Bộ môn: Kỹ Thuật Phần Mềm

Lập trình WWW *Java*

.

Spring MVC - Security

Kỹ thuật Phần mềm – Phạm Quảng Tri

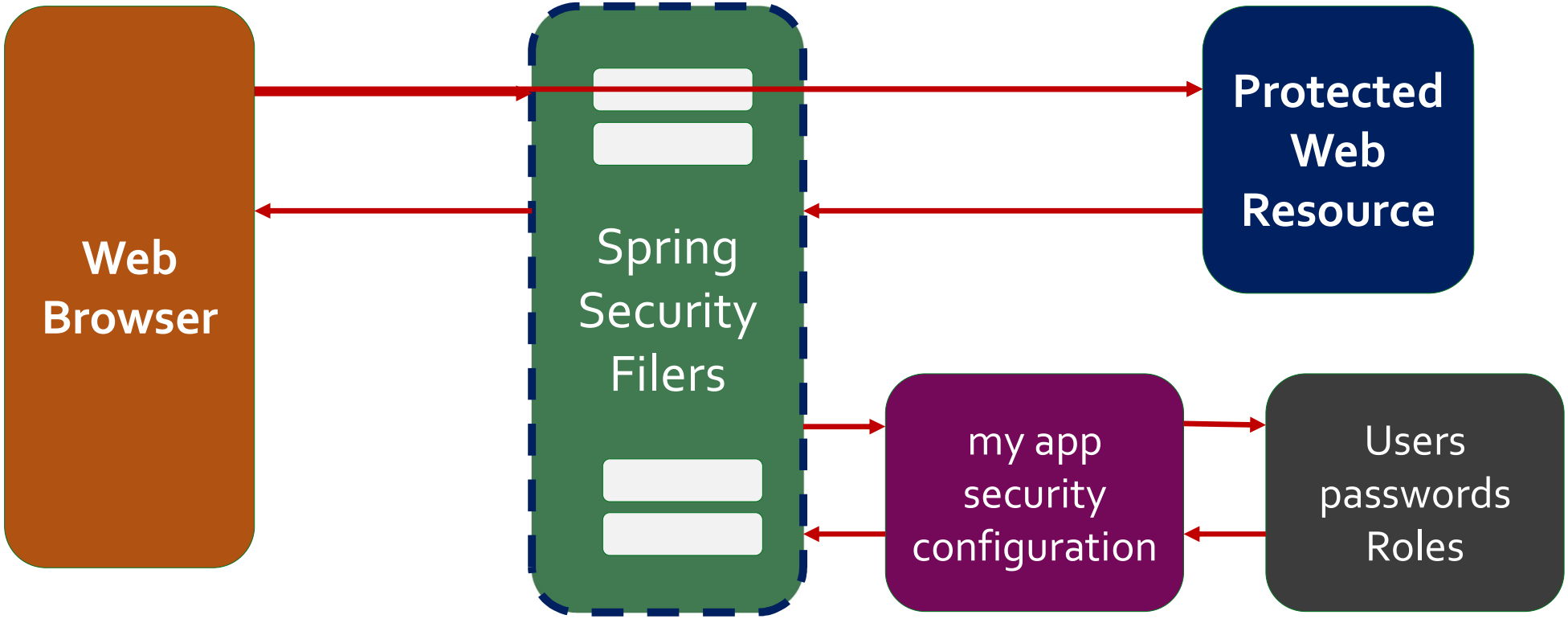
Spring Security Model

- Spring Security **defines a framework** for **security**
- Implemented **using Servlet filters** in the **background**
- Two methods of securing a Web app: **declarative** and **programmatic**

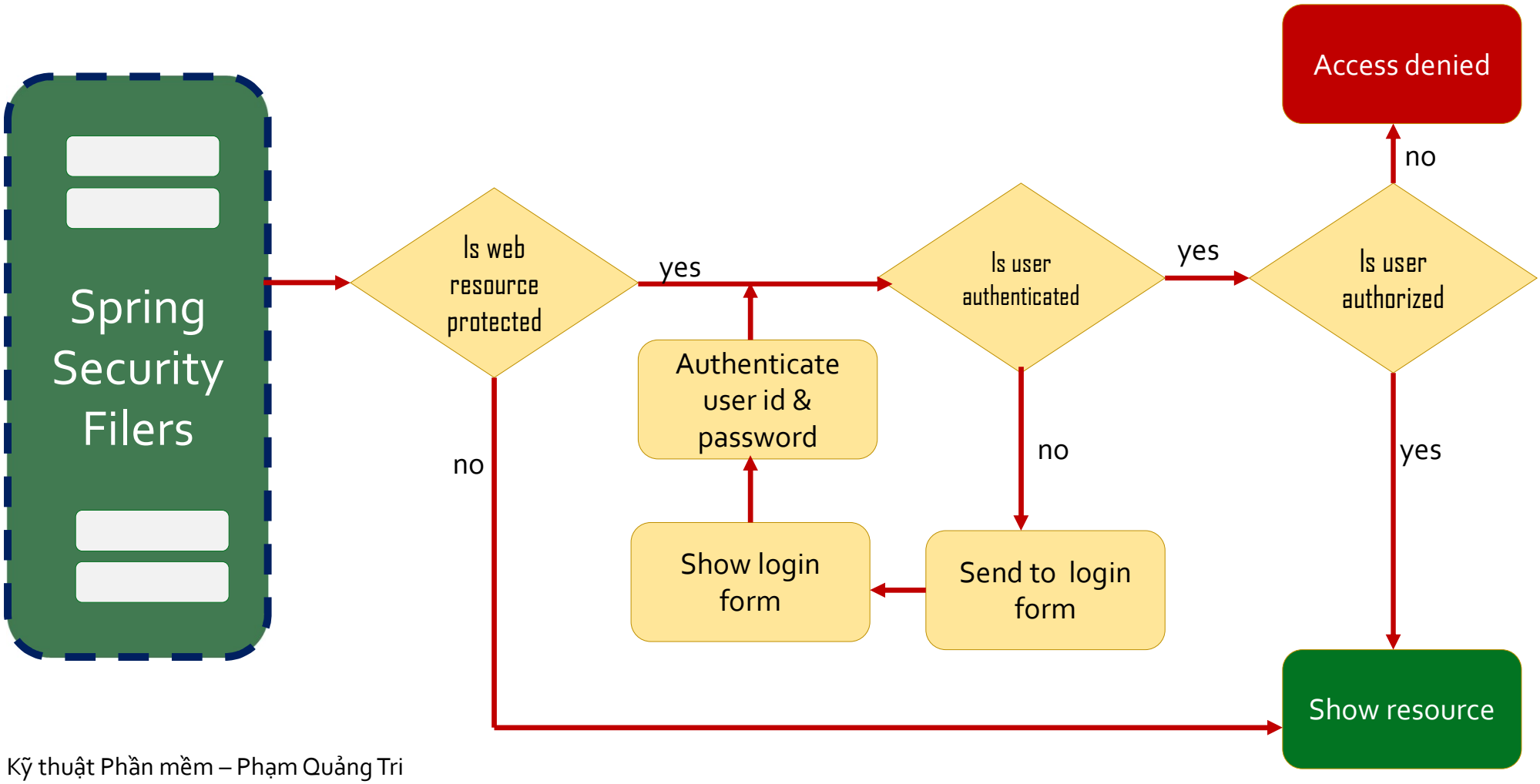
Spring Security with Servlet Filters

- Servlet Filters are used to **pre-process** / **post-process** web requests
- Servlet Filters can **route web requests** based on security logic
- Spring provides a bulk of **security functionality** with **servlet filters**

Spring Security Overview



Spring Security Overview



Security Concepts

- **Authentication**: Check **user id** and **password** with credentials stored in app/db
- **Authorization**: Check to see if user has an **authorized role**

Declarative Security

» Define application's security constraints in configuration

- **All Java config** (@Configuration, no xml)

  or

- **Spring XML config**

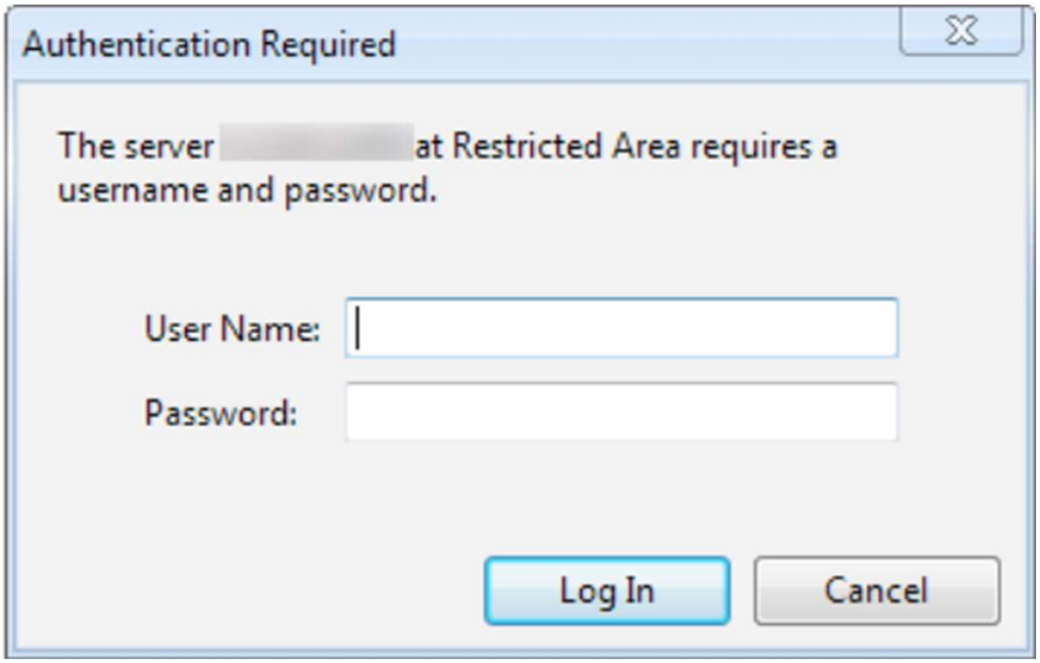
» Provides **separation of concerns** between **application code** and **security**

Different Login Methods

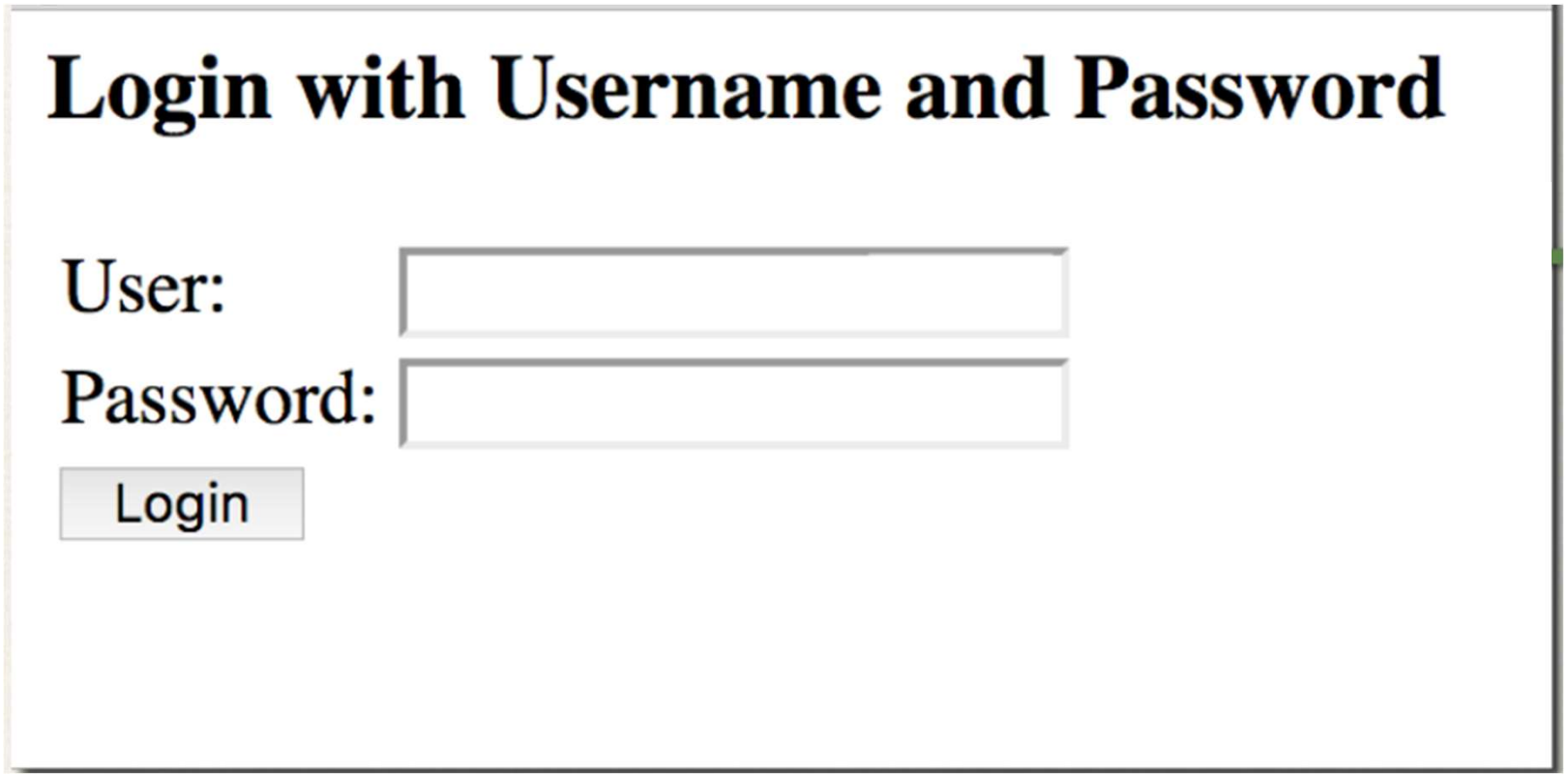
- HTTP Basic Authentication
- **Default login form**: Spring Security provides a default login form
- **Custom login form**: your own look-and-feel, HTML + CSS

