall it has been a mixed experience working on an app project like this.

On the one hand it is nice to utilize the combined knowledge of a group to create something together, but on the other hand it takes a lot of effort to properly coordinate everything.

The work flow of the application has been an issue from the start. Working on multiple features on multiple branches resulted in many merge conflicts and bugs arising while merging the application. This problem aroused time and time again during the development cycle of the project. Especially as the group evolved during the development, which led to changes at highly used interfaces, like the repository.

Working with new technologies and methodologies has been an issue during the project. The different API versions tailors multiple solutions for similar problems, therefore it has been difficult finding resources for the specific problems which rouse while developing the application. This resulted in trial-and-error, which could cost several hours till the right solution was found.

The group chemistry consisted of 2 IKT students, 1 E Student and 1 Exchange student. Because of the different backgrounds, the individual understanding of programming concepts have had different levels. Different starting points has led to some members requiring more time and resources to complete the individual tasks, which overall has had minor effects on the group balance and delivery.

Conclusion | 6

To recap, our app vision was to create a tool which allows users to easily get inspiration for what to cook with the ingredients they had at their disposal. Also our personal vision was to expand our app knowledge over the course of the project.

Our final project meets the vision to pretty decent degree. Although not all of the "quality of life" features were implemented, most of the core features were implemented, at least to an extend where the user would have a good experience.

During the development of the app, we have encountered a lot of challenges, especially when encountering new stuff we had not been working with before. Although there were some difficulties the group learned a lot from these, and hence fulfilled our personal vision.

If the app was to be released, the rest of the core features would be a requirement, but for this assignment it meets the scoped requirements.

Known bugs | 7

After going back from watching a recipe, if you swipe all the way to the right, and then back, the ingredients won't be selected anymore, but the button will still be a search button. These ingredients are still registered in the selected list, so you can add the same ingredient twice.

The default offline list shows right before the loginscreen appears.

After login, the livedata doesn't work. Have to do with the observer being added in the login callback.

The expired ingredients are deleted at irregular intervals, even though the time is set to once a day, at 00:23 am

Sometimes, the recipe link is not opened in our own webview, but opened in chrome.

Going back from a shown recipe will go all the way back to the ingredient lists, not the recipe list from before.

If going back from a recipe in chrome, you will encounter a blank activity.

Going from an anonymous user to logging in, might overwrite some excisting lists in the Firebase database, since new IDs from the room db are not given.

First time you log in, the livedata observer isn't attached properly, meaning no updates at all.

References

[1] Reza Bigdeli. *Android Room: Handling Relations Using LiveData*. 2018-12-09. `https://proandroiddev.com/android-room-handling-relations-using-livedata-2d892e40bd53`: ProAndroidDev.

[2] Stackoverflow. *Update ViewPager dynamically*. Visited: 2018-12-09. `https://stackoverflow.com/questions/10849552/update-viewpager-dynamically`: Stackoverflow.