POLITECNICO
MILANO 1863

Design Document

Sous-chef

Design and Implementation of Mobile application
Academic year 2023 - 2024

19 January 2024
Version 1.0

Authors:
Alzbeta Fekiacova
Christian Lisi

Contents

0.1 Glossary

0.1.1 Acronyms

Acronym	Meaning
FR	Functional Requirement
API	Application programming interface
FAB	Floating action button

0.1.2 Abbreviations

Abbreviations	Term
e.g.	Exempli gratia
i.e.	Id est
w.r.t.	With reference to

0.2 Document Structure

1. ***Introductory***: This part provides an initial understanding of the project's aims and extent. It describes the application purpose, scope and provide an overview of its functions.

2. ***Architectural Design***: This section presents a high-level view of the main components and their interactions. It also contains sequence diagrams of actions that the application offers.

3. ***User Interface Design***: This chapter describes the envisioned appearance of the User Interface for supported devices.

4. ***Testing campaign:*** Description of the Sous Chef app testing process and its coverage.

Chapter 1

Introduction

1.1 Purpose

The aim of this documentation is to inform the reader about the Sous Chef App, designed to empower users in managing their kitchen ingredients with finesse. It goes beyond mere ingredient tracking, offering users tailored recipe recommendations based on the contents of their virtual pantry. Moreover, the app takes a proactive approach by providing timely notifications about impending ingredient expiration dates. The overarching goal is to enhance the culinary journey for users by facilitating optimal utilization of the ingredients at their disposal.

1.2 Scope

The app's scope encompasses a rich array of features, including user registration, virtual pantry creation, personalized recipe recommendations, and proactive notifications for ingredient due dates. The primary thrust is to elevate the cooking experience, enabling users to make the most of their available ingredients. For the implementation, the application exclusively targets Android users and will be developed using the Kotlin programming language.

1.3 Requirements

The following list contains the requirements that the app should satisfy:

FR1	User should be able to sign in/sign up in the app
FR2	User should be able to login to the app
FR3	User should be able to log out from the app
FR4	Upon registration user should be able to select their diets
FR5	Upon registration user should be able to select their allergies
FR6	Logged-in user should be able to add ingredients to their virtual pantry
FR7	Logged-in user should be able to add an expiry date for an ingredient they want to add to the virtual pantry
FR8	Logged-in user should be able to remove ingredients from their virtual pantry
FR9	Logged-in user should be able to search for an ingredient to add to the virtual pantry
FR10	Logged-in user should be able to check the list of ingredients in their virtual pantry
FR11	Logged-in user should be able to check their profile info
FR12	Logged-in user should be able to edit their personal info
FR13	Logged-in user should be able to check the list of available recipes based on their available ingredients
FR14	Logged-in user should be able to check the info of an available recipe
FR15	Logged-in user should be able to select the ingredients they finished after cooking a recipe
FR16	Logged-in user should receive push notification if any ingredients are close to their expiry date

1.4 Features Implemented

The functionalities implemented in the app, trying to follow a possible order of interaction, are:

1.4.1 Login and Registration

Users can sign up (or log in) by using the email address. A “forgot password” protocol is implemented to reset the password when needed. User is able to click on ”Forgot your password?” text to trigger sending of e-mail to reset his password. Also, if the user submits wrong password 3 times, the reset password e-mail is sent. The registration contains three steps, first one requires user to enter his personal data, second to enter his allergies and third one his dietary choice.

1.4.2 Edit personal information

The user can edit his diet/allergies at any time : From the profile page, accessible from the navigation bar at the bottom of the screen. He can see all his present information and if needed, he can modify it by choosing from available options.

1.4.3 Search Ingredients

Logged-in users have access to an extensive list of ingredients accessible from the *SearchIngredient* page. A search bar is available at the top, which can be used to query the system. After querying the system, the app will show a list of ingredients based on the query of the user. Tapping the ingredients card leads to the *AddIngredient* page.

1.4.4 Add ingredients to virtual pantry

Logged-in users can add ingredients to their virtual pantry from the *AddIngredient* page. The user have also the possibility to specify the expiry date of the ingredients so that the app will notify him if we are getting close to that date (3 days) and the ingredients are not yet been used. If the user wants to add an ingredient without a due date, he is informed via pop up window, that notification system would not work when he adds an ingredient without due date.

1.4.5 Retrieve available Recipes

Logged-in user are welcomed by the home page that automatically retrieve the available recipe based on the ingredients in his virtual pantry. In doing so, we reduce the number of interaction needed and present immediately the info the user really wants. Tapping on a recipe card leads to the *RecipeInfo* page.

1.4.6 View recipe's info and complete a recipe

Logged-in user, upon selecting a recipe from the home page, are welcomed by the *RecipeInfo* page. Here, an overview of the recipe with: title, list of ingredient, servings and information about the minutes needed to complete the recipe are displayed. Is also available a step-by-step instruction list that the user can follow in order to cook the recipe. After the cooking is complete, the user can tap on the complete button and notify the system the ingredients that are finished and need to be removed from the virtual pantry.

Chapter 2

Architectural design

In this chapter, we will introduce the architecture of the application in more detail.

2.1 Overview

We have decided to build SousChef as a native android app using Kotlin. Kotlin is a programming language that is officially supported for Android development by Google. It is a modern, concise, and expressive language that is interoperable with Java, the traditional language used for Android development. For the development of the application, 2 different services are used:

- **Recipe Services**: APIs related to the food and recipe data.
- **Firebase Services**: APIs used for user management and for data storage.

The integration of these services is implemented by exploiting asynchronous communication protocols when possible that allows the application to be as responsive and smooth as possible regardless of the performance of the external services.
Due to the application's heavy dependence on these two services, the internet connection is mandatory for using the application and is subject to various checks during its execution.
The internal structure of the app is divided into services, models, screens and viewmodels following the MVVM design pattern.

Model-View-View Model (MVVM) is a software architectural pattern that is commonly used in Android app development. It is specifically designed to work with data-driven user interfaces. The main idea behind MVVM is to separate the logic of the user interface from the underlying data model.

For the sake of clarity and to better match the pattern, the same structure is kept for the source code file structure.

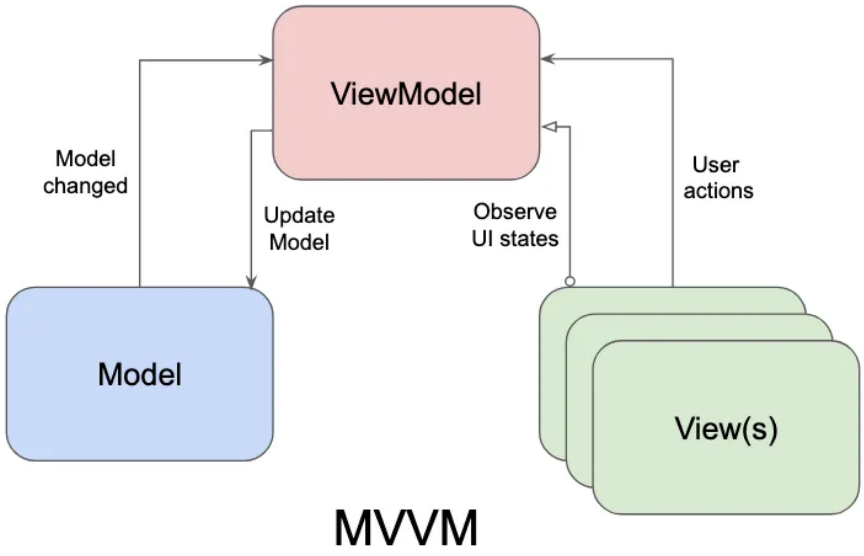


Figure 2.1: MVVM pattern

2.2 Data Model

Data shown in the application pages has been structured using classes which represent the Model part of the MVVM pattern previously illustrated. By following the app requirements (1.3) and designing the view of each page, we ended up defining the following objects:

- *User*, *Allergen* and *DietType*, which include info of registered users and details of their status.
- *Recipe*, which includes data of a specific recipe.
- *Ingredient*, which includes data of a specific ingredient
- *Instruction* and *Analyzed Instruction*, which define the structure of the step by step instructions related to a recipe.
- *Result*, which defines the structure of a general result from an API request.

Data related to food is fetched from the Spoonacular API and both the fetch and parse of the data is done by employing the RetroFit library in order to painlessly obtain the desired structure.

2.3 Data Management

2.3.1 Cloud Firestore

Cloud Firestore is a NoSQL document-based database that stores all the relevant data displayed in the application. The internal structure of the collections consists of a single collection users, meaning each user is represent by one document. The document is created after successful user registration named by unique user ID with this structure:

uniqueID: email, first name, last name, allergies, type of diet and ingredients in virtual pantry.
Pantry and allergies are two arrays.
When the user requests to add an ingredient to his virtual pantry, after checking if not already present, the ingredient will be appended to the array. If it is already present, the due date of the ingredient is modified to a new one.

2.4 External Services

2.4.1 Firebase Authentication

Firebase Authentication provides back-end services to authenticate users to our app. It has been used to support authentication using email and password which are then stored when a user completes the registration phase, together with a unique generated ID.

2.4.2 Spoonacular API

Spoonacular API provides access to various food-related information, such as recipes, nutrition information, meal planning, and more.
Data is retrieved through an HTTP GET call towards a plethora of API endpoints during the app execution. In our case, the endpoint used are:

1. **recipes/findByIngredients** : retrieve basic recipe information from the available ingredients. For the request this parameters are needed:
   - *ingredients* : A comma-separated list of ingredients that the recipes should contain.
   - *ranking* : Whether to maximize used ingredients (1) or minimize missing ingredients (2) first.
   - *number* : The maximum number of recipes to return.
   - *ignorePantry* : Whether to ignore typical pantry items, such as water, salt, flour, etc.
2. **/recipes/informationBulk** : Retrieve all information about multiple recipes at once.
   - *ids* : A comma-separated list of recipe ids.

- *includeNutrition* : If include nutrition data to the recipe information.

3. **/food/ingredients/id/information** : Use an ingredient id to get all available information about an ingredient, such as its image and supermarket aisle.

4. **/food/ingredients/search** : Retrieve all possible ingredient based on the query passed.

   - *query* : The partial or full ingredient name.
   - *number* : The maximum number of ingredients to return.

2.4.3 Firebase Cloud Messaging

Firebase Cloud Messaging is a cross-platform messaging solution that allows reliably to send messages at no cost. In this application, it is the service used to send push notifications to users that have enabled them. This happens thanks to an HTTP POST call towards the Firebase Cloud Messaging (FCM) API that will take care of delivering the built message to the devices specified in the body of the request (fields are title and body of message and list of devices' tokens).

