Tutorial 7: Spring boot (Part 2)

[Experimental Objective]
1. Learn how to use thymeleaf template.
2. Learn how to interact between controllers and html page by thymeleaf template.
3. Learn how to implement basic operation including adding, deleting, updating and selecting by Spring boot.
4. Understand the concept of redirect and learn how to use redirect.
5. Have ability to accomplish other models (such as login model) according to the PurchaseRecord model we given to you.

[How to do it]

Thymeleaf

Step1: Adding a dependency
In pom.xml file
We need to add a new dependency
```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-thymeleaf</artifactId>
</dependency>
```

Step2: Update the yml file
In application.yml file, we need to add
```
spring:
  datasource:
    driver-class-name: com.mysql.cj.jdbc.Driver
    url:
jdbc:mysql://localhost:3306/spring_project?useUnicode=true&characterEnc
oding=utf-8&useSSL=false&serverTimezone=GMT
    username: root
    password: 123456
  jpa:
    hibernate:
      ddl-auto: update
    show-sql: true
  thymeleaf:
    mode: HTML
```

1. Build Purchase Record model.

Find all ()

● **HTML**
Create a html file named **PurchaseRecord**List, in the path:
src/resources/templates, and then add a table that describes the purchase record list as follows

```html
<!DOCTYPE html>
<html lang="en"
xmlns:th="http://www.springframework.org/schema/data/jaxb">
<head>
    <meta charset="UTF-8">
    <title>Purchase Record List</title>
</head>
<body>
<div>
    <h2>Purchase Record List</h2>

    <table>
        <thead>
        <tr>
            <th>Id</th>
            <th>Name</th>
            <th>Date</th>
            <th>Money</th>
            <th>Type</th>
            <th>Description</th>
        </tr>
        </thead>
        <tbody>
        <tr th:each="record,it:${purchaseRecordList}">
            <td th:text="${it.count}">1</td>
            <td th:text="${record.username}">Yueming</td>
            <td th:text="${record.date}">2018-9-10</td>
            <td th:text="${record.money}">100</td>
            <td th:switch="${record.type}">
                <span th:case=0>expense</span>
            <span th:case=1>income</span>
            <td th:text="${record.description}">TaoBao</td>
            </td>

        </tr>
        </tbody>
    </table>
</div>
</body>
</html>
```

Using the template of thymeleaf in html can realize the separate between the front-end and the rear-end. You can have a try by clicking the browser button to test the web page.

```
y>

>

<h2>Purchase Record List</h2>
```

```
th:each
th:text
th:case
```

- Controller

After that, in web layer, please create a java class named PurchaseRecordController as follows.

```java
@Controller
public class PurchaseRecordController {
    @Autowired
    private PurchaseRecordService purchaseRecordService;

    @GetMapping("/allRecord")
    public String list(Model model){
        List<PurchaseRecord>  recordList=purchaseRecordService.findAll();
        model.addAttribute("purchaseRecordList",recordList);
        return "PurchaseRecordList";
    }

}
```

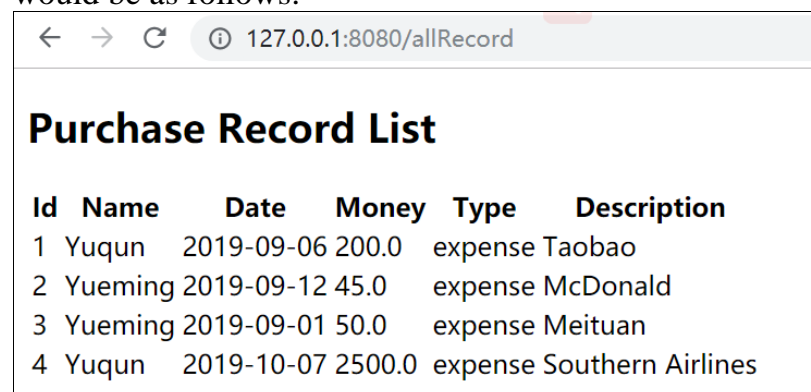After that, you can insert several items into database.
insert into purchase_record (username, date, money, type, description) values ('Yueming','2019-09-01',50,0,'Meituan');
insert into purchase_record (username, date, money, type, description) values ('Yueming','2019-09-12',45,0,'McDonald');
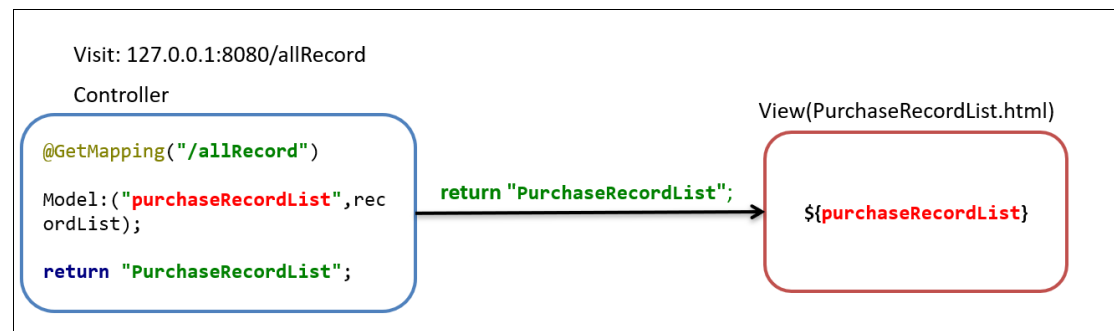insert into purchase_record (username, date, money, type, description) values ('Yuqun','2019-09-06',200,0,'Taobao');
insert into purchase_record (username, date, money, type, description) values ('Yuqun','2019-10-07',2500,0,'Southern Airlines');

