Practical 05

Project 03 - Spring Web Application (Integrating with Hibernate)

‼️Important Note (Before Attempting this Project)

1. Make sure you complete **Practical 04 - Project 02** because this is a continuation from that project.
2. Since this is a continuation from the previous project, the Git remote is pointing to Practical 04 Github classroom. **You need to change the Git remote to Practical 05 Github classroom**
    1. Check the Git remote name in your repository.

        ```
        git remote -v
        ```

        The output will look something like this:

        ```
        origin  https://github.com/user/repo_name.git (fetch)
        origin  https://github.com/user/repo_name.git (push)
        ```

    2. Assuming the remote name is `origin`. Remove the remote URL pointing to the Practical 04 Github classroom based on the remote name. **If your remote name is different, change `origin` to the appropriate remote name stated in `git remote -v`**.

        ```
        git remote remove origin
        ```

3. Go to Practical 05 Github classroom repository and add a new remote URL to the project. **Make sure the URL is for Practical 05, you can accept the invitation to Practical 05 in PBLMS**.

```
git remote add origin <URL>
```

❗ Change `<URL>` to your specific Practical 04 URL.

4. Double check the URL and you can try `git push origin` to the new URL.
5. Then got to your Practical 05 Github classroom repository to check if the files are uploaded there.

Lesson 3.1 - Retrieving all products from Database and Adding product to Database

1. We left off with a lot of commented codes in our `HomeController`. We also have `ProductRepository` interface which extends `CrudRepository` interface.

```
package com.nep.practical.controller;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.ModelMap;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;

import com.nep.practical.data.ProductRepository;
import com.nep.practical.model.Product;

@Controller
public class HomeController {

  @Autowired
  ProductRepository productRepository;

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

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

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

}
```

```
package com.nep.practical.data;

import org.springframework.data.repository.CrudRepository;
import org.springframework.stereotype.Component;

import com.nep.practical.model.Product;

@Component
public interface ProductRepository extends CrudRepository<Product, Integer>
{

}
```

2. The next step is integrate `ProductRepository` interface to `HomeController`. We have `@Autowired` it to the class already from previous practical. Even though we changed it to an interface it still works accordingly. All we need to do is to call the methods provided by `CrudRepository` interface. Lets complete our implementation in method `home()`. To retrieve all products from database, we need to call the `findAll()` method from the CrudRepository.

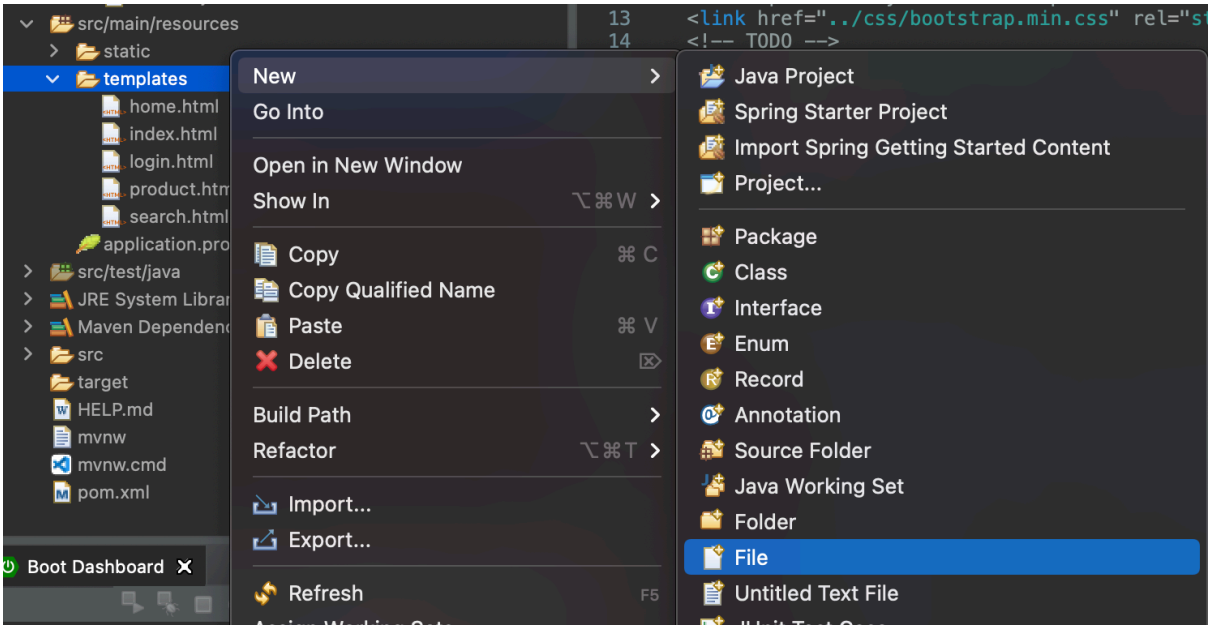
```
package com.nep.practical.controller;

import java.util.List;
```

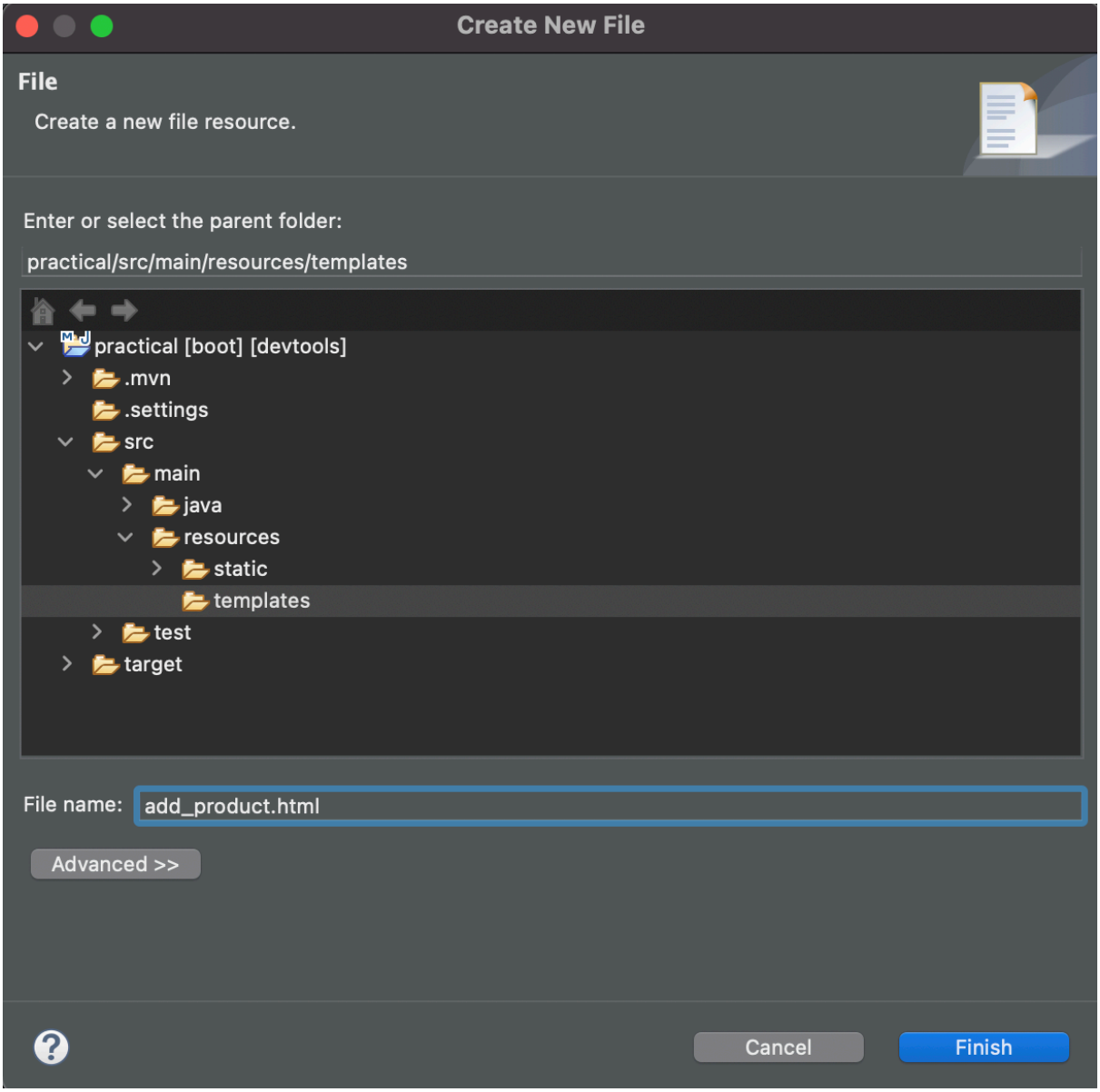
```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.ModelMap;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;

import com.nep.practical.data.ProductRepository;
import com.nep.practical.model.Product;

@Controller
public class HomeController {

  @Autowired
  ProductRepository productRepository;

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

  ...
}
```