2.5 Notification Implementation

Implementing push notifications via Firebase Cloud Messaging (FCM) is crucial for delivering timely updates and information to users, even when they're not actively using the application. This is achieved by leveraging a worker, which operates independently of the app's user interface, ensuring seamless notification delivery without impacting the app's performance.

2.5.1 The Role of the Notification Worker

The notification worker is responsible for managing the process of sending push notifications to users based on certain criteria, such as upcoming expiration dates of ingredients in the user's pantry. It operates on a periodic basis, triggering every 24 hours to ensure timely updates to users. Moreover, the periodic worker has an initial deadline set to be trigger at 10.A.M. every day to ensure the user receive their notification early in the day.

2.5.2 Worker Logic

Upon triggering, the notification worker initiates its tasks with an empty list. Its primary task involves asynchronously fetching the user's pantry data and analyzing the expiration dates of the ingredients. This asynchronous operation allows the worker to efficiently handle potentially large datasets without blocking the main application thread.

The worker then evaluates each ingredient's expiration date to determine if it falls within the upcoming days, specifically today, tomorrow, or the day after tomorrow. For each ingredient meeting these criteria, the ingredient is added to the array, whose length indicates the number of ingredients with upcoming expiration dates.

2.5.3 Notification Sending

Once the asynchronous checks are complete, the worker evaluates the length of the array. If the array is not empty, indicating that there is at least one ingredient with an upcoming expiration date, a notification channel is established. This notification channel is essential for ensuring that notifications are properly categorized and displayed on the user's device. Additionally, it's important to note that the user's device must have the required version of Android to receive these notifications as starting from Android 8.0 (API level 26) notification channels have been introduced. Once the notification channel is set up, a notification is constructed and sent to the user's device.
When the user installs the app and finishes the registration, he is asked to allow receiving notifications. If he allows, then he is subscribed to the Sous Chef app notification channel. Receiving these notifications provide valuable information about the number of ingredients nearing their expiration dates, prompting the user to take necessary actions, such as planning meals or restocking pantry items.

2.6 Dependencies

The most significant dependencies packages included in the application are listed below:

retrofit2	REST client which is used to turn data from the API into callable objects
okhttp3	efficient HTTP & HTTP/2 client with advanced features
firebase-auth	Used for authentication with email and password
firebase-firestore	Necessary to use the Firestore storage
firebase-messaging-ktx	Necessary to use the Firebase Cloud Messaging API
glide	Fast and efficient image loading library focused on smooth scrolling

2.7 Screen Architecture

In this section, the general component organization of the screens is shown. The use of this component framework is used to better generalize on different kinds of screen size and help with code structure for better reuse of it.

As can be seen in the User Interface Paragraph (3), some of the tablet screens are composed of different component positioned side by side rather than in column like the smartphone version. To implement this design, we have chosen to follow the above component framework: a scaffold with a navigation bar that remains constant in most of the pages,

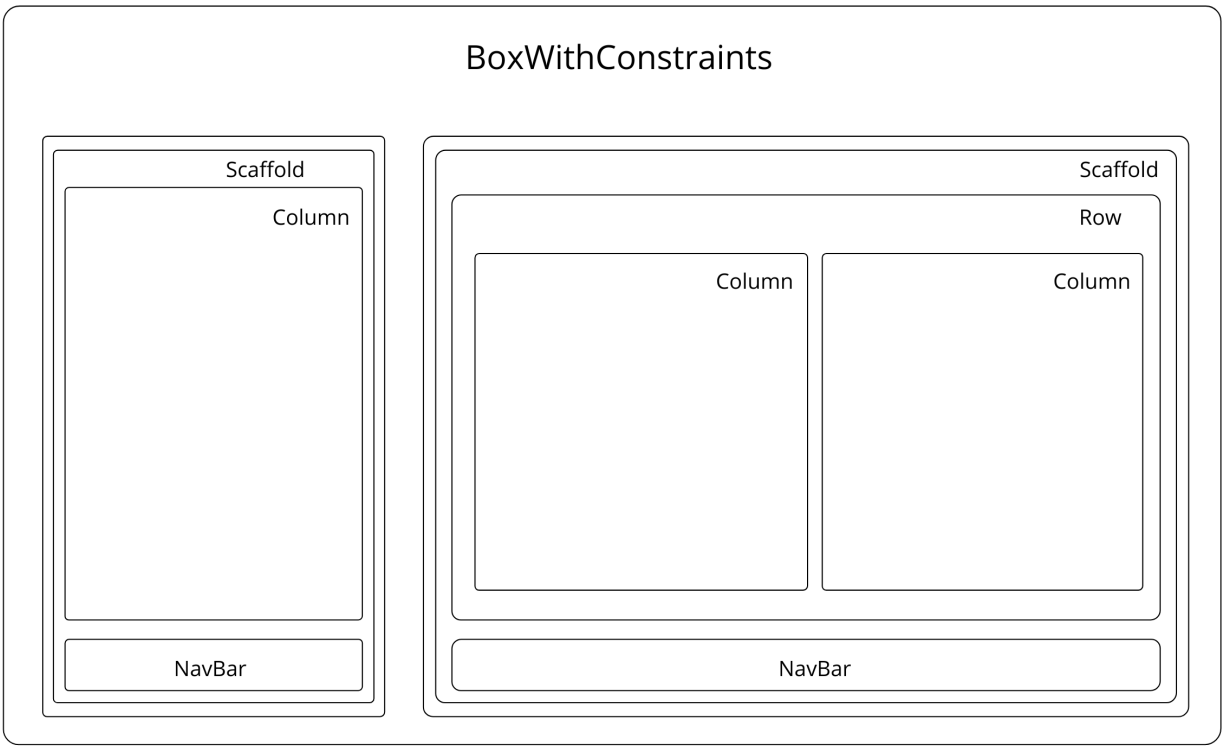


Figure 2.2: Screen philosophy

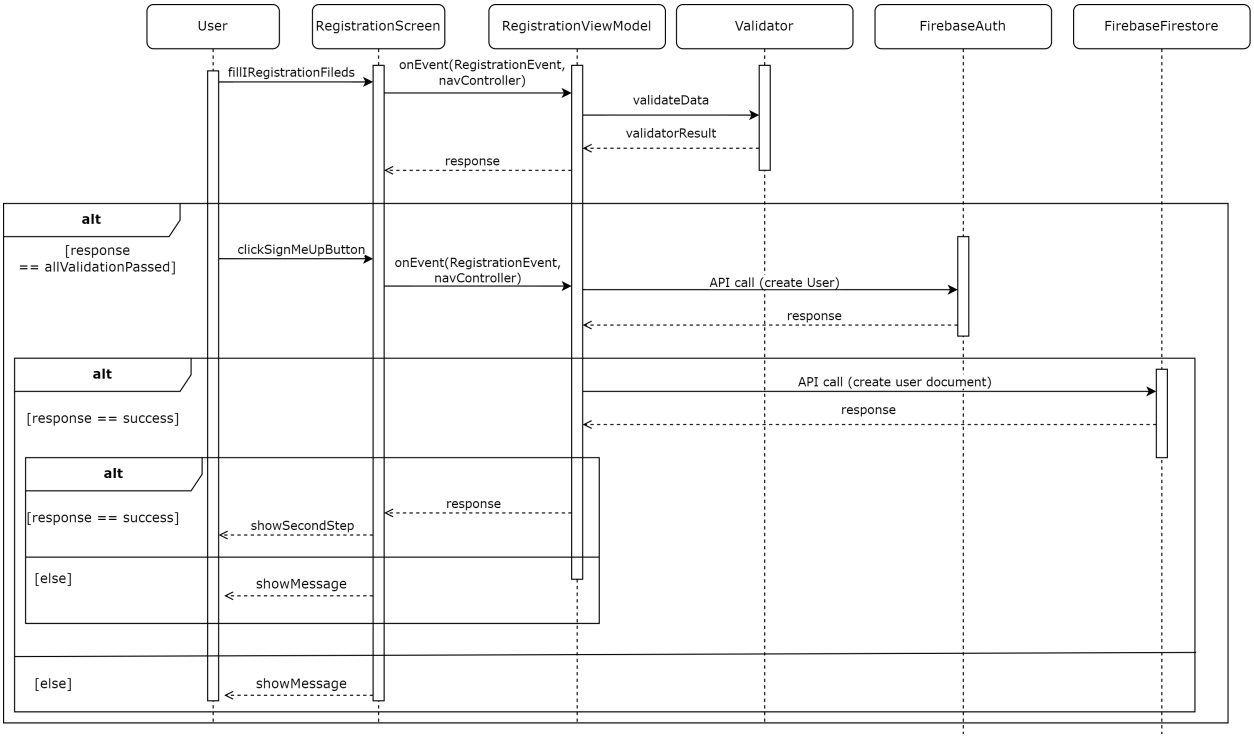
and a column that includes all the main components used in the screen. Both smartphone and tablet pages are then wrapped in a BoxWithConstraints component that allows us to select the right design dynamically according to the screen size and in a transparent way with respect to the design of the page. The choice of wrapping the whole page and not only the different components was made in order to achieve an optimal trade-off between reuse of the code and design flexibility.

