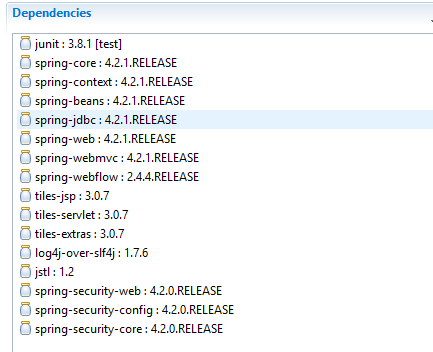
Spring 4.2 Web Application Setup Notes

## Dependencies

These are the dependencies I added. This should allow me to create a Spring web application using Spring Webflow, Spring Security and Apache Tiles. I’ve started off using Spring 4.2 and tiles 3.07. I’ll see if these are compatible and comment on that subsequently. The server started up and the application ran at this point so its a start although nothing is configured currently. I’m going to use XML configuration because that’s what we use in work from what I’ve seen.



## Configuring Basic MVC application

The following need to be created:

* Spring configuration file
* A controller
* Update web.xml
* Move index.jsp wherever your view resolver suggests

### Spring Configuration File

This Spring configuration file scans packages for components, allowing them to be used and it sets up a view resolver which will add a prefix and suffix to a string to resolve the path to a jsp.

This is located here:/src/main/webapp/WEB-INF/mvc-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"*

xsi:schemaLocation=*"http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd*

*http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-4.2.xsd"*>

<context:component-scan base-package=*"com.bwardweb.controller"*></context:component-scan>

<bean class=*"org.springframework.web.servlet.view.InternalResourceViewResolver"*>

<property name=*"prefix"*>

<value>/WEB-INF/jsp/</value>

</property>

<property name=*"suffix"*>

<value>.jsp</value>

</property>

</bean>

</beans>

### Controller

This is a controller (annotated with @Controller) located in the package being scanned by the config file. This is resolving a path defined in the @RequestMapping annotation returning a string which is used to return a jsp. This is located here: /src/main/java/com/bwardweb/controller/

**package** com.bwardweb.controller;

**import** org.springframework.stereotype.Controller;

**import** org.springframework.web.bind.annotation.RequestMapping;

**import** org.springframework.web.bind.annotation.RequestMethod;

@Controller

**public** **class** BaseController {

@RequestMapping(value="/", method=RequestMethod.***GET***)

**public** String home(){

**return** "index";

}

}

### Update the web.xml

Web.xml is updated to reference the dispatcher servlet defined in the mvc-dispatcher-servlet.xml file. This defines the class to use, the mapping and where to find the config file

<web-app xmlns=*"http://java.sun.com/xml/ns/javaee"*

xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xsi:schemaLocation=*"http://java.sun.com/xml/ns/javaee*

*http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd"*

version=*"2.5"*>

<display-name>Fitness Tracker</display-name>

<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>

<listener>

<listener-class>

org.springframework.web.context.ContextLoaderListener

</listener-class>

</listener>

</web-app>

## Adding Logging

To add logging I added the log4j dependency to the pom as follows:

<dependency>

<groupId>log4j</groupId>

<artifactId>log4j</artifactId>

<version>1.2.17</version>

</dependency>

I then created a file called log4j.properties and added to src/main/resources although I think once its on the classpath it should be fine. This file looks lie this:

# Root logger option

log4j.rootLogger=DEBUG, stdout, file

# Redirect log messages to console

log4j.appender.stdout=org.apache.log4j.ConsoleAppender

log4j.appender.stdout.Target=System.out

log4j.appender.stdout.layout=org.apache.log4j.PatternLayout

log4j.appender.stdout.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss} %-5p %c{1}:%L - %m%n

You could also add a redirect to output to file as follows:

# Redirect log messages to a log file, support file rolling.

log4j.appender.file=org.apache.log4j.RollingFileAppender

log4j.appender.file.File=C:\\log4j-application.log

log4j.appender.file.MaxFileSize=5MB

log4j.appender.file.MaxBackupIndex=10

log4j.appender.file.layout=org.apache.log4j.PatternLayout

log4j.appender.file.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss} %-5p %c{1}:%L - %m%n

Finally I added a Logger to my controller like this (with the relevant imports):

**private** **static** **final** Logger ***log*** = Logger.*getLogger*(BaseController.**class**);

Then I could log out as follows:

***log***.debug("Index.jsp");

## Configuring Apache Tiles

To configure Apache Tiles I did the following:

* Created a TilesViewResolver bean in mvc-dispatcher-servlet
* Created a TilesConfigurer bean in mvc-dispatcher-servlet
* Created a tiles.xml file that defined my pages
* Created a default.jsp file that would be the main template
* Amended index.jsp to just have content that would slot into default.jsp

### Amendments to mvc-dispatcher-servlet

This involved creating two beans with the configure pointing to the location of the xml file that defines the pages. The InternalViewResolver could also be removed from this file at this point. The bean definitions are below.

