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](#_Toc126498492)

[1.1: Requirements Overview: 3](#_Toc126498493)

[1.2: Stakeholders: 4](#_Toc126498494)

[1.3: Use Cases: 5](#_Toc126498495)

[1.4: User Stories: 6](#_Toc126498496)

[2. App Development 7](#_Toc126498497)

[Starting the Process: 7](#_Toc126498498)

[2.1: Calendar-Fragment 11](#_Toc126498499)

[2.1.1: Functionality of the Calendar-Fragment: 11](#_Toc126498500)

[2.1.2: Required Layout-Files of the Calendar-Fragment: 21](#_Toc126498501)

[2.2: Recipes-Fragment 25](#_Toc126498502)

[2.2.1: Functionality of the Recipe-Fragment: 25](#_Toc126498503)

[2.2.2: Required Layout-Files of the Recipe-Fragment: 35](#_Toc126498504)

[2.3: Shoppinglist-Fragment 37](#_Toc126498505)

[2.3.1: Functionality of the Shoppinglist-Fragment: 37](#_Toc126498506)

[2.3.2: Required Layout Files of the Shoppinglist-Fragment: 39](#_Toc126498507)

[Glossary: 40](#_Toc126498508)

# 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. |

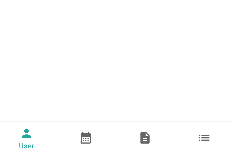
## 1.4: User Stories:

#### 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,

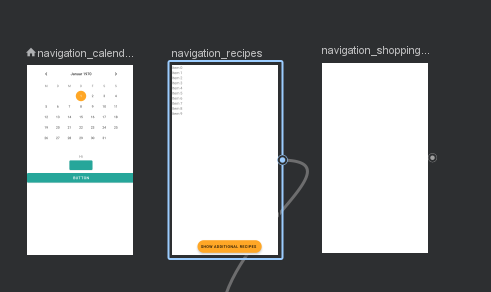
defined with an id, an name (referring to the actual fragment it relates to), a label (the portrayed name when being on that fragment on the top) and the corresponding layout.xml file.

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, let’s 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. The general Structure of it is categories, being the **daily calendar**, the **weekly calendar** and lastly the **monthly calendar**.

Ein Bild, das Text enthält.

Automatisch generierte BeschreibungEin Bild, das Text enthält.

Automatisch generierte BeschreibungBut before we delve into these categories, let’s look at the two files in the data folder. Lets start off with the MealsName class. In here we have a bunch of variables set up that the class requires, such as an UUID.

Now that we have the MealsName class set up , we also want a data class “Meal”, which also uses a UUID as its PrimaryKey.

Now that we looked at the dataclasses briefly, lets start off with the startingpoint of the application, being the **weeklycalendar**.

In the weeklycalendar, we have another class defined as “DayMeal”Ein Bild, das Text enthält.

Automatisch generierte Beschreibung.

It consists of a date-string, aswell as the 3 different meals in a day and an id.

The WeeklyCalendar.kt has most of the functions included in it, so lets look at it next.

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

As we can see, we work with an Instance of the Calendar class which is provided in Kotlin. We grab an instance and then use it to navigate. In doing so we can use the Floating Action Buttons(FAB) that we’ve set up before, to navigate a week back, aswell as a week ahead ( backDate, nextDate).



The little Calender Icon functions as a Button and the onClickListener handles the navigation by creating a new Instance which would then be the Montly view of the Calendar, once again using the Calendar class.

Now as already depicted the Buttons on top had an updateWeek()-function attached to them. Lets see what it does specifically:

Ein Bild, das Text, orange enthält.

Automatisch generierte BeschreibungWe start off by taking a look at an ArrayList of Days and create a function for each of them

We then access the recipeDao to get the function “getMeal” and create an accessable object for us. Using our previously defined DayMeal class, we check if the Database has an entry for the specific date. If it does (here the else), we bind the Data included in the MealsName class with the actual data from the Dao and input it.

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

We then add this Meal to the mealsList and continue the process for every day of the week.

