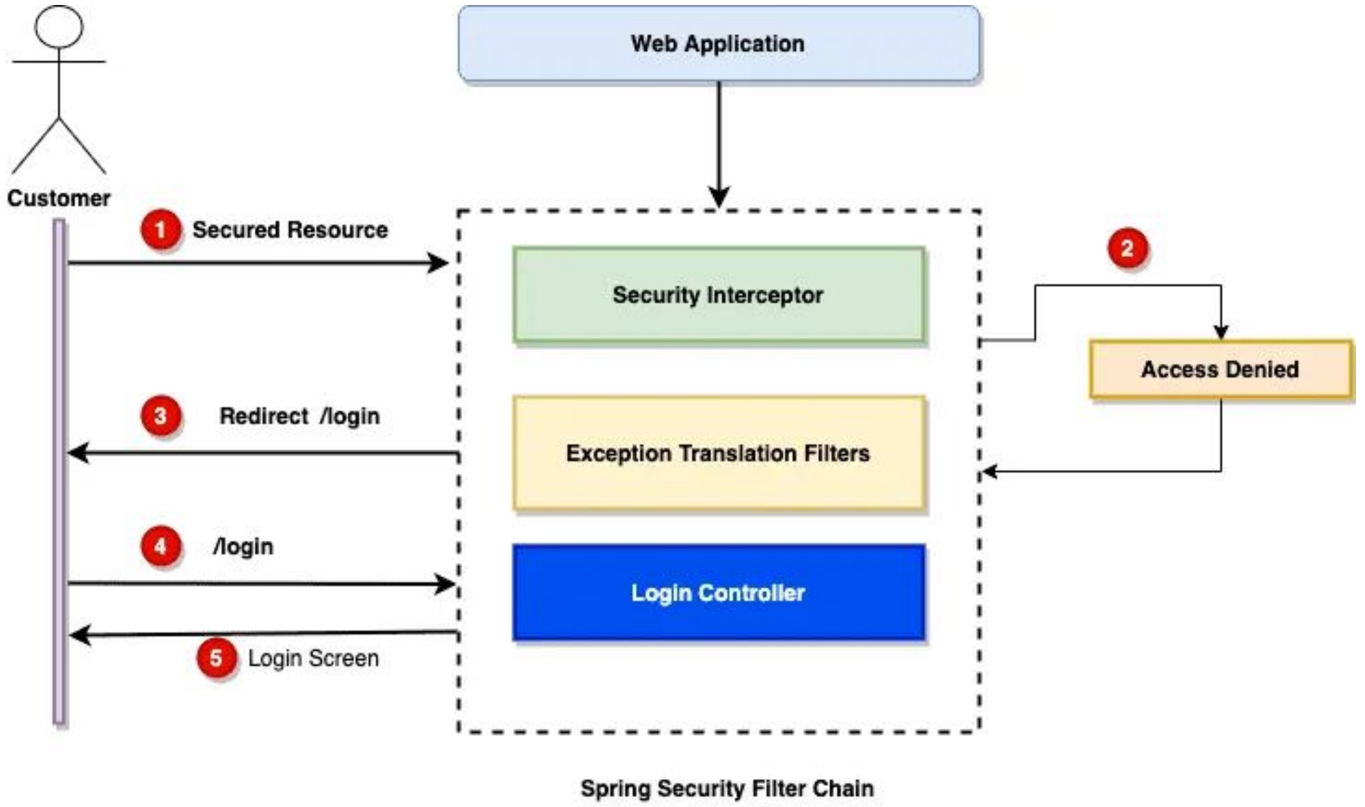
WILD
CODE
SCHOOL

How To Secure Your Web Application

Secure Web Application with Spring Security and OAuth2 (Keycloak) SSO

What is it about?



taken from https://www.javadevjournal.com/spring-security/spring-security-login/

What is it used for?

