# I. Теория

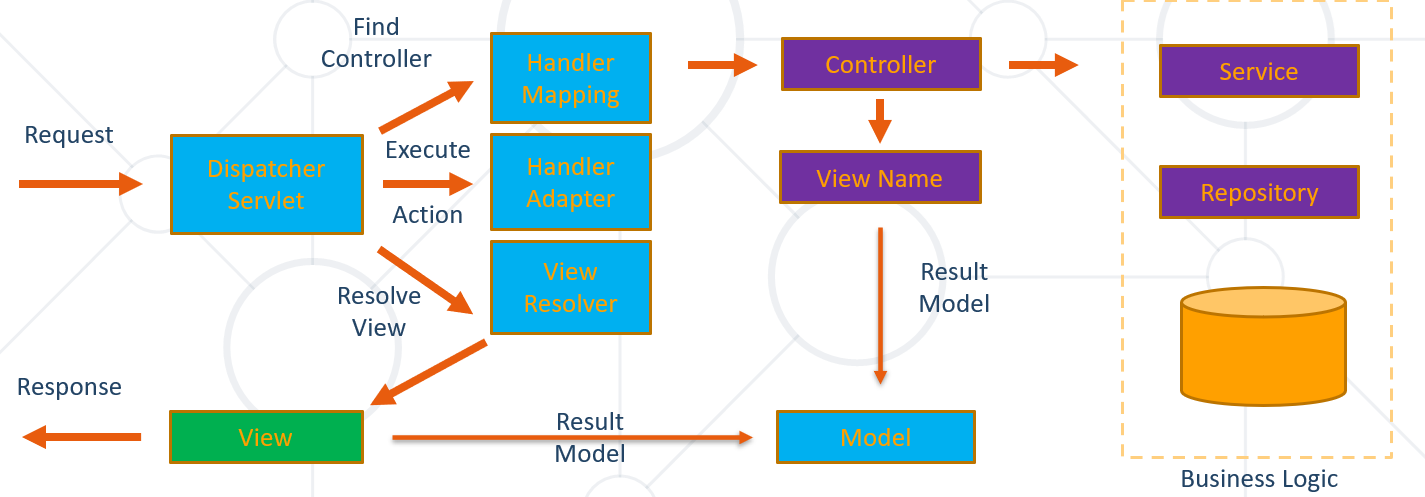
### 1. What is Spring MVC

**MVC е pattern или framework**, чиито **дизайн се базира на DispatchServlet** (Front Controller-a), който препраща **HTTP request-ите на входа на MVC до съответните controller-и**, които отговарят за тях.

**Схемата е следната :**

**HTTP Request-a идва от client-a** и веднага се посреща от **DispatcherServlet-a.** (servlet е Java програма, която работи в Servlet container).**//TODO : Servlet and servlet container?** За да намери правилния controller, който да обработи HTTP заявката, DispatchServlet-a използва помощта на **Handler Mapping-a.** След като намери правилния Controller, той изготвя един модел (например един голям HashMap със стойности вътре). За да го изработи този модел, Controller-ът използва Services, Repositories и базата данни, т.е. бизнес логиката на нашия app. На практика Controller-ът ще извика даден Service, който ще извика Repository и съответната база данни, което ще направи един модел и този модел ще се комбинира с едно View, което ще се изпрати на клиента.

//TO DO : Какво е HandlerMapping ?



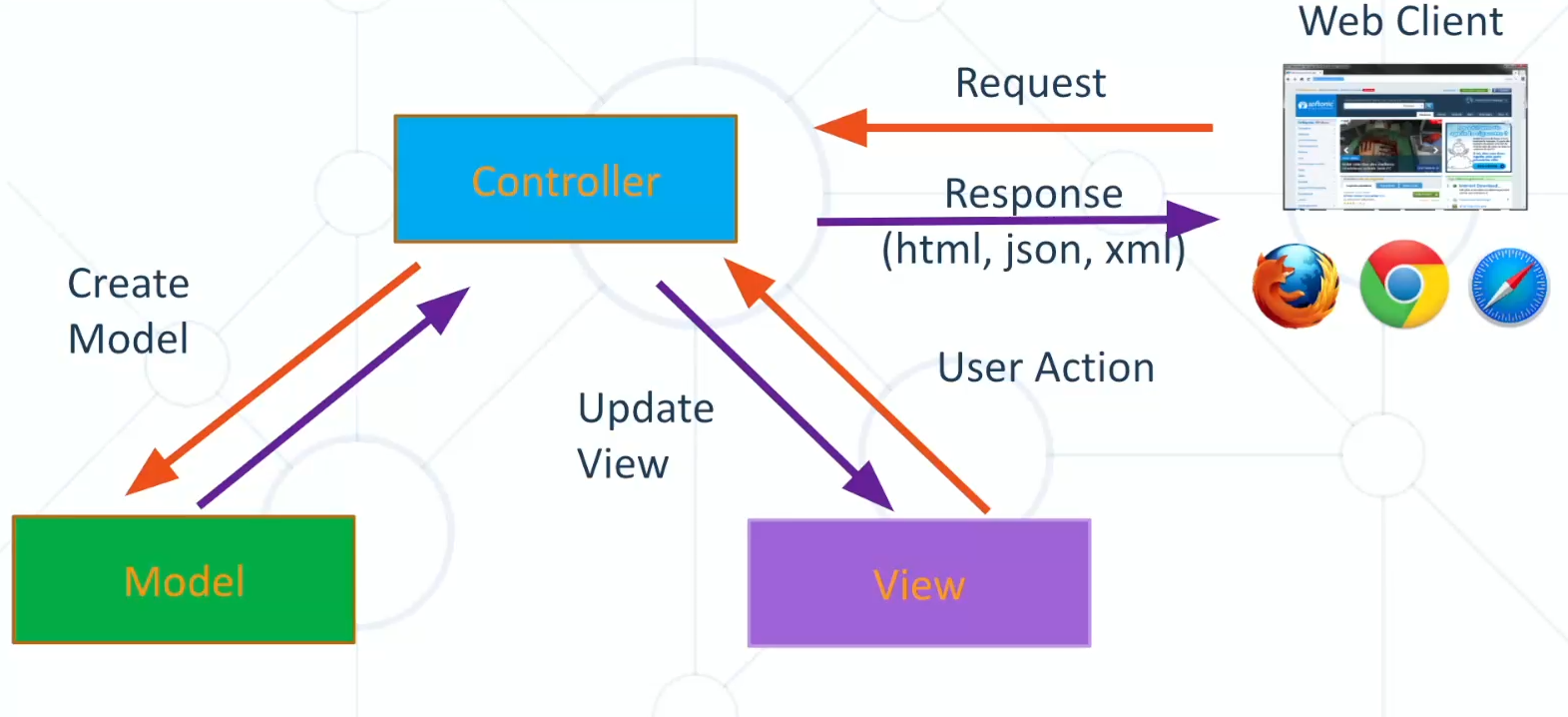
В малко по-суров вид, целият процес може да се представи и по следния начин :

1. Идва **HTTP request от Web client-a**, който се посреща от DispatcherServlet-a и съответния Controller.

