# SPRING MVC

## 1. Creare un progetto Spring MVC base

### 1.1. Creare nuovo progetto in Eclipse

File → New → Other → Dynamic Web Project
Project Name = dare un nome (es. “SpringMVC”)
Target runtime = Apache Tomcat (da installare se non lo hai già fatto)
Dynamic web module versione: 3.0 oppure 3.1
Configuration: Default configuration for Apache Tomcat
→ Next → Next
Spuntare Generate web.xml
→ Finish

#### 1.1.1. Convertire il progetto in un progetto maven

Tasto dx sul progetto → Configure → Convert to Maven Project

### 1.2. Configurare il pom.xml

Aggiungere le dipendenze di Spring e Servlet.

**pom.xml**

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.spring</groupId>
  <artifactId>step32</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <packaging>war</packaging>
  <build>
    <sourceDirectory>src</sourceDirectory>
    <plugins>
      <plugin>
        <artifactId>maven-compiler-plugin</artifactId>
        <version>3.5.1</version>
        <configuration>
          <source>1.7</source>
          <target>1.7</target>
        </configuration>
      </plugin>
      <plugin>
        <artifactId>maven-war-plugin</artifactId>
        <version>3.0.0</version>
        <configuration>
          <warSourceDirectory>WebContent</warSourceDirectory>
        </configuration>
      </plugin>
    </plugins>
  </build>
  <properties>
      <org.springframework-version>4.3.1.RELEASE</org.springframework-version>
```

```
  </properties>

  <dependencies>

      <!-- Spring Framework -->
      <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-context</artifactId>
            <version>${org.springframework-version}</version>
      </dependency>
      <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-aop</artifactId>
            <version>${org.springframework-version}</version>
      </dependency>
      <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-webmvc</artifactId>
            <version>${org.springframework-version}</version>
      </dependency>
            <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-web</artifactId>
            <version>${org.springframework-version}</version>
      </dependency>

      <!-- Servlet -->
      <dependency>
            <groupId>javax.servlet</groupId>
            <artifactId>jstl</artifactId>
            <version>1.2</version>
      </dependency>
      <dependency>
            <groupId>javax.servlet</groupId>
            <artifactId>javax.servlet-api</artifactId>
            <version>3.0.1</version>
            <scope>provided</scope>
      </dependency>

  </dependencies>
</project>
```

Poi aggiornare il progetto:
Tasto dx sul progetto→ Maven → Update Project → spuntare Force Update → Ok

## 1.3. Aggiungere la DispatcherServlet al file web.xml

La Dispatcher Servlet è la servlet principale di spring mvc. É sostanzialmente un Front-Controller attraverso cui passano tutte le richieste dell'applicazione.

**web.xml**

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://xmlns.jcp.org/xml/ns/javaee"
xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd" id="WebApp_ID"
version="3.1">
  <display-name>SpringStep32</display-name>
  <welcome-file-list>
```

```
    <welcome-file>index.html</welcome-file>
    <welcome-file>index.htm</welcome-file>
    <welcome-file>index.jsp</welcome-file>
    <welcome-file>default.html</welcome-file>
    <welcome-file>default.htm</welcome-file>
    <welcome-file>default.jsp</welcome-file>
  </welcome-file-list>

  <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>
```

### 1.3.1. Creare il file dispatcher-servlet.xml

La Dispatcher Servlet ha bisogno di lanciare uno **Spring Application Context. Questo va configurato in un apposito file**, sotto WEB-INF. Dato che nel file web.xml abbiamo chiamato la servlet con “dispatcher”, andiamo sotto WEB-INF e creiamo il file dispatcher-servlet.xml:

**dispatcher-servlet.xml**

```
<beans xmlns="http://www.springframework.org/schema/beans"
      xmlns:mvc="http://www.springframework.org/schema/mvc"
xmlns:context="http://www.springframework.org/schema/context"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/mvc
        http://www.springframework.org/schema/mvc/spring-mvc.xsd
        http://www.springframework.org/schema/context
        http://www.springframework.org/schema/context/spring-context.xsd">

      <context:component-scan base-package="com.spring.step32" />

      <!-- <bean id="viewResolver"
class="org.springframework.web.servlet.view.UrlBasedViewResolver"> -->
      <bean id="viewResolver"
class="org.springframework.web.servlet.view.InternalResourceViewResolver">
            <!-- <property name="viewClass"
value="org.springframework.web.servlet.view.JstlView" /> -->
            <property name="prefix" value="/WEB-INF/views/" />
            <property name="suffix" value=".jsp" />
      </bean>

      <mvc:annotation-driven />

      <context:annotation-config />

</beans>
```

## 1.4. Creare il Controller generale HomeController

Creare la classe HomeController, e annotarla con l'annotazion @Controller (org.springframework.stereotype.Controller)

**HomeController.java**

```
package com.spring.step32;

import java.text.DateFormat;
import java.util.Date;
import java.util.Locale;

import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.RequestMapping;

@Controller
public class HomeController {

	@RequestMapping("/home")
	public String home(Locale locale, Model model) {
		Date date = new Date();
		DateFormat dateFormat =
DateFormat.getDateTimeInstance(DateFormat.LONG, DateFormat.LONG, locale);

		String formattedDate = dateFormat.format(date);

		model.addAttribute("serverTime", formattedDate);

		return "home";
	}

	@RequestMapping(value="/hello")
	@ResponseBody
	public String sayHello() {
		return "Hello World";
	}
}
```

Il Model è qualcosa che viene scambiato/passato tra Controller e Views: se voglio rendere disponibile qualcosa dal Controller alla viste, inserisco quel qualcosa nel Model.

### 1.4.1. Creare la jsp home.jsp

Creare, sotto WEB-INF/views/ la jsp home.jsp

**home.jsp**

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<%@ page session="false" %>
<html>
<head>
	<title>Home</title>
</head>
<body>
<h1>
```

```
      Hello world!
</h1>

<P>  The time on the server is ${serverTime}. </P>
</body>
</html>
```

A questo punto abbiamo tutto il necessario per far funzionare spring.

# 2. Aggiungiamo la gestione della Login

## 2.1. Creare la classe LoginService

Questa classe gestirà, in modo molto spartano, la login. Sostanzialmente si verrà autenticati solo se si inseriscono username “admin” e password “admin”.

**LoginService.java**

```
package com.spring.step32.login;

import org.springframework.stereotype.Service;

@Service
public class LoginService {

      public boolean validateUser(String user, String password) {
            return user.equalsIgnoreCase("admin") && password.equals("admin");
    }
}
```

## 2.2. Creare la classe LoginController

Creare il Controller per intercettare la url “/login”.
Se la chiamata è di tipo GET si verrà reindirizzati sulla pagina login.jsp.
Se la chiamata è di tipo POST viene invocata la LoginService per verificare le credenziali immesse.

**LoginController.java**

```
package com.spring.step32.login;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.ModelMap;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.ResponseBody;
import org.springframework.web.bind.annotation.SessionAttributes;

@Controller
@SessionAttributes("name")
public class LoginController {

      @Autowired
      private LoginService loginService;
```

```
	@RequestMapping(value="/login", method=RequestMethod.GET)
	public String showLoginPage() {
		return "login";
	}

	@RequestMapping(value="/login", method=RequestMethod.POST)
	public String handleLogin(ModelMap model, @RequestParam String name,
	                                          @RequestParam String password) {

		if (!loginService.validateUser(name, password)) {
			model.put("errorMessage", "Invalid Credentials");
			return "login";
		  }

		  model.put("name", name);
		  return "welcome";
	}
}
```

## 2.3. Creare le jsp login e welcome

**login.jsp**

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!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=UTF-8">
<title>Insert title here</title>
</head>
<body>

<p><font color="red">${errorMessage}</font></p>
<form action="/step32/login" method="POST">
	<div>Name: <input name="name" type="text" /></div>
	<div>Password: <input name="password" type="password" /></div>
	<div><input type="submit" value="Login" /></div>
</form>

</body>
</html>
```