HTTP Basic Authentication

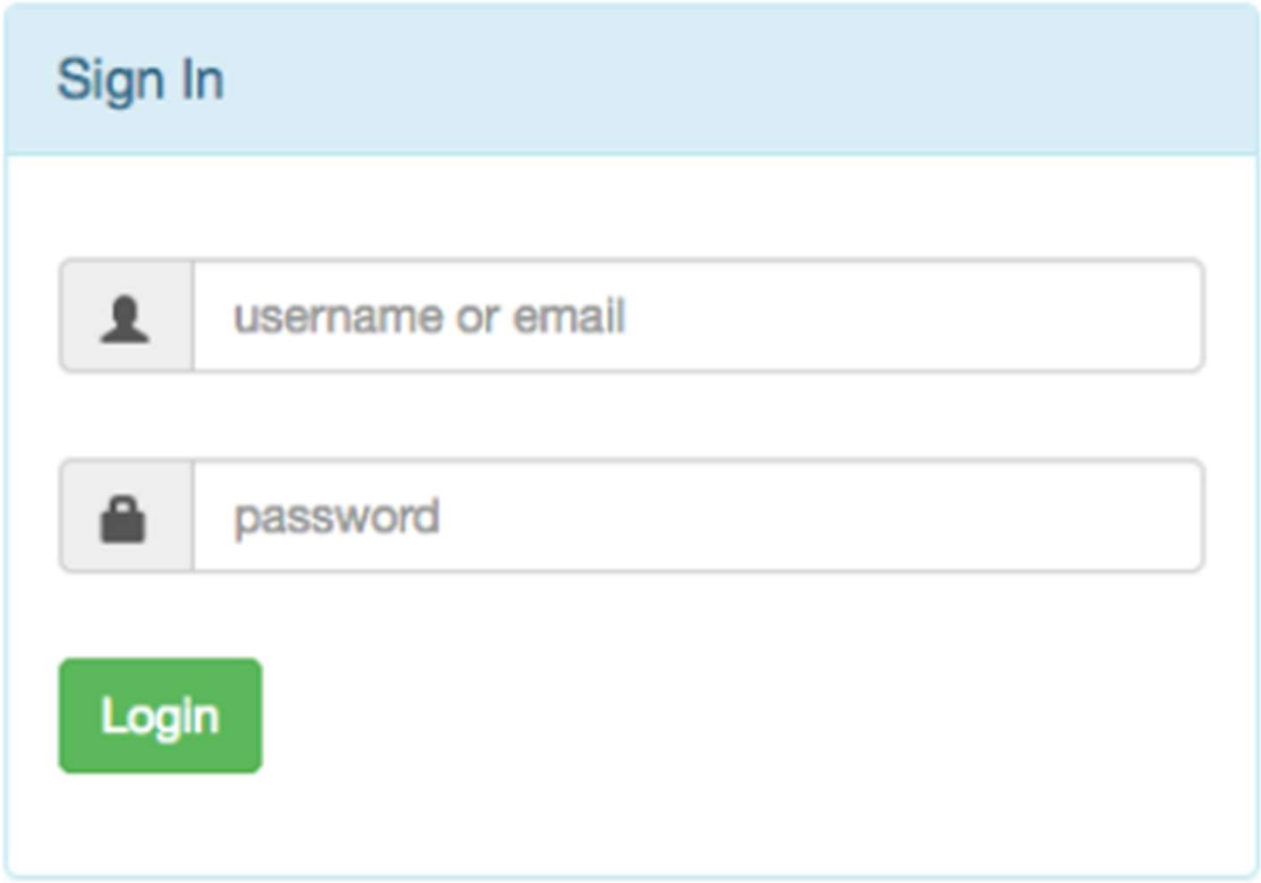
Build-in dialog from browser



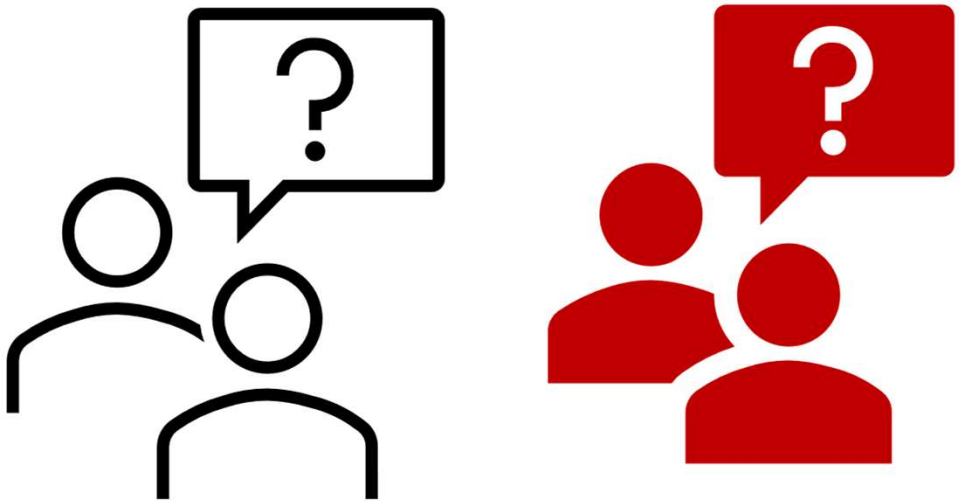
Spring Security - Default Login Form



Your Own Custom Login Form



Finding Compatible version



Add Spring Security Maven Dependencies

```
<properties>
    <springframework.version>5.2.11.RELEASE</springframework.version>
    <springsecurity.version>5.2.8.RELEASE</springsecurity.version>
    <maven.compiler.source>1.8</maven.compiler.source>
    <maven.compiler.target>1.8</maven.compiler.target>
</properties>
```

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

Basic Security

Project: spring-security-demo-basic-security

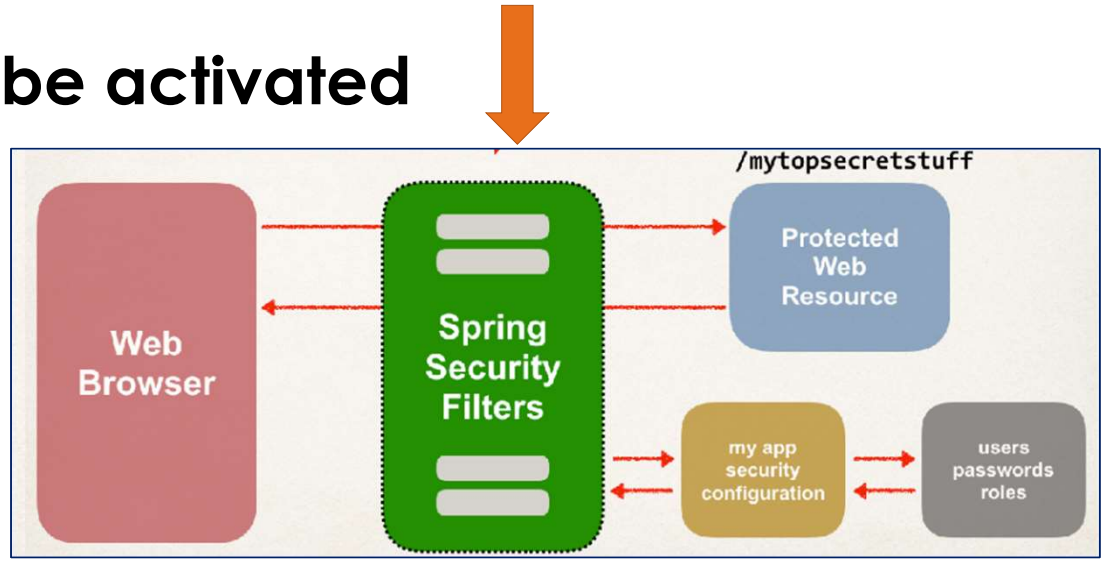
Development Process

1. Create Spring **Security Initializer**
2. Create **Spring Security Configuration** (@Configuration) then add **users**, **passwords** and **roles**

Step 1: Create Spring Security Initializer

» **Spring Security Initializer** is used for **triggering**, or **enabling**, the **Spring Security Filters**

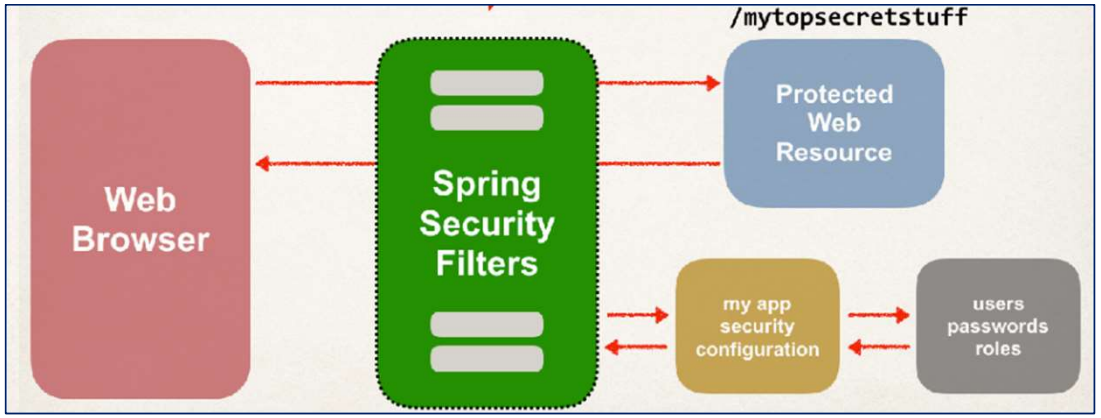
» Without it, the **Spring Security Filters will not be registered, they will not be activated**



Step 1: Create Spring Security Initializer

File: SecurityWebApplicationInitializer.java:

```
import org.springframework.security.web.context.AbstractSecurityWebApplicationInitializer;
public class SecurityWebApplicationInitializer
        extends AbstractSecurityWebApplicationInitializer {
}
```



