Model-View-Controller using SpringMVC

INF5750/9750 - Lecture 3 (Part I)

Problem area

- Mixing application logic and markup is bad practice
  - Harder to change and maintain
  - Error prone
  - Harder to re-use

```
public void doGet( HttpServletRequest request, HttpServletResponse response )
{
    PrintWriter out = response.getWriter();

    out.println( ”<html>\n<body>” );

    if ( request.getParameter( ”foo” ).equals( ”bar” ) )
        out.println( ”<p>Foo is bar!</p>” );
    else
        out.println( ”<p>Foo is not bar!</p>” );

    out.println( ”</body>\n</html>” );
}
```

The MVC pattern

- MVC pattern breaks an application into three parts:
  - Model: The domain object model / service layer
  - View: Template code / markup
  - Controller: Presentation logic / action classes
- MVC defines interaction between components to promote separation of concerns and loose coupling
  - Each file has one responsibility
  - Enables division of labour between programmers and designers
  - Facilitates unit testing
  - Easier to understand, change and debug

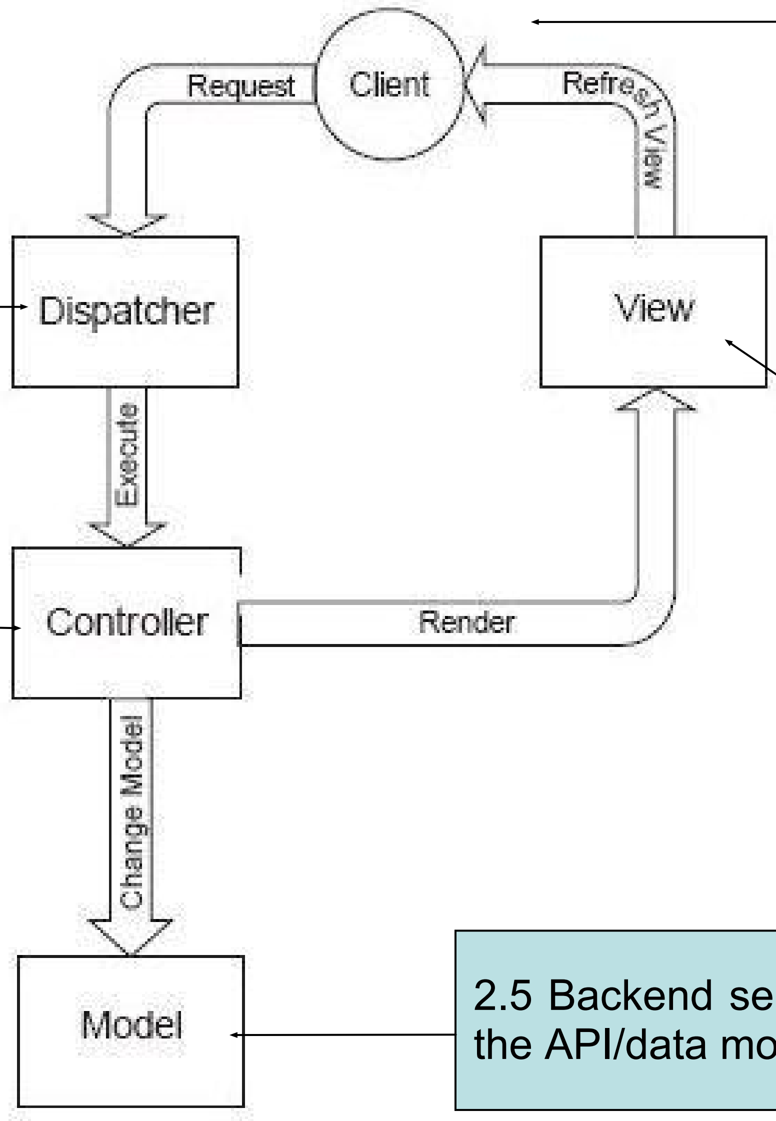
Advantages

Separation of application logic and web design through the *MVC pattern*

- Integration with template languages
- Some MVC frameworks provide built-in components
- Other advantages include:
  - Form validation
  - Error handling
  - Request parameter type conversion
  - Internationalization
  - IDE integration

- We will look at Spring web MVC framework in depth. DHIS2 uses Struts mainly, but uses Spring web MVC for Web-API

MVC with Front Controller



Step 1. Front controller. Maps request URLs to controller classes. Implemented as a servlet.

