NutriApp

Analysis and Design Document

Student:Prodan Andreea

**Group: 30235**

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Version** | **Description** | **Author** |
| 18/03/2018 | <1.0> | Initial Version of Document | Prodan Andreea |
| 25/04/2018 | <2.0> | Revision of iteration 1.1 and 1.2   * used architectural pattern and design patterns * options separation between client and trainer/nutritionist in Project Specification | Prodan Andreea |
|  |  |  |  |
|  |  |  |  |

Table of Contents

[I. Project Specification 4](#_Toc509156559)

[II. Elaboration – Iteration 1.1 5](#_Toc509156560)

[1. Domain Model 5](#_Toc509156561)

[2. Architectural Design 10](#_Toc509156562)

[2.1 Conceptual Architecture 10](#_Toc509156563)

[2.2 Package Design 12](#_Toc509156564)

[2.3 Component and Deployment Diagrams 13](#_Toc509156565)

[III. Elaboration – Iteration 1.2 14](#_Toc509156566)

[1. Design Model 14](#_Toc509156567)

[1.1 Dynamic Behavior 14](#_Toc509156568)

[1.2 Class Design 18](#_Toc509156569)

[2. Data Model 21](#_Toc509156570)

[3. Unit Testing 23](#_Toc509156571)

[IV. Elaboration – Iteration 2 23](#_Toc509156572)

[1. Architectural Design Refinement 23](#_Toc509156573)

[2. Design Model Refinement 23](#_Toc509156574)

[*[Refine the UML class diagram by applying class design principles and GRASP; motivate your choices. Deliver the updated class diagrams.]* 23](#_Toc509156575)

[V. Construction and Transition 24](#_Toc509156576)

[1. System Testing 24](#_Toc509156577)

[2. Future improvements 24](#_Toc509156578)

[VI. Bibliography 25](#_Toc509156579)

# Project Specification

*[Present the project specification]*

The application is designed for the persons who want to sustain a healthy diet on their own or with the help of a trainer or a nutritionist. The application has tow types of users: the regular user/ client, who are interested in following a healthy diet, and the nutritionist/ trainer who helps its users.

The options of the regular user are:

1. Sign Up page
   * Sign In button
2. Logging steps
   * User type
   * Activity type
   * Goal
   * Sex
   * Weight (kg)
   * Allergies
3. Main page
   * Foods list, ordered alphabetically
   * Main menu
     + Sign Out
     + Chat
     + Diary
     + Personal Data
     + Add New Food
     + Delete Food
4. Chat page
   * Invite
   * Send message (+)
   * Send button
5. Diary page
   * Choose another day
   * List with data: added date, food name, quantity(g), macros values
   * Total calories and macros
   * Remained calories and macros
6. Choose another day page
   * List with registered days from which to choose
7. Add to diary page
   * Picture and name of the selected food
   * Quantity space for user input
   * Add button
8. Delete food page
   * Space for data input
   * Delete button
9. Personal data page
   * User’s personal data
   * Necessary macro
   * Change button
10. Calculate page
    * Picture with the name of the choosed food
    * Quantity user input
    * Calculate button

The options for the nutritionist/trainer are:

1. Sign Up page
   * Sign In button
2. Logging steps
   * User type
3. Main page
   * Foods list, ordered alphabetically
   * Main menu
     + Sign Out
     + Chat
     + Foods
     + Add New Food
     + Delete Food
4. Chat page
   * Invite
   * Send message (+)
   * Send button
5. Delete food page
   * Space for data input
   * Delete button
6. Personal data page
   * User’s personal data
   * Necessary macro
   * Change button
7. Calculate page
   * Picture with the name of the choosed food
   * Quantity user input
   * Calculate button

# Elaboration – Iteration 1.1

# Domain Model

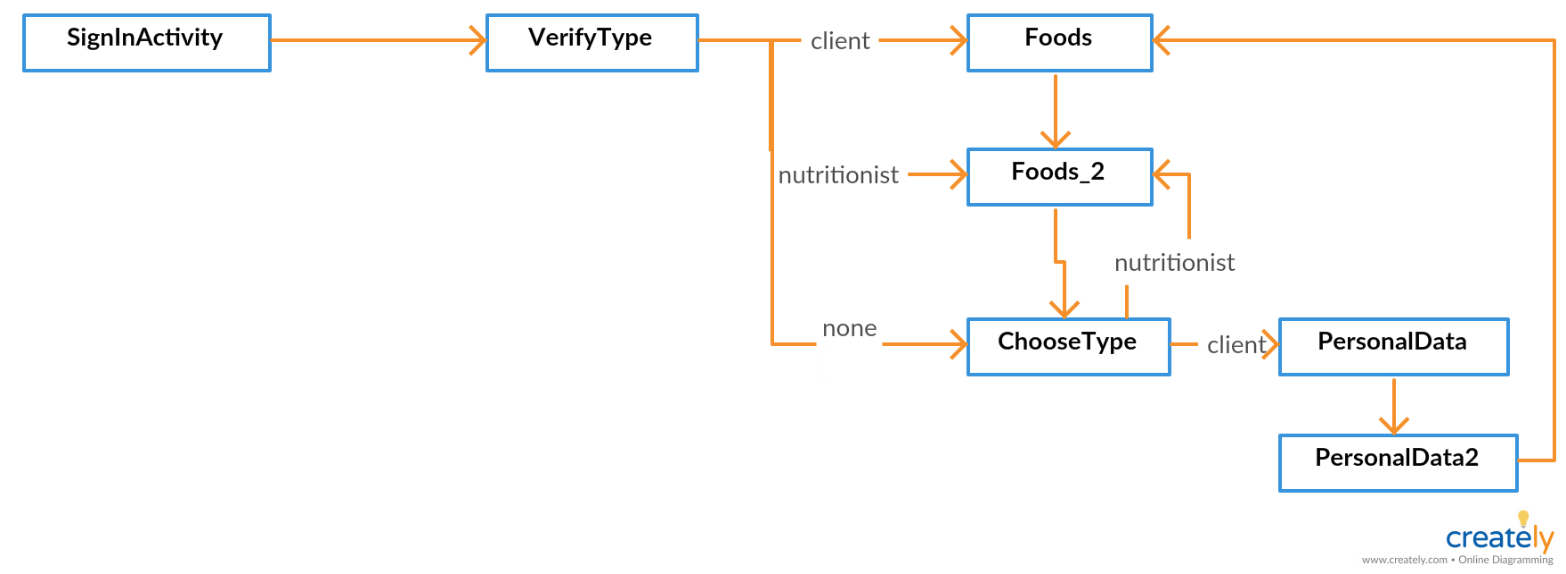
*[Define the domain model and create the conceptual class diagrams]*

after deleting = the access to the class is made after the food is deleted succesfully