Then start the server, and input the URL as 127.0.0.1:8080/allRecord, the page would be as follows:

- The relationship between View (HTML page) and Controller in find all process:

Visit: 127.0.0.1:8080/allRecord

Controller

View(PurchaseRecordList.html)

```
@GetMapping("/allRecord")

Model:("purchaseRecordList",rec
ordList);

return "PurchaseRecordList";
```

return "PurchaseRecordList";

${purchaseRecordList}

2. Add one purchase record

● HTML

Adding following statement just under table in html file

```
<a href="./addPurchaseRecord.html", th:href="@{/insertOneRecord}">add
New</a>
```

In the path: src/resources/templates, please create an html file named
addPurchaseRecord.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Add One Purchase Record</title>
</head>
<body>
<h2>Adding new Purchase Record</h2>
    <form action="/allRecord" method="post">
        <label for="nameId">Name:</label>
        <input type="text" name="username" id="nameId">
        </br>
        <label for="dateId">Date:</label>
        <input type="text" name="date" id="dateId">
        </br>
        <label for="descriptionId">Description:</label>
        <input type="text" name="description" id="descriptionId">
        </br>
        <label for="moneyId">Money:</label>
        <input type="text" name="money" id="moneyId">
        </br>
        <label for="typeId">Type:</label>
        <select name="status" id="typeId">
            <option value="0">expense</option>
            <option value="1">income</option>
        </select>
        </br>
        <button type="submit">submit</button>
    </form>
```
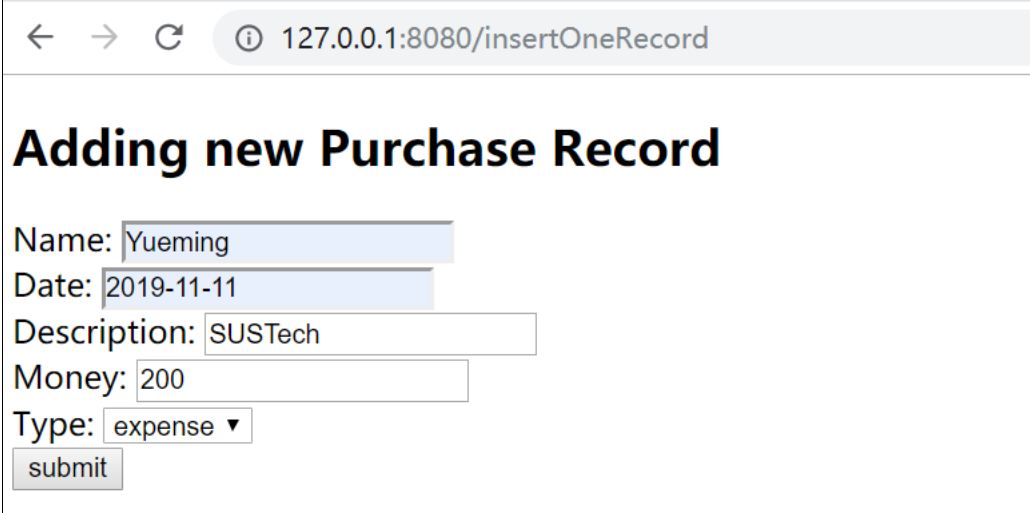
zhuym@sustech.edu.cn

```
</body>
</html>
```

● Controller

In **PurchaseRecordController**, adding following two methods.