2. Controller-ът изчислява кое View трябва да се извика.

3. Създава Model (BindingModel и ServiceModel) на база на View-то.

4. Update-ва View-то и го връща на Web client-a под формата на HTTP Response.



### 2. Inversion of Control

### 3. Spring MVC Controllers

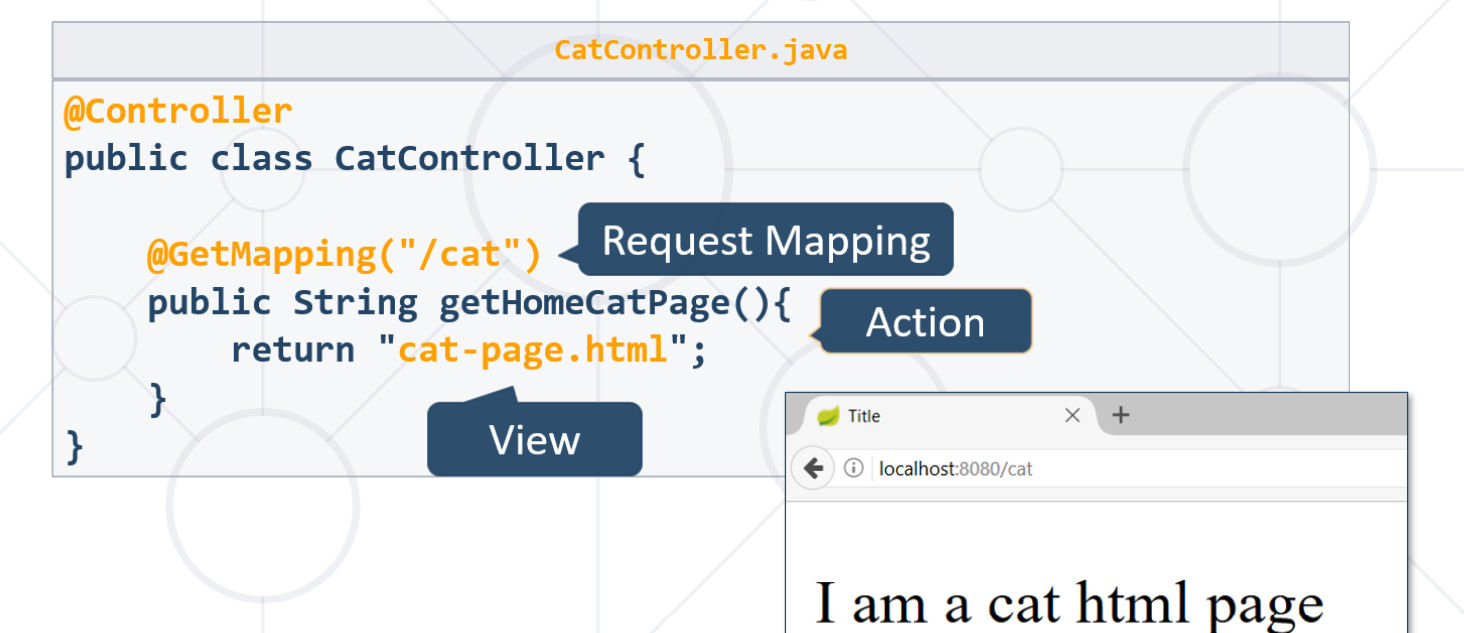
Controllers в Spring отговарят за визуализацията на Front end-a. Те се намират в “web” layer-a и се анотират с @Controller анотация. Анотациите GetMapping, PostMapping, PUT и Delete. се слагат в зависимост от заявката, която се очаква на route-a след тях. Например ако напишем в url bar-a на browsera “localhost:/edit/13”, това би ударило на нашия контролер в долния случай, който връща Get заявка .

//TODO:GetMapping – executes only Get requests.

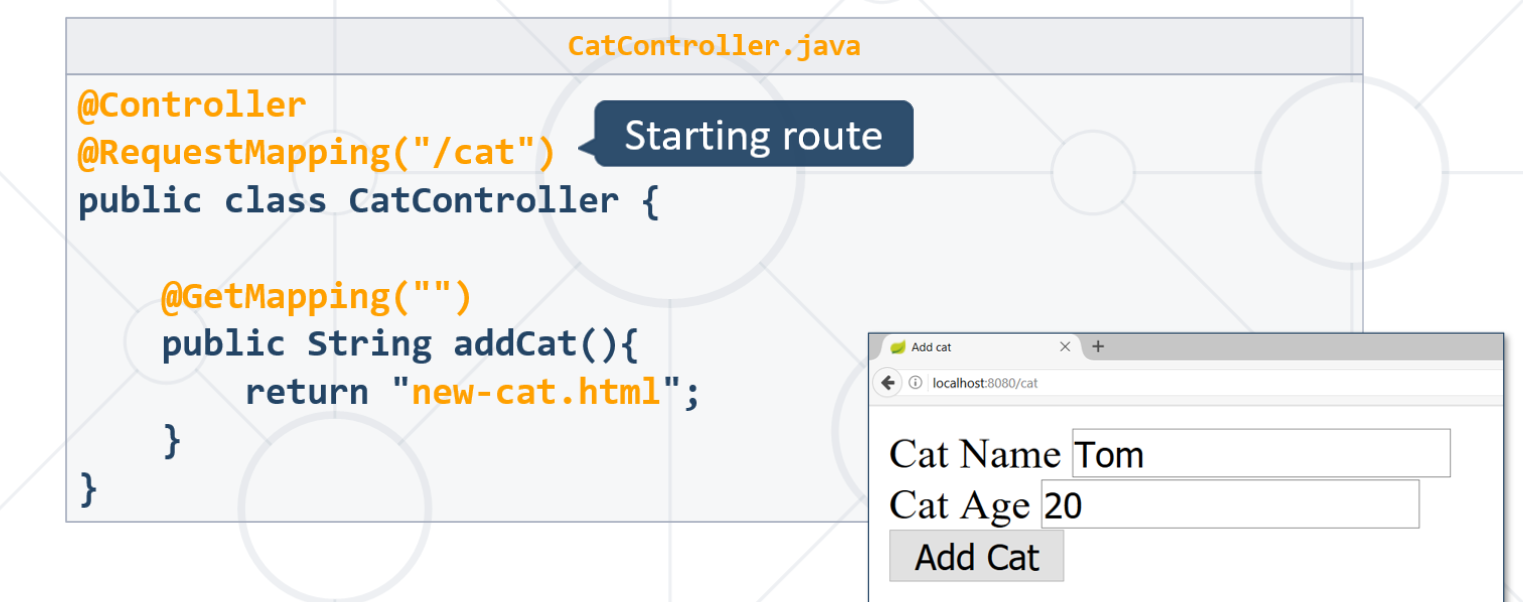
/ @RequestMapping – accepts all types of request methods (get, post, put, delete, head, patch)

