CS544 EA

Applications

Spring Security

Spring Security

- Security, establishing who a user is (authentication), and allowing or disallowing actions (authorization) are vital to any serious application.
- In this Spring Security Module we will look at:
  - Authentication in a web environment
  - Requiring Authorization for certain web pages
  - Requiring Authorization for method calls
  - Defending against common attacks

Basic Example

- We’ll create a basic example to show the essentials of Spring Security
  - Configured with Java Config
  - Configured with XML

- Then we’ll go into the details of the different parts

WebAppInitializer

```
public class MyWebAppInitializer implements WebApplicationInitializer {

     @Override
     public void onStartup(ServletContext container) throws ServletException {
          // Create the 'root' Spring application context
          AnnotationConfigWebApplicationContext rootContext
               = new AnnotationConfigWebApplicationContext();
          rootContext.register(WebConfig.class, SecurityConfig.class);
          container.addListener(new ContextLoaderListener(rootContext));

          // Create the dispatcher servlet
          ServletRegistration.Dynamic appServlet = container.addServlet("mvc",
                    new DispatcherServlet(new GenericWebApplicationContext()));
          appServlet.setLoadOnStartup(1);
          appServlet.addMapping("/");

          container.addFilter("springSecurityFilterChain",
               new DelegatingFilterProxy("springSecurityFilterChain"))
               .addMappingForUrlPatterns(null, false, "/*");
     }
}
```

Load both WebConfig and SecurityConfig

Apply Security Filter to all incoming requests

WebConfig

Normal SpringMVC WebConfig

```
@Configuration
@EnableWebMvc
@ComponentScan("cs544")
public class WebConfig implements WebMvcConfigurer{
    @Bean
    public ViewResolver viewResolver() {
        InternalResourceViewResolver bean = new InternalResourceViewResolver();

        bean.setViewClass(JstlView.class);
        bean.setPrefix("/WEB-INF/view/");
        bean.setSuffix(".jsp");

        return bean;
    }
}
```

You can think of the .and()
as the closing of an XML tag
it takes you back to the root

SecurityConfig

@EnableWebSecurity and
Extends WebSecurityConfigAdapter

Showing
2 of 3
configure
methods
that can be
overridden

```
@Configuration
@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter {
     @Override
     protected void configure(AuthenticationManagerBuilder auth) throws Exception {
          auth.inMemoryAuthentication()
          .withUser("admin").password("{noop}123").roles("USER","ADMIN").and()
          .withUser("user").password("{noop}bla").roles("USER");
     }

     @Override
     protected void configure(HttpSecurity http) throws Exception {
          http
               .authorizeRequests().antMatchers("/important/**").hasRole("USER").and()
               .formLogin().and()
               .logout();
    }
}
```

Creates an inMemory use details
without encoded passwords
(just for demo, not good for production!)

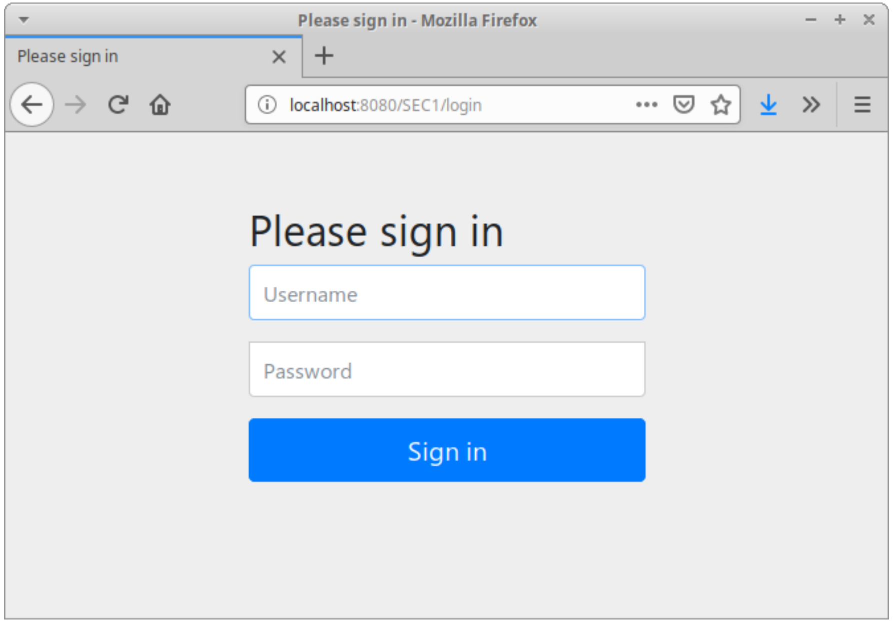
Make sure anyone wanting to access
anything under important
has the USER role

People can login with a form
and logout

Generated login.jsp

You can also write your own

- Spring Security generates a form-login
  - When not logged in and try to access /important/**



