BABEŞ-BOLYAI UNIVERSITY CLUJ-NAPOCA
FACULTY OF MATHEMATICS AND COMPUTER SCIENCE
SPECIALIZATION Computer Science in English

DIPLOMA THESIS

Optimizing Meal Prepping

Supervisor
Lecturer PhD Bădărînză Ioan

Author
Berla Ewald

2022

UNIVERSITATEA BABEŞ-BOLYAI CLUJ-NAPOCA
FACULTATEA DE MATEMATICĂ ŞI INFORMATICĂ
SPECIALIZAREA [Informatica engleza]

LUCRARE DE LICENŢĂ

Optimizarea preparării meselor

Conducător științific
Dr. Lector Bădărînză Ioan

Absolvent
Berla Ewald

2022

ABSTRACT

I declare that this thesis was composed by myself,that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or processional qualification except as specified.

Motivation

Eating is something most of us take for granted. Unless we have specific goals or needs, in general we don't really think too much about it, you throw some eggs, some sunflower oil, maybe a slice or two of bacon and you got yourself a breakfast. For some of us this isn't the case, especially those looking into dieting can be easily overwhelmed by the amount of information on the internet and will soon find that cooking a breakfast that will fit into the meal plan can be quite difficult.

Let's say for example you want to be in a caloric deficit in order to lose some weight, you will soon find that counting calories everyday isn't as easy as it may seem, and planning your meals for your whole week will take some time and patience. And that is if we completely disregard the diversification of meals, which is a very important aspect in order to live a healthy and happy life. Of course, there is the possibility to eat only chicken and rice but if you're someone who likes eating, that's not a viable option.

Or let's take another example, you want to build some muscles, so you know you have to be in a caloric surplus, then you start by eating more, maybe even adding another meal, and based on your metabolic rate, you can find yourself in a place where the numbers on the scale are going up and it's not muscle that you're packing.

A lot of people, myself included, can find themselves in a scenario like the ones presented above quite often, and if you do, it can be irritating since the options are quite limited: you either give up, or you take out your kitchen scale and start calculating the caloric value of everything you eat. Of course, the best option, as stressful as it is, would be the second one. It can be done, and a lot of people have done it, actually some are even enjoying it, but some just don't want to bother that much, some people just want to follow a dieting plan and hopefully make progress.

A possible solution

In this paper, I propose a possible solution for these problems and the concept is pretty simple. By accessing a data set consisting of more than 12000 recipes, considering the user's desire to either gain or lose weight, and calculating the user's

Total Daily Energy Expenditure (TDEE), the application will generate suggestions for what to eat each day in order to meet the user's caloric goals.

Contents

Chapter 1

Introduction

1.1 Some statistics

Eating is the process in which food is ingested to provide an organism with energy in order to survive and grow. This has been the understanding of eating ever since humankind, and it hasn't changed that much since then, we still have to eat in order to survive but now we have the luxury of choosing what to eat and in what quantity.

The process remains the same, you consume food, the body transforms the food into energy, then that energy is used and the surplus, if there is any, is stored as fat. What has changed is the amount of information we have on the topic of nutrition, we learned a lot more about the macronutrients that compose our food, how many kilocalories one should eat, and most importantly how unhealthy poor dietary habits can be.

You would think that with all the information of today's age, we should see a decrease in the rate of people struggling with dietary-related conditions but studies have shown the complete opposite.

Our World in Data [1] published an article that shows how the obesity rate has significantly increased since 1990, and *World Health Organization* concluded in 2021 that the worldwide obesity has nearly tripled since 1975 [2].

In simple terms, weight gain can be determined by an unbalance of energy. Whenever we consume more energy than the energy used by our body to survive and for daily activities, our organism stores that energy for later use and we gain weight. There are two potential drivers of the increase in obesity rates, either we burn less energy through lower activity levels, or our kilocaloric intake has increased.

Over the last few decades, the supplies of calories all over the world has increased, going from 2200kcal per person per day in 1961 to 2800kcal in 2013.

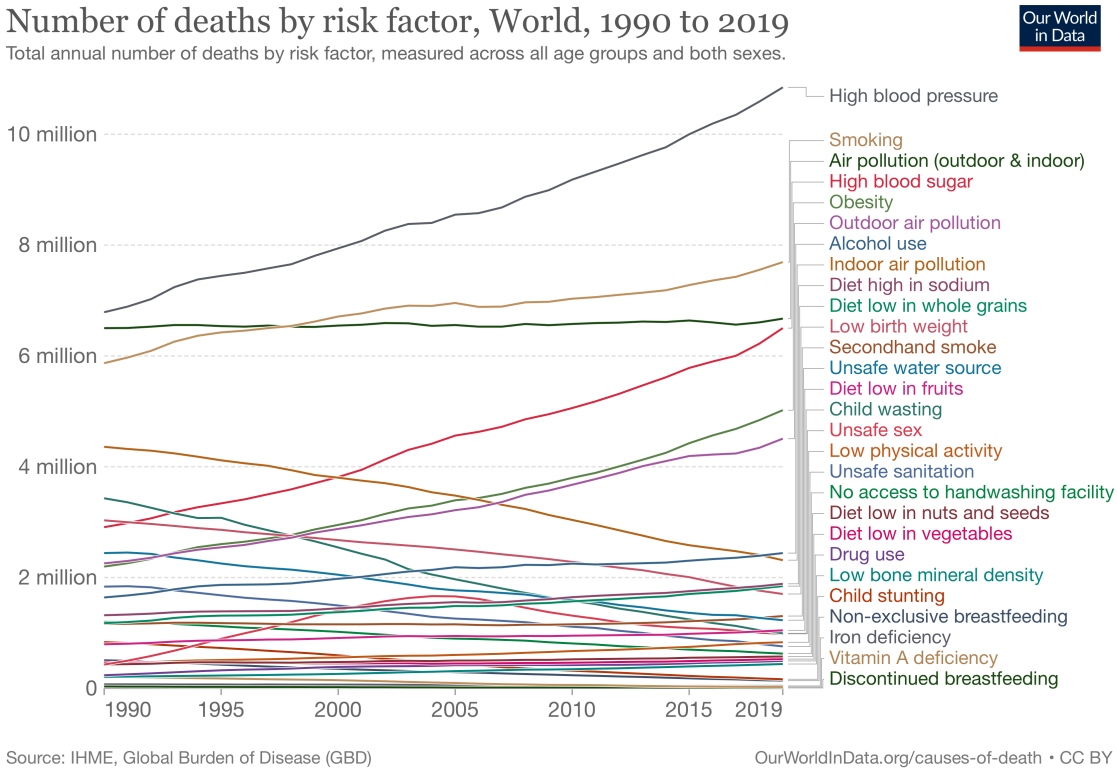

Figure 1.1: Obesity rate since 1990

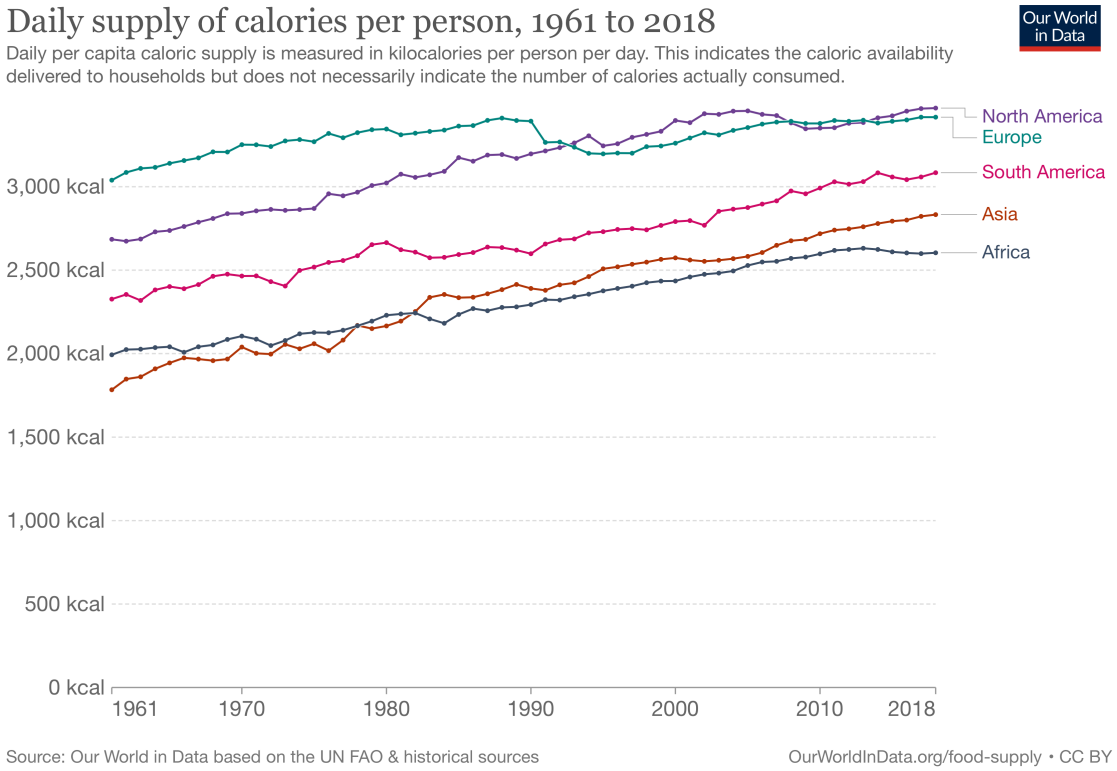

Figure 1.2: Daily supply of calories from 1961 to 2018

In terms of physical activity *World Health Organization* released a study in November of 2020 in which they concluded that globally, 1 in 4 adults do not meet the recommended levels of activity, more than 80% of world's adolescent population is insufficiently physically active and that 5 million deaths a year could be averted if the global population was more active [3].

1.2 The diet problem

There actually isn't any diet problem, the majority of popular diets are working, the problem is whether you can actually stick to it or not, and it's not as easy as you might think. This one often cited July 2005 study in The American Journal of Clinical Nutrition concluded that only about 20% of the people actually succeed in sticking to a diet and maintaining their weight long-term, the other 80% either fail in losing considerable weight or fail in maintaining it, relapsing in less than a year [4].

1.3 Trying to help

I decided to try and tackle the obesity problem while also helping those who want to gain more weight without too much extra fat by approaching it on the nutritional side, to be more specific, I decided to try and make meal planning much easier for everyone, and to take away all the technicalities that go into the process so everyone, without any prior knowledge about how this works, can use it.

Chapter 2

Application Development

2.1 Requirements

The application must help users meet their weight goals and overall help with the obesity problem described above.

2.1.1 Problem Statement

The process of planning what to eat for a week can be a thoughtful and difficult process, especially if the user tries to be careful about their caloric goals in order to meet their desired weight. Moreover, it can take a lot of time calculating the calories attributed to each meal for each day of the week, so the user needs a way of optimizing this process.

They must have the ability to just input data about themselves, about their preferences and the application must suggest meals based on that preferences that will meet their desired goals.

2.1.2 Solution

The application tries to help solving this problem by making the process of tracking what you eat and preparing the food in advance easier. It all starts with the user inputting critical information which will be used for processing the diets, information like weight, height, gender and physical activity.

The user then proceeds to choose what their goal is: to gain weight or to lose it, and how fast do they want to get to it, while keeping in mind that a slower rate tends to work better for most users. After the rate has been established, the application needs to know the user's preference on how many meals do they like in a day and then it's all set up.

With this data, the application uses the Harris-Benedict equation in order to calculate an individual's basal metabolic rate (BMR). This value is then used with different activity factors to determine the total daily energy expenditure (TDEE). Having this value, the application can suggest the user different recipes by searching the set of 1200 recipes, that will fit into the daily caloric budget, taking into account the user's preference for the number of meals in a day and their goal.

2.2 Analysis and design

2.2.1 Users and use cases

There is only one type of user that can create an account, log into application, and use the meal-suggestion algorithm. The application is structured in such a way that a user can interact with the whole application, there is no need for creating an account. However, creating an account comes with the benefit of storing the information about the user for later use. With the user permission granted, the application can store the metrics used by the user in order to make it easier for them to generate meals, the stored fields being automatically taken into account. Another benefit of having an account is saving meal recipes for later reading.

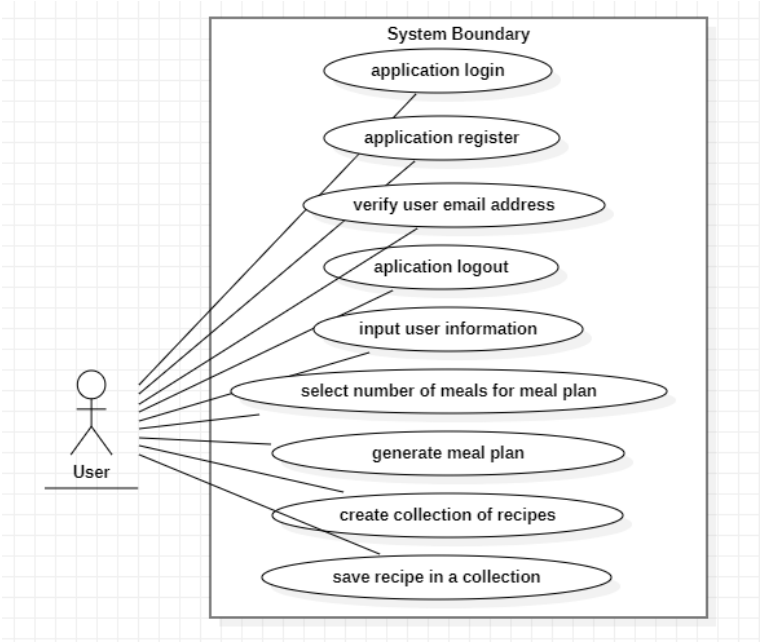

Figure 2.1: Use case diagram

The database contains the following entities:

- User: information about the user
- Authority: types of authorities that exist in the application

- User-Authority: table that connects users with authorities
- Verification Token: table containing information about verification tokens issued by users and their ids

2.2.2 Entities and domain design

User

The user entity represents the basic actor of the application. This class includes attributes like *age, weight, height, gender*, that are stored with the user's approval in order to make the process of meal generation easier. The user also contains an attribute called *authority* used to identify what type of privileges a user has. A more in-depth view of the user entity can be viewed at figure 2.2.

