University for Applied Sciences Informatics Department Applied Informatics

Food planner-Documentation

|  |  |
| --- | --- |
| Tom Brauns | Diana Leticia Aguilar-Jiminez |
| Inf3621@hs-worms.de | Int2482@hs-worms.de |
| 676672 |  |

|  |  |
| --- | --- |
| Professor: | Prof. Dr. Stephan Kurpjuweit |
| Semester: | Winter Semester 2022/23 |
| Due Date: | 15. January 2023 |

Table of Contents

[1. Introduction and Goals: 3](#_Toc124847948)

[1.1: Requirements Overview: 3](#_Toc124847949)

[1.2: Stakeholders: 4](#_Toc124847950)

[1.3: Use Cases: 5](#_Toc124847951)

[1.4: User Stories: 6](#_Toc124847952)

[2. App Development 7](#_Toc124847953)

[Starting the Process: 7](#_Toc124847954)

[2.1: Calendar-Fragment 11](#_Toc124847955)

[2.1.1: Functionality of the Calendar-Fragment: 11](#_Toc124847956)

[2.1.2: Required Layout-Files: 15](#_Toc124847957)

[2.2: Recipes-Fragment 19](#_Toc124847958)

[2.2.1: Functionality of the Recipe-Fragment: 19](#_Toc124847959)

[2.2.2: Required Layout-Files: 24](#_Toc124847960)

[2.3: Shoppinglist-Fragment 26](#_Toc124847961)

[2.4: User-Fragment 27](#_Toc124847962)

[Glossary: 28](#_Toc124847963)

# 1. Introduction and Goals:

The Focus of this App is a working Prototype of a Food planner. That includes a Calendar and Recipes to select. Users should be able to use this app to plan their Meals and know what they require for them. For this to work and be used efficiently, a lot of navigations in between the apps pages is required, considering that The User can start off at

## 1.1: Requirements Overview:

#### Functions include:

* The presentation of a calendar
* The selection of a date
* Select time of day
* The selection of a recipe
* The detailed presentation of the selected recipe
* The option to add ingredients to the shopping list
* The option to access the shopping list
* The option to remove items from the shopping list
* The option to add custom items to the shopping list

#### A summarization of the functional requirements would look as follows:

|  |  |  |  |
| --- | --- | --- | --- |
| ID | Requirement | Description | Reasoning |
| F-1 | Start App | The User starts the application | Every app requires a booting point |
| F-2 | Calendar Tab | The User is presented with the Calendar Fragment. | This is the Apps starting point. |
| F-3 | Highlight current date | The current date is highlighted in the Calendar overview | This helps the user to better grasp the core functionality of the app |
| F-4 | Select date | The User selects the day he wants to interact with. | By selecting the date, the User can further interact with the app |
| F-5 | Select time of day | The User can select between Breakfast, Lunch, and Dinner | Considering that people are expected to eat three meals a day, it is necessary that a minimum of three separate Meals can be selected / added |
| F-6 | Select Recipe | The user adds a recipe to the selected meal slot | Users need to be able to add a recipe in the Meal slot to properly plan their meals |
| F-7 | Remove Recipe | The user removes an already selected recipe from the meal slot | User should be able to remove a recipe from the meal slot in case their plans changed |
| F-8 | Recipe Tab | The User is presented with the Recipe Fragment | One of the implemented Fragments is the Recipe Tab. Users can browse the available recipes |
| F-9 | Browse Recipes | Users can scroll through the available recipes | Considering that there is a bunch of available recipes, it is necessary that you can scroll through for convenience |
| F-10 | Show Details on Recipe | The user can select one of the recipes to get further details regarding the recipe | This provides the user with a more in-depth understanding what’s required for the recipe |
| F-11 | Add to shopping List | Users can click “add to shopping list” when accessing the ingredients | To properly plan a meal, it is necessary that the user can remember the recipes properly |
| F-12 | Add to Meal | User can click a Date and Meal to add this recipe to. | For usability of the app, a navigation from recipes to the calendar makes sense |
| F-13 | shopping list Tab | The user is presented with the shopping list Fragment | This Fragment is required to help the User in planning their meals |
| F-14 | Add Ingredients | The user can add Ingredients | The User can, in addition to adding ingredients by selecting a recipe, add custom ingredients |
| F-15 | Remove Ingredients | The user can remove Ingredients | As plans change, the need to remove ingredients from the list also remains. |
| F-16 | Navigate | The user can Navigate through the application | As a core functionality, users can access the different tabs via the bottom navbar. |

