# Exercises: Web API

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

# Kayzer Watches

Kayzer Inc. is a company that publishes and sells the highest quality of hand watches. You have been given a pretty ugly and probably broken front-end, written by the previous developer, called Ivo, but your employers assure you it works just fine. You have been tasked with implementing a Web API, which can be combined with the Front-End.

## Data Layer

You will have to implement a single entity which will keep the data in the database.

#### Watch

Has an **id**, a **name**, an **image** (String, stored as **image URL**), a **price** (Decimal), a **description** (String) and **views** (long, **initial value** is **0**).

Implement the necessary **Repositories** and **Services**.

## API Endpoints

Let’s start with the **Endpoints** (**Routes**) you need to implement for the **API**.

### GET Route(“/watches/top”)

Returns the **Top 4 watches** in **terms** of **views**, of course, **ordered** by **views**, **descending**. The needed data is:

* **id**
* **name**
* **image**
* **price**

### GET Route(“/watches/all”)

**Returns** all **watches**. The needed data is:

* **id**
* **name**
* **price**

### GET Route(“/watches/details”) (with query (id={id})

**Returns** **details** **about** the **watch** with the **given** (as a **query parameter**) **id**. The needed data is:

* **id**
* **name**
* **image**
* **price**
* **description**
* **views**

**Each time** this **page** is **accessed**, you should **increment** the **views** of the **corresponding** watch **with 1**. This should

happen **before** the **response** is **returned**.

### POST Route(“/watches/add”)

You will receive a **POST** **request** with the **current data**:

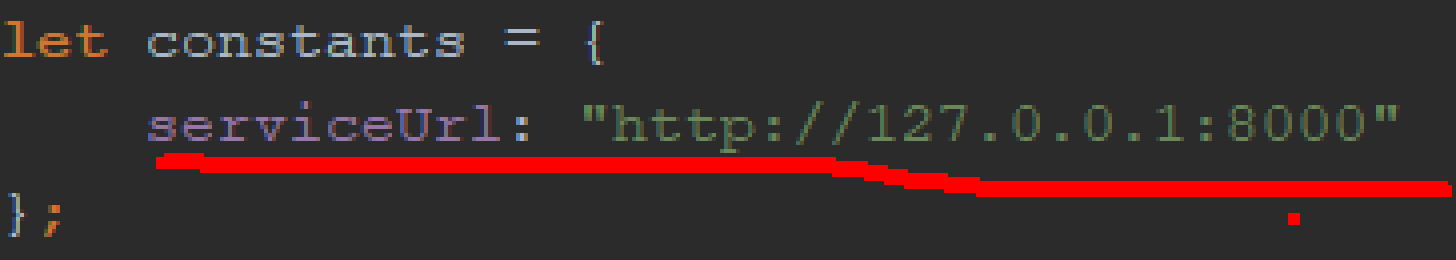
* **name**
* **image**
* **price**
* **description**

You should **create** a **new** **Watch** from the **data**, and **store** it in the **database**.

## Get Familiar with the Front-End

The Front-End is working, somehow… For the purposes of this exercise, it is enough. Get in touch with it, see if you can fix its broken logic, refactor it.

