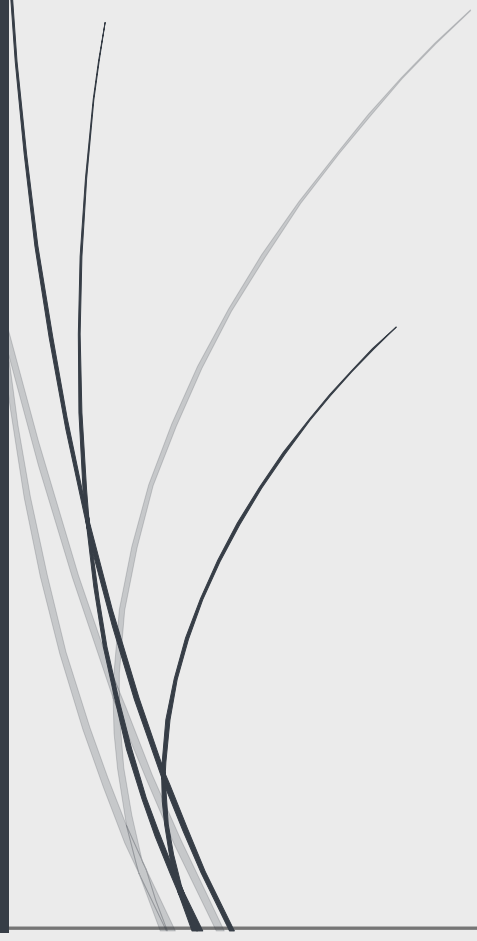
2023/2024

Meal Planner

Technical Documentation



ISPW:

David Pimentel Montes : 54443167M ERASMUS Student

INDEX

1. SOFTWARE REQUIREMENT SPECIFICATION

1.1. INTRODUCTION

1.1.1. AIM OF THE DOCUMENT

The main point of the following document is present objectively the fundamental requirements and specific ones of the system *Meal Planner*. So, we will be doing a review of all the important internal and external aspects of the system with the help of some diagrams.

1.1.2. OVERVIEW OF THE DEFINED SYSTEM

The system here to expose is an application that helps the user to organize all their meals in a calendar, also it helps the user to generate the shopping list and show some statistics that can be very useful for the user. The system has some recipes implemented but the user can also create as much as they want. So, the user can generate their personal and personalized meal schedule. Moreover, the system has a recipe searcher to let the user search the recipes that best fit to their desires and add them to the calendar.

1.1.3. HARDWARE AND SOFTWARE REQUIREMENTS

The basic hardware requirements of the system are a computer that can run JAR files, as well as some space in the disc to save the files with the personal information if the offline version is on.

In the case of the basic software requirements of the system are the oracle JDK SE 21 correctly installed on it to be able to run properly the JAR file, as well as a server or a local server that can holds the DBMS so in the online version of the system the information can be obtained by queries in SQL to the server or local server correctly.

1.1.4. RELATED SYSTEMS, PROS AND CONS

Other systems related to the one described here are:

CONSUM

It is a web page for the organization of the meals of the week. Some of the pros of this system are that it can be personal personalized to a specific user as the number of diners. But as cons it has that you cannot create new recipes and add your own recipes to it, you cannot search for recipes it generates it randomly and the interface it’s quite counterintuitive.

NOTRITION VALUE

This system is a web page for the organization of the meals of the week. But we are in front of a very rudimentary GUI that can lead to the user to get lost. It also has many functional errors and inconsistencies. But in its favor has that the DB that uses is a very big and well implemented one.

1.2. USER STORIES

1.2.1. US-1

As a User **I want** to have a visual form to overview my meals of the day **so that** I can have a quick look at them and change or add in case of need.

1.2.2. US-2

As a User **I want** to create new recipes **so that** I will be able to add my personal recipes to my personal calendar.

1.2.3. US-3

As a User **I want** to have a recipe searcher **so that** I can search for the recipe that best fit to my requirements.

1.3. FUNCTIONAL REQUIREMENTS

1.3.1. FR-1

The system will allow you to log in to your personal account in order to enter in your profile and connect your data.

1.3.2. FR-2

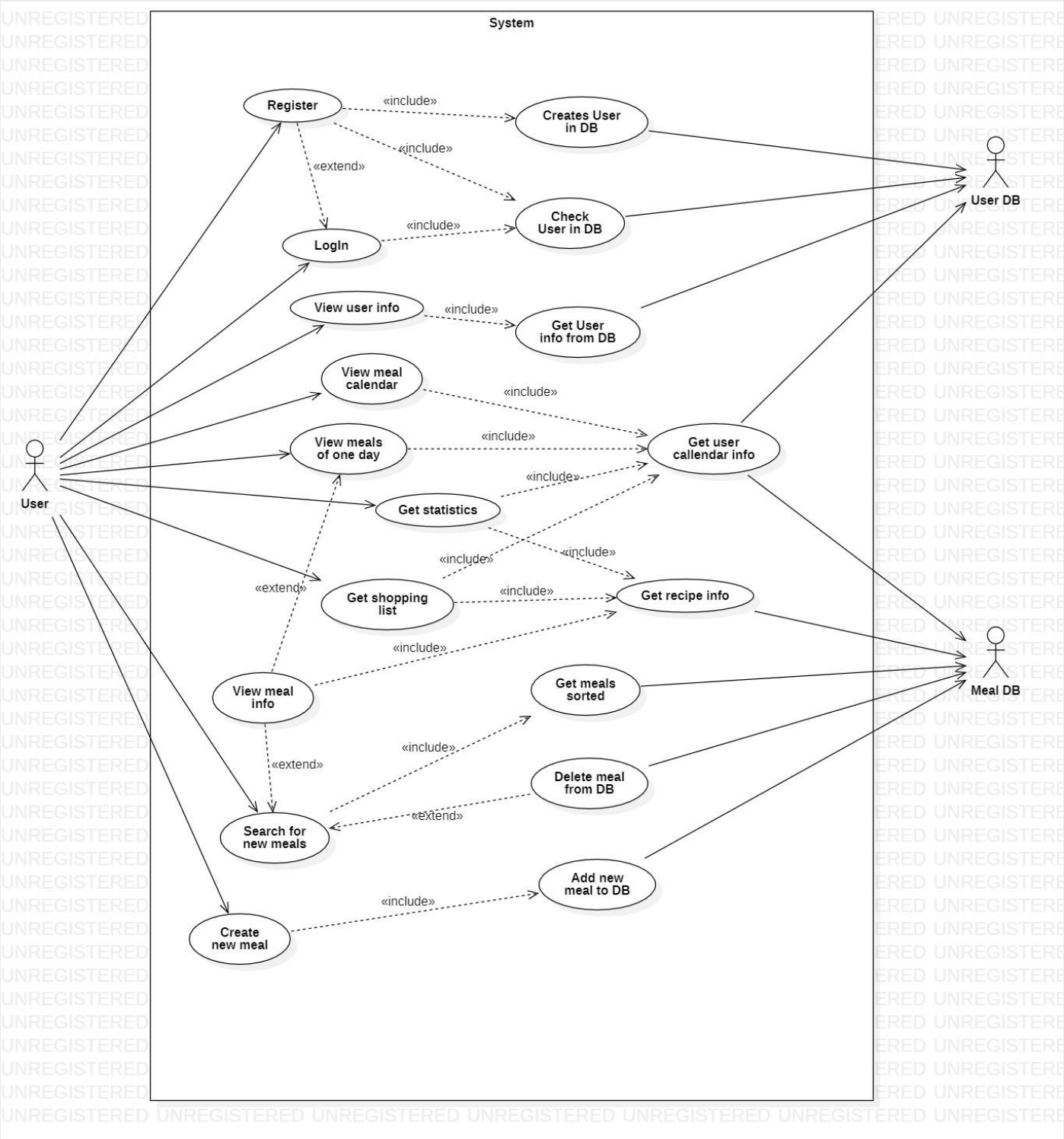
The system will give you access to a wide variety of preloaded and other user's recipes.

1.3.3. FR-3

The system will give you access to a calendar to organize the meals you want to eat during the week.

