Assignment 3 - Clinic Management

Analysis and Design Document

Student: Pop Cristian Constantin

**Group: 30235**

Table of Contents

1. Requirements Analysis 3

1.1 Assignment Specification 3

1.2 Functional Requirements 3

1.3 Non-functional Requirements 3

2. Use-Case Mode 3

3. System Architectural Design 3

4. UML Sequence Diagrams 3

5. Class Design 3

6. Data Model 3

7. System Testing 3

8. Bibliography 3

# Requirements Analysis

## Assignment Specification

Use Java/C# API to design and implement a client-server application for managing the consultations of doctors in a clinic. The application has three types of users: the clinic secretary, the doctors and an administrator.

## Functional Requirements

The administrator can perform the following operations:

* CRUD operations on user accounts
* Edit the schedule for doctors

The secretary can perform the following operations:

* CRUD on patient data
* CRUD on patient consultation (schedule, re-schedule, cancel)
* Notify the doctor that a patient has arrived

The doctor can perform the following operations:

* See the consultations that are attributed to them
* Find consultations of a certain patient
* See observation added to a certain consultation
* Add observations to the consultations
* Receive notifications when patients arrive

## Non-functional Requirements

* The system should deny access to users who are not authenticated
* The system should allow the users to access only the resources and functions they are authorized to use (a administrator cannot edit the patients data for example)
* The system should validate all inputs to prevent abnormal data from being stored in the database
* The system should not allow any kind of overlaps when scheduling a consultation
* The system should inform its users if a certain action was performed successfully or if an error occurred

# Use-Case Model

## Use-case description

Use case: Add a consultation

Level: user-goal

Primary actor: secretary

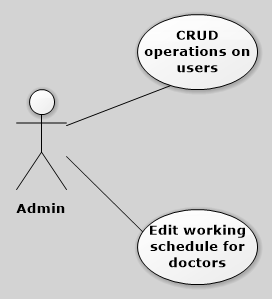
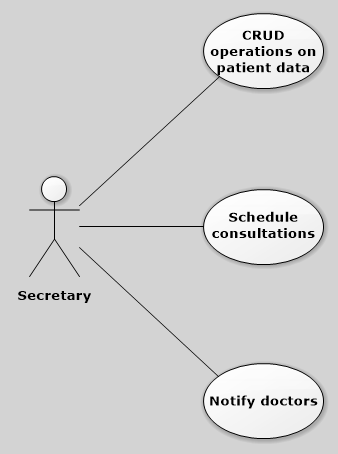
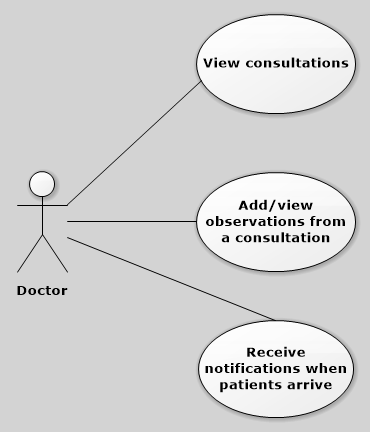
Main success scenario:

* The user goes to the application using a web browser
* The user goes to the login page and enters their credentials
* The front-end application sends the credentials to the server
* The server validates the credentials and returns a token (JWT)
* The user goes to the “Consultations” menu in the navbar and picks the “New consultation” options
* The consultation form is displayed
* The user selects a patient
* The user selects a doctor
* The user selects a day
* The application will display the working hours of the selected doctor for the given day
* If possible the application will also display other consultations scheduled for the same doctor on that day
* The user selects a hour for the consultation and a duration
* The user clicks the “Submit” button
* The server validates the received data and saves the consultation