3. Go to URI `/`. It will load the home page but it doesn't display any data since our database table is empty. Before we add data, lets change the value in `application.properties` for `spring.jpa.hibernate.ddl-auto` from `create-drop` to `update`. **After that reload the server**. Now if we save any data, since the table will no longer being dropped, the saved data are persistence.

```
...

spring.jpa.hibernate.ddl-auto=update
...
```

4. Since there is no data in the table, we need a way to insert data in to it. Since the template for adding data is not included. Lets just create a simple form to reduce complexity towards our html file (Note: it would be better if the add page design is consistent with the other pages). Lets create `add_product.html` in `src/main/resources/templates`.





5. `add_product.html` content is as follows:

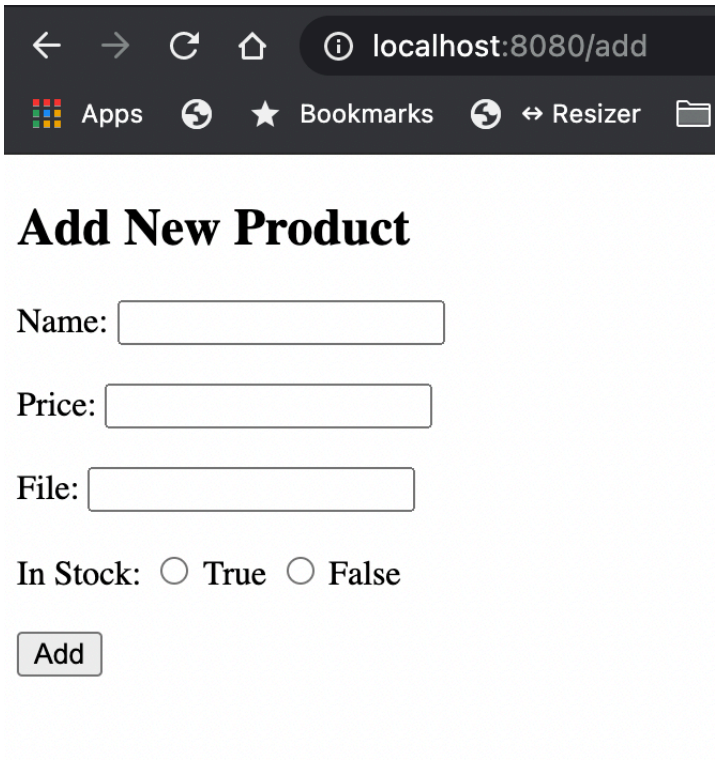
```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Add New Product</title>
</head>
<body>
  <form action="/add" method="post">
        <h2>Add New Product</h2>
        <label>Name:</label>
        <input type="text" name="name" /><br><br>
        <label>Price:</label>
        <input type="text" name="price" /><br><br>
        <label>File:</label>
        <input type="text" name="file" /><br><br>
        <label>In Stock:</label>
        <input type="radio" name="instock" value="true"> True
        <input type="radio" name="instock" value="false"> False<br><br>
        <button type="submit">Add</button>
    </form>
</body>
</html>
```

6. Keep note of the input tag attribute `name` inside the form tag. This will be use in our `HomeController`. If the user click the `Add` button, it will send HTTP Request method post and include the value in the input tag in the HTTP Header. We need to capture this in `HomeController`. First we need to display `add_product.html` in URI `/add`.

```
package com.nep.practical.controller;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.ModelMap;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;

import com.nep.practical.data.ProductRepository;
import com.nep.practical.model.Product;

@Controller
public class HomeController {
  ...
```

```
  @RequestMapping(value="/add")
  public String add() {
    return "add_product";
  }
}
```

7. Go to URI `/add`, it should display the page as follow:

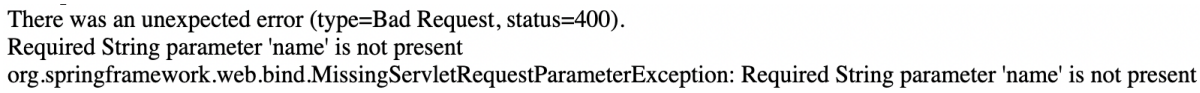


8. Now to capture the values in the HTTP Header (the value sent from input tag), we need to add parameter variable with annotation `@RequestParam`. The **identifier of the parameter variable** needs to be the same as the input tag attribute `name` value.

```
...
@Controller
public class HomeController {
  ...

  @RequestMapping(value="/add")
  public String add(
      @RequestParam String name,
      @RequestParam String price,
      @RequestParam String file,
      @RequestParam String instock
      ) {

    return "add_product";
  }
```

```
}
```

