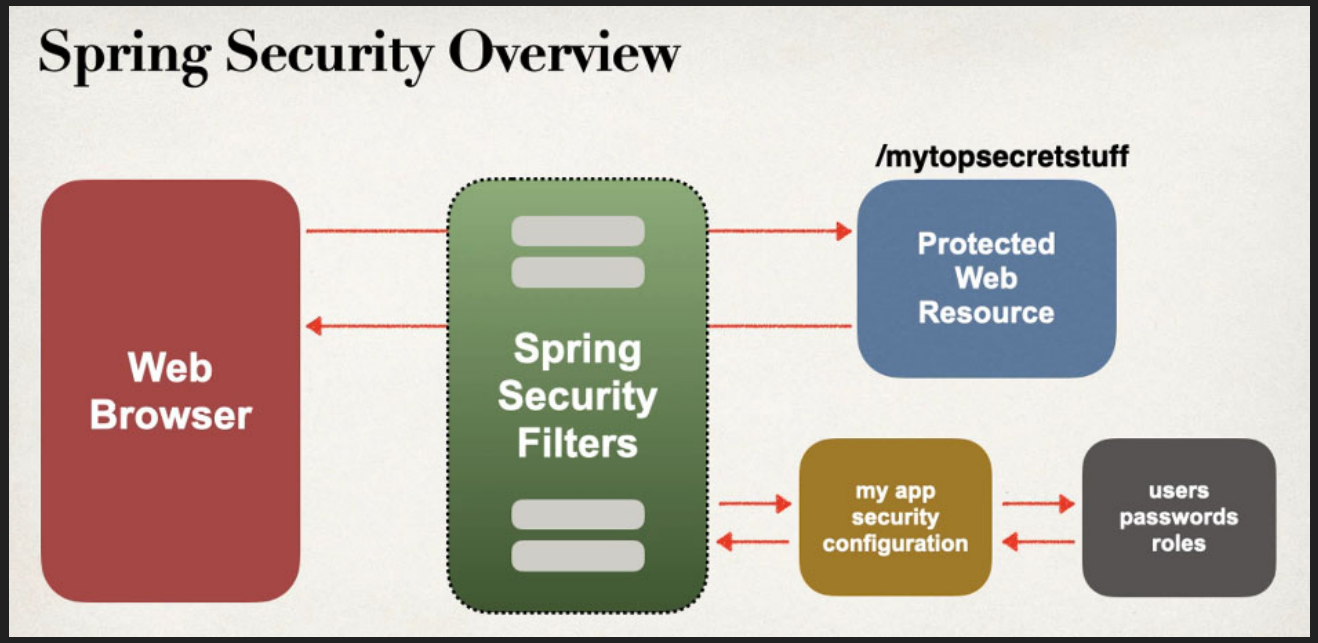
Spring MVC security

Spring Security Model

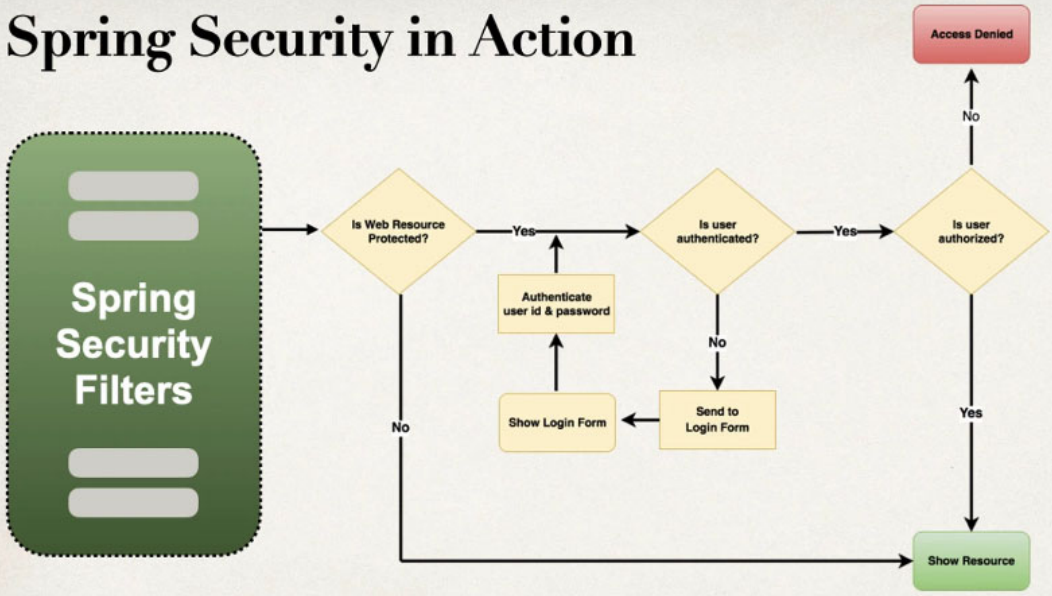
- Spring Security defines a framework for security
- Implemented using Servlet filters in the background
- Two methods of securing an app: declarative and programmatic