<bean id=*"tilesViewReslover"*

class=*"org.springframework.web.servlet.view.tiles3.TilesViewResolver"*>

</bean>

<bean id=*"tilesViewConfigurer"*

class=*"org.springframework.web.servlet.view.tiles3.TilesConfigurer"*>

<property name=*"definitions"*>

<list>

<value>/WEB-INF/views/\*\*/tiles.xml</value>

</list>

</property>

</bean>

### Adding tiles.xml

The tiles.xml file defines the pages, allowing you to specify an overall template which you then slot tiles into. This should be placed in the location pointed to in the path specified in the TilesConfigurer property and looks something like this:

<?xml version=*"1.0"* encoding=*"UTF-8"* ?>

<!DOCTYPE tiles-definitions PUBLIC "-//Apache Software Foundation//DTD Tiles Configuration 3.0//EN" "http://tiles.apache.org/dtds/tiles-config\_3\_0.dtd">

<tiles-definitions>

<!-- Base Definition -->

<definition name=*"base-definition"*

template=*"/WEB-INF/views/tiles/layouts/default.jsp"*>

<put-attribute name=*"title"* value=*""* />

<put-attribute name=*"body"* value=*""* />

</definition>

<!-- Home Page -->

<definition name=*"index"* extends=*"base-definition"*>

<put-attribute name=*"title"* value=*"Welcome"* />

<put-attribute name=*"body"* value=*"/WEB-INF/views/pages/index.jsp"* />

</definition>

</tiles-definitions>

The name of the definition is what’s used by the controller when the view is being resolved. The attributes defined in the definitions override those of the base-definition.

### Adding default.jsp

This is the JSP that you will be slotting tiles into. Note that you need to add the tiles taglib in as I forgot this and I spent a bit of time wondering why the attributes weren’t being set and my pages were blank.

<%@ page language=*"java"* contentType=*"text/html; charset=ISO-8859-1"*

pageEncoding=*"ISO-8859-1"*%>

<%@ taglib uri=*"http://tiles.apache.org/tags-tiles"* prefix=*"tiles"*%>

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">

<html>

<head>

<meta http-equiv=*"Content-Type"* content=*"text/html; charset=ISO-8859-1"*>

<title><tiles:getAsString name=*"title"* /></title>

</head>

<body>

<div>

<tiles:insertAttribute name=*"body"*></tiles:insertAttribute>

</div>

</body>

</html>

### Amending index.jsp

Index.jsp becomes just something very basic at this point being a HTML snippet rather than an actual JSP. This just slots into the tile attribute called body.

<h2>Hello World!</h2>

## Configuring Webflow

So this took a bit of effort. The first thing that caused me to struggle with this was the fact that logging didn’t seem to be occurring with my app beyond what was happening through the logging I configured above. To get this to work I had to exclude a logging dependency for the tiles dependencies in the POM. So the tiles dependencies became as follows:

<dependency>

<groupId>org.apache.tiles</groupId>

<artifactId>tiles-servlet</artifactId>

<version>3.0.7</version>

<exclusions>

<exclusion>

<artifactId>jcl-over-slf4j</artifactId>

<groupId>org.slf4j</groupId>

</exclusion>

</exclusions>

</dependency>

<dependency>

<groupId>org.apache.tiles</groupId>

<artifactId>tiles-extras</artifactId>

<version>3.0.7</version>

<exclusions>

<exclusion>

<artifactId>jcl-over-slf4j</artifactId>

<groupId>org.slf4j</groupId>

</exclusion>

</exclusions>

</dependency>

After this, it became a lot clearer where things were going wrong with my configuration.

### Changes to mvc-dispatcher-servlet

Most of the changes were to the configuration in mvc-dispatcher-servlet.xml. In fact there were even small changes to some of the existing beans that were defined previously for the tiles so I’m going to copy the whole thing in below. Note the addition of the webflow-config namespace.

I also had to add the mvc:annotation-driven tag to get the @Controller to be recognised. I’m not entirely sure why this. The controller was working before I set up webflow but did not seem to be afterwards (the mappings weren’t being recognised). I’ve looked on the internet and see conflicting thoughts on it so I’m not sure. It seems to be required though. Maybe its because of the FlowHandlerMapping bean being added.