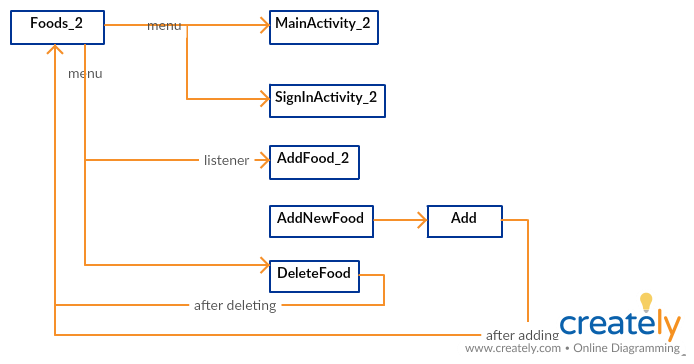
menu = the access is made from the menu

listener = the access is made through the listener, at the screen touching

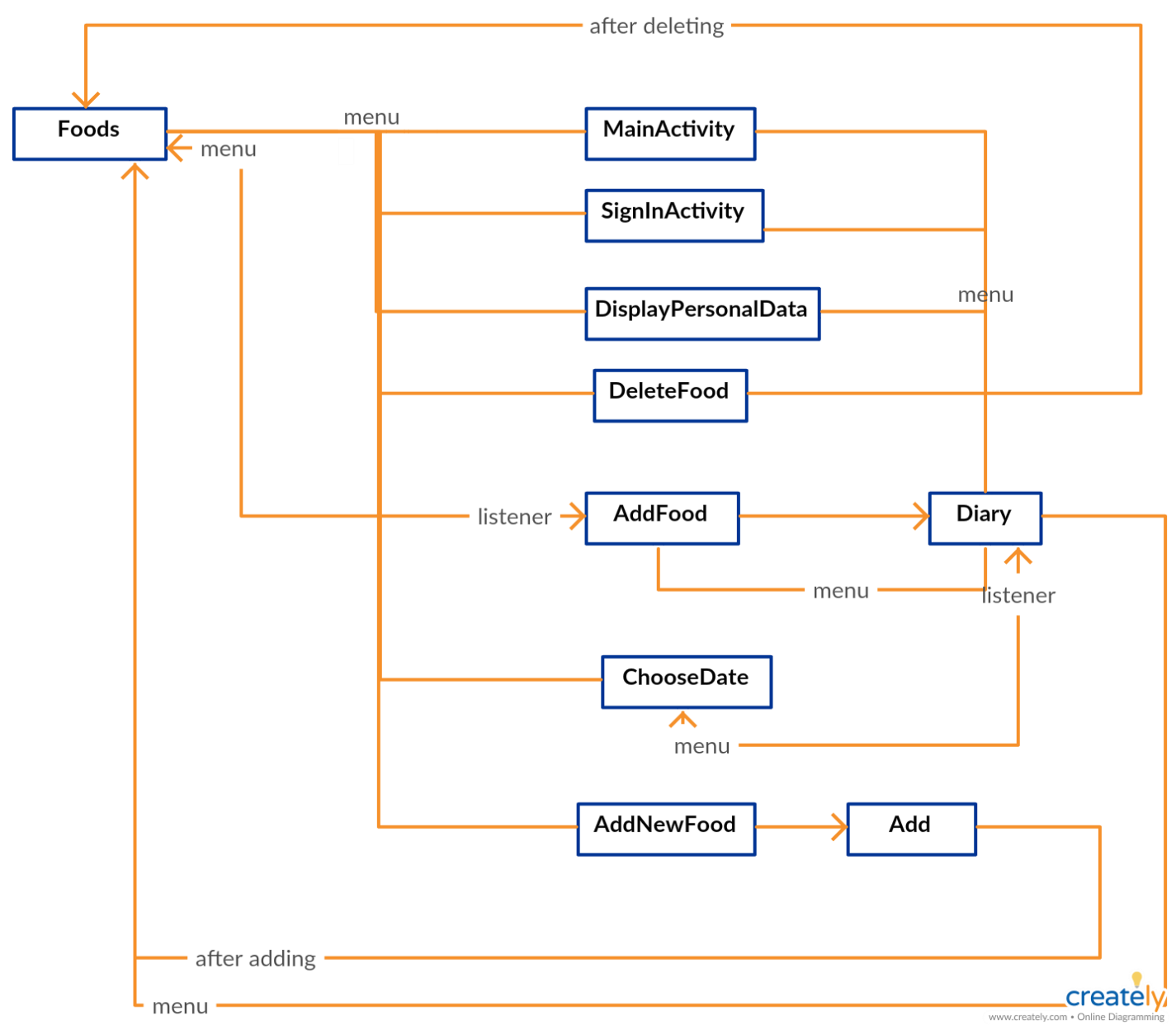
Client Sign In:



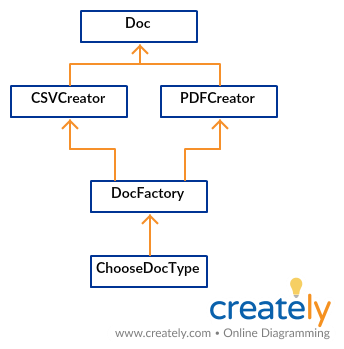
Trainer:



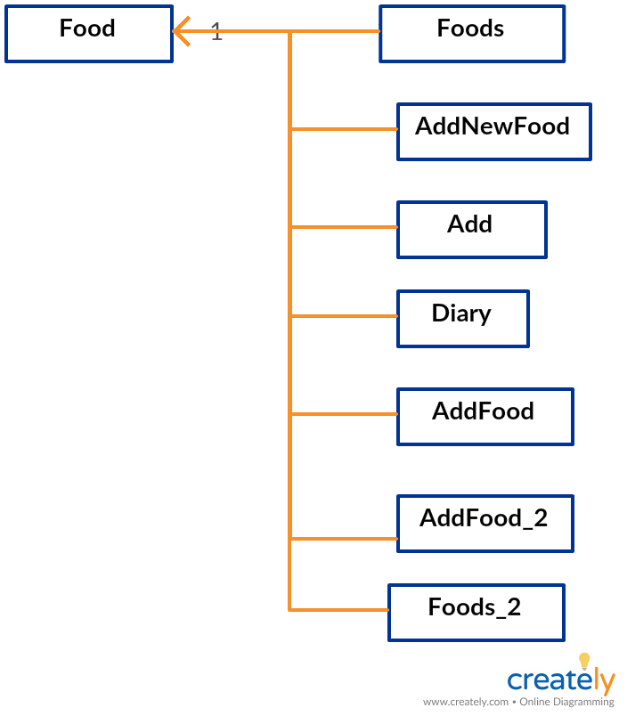
Client:

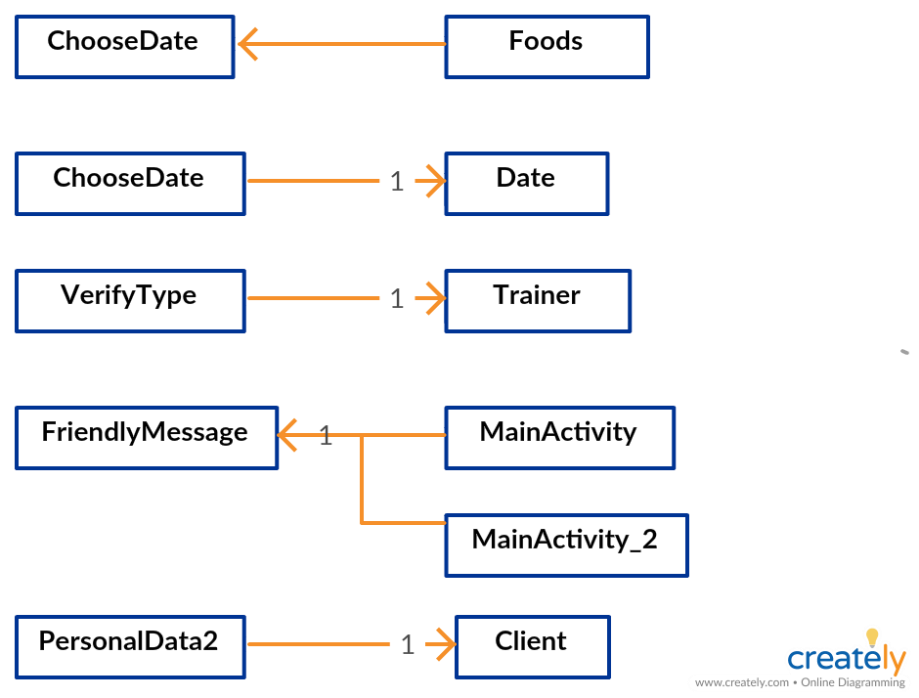


Document Creator

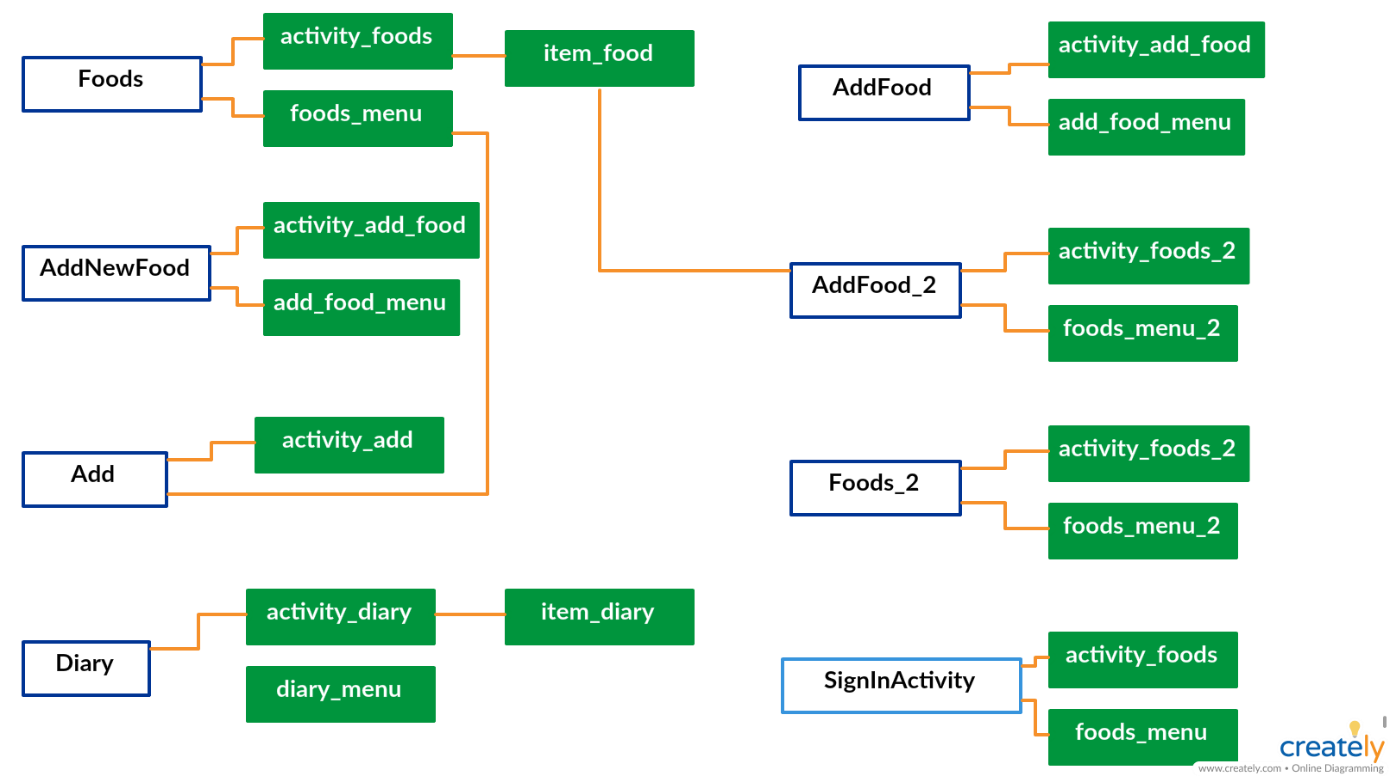
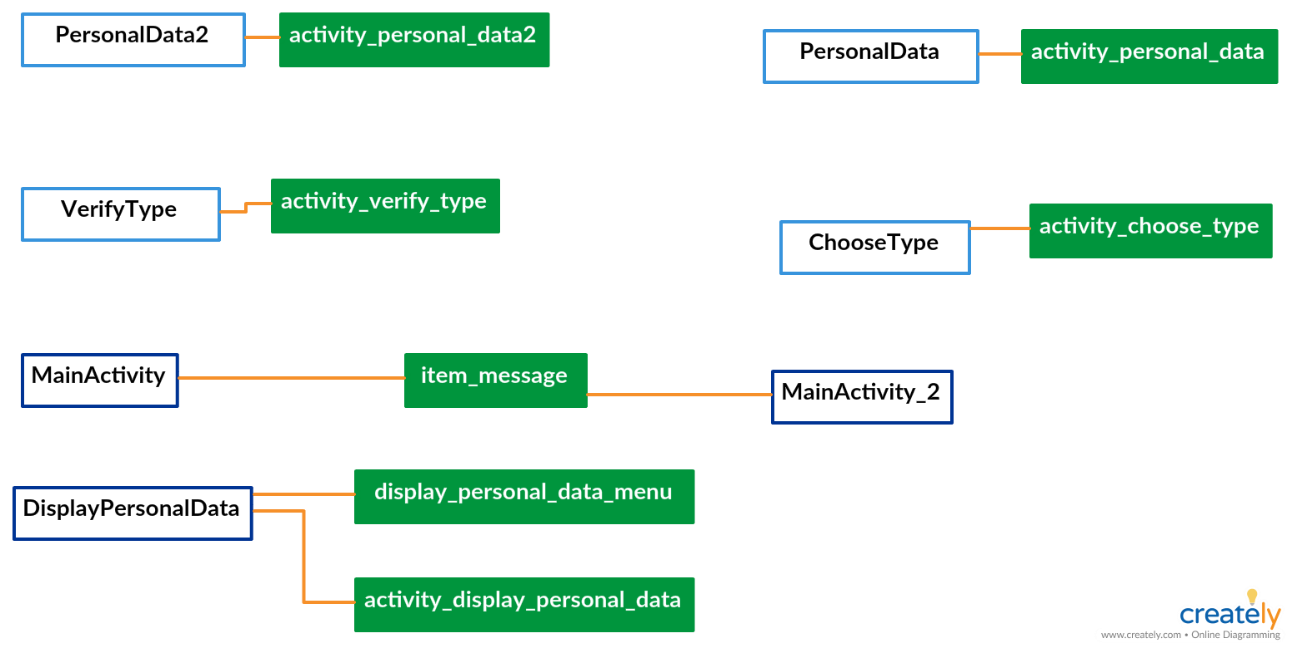