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
/mytopsecretstuff
Web Browser
Spring Security Filters
Protected Web Resource
my app security configuration
users passwords roles

Spring Security in Action



Security Concepts

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

Enabling Spring Security

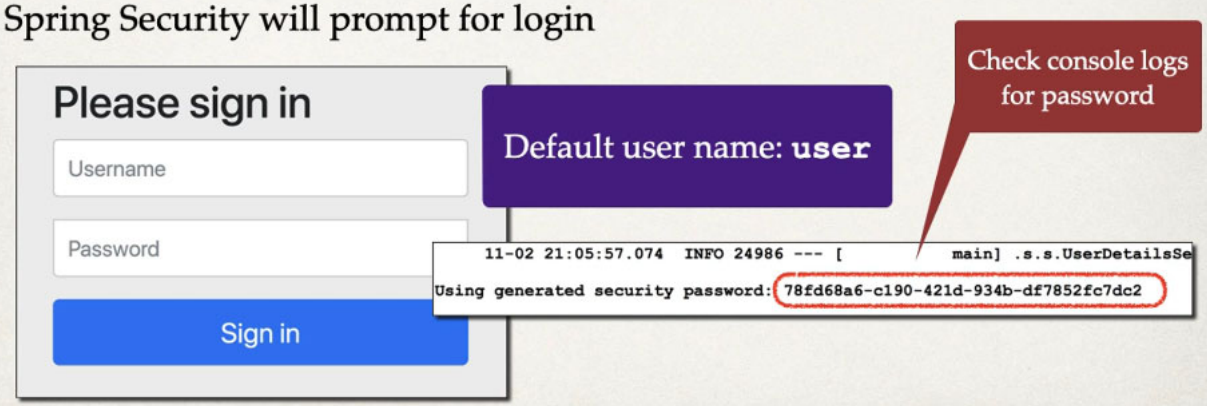
1. Edit **`pom.xml`** and add **`spring-boot-starter-security`**

