Politecnico di Milano

AA 2020-2021

Design and Implementation of Mobile Applications

Dima

Deliverable:	DD
Title:	Design Document
Authors:	Meneghin Giulia & Mauri Giuseppe
Version:	1.0
Date:	05-January-2021
Download page:	`<https://github.com/Ciuse/MauriMeneghin>`
Copyright:	Copyright © 2020, Meneghin Giulia & Mauri Giuseppe – All rights reserved

Contents

1 Introduction

1.1 Scope

Bealthy is an application that allows the user to enter the dishes they consume and monitor the symptoms they experience on a daily basis. The entry of dishes can be done by: searching a predefined list of foods known to our application, scanning a barcode with the camera, or manually entering the information of the dish. The purpose of the application is to collect the data entered by the user, process it and reorganize it in the form of a graph to show the user the correlation between the symptoms experienced and the individual ingredients consumed. The application implements three types of graphs: The first consists of showing which symptoms/ingredients have a higher percentage of the total in a specific period of days. The second type shows which ingredients affect a particular symptom in the selected time period. The third shows of an ingredient what is the likelihood of causing one or more symptoms. In addition, the system allows the user to enter his or her medical treatments that he or she is following and shows past treatments. The system compares the previous month's symptom data with that of the days under treatment to show the effectiveness of the treatment taken.

1.2 Goals

(G1) Allow the user of the application to create a personal account and modify his information.

(G2) Allow the user to change the current day with an interactive calendar.

(G3) Allow the user to add a dish to a specific mealtime of the selected day(Breakfast, Lunch, Snack, Dinner), setting the eaten quantity.

(G4) Allow the user to search a dish.

(G5) Allow the user to scanning a bar code with the camera application.

(G6) Allow the user to create a new dish with a specific image and a lists of ingredients.

(G7) Allow the user to add a symptom in a specific day, setting the intensity, frequency, mealtime occurrences.

(G8) Allow the user to delete the inserted dishes.

(G9) Allow the user to delete the inserted symptoms.

(G10) Allow the user to visualize the percentage of symptoms occurrences and ingredients eaten in a fixed days range (weekly or monthly) or in a days range selected with the calendar.

(G11) Allow the user to visualize the correlation between the symptoms occurrences and the eaten ingredients.

(G12) Allow the user to visualize a page tha shows of an ingredient what is the likelihood of causing one or more symptoms.

(G13) Allow the user to visualize the in progress treatments and the completed ones.

(G14) Allow the user to add a new treatment specifying the diet and medical cure.

(G15) Allow the user to remove a treatment.

(G16) Allow the user to change the profile image.

1.3 Functional Requirement

(G1) Allow the user of the application to create a personal account with an email and password, or with an external services (Google, Github, Twitter).

- The user should be able to register through the mobile application and in order to do that he has to provide a password, and a valid e-mail.

(G2) Allow the user to change the current day

- by clicking another day on the interactive calendar.
- by using the appropriate tab to change the dates.

(G3) Allow the user to add a new dish in a specific mealtime.

- The user has to click on the "+" button of the mealtime he/she has chosen to add a new dish.
- Then he/she can choose the way with which adding the dish. There are 5 ways to add it:

(G4) The first 3 ways consist of searching for the dish:

- Using the **"Search list"** button:
  A list of all the dishes in the online database will appear. After selecting the desired dish and entering the quantity eaten, it will be automatically inserted on the Homepage in the box corresponding to the previously selected mealtime.

- Using the **"Favourites list"** button:
  A user list of all favourites dishes will appear. After selecting the desired dish and entering the quantity eaten, it will be automatically inserted on the Homepage in the box corresponding to the previously selected mealtime.
- Using the **"Your created dishes list"** button:
  A user list of all created dishes will appear. After selecting the desired dish and entering the quantity eaten, it will be automatically inserted on the Homepage in the box corresponding to the previously selected mealtime.

(G5) Using the **"Scanning a barcode"** button:
The user can scan the product eaten and if there is a valid match in the external database "Open food facts" the page containing all the information about it will be shown. The user can edit the image, the name and the list of ingredients provided. Once the quantity eaten has been entered, it will be automatically inserted on the Homepage in the box corresponding to the previously selected mealtime.

(G6) Using the **"Creating new dish"** button:
A form for manually entering the information of the dish eaten will appear: The user can enter a picture (using the mobile camera or uploading it from own image gallery), a name, the quantity eaten and the ingredients with their corresponding quantity.The user must then click on the "create" button to create it and add it on the Homepage in the box corresponding to the previously selected mealtime.

(G7) Allow the user to add a symptom in the selected day

- The user has to click on the "+" button in the symptom box of the HomePage.
- A page with all symptoms provided by the system will open and after clicking on the symptom that the user wants, another page will open.
- the user has to set the various parameters required: the intensity of the perceived pain, the frequency with which the symptom occurred and the time(s) of day it occurred.
- Once you have entered all the parameters, the "save" button will become clickable. The user can click on this button to automatically add the symptom to the Homepage.

(G8) Allow the user to delete a dish

- The user has to click on the dish button to remove its occurrence from a specific time of the selected day.

- On the page with the specific information of that dish, there is a "delete" button on the top right corner of top app bar. The user has to click on it.

(G9) Allow the user to remove a symptom occurrence from a specific day

- The user must click on the symptom button in the Homepage to remove its occurrence.
- On the page with the specific information of that symptom, there is a button to reset all parameters, so the user can delete it.