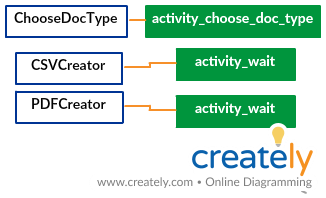
Classes used for data storage





.xml files connection



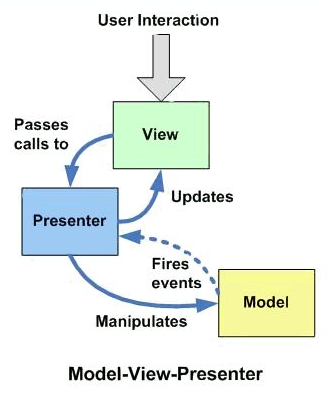


# Architectural Design

## Conceptual Architecture

*[Define the system’s conceptual architecture; use an architectural style and pattern - highlight its use and motivate your choice.]*

As architecture design pattern, the Model-View-Presenter (MVP) is used.



The view is reperesented by the xml files. It is only responsible for presenting the data and routing users actions to the presenter.

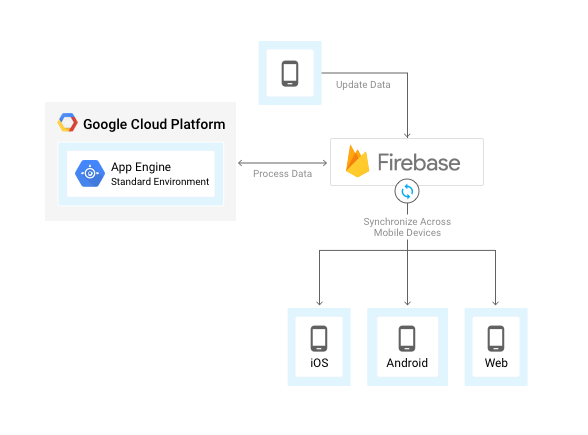
The presenter is represented by the activity clasess and is the middle-man between model and view. All the presentation logic belongs to it. The presenter is responsible for querying the model and updating the view, reacting to user interactions updating the model.

The model is represented by the Food, User, Trainer, Client, Date and Diary classes. It holds the business logic of the application. It controls how data is created, stored and modified.

The main reason for choosing MVP is that in Android there is a problem arising from the fact that Android activities are closely coupled to both interface and data access mechanisms so all logic of view and model remains in activity which generates complexity. For the application to be easily extensible and maintainable, well separated layers must be defined.

MVP makes view independent from the data source. The application is devided into three diffenrent layers, which permits independent testing.

For the connection between multiple devices and acess to the storage, the architecture for the android applications which uses the Firebase for data storage is used.



**The reason choosing Firebase**

**Real Time**

The entire database is backed by a real-time connection to the back end, and if you use the Firebase SDK, you get live updates in your app any time something changes. You don’t have to constantly poll the server or build pull-to-refresh controls into your app, and no need to send a push notification if you want to control the app updates.

