SPRING SECURITY

Instructor:

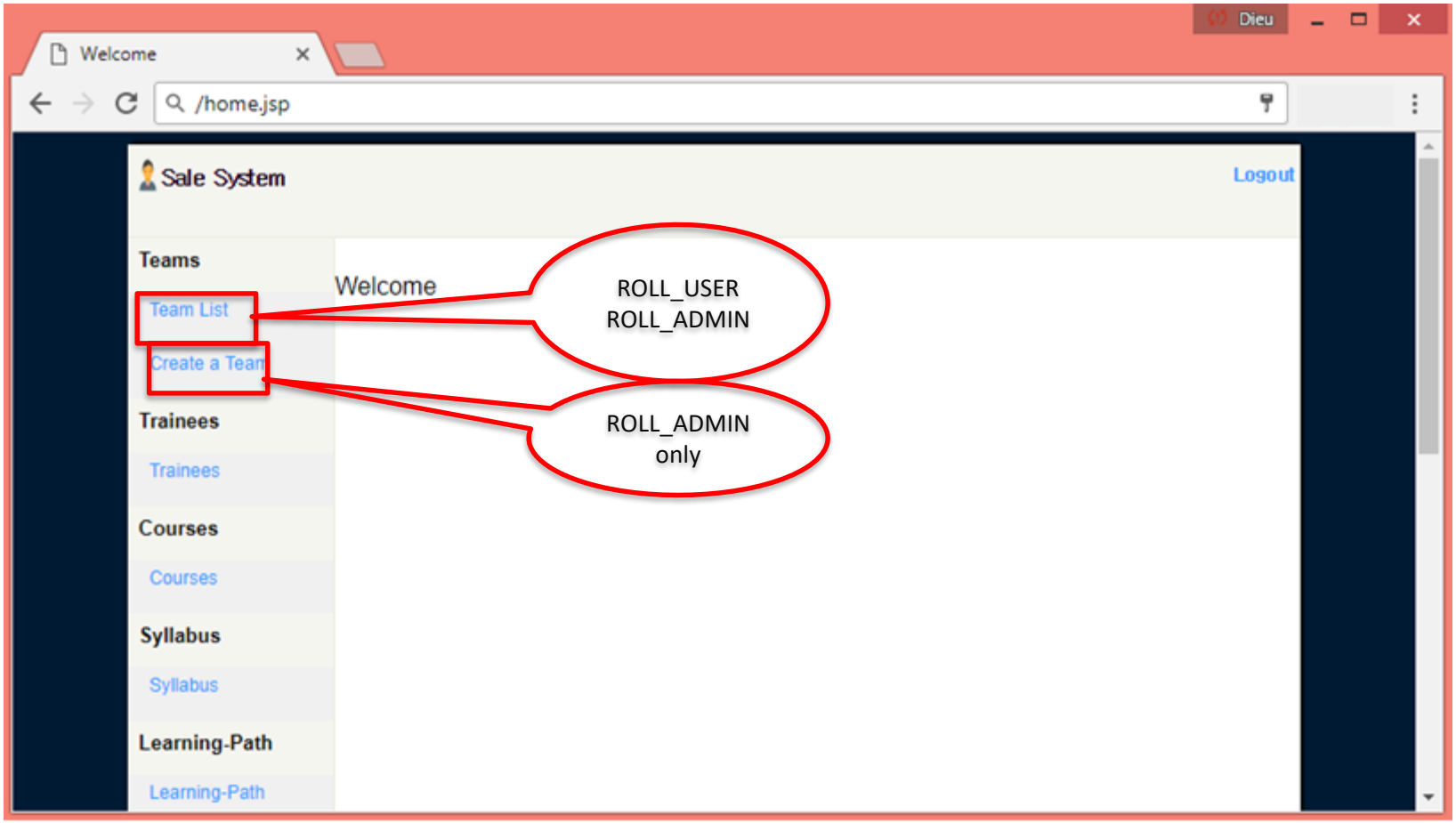
Table Content

Learning Goals

After the course, attendees will be able to:

Understand Spring Security Framework and its core technologies.
Know how to write a Web application with Spring Security.

