**Purpose**

This document will cover our intended design goals, the subsystem decomposition (as UML diagram) and our hardware/software mapping.

**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 2

1.1 Overview 2

1.2 Definitions, acronyms, and abbreviations 2

1.3 References 2

2. Design Goals 2

3. Subsystem decomposition 2

4. Hardware/software mapping 2

5. Persistent data management 2

6. Access control and security 2

7. Global software control 3

8. Boundary conditions 3

**Document History**

|  |  |  |  |
| --- | --- | --- | --- |
| Rev. | Author | Date | Changes |
|  | Marc Roig Kunzmann | 19.07.2021 | Creation of this document for the group project |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

# Introduction

*The purpose of the Introduction is to provide a brief overview of the software architecture. It also provides references to other documents. Our software architecture relies on 3 important key components. We have out Client from which our end user can interact with the system. Then there is the Server that processes the data from the Client and then connects with the SpoonacularServer which helps us retrieving the recipes.*

## Overview

To explain our architecture here in words would be too incomprehensive, so we refer to the subsystem decomposition below. There you can understand our idea more easily.

## Definitions, acronyms, and abbreviations

We did use the standard definitions that should be understandable by everybody (things like Server and Client). Spoonacular is the API we used for querying data.

## References

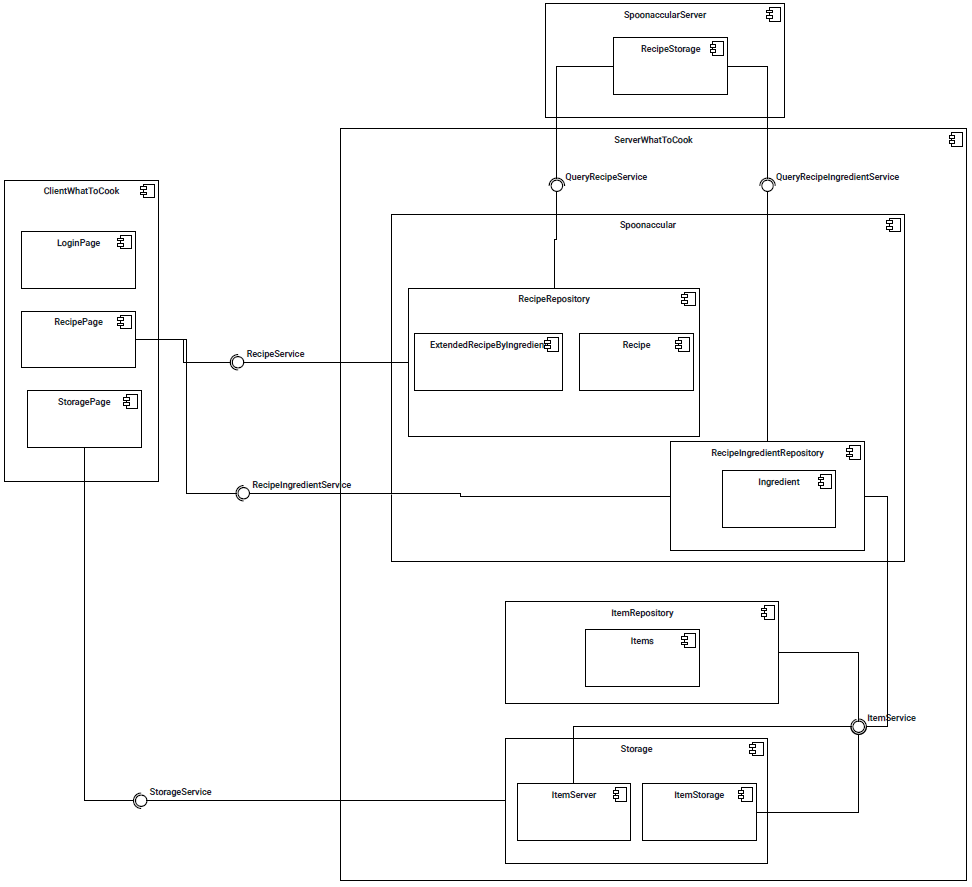
If you want to understand our main idea behind this project and why used this specific structure for our project, we recommend reading our RAD. There we go more into detail about the motivation etc.

