**Purpose**

The system design is documented in the System Design Document (SDD). It describes additional design goals set by the software architect, the subsystem decomposition (with UML class diagrams), hardware/software mapping (with UML deployment diagrams), data management, access control, control flow mechanisms, and boundary conditions. The SDD serves as the binding reference document when architecture-level decisions need to be revisited.

**Audience**

The audience for the SDD includes the system architect and the object designers as well as the project manager.

**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 |
| 1 | Moetaz Khelil | 10.06.2022 | First revision |
| 2 | Leon Körbs | 22.07.2022 | Updated SDD to reflect current system |
|  |  |  |  |
|  |  |  |  |

# Introduction

The system is built on Spring Boot and follows the REST architectural style. It is split in frontend and backend components that interact with our MongoDB data base. The restaurantService class is the main interaction point in our system as it facilitates the data transfer between frontend and data base.

## Overview

## Definitions, acronyms, and abbreviations

## References

# Design Goals

We wanted to make our website as user friendly as possible. This is why our UI is very straight forward and is focusing on the main functionality, which is finding and booking the desired restaurant as quick as possible. It is possible to search for restaurants by different categories but we didn’t implement more advanced search capabilities to keep the UI simple and easy to use.

Furthermore we also had to cut back on portability, as we wanted to use Vaadin as our front end and which is not optimized for mobile development. This has the advantage of having a very efficiently running and at the same time good looking system.

As we were very restrained on time, we had to prioritize the most important features of our application and therefore had to cut back on the calendar reminder functionality.

Our system is also easily maintained as we were very careful selecting a good internal structure with good naming and code style. A lot of our system is self-explaining.

# Subsystem decomposition

*Diagram

Description automatically generated*

# Hardware/software mapping

The systems mongo repository is mapped to a server deployed by MongoDB Inc. This has the advantage that it is always possible to access the required date to keep the system running.

The actual system uses the device it is started from as server to process requests and to interact with the data base.

Our map component is an API provided by google which helps with stability and accessibility.

# Persistent data management

We selected MongoDB as our storage as it is a popular and reliable solution. The objects are stored in JSON format. We communicate with this data base over our ItemRepository interface. RestaurantService is also taking responsibility of deserializing and serializing of the RestaurantItems objects. Using the GET and PUT functionality of restTemplate we can use objects and also save them to our data base.

# Access control and security

As security wasn’t one of the NFRs, we did not pay particular attention to threat retention in our system. Our system also does not have the possibility for users to create accounts which in turn limits the data we store of our customers.

# Global software control

We are using a centralized server that is using polling based request processing.

# Boundary conditions

Running the application is enough to start all components of our system. Our MongoDB is cloud based and always accessible.

If you want to stop the system, you just shut it down and everything closes automatically. There is no need to do anything else.

If errors occur, it is advisable to restart the whole application to resolve the issue.