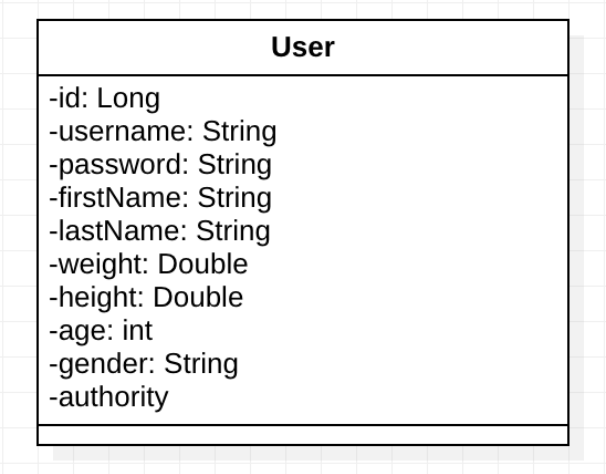

Figure 2.2: User entity

Authority

The authority class implements the Springboot Security interface *GrantedAuthority* and it is used during the authentication phase in order to decide what privileges the user has. The authority class can be seen 2.3

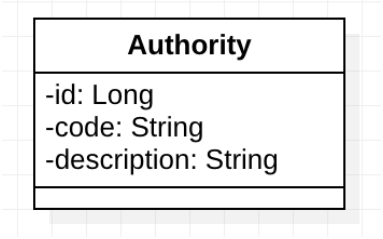

Figure 2.3: Authority entity

Meal

Most of the time recipes in the application are stored into a JSON dataset which we will talk about later. However, when a user decides he wants to save a recipe for later reading, the meal is stored inside the database and inside a given collection. This entity also stores a value named 2.4

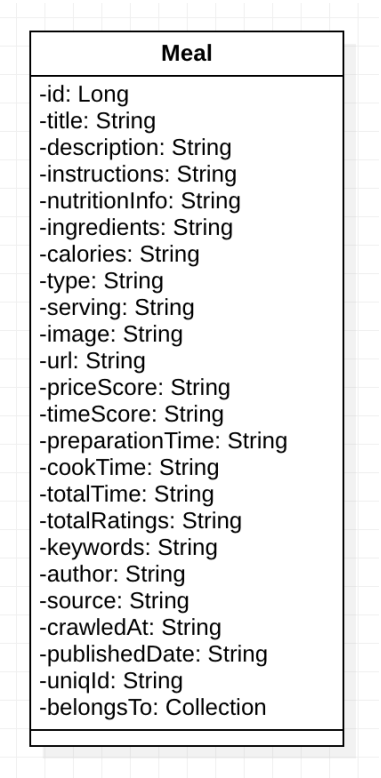

Figure 2.4: Meal entity

Collection

The collection entity is used to associate recipes when the user saves them. This allows the user to categorize saved recipes based on their preference, and in order to help them, the collection also stores a collection name and a description 2.5

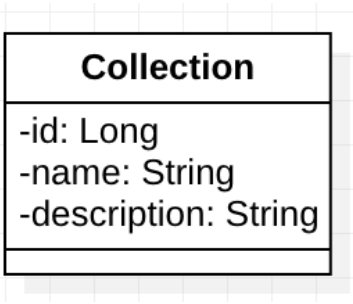

Figure 2.5: Collection entity

Verification Token

This entity is used solely for verification of the email address. This is generated when a new account is created and contains information about the user and the actual token 2.6.

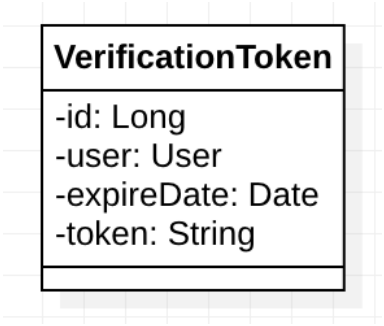

Figure 2.6: Verification token entity

Domain Class Diagram

The class diagram containing all of the above entities can be found in figure 2.7

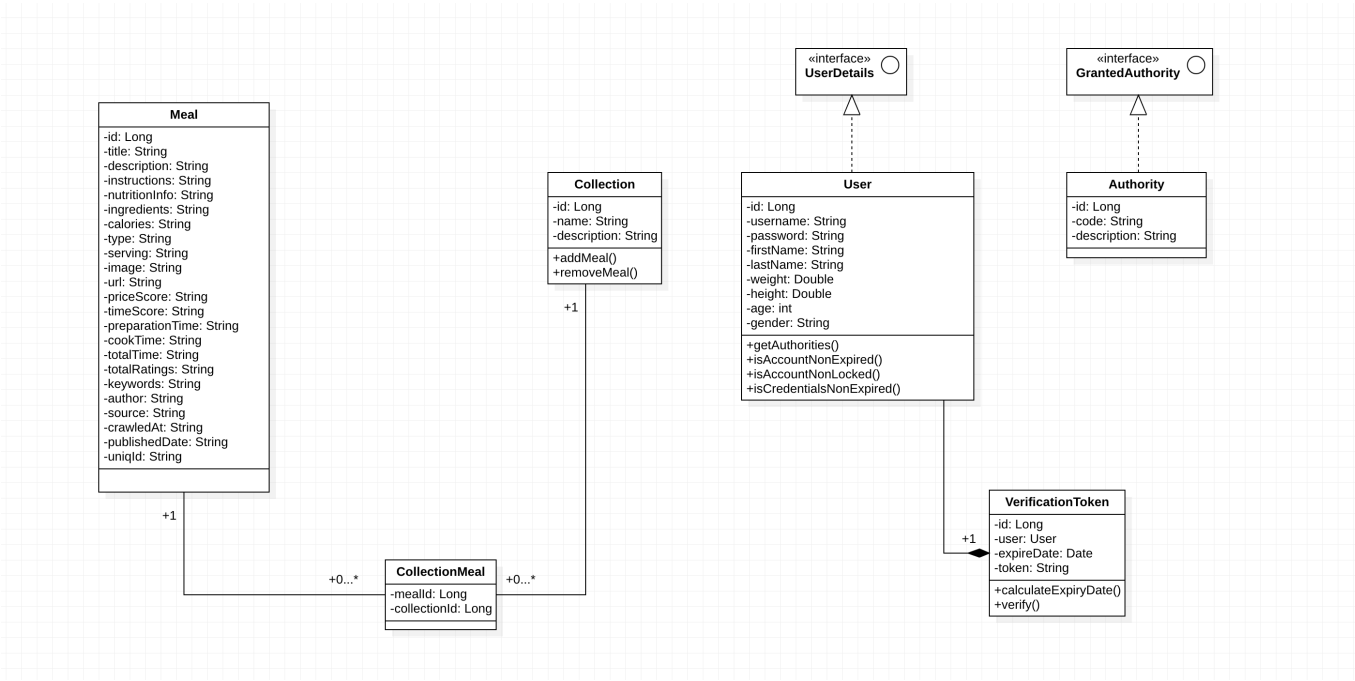

Figure 2.7: Domain class diagram

2.2.3 Use case specifications

Application register

- Description: Will trigger if the user tries to create an account. On the front-end app this means typing the necessary information and hitting register.
- Actors: User
- Pre-conditions: The user has to provide the necessary information and a valid email address as well. All the fields have to be filled, *Full Name* has to be filled with first name and last name separated at least by a space, the *Email* address has to be in a valid format, the *Password* must contain at least 1 uppercase character, 1 special character, 2 digits, 3 lowercase letters and at least 8 characters in total, also the password confirmation and password should be the same.
- Post-conditions: An account will be created. The account will, however, remain disabled until the user verifies their email address.
- Flow: The user navigates to the *Register* page, fills in the information and clicks *Register*.

Verify email address

- Description: Will trigger if the user created an account and verifies their email address by clicking the link in the provided email.
- Actors: User
- Pre-conditions: The user has created an account.
- Post-conditions: The account is verified and the user can now log in.
- Flow: The user navigates to their email inbox, and clicks the link in the provided email.

Application login

- Description: Will trigger if the user has an account and presses the login button on the front-end application.
- Actors: User
- Pre-conditions: The user has a verified account. All fields are required to be filled.

- Post-conditions: The account credentials are checked and the user is logged in.
- Flow: The user navigates to the *Login* page, fills the information and presses *Login*.

Application logout

- Description: Will trigger if the user has an account and presses the logout button on the front-end application.
- Actors: User
- Pre-conditions: The user is logged in into an existing account.
- Post-conditions: The user is logged out of the account, and the front-end app updates accordingly.
- Flow: The user clicks the *Log out* button on the header.

Generate meal plan

- Description: Will trigger if the user fills the information on the meal plan page and presses *Generate*
- Actors: User
- Pre-conditions: The user has to supply preferences and metrics about themselves such as: gender, age, weight, height, activity level, goal, goal tier.
- Post-conditions: If the user is authorized, the meal plan will be displayed, each meal having three recipe suggestions.
- Flow: The user navigates to the *Meal-plan*, fills the needed information and presses *Generate*.

Save user info

- Description: Will trigger if the user types his information on the *Meal-plan* page for the first time, or if the user modifies the already stored values.
- Actors: User
- Pre-conditions: The user is logged in into an existing account.
- Post-conditions: The user information is stored.

- Flow: The user navigates to the *Meal-plan*, fills the needed information and presses *Generate*, if it is the first time the user has generated a meal plan, or if the user updates its stored information, they will be asked for permission to store the information. If permission is granted, the application will store the information and it will be displayed on the *Meal-Plan* page the next time the user visits it.

Create a collection

- Description: The user can create a collection that will store recipes.
- Actors: User
- Pre-conditions: The user is logged in into an existing account. The user also has to supply a name and a description for the collection.
- Post-conditions: A new collection is created with the given information.
- Flow: The user navigates to their profile page, clicks on the *plus* button and provides the needed information for the creation of the collection. Another way of doing it, is by clicking the *Save* button when viewing a recipe. The rest of the process is the same

Save recipe

- Description: The user can save a meal in order to view it later.
- Actors: User
- Pre-conditions: The user is logged in into an existing account. The user also has to supply either a previously created collection or create a new one.
- Post-conditions: The meal is saved into a given collection of meals.
- Flow: The user generates a meal plan, selects a recipe and clicks the *Save* button. From here, the user has to select an existing collection or to create a new one. After a collection has been created, the user has to press the *Save* button on the pop-up and the recipe will be saved.

View all saved meals

- Description: The user can view all previously stored recipes.
- Actors: User

- Pre-conditions: The user is logged in into an existing account. The user needs to have some previously saved recipes and has to click on the collection of meals that they want to see.
- Post-conditions: The collection of recipes is displayed.
- Flow: The user navigates to their profile page and selects a collection. From here the list of recipes will be displayed.

2.2.4 Sequence Diagrams

Login

In figure 2.8 the sequence diagram contains the flow of the login process, containing both cases when credentials are valid or invalid.

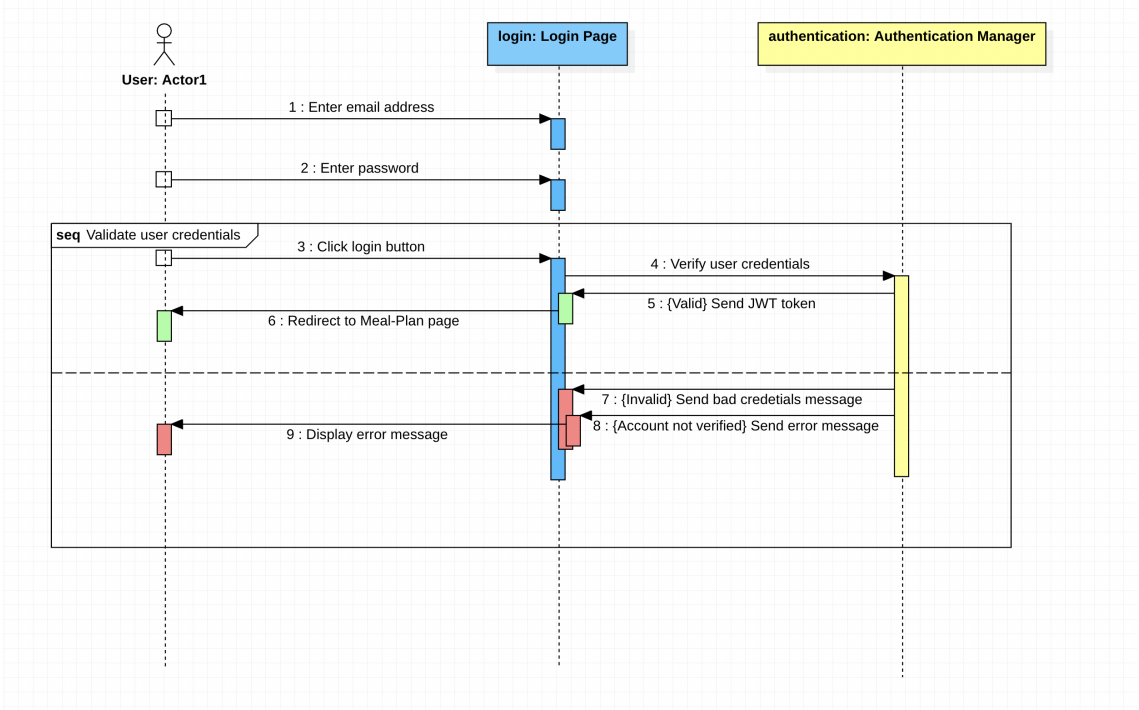

Figure 2.8: Login Sequence Diagram

Register

In figure 2.9 the sequence diagram shows the flow of the register process, if the credentials are valid and the account is created, a confirmation mail will be sent to the user in order to activate the account.

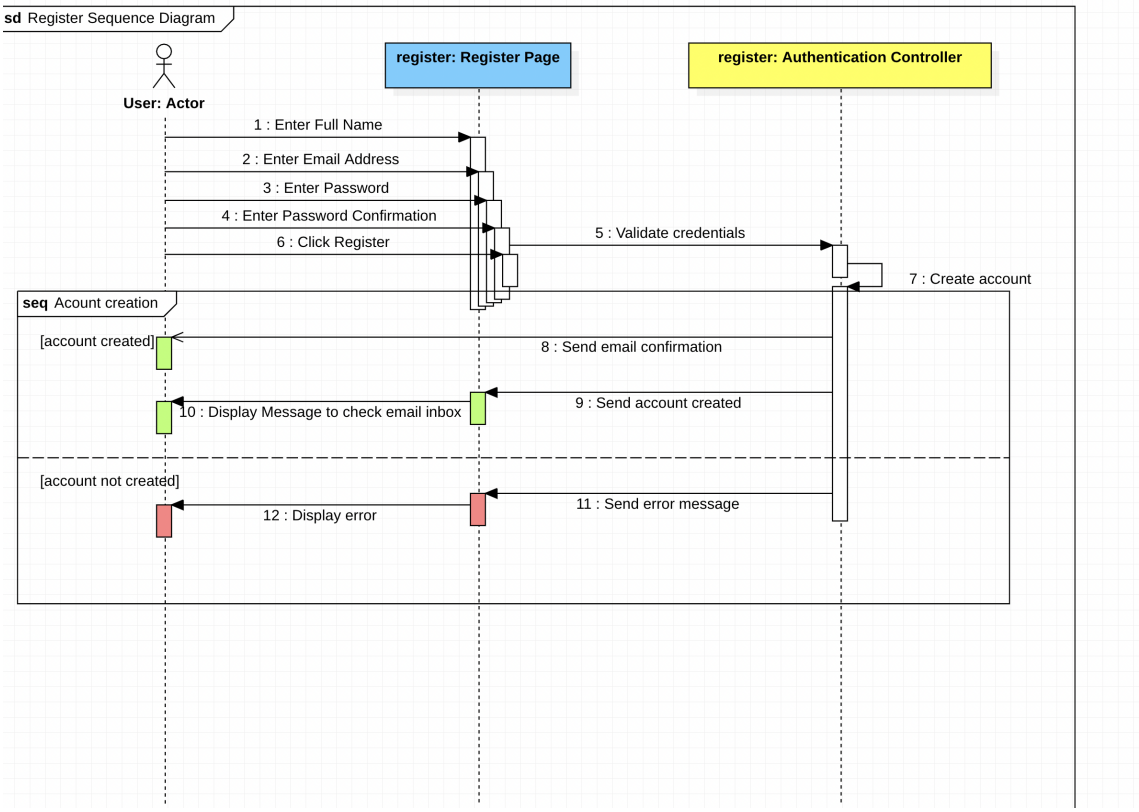

Figure 2.9: Register Sequence Diagram

Email verification

After the registration controller sent an email to the user's email address, the user has to activate the account by clicking the url in the received email. The process is illustrated in figure 2.10.

