Eino Miettinen, 25.4.2020

652898

Computer Science

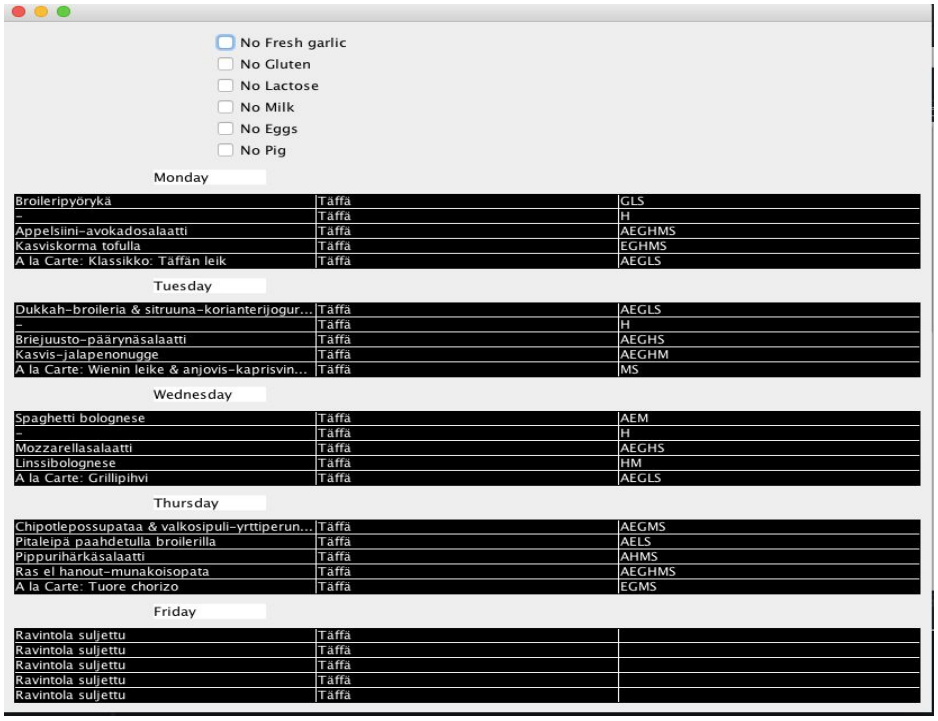
Campus menu application

1. **General description**

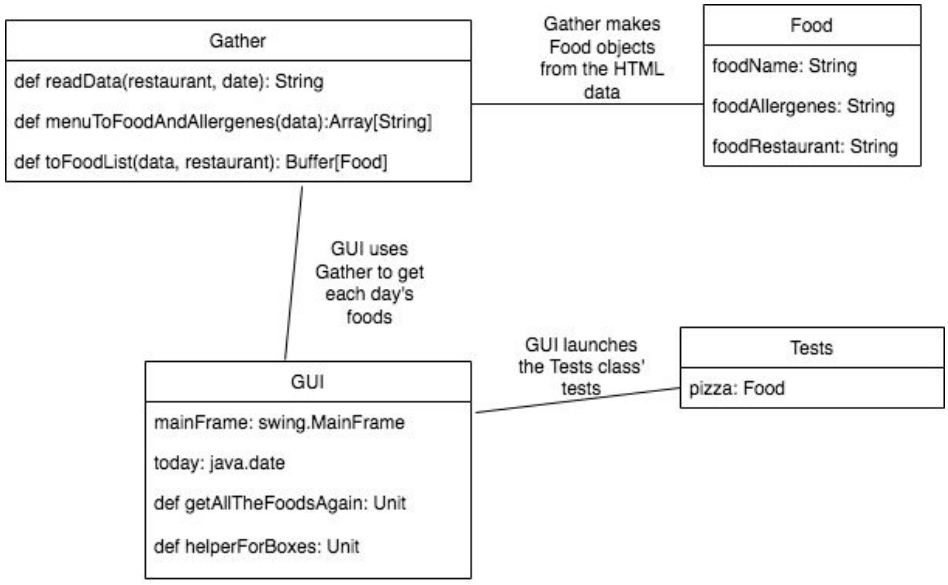
   The application's main goal is to show each weekday's menus for the restaurant Täffä. Due to the corona situtation the plan changed to only show Täffä's foods instead of 4 different restaurants in Otaniemi. The user will see a graphical user interface when he launches the application. The user will be able to filter foods according to their allergenes. The project was done with 'intermediate' level meaning it has a GUI. It also has the filtering feature so it is a bit harder than that.