Introduction



Introduction

- **Spring security** is another major module in spring distribution and is supported only for applications developed using JDK 1.5 or higher.
- **Spring Security** is a framework that focuses on providing both **authentication** and **authorization** to Java EE-based enterprise software applications.
- **Spring security** has been divided into **multiple jars** and you should include them as your application need. Only the core module available in **spring-security-core.jar** is mandatory.

Introduction

- **spring-security-core**

  It contains core authentication and access-contol classes and interfaces.

- **spring-security-web**

  It contains filters and related web-security infrastructure code. It also enable URL based security which we are going to use in this demo.

- **spring-security-config**

  It contains the security namespace parsing code. You need it if you are using the Spring Security XML file for configuration.

- **spring-security-taglibs**

  It provides basic support for accessing security information and applying security constraints in JSPs.

Authentication and Authorization

- It also provides authentication at **view level** and **method level.**
- It can also provide you with a login page! Here are some things that it provides:
  - ✓ Provide capabilities for **login** and **logout**
  - ✓ Control access to a **link based on the role of the user.**
  - ✓ Provide the ability to **hide certain portion of a page** if a user does not have appropriate privileges.
  - ✓ Link to a database for authentication

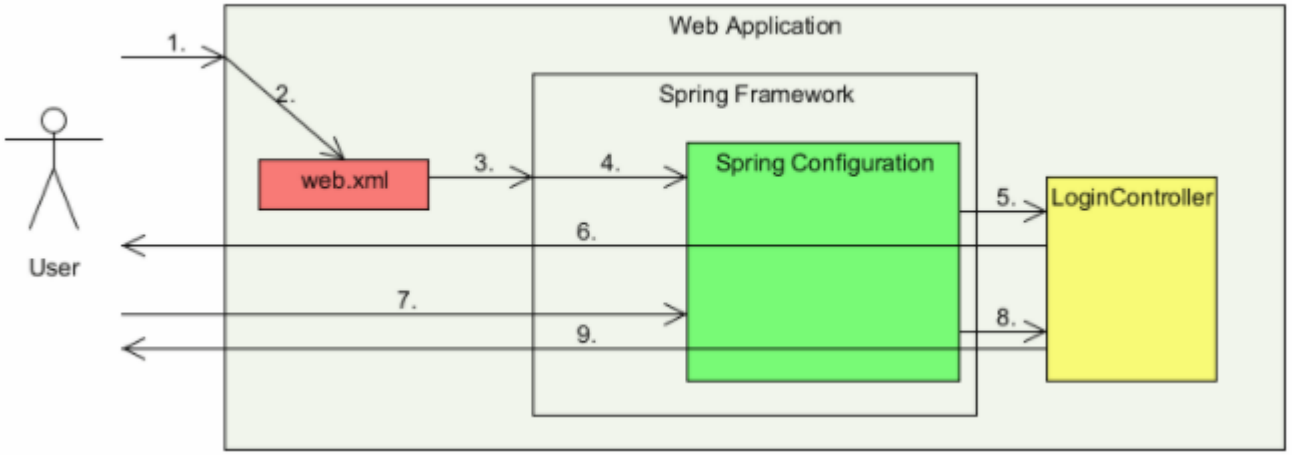
The Security Filter Chain

- Spring Security's web infrastructure is based entirely on standard servlet filters.
- These filters are defined in web.xml file:

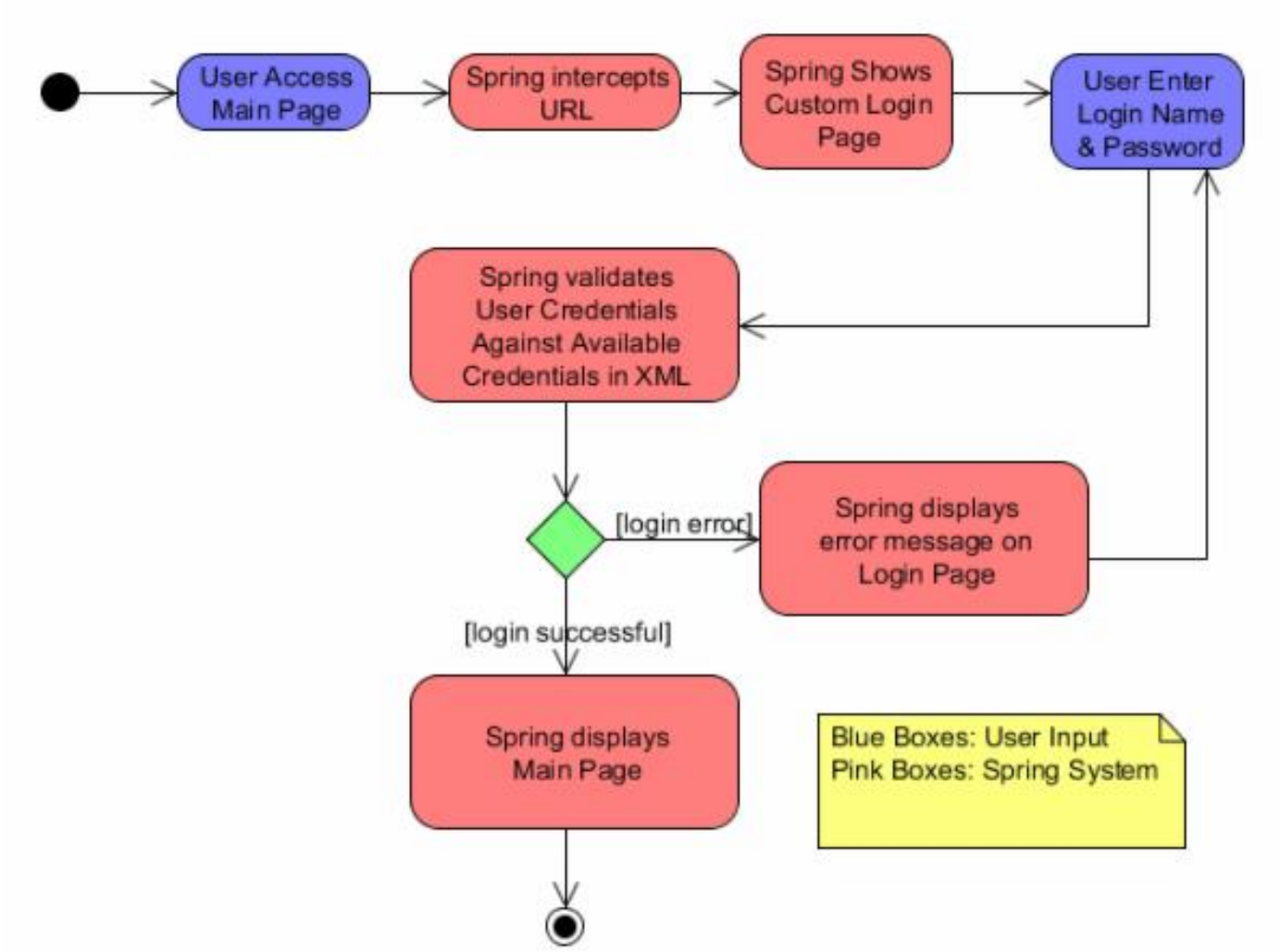
```
<filter>
  <filter-name>myFilter</filter-name>
  <filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-class>
</filter>

<filter-mapping>
  <filter-name>myFilter</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>
```

Interaction Flow

1. User accesses a URL on a web application
2. The web application refers to web.xml
3. The web.xml matches the URL pattern
4. The control is redirected to ***DispatcherServlet*** in Spring framework
5. Spring framework finds that the all **URLs are secured** and hence displays login page to the user
6. The user enters his login name and password
7. Spring framework validates the login name and password by using the entries in Spring configuration XML and redirects to the accessed original URL



Interaction Flow Detail



Login Form Example

- **Only authorized user** should be able to access home screen.
- **Unauthorized users** should be presented with login screen.
  - ✓ **Successful credentials** should forward to home screen.
  - ✓ **Unsuccessful credentials** should forward to access denied screen.
- **There** should be **a link for logout** of the application.