(G10) Allow the user to visualize the percentage of symptoms occurrences and ingredients eaten in a fixed days range (weekly or monthly) or in a days range selected with the calendar.

- The user has to select the statistics icon in the bottom app bar to view the graphs of symptoms and ingredients.
- By clicking the button in the top right corner of the app bar, he can change the period of days (week or month).
- In this page there are two Tabs that allow navigation between groups of content that are related and at the same level of hierarchy: symptoms and ingredients.

(G11) In the first tab "symptoms" there is a pie chart with the percentage of symptoms and under it, there are all the symptoms button. Clicking on one of them will open another bar graph showing the severity of that specific symptom during the selected period of days. Moving from one day to the next will show the list of ingredients that most caused that symptom.

(G12) In the second tab "ingredients" there is a pie chart with the percentage of ingredients and under it, there are all the ingredients button. Clicking on one of them will open another page that shows of an ingredient what is the likelihood of causing one or more symptoms.

(G13) Allow the user to visualize the treatments in progress and the completed ones.

- The user has to select the treatments icon in the bottom app bar to open the page that shows the list of treatments in progress or that ones completed.

(G14) Allow the user to add a new treatment

- The user has to select the treatments icon in the bottom app bar to open the page that shows the list of treatments in progress or that ones completed.
- In that page the user can click on the "+" button in order to create a new treatment.

- Clicking on the button will open the page containing the form for entering a new treatment. The name, start date and end date of the treatment are compulsory, while the description, medical information and diet are optional.

(G15) Allow the user to remove a treatment

- The user has to select the treatments icon in the bottom app bar to open the page that shows the list of treatments in progress or that ones completed.
- In that page the user can click on the treatment he/she wants to delete.
- In the page of treatment information, there is the "delete" button. The user has to click on it.

(G16) Allow the user to change the profile image

- The user has to select the personal info icon in the bottom app bar to open the page that shows the information and the settings of the user.
- In that page the user can click on "camera" button in order to change his/her image profile.
- The user can take a photo or upload an image from his/her gallery.

1.4 Acronyms

- *API:* Application Programming Interface.

1.5 Revision history

- Version 1.0

2 Architectural Design

2.1 Overview

Bealthy application has a two tier architecture.

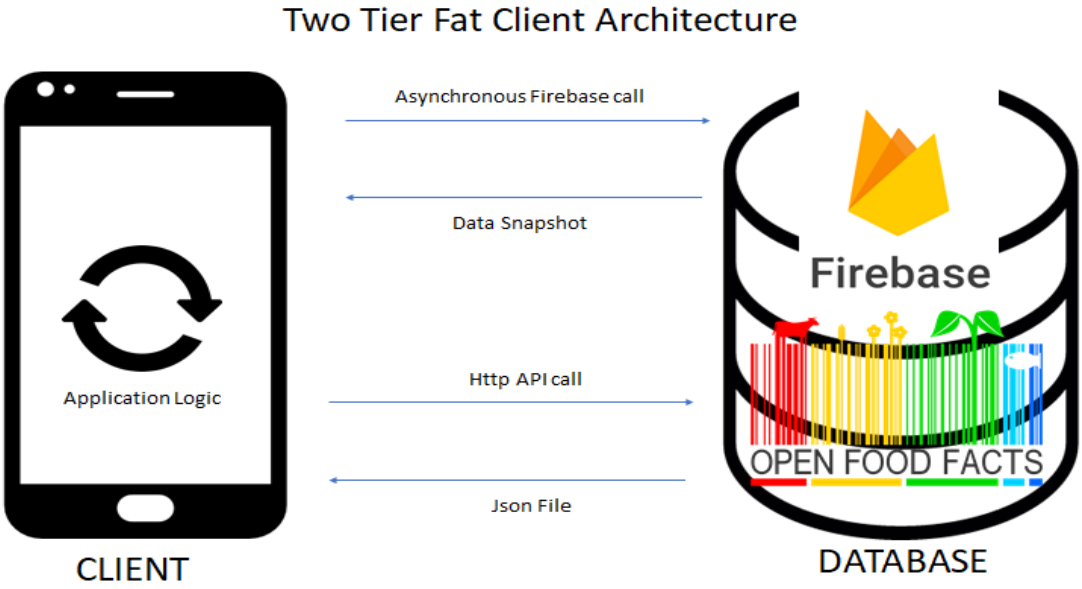


Figure 1: Structure Overview

Bealthy uses a two tier architecture with a fat client. The client tier and the business logic tier are both located within the mobile application, while the database is remote. Most of the data are in the database managed by the provider "Firebase" on which there are all the static data of the predefined dishes, the dishes created by the user, the day by day information of which dishes he eats and which symptoms he shows, and the files of the images of the dishes and the photo of the user. The other part of the data is managed by the provider "OpenFoodFacts", an open source database containing thousands of food information associated with a barcode.

2.2 High level components

The architecture of the high-level components consists of two different types of elements: the External Services and the Mobile Application.

- **The Mobile Application** allows users to interact directly with the system. This component provides a simple and clear view of the various functions that the system provides to the user. We decided to develop a fat client application that handles both the UI and UX part as well as the business logic. A fat client application connects to a external remote database (firebase) in order to sync data or upload

and download user information. Bealthy processes most of data by itself, it does not rely on any servers for processing. The processing of user data, in order to showing the correlation between the symptoms and the ingredients, is done offline.