@ResponseBody

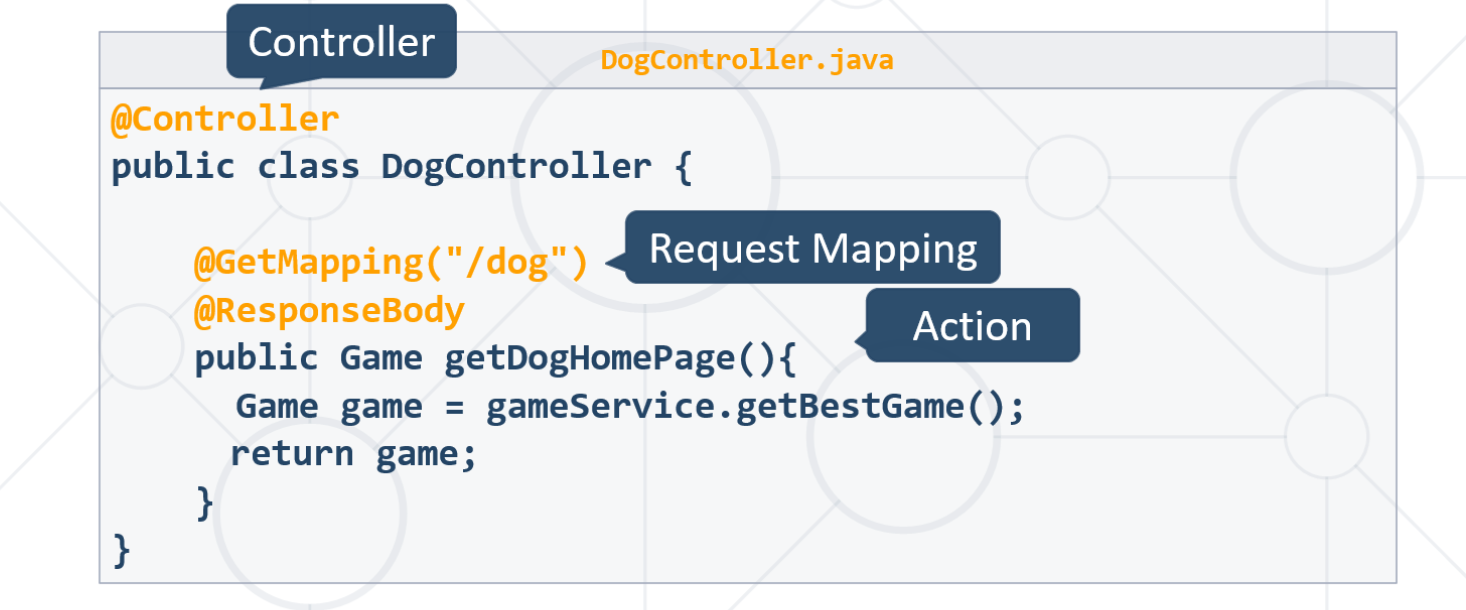
#### Get request



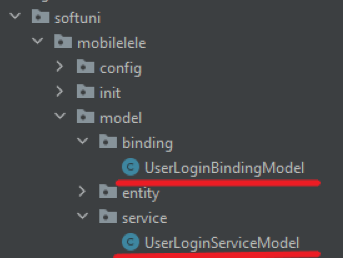
#### Post Request



#### @ResponseBody

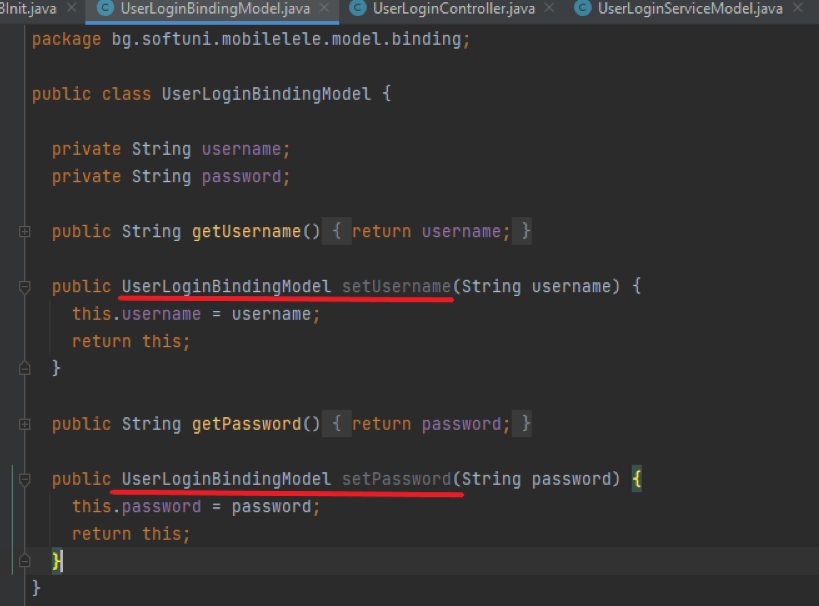


BindingModel and ServiceModel са DTO, които се използват за трансфер на информация между web и service layers.



BindingModel

BindingModel DTO се използва в Controller-а в web layer-a.