## 1.2: Stakeholders:

#### Our potential Stakeholders include:

|  |  |  |
| --- | --- | --- |
| Stakeholders | Type of Stakeholder | Reasoning |
| Students | Primary | As one of the main target groups of this app, Students should be kept in mind |
| Family | Primary | Families are one of the main target groups for this app, as it’s mostly the parents, that plan the meals for the day / upcoming days. |
| Children | Secondary | Considering that they are affected by the Parents using the application and buying ingredients, they should be kept in mind |
| Sponsor | Primary | Considering that they might benefit from the success of this product |
| Media | Secondary | To further increase the success and the popularity of the Application, the project is under the guidelines of major companies (google etc.) to have higher chances of being promoted to customers. |

## 1.3: Use Cases:

#### Considering the in [1.1 defined functional requirements](#_A_summarization_of) the user is left with (but not exclusively) the following use cases

|  |  |  |
| --- | --- | --- |
| ID | Use Case | Description |
| UC-1 | Start App | The User can start the application |
| UC-2 | Select Date | The User can see the Calendar overview and can select a date |
| UC-3 | Select Meal | The User can choose between the 3 Meals, being Breakfast, Lunch and Dinner and select one of them. |
| UC-4 | Select Recipe | The User can select a Recipe after selecting a Meal. |
| UC-5 | Add Ingredients to shopping list | The User can add the ingredients required for the recipe to the shopping list |
| UC-6 | Scroll through Recipes | The User can scroll through the Recipes provided in the Recipes-catalogue. |
| UC-7 | Click Recipe | The User can click on the Recipe to gather further information |
| UC-8 | Add Ingredients to shopping list (Recipe) | The User can add the ingredients required for a recipe to the shopping list |
| UC-9 | Select Date (Recipe) | The User can choose a date and meal to add this recipe to. |
| UC-10 | Add Ingredients | The User can add custom Ingredients to the shopping list |
| UC-11 | Remove Ingredients | The User can remove already existing ingredients from the shopping list. |

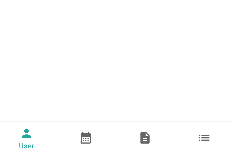
## 1.4: User Stories:

#### The core of each app is to focus on the main functionality at first, so that an application, having the core features implemented, can be used already. To best fulfill this approach, we’ll focus on the most important Use-Cases and User-Stories.

|  |  |  |  |
| --- | --- | --- | --- |
| ID | User-Story | Function | Reference |
| US-1 | As a User I want to be able to select a date to plan the meals for. | Select Date | UC-2, F-4 |
| US-2 | A a User I want to select a time of day to slot the meal into to plan a well-thought-out day. | Select Meal | UC-3, F-5 |
| US-3 | To plan my Day, I want to be able to select a recipe to add it in the Calendar | Select Recipe | UC-4, F-6 |
| US-4 | As a student I want to be able to add the required Ingredients to my shopping list to keep track of what I need to buy later | Add Ingredients to shopping list | UC-5, F-11 |
| US-5 | As a parent I want to scroll through the recipes to find a recipe to prepare. | Browse Recipe, Select Recipe | UC-6, F-9 |
| US-6 | As a User I want to be able to select a recipe to see more details regarding the recipe | Click Recipe,  Show Details on Recipe | UC-7, F-10 |
| US-7 | To plan better I want to add the ingredients to my shopping list to know what I will need to buy | Add to shopping list, Add Ingredients to shopping list (Recipe) | UC-8, F-11 |
| US-8 | To optimally use this app in my daily life I want to add ingredients to my shopping list to buy everything I need. | Add ingredients | UC-10, F-14 |
| US-9 | To effectively use this app, I want to be able to remove ingredients from my shopping list to know what I still need to buy / what I already have. | Remove ingredients | UC-11, F-15 |

# 2. App Development

## Starting the Process:

Ein Bild, das Text enthält.