– There are two different kind of **External Services** that the application uses: The database ones and the authentication ones. The external databases,Firebase and OpenFoodFact, are interrogated by our application when it needs their updated data while the external authentication services are: Google, Github and Twitter.

2.3 Component view

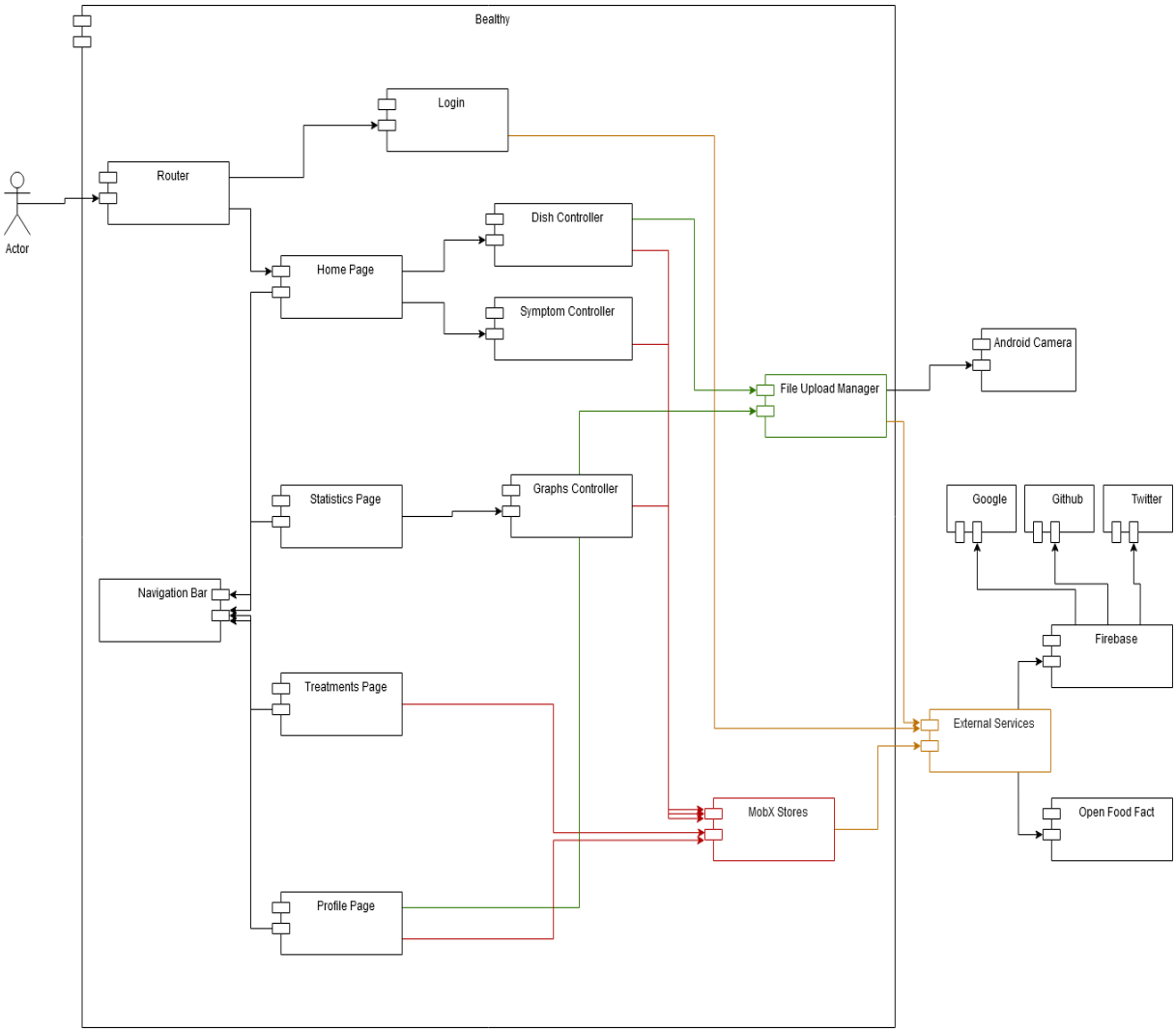


Figure 2: Component Diagram

- Client: the client's device (mobile app).
- Administrator: the administrator's device (mobile app).

2.4 Runtime view

In the following Sequence Diagram for simplicity and clarity, some steps have been omitted in the communication between the components external to the main system and the router; as the main aspects of communication, between the User and the Travlendar system, have been dealt with in the RASD document.

2.4.1 Sequence Diagram 1

This Sequence Diagram deals with the addition of an appointment by the customer. The request is taken on board by the router, which communicates with the appointments component and will show the user the form to fill in the data concerning his new event.
Subsequently, after the user has inserted the various information, the Appointment Controller component will check that it does not overlap with an existing one, in case the Notification Manager will send the user an error.
If the appointment is valid, the data worker will be asked to save the new appointment on the database and the latter will be responsible for providing the travel calculator with all the data necessary to calculate the trip from the previous position to that of the appointment, with the means and preferences that the user has selected.
If it is possible to solve the calculation, the travel itinerary will be saved and the user will receive a confirmation message; if not, it will be notified with a warning.