```
@GetMapping("/insertOneRecord")
public String addPurchaseRecord (){
    return "addPurchaseRecord";
}

@PostMapping("/allRecord")
public String saveNew(PurchaseRecord purchaseRecord){
    purchaseRecordService.save(purchaseRecord);
    return "redirect:/allRecord";
}
```
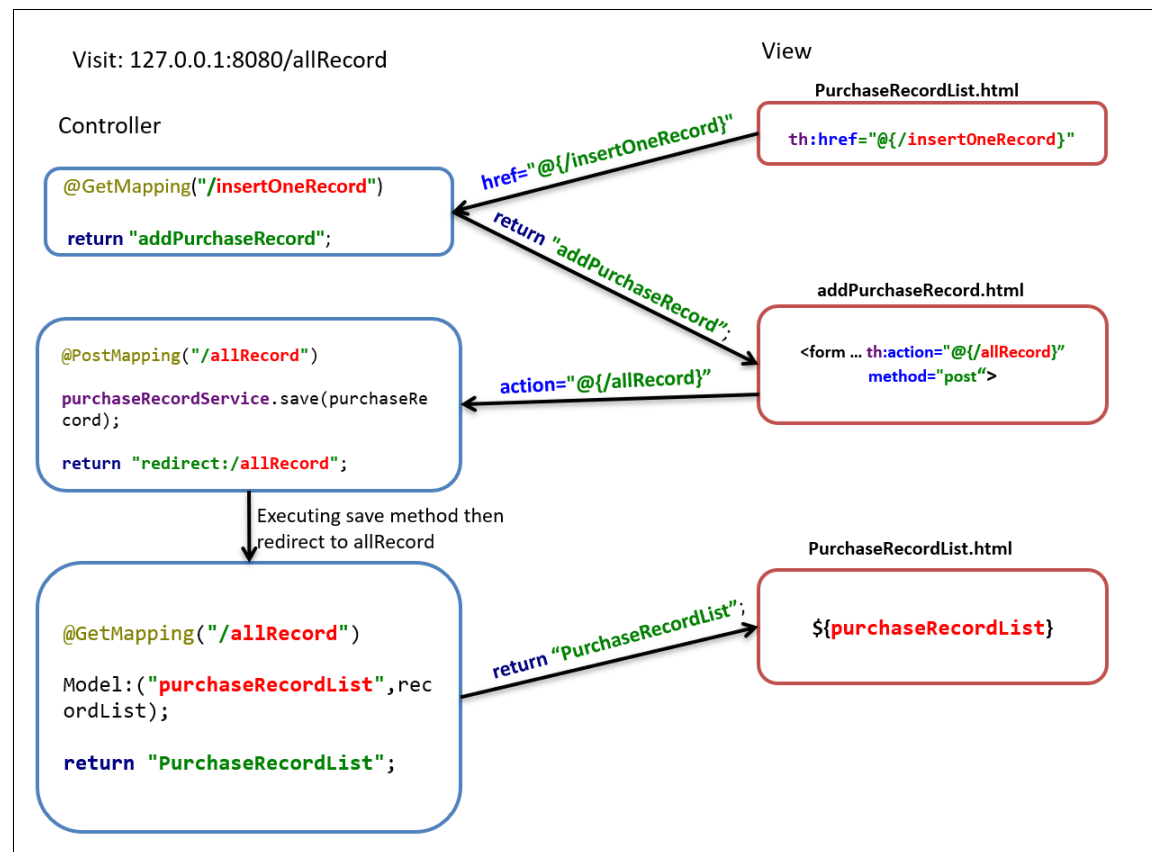
The result will be as follows, and we can fill the form.

● The relationship between View (HTML page) and Controller in find all process:

3. Update one record.

- PurchaseRecordList.html
    1. Adding <**th**>Operation</**th**> at the end of <tr>

    2. Adding a update link in each row:

    ```
    <td>
        <a href="./addPurchaseRecord.html"
    th:href="@{/findOneRecord{id}(id=${record.id})}">update</a>
    </td>
    ```

3. Adding **<p th:text="${msg}"`>** Here is to show message**</p>** just below the link of add New.

```
    <a href="#", th:href="@{/insertOneRecord}">add New</a>
    <p th:text="${msg}"> Here is to show message</p>
</div>
```

● PurchaseRecordService

In PurchaseRecordService, we need to add a new method **findById**, which can return an object of PurchasseRecord according to its id.