2. **User interface**

   The UI is very simple for the user to use. He can only click the filter 'checkButtons' and apart from that the user's goal is only check what food is served each day which the user will be able to see all the time during the usage.



3. **Program structure**



The projects consists of four different classes including the Tests class.

Gather

The gather was three main methods: readData, menuToFoodAndAllergenes and toFoodList. readData method reads data from the server with a GET request. It receives HTML and outputs all the HTML lines it has gotten from the server. The request was made using a library called io and its class Source. menuToFoodAndAllergenes method parses the HTML data into a array of strings that are the meant to be the foods and allergenes. It drops everything before the word ‘title’ in the HTML data because after that the data shows all the foods and their allergenes. I will explain more about this algorithm in the following ‘Algorithms’ section. toFoodList makes the parsed data into Food objects.

Food

Food represents each food. It consists of three main variables: foodName, foodAllergenes, foodRestaurant. foodRestaurant changes the URL’s restaurant’s id number (3 in the case of Täffä) into more human readable version or throws an exception in the case of the restaurant doesn’t exist. Other than that the Food class has ‘get’ methods for each variable so they cannot be modified by the other classes, only ‘viewed’.

GUI

The GUI class uses a library called Swing in order to show all the data to the user. It's has a variable called 'today' and its purpose it is to set the date to the user's current week's monday in order to get all the foods for that same week. After that there is a variable called mainFrame which creates the whole UI. In the code we put all the visual checkBoxes etc. as the contents of the mainFrame and they will be shown in the UI. It consists of checkBoxes, tables and textAreas which all are Swing library's objects. MainFrame also has reactions which act everytime the user does something in the UI, for example clicks a checkBox. getAllTheFoodsAgain method's purpose is to get all the foods data. It is called after the launch of the app and everytime the user sets the checkBox to be 'not checked'. helperForBoxes method does all the work regarding filtering data from the tables, more information about this algorithm in the algorithms section.

Tests

4. **Algorithms**

The most important algorithms of this project are in the Gather and UI classes. Let's start with the Gather class:

Gather: menuToFoodAndAllergenes:

```
def menuToFoodAndAllergenes(data:String):Array[String] = {
    var todaysFoods = (data.split("title").drop(1).mkString)
    var eachFood = todaysFoods.split("}")

    //The first food is differently formatted compared to the others so it
has to be done manually and the rest automatically:
    var first = eachFood(0).drop(2)
                           .takeWhile(_!=']')
                           .dropRight(1)

    var otherFoods = eachFood.drop(1)
                             .map(x=>x.drop(5)
                             .takeWhile(_!=']'))
    //and add the arrays together
    return Array(first)++otherFoods
  }
```

It gets as an input a string of all the HTML data in the website. It drops all the unused data until the key word 'title' and then splits the foods in differenct sections. The first food part is a bit differently formatted than the others so it has to be done manually in the

variable called ‘first’. It drops 2 characters from the beginning and then takes characters until the first ‘]’ character and drops one from the right after that.

Now we have the first food as a string, after that the method’s algorithm gets all the data from the other foods: it drops first the first food since we have it already, after that it maps through the foods and drops first 5 characters of each because they have only unusable characters like “|[[” etc. and after mapping it takes characters from the stream until it finds a ‘]’ character.