Generate meal plan

In figure 2.11 the process of meal plan generation is illustrated. The user inputs the required data, the data is validated and the suggestions of recipes are displayed. The user can choose to store data for future meal-plan generation if they have an account.

Create collection

In figure 2.12 the process of collection creation is illustrated.

Save meal

In figure 2.13 the process of saving a recipe can be seen. The user has to select a collection before saving a meal, any of the previously created collections can be used or the user can create a new one.

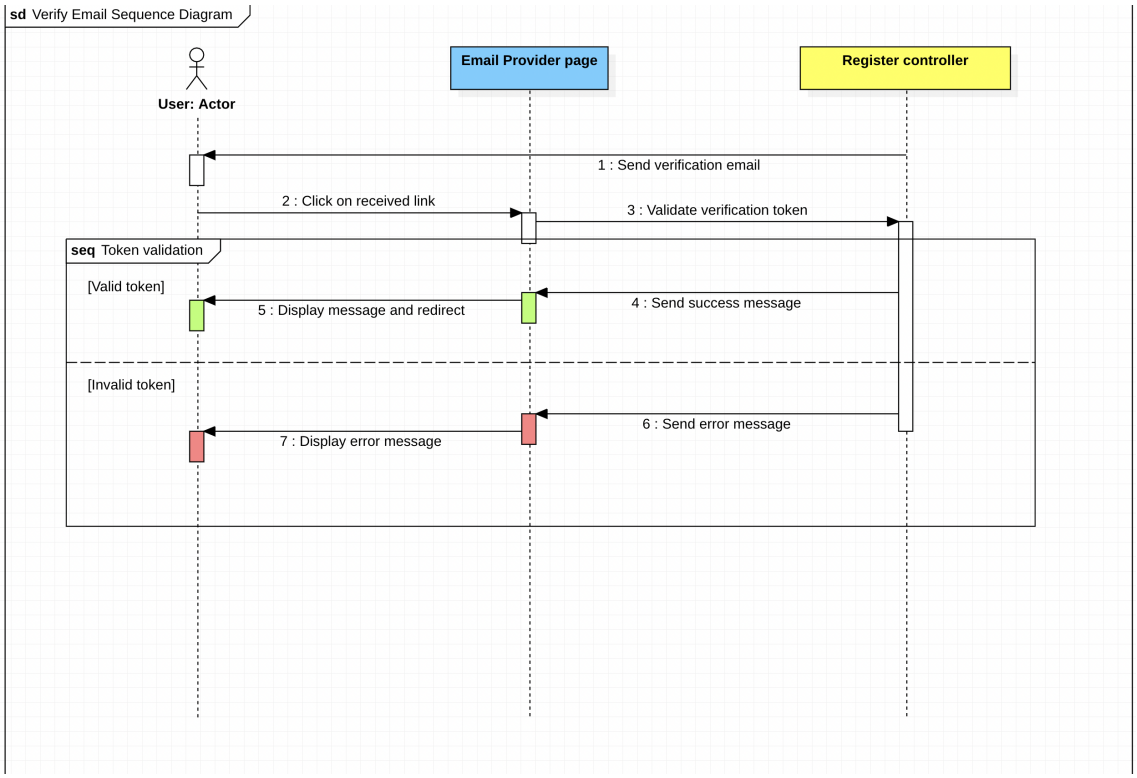

Figure 2.10: Email verification Sequence Diagram

2.3 Back-end Architecture

2.3.1 Technologies

Maven

Maven [5] is an open-source build tool developed and maintained by Apache Group that focuses on the simplification and standardization of the building, publishing and deploying process. Some of the most well-known features of Maven include a growing list of user libraries, setting up projects with ease, dependency management and backwards compatibility. Another reason for which Maven is so well known is the existence of POM (Project Object Model), a XML file containing all the necessary information about the project. To put into perspective the enormous repository of Maven artifacts in Figure 2.14 you can see a fraction of 1% of the whole graph of Maven artifacts.

Spring Boot

Spring Boot [7] is an open-source micro-framework built on top of the Spring framework, it is used to build production ready Spring applications and it comes with many dependencies that can be plugged in a Spring application such as Spring LDAP, Spring Security, etc.

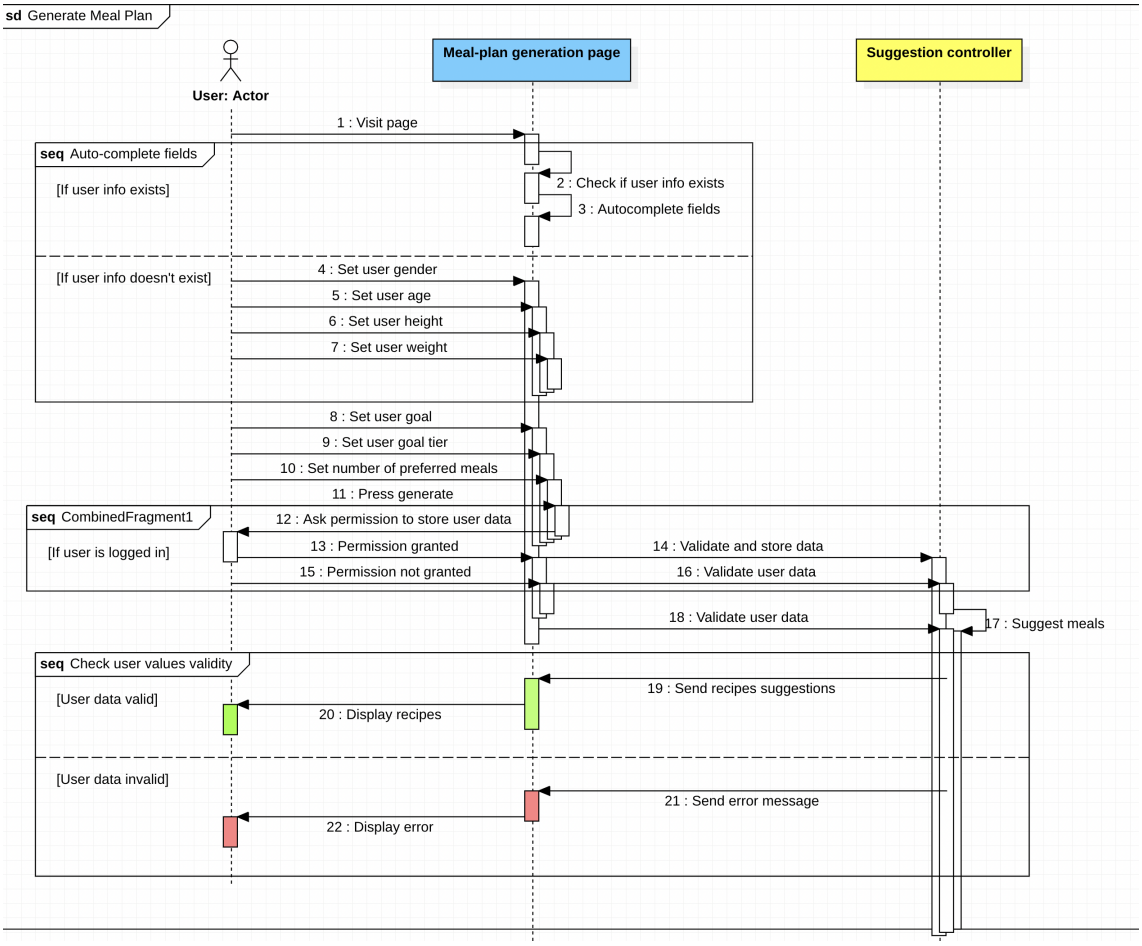

Figure 2.11: Meal plan generation sequence diagram

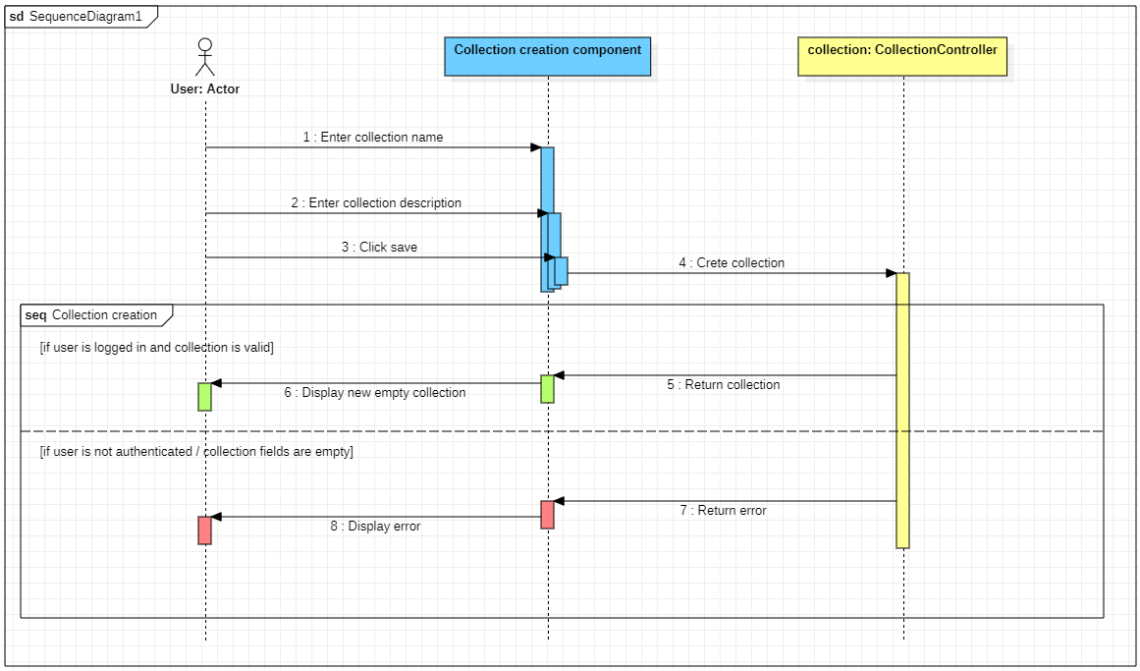

Figure 2.12: Create collection sequence diagram

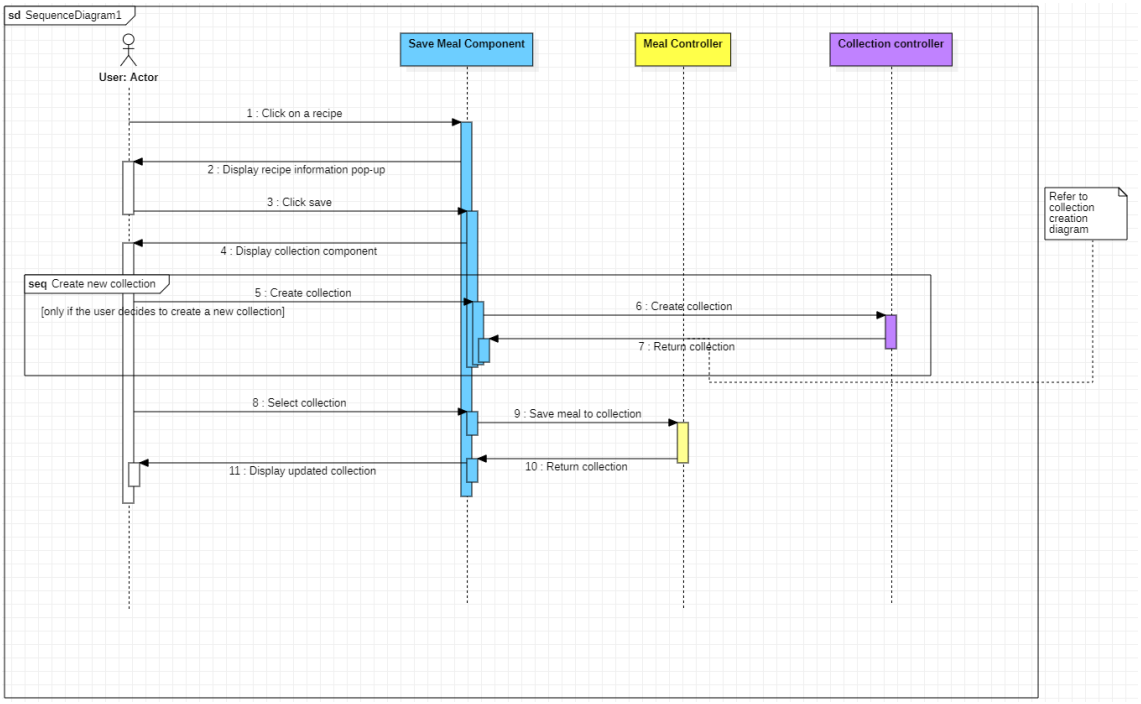

Figure 2.13: Save meal sequence diagram

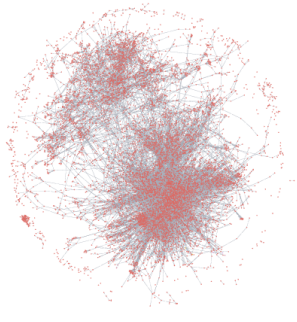

Figure 2.14: Excerpt of 1% of the whole graph of Maven artifacts [6]

MySQL

MySQL is an open source relational database management system developed and maintained by Oracle. As the name says, it is based on Structured Query Language, a language used by programmers in order to create and modify data stored in a relational manner. To be more specific, relational databases means that data is stored in separate tables in which data types may be related to each other.

2.3.2 Design patterns

Design patterns are common, reusable techniques used by software developers as a solution for solving commonly occurring problems in a software design context.