Spring Security

Spring framework for authentication and access control:

- Various means of authentication
- Role-based access control mechanisms
- Protection against certain attacks (XSS, CSRF, Session fixation ...)
- Persistent authentication (*Remember me*)

Purpose: Protection per Default

Limit: application security only

- to be completed with HTTPS, server security, etc ...

Authentification / Who are you?

Authentication

Challenge:

- Is the user (or browser) at the other end of the network who they say they are?

Strategy:

- Ask a question…
- Knowledge of secret information (login / password)
- Authentification HTTP / Form
- Use existing authentication services / mechanisms (LDAP, Active Directory, OAuth 2.0, OpenID, CAS, X.509 ...)

Authorization / Access Control - what can you do?

Authorization

Challenge:

- What is the authenticated user allowed to do?

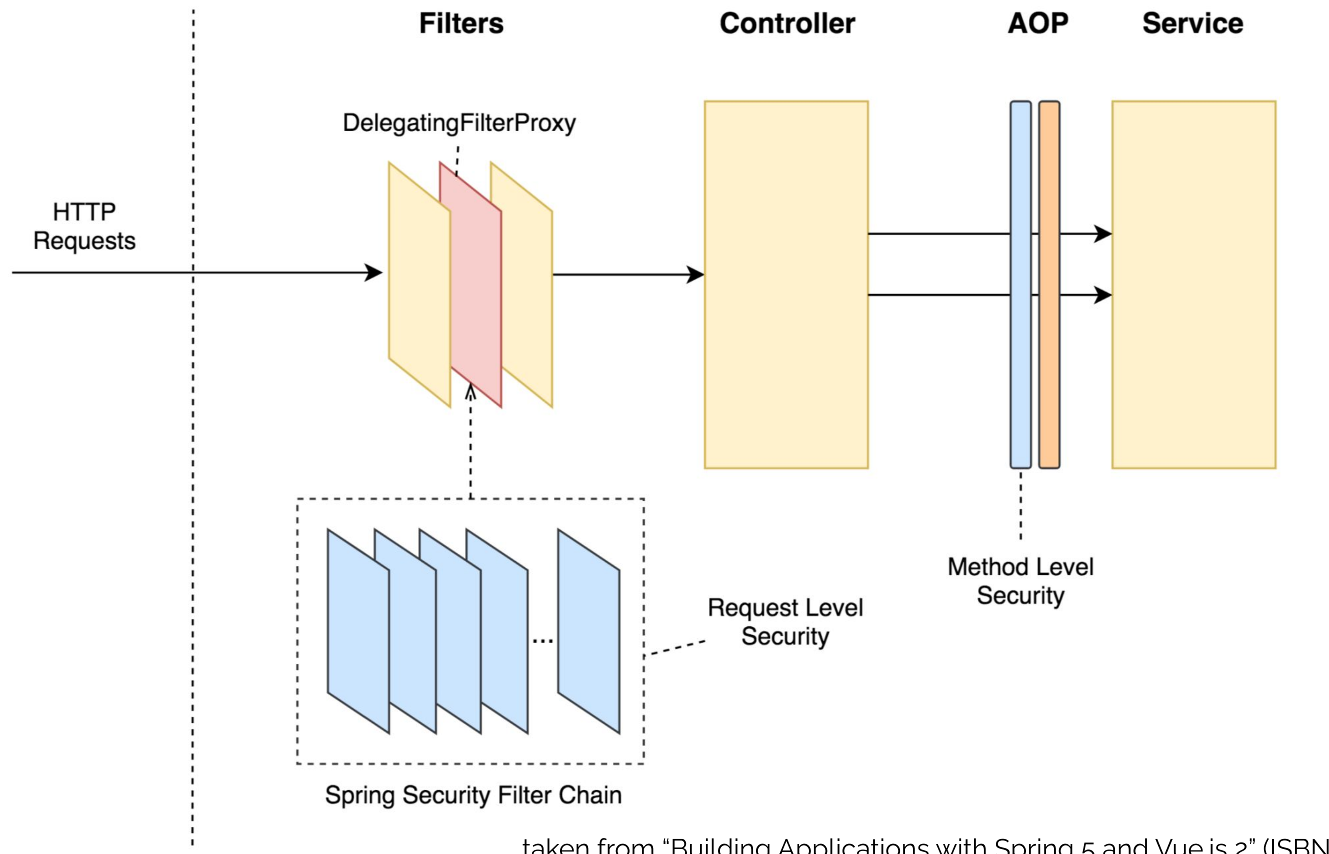
Strategy:

- Use of roles (user groups)

Associate by:

- Explicit Mapping
- URL Mappings
- Annotations on Methods
- Spring Expression Language (Spring EL)

Access Control (Security Level)



How to use it

Spring boot

Spring Initializr :

- Add dependency: Spring Security Default

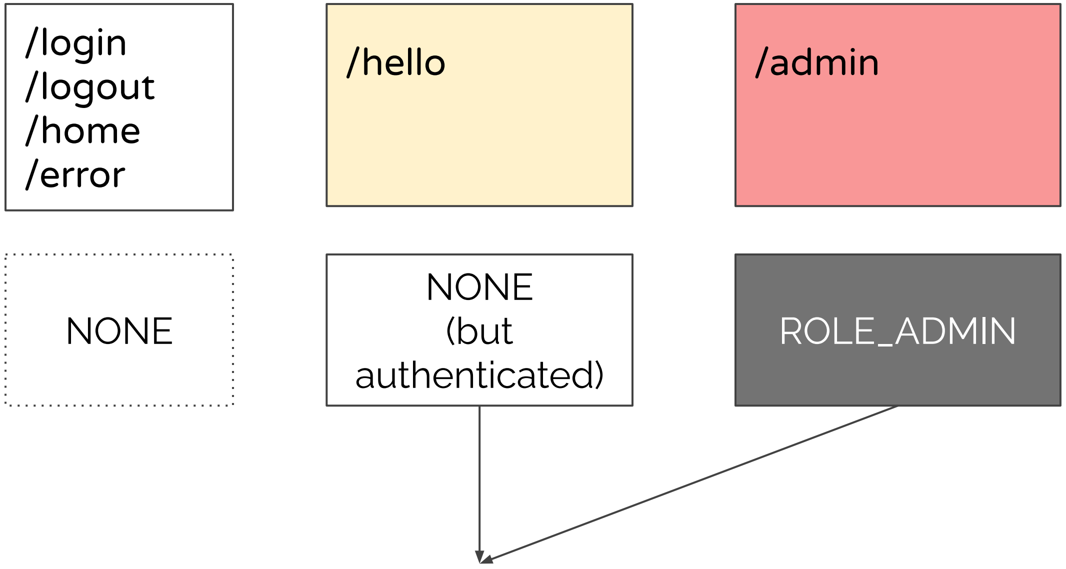
Protection by default:

- All routes require authentication
- Creation of a mapping / login with an authentication form
- Creation of a *GetMapping* / logout which displays a logout button
- Creation of a *PostMapping* / logout which disconnects the user

Setup:

- Creation of a user user with random password
- Using generated security password: xxxxxxxx-xxxx-xxxx-xxxx-xxxxx

Configure your Access Control (Webinar Sample)