Gather: toFoodList:

```
def toFoodList(data:Array[String],restaurant: String): Buffer[Food] = {
    var foodList = Buffer[Food]()

    //getting the wanted food infos here by mapping the HTML data
    var foodNames = data.map(x=>x.drop(1)
                                .takeWhile(_!=',')
                                .dropRight(1))
    var allergenes = data.map(x=>x.dropWhile(_!='[')
                                 .drop(1).split(',')
                                 .mkString
                                 .filter(_.isUpper)) //filtering "
characters
    //add all the gotten data to the foodList
    for(i <- 0 to foodNames.length-3) {
      foodList+=new Food(foodNames(i),allergenes(i),restaurant)
    }

    foodList
  }
```

This algorithm gets all the data we got using the menuToFoodAndAllergenes method and forms it into a buffer of Food objects. This method has two different algorithms that are similar to each other: foodNames and allergenes. foodNames algorithm maps through the data and drops the first character because it is a ‘,’ then takes characters while it finds another ‘,’ and drops one from the right.

The allergenes algorithm is similar: It maps through the data and drops while the character isn’t ‘[‘ because after that the data is the allergene data we want to get. After that it drops one useless character and splits by the ‘,’ character so we get a list of all the allergenes. Because the data consists of the “ characters in all the allergene names we

have to filter them by filtering all the lowercase characters, because the allergenes are uppercase(L,G etc.). And now we have all the allergenes as a string list.

After that we create the food objects with the gotten data in the for-loop.

GUI: today:

```
private val today = {

if(java.time.LocalDate.now.getDayOfWeek==java.time.DayOfWeek.MONDAY) {
              java.time.LocalDate.now
}
else if(java.time.LocalDate.now.getDayOfWeek==java.time.DayOfWeek.TUESDAY)
{
              java.time.LocalDate.now.minusDays(1)
}
elseif(java.time.LocalDate.now.getDayOfWeek==java.time.DayOfWeek.WEDNESDAY)
{
              java.time.LocalDate.now.minusDays(2)
}
else if(java.time.LocalDate.now.getDayOfWeek==java.time.DayOfWeek.THURSDAY)
{
              java.time.LocalDate.now.minusDays(3)
}
else if(java.time.LocalDate.now.getDayOfWeek==java.time.DayOfWeek.FRIDAY) {
              java.time.LocalDate.now.minusDays(4)
}
else if(java.time.LocalDate.now.getDayOfWeek==java.time.DayOfWeek.SATURDAY)
{
              java.time.LocalDate.now.minusDays(5)
}
else {
              java.time.LocalDate.now.minusDays(6)
}

}
```

The today variables algorithm checks first which day today is for the user and then forms a new java.time.DATE for the user that is always monday, for example if the current day was tuesday the algorithm removes one day so the weekday is set as monday.

GUI: getAllTheFoodsAgain:

```
def getAllTheFoodsAgain {
              for(i <- 0 to 4) {
                for(k <- 0 to dayFoods(i).data.size-1) {
                 alltables(i).update(k,0,dayFoods(i).data(k).getName)
                 alltables(i).update(k,1,dayFoods(i).data(k).getRestaurant)
                 alltables(i).update(k,2,dayFoods(i).data(k).getAllergenes)
                }
              }
            }
```

This algorithm contains of two for-loops: the first goes through days monday to friday and the second goes through how many foods each day has so we can update the table's each cell one by one and update its data.

GUI: helperForBoxes:

```
def helperForBoxes(char:Char) = {
              alltables.foreach(x=>
                      for(i<- 0 to x.rowCount-1) {
                        if(  x.apply(i,2).toString().contains(char)) {
                          x.update(i, 0, "filtered")
                          x.update(i, 1, "filtered")
                          x.update(i, 2, "filtered")
                        }
                       }
                      )
            }
```

This algorithm goes through all the foodtables and checks if the row's allergene cell contains the character that is the parameter of this method(char). If it is it changes the data of that cell to 'filtered' with the update method.