Another important thing to note is the change to the tilesViewResolver bean which became a UrlBasedViewResolver type with a property referring to a TilesView class.

<?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:webflow-config=*"http://www.springframework.org/schema/webflow-config"*

xmlns:mvc=*"http://www.springframework.org/schema/mvc"*

xsi:schemaLocation=*"http://www.springframework.org/schema/webflow-config http://www.springframework.org/schema/webflow-config/spring-webflow-config-2.4.xsd*

*http://www.springframework.org/schema/mvc http://www.springframework.org/schema/mvc/spring-mvc-4.2.xsd*

*http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd*

*http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-4.2.xsd"*>

<context:component-scan base-package=*"com.bwardweb.controller"*></context:component-scan>

<mvc:annotation-driven></mvc:annotation-driven>

<bean id=*"tilesViewResolver"*

class=*"org.springframework.web.servlet.view.UrlBasedViewResolver"*>

<property name=*"viewClass"*

value=*"org.springframework.web.servlet.view.tiles3.TilesView"* />

</bean>

<bean id=*"tilesViewConfigurer"*

class=*"org.springframework.web.servlet.view.tiles3.TilesConfigurer"*>

<property name=*"definitions"*>

<list>

<value>/WEB-INF/views/\*\*/tiles.xml</value>

</list>

</property>

</bean>

<!-- Dispatches requests mapped to flows to FlowHandler implementations -->

<bean class=*"org.springframework.webflow.mvc.servlet.FlowHandlerAdapter"*>

<property name=*"flowExecutor"* ref=*"flowExecutor"* />

</bean>

<!-- Maps request paths to flows in the flowRegistry -->

<bean class=*"org.springframework.webflow.mvc.servlet.FlowHandlerMapping"*>

<property name=*"order"* value=*"-1"* />

<property name=*"flowRegistry"* ref=*"flowRegistry"* />

</bean>

<!-- Executes flows: the entry point into the Spring Web Flow system -->

<webflow-config:flow-executor id=*"flowExecutor"*

flow-registry=*"flowRegistry"*>

</webflow-config:flow-executor>

<!-- The registry of executable flow definitions -->

<webflow-config:flow-registry id=*"flowRegistry"*

flow-builder-services=*"flowBuilderServices"* base-path=*"/WEB-INF/flows"*>

<webflow-config:flow-location-pattern value=*"/\*-flow.xml"* />

</webflow-config:flow-registry>

<!-- Plugs in a custom creator for Web Flow views -->

<webflow-config:flow-builder-services id=*"flowBuilderServices"*

view-factory-creator=*"mvcViewFactoryCreator"*

development=*"true"* />

<bean id=*"mvcViewFactoryCreator"*

class=*"org.springframework.webflow.mvc.builder.MvcViewFactoryCreator"*>

<property name=*"viewResolvers"*>

<list>

<ref bean=*"tilesViewResolver"* />

</list>

</property>

<property name=*"useSpringBeanBinding"* value=*"true"* />

</bean>

</beans>

So aside from the webflow-config namespace and the mvc:annotation-driven tag, the following were added here:

* FlowExecutor – I think this is what actually controls the flows
* FlowRegistry – keeps track of which flows are available (you provide a path to your defined flows for this bean to refer to)
* FlowHandlerAdapter – not 100% sure what this is but I think it ties the FlowHandlerMapping bean back to the FlowExecutor bean
* FlowHandlerMapping – maps requests to flows in the registry
* FlowBuilderServices – used to configure views (not required if you aren’t using tiles I don’t think although may be required if configuring validation)
* MvcViewFactoryCreator – references the tilesViewResolver and gets passed to the FlowBuilderServices (again not required if not using tiles I don’ think)

### Adding Flow Definitions

This is quite straightforward. At the moment I just have a couple of flows defined and each only have a couple of states without any means of navigating between them (this will be the next thing to look at). The first defined state will be the entry point for the flow. The view property refers to the tile definition name. Below is the contents of the file called main-flow.xml.

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<flow xmlns=*"http://www.springframework.org/schema/webflow"*

xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xsi:schemaLocation=*"http://www.springframework.org/schema/webflow*

*http://www.springframework.org/schema/webflow/spring-webflow-2.0.xsd"*>

<view-state id=*"start"* view=*"index"*>

</view-state>

<view-state id=*"end"* view=*"test"*>

</view-state>

</flow>

### Amending the Controller

At this point, we have both webflow and MVC configured. So to get your application to use the flow you can redirect to one of your flow definitions from one of the RequestMappings in the controller. Here’s an example below:

@RequestMapping(value="/", method=RequestMethod.***GET***)

**public** String home(){

***log***.debug("main flow");

**return** "redirect:/main-flow";

}

This will then display the first view state defined in that flow.

