**Purpose**

The purpose of this Document lies in the fact that we want to document the results of the requirements elicitation and we also want to document to analysis activities. It also completely describes the system in terms of functional and nonfunctional requirements.

**Audience**

As our target system are all desktop operation systems (Windows, macOS, Unix), either as Java or as browser-based application which communicates with the Spring Boot server application. So, everybody that uses one of the operating systems above is part of our intended target group.

**Table of Contents**

1. Introduction 3

1.1 Purpose of the system 3

1.2 Scope of the system 3

1.3 Objectives and success criteria of the project 3

1.4 Definitions, acronyms, and abbreviations 3

1.5 References 3

1.6 Overview 3

2. Current system 3

3. Proposed system 3

3.1 Overview 3

3.2 Functional requirements 3

3.3 Nonfunctional requirements 3

3.3.1 Usability 4

3.3.2 Reliability 4

3.3.3 Performance 4

3.3.4 Supportability 4

3.3.5 Implementation Requirements 4

3.3.6 Interface Requirements 4

3.3.7 Packaging Requirements 4

3.3.8 Legal Requirements 4

3.4 System models 4

3.4.1 Scenarios 4

3.4.2 Use case model 4

3.4.3 Object model 4

3.4.4 Dynamic model 4

3.4.5 User interface 4

4. Glossary 4

**Document History**

|  |  |  |  |
| --- | --- | --- | --- |
| Rev. | Author | Date | Changes |
|  | Marc Roig Kunzmann | 20.07.2022 | Creation of this document for the team project. |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

# Introduction

*The main idea behind the project was to allow users to track their food more easily without much effort and based on this information our system should recommend the user recipes. The user can enter the food that he has available at home into our database, so it gets processed by our system. The system then calculates the given input and creates a list of recipes that the user can cook with it. Based on this list the end-user can then cook one of the listed meals.*

## Purpose of the system

The main problem was to solve the following problem: People waste too much time on deciding what to cook just to end up cooking the same every day. They need a way to get recipe recommendations from what they have stored in their fridge.

## Scope of the system

The scope of the system will be explained in further detail if we look at the non-functional and functional requirements that we have set ourselves. With this list you can get a good look at what we want to achieve in the project work and what our goals are.

## Objectives and success criteria of the project

Our main objective is to offer our end users a good experience and actually help people solve the problem already mentioned above. We define the success of our project based on the number of implemented non-functional and functional requirements. But even if we don`t achieve all of our set goals, our main focus is that our system fulfills the basic functionalities like adding food deleting food or requesting recipes.

## Definitions, acronyms, and abbreviations

We think definitions were explained enough in detail and therefore we don’t need any acronyms or abbreviations. Our motivation should be easily understandable in words.

## References

We do not have much external references. We only used the Spoonacular API: https://spoonacular.com/food-api

## Overview

As we already explained our motivation behind the system in the parts above, we have covered the basic

# Current system

*With our system, we are currently able to let the user add new foods independently and then recommend recipes to the user, so our main target of this project is already met. We can provide a working system, with the help of which our users can get some help in everyday life in relation to the topic of food. Only minor things like a login service between our client and our server have not yet been implemented due to time problems. Additionally, we also have some more ideas in our minds that could be implemented even after this team project. Nevertheless, our system is already good and intuitive to use for the end user.*

# Proposed system

*In this part, we will get into further detail about the requirements and our analysis model from our team project.*

## Overview

*The system allows its users to enter some input in the form of food, that the user has at home. He can enter the name of the food and the given weight. He can also define some regulations about his diet for example “gluten-free”. Based on this combination of data our systems searches for recipes that match this metric. Then these recipes will get recommend to our user.*

## Functional requirements

We have come up with a series of functional requirements for our project, which we will briefly list below:

* **Track items**: The user can add items from its fridge to a database
* **Query recipes:** Recipes can be queried using the saved items in the database
* **Filter recipes based on diet**: The queried recipes can be filtered to fit a certain diet
* **Save recipes**: The user can save recipes and find them in a simple favorites tab
* **Scan barcodes of products**: The user can add items to the database through scanning the barcode of the product
* **Get random recipe**: The system displays a random recipe to the user which can be cooked right on the spot. The random recipe follows the users´ preferences if the user saved them
* **User profile with preferences**: The user can safe their preferred diet which will be respected in querying a random recipe or added automatically as a filter on queried recipes.
* **Remove used ingredients**: The used ingredients in a recipe can be removed in the quantity given in the recipe from the database automatically after cooking

## Nonfunctional requirements

Similar to the functional requirements already mentioned above, the non-functional requirements are also briefly listed here:

* **Usability**: The system should be intuitive to use, and the user interface should be easy to understand. Simple interactions should be completed in less than three clicks. Complex interactions should be completed in less than six clicks.
* **Conformance to guidelines**: The design of the system should conform to the typical usability guidelines such as Nielsen´s usability heuristics.
* **Server system**: A server subsystem with a couple of services must be used in the system. However, additional services like recipes and product information should be obtained from external services.