Step 2. Loads the Model & returns a ModelandView Interacts with backend services of the system.

2.5 Backend services working with the API/data model

Step 3. Web page template displays View (JSP or Velocity), which uses model for data

Step 4. Web browser displays output.

DispatcherServlet in web.xml

- Web applications define servlets in web.xml
- Maps URL patterns to servlets

- WebApplicationContext is an extension of ApplicationContext for features of *Servlets* and *themes*

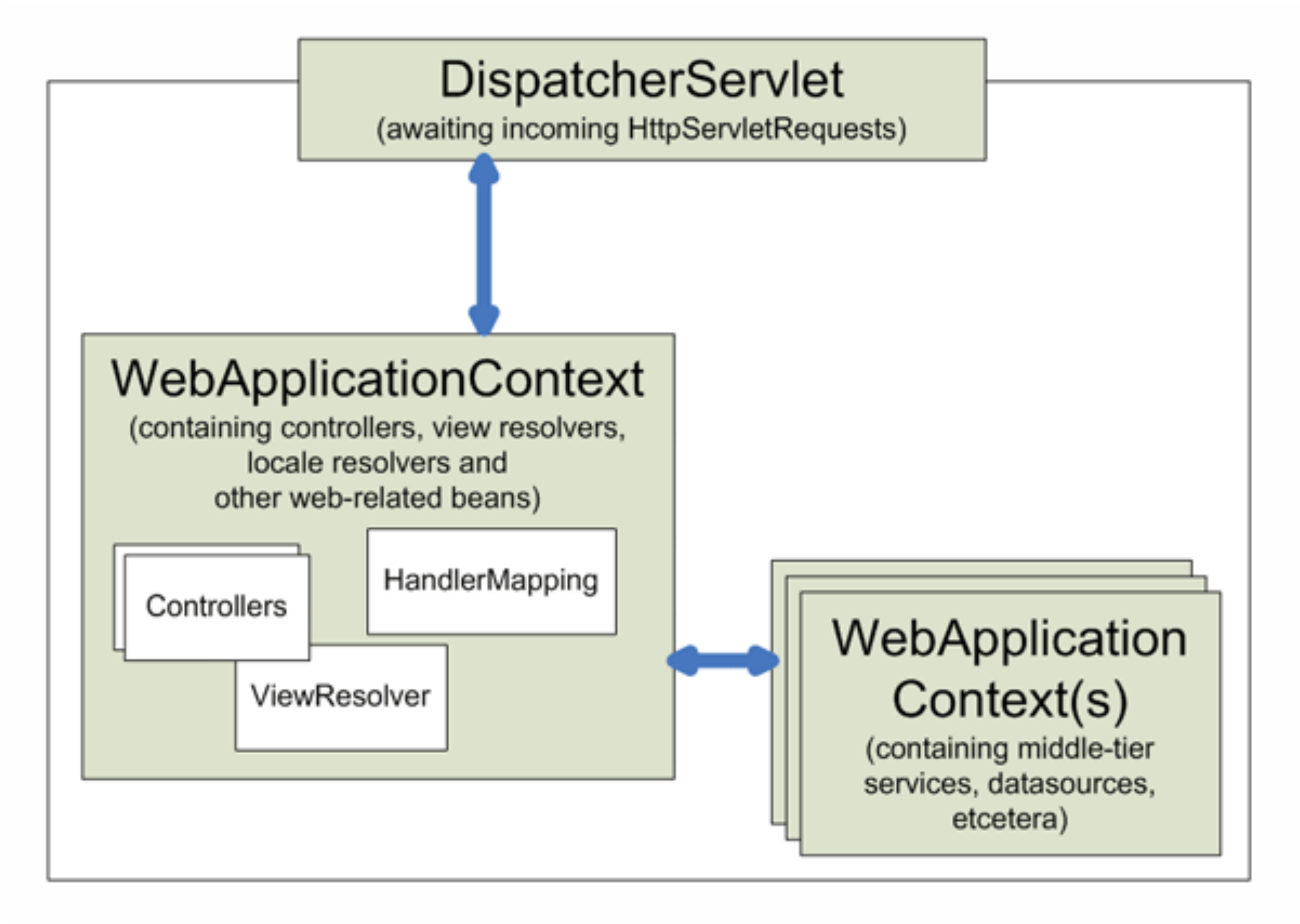
```
<web-app>
...
   <servlet>
      <servlet-name>mvc-dispatcher</servlet-name>
      <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
      <load-on-startup>1</load-on-startup>
   </servlet>

   <servlet-mapping>
      <servlet-name>mvc-dispatcher</servlet-name>
      <url-pattern>/</url-pattern>
   </servlet-mapping>

   <context-param>
      <param-name>contextConfigLocation</param-name>
      <param-value>/WEB-INF/mvc-dispatcher-servlet.xml</param-value>
   </context-param>
...
</web-app>
```