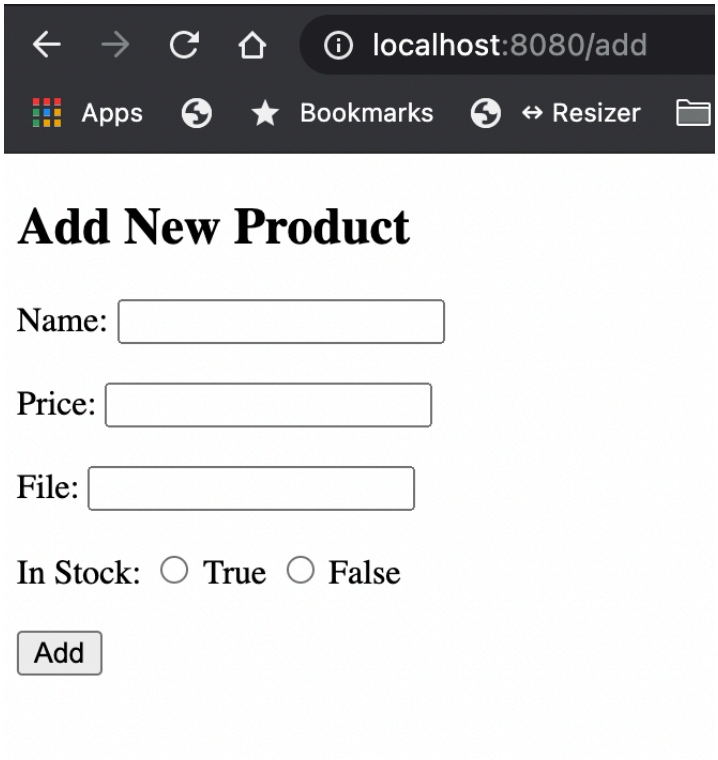
9. But since it is the same URI `/add`, if we initially go to that URI, no data is included. This will cause an error.



10. So we need to set the `@RequestParam` as not required.

```
...
@Controller
public class HomeController {
  ...

  @RequestMapping(value="/add")
  public String add(
      @RequestParam(required=false) String name,
      @RequestParam(required=false) String price,
      @RequestParam(required=false) String file,
      @RequestParam(required=false) String instock
      ) {

    return "add_product";
  }
}
```

11. Go to URI `/add`, there are no more errors.



12. Lets add an implementation where if there are no data being passed, just show the page and if some data is missing or invalid show a warning message. To indicate warning message, lets declare a boolean variable. If all data are valid, add product is successful. We need to indicate that as well. Lets declare another boolean variable.

```
...
@Controller
public class HomeController {
  ...

  @RequestMapping(value="/add")
  public String add(
      @RequestParam(required=false) String name,
      @RequestParam(required=false) String price,
      @RequestParam(required=false) String file,
      @RequestParam(required=false) String instock
      ) {
    boolean success = false;
    boolean fail = false;
    if(name == null && price == null
       && file == null && instock == null) {
      success = false;
      fail = false;
    } else {
      if(name == null || price == null
         || file == null || instock == null) {
        success = false;
        fail = true;
```

```
      } else {
        try {
          double priceInDouble = Double.parseDouble(price);
          boolean instockInBoolean = Boolean.parseBoolean(instock);



          success = true;
          fail = false;
        } catch (NumberFormatException e) {
          success = false;
          fail = true;
        }
      }
    }
    return "add_product";
  }
}
```

13. Lets pass the boolean variables to `add_product.html`.

```
...
@Controller
public class HomeController {
  ...

  @RequestMapping(value="/add")
  public String add(
      ...
      @RequestParam(required=false) String instock,
      ModelMap modelMap
      ) {
    boolean success = false;
    boolean fail = false;
    if(name == null && price == null
       && file == null && instock == null) {
      ...
    }
    modelMap.put("success", success);
    modelMap.put("fail", fail);
    return "add_product";
  }
}
```

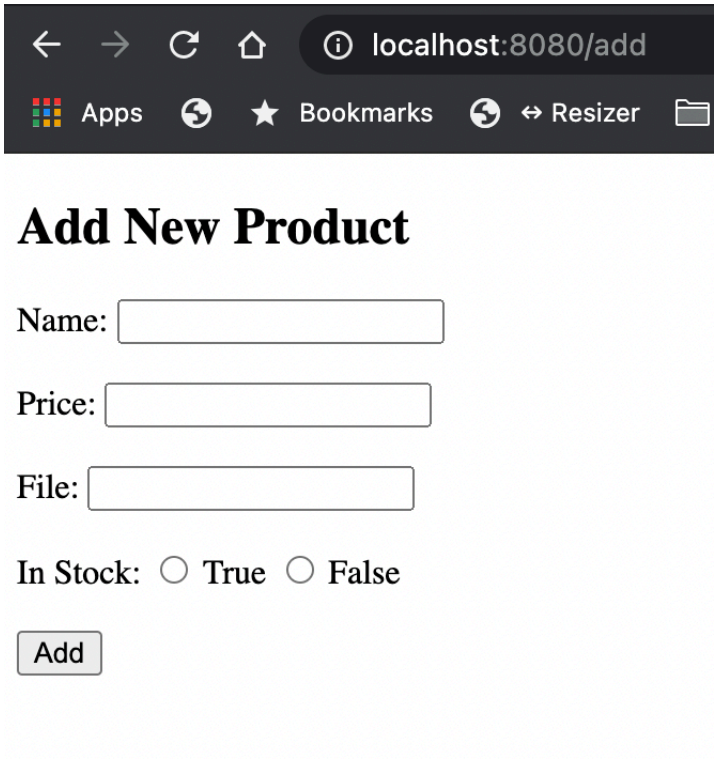
14. `add_student.html` have to handle this boolean. Lets add two label to indicate success add or not below the form tag.

```
...
<body>
  <form action="/add" method="post">
      ...
  </form>
  <label>Add new product successful</label>
  <label>Add new product failed</label>
</body>
</html>
```

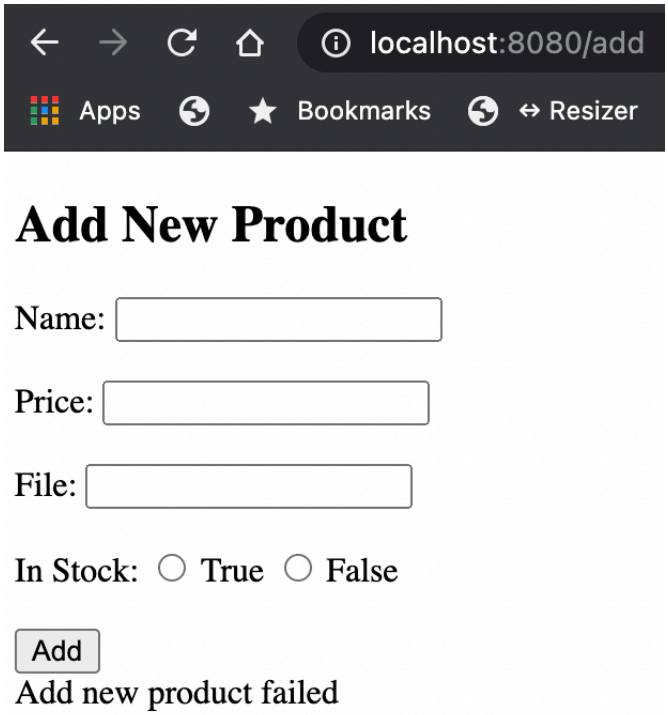
15. Now we want if success or fail it shows the respective label. Thymeleaf has an if statement where we can put as an attribute `th:if`.

```
...
<body>
  <form action="/add" method="post">
      ...
  </form>
  <label th:if="${success == true}">Add new product successful</label>
  <label th:if="${fail == true}">Add new product failed</label>
</body>
</html>
```

