Spring Security Overview

You will learn how to …

- Secure Spring MVC Web Apps
- Develop login pages (default and custom)
- Define users and roles with simple authentication
- Protect URLs based on role
- Use JSP tags to hide/show content based on role
- Store users, passwords and roles in DB (plain-text -> encrypted)

Practical Results

- Cover the most common Spring Security tasks that you will need on daily projects
- Not an A to Z reference … for that you can see **Spring Security Reference Manual**

http://www.luv2code.com/spring-security-reference-manual

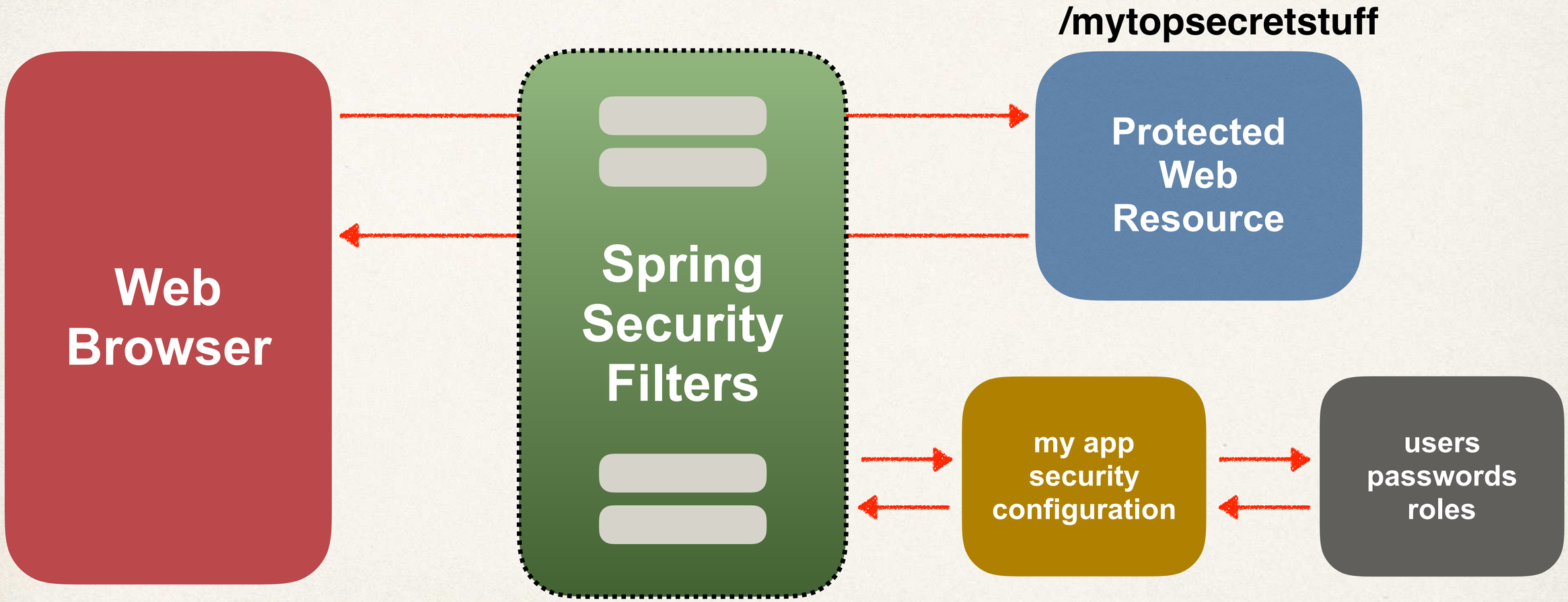
Spring Security Model

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

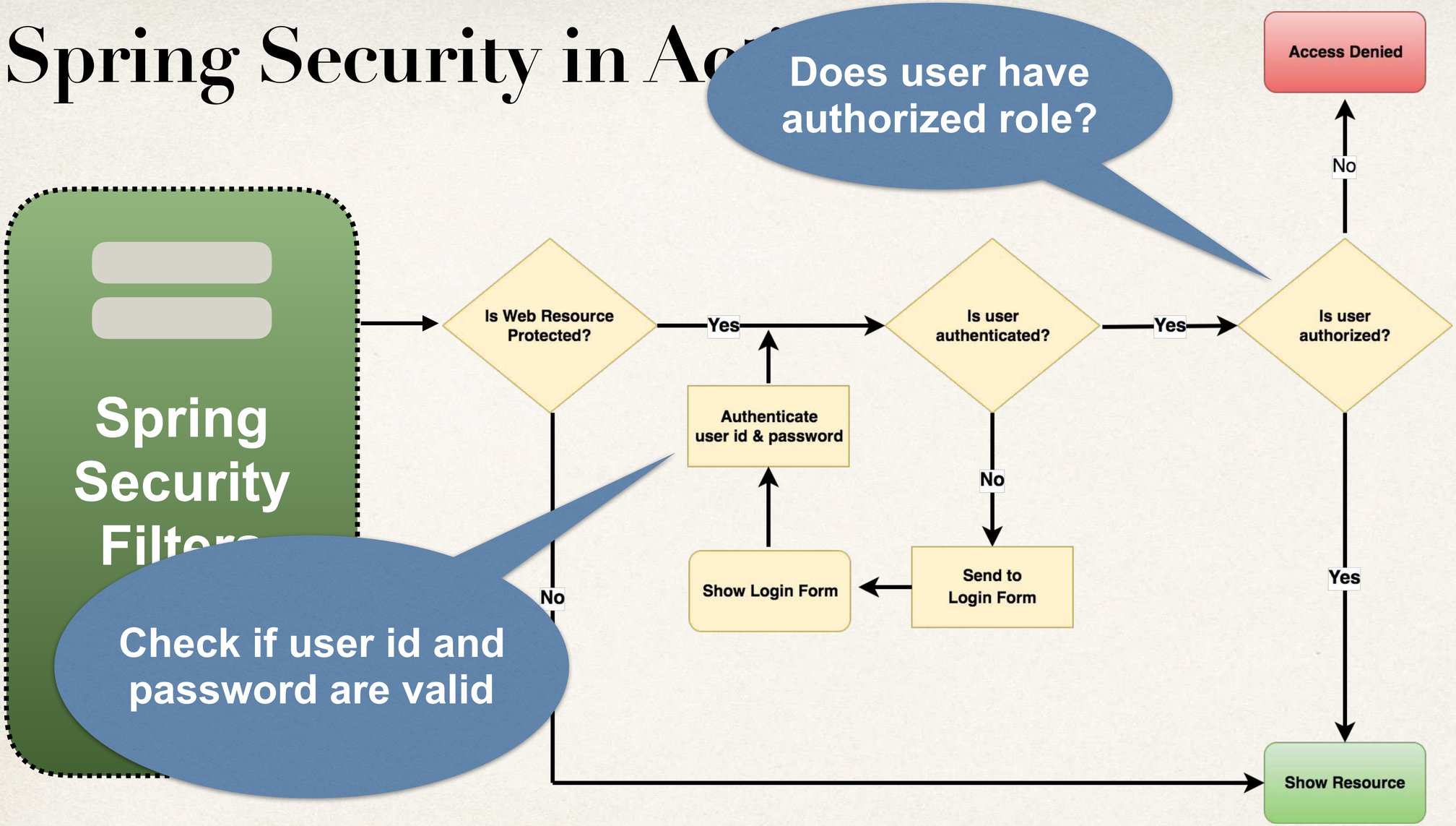
Spring Security with Servlet Filters

- 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 in Ac



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

Programmatic Security

- Spring Security provides an API for custom application coding
- Provides greater customization for specific app requirements

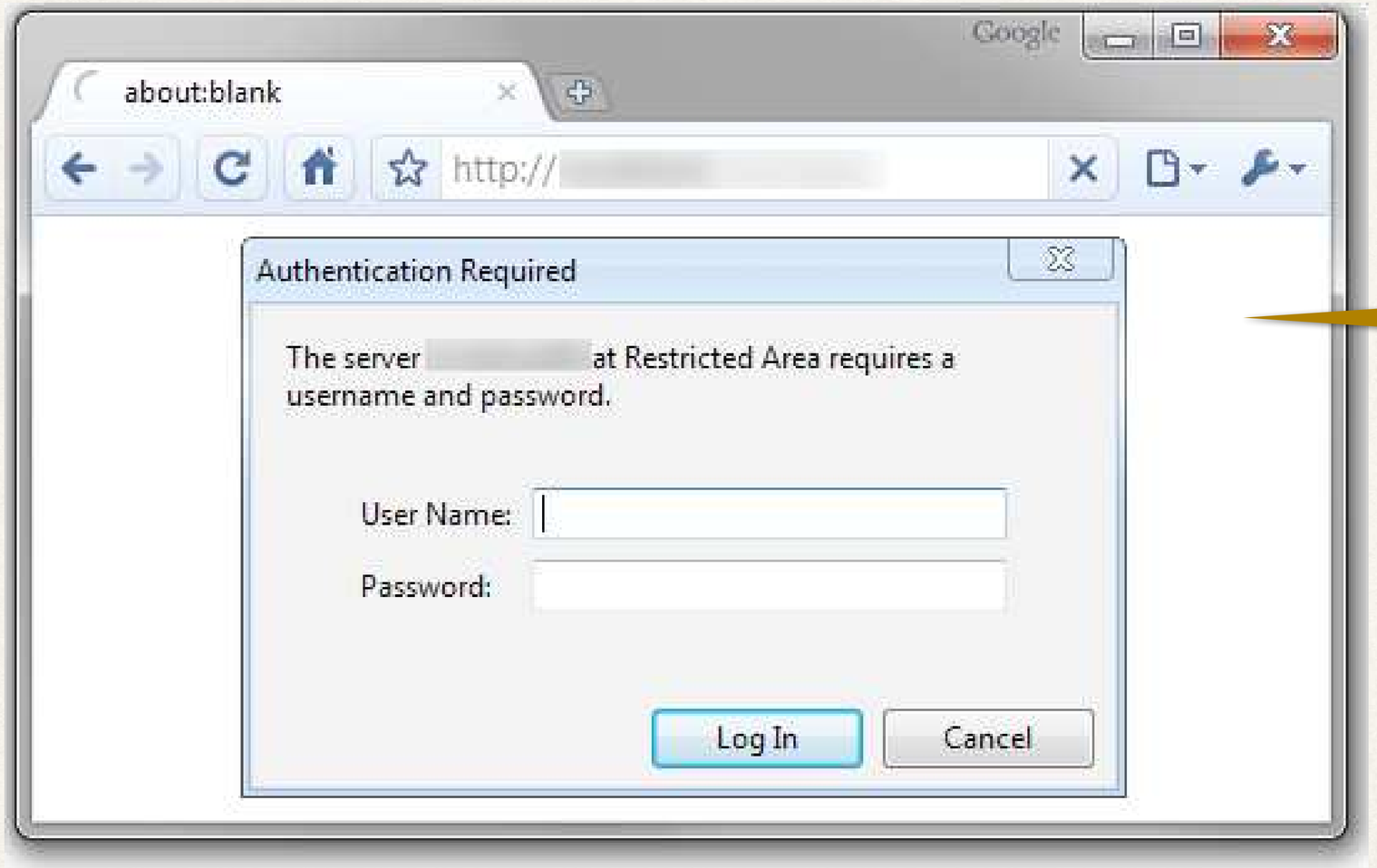
Different Login Methods

- HTTP Basic Authentication

- Default login form
  - Spring Security provides a default login form

- Custom login form
  - your own look-and-feel, HTML + CSS

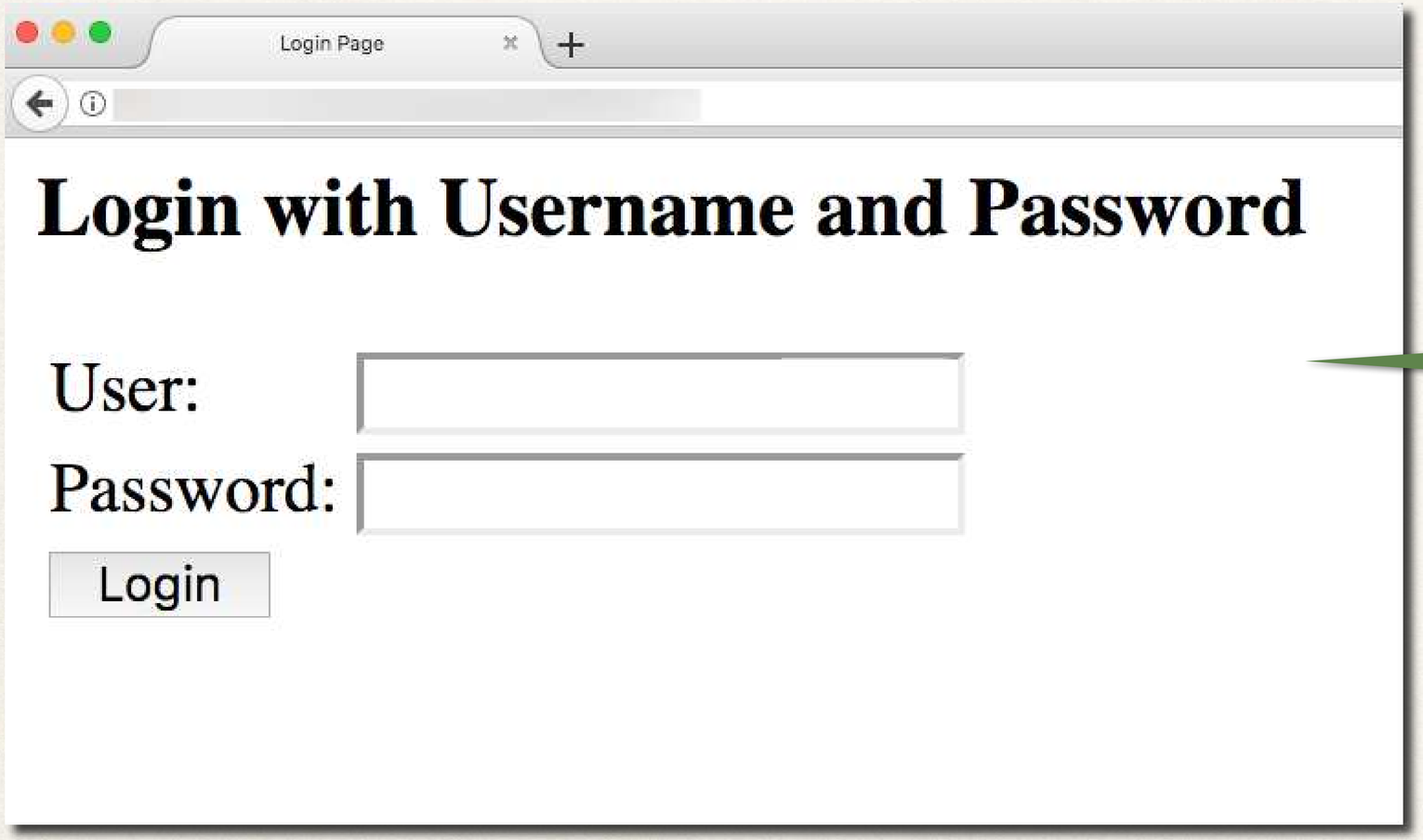
HTTP Basic Authentication



Built-in Dialog from browser

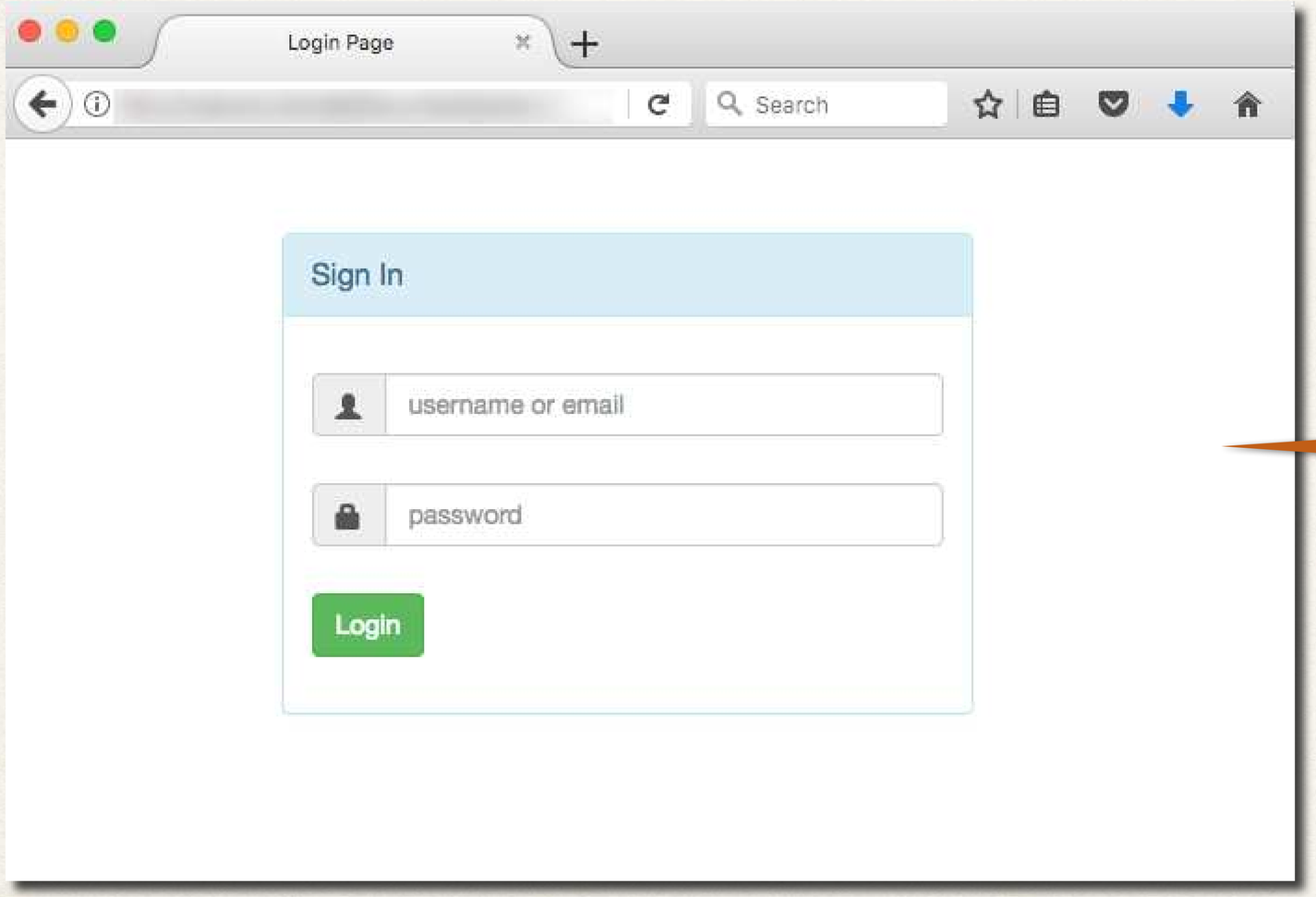
bleh!

Spring Security - Default Login Form



Good for quick start

Your Own Custom Login Form



Your own custom look-and-feel HTML + CSS

Authentication and Authorization

- In-memory
- JDBC
- LDAP
- Custom / Pluggable
- *others …*

users
passwords
roles

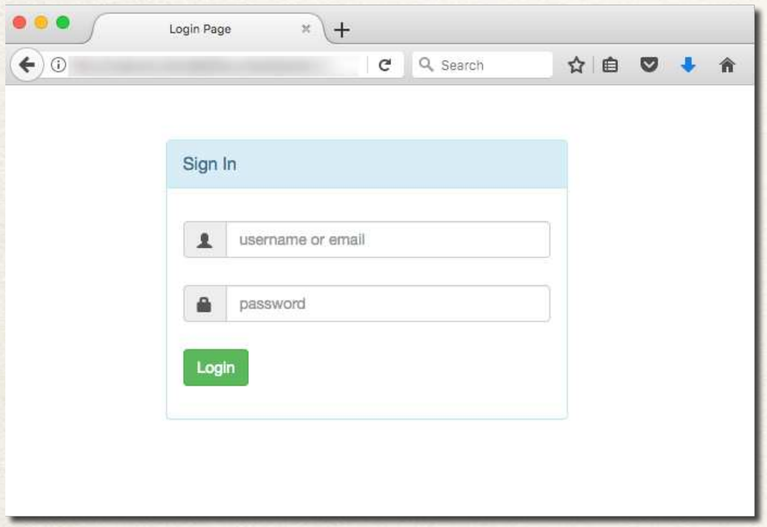
We will cover password storage in DB as plain-text AND encrypted

Spring Security Demo

Recap of our Game Plan

- Secure Spring MVC Web Apps
- Develop login pages (default and custom)
- Define users and roles with simple authentication
- Protect URLs based on role
- Use JSP tags to hide/show content based on role
- Store users, passwords and roles in DB (plain-text -> encrypted)

Our Example



/

Home Page

role: EMPLOYEE

/leaders

Leadership Retreat Page

role: MANAGER

/systems

Admin Holiday Cruise

role: ADMIN

Spring MVC App - Java Config

Java Configuration

Java Configuration

- Instead of configuring Spring MVC app using XML

Java Configuration

- Instead of configuring Spring MVC app using XML
  - web.xml

Java Configuration

- Instead of configuring Spring MVC app using XML
  - web.xml
  - spring-mvc-demo-servlet.xml

Java Configuration

- Instead of configuring Spring MVC app using XML
  - web.xml
  - spring-mvc-demo-servlet.xml

Java Configuration

- Instead of configuring Spring MVC app using XML
  - web.xml
  - spring-mvc-demo-servlet.xml

- Configure the Spring MVC app with Java code

Java Configuration

- Instead of configuring Spring MVC app using XML
  - web.xml
  - spring-mvc-demo-servlet.xml

- Configure the Spring MVC app with Java code

No XML!

Development Process

Step-By-Step

Development Process

Step-By-Step

1. Add Maven dependencies for Spring MVC Web App

Development Process

Step-By-Step

1. Add Maven dependencies for Spring MVC Web App

2. Create Spring App Configuration (@Configuration)

Development Process

Step-By-Step

1. Add Maven dependencies for Spring MVC Web App

2. Create Spring App Configuration (@Configuration)

3. Create Spring Dispatcher Servlet Initializer

Development Process

Step-By-Step

1. Add Maven dependencies for Spring MVC Web App

2. Create Spring App Configuration (@Configuration)

3. Create Spring Dispatcher Servlet Initializer

4. Develop our Spring controller

Development Process

Step-By-Step

1. Add Maven dependencies for Spring MVC Web App

2. Create Spring App Configuration (@Configuration)

3. Create Spring Dispatcher Servlet Initializer

4. Develop our Spring controller

5. Develop our JSP view page

Step 1: Add Maven dependencies for Spring MVC Web App

Step 1: Add Maven dependencies for Spring MVC Web App

spring-

Step 1: Add Maven dependencies for Spring MVC Web App

spring-

jstl

Step 1: Add Maven dependencies for Spring MVC Web App

spring-

javax.servlet-api

jstl

Step 1: Add Maven dependencies for Spring MVC Web App

spring-

javax.servlet-api

jstl

javax.servlet.jsp-api

Step 1: Add Maven dependencies for Spring MVC Web App

File: pom.xml

```
<!-- Spring MVC support -->
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-webmvc</artifactId>
    <version>...</version>
</dependency>
```

Step 1: Add Maven dependencies for Spring MVC Web App

File: pom.xml

```xml
<!-- Spring MVC support -->
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-webmvc</artifactId>
    <version>...</version>
</dependency>
```

Will load all supporting dependencies: spring-core, logging etc …

Step 1: Add Maven dependencies for Spring MVC Web App

File: pom.xml

```
<!-- Spring MVC support -->
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-webmvc</artifactId>
    <version>...</version>
</dependency>
```

Step 1: Add Maven dependencies for Spring MVC Web App

File: pom.xml

```
<!-- Spring MVC support -->
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-webmvc</artifactId>
    <version>...</version>
</dependency>

<!-- Servlet, JSP and JSTL support -->
<dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>javax.servlet-api</artifactId>
    <version>...</version>
</dependency>

<dependency>
    <groupId>javax.servlet.jsp</groupId>
    <artifactId>javax.servlet.jsp-api</artifactId>
    <version>...</version>
</dependency>

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

Add Servlet, JSP and JSTL support

Customize Maven Build

File: pom.xml

```
<build>

    <pluginManagement>
        <plugins>





        </plugins>
    </pluginManagement>
```

Customize Maven Build

File: pom.xml

```
<build>

    <pluginManagement>
        <plugins>





        </plugins>
    </pluginManagement>
```

Need to customize Maven build

Since we are not using web.xml

Customize Maven Build

File: pom.xml

```
<build>

    <pluginManagement>
        <plugins>

            <plugin>
                <!-- Add Maven coordinates (GAV) for: maven-war-plugin -->
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-war-plugin</artifactId>
                <version>3.2.0</version>
            </plugin>

        </plugins>
    </pluginManagement>
```

Need to customize Maven build

Since we are not using web.xml

Customize Maven Build

File: pom.xml

```
<build>

    <pluginManagement>
        <plugins>

            <plugin>
                <!-- Add Maven coordinates (GAV) for: maven-war-plugin -->
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-war-plugin</artifactId>
                <version>3.2.0</version>
            </plugin>

        </plugins>
    </pluginManagement>
```

Need to customize Maven build

Since we are not using web.xml

Must add Maven WAR plugin

XML config to Java config

XML config to Java config

web.xml

spring-mvc-demo-servlet.xml

XML config to Java config

web.xml

spring-mvc-demo-servlet.xml

Java Config

Spring
@Configuration

Spring Dispatcher
Servlet Initializer

XML config to Java config

Java Config

Spring
@Configuration

Spring Dispatcher
Servlet Initializer

XML config to Java config

Java Config

Spring
@Configuration

Spring Dispatcher
Servlet Initializer

No XML!

Flash Back to XML config (the old way)

File: spring-mvc-demo-servlet.xml

```
<beans>

</beans>
```

Flash Back to XML config (the old way)

File: spring-mvc-demo-servlet.xml

```
<beans>

</beans>
```

Just an FYI

Flash Back to XML config (the old way)

File: spring-mvc-demo-servlet.xml

```
<beans>

  <!-- Add support for component scanning -->
  <context:component-scan base-package="com.luv2code.springdemo" />

</beans>
```

Just an FYI

Flash Back to XML config (the old way)

File: spring-mvc-demo-servlet.xml

```
<beans>

    <!-- Add support for component scanning -->
    <context:component-scan base-package="com.luv2code.springdemo" />

    <!-- Add support for conversion, formatting and validation support -->
    <mvc:annotation-driven/>



</beans>
```

Just an FYI

Flash Back to XML config (the old way)

File: spring-mvc-demo-servlet.xml

```
<beans>

   <!-- Add support for component scanning -->
   <context:component-scan base-package="com.luv2code.springdemo" />

   <!-- Add support for conversion, formatting and validation support -->
   <mvc:annotation-driven/>

   <!-- Define Spring MVC view resolver -->
   <bean
      class="org.springframework.web.servlet.view.InternalResourceViewResolver">
      <property name="prefix" value="/WEB-INF/view/" />
      <property name="suffix" value=".jsp" />
   </bean>

</beans>
```

Just an FYI

Enabling the MVC Java Config

Enabling the MVC Java Config

```
@EnableWebMvc
```

Enabling the MVC Java Config

@EnableWebMvc

- Provides similar support to **<mvc:annotation-driven />** in XML.

Enabling the MVC Java Config

@EnableWebMvc

- Provides similar support to **<mvc:annotation-driven />** in XML.
- Adds conversion, formatting and validation support

Enabling the MVC Java Config

@EnableWebMvc

- Provides similar support to **<mvc:annotation-driven />** in XML.
- Adds conversion, formatting and validation support
- Processing of @Controller classes and @RequestMapping etc ... methods

Step 2: Create Spring App Configuration

File: DemoAppConfig.java

```
@Configuration
@EnableWebMvc
@ComponentScan(basePackages="com.luv2code.springsecurity.demo")
public class DemoAppConfig {

}
```

Step 2: Create Spring App Configuration

File: DemoAppConfig.java

```
@Configuration
@EnableWebMvc
@ComponentScan(basePacka
public class DemoAppConfig {

}
```

provides similar support to
<mvc:annotation-driven />

Step 2: Create Spring App Configuration

File: DemoAppConfig.java

```
@Configuration
@EnableWebMvc
@ComponentScan(basePackages="com.luv2code.springsecurity.demo")
public class DemoAppConfig {

}
```

Step 2: Create Spring App Configuration

File: DemoAppConfig.java

```
@Configuration
@EnableWebMvc
@ComponentScan(basePackages="com.luv2code.springsecurity.demo")
public class DemoAppConfig {

    // define a bean for ViewResolver



}
```

Step 2: Create Spring App Configuration

File: DemoAppConfig.java

```
@Configuration
@EnableWebMvc
@ComponentScan(basePackages="com.luv2code.springsecurity.demo")
public class DemoAppConfig {

    // define a bean for ViewResolver

    @Bean
    public ViewResolver viewResolver() {

        InternalResourceViewResolver viewResolver = new InternalResourceViewResolver();

        viewResolver.setPrefix("/WEB-INF/view/");
        viewResolver.setSuffix(".jsp");

        return viewResolver;
    }

}
```

View Resolver Configs - Explained

```
InternalResourceViewResolver viewResolver = new InternalResourceViewResolver();

viewResolver.setPrefix("/WEB-INF/view/");
viewResolver.setSuffix(".jsp");
```

View Resolver Configs - Explained

```
InternalResourceViewResolver viewResolver = new InternalResourceViewResolver();

viewResolver.setPrefix("/WEB-INF/view/");
viewResolver.setSuffix(".jsp");
```

/WEB-INF/view/ show-student-list .jsp

View name

Flash Back to XML config (the old way)

File: web.xml

```
<web-app>

</web-app>
```

Flash Back to XML config (the old way)

File: web.xml

```
<web-app>

</web-app>
```

Just an FYI

Flash Back to XML config (the old way)

File: web.xml

Just an FYI

```
<web-app>

    <servlet>
        <servlet-name>dispatcher</servlet-name>
        <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>

        <init-param>
            <param-name>contextConfigLocation</param-name>
            <param-value>/WEB-INF/spring-mvc-demo-servlet.xml</param-value>
        </init-param>

        <load-on-startup>1</load-on-startup>
    </servlet>





</web-app>
```

Flash Back to XML config (the old way)

File: web.xml

Just an FYI

```
<web-app>

    <servlet>
        <servlet-name>dispatcher</servlet-name>
        <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>

        <init-param>
            <param-name>contextConfigLocation</param-name>
            <param-value>/WEB-INF/spring-mvc-demo-servlet.xml</param-value>
        </init-param>

        <load-on-startup>1</load-on-startup>
    </servlet>

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

</web-app>
```

Web App Initializer

Web App Initializer

- Spring MVC provides support for web app initialization

Web App Initializer

- Spring MVC provides support for web app initialization
- Makes sure your code is automatically detected

Web App Initializer

- Spring MVC provides support for web app initialization
- Makes sure your code is automatically detected
- Your code is used to initialize the servlet container

Web App Initializer

- Spring MVC provides support for web app initialization
- Makes sure your code is automatically detected
- Your code is used to initialize the servlet container

AbstractAnnotationConfigDispatcherServletInitializer

Web App Initializer (more info)

AbstractAnnotationConfigDispatcherServletInitializer

Web App Initializer (more info)

AbstractAnnotationConfigDispatcherServletInitializer

- Your TO DO list

Web App Initializer (more info)

AbstractAnnotationConfigDispatcherServletInitializer

- Your TO DO list
  - Extend this abstract base class

Web App Initializer (more info)

AbstractAnnotationConfigDispatcherServletInitializer

- Your TO DO list
  - Extend this abstract base class
  - Override required methods

Web App Initializer (more info)

AbstractAnnotationConfigDispatcherServletInitializer

- Your TO DO list
  - Extend this abstract base class
  - Override required methods
  - Specify servlet mapping and location of your app config

Step 3: Create Spring Dispatcher Servlet Initializer

File:MySpringMvcDispatcherServletInitializer.java

```
import org.springframework.web.servlet.support.AbstractAnnotationConfigDispatcherServletInitializer;

public class MySpringMvcDispatcherServletInitializer extends AbstractAnnotationConfigDispatcherServletInitializer {

}
```

Step 3: Create Spring Dispatcher Servlet Initializer

File:MySpringMvcDispatcherServletInitializer.java

```
import org.springframework.web.servlet.support.AbstractAnnotationConfigDispatcherServletInitializer;

public class MySpringMvcDispatcherServletInitializer extends AbstractAnnotationConfigDispatcherServletInitializer {

    @Override
    protected Class<?>[] getRootConfigClasses() {
        // TODO Auto-generated method stub
        return null;
    }

}
```

Step 3: Create Spring Dispatcher Servlet Initializer

File:MySpringMvcDispatcherServletInitializer.java

```
import org.springframework.web.servlet.support.AbstractAnnotationConfigDispatcherServletInitializer;

public class MySpringMvcDispatcherServletInitializer extends AbstractAnnotationConfigDispatcherServletInitializer {

    @Override
    protected Class<?>[] getRootConfigClasses() {
        // TODO Auto-generated method stub
        return null;
    }

    @Override
    protected Class<?>[] getServletConfigClasses() {
        return new Class[] { DemoAppConfig.class };
    }

}
```

Step 3: Create Spring Dispatcher Servlet Initializer

File:MySpringMvcDispatcherServletInitializer.java