5. **Data structures**

   This project uses Strings, Buffers and Food-objects as its primary data structures. Strings are used when we get the HTML data from the server and we parse it into smaller pieces => Buffer of strings. After that we form the strings into Food-objects which

is our own data structure. It contains the following data: Name, Allergenes and the restaurant. The data is used when forming the UI's foodtables.

I could have used JSON format for HTML to String part and instead of string I could have used JSON. I felt like making it JSON would have needed a little bit more work even though there are several ready-made libraries for JSON. The HTML was fairly clear to just parse with simple string methods that the scala's default libary has.

For buffers I could have used any other 'list' data structure as well but buffers were more familiar to me before the project. Because we aren't dealing with high amounts of data(each day has maximum of 10 different foods) the decision of using buffers didn't make a differenct speed-wise. If the data would have been million times bigger we should have used for example hashSets to make the data handling faster.

6. **Files and Internet access**

If the program accesses Internet to read/write data, e.g. web pages, explain the relevant things here as appropriate.

The application's class called Gather does all the work regarding reading data from a webpage. The Gather class has a method called readData which uses libraries called: 'io.Source' and 'java.net.URL':

```
def readData(restaurant:String,date: String): String = {
    val url = "https://kitchen.kanttiinit.fi/restaurants/"+restaurant+
"/menu?day="+date
    //without properties sending a GET request wouldn't work:
    val requestProperties = Map(
      "User-Agent" -> "Mozilla/5.0 (Macintosh; Intel Mac OS X 10.12;
rv:73.0) Gecko/20100101 Firefox/73.0)"
    )
    //open TCP connection to the wanted server
    val connection = new URL(url).openConnection
    requestProperties.foreach({
      case (name, value) => connection.setRequestProperty(name, value)
    })

    //and finally get the lines from the html

Source.fromInputStream(connection.getInputStream).getLines.mkString("\n")
  }
```

You can see from the method that it has first a variable called ‘url’ which is the website we cant to connect to. Here restaurant is a number for example 3 for ‘Täffä’ and date is in the format of: 2020-04-25 for exampe. Then we need a request properties because otherwise the server wouldn’t allow us to access its data because it doesn’t know what is connecting to it. After that we opne the TCP connection by making a new URL object and using its method ‘openConnection’ and then it creates a socket for us. Now that we have a connection to the server, we can do a GET request for the HTML data by calling the Source.fromInputStream method with the parameter: connection.getInputStream and there we have it, all the data as string data structure.

7. **Testing**

For the testing the application has its own ‘Tests’ class. It consists of different kinds of tests for each part of the project. The tests run the second a user launches the application and he will be notified in the console whether the tests were competed succesfully or otherwise.

For the GUI we tests the correctness of the today variable and it should always equial to java.time.dayOfWeek.MONDAY. Also we check that the GUI should have foods for 5 different weekdays. For the Food class we create a good food object and a bad food object and do tests for those: For the good one we test that all the variables are set as correct ones. For the bad one we check that it will throw an exception if it gets a bad food object. Meaning for example a bad restaurant name that doesn’t exist.

For the Gather class we test that the correct amount of foodtables are made.

If during the launch of the application any of the tests won’t pass the user will be notified in the console with an appropriate message saying where the problem is.

8. **Known bugs and missing features**

The project misses the feature that was supposed to be done called ‘favourite setting’. This wasn’t one because setting a favourite restaurant wouldn’t be very useful because only Täffä currently has a menu for each weekday’s foods and other campus restaurants are shut down. Also missing the feature of viewing other restaurants but that wasn’t possible due the corona situation since all other campus restaurants are closed.

One known bug is that the user can modify the UI's tables and for example delete text from each cell or edit it. This isn't a big bug but something that should be considered. Because the user uses the app for himself this won't be bad for other users because they have a separate 'own' application and the changes won't be visible to them.