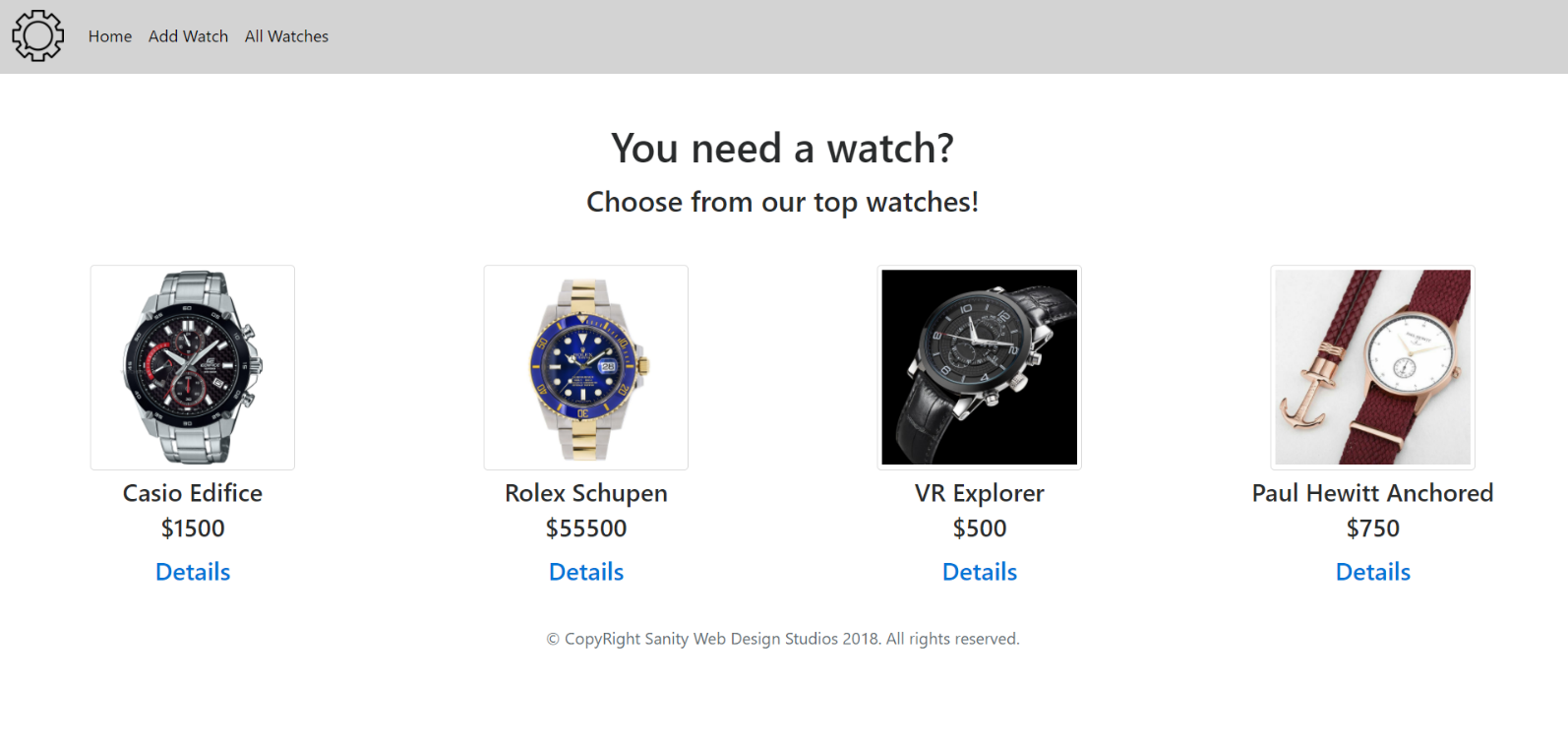
The least you will need to do, is most likely **change** the serviceUrl, which is located in a constants object (which is **not constant** at all, hint, hint … 2 rows above), in the app.js file.



## How Everything Should Look

Here’s a few screenshots about how everything should look, if the API is implemented successfully.