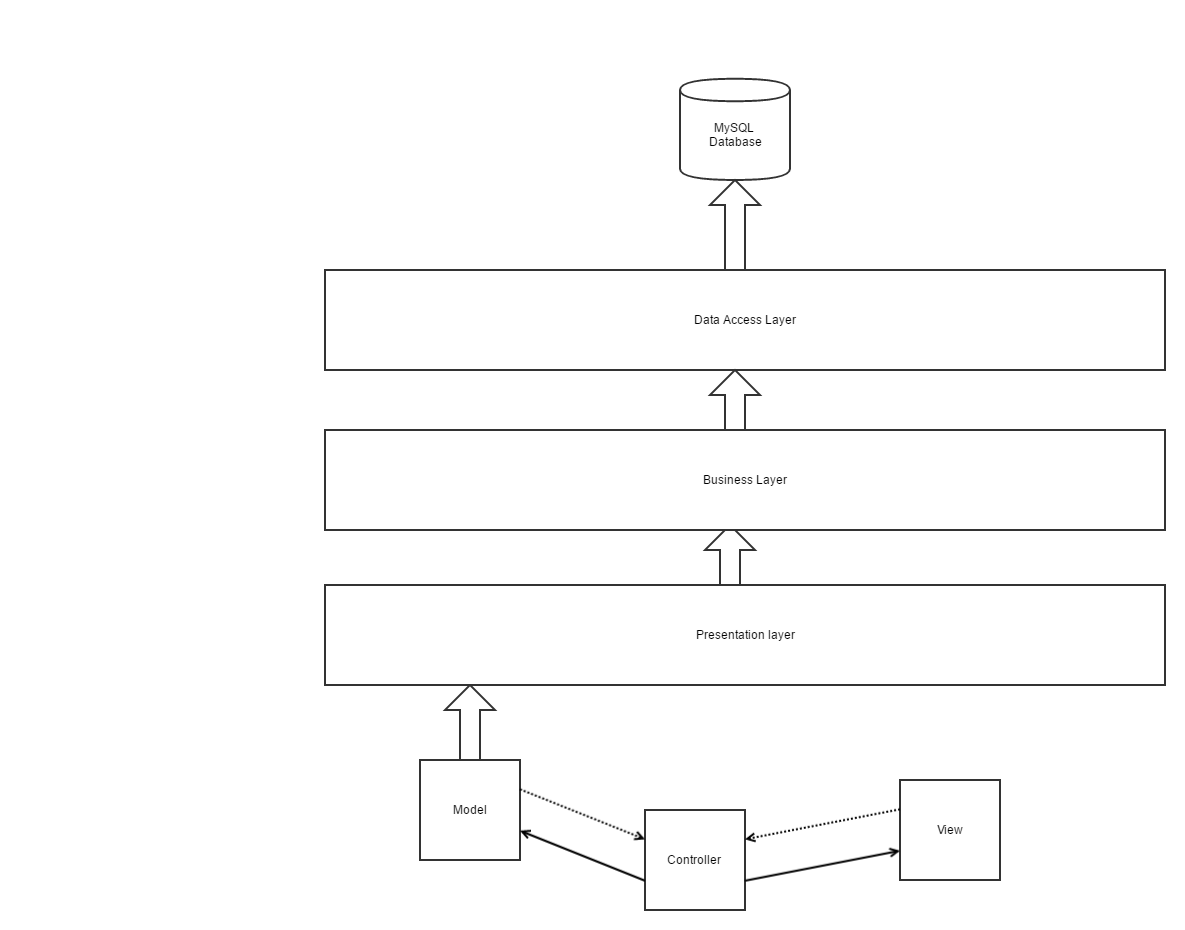
Extensions:

* The user is not a secretary – access to the consultation data will be denied
* The user does not select any doctor/patient/date – the application will not send data to the server until all the fields in the form are filled
* The consultation is outside the doctor’s schedule – display an error message. Don’t store the consultation into the database
* The consultation overlaps with another one – display an error message. Don’t store the consultation into the database

## Use-case diagrams

* administrator
* secretary
* doctor

# System Architectural Design

Conceptual architecture diagram

## Architectural Pattern Description

The two main architectural patterns used for designing this system were Layers and Model-View-Controller.