9. **3 best sides and 3 weaknesses**

   3 best sides: I feel like the Gather class was implemented very well. The connection was opened smoothly in the method and the data from the HTML was parsed rather clearly in the methods menuToFoodAndAllergenes and toFoodList. Third thing I am proud of is the filtering feature of the UI. The user can filter for example if he doesn't want to eat anything with gluten he can just filter all of those foods by clicking the checkbox.

   3 weaknesses: The application should have more thorough tests, it doesn't cover all the sides of the project. The UI could be more creative instead of just tables of foods etc. The project should also have atleast one more feature for the user for example setting restaurant.

10. **Deviations from the plan, realized process and schedule**

    0-2: Starting was fairly hard and slow, made the class structure but didn't do methods for them

    2-4: Connection working and getting the data from the wanted server

    4-6: methods for classes working + GUI implementation

    6-8 GUI

    8-10 fixing bugs and testing

    Comparing to the time schedule before the project, it differs a little but not too much. The start was slow because I had a few other courses' exams. Apart from that the timeline was faily well followed. A few slowdown every now and then but other than that I made the project in time. The order was progress was followed directly as estimated, I feel like it was well done and felt intuitive. I learned that starting is hard and slow, there are many things to consider when starting one's very own project, how to add the libraries in use etc.. So start working on the project as soon as you have time! Don't wait a week before starting.

11. **Final evaluation**

    All in all I feel like achieving the main goals of this project. I made a fairly good UI that

has a few features for the user to use and the project in general is well structured. The program could be improved UI-wise and restaurant-wise. UI wise it could be a little bit more userfriendly and not so 'blocky'. Maybe edit your own UI elements? Restaurant-wise it only uses one restaurant's data because of the corona situation. Maybe in the future it can be easily modified into a multiple restaurants data.

The solution methods were well made in my opinion, they were clear and easy to use and well documented. Data strucutres could have used more 'popular' methods like JSON instead of plain strings. Class structure is intuitive and easy to see what happens where. Maybe the GUI class is a bit messy because of all the swing elements, maybe should have separated it into a few files instead of one. Changes are easy to make, the methods take parameters like date and restaurant, so it is easy to adapt more restaurants into the program. UI is also very easy to modify. All the elements are in order and easily modified.

What would I do different? Probably the tests. I would make them more thorough and they would then check for more possible situations in the program.

12. **References**

Connection library: https://www.scala-lang.org/api/current/scala/io/Source.html (read:14.2.2020)