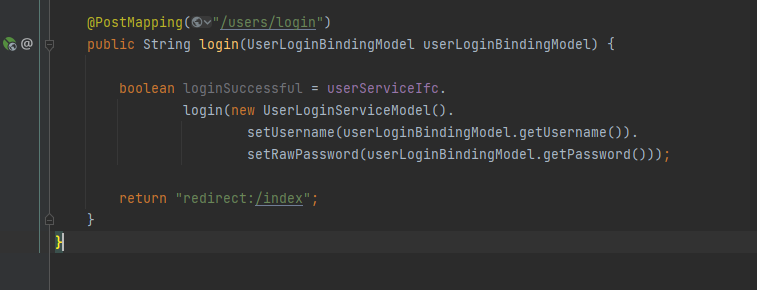


Полетата му (в сл. username и password) се определят от полетата, които се намират в html формата :

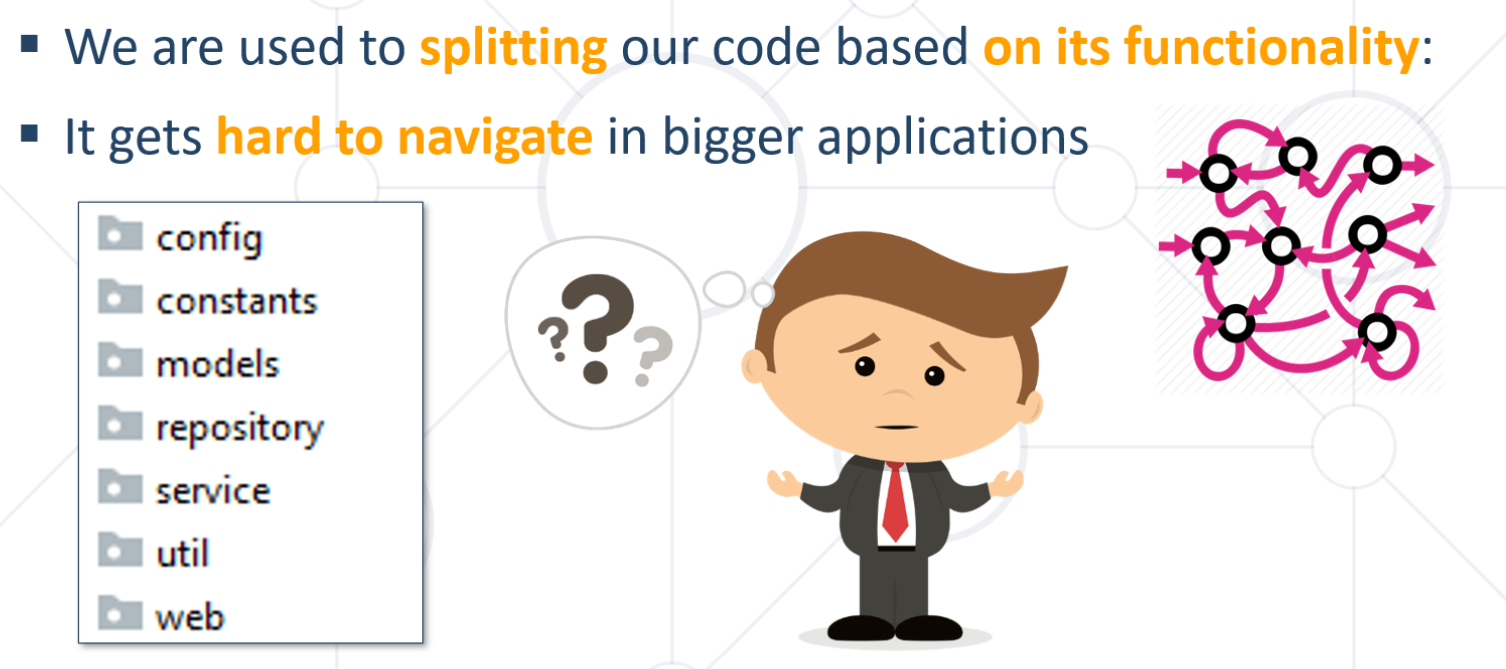


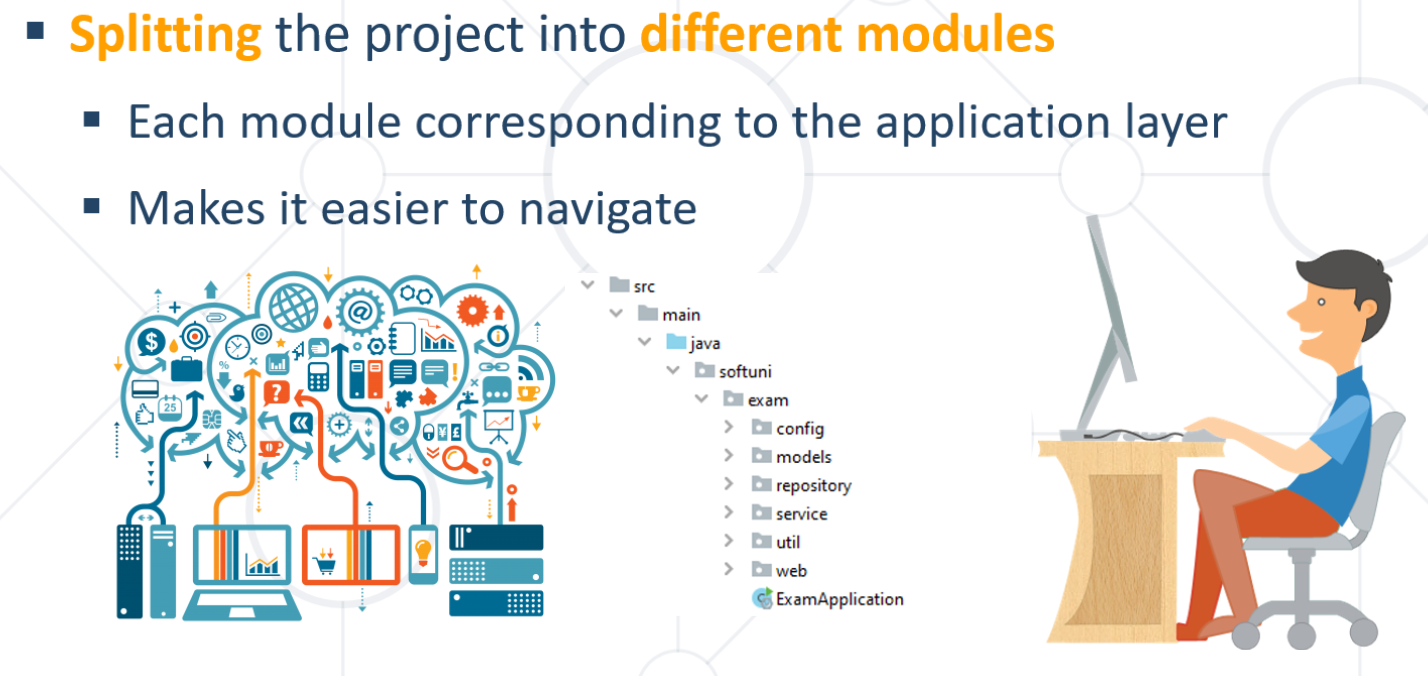
В примера ще напишем такъв UserBinding Model, който да взима Username и Password, които ще дойдат от Web formata на адрес localhost/users/login.