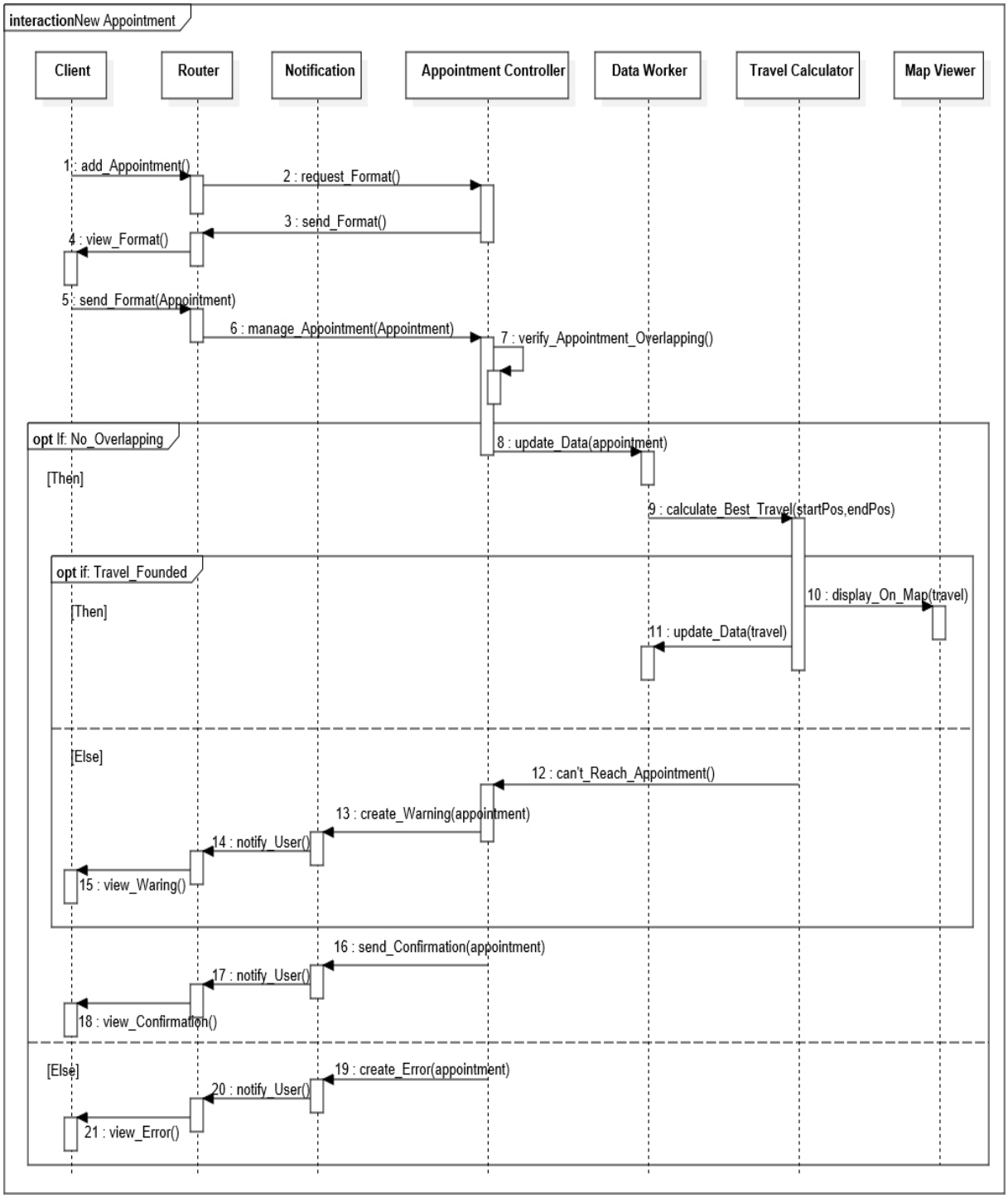
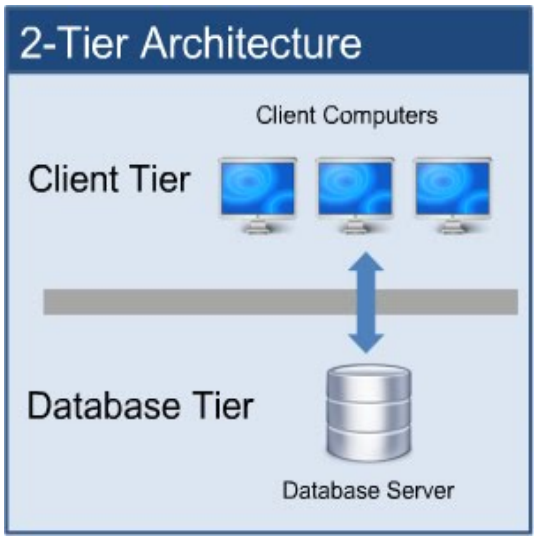


Figure 3: Sequence Diagram 1

2.5 Styles and patterns

2.5.1 Overall Architecture



Our application has been divided into 2 tiers: (fat client)

1. **Database tier (DAL: Data Access Layer)**
   Here information is stored and retrieved from an external database. The information is then passed to the logic tier for processing, and then eventually back to the user.