Lastly we have an onViewCreated-Function in the WeeklyCalender.kt which has an adapter. This adapter has an onClickListener – being a function that will be triggered once a Meal is clicked.

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

We grab the corresponding month and year aswell as the day from the dayMeal.kt and create the DailyViewFragment ( so basicly we navigate to the next Fragment ). But before we go over to the DailyCalender, lets take one last Look at the MealsAdapter.kt:

Ein Bild, das Text enthält.

Automatisch generierte BeschreibungIn the MealsAdapter.kt, we grab the card\_weekly.xml( so the visual design of the data we input), and fill it with data:

As previously mentioned in the WeeklyCalender.kt, the cards are created as an object. If the onClick of a meal is clicked, we will switch to the daily View ( that specific date will be opened).

Now that we clicked a day, lets check out the **DailyCalendar**: Similar to the Weeklycalendar, this

Ein Bild, das Text enthält.

Automatisch generierte Beschreibungfragment also has a class of its own, containing the Icon of the Meal, the headline, the Recipename ( if already existing), aswell as the meals and the mealsId for the dropdownmenu.

In the **DailyCalender.kt,** we start off by assuming that the input has the following data given to us:

day, month and year.

Ein Bild, das Text enthält.

Automatisch generierte BeschreibungWe then grab ourselves an instance of the Calendar Fragment again and fill it with a preset of data.

After that we get the actual data by accessing “Day\_of\_month and the time from the calendar-class.



At the bottom of the Fragment there is a SaveMeals button. In its onClickListener, we define how we actually add a Meal to the Day or rather the meals.

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

We work with the recipeDao functions, depending on if we already have a meal in that slot or not. Either way we require the day, month and year of that specific day, If there is no meal, we attempt to add a Meal with a specific UUID to one of the three slots (being breakfast, lunch or dinner). After we are done, we add the Meal using the addMeal function.

IF there already is a meal selected, we instead access the updateMeal function, changing the meal previously set. We then add a toast to let the user know it worked out.

Ein Bild, das Text enthält.

Automatisch generierte BeschreibungIn order to actually have recipes showing, we use the recipeDao again and get the Recipes, to then show the Recipes.

The showRecipes-Function looks asfollows:

Ein Bild, das Text enthält.

Automatisch generierte BeschreibungTheshowRecipes-function receives the recipes gathered by the Dao and bind the title of the recipe to the food and the id of the recipe to the uuid for each recipe.

Considering that we have an showRecipes-Function, we have to assume that a lot of the slots are initially empty. So for those cases where users havent selected meals for a day yet, they are empty. For these cases, we use the MealConstants.kt:

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

With the no\_selection\_meal option, we let the user – similar to a hint – know that he can add a meal here.



We then partially inflate the layout by adding all three different meals with the corresponding icons, and start the actual cration of the Daily cards.

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

In the case that our mealDb is empty, by default we suggest to the user, that he can Select a meal. If there is an entry in the mealDb, we check if the mealDb(being in this face the function of getMeal) has meals saves for this slot. If none are selected we once again hint, that the user can select a meal.

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

We then continue to add all of the different elements together, working with an Index and iterating over them one by one. This will end up in resulting in the final card. We fill up the elemts required in the DailyMealModel, being the Mealtype( referenced by index), the corresponding icon, the defaults and the foods and uuids that are addable.

Now let’s take a look at the **DailyMealAdapter**:

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

The ViewHolder itself links the data to the layout File. Furthermore we have an onMealListenerEin Bild, das Text enthält.

Automatisch generierte Beschreibung,

which is required for the different buttons.

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Forr this we have the onRecipeSelected-Function, the onShoppingListSelected-Function, aswell as the onSelectedRecipe-Function.

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

For the addShopping and the seeRecipe-Functions we use the mealSpinner to navigate us, using the UUID of the DailyMealModel with the item.mealsId.

Now that we have had a glimpse into the structure of the DailyMealAdapter.kt, lets look at how those functions look like in the DailyCalender.kt:

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

If the user presses the “Recipe-Detail”-Button without selecting a recipe, a Snackbar appears letting the user know that no meal has been selected. Otherwhise, a new Instance is created with the corresponding id referencing to the selected recipe. So that the fragment “RecipeDetail” is opened portraying the information required of the recipe.