2.8 Sequence Diagram

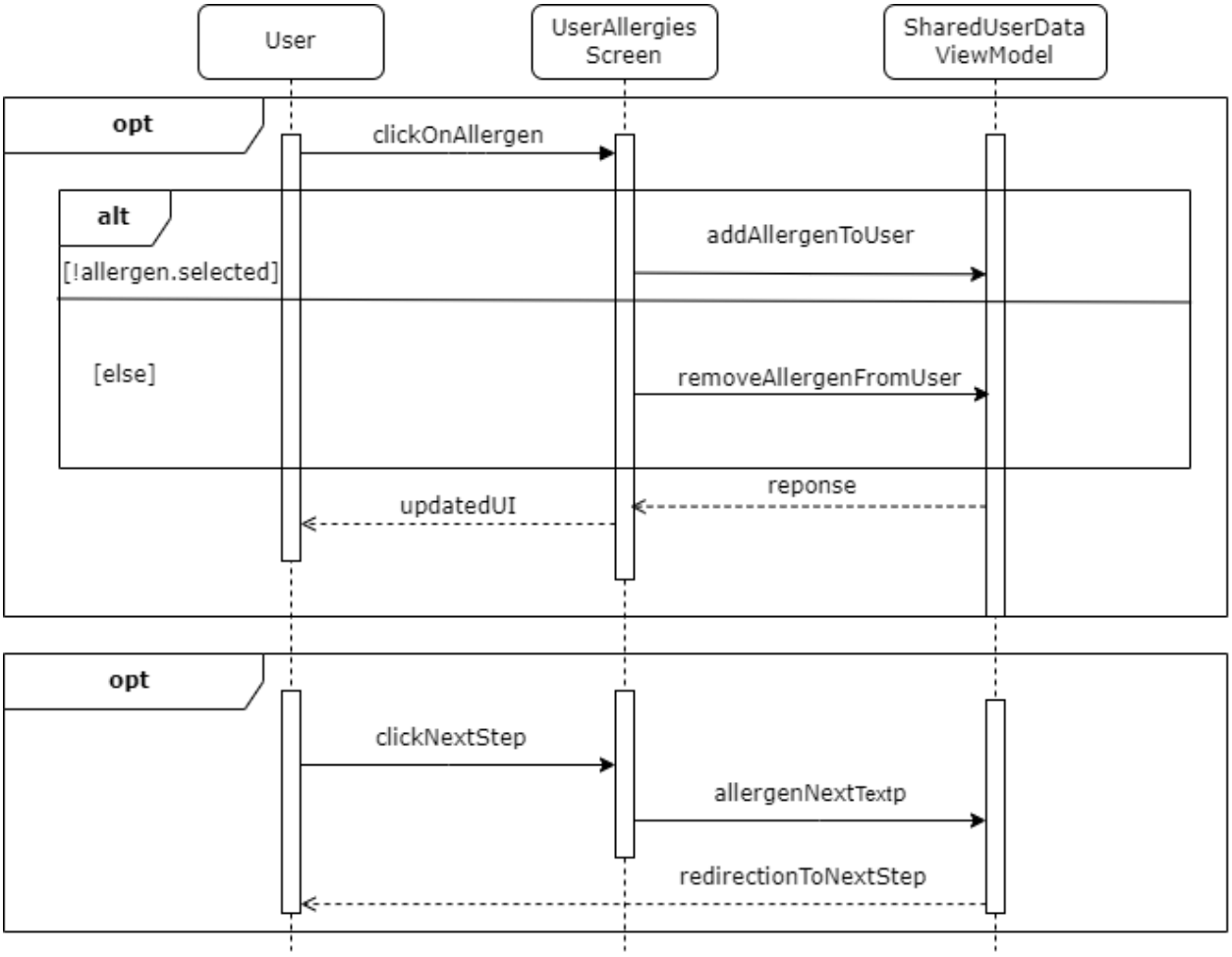
Sequence diagrams offer a visual representation of how different components or objects interact and cooperate within our system; these diagrams provide a clear understanding of the system's behavior, communication patterns, and the responsibilities of each element. For these reasons, the sequence diagram of the most relevant actions of our application is shown.

2.8.1 Registration

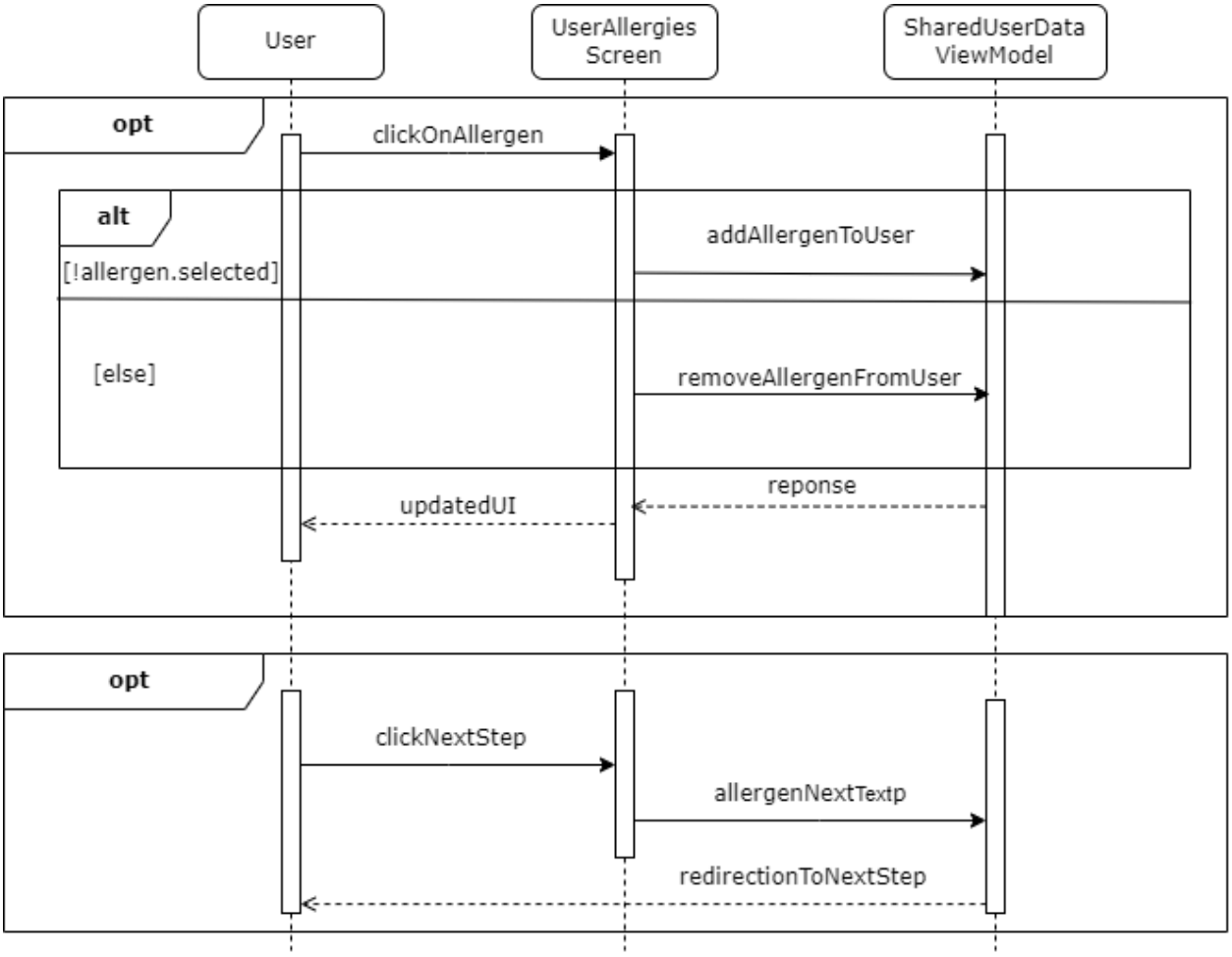
First step



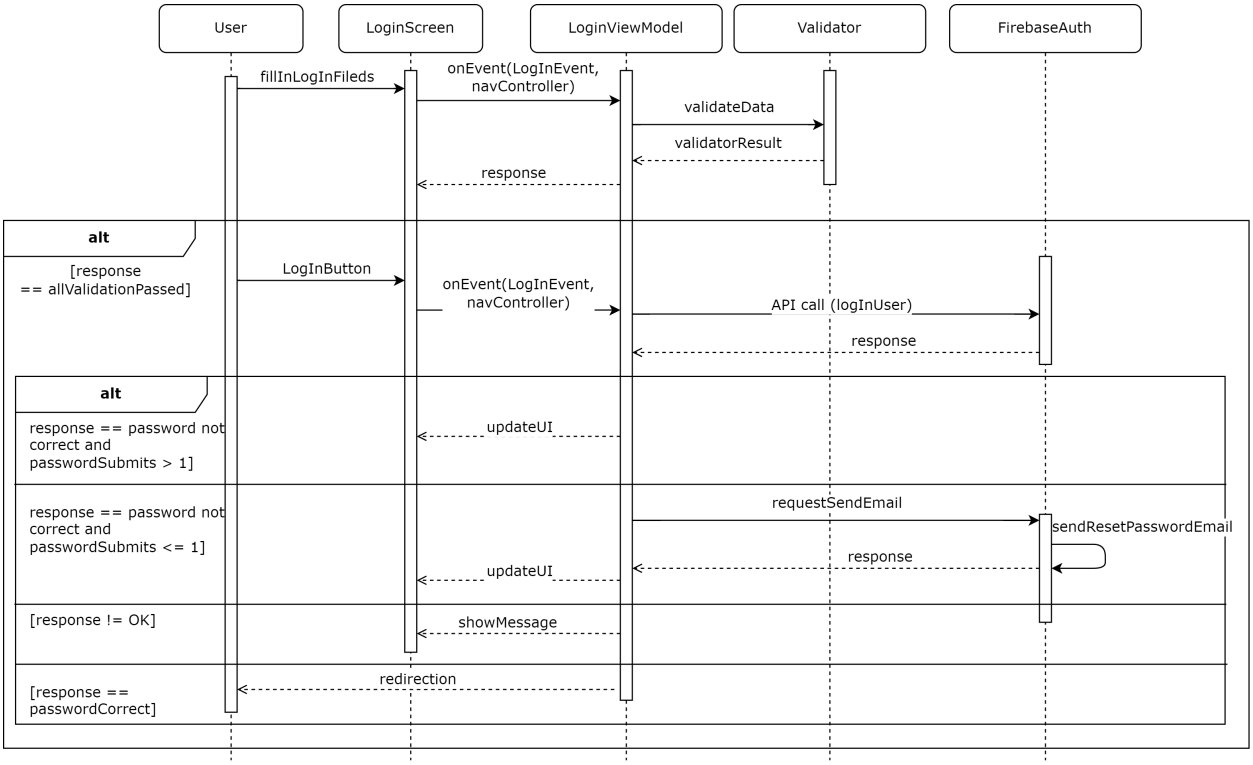
Second step


User
UserAllergies Screen
SharedUserData ViewModel
opt
clickOnAllergen
alt
addAllergenToUser
[!allergen.selected]
[else]
removeAllergenFromUser
reponse
updatedUI
opt
clickNextStep
allergenNextTextp
redirectionToNextStep

