Java Web Lab Work

---

1) Project skeleton introduction
Take a look at pom.xml

For including Bootstrap and jQuery files to the project maven dependencies is used as web jars packages.

```
<dependency>
    <groupId>org.webjars</groupId>
    <artifactId>bootstrap</artifactId>
    <version>3.2.0</version>
    <exclusions>
        <exclusion>
            <groupId>org.webjars</groupId>
            <artifactId>jquery</artifactId>
        </exclusion>
    </exclusions>
</dependency>
<dependency>
    <groupId>org.webjars</groupId>
    <artifactId>jquery</artifactId>
    <version>2.1.1</version>
</dependency>
```

Also as a review take a look at spring and jackson dependency

Another resources we'll use are placed in ${project.dit}/webapp/WEB-INF/resources/core/

In ${project.dit}/webapp/WEB-INF/mvc-web-dispatcher-servlet.xml to let resources be downloaded without accessing controller we have next lines

```
<mvc:resources mapping="/resources/**" location="/resources/" />
<mvc:resources mapping="/webjars/**" location="classpath:/META-INF/resources/webjars/" />
```

Project contains next packages:
- controller - contains project controller
- exceptions - contains custom exceptions
- model - contains inner DTO's
- service - contains project services

2) Overview pub.jsp page

Following lines produce resource import into page

In header:

```
<head>
    <title>Java Pub</title>
    <meta name="viewport" content="initial-scale=1, maximum-scale=1">
    <link rel='stylesheet' href='webjars/bootstrap/3.2.0/css/bootstrap.min.css'>
    <link rel="stylesheet" href="/resources/core/my-style.css">
</head>
```

In page body:

```
<script type="text/javascript" src="webjars/jquery/2.1.1/jquery.min.js"></script>
<script type="text/javascript" src="webjars/bootstrap/3.2.0/js/bootstrap.min.js"></script>
<script type="text/javascript" src="/resources/core/my-script.js"></script>
```

Tabs are created with Twitter Bootstrap styling and code looks like

```
<div class="row">
    <div class="col-lg-offset-1 col-lg-10">
        <ul class="nav nav-tabs">
            <li role="presentation" class="active"><a href="#" id="order-tab">Make an order</a></li>
            <li role="presentation"><a href="#" id="status-tab">Current beer status</a></li>
            <li role="presentation"><a href="#" id="contact-tab">Contact information</a></li>
        </ul>
    </div>
</div>
```

Each tab connected to concrete tab-link and implemented with div block

```
<div class="top-buffer tab-panel" id="order-panel"...>
<div class="top-buffer tab-panel hidden" id="status-panel"...>
<div class="top-buffer tab-panel hidden" id="contact-panel"...>
```

3) Task 1. Implement tab switching with JavaScript functions

In *my-style.css* you already have class, that can make component hidden on attaching:

```
.hidden {
    display: none;
}
```

So the task is to write JavaScript function, that will add this class to tabs need to be hidden and remove from that need to be shown. Also don't forget to save tab buttons style (selected tab should have class *active*)

Hints:

- To add class to element you can use

```
$('.tab-panel').addClass("hidden");
```

- To delete class

```
$('.nav-tabs li').removeClass("active");
```

Functions can be written in *my-script.js* file within *$(document).ready(function(){}*

4) Task 2. Add initial data to the page on load.