```java
public PurchaseRecord findById(long id);
```

● PurchaseRecordServiceImpl

Implements the abstract method findById
```java
@Override
public PurchaseRecord findById(long id) {
    return purchaseRecordRepository.findById(id).get();
}
```

● PurchaseRecordController

1. Adding a method `findPurchaseRecord`, which can get an id from `th:href="@{/findOneRecord{id}(id=${record.id})}"` and then we can pass the id as parameter to find the corresponding object. The method has an annotation `@PathVariable long id`, which means `id` is an attribute we get from web page and the value of which is **${record.id}**.

```java
@GetMapping("/findOneRecord{id}")
public String findPurchaseRecord(@PathVariable long id, Model model){
    PurchaseRecord purchaseRecord=purchaseRecordService.findById(id);
    model.addAttribute("purchaseRecord",purchaseRecord);
    return "addPurchaseRecord";
}
```

2. After that, modify `saveNew` method. Adding a RedirectArributes in saveNew method for passing a flash attribute from controller to html file.
```java
@PostMapping("/allRecord")
public String saveNew(PurchaseRecord purchaseRecord,
RedirectAttributes attributes){
    purchaseRecordService.save(purchaseRecord);
    attributes.addFlashAttribute("msg","Update has been saved");
    return "redirect:/allRecord";
}
```

3. Modify method `addPurchaseRecord` by passing an object as an attribute, because the we plan to modified **addPurchaseRecord.html** file (we would modify the html file later). If we add `th:object="${purchaseRecord}"` in the form tag, it needs to pass an object of PurchaseRecord, otherwise, it

can only be accepted in update operation but will raise an Exception when doing add operation.

```java
@GetMapping("/insertOneRecord")
public String addPurchaseRecord(Model model){
    model.addAttribute("purchaseRecord",new PurchaseRecord());
    return "addPurchaseRecord";
}
```

- addPurchaseRecord.html

Here is the process for passing an object from controller to html file by using the tag `th:object`.

1. Firstly, we need to set an attribute in the Model as following type:
   ```java
   model.addAttribute("purchaseRecord", new PurchaseRecord());
   ```
2. Modify .html file, and adding `th:action="@{/allRecord}"` `th:object="${purchaseRecord}"` in the form tag, then we can receive the field value of purchaseRecord by `th:field="*{field name }"`

We modify the Html file as follows:

```html
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.w3.org/1999/xhtml">
<head>
    <meta charset="UTF-8">
    <title>Add One Purchase Record</title>
</head>
<body>
<h2>Adding new Purchase Record</h2>
    <form th:action="@{/allRecord}" th:object="${purchaseRecord}" method="post">
        <input type="hidden" th:value="*{id}" name="id">
        <label for="nameId">Name:</label>
        <input type="text" name="username" id="nameId" th:field="*{username}" >
        <br>
        <label for="dateId">Date:</label>
        <input type="text" name="date" id="dateId" th:field="*{date}">
        <br>
        <label for="descriptionId">Description:</label>
        <input type="text" name="description" id="descriptionId" th:field="*{description}">
        <br>
        <label for="moneyId">Money:</label>
        <input type="text" name="money" id="moneyId" th:field="*{money}">
        <br>
        <label for="typeId">Type:</label>
        <select name="status" id="typeId" th:field="*{type}">
            <option value="0" th:selected="(*{type}=='0')">expense</option>
            <option value="1" th:selected="(*{type}=='1')">income</option>
        </select>
        <br>
        <button type="submit">submit</button>
```
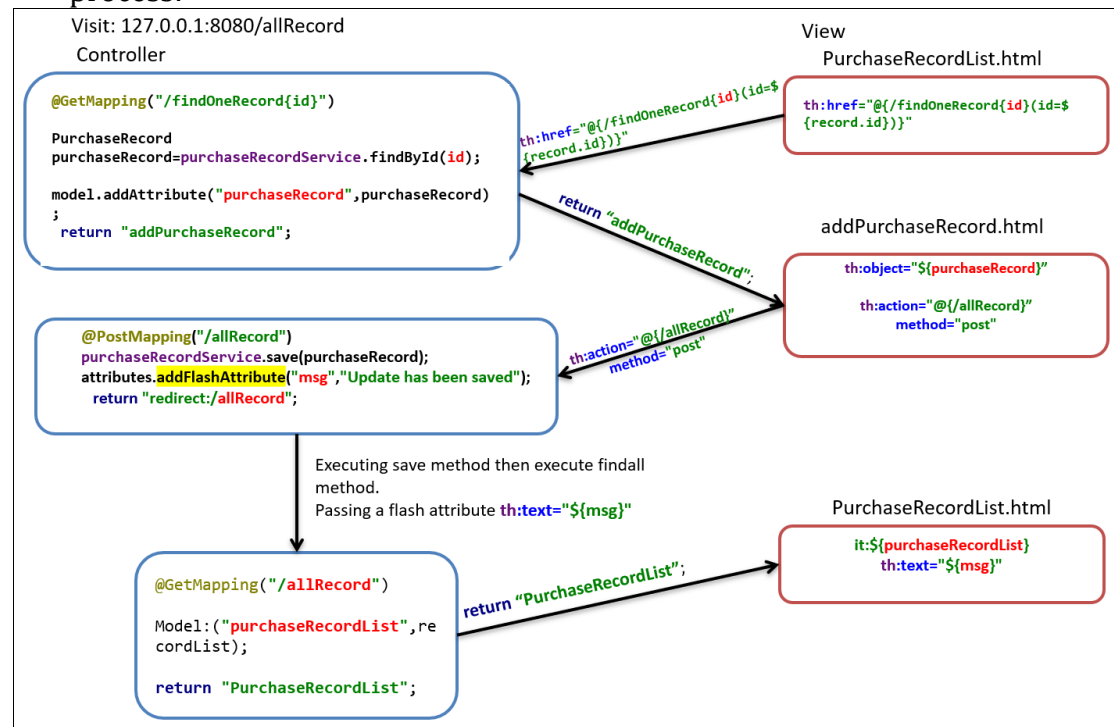
zhuym@sustech.edu.cn