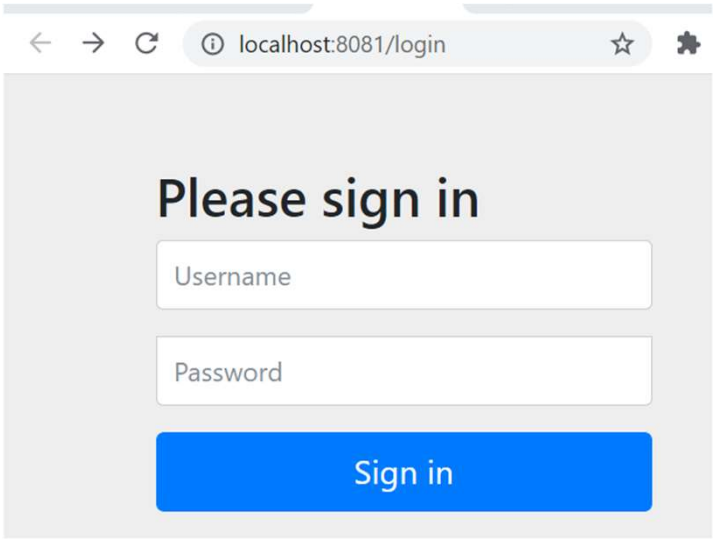
Step 2: Create Spring Security Configuration…

File: DemoSecurityConfig.java:

```
@Configuration
@EnableWebSecurity
public class DemoSecurityConfig extends WebSecurityConfigurerAdapter {
    @Override
    protected void configure(AuthenticationManagerBuilder auth) throws Exception {
            // add our users for in memory authentication
        UserBuilder users = User.withDefaultPasswordEncoder();
        auth.inMemoryAuthentication()
                .withUser(users.username("john").password("test123").roles("EMPLOYEE"))
                .withUser(users.username("mary").password("test123").roles("MANAGER"))
                .withUser(users.username("susan").password("test123").roles("ADMIN"));    }
}
```

Run Project

```
@Configuration
@EnableWebSecurity
public class DemoSecurityConfig extends WebSecurityConfigurerAdapter {
   @Override
   protected void configure(AuthenticationManagerBuilder auth) throws Exception {
         // add our users for in memory authentication
      UserBuilder users = User.withDefaultPasswordEncoder();
     auth.inMemoryAuthentication()
            .withUser(users.username("john").password("test123").roles("EMPLOYEE"))
            .withUser(users.username("mary").password("test123").roles("MANAGER"))
            .withUser(users.username("susan").password("test123").roles("ADMIN"));    }
}
```

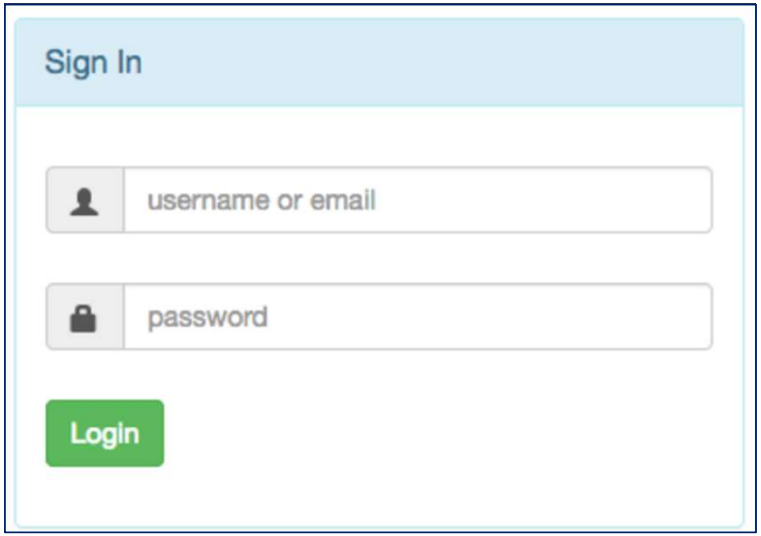


Spring Security - Custom Login Form

(Project spring-security-demo-custom-login-form-security)

Development Process

- Modify **Spring Security Configuration** to **reference custom login form**
- Develop a **Controller** to **show** the **custom login form**
- Create **custom login form**
  - ✤ HTML (CSS optional)
  - ✤ Spring MVC form tag <form:form>



Step 1: Modify Spring Security Configuration …

File: DemoSecurityConfig.java

```
@Configuration
@EnableWebSecurity
public class DemoSecurityConfig extends WebSecurityConfigurerAdapter {
    @Override
    protected void configure(AuthenticationManagerBuilder auth) throws Exception {...7 lines }
```

Method Description	Method Description
configure(AuthenticationManagerBuilder)	Configure users (in memory, database, ldap, etc)
configure(HttpSecurity)	Configure security of web paths in application, login, logout etc

Step 1: Modify Spring Security Configuration …

File: DemoSecurityConfig.java

```
@Configuration
@EnableWebSecurity
public class DemoSecurityConfig extends WebSecurityConfigurerAdapter {
@Override
    protected void configure(AuthenticationManagerBuilder auth) throws Exception {...7 lines }
@Override
protected void configure(HttpSecurity http) throws Exception {
    http.authorizeRequests()
        .anyRequest().authenticated()
        .and()
        .formLogin()
                .loginPage("/showMyLoginPage")
                .loginProcessingUrl("/authenticateTheUser")
                .permitAll(); }
}
```

Configure security of web paths

access based on
pServletRequest

Any request to the app
must be authenticated
(

We are customizing the form
login process

Login form should post data to this URL for processing (check user id & password)

Step 1: Modify Spring Security Configuration …

Login processing URL will be handled by Spring Security Filters

File: DemoSecurityConfig.jav

```
protected void configure( Security http) throws E
    http.authorizeRequests
        .anyRequest().authen ted()
        .and()
        .formLogin()
            .loginPage("/showMyLoginPage")
            .loginProcessingUrl("/authenticateTheUser")
            .permitAll(); }
}
```

You can give any values for these configuration- Just stay consistency in your app

Step 1: Modify Spring Security Configuration …

File: DemoSecurityConfig.java:

```
protected void configure(HttpSecurity http) throws Exception {
    http.authorizeRequests()
        .anyRequest().authenticated()
        .and()
        .formLogin()
            .loginPage("/showMyLoginPage")
            .loginProcessingUrl("/authenticateTheUser")
            .permitAll(); }
}
```

Step 2: Develop a Controller to show the custom login form

File: LoginController.java

```
@Controller
public class LoginController {
    @GetMapping("/showMyLoginPage")
    public String showMyLoginPage() {
        return "plain-login"; }
}
```

```
@Override
protected void configure(HttpSecurity http) throws Exception {
    http.authorizeRequests().anyRequest().authenticated()
        .and().formLogin()
            .loginPage("/showMyLoginPage")
            .loginProcessingUrl("/authenticateTheUser")
            .permitAll(); }
```

View name

/WEB-INF/view/plain-login.jsp

Step 3: Create custom login form

File: plain-login.jsp

```
<form:form action="${pageContext.request.contextPath}/authenticateTheUser"
           method="POST">
    ...
</form:form>
```

Must POST the data

```
@Override
protected void configure(HttpSecurity http) throws Exception {
    http.authorizeRequests().anyRequest().authenticated()
        .and().formLogin()
            .loginPage("/showMyLoginPage")
            .loginProcessingUrl("/authenticateTheUser")
            .permitAll(); }
```

Step 3: Create custom login form

» Spring Security defines **default names** for login form fields

- User name field: **username**
- Password field: **password**

File: plain-login.jsp

```
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form" %>
...
<form:form action="${pageContext.request.contextPath}/authenticateTheUser"
           method="POST">

  <p>
    User name: <input type="text" name="username" />
  </p>

  <p>
    Password: <input type="password" name="password" />
  </p>

  <input type="submit" value="Login" />

</form:form>
```

Step 3: Create custom login form – Fail login

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

```
<!-- Check for login error -->
<c:if test="${param.error != null}">
    <i class="failed">Sorry! You entered invalid username/password.</i>
</c:if>
```

Bootstrap, Css, Jquery for Login Form

» Use CDN:

Edit file Jsp (fancy-login.jsp)

```
<link rel="stylesheet"
    href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css">
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.2.0/jquery.min.js"></script>
<script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js"></script>
```

Bootstrap, Css, Jquery for Login Form

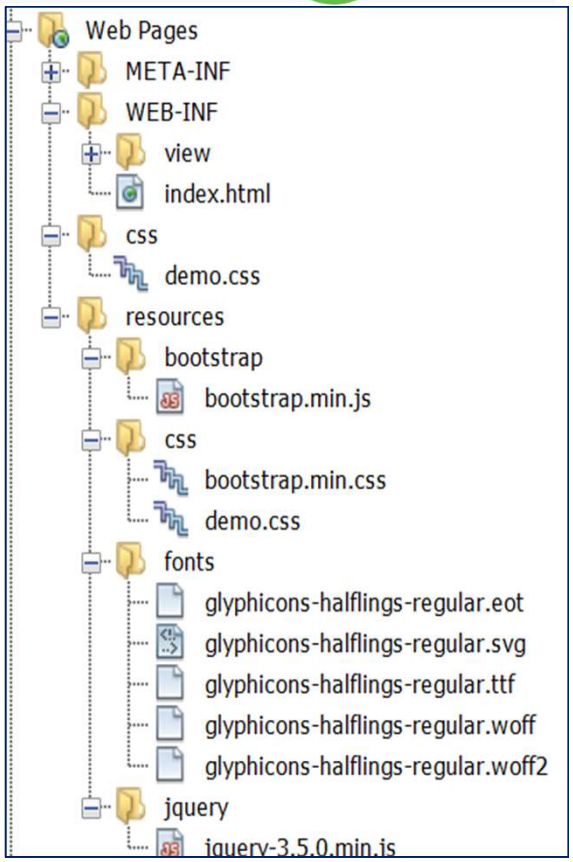
Use Static resource: [1]

- Copy css. Bootstrap, jquery files to folders
- Override method **addResourceHandlers** (file DemoAppConfig.java)

```
public void addResourceHandlers(final ResourceHandlerRegistry registry) {
   registry.addResourceHandler("/resources/**").addResourceLocations("/resources/");
}
```

XML config

```
<!-- Add support for reading web resources: css, images, js, etc ... -->
        <mvc:resources location="/resources/" mapping="/resources/**"></mvc:resources>
```



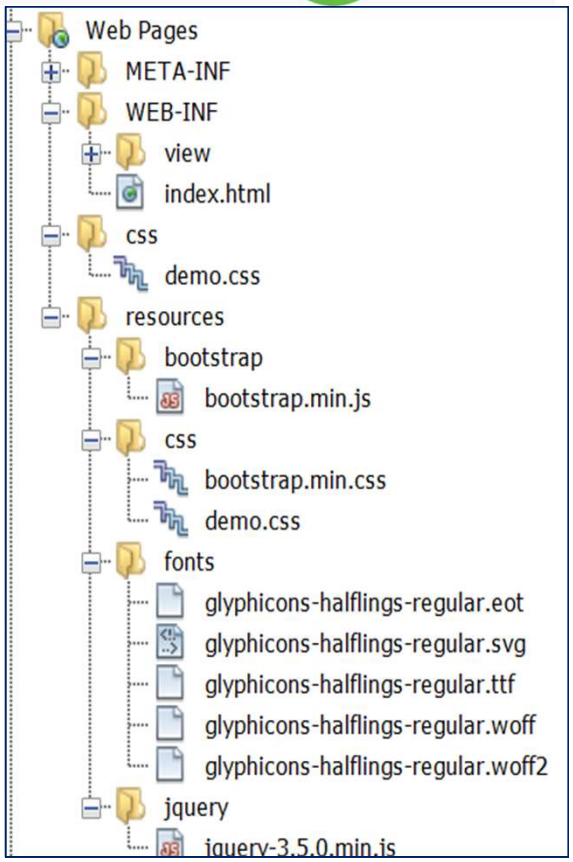
Bootstrap, Css, Jquery for Login Form

Use Static resource: [2]

» Edit file **DemoSecurityConfig.java** to allow **unauthenticated requests** (permit all) to the **"/resources/** " directory**

»

```
protected void configure(HttpSecurity http) throws Exception {
    http.authorizeRequests()
        .antMatchers("/resources/css/**").permitAll()
        .antMatchers("/resources/bootstrap/**").permitAll()
        .antMatchers("/resources/jquery/**").permitAll()
        .antMatchers("/resources/fonts/**").permitAll()
        .anyRequest().authenticated()
        .and().formLogin()
            .loginPage("/showMyLoginPage")
            .loginProcessingUrl("/authenticateTheUser")
            .permitAll(); }
```