Both the Front-end and Back-end implementations are built with the help of design patterns. On the Back-end side of the application, some of the more mainstream design patterns are implemented, while on the Front-end application, the design patterns are not so well-known and sometimes are referred to as design techniques.

Adapter

Starting off with one of the most popular structural design patterns, the *Adapter pattern* [8] is a common solution for communication between objects with incompatible interfaces. In this application, this is exactly the case, since the dataset of meal recipes stores data in *JSON* format, and since the algorithm used to suggest recipes needs objects of type *MealDTO* to operate, there is a need for an adapter class which takes the JSON data retrieved by the meal repository and converts it into an list of *MealDTO* objects

The adapter class in this case is named *MealJsonDTOAdapter* and it exposes only one public method: *convert* which takes an array of *JSON* objects and returns a list of converted *MealDTO* objects.

Builder

The *Bridge pattern* [9] is one of the creational design patterns which helps in constructing complex objects step-by-step. Using the same code for construction, this pattern allows the production of different types of representation of an entity.

In the case of this application, there are a number of ways to calculate the *Total Daily Energy Expenditure*, each approach has its ups and downs and the differences are not that great.

Take for example the Harris-Benedict formula and the revised one, both are using the same principles but the multipliers are different. In this case, the code for the *TDEE* calculator was extracted into a separate interface and classes which have

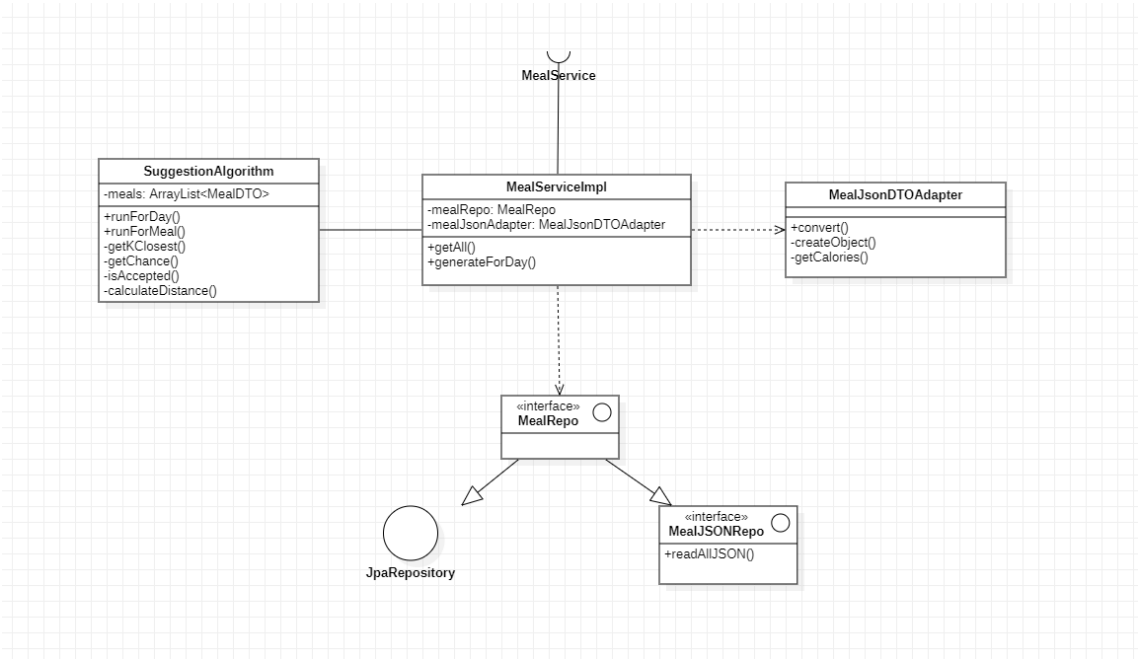

Figure 2.15: Integration of adapter design pattern

to implement it, an interface for the calculator builder was defined and each type of formula will have to implement its own concrete builder. For example: *HarrisBenedictBuilder* and *HarrisBenedictRevisedBuilder* which will build an object of type *CaloricCalculatorInterface* with their own multipliers. On top of that, another object named *Director* is created in order to establish what builder to use and start the building process.

2.4 Front-end Architecture

2.4.1 Technologies

ReactJS

ReactJS or React.js [10] is a well-known, free, open-source JavaScript front-end library used mainly for building and designing user interfaces. It was created by Jordan Walke at Facebook in 2011 for their own use, and today is maintained by over a thousand open-source contributors alongside with a small and dedicated team working full-time at Meta (formerly Facebook).

Flexibility is one of the most important benefits of using React, and to further illustrate that we can think of a React component as anything we want into a web application, it can either be a text, button, grid, etc. Also, since the community has grown a lot over the years, even though React was created mainly for web application, there are tools like Electron which lets you create desktop application, or React

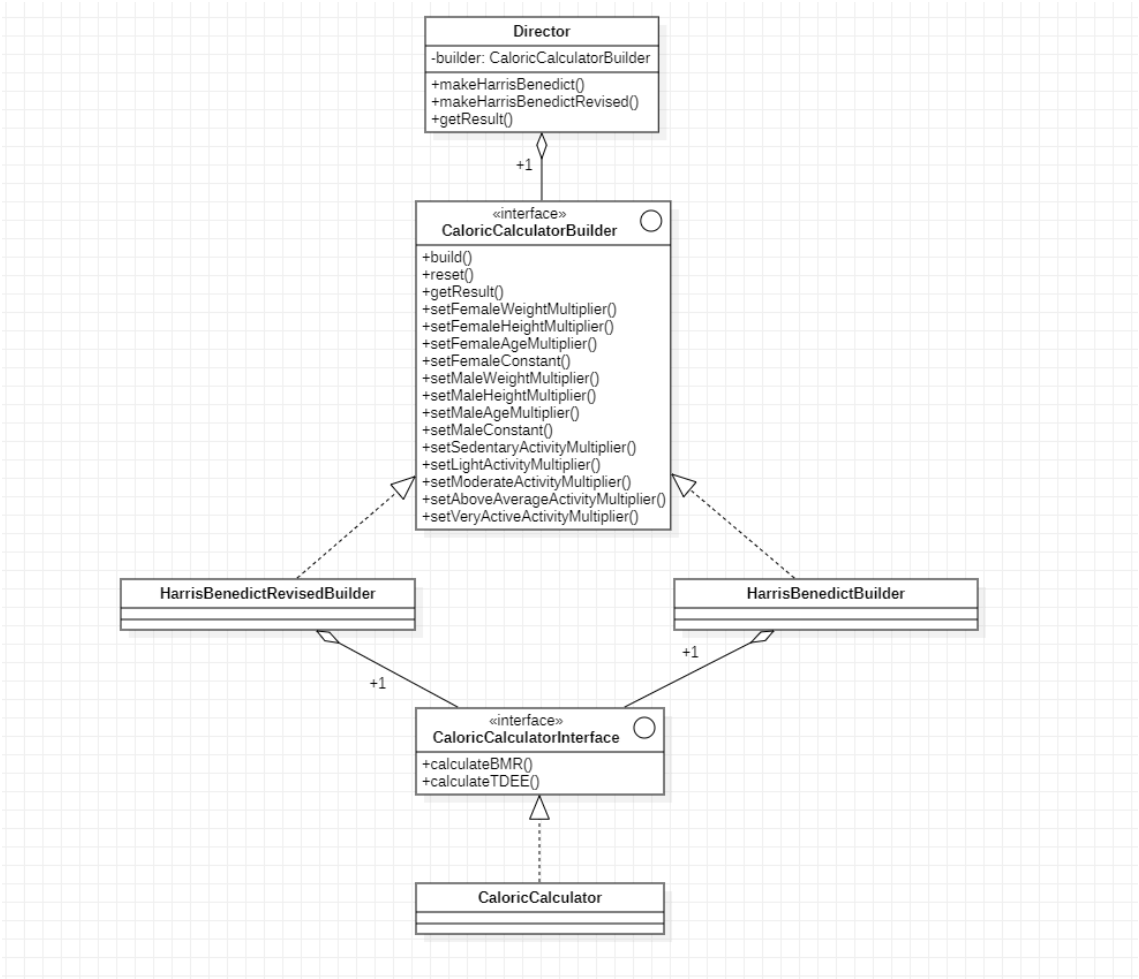

Figure 2.16: Integration of builder design pattern

Native for mobile applications.

Another benefit is the support it gets, both from Meta's resources (top four contributors of React are developers from Meta) and from open source community (the repository is one of the top five repos on GitHub).

ReactJS also has great performance since the developers figured out a way of making the DOM update process more efficient and intelligent. The way this is done is when a component's state is changed, React finds the cheapest way to update the Document Object Model tree, thus avoiding layout trashing (when the browser has to update the position of every element in the DOM) by updating only the component that needs to change.

2.4.2 Design patterns

Redux

Redux [11] is an open-source library used for centralizing application state mostly in ReactJS or AngularJS front-end applications created by Andrew Clark and Dan Abramov and its primary maintainers since 2016 are Tim Dorr and Mark Erikson.

Redux is best used when the application state is updated frequently, when large parts of the app state needs to be in multiple parts of the application, and when the logic to update the state is too complex. In the case of this application, the redux library is used paired with *redux-persist* npm package in order to persist data about the user and update the components according to it.

The one-way data flow works like this:

- The state of the application represents the condition of the application at a specific time stamp
- The User Interface is rendered according to the state
- When an event happens, the state is updated
- The User Interface updates itself based on the new updated state

Redux is also characterized by the fact that the global state acts as a single source of truth through the application. This basically means that any peace of data exist only in one location and it is not duplicated. This makes debugging, inspecting the application state and centralizing the login that is interacting with the whole application easier.

Another advantage is the fact that the state is read-only, and it can only be changed if an action has been dispatched. This way, the user interface won't accidentally overwrite the state and tracing a state update is easier.

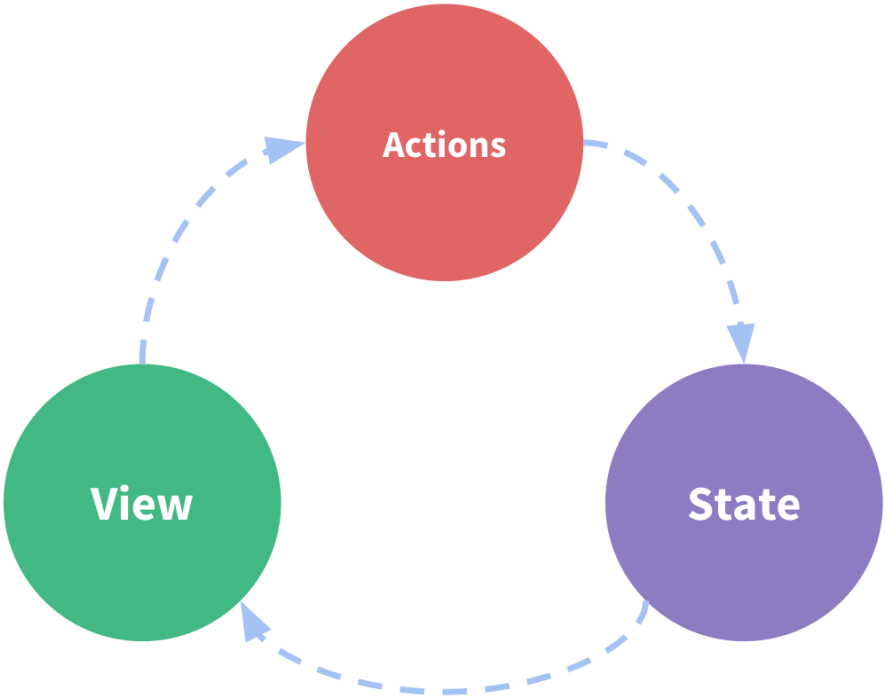

Figure 2.17: Integration of builder design pattern

Compound Components

Compound components work together and share a state in order to achieve a common goal. For example the *select* and *option* HTML elements don't function properly on their own, but together they interact and create a drop-down menu.

This design pattern can be seen in many popular React UI Libraries such as Material UI, React Bootstrap or Ant Design.

In this application, this pattern can be seen in a couple of places, most notably, the Modal used for displaying information about a recipe is built using this approach.

```
export default function MealModal(props) {
    const {item, onClose, open} = props;
    const instructions = item.instructions.split('|');
    const ingredients = item.ingredients.split('|');
    const nutritionInfo = item.nutritionInfo.split('|');

    return (
        <Modal open={open} onClose={onClose}>
            <div  className="meal-modal flex-row flex-space-between">
                <div className='meal-modal-left'>
                    <div className='meal-modal-left-image'>
                        <img src={item.image}/>
                    </div>
                    <div className='meal-modal-left-info'>
                        {item.prepTime && <MealModalInfo icon={refrigerator} label={'Preparation Time'} info={item.prepTime}/>}
                        {item.cookTime && <MealModalInfo icon={timer} label={'Cook Time'} info={item.cookTime}/>}
                        {item.timeScore && <MealModalInfo icon={stopwatch} label={'Time score'} info={`${item.timeScore} / 5`}/>}
                        {item.priceScore && <MealModalInfo icon={dollar} label={'Price score'} info={`${item.priceScore} / 5`}/>}
                        {item.kcalories && <MealModalInfo icon={chart} label={"Calories"} info={item.kcalories}/>}
                        {item.serving && <MealModalInfo icon={bowl} label={'Serving'} info={item.serving}/>}
                        {/* {item.type && <MealModalInfo icon={tag} label={'Type'} info={item.type}/>} */}
                    </div>
                </div>
```

Figure 2.18: Integration of compound component design pattern

This implementation illustrates how *Modal* and *MealModalInfo* components work together to create a modal containing information about one recipe. *MealModal* can't exist without *MealModalInfo* and *MealModalInfo* doesn't make much sense without *MealModal*. The Result can be seen at figure 2.19, *MealModal* being the entire pop-up and *MealModalInfo* being the component rendered below the image.

