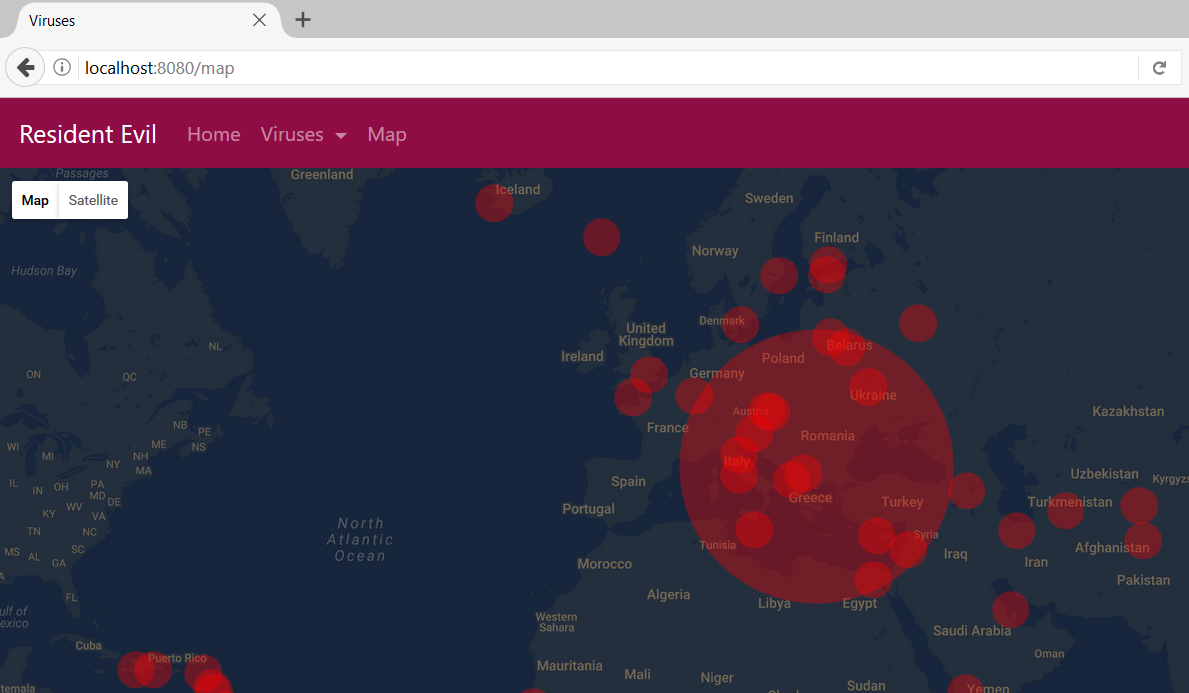
# Project: Resident Evil

Resident Evil is a system that registers virus spreads across the world. It is a significantly big project, and as such it will have several parts.



# Exercises: Filters

Problems for exercises and homework for the [“Java MVC Frameworks - Spring” course @ SoftUni](https://softuni.bg/trainings/1538/java-mvc-frameworks-spring-march-2017).

The Resident Evil Project is a pretty serious project. As such, it must keep track of all its clients. It would also be good if its design was a bit more intuitive. It would be incredibly good, if it had some authentication.

**NOTE**: As you will be implementing a few more pages in this exercise, there is something you should know. You know that the desing (like **color theme**, **page layout** etc.) does **not** matter much in the Resident Evil Project, however, you should make it atleast a bit **user friendly**.

In other words, design the web pages in whatever way you want, with whatever tools you prefer, but make **GOOD DESIGN**. 😉

## User Model

Implement a User model, which you will use as the main **Authentication model** in the Resident Evil Project. In future this User will be changed, but for now let it have the following properties:

* Username
* Password
* Email
* Role – (USER / ADMIN)

Of course, you will need to add the corresponding **Repositories** and **Services**, as the **User** will be **persisted** in the **Database**.

## Register Page

Implement a simple Register Page. There will be no example screenshot, as the design does not matter.   
It should hold the following input fields:

* Username
* Password
* Confirm Password
* Email

## Login Page

Implement a simple Login Page. There will be no example screenshot, as the design does not matter.   
It should hold the following input fields:

* Username
* Password

## The Users Controller

Implement a Controller for the Users, which will hold functionalities (Get / Post Mappings) for Login / Register / Logout.

## User Authentication Logic

Naturally the User’s Login and Logout logic will be controlled from the Session.

## User Roles Logic

By default, the User’s Role should be “USER”. Only though the **Database** **directly**, should it be set as “ADMIN”. This is for the current Exercise’s needs.

## Pre-Authenticate

Implement an annotation called @PreAuthenticate. The annotation should have the following fields:

* loggedIn – boolean (**default** – false)
* inRole – String (**default** – “USER”)

## Authentication Interceptor

Implement an Interceptor, which checks if the current action has a @PreAuthenticate annotation. If the annotation is **present**, you should check its fields.

The Interceptor should check If loggedIn is set to true, then you must check if there is a **logged** **in** User.

The interceptor should get the **currently logged** **in** User’s Role from the **Database**, and **check** if it **equals** the **given** one.

## \*Enhancing Authentication

Enhance the **Authentication logic**, by extending the functionality around the @PreAuthenticate annotation. **Configure** your business logic, so that the **annotation** can be placed on **controllers** **as well**. This **should not affect** its **ability** to be **placed on actions**. In fact, the **interceptor** should check **both controller** and **action** for the annotation.

The **action’s annotation** is with **higher priority** than the **controller’s annotation**.