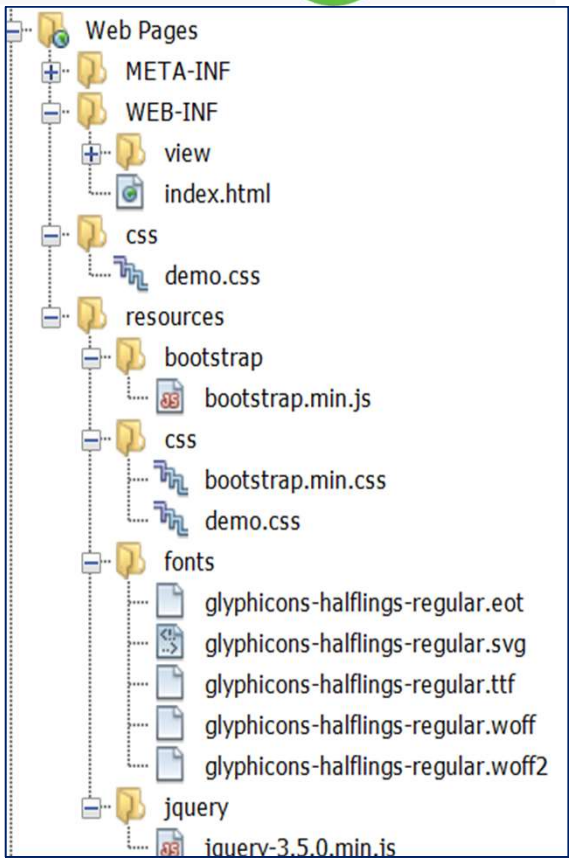


Bootstrap, Css, Jquery for Login Form

Use Static resource: [3]

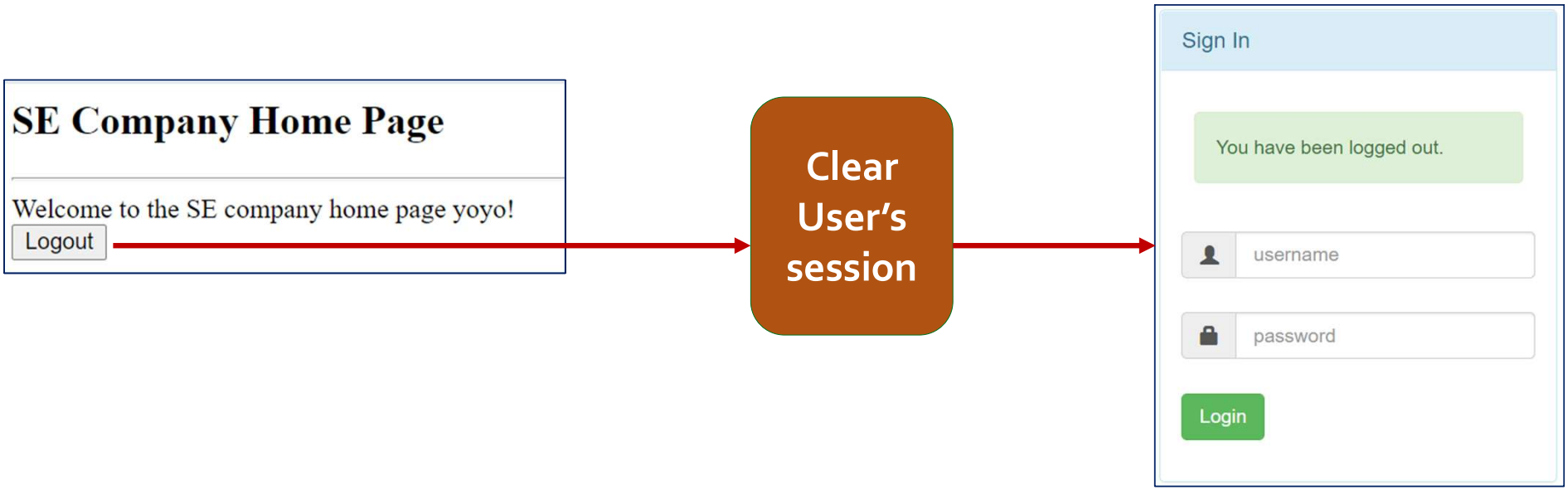
» Edit jsp file (fancy-login-local-css.jsp):

```
<!-- Reference Bootstrap files -->
<link href="${pageContext.request.contextPath}/resources/css/bootstrap.min.css"  rel="stylesheet" type="text/css">
<script src="<c:url value='/resources/jquery/jquery-3.5.0.min.js' />" ></script>
<script src="<c:url value='/resources/bootstrap/bootstrap.min.js' />" ></script>
```



Spring Security – Logging Out
(project: spring-security-demo-custom-logout)

Logging Out


SE Company Home Page
Welcome to the SE company home page yoyo!
Logout
Clear User's session
Sign In
You have been logged out.
username
password
Login

Development Process

- Add **logout support** to **Spring Security Configuration**
- Add **logout button** to JSP page
- Update **login form** to display "**logged out**" message

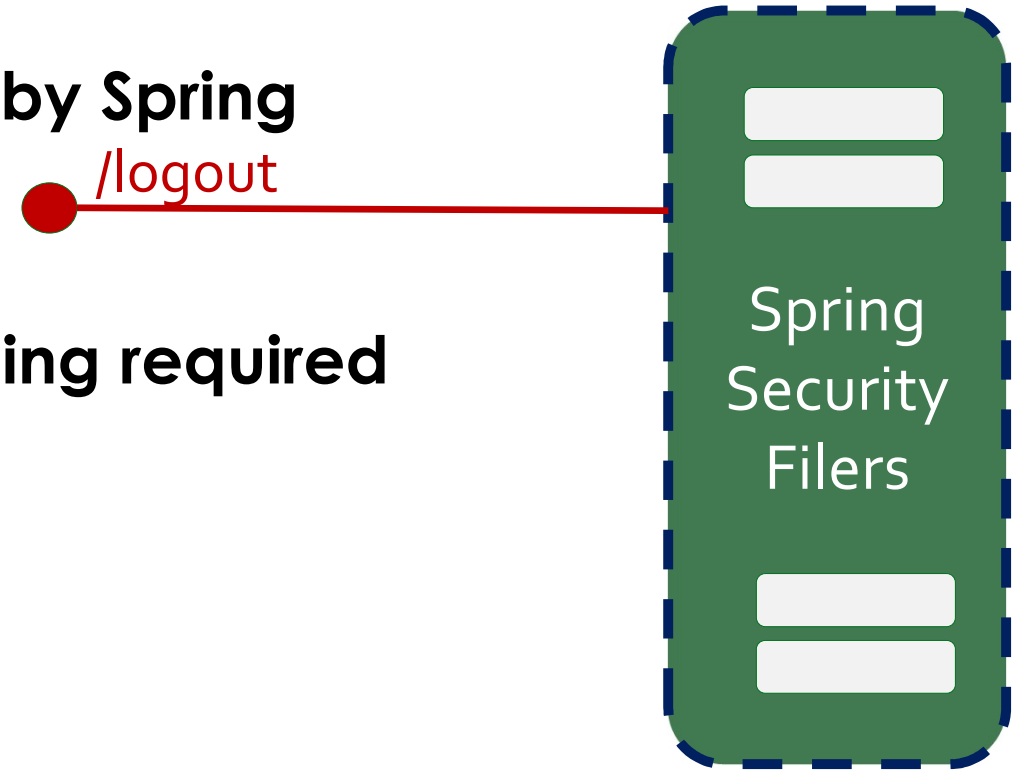
Step 1: Add logout support to Spring Security Configuration

File: DemoSecurityConfig.java

```
@Override
protected void configure(HttpSecurity http) throws Exception {
    http.authorizeRequests()
            .antMatchers("/resources/css/**").permitAll()
            .antMatchers("/resources/bootstrap/**").permitAll()
            .antMatchers("/resources/jquery/**").permitAll()
            .antMatchers("/resources/fonts/**").permitAll()
            .anyRequest().authenticated()
            .and().formLogin()
                .loginPage("/showMyLoginPage")
                .loginProcessingUrl("/authenticateTheUser")
                .permitAll()
            .and().logout()
                .permitAll(); }
```

Step 2: Add logout button to JSP page

- Send data to **default logout URL: /logout**
- Logout URL will be **handled by Spring Security Filters**
- You get it for free … **no coding required**



Step 2: Add logout button to JSP page

Send data to default logout URL: /logout

File: home.jsp

```
<form:form action="${pageContext.request.contextPath}/logout"
           method="POST">

      <input type="submit" value="Logout" />

</form:form>
```

Step 2: Add logout button to JSP page – Logout Process

» When a logout is processed, by default Spring Security will …

- **Invalidate user's HTTP session** and **remove session cookies**, etc
- Send **user back** to **login page**
- Append a **logout parameter**: **?logout**

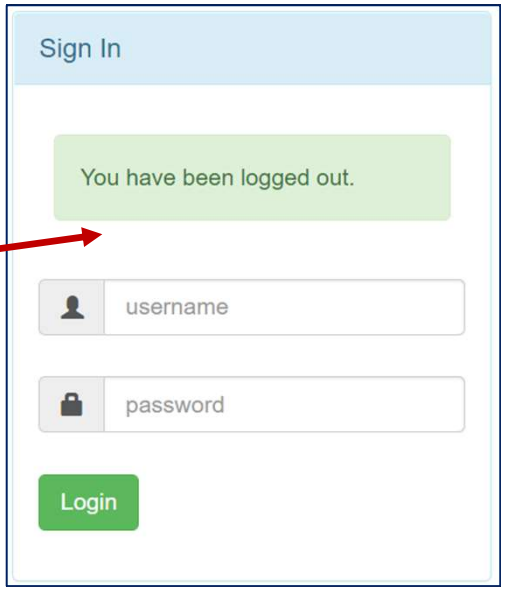
Step 3: Update login form to display "logged out" message

Update login form

- Check the logout parameter
- If logout parameter exists, show "logged out" message

File: fancy-login-local-css.jsp

```
<!-- Check for logout -->
<c:if test="${param.logout != null}">
    <div class="alert alert-success col-xs-offset-1 col-xs-10">
        You have been logged out.
    </div>
</c:if>
```



localhost:8081/spring-security-demo-custom-login-form-security/showMyLoginPage?logout

Spring Security – Cross Site Request Forgery (CSRF)

Spring Security

- Spring Security protects against **Cross-Site Request Forgery (CSRF)**
- **What is CSRF?**

A security attack where an evil website tricks you into executing an action on a web application that you are currently logged in.

CSRF Examples

You are logged into your banking app

» **tricked into sending money to another person**

You are logged into an e-commerce app

» **tricked into purchasing unwanted items**

CSRF Protection

To protect against CSRF attacks

- Embed additional **authentication data/token** into all **HTML forms**
- On **subsequent requests**, **web app** will **verify token** before processing

Spring Security's CSRF Protection

» CSRF protection is **enabled by default** in **Spring Security**

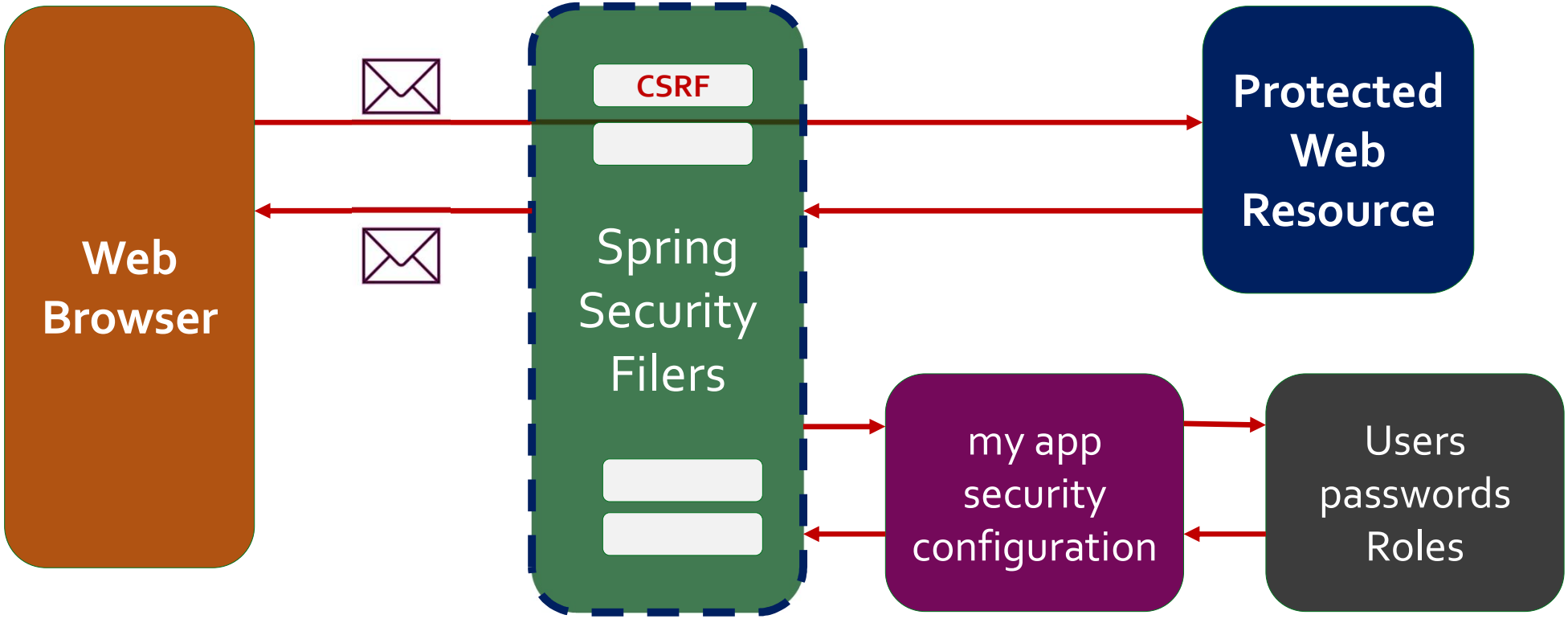
» Spring Security uses the **Synchronizer Token Pattern**