The Spring DispatcherServlet

The URL to be “captured” by DispatcherServlet

Finds the file in WEB-INF *[servlet-name]-servlet.xml* to initiate beans

WebApplicationContext Internals


DispatcherServlet
(awaiting incoming HttpServletRequests)
WebApplicationContext
(containing controllers, view resolvers, locale resolvers and other web-related beans)
Controllers
HandlerMapping
ViewResolver
WebApplication Context(s)
(containing middle-tier services, datasources, etcetera)

Overriding *DispatcherServlet* defaults

- DispatcherServlet initiates with default configuration. Overiding it through the *[servlet-name]-servlet.xml bean*
- Configuring *ViewResolver* is basic step

Different types of *ViewResolver*. Following 2 basic ones:

- *InternalResourceViewResolver (for jsp, css, images etc)*
- *ContentNegotiatingViewResolver (for ContentType response, useful for REST APIs)*

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

From Assignment 1:
if the Controller returns “**index**”, InternalResourceViewResolver tries to find file as view **/WEB-INF/pages/index.jsp**

Controllers

- The central components of MVC
- Simply add **@Controller** annotation to a class
- Use **@RequestMapping** to map methods to url

```
<beans…>
<context:component-scan base-package="no.uio.inf5750.assignment1"/>
...
</beans>
```

Get all the **@Controller** annotated classes accessible as beans

```
@Controller
public class BaseController {

        @RequestMapping(value="/")
        public String welcome(ModelMap model) {

                model.addAttribute("message", "Whaddap!!");

                //Spring uses InternalResourceViewResolver and return back index.jsp
                return "index";
        }
…
}
```

More detailed Url Mapping

- @RequestMapping also accepts the following parameters:
  - method (GET/POST/PUT/DELETE...)
  - produces (mimeType)
  - consumes (mimeType)
  - params
  - headers

```
...
     @RequestMapping(value="/", method = RequestMethod.GET, produces = "text/html")
     public String welcome(ModelMap model) {

          model.addAttribute("message", "Whaddap!!");

          //Spring uses InternalResourceViewResolver and return back index.jsp
          return "index";
     }
…
}
```

- We look at these parameters in depth, in next presentation for REST Web Services using SpringMVC lecture slides

Url Templates in Controllers

- @PathVariable - to map variables in URL paths
- path variables can also be Regular Expressions
- You can also do as follows
  - /message/*/user/{name}

- You can also use comma-separated URL parameters (also called Matrix-Variables)
  - To do this, make setRemoveSemicolonContent=false for *RequestMappingHandlerMapping*

```
// GET = /message/lars;friends=bob,rob,andy
@RequestMapping(value="/message/{name}", method = RequestMethod.GET)
public String welcome( @PathVariable String name,  @MatrixVariable String[] friends, ModelMap model) {

     model.addAttribute("message", "Hello " + name + " from " + friends[0] + " & " + friends[1]);
     //Spring uses InternalResourceViewResolver and return back index.jsp
     return "index";
}
```