As the onShoppingListSelected function will be discussed later in the Documentation ( see RecipeList-Fragment), I will shorten the Documentation here:

Ein Bild, das Text enthält.

Automatisch generierte BeschreibungIf there is a meal selected, the user can click on the shoppinglist-Button to have a popup asking him if he wants to add the ingredients to the shoppinglist. If he decides to press yes, each of the items is iterated and added to the ShoppingList with a corresponding reference ID of the Item. Afterwards there will be a toast letting the users know that the ingredients have been added.

Lastly there is the onSelectRecipe-function, which handles the structure of where the meals belong, or provide a Toast in case something went wrong.

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Now that the WeeklyCalender works and the DailyCalender works aswell, lets take one more look at the **MonthlyCalendar:**

Similar to the two previously discussed calendars, we have a DayCellModel-class here:

Ein Bild, das Text enthält.

Automatisch generierte BeschreibungUnlike the daily and the weekly calendar, we are not too concerned with the meals in this one. In here we have three booleans checking If the meals have datainputs, and lastly the day of the Calendar it references.

We once again grab an instance of the Calendar and fill it with a preset of data:

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Then we fill it with the actual data:

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

We then integrate the navigation by having the backDate and nextDate-Floating Action Buttons accessing the calendar.add functions.

To navigate back to the weeklyCalendar, the user can click on the Icon, which is a Button.

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

As shown in the FloatingActionButtons, we accessed a function named updateMonth():

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

In it, we access the recipeDao to get the month, aswell as the year. In addition to that we added a little log entry to show which days have meals selected already.

We then save the days where meals ( MonthMeals) are saved in the Calendar, and give that information over to the CalendarCellAdapter.

The function getDays() is used to create the monthly calendar. Relevant in it, there is the days.add() function, in which we can initialize the entries using the DayCellModel-class again:

If any meals are selected, we save those days:

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Of course you can also navigate to the specific dates being in the monthlyCalendar-View:

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

We therefore use the onItemClick-Function (item referring to the DaycellModel), and grab the designated day, month and year, before using these information to access the DailyCalenders “NewInstance”-Function.

As previously mentioned, the CalenderFragment uses the **CalendarCellAdapter.kt**:

Firstly, we fill each of the Cells with the data and link it to the layout-File:

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

We also define how the icons are coloured. If a meal is set, it has a different colour than if no recipe is selected:

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Considering that it is a Calendar and not every month has the same amount of days, the days that are not existing in this month are invisible, so that only the relevant days are portrayed:

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

If a recipe is then selected, the colour of the icon of the corresponding mealslot is then changed as follows:

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

### 2.1.2: Required Layout-Files of the Calendar-Fragment:

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.

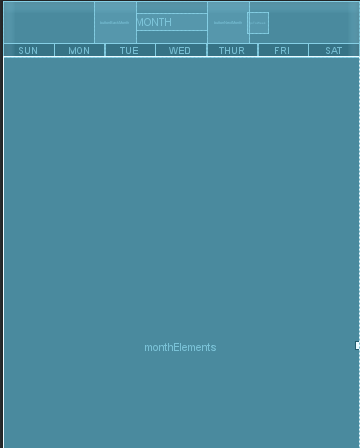
To best explain this, we look at the Breakfast-Row of the CardView-widget:

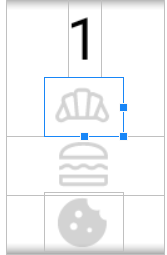
Ein Bild, das Text enthält.

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 the fragment\_calender.xml we have the month presented Using FloatingActionButtons for the navtigation between months. We also have the data sorted in Columns as to better represent the weekdays.

We then use a Recyclerview to fill the MonthElements. For this we use the calender\_cell.xml. In here we have a TextView to handle the date and three Icons representing the specific meals in that date. If a meal has been selected for either of these meals, it will be highlighted as such:

Ein Bild, das Tisch enthält.