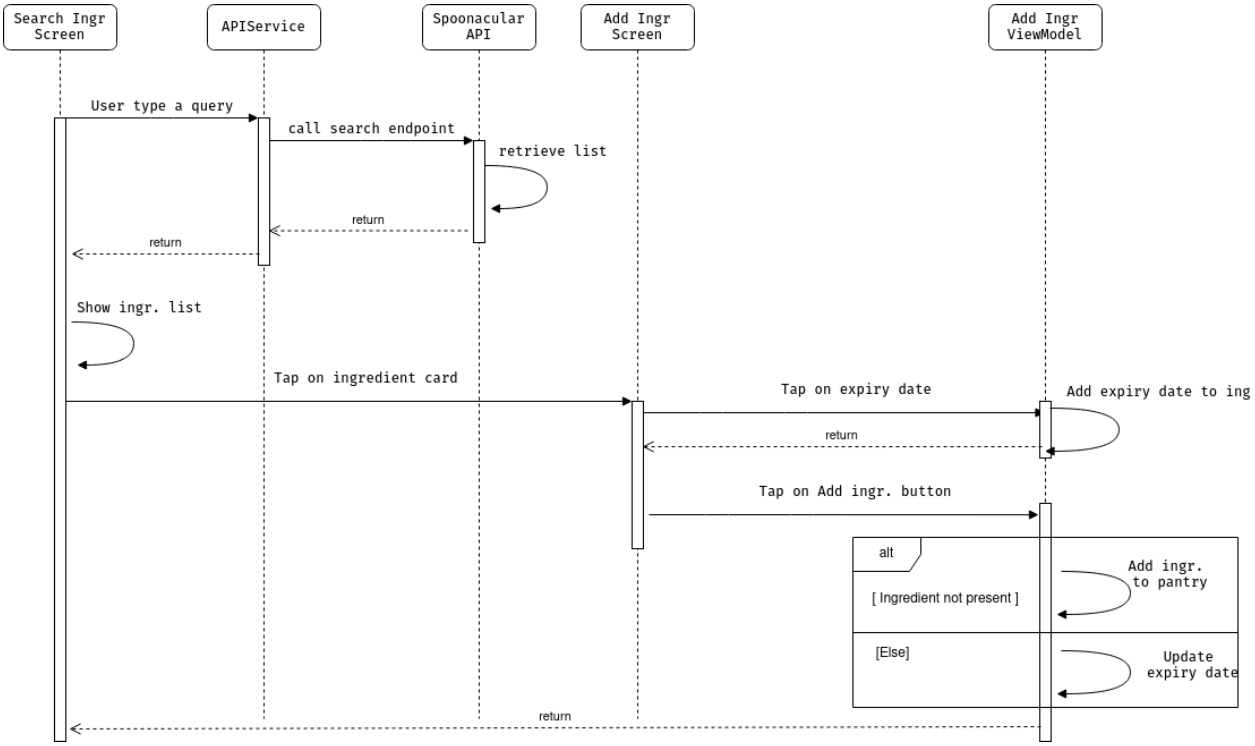
Third step


User
UserAllergies Screen
SharedUserData ViewModel
opt
clickOnAllergen
alt
addAllergenToUser
[!allergen.selected]
[else]
removeAllergenFromUser
reponse
updatedUI
opt
clickNextStep
allergenNextTextp
redirectionToNextStep

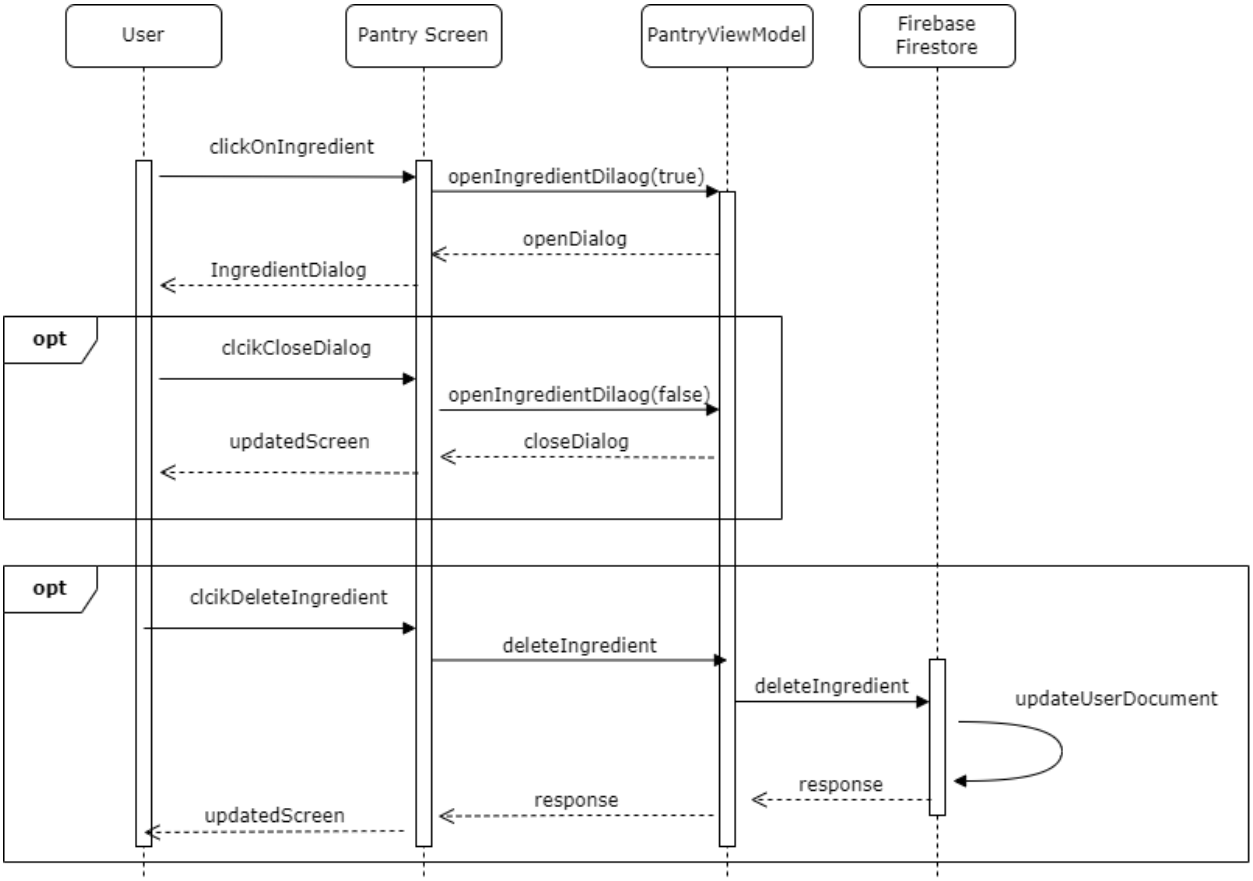
2.8.2 Login


User
LoginScreen
LoginViewModel
Validator
FirebaseAuth
fillInLogInFileds
onEvent(LogInEvent, navController)
validateData
validatorResult
response
alt
[response == allValidationPassed]
LogInButton
onEvent(LogInEvent, navController)
API call (logInUser)
response
alt
response == password not correct and passwordSubmits > 1]
updateUI
response == password not correct and passwordSubmits <= 1]
requestSendEmail
sendResetPasswordEmail
response
updateUI
[response != OK]
showMessage
redirection
[response == passwordCorrect]