1.4. USE CASES

1.4.1. OVERVIEW DIAGRAM



1.4.2. INTERNAL STEPS

Name: View the user meal calendar

1) The system gets all the information saved in the User Data Base related to the personal calendar.
2) The system creates the calendar with all the information saved in it.

3) The User can navigate between the weeks to access a more distant day.
4) The User chooses the day he/she wishes to obtain more detailed information on the meal plan.
5) The system shows a schedule of the meals for that day.
6) The User can view/modify/delete the meals in the schedule.
7) The User can obtain more detailed information in one specific meal to get the steps of the recipe or the ingredients.
8) The system saves all the changes made by the User in his/her personal calendar.

Extensions:

2a. The User has no saved calendar data: The system creates an empty calendar for the User.

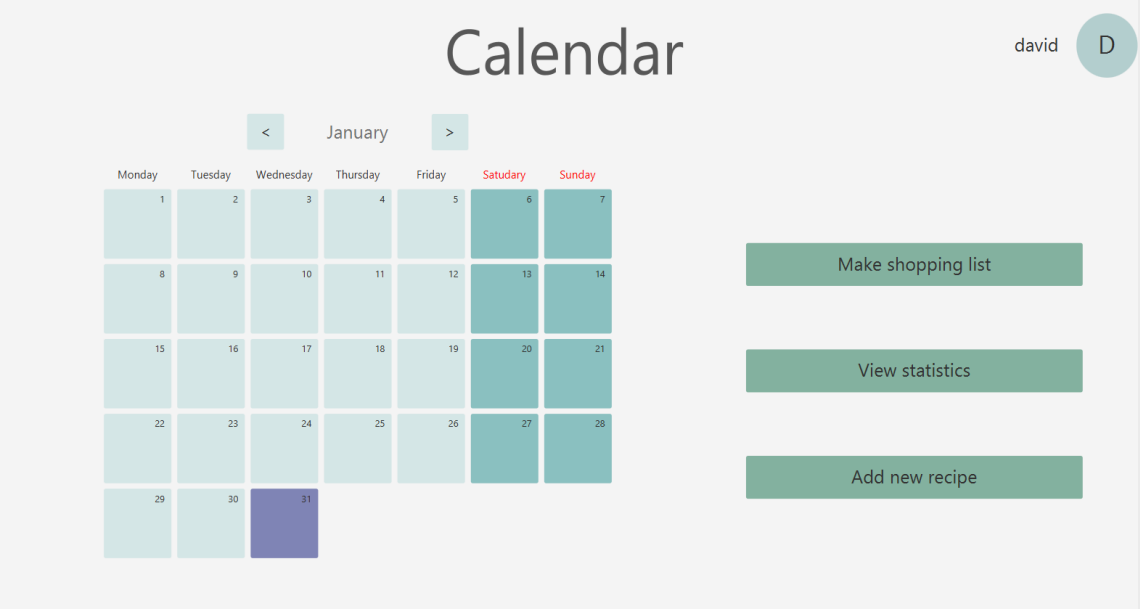
6a. The User wants to view a meal: The system shows a preview of the selected recipe so that the user can check how it is made and what does it needs.

6b. The User wants to modify a meal: The system provides a form so that the User can choose the specifications of the recipe to add to their day schedule.

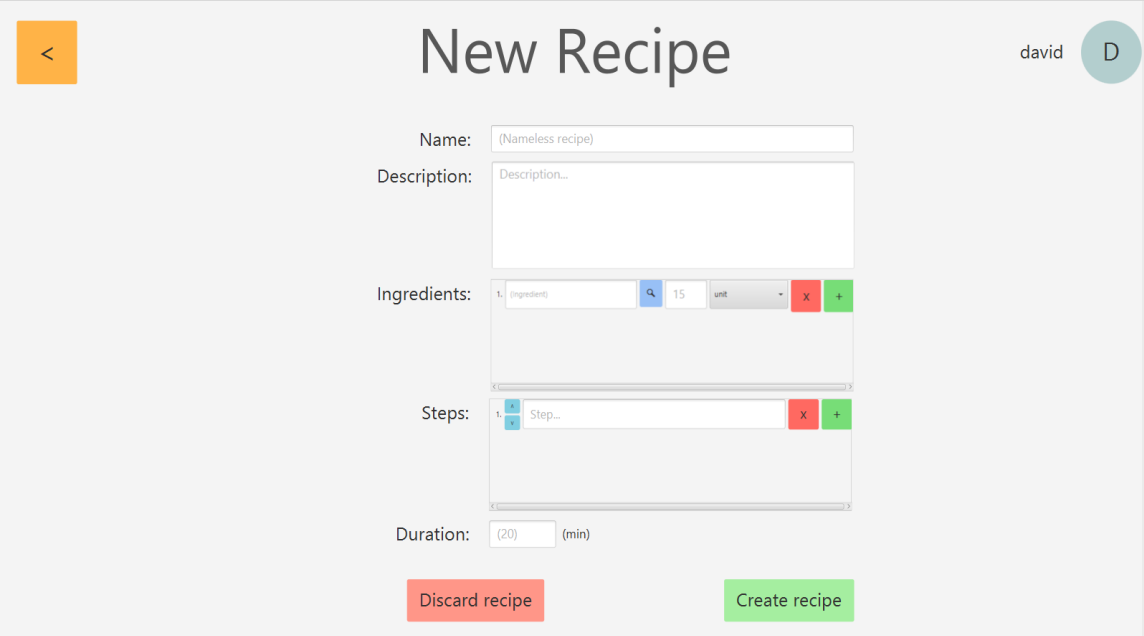
6c. The User wants to delete a meal: The system deletes that meal form de personal meal calendar of the User.

2. STORYBOARDS

The first Storyboard is the main menu screen, where there Will be show a calendar of the month with the information of the meals that the user will take. As well as buttons to grant the user access to the rest of the functionalities:



The second Storyboard is the form to generate a new recipe and add it to the DB, to create your own recipes and to use them in your calendar meal planner:



3. DESIGN

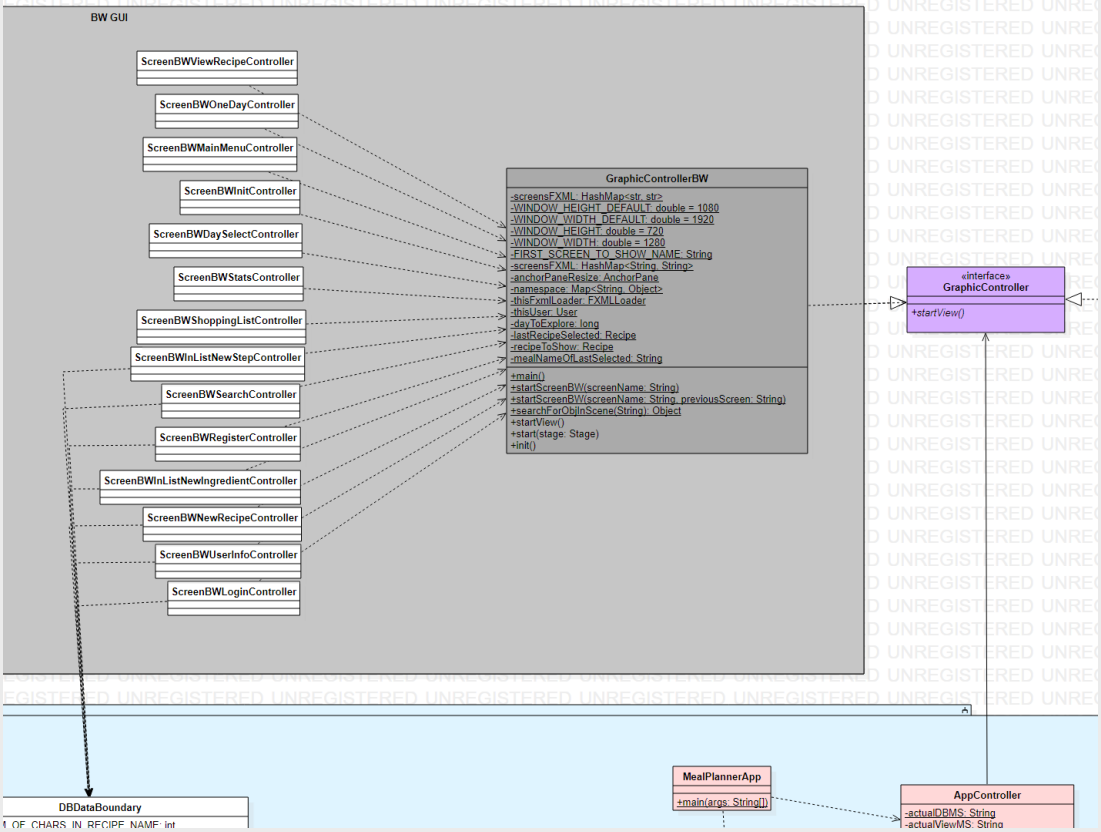
In this part the principal internal characteristics of the system will be exposed, with some diagrams to express it.