Automatisch generierte BeschreibungNow that the montly and weekly views have been discussed, lets take a look at the .xml-Files required for the Daily View. For this view, we have the fragment\_daily\_calender.xml. It consists of a TextView presenting the Current Date, followed by a Textview portraying the Month and year, and lastly a Recyclerview. At the bottom of the Fragment there is also a “Save Meals” Button.



Ein Bild, das Text enthält.

Automatisch generierte BeschreibungNow that we have the structure of it set up, let’s fill the Recyclerview. For this we used the meal\_selection\_card.xml: Each of the entries looks somewhat like this: Now lets discuss the actual functions: We have an Image representing the meal as well as a TextView defining the Meal (In this case Breakfast).

In the next line we have a Spinner, being a sort of dropdown menu, where the user will be able to select the meals from lateron. Furthermore there are two ImageViews, used to represent functions for the actual application. In actuality it will look something like this in the final Version:

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

## 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).

With the continued work on the project, a lot of structures changed in the process. Lets go over the biggest changes:

Ein Bild, das Text enthält.

Automatisch generierte BeschreibungDue to our Recipe-class being used in an unpractical way, we developed an “RecipePreset” class that would initialize the Recipes

We get the Recipes and initialise all of them by grabbing the regarding data and end up with a recipe element containing the actual recipes. We then insert them into the Database:

Besides that, functions have been added adressing key functionalities.

Ein Bild, das Text enthält.

Automatisch generierte BeschreibungLet’s start by talking about the RecipeDetail.kt. This adresses the functionality, that if you click on a recipe, you would be able to see a detailed overview of that recipe:

In this Class, you would then have all of the important information displayed that are required from the fragment\_recipe\_detail.xml file.

In addition to that, there are two buttons included in this fragmen, giving you the option to edit recipes, aswell as add the required ingredients to the shoppinglist.

The editRecipe Function is handled by creating a new Instance with the same uuid and origin: Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Ein Bild, das Text enthält.

Automatisch generierte BeschreibungNow to the shoppingButton:

As the name suggests the general design of it is in a button. The user gets offered with the choices to either Add the ingredients by clicking Yes or click No (which will be shown in a second). If the user clicks Yes, then all the items counted in the “ingredient” tab are grabbed and then added to the ShoppingItem-list.

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung We then access the recipeDao to call addShoppingCard with the Items we just added and add a toast, aswell as a no Button ( teasered earlier).

We also have an “AddRecipe”-class which allows us to add new Recipes to the List:

For this we have different “EditText”-variables … being elements the user can customize: Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Ein Bild, das Text enthält.

Automatisch generierte BeschreibungWe then continue to continue to check for the steps and the ingredients that they are containing information and that they are edited properly:

Now that those are customized properly, we are checking with a validator that none of the fields are empty: Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

And if everything is valid, we add it to the Recipedatabase and give the user a toast to let him know it worked.Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Lastly, there is the EditRecipe.kt . It works in a rather similar way to the AddRecipe, except that an already existing Recipe is being pulled for this:Ein Bild, das Text, drinnen, Screenshot, gestapelt enthält.

Automatisch generierte BeschreibungThe user is then able to – in a similar fashion to before – edit the fields such as title and description – and after he is done and presses the saveRecipe-Button, all of the inputs are validated again and if everything is valid, we let the user know it worked and change the initial recipe by creating a NewInstance of the recipe.Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

### 2.2.2: Required Layout-Files of the Recipe-Fragment:

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.

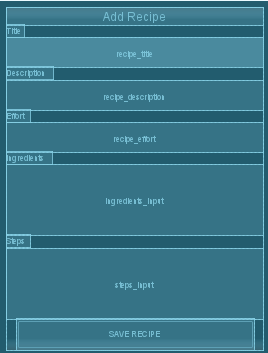
Automatisch generierte Beschreibung

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 BeschreibungThe general object is a CardView. All of the Elements included in the Recipe are inside of a clickable button. This helps the “setOnClickListener”-Function that we discussed earlier.