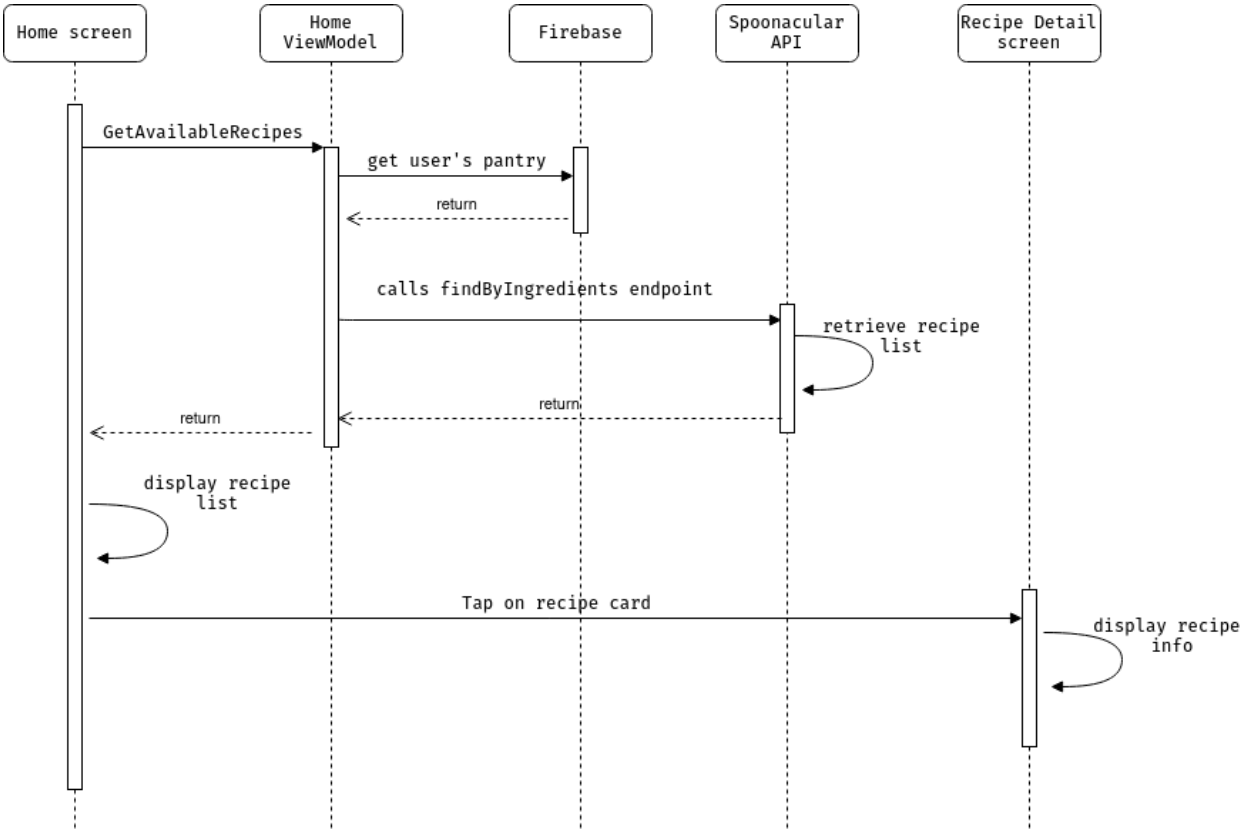
2.8.3 Search and add ingredient



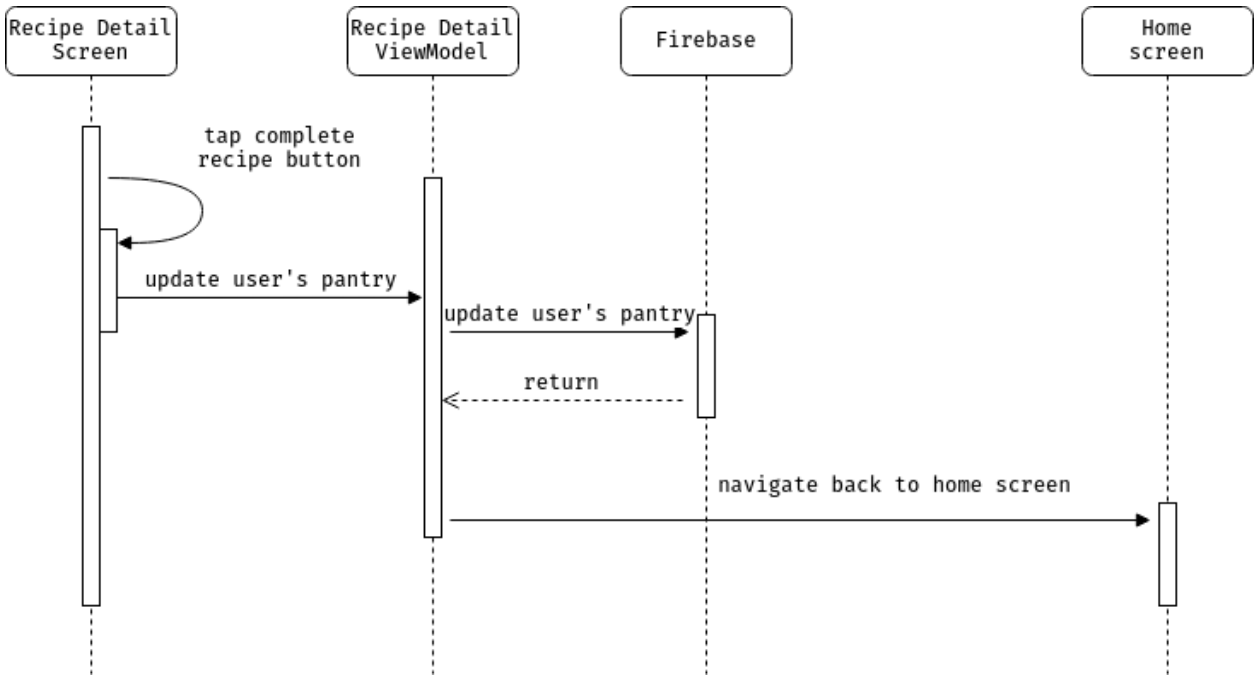
2.8.4 Delete an ingredient from pantry



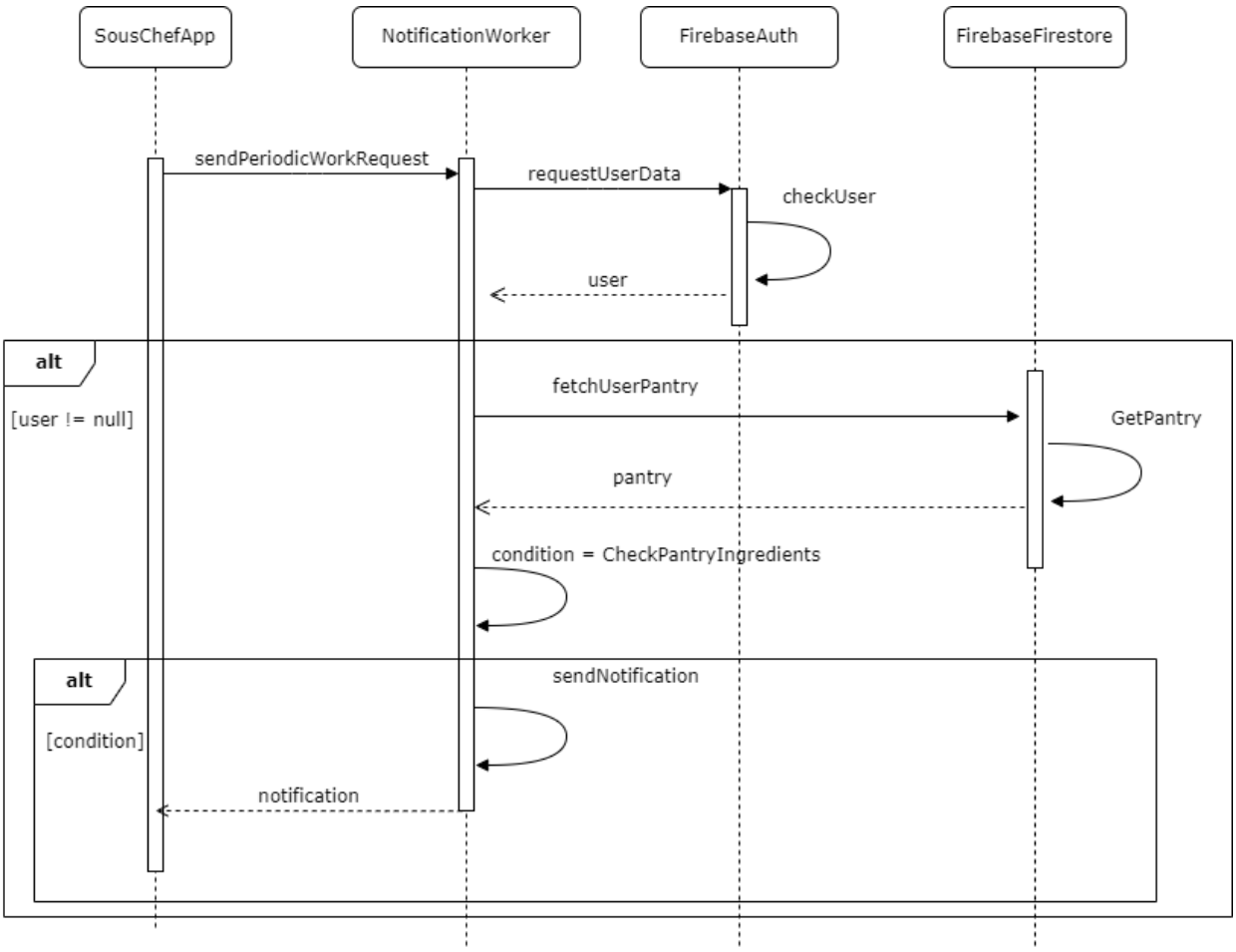
2.8.5 Get available recipes



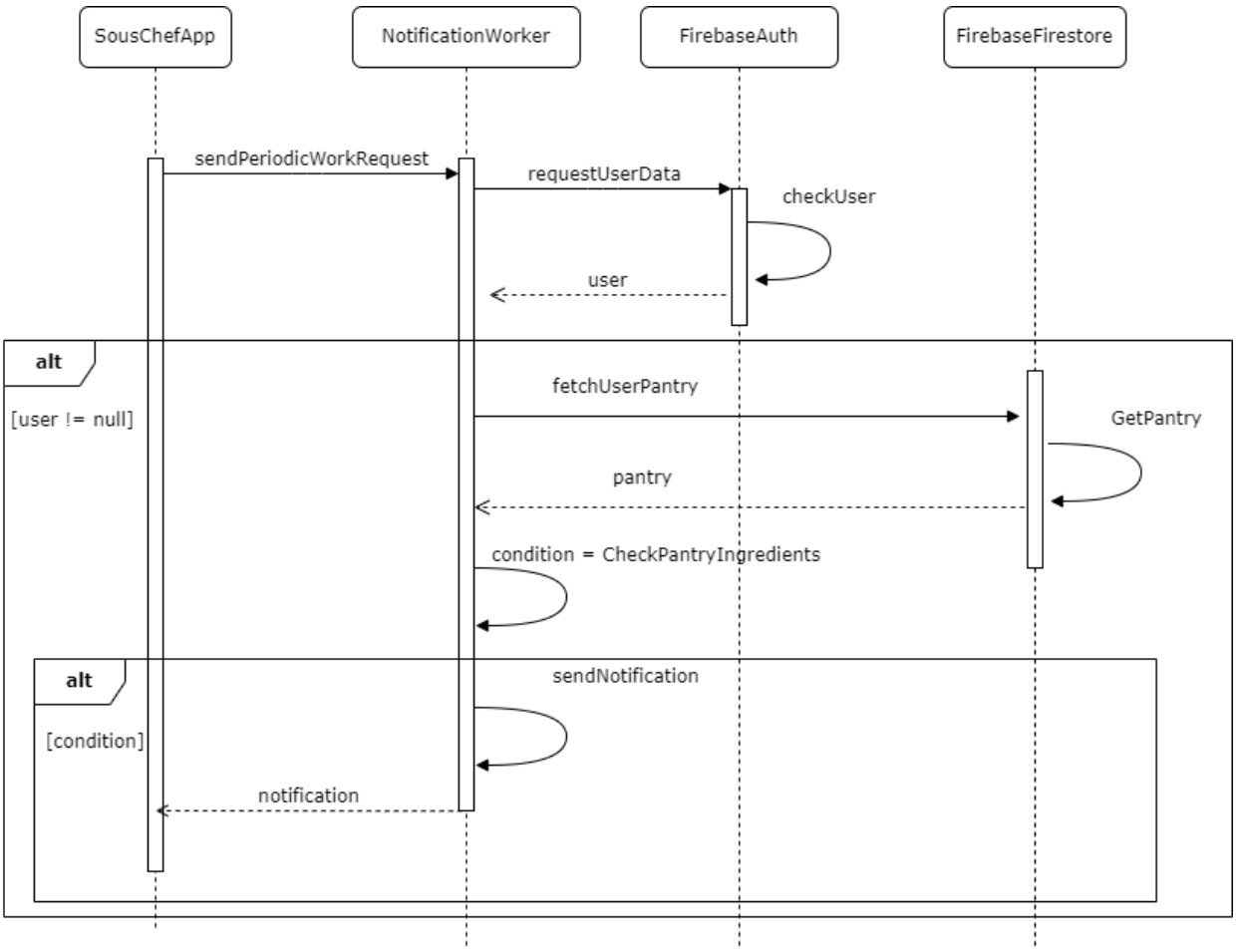
2.8.6 Complete a recipe



2.8.7 Notification



2.8.8 Delete ingredients when recipe is completed



Chapter 3

User Interface design

In this chapter, we present the design of the Sous chef app screens. Our design inspiration draws from an existing and well-crafted design available on Figma. You can explore the original design here on the official site.

The design we have chosen incorporates several key features that caught our attention. Elements such as sign-up flow and color design have influenced our approach in crafting the Sous chef app screens.

While drawing inspiration from the Figma design, we have customized and adapted elements to align with our requirements and branding of the Sous chef app. This ensures a tailored and cohesive user experience for our app users.

We would like to extend gratitude to the original designers behind the Figma Cooking App UI for their work. Their design has inspired us in designing the Sous chef app screens, and we appreciate their creativity.