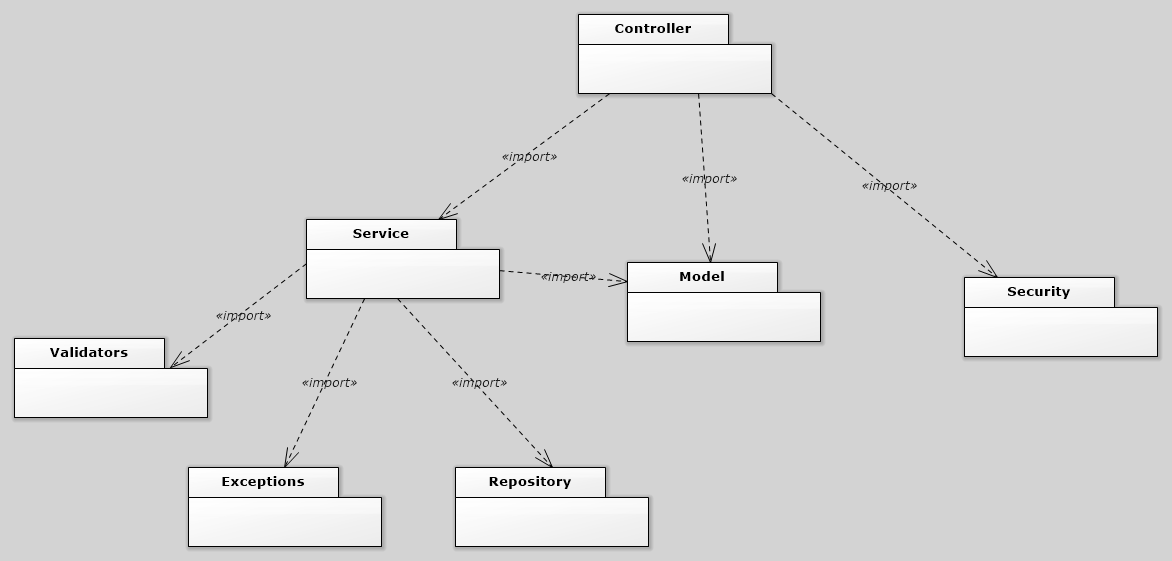
The Layers pattern was used to group the server application login into three categories:

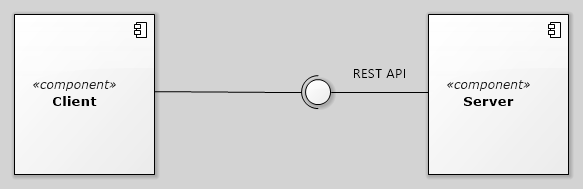
* The Data Access Layer was used to provide access to the data stored in a MySQL database. This was done by using the Java Persistence API (JPA) and Hibernate
* The Business Logic Layer contains the logic of the application that describes how the data should be manipulated and how transactions should be performed
* The Presentation Layer contains the logic used for communicating with the user. In this case the Presentation Layer contained REST Controllers used to exchange data with the frontend application

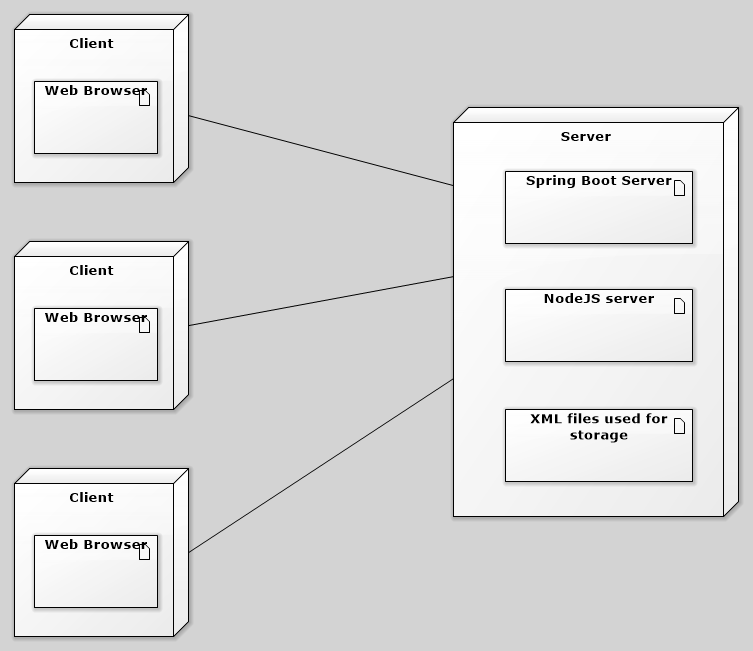
The Model-View-Controller pattern was used for building the frontend application. This pattern grouped the frontend into three categories:

* The Model was the part that held the data that came from the backend. In this case the models were the Angular services that exchanged data with the backend
* The View was the part of the application that interacted with the user. In this case the views were the HTML template used by the Angular components
* The Controller was the part of the application that link the models and the views. It should take data from the view and use it to update the model and also make the view update to reflect the changes from the model. In this case the controllers were the Angular components.

