COMP S380F Web Applications: Design and Development
Lab 8: Spring MVC Web Framework

We will use Spring MVC to implement the lab exercises in Lab 7. The following topics are covered:

- Using @GetMapping and @PostMapping
- Accessing request and session objects in Controller Methods
- URL redirect the String prefix "redirect:"

Task 1: Setting up for Spring MVC

Download the branch "**lab07ans**" of GitHub repository "**https://github.com/cskeith/380_2024.git**".

1. In **settings.gradle**, make the following update:

```
rootProject.name = "Lab08"
```

2. In **build.gradle**, update the **sourceCompatibility** and **targetComptaibility** to 17, which are required for using the latest version of Spring MVC. Then, add the property **springVersion** and dependency for Spring MVC. **Reload the project in the Gradle window**.

```
ext {
    springVersion = '6.1.4'
    …
}

sourceCompatibility = '17'
targetCompatibility = '17'

…
dependencies {
    …
    implementation "org.springframework:spring-webmvc:${springVersion}"
}
```

3. In **web.xml**, add the definition of the **DispatcherServlet**:

```
<servlet>
    <servlet-name>dispatcher</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
    <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
    <servlet-name>dispatcher</servlet-name>
    <url-pattern>/</url-pattern>
</servlet-mapping>
```

4. As the **DispatcherServlet** is named **dispatcher**, we need a Spring XML configuration file "/WEB-INF/**dispatcher**-servlet.xml" to set up the **Spring application context**:
    - Right-click on "**WEB-INF**", and select **New** > **XML Configuration File** > **Spring Config** .
    - Set its file name to "**dispatcher-servlet**".

- Add the followings (add the Spring namespace **context** and **mvc** using Alt + Enter):

```
<context:component-scan base-package="hkmu.comps380f" />
<mvc:annotation-driven />

<bean id="jspViewResolver"
      class="org.springframework.web.servlet.view.InternalResourceViewResolver">
    <property name="prefix" value="/WEB-INF/jsp/" />
    <property name="suffix" value=".jsp" />
</bean>
```

- `<context:component-scan base-package="..." />` tells Spring to automatically create Spring beans (e.g., controllers) in the specified packages.
- `<mvc:annotation-driven />` declares explicit support for annotation-driven Spring MVC, e.g., `@Controller` for defining a controller class, and `@GetMapping` for handler mapping.
- The Spring bean `jspViewResolver` resolves a view name to a view implementation.

5. In the package **hkmu.comps380f**, create the controller **MyController**, which has a method mapping the URL pattern "/" to show the session activity JSP page for HTTP GET requests, using `@GetMapping`. Note that we need to delete **index.jsp** to make the code work.

```
@Controller
public class MyController {

    @GetMapping("/")
    public String index() {
        return "viewSessionActivity";
    }
}
```

Task 2: Revisiting the exercise on multi-value request parameter

We will move the code in the servlet **MultiValueParameterServlet** to the controller **MyController**.

1. Delete the servlet **MultiValueParameterServlet.**

2. Add the following controller methods to the controller **MyController**. Note that `@GetMapping` is used for handling HTTP GET methods, while `@PostMapping` is used for handling HTTP POST methods.

```
@GetMapping("/checkboxes")
public String multiValueForm() {
    return "MultiValueForm";
}

@PostMapping("/checkboxes")
public String multiValueResult() {
    return "MultiValueResult";
}
```

Task 3: Revisiting the exercise on tracking session activity

We will move the code in the servlet **ActivityServlet** to the controller **MyController**.

1. Add the following controller method to the controller **MyController**. Note that Spring controller provides flexible method arguments, and you can add a method argument (e.g., the session object in red below) when you need to access it within the controller method.

```
@GetMapping("/do/*")
public String recordSessionActivity(HttpServletRequest request, HttpSession session) {
    if (session.getAttribute("activity") == null)
        session.setAttribute("activity", new CopyOnWriteArrayList<PageVisit>());

    @SuppressWarnings("unchecked")
    CopyOnWriteArrayList<PageVisit> visits
            = (CopyOnWriteArrayList<PageVisit>) session.getAttribute("activity");
    if (!visits.isEmpty()) {
        PageVisit last = visits.get(visits.size() - 1);
        last.setLeftTimestamp(System.currentTimeMillis());
    }

    PageVisit now = new PageVisit();
    now.setEnteredTimestamp(System.currentTimeMillis());
    if (request.getQueryString() == null) now.setRequest(request.getRequestURL().toString());
    else now.setRequest(request.getRequestURL() + "?" + request.getQueryString());
    try {
        now.setIpAddress(InetAddress.getByName(request.getRemoteAddr()));
    } catch (UnknownHostException e) {
        e.printStackTrace();
    }
    visits.add(now);
    return "viewSessionActivity";
}
```

2. Delete the servlet **ActivityServlet.**

Task 4: Revisiting the exercise on shopping cart

We will move the code in the servlet **StoreServlet** to the controller **MyController**.

1. Add the following instance variable, constructor, and controller method to the controller **MyController**.

```
@Controller
public class MyController {

    private final Map<Integer, String> products = new ConcurrentHashMap<>();

    public MyController() {
        this.products.put(1, "Sandpaper");
        this.products.put(2, "Nails");
        this.products.put(3, "Glue");
        this.products.put(4, "Paint");
        this.products.put(5, "Tape");
    }

   ...
```

```
@GetMapping("/shop")
public String shop(HttpServletRequest request, HttpSession session) {
    String action = request.getParameter("action");
    if (action == null)
        action = "browse";

    switch (action) {
        case "addToCart":
            return this.addToCart(request, session);
        case "viewCart":
            return this.viewCart(request);
        case "emptyCart":
            return this.emptyCart(session);
        case "browse":
        default:
            return this.browse(request);
    }
}
```

2. We update the **addToCart** method, as follows. Here, instead of returning a view name, we can use the prefix `redirect:` on a URL pattern to redirect the user to a URL.

```
private String addToCart(HttpServletRequest request, HttpSession session) {
    int productId;
    try {
        productId = Integer.parseInt(request.getParameter("productId"));
    } catch (Exception e) {
        return "redirect:/shop";
    }

    if (session.getAttribute("cart") == null)
        session.setAttribute("cart", new ConcurrentHashMap<>());

    @SuppressWarnings("unchecked")
    Map<Integer, Integer> cart
            = (Map<Integer, Integer>) session.getAttribute("cart");
    if (!cart.containsKey(productId))
        cart.put(productId, 0);
    cart.put(productId, cart.get(productId) + 1);

    return "redirect:/shop?action=viewCart";
}
```

3. Update the methods `viewCart`, `browse`, and `emptyCart` similarly.

4. Delete the servlet **StoreServlet.**