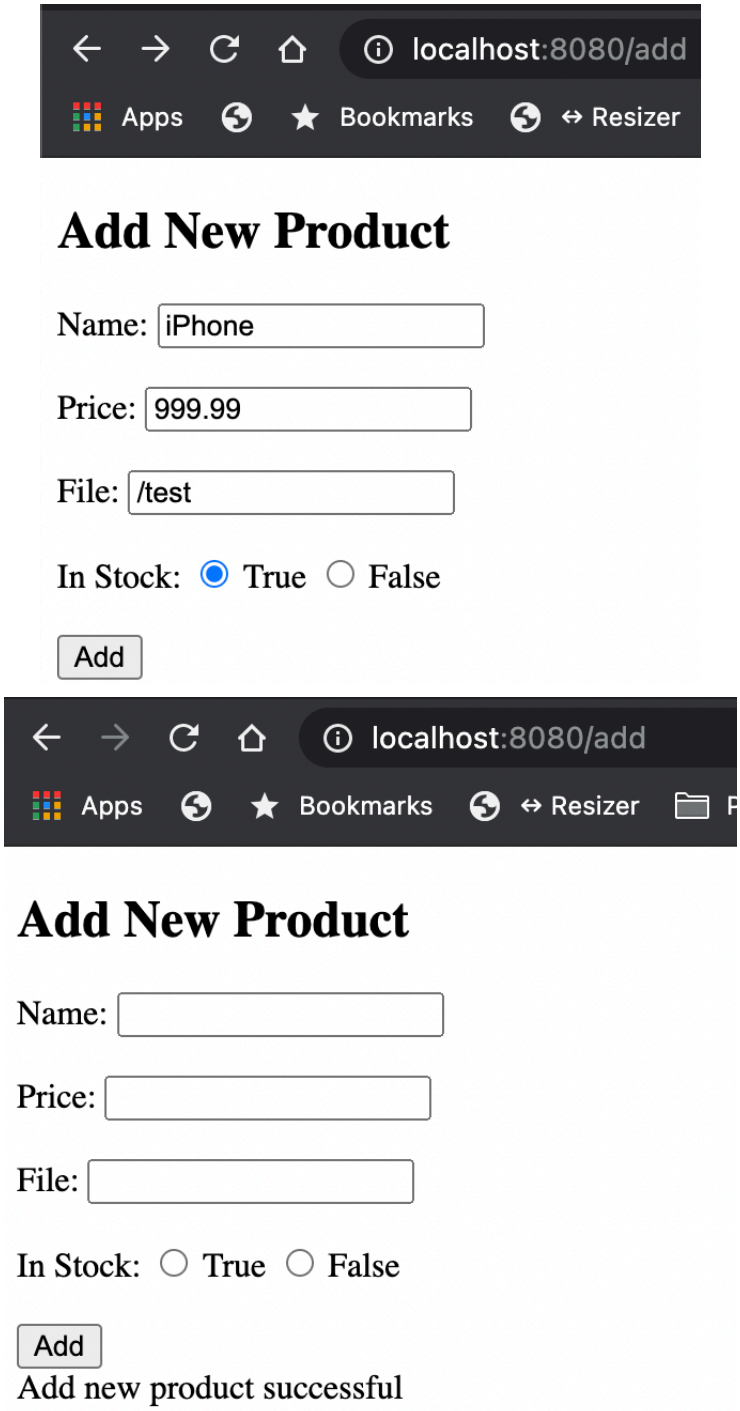
16. Go to URI `/add`.
    - Initially no label is displayed.



- When user click Add button without filling in anything or missed one of the input or invalid value is added (Example: user filled price as text). It will display the fail message.



- When the user filled all accordingly and click Add.



17. Now lets add these data to the database.

```
...
@Controller
public class HomeController {
  ...

  @RequestMapping(value="/add")
  public String add(
      @RequestParam(required=false) String name,
      @RequestParam(required=false) String price,
      @RequestParam(required=false) String file,
      @RequestParam(required=false) String instock
```

```
      ) {
    ...
    if(name == null && price == null
       && file == null && instock == null) {
      ...
        try {
          double priceInDouble = Double.parseDouble(price);
          boolean instockInBoolean = Boolean.parseBoolean(instock);

          Product product = new Product(name, priceInDouble,
                    file, instockInBoolean);
          productRepository.save(product);

          success = true;
          fail = false;
        } catch (NumberFormatException e) {
          ...
        }
      }
    }
    modelMap.put("success", success);
    modelMap.put("fail", fail);
    return "add_product";
  }
}
```

18. Lets add the data to the database.
    1. **Name**: Apple iPhone 6s

       Price: 1210.50

       File: /images/iphone6s.png

       Instock: True
    2. **Name**: Apple iPad Pro

       Price: 1310.99

       File: /images/ipadpro.png

       Instock: True
    3. **Name**: Samsung Galaxy S7 Edge

       Price: 835.92

       File: /images/samsungs7edge.png

       Instock: False
    4. **Name**: Samsung Galaxy Note 7

       Price: 1035.92

File: /images/samsungnote7.png

Instock: True

5. **Name**: HTC One X9

   Price: 850.20

   File: /images/htconex9.png

   Instock: False

6. **Name**: Oppo F1s

   Price: 400.28

   File: /images/oppof1s.png

   Instock: True

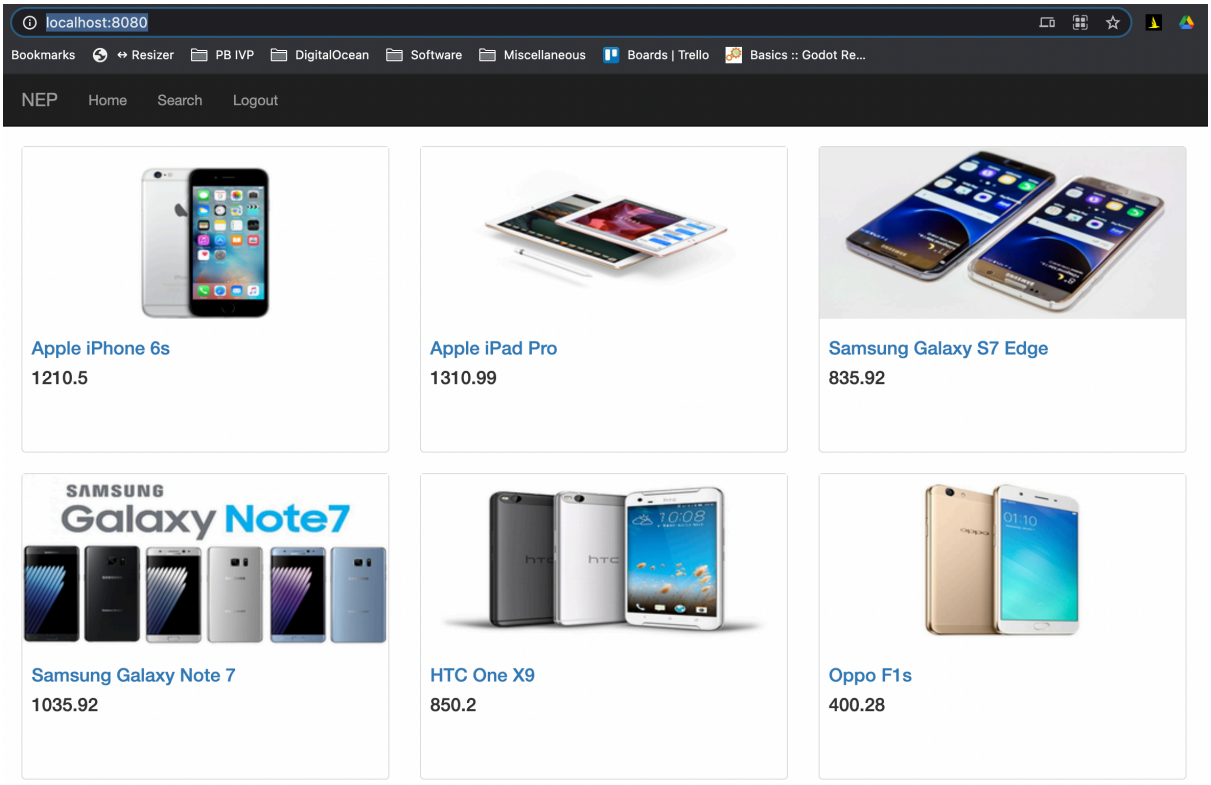
Add the data above using the form as per example below:

Make sure to click the Add button to add the product.

19. The MySQL table should contain data that we added. (The id might differ from this example).

id	file	in_stock	name	price
2	/images/iphone6s.png	1	Apple iPhone 6s	1210.50
3	/images/ipadpro.png	1	Apple iPad Pro	1310.99
4	/images/samsungs7edge.png	0	Samsung Galaxy S7 Edge	835.92
5	/images/samsungnote7.png	1	Samsung Galaxy Note 7	1035.92
6	/images/htconex9.png	0	HTC One X9	850.20
7	/images/oppof1s.png	1	Oppo F1s	400.28

20. Going back to URI `/`. The products should be displayed here.



> ## ⚠ Milestone:
>
> Commit the changes made to the project folder with message "Lesson 3.1 - Retrieving all products from Database and Adding product to Database".
>
> Hint: Add the files to the staging area first, `git add <files>` and then proceed to commit, `git commit -m "Lesson 3.1 - Retrieving all products from Database and Adding product to Database"`.

Lesson 3.2 - Deleting product from Database

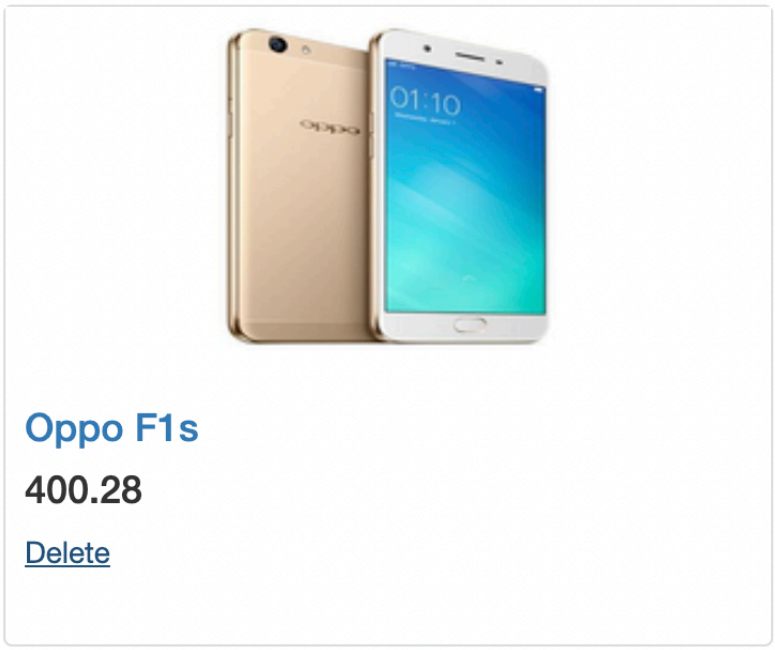
1. Lets add a delete link in the `index.html`.

```
...
                                <!-- TODO -->
                                <!-- Slide #32 Example Data to Thymeleaf
Templates -->
                                <h4 th:text="${ product.price }"></h4>
                                <a th:href="@{'/delete/' +
${product.id}}">Delete</a>
                            </div>
                        </div>
                    </div>
                </div>
            </div>
        </div>
...
```

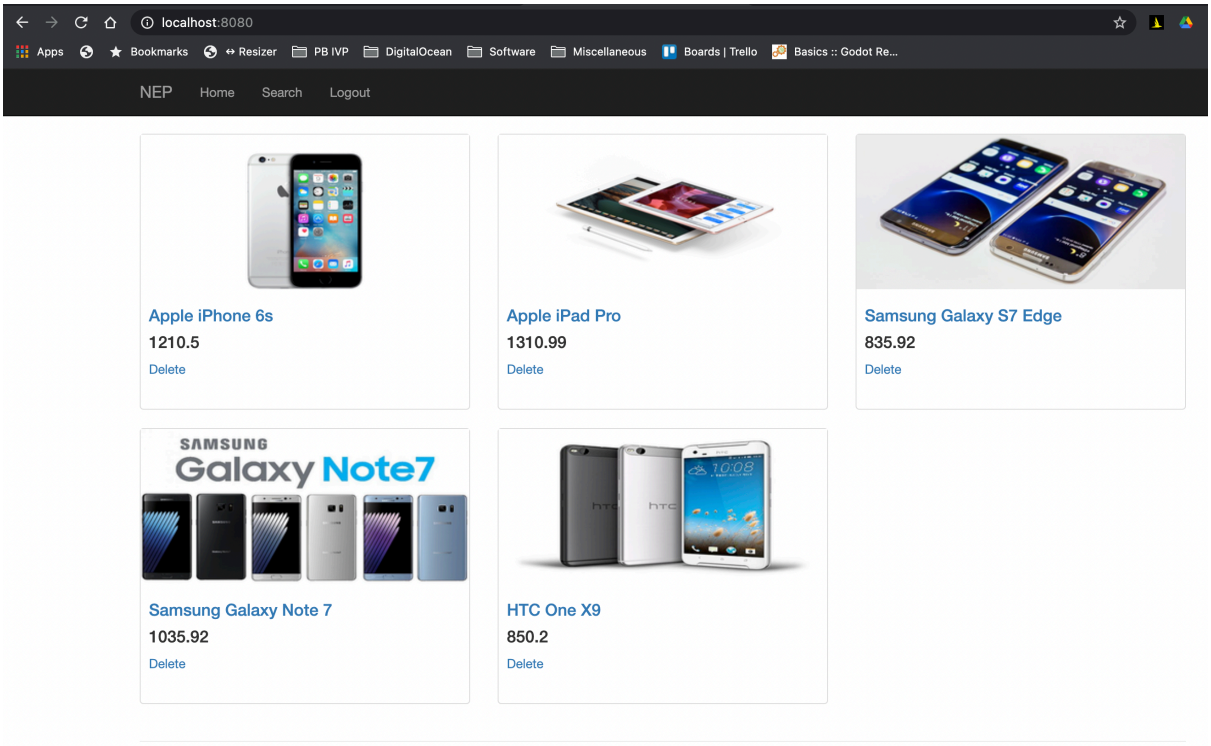
2. Next we need to handle the URI `/delete/{id}` in `HomeController`.

```
...
@Controller
public class HomeController {
...
  @RequestMapping(value="/delete/{id}")
  public String delete(
      @PathVariable int id
      ) {
    productRepository.deleteById(id);
    return "redirect:/";
  }
}
```

