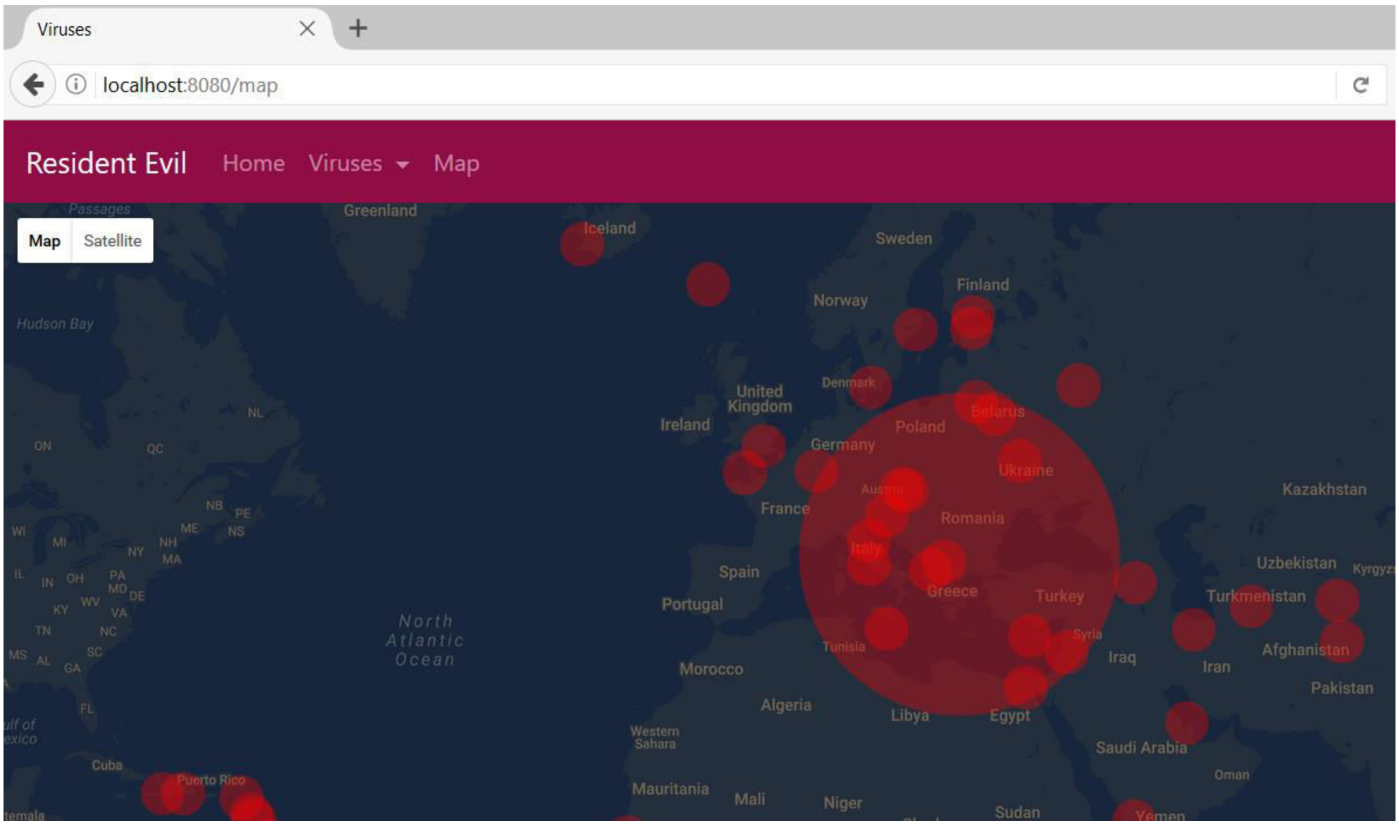
Project: Resident Evil

`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: Security

Problems for exercises and homework for the ["Java MVC Frameworks - Spring" course @ SoftUni](#).

The **`Resident Evil`** Project is a pretty serious project, as you've already heard. As such, it would need to be secured with authentication and authorization measures.

1. 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**.

2. Roles

There should be three main **roles** in your application:

- **ADMIN** – should be able to access **everything**.
    - Has all the rights of a **USER**.
    - Has all the rights of a **MODERATOR**.
- **USER** – should be able to access [**/viruses/show**], [**/**], [**/logout**].
    - This **Role** is set by **default**, upon **Register**.
- **MODERATOR** – should be able to access [**/viruses/add**], [**/viruses/edit**], [**/viruses/delete**].
    - Has all the rights of a **USER**.

Anonymous (not **logged-in**) clients should be able to access [**/login**], [**/register**], and [**/**].

Assign the roles from the **database** for now.

3. 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**

4. 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**

5. The Users Controller

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

6. Custom Authorization

Create a **simple page** (for example on route [**/unauthorized**] for **access denial**. If, for example, a **USER** tries to access one of the **MODERATOR** functionalities, you should **redirect** to **that page**.

7. Users Page

Create a page that **lists all** the **Users**. It should be **accessible only** for **Admins** and (only Admins of course) should see it as an element of the navigation bar.

8. UI Authorization

Edit the **home view** by using **Thymeleaf Security**. Change the **navigation bar**, adding the following authentication measures.

- If you are **anonymous** you should see:
    - [**Home**] section (**Guest**).

- [**Register**] section.
- [**Login**] section.

- If you are **logged in**, but with **USER** role, you should see:
  - The [**Home**] section (**User**).
  - The [**Viruses**] section (only with [**Show**] action).
  - The [**Logout**] section.
- If you are logged in, but with **MODERATOR** role, you should see:
  - The [**Home**] section (**User**).
  - The [**Viruses**] section (with both [**Show**] and [**Add**] actions).
  - The [**Logout**] section.
- If you are **Admin** you should see **all** sections, including the [**Users**] section.

9. Edit User Permissions

The page should **visualize basic data** about the users (for example in a **table**) like **`Username`** and **`Role`**. The **`Admins`** should be able to edit the **`Role`** of the **`Users`**, making them **`Users`** or **`Moderators`** or **`Admins`**.

You should **NOT be able** to edit your own **`Role`**.

10. * Secure the Admin Functionality

This task is designed to **secure** the **Admin functionality**, so that the **`Admins`** don't make **critical mistakes**. Such security **should exist** in **every application**. For example, an **`Admin`** should **not be able** to delete all other admins.

Implement **only 1** of the 2 choices stated below, as the **2nd choice** will **replace** the **functionality** from the **first**, and **visa versa**.

Choice A: Secure the Admins

In the **`Users`** section from the **previous task**, **secure** the **`Admins`**, by making their **`Role` unchangeable**.
This functionality should **trigger instantly**, when a **`User`** is made **`Admin`**.

Example:

- **`Admin Pesho`** makes **`User Gosho`** – **`Admin`**.
- **`Admin Pesho`** is no longer capable of editing **`Admin Gosho`**'s **`Role`**.
- **`Admin Gosho`** cannot edit **`Admin Pesho`**'s role either.

Choice B: Secure the Root

Create a 4th Role, called "**ROOT**".

- The **ROOT** role should have the same permissions as the **`Admin`** role.
- The **ROOT `User`**'s **`Role`** should **NOT** be modifiable by any **`Admins`**. **`Admins`** can modify everyone else.
- The **ROOT `User`** should be able to modify all other **`Users`**'s **`Roles`**, without exception (even the **`Admins`**).
- The **ROOT `User`** should be created by:
  - Being seeded with the **initial application start-up**
  - Being the **first-registered `User`**.