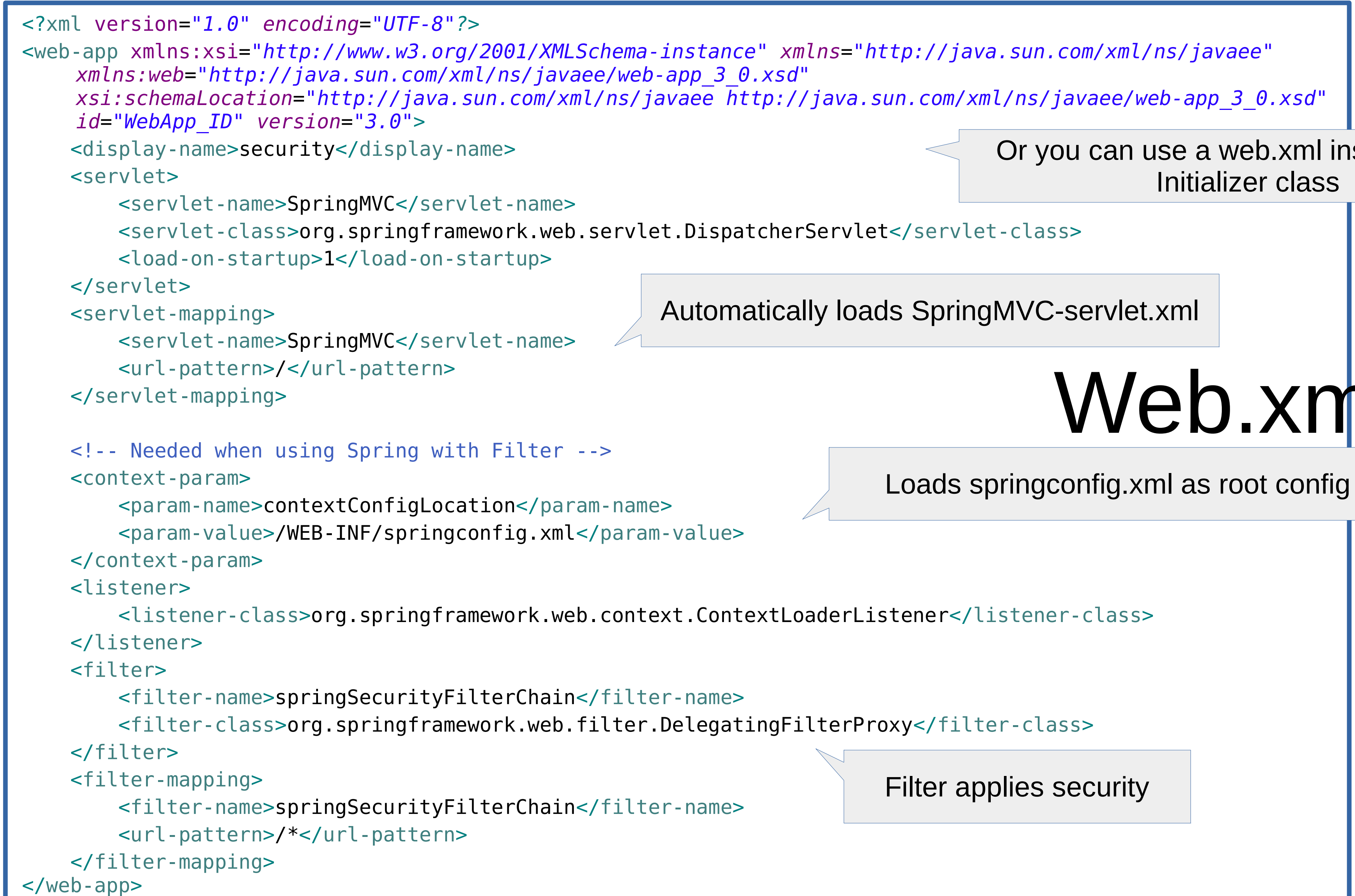
Please sign in

Username

Password

Sign in

Web.xml



```xml
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://java.sun.com/xml/ns/javaee"
     xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
     xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
     id="WebApp_ID" version="3.0">
    <display-name>security</display-name>
    <servlet>
        <servlet-name>SpringMVC</servlet-name>
        <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>SpringMVC</servlet-name>
        <url-pattern>/</url-pattern>
    </servlet-mapping>

    <!-- Needed when using Spring with Filter -->
    <context-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>/WEB-INF/springconfig.xml</param-value>
    </context-param>
    <listener>
        <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
    </listener>
    <filter>
        <filter-name>springSecurityFilterChain</filter-name>
        <filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-class>
    </filter>
    <filter-mapping>
        <filter-name>springSecurityFilterChain</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>
</web-app>
```