Model

- Controllers and view share a Java object referred as model, ('M' in MVC)
- A model can be of the type *Model* or can be a *Map* that can represent the model.
- The view uses this to display dynamic data that has been given by the controller

```
// Controller
@RequestMapping(value = "/{name}", method = RequestMethod.GET)
        public String welcome(@PathVariable String name, ModelMap model) {

        model.addAttribute("message", "Hello " + name);

        return "index";
}
```

```
<%-- VIEW--%>

<html>
   <body>
       <h1>{message}</h1>
   </body>
</html>
```

@ModelAttribute from Controller

- You can also use ***@ModelAttribute*** in controller to directly load URL value into the model

- A Model can represent objects that can be retrieved from database or files as well

- Model should not have logic, rather the controller should get the model and “transform” the model based on the request, while sending it to the View

View

- Spring MVC integrates with many view technologies:
  - JSP
  - Velocity
  - Freemarker
  - JasperReports
- Values sent to controller with POST or GET as usual
- Values made available to the view by the controller

View

- *Velocity* is a popular template engine and language
- JSP commonly uses JSTL as a expression language
- “Templating” allows sharing dynamic page fragments

The Controller returns invertedWord

```
public String getInvertedWord() {
        return this.invertedWord;
   }
```

The request is mapped to the Controller path

Corresponding name is mapped in the Controller

Controller returned model can be accessed in the View

```
<html>

<body>

    <form method=”post” action=”/invertString”>

    <p><input type=”text” name=”word”></p>

    <p><input type=”submit” value=”Invert”></p>

    </form>

    <p>${invertedWord}</p>

</body>

</html>
```

JSTL

- Although JSTL is huge, we'll try to cover small part of it
- Include JSTL as part of your maven dependency

- prefix "c" can be used for core language
- prefix "fn" for using JSTL functions

```
<dependency>
   <groupId>javax.servlet</groupId>
   <artifactId>jstl</artifactId>
   <version>1.2</version>
   <scope>provided</scope>
</dependency>
```

At the top of the JSP file, one needs to add the following lines

```
<?xml version="1.0" encoding="UTF-8" ?>
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
<%@taglib prefix="fn" uri="http://java.sun.com/jsp/jstl/functions"%>

<c:forEach var="i" begin="0" end="5">
         Item <c:out value="${i}"/><p>
</c:forEach>

<c:if test="${fn:length(friends) > 0}" >
     <%@include file="welcome.jsp" %>
</c:if>
```

Interceptors

- Requests or Response can be worked on through the use of Interceptors
- They are generic, sharable “controller-like” components that are useful for doing authentication or security or validation
- Suppose you want to stop all requests between 23:00 and 06:00, you could write an Interceptor as follows

```
<beans>
  <bean id="handlerMapping"
      class="org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerMapping">
    <property name="interceptors">
      <list>
        <ref bean="officeHoursInterceptor"/>
      </list>
    </property>
  </bean>

  <bean id="officeHoursInterceptor"
      class="no.uio.inf5750.TimeBasedAccessInterceptor">
    <property name="openingTime" value="6"/>
    <property name="closingTime" value="23"/>
  </bean>
<beans>
```

Interceptor class

```
package no.uio.inf5750;
….
public class TimeBasedAccessInterceptor extends HandlerInterceptorAdapter {

    private int openingTime;
    private int closingTime;

    public void setOpeningTime(int openingTime) {
        this.openingTime = openingTime;
    }

    public void setClosingTime(int closingTime) {
        this.closingTime = closingTime;
    }

    public boolean preHandle(HttpServletRequest request, HttpServletResponse response, Object handler)
              throws Exception {

        Calendar cal = Calendar.getInstance();
        int hour = cal.get(HOUR_OF_DAY);
        if (openingTime <= hour && hour < closingTime) {
            return true;
        } else {
            response.sendRedirect("outsideOfficeHours");
            return false;
        }
    }
}
```

Resources

- Spring MVC docs - http://docs.spring.io/spring/docs/3.2.x/spring-framework-reference/html/mvc.html

- JSTL docs - http://docs.oracle.com/javaee/5/tutorial/doc/bnakh.html

-