As per Curriculam

1. Spring MVC Architecture

- **Explanation**: Spring MVC follows the **Model-View-Controller pattern**. The architecture involves a **DispatcherServlet** handling requests, which then forwards them to **controllers**, and eventually returns a **view** to the user.

2. Request Life Cycle in Spring MVC

- **Steps**:
    1. The user sends a request to the server.
    2. The DispatcherServlet intercepts the request.
    3. It consults HandlerMapping to find the correct controller.
    4. The controller processes the request and returns a ModelAndView object.
    5. The ViewResolver resolves the view.
    6. The response is rendered back to the user.

3. Introduction to DispatcherServlet

- **Code Example** (web.xml configuration):

```
<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
                             http://xmlns.jcp.org/xml/ns/javaee/web-
app_4_0.xsd"
         version="4.0">

    <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>
```

```
</web-app>
```

4. WebApplicationContext and Context Hierarchy

- **Explanation**: **WebApplicationContext** is a specialized version of **ApplicationContext** used in **Spring MVC**. It is loaded by **DispatcherServlet** and manages web-related beans like controllers and view resolvers.
- **Code Example (dispatcher-servlet.xml):**

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:context="http://www.springframework.org/schema/context"
       xmlns:mvc="http://www.springframework.org/schema/mvc"
       xsi:schemaLocation="
         http://www.springframework.org/schema/beans
         https://www.springframework.org/schema/beans/spring-
beans.xsd
         http://www.springframework.org/schema/context
         https://www.springframework.org/schema/context/spring-
context.xsd
         http://www.springframework.org/schema/mvc
         https://www.springframework.org/schema/mvc/spring-mvc.xsd">

    <!-- Enable Spring MVC Annotations -->
    <mvc:annotation-driven />

    <!-- Component Scanning -->
    <context:component-scan base-package="com.coforge.controller" />

    <!-- View Resolver Configuration -->
    <bean id="viewResolver"
class="org.springframework.web.servlet.view.InternalResourceViewReso
lver">
        <property name="prefix" value="/WEB-INF/views/" />
        <property name="suffix" value=".jsp" />
    </bean>

    <!-- Register the interceptor with Spring MVC -->
    <mvc:interceptors>
        <mvc:interceptor>
            <mvc:mapping path="/*" />
            <bean class="com.coforge.interceptor.CustomInterceptor"
/>
        </mvc:interceptor>
    </mvc:interceptors>


</beans>
```

5. Defining a Controller with @Controller

- **Code Example**:

```
// Home page mapping
    @GetMapping("/")
    public String home() {
        return "index";

    }
```

6. Using @RequestMapping, @RequestParam, @ModelAttribute

- **Code Example**:

```
// Get user by ID
    @GetMapping("/user")
    public String getUser(@RequestParam("id") String userId,
Model model) {
        model.addAttribute("userId", userId);
        return "user"; // Returns user.jsp
    }

    // Add user handling (for the form in index.jsp)
    @PostMapping("/addUser")
    public String addUser(@RequestParam("name") String
userName, Model model) {
        model.addAttribute("userName", userName);
        return "userAdded";

    }
```

7. Using @RequestBody, @PathVariable, @CookieValue

- **Code Example**:

```
@RestController
public class ApiController {
   @PostMapping("/createUser")
   public String createUser(@RequestBody User user) {
      return "User created: " + user.getName();
   }
```

```
    @GetMapping("/user/{id}")
    public String getUserById(@PathVariable("id") String id) {
        return "User ID: " + id;
    }

    @GetMapping("/cookie")
    public String getCookie(@CookieValue("sessionId") String
sessionId) {
        return "Session ID: " + sessionId;
    }
}
```

8. Using @RequestHeader, @ResponseBody, @ModelAttribute

- **Code Example**:

```
@RestController
public class HeaderController {
    @GetMapping("/headers")
    public String headers(@RequestHeader("User-Agent") String
userAgent) {
        return "User-Agent: " + userAgent;
    }

    @GetMapping("/response")
    public @ResponseBody String responseBody() {
        return "This is a response body";
    }
}
```

9. Handler Mapping

- **Explanation**: Spring MVC uses HandlerMapping to map requests to their respective controllers. By default, Spring uses RequestMappingHandlerMapping.

10. Interceptors in Spring MVC

- **Explanation**: Interceptors in Spring MVC can pre-process (before the controller) and post-process (after the controller) requests.
- **Code Example**:

```
package com.coforge.interceptor;



import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.springframework.web.servlet.HandlerInterceptor;
import org.springframework.web.servlet.ModelAndView;

public class CustomInterceptor implements HandlerInterceptor {

    @Override
    public boolean preHandle(HttpServletRequest request,
HttpServletResponse response, Object handler) {
        System.out.println("Pre Handle method: Before
Controller execution");
        // Check if a user is logged in, for example
        // If not, you can send a redirect to a login page
        return true;
    }

    @Override
    public void postHandle(HttpServletRequest request,
HttpServletResponse response, Object handler, ModelAndView
modelAndView) {
        System.out.println("Post Handle method: After
Controller execution, before View rendering");
        // Modify the response or add common attributes to the
model
    }

    @Override
    public void afterCompletion(HttpServletRequest request,
HttpServletResponse response, Object handler, Exception ex) {
        System.out.println("After Completion method: After
View rendering");
        // Cleanup resources if needed
    }
}
```

11. Request Mapping Attributes (value, method, params, headers)

- **Code Example**:

```
@RequestMapping(value = "/users", method =
RequestMethod.GET, params = "version=1", headers =
"Accept=application/json")
public String getUsers() {
   return "usersList";
}
```

12. Life Cycle of the Interceptor: Methods

- **Methods**:
  1. **preHandle():** Called before the controller method.
  2. **postHandle():** Called after the controller method but before **the view is rendered.**
  3. **afterCompletion():** Called after the view is rendered.

13. Implementing Handler Interceptor

- **Code Example**:

```
package com.coforge.interceptor;


import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.springframework.web.servlet.HandlerInterceptor;
import org.springframework.web.servlet.ModelAndView;

public class CustomInterceptor implements HandlerInterceptor {

    @Override
    public boolean preHandle(HttpServletRequest request,
HttpServletResponse response, Object handler) {
```

```
        System.out.println("Pre Handle method: Before
Controller execution");
        // Check if a user is logged in, for example
        // If not, you can send a redirect to a login page
        return true;
    }

    @Override
    public void postHandle(HttpServletRequest request,
HttpServletResponse response, Object handler, ModelAndView
modelAndView) {
        System.out.println("Post Handle method: After
Controller execution, before View rendering");
        // Modify the response or add common attributes to the
model
    }

    @Override
    public void afterCompletion(HttpServletRequest request,
HttpServletResponse response, Object handler, Exception ex) {
        System.out.println("After Completion method: After
View rendering");
        // Cleanup resources if needed
    }
}
```

14. Registering Interceptor and Ordering

- **Configuration Example**:

```
@Configuration
public class WebConfig implements WebMvcConfigurer {
   @Override
   public void addInterceptors(InterceptorRegistry registry) {
      registry.addInterceptor(new CustomInterceptor()).order(1);
   }
}
```

```
<!-- Register the interceptor with Spring MVC -->
    <mvc:interceptors>
        <mvc:interceptor>
            <mvc:mapping path="/*" />
```

```
            <bean
class="com.coforge.interceptor.CustomInterceptor" />
        </mvc:interceptor>
    </mvc:interceptors>
```

15. View Resolvers

- **Explanation**: The **ViewResolver** maps the logical view name returned by a controller to an actual view.
- **Code Example**:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xmlns:context="http://www.springframework.org/schema/context"
       xmlns:mvc="http://www.springframework.org/schema/mvc"
       xsi:schemaLocation="
         http://www.springframework.org/schema/beans
         https://www.springframework.org/schema/beans/spring-
beans.xsd
         http://www.springframework.org/schema/context

https://www.springframework.org/schema/context/spring-
context.xsd
         http://www.springframework.org/schema/mvc
         https://www.springframework.org/schema/mvc/spring-
mvc.xsd">

    <!-- Enable Spring MVC Annotations -->
    <mvc:annotation-driven />

    <!-- Component Scanning -->
    <context:component-scan base-
package="com.coforge.controller" />

    <!-- View Resolver Configuration -->
    <bean id="viewResolver"
class="org.springframework.web.servlet.view.InternalResourceVi
ewResolver">
        <property name="prefix" value="/WEB-INF/views/" />
        <property name="suffix" value=".jsp" />
    </bean>
```