```
        </form>

</body>
</html>
```

- The relationship between View (HTML page) and Controller in update process:

4. Delete one PurchaseRecord.

- PurchaseRecordList.html
    1. Adding <**th**>Operation</**th**> at the end of <tr>

    2. Adding a button in each row:
Adding a <td> at the bottom of <tr></tr>

```
<td>
    <form th:action="@{/deleteOneRecord{id}(id=${record.id})}"
method="get">
        <button type="submit">Delete</button>
    </form>
</td>
```
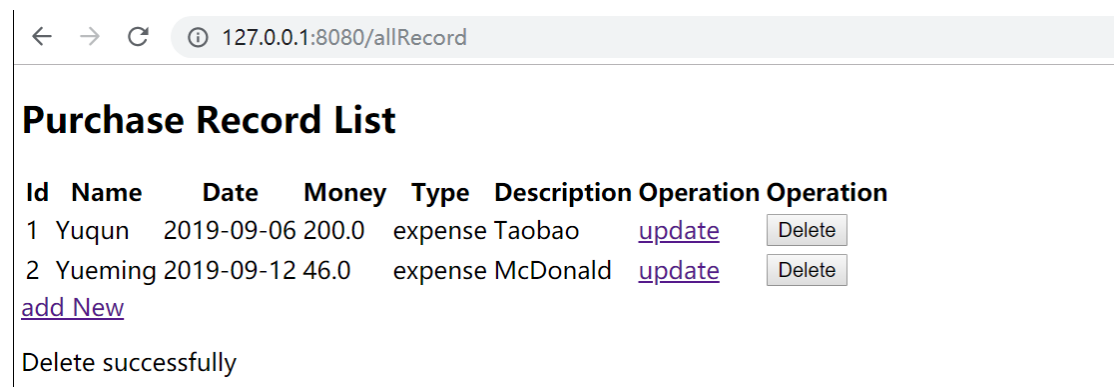
- PurchaseRecordController

It is similar to update a row. We add a method **deleteRecord**, which can delete a row according to the @PathVariable id.