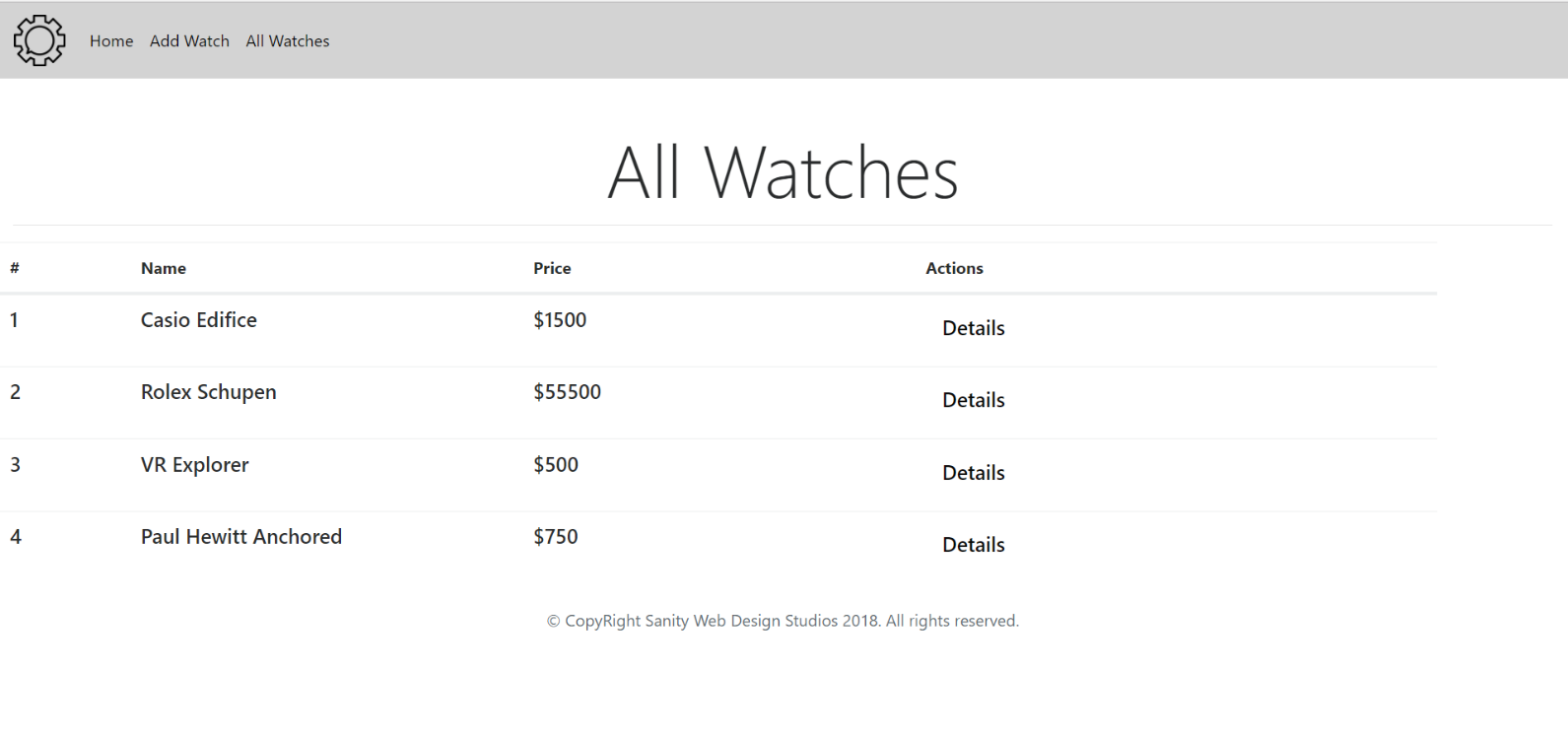
**Index** (“**/**”)