В web controller-a, Binding моделът се подава като параметър на PostMapping заявката. Данните, които ще вземем чрез PostMapping на адрес users/login, ще бъдат присвоени на UserLoginServiceModel, като първо ги get-нем от BindingModel-а, и после ги set-нем на нова инстанция на ServiceLoginModel с методите setUsername() и setRawPassword(). По този начин успяваме с две DTO-та, да вземем данните от web layera (userLoginBindingModel) и да ги присвоим в service layera чрез userLoginServiceModel. Ако успешно сме взели денните, boolean методът в userServiceFc ще върне true стойност. В случая, въпреки че userLoginBIndingModel и userLoginServiceModel са на практика еднакви, трябва винаги да правим 2 DTO-та, с които да разграничаваме входните данни от web и service layer-ите.



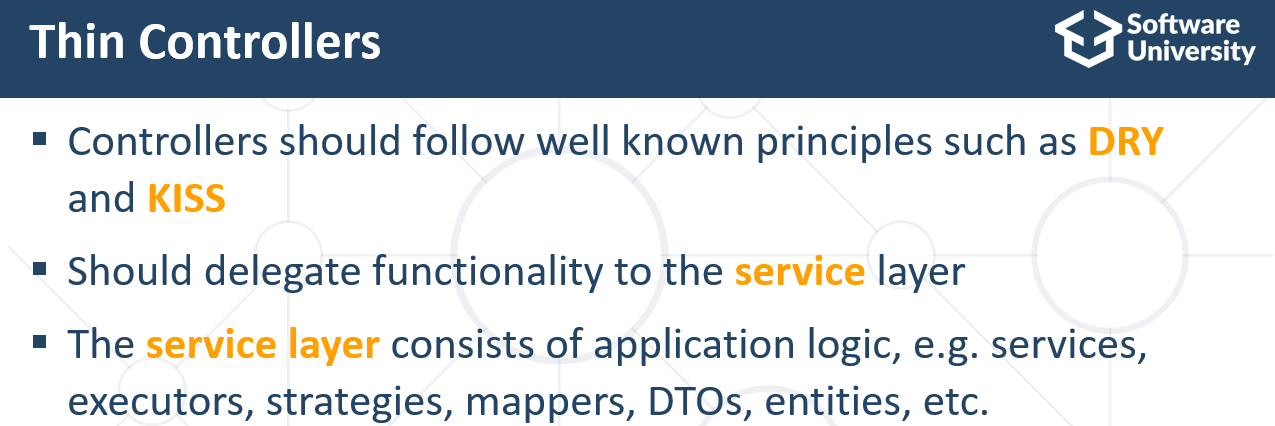
### 4. Layers – dividing code





### 5. Thin Controllers

Web controllers трябва да бъдат прости, да не се повтарят и да делегират функционалности на Service layer. Делегирането се извършва чрез Binding и Service модели.



# II. Lab: Spring Introduction MVC

## MobiLeLeLe web application

MobiLeLeLe is an application in which you register cars, with several properties.

You will have to create a simple application which has several pages and some object entities.

### Data

This is the data layer of the application. There are some data object for you to implement.

#### Brand

Create a Brand class, which holds the following properties:

* id – a **uuid or number**.
* name – a **name of brand**.
* **created** – a **date and time**.
* modified – a **date and time**.

#### Model

Create a Model class, which holds the following properties:

* id – **uuid or number**.
* name – a **model name**.
* category – an enumeration (Car, Buss, Truck, Motorcycle)
* imageUrl – the **url of image** with size between 8 and 512 characters.
* startYear – a **number**.
* endYear – a **number**.
* **created** – a **date and time**.
* modified – a **date and time**.
* brand – a **model** **brand**.

#### Offer

Create a Model class, which holds the following properties:

* id – **uuid or number**.
* description – some **text**.
* engine – **enumerated** value (GASOLINE, DIESEL, ELECTRIC, HYBRID).
* imageUrl – the **url of image**.
* mileage – a **number**.
* price – the **price of the offer**.
* transmission – **enumerated** value (MANUAL, AUTOMATIC).
* year – the **year** of offered car.
* **created** – a **date and time**.
* modified – a **date and time**.
* model – the **model of a car**.
* seller – a **user that sells the car.**

#### User

Create a User class, which holds the following properties:

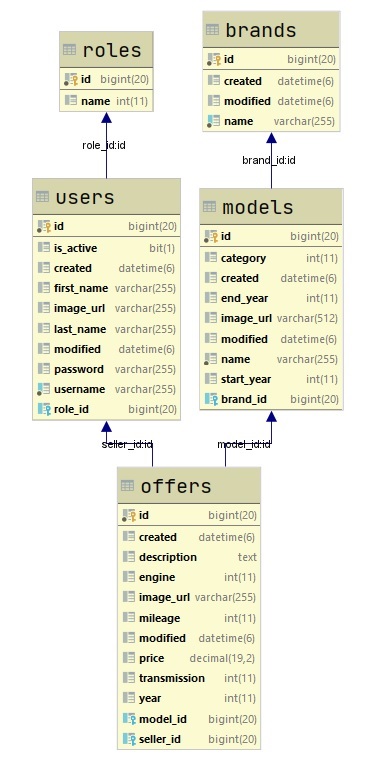
* id – **uuid or number**.
* **username** – username of the **user**.
* **password** – password of the **user**.
* **firstName** – first name of the **user**.
* **lastName** – last name of the **user**.
* isActive – **true OR false**.
* role – **user's role (User or Admin)**.
* imageUrl – a url of user's picture.
* **created** – a **date and time**.
* modified – a **date and time**.

#### UserRole

Create a UserRole class, which holds the following properties:

* id – **uuid or number**.
* **role** – **enumerated** value.