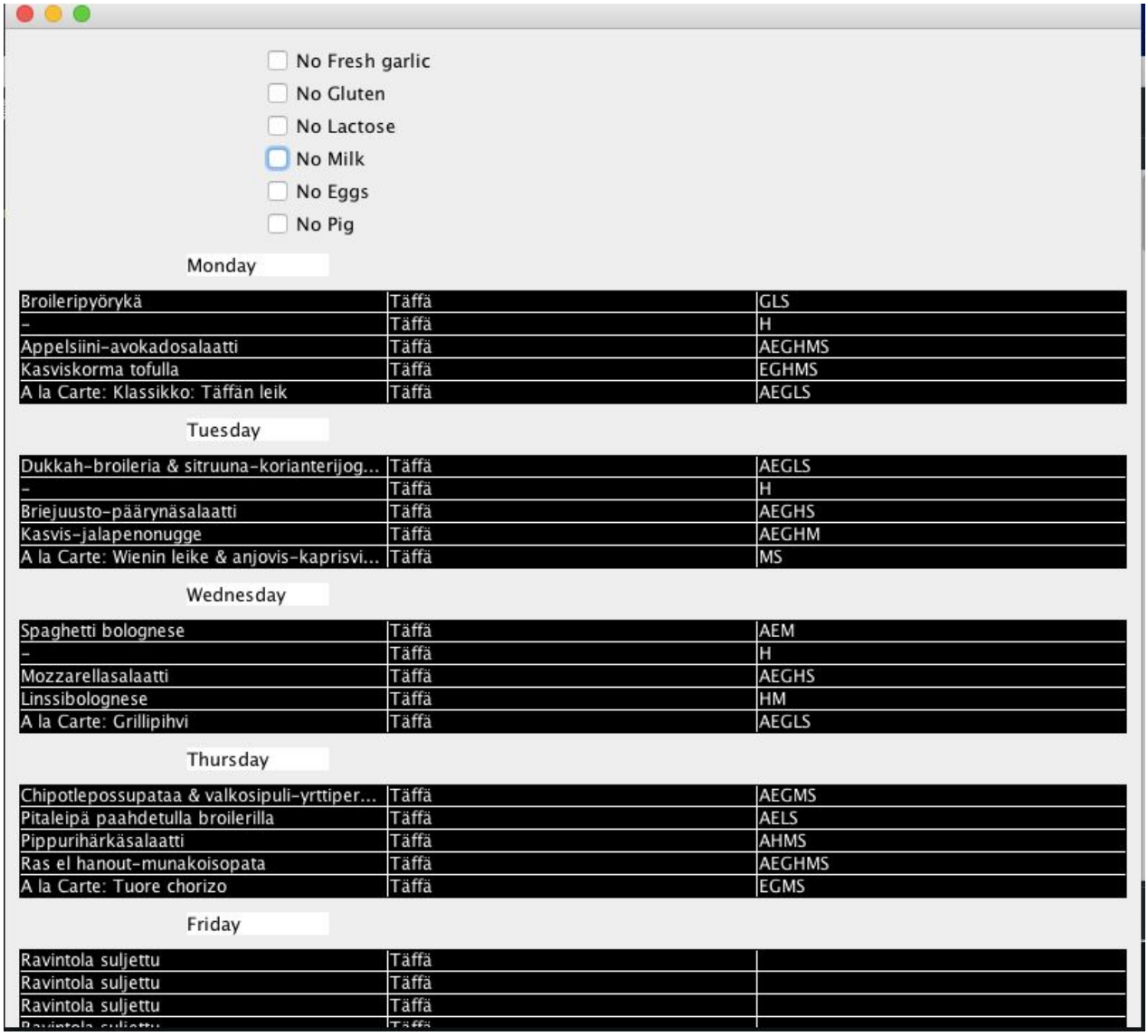
UI library: https://www.scala-lang.org/api/2.9.1/scala/swing/package.html (read: 20.3.2020)

UI colors etc. https://docs.oracle.com/javase/7/docs/api/java/awt/package-summary.html (read 25.3.2020)