**Speed**

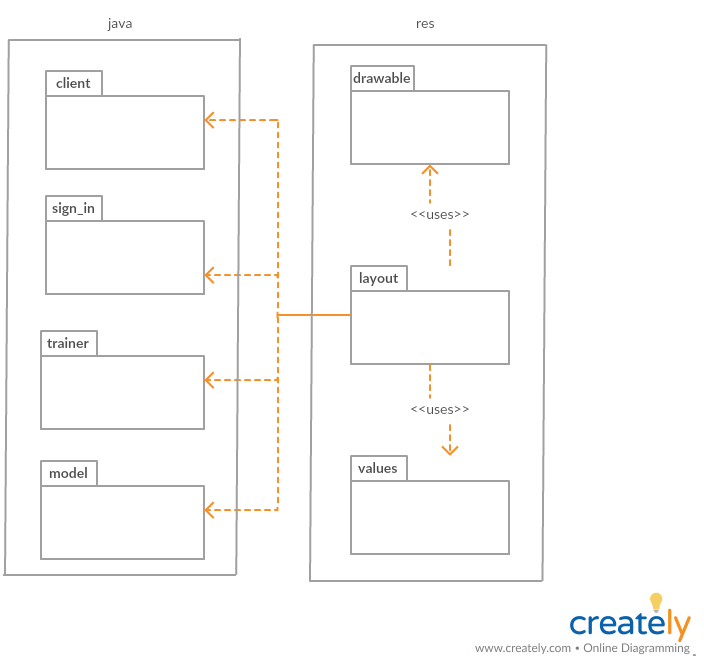
You can hit the ground running, writing only native code, and fly through screens in your app. The SDK handles all network traffic, which saves you from having to write a networking layer. Since the Firebase database is schemaless, you can define your data model in your app without having to make changes to any server code.

**Cross Platform**

Building for multiple platforms using the same database and backend is a breeze with Firebase. They have an SDK for all the popular languages, and they are pretty close to feature parity across all platforms.