Considering that once again the AddRecipe-Fragment is quite similar to the EditRecipe-Fragment, the Layout-Files are rather similar aswell. So we will only look at one of them:

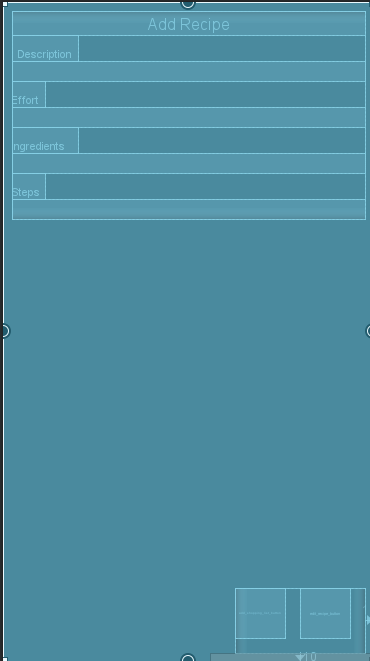
Lets take a second to go over the components included in this: There is the Header serving as a reminder where the user is right now. Then there are several TextViews, each functioning as EditTexts as to give the User further detail to the functionality of the inputs.

At the end of it there is a Button to save the recipe.

Ein Bild, das Text enthält.

Automatisch generierte BeschreibungThis entire layout file is encapsulated in a Scrollview.

The Edit-Recipe-Layout.xml file has the same inputs with the addition of a second button below the “Save Recipe”-Button, being the “Delete Recipe”-Button.

The Recipe\_Detail-Fragment-Layout.xml uses the ScrollView aswell as a RelativeLayout. The TextViews are getting filled with the actual inputs of the recipe (except for the Lines of “Description”, “Effort”, “Ingredients” and “Steps”), and the “AddRecipe” gets replaced with the name of the Recipe selected.

The two Floating Action Buttons on the bottom are inside of a LinearLayout to best align them to the bottom right. Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

They have buttons to better explain their functionality. 

## 2.3: Shoppinglist-Fragment

### 2.3.1: Functionality of the Shoppinglist-Fragment:

While this Fragment relies on the input from Recipes and the Calender, it is initially empty. The User is still able to add ingredients to it by pressing the “NewItemButton” (yellow button bottom right). In the ShoppingListFragment, this button is defined accordingly:Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

For this button we used a Alertdialog.builder:

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

If the User chooses to click Add, his input is added to the Shoppinglist and the Shoppinglist is updated. This update function is defined as follows:Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

The Shoppinglist-Class itself consists of two variables: an Item (String) and a Count (Int). So the Ingredient and the amount. This leaves us with the “Shopping Item” consisting of the id and the Item Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

as a String

Ein Bild, das Text enthält.

Automatisch generierte BeschreibungTo add an Item to the Shoppinglist, we access the addShoppingItem function defined in the recipeDao. We then add a Toast to the process to let the User know, that the ingredient has been added. We also added a Undo button if the user wants to Undo his last action and update the Shoppinglist.

To remove an Item, the onSubItem function is accessed, working in a similar way as the previously defined “Undo” action aswell as having a Toast added to it.

Ein Bild, das Text enthält.

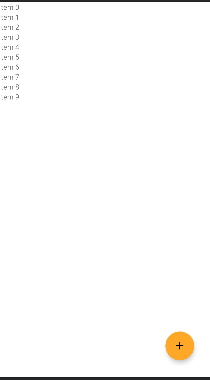
Automatisch generierte Beschreibung

In the ShoppingListFragment.kt those two functions are handled by holders with listeners attached to them

### 2.3.2: Required Layout Files of the Shoppinglist-Fragment:

The Shoppinglist consists of the “fragment\_shoppinglist.xml” general Fragment and the “card\_shoppinglist.xml” being the entries in the list.

Let’s start off with the fragment\_shoppinglist.xml:

This Fragment uses a Recyclerview similar to the ones we have discussed in the Calender and the Recipe-Fragments before. In addition to that, it has a FloatingActionButton at the bottom right.

The card\_shoppinlist.xml uses the CardView, something weve been familiar with from the Calender-Fragment already.



It consists of an Icon, The Itemname aswell as the two buttons, being the “item\_sub\_qty” and the “item\_add\_qty”, aswell as a TextView in the middle representing the current Item-Quantity.

# 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.