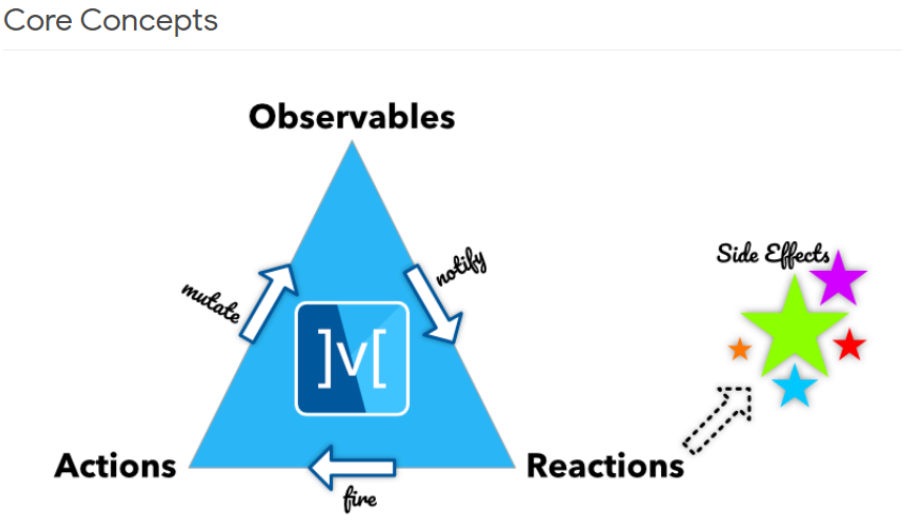
2. **Business Logic (BLL: Business Logic Layer) and Client tier (interface to BLL)**
   The highest level of the application is the user interface, whose main function is to take care of all the user's requests, and to give him the possibility to interact quickly with the system. The user will then be able to view own Symptoms, eaten dishes and statistics of a certain time period whenever he/she wishes. Business Logic Tier is an important part of the Client Tier. It is managed by the "store" classes of our application through the use of **MobX** that is a state-management library that makes it simple to connect the reactive data of application with the UI. All of these classes coordinate the application, process commands, make logical decisions and assessments and perform the calculations in order to elaborate user statistics chart of symptoms and ingredients. This level is mainly concerned with: processing the user's data and information, managing the interactions between our system and that of external Databases services.

2.5.2 Design decisions and patterns

- We decided to use a **2 two tier architecture** because our application does not need a real external server to process the data or do complex calculations. The app does not need a lot of computing power. The data collected on the online database is downloaded as often as needed and is reorganised in such a way as to show statistics. In the end, these are relatively simple and inexpensive calculations.

- We decided to use **Flutter** because it is Google's UI toolkit used for building natively compiled applications for mobile from a single codebase. It is characterize by a fast development that allow to use a rich set of fully-customizable widgets in order to build native interfaces.

- Flutter provides a lot of flexibility in deciding how to organize and architect apps. We decided to use the library **MobX** because it is a popular state management library for Dart and Flutter apps. It has also been recognized as a Flutter Favorite package. MobX relies on 3 core concepts: Observables, Actions and Reactions. Observables represent the reactive state of your application. Actions are semantic operations that modify these observables and Reactions are the listeners that "react" to the change in Observables, by updating UI or firing network operations. Reactions are considered as the side-effects in a MobX-based application.



- We decided to use **Firebase Cloud Storage API** that let us to upload users' data to firebase cloud and download it. It is a flexible and scalable NoSql database for mobile development. We protect the data for users logged in our app throght the use of certain rules with firebase authentication. Security, of course, is our first concern. All transfers are performed over a secure connection. Firebase SDKs for Cloud Storage add Google security to file uploads and downloads for your Firebase

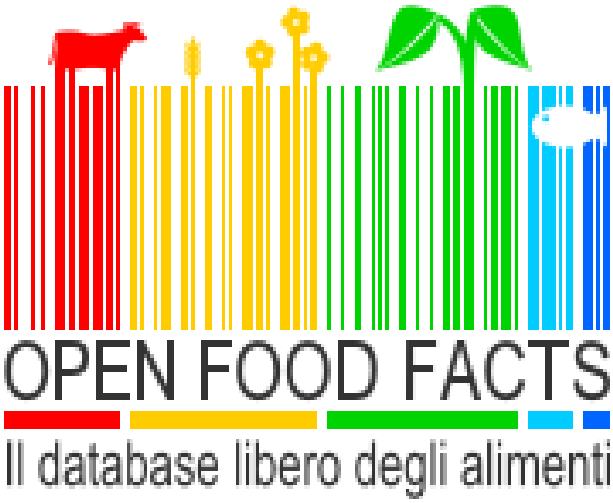
apps, regardless of network quality. We use that SDKs to store images and other user-generated content like dishes and symptoms. Firebase Cloud is also use to share standard dishes and symptoms between all logged users. We developers have to mantain them updated. In order to integrate firebase into our application we used FlutterFire that is a set of Flutter plugins which connect our Flutter application to Firebase:

FlutterFire Overview

- cloud_firestore: 0.14.1+3
- firebase_storage: 5.1.0
- firebase_core: 0.5.0
- firebase_auth: 0.18.3

To handle the authentication of external services (google, github and twitter) we also imported the firebase plugin: lit_firebase_auth: 0.3.0

– We have also integrated **Open Food Facts** which is a database of food products with ingredients, allergens, nutritional information and all the details of information that can be found on product labels. We used Flutter package for the Open Food Facts API. It can let us to Easily access to more than 1.5 million products from all around the world. Open Food Facts is powered by global contributors and is constantly growing thanks to them. The user of our application can scan the barcode of a food and thanks to the integration of our system with Open Food Facts database we can show him/her the ingredients.



2.6 Software System Attributes

2.6.1 Reliability

The users age range is 16-40 years old, so the system will be designed to be always reliable:

- On workday, in the early morning (7-10) and in the late afternoon (16-19);
- On weekend, from the midday to middle evening (11-16) and in the evening (19-24);

Because during the working day the user will use the application mainly for register his working or studying appointment and some personal engagements.
While in the weekend the user will register his leisure events and night friends meeting.