# Design Goals

*The highest priority lies in the reliability and the consistency of our system. We want that the user can rely on our system at any given time. If the system fails, we have a huge problem that we try to avoid. But we also want that our useability is very good because we our system to be a advantage for our user not a burden. If the user has problems in using our system, he may neglect it and use another system. The worst case that could happen. So, we also got our functionalities that we want to implement into our system but in favor of the 2 factors above we need to make a tradeoff. We would rather sacrifice features than having problems with the other two factors.*

# Subsystem decomposition

*The structure of our subsystem decomposition can be seen on the following diagram:*



*So, our system consists of 3 core parts, the ClientWhatToCook, the SpoonacularServer and the WhatToCookServer. The client itself is not really complex it consists of the 3 different Pages, the Recipe Page that uses the recipe service of the Recipe repository and the Recipe ingredient service, the Storage Page that uses the Storage service provided by the Storage and the login page.*

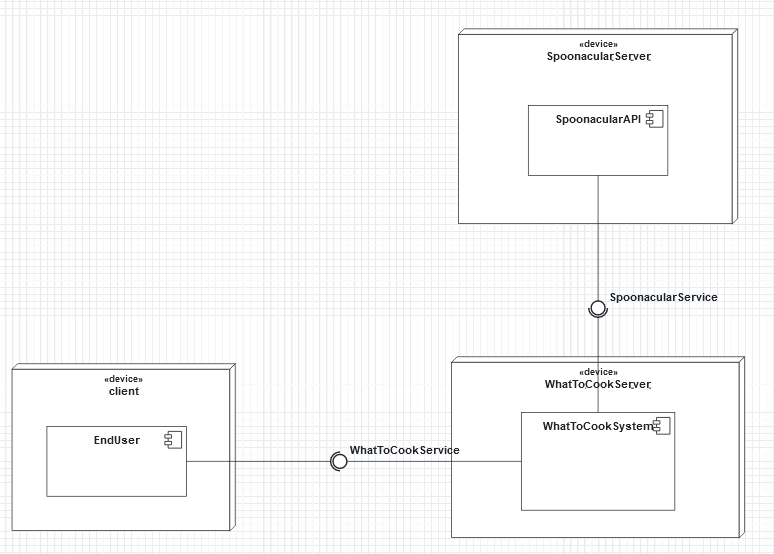
*The spoonacular server uses the query recipe service provided by the recipe repository and the query recipe ingredient service provided by our recipe ingredient repository.*

*Inside of our WhatToCook server we have the item service that is provided by the item repository and used by the Storage and the Recipe ingredient repository.*

*In case that the quality of the picture is not detailed enough for you in order to understand our subsystem decomposition. Here you can get a better insight:*

**

# Hardware/software mapping

*The hardware mapping relies on 3 key components. We have the software part of our spoonacular service that is mapped to a server that is run by the spoonacular provider. Then we have our WhatToCook service that is mapped onto our own server. At least we have our end users that have their software mapped on the device that they use i.e., phone, tablet, desktop etc.* 

# Persistent data management

We have an internal data base that stores our inserted items. We also take use of the Spoonacular API that allows us to take recipes out of their database.

# Access control and security

*This section describes the access control and security issues based on the nonfunctional requirements in the requirements analysis document. It also describes the implementation of the access matrix based on capabilities or access control lists, the selection of authentication mechanisms and the use of encryption algorithms. We have not been able to discover any specific security problems during the implementation, our databases should contain protective mechanism that deny fraudulent activity and the API we use has of course also been designed to be secure.*

# Global software control

*Because a user can only add one item at a time, we should be linear as from there the request goes through our system to Spoonacular and from there the date comes back to us. If several users should add items at the same time, then this should be also regulated via the database, which can usually also handle it. The data flow is happening with queries.*

# Boundary conditions

*The part of the service that is responsible for the end user, will be started by our end user himself when he opens the app. The database behind our system needs to be started separately and will need to run at all time to ensure that the given input is actually stored. If this part fails our system will not break but it will prevent them from adding their food into our database thus making our system useless. We can also easily just deactivate our database if we feel to urge to do so.*