In addition, we also set ourselves some additional constraints:

* The version control system must be git
* Source Code Documentation must be in HTML format
* The server system must use the Spring Boot framework

### Usability

We already mentioned the usability aspect above and explained in detail what we expect our system to have.

### Reliability

Our system should of course be reliable as possible. Reliability is one of our priorities.

### Performance

Similar to the Reliability aspect, our system should be as performant as possible. We also define it as crucial.

### Supportability

As we explained our target group beforehand, we want our system to be able to run on all kind of operating systems. This is crucial for us in order to fully reach our target group

### Implementation Requirements

In the additional constraints we have already mentioned what we specifically expect from the implementation. The most important thing is that we want to work with Spring Boot.

### Interface Requirements

We had not defined some special interface requirements. Over the course of the project, we did not have any interface requirements. So, this point is not that relevant.

### Packaging Requirements

Similar to the interface requirements we did not specify the packaging requirements. So this point is also negligible.

### Legal Requirements

There a not strict legal requirements that we set ourselves. But we want to clarify that we try to avoid in breaking some kind of law or stealing code.

## System models

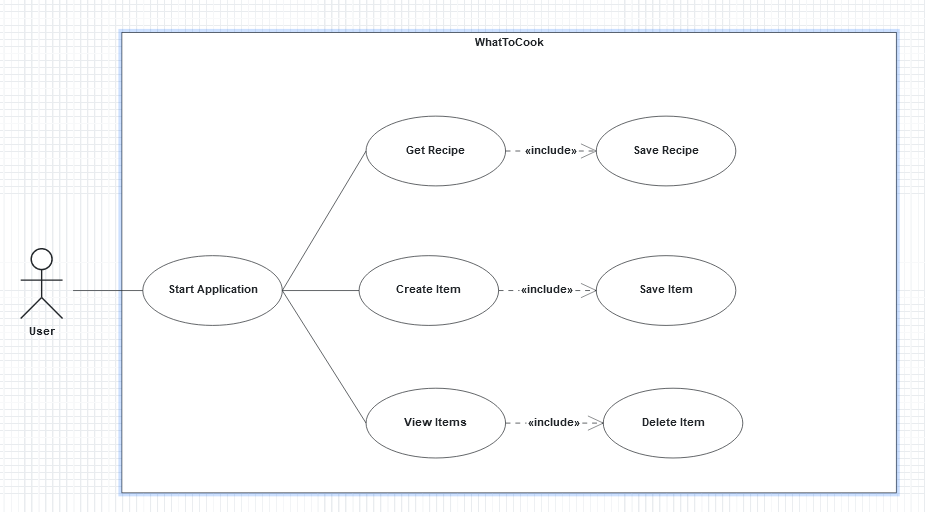
### Scenarios

This part is going to be covered in the following subsections of system models, so we will take a look at our use case, our object model, our dynamic models and about our subsystem decomposition. We will also provide some insight in the user interface.

But we also have defined the following scenario:

Lisa is a hard-working single mother juggling her kids and work. After a rough day at work, she usually has no energy left to waste on decisions, but she wants to feed her kids healthy food. She has a meal plan that she cycles through every week. After a few months she gets bored cooking the same every week. She needs a tool that makes it easy for her to find new recipes quickly with the same items in her fridge. A friend of her suggests she should start using “whattocook”. Lise can quickly fill her digital fridge through scanning barcodes or just entering the name. After tracking all items stored in her fridge, she wants to cook. She opens the app which serves her a recipe she can start with right on the spot. Lisa can also fetch more recipes based on her diet. She chooses one recipe with a compelling picture and starts to cook. The used ingredients get automatically removed from the digital fridge. Lisa and the kids liked the recipe. Lisa saves the recipe so she can cook it again.