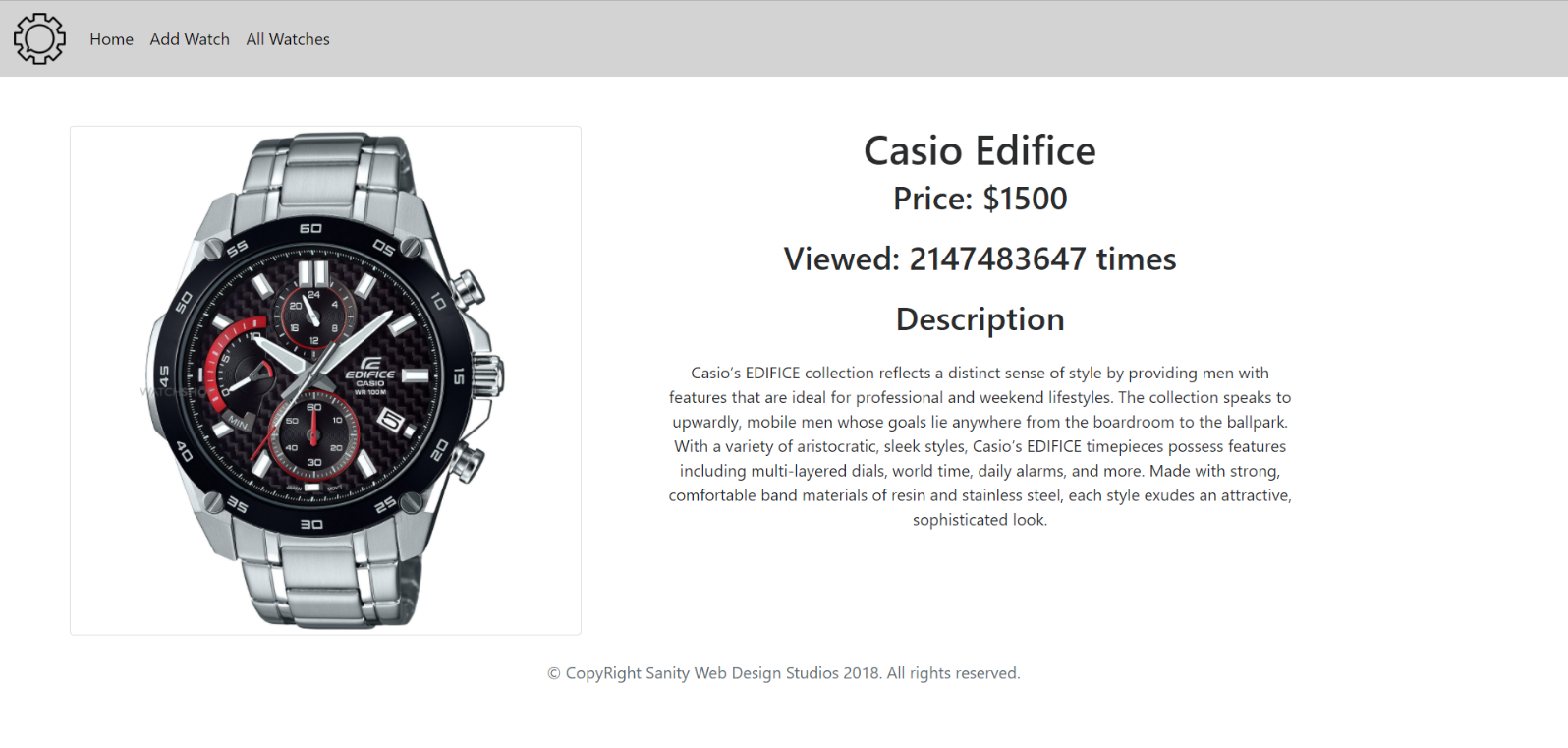
**Add Watch** (“**/watches/add**”)