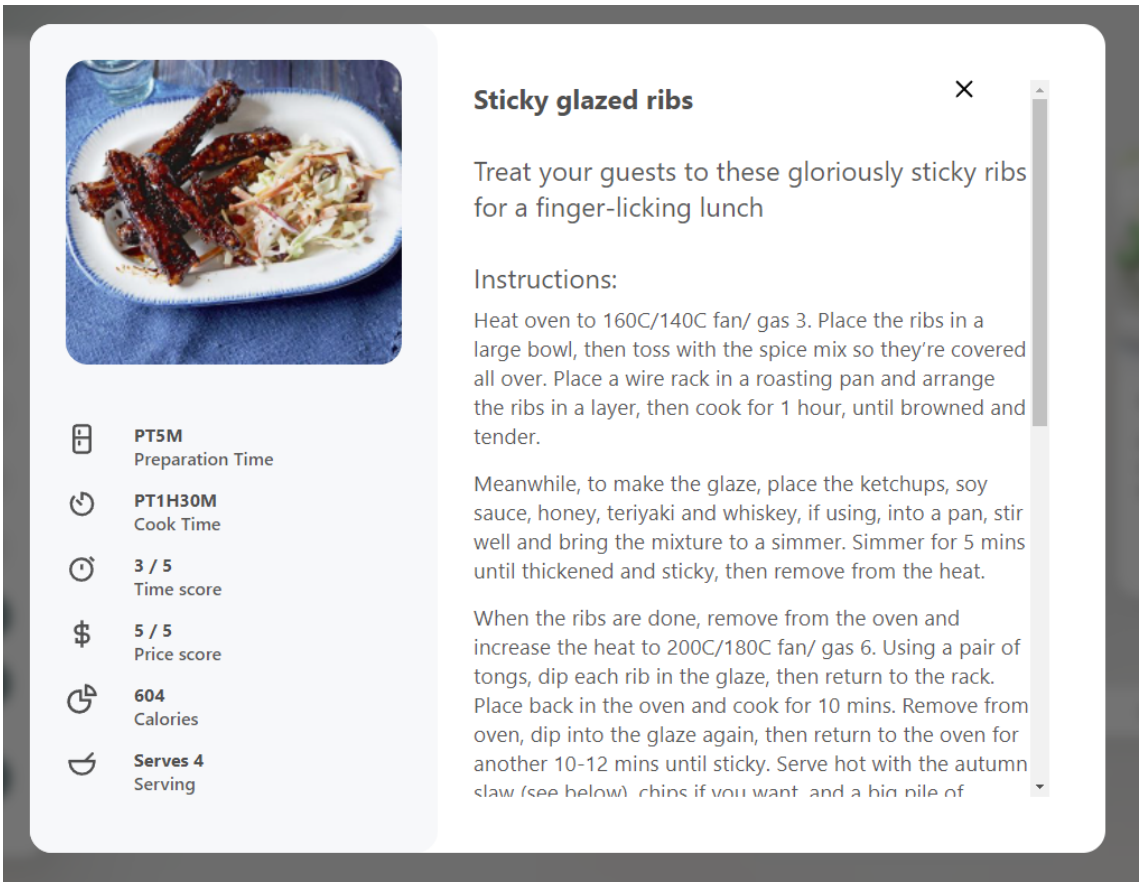

Figure 2.19: Recipe information modal

2.5 Dataset

Since the main purpose of this application is to aid users when creating a meal plan and recommending food recipes to them, I decided that it's best to start with an existing dataset of recipes and add to that. While searching the internet for a dataset containing at least 5000 recipes with nutrition information, descriptions, and instructions on how to cook them I came across a set of data containing 14000+ recipes from *BBC Food* [12]. The dataset contains what I was searching for and more, in total there were 18 fields stored in JSON format which along other information includes: an image of the recipe, information about how much time it takes to prepare and cook, ingredients, instructions and so on.

After downloading the dataset, I created a python script that reads the keywords of each recipe and categorizes it into snacks, breakfasts, desserts, drinks and other. I decided to remove the drinks from the dataset since they didn't fit into the purpose of the application so after removing them there were 11954 recipes left.

The python script also analyzes the total time to cook and prepare a recipe and assigns a score ranging from 0 to 5. The encoding of the time information is *ISO 8601*([13]), an international standard covering the worldwide exchange of date and time-related data maintained by *International Organization for Standardization (ISO)*. For parsing this data I used Metomi's ISO Datetime parser implementation ([14]).

After parsing the time data, the score is assigned based on 5 intervals:

- Recipes that take less than 30 minutes to prepare and cook
- Recipes that take between 30 and 60 minutes to prepare and cook
- Recipes that take between 1 hour and 2 hours to prepare and cook
- Recipes that take between 2 hours and 4 hours to prepare and cook
- Recipes that take between more than 4 hours to prepare

At this stage, the recipes are also assigned a random price rating from 0 to 5 that will be used to demonstrate the suggestion algorithm. When releasing the application, the default value will be 0 and the user will have the possibility to give a rating between 0 to 5. The displayed rating will be the average of all of the previous price ratings assigned by the users.

An example of a food recipe from the dataset would be the *Sesame spinach* [15] (see figure 2.20)

- title : Sesame spinach
- total time : PT6M
- cook time : PT1M
- serving : 2
- crawled at : 17/12/2021 00:43:57
- description : Flavoured traditionally with sesame, soy sauce, vinegar and sugar, this quick and simple dish is a classic and versatile Korean side
- url : https://www.bbcgoodfood.com/recipes/sesame-spinach

- nutritions info : calories:104 calories — fatContent:8 grams fat — saturatedFatContent:1 grams saturated fat — carbohydrateContent:3 grams carbohydrates — sugarContent:3 grams sugar — fiberContent:3 grams fiber — proteinContent:4 grams protein — sodiumContent:1.1 milligram of sodium
- image : https://images.immediate.co.uk/production/volatile/sites/30/2020/08/sesame-spinach-b5d9949.jpg
- ingredients : 1 tbsp toasted sesame oil—½ tbsp soy sauce—½ tsp toasted sesame seeds , crushed—½ tsp rice vinegar—½ tsp golden caster sugar—1 garlic clove , grated—225g spinach , stem ends trimmed
- unique id : 82cb5e7a-2d4d-5197-bd49-c5502c373a8a
- source : BBC Good Food
- author : Judy Joo
- preparation time : PT5M
- published date : 2015-06-18T16:01:47+00:00
- keywords : 1 of 5-a-day, 2 servings, 200 kcal or less, Folate, Judy Joo, Korean side dish, Quick, spinach stir-fry, Summer, Under 10 minutes
- total ratings : 10 ratings
- instructions : Bring a large pot of water to the boil and fill a large bowl with cold water and a handful of ice. Meanwhile, in a medium bowl, stir together the sesame oil, soy sauce, sesame seeds, vinegar, sugar, garlic and pepper to taste, until the sugar has dissolved. Set the dressing aside. — Blanch the spinach in the boiling water until just wilted, then scoop straight into the cold water. Drain well and squeeze out any excess water. Gently loosen the clumps of spinach with your fingers, transfer to the bowl of sesame dressing and toss together. Cover and chill for about 1 hr to let the flavours mingle.
- type : other
- time score : 5
- price score : 5

Figure 2.20: Image of a an example of food recipe

2.6 Implementation

In the implementation part of this thesis I will briefly describe the Login the Register processes and focus more on the implementation of the Meal Suggestion Algorithm. The algorithm contains the following steps:

- Calculating the Total Daily Energy Expenditure of a person
- Determining how many calories to eat for a given goal
- Splitting the previously calculated values on the chosen number of meals for a day
- Creating the meal targets
- Determining suggestions for each meal

2.6.1 Determining Total Daily Energy Expenditure

TDEE (Total Daily Energy Expenditure) is a value that represents how many calories a certain person needs to consume in a day in order to survive and maintain their weight at a certain activity level. Therefore by having this value and subtracting or adding from it, one can calculate how many calories to eat in a day in order to meet a certain goal. First, BMR (Basal Metabolic Rate) has to be calculated to obtain this value.

Basal Metabolic Rate is the rate of energy expenditure per unit of time, but contrary to what TDEE is, BMR is calculated at rest instead of at a certain activity level (see 2.21).

```
28
29      public Double calculateBMR(char gender, Double weight, Double height, int age){
30          switch(gender){
31              case 'm':
32                  return maleConstant + (maleWeightMultiplier * weight) + (maleHeightMultiplier * height) - (maleAgeMultiplier * age);
33              case 'f':
34                  return femaleConstant + (femaleWeightMultiplier * weight) + (femaleHeightMultiplier * height) -(femaleAgeMultiplier * age);
35          }
36          return null;
37      }
38
39      public Double calculateTDEE(char gender, Double weight, Double height, int age, String activityType){
40          Double bmr = this.calculateBMR(gender, weight, height, age);
41          switch(activityType){
42              case "SEDENTARY":
43                  return this.SEDENTARY_MULTIPLIER * bmr;
44              case "LIGHT_ACTIVITY":
45                  return this.LIGHT_ACTIVITY_MULTIPLIER * bmr;
46              case "MODERATE_ACTIVITY":
47                  return this.MODERATE_ACTIVITY_MULTIPLIER * bmr;
48              case "ABOVE_AVERAGE_ACTIVITY":
49                  return this.ABOVE_AVERAGE_ACTIVITY_MULTIPLIER * bmr;
50              case "VERY_ACTIVE":
51                  return this.VERY_ACTIVE_ACTIVITY_MULTIPLIER * bmr;
52          }
53          return bmr;
54      }
```

Figure 2.21: Code for calculating BMR and TDEE

There are a number of ways to calculate BRM and one of the most popular is using the Harris-Benedict formula. I decided on this approach since it works well for

the majority of people ([16] this study found that HBE is the most accurate from various prediction equations). However, there are cases in which this formula doesn't hold well (an example of a group where this approximation doesn't hold is in elderly african american people [17]) and other equations might work better. The implementation is using the *Builder* design pattern such that in the future I will be able to add other types of formulas like Harris-Benedict revised equation or the Mifflin St. Jeor equation, each with their own advantages. The design pattern used in this situation fits the needed flexibility since all of the above-mentioned formulas follow the same principle but the multipliers for the values are different.

2.6.2 Calorie intake for a goal

The next step is to determine what daily caloric intake an individual needs in order to meet their goal. There are two goals (one can either choose to lose weight or gain weight) and two goal-tiers which work both ways (either 0.25kg or 0.5kg). There are two goal tiers:

- The first goal tier: 0.25kg/week
- The second goal tier: 0.5kg/week

What that means is that after choosing a goal, an individual can decide how much weight to gain/lose in a week. This is calculated by adding or subtracting 11% of the TDEE for the first goal tier (0.25kg/week) or 22% of the TDEE for the second goal tier (0.5kg/week).

Getting the perfectly fit food recipes for a given goal is only one part of the problem. The other part consists in the diversification of food. What that means is that for the same input getting the same suggestions is not a good solution, instead, some sort of mechanism to diversify the result has to be implemented without altering too much the suggestions.

For this purpose, at the end of the process, before returning the determined value, a constant is generated randomly between -10 and +10 and it is added to the final result. This decreases the possibility that two runs of the algorithm using the same input will return the same result. It is a simple but effective trick and will not influence the final result by much.

This process can be seen in figure 2.22, the main function of the mechanism being *calculate*. This class has to be provided a *GoalCalculator* object, either one for losing weight or one for gaining weight.

2.6.3 Splitting for desired number of meals

Some people prefer eating three meals a day, while others prefer eating four

```java
/**
 * Calculates the value of calories for a given goal
 * @param TDEE - Total Daily Energy Expenditure
 * @return <code>double</code> value of calories for a given goal
 */
public Double calculate(Double TDEE){
    int percentage = this.calculatePercentage();
    Double calories = this.goalCalorieCalculator.calculateCalories(percentage, TDEE);      ewald30, last week • minor changes …
    return this.randomize(calories);
}

/**
 * Calculates the percentage for each given goal
 * @return <code>Double</code> the percentage for each goal
 */
private int calculatePercentage(){
    switch (goalTier){
        case FIRST_GOAL_TIER:
            return FIRST_GOAL_TIER_PERCENTAGE;
        case SECOND_GOAL_TIER:
            return SECOND_GOAL_TIER_PERCENTAGE;
    }
    return 0;
}

/**
 * Since a difference of +-10 calories for a day is not that big, this function is used to create results that differ more
 * @param TDEE - Total Daily Energy Expenditure: this will be altered by either adding or removing 10
 * @return <code>double</code> representing the modified value of TDEE
 */
private Double randomize(Double TDEE){
    Random random = new Random();
    int value = random.nextInt((MAX_RANDOMIZE_VALUE - MIN_RANDOMIZE_VALUE) + 1) + MIN_RANDOMIZE_VALUE;
    return TDEE + value;
}
```

Figure 2.22: Code for calculating the calorie intake for a given goal and tier

or maybe five. Since the purpose of the application is to make meal planning as easy as possible for most people, the algorithm will take into account the user's preference for how many meals they eat in a day. These can be the main three meals of the day (breakfast, lunch, dinner) and a maximum of two snacks: one in the morning and one in the afternoon. This means that splitting the previously calculated calorie intake can be done in 3 ways. For each case, and for each meal, there is assigned a range of percentages and a number belonging to that interval will be chosen randomly. That number will represent what percent of the total caloric intake for a day will be given to that meal.

The way the calories are divided was inspired by Omni calculator's webpage ([18]) and was tweaked by following a dietitian's advice. For three meals:

- breakfast: between 30% and 33%
- lunch: between 37% and 40%
- dinner: between 28% and 32%

For four meals:

- breakfast: between 26% and 28%
- snack: between 7% and 9%
- lunch: between 36% and 39%
- dinner: between 26% and 29%

For five meals:

- breakfast: between 27% and 29%
- snack: between 7% and 9%
- lunch: between 37% and 39%
- snack: between 7% and 9%
- dinner: between 17% and 19%

This mode of splitting the meals makes sure that we can have at most 105% of the total caloric intake in a day and at least 95%.

2.6.4 Creating the target meals

This part of the process goes hand in hand with the splitting of caloric intake in a number of meals. After doing so, we have to create some target meals.

A target meal, in our implementation, is a DTO (Data Transfer Object) for a meal object that contains only the necessary information that is needed for the algorithm to run: number of calories, time score, price score, and type (see 2.23).

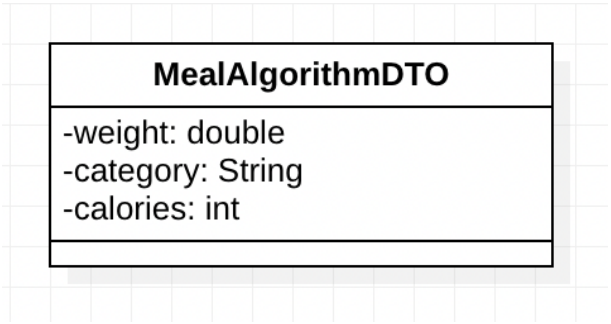

Figure 2.23: Class Diagram of MealAlgorithmDTO

These are used as reference for the algorithm when searching for recipes that are close to them. One aspect that is important to mention here is what is the *weight* attribute and how it is computed.

2.6.5 The weight of a meal

When suggesting meals to the users, the algorithm takes into account three different values:

- The price rating of a meal

- The time score
- The number of calories of a meal

Since the algorithm essentially tries to find *k* recipes closest to our target meal while taking into account the price score of the meal, the time score of the meal, and the number of calories, we need a single value to refer to when calculating the distance between them. The *weight* of a meal is the value that takes into account the above mentioned three values. The next steps are used in the computation of this value:

First, all the input data is normalized, then, the values are multiplied by some factors selected by the user in the settings (I will talk about all this in depth in the section about Tweaks and Enhancements) and the values are added to create a number representing the weight of the meal.