Automatisch generierte BeschreibungWe did a bunch of sketches to have a clear understanding of how we wanted the app to work. This led to the general understanding that we had the need for a Calendar-Fragment, a Recipes-Fragment, and a shopping list Fragment as the core-features. Those three Recipes would need to interact with one another (some of the Navigations were discussed in the [Use Cases](#_Considering_the_in) and the [functional requirements](#_A_summarization_of) already). In order to help the navigation process, we agreed on using Fragments and implemented a BottomNavigationView. After finding an example project that had a [Bottom Navigation View](#_Glossary:), we started to work on those individual portions of the application. Understanding how the components work with one another and how the navigation actually works was key.

Ein Bild, das Text enthält.

Automatisch generierte BeschreibungTo best explain how the actual Navigation works, lets take a look at the mobile\_navigation.xml

Ein Bild, das Text enthält.

Automatisch generierte BeschreibungIn the actual navigation, there are fragments,

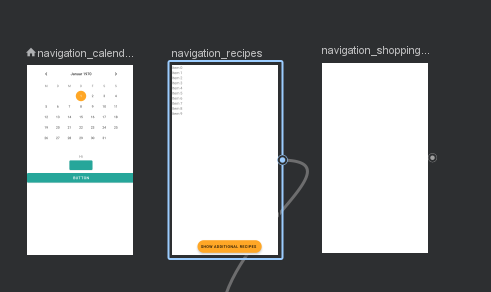
defined with an id, an name (referring to the actual fragment it relates to), a label (the portrayed name when being on that fragment on the top) and the corresponding layout.xml file.

Another file of great importance is the bottom\_nav\_menu.xml. Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

In here we have the “item” containers, that each have an id, an icon and a title.

That being said we were left with three main Fragments to work on, the Calendar-Fragment, the Recipes-Fragment, and the ShoppingList-Fragment.



Ein Bild, das Text enthält.

Automatisch generierte BeschreibungConsidering that we talked about each of the individual fragments that are of great importance, let’s take a look at the main activity. Let’s start off by taking one more look at the activity\_main.xml.

Ein Bild, das Text enthält.

Automatisch generierte BeschreibungThis xml-File uses a Linear Layout, due to it being easier to understand and customize than the alternative ( being ConstraintLayout). Now that we know what the activity\_main.xml looks like, lets look at our MainActivity.kt.

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung Let’s Focus on how the actual Navigation works: The Component that handles the navigation is the “FindNavController” function. This has the ID “nav\_host\_fragment\_activity\_main”, which refers to this ID in the activity\_main.xml. This uses the androidx library to handle the navigation and uses the navGraph to handle the defined navigation. As we have previously explained the mobile\_navigation.xml, I’ll not talk about it further.

Ein Bild, das Text enthält.

Automatisch generierte BeschreibungNow that we know more about how the “findNavController” function works and which data it grabs, lets look at the AppBarConfiguration ( being the Element that handles the Fragments that are portrayed in the BottomNavigationView. So the App Bar has a set of 4 different Fragments, those being referenced in here via IDs, which are used in the Navbar-Element.

Now that we have discussed how the ActivityMain works and how the corresponding .xml files handle the navigation, lets take a look at the fragments.

## 2.1: Calendar-Fragment

### 2.1.1: Functionality of the Calendar-Fragment:

Considering that this is one of the two core features, lets start off by talking about the Calendar and its importance to this application. We decided to declare the Calendar as the starting point of this app. Users navigate the application starting at the Calendar-Tab.

Let’s firstly look at the DayMeal-Class

Ein Bild, das Text enthält.

Automatisch generierte BeschreibungThis Class has a date-string, as well as a breakfast, a lunch- and a dinner-String and an id.

Ein Bild, das Text enthält.

Automatisch generierte BeschreibungIn addition to the DayMeal-Class there is also the MealClickListener.kt-File. This one is pretty basic in its functionality aswell, only including an onClick function to select the DayMeal (being the selected Meal).

Now to the rather large Files of the Calendar-Fragment: We begin by taking a look at the CalendarFragment.kt itself. In here we have the onCreateView. Ein Bild, das Text enthält.