**welcome.jsp**

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!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=UTF-8">
<title>Insert title here</title>
</head>
<body>

Welcome ${name}. You are now authenticated.
<br>
<a href="/step32/lista-esami">Click here</a> to start maintaining your todo's.
```

```
</body>
</html>
```

# 3. Aggiungiamo la funzione per gestire gli esami, compresa la validazione delle Form

## 3.1. Il Model

Creiamo la classe Esame, per rappresentare oggetti di tipo Esame, e usiamo le annotation per la **validazione**.
Una cosa importante da ricordare è che l'oggetto che si vuole validare deve avere il costruttore di default (quello *no-args*).

**Esame.java**

```
package com.spring.step32.model;

import java.util.Date;

import javax.validation.constraints.Size;

public class Esame {

	private int id;
	@Size(min=6, message="Enter atleast 6 characters")
	private String desc;
	private Date data;
	private boolean superato;
	private String studente;

	public Esame() {
		super();
	}

	public Esame(int id, String desc, Date data, boolean superato, String
studente) {
	  super();
	  this.id = id;
	  this.desc = desc;
	  this.data = data;
	  this.superato = superato;
	  this.studente = studente;
    }


	// Getter e Setter


	@Override
	public int hashCode() {
		final int prime = 31;
		int result = 1;
		result = prime * result + id;
		return result;
	}

	@Override
	public boolean equals(Object obj) {
		if (this == obj)
```

```
                return true;
            if (obj == null)
                return false;
            if (getClass() != obj.getClass())
                return false;
            Esame other = (Esame) obj;
            if (id != other.id)
                return false;
            return true;
      }

      @Override
      public String toString() {
            return String.format(
                        "Esame [id=%s, desc=%s, data=%s, superato=%s, studente=
%s]",
                        id, desc, data, superato, studente);
      }
}
```

### 3.1.1. Aggiungiamo le dipendenze per la validazione

Per la validazione si usa il pacchetto **javax.validation**, ma abbiamo bisogna anche di una sua implementazione. In particolare useremo quella di **hibernate**. Per farlo occorre aggiungerla al pom:

**pom.xml**

```
            ...

            <!-- Hibernate Validator -->
            <dependency>
                  <groupId>org.hibernate</groupId>
                  <artifactId>hibernate-validator</artifactId>
                  <version>5.0.2.Final</version>
            </dependency>

            ...
```

## 3.2. Creare la classe EsameService e EsameController

Questa classe gestirà, in modo molto spartano, le operazioni sugli esami. Inizializza una lista di 3 esami, e implementa le funzioni di aggiunta/modifica/rimozione.

**LoginService.java**

```
package com.spring.step32.esame;

import java.util.Date;
import java.util.Iterator;
import java.util.ArrayList;
import java.util.List;

import org.springframework.stereotype.Service;

import com.spring.step32.model.Esame;


@Service
public class EsameService {
```

```
    private static List<Esame> esami = new ArrayList<Esame>();
    private static int esameCount = 3;

    static {
      esami.add(new Esame(1, "Learn Spring MVC", new Date(), false, "admin"));
      esami.add(new Esame(2, "Learn Struts", new Date(), false, "admin"));
      esami.add(new Esame(3, "Learn Hibernate", new Date(), false, "admin"));
    }


    public Esame cercaEsame(int id) {
        for (Esame esame : esami) {
            if (esame.getId() == id)
                  return esame;
        }
        return null;
    }


    public List<Esame> cercaEsami(String studente) {
        List<Esame> filteredEsami = new ArrayList<Esame>();
        for (Esame esame : esami) {
            if (esame.getStudente().equals(studente))
                  filteredEsami.add(esame);
        }
        return filteredEsami;
    }