2.6.6 Suggestion algorithm

This is the essential part of the application, all the steps of the implementation from above are used in this algorithm. In this part, I will describe the algorithm itself, without any improvements and in the next section, I will talk about all the modifications made in order to get better results.

The main idea of the algorithm is to find the closest *k* elements to our target meal. This can be done in a couple of ways:

One way is to sort the array of items and use a binary search over it to find the closest values. Depending on the search algorithm used, this method would produce a complexity of minimum O(N log N), and since the n, in this case, is about 13000, this solution does not suffice.

Max Heap and Priority Queue

Another solution and the one used here is to use a *Priority Queue* implemented as a *Max Heap* in order to keep the k closest elements there at all times and just update the heap accordingly. This solution produces the complexity of O(N log K).

A priority queue is an abstract data type, similar to stack or queues in which each object also has a *priority* tied to it. In this case, the closer the object is to the target meal, the greater the priority (the previously calculated *weight* value is used to establish this part). There are a couple of implementation methods for the priority queue, but one of the most common and most efficient is using a heap.

A max heap is a specialized tree-based data structure representing an almost complete tree that satisfies the property: in a max heap, for any given node A, if B is the parent node of A, then the value of B is greater than or equal to the value of A. It

is the most efficient implementation of *priority queue* which is an abstract data type that is actually often referred to as *heap*. One of the most common misconceptions about the Heap is that it is an ordered data structure, however, the heap is actually only partially ordered (see 2.24), the highest priority element being stored at the root.

The most common and useful applications of the heap are when it is necessary to modify the element with the highest priority repeatedly, in this application it means to replace the root of the heap with an element closer to our target meal. Since the operations for insert and removal of this implementation of the priority queue are O(log N), this is why this approach is a good fit for the algorithm.

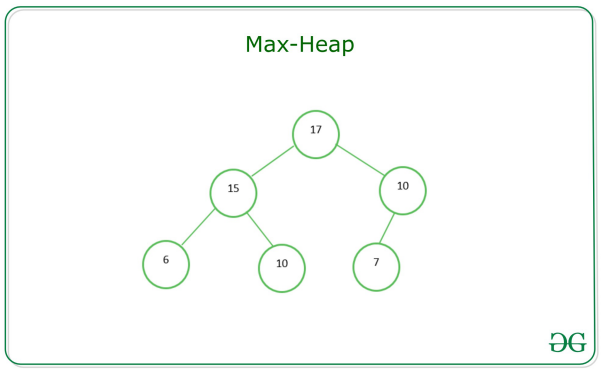

Figure 2.24: Illustration of the Max Heap by GeeksforGeeks

The algorithm

The main steps of the suggestion algorithm are:

- Create a *Max Heap* containing *k* pairs of objects and differences between them and the target meal that fit the goal chosen by the user
- For each k+1 meal candidate do the following:
    1. Calculate the distance between the target meal and the candidate
    2. Check to see if the item is in the maximum caloric surplus and minimum caloric deficit allowed
    3. If this is not true or the difference is bigger than the root of the heap ignore
    4. Otherwise remove the root, insert the new element and remove the chosen element from the list in order to avoid duplicates
    5. If the distance from the root to target is less than *eps* stop the execution
- In the end, the heap should contain exactly k elements

For the implementation of Priority Queue I chose the one from *java.util* package, and it will hold pair of Meals and their distance to the target meal. According to the documentation [19], by default the elements are stored in their natural order but, by supplying the queue with an instance of *Comparator* class, the items will be stored based on their distance (see 2.25).

See [1] for pseudocode and 2.26 for implementation.

```
75          // Make a max heap.
76          PriorityQueue<Pair<MealDTO, Double>> pq = new PriorityQueue<>(
                    You, 16 seconds ago | 2 authors (Berla Ewald and others)
77                  new Comparator<Pair<MealDTO, Double>>() {
78                      public int compare(Pair<MealDTO, Double> p1, Pair<MealDTO, Double> p2) {
79                          return p2.getValue().compareTo(
80                                  p1.getValue());
81                      }
82                  });
```

Figure 2.25: The initialisation of Priority Queue with Comparator

2.6.7 Tweaks and enhancements

I decided that simply pressing a button and seeing what the results of running the algorithm are isn't a feasible way to test the accuracy and diversity of the suggestions, so I implemented a hidden menu in the front-end application that will show only if the user's role is either *Administrator* or *Developer*. This menu allows running simulations of the algorithm and plots two charts: one for displaying how many times each meal is suggested, and one with the accuracy of the suggestions (how close are the suggestions to the desired caloric intake). These plots will help in tweaking the algorithm in such a way that the results will be better.

Data normalization

Since the calculation of the *weight* of a meal contains the time score, the price score and these three values all belong to different intervals, I decided that data normalization ([20]) is a must in order to bring them as close as possible.

Algorithm 1 Suggestion Algorithm pseudocode

```
Require: k ≥ 0
  function GetKClosest(k, target)
      result ← new [];
      pq ← new PriorityQueue;                ▷ Contains Pairs of Meal and Diff to target
      index ← 0;
      number ← 0;
      while number < k and index < Meals.size() do          ▷ Insert k initial items
          meal ← Meals[index];
          if ChooseBasedOnGoal(meal, target) and meal.type ≠ dessert then
              pq.insert([meal, difference]);
              number := number + 1;
          end if
          index := index + 1;
      end while
      for i:=k to Meals.size() do                          ▷ Process remaining items
          diff ← |CalculateDistance(Meals[i], target)|;
          if !IsAccepted(diff, Meals[i], pq[0]) then
              continue;                          ▷ Skip items that don't fit the criteria
          end if
          meal ← Meals[i];
          pq.pop();                                                    ▷ Remove root
          pq.insert([meal, diff]);                                     ▷ Insert new item
          Meals.remove(meal);                    ▷ Remove the already suggested meal
          if diff < eps then
              break;                             ▷ Stop if result is precise enough
          end if
      end for
      while !pq.IsEmpty() do
          result.insert(pq.pop().value);                    ▷ Construct result array
      end while
      return result;
  end function
```

```java
/**
 * Returns the closest k meals for a target meal
 * @param target - representing the target used as a reference by the algorithm when searching for related meals
 * @param k - the number of recommendations for the target meal
 * @return <code>ArrayList<MealDTO></code> of recommendations for the target meal
 */
private ArrayList<MealDTO> getKclosest(MealAlgorithmDTO target, int k) {
    ArrayList<MealDTO> result = new ArrayList<>();

    // Create a max heap.
    PriorityQueue<Pair<MealDTO, Double>> pq = new PriorityQueue<>(
            You, 13 seconds ago | 2 authors (Berla Ewald and others)
            new Comparator<Pair<MealDTO, Double>>() {       // By supplying a comparator the order of the pairs will be based
                public int compare(Pair<MealDTO, Double> p1, Pair<MealDTO, Double> p2) { // on their respective difference
                    return p2.getValue().compareTo(
                            p1.getValue());
                }
            });

    // Build heap of difference with first k elements that fit into the desired caloric range
    int index=0;
    int nb=0;
    while(nb < k && index < this.meals.size()){
        MealDTO meal = this.meals.get(index);
        if(chooseBasedOnGoal(meal, target) && (!meal.getType().equals("dessert"))){
            pq.offer(new Pair<MealDTO, Double>(this.meals.get(index), Math.abs(calculateDistance(this.meals.get(index), target))));
            nb++;
        }
        index++;
    }

    for (int i = k; i < this.meals.size(); i++) {       // Now process remaining elements.
        Double diff = Math.abs(calculateDistance(this.meals.get(i), target));

        // If difference with current element is more than root, then ignore it.
        if (!isAccepted(diff, this.meals.get(i), target, pq.peek().getValue()))
            continue;

        // Else remove root and insert
        MealDTO meal = this.meals.get(i);
        pq.poll();
        pq.offer(new Pair<MealDTO, Double>(this.meals.get(i), diff));
        this.meals.removeIf(m -> m.getUniq_id().equals(meal.getUniq_id())); // removes the already chosen meal

        // check if we found the closest matches that correspond to the epsilon and stop the search
        Double rootDifference = pq.peek().getValue();
        if (rootDifference < eps)
            break;
    }

    while (!pq.isEmpty()) {         // Get the contents of heap.
        result.add(pq.poll().getKey());
    }

    return result;
}
```

Figure 2.26: Algorithm implementation

The *scale to a range* method is used for the time score and price score since the values are pretty uniformly distributed and the bounds of the intervals are exactly 0 and 5. However, for the calories, *logarithmic scaling* makes much more sense since the interval is way bigger and most of the recipes have a caloric value that ranges between 300 and 600, so the data is not so uniformly distributed.

After applying this transformations and running the algorithm for a thousand iterations and for a target meal with five hundred calories we observe the graphs at figure 2.27.

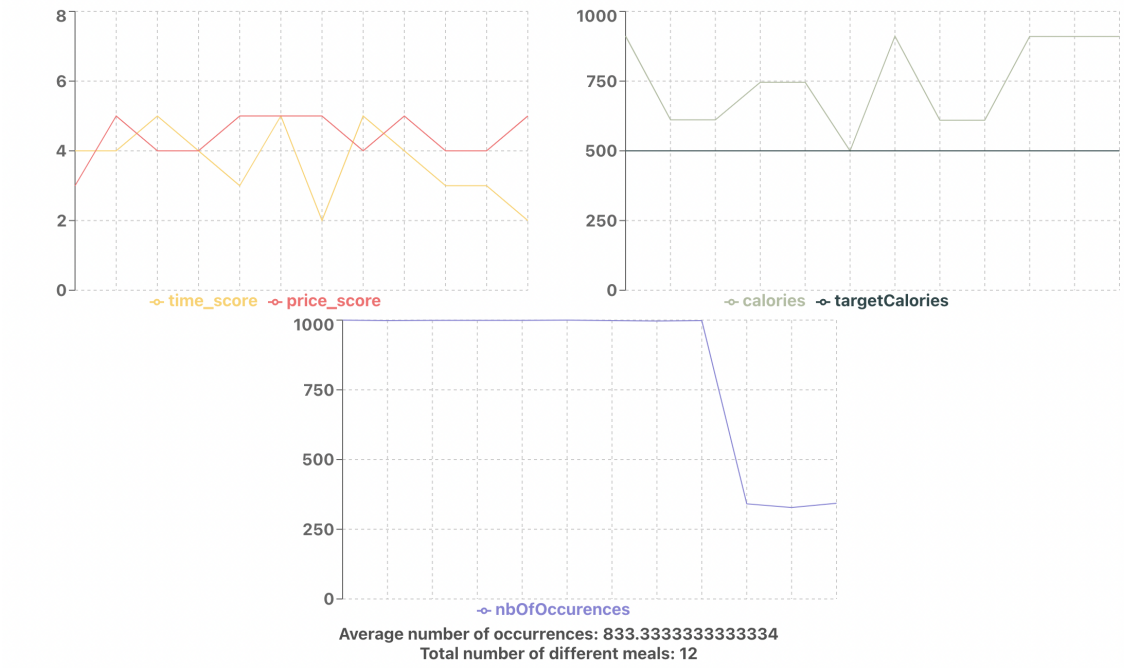

Figure 2.27: Graphs of 1000 iterations — no enhancements

A few key takeaways from these graphs are the fact that for a thousand suggestions with the same input and no enhancements the algorithm returned the same twelve recipes while also being very unprecise about the caloric intake of the meals (the values range from 500 to 1000).

Getting more results

The problem of getting more results can be solved quite easily since we know that before the search for suggestions happens, the list of meals is shuffled but the results are almost exactly the same for each run. This is because the algorithm tries to find close weights down to every decimal possible and for the same input, it finds the same 10 to 15 meals. A simple solution is to add an *Epsilon* value and to stop the algorithm whenever we have k meals that are close enough to the target meal. After doing so, we can see a massive improvement in the diversity of suggested meals (see figure 2.28), the algorithm now returns over three thousand suggestions

but they are very far away from what we are looking for (the input is five hundred calories and the algorithm returns recipes with calories from 300 to 1400). However, another improvement to notice is how much the average occurrence of a meal has decreased, from 800 times in the previous state to 2.7, this means that the duplicate suggestions are much more uniformly distributed across iterations.

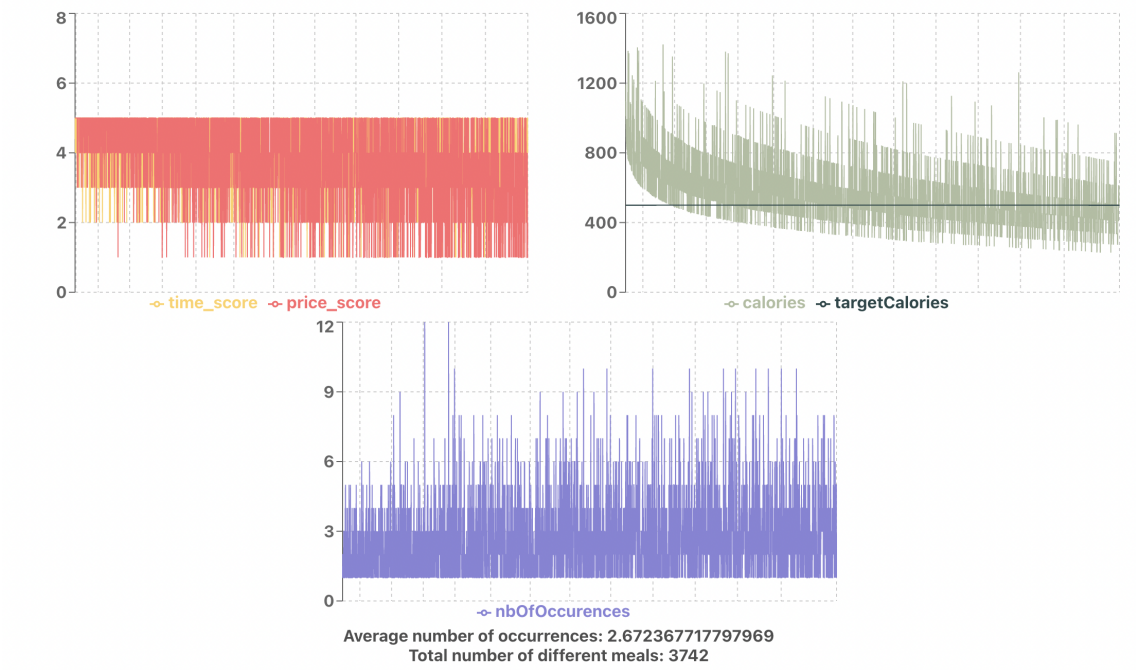

Figure 2.28: 1000 iterations — added epsilon value

Getting accurate

The inaccuracy is a side effect of normalizing the data since now the three components of the *weight* value have the same magnitude in calculating it (a weight of 2.69 can be obtained either with a recipe having 500 calories and the two scores to zero, or with a recipe having 200 calories and the scores of two out of five). Since the calories of a meal is the most important component when we search for suggestions, I added a multiplier to it that ranges from five to ten, depending on how close the user wants the suggestions to be, or how diverse. I decided that it's best to leave this trade-off between diversity and accuracy to the user's choice, so under *My profile* tab and settings, there is a slider that will adjust how the algorithm will search for suggestions.