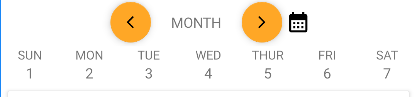
Automatisch generierte Beschreibung

As portrayed on the previous Code-Snippet, we use the inflater for the view and inflate the fragment by making use of our fragment\_calender layout-file(See here for the exact layout of this).

We also need some data for the Calendar, which we get by referencing the Calender.class that android provides, aswell as linking some of these variables to the fragment\_calender layout file.

For the formatting we use the SimpleDateFormat.classEin Bild, das Text enthält.

Automatisch generierte Beschreibung. MonthlyCalendar references the CalendarView.class, which we use as a widget for the Monthly Calender. With the setOnDateChangeListener, we update the Date to the current date (using a function of the CalendarView.class). With the buttonPanel, we add an setOnClickListener to our button in the layout-File and add a Toast to it.

Ein Bild, das Text enthält.

Automatisch generierte BeschreibungNow that we have discussed the monthly view of the Calender in length, let us look at the WeeklyCalender.kt.

Considering that we worked with tables, there are a bunch of layout-Elements that we relate to to properly relay our information to lateron.

Ein Bild, das Text enthält.

Automatisch generierte BeschreibungAfterwards we initialize the Elements with the accurate data, by making use of the Calendar.class aswell as the SimpleDateFormat.class.

Now that we have all those parts set up, we can start with the functionality. Lets begin by discussing the Navigation of the Calendar. You can swap the inspected week by clicking the FloatingActionButtons.

Ein Bild, das Text enthält.

Automatisch generierte BeschreibungConsidering as their functions are similar, let’s look at one of them:

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung Seeing as FloatingActionButtons are buttons in its core, we added a setOnClickListener to it and navigate the date by using the calendars functions and adding – or in this case subtracting a week from it. We then update the date and use the formatter. The result of this then gets added to the layout-File and we add a Toast.

In addition to those buttons, we made a function named “fillMeals” to have an placeholder for some of the days to be filled with custom recipes later. This is meant to show the users how the overall design will look and how it functions. Those placeholders are added to the mealsList lateron.

Lastly, we took to the onViewCreated function. This function fills the Meals. In addition to that it connects to the layout-file and links the recyclerView.java (the class) to our specific RecyclerView (mealsRecycle). We then make use of the MealsAdapter class, which we will explain shortly.

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Now, how does the MealsAdapter work. The MealsAdapter.kt consists of two parts. There is the MealsAdapter-class, and the MealsViewHolder-class.

Ein Bild, das Text, Screenshot, drinnen enthält.

Automatisch generierte Beschreibung

Besides the LayoutInflater and the Bindings created in this MealsAdapter-class, there is the onBindViewHolder-Function, which uses the ViewHolder initialized earlier and linked to the layout file, and fills it with the actual meals at their corresponding positions. Ein Bild, das Text enthält.

Automatisch generierte BeschreibungIn this binMeal-Function the Data of the meals gets linked to the current meal by relating to the card\_weekly.xml and pulling the data and filling it with the actual data. After the date and the three meals are implemented for that date, an setOnClickListener is set up to be used lateron.

### 2.1.2: Required Layout-Files:

In this Fragment there are three important layout-Files. We begin by talking about the fragment\_weekly\_calender.xml:

Ein Bild, das Text enthält.

Automatisch generierte BeschreibungEin Bild, das Tisch enthält.

Automatisch generierte BeschreibungAs this is a very important File, we start discussing it line by line. Let’s start with the top. There are two Floating Action Buttons, which help with the Date-Navigation.

In Addition to the Floating Action Buttons, there is also an ImageButton. Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

In the next line, we use an TableLayout. We have two Columns for this, one being for the Days of the Week, and one for the actual Numbers.

Ein Bild, das Text enthält.

Automatisch generierte BeschreibungAn example for the Week-Elements would be this TextView-Element. Considering that this is a TableLayout, we are using TableRows to encapsulate our Elements (in this Case the TextView-Elements) in.

Like the Week-Elements, we have another TextView-Element, that contains the Numbers.

Ein Bild, das Text enthält.

Automatisch generierte BeschreibungIn addition to the layout-specifications such as width, height, and weight, we also have an id for the TextView-Elements on the Numbers of the Days.