2.6.2 Availability

The system must guarantee a 24/7 service. Very small interrupt of the service during the day will be acceptable and tolerated. But when an issue occurs, the system must respond correctly after a maximum of 3 user attempts.

2.6.3 Security

Users credentials and external services account will be cryptate and stored in a reserved and protect area of the system database. Also, the privacy information about the user movement and his localization must be all encrypted and totally protected.

2.6.4 Maintainability

Our system uses many external APIs, so the maintainability of our software is very much dependent on this factor. Our system must always be updated with external API interfaces, and it must always be able to interpret and exploit the data it obtains from them. Generally, there should not be too much invasive maintenance processes, as it uses very popular APIs used by many other software, whose changes are often minimal and well documented.

2.6.5 Portability

The system in terms of portability shall be very flexible. The application logic and the system interfaces are abstractly separated; so, the application porting consist only in the re-adapting of the user interface with the new operating system. Also, could be necessary to reimplement some of the system interaction with the external services.

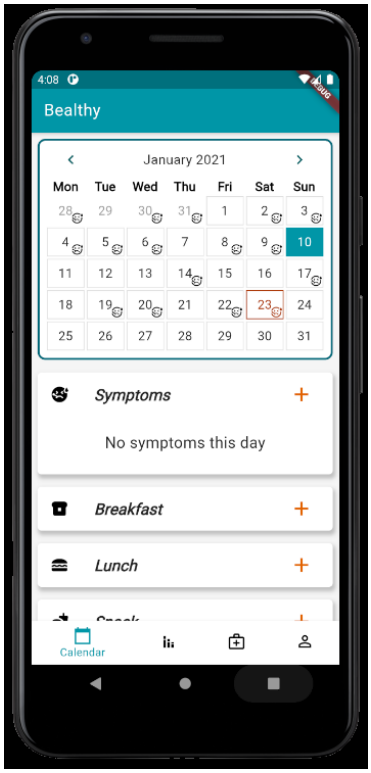
3 Specific Requirement

3.1 User characteristics

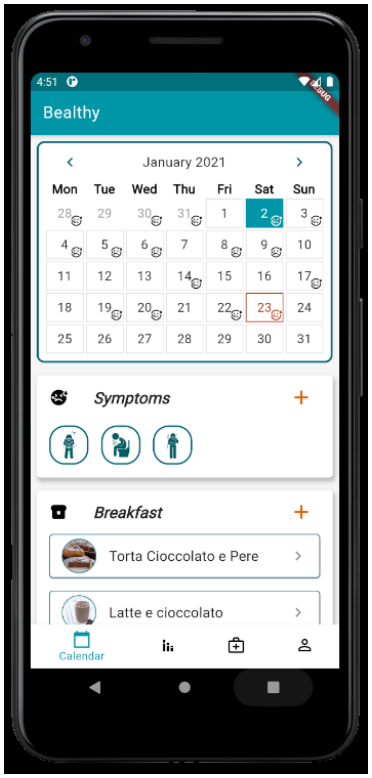
The main actor that interact with the system is the **Registered User** of the application. The registered user can access to all the function of the application, and can link external available services to his account in the login page.
The application is designed for all those users who suffer from eating problems or chronic disorders in their daily diet and who experience frequent symptoms. The application would help them monitor the chronic pattern of symptoms and identify a correlation between what they eat and the symptoms they experience so that they can follow a food diet that may alleviate their symptoms in some way.

