

NS4307

Network Programming

Java Web Programming III

Disclaimer

- Some parts of the notes is based on the online course provided in Treehouse (teamtreehouse.com).
 - **Course:** Spring
 - **Teacher:** Chris Ramacciotti

Data Repository

- At the moment, it is only able to display the details of one hard coded object. But what if we want the controller method and template to control the display the details of any object from a list of objects according to the URL requested?
- A repository is essentially a collection that we've stored somewhere in memory or in a database.
 - Lets create a new package called **data** in src/main/java directory and create a Java class into the package.
 - In here, we'll implement a data structure to store the list of data.
 - Implement methods to retrieve necessary data.

Data Repository (cont.)

- This class will act as both the storage device for objects and a methods for interacting with those objects.
- For now we are going to store it within the application using static Java list but in the future we will be interacting with a database.
- To reduce our coding, we are going to allow Spring to initialize fields for us if it find a spring component of the same class for the field.
 - We add @Component annotation to the class.

Example Data Repository

- For this example:
 - We are going to create a class called ProductRepository.java into data package.
 - Add @Component annotation to the class.
 - Create a constant data structure to store all products.

```
@Component
public class ProductRepository {

    private static final List<Product> ALL_PRODUCTS = Arrays.asList(
        new Product("Apple iPhone 6s", 1210.50, "iphone6s.png", true),
        new Product("Apple iPad Pro", 1310.99, "ipadpro.png", true),
        new Product("Samsung Galaxy S7 Edge", 835.92, "samsungs7edge.png", false),
        new Product("Samsung Galaxy Note 7", 1035.92, "samsungnote7.png", true),
        new Product("HTC One X9", 850.20, "htconex9.png", false),
        new Product("Oppo F1s", 400.28, "oppof1s.png", true)
    );
}
```

Example Data Repository (cont.)

- Next,
 - Add a getter method to allow retrieval of all of the products.

```
@Component
public class ProductRepository {

    private static final List<Product> ALL_PRODUCTS = Arrays.asList(
        new Product("Apple iPhone 6s", 1210.50, "iphone6s.png", true),
        new Product("Apple iPad Pro", 1310.99, "ipadpro.png", true),
        new Product("Samsung Galaxy S7 Edge", 835.92, "samsungs7edge.png", false),
        new Product("Samsung Galaxy Note 7", 1035.92, "samsungnote7.png", true),
        new Product("HTC One X9", 850.20, "htconex9.png", false),
        new Product("Oppo F1s", 400.28, "oppof1s.png", true)
    );

    public List<Product> getAllProducts() {
        return ALL_PRODUCTS;
    }
}
```

Data Repository in Controller

- Since we set the repository class as Spring Component (using `@Component` annotation):
 - All we need is to declare the repository as the controller field and annotate it with `@Autowired` for Spring to initialise the field.
- For Thymeleaf to access the repository in the controller, you need to pass a `ModelMap` object as the parameter.
 - Then use the `ModelMap` `put` method to insert the repository into it.
 - Example: `modelMap.put("products", productList);`
 - "products" is the key for the repository object `productList` in the map.
- The same as previous, the key mentioned in the `ModelMap` can be used by Thymeleaf to access the data repository.

Example Data Repository in Controller

- Add a field with ProductRepository and a @Autowired annotation.
- Change the home method we created earlier.
 - Instead we create a single product object, we declare a List object with Product data type.
 - Then we use the getter method from the repository object.
 - Then put the List into the ModelMap.

Example Data Repository in Controller (cont.)

```
@Controller
public class MainController {
    @Autowired
    private ProductRepository productRepository;

    @RequestMapping(value="/")
    public String home(ModelMap modelMap) {
        List<Product> p = productRepository.getAllProducts();
        modelMap.put("products", p);
        return "index";
    }
}
```

List of Data to Thymeleaf Templates

- In the HTML file, to get the list of data in repository using Thymeleaf. You need to access them using a for each loop.
- To implement the for each loop in HTML file.
 - Add attribute into the outer tag where each data will be displayed **th:each=""**.
 - The value inside the attribute will be **th:each="data : \${listOfData}"**.
 - Then access the data the same as previous
 - Example: `<h4 th:text="${data.field}"></h4>`
 - ↑
Instance Variable in the object class.
 - ↑
For each data object, the fields can be access through this object.
 - ↑
The key stated in ModelMap object

Example List of Data to Thymeleaf Templates

- Lets edit the index.html file provided in the template.
 - Line 81: Add attribute **th:each="product : \${products}"**.
- Previously we have implemented Line 83, 85 and 86 to retrieve Product picFile, name and price respectively. So we don't need to add further.