Most of this screen is filled with the RecyclerView, which is called mealsRecycle. The contents of these are filled in the card\_weekly.xml

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Let’s look at the contents in specific:

Ein Bild, das Tisch enthält.

Automatisch generierte BeschreibungThe entire object is considered as an CardView-widget. In this widget, we have TableRows.

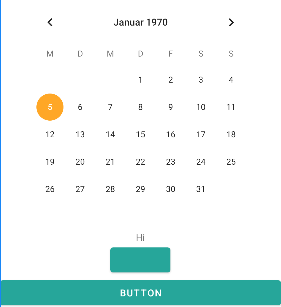
To best explain this, we look at the Breakfast-Row of the CardView-widget:

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

In here, we have the image for a croissant as an ImageView-Element, followed by an TextView-Element that is the time of meal, and then there is the potentially selected Recipe.

Now as you might have noticed already with the little Calendar-icon at the “fragment\_weekly\_calender.xml, there is an additional view for the Calender, which portrays the monthly view (this layout file is being called “fragment\_calender”)

In its core, it looks like this. But let’s get into the details of it.

Ein Bild, das Text enthält.

Automatisch generierte BeschreibungFor the creation of the monthly Calender we make use of the CalenderView, which is an preexisting android widget.

As further Elements this layout contains a TextView and two Buttons, but seeing as both of these aren’t of great complexity I’ll spare the reader the snippets of those.

## 2.2: Recipes-Fragment

### 2.2.1: Functionality of the Recipe-Fragment:

Thinking about how the Recipes should work, we realized that we would need a Database, including the corresponding structures. The first attempt of the recipes\_catalog.json looked as follows:

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Each of the recipes has a product\_id (which is Unique), a title, an description and ingredients, aswell as the potential for an image\_file\_url.

Ein Bild, das Text enthält.

Automatisch generierte BeschreibungEin Bild, das Text enthält.

Automatisch generierte BeschreibungWhile the first realization of this relied on a solution without using a DataAccessObject (Dao) or Room, this later got changed to an approach using room by closely working with projects presented in the lecture (e.g., Criminal Intent). In the process of realization, a bunch of files were added. To cover some of them, let’s look at the Recipe class itself, the object that we consider the “Recipe”

The data class “Recipe” consists of the id, the title and further information being the description, ingredients and the imageFileURL.

This mostly stayed the same throughout the process of the Fragment, except for some variables being added and others being removed in the process of creation.

Ein Bild, das Text enthält.

Automatisch generierte BeschreibungConsidering we are looking at the data folder at the time being, lets continue with the RecipeDao. In here we have the actual Database accepts. So the link between the Room and the recipes\_catalog.json. This includes the “getRecipes” function that pulls the entire list of all Recipes, then we have the selection of a specific Recipe (with the ID of the Recipe for the RecipeDetailFragment and the Navigation). Lastly there is the addRecipe function to add a new Recipe lateron.

Ein Bild, das Text enthält.

Automatisch generierte BeschreibungAs a third component of the data folder, there is also a RecipeDatabase-File, that generally defines the RecipeDatabase as a subclass of the RoomDatabase.

Ein Bild, das Text enthält.

Automatisch generierte BeschreibungNextup there is the RecipeRepository.kt (in the model-Folder).

The RecipeRepository.kt handles the contruction aswell as the actual functions of the Functions (getRecipe, addRecipe). These functions have the actual parameters and the database accesses. So this would be the File that actually uses the RecipeDao.

Ein Bild, das Text enthält.

Automatisch generierte BeschreibungBefore we can talk about the RecipeListFragment, we need to take a look at the RecipeViewModel.kt(in the viewmodel-Folder).

In this class we access the recipeRepository and access the recipes (defined as \_recipes) and initialize them after with the init and then the getRecipes().collect, and declaring values to each of them ( creation of the object). Considering that a new Recipe wouldn’t be in this initialization process, there is also an “addRecipe” function.

Ein Bild, das Text enthält.