For example for a multiplier of *5* after a thousand iteration the algorithm will return about 2000 suggestions with calories from 800 to 420 (see figure 2.29 the graph for calories is sorted based on the calories of each meal to be easier to interpret).

Running the algorithm again but with a multiplier of *10*, the yielded results now contain meals with calories ranging from 630 to 470, and looking at the graph, you

can see the results are on average much more closer to the targeted line (see 2.30).

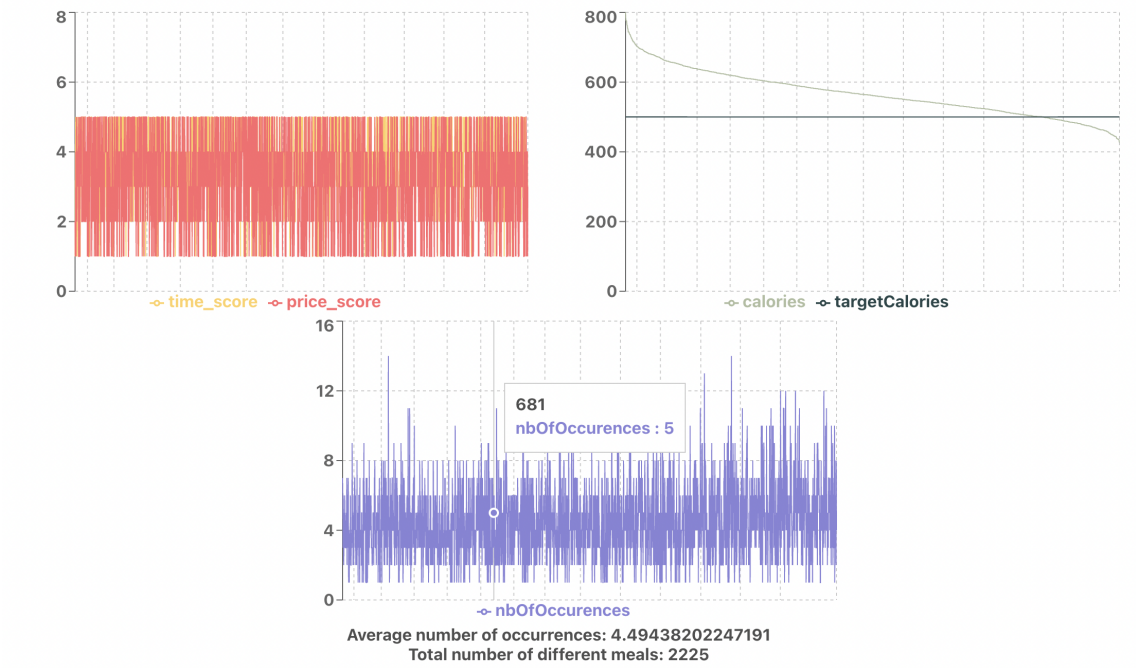

Figure 2.29: 1000 iterations — added 5x multiplier for calories

Limiting the results

Even if the overall accuracy of the algorithm has increased by a lot in the previous step, we still have to limit the surplus or the deficit of the suggested meals over our targeted meal to a smaller value as getting a meal with 800 calories instead of one of 500 calories isn't a good idea. Since we also want to allow the flexibility of the user to decide if he wants more diversity or more accuracy, based on the chose multiplier, a value has to be determined such that the the suggested meal's calories vary in range.

I decided that for the users who want more diversity 80 calories should be enough, while for those who want more accurate results, about 30 calories is the way to go. To obtain these values from the multiplier, we simply multiply the chosen factor with 10 and subtract the value from 130 (such that the minimum interval is at least -30, +30).

Running the algorithm now with 10x multiplier, we can see that the total number of meals has decreased by more than 500 but the values are much more accurate and the meal with the biggest number of calories has 530 calories (see figure 2.31).

Taking user's preference into account

The last improvement added to the algorithm is adding a multiplier to the time

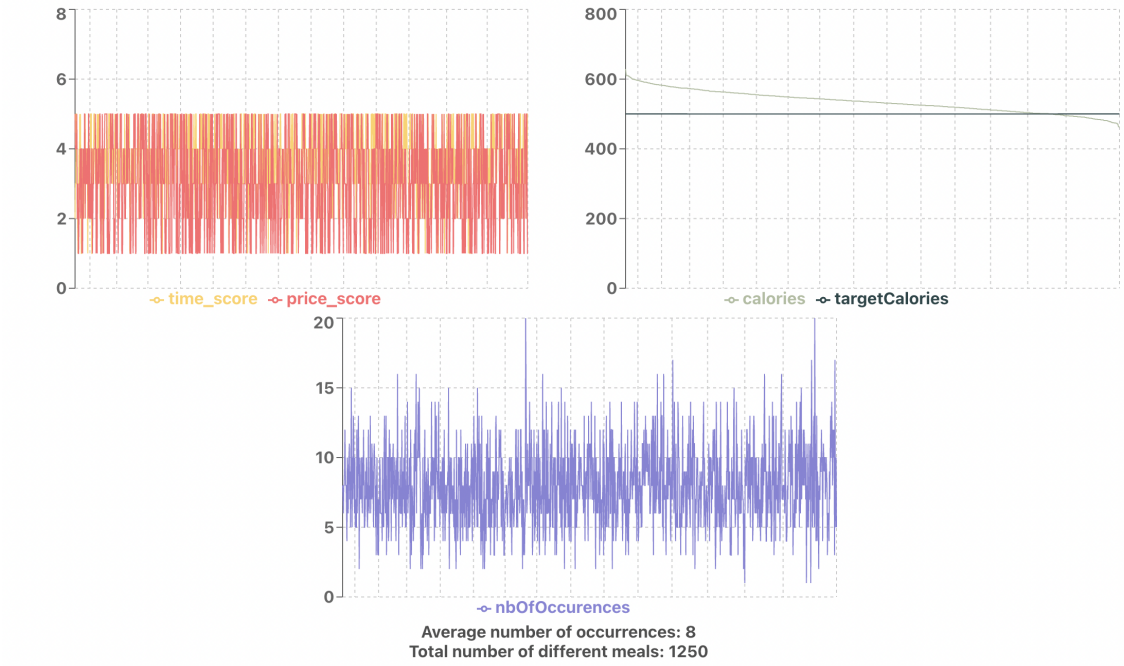

Figure 2.30: 1000 iterations — added 10x multiplier for calories

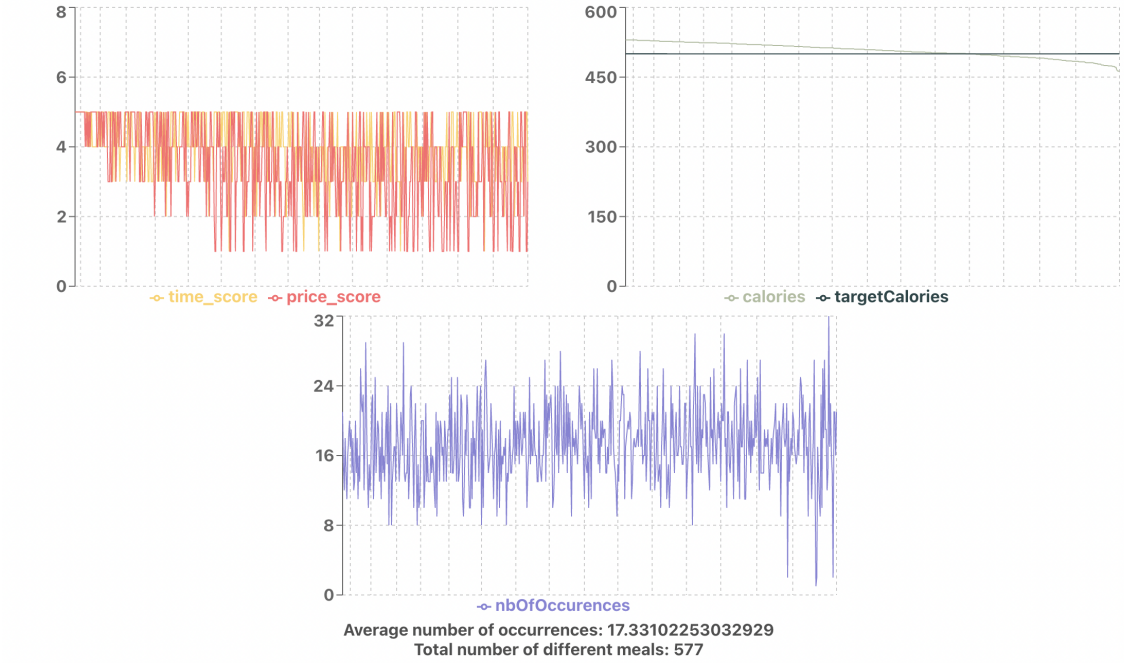

Figure 2.31: 1000 iterations — added limiter

score and the price rating. In the above graph, the time and price scores are uniformly distributed between 0 to 5, however, a user might want to see only meals that take less time to cook. By adding a multiplier between 1 to 5 to each of those, the user can select how much magnitude each of the scores have.

The main benefit of this approach instead of using a simple filter is the flexibility, a multiplier of 5x will yield results that have a score of 5 and sometimes 4, there is no strict policy. This way, we get to keep more diversity over results.

Running the algorithm with a caloric multiplier of 10x and a price rating multiplier of 5x will result in much less results being shown, but most of the results have a higher price rating (see figure 2.32).

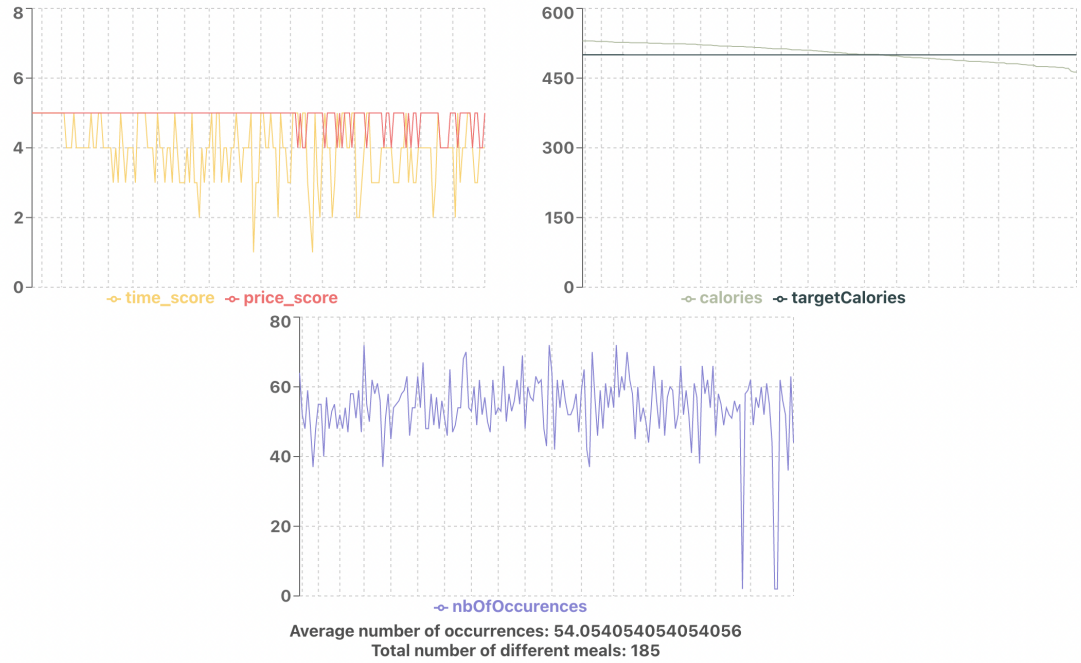

Figure 2.32: 1000 iterations — added multiplier for time and price fields

2.6.8 Register and Login

For the *Register* and *Login* actions I used the implementations from *Spring Security* [21] (see figure 2.34).

I started by making my *User* class implement the *UserDetails* interface from the spring security core package (see figure 2.7). From here, I implemented the necessary methods and I used inheritence to extend the *UserDetailsService*. Then, I created

my own implementation for the JWT token (see figure 2.33) and in the class that extends the *WebSecurityConfigurerAdapter* implementation, I overrode the *configure* method and I added a custom filter that will be used later to check the JWT token for authorization.

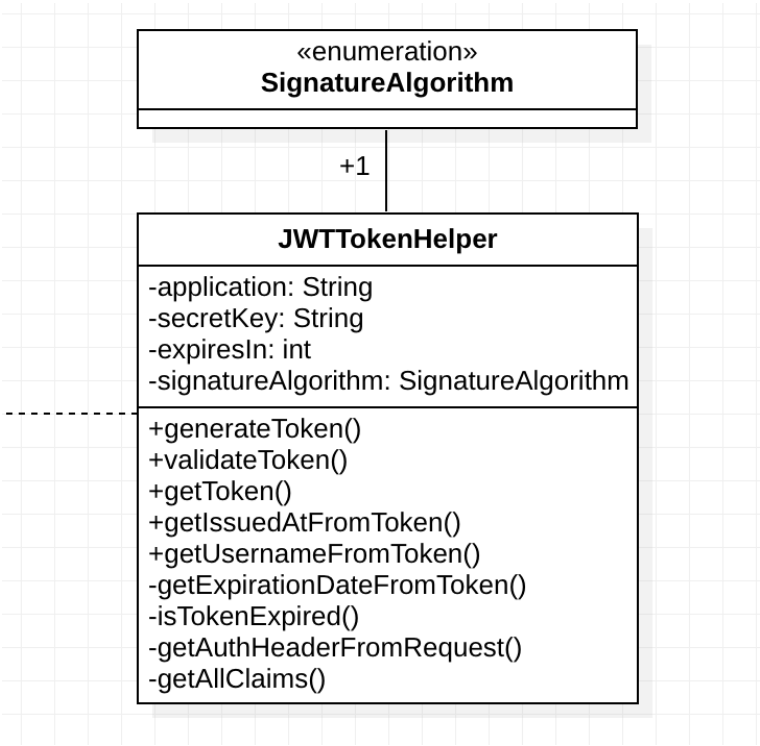

Figure 2.33: Class diagram for JWT token

For *Register*, I simply implemented an insert into the user table using a custom DTO. From here, if the insertion succeded, the application will publish using *ApplicationEventPublisher*, a new custom event: *RegistrationCompleteEvent* and another component: *RegistrationListener* that implements the *ApplicationListener* interface, will listen for that event. When the event is issued, the email confirmation behaviour will start, a new *VerificationToken* will be created and sent to the user's email address. The user will be able to log in successfly only after verifying the email address. Also, the token expires after 24 hours.