3. Lets try deleting Opponents F1s by clicking the Delete link.



4. Once clicked, it should be removed from Database.



Inside database also removed

id	file	in_stock	name	price
2	/images/iphone6s.png	1	Apple iPhone 6s	1210.50
3	/images/ipadpro.png	1	Apple iPad Pro	1310.99
4	/images/samsungs7edge.png	0	Samsung Galaxy S7 Edge	835.92
5	/images/samsungnote7.png	1	Samsung Galaxy Note 7	1035.92
6	/images/htconex9.png	0	HTC One X9	850.20

> ## ⚠️ Milestone:
>
> Commit the changes made to the project folder with message "Lesson 3.2 - Deleting product from Database".
>
> Hint: Add the files to the staging area first, `git add <files>` and then proceed to commit, `git commit -m "Lesson 3.2 - Deleting product from Database"`.

Lesson 3.3 - Editing product details in Database

1. Same case as adding product, we don't have a page to edit product. Create `edit_product.html` and its content is as follows (Its almost all the same as `add_product.html`):

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Edit Product</title>
</head>
<body>
  <form action="/edit" method="post">
        <h2>Edit Product</h2>
        <label>Name:</label>
        <input type="text" name="name" /><br><br>
        <label>Price:</label>
        <input type="text" name="price" /><br><br>
        <label>File:</label>
        <input type="text" name="file" /><br><br>
        <label>In Stock:</label>
        <input type="radio" name="instock" value="true"> True
        <input type="radio" name="instock" value="false"> False<br><br>
        <button type="submit">Edit</button>
    </form>
    <label th:if="${success == true}">Edit product successful</label>
    <label th:if="${fail == true}">Edit product failed</label>
</body>
</html>
```

2. Now in `index.html`, lets add the Edit hyperlink.

```
...
                                <!-- TODO -->
                                <!-- Slide #32 Example Data to Thymeleaf
Templates -->
                                <h4 th:text="${ product.price }"></h4>
```

```
                                    <a th:href="@{'/delete/' +
${product.id}}">Delete</a>
                                    <br><a th:href="@{'/edit/' +
${product.id}}">Edit</a>
                                </div>
                            </div>
                        </div>
                    </div>
                </div>
            </div>
        ...
```

3. Lets handle the URI `/edit/{id}` in `HomeController`.

```
...

@Controller
public class HomeController {
...

  @RequestMapping(value="/edit/{id}")
  public String edit(
      @PathVariable int id
      ) {

    return "edit_product";
  }
}
```

4. We need to populate the `edit_product.html` with product details. So we need to retrieve the product and pass to the html file.

```
...

@Controller
public class HomeController {
...

  @RequestMapping(value="/edit/{id}")
  public String edit(
      @PathVariable int id,
      ModelMap modelMap
      ) {
    Product product = productRepository.findById(id).get();
```

```
    modelMap.put("product", product);
    return "edit_product";
  }
}
```

5. We need to generate the action based on product id.

```
...
<body>
  <form th:action="@{'/edit/' + ${product.id}}" method="post">
  ...
  </form>
</body>
</html>
```

6. Display the product details in `edit_product.html`. For radio button, we need to check if inStock is true or not then we assign to attribute `th:checked`. This will set the radio button to be checked if the condition is true.

```
...
<body>
    <form th:action="@{'/edit/' + ${product.id}}" method="post">
        <h2>Edit Product</h2>
        <label>Name:</label>
        <input type="text" name="name" th:value="${ product.name }"/><br>
<br>
        <label>Price:</label>
        <input type="text" name="price" th:value="${ product.price }"/><br>
<br>
        <label>File:</label>
        <input type="text" name="file" th:value="${ product.file }"/><br>
<br>
        <label>In Stock:</label>
        <input type="radio" name="instock" value="true"
th:checked="${product.inStock == true}"> True
        <input type="radio" name="instock" value="false"
th:checked="${product.inStock == false}"> False<br><br>
        <button type="submit">Edit</button>
    </form>
    <label th:if="${success == true}">Edit product successful</label>
    <label th:if="${fail == true}">Edit product failed</label>
</body>
</html>
```

7. Lets further implement the `edit()` method in `HomeController`. First we need to retrieve the specific product based on the id.

```
...
@Controller
public class HomeController {
...

    @RequestMapping(value="/edit/{id}")
  public String edit(
      @PathVariable int id,
      ModelMap modelMap,
      @RequestParam(required=false) String name,
      @RequestParam(required=false) String price,
      @RequestParam(required=false) String file,
      @RequestParam(required=false) String instock
      ) {
    Product product = productRepository.findById(id).get();
    boolean success = false;
    boolean fail = false;
    if(product != null) {
      if(name == null && price == null
          && file == null && instock == null) {
        success = false;
        fail = false;
      } else {
        if(name == null || price == null
            || file == null || instock == null) {
          success = false;
          fail = true;
        } else {
          try {
            double priceInDouble = Double.parseDouble(price);
            boolean instockInBoolean = Boolean.parseBoolean(instock);



            success = true;
            fail = false;
          } catch (NumberFormatException e) {
            success = false;
            fail = true;
          }
        }
      }
      modelMap.put("success", success);
      modelMap.put("fail", fail);
      modelMap.put("product", product);
      return "edit_product";
```

```
    } else {
      return "redirect:/";
    }
  }
}
```

It is almost the same as `add{}` method so I don;t need to explain further.

8. To edit the database content, we need to access the product object and use the setter method.

```
...
@Controller
public class HomeController {
...

    @RequestMapping(value="/edit/{id}")
  public String edit(
      @PathVariable int id,
      ModelMap modelMap,
      @RequestParam(required=false) String name,
      @RequestParam(required=false) String price,
      @RequestParam(required=false) String file,
      @RequestParam(required=false) String instock
      ) {
    Product product = productRepository.findById(id).get();
    boolean success = false;
    boolean fail = false;
    if(product != null) {
      if(name == null && price == null
          && file == null && instock == null) {
        success = false;
        fail = false;
      } else {
        if(name == null || price == null
            || file == null || instock == null) {
          success = false;
          fail = true;
        } else {
          try {
            double priceInDouble = Double.parseDouble(price);
            boolean instockInBoolean = Boolean.parseBoolean(instock);

            product.setName(name);
            product.setPrice(priceInDouble);
            product.setFile(file);
            product.setInStock(instockInBoolean);
            productRepository.save(product);
```