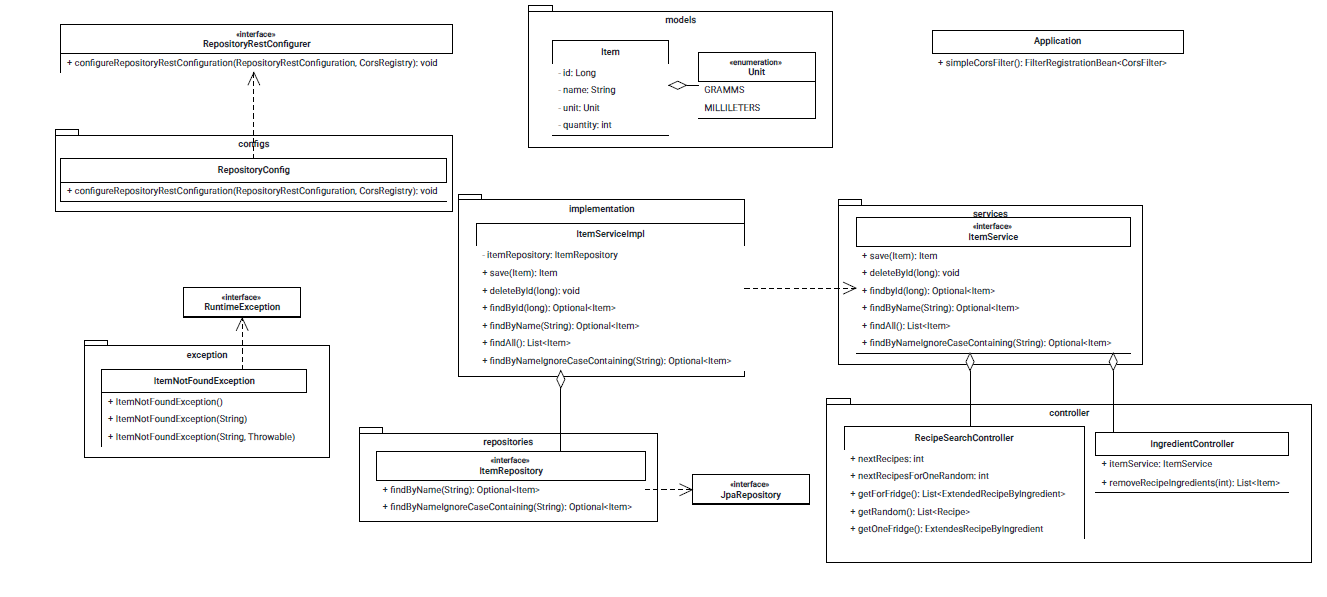
### Use case model



Here is our use case model, our customer starts the application and from here he can either search for an item that has already been created, create a new item himself or he can view all items that have already been created. Additionally, here is an alternative document for our use case diagram:



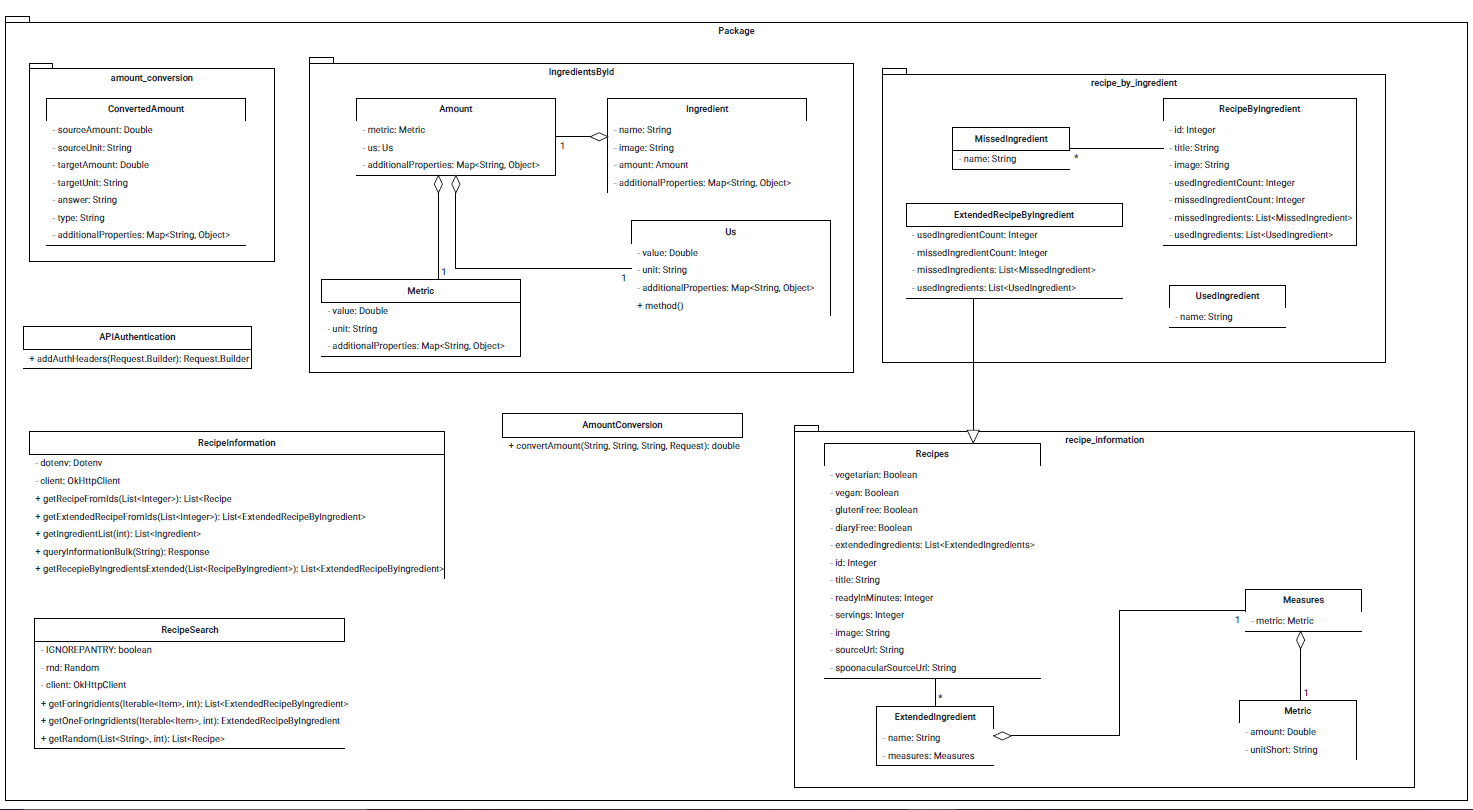
### Object model

**Part1**: 

The first part of our object model refers to the part of the system that was already included in the early development and on the other hand it is also the part that helps us internally to implement our own functionalities, the other in turn helps us with our Spoonacular API to interact.

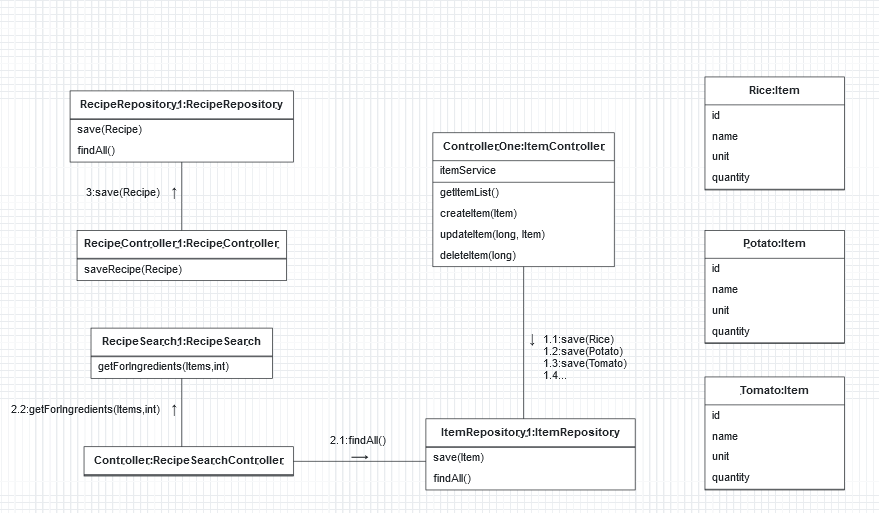
Unfortunately, part of the image quality is lost when you try to zoom out in Apollon, which is why here is a PDF where you can take a better look at our object model:



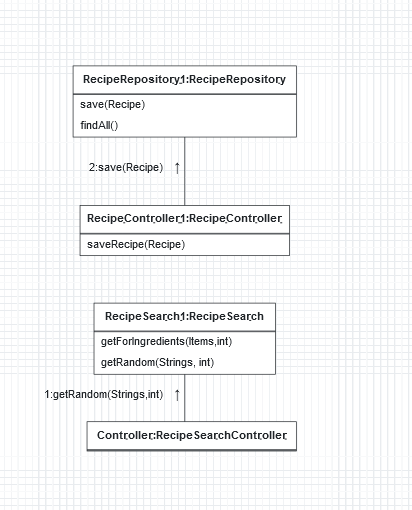
**Part2**: 

As already explained this part is used for the interaction with the Spoonacular API. Note: (You can see the connection between Spoonacular and our system in more detail in the subsystem decomposition)