```
import org.springframework.web.servlet.support.AbstractAnnotationConfigDispatcherServletInitializer;

public class MySpringMvcDispatcherServletInitializer extends AbstractAnnotationConfigDispatcherServletInitializer {

    @Override
    protected Class<?>[] getRootConfigClasses() {
        // TODO Auto-generated method stub
        return null;
    }

    @Override
    protected Class<?>[] getServletConfigClasses() {
        return new Class[] { DemoAppConfig.class };
    }



}
```

Our config class from Step 2

Step 3: Create Spring Dispatcher Servlet Initializer

File:MySpringMvcDispatcherServletInitializer.java

```
import org.springframework.web.servlet.support.AbstractAnnotationConfigDispatcherServletInitializer;

public class MySpringMvcDispatcherServletInitializer extends AbstractAnnotationConfigDispatcherServletInitializer {

    @Override
    protected Class<?>[] getRootConfigClasses() {
        // TODO Auto-generated method stub
        return null;
    }

    @Override
    protected Class<?>[] getServletConfigClasses() {
        return new Class[] { DemoAppConfig.class };
    }

}
```

Step 3: Create Spring Dispatcher Servlet Initializer

File:MySpringMvcDispatcherServletInitializer.java

```
import org.springframework.web.servlet.support.AbstractAnnotationConfigDispatcherServletInitializer;

public class MySpringMvcDispatcherServletInitializer extends AbstractAnnotationConfigDispatcherServletInitializer {

    @Override
    protected Class<?>[] getRootConfigClasses() {
        // TODO Auto-generated method stub
        return null;
    }

    @Override
    protected Class<?>[] getServletConfigClasses() {
        return new Class[] { DemoAppConfig.class };
    }

    @Override
    protected String[] getServletMappings() {
        return new String[] { "/" };
    }

}
```

Step 3: Create Spring Dispatcher Servlet Initializer

File:MySpringMvcDispatcherServletInitializer.java

```java
import org.springframework.web.servlet.support.AbstractAnnotationConfigDispatcherServletInitializer;

public class MySpringMvcDispatcherServletInitializer extends AbstractAnnotationConfigDispatcherServletInitializer {

    @Override
    protected Class<?>[] getRootConfigClasses() {
        // TODO Auto-generated method stub
        return null;
    }

    @Override
    protected Class<?>[] getServletConfigClasses() {
        return new Class[] { DemoAppConfig.class };
    }

    @Override
    protected String[] getServletMappings() {
        return new String[] { "/" };
    }

}
```

```xml
<servlet>
  <servlet-name>dispatcher</servlet-name>
  <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>

  <init-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>/WEB-INF/spring-mvc-demo-servlet.xml</param-value>
  </init-param>

  <load-on-startup>1</load-on-startup>
</servlet>

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

Step 3: Create Spring Dispatcher Servlet Initializer

File:MySpringMvcDispatcherServletInitializer.java

```java
import org.springframework.web.servlet.support.AbstractAnnotationConfigDispatcherServletInitializer;

public class MySpringMvcDispatcherServletInitializer extends AbstractAnnotationConfigDispatcherServletInitializer {

    @Override
    protected Class<?>[] getRootConfigClasses() {
        // TODO Auto-generated method stub
        return null;
    }

    @Override
    protected Class<?>[] getServletConfigClasses() {
        return new Class[] { DemoAppConfig.class };
    }

    @Override
    protected String[] getServletMappings() {
        return new String[] { "/" };
    }

}
```

```xml
<servlet>
  <servlet-name>dispatcher</servlet-name>
  <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>

  <init-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>/WEB-INF/spring-mvc-demo-servlet.xml</param-value>
  </init-param>

  <load-on-startup>1</load-on-startup>
</servlet>

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

Step 3: Create Spring Dispatcher Servlet Initializer

File:MySpringMvcDispatcherServletInitializer.java

```java
import org.springframework.web.servlet.support.AbstractAnnotationConfigDispatcherServletInitializer;

public class MySpringMvcDispatcherServletInitializer extends AbstractAnnotationConfigDispatcherServletInitializer {

    @Override
    protected Class<?>[] getRootConfigClasses() {
        // TODO Auto-generated method stub
        return null;
    }

    @Override
    protected Class<?>[] getServletConfigClasses() {
        return new Class[] { DemoAppConfig.class };
    }

    @Override
    protected String[] getServletMappings() {
        return new String[] { "/" };
    }

}
```

```xml
<servlet>
  <servlet-name>dispatcher</servlet-name>
  <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>

  <init-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>/WEB-INF/spring-mvc-demo-servlet.xml</param-value>
  </init-param>

  <load-on-startup>1</load-on-startup>
</servlet>
```

```xml
<servlet-mapping>
  <servlet-name>dispatcher</servlet-name>
  <url-pattern>/</url-pattern>
</servlet-mapping>
```

Whew!!!!

Now for the easy stuff

Create Spring MVC Controller and view page

easy peazy ...

Step 4: Develop our Spring Controller

File: DemoController.java

```
@Controller
public class DemoController {



}
```

Step 4: Develop our Spring Controller

File: DemoController.java

```
@Controller
public class DemoController {

    @GetMapping("/")
    public String showHome() {

        return "home";
    }

}
```

Step 4: Develop our Spring Controller

File: DemoController.java

```
@Controller
public class DemoController {

    @GetMapping("/")
    public String showHome() {

        return "home";
    }

}
```

/WEB-INF/view/ home .jsp

Step 4: Develop our Spring Controller

File: DemoController.java

```
@Controller
public class DemoController {

    @GetMapping("/")
    public String showHome() {

        return "home";
    }

}
```

View name

/WEB-INF/view/ home .jsp

Step 5: Develop our JSP view page

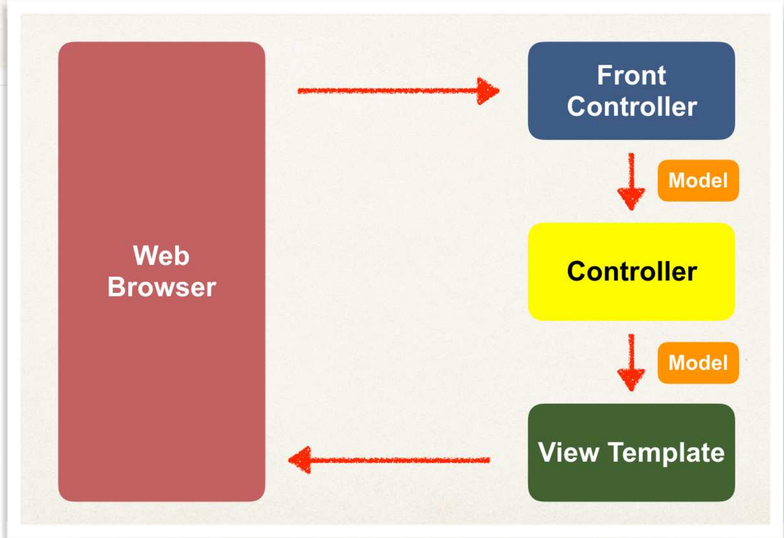
File: /WEB-INF/view/home.jsp

```
<html>

<body>

   Welcome to the luv2code company home page!

</body>

</html>
```

Step 5: Develop our JSP view page

File: /WEB-INF/view/home.jsp

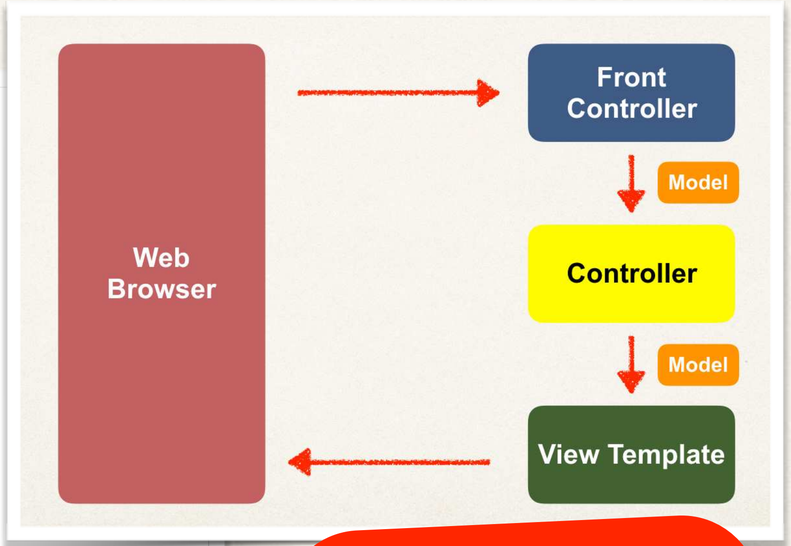
```
<html>

<body>

   Welcome to the luv2code company home page!

</body>

</html>
```



Step 5: Develop our JSP view page

File: /WEB-INF/view/home.jsp

```
<html>

<body>

   Welcome to the luv2code company home page!

</body>

</html>
```

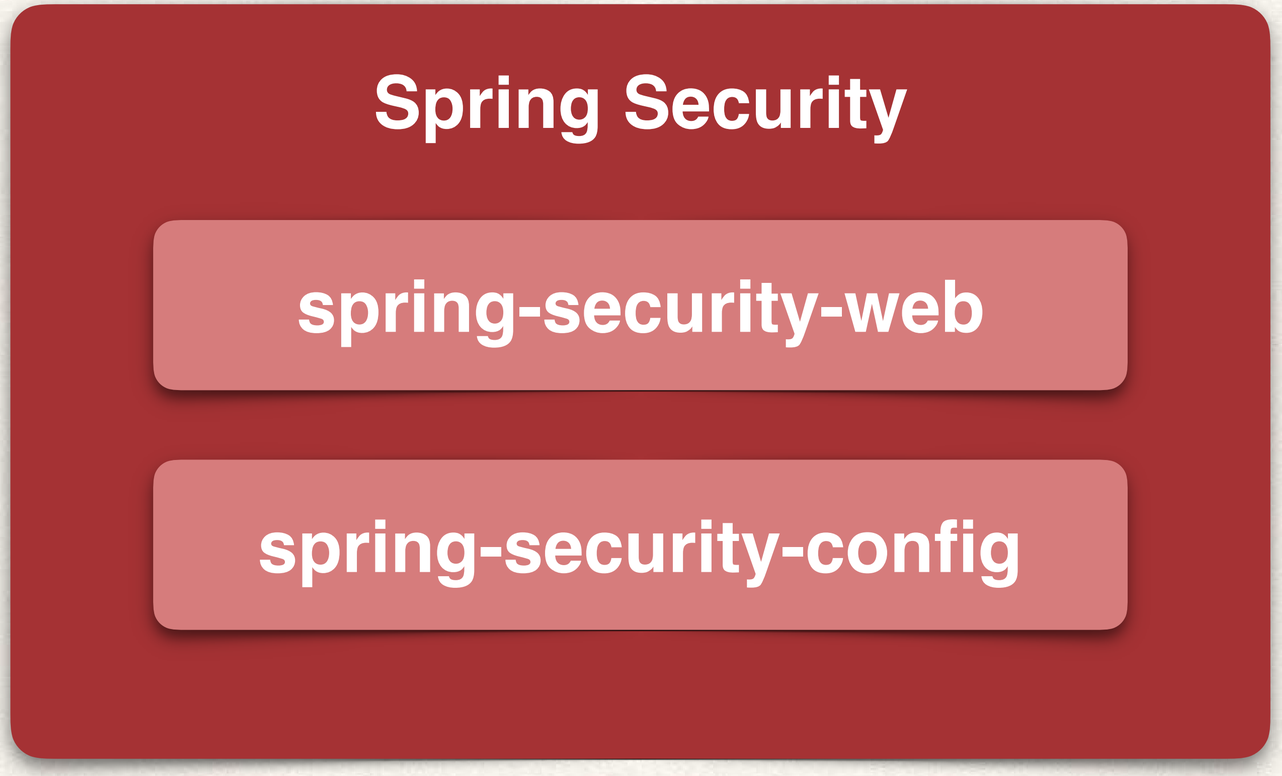


No XML!

Spring Security Project Setup

Add Maven dependencies for Spring Security

Add Maven dependencies for Spring Security



Spring Projects

Spring Projects

Spring Security

Spring Projects

Spring Security

Spring Framework (core, aop, mvc…)

Spring Projects

Spring Security

Spring Framework (core, aop, mvc…)

Spring Projects

Spring Security

Spring Framework (core, aop, mvc…)

- These are two <u>separate</u> projects

Spring Projects

Spring Security

Spring Framework (core, aop, mvc…)

- These are two separate projects
- The projects are on different release cycles

Spring Projects

Spring Security

Spring Framework (core, aop, mvc…)

- These are two separate projects
- The projects are on different release cycles
- Version numbers between projects are generally not in sync (sigh…)

Spring Projects

Spring Security

Spring Framework (core, aop, mvc…)

- These are two separate projects
- The projects are on different release cycles
- Version numbers between projects are generally not in sync (sigh…)
  - Spring team is working to resolve this for future releases …

Spring Security

Spring Security

Spring Framework
(core, aop, web…)

Spring Security

Spring Security

Spring Framework (core, aop, web…)

- Common pitfall is using incompatible projects

Spring Security

Spring Security

Spring Framework (core, aop, web…)

- Common pitfall is using incompatible projects
- Need to find compatible version

Finding Compatible version

Finding Compatible version

- Find desired version of Spring Security in Maven Central Repo
  - spring-security-web

Finding Compatible version

- Find desired version of Spring Security in Maven Central Repo
  - spring-security-web
- Look at the project POM file
  - Find supporting Spring Framework version

Finding Compatible version

Finding Compatible version

org.springframework.security : spring-security-web : 5.0.0.RELEASE

Click on a link above to browse the repository.

Project Information

GroupId:	org.springframework.security
ArtifactId:	spring-security-web
Version:	5.0.0.RELEASE

Dependency Information

Project Object Model (POM)

```
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-context</artifactId>
  <scope>compile</scope>
  <version>5.0.2.RELEASE</version>
</dependency>
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-core</artifactId>
  <scope>compile</scope>
```

Finding Compatible version

org.springframework.security : spring-security-web : 5.0.0.RELEASE

Click on a link above to browse the repository.

Project Information

GroupId:	org.springframework.security
ArtifactId:	spring-security-web
Version:	5.0.0.RELEASE

Dependency Information

Project Object Model (POM)

```
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-context</artifactId>
  <scope>compile</scope>
  <version>5.0.2.RELEASE</version>
</dependency>
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-core</artifactId>
  <scope>compile</scope>
```

Spring Security version

Finding Compatible version

org.springframework.security : spring-security-web : 5.0.0.RELEASE

Click on a link above to browse the repository.

Project Information

GroupId:	org.springframework.security
ArtifactId:	spring-security-web
Version:	5.0.0.RELEASE

Dependency Information

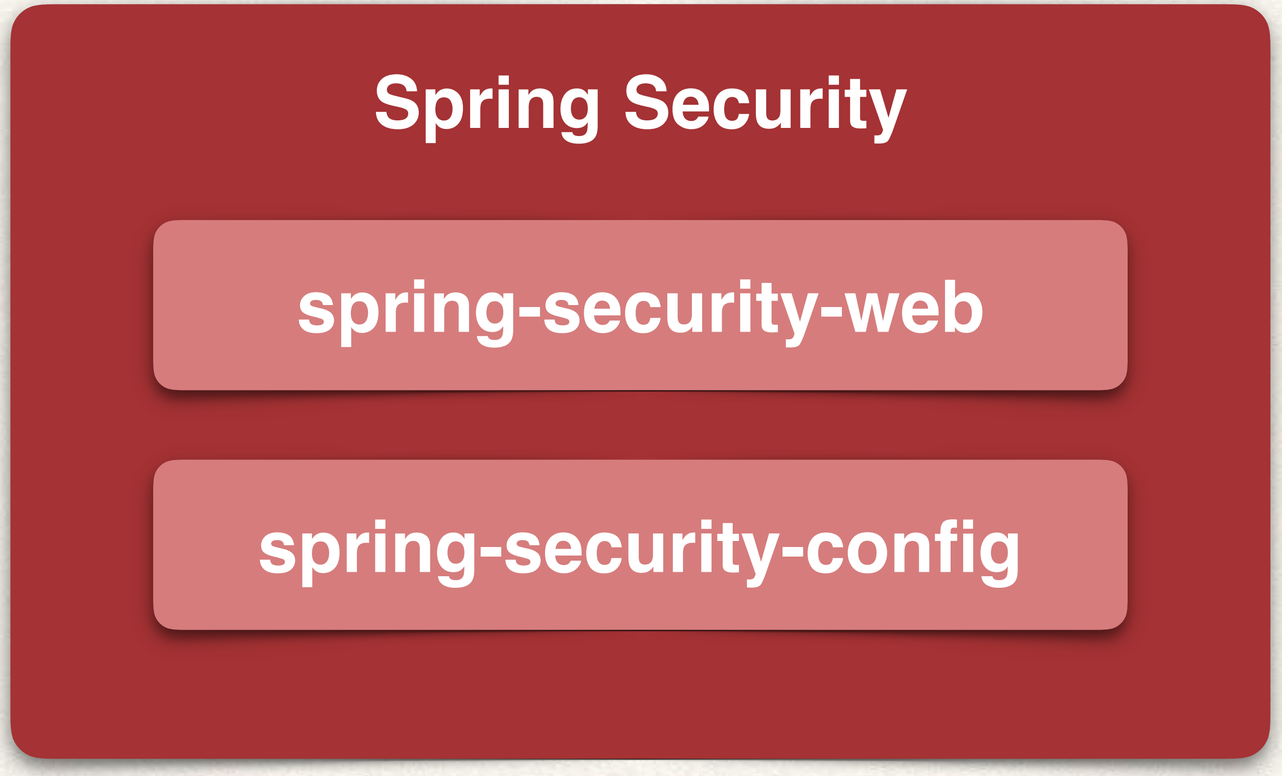
Project Object Model (POM)

```
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-context</artifactId>
  <scope>compile</scope>
  <version>5.0.2.RELEASE</version>
</dependency>
<dependency>
  <groupId>org.springf
  <artifactId>spring-c
  <scope>compile</sc
```

Spring Security version

Spring Framework version

Add Maven dependencies for Spring Security



Add Maven dependencies for Spring Security

File: pom.xml

```
<properties>
    <springframework.version>5.0.2.RELEASE</springframework.version>
    <springsecurity.version>5.0.0.RELEASE</springsecurity.version>
</properties>

...
```

Add Maven dependencies for Spring Security

File: pom.xml

```
<properties>
    <springframework.version>5.0.2.RELEASE</springframework.version>
    <springsecurity.version>5.0.0.RELEASE</springsecurity.version>
</properties>

...

    <!-- Spring -->
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-webmvc</artifactId>
        <version>${springframework.version}</version>
    </dependency>
```

Add Maven dependencies for Spring Security

File: pom.xml

```
<properties>
    <springframework.version>5.0.2.RELEASE</springframework.version>
    <springsecurity.version>5.0.0.RELEASE</springsecurity.version>
</properties>

...

    <!-- 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>
```

Add Maven dependencies for Spring Security

File: pom.xml

```
<properties>
    <springframework.version>5.0.2.RELEASE</springframework.version>
    <springsecurity.version>5.0.0.RELEASE</springsecurity.version>
</properties>

...

    <!-- 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>
```

Configuring Basic Security

Development Process

Step-By-Step

Development Process

Step-By-Step

1. Create Spring Security Initializer

Development Process

Step-By-Step

1. Create Spring Security Initializer

2. Create Spring Security Configuration (@Configuration)

Development Process

Step-By-Step

1. Create Spring Security Initializer

2. Create Spring Security Configuration (@Configuration)

3. Add users, passwords and roles

Spring Security Web App Initializer

Spring Security Web App Initializer

- Spring Security provides support for security initialization

Spring Security Web App Initializer

- Spring Security provides support for security initialization
- Your security code is used to initialize the servlet container

Spring Security Web App Initializer

- Spring Security provides support for security initialization
- Your security code is used to initialize the servlet container
- Special class to register the **Spring Security Filters**

Spring Security Web App Initializer

- Spring Security provides support for security initialization
- Your security code is used to initialize the servlet container
- Special class to register the **Spring Security Filters**

AbstractSecurityWebApplicationInitializer

Spring Security Web App Initializer

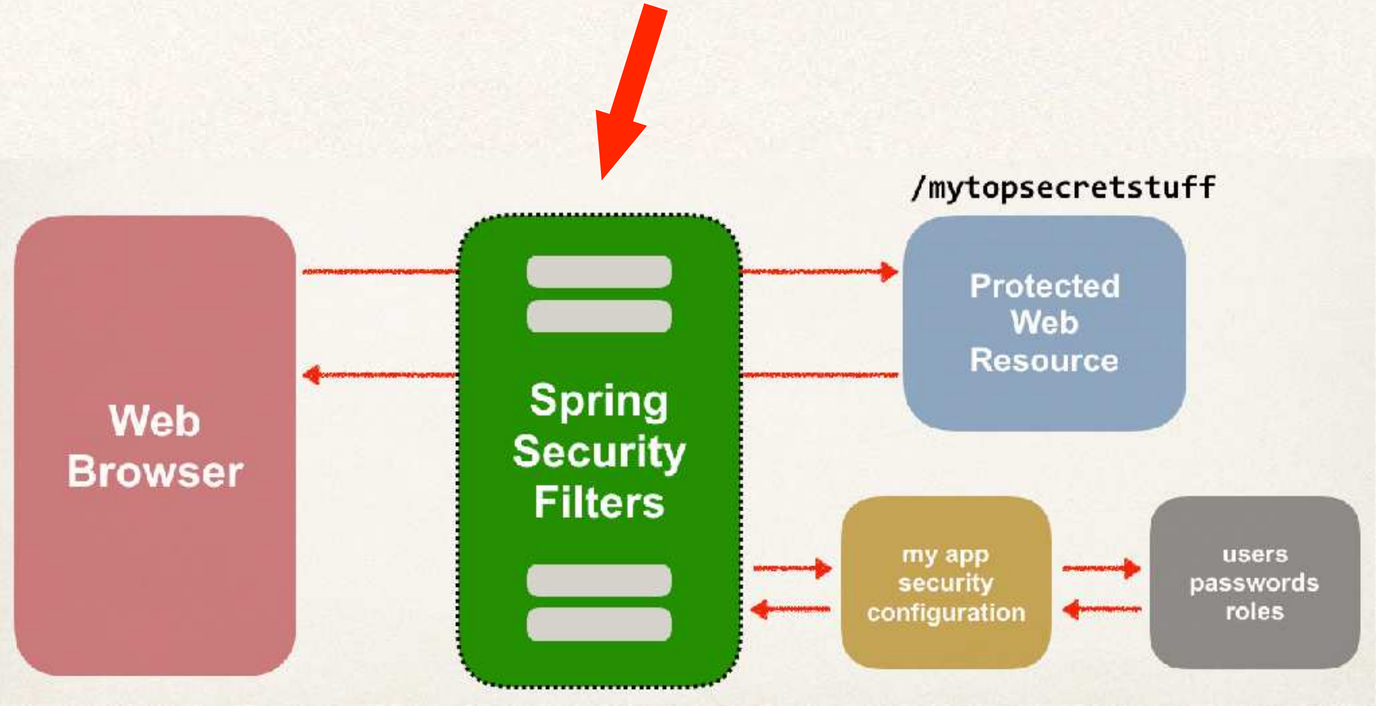
Spring Security Web App Initializer

AbstractSecurityWebApplicationInitializer

Spring Security Web App Initializer

AbstractSecurityWebApplicationInitializer

- Special class to register the **Spring Security Filters**



Spring Security Web App Initializer (more info)

AbstractSecurityWebApplicationInitializer

Spring Security Web App Initializer (more info)

AbstractSecurityWebApplicationInitializer

- Your TO DO list

Spring Security Web App Initializer (more info)

AbstractSecurityWebApplicationInitializer

- Your TO DO list
  - Extend this abstract base class

Spring Security Web App Initializer (more info)

AbstractSecurityWebApplicationInitializer

- Your TO DO list
  - Extend this abstract base class
  - that's it!

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

```
import org.springframework.context.annotation.Configuration;
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;

@Configuration
@EnableWebSecurity
public class DemoSecurityConfig extends WebSecurityConfigurerAdapter {

}
```

Step 3: Add users, passwords and roles

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









  }
```

Step 3: Add users, passwords and roles

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"));

  }
```

Step 3: Add users, passwords and roles

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("
      .withUser(users.username("susan").password("test123").roles(

  }
```

We will add DB support in later videos

(plaintext and encrypted)

Step 3: Add users, passwords and roles

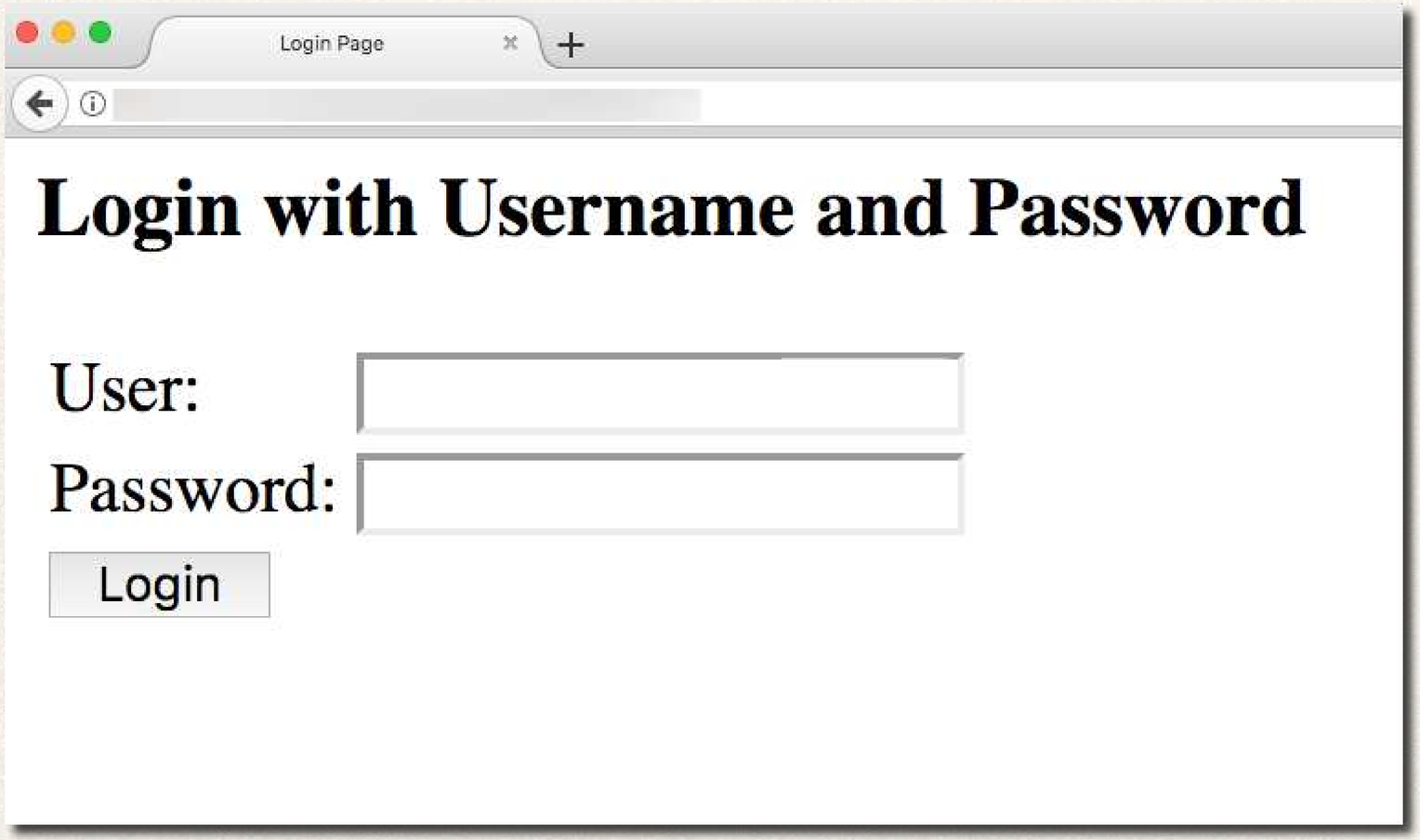
File: DemoSecurityConfig.java

```
@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"));

  }
```