- Each request **includes a session cookie** and **randomly generated token**

- For request processing, Spring Security **verifies token before processing**

- All of this is **handled** by **Spring Security Filters**

Spring Security's CSRF Protection



When to use CSRF Protection?

» The Spring Security team recommends

- **Use CSRF protection for any normal browser web requests**

» If you are building a service for non-browser clients

- you may want to disable CSRF protection (**after careful review**)

Use Spring Security CSRF Protection

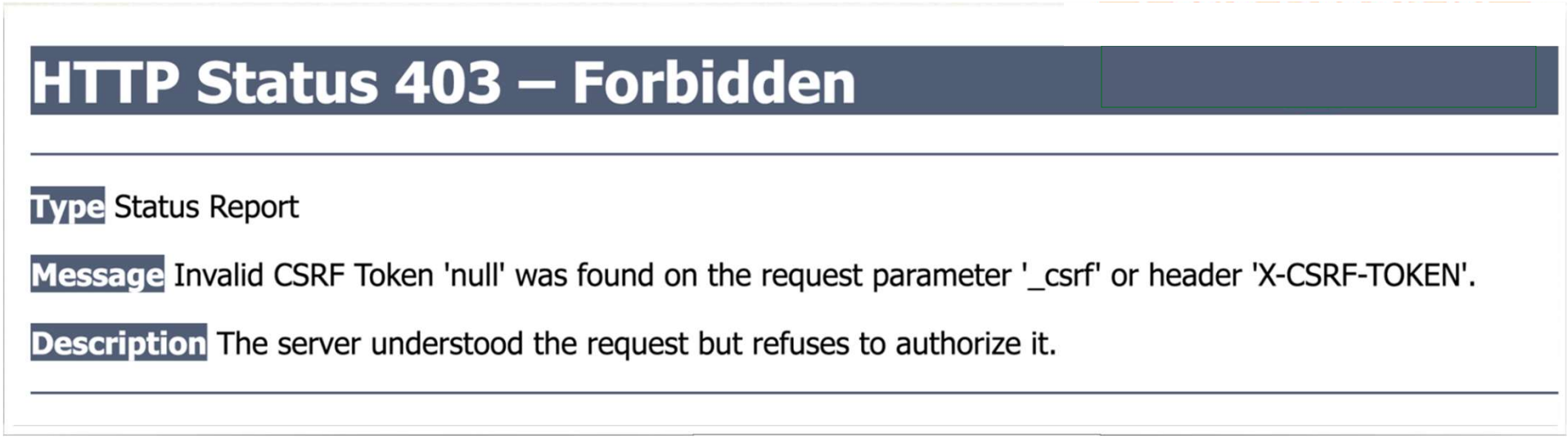
» For form submissions use **POST** instead of GET

- Include CSRF token in form submission

» • <form:form> **automagically adds CSRF token**

» • If you don't use <form:form>, you **must manually add CSRF token**

What happens if you don't include CSRF token?



View CSRF token

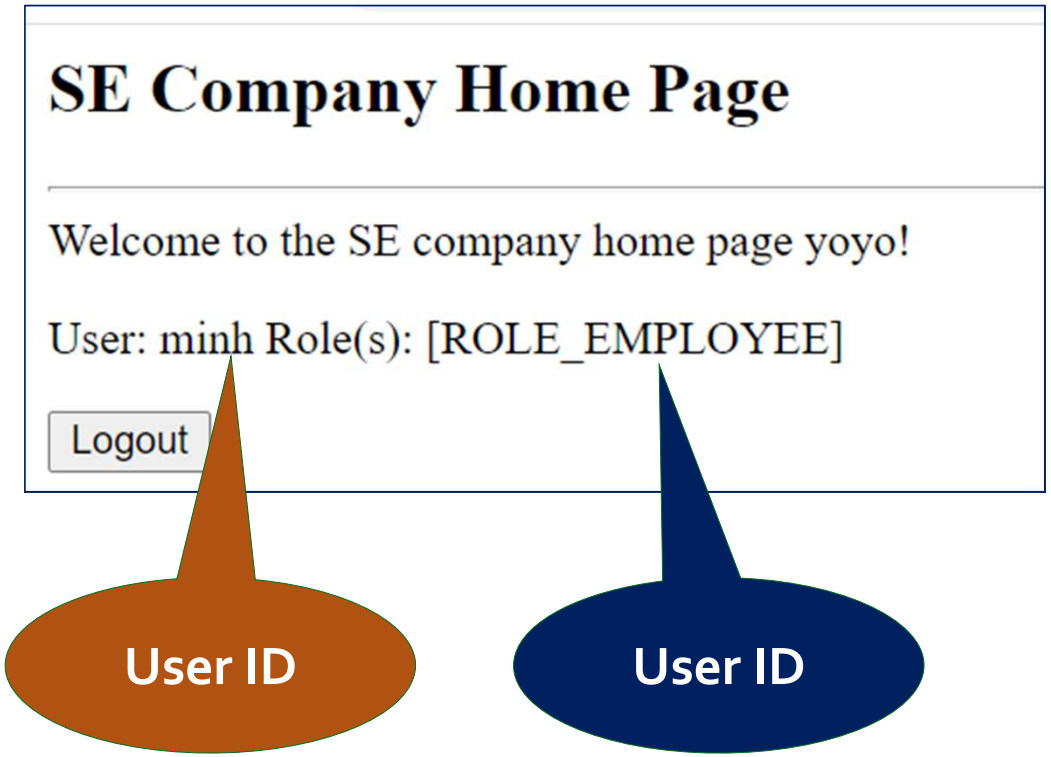
Manual add CSRF token (not using of Spring form:form)


```
<form action="${pageContext.request.contextPath}/authenticateTheUser"
      method="POST" class="form-horizontal">

    <!-- Place for messages: error, alert etc ... -->
    <...18 lines />
    <!-- User name -->
    <...5 lines />
    <!-- Password -->
    <...5 lines />
    <!-- Login/Submit Button -->
    <...5 lines />
    <!-- Imanually adding tokens ... -->
    <input type="hidden"
           name="${_csrf.parameterName}"
           value="${_csrf.token}" />
</form>
```

Spring Security – Display User ID and Roles
(project: spring-security-demo-display-user-role)

Display User ID and Roles



Development Process

- Update POM file for **Spring Security JSP Tag Library**
- Add **Spring Security JSP Tag** Library to JSP page
- **Display User ID** and **Display User Roles**

Step 1: Update POM file for Spring Security JSP Tag Library

File: pom.xml

```
<!-- Add Spring Security Taglibs support -->
<dependency>
    <groupId>org.springframework.security</groupId>
    <artifactId>spring-security-taglibs</artifactId>
    <version>${springsecurity.version}</version>
</dependency>
```

Step 2: Add Spring Security JSP Tag Library to JSP page

File: home.jsp

```
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form" %>
<%@ taglib prefix="security" uri="http://www.springframework.org/security/tags" %>
<html>
    <head>
    <body>
        <h2> SE Company Home Page</h2>
        <hr>
        Welcome to the SE company home page yoyo!
        <!-- display user name and role -->
    <p>
            User: <security:authentication property="principal.username" />
            Role(s): <security:authentication property="principal.authorities" />
    </p>
        <!-- Add a logout button -->
    <form:form action="${pageContext.request.contextPath}/logout"
    </body>
</html>
```

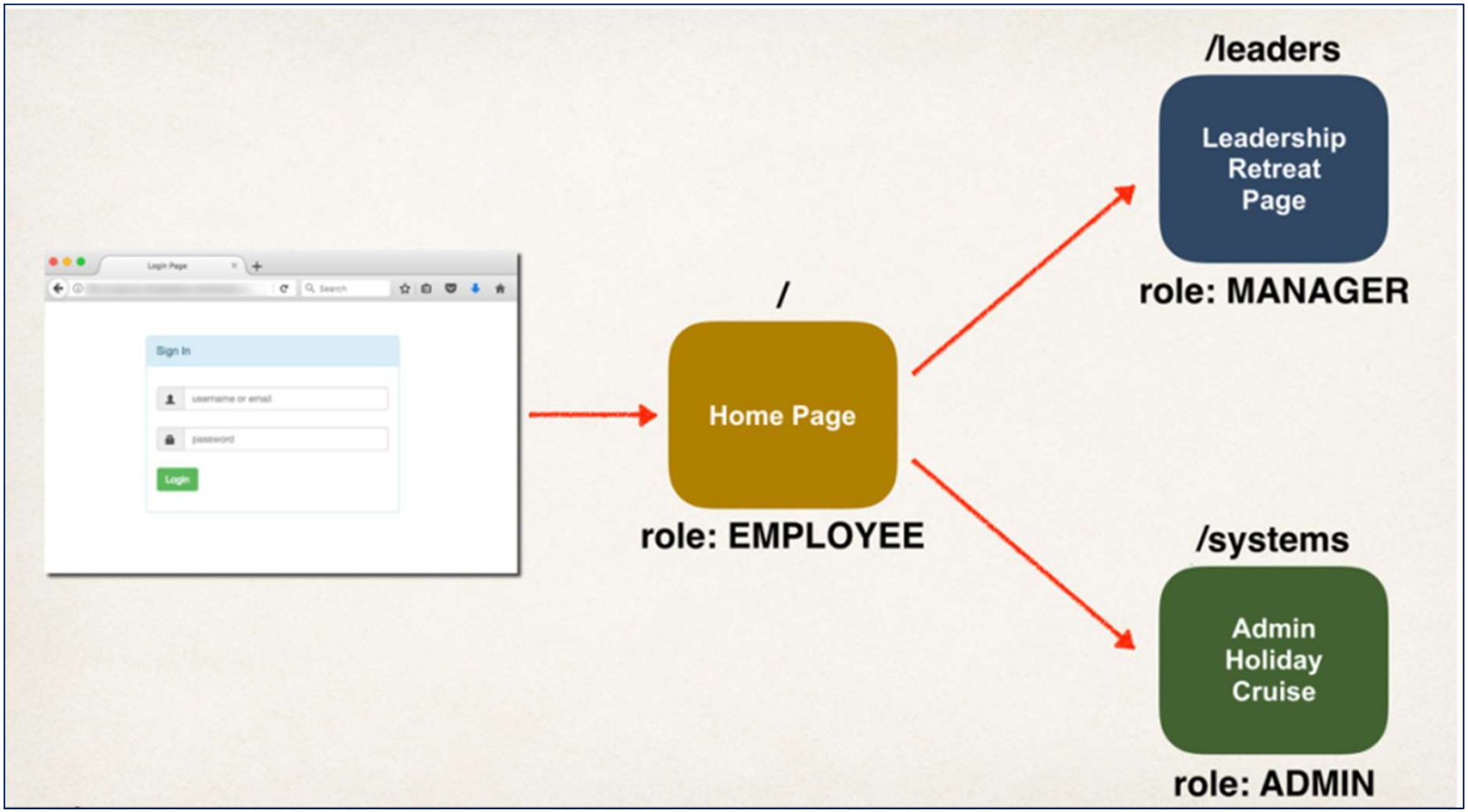
Step 2: Display User ID and Display User Roles

