**Introduction**

As an introduction to this document, I would first like to say that this is mainly a self-educational project, which goal is to learn and better understand Spring Boot, and its components.

This project is not deployed at any server and there is no intention of doing so on this version. However, there are plans for doing so on future iterations on this project.

**Project overview**

The project contains three major components Web(Thymeleaf), database(MongoDB), and server(Spring boot). This document aims to describe and clarify the different parts of the project with a clear focus on Spring Boot. The following text will describe the technical solution of the project class by class.

**Database**

The database in use for this project is a small NoSql database that is running on cloud.mongdb.com. The choice of database is based on the expected number of entries on it. Since the number of cheap beers could be considered as constant and rather few, I decided that a NoSql database should be a good option.

**Spring Boot**

This part contains six different classes.

* Beers
* BeerRepo
* BeerService
* AddBeerController
* BeerWebController
* DemoApplication

**Class Beers**

This class is the database representation in the project. It contains all the fields that exist in the collection on which it performs actions. There are some annotations in here that are worth a comment.

* @Getter
* @Setter
* @NoArgsConstructor

These annotations make it possible to remove getters, setters, and the empty constructor that otherwise would have been necessary. I am aware that these annotations could have been replaced with @Data.

**@**Document**(**collection = “beers”**)**

This annotation declares two things. First, it is MongoDB. And second, it’s working towards the collection “beers”.

**@**Repository

Defines that we are handling the database access layer.

**Class BeerRepo**

Is an interface that extends MongoRepository to enable CRUD operations. This class has one abstract class:

List<Beers> findByCountry(String country);

When using this method you will receive a list of beers that satisfies the String Country condition.

**@**Repository

Defines that we are handling the database access layer.

**Class BeerService**

Offers different services to classes that use it. It is injected with a BeerRepo and offers three different methods to use.

public Beers findBeerById(String id){

return beerRepo.findById(id).get();

}

public List<Beers> getAllBeersFromCountry(String country){

return beerRepo.findByCountry(country);

}

public List<Beers> getAllBeers(){

return beerRepo.findAll();

}

@Service

This indicates that we are in the service layer and that is where we do business logic. Nothing advanced here thou. Only different requests of data.

**Class BeerWebController**

This is a controller class and serves the webpage with data. It uses two different mappings.

@RequestMapping(“/index”) and @RequestMapping(“/oneBeer”).

The mapping using the index sends all available beers to the Thymeleaf page. Note that the return string must use the same name as the Html-page.

@GetMapping("/index")

public String index(Model model){

model.addAttribute("index", beerService.getAllBeers());

return "index";

}

The mapping connected to oneBeer needs input. In this case the id of the requested beer. If that is passed in it will return the beer model.

@GetMapping("/oneBeer")

public String getOneBeer(@RequestParam(value = "id", required = true) String id, Model model){

model.addAttribute("beers", beerService.findBeerById(id));

return "beers";

}

**Class AddBeerController**

This controller class is responsible för adding beer objects to the database. It takes input via @PostMapping(“/addBeers”) and then uses the beerRepo to add them to the database.

@PostMapping("/addBeers")

public List<Beers> addBeers(@RequestBody List<Beers> beerList){

return beerRepo.saveAll(beerList);

}

**Thymeleaf**

The template engine chosen for the project is Thymeleaf. The choice was rather simple since Spring Boot has built-in dependencies for it. To access data from the server simply use the prefix “th:” in front of most Html tags and it will work just nicely.