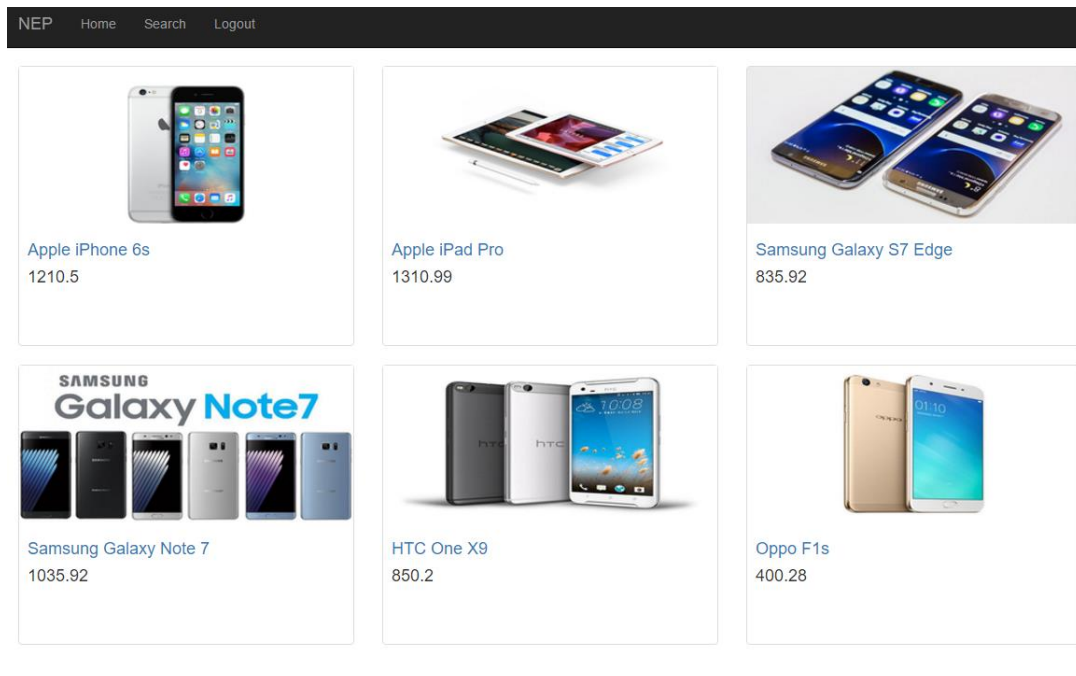
Example List of Data to Thymeleaf Templates (cont.)

- Added attribute.

```
<!-- Slide #44 Example List of Data to Thymeleaf Templates -->
<div class="col-sm-4 col-lg-4 col-md-4" th:each="product : ${products}">
  <div class="thumbnail">
    <!-- Slide #32 Example Data to Thymeleaf Templates -->
    
    <div class="caption">
      <!-- Slide #32 Example Data to Thymeleaf Templates -->
      <!-- Slide #51 Example Dynamic Page (cont.) -->
      <a href="product.html"><h4 th:text="${product.name}"></h4></a>
      <!-- Slide #32 Example Data to Thymeleaf Templates -->
      <h4 th:text="${product.price}"></h4>
    </div>
  </div>
</div>
</div>
```

Example List of Data to Thymeleaf Templates

- When you run and access the web application.
 - The page will display the All the list of Product repository that you added into the Controller ModelAndView earlier.




Dynamic Page

- Now that you have implemented a list of data repository to your page. But what if you want to allow user to click one of the data and go to a page where it shows the clicked data details only.
- One way to do it is, to have a dynamic Uniform Resource Identifier (URI) depending on the data clicked.
 - Example: A home page (/) where it has a list of Product data and one of it has a name called Samsung and another one called Apple.
 - When the user clicked on Samsung, it will go to URI (/product/Samsung).
 - When the user clicked on Apple, it will go to URI (/product/Apple).

Dynamic Page (cont.)