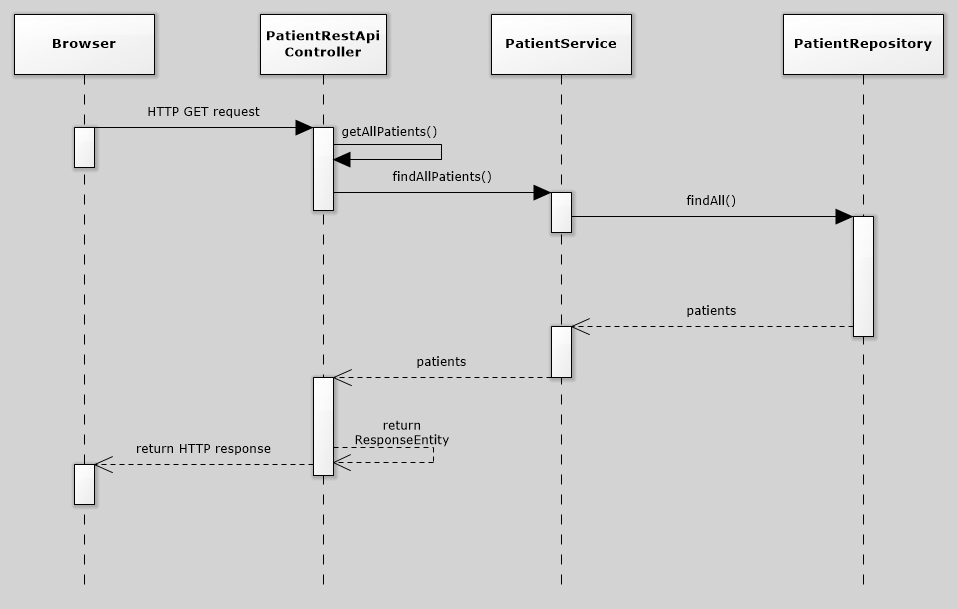
## Diagrams

Package diagram

Component diagram

Deployment diagram

# UML Sequence Diagrams

Sequence diagram for retrieving data for all the patients