The logo and icon of our application has been made via Canva[1], where we also made icons for the different types of diets. Other icons that were used are from a web page containing multiple icons free to use [2].

Most of our application screen composition depends on the device screen size. When the screen is wider (e.g. tablet), the user is able to see more.

[1] https://www.canva.com/

[2] https://www.svgrepo.com/

3.1 Screens preview

3.1.1 Welcome Screen

Welcome screen of the Sous Chef app. It displays the logo and a button leading the user to sign up screen.

Figure 3.1: Mobile preview

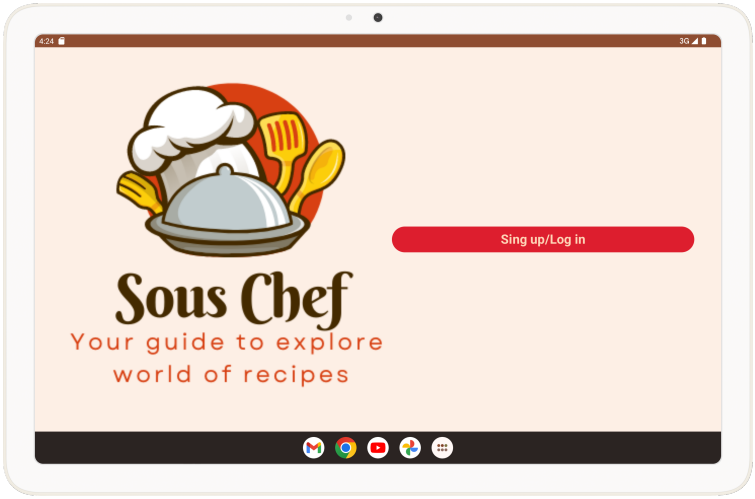


Figure 3.2: Tablet preview

3.1.2 Registration Screen

The registration screen of the Sous Chef app. It contains fields for first name. last name, e-mail and password, all the fields are required. There is a button to perform registration and clickable text to redirect to log in screen. The tablet view also contains the logo of the application on the left side.

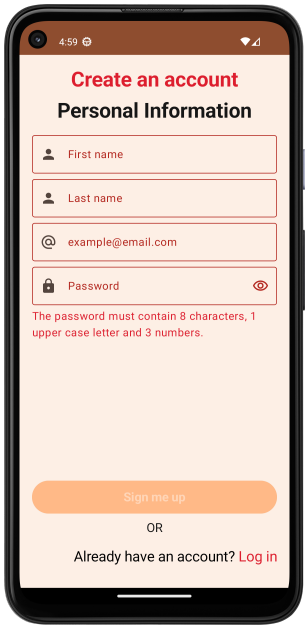


Figure 3.3: Mobile preview

Figure 3.4: Tablet preview

3.1.3 Allergens and Diet Screen

The registration process of the Sous Chef app has three steps. First step is filling in the personal data, then the user is asked to fill in his allergies and diet type.(User can choose to select none of the allergies)

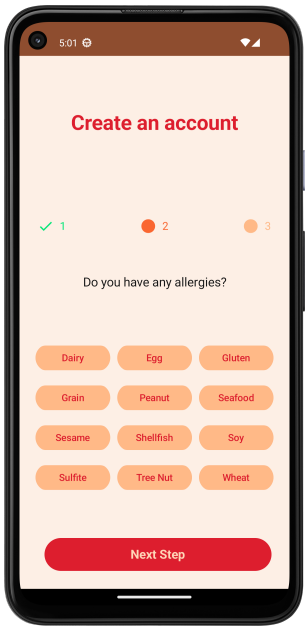


Figure 3.5: Mobile preview

Figure 3.6: Tablet preview

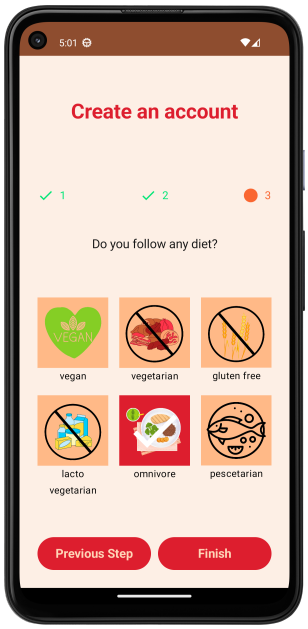


Figure 3.7: Mobile preview

Figure 3.8: Tablet preview

3.1.4 Login Screen

The login screen of the Sous Chef app. It contains fields for e-mail and password. It also contains two clickable texts, one to request password reset, the other one to redirect to the sign-up screen. Same as registration screen, it contains a logo of the application on the left side.

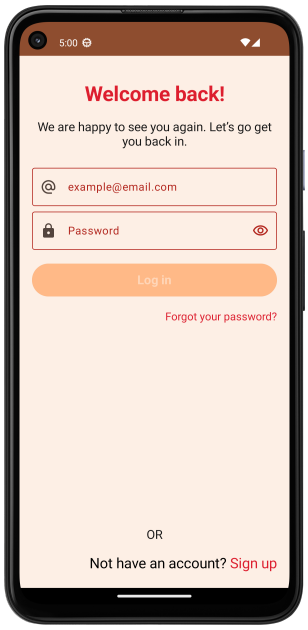


Figure 3.9: Mobile preview

Figure 3.10: Tablet preview

3.1.5 Home Screen

Home screen is the main page of the Sous Chef app, and it displayed when the user is logged in. It contains a bottom navigation bar, from which the user can access other screens with ease. The content of the view depends on the user's pantry. As soon as the screen is open, it retrieves all the ingredients the user has in his pantry and query the Spoonacular API retrieving the available recipe. If none is available, the user is notified with a message.

Figure 3.11: Mobile preview

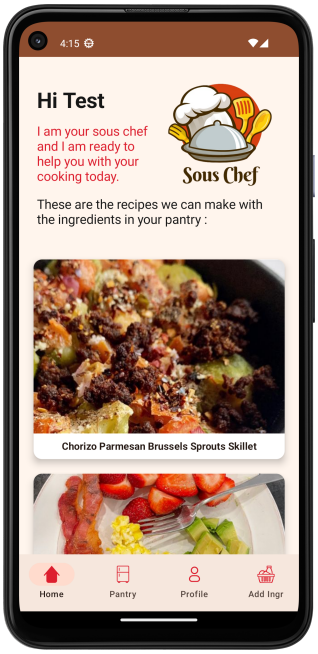


Figure 3.12: Tablet preview

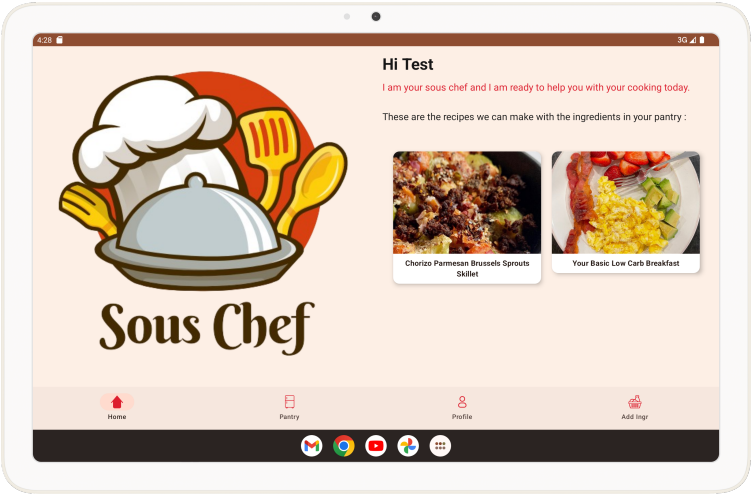


Figure 3.13: Tablet preview

3.1.6 Profile Screen

In the Profile screen, user can see and also modify information about him. The information about his diet and allergies can be modified at any time by tapping on the respective button. The screen displays a dialog box where the user can select new information to be entered into the system. It also contains Logout button.

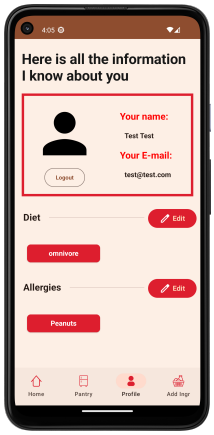


Figure 3.14: Mobile preview

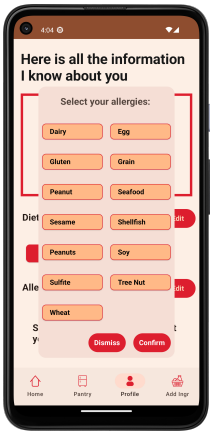


Figure 3.15: Mobile preview

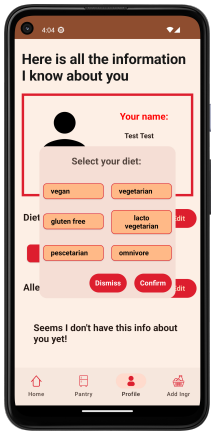


Figure 3.16: Mobile preview

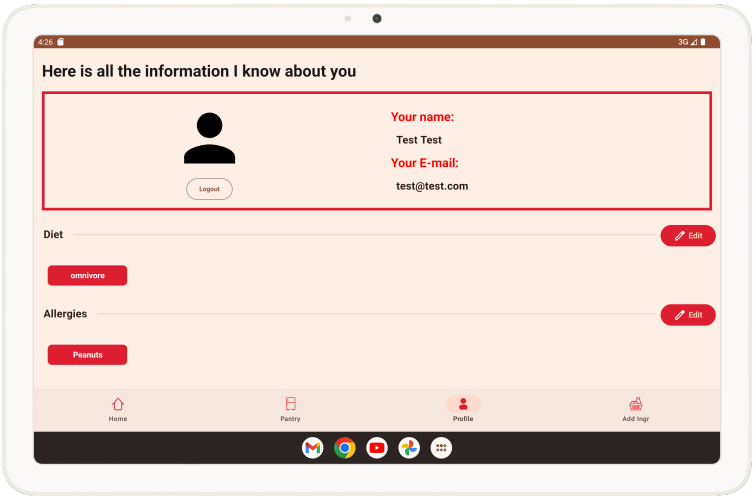


Figure 3.17: Tablet preview

3.1.7 Search ingredient screen

In the Search ingredient screen, the user can search for an ingredient to add to his virtual pantry. By querying the system using the search bar, the view will call the right Spoonacular API, get all the ingredients that match the query entered and populate the grid. By tapping on the ingredient card, the add ingredient screen is shown.