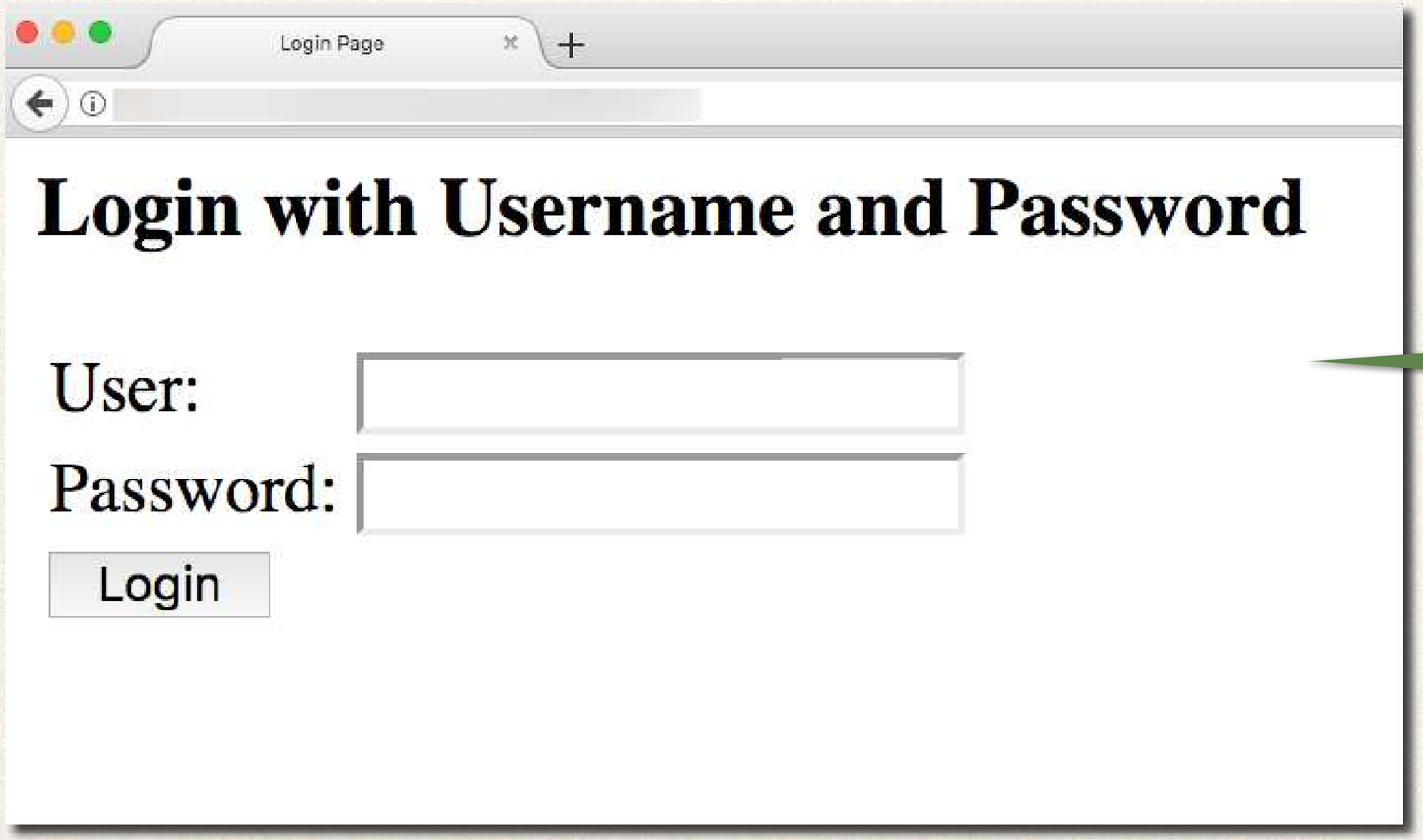
Spring Security - Custom Login Form

Spring Security - Default Login Form

Spring Security - Default Login Form



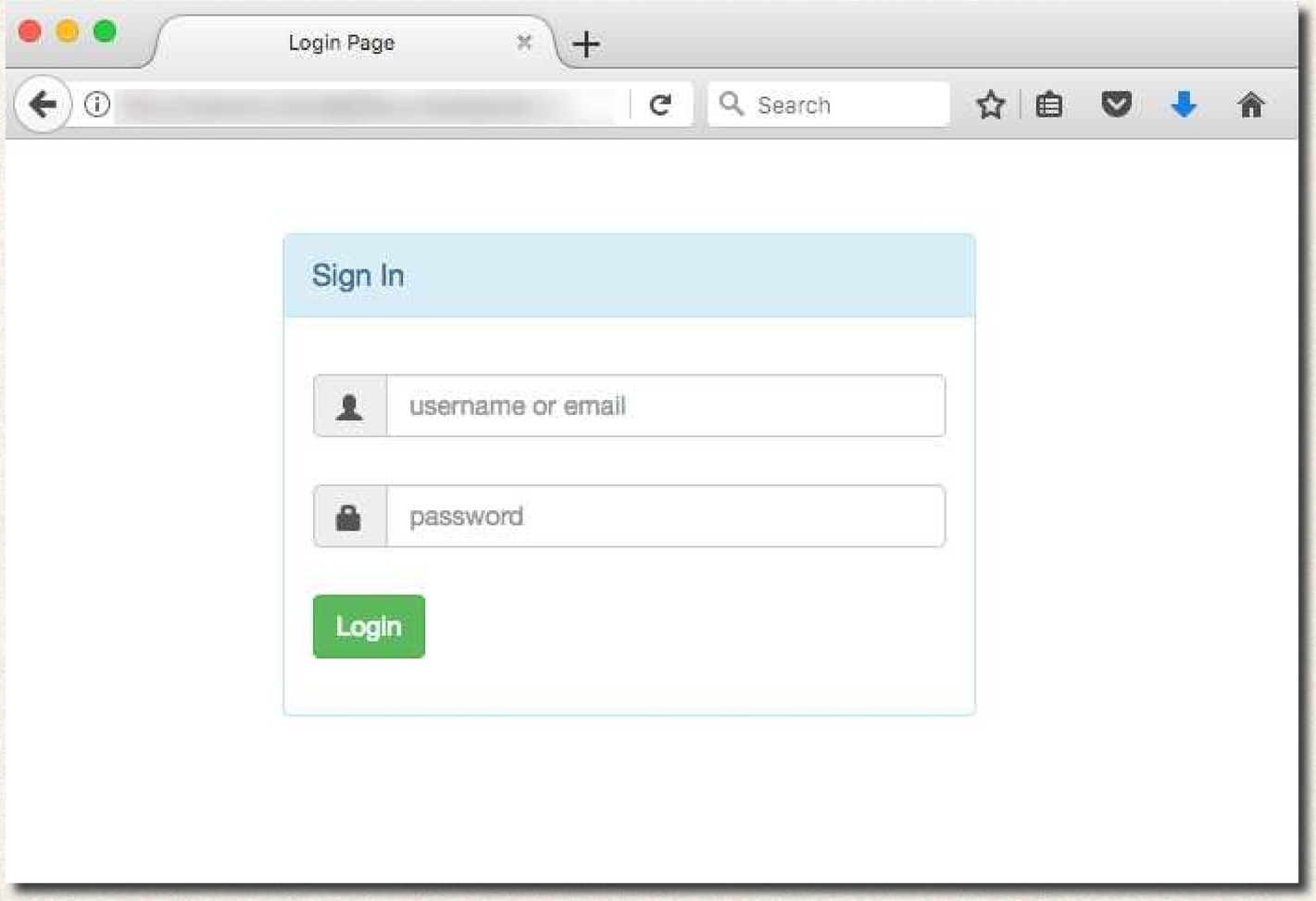
Spring Security - Default Login Form



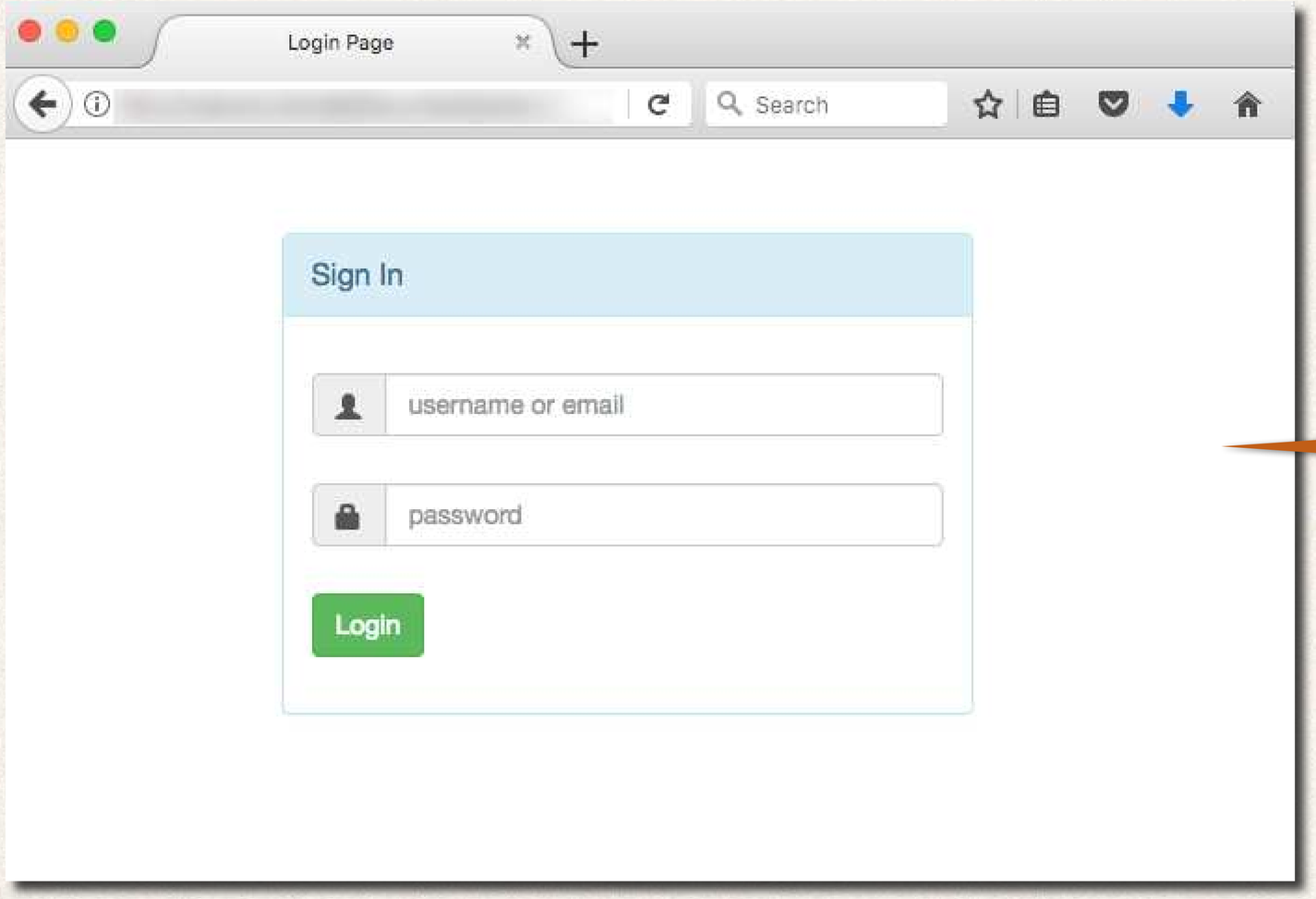
Good for quick start

Your Own Custom Login Form

Your Own Custom Login Form



Your Own Custom Login Form



Your own custom look-and-feel HTML + CSS

Spring Security Overview

Spring Security Overview

Web
Browser

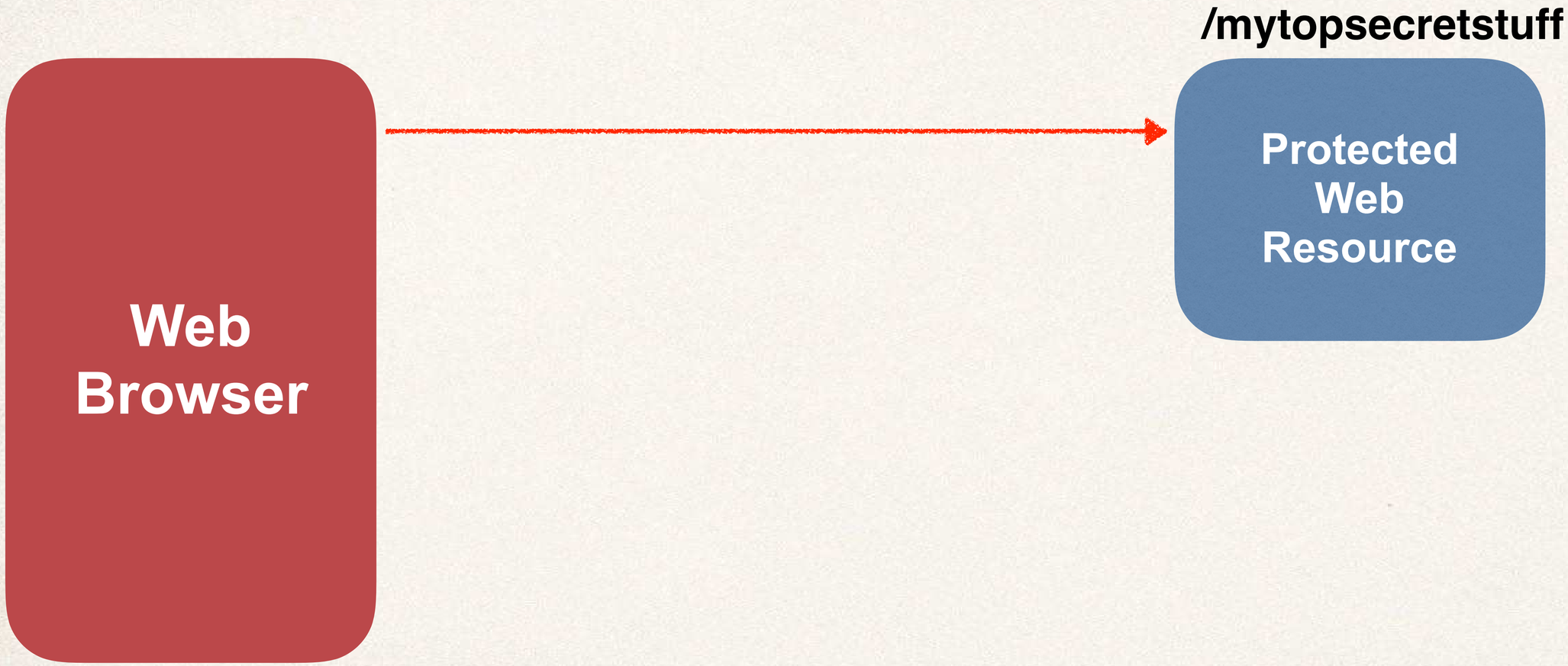
Spring Security Overview

/mytopsecretstuff

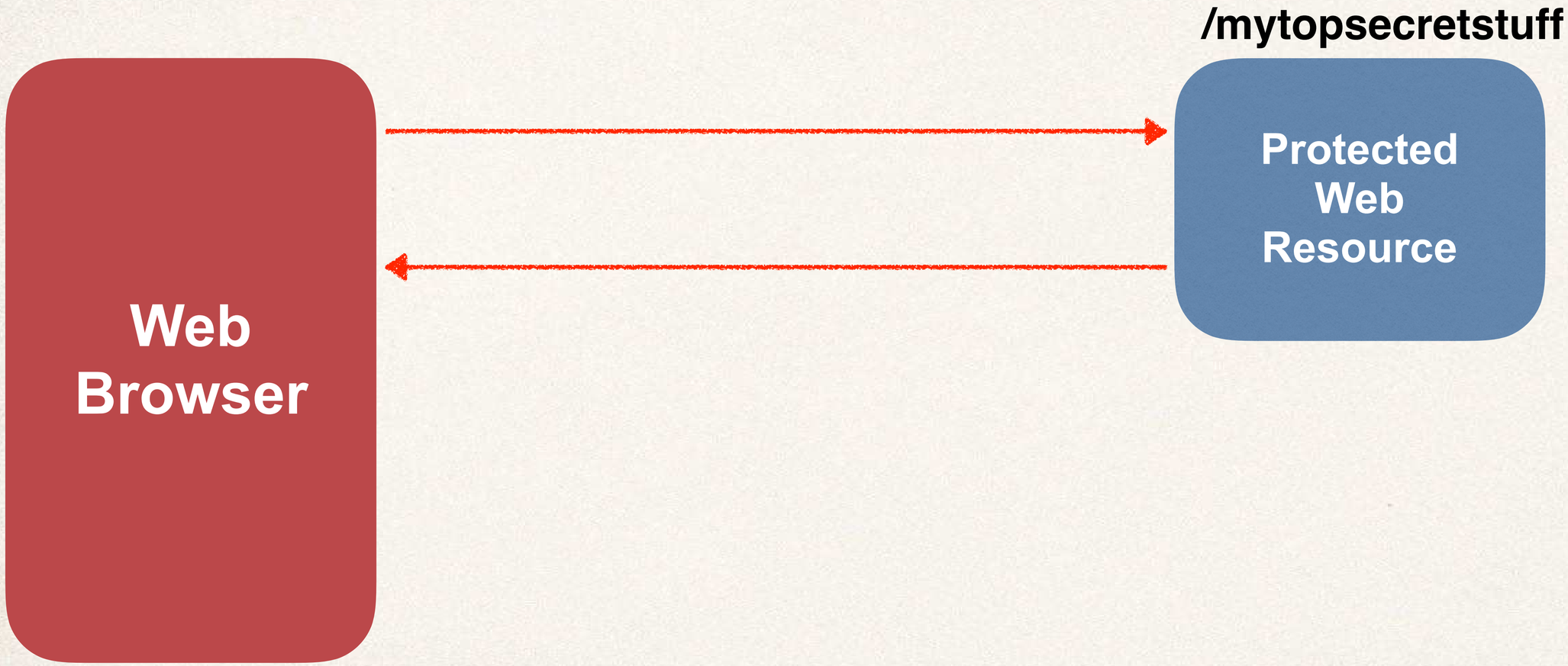
Web Browser

Protected Web Resource

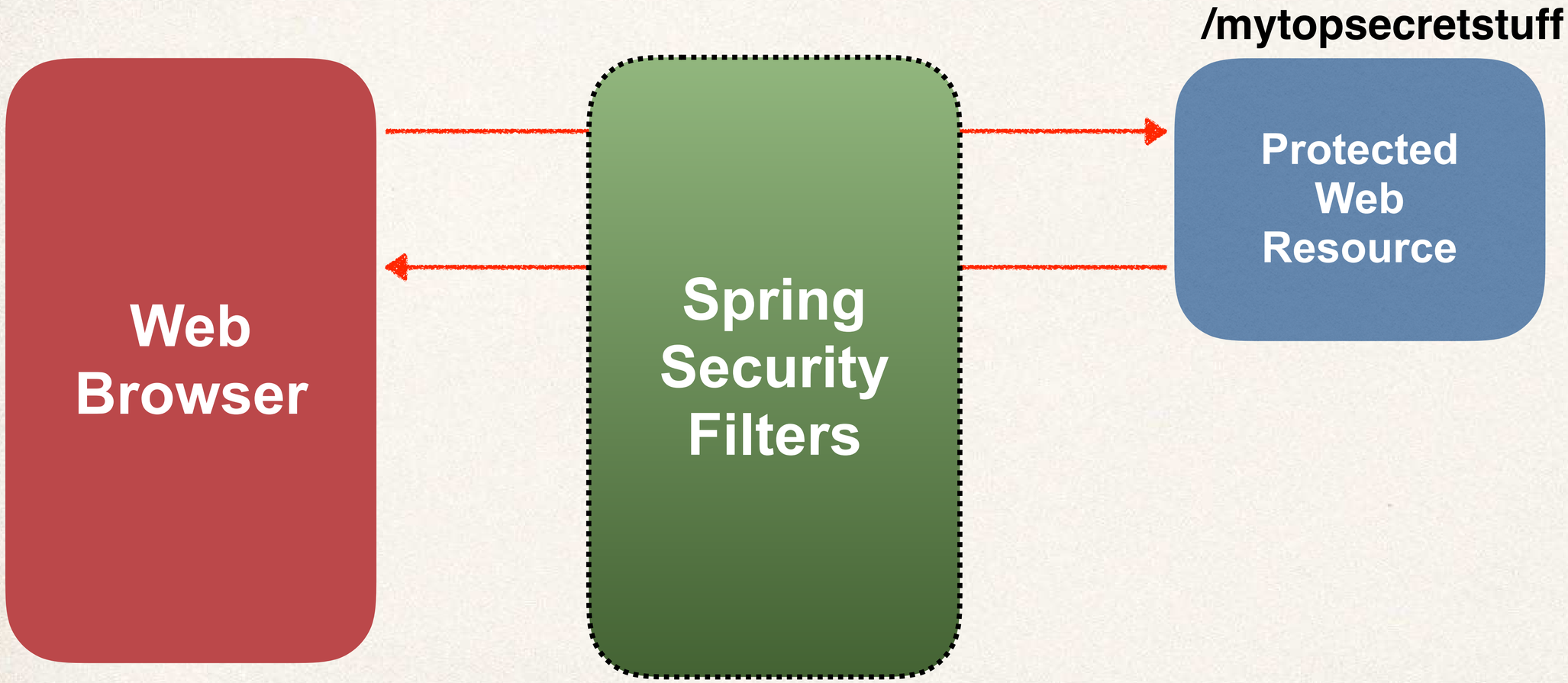
Spring Security Overview



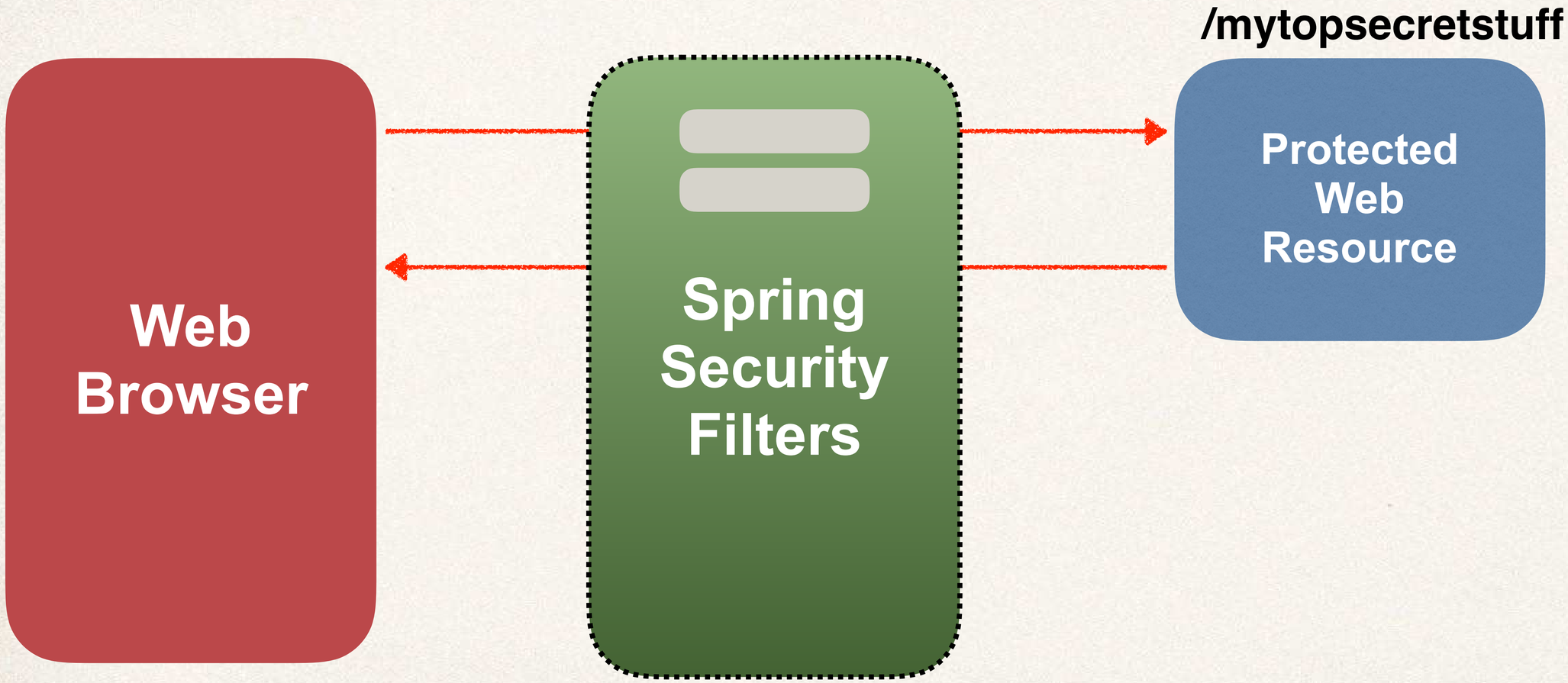
Spring Security Overview



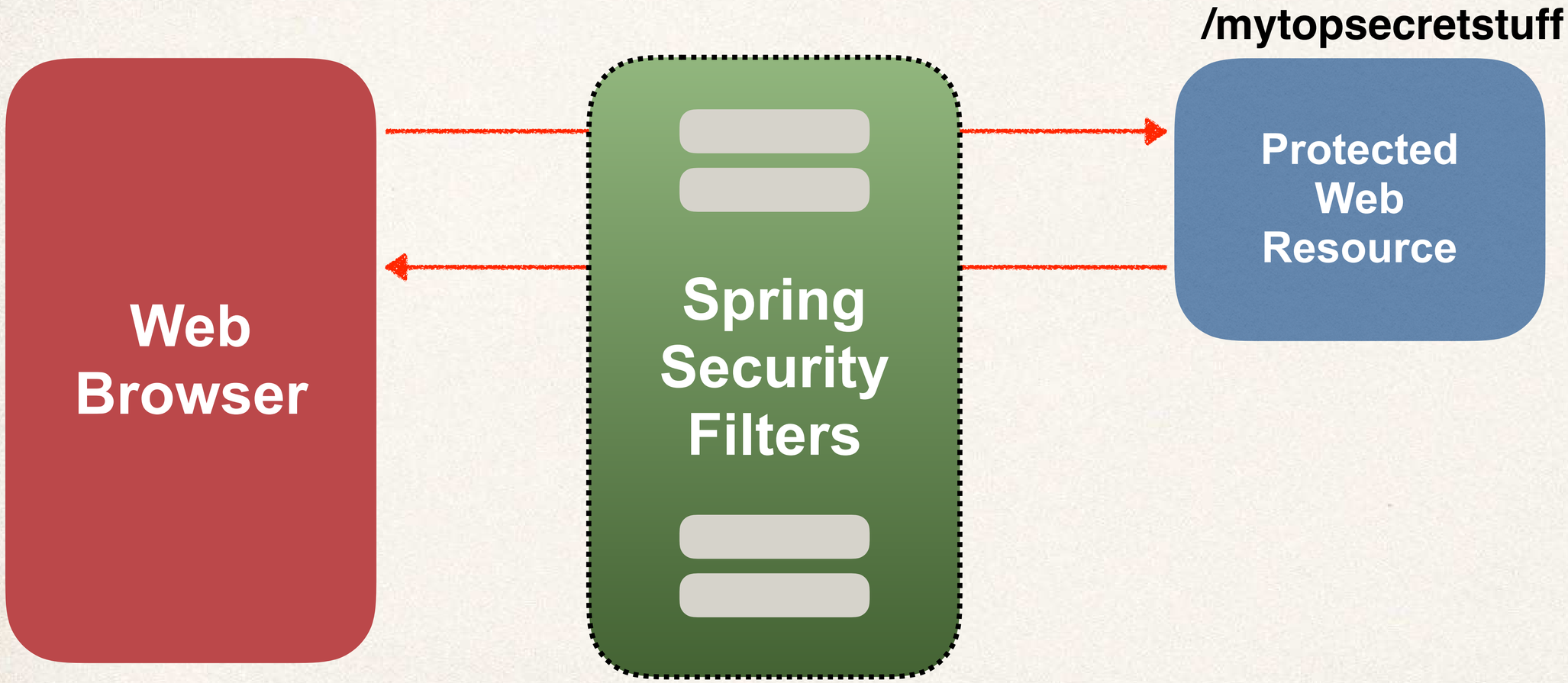
Spring Security Overview



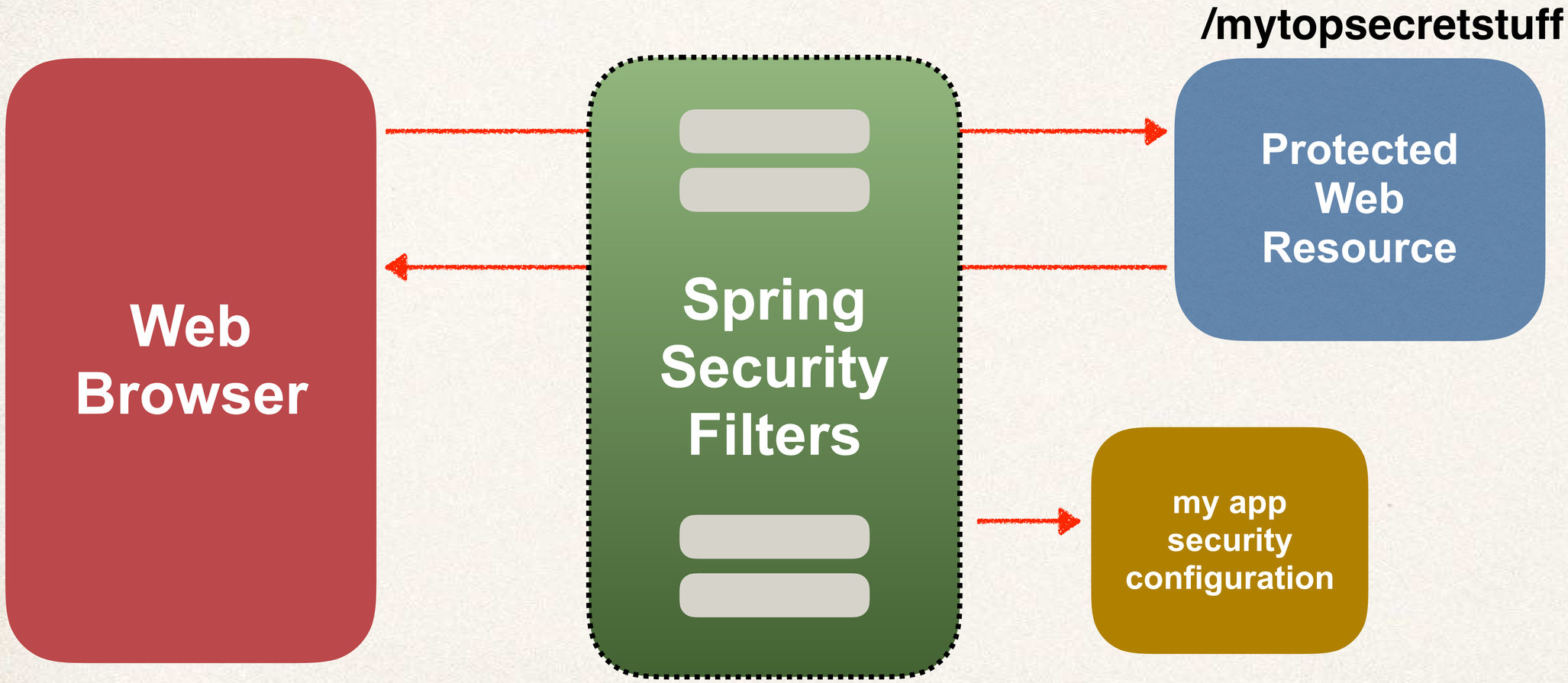
Spring Security Overview



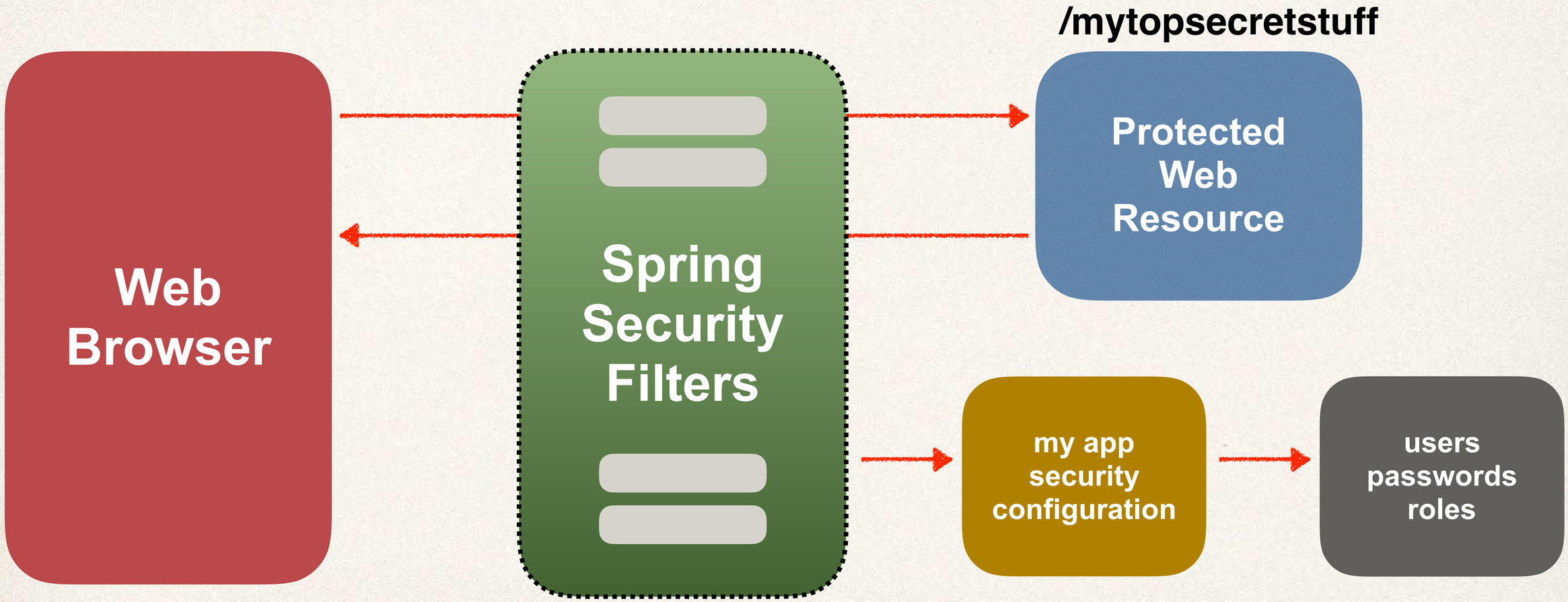
Spring Security Overview



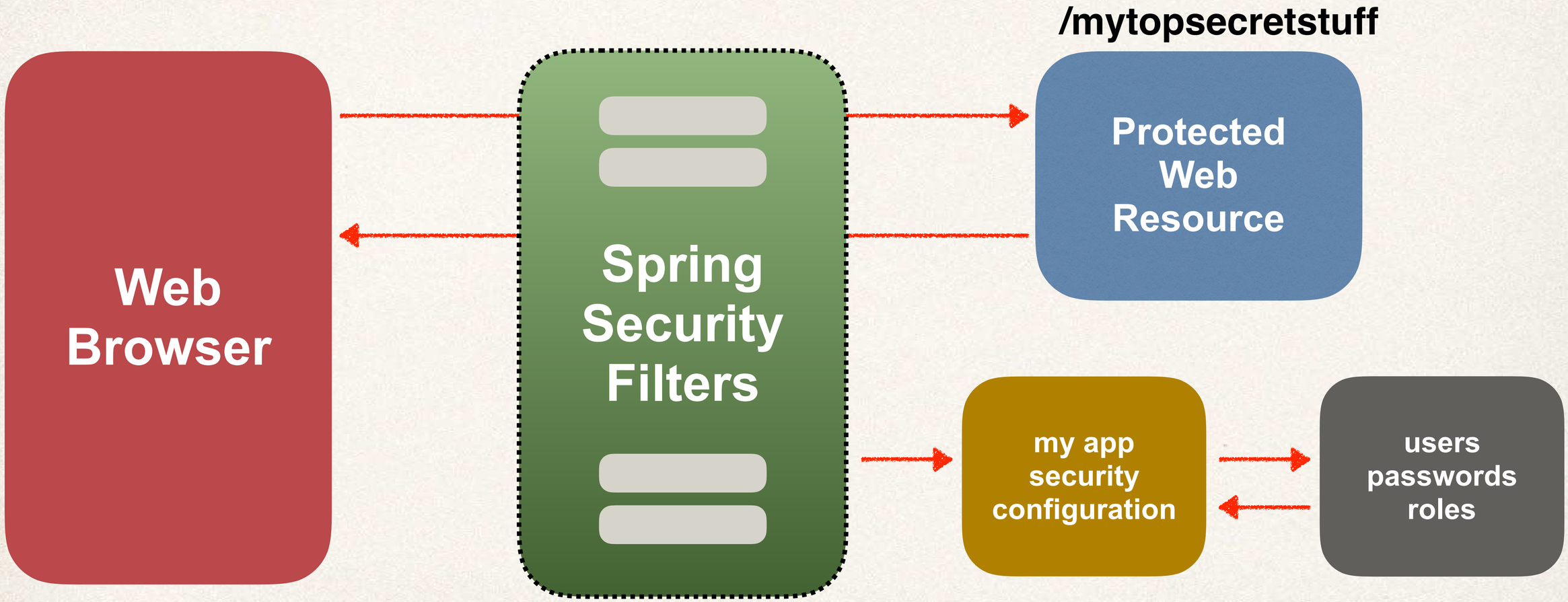
Spring Security Overview



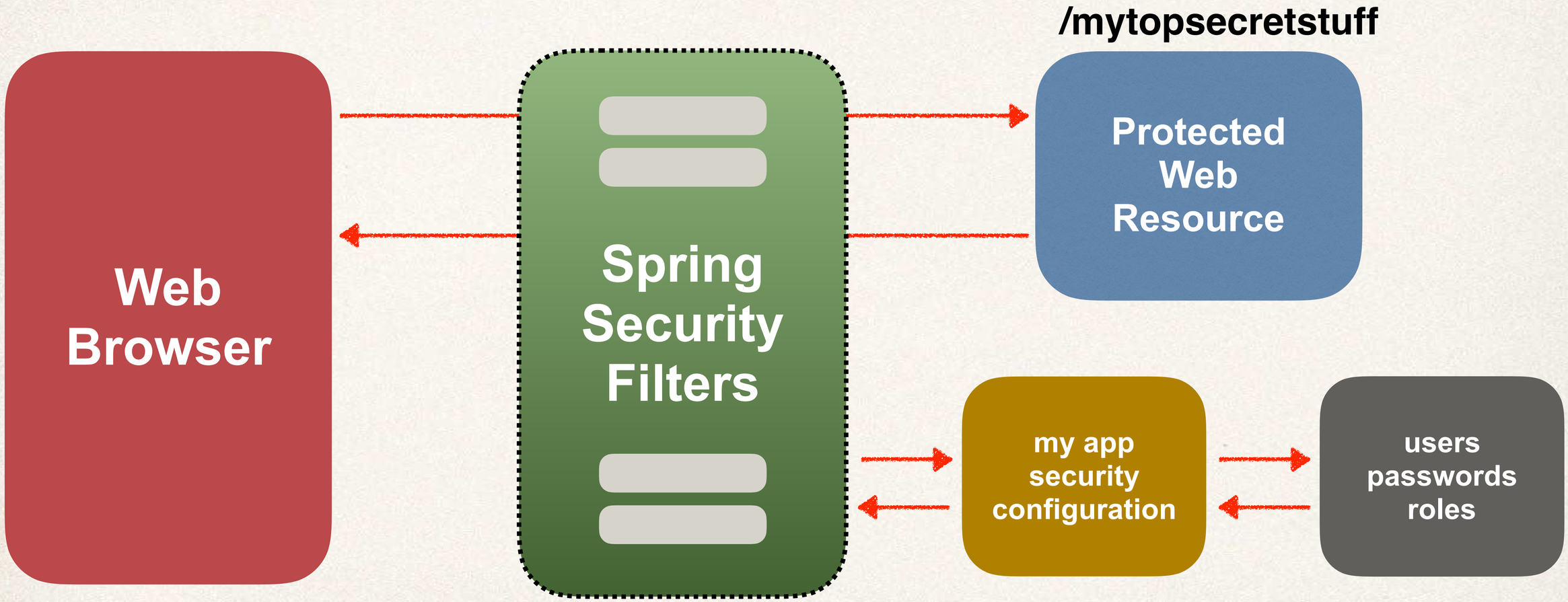
Spring Security Overview



Spring Security Overview

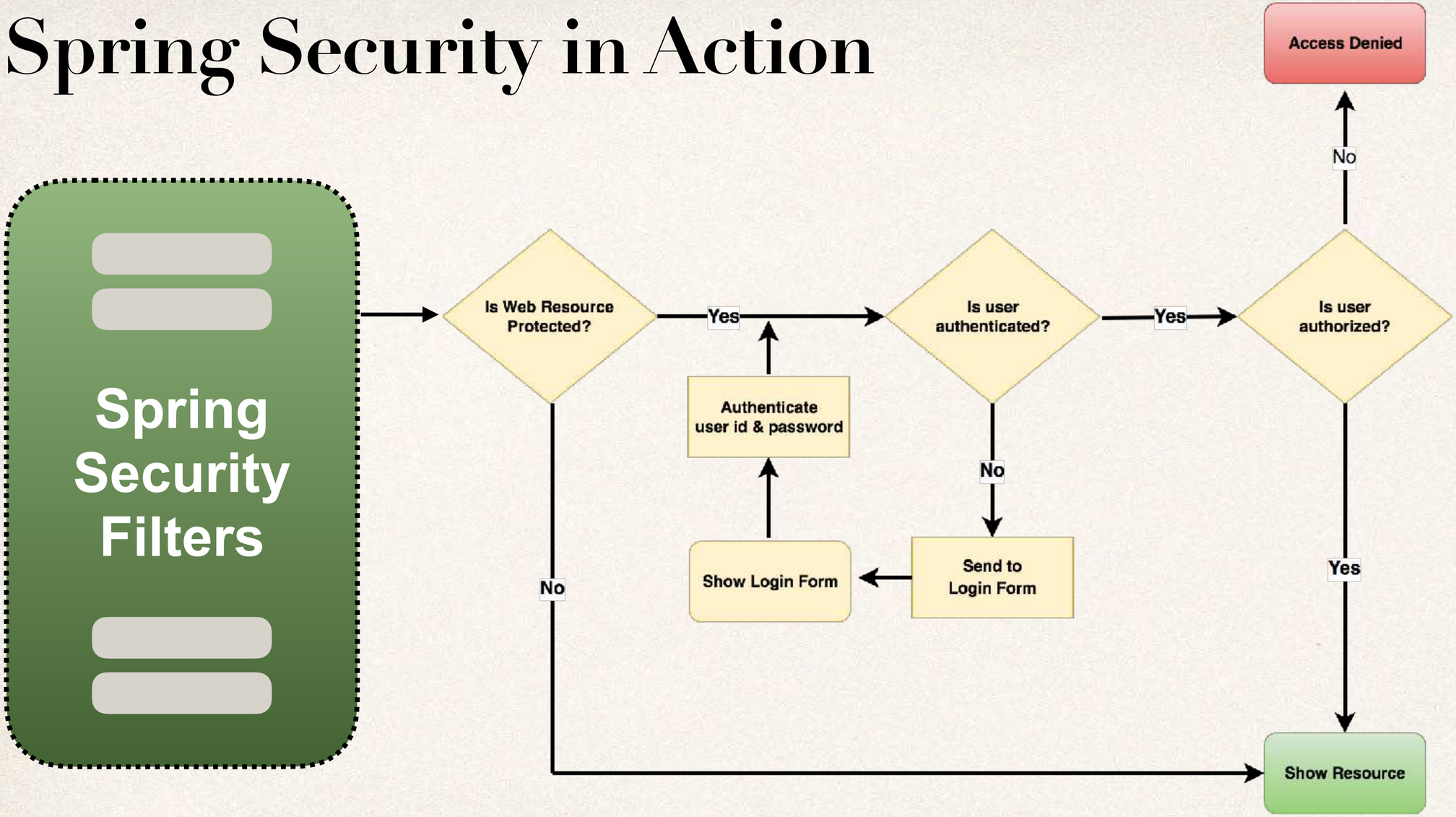


Spring Security Overview

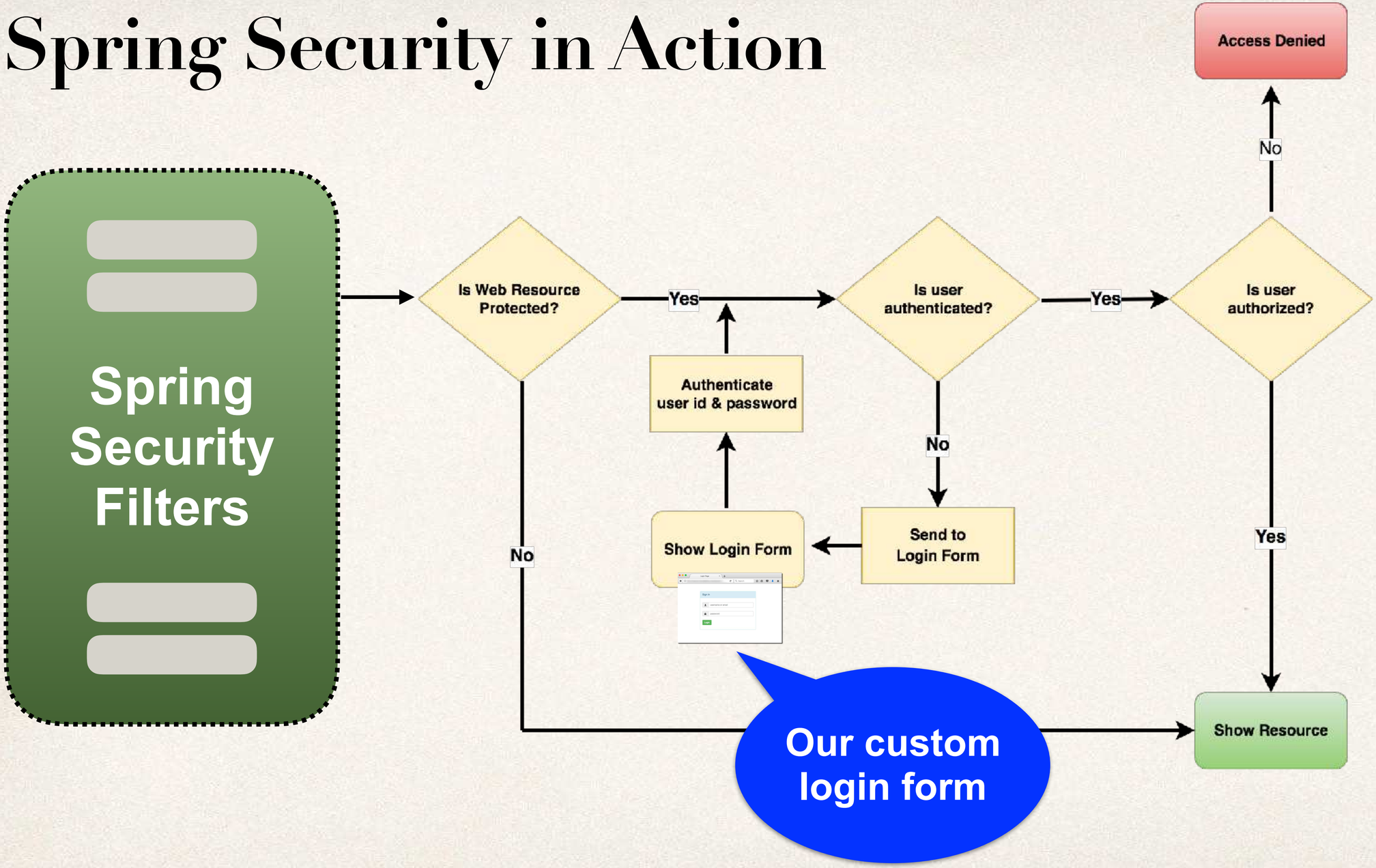


Spring Security in Action

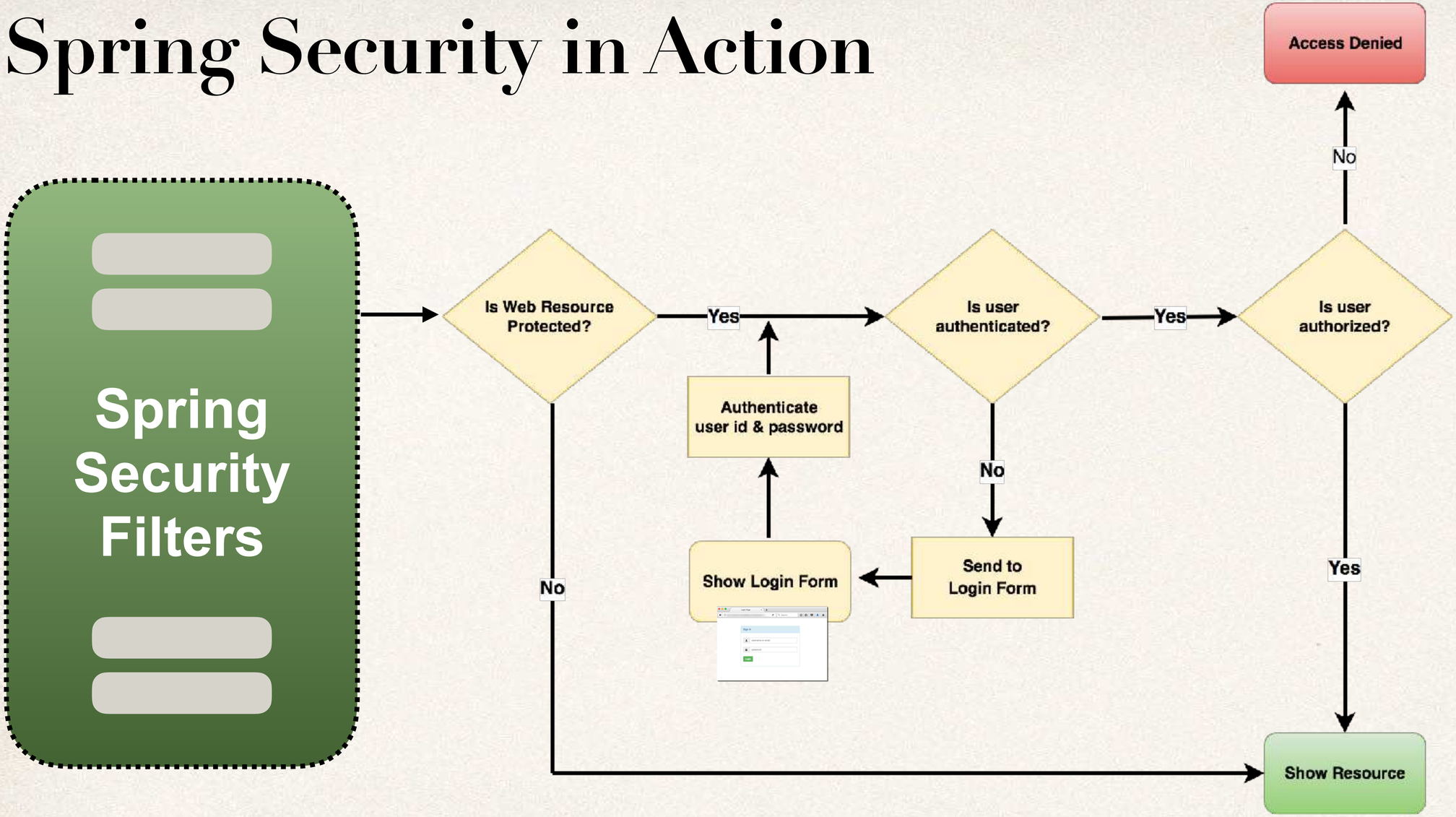
Spring Security in Action



Spring Security in Action



Spring Security in Action



Development Process

Step-By-Step

Development Process

Step-By-Step

1. Modify Spring Security Configuration to reference custom login form

Development Process

Step-By-Step

1. Modify Spring Security Configuration to reference custom login form

2. Develop a Controller to show the custom login form

Development Process

Step-By-Step

1. Modify Spring Security Configuration to reference custom login form

2. Develop a Controller to show the custom login form

3. Create custom login form

Development Process

Step-By-Step

1. Modify Spring Security Configuration to reference custom login form

2. Develop a Controller to show the custom login form

3. Create custom login form
   - HTML (CSS optional)

Development Process

Step-By-Step

1. Modify Spring Security Configuration to reference custom login form

2. Develop a Controller to show the custom login form

3. Create custom login form
   - HTML (CSS optional)
   - Spring MVC form tag <form:form>

Step 1: Modify Spring Security Configuration

Step 1: Modify Spring Security Configuration

File: DemoSecurityConfig.java

```
@Configuration
@EnableWebSecurity
public class DemoSecurityConfig extends WebSecurityConfigurerAdapter {

  // override method to configure users for in-memory authentication …

}
```

Step 1: Modify Spring Security Configuration

File: DemoSecurityConfig.java

```
@Configuration
@EnableWebSecurity
public class DemoSecurityConfig extends WebSecurityConfigurerAdapter {

  // override method to configure users for in-memory authentication …

}
```

Method	Description

Step 1: Modify Spring Security Configuration

File: DemoSecurityConfig.java

```
@Configuration
@EnableWebSecurity
public class DemoSecurityConfig extends WebSecurityConfigurerAdapter {

  // override method to configure users for in-memory authentication …

}
```

Method	Description
`configure(AuthenticationManagerBuilder)`	Configure users (in memory, database, ldap, etc)

Step 1: Modify Spring Security Configuration

File: DemoSecurityConfig.java

```
@Configuration
@EnableWebSecurity
public class DemoSecurityConfig extends WebSecurityConfigurerAdapter {

  // override method to configure users for in-memory authentication …

}
```

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 {

      // add our users for in memory authentication
      …
   }









}
```

Step 1: Modify 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
      …
   }

   @Override
   protected void configure(HttpSecurity http) throws Exception {

      http.authorizeRequests()
            .anyRequest().authenticated()
          .and()
          .formLogin()
            .loginPage("/showMyLoginPage")
            .loginProcessingUrl("/authenticateTheUser")
            .permitAll();

   }
}
```

Step 1: Modify 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
      …
   }

   @Override
   protected void configure(HttpSecurity http) throws Exception {

      http.authorizeRequests()
            .anyRequest().authenticated()
          .and()
          .formLogin()
            .loginPage("/showMyLoginPage")
            .loginProcessingUrl("/authenticateTheUser")
            .permitAll();

   }
}
```

Configure security of web paths in application, login, logout etc

Step 1: Modify Spring Security Configuration

```
@Override
protected void configure(HttpSecurity http) throws Exception {

  http.authorizeRequests()
        .anyRequest().authenticated()
      .and()
      .formLogin()
        .loginPage("/showMyLoginPage")
        .loginProcessingUrl("/authenticateTheUser")
        .permitAll();

}
```

Step 1: Modify Spring Security Configuration

```
@Override
protected void configure(HttpSecurity http) throws Exception {

  http.authorizeRequests()
        .anyRequest().authenticated()
      .and()
      .formLogin()
        .loginPage("/showMyLoginPage")
        .loginProcessingUrl("/authenticateTheUser")
        .permitAll();

}
```

Restrict access based on the HttpServletRequest

Step 1: Modify Spring Security Configuration

```
@Override
protected void configure(HttpSecurity http) throws Exception {

  http.authorizeRequests()
        .anyRequest().authenticated()
      .and()
      .formLogin()
        .loginPage("/showMyLoginPage")
        .loginProcessingUrl("/authenticateTheUser")
        .permitAll();

}
```

Any request to the app must be authenticated (ie logged in)

Step 1: Modify Spring Security Configuration

```
@Override
protected void configure(HttpSecurity http) throws Exception {

  http.authorizeRequests()
        .anyRequest().authenticat
      .and()
      .formLogin()
        .loginPage("/showMyLoginPage")
        .loginProcessingUrl("/authenticateTheUser")
        .permitAll();

}
```

We are customizing the form login process

Step 1: Modify Spring Security Configuration

```