- Before we create and handle dynamic URI, create a method to retrieve a single data from the data repository based on what field.
 - Example: Retrieve a Product object from Project repository based on the name of the product.
- To generate a dynamic URI, in your html file.
 - Use a hyperlink tag with attribute as follows.
 - `<a th:href="@{'/URI/' + ${data.field}}">Show the text here`



Any URI at
the front.

Concatenated with data.

Dynamic Page (cont.)

- To handle request of a dynamic URI, in your controller.
 - Add an annotation **@RequestMapping(value="/URI/{field}")**.
 - Create a method with parameter variable:
 - **@PathVariable String field**
 - **ModelMap modelMap**
 - Retrieve the data based on the field.
 - Put it into the ModelMap object.
 - Send a response.

↑
This is a path variable. So any value inputted here will be passed into the method.

↙
@PathVariable annotation is for notifying that the method will get the data stated in URI.

Example Dynamic Page

- Lets add a method in the ProductRepository:
 - The following method will find product in the list of products where the name is equal to the value passed in the parameter.

```
public Product findByName(String name) {  
    for(Product product : ALL_PRODUCTS) {  
        if(product.getName().equals(name)) {  
            return product;  
        }  
    }  
    return null;  
}
```

Example Dynamic Page (cont.)

- In index.html
 - With comment `<!-- Slide #51 Example Dynamic Page (cont.) -->`
 - Add a hyperlink tag `<a>` and put it outside of the `<h4>` tag.
 - Then add an attribute `th:href` with value `@{'/product/' + ${product.name}}`.

Concatenated with the name of the product. It will generate a hyperlink for each product using the product name.

Product details URI

- In product.html
 - With comment `<!-- Slide #51 Example Dynamic Page (cont.) -->`
 - Make changes same as of Slide #32.

Example Dynamic Page (cont.)

- In controller class:
 - Add an annotation `@RequestMapping(value="/product/{name}")`.
 - Create a method: `public String getProduct(@PathVariable String name, ModelMap modelMap) { }`
 - Retrieve the Product based on the name of the product from the Product Repository.
 - `Product p = productRepository.findByName(name);`
 - Put the Product into the ModelMap.
 - `modelMap.put("product", p);`
 - Return `product.html`.

Example Dynamic Page (cont.)

- Changes in ProductRepository:

```
@Component
public class ProductRepository {
    private static final List<Product> ALL_PRODUCTS = Arrays.asList(
        new Product("Apple iPhone 6s", 1210.50, "iphone6s.png"),
        new Product("Apple iPad Pro", 1310.99, "ipadpro.png"),
        new Product("Samsung Galaxy S7 Edge", 835.92, "samsungs7edge.png"),
        new Product("Samsung Galaxy Note 7", 1035.92, "samsungnote7.png"),
        new Product("HTC One X9", 850.20, "htconex9.png"),
        new Product("Oppo F1s", 400.28, "oppof1s.png")
    );

    public Product findByName(String name) {
        for(Product product : ALL_PRODUCTS) {
            if(product.getName().equals(name)) {
                return product;
            }
        }
        return null;
    }

    public List<Product> getAllProducts() {
        return ALL_PRODUCTS;
    }
}
```

Example Dynamic Page (cont.)

- Changes in index.html

```

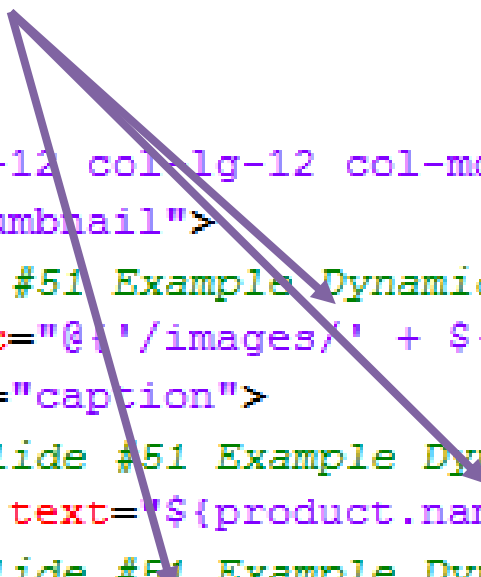
<!-- Slide #44 Example List of Data to Thymeleaf Templates -->
<div class="col-sm-4 col-lg-4 col-md-4" th:each="product : ${products}">
  <div class="thumbnail">
    <!-- Slide #32 Example Data to Thymeleaf Templates -->
    
    <div class="caption">
      <!-- Slide #32 Example Data to Thymeleaf Templates -->
      <!-- Slide #51 Example Dynamic Page (cont.) -->
      <a th:href="@{'/product/' + ${product.name}}"><h4 th:text="${product.name}"</h4></a>
      <!-- Slide #32 Example Data to Thymeleaf Templates -->
      <h4 th:text="${product.price}"</h4>
    </div>
  </div>
</div>

```

Example Dynamic Page (cont.)