This is an example of ER Diagram



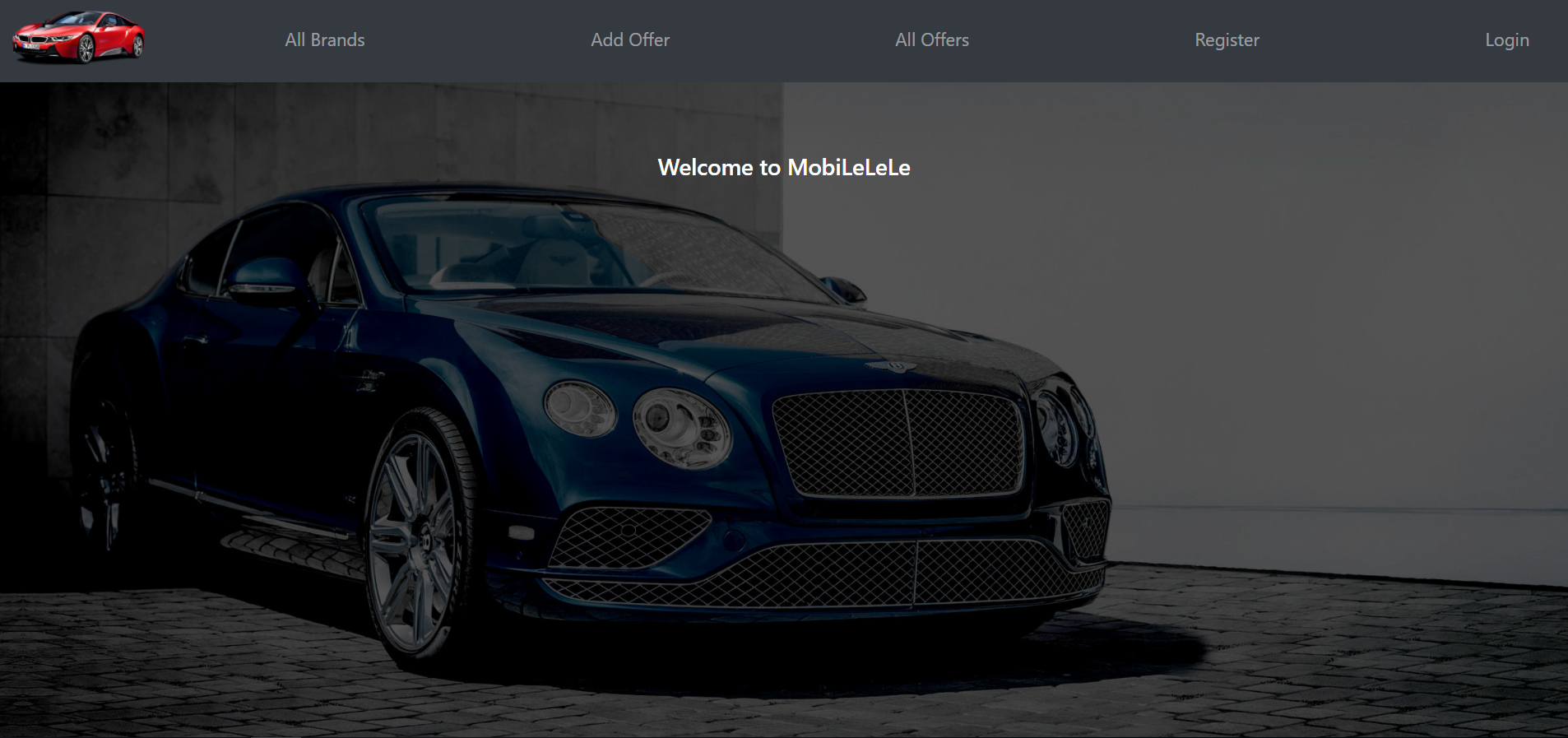
### Populate DB

Create Data Initializer class, that populate the DB with information about cars when application starts for the first time.

### Home/index - route ("/")

It should support only a **GET** request.

It should return the following HTML page, upon a **GET** request.

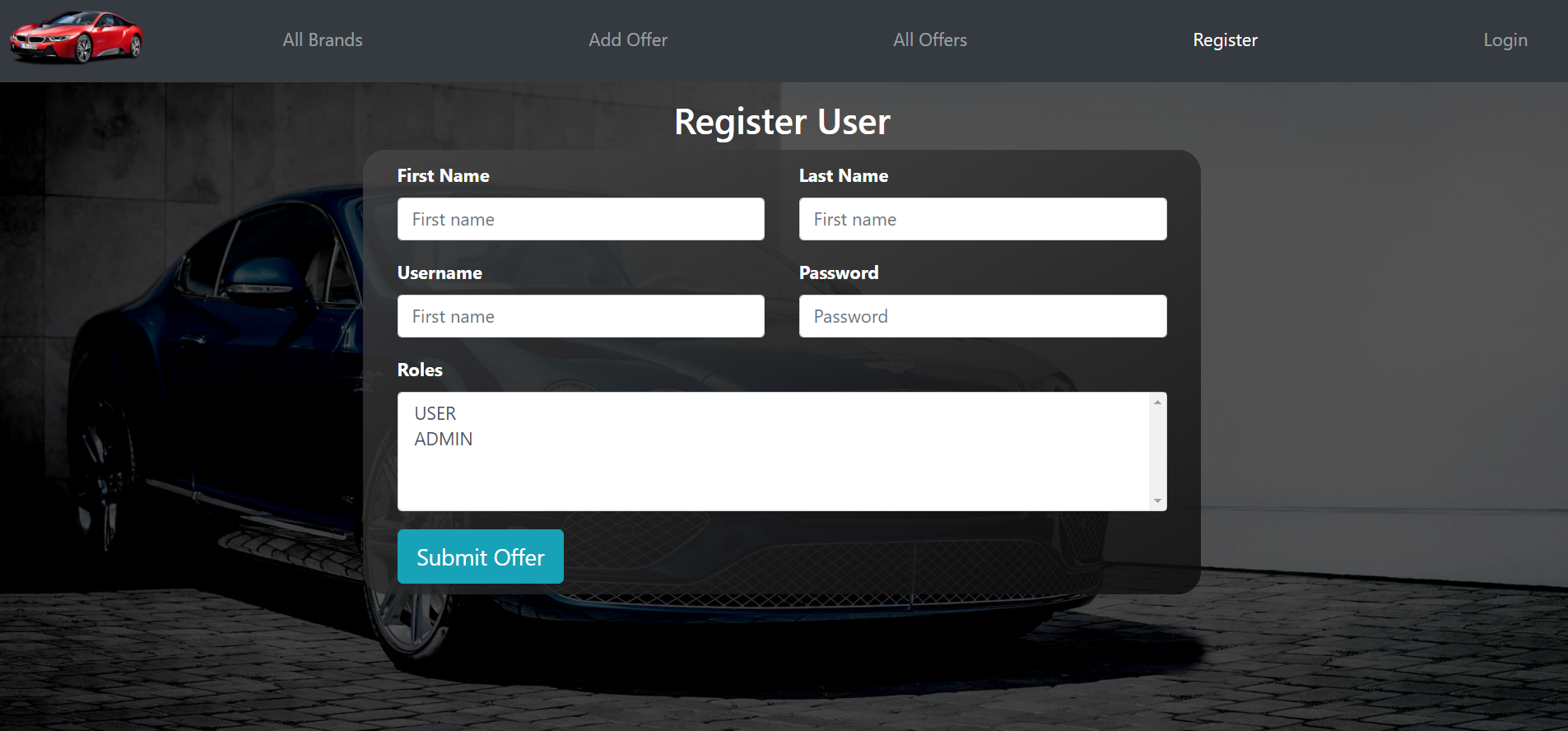


### Register User - route ("/users/register").

It should support only a **GET & POST** request.