(1) Update project dependencies in pom.xml

```
<security.version>4.2.3.RELEASE</security.version>
// ..
<!--Security -->
<dependency>
	<groupid>org.springframework.security</groupid>
	<artifactid>spring-security-core</artifactid>
	<version>${spring.version}</version>
	<type>jar</type>
	<scope>compile</scope>
</dependency>
<dependency>
	<groupid>org.springframework.security</groupid>
	<artifactid>spring-security-web</artifactid>
	<version>${spring.version}</version>
	<type>jar</type>
	<scope>compile</scope>
</dependency>
```

```
<dependency>
	<groupid>org.springframework.security</groupid>
	<artifactid>spring-security-config</artifactid>
	<version>${spring.version}</version>
	<type>jar</type>
	<scope>compile</scope>
</dependency>
<dependency>
	<groupid>org.springframework.security</groupid>
	<artifactid>spring-security-taglibs</artifactid>
	<version>${spring.version}</version>
	<type>jar</type>
	<scope>compile</scope>
</dependency>
```

(2) Configure DelegatingFilterProxy in web.xml

spring-servlet.xml | web.xml

```
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns="http://java.sun.com/xml/ns/javaee"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
    version="3.0">
    <display-name>Archetype Created Web Application</display-name>

    <welcome-file-list>
        <welcome-file>/views/login.jsp</welcome-file>
    </welcome-file-list>

    <servlet>
        <servlet-name>spring</servlet-name>
        <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
        <init-param>
                    <param-name>contextConfigLocation</param-name>
                    <param-value> /WEB-INF/spring-servlet.xml </param-value>
        </init-param>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>spring</servlet-name>
        <url-pattern>/</url-pattern>
    </servlet-mapping>

    <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>
```

(3) Add security configuration

- Namespace:

  xmlns:security="http://www.springframework.org/schema/security"

- and

  http://www.springframework.org/schema/security
  http://www.springframework.org/schema/security/spring-security.xsd

```
74      <!--SECURITY  -->
75      <security:http auto-config="true"  use-expressions="true">
76          <security:csrf disabled="true"/>
77          <security:intercept-url pattern="/teams" access="hasRole('ROLE_USER')" />
78          <security:intercept-url pattern="/addTeam*" access="hasAnyRole(ROLE_ADMIN)" />
79          <security:intercept-url pattern="/home" access="hasAnyRole('ROLE_ADMIN,ROLE_USER')" />
80          <security:intercept-url pattern="/accessdenied" access="permitAll" />
81          <security:form-login    login-page="/views/login.jsp"
82                                  login-processing-url="/appLogin"
83                                  default-target-url="/home"
84                                  username-parameter="username" password-parameter="password"
85                                  authentication-failure-url="/gotologin" />
86          <security:logout logout-success-url="/logout" />
87      </security:http>
88
89      <security:authentication-manager>
90          <security:authentication-provider>
91              <security:user-service>
92                  <security:user name="admin" password="admin" authorities="ROLE_ADMIN" />
93                  <security:user name="sa" password="sa" authorities="ROLE_USER" />
94              </security:user-service>
95          </security:authentication-provider>
96      </security:authentication-manager>
```

(3) Add security configuration

- **http**: Include configuration related url level security. This element is the parent for all web-related namespace functionality.
- **auto-config**: Inclused some basic services. It is shorthand for –
  - ✓ <http>
  - ✓ <form-login />
  - ✓ <http-basic />
  - ✓ <logout />
  - ✓ </http>
- **use-expressions**: It is here to use expressions to secure individual URLs. These expressions can be e.g. hasRole([role]), hasAnyRole([role1,role2]), permitAll, denyAll etc.