### Dynamic model

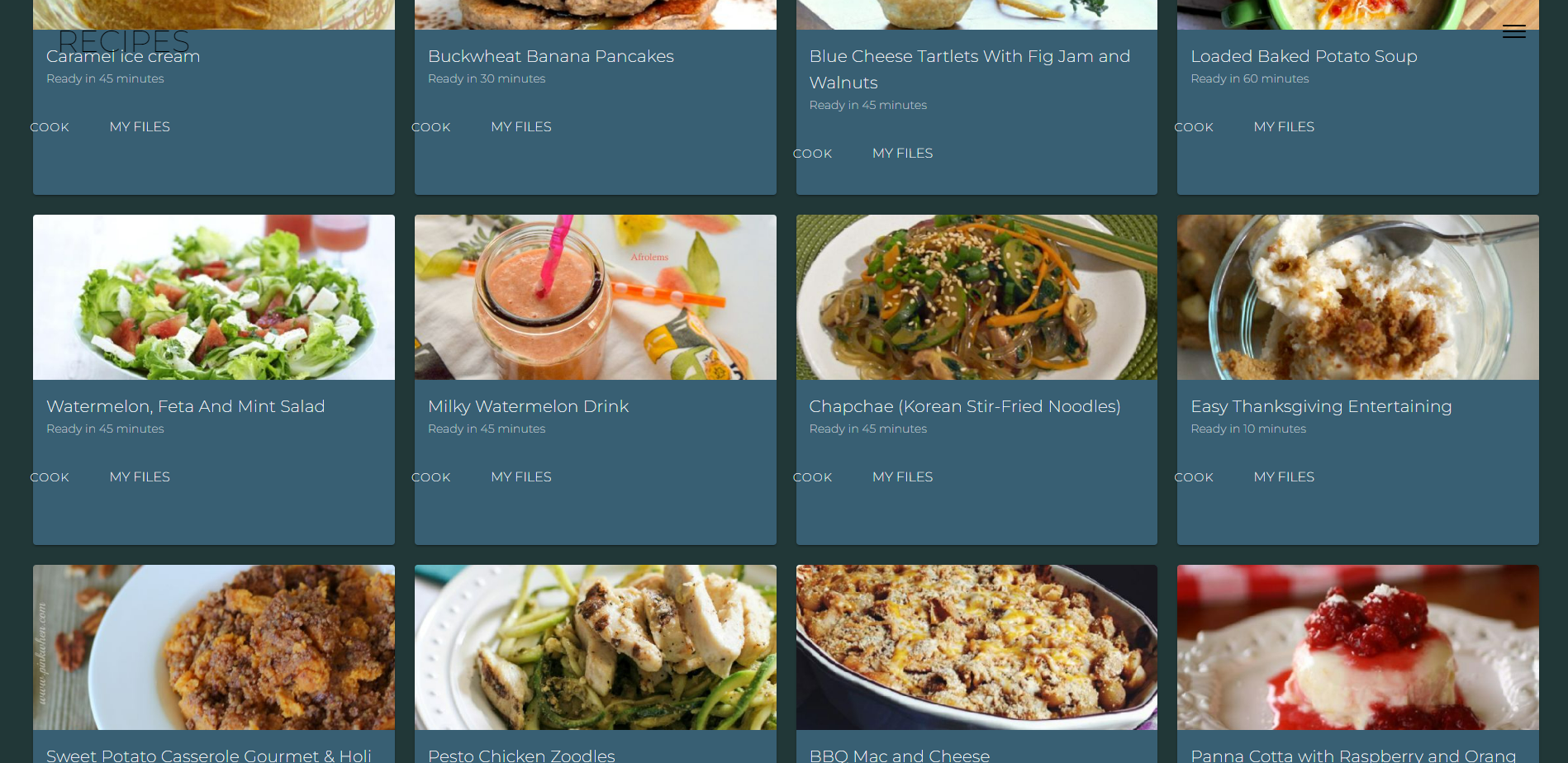


This dynamic shows the first case of our system. We have our end user who wants to add some Items to our Database. His input in the client triggers the save method in the server and thus ensures that our food is saved. A recipe is generated via the RecipeSearchController and the RecipeSearch. Lastly, this recipe is stored via the RecipeController and the RecipeRepository.



In our second use case we do not have the intention to save items on our own. Instead, we want to generate a random recipe. The input of the user´s client triggers the getRandom Method of RecipeSearch which in connection with the RecipeSearchController returns a random recipe. As we want to save it, we have the same cycle as in case 1, with the help of the RecipeRepository and the RecipeController, our recipe gets saved.

### User interface



Sampel picture what the user sees, when he opens our service. He sees the recipes and the associated cooking time.

# Glossary

*Spoonacular – API used for querying items/recipes*