File: home.jsp

```
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form" %>
<%@ taglib prefix="security" uri="http://www.springframework.org/security/tags" %>
<html>
    <head>
    <body>
        <h2> SE Company Home Page</h2>
        <hr>
        Welcome to the SE company home page yoyo!
        <!-- display user name and role -->
    <p>
            User: <security:authentication property="principal.username" />
            Role(s): <security:authentication property="principal.authorities" />
    </p>
        <!-- Add a logout button -->
    <form:form action="${pageContext.request.contextPath}/logout"
    </body>
</html>
```

Spring Security – Restrict Access based on Role
(Project: spring-security-demo-restrict-access)

Our Example


/leaders
Leadership Retreat Page
role: MANAGER
/
Home Page
role: EMPLOYEE
/systems
Admin Holiday Cruise
role: ADMIN
Sign In
Login

Development Process

- Create **supporting controller** code and **view pages**
- Update **user roles**
- Restrict **Access based on Roles**

Step 1: Create supporting controller code and view pages

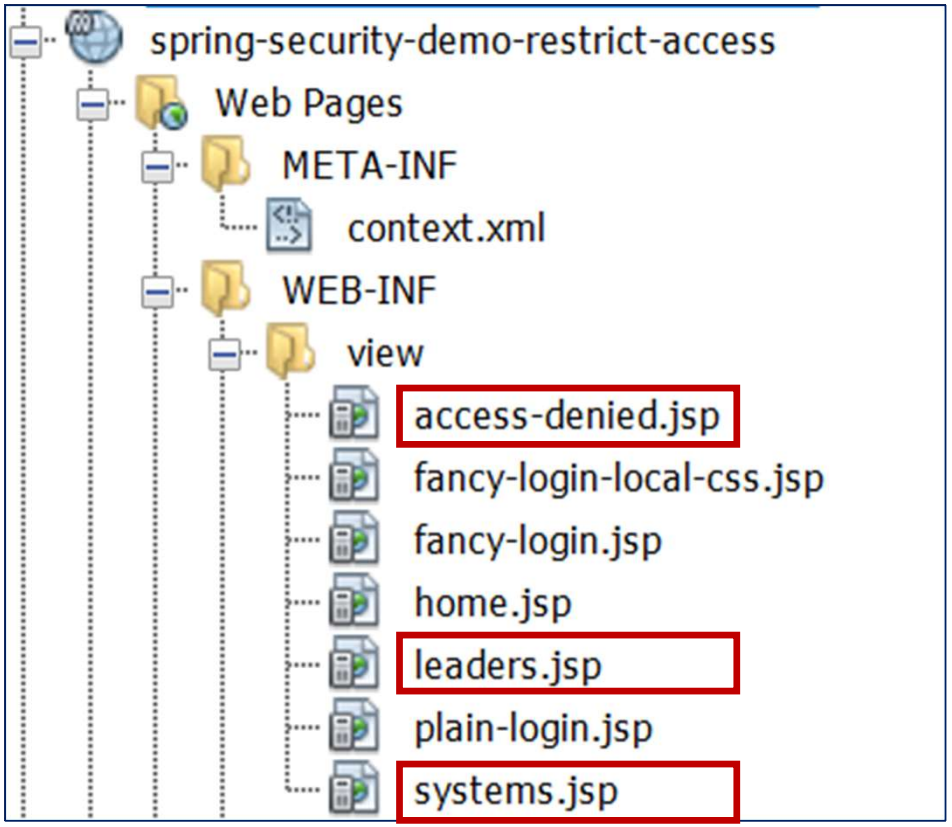
» Create **supporting controller**

File: DemoController.java

```
@Controller
public class DemoController {
   @GetMapping("/")
   public String showHome() {
         return "home";
   }
   // add request mapping for /leaders
   @GetMapping("/leaders")
   public String showLeaders() {
         return "leaders";
   }
   // add request mapping for /systems
   @GetMapping("/systems")
   public String showSystems() {
         return "systems";
   }
   @GetMapping("/access-denied")
   public String showAccessDenied() {
      return "access-denied";
   }
}
```

Step 1: Create supporting controller code and view pages

» Create **view pages**



Step 3: Restrict Access based on Roles

» Update your Spring Security Java configuration file (.java)

- General Syntax

Single role

```
antMatchers(<< add path to match on >>).hasRole(<< authorized role >>)
```

```
antMatchers(<< add path to match on >>).hasAnyRole(<< list of authorized roles >>)
```

Any role in the list, comma-delimited list

Step 3: Restrict Access based on Roles

File: DemoSecurityConfig.java

```
@Override
protected void configure(HttpSecurity http) throws Exception {
   http.authorizeRequests()
         .antMatchers("/resources/css/**").permitAll()
         .antMatchers("/resources/bootstrap/**").permitAll()
         .antMatchers("/resources/jquery/**").permitAll()
         .antMatchers("/resources/fonts/**").permitAll()
         .antMatchers("/").hasRole("EMPLOYEE")
         .antMatchers("/leaders/**").hasRole("MANAGER")
         .antMatchers("/systems/**").hasRole("ADMIN")
         .anyRequest().authenticated()
         .and().formLogin()
            .loginPage("/showMyLoginPage")
            .loginProcessingUrl("/authenticateTheUser")
            .permitAll()
          .and().logout()
            .permitAll()
         .and()
            .exceptionHandling().accessDeniedPage("/access-denied");}
```

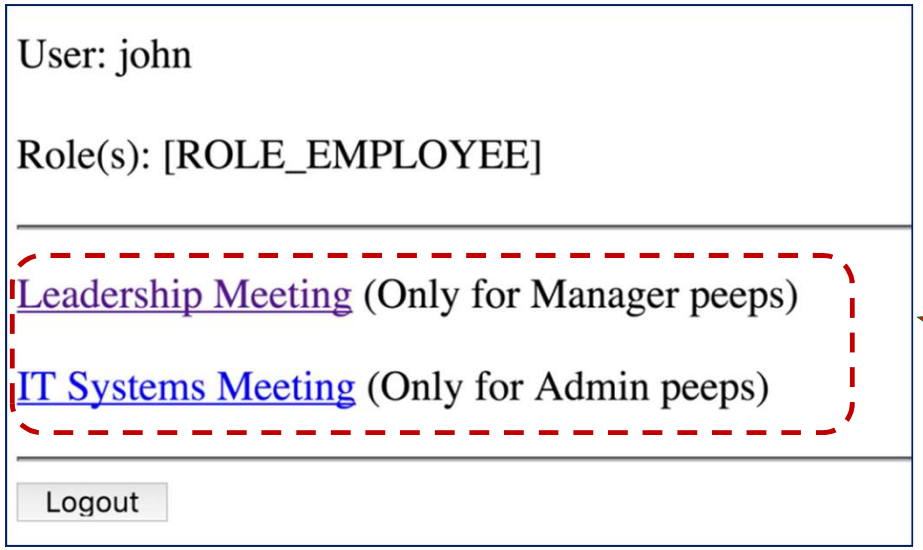
Step 3: Restrict Access based on Roles

File: DemoSecurityConfig.java

```
@Override
protected void configure(AuthenticationManagerBuilder auth) throws Exception {
        // add our users for in memory authentication
    UserBuilder users = User.withDefaultPasswordEncoder();
    auth.inMemoryAuthentication()
        .withUser(users.username("john").password("test123").roles("EMPLOYEE"))
        .withUser(users.username("mary").password("test123").roles("EMPLOYEE", "MANAGER"))
        .withUser(users.username("susan").password("test123").roles("EMPLOYEE", "ADMIN"));    }
```

Display Content Based on Roles

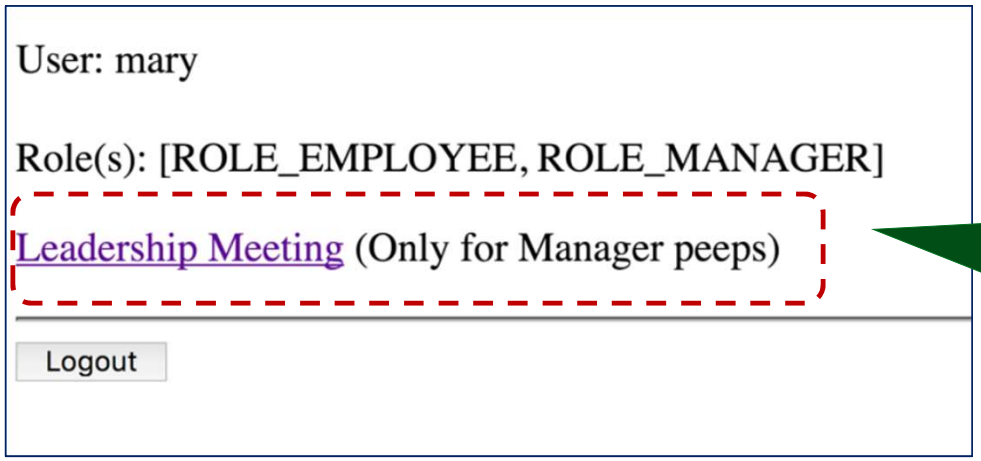
Why Show These Links?



Since John is an employee

he shouldn't be able to see this content / links

Display Content Based on Roles

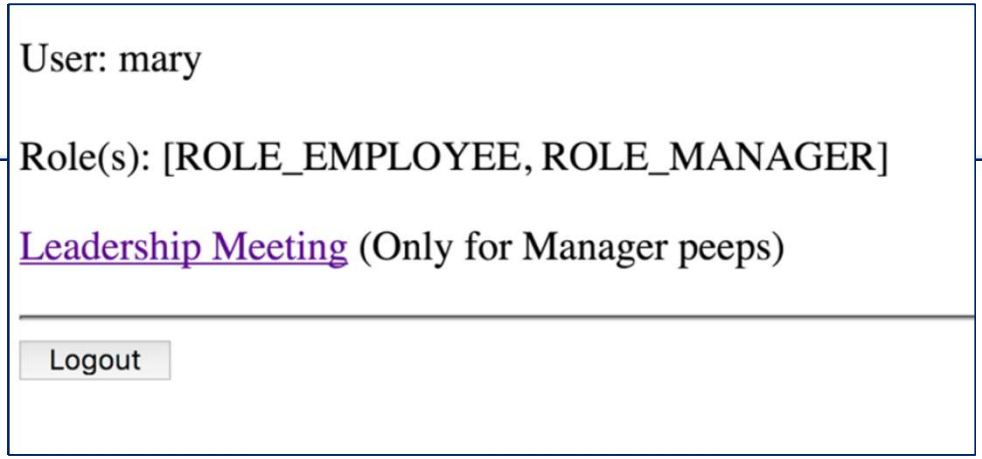


Mary can see MANAGER content

Spring Security JSP Tags

File: home.jsp

```
<security:authorize access="hasRole('MANAGER')">
    <!-- Add a link to point to /leaders ... this is for the managers -->
    <p>
        <a href="${pageContext.request.contextPath}/leaders">Leadership Meeting</a>
        (Only for Manager peeps)
    </p>
</security:authorize>
```



Spring Security JSP Tags

File: home.jsp

```
<security:authorize access="hasRole('ADMIN')">
    <!-- Add a link to point to /systems ... this is for the admins -->
    <p>
        <a href="${pageContext.request.contextPath}/systems">IT Systems Meeting</a>
        (Only for Admin peeps)
    </p>
</security:authorize>
```

Spring Security – User Accounts Stored in Database (Project: spring-security-demo-jdbc-authentication)

Database Access

- So far, our **user accounts** were **hard coded** in **Java source code**
- We want to add **database access Advance**