To not execute additional AJAX call from page it's possible to add some starter static information on JSP processing. In current application we have several places to show information from the server: fill in dropdown list with beer kinds on the first tab; fill in current bar status on the second tab; fill contact information on the third tab

First of all let's take a look at server part. BarService class already has methods that could return required data. These methods took data from class variables, so add some default values for them to see that data on the page.
The next step will be also on server side, in MainController class. The way to pass data to UI is to attach data to response:

```
@RequestMapping(value = "/pub", method = RequestMethod.GET)
public String getPub(Model model) {
    model.addAttribute("beerKinds", bar.getBeerKinds());
    model.addAttribute("contactInformation", bar.getContactInformation());
    model.addAttribute("barStatus", bar.getBarStatus());
    return "pub";
}
```

Second, is how to add this value to *pub.jsp*. First of all we should import tag lib for some automation feature, and for all entities we have to use. Put this data to the *head* tag of *pub.jsp*:

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<jsp:useBean id="beerKinds" scope="request" type="java.util.List"/>
<jsp:useBean id="contactInformation" scope="request" type="ua.skillsup.practice.rest.model.ContactInformation"/>
<jsp:useBean id="barStatus" scope="request" type="ua.skillsup.practice.rest.model.BarStatus"/>
```

On the first tab to fill dropdown with beer kinds we can use forEach iteration:

```
<select class="form-control" id="kindSelect">
    <c:forEach items="${beerKinds}" var="kind">
        <option>${kind}</option>
    </c:forEach>
</select>
```

In case of the second tab, where data just in DTO and should be put to the table the next construction can be used:

```
<tr>
    <td>Dark</td>
    <td id="darkAmount">${barStatus.glassesOfDark}</td>
</tr>
```

The same is for the third tab:

```
<li class="list-group-item">Country : ${contactInformation.country}</li>
<li class="list-group-item">City : ${contactInformation.city}</li>
```

5) Task 3. Implement resource management with Spring resource handling

So to not hardcode values like contact information etc. in class, its better to have them placed in separate properties file. For example in resources we have bar.properties file, so put there data like:

```
contact.country=Ukraine
contact.city=Dnipropetrovsk
contact.phone.number=+380939218051
contact.address=Barykadnaya street, 2a
```

And in BarService on internal fields add following annotations:

```
@Value("${contact.country}")
private String country;
```

6) Task 4. Execute client's order

First of all add method in controller to receive clients data and prepare response.

```
@ResponseBody
@RequestMapping(value = "/clientOrder", method = RequestMethod.POST)
public ResponseMessage executeClientOrder(@RequestBody ClientOrder clientOrder) {
    return ResponseMessage.okMessage(bar.executeClientOrder(clientOrder));
}
```

Take a look, that for some reasons exception can be performed during request processing. For correct exception handling and possibility not to show to client error page with exception stack trace in Spring MVC exists following mechanism:

```
@ResponseBody
@ExceptionHandler(Exception.class)
public ResponseMessage handleException(Exception e) {
    return ResponseMessage.errorMessage(e.getMessage());
}
```

To submit this order from client side you should use *ajax* request with JavaScript, for example:

```
$('#submit-order').click(function () {
    var kind = $('#kindSelect option:selected').text();
    var count = $('#count').val();
    $.ajax({
        type: "POST",
        url: '/clientOrder',
        contentType: "application/json; charset=utf-8",
        dataType: "json",
        data: '{"kind": "' + kind + '", "count": "' + count + '"}',
        success: function(data) {
            if (data.status === 'OK') {
                updateStatus(data.barStatus);
                displayClientOrderSuccess();
            } else {
                displayError(data.errorMessage);
            }
    }});
});
```

The idea is to send from backend current status of application and with this data update second tab - implement updateStatus() method. Also for displaying error or success after client action use concrete text block:

```
<div class="col-lg-offset-3 col-lg-6">
    <p id="client-order-response" class="text-centred"></p>
</div>
```

Use bootstrap styles for showing type of the message: **bg-success** and **bg-danger**.

7) Task 5. Implement functionality for Refresh button from second tab

Add functionality to Refresh button, that will send GET request to server and receive actual application status. Use this status to update data on the second tab.

No example this time :)

8) Task 6. Implement functionality for Refill Bar request on modal from the third tab.

Implement function for Request button from Refill modal. On button click POST ajax request should be send to server side and beer's count should be updated with values, retrieved from modal form.

9) Task 7. Add client-side checks for input values.