```java
@GetMapping("/deleteOneRecord{id}")
public String deletePurchaseRecord(@PathVariable long
id,RedirectAttributes attributes){
    purchaseRecordService.deleteById(id);
    attributes.addFlashAttribute("msg","Delete successfully");
    return "redirect:/allRecord";
}
```

When we restart the server, the result would as following graph

User model:

Build Entity Class

- In domain package, create a class named **User** and adding following attributes and the constructors. Don't forget to add getter and setter methods of all private fields:

```java
@Entity
public class User {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;
    @Column(unique=true)
    private String username;
    private String password;

    public User() {
    }

    public User(String username, String password) {
        this.username = username;
        this.password = password;
    }
}
```

zhuym@sustech.edu.cn

● You can create another class named **UserForm**, which needn't to be persistent in database, only for passing object from register form. Don't forget to add getter and setter methods of all private fields

```java
public class UserForm {
    @NotBlank(message = "Username shouldn't be null")
    private String username;
    @Length(min = 6, message = "Password need at least 6 bits")
    private String password;
    @NotBlank(message = "Confirm password shouldn't be null")
    private String confirmPassword;
}
```

Build User Repository

In the repository layer, please create an interface as follows: I have added those two methods, which may be used in following design.

```java
public interface UserRepository extends JpaRepository<User, Integer> {
    public User findUserByUsername(String username);
    public User findUserByUsernameAndPassword(String username, String password);
}
```

Build User Service

In the service layer, please create an interface named **UserService** and a class named **UserServiceImpl**.

```java
public interface UserService {
    public void save(User user);
    public boolean checkLogin(User user);
    public User findByUsername(String username);
}
```

```java
@Service
public class UserServiceImpl implements UserService {
    @Autowired
    private UserRepository userRepository;

    @Override
    public void save(User user) {
        userRepository.save(user);
    }

    @Override
    public boolean checkLogin(User user) {
        User u =
userRepository.findUserByUsernameAndPassword(user.getUsername(),
user.getPassword());
        return u != null;
    }

    @Override
    public User findByUsername(String username) {
        return userRepository.findUserByUsername(username);
```

```
        }
}
```

Build User Controller

Create a class named **UserController**, which is being used for interacting with html file. Here we only exercise the register part, then you can add two methods for register as follows:

When we visit from url: 127.0.0.1:8080/register, it will jump to the register.html.

```
@GetMapping("/register")
public String registerPage() {
    return "register";
}
```

@Valid is for checking the whether the Bean object passing from web is correct. Here the Bean object is a type of UserForm, and we had added some annotations for checking such as @NotBlank

BindingResult can record the error of field, from which we can get the error messages having designed.

The following method is not the whole checking process, which only check the annotations defined in **UserForm**, and you need to add other necessary logic to complete it, such as checking whether two passwords are same, checking UK constraint for username, etc.

```
@RequestMapping(params = "submit", method = RequestMethod.POST)
@PostMapping("/register")
public String registerUser(@Valid UserForm userForm, BindingResult
result, RedirectAttributes attributes) {
    attributes.addFlashAttribute("username", userForm.getUsername());
    if (result.hasErrors()) {
        List<FieldError> errors = result.getFieldErrors();
        attributes.addFlashAttribute("errorMsg",
errors.get(0).getDefaultMessage());
        return "redirect:/register";
    } else  {
        return "redirect:/login ";
        }
}
```

Html Files

In template, create a html file named register, and another html file named login.

- register.html

```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.w3.org/1999/xhtml">
<head>
    <meta charset="UTF-8">
    <title>Register</title>
</head>
<body>
<h2>Register A User</h2>
```

```html
<form th:action="register" method="post">
    <label for="nameId">Name:</label>
    <input type="text" name="username" id="nameId"
th:value="${username}"><br>
    <label for="passwordId">Password:</label>
    <input type="password" name="password" id="passwordId"><br>
    <label for="confirmPasswordId">Confirm Password:</label>
    <input type="password" name="confirmPassword"
id="confirmPasswordId"><br>
    <input type="submit" name="submit" value="Submit">
    <input type="submit" name="back" value="Back">
    <input type="submit" name="clear" value="Clear">
</form>
<p th:text="${errorMsg}">Error message is here</p>
</body>
</html>
```

● login.html
```html
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.w3.org/1999/xhtml">
<head>
    <meta charset="UTF-8">
    <title>Login</title>
</head>
<body>
<h2>Please Login:</h2>
    <form method="post">
        <label for="nameId">Name:</label>
        <input type="text" name="username" id="nameId"
th:value="${username}">
        <label for="passwordId">Password:</label>
        <input type="password" name="password" id="passwordId">
        <button type="submit">Login</button>
    </form>
</body>
</html>
```

Your work:
1. You need to add additional methods in UserController.java to complete this project especially in login process.
2. Those two html files, especially Login.html, is only a template, you need to add necessary elements of thymeleaf template to get attributes from controller.

Further Learning and resource:
https://www.thymeleaf.org/doc/tutorials/3.0/usingthymeleaf.html#standard-expression-syntax