- Changes in product.html

```
<div class="col-sm-12 col-lg-12 col-md-12">
  <div class="thumbnail">
    <!-- Slide #51 Example Dynamic Page (cont.) -->
    
    <div class="caption">
      <!-- Slide #51 Example Dynamic Page (cont.) -->
      <h4 th:text="${product.name}"></h4>
      <!-- Slide #51 Example Dynamic Page (cont.) -->
      <h4 th:text="${product.price}"></h4>
    </div>
  </div>
</div>
```



Example Dynamic Page (cont.)

- Changes in controller class:

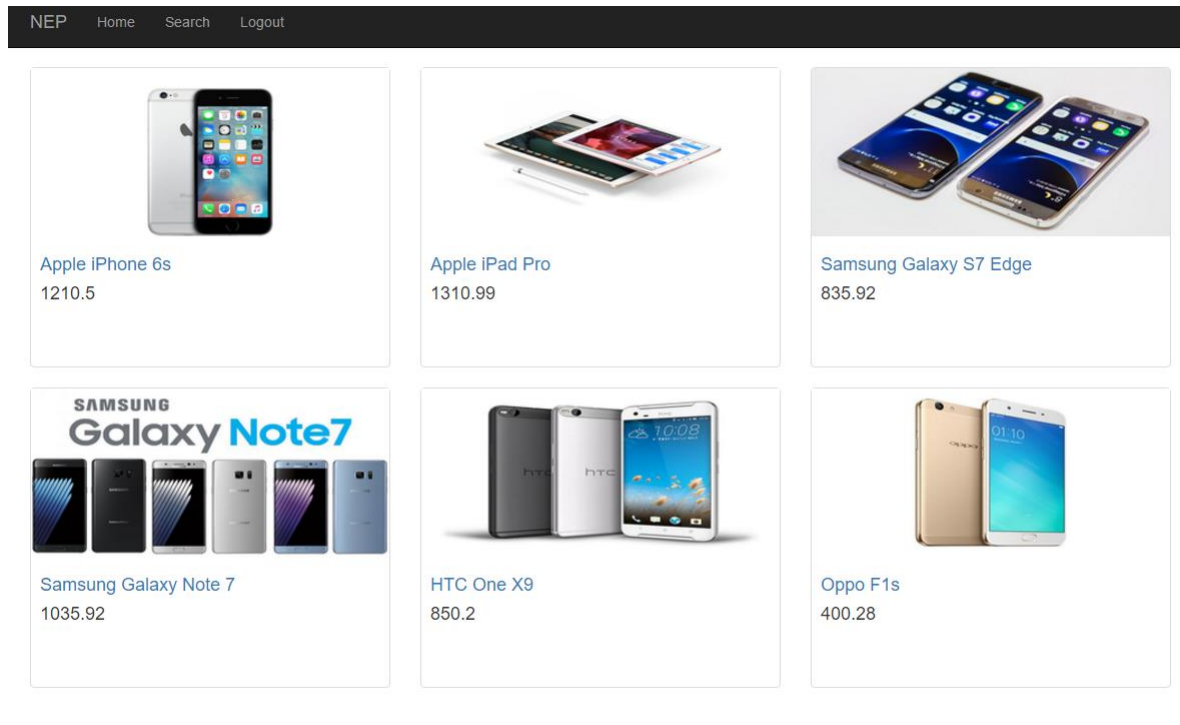
```
@Controller
public class MainController {
    @Autowired
    private ProductRepository productRepository;

    @RequestMapping(value="/")
    public String allProducts(ModelMap modelMap) {
        List<Product> p = productRepository.getAllProducts();
        modelMap.put("products", p);
        return "index";
    }

    @RequestMapping(value="/product/{name}")
    public String getProduct(@PathVariable String name, ModelMap modelMap) {
        Product p = productRepository.findByName(name);
        modelMap.put("product", p);
        return "product";
    }
}
```