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-security</artifactId>
</dependency>
```

2. This will ***automagically*** secure all endpoints for application

Secured Endpoints

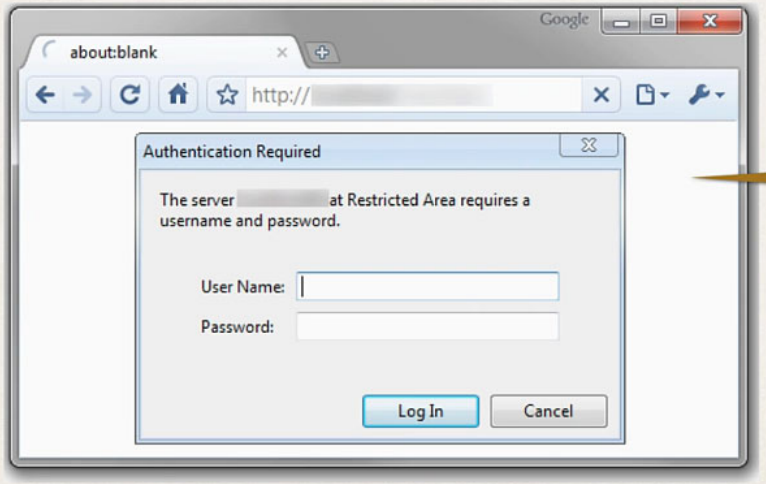
- Now when you access your application
- Spring Security will prompt for login



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

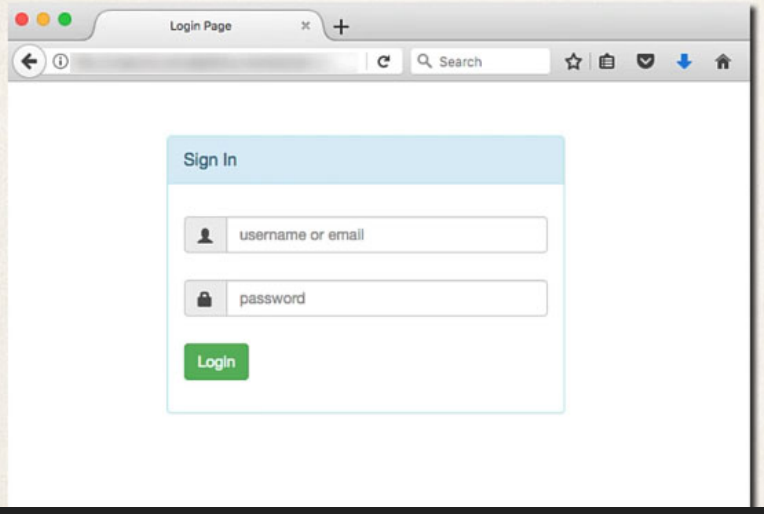


Built-in Dialog from browser

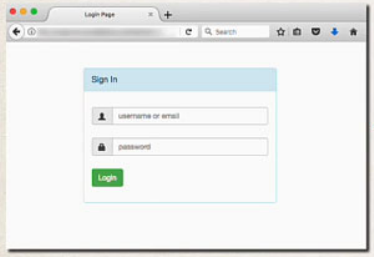
bleh!

Spring Security - Default Login Form

Your Own Custom Login Form



Our Example



/

Home Page

role: EMPLOYEE

/leaders

Leadership Retreat Page

role: MANAGER

/systems

Admin Holiday Cruise

role: ADMIN

Development Process

Step-By-Step

1. Create project at Spring Initializr website
    1. Add Maven dependencies for Spring MVC Web App, Security, Thymeleaf
2. Develop our Spring controller
3. Develop our Thymeleaf view page

Step 1: Add Maven dependencies for Spring MVC Web App

File: pom.xml

```
...
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-web</artifactId>
</dependency>

<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-thymeleaf</artifactId>
</dependency>

<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-security</artifactId>
</dependency>

<dependency>
  <groupId>org.thymeleaf.extras</groupId>
  <artifactId>thymeleaf-extras-springsecurity6</artifactId>
</dependency>
...
```

Spring MVC web support

Thymeleaf view support

Spring Security support

Thymeleaf Security support

Step 2: Develop our Spring Controller

File: DemoController.java