3.2 User Interface Design

1)Home page



(a) **Empty day**



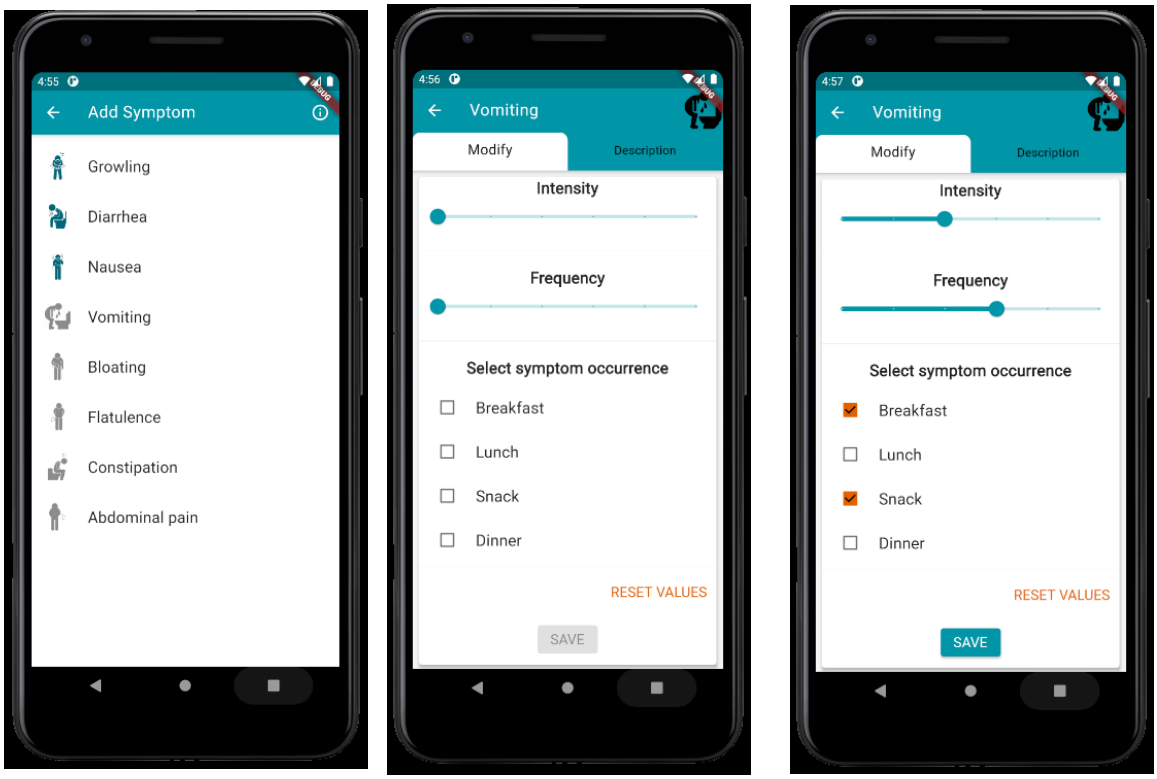
(b) **Day with symptoms and dishes**

- This is the main screen of the application, from this screen the user can navigate through the calendar by simply clicking on any day. The side

arrows allow the user to change the month selected on the calendar. Underneath the calendar, the user can both view and keep track of his or her symptoms and meals taken in the past days and enter new ones using the appropriate buttons. Scrolling upwards, the calendar is reduced to a simple bar showing the day selected by the user. Again, the user can use the side arrows to change the day. The calendar also shows with a specific icon the presence or absence of a symptom on a given day so that it is easy to see which days the user has been sickest.

- With the "+" button in the symptoms box the user can add a symptom.
- With the "+" button in the mealtime boxes (breakfast,lunch,snach and dinner) the user can add a dish.
- With the bottom bar it will be possible to navigate through the various screens(calendar , statistics , treatments and personal page).

2)Add symptom page

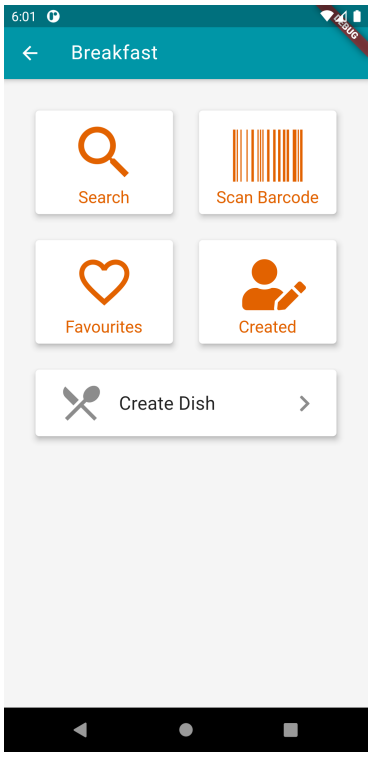


(a) **All symptoms** (b) **Add symptom** (c) **Set parameters**

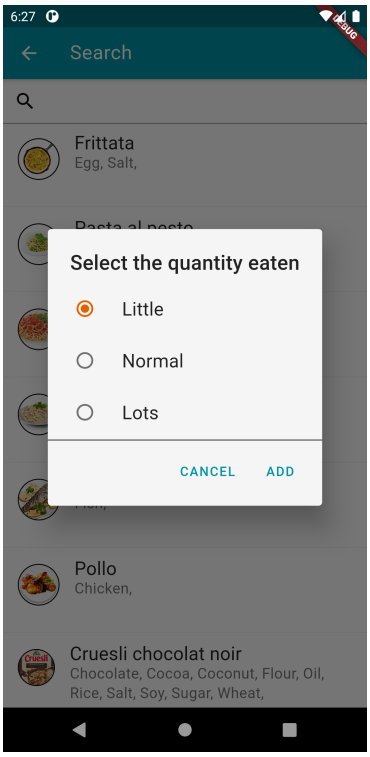
- The first picture shows the all symptoms screen that is displayed after clicking on the "+" button in the symptom box under the calendar in the Homepage screen.

- After clicking on a specific symptom, a form will appear. The second picture shows this form that the user needs to complete in order to be able to add a new symptom in the selected day.
- It's necessary set all the parameters: intensity, frequency and when the symptom occurs in order to click on the "save" button.