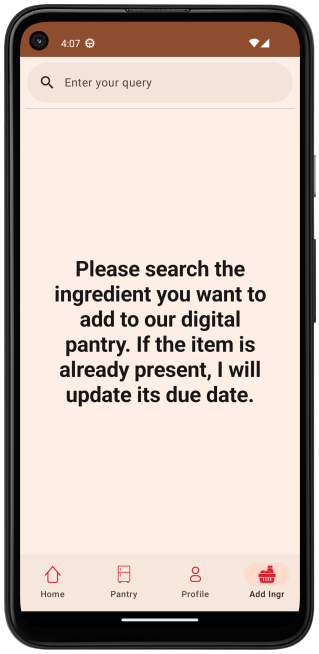


Figure 3.18: Mobile preview

Figure 3.19: Tablet preview

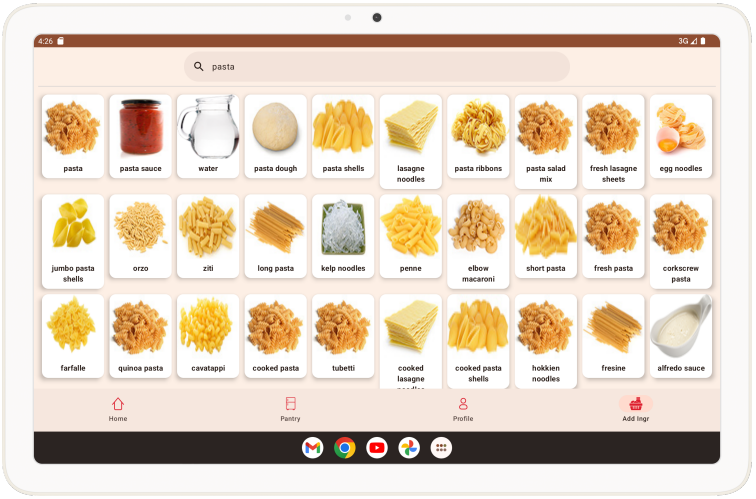


Figure 3.20: Tablet preview

3.1.8 Add Ingredient Screen

This Add ingredient screen shows the name and image of the ingredient selected. The user can also choose to inform the app of the ingredient expiry date by tapping on the dedicated button and select the date from the dialog that the app will present him. Doing so, the app will be able to informs the user (via a notification) if any ingredient in his pantry is near its expiration date.

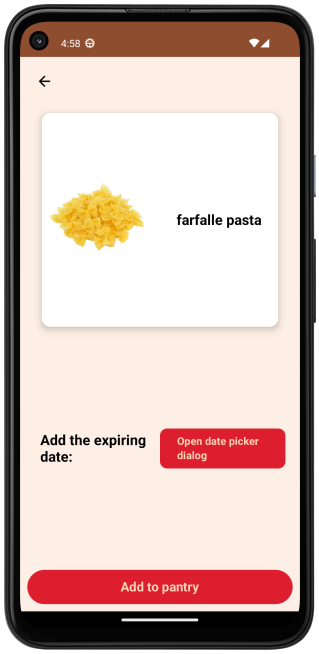


Figure 3.21: Mobile preview

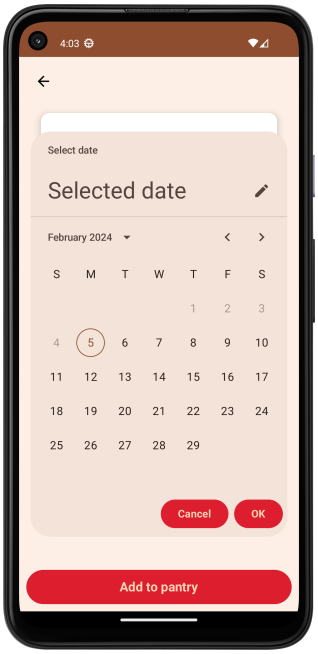


Figure 3.22: Mobile preview

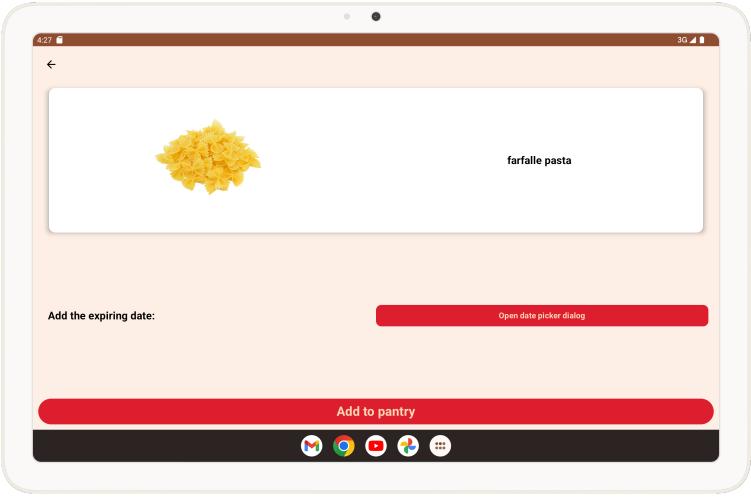


Figure 3.23: Tablet preview

3.1.9 Pantry Screen

The pantry screen shows the list of ingredients in the user's pantry, with due dates (if available). If the pantry is empty, the screen show an appropriate message to the user.

Figure 3.24: Mobile preview

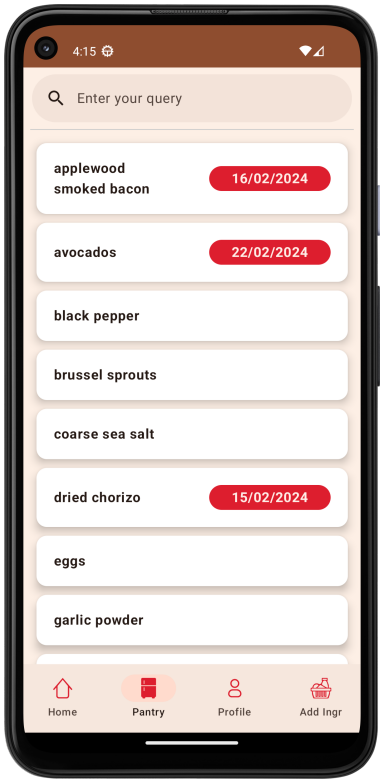


Figure 3.25: Mobile preview

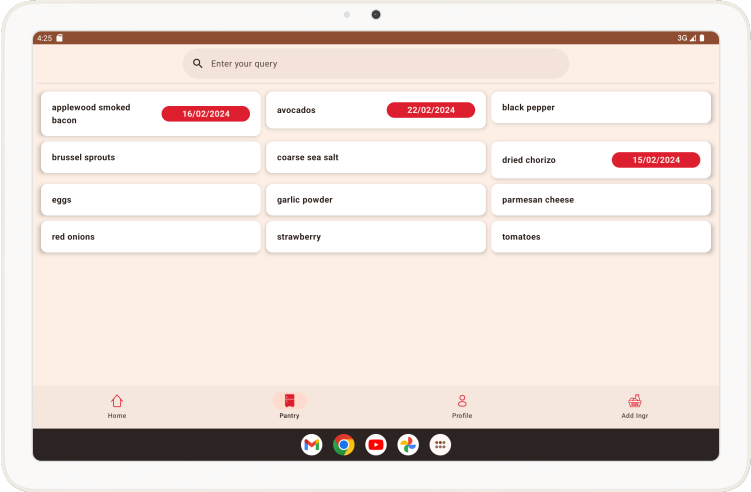


Figure 3.26: Tablet preview

The click on ingredient in pantry opens a dialog providing option to delete the ingredient from the pantry.

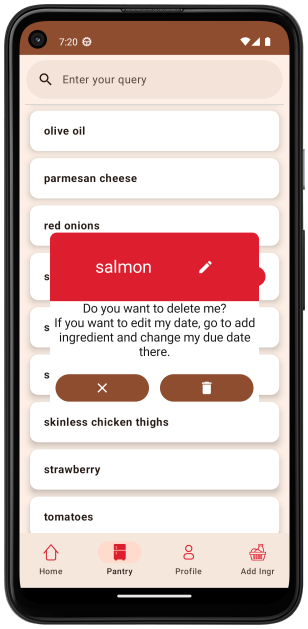


Figure 3.27: Mobile preview

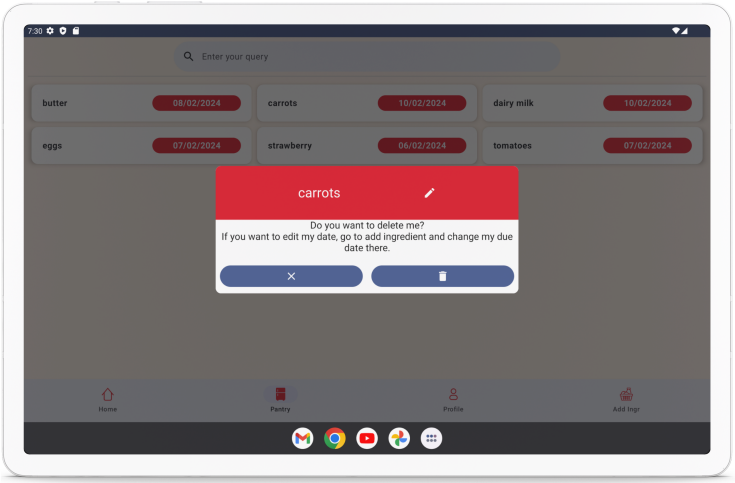


Figure 3.28: Tablet preview

3.1.10 Recipe detail Screen

The recipe detail screen displays all the information necessary for successful cooking the recipe with step-by-step instructions. By tapping the Expandable FAB button at the bottom right, the user notifies the app that he has finished cooking. Then the app ask the user - via a dialog - to select which ingredient he has finished in his pantry and need to be removed.