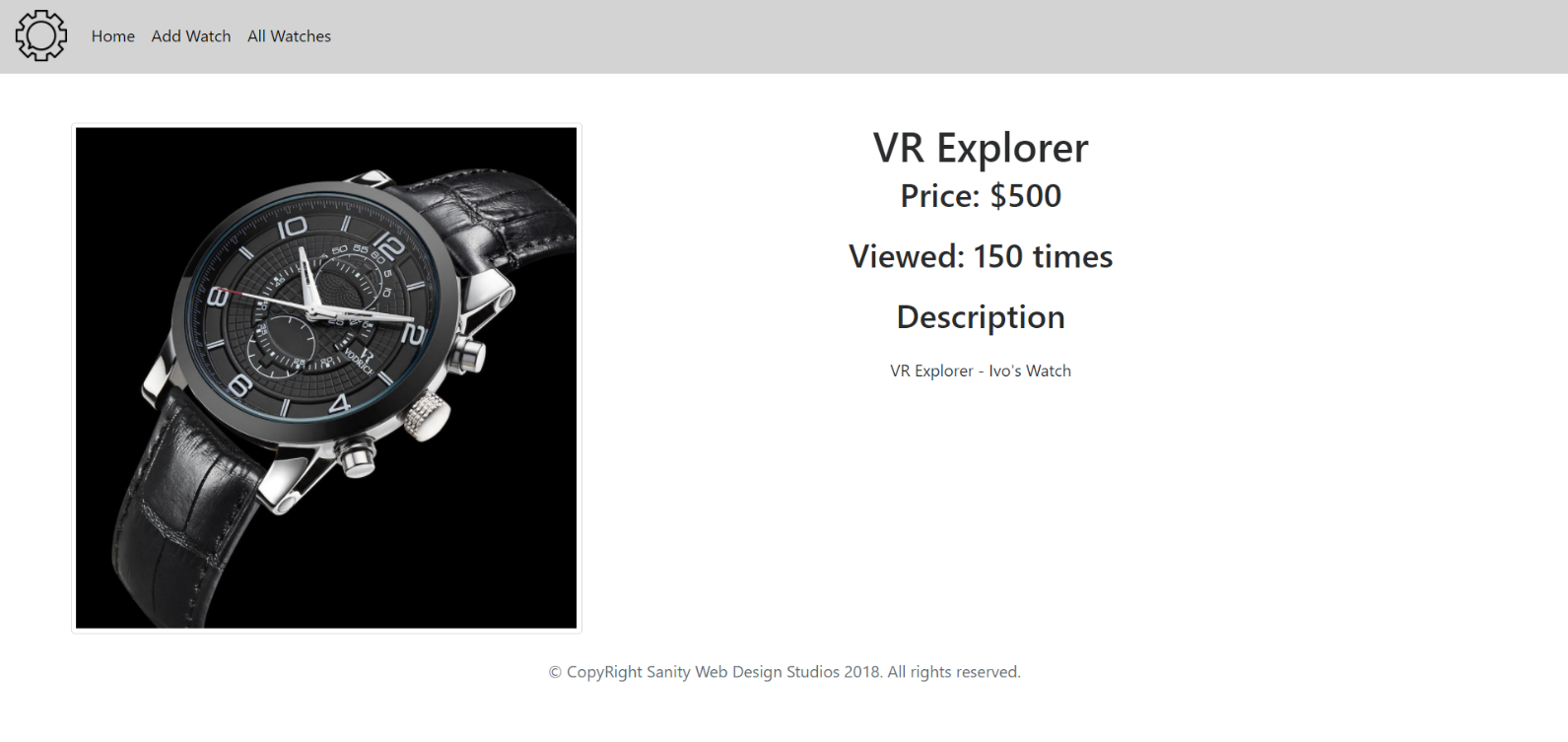


**All Watches** (“**/watches/all**”)



**Watch Details** (“**/watches/details?id={id}**”)





If you’ve implemented everything correctly, as it is said above, you shouldn’t have any problems with the front-end.

## Fix the JavaScript \*

You might have noticed that if you refresh the page, it does not reload the resources. That is due to the fact that the Front-End depends heavily on the **window hashchange** event. There is a workaround for this, try to fix it.

## Use Spring Data REST \*

You should be familiar with **Spring Data REST** by this moment. Well, try to implement the **Back-End** (the **API**) with **Spring Data REST**. That would remove certain elements from your application and make it quite lightweight.