```
@Controller
public class DemoController {

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

      return "home";
   }

}
```

View name

src/main/resources/templates/home.html

Project

- ● Gradle - Groovy
- ○ Gradle - Kotlin
- ○ Maven

Language

- ● Java
- ○ Kotlin
- ○ Groovy

Spring Boot

- ○ 3.2.0 (SNAPSHOT)
- ○ 3.2.0 (RC1)
- ○ 3.1.6 (SNAPSHOT)
- ● 3.1.5
- ○ 3.0.13 (SNAPSHOT)
- ○ 3.0.12
- ○ 2.7.18 (SNAPSHOT)
- ○ 2.7.17

Project Metadata

Group	com.jac.springboot
Artifact	demosecurity
Name	demosecurity
Description	Demo project for Spring Boot
Package name	com.jac.springboot.demosecurity
Packaging	● Jar ○ War
Java	○ 21 ● 17 ○ 11 ○ 8

Dependencies ADD DEPENDENCIES... CTRL + B

Spring Web WEB
Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.

Thymeleaf TEMPLATE ENGINES
A modern server-side Java template engine for both web and standalone environments. Allows HTML to be correctly displayed in browsers and as static prototypes.

Spring Security SECURITY
Highly customizable authentication and access-control framework for Spring applications.

Spring Boot DevTools DEVELOPER TOOLS
Provides fast application restarts, LiveReload, and configurations for enhanced development experience.

Our Users

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

We can give ANY names for user roles

Development Process

Step-By-Step

1. Create Spring Security Configuration (@Configuration)

2. Add users, passwords and roles

Step 1: Create Spring Security Configuration

File: DemoSecurityConfig.java

```
import org.springframework.context.annotation.Configuration;

@Configuration
public class DemoSecurityConfig {

  // add our security configurations here …

}
```

Spring Security Password Storage

- In Spring Security, passwords are stored using a specific format

```
{id}encodedPassword
```

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

Password Example

The encoding algorithm id

The password

```
{noop}test123
```

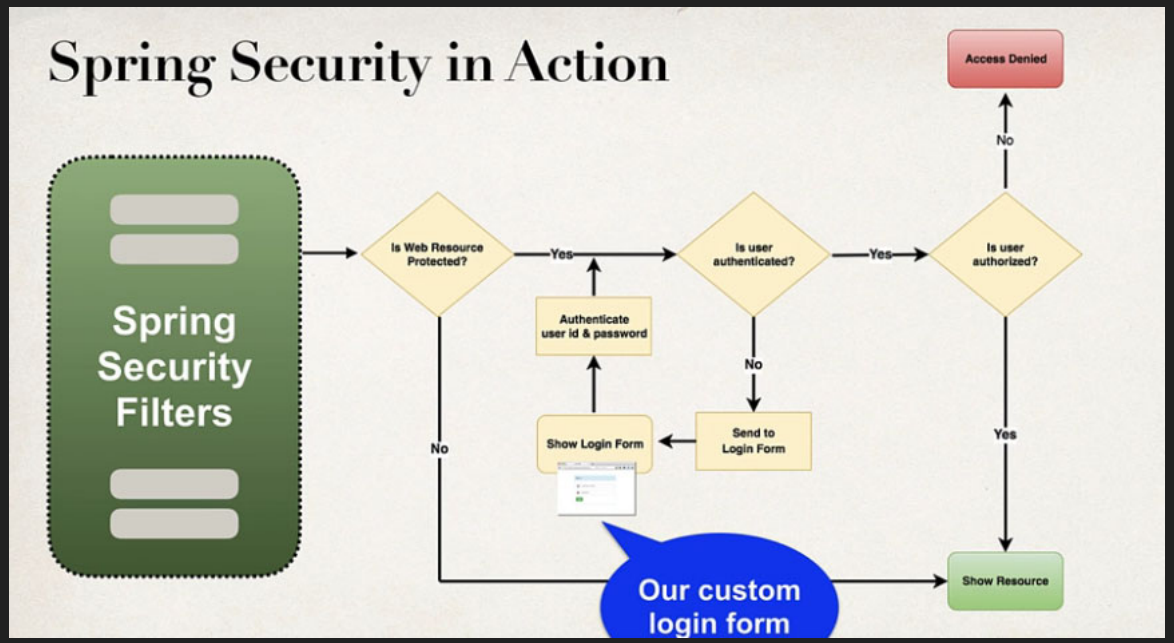
Let's Spring Security know the passwords are stored as plain text (noop)

Step 2: Add users, passwords and roles

File: DemoSecurityConfig.java