It should return the following HTML page, upon a **GET** request.

First we need to add some users in our DB.



**Hint section:**

* **Because you will learn Thymeleaf in details on the next next lecture, we'll give you hints on how to implement some things**
* **Do not forget to add Thymeleaf in you pom.xml file**
* **Do not forget to add Thymeleaf name spaces:**



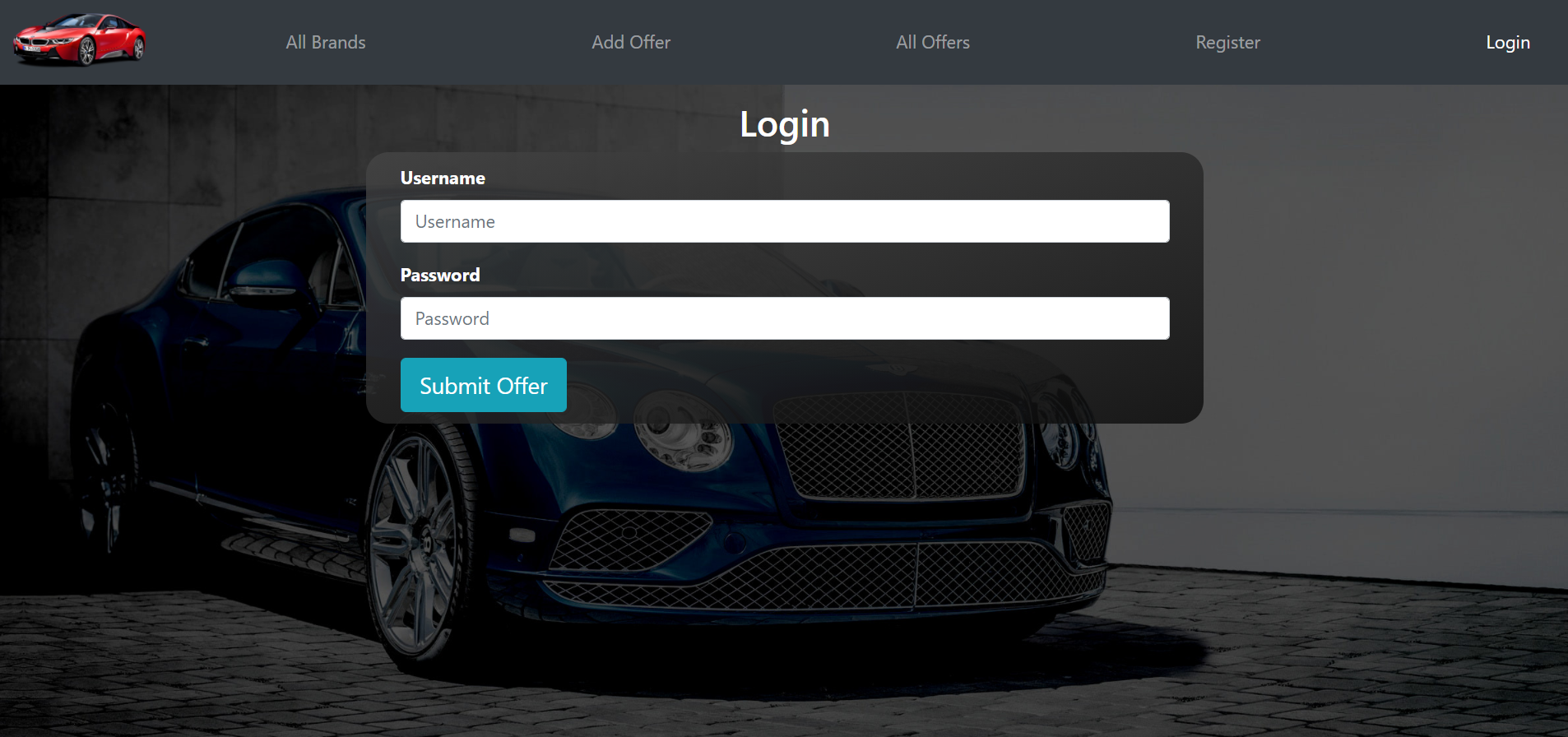
* Also you need to add in are the html form action and method (remember last lecture)**:**



### Login - route ("/users/login")

It should support only a **GET & POST** request.

It should return the following HTML page, upon a **GET** request.



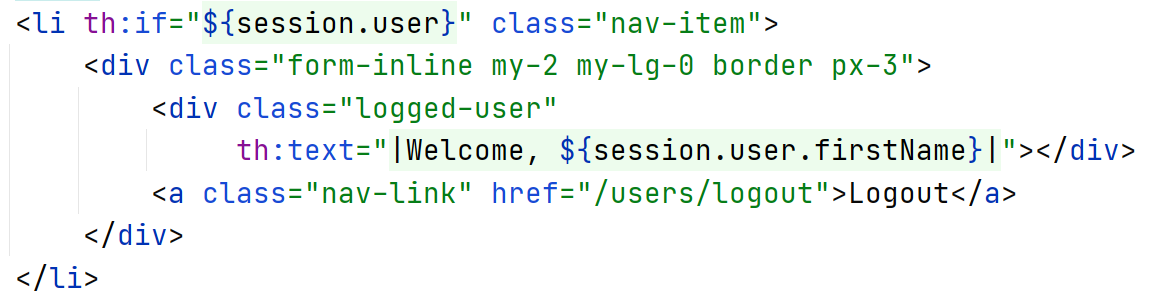
### Navigation for login user

When a user logs in, in the application, he cannot see the Register and Login buttons, but Logout.

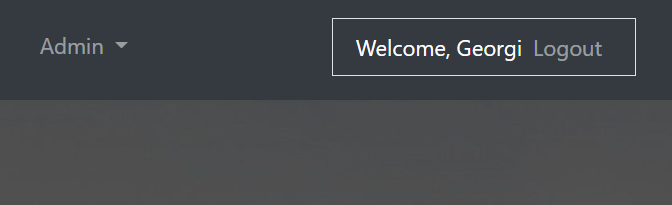
Also, if he has an Admin role, he can see the Admin dropdown.

Because you will learn Thymeleaf in the next lesson, we will give you a little hint how to do this point.