@Override
protected void configure(HttpSecurity ht

  http.authorizeRequests()
        .anyRequest().authenticated()
      .and()
      .formLogin()
        .loginPage("/showMyLoginPage")
        .loginProcessingUrl("/authenticateTheUser")
        .permitAll();

}
```

Show our custom form at the request mapping

"/showMyLoginPage"

Step 1: Modify Spring Security Configuration

```
@Override
protected void configure(HttpSecurity http

  http.authorizeRequests()
        .anyRequest().authenticated()
      .and()
      .formLogin()
        .loginPage("/showMyLoginPage")
        .loginProcessingUrl("/authenticateTheUser")
        .permitAll();

}
```

Login form should POST data to this URL for processing

(check user id and password)

Step 1: Modify Spring Security Configuration

```
@Override
protected void configure(HttpSecur

  http.authorizeRequests()
        .anyRequest().authenticated
      .and()
      .formLogin()
        .loginPage("/showMyLoginPage")
        .loginProcessingUrl("/authenticateTheUser")
        .permitAll();

}
```

You can give ANY values
for this configuration.

Just stay consistent in your app

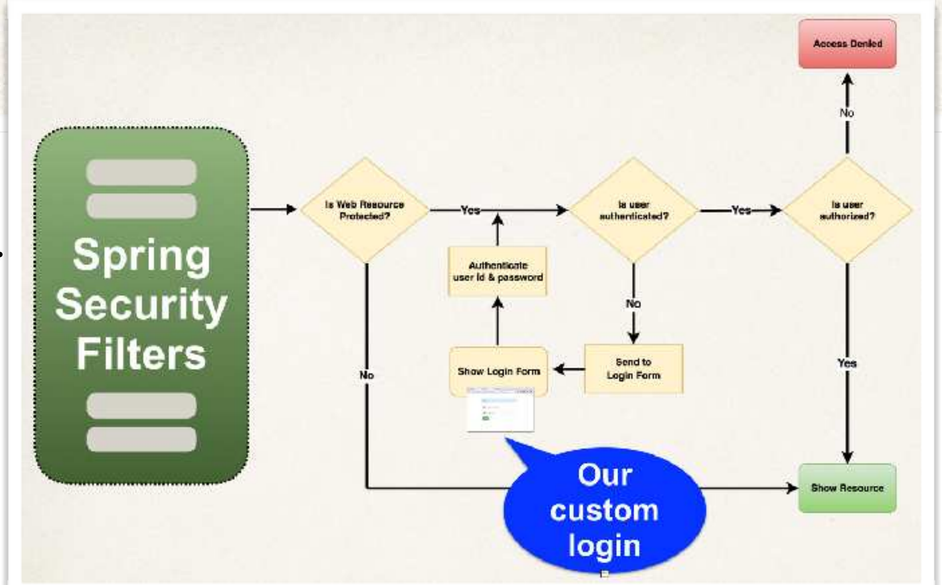
Step 1: Modify Spring Security Configuration

```
@Override
protected void configure(HttpSecurity http) throws Exception {

  http.authorizeRequests()
        .anyRequest().authenticated()
      .and()
      .formLogin()
        .loginPage("/showMyLoginPage")
        .loginProcessingUrl("/authenticateTheUser")
        .permitAll();

}
```

Allow everyone to see login page.
No need to be logged in.

Step 1: Modify Spring Security Configuration

```
@Override
protected void configure(HttpSecurity

  http.authorizeRequests()
        .anyRequest().authenticated()
      .and()
      .formLogin()
        .loginPage("/showMyLoginPage")
        .loginProcessingUrl("/authenticateTheUser")
        .permitAll();

}
```



Step 1: Modify Spring Security Configuration

```
@Override
protected void configure(HttpSecuri

  http.authorizeRequests()
        .anyRequest().authenticated(
      .and()
      .formLogin()
        .loginPage("/showMyLoginPage")
        .loginProcessingUrl("/authenticateTheUser")
        .permitAll();

}
```

TO DO
We need to create a controller for this request mapping

"/showMyLoginPage"

Step 1: Modify Spring Security Configuration

```
@Override
protected void configure(HttpSecuri

  http.authorizeRequests()
        .anyRequest().authenticated(
      .and()
      .formLogin()
        .loginPage("/showMyLoginPage")
        .loginProcessingUrl("/authenticateTheUser")
        .permitAll();

}
```

TO DO
We need to create a controller for this request mapping

"/showMyLoginPage"

No Controller Request Mapping required for this.
We get this for free :-)

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

File: LoginController.java

```
@Controller
public class LoginController {





}
```

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";

  }
}
```

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";

  }
}
```

```
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";

  }
}
```

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";

  }
}
```

View name

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

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

File: LoginController.java

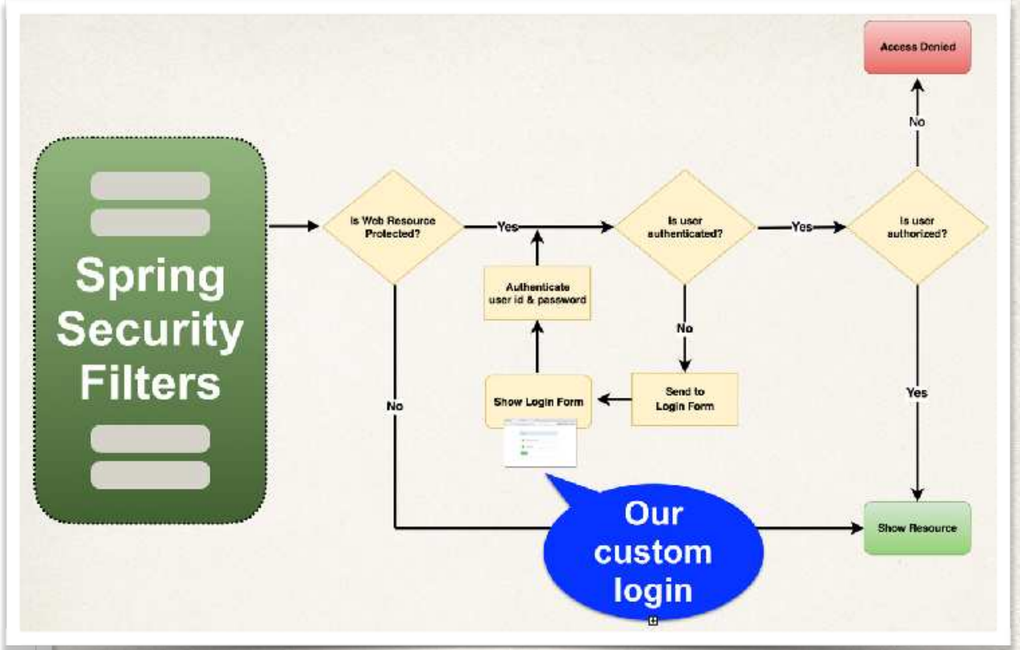
```
@Controller
public class LoginController {

  @GetMapping("/showMyLoginPage")
  public String showMyLoginPage() {

    return "plain-login";

  }
}
```

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

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";

  }
}
```



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

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";

  }
}
```

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

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";

  }
}
```

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

TO DO
We need to create this file

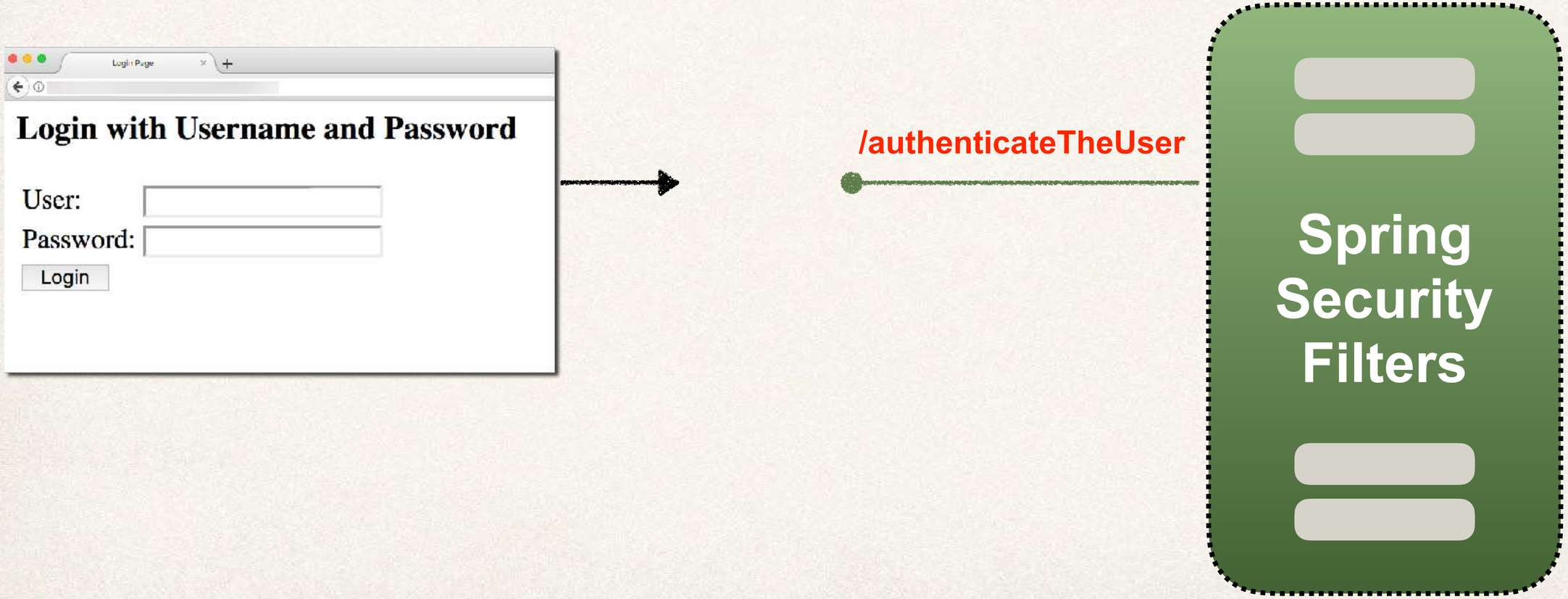
Step 3: Create custom login form

Step 3: Create custom login form

- Send data to login processing URL: **/authenticateTheUser**

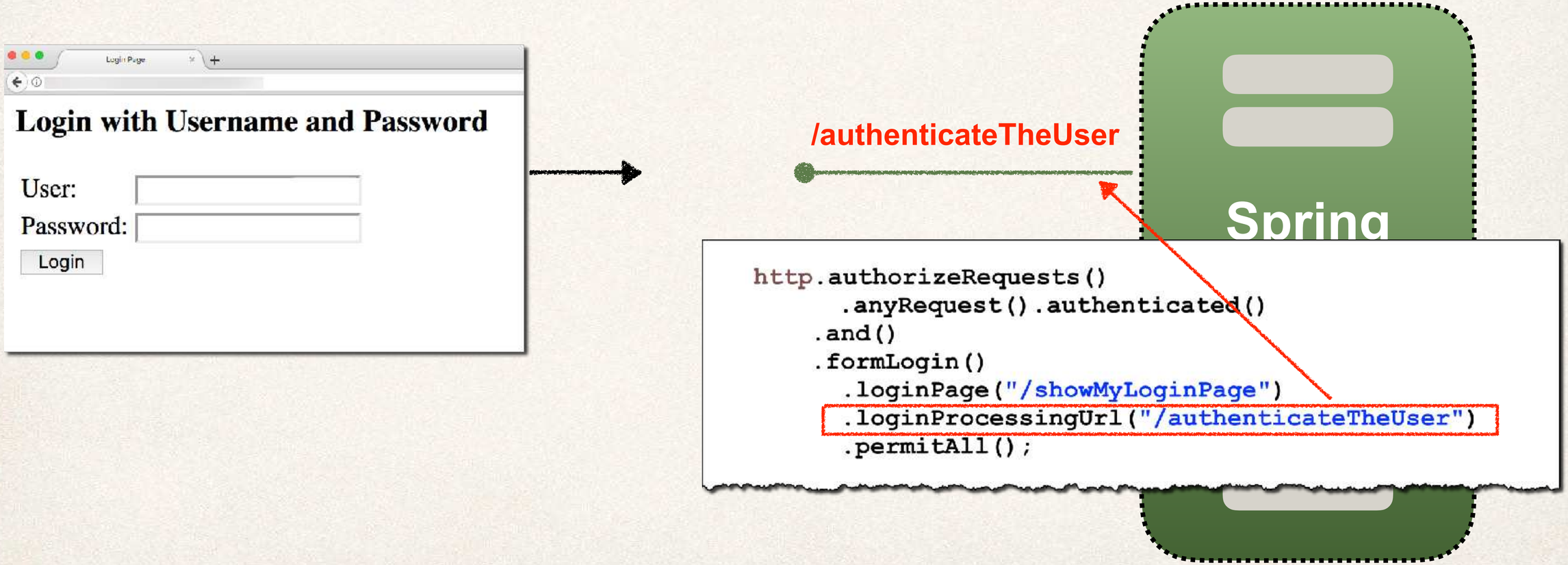
Step 3: Create custom login form

- Send data to login processing URL: **/authenticateTheUser**



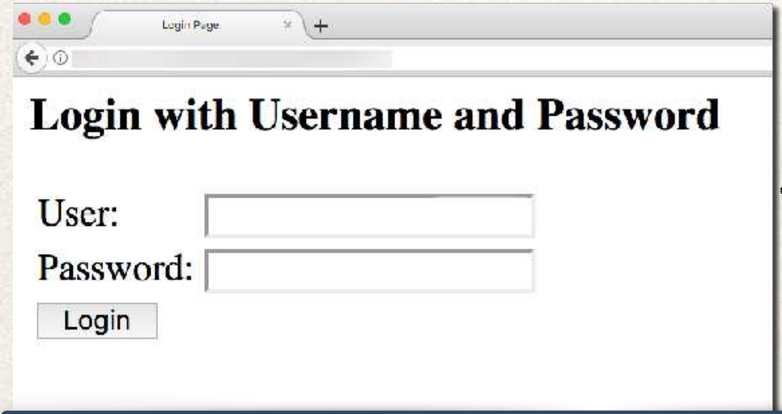
Step 3: Create custom login form

- Send data to login processing URL: **/authenticateTheUser**



Step 3: Create custom login form

- Send data to login processing URL: **/authenticateTheUser**



/authenticateTheUser

Spring

```
http.authorizeRequests()
      .anyRequest().authenticated()
    .and()
    .formLogin()
      .loginPage("/showMyLoginPage")
      .loginProcessingUrl("/authenticateTheUser")
      .permitAll();
```

You can give ANY values for this configuration.

Just stay consistent in your app

Step 3: Create custom login form

Step 3: Create custom login form

- Send data to login processing URL: **/authenticateTheUser**

Step 3: Create custom login form

- Send data to login processing URL: **/authenticateTheUser**

- Login processing URL will be handled by Spring Security Filters

Step 3: Create custom login form

- Send data to login processing URL: **/authenticateTheUser**
- Login processing URL will be handled by Spring Security Filters
- You get it for free … no coding required

Step 3: Create custom login form

- Send data to login processing URL: **/authenticateTheUser**
- Login processing URL will be handled by Spring Security Filters
- You get it for free … no coding required

This is
Spring Security magic …
LOL

Step 3: Create custom login form

Step 3: Create custom login form

- Send data to login processing URL: **/authenticateTheUser**

Step 3: Create custom login form

- Send data to login processing URL: **/authenticateTheUser**

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

Step 3: Create custom login form

- Send data to login processing URL: **/authenticateTheUser**
  - Must **POST** the data

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

Step 3: Create custom login form

- Send data to login processing URL: **/authenticateTheUser**
  - Must **POST** the data

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