```java
@Configuration
public class DemoSecurityConfig {

    @Bean
    public InMemoryUserDetailsManager userDetailsManager() {

        UserDetails john = User.builder()
                .username("john")
                .password("{noop}test123")
                .roles("EMPLOYEE")
                .build();

        UserDetails mary = User.builder()
                .username("mary")
                .password("{noop}test123")
                .roles("EMPLOYEE", "MANAGER")
                .build();

        UserDetails susan = User.builder()
                .username("susan")
                .password("{noop}test123")
                .roles("EMPLOYEE", "MANAGER", "ADMIN")
                .build();

        return new InMemoryUserDetailsManager(john, mary, susan);
```

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


Spring Security in Action
Spring Security Filters
Is Web Resource Protected?
Yes
No
Authenticate user id & password
Is user authenticated?
Yes
No
Send to Login Form
Show Login Form
Is user authorized?
No
Yes
Access Denied
Show Resource
Our 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)

Step 1: Modify Spring Security Configuration

File: DemoSecurityConfig.java

```
@Bean
public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {

    http.authorizeHttpRequests(configurer ->
                    configurer
                            .anyRequest().authenticated()
            )
            .formLogin(form ->
                    form
                            .loginPage("/showMyLoginPage")
                            .loginProcessingUrl("/authenticateTheUser")
                            .permitAll()
            );

    return http.build();
}
```

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";

  }
}
```

```
.loginPage("/showMyLoginPage")
.loginProcessingUrl("/authenticateTheUser")
.permitAll()
```

- 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

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

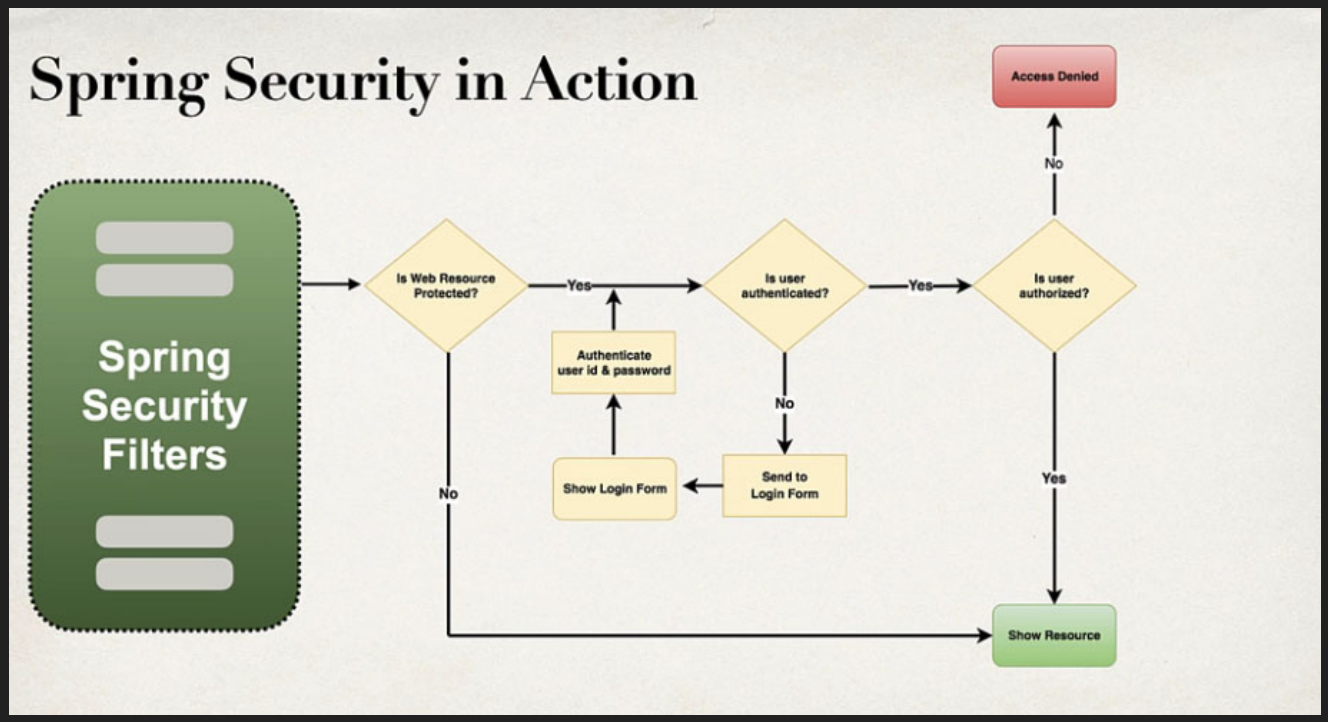
```
<form action="#" th:action="@{/authenticateTheUser}"
          method="POST">
   ...
</form>
```

```
.loginPage("/showMyLoginPage")
.loginProcessingUrl("/authenticateTheUser")
.permitAll()
```

