Made by: Georgi Zhizgov

Design Document

Version 2.0

Table of Contents

[1. How is SOLID guaranteed? 2](#_Toc118550930)

[1.1 Single-responsibility (S) 2](#_Toc118550931)

[1.2 Open-close principle (O) 2](#_Toc118550932)

[1.3 Liskov substitution (L) 2](#_Toc118550933)

[1.4 Interface segregation (I) 2](#_Toc118550934)

[1.5 Dependency inversion (D) 3](#_Toc118550935)

[2 C4 Architecture 3](#_Toc118550936)

[2.1 System Context C1(System, users, dependencies) 3](#_Toc118550937)

[2.2 Containers and technology C2 4](#_Toc118550938)

[2.3 Components C3 6](#_Toc118550939)

# How is SOLID guaranteed?

What is SOLID? SOLID is an acronym for a group of five good principles (rules) in computer programming. SOLID allows programmers to write code that is easier to understand and change later on. Solid is often used with systems that use an object-oriented design.

## 1.1 Single-responsibility (S)

* Single-responsibility applies when there are classes depending on other ones so each class should remain responsible only for one purpose which has to do only changes to itself with only his information. In my case I use for each object a different service, repository and controller class plus each object works and is responsible only for its information.

## 1.2 Open-close principle (O)

* In [object-oriented programming](https://en.wikipedia.org/wiki/Object-oriented_programming), the open–closed principle states "software entities (classes, modules, functions, etc.) should be open for extension, but closed for modification" that is, such an entity can allow its behavior to be extended without modifying its [source code](https://en.wikipedia.org/wiki/Source_code). The open-close principle is mainly used via inheritance which is not applicable for my project.

## Liskov substitution (L)

* The Liskov substitution principle (LSP) is a particular definition of a subtyping relation, called**strong behavioral subtyping**, that was initially introduced by Barbara Liskov. When inheritance is not applied, so is not Liskov Substitution.

## Interface segregation (I)

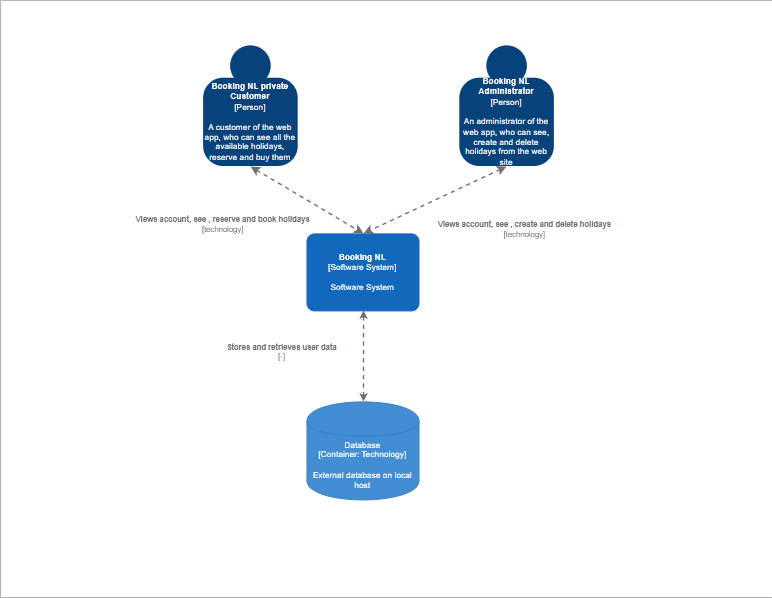
* In the field of [software engineering](https://en.wikipedia.org/wiki/Software_engineering), the interface segregation principle (ISP) states that no code should be forced to depend on [methods](https://en.wikipedia.org/wiki/Method_(computer_programming)) it does not use. ISP splits [interfaces](https://en.wikipedia.org/wiki/Interface_(computing)) that are very large into smaller and more specific ones so that clients will only have to know about the methods that are of interest to them. Such shrunken interfaces are also called role interfaces. Each service class implements separate service interface and each repository is represented by separate interfaces.

## 1.5 Dependency inversion (D)

* Dependency inversion talks about the coupling between the different classes or modules. It focuses on the approach where the higher classes are not dependent on the lower classes instead depend upon the abstraction of the lower classes.