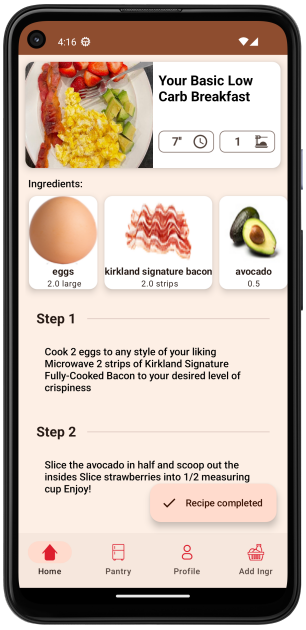


Figure 3.29: Mobile preview

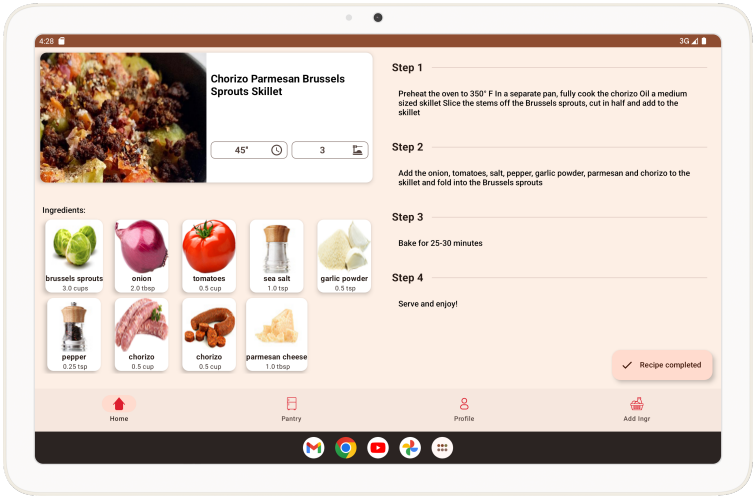


Figure 3.30: Tablet preview

3.1.11 Notification acceptance Screen

In order to receive notifications, user must accept receiving them. User can receive the notifications both in and also outside the application.

Figure 3.31: Mobile preview

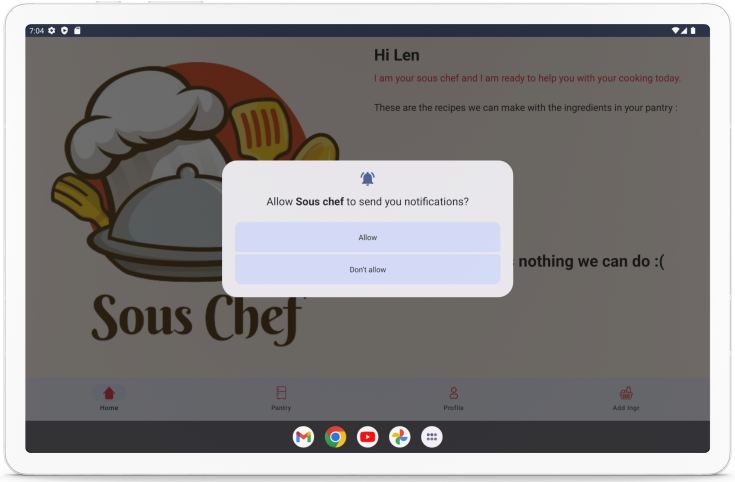


Figure 3.32: Tablet preview

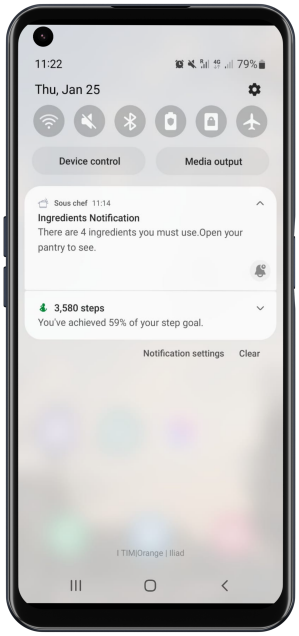


Figure 3.33: Mobile preview

Figure 3.34: Tablet preview

3.1.12 No internet Connection Screen

The app needs internet connection for its functioning. If the user looses it, a screen informing the user of such event is displayed asking to check the network connectivity and restart the app. Also if a call is done to API and it fails due to no Internet connection, user is informed of such behaviour via a standard alert Window.

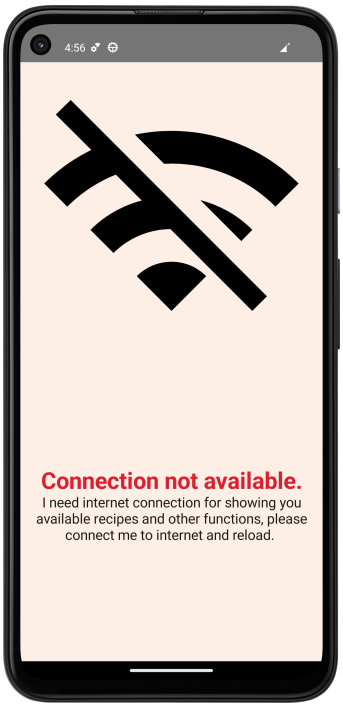


Figure 3.35: Mobile preview

Figure 3.36: Tablet preview

Chapter 4

Testing Campaign

In this section we provide an overview of the testing performed over the Sous chef app.

4.1 Unit Tests

Due to the application strong dependence on external service and the mock environment limitation employed during tests, our testing strategy for some part of the application were limited, making meaningful unit test impractical for those scenarios. Mocking Firebase calls can provide a workaround to some extent, but it may not fully capture the nuances and intricacies of the actual service behavior. As a result, unit tests may not accurately reflect the real-world interactions and edge cases that the application encounters during runtime.

We performed both Local and Instrumented Unit tests.

4.1.1 Local Unit Tests

Local unit tests are those that do not require emulator or physical device for its execution. These are the local tests we performed:

Test Case	Description
APIServiceTest	Verify the correctness of the Retrofit setup.
RecipeAPICallsTest	Verify the correctness during the retrieval of info from Spoonacular API endpoints.
ValidatorTest	Verify the validator correctness with a plethora of possible data insertions by the user.
Models test	Verify their initialization and correctness of the data during possible application interactions.

Test Case	Description
Models test	Verify their initialization and correctness of the data during possible application interactions.
LogInUIStateTest	Verify state of the UI during Login phase.
RegistrationUIStateTest	Verify state of the UI during Registration phase.
LogInViewModelTest	Verify correctness of events on email and password changes.
SharedUserDataViewModelTest	Verify the correctness during registration phase 2 and 3 (allergen and diet selection).
SearchIngredientViewModelTest	Verify the correctness of the data retrieval when user try to search for an ingredient.

4.1.2 Instrumented Unit Tests

Instrumented unit tests, on the other hand, require emulator or physical device for its execution. The app is built and installed alongside a test app that injects commands and reads the state. We performed those test to verify UI itself and interaction with it. However we were to test the screens that require view models with hilt dependency injection of application context due to having not enough skills in this field and shortage of time.

These are the screens we tested in order to verify all the fields are correctly displayed on both phone and tablet devices along with the the messages and dialog that appear when a certain flag is set :

- Splash Screen
- Welcome Screen
- Login Screen
- Sign up Screen
- Sign up Successful Screen
- User Allergies Screen
- User Diet Screen
-

4.2 End to End testing

As mentioned in Section 4.1 the challenges encountered in unit testing external services, particularly Firebase, led us to prioritize end-to-end testing for functions reliant on Firebase calls.

End-to-end testing involves executing tests that simulate real user interactions and validate the behavior of the application across its entire stack, including interactions with external services like Firebase. By exercising the application in a manner that closely mirrors its usage in a production environment, end-to-end tests offer insights into how well the application performs under realistic conditions and whether it meets the desired functional requirements.

The tested behaviour includes:

Test Case	Description
Sign up - first step	Upon trying to register with an already registered email address, verify correct information is displayed to the user when attempting to register with an already registered email address.
Sign up - first step	Includes check of Firebase authentication to verify the user is created with a unique ID.
Sign up - finish	Verify that the document is correctly created in Firestore and its name corresponds to the user's ID.
Log in	Verify the ability to log in with a registered email address.
Log in	Verify that the correct message is displayed when the user enters the wrong password and that the variable representing the password submission is correctly updated.
Forgotten password - click	Verify that an email is sent to the user with a prompt to reset the password.
Forgotten password - 3 times wrong submission	Verify that an email is sent to the user with a prompt to reset the password after three incorrect submissions.
Profile update - diet type	Verify that the user's document is properly updated in Firebase Firestore when updating the diet type.
Profile update - allergen	Verify that the user's document is properly updated in Firebase Firestore when updating allergen information.

Test Case	Description
Pantry - add ingredient	Verify that the user's document, specifically the pantry array, is properly updated in Firebase Firestore when adding an ingredient.
Pantry - delete ingredient	Verify that the user's document, specifically the pantry array, is properly updated in Firebase Firestore when deleting an ingredient.
Home - empty page	Verify that if the user's pantry is empty, the proper message is displayed.
Home - not empty page	Verify that the user can see all the ingredients stored in their Firebase Firestore document.
Recipe - detail	Verify that the user is able to click and see details upon clicking on a recipe.
Recipe - finish	Verify that the user's pantry array in Firebase Firestore is properly updated, meaning used-up ingredients are deleted.
Notification - window	Verify that the user is displayed a notification permission window upon installation.
Notification - allow	Verify that the user receives a notification when the condition is met.
Notification - allow	Verify that the user does not receive a notification when the condition is not met.
Notification - deny	Verify that the user does not receive a notification when the condition is met.

Table 4.1: Test Cases

4.3 Coverage analysis

As mentioned in through all this chapter, we encountered problems during the testing part, therefore the coverage of our tests is not that high. The classes which are not dependent

on Firebase and dependency injection were tested with almost 100% coverage, however the classes with such dependencies we were unable to test other than manually.

Chapter 5

Future development

As cooking is a broad field, there are multiple features that could be added to the Sous Chef application which would possibly make the application more appealing for its users.
From the technical point of view, during the last part of the development process we focused on testing, and realized that some of the source code could be improved as we encountered challenges during writing of the tests. The possible future development therefore would include creating a layer between the view model and external entities to create a better environment for testing code of the application. With such improvement, the test coverage could also be broadened. The features that could be added would also require creating a paid account on Spoonacular API as the number of calls would be increased.
Possible features to be added to the application would be:

- Custom Notification Timing: Enabling users to set specific times for receiving notifications would enhance the app's flexibility and user control, allowing for tailored reminders based on individual preferences and schedules.
- Enhanced Pantry Management: Introducing the capability to add multiple instances of the same item to the pantry would streamline inventory management for users with recurring ingredients or varying quantities.
- Recipe Recommendations with Incomplete Pantry: Providing users with access to recipes even when their pantry lacks certain ingredients would encourage exploration and creativity in the kitchen, inspiring users to experiment with diverse culinary combinations.
- Integrated Shopping List: Integrating a built-in shopping list feature within the application would simplify meal planning and grocery shopping, allowing users to seamlessly transition from recipe discovery to ingredient procurement.

Chapter 6

References

- Runtime view diagrams made with DrawIO
- User interface design made with Figma
- Software Engineering 2 course material - Politecnico di Milano 2023-2024
- App icons and icons and other custom design items - Canva
- Development guide and testing guide form official Android developer site [1]

[1] https://developer.android.com/