- 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" />
```


Spring Security in Action
Spring Security Filters
Is Web Resource Protected?
Yes
No
Authenticate user id & password
Show Login Form
Is user authenticated?
Yes
No
Send to Login Form
Is user authorized?
No
Yes
Access Denied
Show Resource

Context Path

- Is the same as context root : root path for your application

Context Root: my-ecommerce-app

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

Gives us access to context path dynamically

```
<form action="#" th:action="@{/authenticateTheUser}"
          method="POST">
  ...
```

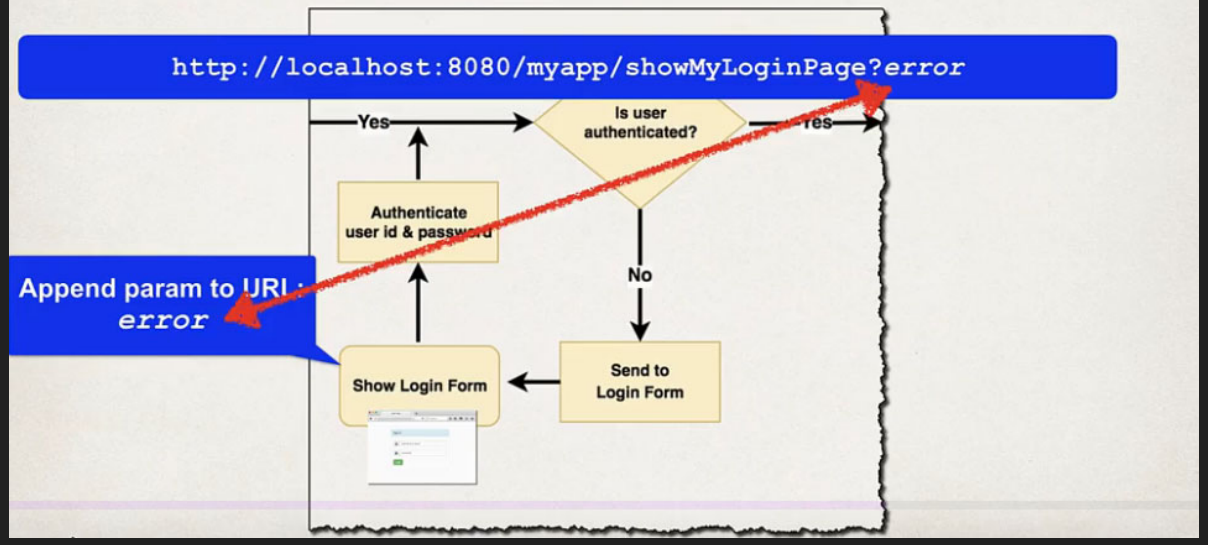
```java
@Bean
public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {

    http.authorizeHttpRequests(configurer ->
                    configurer
                            .anyRequest().authenticated()
            )
            .formLogin(form ->
                    form
                            .loginPage("/showMyLoginPage")
                            .loginProcessingUrl("/authenticateTheUser")
                            .permitAll()
            );
```

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

Failed Login

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

Failed Login



Step 1: Modify form - check for error

File: src/main/resources/templates/plain-login.html

If error param then show message