3.1. CLASS DIAGRAM

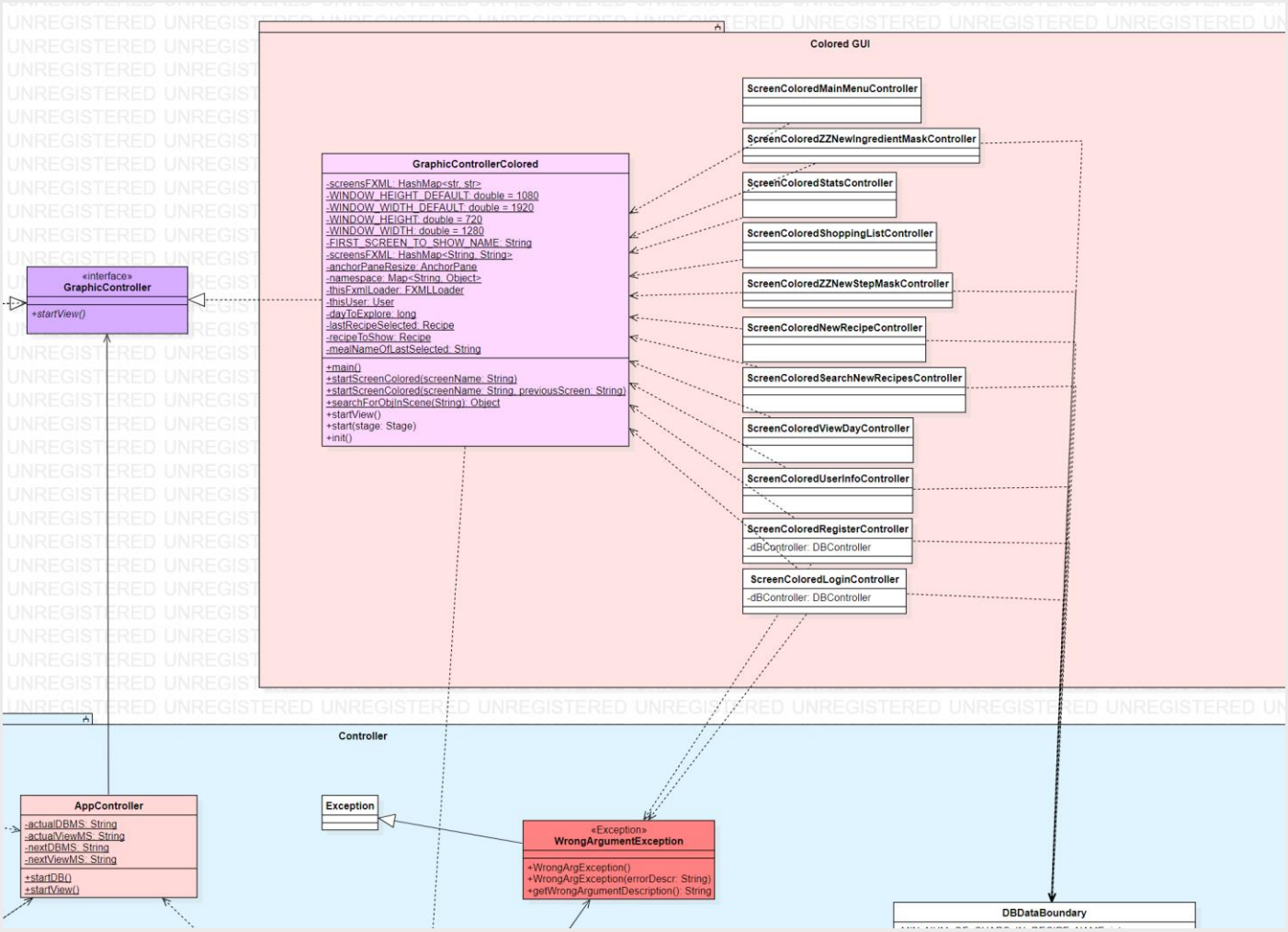
3.1.1. VOPC (ANALYSIS)

This diagram is attached near to this document to be able to have a closer look to it.

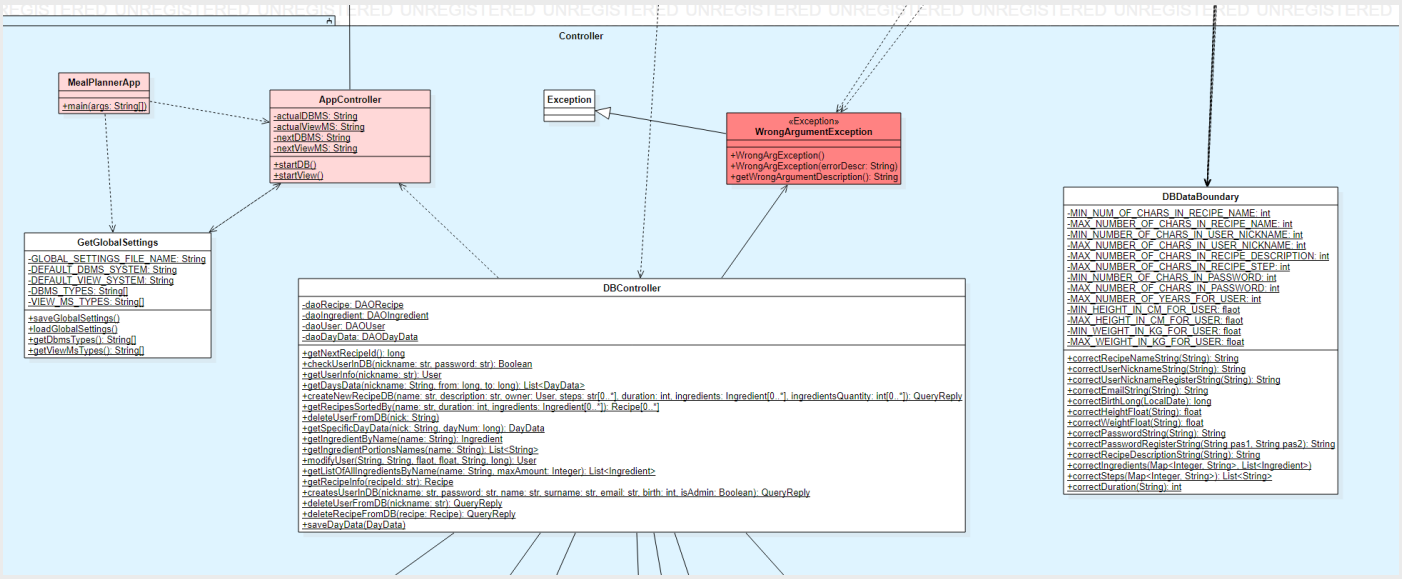
The *BW GUI* part:



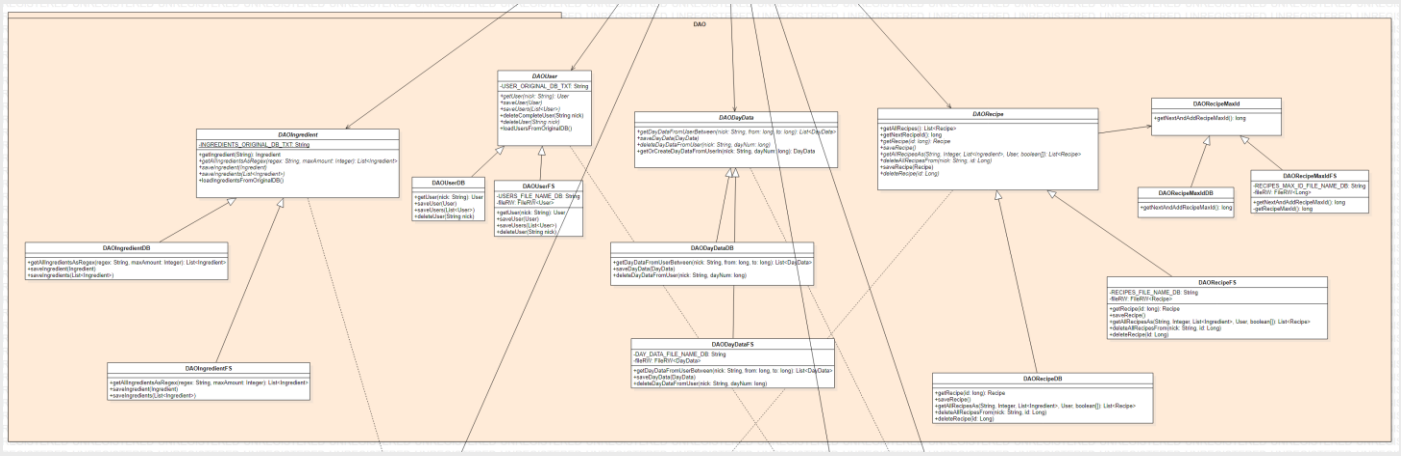
The *Colored GUI* part:



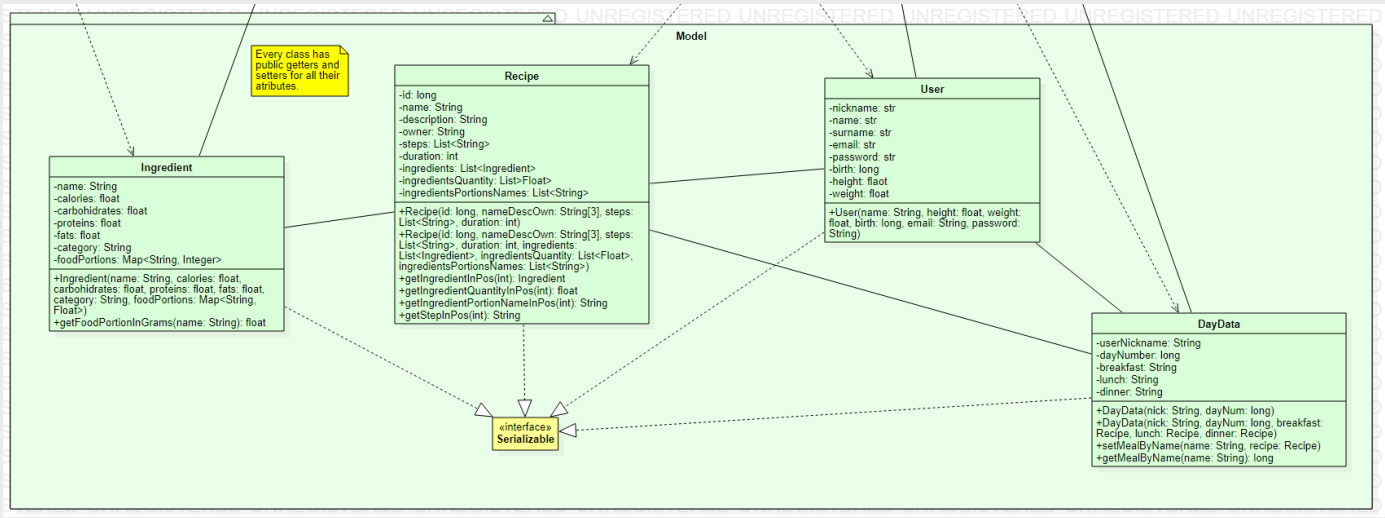
The *Controller* part:



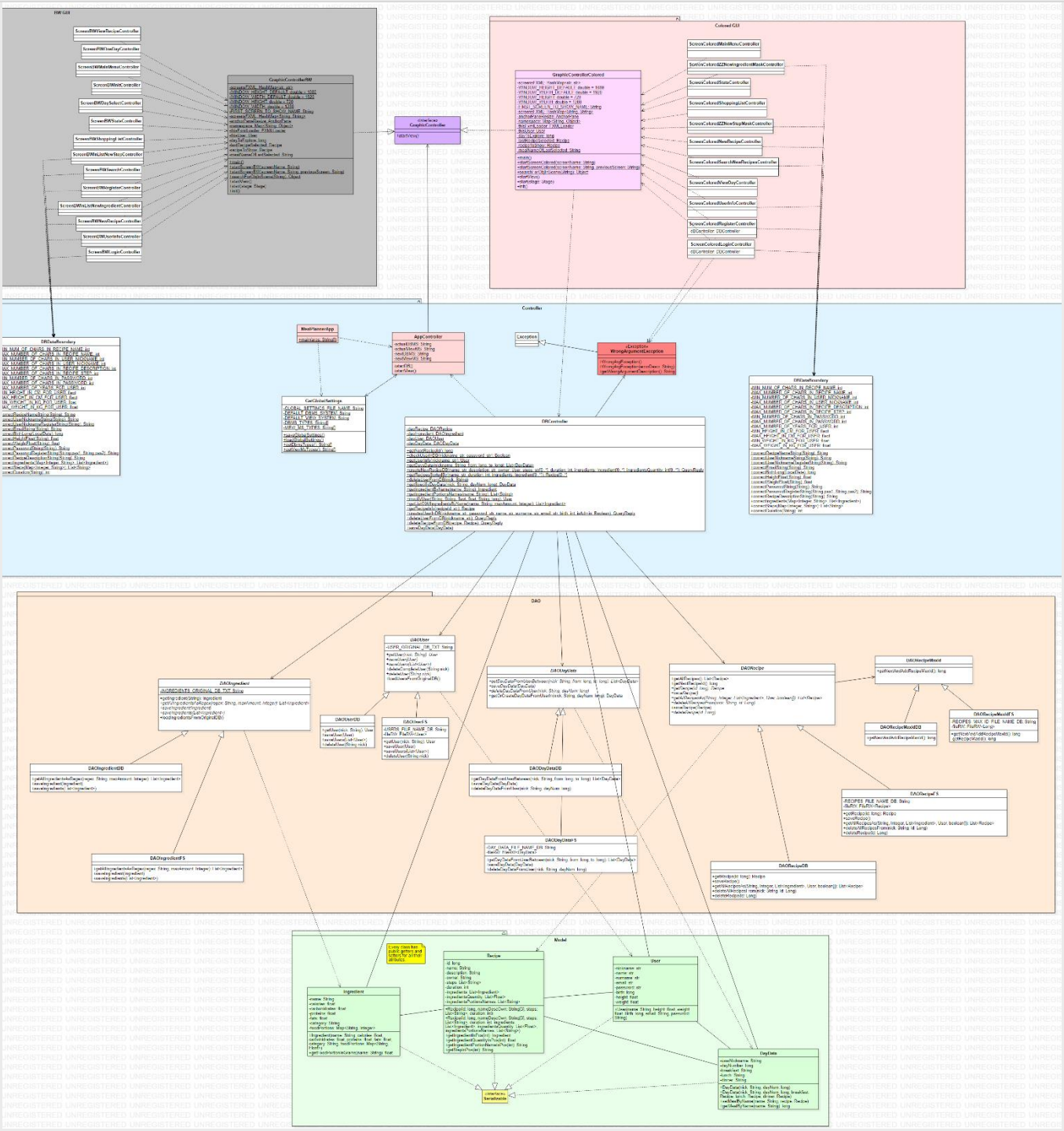
The *DAO* part:



The *Model* part:



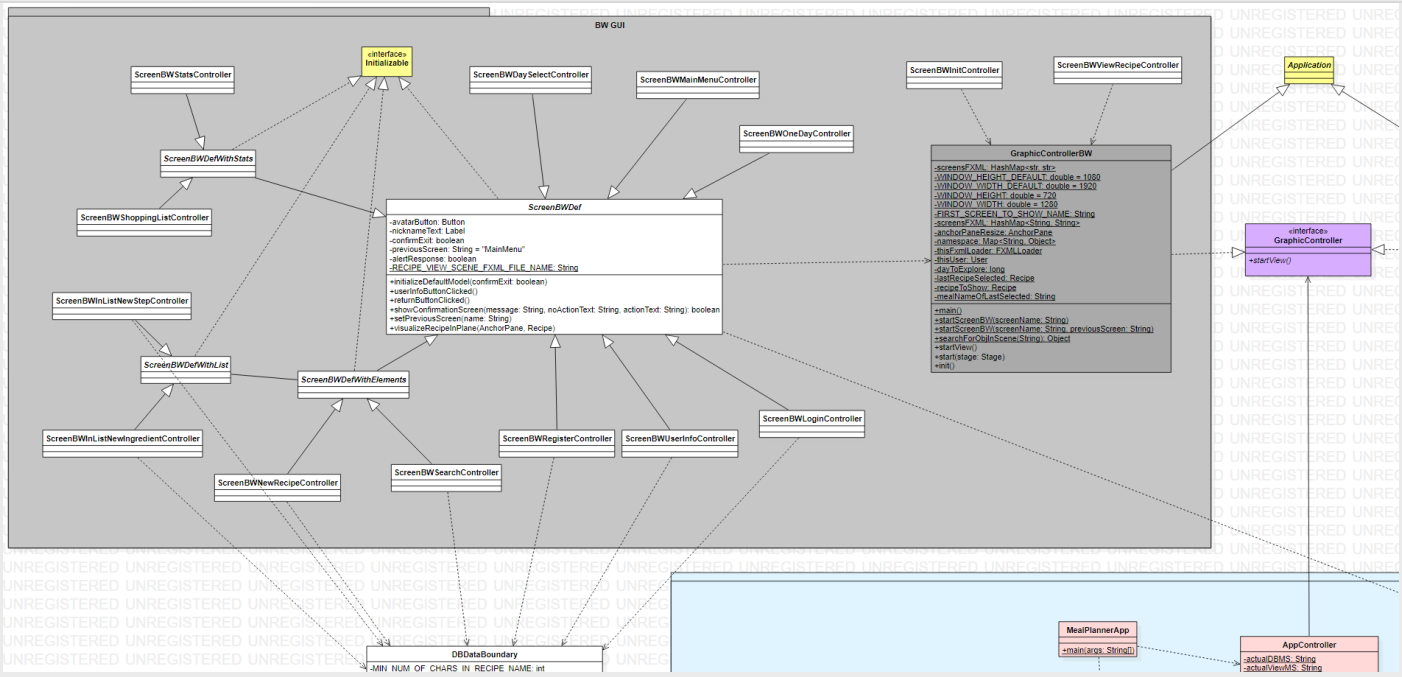
The global view of the class diagram:



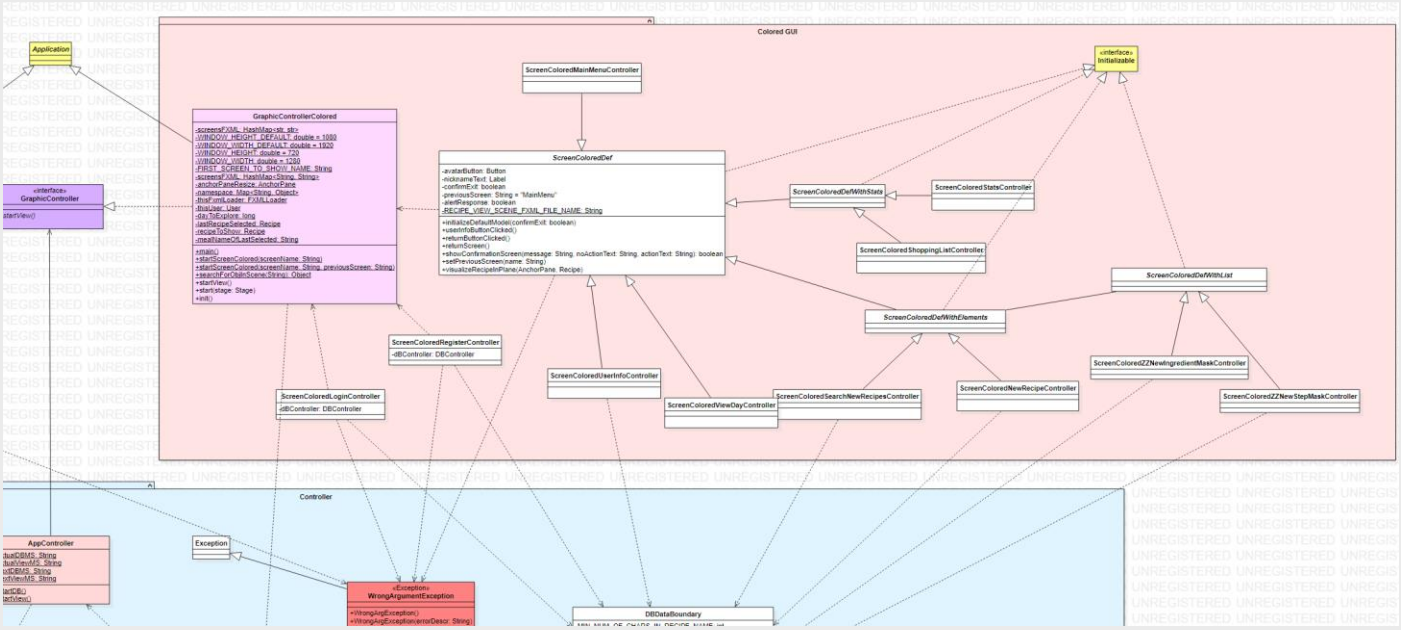
3.1.2. DESIGN-LEVEL DIAGRAM

This diagram is attached near to this document to be able to have a closer look to it.

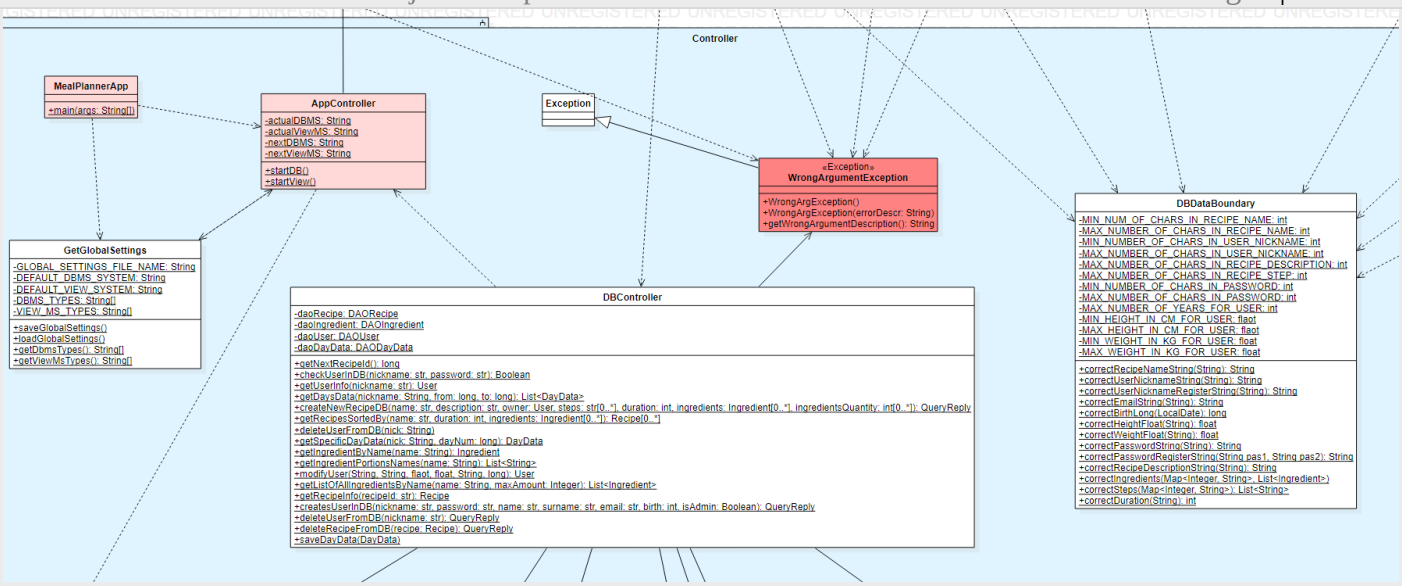
The *BW GUI* part:



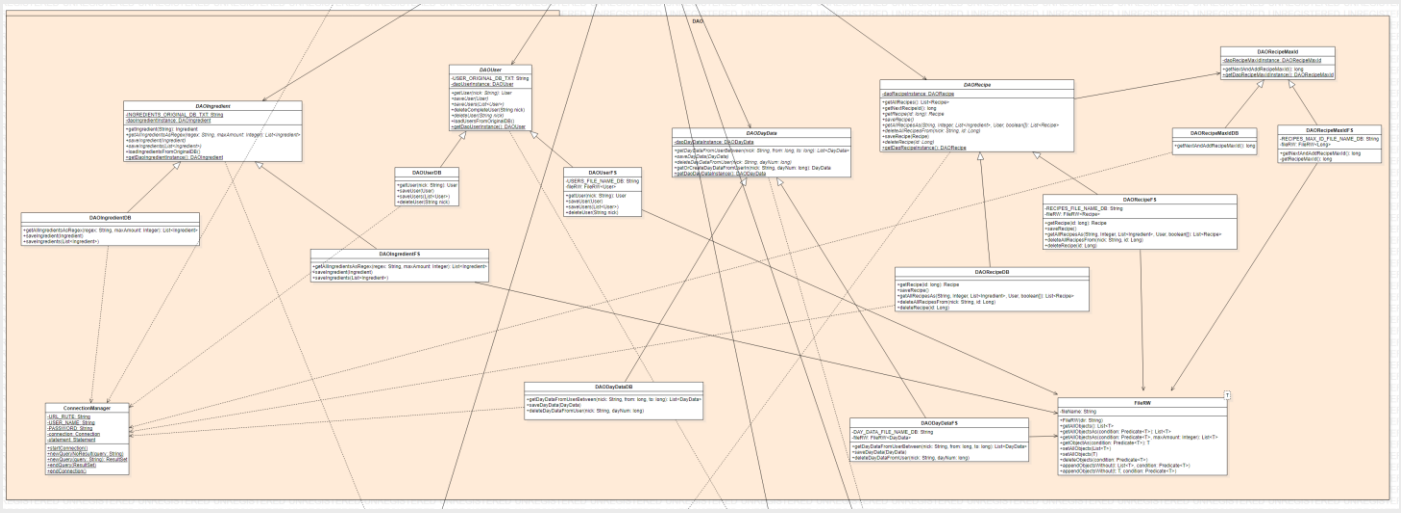
The *Colored GUI* part:



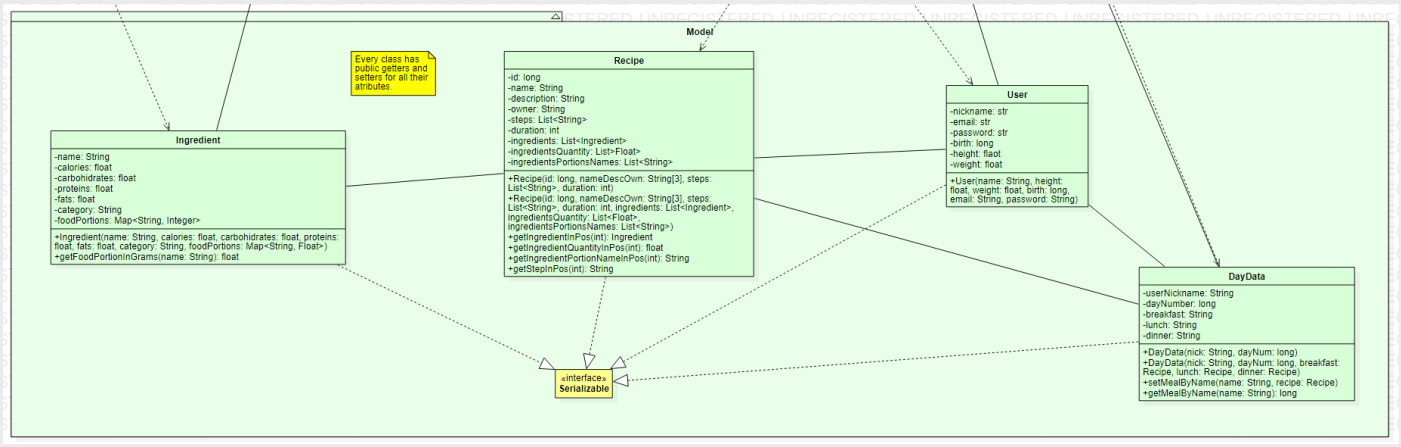
The *Controller* part:



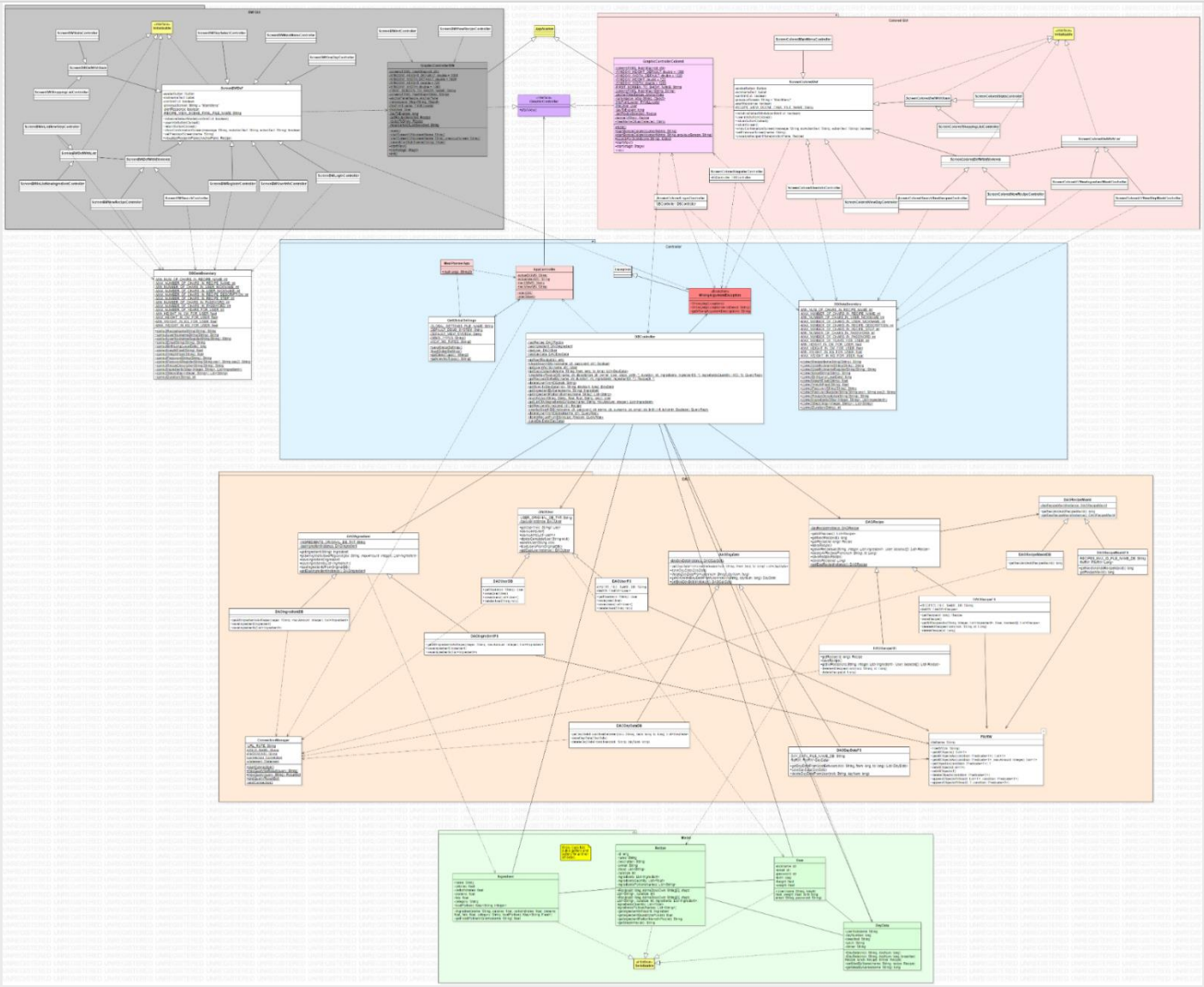
The *DAO* part:



The *Model* part:



And the global view of the class diagram complete is the following:



3.2. DESIGN PATTERNS

In the context of the system it has been implemented the *Singleton* pattern, in which the main point of it is to make sure that there is one and only one instance of one type of object, so all the actions in that object are correctly executed without incoherence. Without the need to be send as parameter in every constructor of the classes that use it. This pattern was a big help to crate the DAO classes so every class that wants to access to some sort of information from the data base don't need to create another instance of it or have it as parameter.