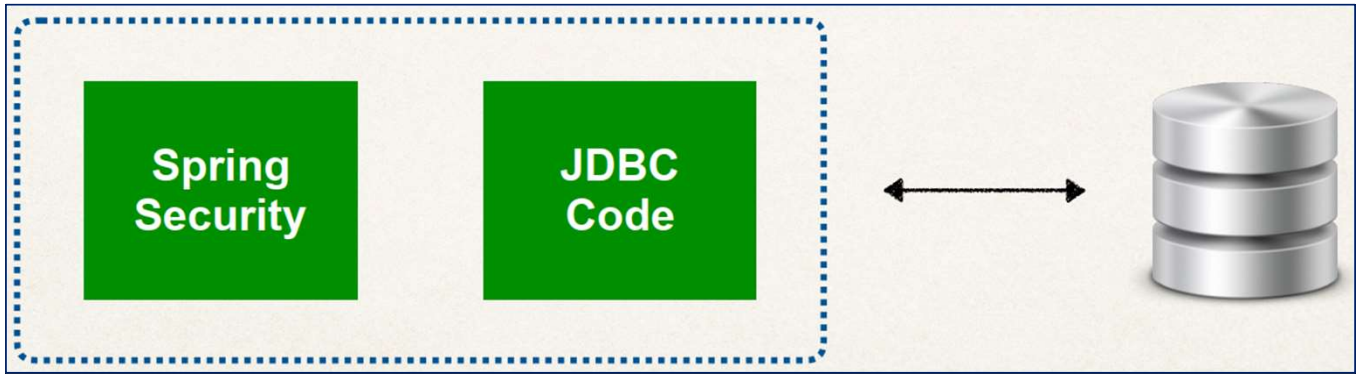
Recall Our User Roles

User ID	Password	Roles
john	test123	EMPLOYEE
mary	test123	EMPLOYEE, MANAGER
susan	test123	EMPLOYEE, ADMIN

Database Support in Spring Security

» Spring Security can **read user account info from database**

» By default, you **have to follow Spring Security's predefined table schemas**

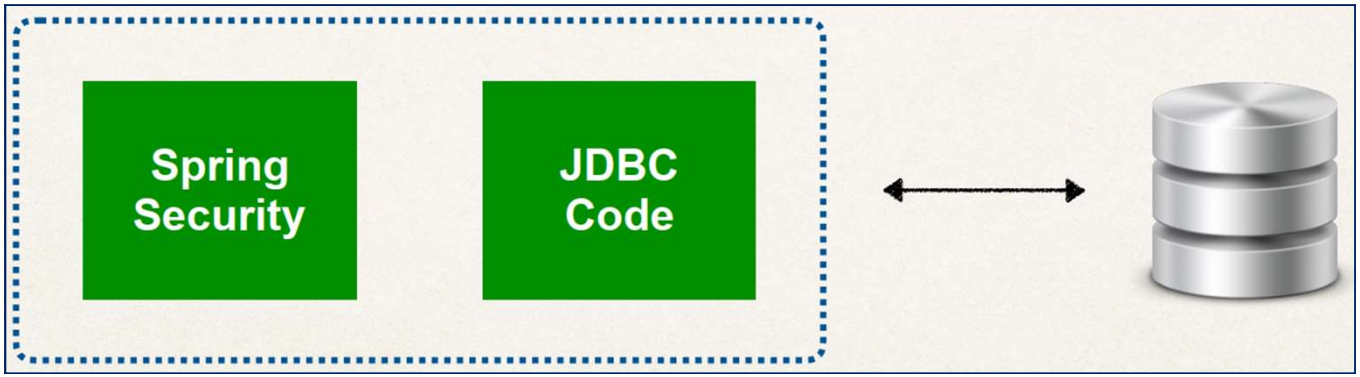


Customize Database Access with Spring Security

- Can also **customize** the **table schemas**
- Useful if you have **custom tables specific to your project**
- You will be responsible for developing the code to access the data
- JDBC, Hibernate etc …

Database Support in Spring Security

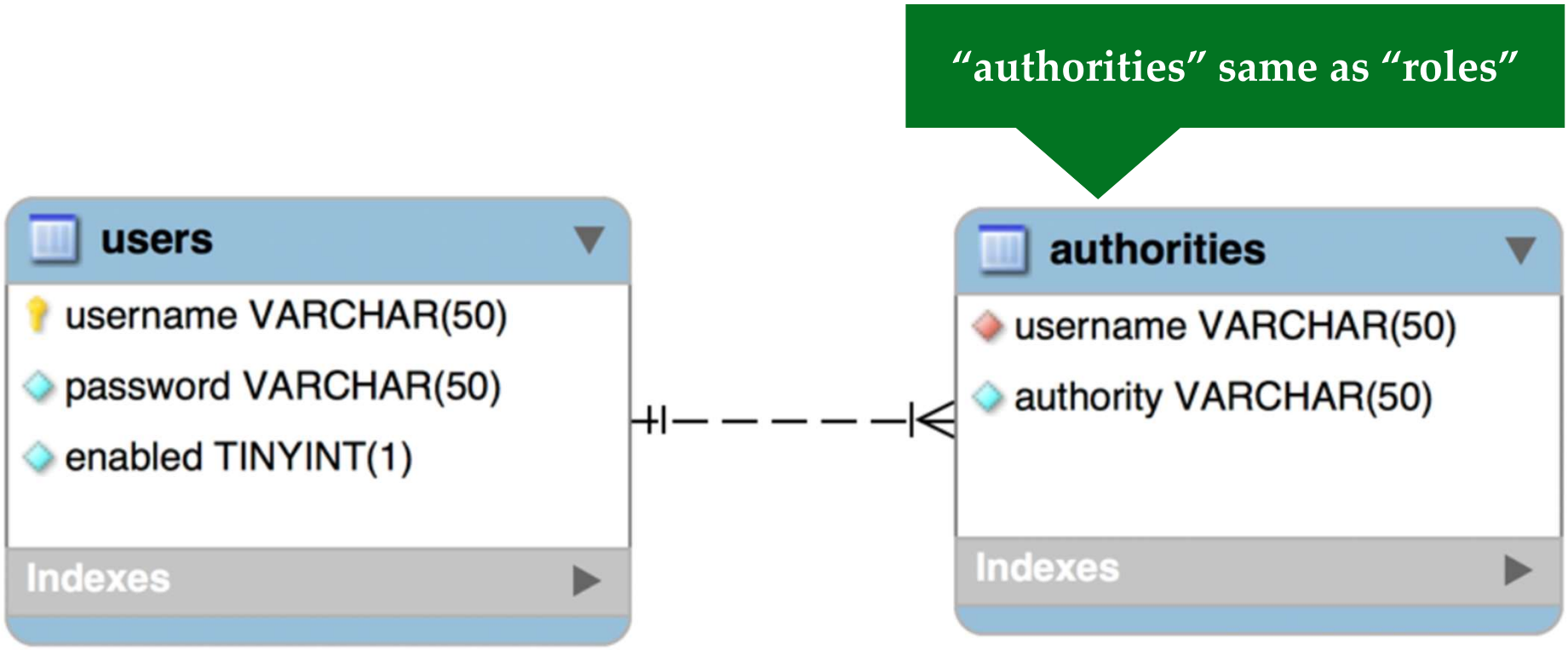
» Follow Spring Security's predefined table schemas



Development Process

- **Setup database tables**
- Add **database support** to **Maven POM** file
- Create **JDBC properties** file
- Define **DataSource in Spring Configuration**
- Update **Spring Security Configuration** to **use JDBC**

Default Spring Security Database Schema



Step 1: Setup Database

» Create Database: `spring_security_demo_plaintext`

» Create table: `users`

```
CREATE TABLE [dbo].[users](
[username] [varchar](50) NOT NULL,
[password] [varchar](50) NULL,
[enabled] [tinyint] NULL,
 CONSTRAINT [PK_users] PRIMARY KEY CLUSTERED
(
[username] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]
```

Step 1: Setup Database

» Create table: authorities

```
CREATE TABLE [dbo].[authorities](
[usename] [varchar](50) NULL,
[authority] [varchar](50) NULL
) ON [PRIMARY]

GO
ALTER TABLE [dbo].[authorities]  WITH CHECK ADD  CONSTRAINT [FK_authorities_users] FOREIGN
KEY([username])
REFERENCES [dbo].[users] ([username])
GO
ALTER TABLE [dbo].[authorities] CHECK CONSTRAINT [FK_authorities_users]
```

Step 1: Setup Database

Insert data into users table:

The encoding algorithm id

```
INSERT INTO users
VALUES
('john','{noop}test123',1),
('mary','{noop}test123',1),
('susan','{noop}test123',1);
```

The password

ID	Description
noop	Plain text passwords
bcrypt	BCrypt password hashing
...	...

one-way hashing
or one-way encryption

Step 1: Setup Database

Insert data into authorities table:

```
INSERT INTO authorities
VALUES
('john','ROLE_EMPLOYEE'),
('mary','ROLE_EMPLOYEE'),
('mary','ROLE_MANAGER'),
('susan','ROLE_EMPLOYEE'),
('susan','ROLE_ADMIN');
```

ROLE_ prefix

User ID	Password	Roles
john	test123	EMPLOYEE
mary	test123	EMPLOYEE, MANAGER
susan	test123	EMPLOYEE, ADMIN

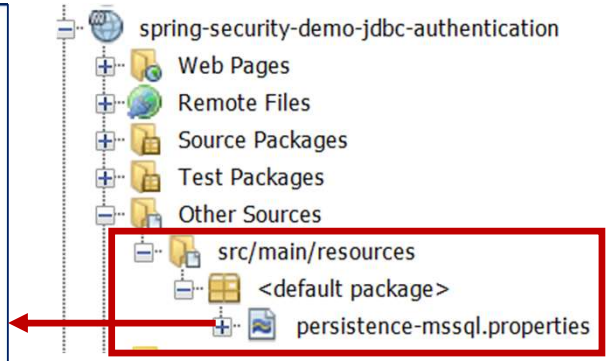
Step 2: Add database support to Maven POM file

File: pom.xml

```
<!-- Add MySQL and C3P0 support -->
        <!-- https://mvnrepository.com/artifact/com.microsoft.sqlserver/mssql-jdbc -->
        <dependency>
           <groupId>com.microsoft.sqlserver</groupId>
           <artifactId>mssql-jdbc</artifactId>
           <version>8.4.1.jre11</version>
        </dependency>
        <!-- https://mvnrepository.com/artifact/com.mchange/c3p0 -->
        <dependency>
              <groupId>com.mchange</groupId>
              <artifactId>c3p0</artifactId>
              <version>0.9.5.2</version>
        </dependency>
        <!-- Servlet, JSP and JSTL support -->
        <dependency>
```

Step 3: Create JDBC properties file

```
#
# JDBC connection properties
#
jdbc.driver=com.microsoft.sqlserver.jdbc.SQLServerDriver
jdbc.url=jdbc:sqlserver://localhost:1433;databaseName=spring_security_demo_plaintext
jdbc.user=springstudent
jdbc.password=springstudent
#
# Connection pool properties
#
connection.pool.initialPoolSize=5
connection.pool.minPoolSize=5
connection.pool.maxPoolSize=20
connection.pool.maxIdleTime=3000
```



Step 4: Define DataSource in Spring Configuration

File: DemoAppConfig.java

```
@Bean
public DataSource securityDataSource() {
    // create connection pool
    ComboPooledDataSource securityDataSource
      = new ComboPooledDataSource();
    // set the jdbc driver class
    try {
        securityDataSource.setDriverClass(env.getProperty("jdbc.driver"));
    } catch (PropertyVetoException exc) {
        throw new RuntimeException(exc);   }
  // set database connection props
    securityDataSource.setJdbcUrl(env.getProperty("jdbc.url"));
    securityDataSource.setUser(env.getProperty("jdbc.user"));
    securityDataSource.setPassword(env.getProperty("jdbc.password"));
    // set connection pool props
    securityDataSource.setInitialPoolSize(
            getIntProperty("connection.pool.initialPoolSize"));
    securityDataSource.setMinPoolSize(
            getIntProperty("connection.pool.minPoolSize"));
    securityDataSource.setMaxPoolSize(
            getIntProperty("connection.pool.maxPoolSize"));
    securityDataSource.setMaxIdleTime(
            getIntProperty("connection.pool.maxIdleTime"));
```

```
// log the connection props
//just to make sure we are REALLY reading data from properties file
logger.info(">>> jdbc.url=" + env.getProperty("jdbc.url"));
logger.info(">>> jdbc.user=" + env.getProperty("jdbc.user"));
return securityDataSource;
}
```

```
jdbc.driver=com.microsoft.sqlserver.jdbc.SQLServerDriver
jdbc.url=jdbc:sqlserver://localhost:1433;databaseName=spring_security_demo_plaintext
jdbc.user=springstudent
jdbc.password=springstudent
connection.pool.initialPoolSize=5
connection.pool.minPoolSize=5
connection.pool.maxPoolSize=20
connection.pool.maxIdleTime=3000
```