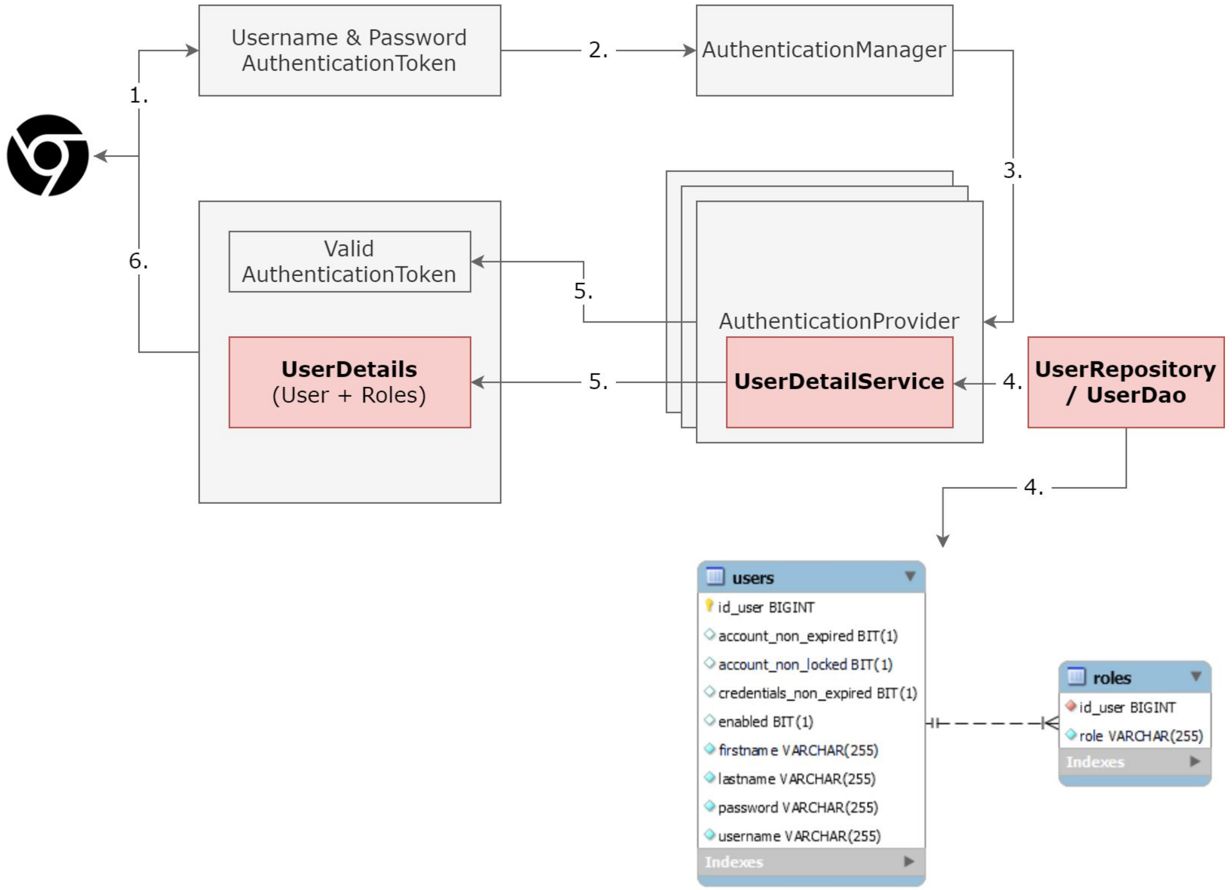
URL-based Security

WILD CODE SCHOOL

Authentification

User Management

Authentication Flow with Spring Security



User management

The User class

- Represents the users of the application
- Characterized (attributes) by at least:
    - *username*
    - *password*
    - *GrantedAuthority (permissions)*
- Class [org.springframework.security.core.userdetails.User]
- Can be used directly or extended (inheritance)
- Facilitates **UserBuilder** pattern
- Implements **UserDetails**

UserDetails

The UserDetails interface

- Define possible interactions with Users (eg. **User**)
- Interface org.springframework.security.core.userdetails.UserDetails
- Must be implemented by your User class
- For Spring Security: accounts are UserDetails (polymorphism)

```
public String getUsername(); // return username (never null)
public String getPassword(); // return password
public Collection<? extends GrantedAuthority> getAuthorities(); // return roles list (never null)

// User states preventing authentication
public boolean isAccountNonExpired();
public boolean isAccountNonLocked();
public boolean isCredentialsNonExpired(); // password expired
public boolean isEnabled();
```