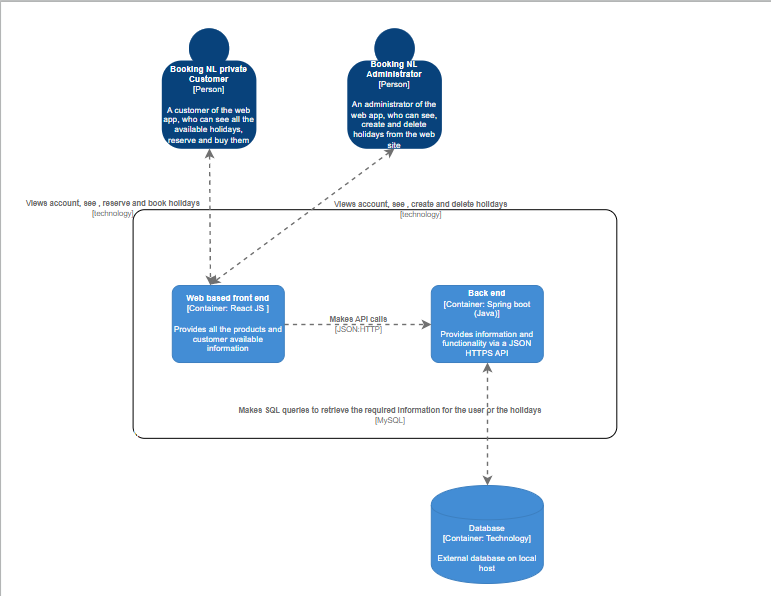
# C4 Architecture

## 2.1 System Context C1(System, users, dependencies)



The two types of users are a (regular) customer and an administrator. Both interact with the web app which corresponds with the database.

## 2.2 Containers and technology C2



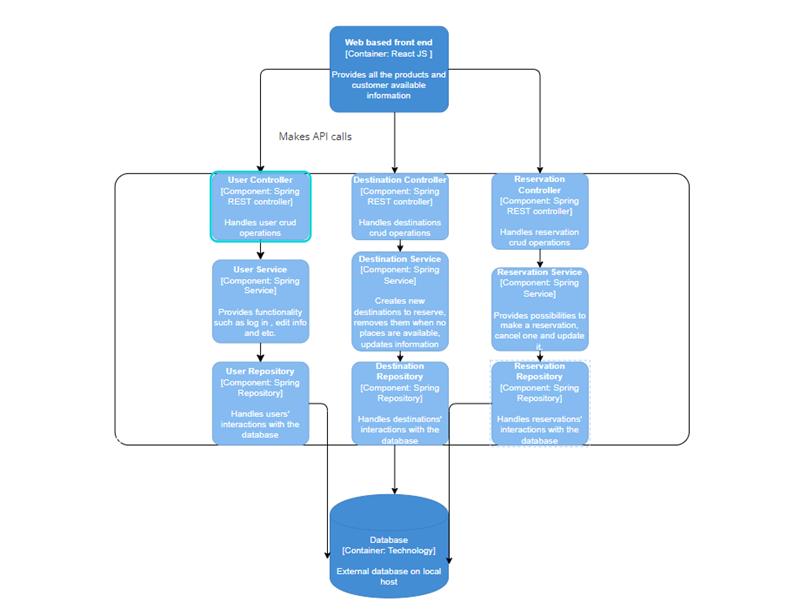
The web app, which is being used by both types of users, is separated into two parts, front end which is made for users benefit in order to communicate with the system easily and a back end which proceeds and retrieves the data from the database it is connected to.

What the client sees (the front end) is based on ReactJS because it comes with helpful developer toolset, it is fast, user friendly and responsive. Perfect for single page applications.

On the other hand, the back end (where the magic happens) is built using Spring and programmed in Java language. How does it contribute? Spring provides packages and libraries which require less code in order to implement faster and take less space (e.g. using Lombok you can get rid of implementing constructors or basic getters and setters just by using annotations). What is more, it is the framework we can use so we can create a RESTful API layer which corresponds with the commands sent from the front end.

To conclude the choice for Spring is that it also does not require data access layer implementation due to the fact that it provides JPA (Java persistence API), which has already implemented most of the CRUD functionalities and you save time from creating a separate layer with connection to the database, writing SQL queries and even database creation, because while we use JPA and the right annotations it creates a database for each entity with the columns we require.

## Components C3



When the front end sends API calls to the back end it goes through all the layers, which end in the repository which corresponds with the database.

Spring recognizes the different layers by their annotation, every business-logic layer requires a @Service annotation, every data access layer(DAL) requires a @Repository annotation.