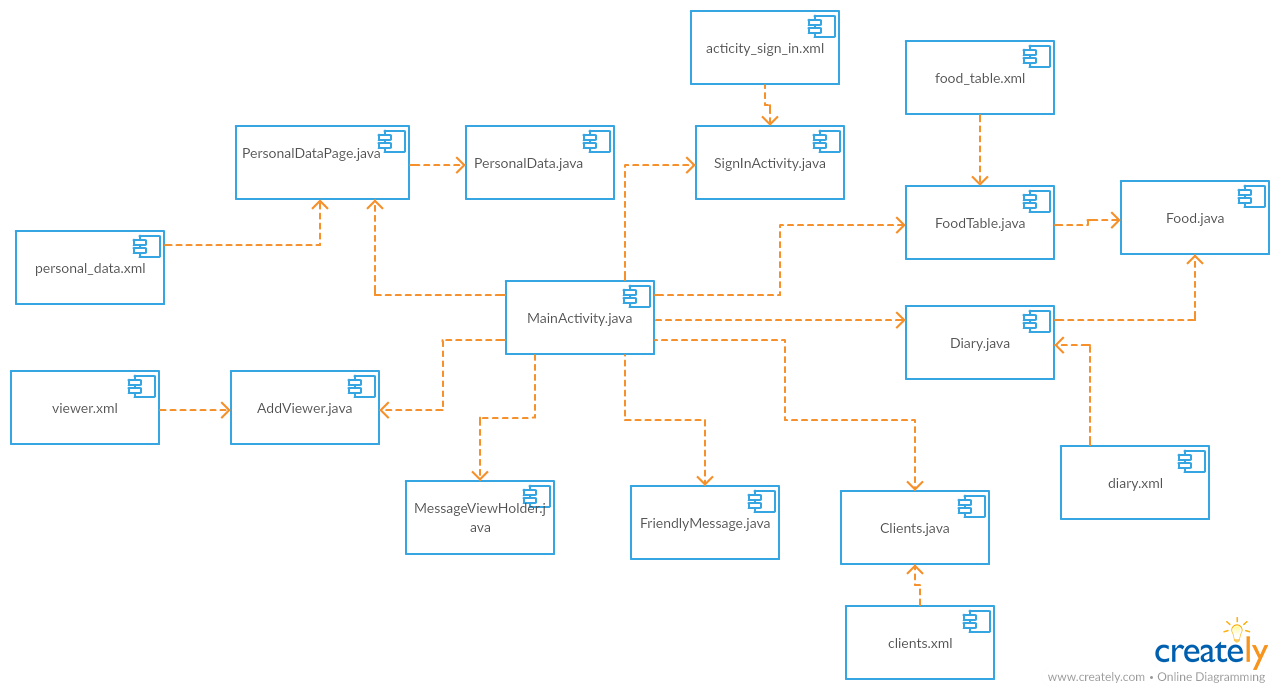
## Package Design

*[Create a package diagram]*



## Component and Deployment Diagrams

*[Create the component and deployment diagrams.]*



# C:\Users\Andreea\Desktop\deployment.png

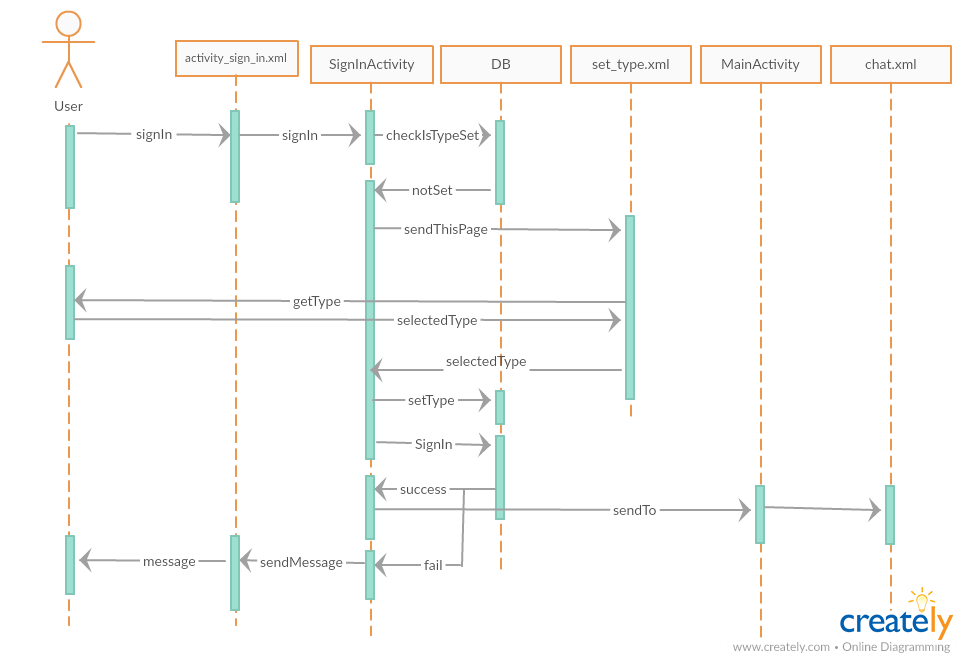
# Elaboration – Iteration 1.2

# Design Model

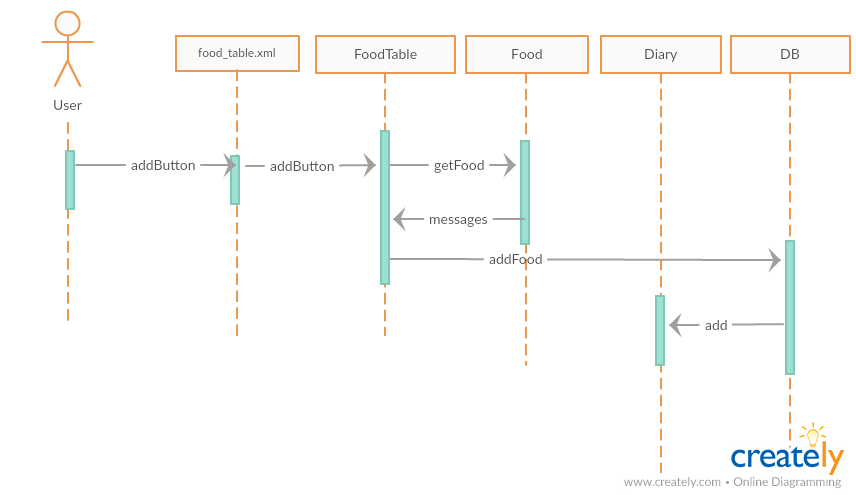
## Dynamic Behavior

*[Create the interaction diagrams (1 sequence, 1 communication diagrams) for 2 relevant scenarios]*