    public void aggiungiEsame(String nome, String desc, Date data, boolean
superato) {
        esami.add(new Esame(++esameCount, desc, data, superato, "admin"));
    }


    public void aggiornaEsame(Esame esame) {
        esami.remove(esame);
        esami.add(esame);
    }


    public void rimuoviEsame(int id) {
        Iterator<Esame> iterator = esami.iterator();
        while (iterator.hasNext()) {
            Esame esame = iterator.next();
            if (esame.getId() == id) {
                iterator.remove();
            }
        }
    }
}
```

Creare poi il Controller per gestire gli esami e la validazione.

**EsameController-java**

```
package com.spring.step32.esame;

import java.util.Date;
```

```java
import javax.validation.Valid;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.ModelMap;
import org.springframework.validation.BindingResult;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.SessionAttributes;

import com.spring.step32.model.Esame;


@Controller

@SessionAttributes("name")
public class EsameController {

	@Autowired
	private EsameService service;

	@RequestMapping(value="/lista-esami", method=RequestMethod.GET)
	public String getListaEsami(ModelMap model) {
		String studente = (String)model.get("name");
	  model.addAttribute("esami", service.cercaEsami(studente));
	  return "lista-esami";
	}


	@RequestMapping(value = "/add-esame", method = RequestMethod.GET)
    public String mostraPaginaAddEsame(ModelMap model) {
        model.addAttribute("esame", new Esame());
        return "esame";
    }


	@RequestMapping(value = "/add-esame", method = RequestMethod.POST)
    public String addEsame(ModelMap model, @Valid Esame esame, BindingResult
result) {

        if (result.hasErrors())
            return "esame";

        service.aggiungiEsame((String)model.get("desc"), esame.getDesc(), new
Date(), false);
        model.clear
        model.clear();  // to prevent request parameter "studente" to be passed
        return "redirect:/lista-esami";
    }


	@RequestMapping(value = "/delete-esame", method = RequestMethod.GET)
    public String deleteEsame(@RequestParam int id) {
        service.rimuoviEsame(id);

        return "redirect:/lista-esami";
    }

}
```

## 3.3. Creare le jsp per gestire gli esami

Creare le jsp esame.jsp (con una form per aggiungere l'esame) e lista-esami.jsp (per visualizzare/editare gli esami).

**esame.jsp**

```
<%@taglib uri="http://www.springframework.org/tags/form" prefix="form"%>

<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!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=UTF-8">
<title>Insert title here</title>
</head>
<body>

<form:form method="post" commandName="esame">
      <form:label path="desc">Description:</form:label>
      <form:input path="desc" type="text" />
      <form:errors path="desc" font-color="red" />

      <button type="submit">Add</button>
</form:form>

</body>
</html>
```

**lista-esami.jsp**

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>

<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!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=UTF-8">
<title>Insert title here</title>
</head>
<body>

<table>
      <caption>I tuoi Esami</caption>
      <tr>
            <th>Descriziome</th>
            <th>Data</th>
            <th>Superato</th>
            <th></th>
      </tr>
      <c:forEach items="${esami}" var="esame">
      <tr>
            <td>${esame.desc}</td>
            <td>${esame.data}</td>
            <td>${esame.superato}</td>
```

```
            <td><a type="button" href="/step32/delete-esame?id=$
{esame.id}">Delete</a></td>
      </tr>
      </c:forEach>
</table>

<div><a type="button" href="/step32/add-esame">Add</a></div>

</body>
</html>
```

# 4. Usiamo Spring Security

## 4.1. Aggiungere la dipendenza nel pom

Per usare Spring Security, prima di tutto aggiungere le dipendenze nel pom.xml:

**FILE pom.xml**

```
      ...

      <org.springframework.security-
version>4.0.1.RELEASE</org.springframework.security-version>

      ...

      <!-- Spring Security -->
      <dependency>
            <groupId>org.springframework.security</groupId>
            <artifactId>spring-security-web</artifactId>
            <version>${org.springframework.security-version}</version>
      </dependency>
      <dependency>
            <groupId>org.springframework.security</groupId>
            <artifactId>spring-security-config</artifactId>
            <version>${org.springframework.security-version}</version>
      </dependency>

      ...
```

## 4.2. Configurare Spring Security

Creare la classe **SecurityConfiguration**, che è sostanzialmente la configurazione di Spring Security. This configuration creates a Servlet Filter known as the springSecurityFilterChain which is responsible for all the security (protecting the application URLs, validating submitted username and passwords, redirecting to the log in form, etc) within our application.
Qui andremo a definire due aspetti:

1. Chi sono gli utenti validi (che possono accedere all'applicativo) e quali sono i loro ruoli. Questo è definito nel metodo **configeGlobalSecurity()**.
2. Quali sono gli url accessibili da chiunque, e quali quelli che devono subire un controllo di sicurezza. Questo è definito nel metodo c**onfige()**.

**SecurityConfiguration.java**

```
package com.spring.step32.security;
```

```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Configuration;
import
org.springframework.security.config.annotation.authentication.builders.Authenti
cationManagerBuilder;
import
org.springframework.security.config.annotation.web.builders.HttpSecurity;
import
org.springframework.security.config.annotation.web.configuration.EnableWebSecur
ity;
import
org.springframework.security.config.annotation.web.configuration.WebSecurityCon
figurerAdapter;

@Configuration
@EnableWebSecurity
public class SecurityConfiguration extends WebSecurityConfigurerAdapter {

   @Autowired
   public void configureGlobalSecurity(AuthenticationManagerBuilder auth)
                                                                  throws Exception {
      auth
            .inMemoryAuthentication()
                  .withUser("admin").password("admin").roles("USER", "ADMIN");
   }

   @Override
   protected void configure(HttpSecurity http) throws Exception {
      http
            .authorizeRequests()
                  .antMatchers("/login").permitAll()
                  .antMatchers("/", "/*esam*/**").access("hasRole('USER')")
                  .and()
            .formLogin();
   }

}
```

Le richieste passano prima di tutto per Spring Security: se l'utente non è loggato, allora verrà automaticamente rediretto alla pagina di login.

## 4.3. Abilitare Spring Security

Bisogna dire a Spring di eseguire Spring Security prima di ogni richiesta. E come si fa questo? Introducendo un filtro! Cioè aggiungeremo Spring Security come un filtro. In pratica, nel file web.xml configuriamo un filtro

**web.xml**

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://xmlns.jcp.org/xml/ns/javaee"
xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd" id="WebApp_ID"
version="3.1">
  <display-name>SpringStep32</display-name>
  <welcome-file-list>
```

```
    <welcome-file>index.html</welcome-file>
    <welcome-file>index.htm</welcome-file>
    <welcome-file>index.jsp</welcome-file>
    <welcome-file>default.html</welcome-file>
    <welcome-file>default.htm</welcome-file>
    <welcome-file>default.jsp</welcome-file>
  </welcome-file-list>

  <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>


  <!-- Spring Security -->
  <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>

</web-app>
```

## 4.4. Rivedere il LoginController

A questo punto la login viene gestita da Spring Security. Cioè la pagina di login che vediamo non è più la nostra login.jsp ma una pagina gestita interamente da Spring Security, così come tutta la logica del login è gestita ora da Spring Secuirity.
Per questo **i metodi che intercettano la url "/login"** non hanno più senso di esistere, e p**ossono essere rimossi. Così come può essere rimossa anche l'intera classe LoginService**.
Allo stesso tempo ne approfittiamo per raffinare il codice ed eliminare lo username scolpito nel codice: l'utente loggato ce lo facciamo dare da Spring Security con l'oggetto "principal".
Inoltre, dato che se la validazione delle credenziali ha successo Spring Security ci rediriga sulla root, aggiungiamo un metodo che intercetta la url "/" e che ci rimanda sulla welcome.jsp.