Automatisch generierte BeschreibungNow to the actual functionality of the Recipe-Fragment. First off, lets start by taking a look at the RecipeListFragment. To focus on the core functionality we once again shortened the scope of this documentation, and are only going to talk about the “onViewCreated”.

onViewCreated would be considered the “STARTED” state, as the Fragment has to be running already. Now we access the recipeViewModel and grab the initialized list of Recipes. We fill these Recipes into the recipesListRecyclerView ( We will discuss the way that this looks like lateron). The File “RecipeListAdapter” majorly helps with the contents of the recipesListRecyclerView( We will also talk about this File lateron).Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Now to the core Feature: If the User clicks on an actual Recipe in the List, he will be navigating to a different “Fragment” that shows him an in-depth overview of the Recipe he selected (the RecipeDetailFragment (still under construction). In order to realize that navigation, we use the findNavController().navigate(RecipeListFragmentDirections.showRecipeDetail(recipeId)). Let’s break this line down into its components: the RecipeListFragmentDirections refers to the Fragment “RecipesFragment” defined in the mobile\_navigation.xml. In this Fragment there is an action, also commonly called a function, that is called “show\_recipe\_detail”. This function leads to the recipeDetailFragment (the destination of the navigation).

Ein Bild, das Text enthält.

Automatisch generierte BeschreibungIn order for this navigation to work properly, the function requires an recipeId, so that the App knows which Recipe to navigate to.

Ein Bild, das Text enthält.

Automatisch generierte BeschreibungWith the core functions outlined, let’s take another look at the parts we’ve skipped over. As previously discussed, there is the RecipeListAdapter.kt that helps with the contents of the RecyclerView. The thing that is most interesting for us in this is the “RecipeHolder”-portion.

Ein Bild, das Text enthält.

Automatisch generierte BeschreibungRecipeHolder is another class in the RecipeListAdapter-File.

In it, we have the function bind that receives all of the elements that are included in the recipe aswell as the actual settings for the Recipe (an setOnClickListener that is used for the navigation lateron in the way that it provides the recipe.id of the clicked Recipe).

### 2.2.2: Required Layout-Files:

Now that we have discussed all of the files that are included in the recipes folder, lets get over the used .xml files for this Fragment.

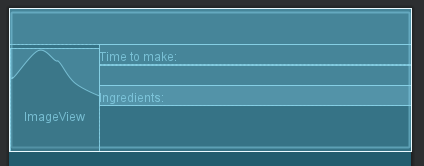
First of all there is the fragment\_recipes.xml. In here we define the RecyclerView. This includes several layout constraints aswell as scrollbar customizations and settings. Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

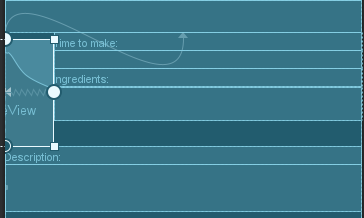
Now to actually customize the RecyclerView-Elements, we take a look at the recipes\_list\_item.xml. Considering that we do not wan’t to overexaggerate, lets only look at two Elements.

Ein Bild, das Text enthält.

Automatisch generierte BeschreibungAll of the Elements included in the Recipe are inside of a clickable button. This helps the “setOnClickListener”-Function that we discussed earlier.

To Quickly go over the Elements, there is the name of the Recipe, an Image of the Recipe, aswell as Captions regarding Time to make aswell as Ingredients with their respected informations.

Lastly, there is the fragment\_recipe\_detail.xml, so the Layout for the selected Recipe.

This contains the Information already provided in the recipe\_list\_item.xml aswell as an in-depth-description on how to actually make the meal.

## 2.3: Shoppinglist-Fragment

While this Fragment is one of the core aspects of the app, it is not the first concern, due to it relying on the Recipes and the Calendar. The only function that it could provide would be the option to add new ingredients to the list, which wasn’t the first approach we took to it, due to the app being focussed on a Foodplanner.

## 2.4: User-Fragment

While this app would optimally have a User-Tab in which the User can customize his settings / log in to save his recipes by binding them to a profile, it is not an essential for the first prototype of our application, so the focus shifts away from this Fragment.

# Glossary:

Bottom Navigation View = An Element that handles the navigation between the Fragment “pages”, by providing some Icons and spacing for each of the fragments included in the navigation bar

FloatingActionButton = A clickable Button that is shown over the original content, hovering over it.