Mapping to Roles

Roles / GrantedAuthority

Idea:

- The rights are not directly associated with the users
- Using user groups (roles)

Group:

- Interface GrantedAuthority
- Implementation (in general) is built with a String
- Example: SimpleGrantedAuthority
- Character strings prefixed by “ROLE_”

```
public String getAuthority(); // return the String representation of the Authority
```

Creating users (of the User class)

Class UserBuilder

Nested class in **User**
Get the **UserBuilder** :

- via **User** class methods

```
// User class
public static UserBuilder withDefaultPasswordEncoder();
// Deprecated because not for production use. i.e. Use for prototypes, tests, demos, etc....
// Won't disappear, so in some cases, could be a tolerable warning

// UserBuilder methods - return UserBuilder for chaining
public UserBuilder username(String username); // Specify the username (mandatory)
public UserBuilder password(String password); // Specify the password (mandatory)
public UserBuilder roles(String... role);
// Specify the roles with their names (short), the ROLE_ prefix will be added by the method
// You don't have to create GrantedAuthority yourself
public UserDetails build(); // Create the user - The UserBuilder can be reused for another creation
```

Access users

Interface UserDetailsService

Defined the way to retrieve users (**UserDetails**)

```
// Get a user by username - Only method in UserDetailsService
public UserDetails loadUserByUsername(String username);
```

Interface UserDetailsManager (extends UserDetailsService)

Adds the means to create / modify / delete users (**UserDetails**)

```
public void createUser(UserDetails user);
public void deleteUser(String username);
public void updateUser(UserDetails user);
public boolean userExists(String username);
public void changePassword(String oldPassword, String newPassword);
```

User storage

Class InMemoryUserDetailsManager

Class allowing the use of users stored in memory
Implements:

- **UserDetailsManager** (ie. **UserDetailsService**)
- **UserDetailsPasswordService** (modification of password without the previous password)

```
// Method from UserDetailsPasswordService
public void updatePassword(UserDetails user, String password);
```

- **InMemoryUserDetailsManager** is an implementation of **UserDetailsService**
- Other implementations exist
- You can create your own implementation

Introduction to Spring Beans

Bean

Spring Bean

- Java object
- Managed by Spring via Dependency Injection
- Instantiated, configured ... managed by **Spring IoC container**

Annotation @Bean

Method annotation

- Declare a bean
- Indicate how to get it

```
@Bean
public SomeBean getSomeBean() {
        return new SomeBean();
}
```

Configure your Authentication

Example with users in memory

```
// Security configuration class

@Configuration // To indicate this class contains configuration for Spring IoC Container
@EnableWebSecurity  // To indicate we'll have Spring Security configuration in this class
public class MySecurityConfig {

    // Creating some users
    @Bean
    public UserDetailsService userDetailsService() {
        UserBuilder users = Users.withDefaultPasswordEncoder();
        InMemoryUserDetailsManager manager = new InMemoryUserDetailsManager();
        manager.createUser(users.username("hubert").password("OSS117").roles("AGENT").build());
        manager.createUser(users.username("armand").password("blanquette").roles("CHEF").build());
        return manager;
    }
}
```