```
http.authorizeRequests()
      .anyRequest().authenticated()
    .and()
    .formLogin()
      .loginPage("/showMyLoginPage")
      .loginProcessingUrl("/authenticateTheUser")
      .permitAll();
```

Step 3: Create custom login form

Best Practice

Step 3: Create custom login form

Best Practice

- **Best practice** is to use the Spring MVC Form tag `<form:form>`

Step 3: Create custom login form

Best Practice

- **Best practice** is to use the Spring MVC Form tag `<form:form>`
  - Provides automatic support for security defenses *(more on this later)*

Step 3: Create custom login form

Best Practice

- **Best practice** is to use the Spring MVC Form tag `<form:form>`
  - Provides automatic support for security defenses (*more on this later*)

```
<form:form action="…" method="…" >
   …
</form:form>
```

Step 3: Create custom login form

Step 3: Create custom login form

- Spring Security defines default names for login form fields

Step 3: Create custom login form

- Spring Security defines default names for login form fields
  - User name field: `username`

Step 3: Create custom login form

- Spring Security defines default names for login form fields
  - User name field: `username`
  - Password field: `password`

Step 3: Create custom login form

- Spring Security defines default names for login form fields
  - User name field: **`username`**
  - Password field: **`password`**

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

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

Step 3: Create custom login form

- Spring Security defines default names for login form fields
  - User name field: **`username`**
  - Password field: **`password`**

Spring Security Filters will read the form data and authenticate the user

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

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

Step 3: Create custom login form

Pull It All Together

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

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

Step 3: Create custom login form

Pull It All Together

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

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









</form:form>
…
```

Step 3: Create custom login form

Pull It All Together

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

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

</form:form>
…
```

```
http.authorizeRequests()
      .anyRequest().authenticated()
    .and()
    .formLogin()
      .loginPage("/showMyLoginPage")
      .loginProcessingUrl("/authenticateTheUser")
      .permitAll();
```

Step 3: Create custom login form

Pull It All Together

File: WEB-INF/view/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

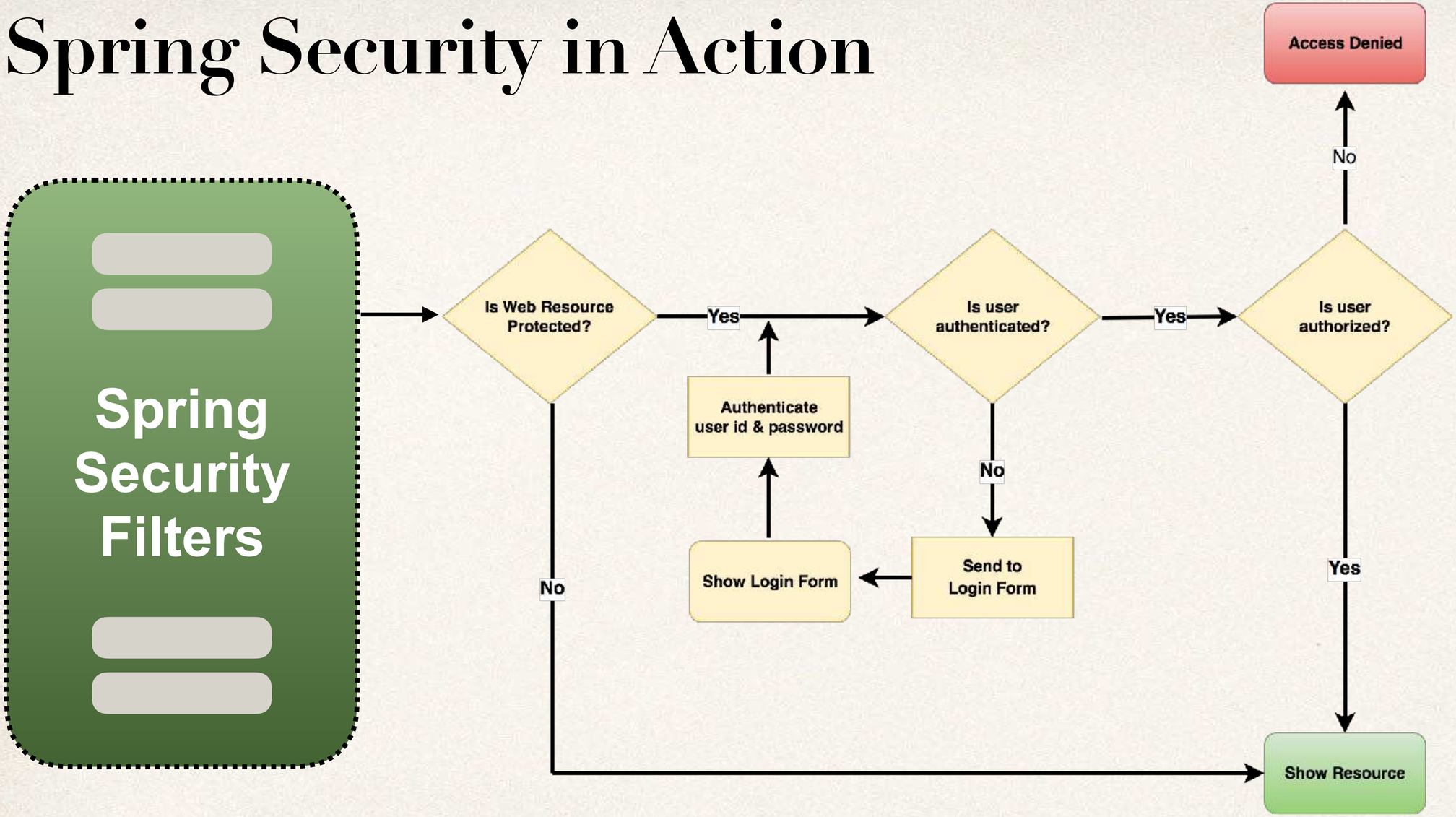
Pull It All Together

File: WEB-INF/view/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>
…
```

Spring Security in Action

Spring Security in Action



More Info on Context Path

More Info on Context Path

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

More Info on Context Path

What is this???

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

More Info on Context Path

More Info on Context Path

What is "Context Root"

The root path for your web application

Context Root: my-ecommerce-app

http://localhost:8080/my-ecommerce-app

More Info on Context Path

Context Path
is same thing as
Context Root

What is "Context Root"

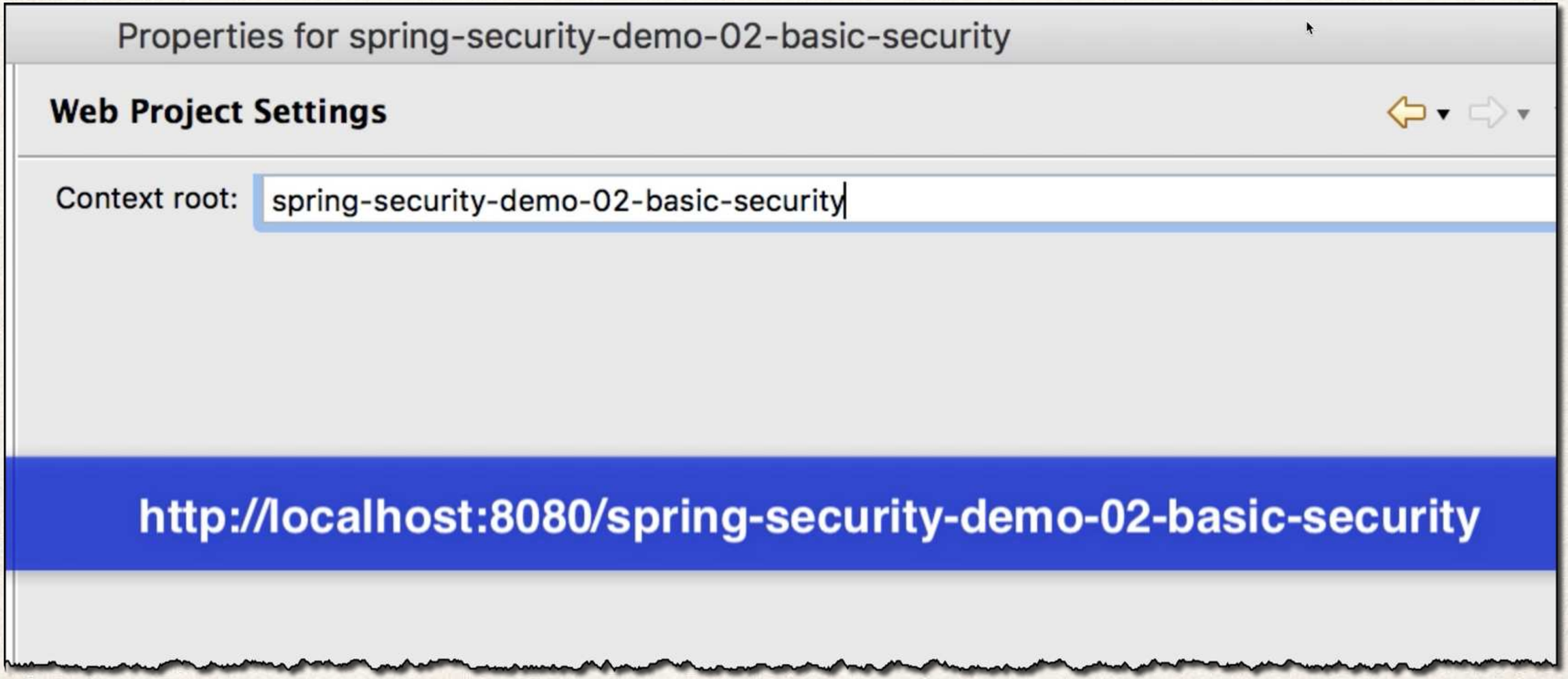
The root path for your web application

Context Root: my-ecommerce-app

http://localhost:8080/my-ecommerce-app

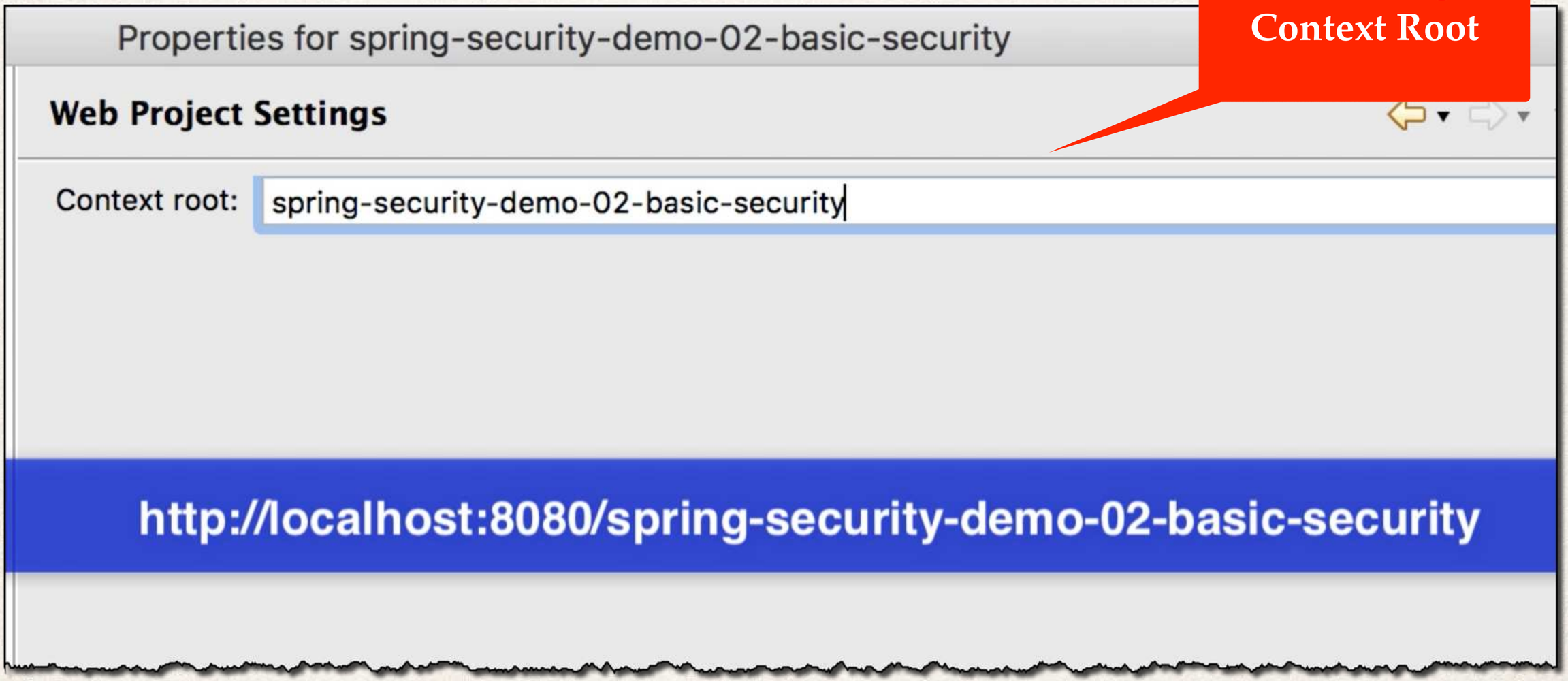
More Info on Context Path

More Info on Context Path



More Info on Context Path

Context Path
is same thing as
Context Root



More Info on Context Path

More Info on Context Path

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

More Info on Context Path

Gives us access to context path dynamically

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

Why use Context Path?

Best Practice

Why use Context Path?

Best Practice

- Allows us to dynamically reference context path of application

Why use Context Path?

Best Practice

- Allows us to dynamically reference context path of application

- Helps to keep links relative to application context path

Why use Context Path?

Best Practice

- Allows us to dynamically reference context path of application
- Helps to keep links relative to application context path
- If you change context path of app, then links will still work

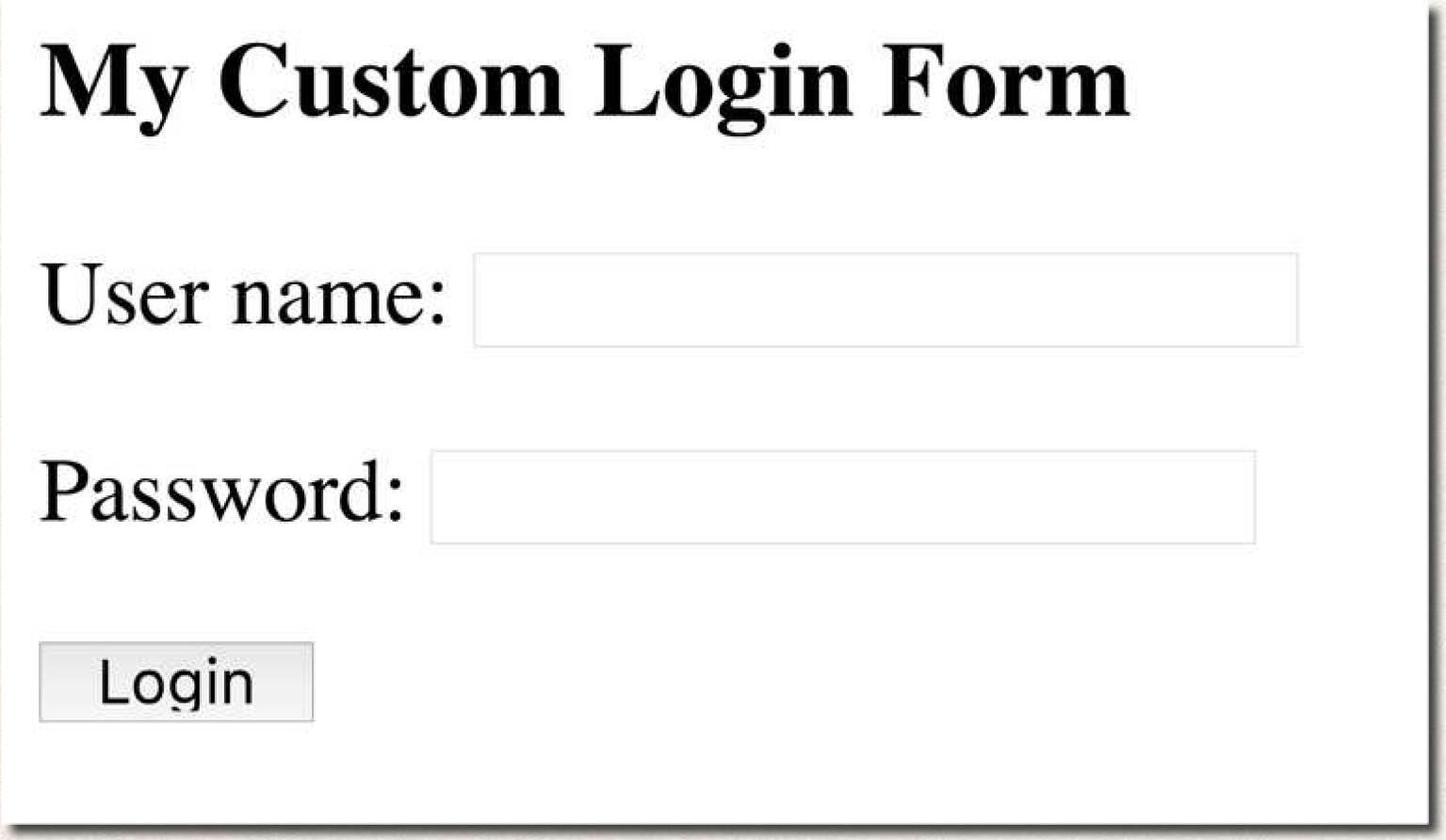
Why use Context Path?

Best Practice

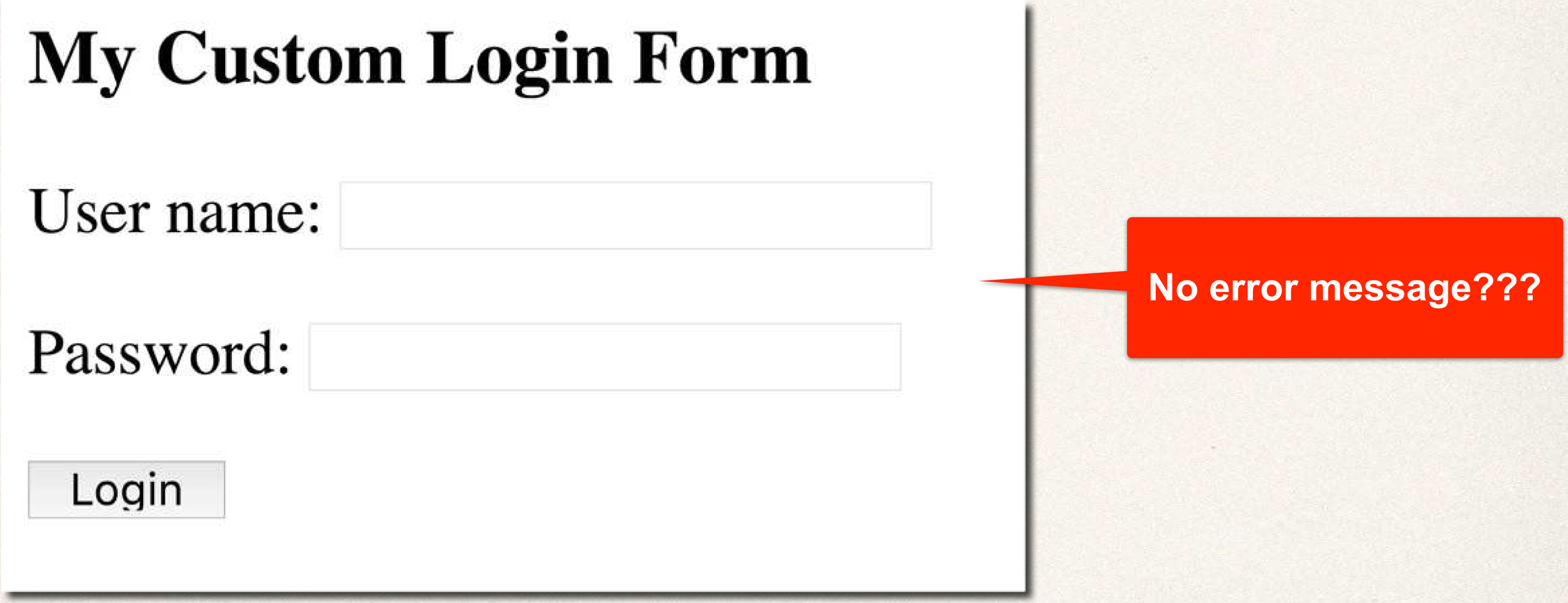
- Allows us to dynamically reference context path of application
- Helps to keep links relative to application context path
- If you change context path of app, then links will still work
- Much better than hard-coding context path …

Spring Security - Show Login Error

No error message for failed login - What???



No error message for failed login - What???



We need an error message … ASAP!!!

My Custom Login Form

Sorry! You entered invalid username/password.

User name:

Password:

Login

We need an error message … ASAP!!!

My Custom Login Form

Sorry! You entered invalid username/password.

User name:

Password:

Login

We'll add this support

Failed Login

Failed Login

- When login fails, by default Spring Security will …

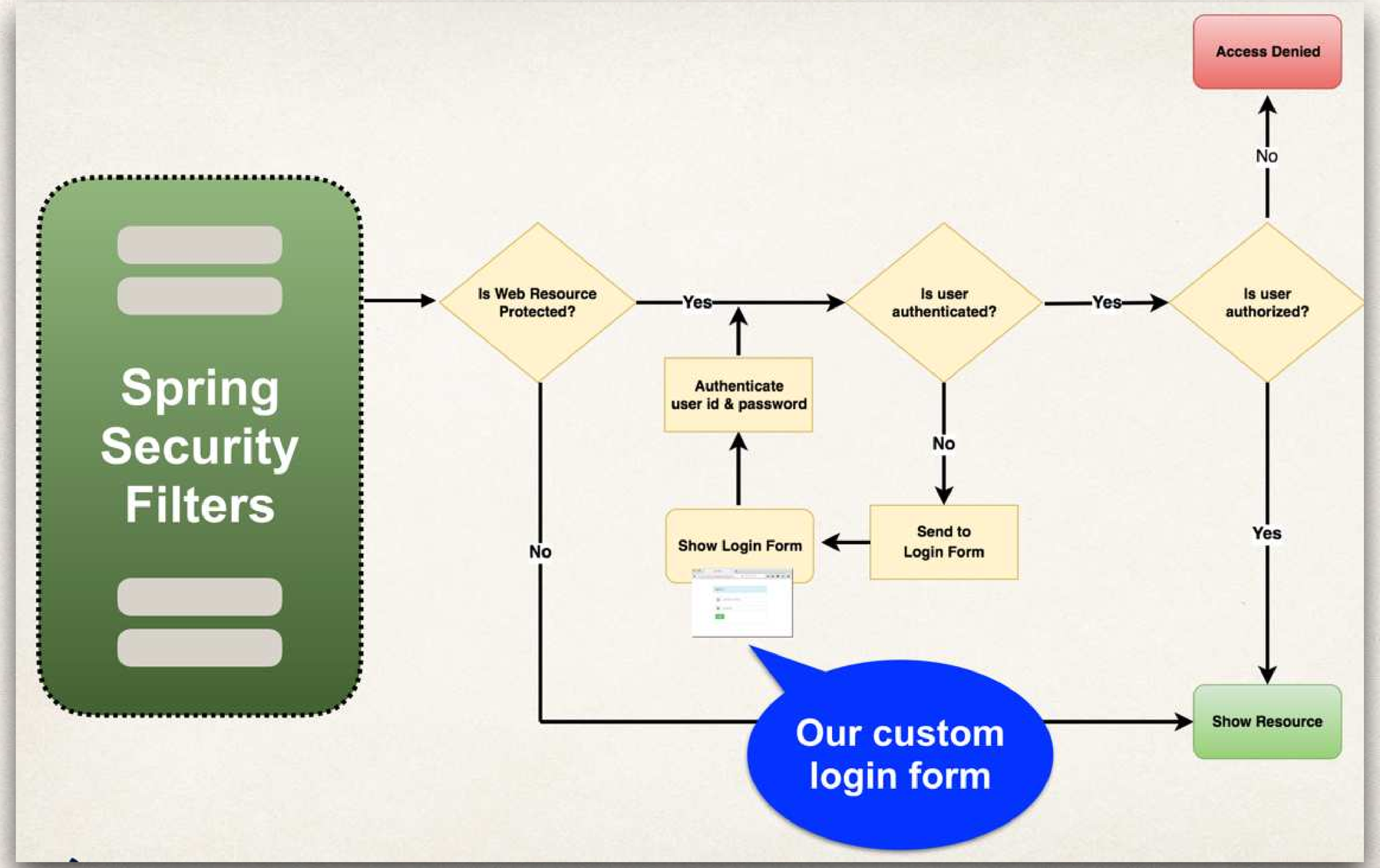
Failed Login

- When login fails, by default Spring Security will …
- Send user back to your login page

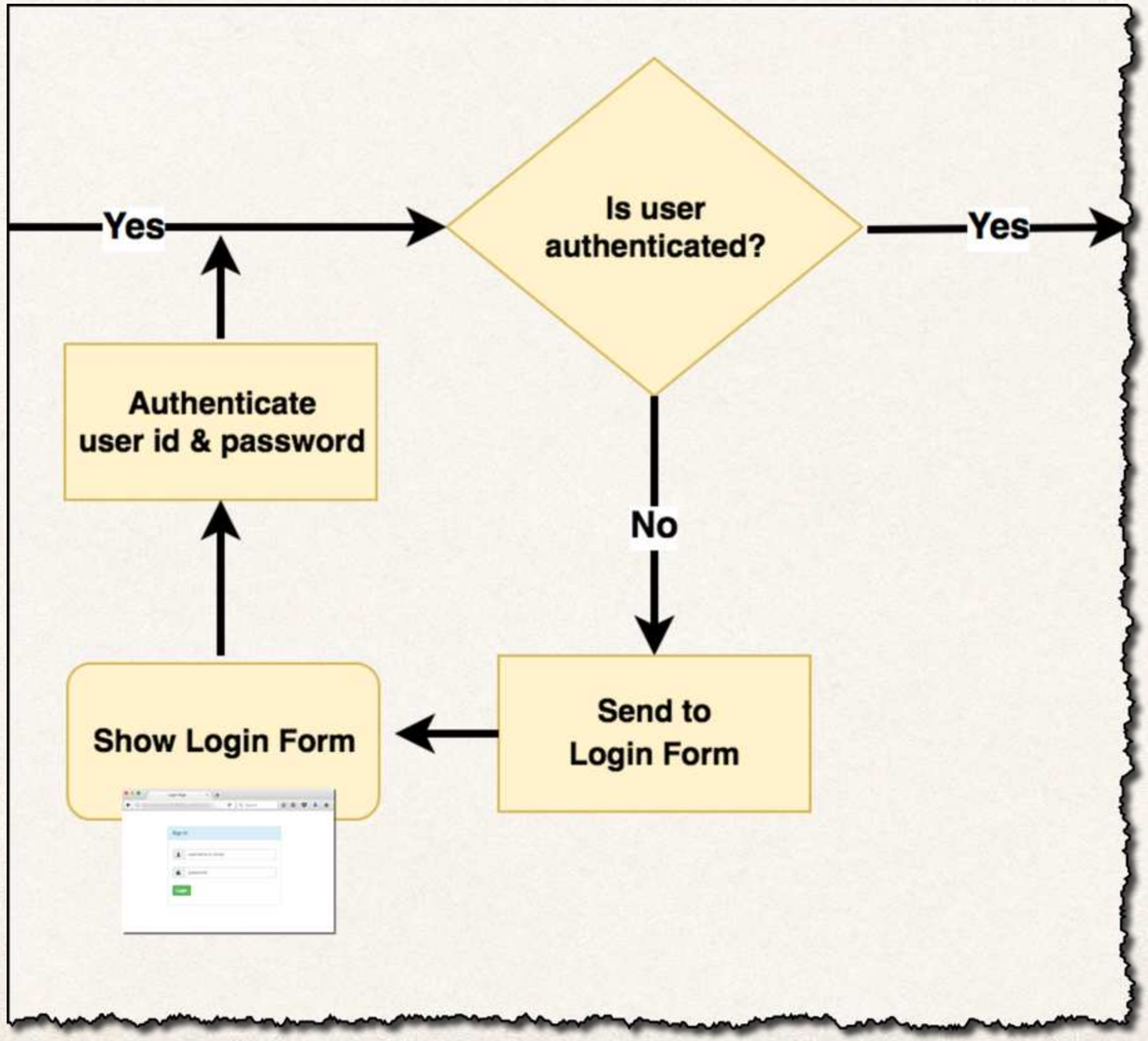
Failed Login

- When login fails, by default Spring Security will …
- Send user back to your login page
- Append an error parameter: **`?error`**