## Moving Between Flows and MVC

Next up is configuring the flows to transition between views and also move back to MVC or other flows. This involved making some amendments to the flow definitions, adding transitions to view states and also end-states when redirecting away from a flow.

### Amending the main-flow.xml

The changes made here are as discussed above. One thing to note, you need to end the state or the redirect won’t work from what I can tell. Even having a link on a flow page seemed to return a 404 for an internal resource.

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<flow xmlns=*"http://www.springframework.org/schema/webflow"*

xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xsi:schemaLocation=*"http://www.springframework.org/schema/webflow*

*http://www.springframework.org/schema/webflow/spring-webflow-2.0.xsd"*>

<view-state id=*"start"* view=*"index"*>

<transition on=*"nextScreen"* to=*"flowEnd"*></transition>

<transition on=*"mvc"* to=*"mvcEndFlow"*></transition>

<transition on=*"nextFlow"* to=*"nextFlowEndFlow"*></transition>

</view-state>

<view-state id=*"flowEnd"* view=*"flowEnd"*>

<transition on=*"mvc"* to=*"mvcEndFlow"*></transition>

</view-state>

<end-state id=*"mvcEndFlow"* view=*"externalRedirect:index"*></end-state>

<end-state id=*"nextFlowEndFlow"* view=*"externalRedirect:test"*></end-state>

</flow>

### Amending the jsp

I set up a bit of a loop where two flows could interact with each other and also exit to MVC. The concept is the same though so I’m just going to provide one JSP. I’ll try to leave this loop intact in the code even though I won’t provide any links to it on the pages when the application is doing more than it is now and I’ll probably rename stuff a bit (i.e. you’ll need to know the entry point to hit it).

<h2>Hello World!</h2>

<form action=*"${flowExecutionUrl}"* method=*"post"*>

<button class=*"btn btn-primary"*>Next in Flow</button>

<input type=*"hidden"* name=*"\_flowExecutionKey"* value=*"${flowExecutionKey}"*/>

<input type=*"hidden"* name=*"\_eventId"* value=*"nextScreen"*>

</form>

<form action=*"*${flowExecutionUrl}*"* method=*"post"*>

<button class=*"btn btn-primary"*>MVC</button>

<input type=*"hidden"* name=*"\_flowExecutionKey"* value=*"*${flowExecutionKey}*"*/>

<input type=*"hidden"* name=*"\_eventId"* value=*"mvc"*>

</form>

<form action=*"*${flowExecutionUrl}*"* method=*"post"*>

<button class=*"btn btn-primary"*>New Flow</button>

<input type=*"hidden"* name=*"\_flowExecutionKey"* value=*"*${flowExecutionKey}*"*/>

<input type=*"hidden"* name=*"\_eventId"* value=*"nextFlow"*>

</form>

The main things of interest here are the action and the hidden inputs. The \_eventId refers to the name of the ‘on’ in the transition defined in the flow. I’m not 100% sure on the flowExecutionUrl and flowExecutionKey but I presume these are webflows way ok keeping track of the flow.

## Configuring Spring Security

Next up was configuring spring security. This wasn’t too hard and involved creating a security-config.xml file primarily and then making some amendments to web.xml and the login screen. One thing I had an issue with was whenever I use a form to move to another screen I needed to include a hidden input with a name of \_csrf. It looks like this:

<input name=*"\_csrf"* type=*"hidden"* value=*"*${\_csrf.token}*"* />

Other than that it was relatively straightforward. Here is the rest of the constituent parts.

### Changes to web.xml

<context-param>

<param-name>contextConfigLocation</param-name>

<param-value>/WEB-INF/spring/\*.xml</param-value>

</context-param>

<filter>

<filter-name>springSecurityFilterChain</filter-name>

<filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-class>

</filter>

<filter-mapping>

<filter-name>springSecurityFilterChain</filter-name>

<url-pattern>/\*</url-pattern>

</filter-mapping>

### The security-config.xml

This is located in the directory specified for the contextConfigLocation in web.xml (i.e. WEB-INF/spring/\*.xml). At the moment this is using a user hardcoded into the authentication-manager. This will be swapped out for a database at some stage. The security:http section defines the intercept URLs and the rules around them. The order of these is important. The login and access denied pages are also defined here (if you don’t define your own login page you are provided with a default).