WILD CODE SCHOOL

Password hashes

Password Encoding

The problem of storing passwords

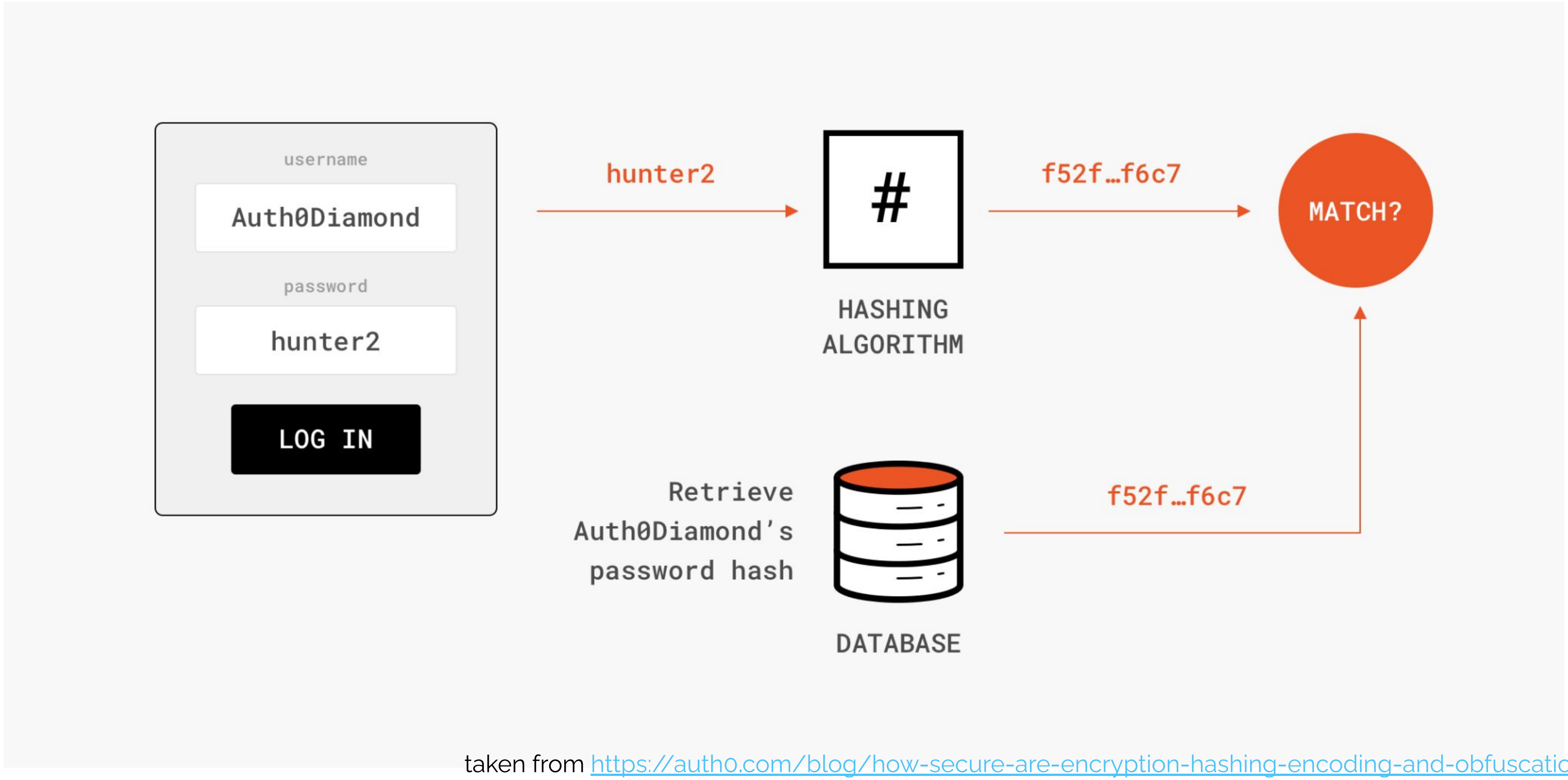
Why passwords?