The classes *DAODayData*, *DAOIngredient*, *DAORecipe*, *DAORecipeMaxId* y *DAOUser* implements this pattern.

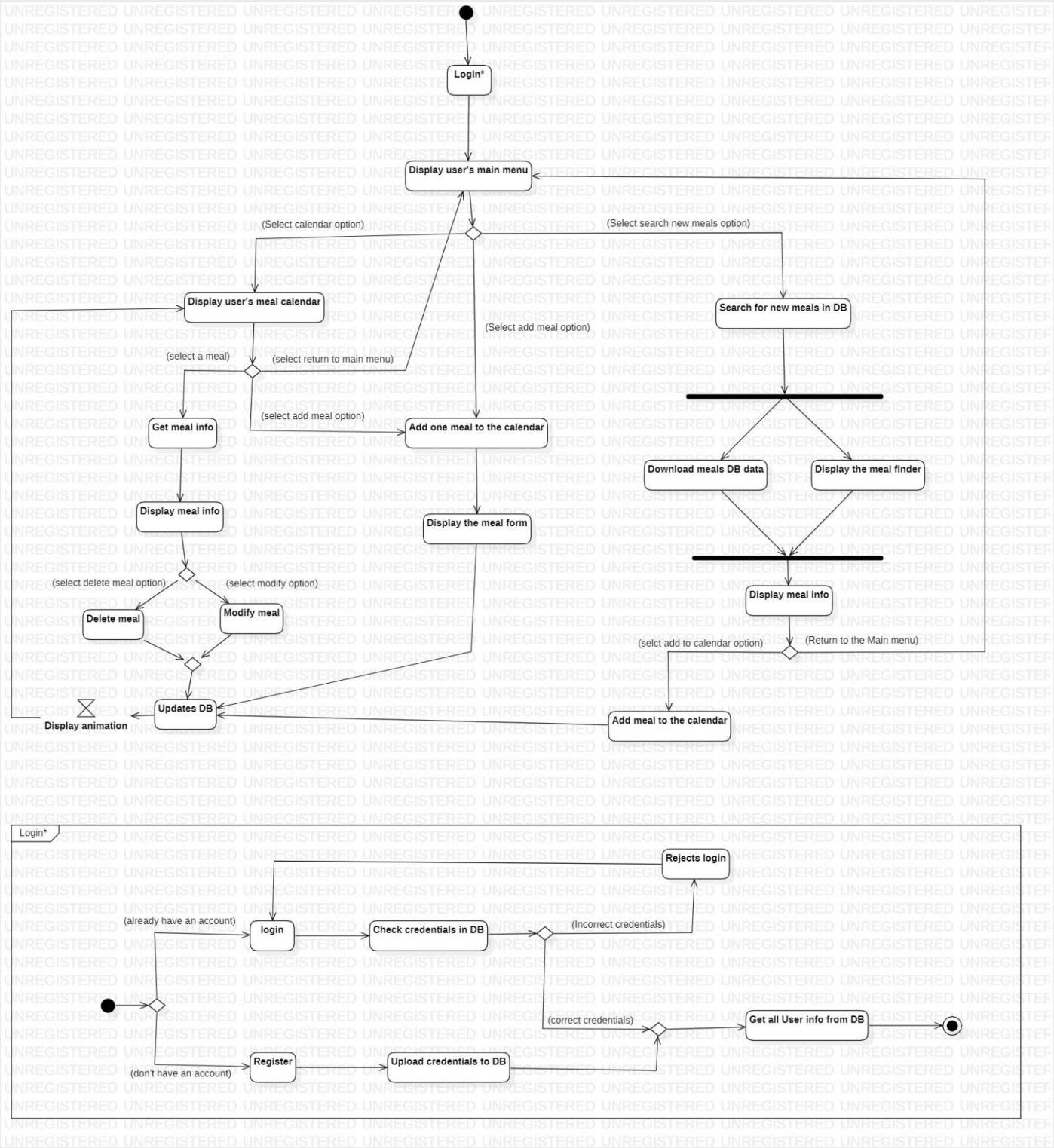
Moreover, we also consider the use of other patterns such as the *Factory Method* in the context of create the instances of the entities bean classes, but due to the simplicity of these classes it was consider not the optimal solution and not the best implementation.

Another pattern that was consider was the *Decorator* pattern to use it in the context of the GUI interface so that the screens were surrounded by other screens inheriting the queries, but in the context of the system was going to generate more complexity than the one that should delete.

3.3. ACTIVITY DIAGRAM

This diagram is attached near to this document to be able to have a closer look to it.

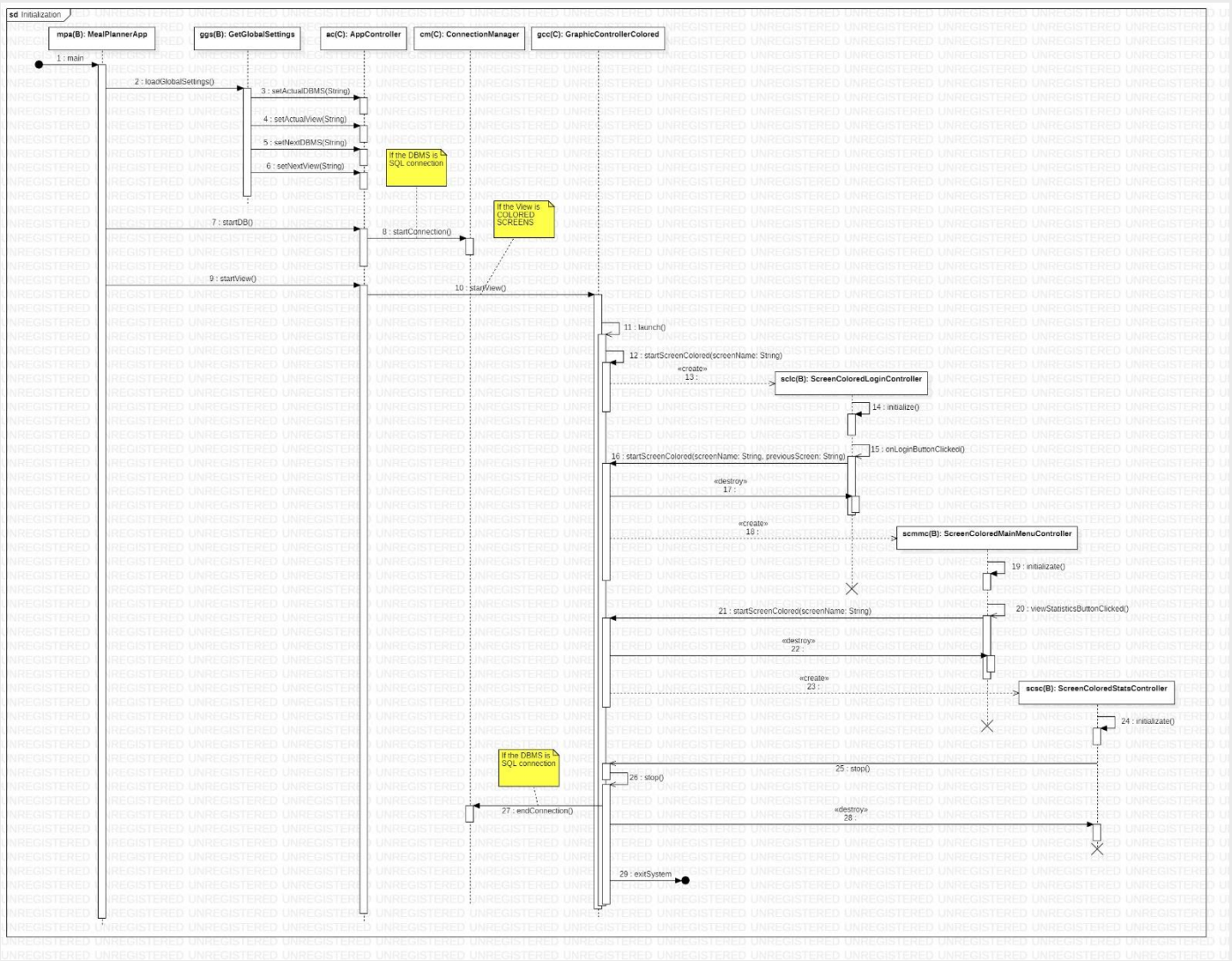
The following activity diagram describes the main branch of the login/register and the main functionalities of the system:



3.4. SEQUENCE DIAGRAM

This diagram is attached near to this document to be able to have a closer look to it.