<?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:security=*"http://www.springframework.org/schema/security"*

xsi:schemaLocation=*"http://www.springframework.org/schema/security http://www.springframework.org/schema/security/spring-security-4.2.xsd*

*http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd"*>

<security:authentication-manager>

<security:authentication-provider>

<security:user-service>

<security:user name=*"user"* authorities=*"ROLE\_USER"*

password=*"password"* />

</security:user-service>

</security:authentication-provider>

</security:authentication-manager>

<security:http>

<security:intercept-url pattern=*"/register"* access=*"permitAll"* />

<security:intercept-url pattern=*"/leaderboard"* access=*"permitAll"* />

<security:intercept-url pattern=*"/login"* access=*"permitAll"* />

<security:intercept-url pattern=*"/denied"* access=*"permitAll"* />

<security:intercept-url pattern=*"/"* access=*"permitAll"* />

<security:intercept-url pattern=*"/mvc"* access=*"permitAll"* />

<security:intercept-url pattern=*"/test1"* access=*"permitAll"* />

<security:intercept-url pattern=*"/test2"* access=*"permitAll"* />

<security:intercept-url pattern=*"/hub"* access=*"isAuthenticated()"* />

<security:intercept-url pattern=*"/main-flow"* access=*"isAuthenticated()"* />

<security:intercept-url pattern=*"/\*\*"* access=*"denyAll"* />

<security:form-login login-page=*"/login"*

authentication-failure-url=*"/login?error=true"*

default-target-url=*"/"*

/>

<security:access-denied-handler error-page=*"/denied"* />

</security:http>

</beans>

### The Login JSP

The key thing here seems to be matching the names of the elements and the hidden input. I based this off the default one.

<form name=*'f'* action=*'/fitness-tracker/login'* method=*'POST'*>

<table>

<tr><td>User:</td><td><input type=*'text'* name=*'username'* value=*''*></td></tr>

<tr><td>Password:</td><td><input type=*'password'* name=*'password'*/></td></tr>

<tr><td colspan=*'2'*><input name=*"submit"* type=*"submit"* value=*"Login"*/></td></tr>

</table>

<input name=*"\_csrf"* type=*"hidden"* value=*"*${\_csrf.token}*"* />

</form>

## Configuring Database Connection

I configured a simple database connection to a MySQL database and accessed it using JdbcTemplate. At the moment I’ve just hardcoded the connection details but I intend to set up a property file for this at some point. I may also explore JPA. To configure the database, the following steps were required:

* Add mysql-connector-java to pom.xml (note I used version 5.1.39 as originally I used 6.0.6 and it threw an exception related to Timezones!)
* Add commons-dbcp to pom.xml
* Add spring-jdbc to pom.xml
* Add a BasicDataSource Bean to the mvc-dispatcher-servlet.xml (its properties include the connection details)
* Add a JdbcTemplate Bean to the mvc-dispatcher-servlet.xml (its takes the datasource as a property)
* Create a DAO object annotated with @Component with the JdbcTemplate @Autowired in and make a call to the database within it
* @Autowire the DAO object into the Controller and call it

### Changes to pom.xml

Note I’d actually already added spring-jdbc

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

<version>5.1.39</version>

</dependency>

<dependency>

<groupId>commons-dbcp</groupId>

<artifactId>commons-dbcp</artifactId>

<version>1.4</version>

</dependency>

</dependencies>

### Changes to mvc-dispatcher-servlet.xml

<bean id=*"dataSource"* class=*"org.apache.commons.dbcp.BasicDataSource"*>

<property name=*"url"* value=*"jdbc:mysql://localhost:3306/fitness\_tracker"*></property>

<property name=*"username"* value=*"root"*></property>

<property name=*"password"* value=*"password"*></property>

<property name=*"driverClassName"* value=*"com.mysql.jdbc.Driver"*></property>

</bean>

<bean id=*"jdbcTemplate"*

class=*"org.springframework.jdbc.core.JdbcTemplate"*>

<property name=*"dataSource"* ref=*"dataSource"*></property>

</bean>

### The DAO

**package** com.bwardweb.fitnesstracker.dao;

**import** org.springframework.beans.factory.annotation.Autowired;

**import** org.springframework.jdbc.core.JdbcTemplate;

**import** org.springframework.stereotype.Component;

@Component("TestDao")

**public** **class** TestDao {

@Autowired

**private** JdbcTemplate jdbcTemplate;

**public** **int** getUsersCount(){

**int** rowCount = **this**.jdbcTemplate.queryForObject("select count(\*) from users", Integer.**class**);

**return** rowCount;

}

}

### Changes to Controller

@Autowired

**private** TestDao testDao;

@RequestMapping(value="/", method=RequestMethod.***GET***)

**public** String home(){

***log***.debug("homepage");

***log***.debug("Users in database: " + testDao.getUsersCount());

**return** "index";

}