```
package com.spring.step32.login;

import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.stereotype.Controller;
import org.springframework.ui.ModelMap;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;

@Controller
@SessionAttributes("name")
```

```
public class LoginController {

	private String getLoggedInUserName() {
	  Object principal = SecurityContextHolder.getContext()
			  .getAuthentication().getPrincipal();

	  if (principal instanceof UserDetails)
		  return ((UserDetails) principal).getUsername();

	  return principal.toString();
    }


	@RequestMapping(value = "/", method=RequestMethod.GET)
	public String showWelcomePage(ModelMap model) {
		model.put("name", getLoggedInUserName());
		return "welcome";
	}
}
```

## 4.5. Aggiungiamo il Logout

Visto che ci siamo, aggiungiamo anche la funzionalità di Logout, sempre gestita da Spring Security. Creiamo quindi un Controller apposito per questo.

**LogoutController.java**

```
package com.spring.step32.login;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.springframework.security.core.Authentication;
import org.springframework.security.core.context.SecurityContextHolder;
import
org.springframework.security.web.authentication.logout.SecurityContextLogoutHan
dler;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;

@Controller
public class LogoutController {

	@RequestMapping(value = "/logout", method=RequestMethod.GET)
	public String logout(HttpServletRequest request, HttpServletResponse
response) {
		Authentication auth = SecurityContextHolder.getContext()
				.getAuthentication();

		if (auth != null) {
			new SecurityContextLogoutHandler().logout(request, response,
auth);
			// inoltre per essere sicuri, killiamo la sessione corrente
			request.getSession().invalidate();
		}

		// redirect to home page
		return "redirect:/";
```

```
	}
}
```

# 5. Aggiungiamo il logging con log4j

## 5.1. Aggiungere la dipendenza nel pom

Per usare log4j, prima di tutto aggiungere la dipendenza nel pom.xml:

**pom.xml**

```
		...

		<!-- Logging -->
		<dependency>
			<groupId>log4j</groupId>
			<artifactId>log4j</artifactId>
			<version>1.2.17</version>
		</dependency>

		...
```

## 5.2. Crea il file di properties

Creare il file log4j.properties oppure log4j.xml, e salvarlo dentro /WEB-INF/classes/

**log4j.properties**

```
log4j.rootLogger=INFO, console

log4j.appender.console=org.apache.log4j.ConsoleAppender
log4j.appender.console.layout=org.apache.log4j.PatternLayout
log4j.appender.console.layout.ConversionPattern=%-7p %d [%t] %c %x - %m%n

log4j.logger.org.springframework.security=DEBUG, console

#TRACE
#DEBUG
#INFO
#WARN
#ERROR
```

**Nota: tutto ciò che si mette dentro resources sarà disponibile nel classpath.**
**Invece tutto ciò che si trova dentro src/main/java è il codice sorgente.**

# 6. Gestione delle eccezioni

Quando creiamo un applicazione è molto importante gestire le eccezioni.
Una buona gestione delle eccezioni presuppone che:

- durante l'eccezione:
  - loggare l'eccezione
  - fornire all'utente un qualche riferimento dell'eccezione (tipicamente un ID) che lui possa usare successivamente per chiedere supporto

- dopo (team di supporto):
  - dato un ID, risalire facilmente al contesto dell'eccezione

Altre linee guida sono:
- non gestire una eccezione fino a che non sai cosa fare con essa
- non gestire eccezioni nei Controller o nei Service se non puoi dargli un valore aggiunto
- usare un **gestore globale** delle eccezioni che mostri un errore all'utente
- usare un **gestore specifico** solo per quelle eccezioni che gestiscono funzionalità di business

Il try-catch ormai è obsoleto e complicato da gestire.
In Spring MVC la gestione delle eccezioni può essere applicata a due livelli:
- **Global Exception Handling**: generale per tutti i Controller
  - basta creare una classe, annotandola con **@ControllerAdvice**, e implementare la gestione delle eccezioni con dei metodi annotati con **@ExceptionHandler**
  - tutto ciò che si trova all'interno di questa classe verrà applicata a tutti i Controller dell'applicazione.
- **Local Exception Handling**: specifica per un determinato Controller
  - basta creare un metodo per gestire l'eccezione all'interno dello specifico Controller, e annotare tale metodo con **@ExceptionHandler**

Attenzione: se un'eccezione viene sollevata a livello di JSP, nel browser verrà visualizzato tutto lo stack trace, che non è carino. Per gestire le eccezioni anche all'interno delle JSP basta aggiungere un pezzo di configurazione all'interno del file web.xml.

## 6.1. Gestione eccezioni a livello Globale

Creare una classe per gestire le eccezioni a livello globale, ossia per tutti i Controller. La gestione avviene loggando l'eccezione e riportando l'utente ad una pagina di errore *generica*.

**GlobalExceptionController.java**