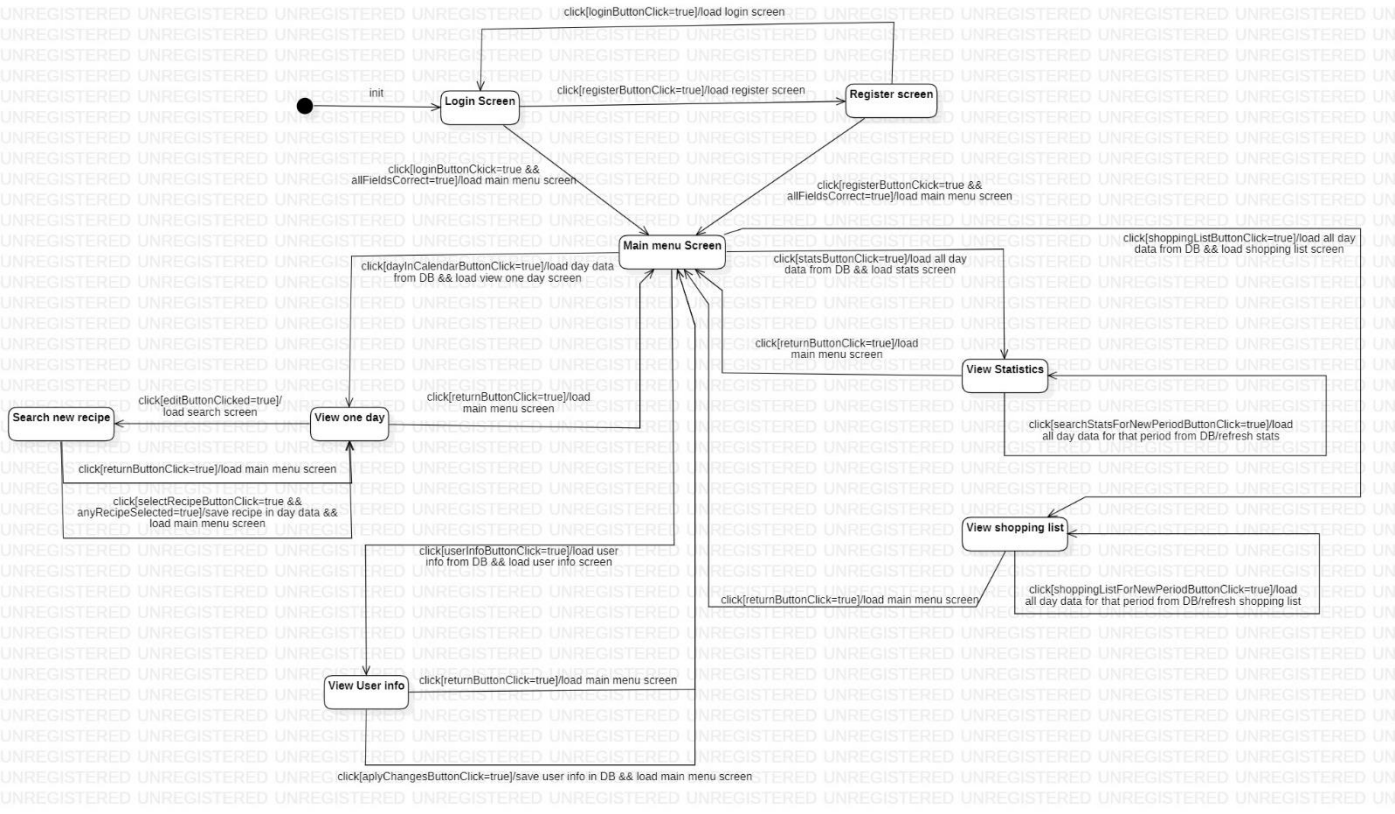
The main branch of the system is described in the following sequence diagram, in which is shown the initialization of the controllers of the system in the case of the DBMS of SQL and the view of colored form, and logs in, and goes to the statistics screen and then they close the window.



3.5. STATE DIAGRAM

This diagram is attached near to this document to be able to have a closer look to it.

The following state diagram is the main functionalities of the system:



4. TESTING

In the testing part we have three test classes in which is tested some of the most important or relevant functions of the classes.

In first place, we have the *DBControllerTest*, that it oversees testing the *DBController* class and the functions: *checkUserInDB(), getUserInfo()* and *deleteUserFromDB().*

In second place we have the *DAOUserTest*, that it oversees testing the *DAOUser* class and the function: *loadUsersFromOriginalDB().*

In third and last place we have the *DBDataBoundaryTest*, that oversees testing the *DBDataBoundary* class and the functions: *correctRecipeNameString(), correctUserNicknameString(), correctEmailString(), correctBirthLong(), correctHeightFloat()* and *correctDuration().*

5. EXCEPTIONS

In Handling Exceptions part, we have two types of handling in the system.

The first type is the dedicated class that inherit the functionalities of *Exception* class, in this case this class is *WrongArgException*. It oversees handling the exceptions that occurs when the user inserts a wrong formatted string into the fields of any of the forms or the searcher screen. It is used by *DBDataBoundary*, to check whether a string fits the restrictions of their type.

The second type of handling exceptions is not just the catching and throwing the exception to superior functions, is the realization of the correct coding in the case of any of the tried scripts fails. In this case we have the example of reading information from a file, in the case of anything of the reading file part fails, the resulting entity will be a null entity to let the other classes that it was impossible to search for that entity.

6. DATA BASES

We have two types of data bases in the described system:

6.1. FILE SYSTEM

The file system is a set of files with the information of the entities. This file distribution is the following:

1) *fileData/connectionSettings/connectionSettings.txt*

   It contains the direction to the server, the username, and the password to connect to it.

2) *fileData/fileDataBase/dayDataFromUserInfo_DB.dayDataInfo*
   It is a binary file with the serialization of the *DayData* objects.
3) *fileData/fileDataBase/ ingredientsInfo_DB.ingredientsInfo*

   It is a binary file with the serialization of the *Ingredient* objects.

4) *fileData/fileDataBase/ recipesInfo_DB.recipesInfo*
   It is a binary file with the serialization of the *Recipe* objects.
5) *fileData/fileDataBase/ recipesMaxId_DB.recipesMaxId*
   It is a binary file with the serialization of the max id available for recipes.
6) *fileData/fileDataBase/ usersInfo_DB.usersInfo*
   It is a binary file with the serialization of the *User* objects.
7) *fileData/globalSettings/globalSettings.globalSettings*
   It is a binary file with the serialization of the GUI and the DBMS to use.
8) *fileData/ originalDataToDB/ ingredientsOriginalDB.txt*
   It is the original DB for the ingredients.
9) *fileData/ originalDataToDB/ usersOriginalDB.txt*
   It is the original DB for the users.

6.2. MYSQL

The other type of DBMS in the system is the connection to a *MySQL server* to access the information. Internally this SQL must follow the struct specified in the file ***MySQL_Code.sql***.

7. SONAR CLOUD

To check the correct behavior of this project, it was used *SonarCloud* system. This system points out the possible bugs, possible vulnerabilities, and review that the functions are well implemented. You can check the results of the project in the following link: https://sonarcloud.io/project/activity?id=daaaviid-03_MealPlannerFX.