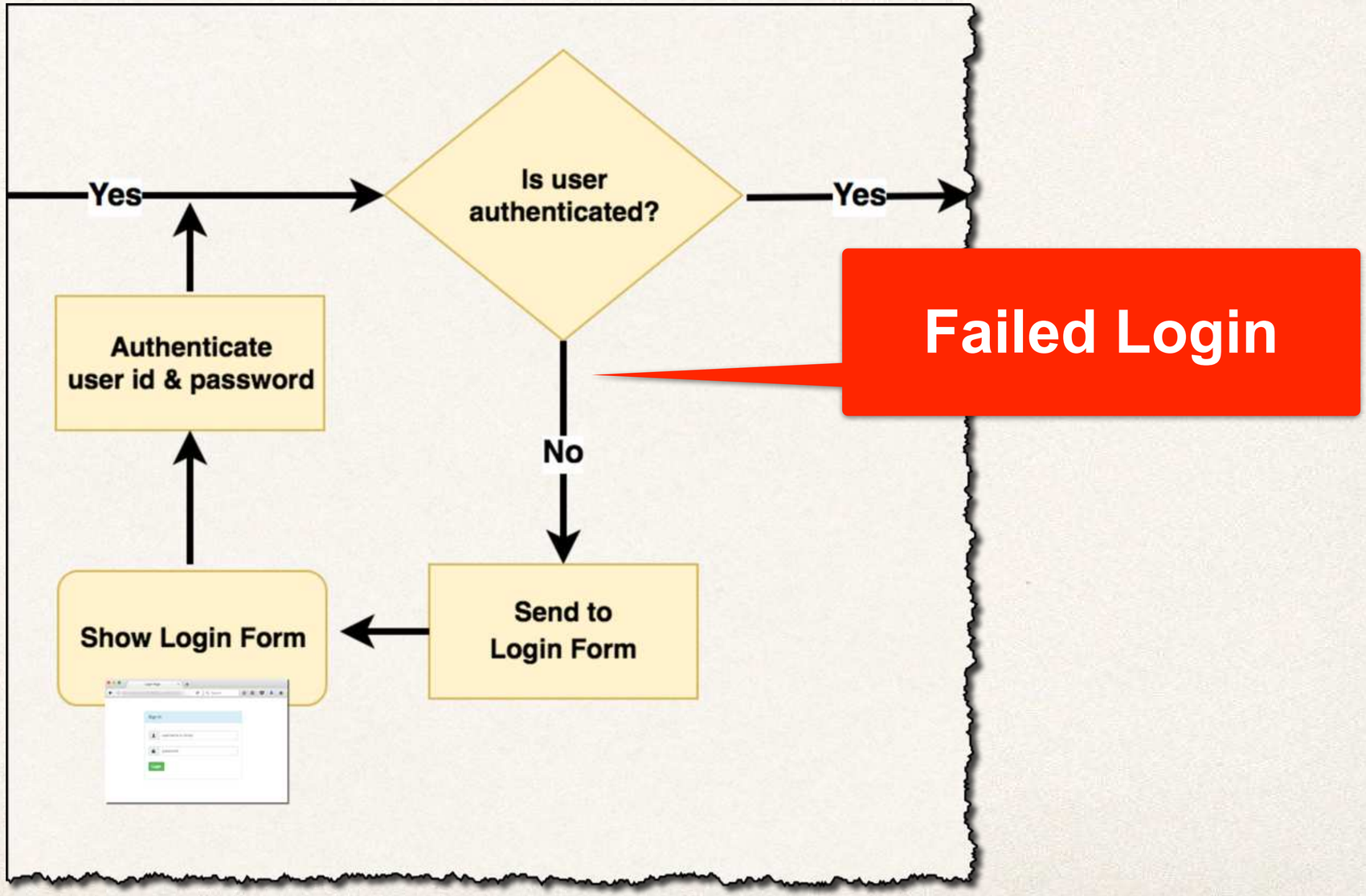
Failed Login



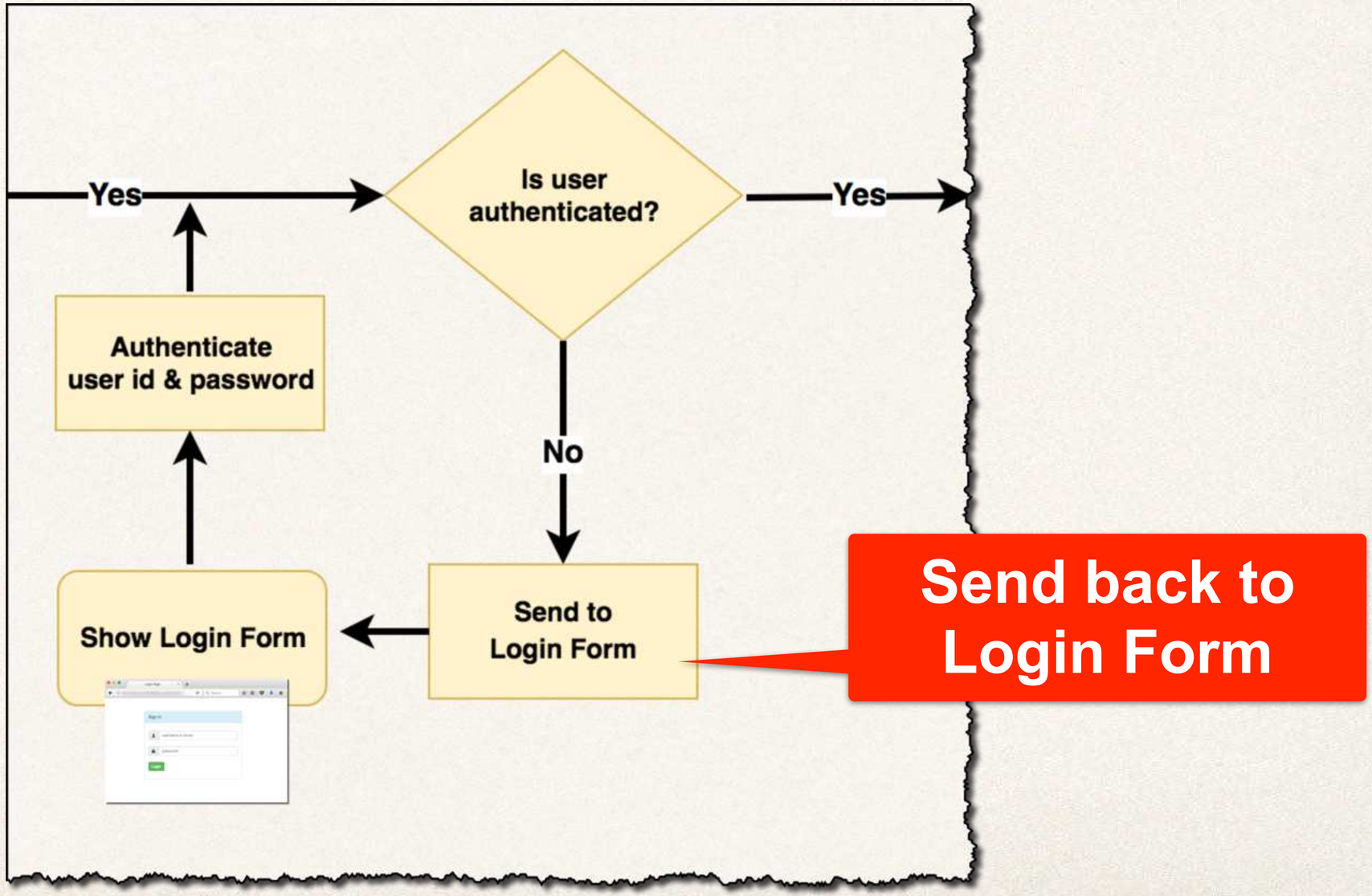
Failed Login



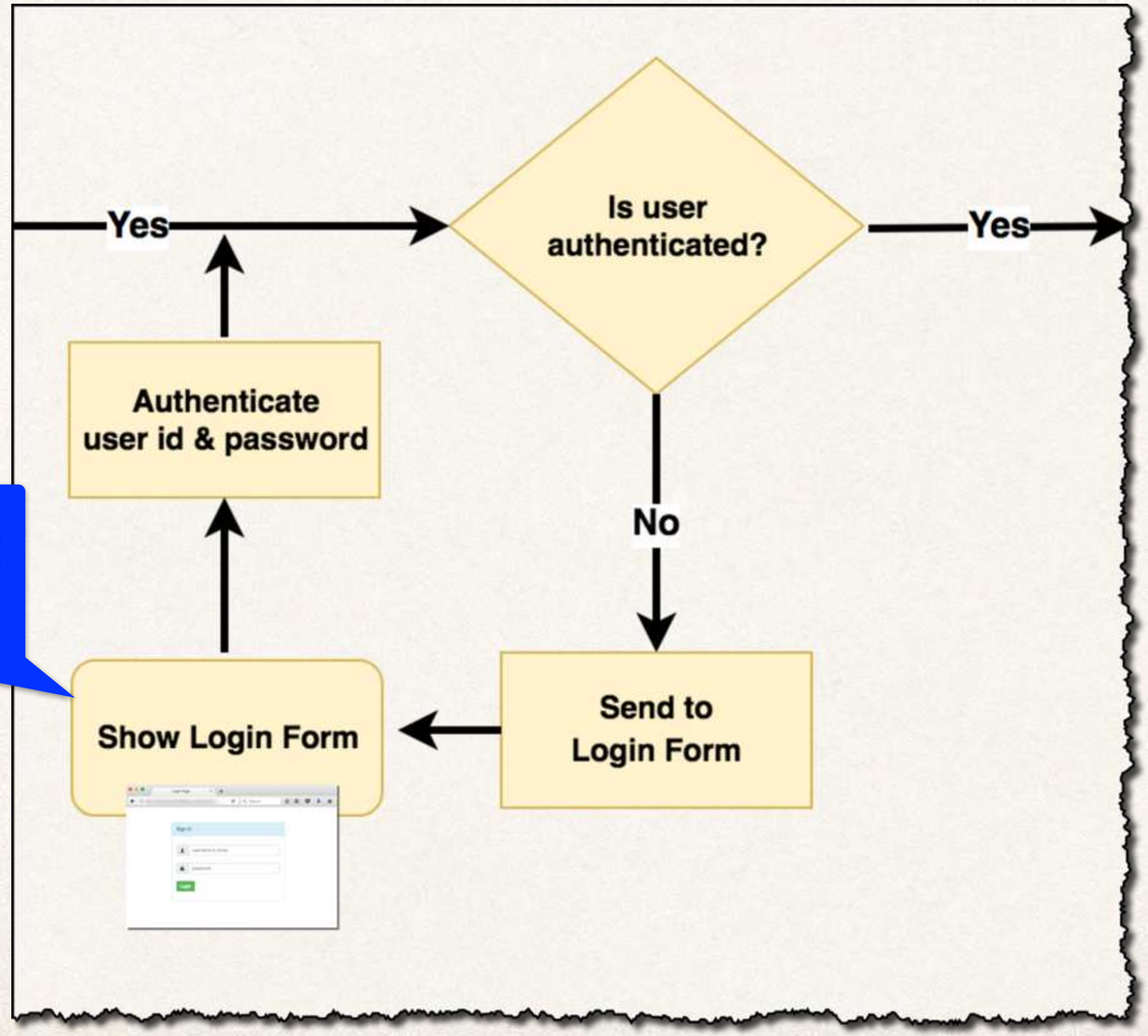
Failed Login



Failed Login



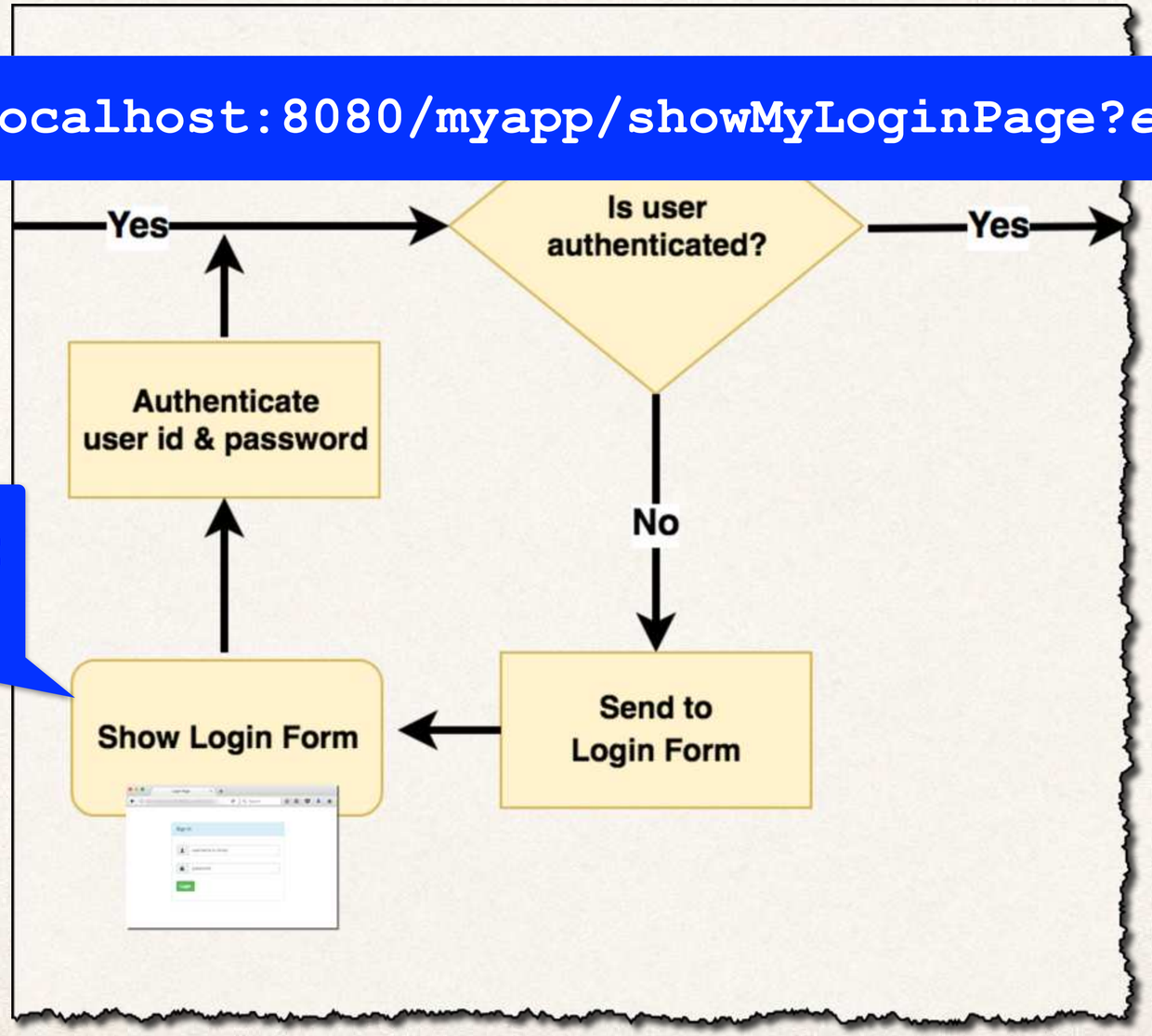
Failed Login



Append param to URL:
error

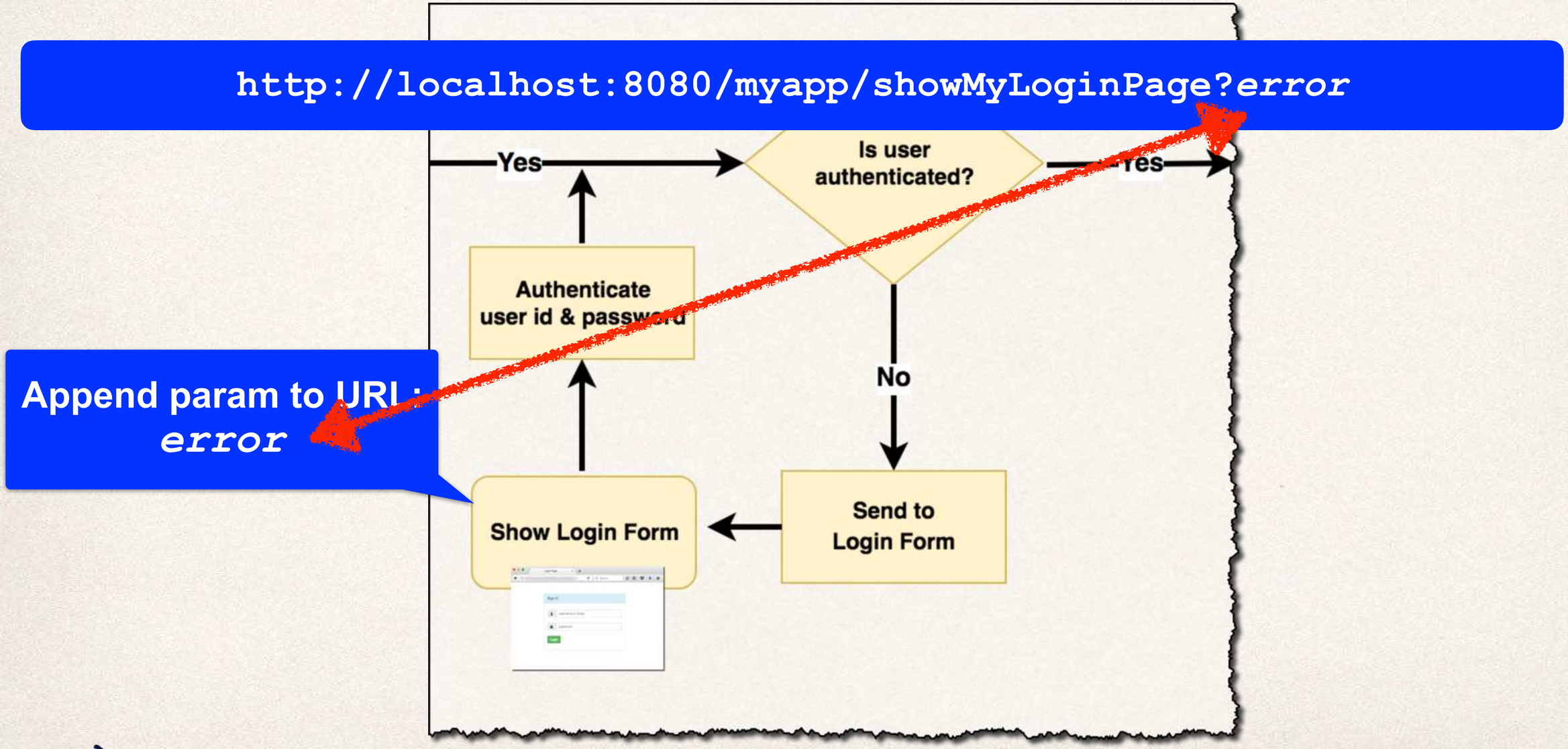
Failed Login

`http://localhost:8080/myapp/showMyLoginPage?error`



Append param to URL: *error*

Failed Login



Development Process

Step-By-Step

Development Process

Step-By-Step

1. Modify custom login form

Development Process

Step-By-Step

1. Modify custom login form
   1. Check the **error** parameter

Development Process

Step-By-Step

1. Modify custom login form
    1. Check the **error** parameter
    2. If **error** parameter exists, show an error message

Development Process

Step-By-Step

1. Modify custom login form

1. Check the **error** parameter
2. If **error** parameter exists, show an error message

JSTL can help with this :-)

Step 1: Modify form - check for error

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

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

...
```

Step 1: Modify form - check for error

Use JSTL

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

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

...
```

Step 1: Modify form - check for error

Use JSTL

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

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



</form:form>
…
```

Step 1: Modify form - check for error

Use JSTL

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

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

  <c:if test="${param.error != null}”>

    <i>Sorry! You entered invalid username/password.</i>

  </c:if>

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


</form:form>
…
```

Step 1: Modify form - check for error

Use JSTL

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

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

  <c:if test="${param.error != null}”>

    <i>Sorry! You entered invalid username/password.</i>

  </c:if>

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


</form:form>
…
```

If error param then show message

Step 1: Modify form - check for error

Use JSTL

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

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

  <c:if test="${param.error != null}”>

    <i>Sorry! You entered invalid username/password.</i>

  </c:if>

    User name: <input type="text" name="username" />
    Password: <input type="password" name="password" />
```

If error param then show message

http://localhost:8080/myapp/showMyLoginPage?*error*

…

Step 1: Modify form - check for error

Use JSTL

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

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

  <c:if test="${param.error != null}”>

    <i>Sorry! You entered invalid username/password.</i>

  </c:if>

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

…
```

If error param then show message

http://localhost:8080/myapp/showMyLoginPage?*error*

Step 1: Modify form - check for error

Use JSTL

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

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

  <c:if test="${param.error != null}”>

    <i>Sorry! You entered invalid username/password.</i>

  </c:if>

    User name: <input type="text" name="username" />
    Password: <input type="password" name="password" />
```

If error param then show message

http://localhost:8080/myapp/showMyLoginPage?*error*

…

Spring Security - Bootstrap Login Page

Let's Make our Login Form Beautiful

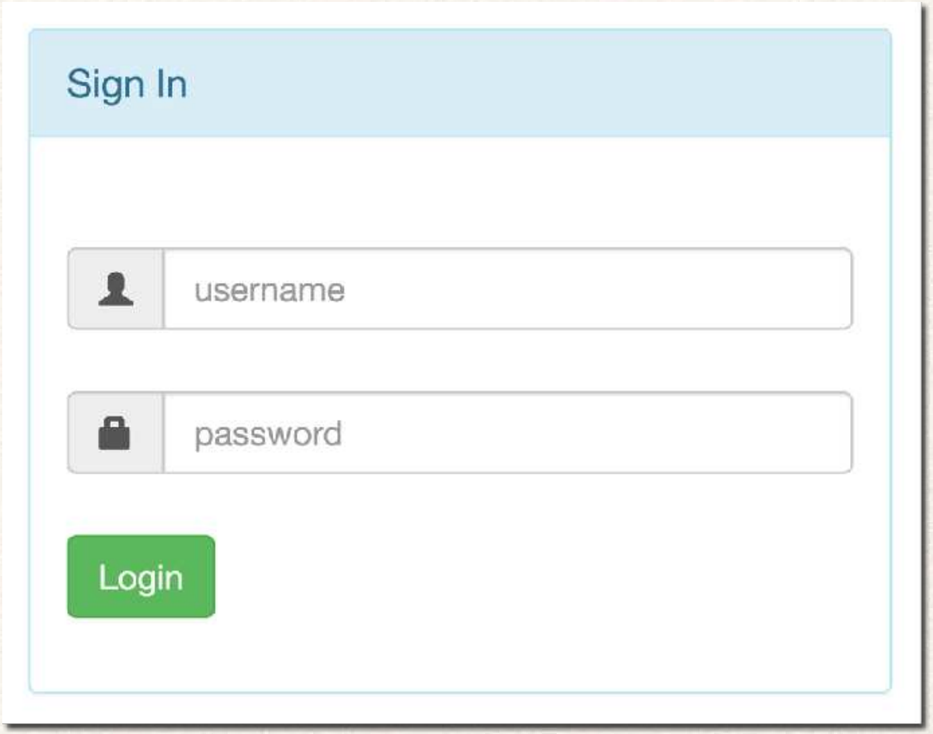
Let's Make our Login Form Beautiful

Before

Let's Make our Login Form Beautiful

Before

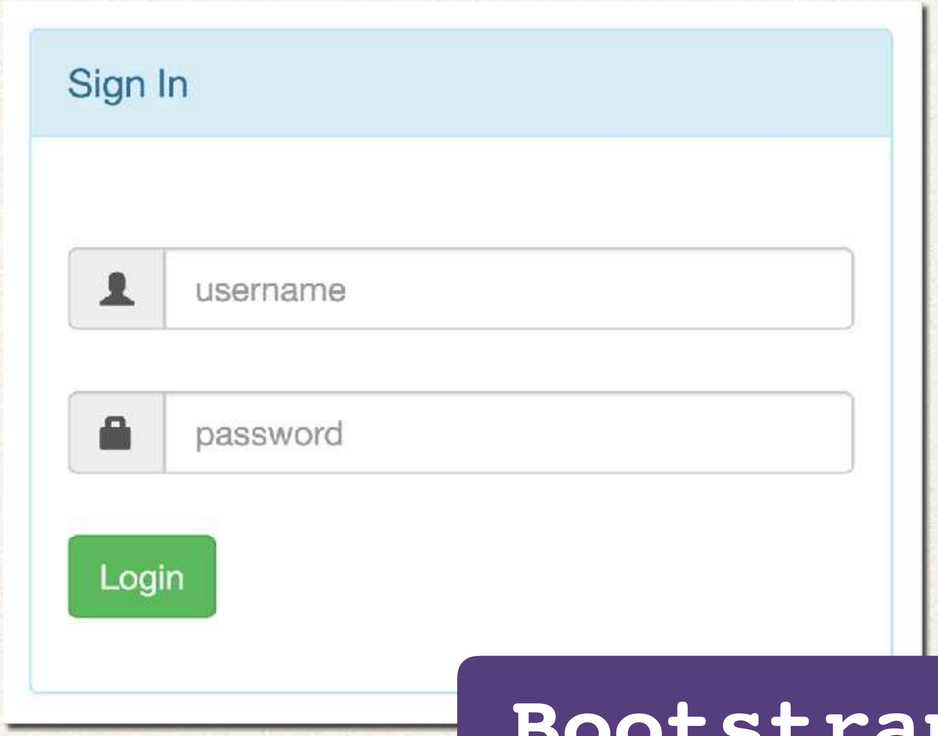
After



Let's Make our Login Form Beautiful

Before

After



Bootstrap

Bootstrap

Bootstrap

- Bootstrap is a web framework that includes CSS styles and JavaScript

Bootstrap

- Bootstrap is a web framework that includes CSS styles and JavaScript
- Focused on front-end UI

Bootstrap

- Bootstrap is a web framework that includes CSS styles and JavaScript
- Focused on front-end UI
- For our project, you do not need any experience with Bootstrap :-)

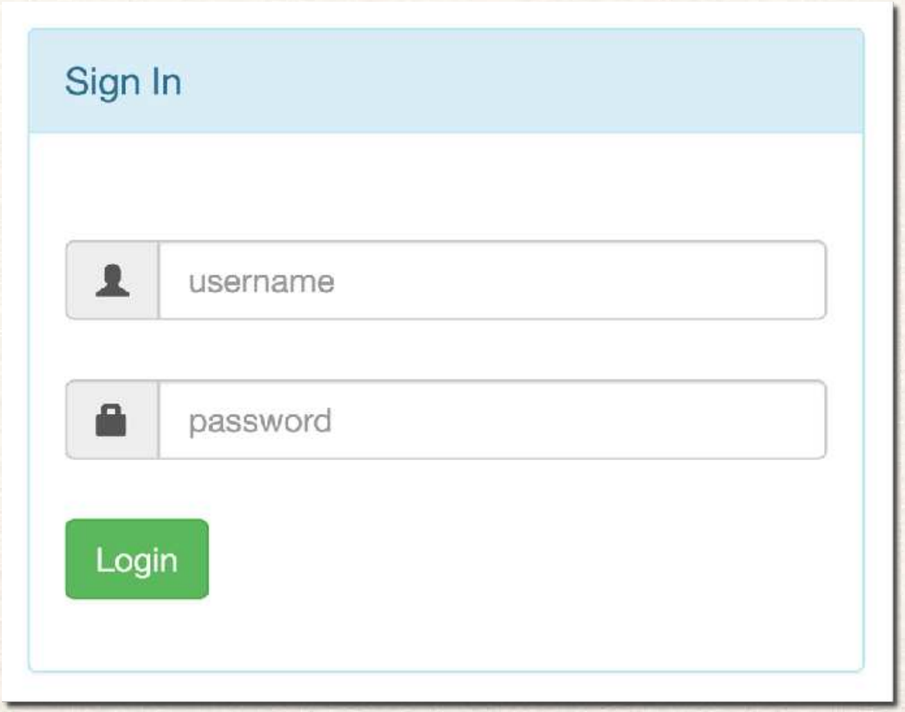
Our Game Plan

Our Game Plan

- My friend, Alan, has helped us out

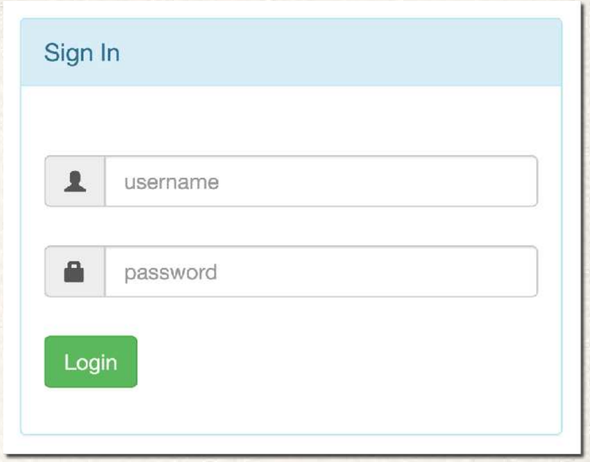
Our Game Plan

- My friend, Alan, has helped us out
  - He created an HTML login form using Bootstrap



Our Game Plan

- My friend, Alan, has helped us out
  - He created an HTML login form using Bootstrap



- We'll modify the form to use Spring Security for our project

Development Process

Step-By-Step

Development Process

Step-By-Step

1. Modify form and add support for Spring MVC form tags

Development Process

Step-By-Step

1. Modify form and add support for Spring MVC form tags

2. Update form to point to our login processing URL

Development Process

Step-By-Step

1. Modify form and add support for Spring MVC form tags

2. Update form to point to our login processing URL

3. Verify form fields for username and password

Development Process

Step-By-Step

1. Modify form and add support for Spring MVC form tags

2. Update form to point to our login processing URL

3. Verify form fields for username and password

4. Change our controller to use our new Bootstrap login form

Bootstrap Tutorials and Docs

Bootstrap Tutorials and Docs

- There are tons of free tutorials online: “bootstrap tutorial”

Bootstrap Tutorials and Docs

- There are tons of free tutorials online: “bootstrap tutorial”

www.w3schools.com/bootstrap

Bootstrap Tutorials and Docs

- There are tons of free tutorials online: “bootstrap tutorial”

www.w3schools.com/bootstrap

- Bootstrap Official Documentation

Bootstrap Tutorials and Docs

- There are tons of free tutorials online: “bootstrap tutorial”

www.w3schools.com/bootstrap

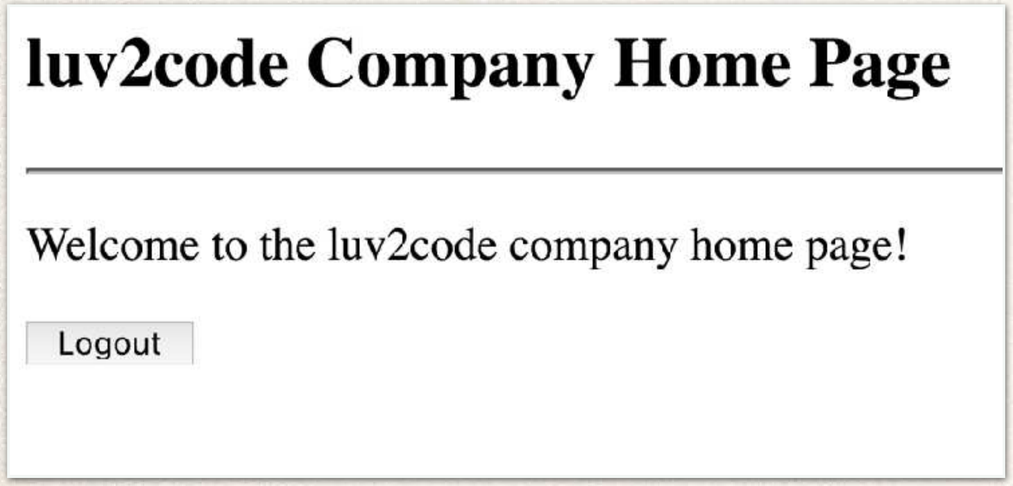
- Bootstrap Official Documentation

www.getbootstrap.com/docs

Spring Security - Logout

Logging Out

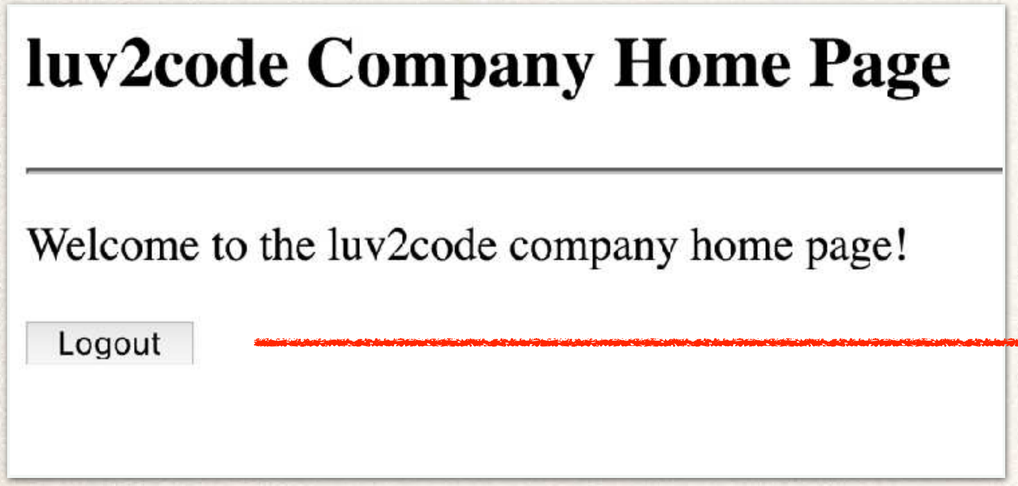
Logging Out


luv2code Company Home Page
Welcome to the luv2code company home page!
Logout

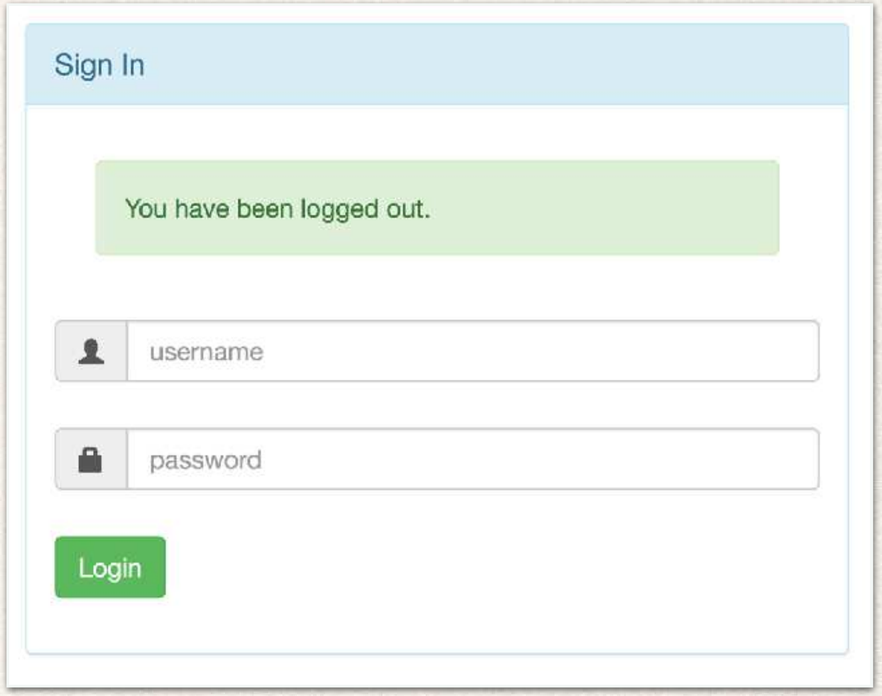
Logging Out

luv2code Company Home Page
Welcome to the luv2code company home page!
Logout
Clear user's session

Logging Out



Clear user's session



Development Process

Step-By-Step

Development Process

Step-By-Step

1. Add logout support to Spring Security Configuration

Development Process

Step-By-Step

1. Add logout support to Spring Security Configuration

2. Add logout button to JSP page

Development Process

Step-By-Step

1. Add logout support to Spring Security Configuration

2. Add logout button to JSP page

3. Update login form to display “logged out” message

Step 1: Add Logout support to Spring Security Configuration

```
@Override
protected void configure(HttpSecurity http) throws Exception {

  http.authorizeRequests()
         .anyRequest().authenticated()
       .and()
       .formLogin()
         .loginPage("/showMyLoginPage")
         .loginProcessingUrl("/authenticateTheUser")
         .permitAll()


}
```

Step 1: Add Logout support to Spring Security Configuration

```
@Override
protected void configure(HttpSecurity http) throws Exception {

  http.authorizeRequests()
         .anyRequest().authenticated()
       .and()
       .formLogin()
         .loginPage("/showMyLoginPage")
         .loginProcessingUrl("/authenticateTheUser")
         .permitAll()
       .and()
       .logout().permitAll();
}
```

Step 1: Add Logout support to Spring Security Configuration

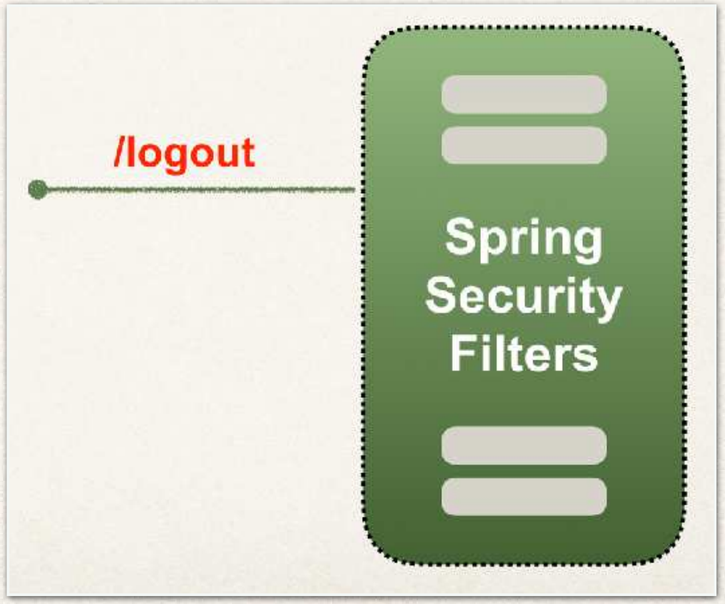
```
@Override
protected void configure(HttpSecurity http) throws Exception {

  http.authorizeRequests()
         .anyRequest().authenticated()
       .and()
       .formLogin()
         .loginPage("/showMyLoginPage")
         .loginProcessingUrl("/authenticateTheUser")
         .permitAll()
       .and()
       .logout().permitAll();
}
```