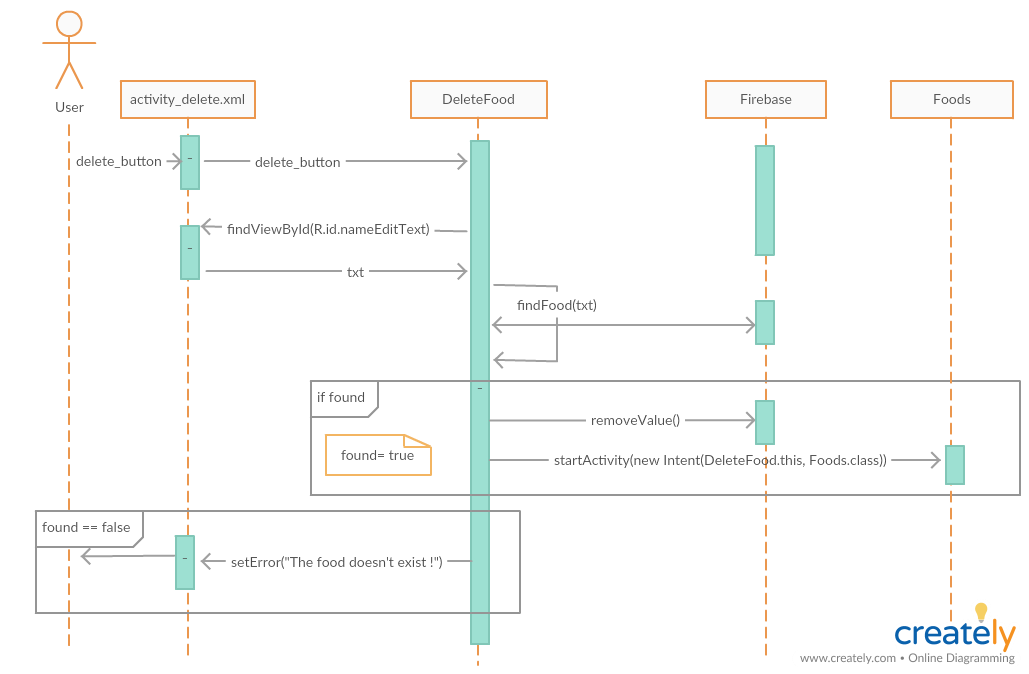
Logging

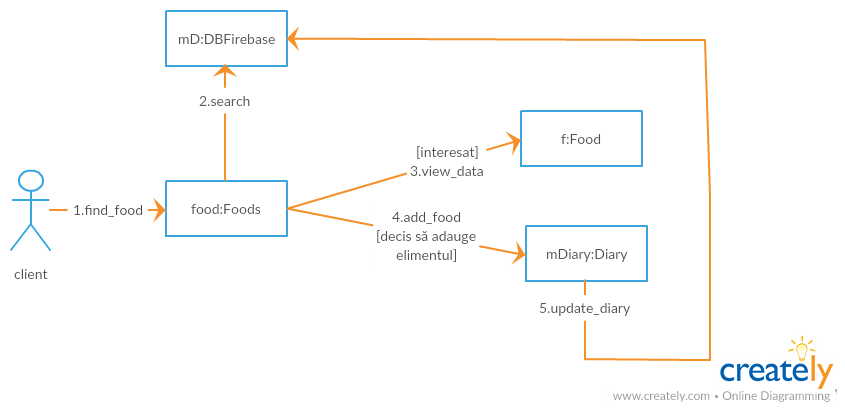


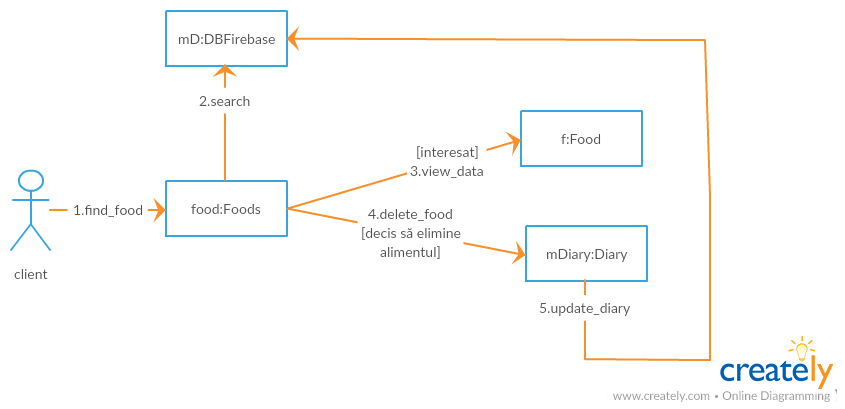
Add a new food to diary



Delete food

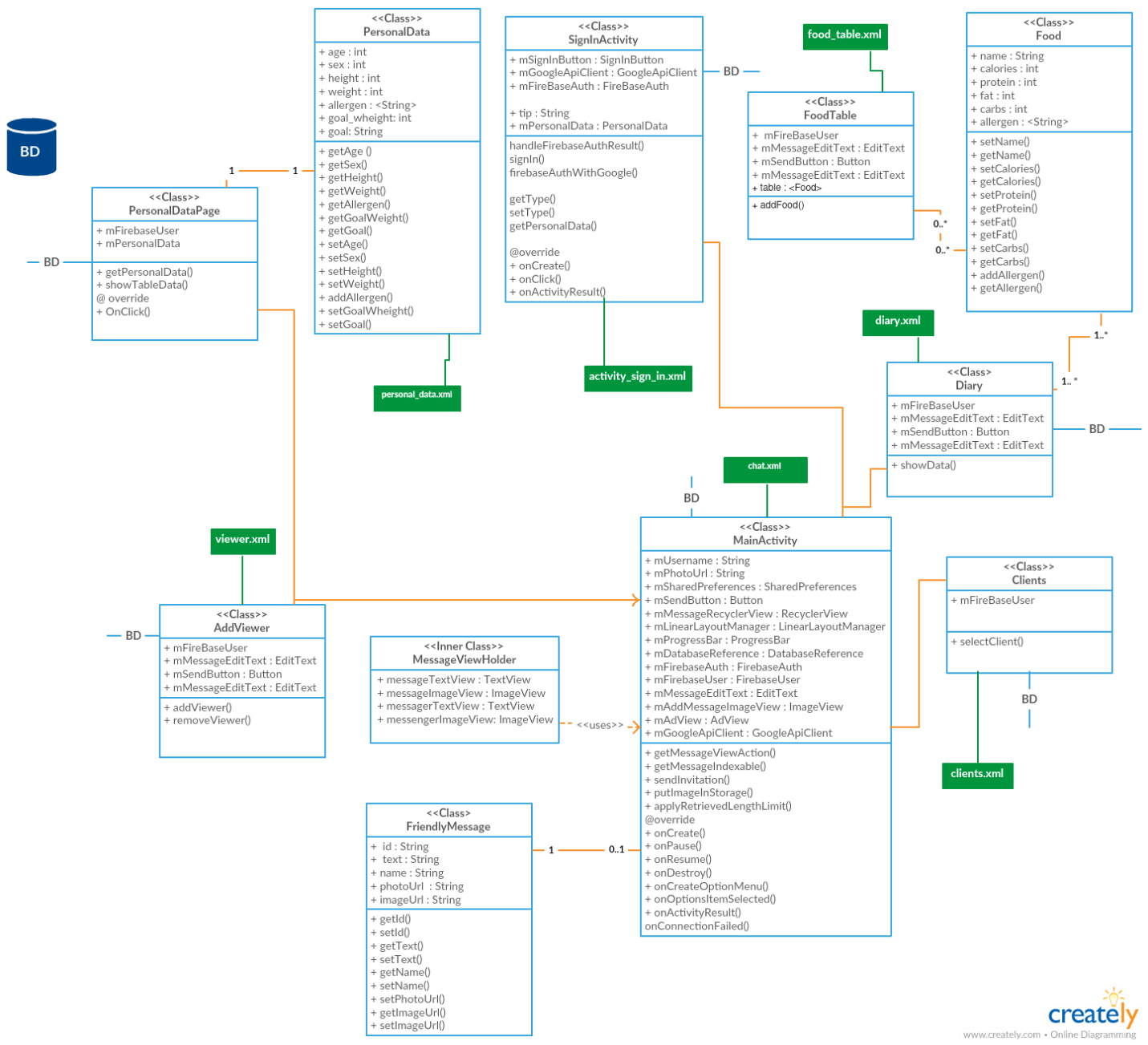


Add new food

Delete food

## Class Design

*[Create the UML class diagram; apply GoF patterns and motivate your choice]*

**

The most used design patterns in Android application are creational patterns, followed by structural design patterns, from which RecyclerViewAdapter is the most common used pattern. Therefore the application uses patterns from this two categories.

**Creational**

* Factory

The user can generate a report with the diary content, in pdf or csv format. The creation of the report is controled by ReportFactory.

|  |
| --- |
| **<<Interface>>**  **Doc** |
|  |
|  |
| + createDoc(String username); |

|  |
| --- |
| **ReportFactory** |
|  |
|  |
| +createDoc(String Type): Class |

creates

|  |
| --- |
| **PDFDoc** |
|  |
|  |
| +createDoc(String username): void  -returnBack(): void  -createPdfWrapper(): void  -onRequestPermissionsResult(): void |

|  |
| --- |
| **CSVDoc** |
|  |
|  |
| +createDoc(String username): void  -writeCsvHeader(): void  -writeCsvData(): void  -returnBack(): void |

* Singleton

When a person signs up as a trainer, the application stores the user’s data into the class Trainer. Because only one instance is needed, singleton pattern is used to control the creation.

|  |
| --- |
| **Trainer** |
| -trainerInstance: Trainer |
| -Trainer() |
| getInstance(): Trainer  +setName(String)  +setId(String)  +getName  +getId |

* Builder (AddFood)

Since the Food class has constructors which are needed, problems can arise. Even if at a first glance it looks fine whith time, if the number of the fields increases, the code becomes harder to read and maintain. More importantly, the code becomes increasingly harder for clients. Which constructor should be used? The one with 4 parameters? The one with 5? What is the default value for those parameters where you don’t pass an explicit value? If you want for Food to set value for macros but not for image, you have to call the constructor that takes all the parameters and pass default value for the image. Additionally, several parameters with the same type can be confusing.

One soluton would be to have an empty constructor and have setters and getters for every atttribute. One problem to this solution, however, is that if you want to create a Food object with values for all the attributes then the object will not have a complete state until all the set methods have been invoked. This means that some part of the application might see this object and assume that is already constructed while that’s not the case.