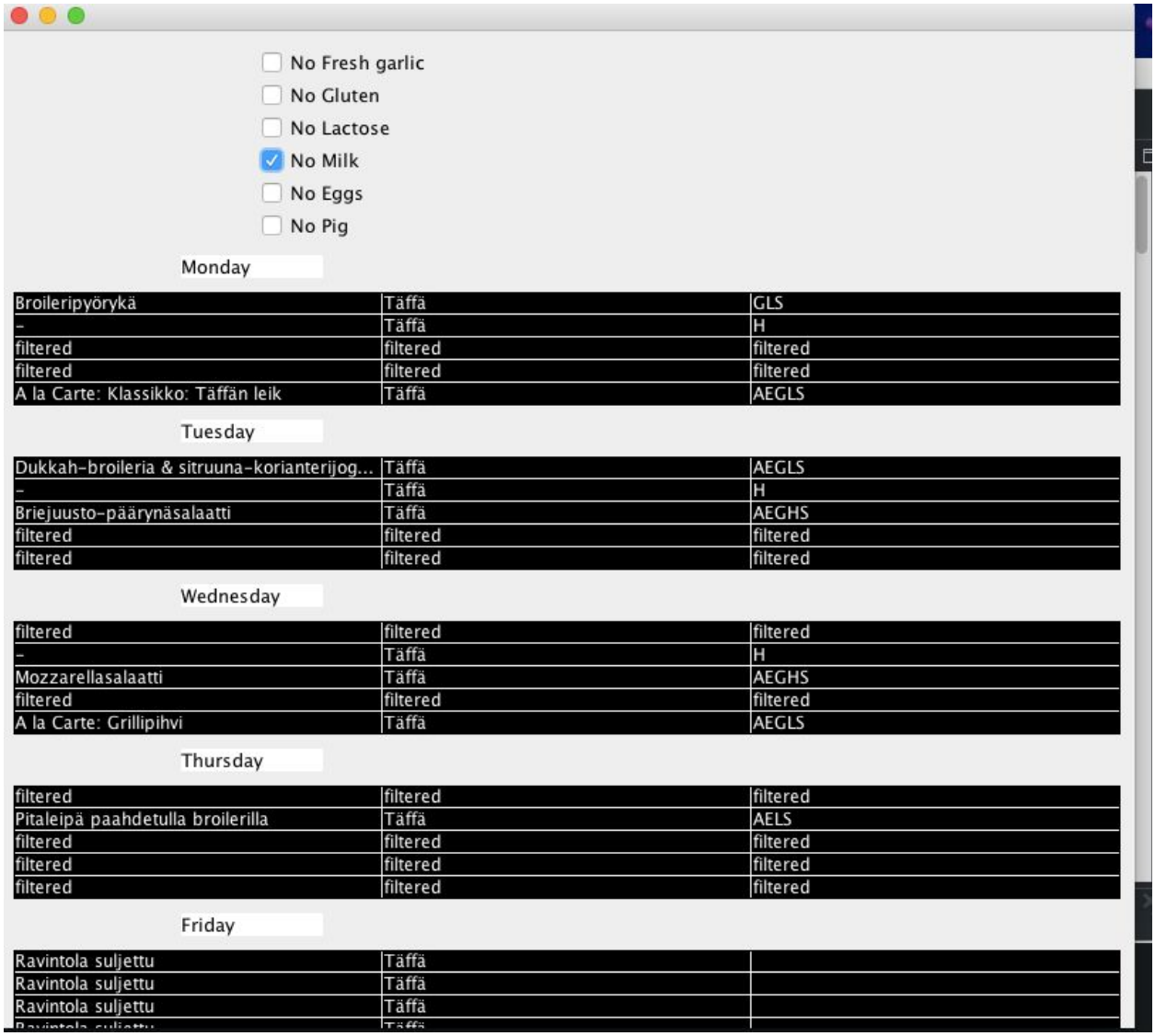
Testing tips: http://www.scalatest.org/user_guide/writing_your_first_test (read 16.4.2020)

13. **Appendixes**

Main UI screen:



And after a user decided to filter all the foods with milk in it:


No Fresh garlic
No Gluten
No Lactose
No Milk
No Eggs
No Pig
Monday
Broileripyörykä Täffä GLS
- Täffä H
filtered filtered filtered
filtered filtered filtered
A la Carte: Klassikko: Täffän leik Täffä AEGLS
Tuesday
Dukkah-broileria & sitruuna-korianterijog... Täffä AEGLS
- Täffä H
Briejuusto-päärynäsalaatti Täffä AEGHS
filtered filtered filtered
filtered filtered filtered
Wednesday
filtered filtered filtered
- Täffä H
Mozzarellasalaatti Täffä AEGHS
filtered filtered filtered
A la Carte: Grillipihvi Täffä AEGLS
Thursday
filtered filtered filtered
Pitaleipä paahdetulla broilerilla Täffä AELS
filtered filtered filtered
filtered filtered filtered
filtered filtered filtered
Friday
Ravintola suljettu Täffä
Ravintola suljettu Täffä
Ravintola suljettu Täffä