Step 5: Update Spring Security Configuration to use JDBC

File: DemoSecurityConfig.java

```
@Configuration
@EnableWebSecurity
public class DemoSecurityConfig extends WebSecurityConfigurerAdapter {
// add a reference to our security data source
@Autowired
private DataSource securityDataSource;
@Override
 protected void configure(AuthenticationManagerBuilder auth) throws Exception {
    // use jdbc authentication
    auth.jdbcAuthentication().dataSource(securityDataSource); }
```

Tell Spring security to JDBC authentication with our data sources

No longer hard-coding users

Spring Security – Password Encryption

(Project: spring-security-demo-jdbc-encrypt-password)

Password Storage - Best Practice

- **The** best practice is store passwords in an **encrypted format**
- **Spring Security recommends** using the popular **bcrypt** algorithm

username	password
john	{bcrypt}$2a$10$12tNGY/2mOywEOYufGcLnuICjFI/Z3F/FWI4UAq2zcqUwZbFm/mdW
mary	{bcrypt}$2a$10$9bWMlwdxwB.y6cUMSNjXTe4sFqkhWqXZSQpsKs6Pz6BbXklMggEl6
susan	{bcrypt}$2a$10$12tNGY/2mOywEOYufGcLnuICjFI/Z3F/FWI4UAq2zcqUwZbFm/mdW

Encrypted version of password

bcrypt

- Performs **one-way encrypted hashing**
- Adds a **random salt** to the password for **additional protection**
- **Includes support** to **defeat brute force attacks**

How to Get a Bcrypt password

- You have a plaintext password and you want to encrypt using bcrypt
- **Option 1**: Use a **website utility** to perform the encryption
- **Option 2**: **Write Java code** to perform the encryption

How to Get a Bcrypt password - Website

- Visit: https://www.bcryptcalculator.com/encode
- Enter your plaintext password
- The website will generate a bcrypt password for you

Development Process

- **Setup database tables**
- Modify DDL for password field, **length** should be **68**
- Modify database properties file to point to **new database schema**

Step 1: Modify DDL for password field, length should be 68


```
CREATE TABLE users (
'username' varchar(50) NOT NULL,
'password' char(68) NOT NULL,
'enabled' tinyint(1) NOT NULL,

PRIMARY KEY ('username'))
```

The encrypted password

```
INSERT INTO users VALUES
('john','{bcrypt}$2a$04$eFytJDGtjbThXa80FyOOBuFdK2IwjyWefYkMpiBEFlpBwDH.5PM0K',1),

('mary','{bcrypt}$2a$04$eFytJDGtjbThXa80FyOOBuFdK2IwjyWefYkMpiBEFlpBwDH.5PM0K',1),

('susan','{bcrypt}$2a$04$eFytJDGtjbThXa80FyOOBuFdK2IwjyWefYkMpiBEFlpBwDH.5PM0K',1)
;
```

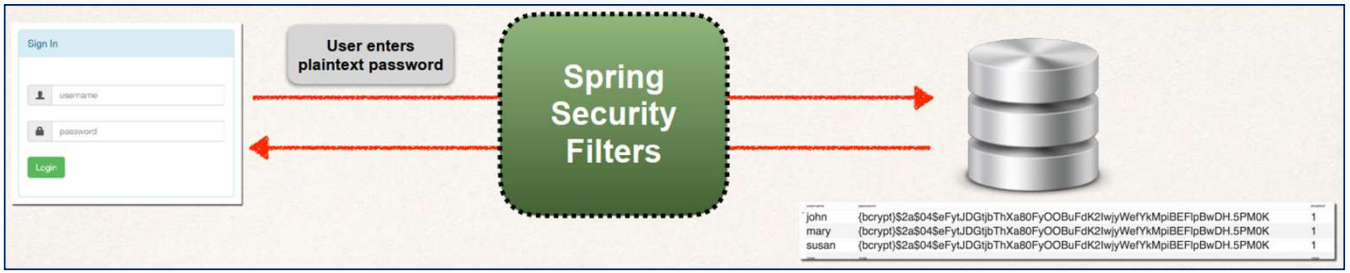
The encoding algorithm id

Step 2: Point to New Database Schema

File: src/main/resources/persistence-mysql.properties

```
#
# JDBC connection properties
#
jdbc.driver=com.mysql.jdbc.Driver
jdbc.url=jdbc:sqlserver://localhost:1433;databaseName=spring_security_demo_encrypted
jdbc.user=springstudent
jdbc.password=springstudent
#
# Connection pool properties
#
connection.pool.initialPoolSize=5
connection.pool.minPoolSize=5
connection.pool.maxPoolSize=20
connection.pool.maxIdleTime=3000
```

Spring Security Login Process



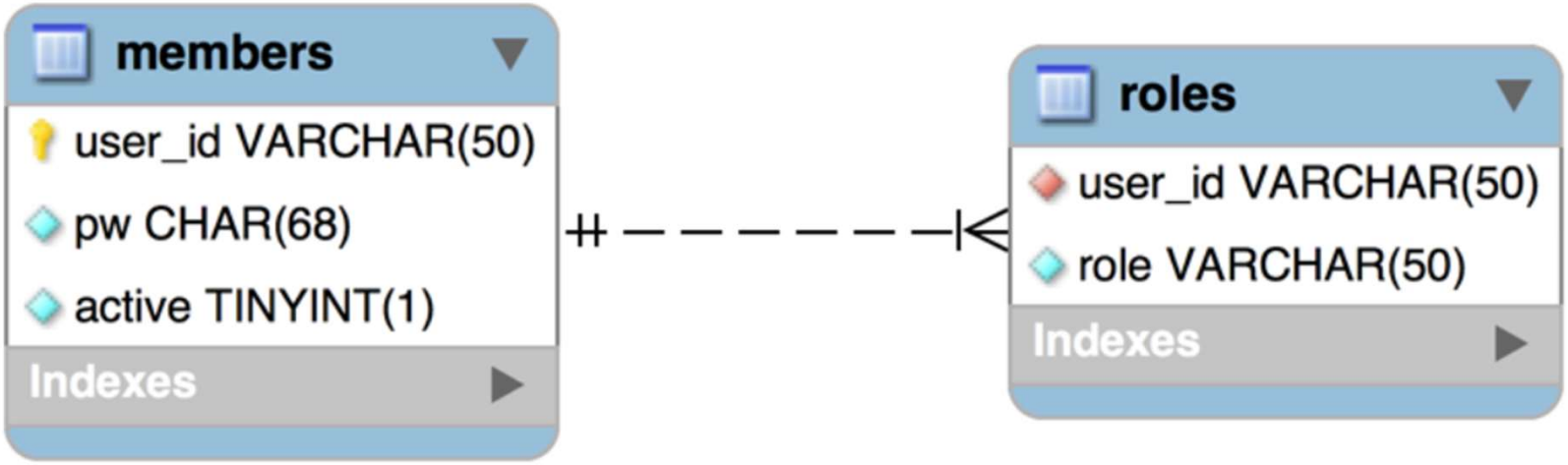
1. Retrieve password from db for the user
2. Read the encoding algorithm id (bcrypt etc)
3. For case of bcrypt, encrypt plaintext password from login form (using salt from db password)
4. Compare encrypted password from login form WITH encrypted password from db
5. If there is a match, login successful
6. If no match, login NOT successful

Spring Security – Custom Tables

Custom Tables

» What if we have our own custom tables?

» Our own custom column names?



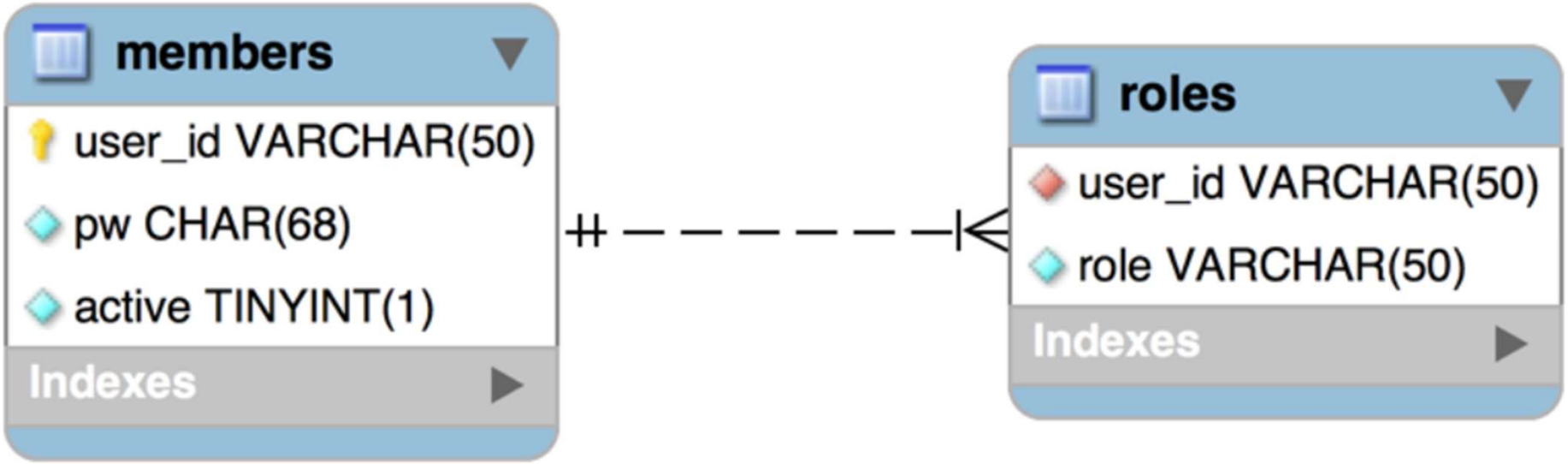
For Security Schema Customization

- Tell Spring how to query your custom tables
- Provide query to find user by user name
- Provide query to find authorities / roles by user name

Development Process

- Create our **custom tables** with SQL
- Update **JDBC properties file** to point to new database schema
- Update **Spring Security Configuration**
  - Provide query to find user by user name
  - Provide query to find authorities / roles by user name

Step 1: Create our custom tables

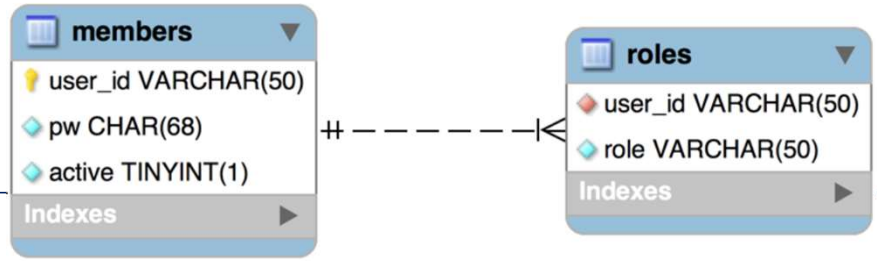


Step 2: Update JDBC properties file

File: src/main/resources/persistence-mysql.properties

```
#
# JDBC connection properties
#
jdbc.driver=com.mysql.jdbc.Driver
jdbc.url=jdbc:sqlserver://localhost:1433;databaseName=spring_security_demo_encrypted_custom
jdbc.user=springstudent
jdbc.password=springstudent
#
# Connection pool properties
#
connection.pool.initialPoolSize=5
connection.pool.minPoolSize=5
connection.pool.maxPoolSize=20
connection.pool.maxIdleTime=3000
```

Step 3: Update Spring Security Configuration



File: DemoSecurityConfig.java

```
@Configuration
@EnableWebSecurity
public class DemoSecurityConfig extends WebSecurityConfigurerAdapter {
@Autowired
private DataSource securityDataSource;
@Override
protected void configure(AuthenticationManagerBuilder auth) throws Exception {
auth.jdbcAuthentication().dataSource(securityDataSource)
.usersByUsernameQuery("select user_id, pw, active from members where user_id=?")
.authoritiesByUsernameQuery("select user_id, role from roles where user_id=?");
}
…
}
```

How to find users

How to find roles

Questions