Add logout support for default URL `/logout`

Step 2: Add logout button

Step 2: Add logout button

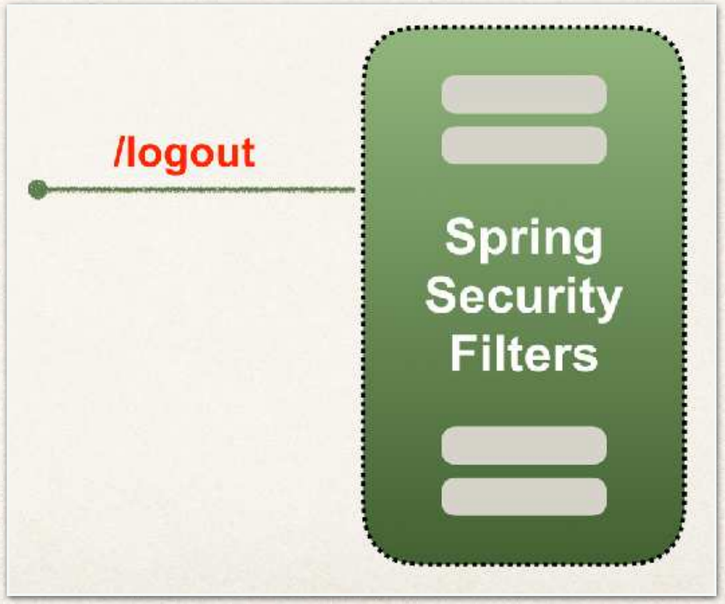
- Send data to default logout URL: **/logout**



Step 2: Add logout button

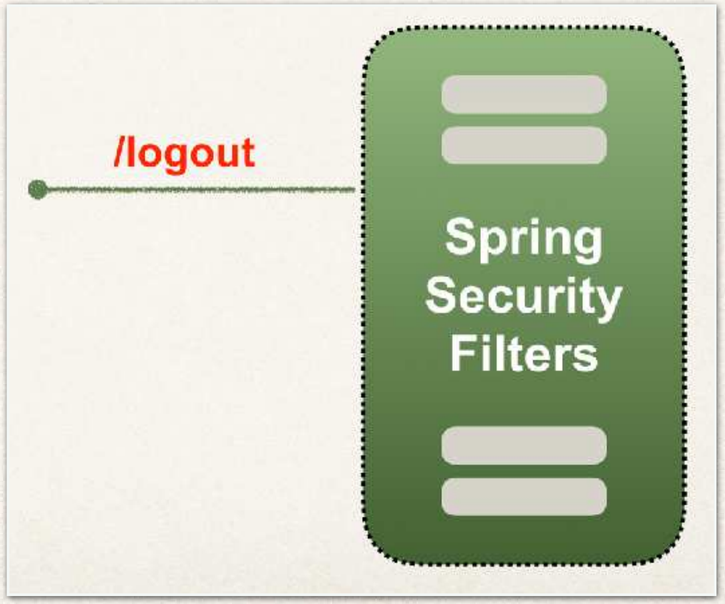
- Send data to default logout URL: **/logout**

- Logout URL will be handled by Spring Security Filters



Step 2: Add logout button

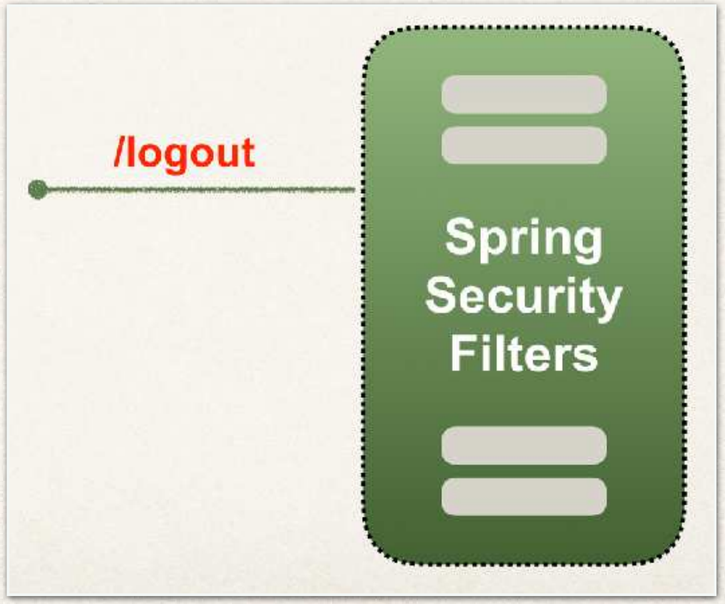
- Send data to default logout URL: **/logout**
- Logout URL will be handled by Spring Security Filters
- You get it for free … no coding required



Step 2: Add logout button

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

This is
Spring Security magic …
LOL



Step 2: Add logout button

Step 2: Add logout button

- Send data to default logout URL: **/logout**

Step 2: Add logout button

- Send data to default logout URL: **/logout**

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

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

</form:form>
```

Step 2: Add logout button

- Send data to default logout URL: **/logout**
  - By default, must use **POST** method

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

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

</form:form>
```

Step 2: Add logout button

- Send data to default logout URL: **/logout**
  - By default, must use **POST** method

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

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

</form:form>
```

GET method is disabled by default

Logout process

Logout process

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

Logout process

- When a logout is processed, by default Spring Security will …
- Invalidate user's HTTP session and remove session cookies, etc

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 your login 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 your login page
- Append a logout parameter: **`?logout`**

Step 3: Update login form to display “logged out” message

Step 3: Update login form to display “logged out” message

1. Update login form

Step 3: Update login form to display “logged out” message

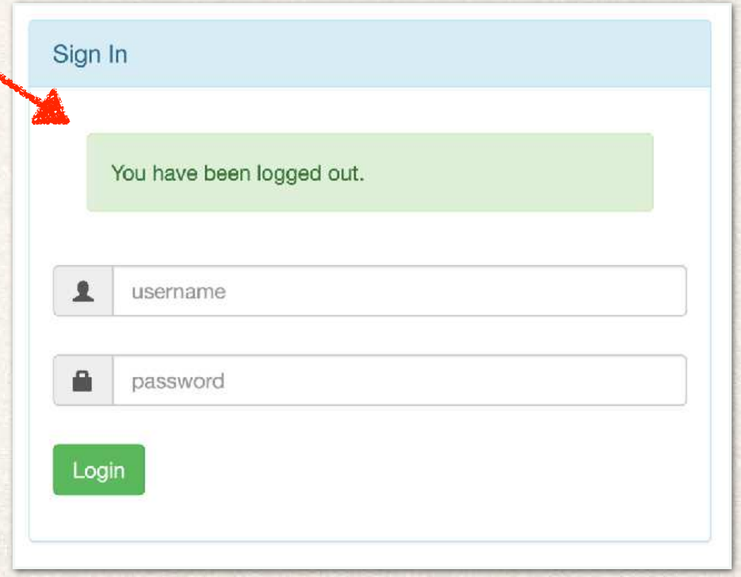
1. Update login form
    1. Check the **logout** parameter

Step 3: Update login form to display “logged out” message

1. Update login form
   1. Check the **logout** parameter
   2. If **logout** parameter exists, show “logged out” message

Step 3: Update login form to display “logged out” message

1. Update login form
   1. Check the **logout** parameter
   2. If **logout** parameter exists, show “logged out” message

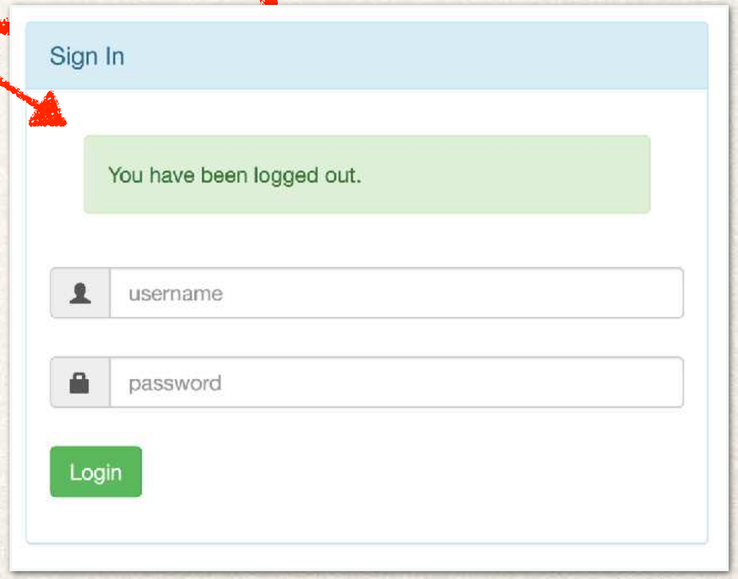


Step 3: Update login form to display "logged out" message

1. Update login form

1. Check the **logout** parameter
2. If **logout** parameter exists, show "logged out" message

JSTL can help with this :-)



Modify Login form - check for "logout"

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

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

...
```

Modify Login form - check for "logout"

Use JSTL

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

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

...
```

Modify Login form - check for "logout"

Use JSTL

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

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



</form:form>
…
```

Modify Login form - check for "logout"

Use JSTL

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

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

  <c:if test="${param.logout != null}">

    <i>You have been logged out.</i>

  </c:if>

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


</form:form>
…
```

Modify Login form - check for "logout"

Use JSTL

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

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

  <c:if test="${param.logout != null}">

    <i>You have been logged out.</i>

  </c:if>

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


</form:form>
…
```

If logout param then show message

Modify Login form - check for "logout"

Use JSTL

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

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

  <c:if test="${param.logout != null}">

    <i>You have been logged out.</i>

  </c:if>

    User name: <input type="text" name="username" />
    Password: <input type="password" name="password" />
```

If logout param then show message

http://localhost:8080/myapp/showMyLoginPage?logout

…

Modify Login form - check for "logout"

Use JSTL

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

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

  <c:if test="${param.logout != null}">

    <i>You have been logged out.</i>

  </c:if>

    User name: <input type="text" name="username" />
    Password: <input type="password" name="password" />
```

If logout param then show message

http://localhost:8080/myapp/showMyLoginPage?logout

```
…
```

Cross Site Request Forgery (CSRF)

Earlier in the course …

Earlier in the course …

Step 3: Create custom login form

Best Practice

- **Best practice** is to use the Spring MVC Form tag `<form:form>`
  - Provides automatic support for security defenses *(more on this later)*

```
<form:form action="…" method="…" >
  …
</form:form>
```

Earlier in the course …

Step 3: Create custom login form

Best Practice

- **Best practice** is to use the Spring MVC Form tag `<form:form>`
  - Provides automatic support for security defenses *(more on this later)*

```
<form:form action="…" method="…" >
  …
</form:form>
```

Spring Security

Spring Security

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

What is 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

CSRF Examples

- You are logged into your banking app

CSRF Examples

- You are logged into your banking app
  - tricked into sending money to another person

CSRF Examples

- You are logged into your banking app
  - tricked into sending money to another person

- You are logged into an e-commerce app

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

CSRF Protection

- To protect against CSRF attacks

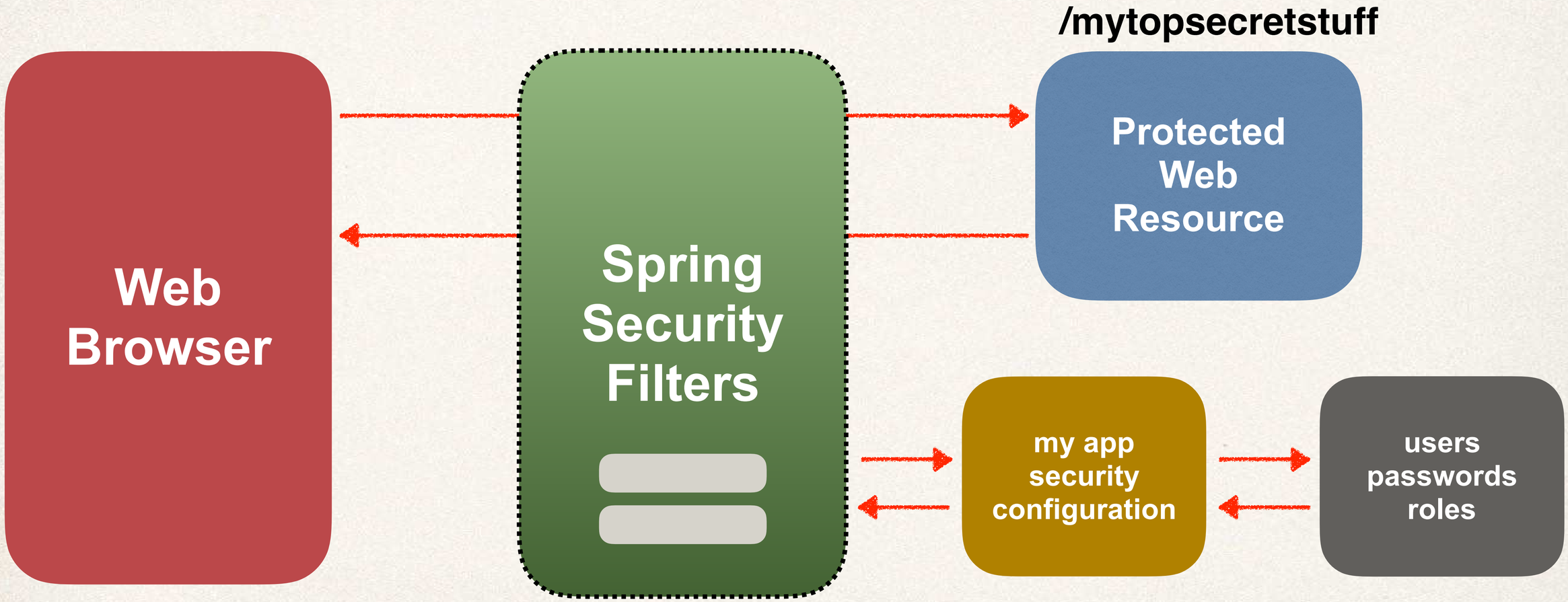
CSRF Protection

- To protect against CSRF attacks
- Embed additional authentication data/token into all HTML forms

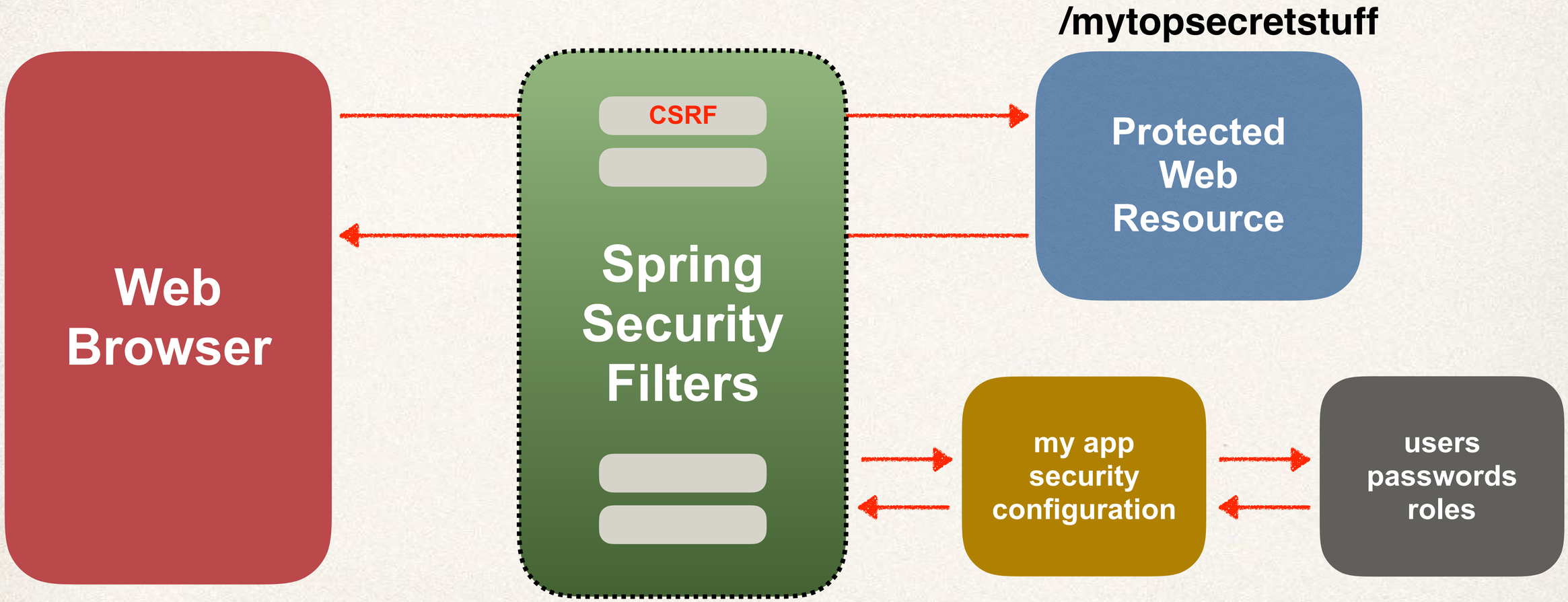
CSRF Protection

- To protect against CSRF attacks
- Embed additional authentication data/token into all HTML forms
- On subsequent requests, web app will verify token before processing

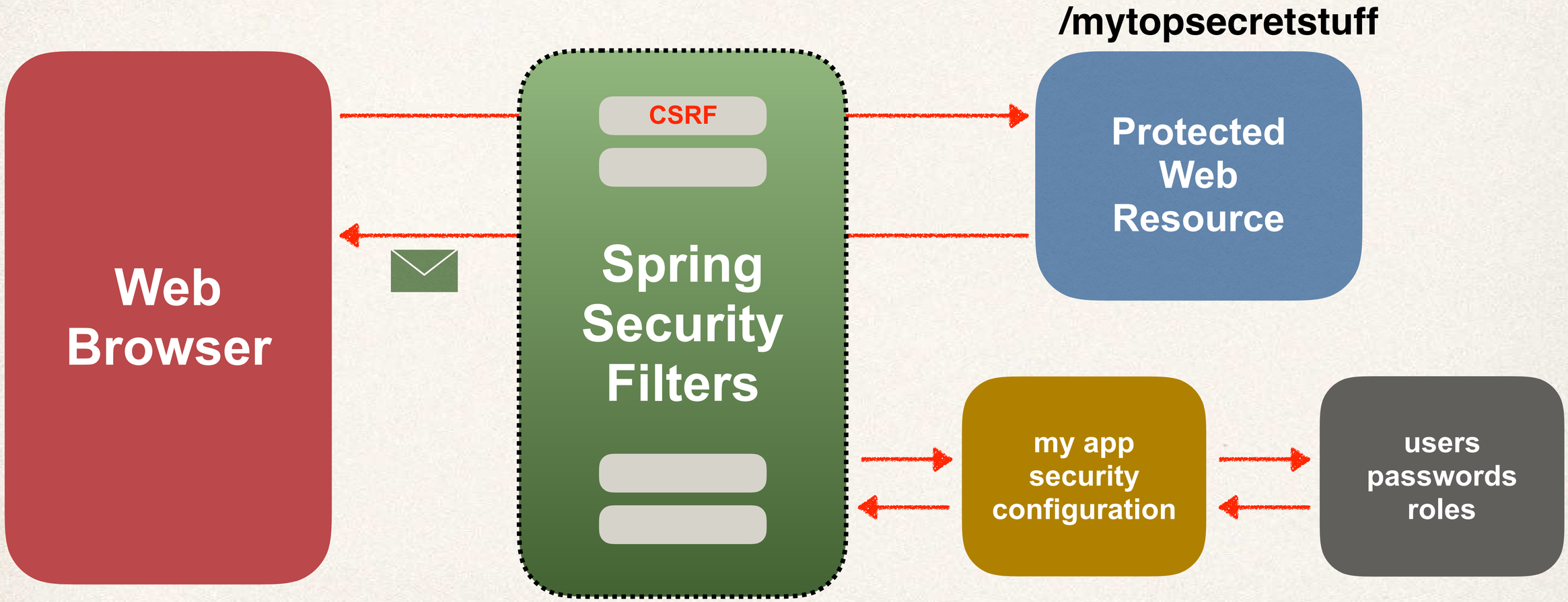
Spring Security Overview



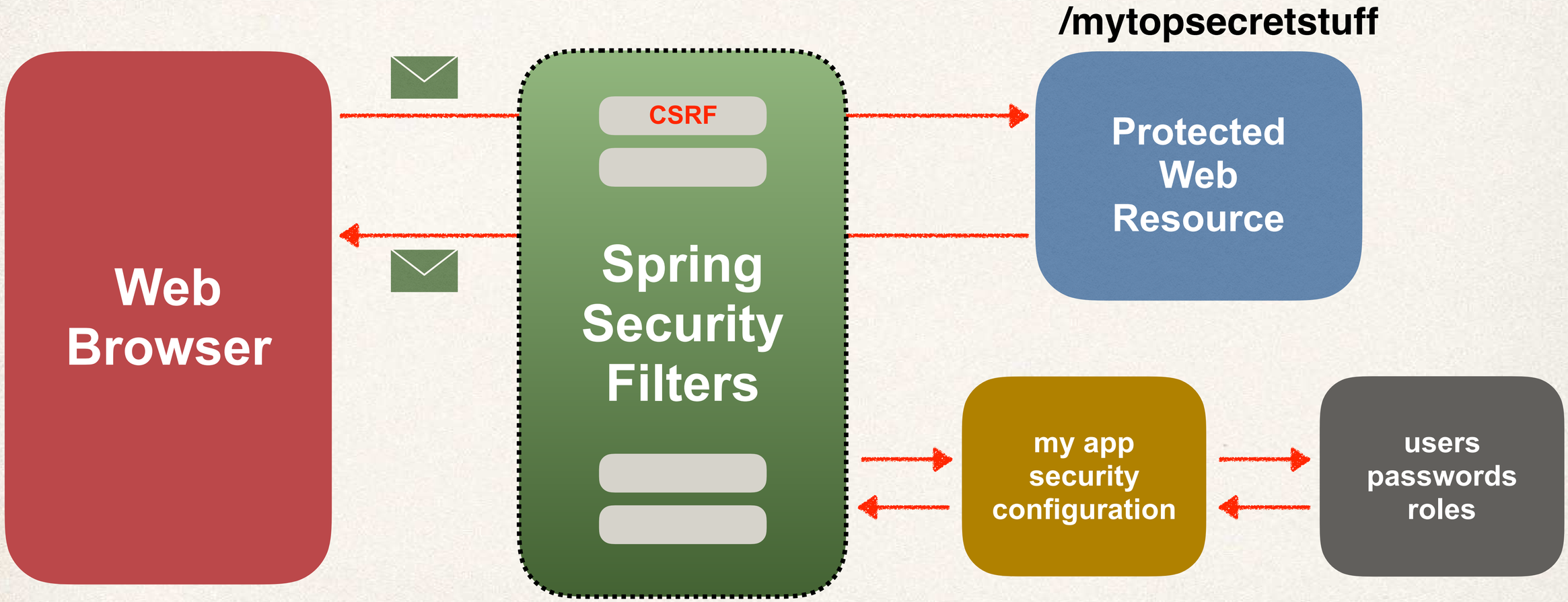
Spring Security Overview



Spring Security Overview



Spring Security Overview



Spring Security's CSRF Protection

Spring Security's CSRF Protection

- CSRF protection is enabled by default in Spring Security

Spring Security's CSRF Protection

- CSRF protection is enabled by default in Spring Security
- Spring Security uses the Synchronizer Token Pattern

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

Spring Security's CSRF Protection

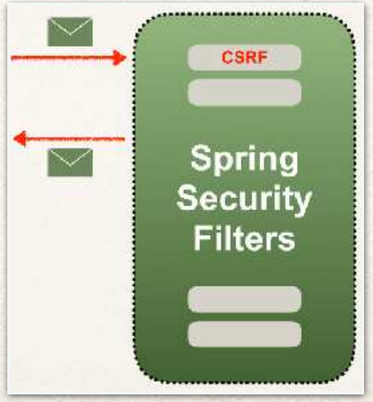
- CSRF protection is enabled by default in Spring Security
- Spring Security uses the Synchronizer Token Pattern
  - Each request includes a session cookie and randomly generated token
- For request processing, Spring Security verifies token before processing

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

- 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



When to use CSRF Protection?

When to use CSRF Protection?

- The Spring Security team recommends

When to use CSRF Protection?

- The Spring Security team recommends
- Use CSRF protection for any normal browser web requests

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

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

Use Spring Security CSRF Protection

- For form submissions use POST instead of GET

Use Spring Security CSRF Protection

- For form submissions use POST instead of GET
- Include CSRF token in form submission

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

Best Practice

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

Best Practice

- If you don't use `<form:form>`, you must manually add CSRF token

Manually add CSRF token

```
<form action="..." method="POST">

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

</form>
```

What happens if you don't include CSRF token?

What happens if you don't include CSRF token?

Let's Break It!

What happens if you don't include CSRF token?

Let's Break It!

HTTP Status 403 – Forbidden

Type Status Report

Message Invalid CSRF Token 'null' was found on the request parameter '_csrf' or header 'X-CSRF-TOKEN'.

Description The server understood the request but refuses to authorize it.

What happens if you don't include CSRF token?

Let's Break It!

HTTP Status 403 – Forbidden

Type Status Report

Message Invalid CSRF Token 'null' was found on the request parameter '_csrf' or header 'X-CSRF-TOKEN'.

Description The server understood the request but refuses to authorize it.

Token was not included

CSRF Resources

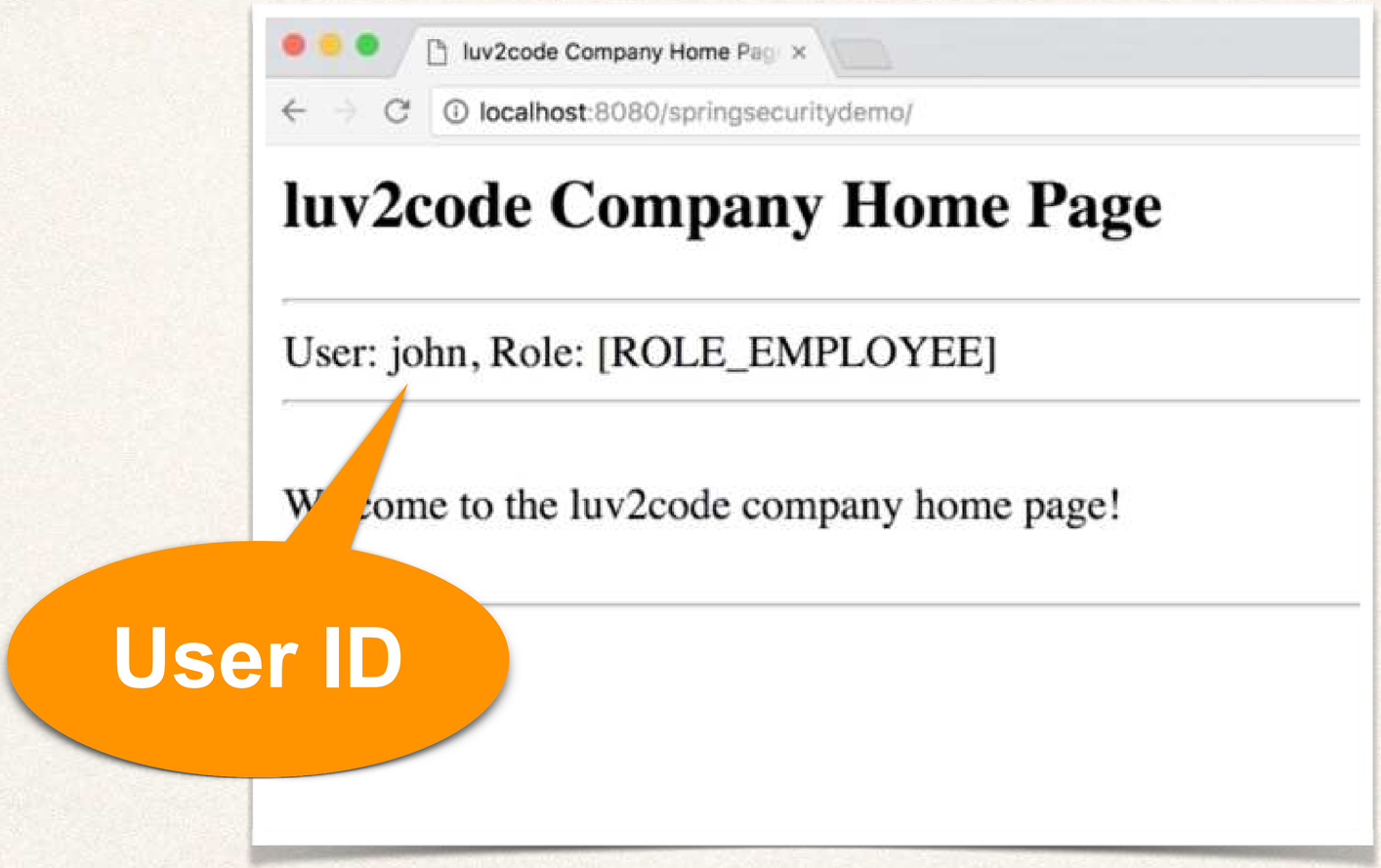
- CSRF Security Reference
  - `https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)`
- Spring Security CSRF Support
  - `https://docs.spring.io/spring-security/site/docs/current/reference/htmlsingle/#csrf`

Display User ID and Roles

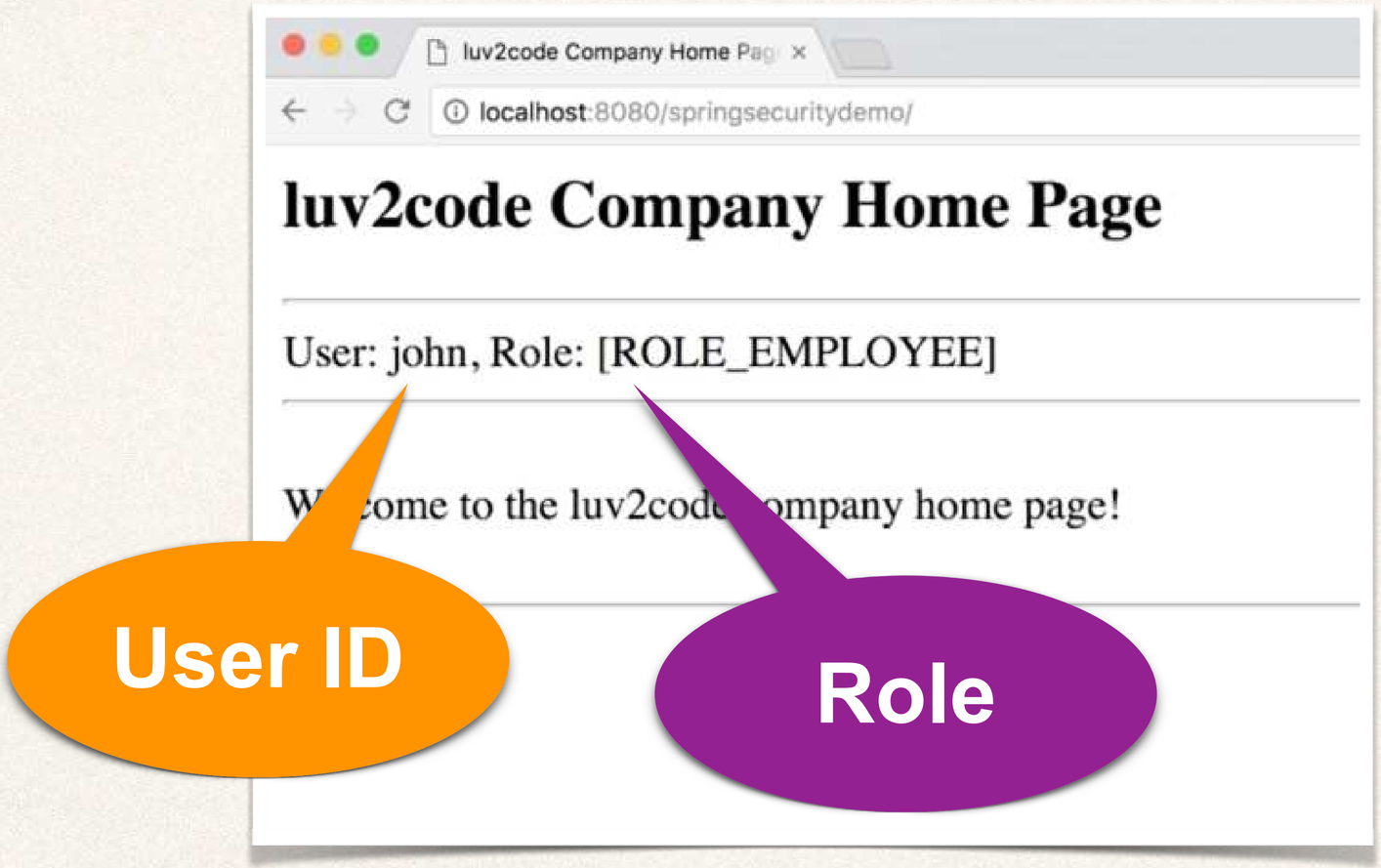
Display User ID and Roles

Display User ID and Roles

Display User ID and Roles



Display User ID and Roles



Spring Security

Spring Security

- Spring Security provides JSP custom tags for accessing user id and roles

Development Process

Step-By-Step

Development Process

Step-By-Step

1. Update POM file for Spring Security JSP Tag Library

Development Process

Step-By-Step

1. Update POM file for Spring Security JSP Tag Library

2. Add Spring Security JSP Tag Library to JSP page

Development Process

Step-By-Step

1. Update POM file for Spring Security JSP Tag Library

2. Add Spring Security JSP Tag Library to JSP page

3. Display User ID

Development Process

Step-By-Step

1. Update POM file for Spring Security JSP Tag Library

2. Add Spring Security JSP Tag Library to JSP page

3. Display User ID

4. Display User Roles

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

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

File: pom.xml

```
<dependency>
    <groupId>org.springframework.security</groupId>
    <artifactId>spring-security-taglibs</artifactId>
    <version>${springsecurity.version}</version>
</dependency>
```