Explain: Common Built-In Expressions

- The base class for expression root objects is ***SecurityExpressionRoot***. This provides some common expressions which are available in both web and method security.

```
<http auto-config="true"  use-expressions="true">
```

Expression	Description
hasRole([role])	Returns true if the current principal has the specified role.
hasAnyRole([role1,role2])	Returns true if the current principal has any of the supplied roles (given as a comma-separated list of strings)
principal	Allows direct access to the principal object representing the current user
authentication	Allows direct access to the current Authentication object obtained from the SecurityContext
permitAll	Always evaluates to true
denyAll	Always evaluates to false
isAnonymous()	Returns true if the current principal is an anonymous user
isRememberMe()	Returns true if the current principal is a remember-me user
isAuthenticated()	Returns true if the user is not anonymous
isFullyAuthenticated()	Returns true if the user is not an anonymous or a remember-me user

Explain: Web Security Expressions

- To use expressions to secure individual **URLs,** you would first need to set the ***use-expressions*** attribute in the ***<http>*** element to **true.**

```
<http auto-config="true" use-expressions="true">
        <intercept-url pattern="/login" access="permitAll" />
        <intercept-url pattern="/logout" access="permitAll" />
</http>
```

- **Spring Security** will then expect the ***access*** attributes of the ***<intercept-url>*** elements to contain Spring EL expressions.
- Example:

```
<http use-expressions="true">
    <intercept-url pattern="/admin*"
        access="hasRole('admin') and hasIpAddress('192.168.1.0/24')"/>
    ...
</http>
```

(3) Add security configuration

- **intercept-url**: This will match the requested url pattern from request and will decide what action to take based on access value.
- **form-login:** This will come into picture **when user will try to access any secured URL**. A login page mapped to "*login-page*" attribute will be served for authentication check. If not provided, spring will provide an inbuilt login page to the user. It also contains an attribute for default target if login success, or login failure due to invalid.
- **access-denied-handler:** redirect the user to other page when "**Etat HTTP 403 - Access is denied**";
- **logout:** This will help to find the next view if logout is called in the application.

Spring Security with JDBC

Create 2 tables:

```
CREATE TABLE [dbo].[Users](
    [user_id] [int] IDENTITY(1,1) PRIMARY KEY NOT NULL,
    [user_name] [varchar](50) NOT NULL,
    [email] [varchar](50) NOT NULL,
    [password] [varchar](50) NOT NULL,
    [enabled] [tinyint] NOT NULL,
    [passwords] [varchar](255) NULL,
    [users_name] [varchar](255) NULL,
)

CREATE TABLE [dbo].[Users_Roles](
    [user_role_id] [int] PRIMARY KEY NOT NULL,
    [user_id] [int] NOT NULL,
    [authority] [varchar](45) NOT NULL,
    CONSTRAINT uni_user UNIQUE([authority], [user_id]),
    CONSTRAINT fk_user FOREIGN KEY ([user_id]) REFERENCES dbo.Users([user_id])
)
```

Spring Security with JDBC

- Some initial data to start with:

```
INSERT [dbo].[Users] ([user_id], [user_name], [email], [password], [enabled])
VALUES   (1, N'dieunguyen', N'dieun@fsoft.com.vn', N'12345', 1),
         (2, N'dieunt1', N'dieunt1@fsoft.com.vn ', N'sa', 1),
         (3, N'admin', N'admin@fsoft.com', N'admin', 1)


INSERT INTO dbo.Users_Roles([user_role_id], [user_id], [authority])
VALUES (100, 2, 'ROLE_ADMIN'),
       (101, 3, 'ROLE_ADMIN'),
       (102, 1, 'ROLE_USER');
```

Update security configuration

```
<security:authentication-manager>
    <security:authentication-provider>
        <security:jdbc-user-service
            data-source-ref="dataSource"
            users-by-username-query="select user_name, password, enabled from USERS
                                     where user_name = ?"
            authorities-by-username-query="select u.user_name, ur.authority from
                        USERS u, USERS_ROLES ur where u.user_id = ur.user_id
                        and u.user_name = ? " />

    </security:authentication-provider>
</security:authentication-manager>
```

SUMMAY

1. Introduction
2. Authentication and Authorization
3. The Security Filter Chain
4. Interaction Flow
5. Spring Security with JDBC

Thank you