Hint Section:



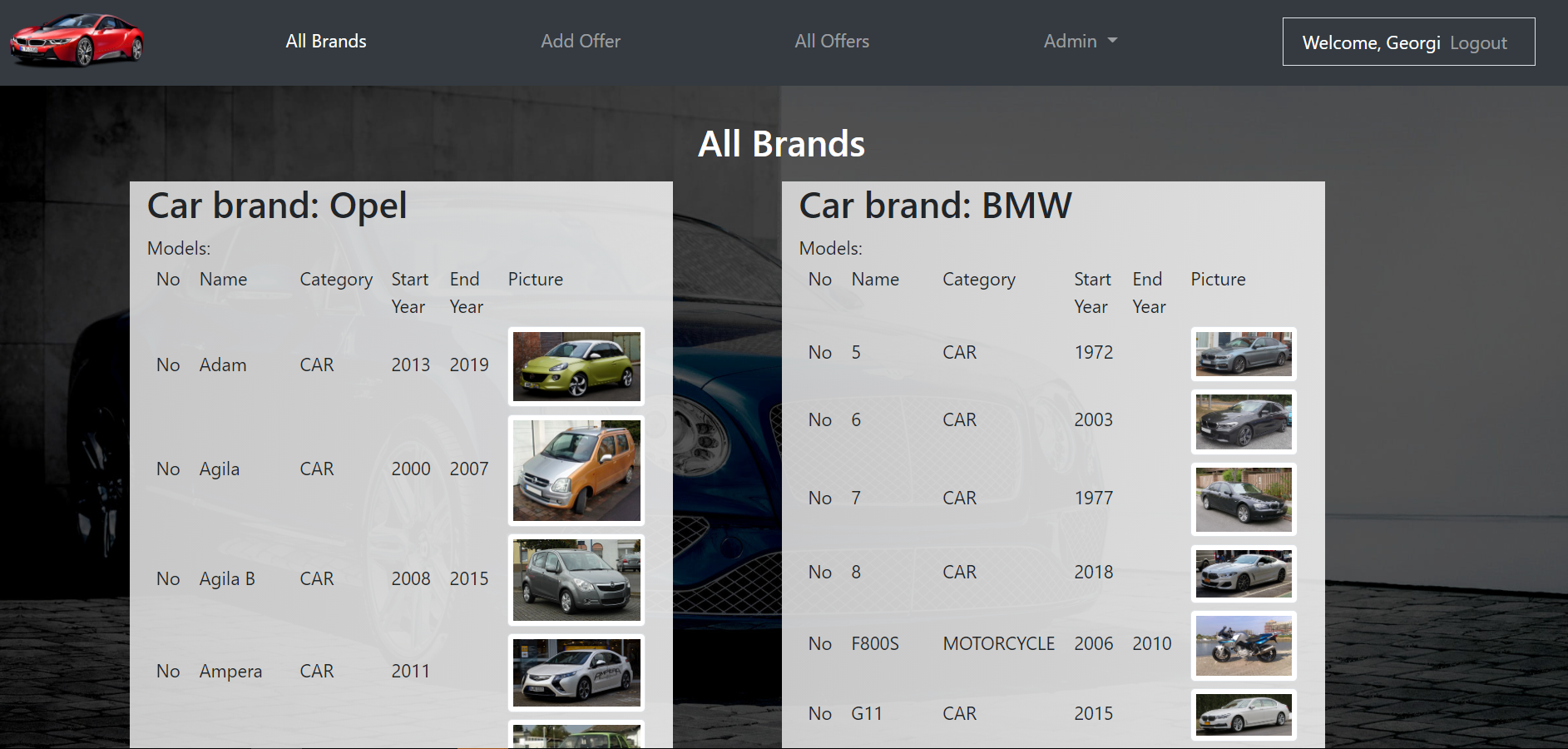
Expected result for login user



### All brands and models in out DB - route ("/brands/all").

It should support only a **GET** request.

It should return the following HTML page, upon a **GET** request.

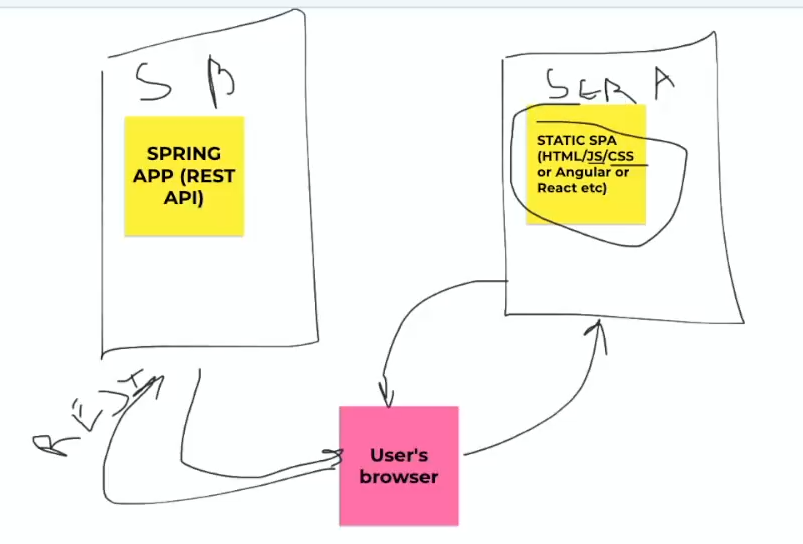


We continue with more functionality on the next lab. ☺

# III. Въпроси

## 1. Single Page Application и Back end

//TO DO : SPA and thymeLeaf ?



От Users browser-a се изпраща HTTP Request до Server A, където се намира нашето SPA приложение. То се сваля в browser-a на User-a като програма. След това Browser-a започва да изпраща REST заявки до Server B, където се намира нашия back end. Това е най-използваният pattern в практиката за визуализиране на Front end, който се нарича REST. По принцип е възможно и Front end и Back end частите да оперират на един сървър.

## 2. Lombok анотации

## 3. Repository extend JPARepository <Entity, Long> refresh

## 3. uuid ? advantages over primary id

## 4. AutoConfiguration Spring boot dependency

(auto-configured) – те вдигат инстанциите в Spring (= new () ).

## 5. Rest API ?

Вика се HTTP request, който връща JSON (не връща HTML).

## 6. SPA и Spring

## 7. Web Servers explaining ?

MySQL Localhost ? What is Localhost ?

## 8. Какво значи да се хешира парола ?

Прави се с цел да се

Pbkdf2PasswordEncoder

## 9. Css bootstrap

## 7. DispatchServlet

## 8. Application Context