```
            success = true;
            fail = false;
          } catch (NumberFormatException e) {
            success = false;
            fail = true;
          }
        }
      }
      modelMap.put("success", success);
      modelMap.put("fail", fail);
      modelMap.put("product", product);
      return "edit_product";
    } else {
      return "redirect:/";
    }
  }
}
```

9. Go to URI `/edit/{id}`. Now we should be able to edit product.

id	file	in_stock	name	price
2	/images/iphone6s.png	1	Apple iPhone 6s	1210.50
3	/images/ipadpro.png	1	Apple iPad Pro	1310.99
4	/images/samsungs7edge.png	0	Samsung Galaxy S7 Edge	835.92
5	/images/samsungnote7.png	1	Samsung Galaxy Note 7	1035.92
6	/images/htconex9.png	0	HTC One X9	850.20

Edit Product

Name: Apple iPhone 6s

Price: 1213.55

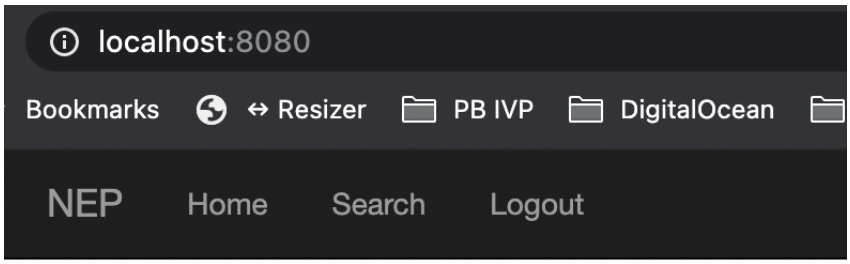
File: /images/iphone6s.png

In Stock: True False

Edit

Edit product successful

id	file	in_stock	name	price
2	/images/iphone6s.png	1	Apple iPhone 6s	1213.55
3	/images/ipadpro.png	1	Apple iPad Pro	1310.99
4	/images/samsungs7edge.png	0	Samsung Galaxy S7 Edge	835.92
5	/images/samsungnote7.png	1	Samsung Galaxy Note 7	1035.92
6	/images/htconex9.png	0	HTC One X9	850.20



Apple iPhone 6s

1213.55

Delete

Edit

> ## ⚠️ Milestone:
>
> Commit the changes made to the project folder with message "Lesson 3.3 - Editing product details in Database".
>
> Hint: Add the files to the staging area first, `git add <files>` and then proceed to commit, `git commit -m "Lesson 3.3 - Editing product details in Database"`.

Lesson 3.4 - Finding product by name

1. So far to find a product, its either uses `findById()` or `findAll()` method of `CrudRepository`. We can define our own method by referring the rules in https://docs.spring.io/spring-data/data-commons/docs/current/reference/html/#repositories.query-methods.query-creation . Lets add an abstract method to find product by name in `ProductRepository`.

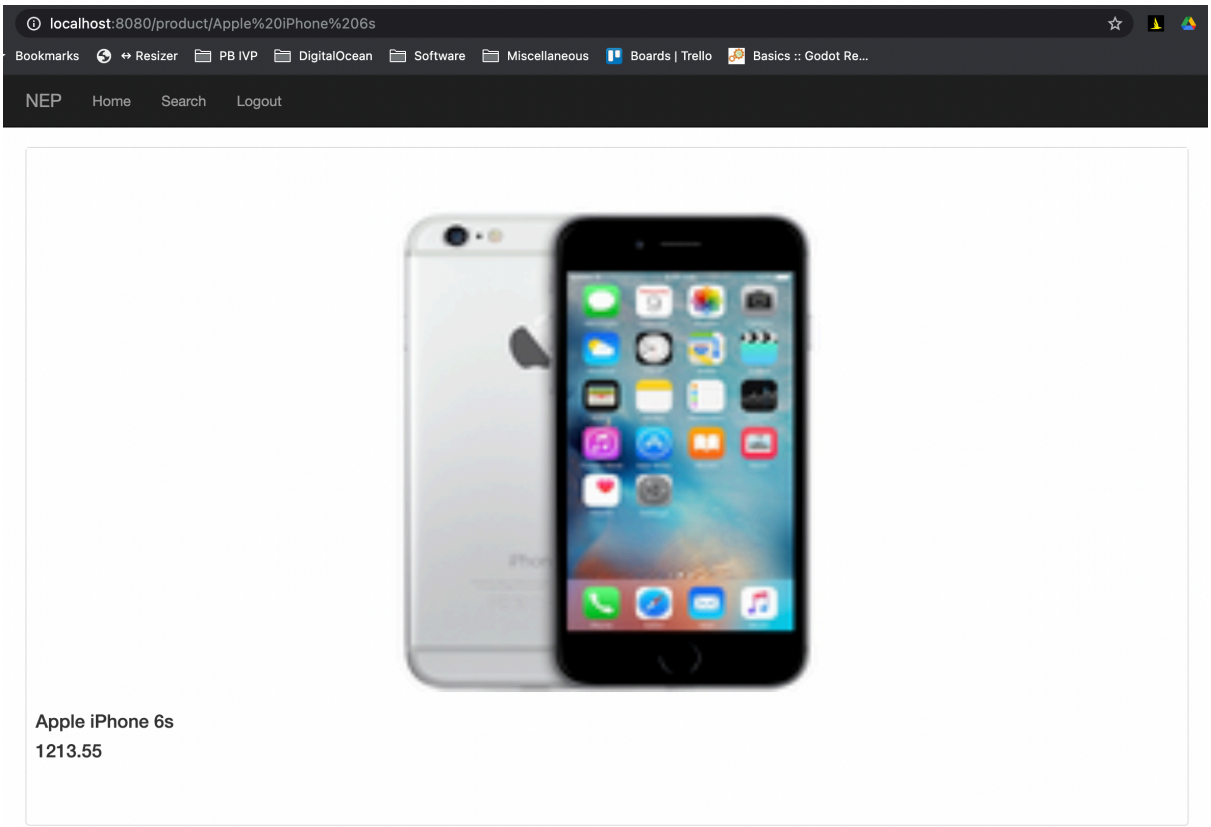
```
package com.nep.practical.data;

import java.util.List;

import org.springframework.data.repository.CrudRepository;
import org.springframework.stereotype.Component;

import com.nep.practical.model.Product;

@Component
public interface ProductRepository extends CrudRepository<Product, Integer>
{

  List<Product> findByName(String name);

}
```

2. Lets fix the method `product` in `HomeController`. Since the method `findByName` is the same, we could just uncomment the codes inside it. In this case, we need the specific detail of the product. So we will retrieve one of it which is index 0 of the list.

```
...
@Controller
public class HomeController {
...