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

File: pom.xml

```
<dependency>
    <groupId>org.springframework.security</groupId>
    <artifactId>spring-security-taglibs</artifactId>
    <version>${springsecurity.version}</version>
</dependency>
```

Spring Security Taglibs

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

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

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

Step 3: Display User ID

Step 3: Display User ID

File: home.jsp

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

User: <security:authentication property="principal.username" />
```

Step 3: Display User ID

File: home.jsp

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

User: <security:authentication property="principal.username" />
```

User ID

Step 4: Display User Roles

Step 4: Display User Roles

File: home.jsp

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

Role(s): <security:authentication property="principal.authorities" />
```

Step 4: Display User Roles

File: home.jsp

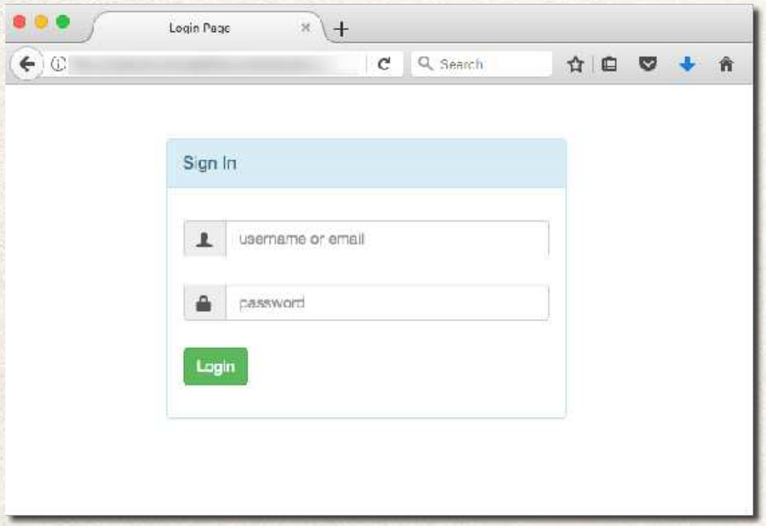
```
<%@ taglib prefix="security"
            uri="http://www.springframework.org/security/tags" %>
…

Role(s): <security:authentication property="principal.authorities" />
```

User Roles

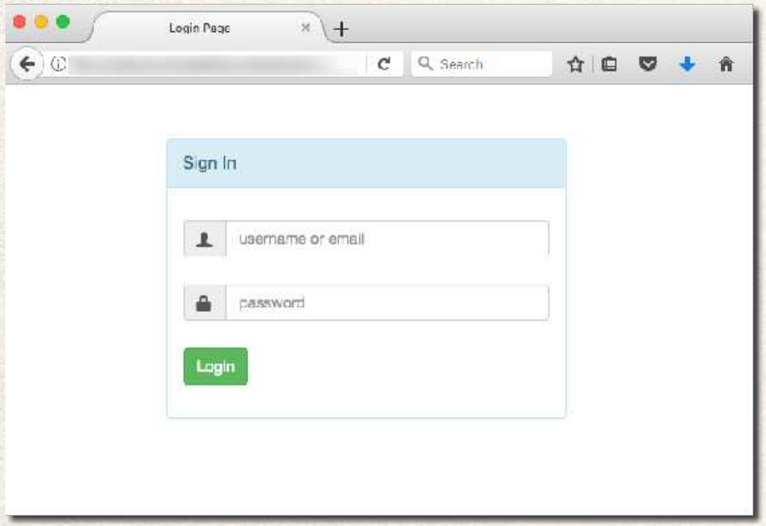
Restrict Access Based on Roles

Our Example

Our Example



Our Example

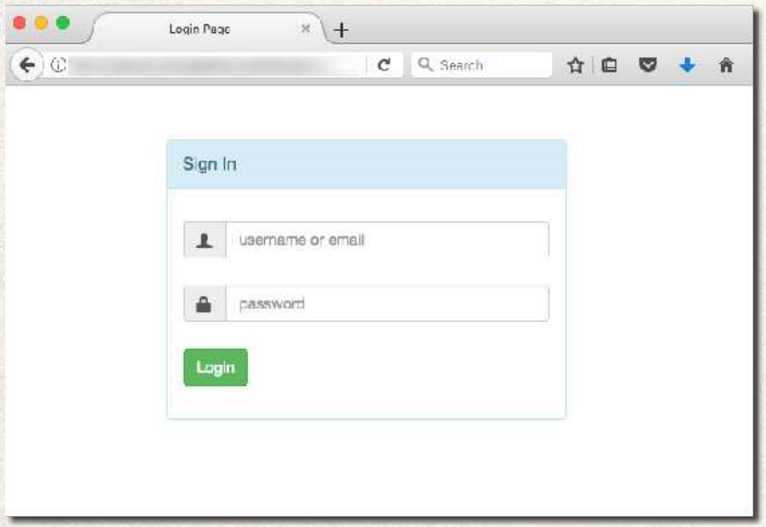


/

Home Page

role: EMPLOYEE

Our Example



/

Home Page

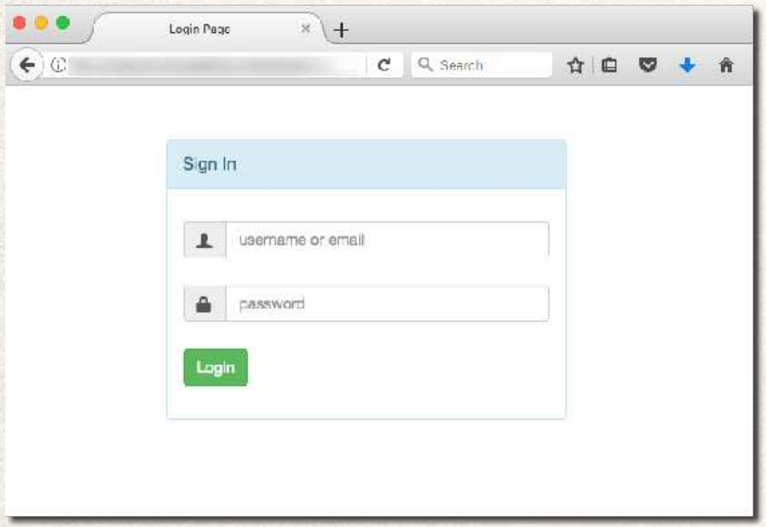
role: EMPLOYEE

/leaders

Leadership Retreat Page

role: MANAGER

Our Example



/

Home Page

role: EMPLOYEE

/leaders

Leadership Retreat Page

role: MANAGER

/systems

Admin Holiday Cruise

role: ADMIN

Development Process

Step-By-Step

Development Process

Step-By-Step

1. Create supporting controller code and view pages

Development Process

Step-By-Step

1. Create supporting controller code and view pages

2. Update user roles

Development Process

Step-By-Step

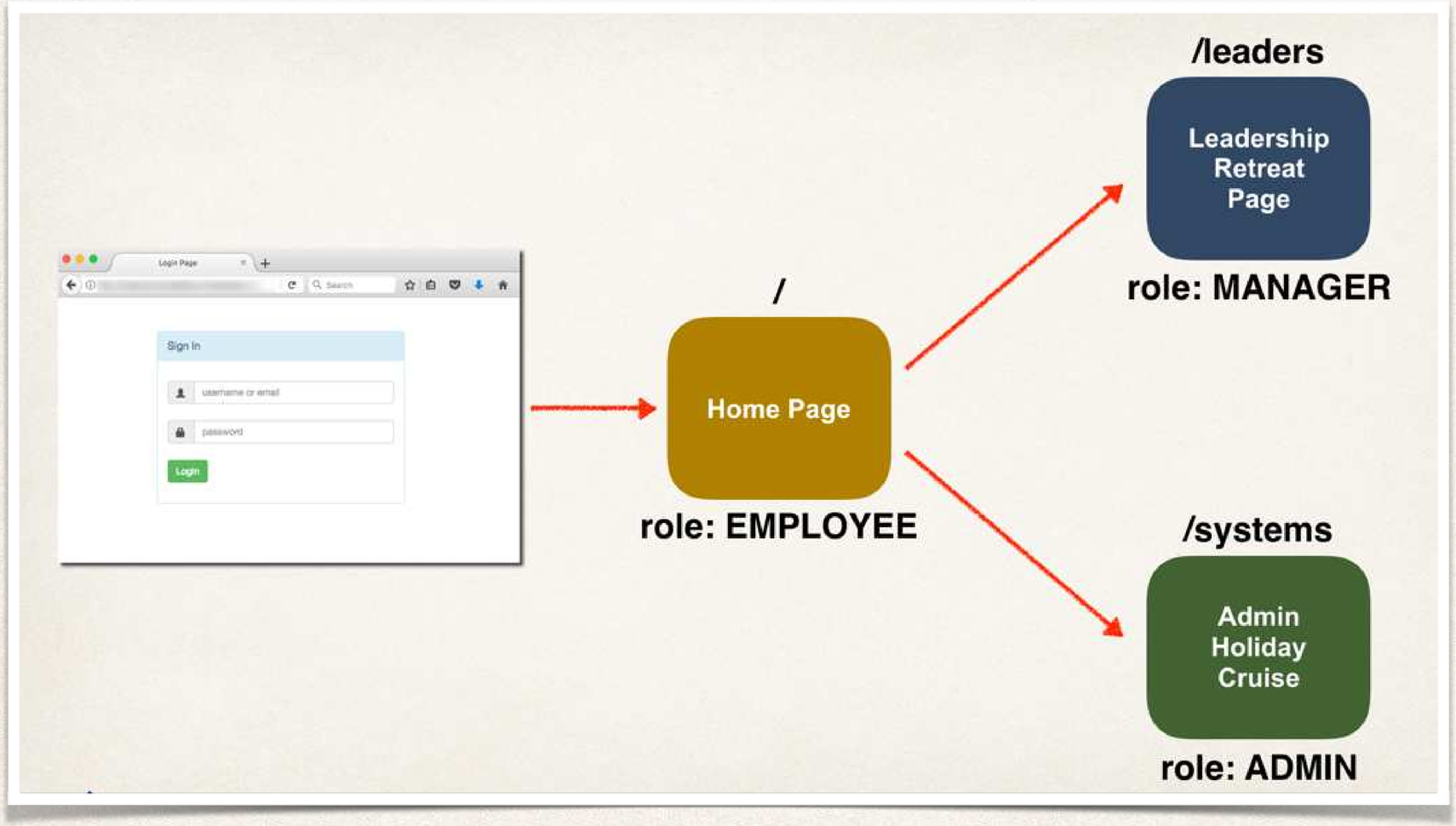
1. Create supporting controller code and view pages

2. Update user roles

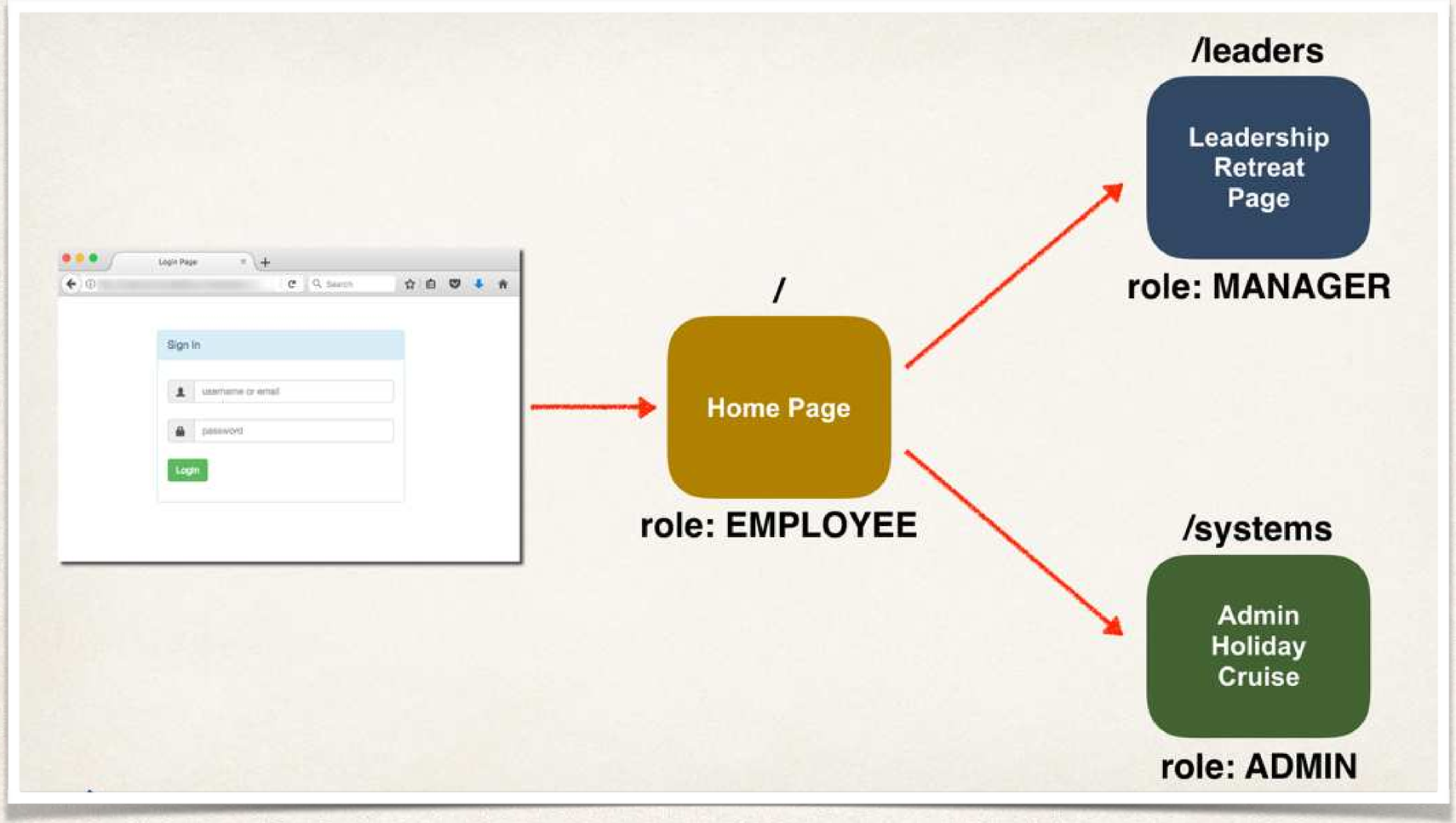
3. Restrict Access based on Roles

Step 1: Create Supporting Controller code and View Pages

Step 1: Create Supporting Controller code and View Pages



Step 1: Create Supporting Controller code and View Pages



Cover these steps in the video

Step 2: Update Our User Roles

Step 2: Update Our User Roles

User ID	Password	Roles

Step 2: Update Our User Roles

User ID	Password	Roles
`john`	`test123`	`EMPLOYEE`

Step 2: Update Our User Roles

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

Step 2: Update Our User Roles

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

Step 2: Update Our User Roles

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

We can give ANY names for user roles

Step 2: Update Our User Roles

File: DemoSecurityConfig.java

```
@Override
protected void configure(AuthenticationManagerBuilder auth) throws Exception {

  // add our users for in memory authentication





}
```

Step 2: Update Our User Roles

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

File: DemoSecurityConfig.java

```
@Override
protected void configure(AuthenticationManagerBuilder auth) throws Exception {

  // add our users for in memory authentication





}
```

Step 2: Update Our User Roles

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

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"));

}
```

Step 2: Update Our User Roles

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

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"));

}
```

Step 3: Restricting Access to Roles

Step 3: Restricting Access to Roles

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

Step 3: Restricting Access to Roles

- Update your Spring Security Java configuration file (.java)
- General Syntax

Step 3: Restricting Access to Roles

- Update your Spring Security Java configuration file (.java)
- General Syntax

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

Step 3: Restricting Access to Roles

- Update your Spring Security Java configuration file (.java)
- General Syntax

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

Restrict access to a given path "/systems/"**

Step 3: Restricting Access to Roles

- Update your Spring Security Java configuration file (.java)
- General Syntax

Single role

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

Restrict access to a given path "/systems/**"

Step 3: Restricting Access to Roles

- Update your Spring Security Java configuration file (.java)
- General Syntax

Single role

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

Restrict access to a given path "/systems/**"

"ADMIN"

Step 3: Restricting Access to Roles

Step 3: Restricting Access to Roles

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

Step 3: Restricting Access to Roles

Any role in the list, comma-delimited list

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

Step 3: Restricting Access to Roles

Any role in the list, comma-delimited list

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

“ADMIN”, “DEVELOPER”, “VIP”, “PLATINUM”

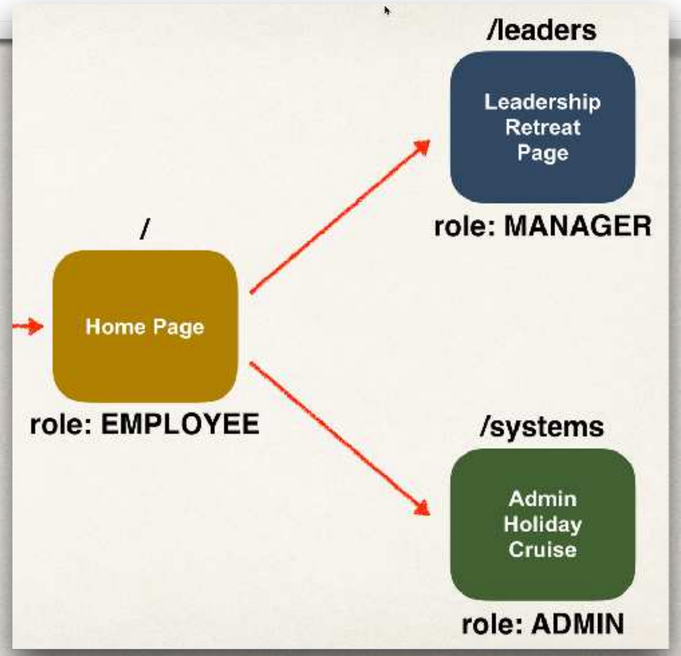
Restrict Path to EMPLOYEE

Restrict Path to EMPLOYEE

```
antMatchers("/").hasRole("EMPLOYEE")
```

Restrict Path to EMPLOYEE

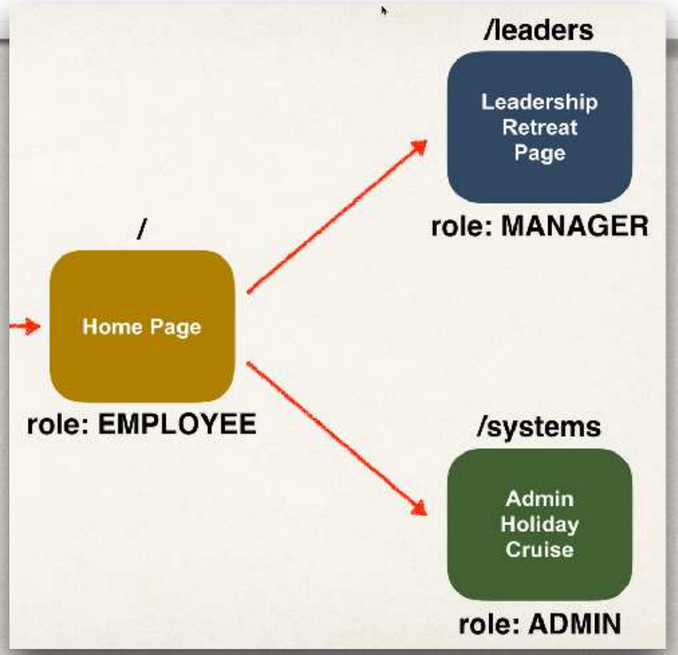
```
antMatchers("/").hasRole("EMPLOYEE")
```



Restrict Path to EMPLOYEE

```
antMatchers("/").hasRole("EMPLOYEE")
```

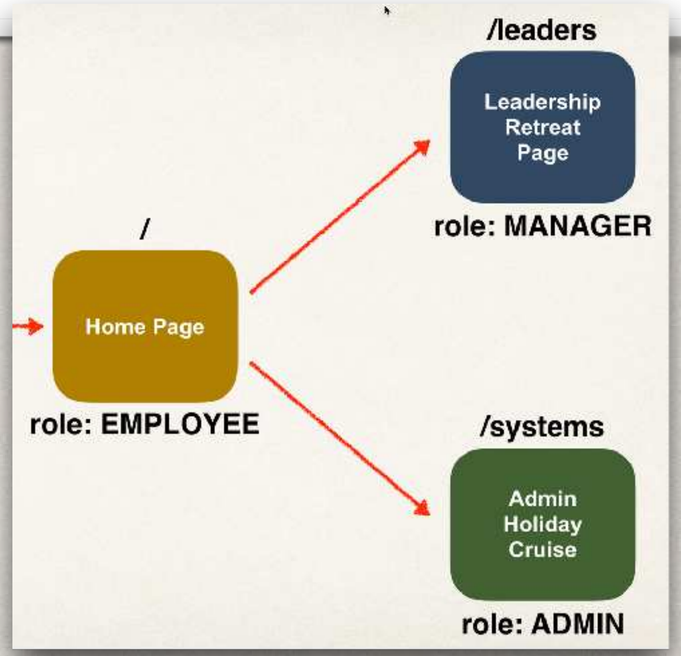
Match on root path: /



Restrict Path /leaders to MANAGER

Restrict Path /leaders to MANAGER

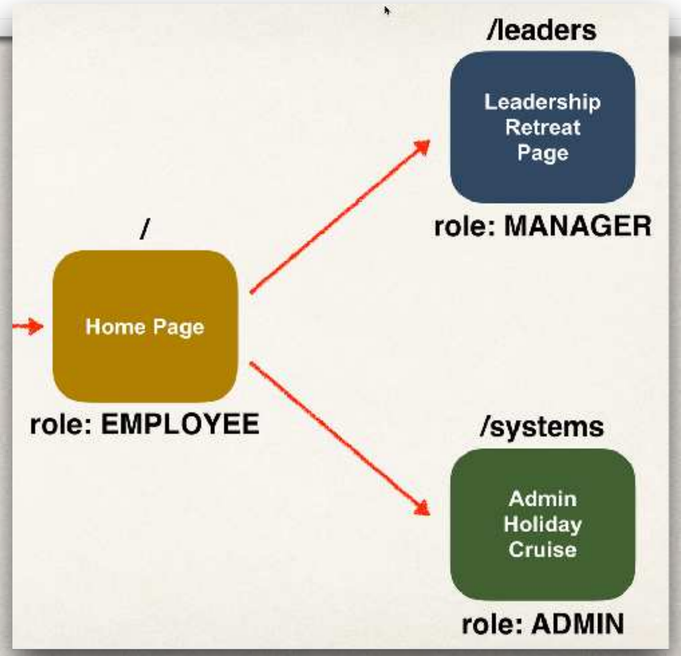
```
antMatchers("/leaders/**").hasRole("MANAGER")
```



Restrict Path /leaders to MANAGER

```
antMatchers("/leaders/**").hasRole("MANAGER")
```

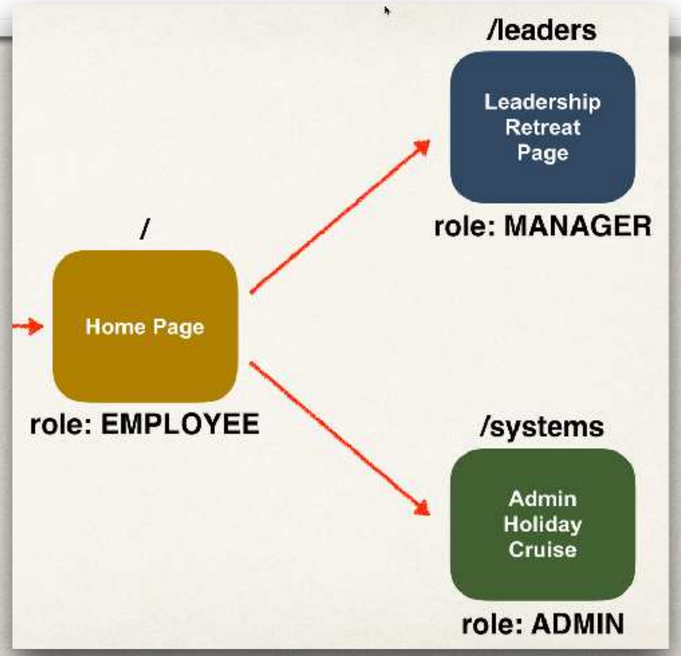
Match on path: /leaders
And all sub-directories (**)



Restrict Path /systems to ADMIN

Restrict Path /systems to ADMIN

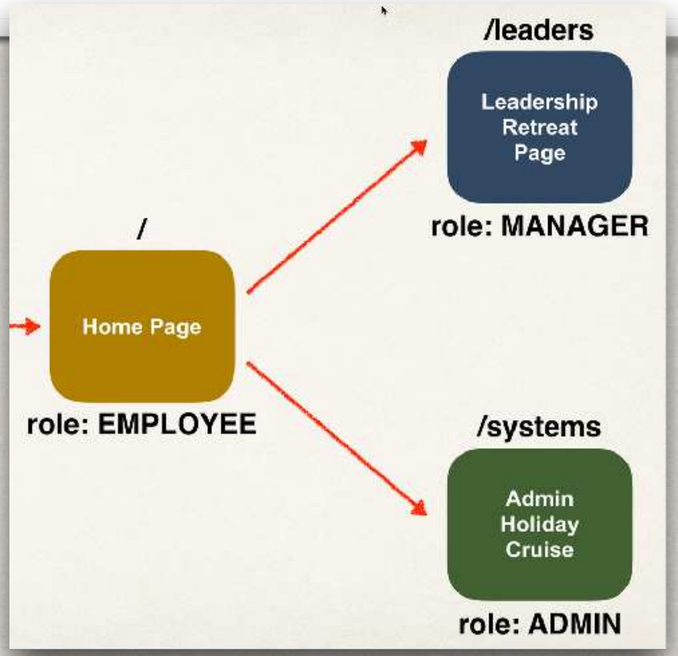
```
antMatchers("/systems/**").hasRole("ADMIN")
```



Restrict Path /systems to ADMIN

```
antMatchers("/systems/**").hasRole("ADMIN")
```

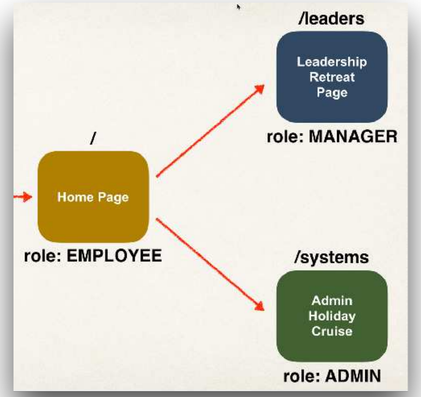
Match on path: /systems
And all sub-directories (**)



Pull It Together

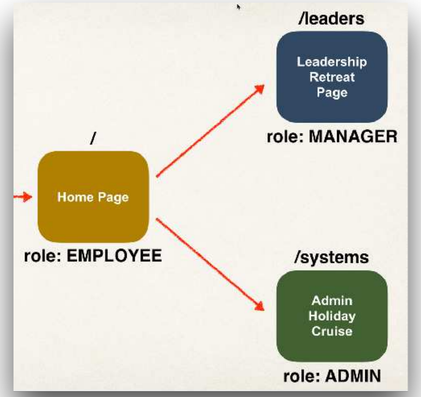
Pull It Together

```
@Override
protected void configure(HttpSecurity http) throws Exception {

  http.authorizeRequests()
    .antMatchers("/").hasRole("EMPLOYEE")
    .antMatchers("/leaders/**").hasRole("MANAGER")
    .antMatchers("/systems/**").hasRole("ADMIN")
    .and()
    .formLogin()
   …
}
```



Pull It Together

```
@Override
protected void configure(HttpSecurity http) throws Exception {

  http.authorizeRequests()
    .antMatchers("/").hasRole("EMPLOYEE")
    .antMatchers("/leaders/**").hasRole("MANAGER")
    .antMatchers("/systems/**").hasRole("ADMIN")
    .and()
    .formLogin()
   ...
}
```




```
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"));
```

Custom Access Denied Page

Default Access Denied Page

Default Access Denied Page

HTTP Status 403 – Forbidden

Type Status Report

Message Forbidden

Description The server understood the request but refuses to authorize it.

Custom Access Denied Page

Custom Access Denied Page

Access Denied - You are not authorized to access this resource.

Back to Home Page

Custom Access Denied Page

Access Denied - You are not authorized to access this resource.

Back to Home Page

You can customize this page

HTML + CSS

Development Process

Step-By-Step

Development Process

Step-By-Step

1. Configure custom page for access denied

Development Process

Step-By-Step

1. Configure custom page for access denied

2. Create supporting controller code and view page

Step 1: Configure custom page access denied

Step 1: Configure custom page access denied

```
@Override
protected void configure(HttpSecurity http) throws Exception {

  …
  .exceptionHandling()
    .accessDeniedPage("/access-denied");

}
```

Step 1: Configure custom page access denied

```
@Override
protected void configure(HttpSecurity http) throws Exception {

  …
  .exceptionHandling()
    .accessDeniedPage("/access-denied");

}
```

Our request mapping path

Step 2: Create Supporting Controller code and View Page

Step 2: Create Supporting Controller code and View Page

Cover these steps in the video

Display Content Based on Roles

Why Show These Links?

User: john

Role(s): [ROLE_EMPLOYEE]

Leadership Meeting (Only for Manager peeps)

IT Systems Meeting (Only for Admin peeps)

Logout

Why Show These Links?

User: john

Role(s): [ROLE_EMPLOYEE]

Leadership Meeting (Only for Manager peeps)

IT Systems Meeting (Only for Admin peeps)

Logout

Since John is an employee,

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

Display Content Based on Roles

User: mary

Role(s): [ROLE_EMPLOYEE, ROLE_MANAGER]

Leadership Meeting (Only for Manager peeps)

Logout

Display Content Based on Roles

Logged in as Mary

User: mary

Role(s): [ROLE_EMPLOYEE, ROLE_MANAGER]

Leadership Meeting (Only for Manager peeps)

Logout

Display Content Based on Roles

Logged in as Mary

User: mary

Role(s): [ROLE_EMPLOYEE, ROLE_MANAGER]

Leadership Meeting (Only for Manager peeps)

Logout

Mary can see MANAGER content

Spring Security JSP Tags

User: mary

Role(s): [ROLE_EMPLOYEE, ROLE_MANAGER]

Leadership Meeting (Only for Manager peeps)

Spring Security JSP Tags

```
…
<security:authorize access="hasRole('MANAGER')">

  <p>
      <a href="${pageContext.request.contextPath}/leaders">
        Leadership Meeting
      </a>
      (Only for Manager peeps)
  </p>

 </security:authorize>
```

User: mary

Role(s): [ROLE_EMPLOYEE, ROLE_MANAGER]

Leadership Meeting (Only for Manager peeps)

Spring Security JSP Tags

Only show this section for users with MANAGER role

```
...
<security:authorize access="hasRole('MANAGER')">

  <p>
     <a href="${pageContext.request.contextPath}/leaders">
       Leadership Meeting
     </a>
     (Only for Manager peeps)
  </p>

 </security:authorize>
```

User: mary

Role(s): [ROLE_EMPLOYEE, ROLE_MANAGER]

Leadership Meeting (Only for Manager peeps)

Display Content Based on Roles

User: susan

Role(s): [ROLE_ADMIN, ROLE_EMPLOYEE]

IT Systems Meeting (Only for Admin peeps)

Display Content Based on Roles

Logged in as Susan

User: susan

Role(s): [ROLE_ADMIN, ROLE_EMPLOYEE]

IT Systems Meeting (Only for Admin peeps)

Display Content Based on Roles

Logged in as Susan

User: susan

Role(s): [ROLE_ADMIN, ROLE_EMPLOYEE]

IT Systems Meeting (Only for Admin peeps)

Susan can see ADMIN content

Spring Security JSP Tags

User: susan

Role(s): [ROLE_ADMIN, ROLE_EMPLOYEE]

IT Systems Meeting (Only for Admin peeps)

Spring Security JSP Tags

```
…
<security:authorize access="hasRole('ADMIN')">

  <p>
     <a href="${pageContext.request.contextPath}/systems">
       IT Systems Meeting
     </a>
    (Only for Admin peeps)
  </p>

 </security:authorize>
```

User: susan

Role(s): [ROLE_ADMIN, ROLE_EMPLOYEE]

IT Systems Meeting (Only for Admin peeps)

Spring Security JSP Tags

Only show this section for users with ADMIN role

```
...
<security:authorize access="hasRole('ADMIN')">

  <p>
     <a href="${pageContext.request.contextPath}/systems">
       IT Systems Meeting
     </a>
    (Only for Admin peeps)
  </p>

 </security:authorize>
```

User: susan

Role(s): [ROLE_ADMIN, ROLE_EMPLOYEE]

IT Systems Meeting (Only for Admin peeps)