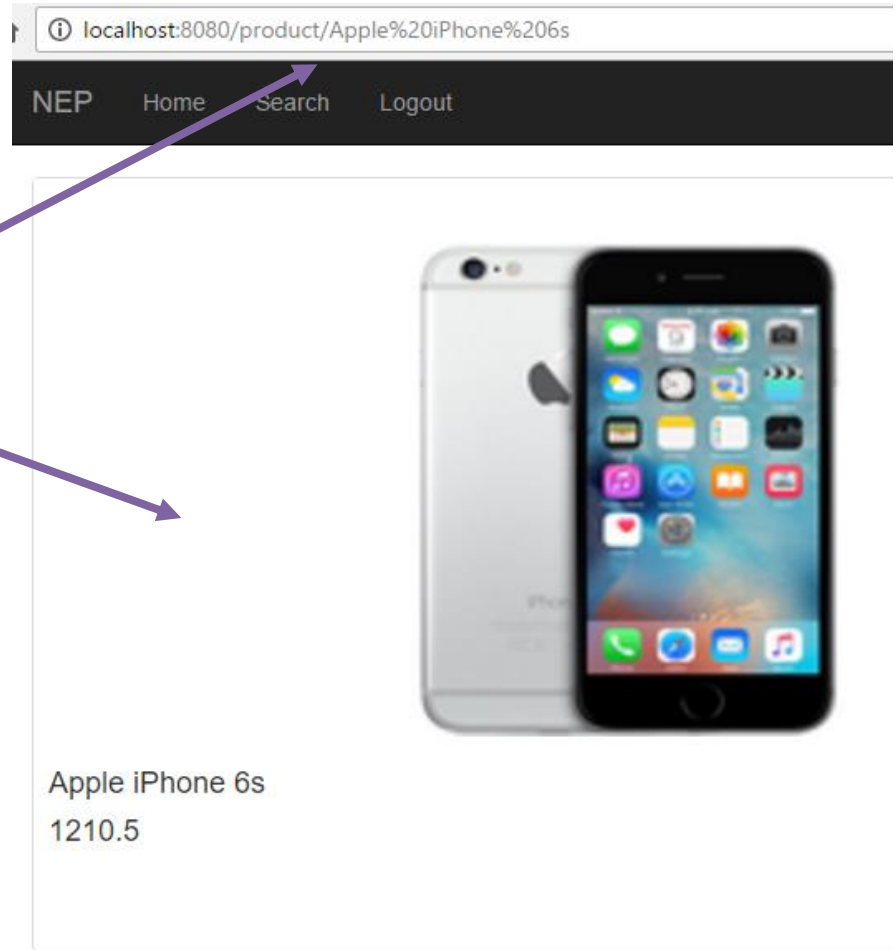
Example Dynamic Page Result

- The each name of the products will have hyperlink with URI to the name of the product.



Example Dynamic Page Result

- This page will be displayed with the details of a single Product.
 - URL with product name.
 - Single Product details.



Send Data with HTTP Request

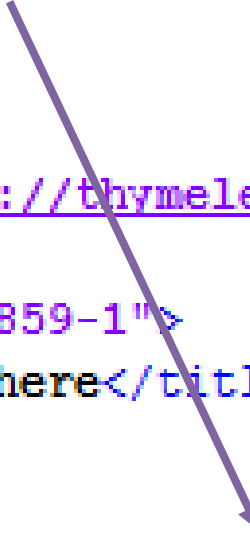
- In some websites, they provide some forms that user can input data into it. So far, we've simply requests to view certain resources where no additional data is supplied on the request.
- There are two methods to supply additional data on the request:
 - **GET Method:** Retrieve whatever information (in the form of an entity) is identified by the Request-URI.
 - **POST Method:** Used to request that the origin server accept the entity enclosed in the request as a new subordinate of the resource identified by the Request-URI in the Request-Line.

Send Data with HTTP Request (cont.)

- Usually these methods is implemented in HTML form tag.

- Example:

```
<!DOCTYPE html>
<html xmlns:th="http://thymeleaf.org">
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
<form action="/find" method="get">
    <input type="text" name="firstName"/>
</form>
</body>
</html>
```



Send Data with HTTP Request (cont.)