```
...
<form ...>

  <div th:if="${param.error}">

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

  </div>

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

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

Step 1: Add Logout support to Spring Security Configuration

File: DemoSecurityConfig.java

```
@Bean
public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {

    http.authorizeHttpRequests(configurer ->
                    configurer
                            .anyRequest().authenticated()
            )
            .formLogin(form ->
                    form
                            .loginPage("/showMyLoginPage")
                            .loginProcessingUrl("/authenticateTheUser")
                            .permitAll()
            )
            .logout(logout -> logout.permitAll()
            );

    return http.build();
}
```

Step 2: Add logout button

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

```
<form action="#" th:action="@{/logout}" method="POST">
      <input type="submit" value="Logout" />
</form>
```

Need to use a form for logout

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`

Modify Login form - check for "logout"

File: src/main/resources/templates/plain-login.html

```
…
<form … th:action="…" method="…">
  <div th:if="${param.logout}">
    <i>You have been logged out.</i>
  </div>
    User name: <input type="text" name="username" />
    Password: <input type="password" name="password" />
```

If logout param then show message

http://localhost:8080/showMyLoginPage?logout

Step 1: Display User ID

File: home.html

```
...

User: <span sec:authentication="principal.username"></span>
```

User ID

Step 2: Display User Roles

File: home.html

```
...
Role(s): <span sec:authentication="principal.authorities"></span>
```

Our Example

/

Home Page

role: EMPLOYEE

/leaders

Leadership Retreat Page

role: MANAGER

/systems

Admin Holiday Cruise

role: ADMIN

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

Single role

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

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

"ADMIN"

Any role in the list, comma-delimited list

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

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

```java
@Bean
public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {

    http.authorizeHttpRequests(configurer ->
                    configurer
                            .requestMatchers("/").hasRole("EMPLOYEE")
                            .requestMatchers("/leaders/**").hasRole("MANAGER")
                            .requestMatchers("/systems/**").hasRole("ADMIN")
                            .anyRequest().authenticated()
            )

  ...
}
```

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

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

Sign In

username

password

Login

/

Leadership Meeting (Only for Manager peeps)

IT Systems Meeting (Only for Admin peeps)

role: EMPLOYEE

/leaders

luv2code LEADERS Home Page

See you in Brazil ... for our annual Leadership retreat!
Keep this trip a secret, don't tell the regular employees LOL :-)

Back to Home Page

role: MANAGER

/systems

luv2code SYSTEMS Home Page

We have our annual holiday Caribbean cruise coming up. Register now!
Keep this trip a secret, don't tell the regular employees LOL :-)

Back to Home Page

Not showing the unauthorized content

```
...
<div sec:authorize="hasRole('MANAGER')">

  <p>
      <a th:href="@{/leaders}">
        Leadership Meeting
      </a>
      (Only for Manager peeps)
  </p>

</div>
```

User: mary

Role(s): [ROLE_EMPLOYEE, ROLE_MANAGER]

Leadership Meeting (Only for Manager peeps)

Database Support in Spring Security

Out-of-the-box

- Spring Security can read user account info from database
- By default, you have to follow Spring Security's predefined table schemas