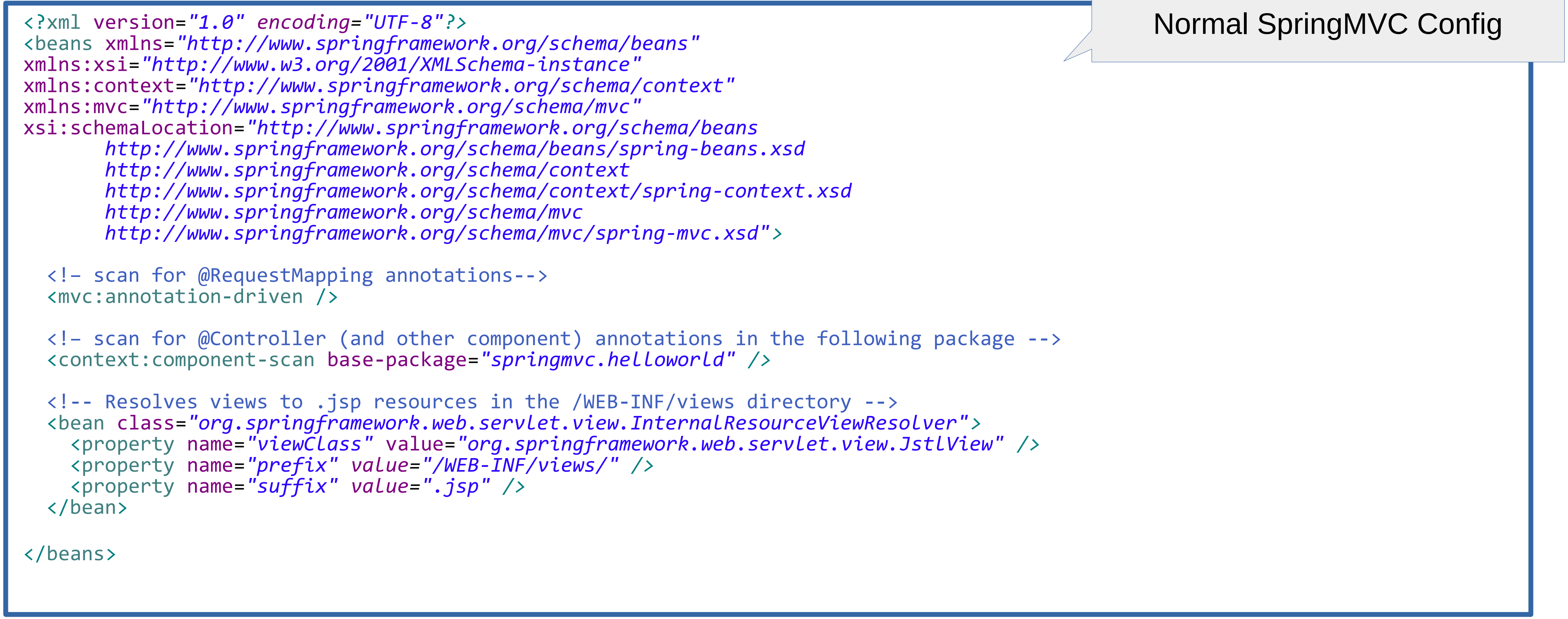
Or you can use a web.xml instead of the Initializer class

Automatically loads SpringMVC-servlet.xml

Loads springconfig.xml as root config

Filter applies security

SpringMVC-servlet.xml



```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:context="http://www.springframework.org/schema/context"
xmlns:mvc="http://www.springframework.org/schema/mvc"
xsi:schemaLocation="http://www.springframework.org/schema/beans
       http://www.springframework.org/schema/beans/spring-beans.xsd
       http://www.springframework.org/schema/context
       http://www.springframework.org/schema/context/spring-context.xsd
       http://www.springframework.org/schema/mvc
       http://www.springframework.org/schema/mvc/spring-mvc.xsd">

  <!- scan for @RequestMapping annotations-->
  <mvc:annotation-driven />

  <!- scan for @Controller (and other component) annotations in the following package -->
  <context:component-scan base-package="springmvc.helloworld" />

  <!-- Resolves views to .jsp resources in the /WEB-INF/views directory -->
  <bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">
    <property name="viewClass" value="org.springframework.web.servlet.view.JstlView" />
    <property name="prefix" value="/WEB-INF/views/" />
    <property name="suffix" value=".jsp" />
  </bean>

</beans>
```

Normal SpringMVC Config

Springconfig.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:sec="http://www.springframework.org/schema/security"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.springframework.org/schema/beans
     http://www.springframework.org/schema/beans/spring-beans.xsd
     http://www.springframework.org/schema/security
     http://www.springframework.org/schema/security/spring-security.xsd">

    <sec:http>
        <sec:intercept-url pattern="/important/**" access=“ROLE_USER"/>
        <sec:form-login />
        <sec:logout />
    </sec:http>

    <sec:authentication-manager>
        <sec:authentication-provider>
            <sec:user-service>
                <sec:user name="test" password="{noop}123" authorities="ROLE_USER, ROLE_ADMIN" />
                <sec:user name="bob" password="{noop}bobiscool" authorities="ROLE_USER" />
            </sec:user-service>
        </sec:authentication-provider>
    </sec:authentication-manager>
</beans>
```

Security namespace

http elements specify url patterns for security

Authentication manager / provider configuration