The secons solution is the builder pattern. The Food constructor is private, therefore this class can not be directly instantiated from the outside. Because the class is used by Firebase to map the data from the database to this class, the attributes cannot be made final and the default constructor must have public access. Still the attributes are private and can be accessed using getters.

|  |
| --- |
| **Food** |
|  |
| +Food()  -Food(Builder) |
| +getName(): String  +getCal():String  +getCarb():String  +getProt():String  +getFat():String +getImgUrl():String  +getAllergens():String   |  | | --- | | **Builder** | | - name : String  -cal : float  -carb : float  -prot : float  -fat : float  -imgUrl : String  -allergens : String | | +Builder(String, Float, Float, Float, Float)  +Builder() | | +setName(String) : Builder  +setCal(float) : Builder  +setProt(float) : Builder  +setFat(float) : Builder  +setImgUrl(String) : Builder  +create() : User | |

**Structural**

The data is listed in a grid manner using RecyclerView. In order to connect the data to the RecyclerView and determine the ViewHolder which is needed to be used to display the data, an adapter is needed.

One solution would be to create a separate class which extends RecyclerView.Adapter. The second, which is used is used in this application, is to create an anonymous inner class. The class must override the next methods:

* + onCreateViewHolder(ViewGroup parent, int viewType) – is called right when the adapter is created and is used to initialize the ViewHolder
  + onBindViewHolder(RecyclerView.ViewHolder holder, int position) – is called for each ViewHolder to bind it to the adapter; this is where the data is passed to the ViewHolder

**Bihevioral**

* Observer

# Data Model

*[Create the data model for the system.]*

The data is stored in a JSON tree, being the structure used by Firebase for data storage. Unlike a SQL database, there are no tables or records. When you add data to the JSON tree, it becomes a node in the existing JSON structure with an associated key. You can provide your own keys, such as user IDs or semantic names, or they can be provided for you using [push()](https://firebase.google.com/docs/reference/js/firebase.database.Reference#push).

In this app the keys are clients, client\_class\_id, dates, user\_name, date\_class\_id, etc. Each key can have one or more children. For example the key clients has as child client\_class\_id and client\_class\_id has as childs activity, allergies, cal, carb, fat, name, prot, scope, sex and weight.

* **clients**
  + **client\_class\_id**
    - activity: String
    - allergies: String
    - cal: Float
    - carb: Float
    - fat: Float
    - name: String
    - prot: Float
    - scope: String
    - sex: String
    - weight: String
* **dates**
  + **user\_name**
    - **date\_class\_id**
      * date: String
* **diary**
  + **user\_name**
    - **date**
      * **food\_class\_id**
        + cal: Float
        + carb: Float
        + fat: Float
        + name: String
        + prot: Float
        + quantity: Float
* **foods**
  + **food\_name**
    - allergy: String
    - cal: Float
    - carb: Float
    - fat: Float
    - name: String
    - photo: String
    - prot: Float
* **messages**
  + **friendlyMessage\_class\_id**
    - name: String
    - photoUrl: String
    - text: String
* **trainers**
  + trainer\_class\_id
  + name: String

# Unit Testing

*[Present the used testing methods and the associated test case scenarios.]*

|  |  |  |
| --- | --- | --- |
| Test scenario | Bug | Solution |
| In Add Food is tested the introduction of a food by typing an incorrect data: 12t | The cases where incorrect data is introduced are not treated. | The exception is captured for treatment.  When capturing, a warning appears on the screen |
| In Diary, you navigate on the screen by scrolling up to see all of the data. | When navigating, the data in the Total table changes.  Adding data to the table was done in the method that displays the data in the Recycler table. This method is dynamic, every time it is called (when browsing through the table), it will add to the Total data again those displayed in the Recycler. | A separate method is used, which is called once, to take the necessary data and display it in the Total table |
| In Generate Document, PDF button is pressed to generate a PDF file with the consumed foods from the diary. No file is generated. | In order to save a file to the sd storage, the app needs permission to write | Read and write permissions are introduced in AndroidManifest. Also, a new method, onRequestPermissionsResult  , is introduced, which veriffies if the app has been granted permission. If not, it will open a dialog box to ask for it. |

# Elaboration – Iteration 2

# Architectural Design Refinement

*[Refine the architectural design: conceptual architecture, package design (consider package design principles), component and deployment diagrams. Motivate the changes that have been made.]*

All the users have access to the foods data, being able to see, add or delete a food. Therefore a change made by one user is seen by the other users. In order to controll the usage of this data, the task of controll have been sent to the firebase, which provides tools for data sharing.

resource access control

# Design Model Refinement

## *[Refine the UML class diagram by applying class design principles and GRASP; motivate your choices. Deliver the updated class diagrams.]*

Low coupling is applied for xml files. Therefore each file  has no knowledge of the other xml files.

# Construction and Transition

# System Testing

*[Describe how you applied integration testing and present the associated test case scenarios.]*

The next steps have been taken:

1. Understand the architecture of your application.
2. Identify the modules
3. Understand what each module does
4. Understand how the data is transferred from one module to another.
5. Understand how the data is entered and received into the system ( entry point and exit point of the application)
6. Segregate the application to suit your testing needs.
7. Identify and create the test conditions
8. Take one condition at a time and write down the test cases.

The choosed method is bottom-up

Bottom-up testing, as the name suggests starts from the lowest or the innermost unit of the application, and gradually moves up. The Integration testing starts from the lowest module and gradually progresses towards the upper modules of the application. This integration continues till all the modules are integrated and the entire application is tested as a single unit.

The reason for choosing this approach is that, if a major fault exists at the lowest unit of the program, it is easier to detect it, and corrective measures can be taken.

Google provides an [Android Testing framework](http://developer.android.com/tools/testing/testing_android.html) that is part of the Android SDK and is built on top of standard JUnit testing extended with a instrumentation framework and Android-specific testing classes. For testing this app, Espresso Test Recorder is used, which is a library for UI instrumentation testing**.** By recording a test scenario, interactions with a device can be recorded and assertions can be added to verify UI elements in particular snapshots of the app. Espresso then takes the saved recording and automatically generates a corresponding UI test that you can run to test the app.

# Future improvements

*[Present future improvements for the system]*

A future improvement can be obtaining a better cohesion.

# Bibliography

<https://developer.android.com/studio/intro/>

<https://www.raizlabs.com/dev/2016/12/firebase-case-study/>

<https://medium.com/indianic/mvp-model-view-presenter-in-android-6487b9fafcbb>

<http://www.blackwasp.co.uk/gofpatterns.aspx>

<https://jlordiales.me/2012/12/13/the-builder-pattern-in-practice/>

<https://www.thedroidsonroids.com/blog/design-patterns-in-android-builder>

<https://developer.android.com/studio/test/espresso-test-recorder>