Development Process

Step-By-Step

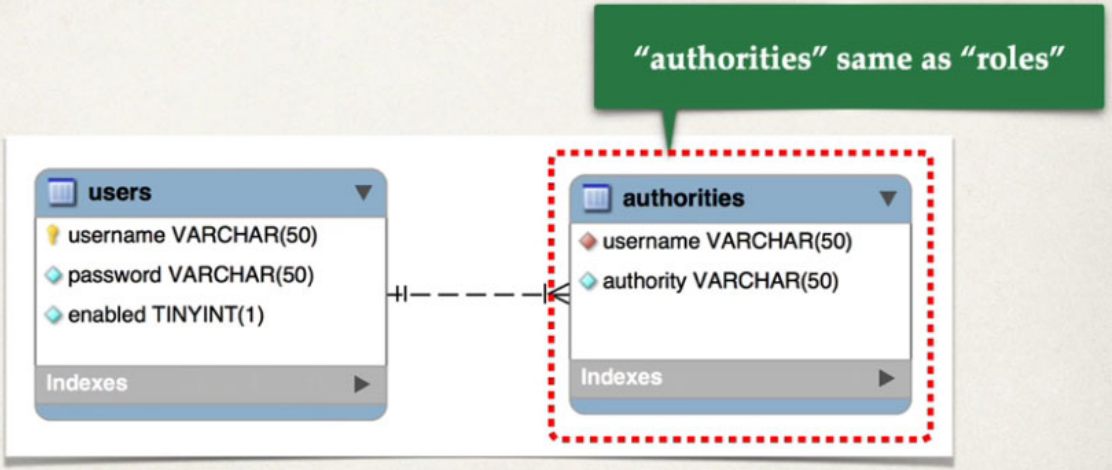
1. Develop SQL Script to set up database tables

2. Add database support to Maven POM file

3.Create JDBC properties file

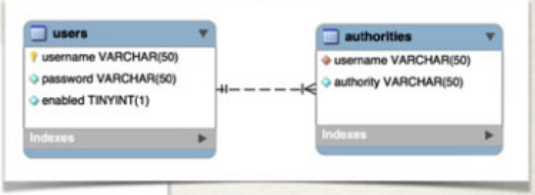
4. Update Spring Security Configuration to use JDBC

Default Spring Security Database Schema

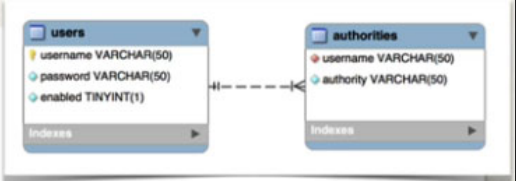


Step 1: Develop SQL Script to setup database tables

```sql
CREATE TABLE `users` (
  `username` varchar(50) NOT NULL,
  `password` varchar(50) NOT NULL,
  `enabled` tinyint NOT NULL,

  PRIMARY KEY (`username`)

) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```



Step 1: Develop SQL Script to setup database tables



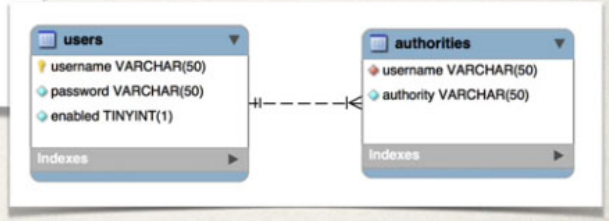

```sql
CREATE TABLE `authorities` (
  `username` varchar(50) NOT NULL,
  `authority` varchar(50) NOT NULL,

  UNIQUE KEY `authorities_idx_1` (`username`,`authority`),

  CONSTRAINT `authorities_ibfk_1`
  FOREIGN KEY (`username`)
  REFERENCES `users` (`username`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

Step 1: Develop SQL Script to setup database tables

"authorities" same as "roles"

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

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



```
@Configuration
public class DemoSecurityConfig {

    @Bean
    public UserDetailsManager userDetailsManager(DataSource dataSource) {

        return new JdbcUserDetailsManager(dataSource);
    }

    …
}
```

Inject data source
Auto-configured by Spring Boot

Tell Spring Security to use
JDBC authentication
with our data source

Spring Security Team Recommendation

- Spring Security recommends using the popular **bcrypt** algorithm
- bcrypt
  - Performs one-way encrypted hashing
  - Adds a random salt to the password for additional protection
  - Includes support to defeat brute force attacks

BCrypt

- https://bcrypt.online/

Modify DDL for Password Field

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

  PRIMARY KEY (`username`)

) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

Password column must be at least 68 chars wide

{bcrypt} - 8 chars
encodedPassword - 60 chars

Spring Security Login Process

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

Note:
The password from db is NEVER decrypted

Because bcrypt is a one-way encryption algorithm