- To handle the GET/POST HTTP request, in the controller add **@RequestParam** annotation on a parameter variable to the method that will handle the request.
 - The parameter variable should be the same as name attribute in form component.
- It is possible make the @RequestParam as required or optional.
 - **@RequestParam(required=false)**
 - or
 - **@RequestParam(required=true)**

Example Send Data with HTTP Request

- Lets add a method in the ProductRepository:
 - The following method will find product in the list of products where the value passed in the parameter contain in the name.

```
public List<Product> findContainName(String name) {  
    List<Product> products = new ArrayList<Product>();  
    for(Product product : ALL_PRODUCTS) {  
        if(product.getName().contains(name)) {  
            products.add(product);  
        }  
    }  
    return products;  
}
```

Example Send Data with HTTP Request (cont.)

- In search.html
 - With comment <!-- Slide #63 Example Send Data with HTTP Request (cont.) -->
 - Make changes same as of Slide #32, #44, #51 in index.html.

Example Send Data with HTTP Request (cont.)

- In controller class:
 - Add an annotation `@RequestMapping(value="/search")`.
 - Create a method: `public String searchProducts(@RequestParam(required=false) String productName, ModelMap modelMap) { }`
 - Check if there is currently no HTTP Request, get all products.
 - If there is HTTP Request, get the products that contain the searched value.
 - Return product.html.

Example Send Data with HTTP Request (cont.)

- Changes in ProductRepository.

```
@Component
public class ProductRepository {

    private static final List<Product> ALL_PRODUCTS = Arrays.asList(
        new Product("Apple iPhone 6s", 1210.50, "iphone6s.png"),
        new Product("Apple iPad Pro", 1310.99, "ipadpro.png"),
        new Product("Samsung Galaxy S7 Edge", 835.92, "samsungs7edge.png"),
        new Product("Samsung Galaxy Note 7", 1035.92, "samsungnote7.png"),
        new Product("HTC One X9", 850.20, "htconex9.png"),
        new Product("Oppo F1s", 400.28, "oppof1s.png")
    );

    public Product findByName(String name) {

    }

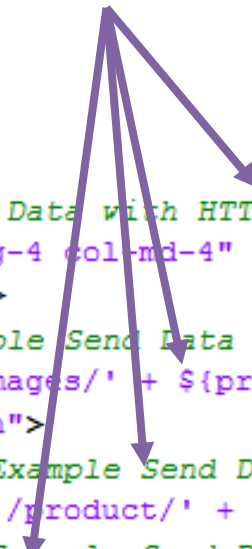
    public List<Product> findProducts(String name) {
        List<Product> products = new ArrayList<Product>();
        for(Product product : ALL_PRODUCTS) {
            if(product.getName().contains(name)) {
                products.add(product);
            }
        }
        return products;
    }

    public List<Product> getAllProducts() {

    }
}
```


Example Send Data with HTTP Request (cont.)

- Changes in search.html



```
<!-- Slide 63 Example Send Data with HTTP Request (cont.) -->
<div class="col-sm-4 col-lg-4 col-md-4" th:each="product : ${products}">
  <div class="thumbnail">
    <!-- Slide 63 Example Send Data with HTTP Request (cont.) -->
    
    <div class="caption">
      <!-- Slide 63 Example Send Data with HTTP Request (cont.) -->
      <a th:href="@{'/product/' + ${product.name}}"><h4 th:text="${product.name}"</h4></a>
      <!-- Slide 63 Example Send Data with HTTP Request (cont.) -->
      <h4 th:text="${product.price}"</h4>
    </div>
  </div>
</div>
```

Example Send Data with HTTP Request (cont.)

- Changes in controller class.

```
@Controller
public class MainController {
    @Autowired
    private ProductRepository productRepository;

    @RequestMapping(value="/")
    public String allProducts(ModelMap modelMap) {

        @RequestMapping(value="/product/{name}")
        public String getProduct(@PathVariable String name, ModelMap modelMap) {

            @RequestMapping(value="/search")
            public String searchProducts(@RequestParam(required=false) String productName, ModelMap modelMap) {
                if(productName == null) {
                    List<Product> p = productRepository.getAllProducts();
                    modelMap.put("products", p);
                } else {
                    List<Product> p = productRepository.findContainName(productName);
                    modelMap.put("products", p);
                }
                return "search";
            }
        }
    }
}
```