```
package com.spring.step32.exception;

import javax.servlet.http.HttpServletRequest;

import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;
import org.springframework.web.bind.annotation.ControllerAdvice;
import org.springframework.web.bind.annotation.ExceptionHandler;

@ControllerAdvice
public class GlobalExceptionController {

      private Log logger = LogFactory.getLog(GlobalExceptionController.class);


    @ExceptionHandler(value = Exception.class)
    public String handleError(HttpServletRequest req, Exception exception) {
        logger.error("Request: " + req.getRequestURL() + " Raised an exception:
", exception);
        return "error";
    }

}
```

### 6.1.1. Creare la jsp error.jsp

**error.jsp**

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!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=UTF-8">
<title>Insert title here</title>
</head>
<body>

<%@ include file="common/navigation.jspf"%>

<div>Application has encountered an error.</div>
<div>Please contact support on ...</div>

</body>
</html>
```

## 6.2. Gestione eccezioni a livello Locale

Supponiamo di voler gestire le eccezioni relative agli esami. Bene, basta ggiungere un metodo per gestire l'eccezione all'interno di EsameController e annotarlo con @ExceptionHandler.
La gestione avviene loggando l'eccezione e riportando l'utente ad una pagina di errore *specifica* per gli esami.

### 6.2.1. Creare la jsp error-esame.jsp

**error-esame.jsp**

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!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=UTF-8">
<title>Insert title here</title>
</head>
<body>

<%@ include file="common/navigation.jspf"%>

<div>SPECIFIC Exception Page.</div>
<div>Please contact support on ...</div>

</body>
</html>
```

## 6.3. Gestione eccezioni a livello di JSP

Occorre aggiungere la configurazione della pagina di errore nel file web.xml.
Facciamo in modo che se una jsp solleva una eccezione, all'utente verrà visualizzata la pagina generica error.jsp.

**web.xml**

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://xmlns.jcp.org/xml/ns/javaee"
xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd" id="WebApp_ID"
version="3.1">
  <display-name>SpringStep32</display-name>
  <welcome-file-list>
    <welcome-file>index.html</welcome-file>
    <welcome-file>index.htm</welcome-file>
    <welcome-file>index.jsp</welcome-file>
    <welcome-file>default.html</welcome-file>
    <welcome-file>default.htm</welcome-file>
    <welcome-file>default.jsp</welcome-file>
  </welcome-file-list>

  <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>


  <!-- Spring Security -->
  <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>

  <!-- JSP Exception Handling -->
  <error-page>
      <location>/WEB-INF/views/error.jsp</location>
  </error-page>

</web-app>
```

## 7. Aggiungere la parte grafica

Per aggiungere la parte grafica all'applicativo (css, js, jquery, ecc) guarda il video
https://www.youtube.com/watch?v=thME7hC3hIQ