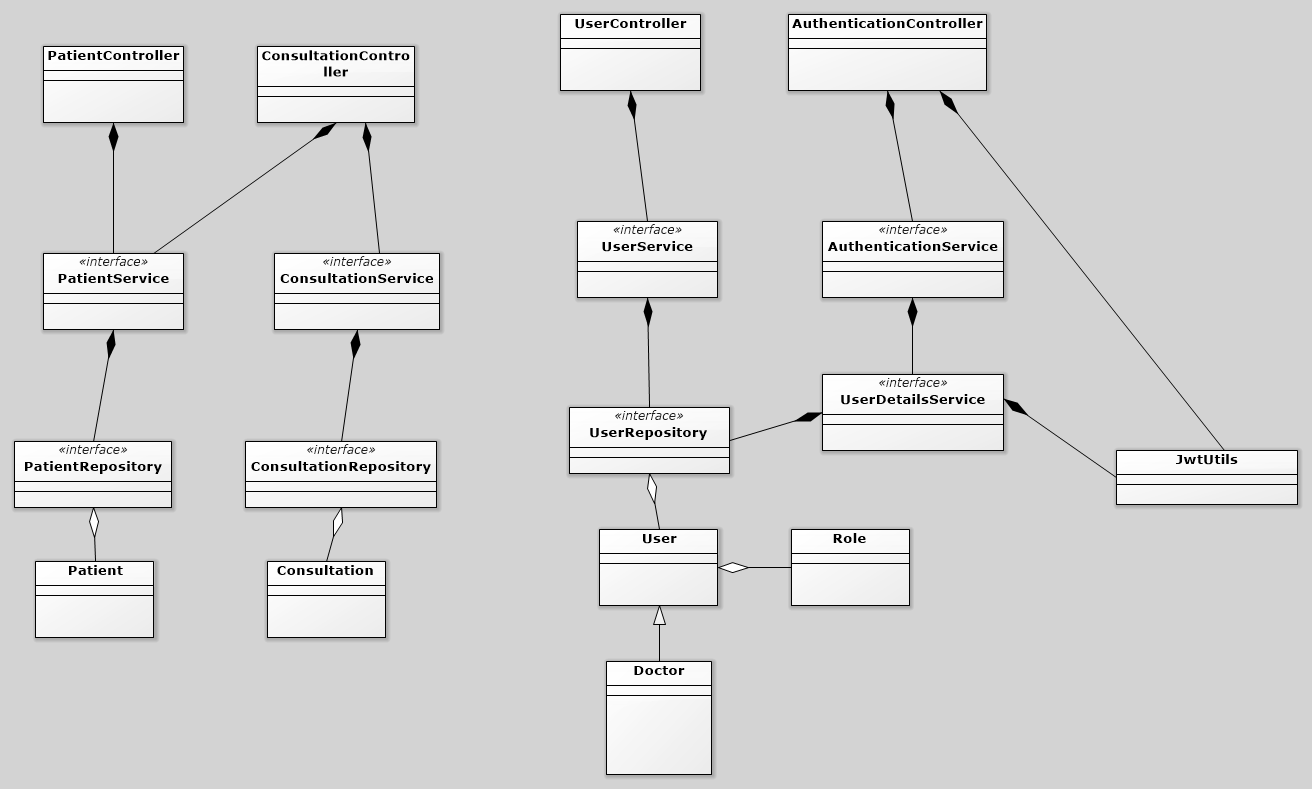
5. Class Design

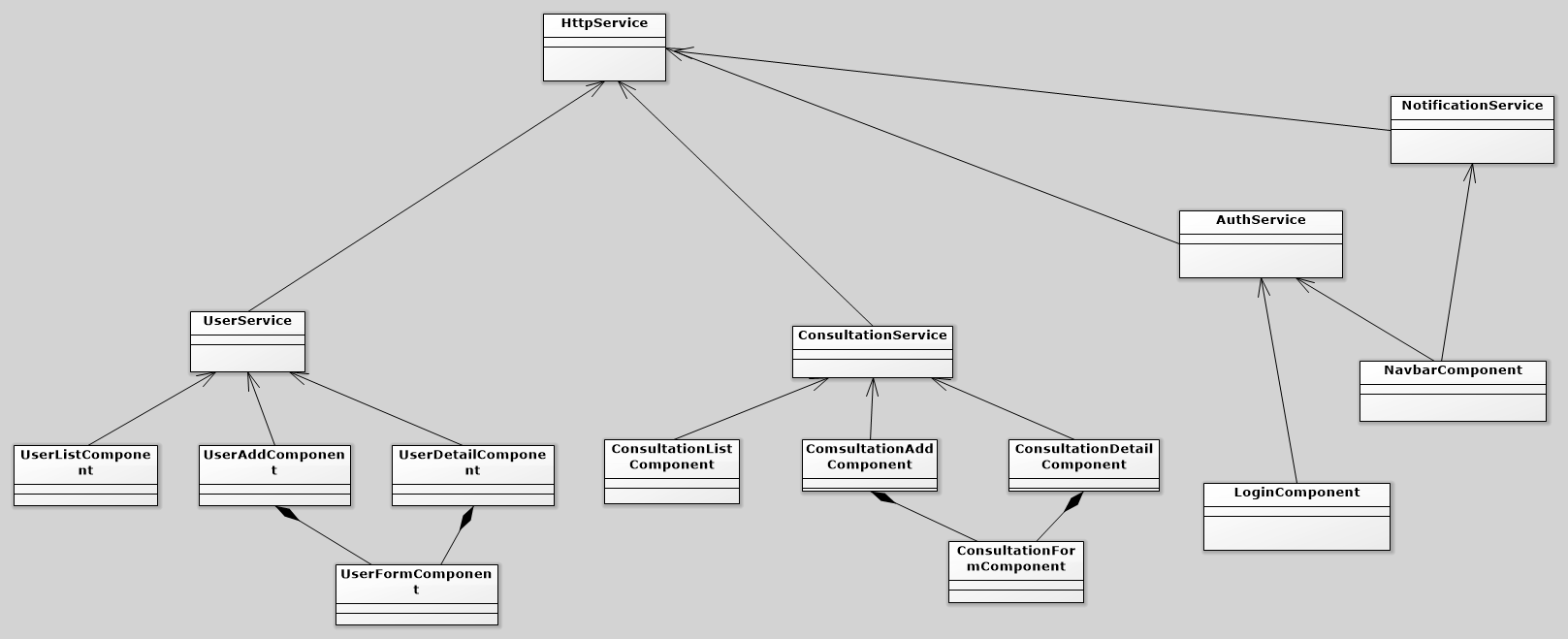
**5.1 Design Patterns Description**

The main design patterns used were:

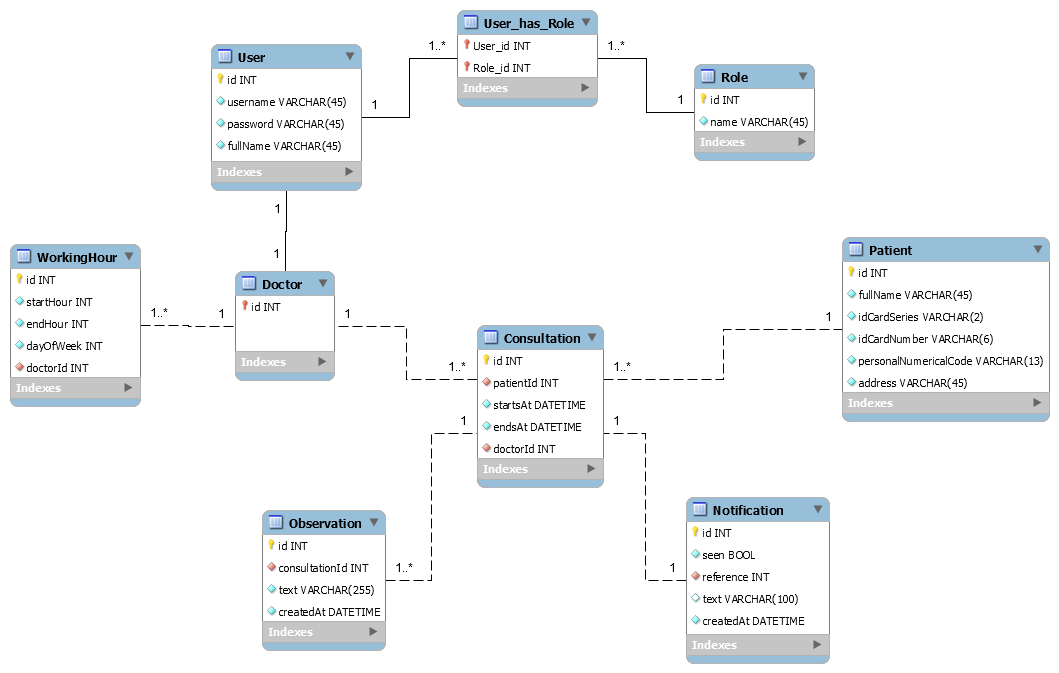
* Data mapper – the underlying pattern used by Hibernate. It is used to map a table (or more) from a database to a Java Object. The mapping should be guided by various annotations
* Builder – used to create objects without the need of using the constructor that has a large number of arguments. Mainly used inside unit tests.
* Observer – used in this application to send notifications to its users (notify users when patients arrive). This is implemented by sending AJAX requests to the server at a small interval of time to check if there is any new information (polling)

**5.2 UML Class Diagram**

Class diagram for server application (backend)

Class diagram for client application (frontend)

6. Data Model

**

7. System Testing

The important services from this project were tested by creating unit tests with Junit4 and SpringBootTest. Also, the tests were ran on a different Spring profile that used a H2 in-memory database instead of the regular MySQL database used in the default profile.

8. Bibliography

<https://cli.angular.io/>

<https://docs.jboss.org/hibernate/stable/annotations/reference/en/html_single/>

<https://auth0.com/blog/securing-spring-boot-with-jwts/>

<http://websystique.com/spring-boot/spring-boot-angularjs-spring-data-jpa-crud-app-example/>

<http://websystique.com/spring-boot/spring-boot-rest-api-example/>

<https://angular.io/docs/ts/latest/quickstart.html>

<https://getbootstrap.com/>

<https://docs.spring.io/spring-boot/docs/current/reference/html/boot-features-testing.html>