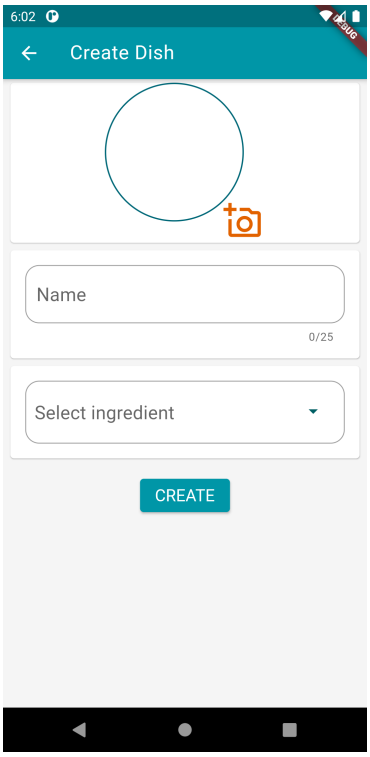
2)Add Dish page



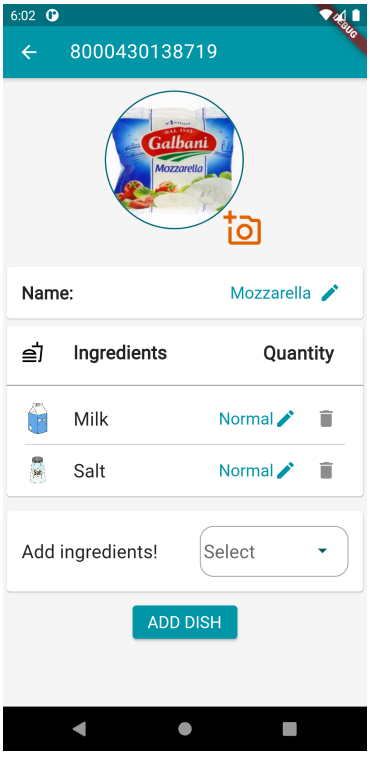
(a) **Add dish**



(b) **Search Dishes**



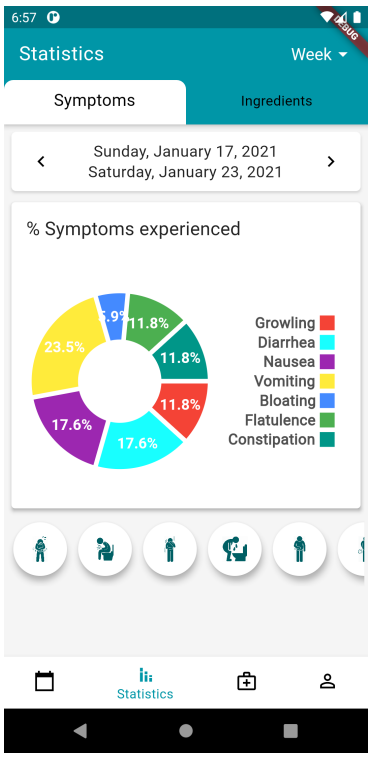
(c) **Create dish**



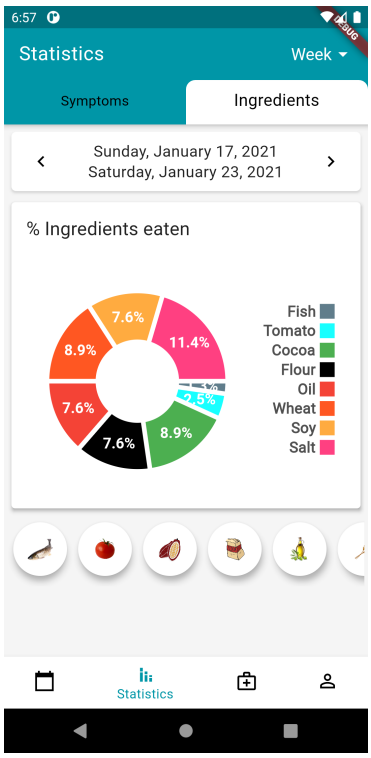
(d) **Scanned dish**

- The first picture shows the five ways to add a new dish in a specific mealtime of selected day. Three methods are similar and consists of a research of dish into a list(Search,favourite and created dishes buttons) the last two ones are two actions that the user can do. The user in fact can scan the barcode of a dish or can create a customizable dish inserting an image, a name, the list of ingredients and their quantity.
- The three searching ways are displayed with the same design. In each of them there is the list of dishes with which the user can interact. Clicking on one of the dishes, a popup will open that asks for the eaten quantity.
- After clicking on "create" button, a form will appear. The picture shows this form that the user needs to complete in order to be able to add a new dish in the selected day. This form let the user to customize a new dish with an image that can take with his/her camera or uploading an image from the gallery, inserting a name and the list of ingredients.
- The last picture shows the page of a dish scanned by the user with all the informations that have been collected by the Open Food Facts database. If the user wants, he/she can also insert other ingredients.

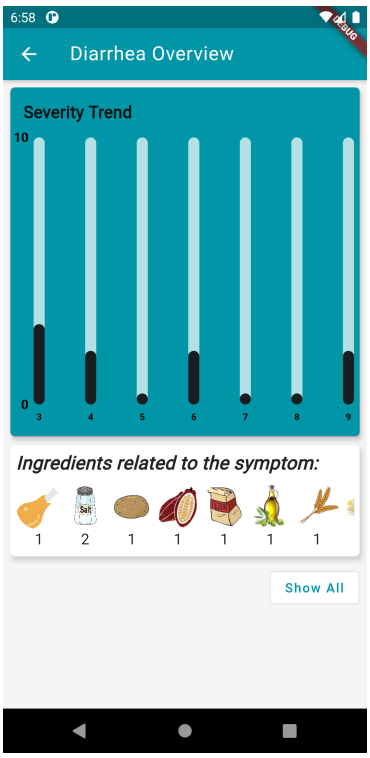
2)Visualize statistics



(a) **All symptoms**



(b) **Add symptom**



(c) **Set parameters**

-

-
-

4 Implementation,Integration and Test Plan:

4.1 Elements to be integrated

4.2 Integration Testing Strategy

4.3 Component Integration

4.4 Used Tools

- StarUml 2.8.0
- Miktex 2.9.6361
- Texmaker 5.0.2
- DeepL
- GitHubDesktop 1.0.6
- AdobePhotoshop CC 2017
- Power Point