2.7 Testing

2.7.1 Session-Based Test Management

Software Testing Times[22] says about Session Based Test Management that it is a formalized approach that uses the concept of charters and the sessions for performing the Exploratory Testing.A session is not a test case or bug report. It is the

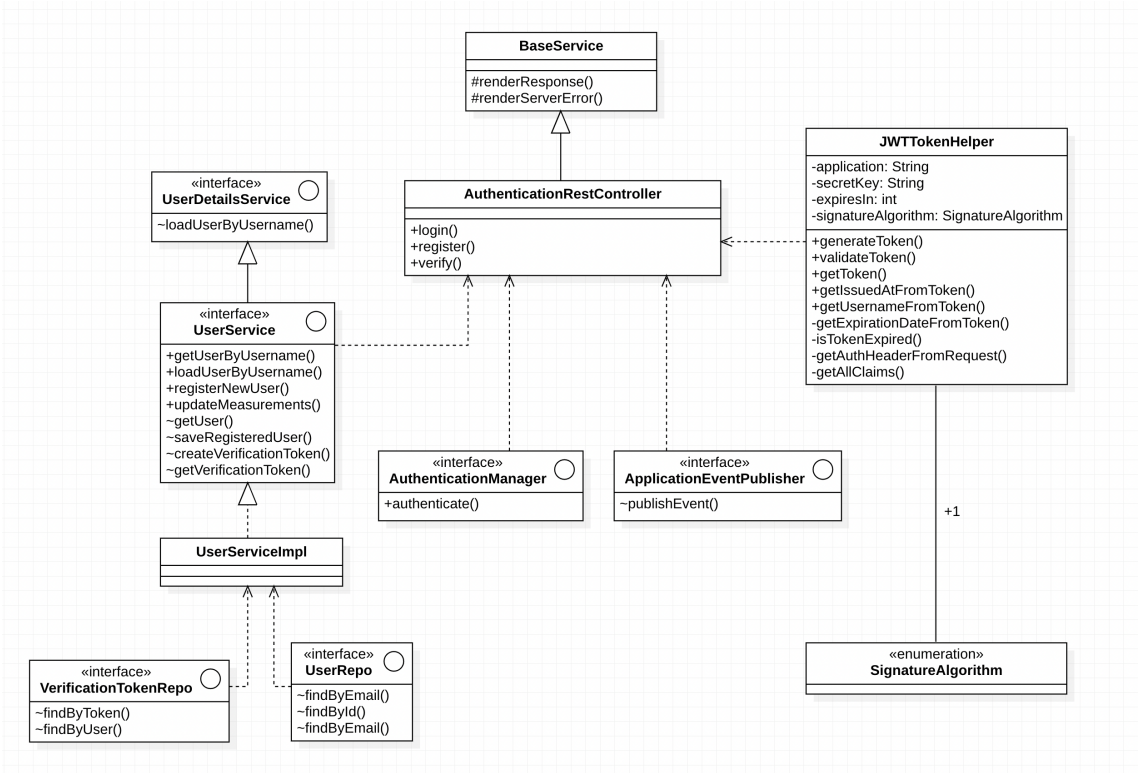

Figure 2.34: Class diagram for the Register and Login processes

reviewable product produced by chartered and uninterrupted test effort. For guidance I also used the article by Jonathan Bach: Session Based Test-Management [23]. A report of two completed sessions can be seen in figure 2.35.

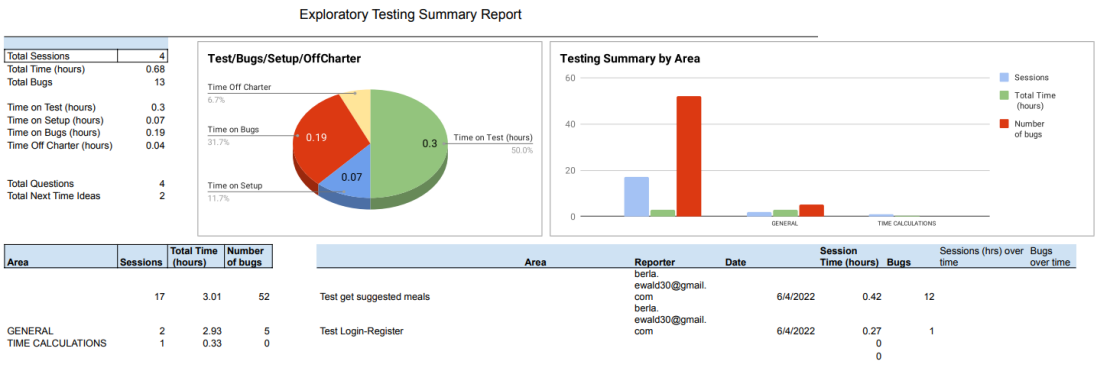

Figure 2.35: SBTM report

2.8 Deployment

2.8.1 Heroku

Heroku is a PaaS (Platform as a Service) and one of the first cloud platforms

that supports several programming languages such as Ruby, Java, Node.js, Clojure, Python, Php and Go. Both of the front-end and back-end applications are hosted using Heroku. The reason this platform was chosen is mainly because of the simplicity of the deploying process. By abstracting away the servers in some sort of smart containers called *dynos*, Heroku takes the responsibility of making sure they are secure, and by making the process of deploying easier and allowing the developer to focus more on coding instead of infrastructure.

Another big benefit of using Heroku for deployment is that it is scalable, both in horizontal and vertical environments, this being done by either upgrading the dynos or adding new ones.

2.8.2 ClearDB

ClearDB is a high performance Database as a Service system used for MySQL powered applications. It stores and manages MySQL dabatases for you thus removing the need to deal with database servers, advanced storage, It support and database failures.

ClearDB can be found on Heroku as a free addon and therefore it is used by me to host the application database.

2.8.3 Some prerequisites

Before deploying the application, the user needs to install heroku-cli navigate to the application directory and type the command 'heroku create app-name'. This command is needed for both the back-end application and front-end application, and will create the heroku application alongside with adding the heroku git remote. See 2.36 for an example.

```
$ mkdir example
$ cd example
$ git init
$ heroku apps:create example
Creating ⬢ example... done
https://example.herokuapp.com/ | https://git.heroku.com/example.git
Git remote heroku added
```

Figure 2.36: Example of using 'heroku create' command

After the heroku application has been created the user can push the appication code to the heroku remote by typing the following command in the terminal: 'git push heroku'. This command will push all the existing code from the main branch of the application to the heroku remote and then the building process of the

app will start. After the build has finished, the application will try and start. By running 'heroku info' one can view information about the application, such as the URL that is used to access it. (See 2.37) At this point the application is deployed,

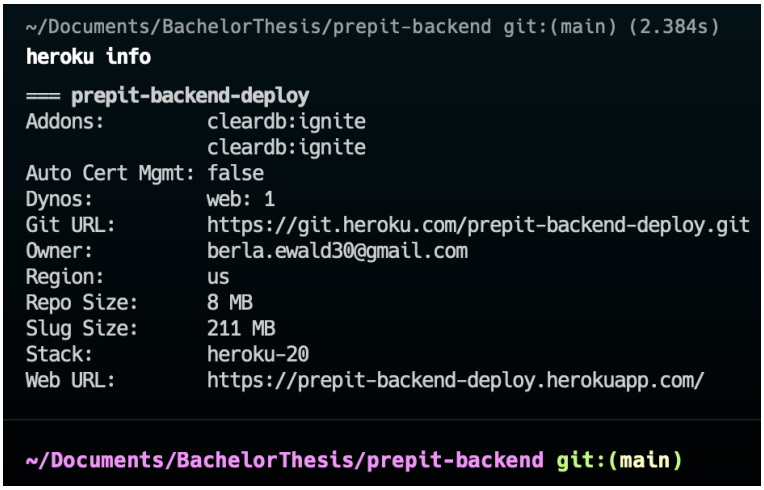

Figure 2.37: Example of using 'heroku info' command

these steps are the same for both applications. However, at this point when visiting the web URL of the application an error will be displayed since some configurations are missing.

2.8.4 Deploying the Database

There are a few ways to deploy a MySQL database on heroku but all start by installing the ClearDB addon from Heroku website. After installing the package, the user has to connect to the provided database in MySQL Workbench using the credentials given by ClearDB on their website.

After a connection has been established, the data can be exported into a file from the local database and imported back into the remote database. Sometimes the local databse and the remote one will use different encodings, thus the user has to make sure that the exported file has the same encoding used in SQL statements as the remote database.

2.8.5 Back-end

As a security measure, on the back-end application the secrets are stored in a separate file named *secrets.properties*. This file includes information like: email username, email password, JWT secret key, datasource url username and password and the prefix of the URL used for verification of the email address. This file is included in .gitignore such as it is not available to public, but the application will not work without it.

The solution is to provide this information in the form of configuration variables so that they are added when the application is started. To do so, the user has to navigate to Heroku website, select the application click on the *Settings* tab and select *Reveal Config Vars*. Here the user has to supply:

- application.verification.url - This will contain the deployed app URL prefix (all info before the application context:
  PrepIt)
- jwt.auth.secret_key - a secrey key used to decode the JWT tokens
- spring.mail.username - the username of the application email
- spring.mail.password - the password of the application email

In figure 2.38 the configured variables are show. Besides the mentioned variables there are two more values configured from the ClearDB addon. These are used in order to connect the application to the remote database and are configured automatically when ClearDB is installed for the application. After these are configured,

Figure 2.38: Example of stored configuration variables

the back-end application should be deployed and working successfully.

2.8.6 Front-end

The process for deploying the front-end application is pretty straight forward. After creating the Heroku application and pushing the code to the Heroku remote, the user also has to setup a configuration variable. This is used in order to distinguish from the *production* and *development* environments. The variable is located in

a .env file at the root of the directory that is also included in .gitignore, so for development purposes the user has to create the file and set the value to *development*. For deploying, the user has to navigate to the heroku website and under config vars add the following entry:

REACT_APP_API_ENV = production

An example can be found at 2.39.

Figure 2.39: Example of stored configuration variables

This is required such that the correct back-end application URL is used. When the variable is set to development the URL for the local instance of the back-end application is used, otherwise the URL for the deployed instance is used (see 2.40).

```
12  const BASE_URL = process.env.REACT_APP_API_ENV=== 'development' ?
13              'http://localhost:8080/PrepIt' :
14              'https://prepit-backend-deploy.herokuapp.com/PrepIt';
```

Figure 2.40: Instruction for determining what URL to use

Chapter 3

Conclusion

3.1 What was achieved

The main goal of the thesis was developing an application that would make meal planning much easier for people regardless of their knowledge on the subject. This was achieved by implementing an algorithm that by having only marginal information about the user (age, height, weight, activity level and their desire to lose or gain weight) builds a complete and diversified meal plan, along with up to nine recommendations for each meal of the day.

The algorithm was then tweaked and enhanced in order to improve as much as possible the accuracy while maintaining the diversity of the results. However, since some users would prefer more accurate results while others would prefer for them to be more diversified, the option to choose between the two was implemented to allow for more flexibility and to please a greater mass of users.

Since there are so many food recipes, the option to save a meal for later reading was introduce such that a user can come back later to a recipe that they enjoyed.

3.2 What can be improved

While there are about 12000 food recipes in the dataset, one of the most important improvements would be to let users create new recipes. This would also be a good time to make use of the role based system, a simple user would create a meal and a manager would verify and accept or decline it.

Another idea for improvement would be to ask the users to input their weight once a week and create a graph with the current progress that can be displayed under *My Profile* page.

The front-end application is responsive and works for mobile devices, but a new mobile only application would be a big improvement of the user experience. A good

option for building the application would be the React Native framework which allows creating cross-platform mobile applications.

Bibliography

[1] Hannah Ritchie and Max Roser. Obesity. *Our World in Data*, 2017. https://ourworldindata.org/obesity.

[2] Obesity and overweight. https://www.who.int/news-room/fact-sheets/detail/obesity-and-overweight. Accessed: 2022-06-05.

[3] Physical activity. https://www.who.int/news-room/fact-sheets/detail/physical-activity. Accessed: 2022-06-05.

[4] Rena R Wing and Suzanne Phelan. Long-term weight loss maintenance. *The American Journal of Clinical Nutrition*, 82(1):222S–225S, 07 2005.

[5] Maven documentation. https://maven.apache.org/guides/index.html. Accessed: 2022-04-05.

[6] Amine Benelallam, Nicolas Harrand, César Soto-Valero, Benoît Baudry, and Olivier Barais. The maven dependency graph: A temporal graph-based representation of maven central. *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*, pages 344–348, 2019.

[7] Spring boot documentation. https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/#legal. Accessed: 2022-04-05.

[8] Refactoring guru adapter pattern. https://refactoring.guru/design-patterns/adapter. Accessed: 2022-04-05.

[9] Refactoring guru builder pattern. https://refactoring.guru/design-patterns/builder. Accessed: 2022-04-05.

[10] Reactjs documentation. https://reactjs.org. Accessed: 2022-04-05.

[11] Redux documentation. https://redux.js.org. Accessed: 2022-04-05.

[12] BBC Food recipes dataset. https://crawlfeeds.com/datasets/bbc-food-recipes-dataset. Accessed: 2022-04-01.

[13] ISO 8601:2004 data elements and interchange formats — information interchange — representation of dates and times. https://www.iso.org/standard/40874.html. Accessed: 2022-06-04.

[14] Metmomi iso datetime parser. https://github.com/metomi/isodatetime. Accessed: 2022-06-04.

[15] Bbc good food sesame spinach. https://www.bbcgoodfood.com/recipes/sesame-spinach. Accessed: 2022-06-08.

[16] Eun Kyung Kim Sun Hee Lee. Accuracy of predictive equations for resting metabolic rates and daily energy expenditures of police officials doing shift work by type of work. *Clinical nutrition research vol. 1,1*, 2012.

[17] Joan Bader Charlene Compher, Robert Cato and Bruce Kinosian. Harris-benedict equations do not adequately predict energy requirements in elderly hospitalized african americans. *Journal of the National Medical Association*, 2004.

[18] Omni calculator meal calorie calculator. https://www.omnicalculator.com/health/meal-calorie. Accessed: 2022-06-01.

[19] Priority Queue documentation. https://docs.oracle.com/javase/7/docs/api/java/util/PriorityQueue.html. Accessed: 2022-06-03.

[20] Normalization techniques. https://developers.google.com/machine-learning/data-prep/transform/normalization. Accessed: 2022-06-03.

[21] Spring security documentation. https://www.baeldung.com/security-spring. Accessed: 2022-04-05.

[22] Session based test management (sbtm). http://softwaretestingtimes.com/2012/04/session-based-test-management-sb.html. Accessed: 2022-06-04.

[23] Jonathan Bach. Session-based test management. *Software Testing and Quality Engineering (STQE)*, 2000.