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

```
    Product product = productRepository.findByName(name).get(0);
    modelMap.put("product", product);
    return "product";
  }

...
}
```

3. Go to URI `/product/{name}`. It will display the specific product details.



4. For `search` method in `HomeController` we need to remove the commented part.

```
...
@Controller
public class HomeController {
...

  @RequestMapping(value="/search")
  public String search(@RequestParam(required=false) String productName,
ModelMap modelMap) {

    return "search";
  }

...
}
```

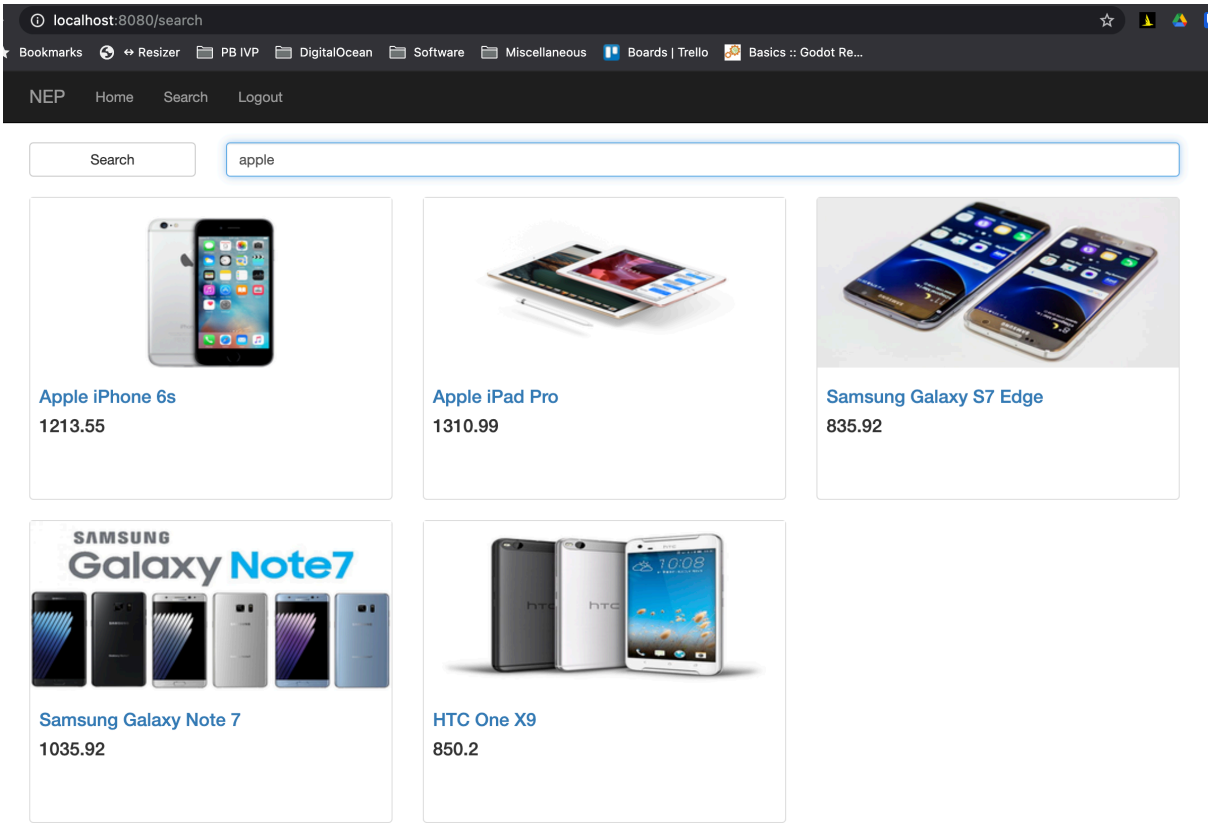
5. To search specific word and get list of products that contain the word. We need to define another method in `ProductRepository`. Since we will use `LIKE` to use wildcard `%` or `_` in MySQL. We need a method that use `LIKE` as well.

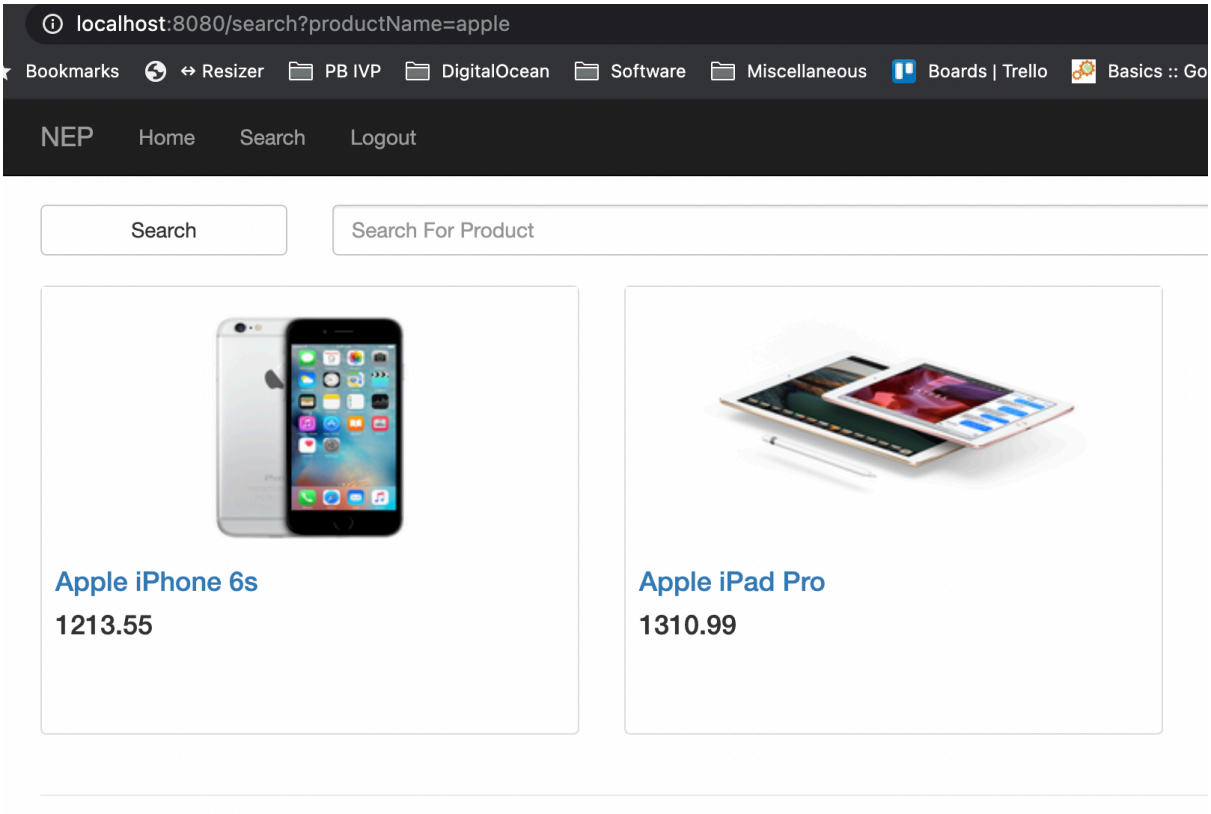
```
package com.nep.practical.data;

import java.util.List;

import org.springframework.data.repository.CrudRepository;
import org.springframework.stereotype.Component;

import com.nep.practical.model.Product;

@Component
public interface ProductRepository extends CrudRepository<Product, Integer>
{

  List<Product> findByName(String name);
  List<Product> findByNameLikeIgnoreCase(String name);

}
```

6. At the beginning of the search, meaning no data sent for finding name. It should show all products. When user type specific word, it should show products that contain the specific word. Lets implement these in `HomeController` method `search`.

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

7. Go to URI `/search`, at the beginning it will show all products. But when user type `apple` in the search textfield and press search. It will only show products that name contain `apple`.





⚠️ Milestone:

Commit the changes made to the project folder with message "Lesson 3.4 - Finding product by name".

Hint: Add the files to the staging area first, `git add <files>` and then proceed to commit, `git commit -m "Lesson 3.4 - Finding product by name"`.