- Authenticate a user based on a secret
- Known to the user only (compare: knowledge, ownership, inference)

The server must verify this secret…

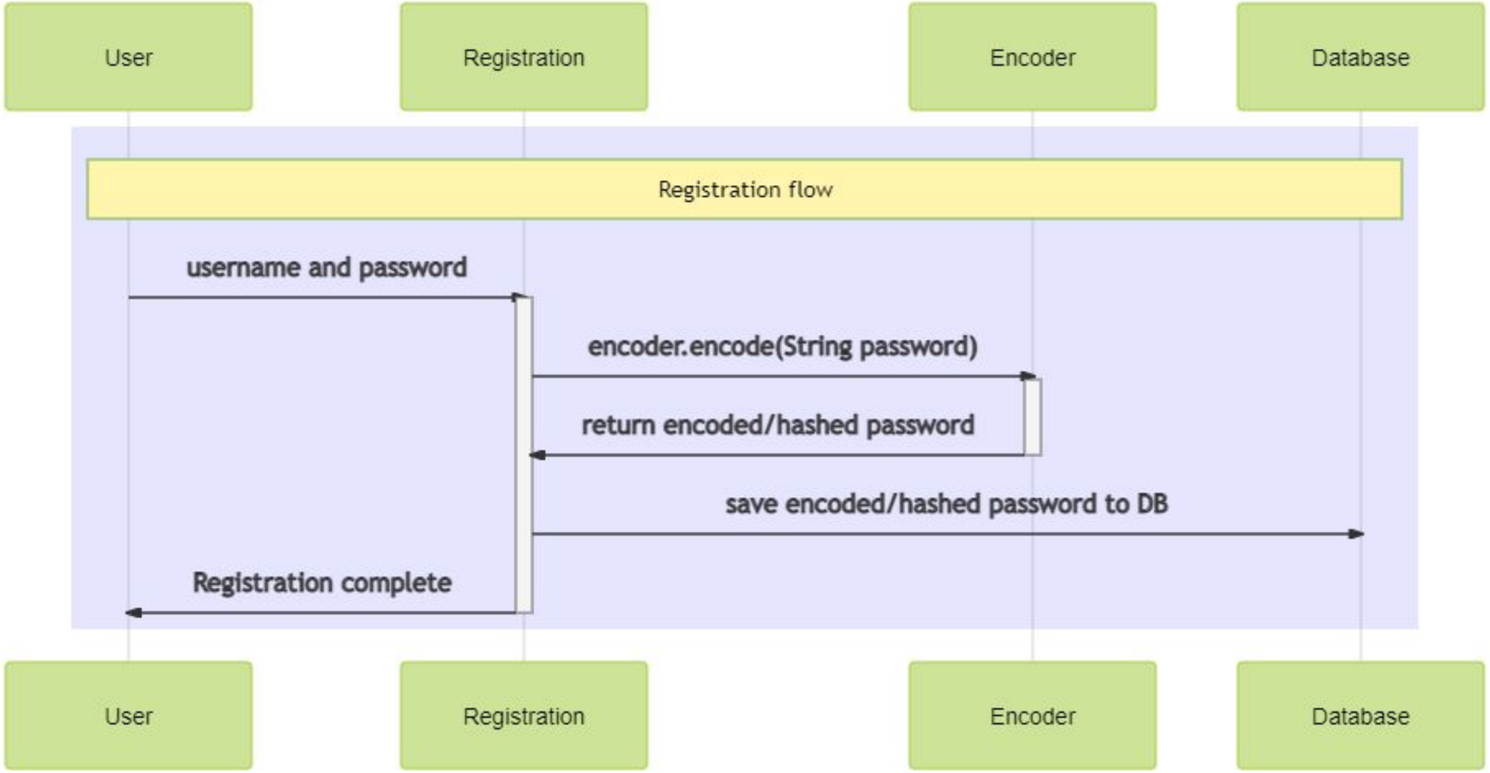
- **Assumption:** the server also knows this secret
- **Problems:**
  - No human intervention possible so the password must be stored.
  - In practice, users re-use their passwords…
- **Solution :**
  - Being able to compare the password without knowing the password

The problem of storing passwords



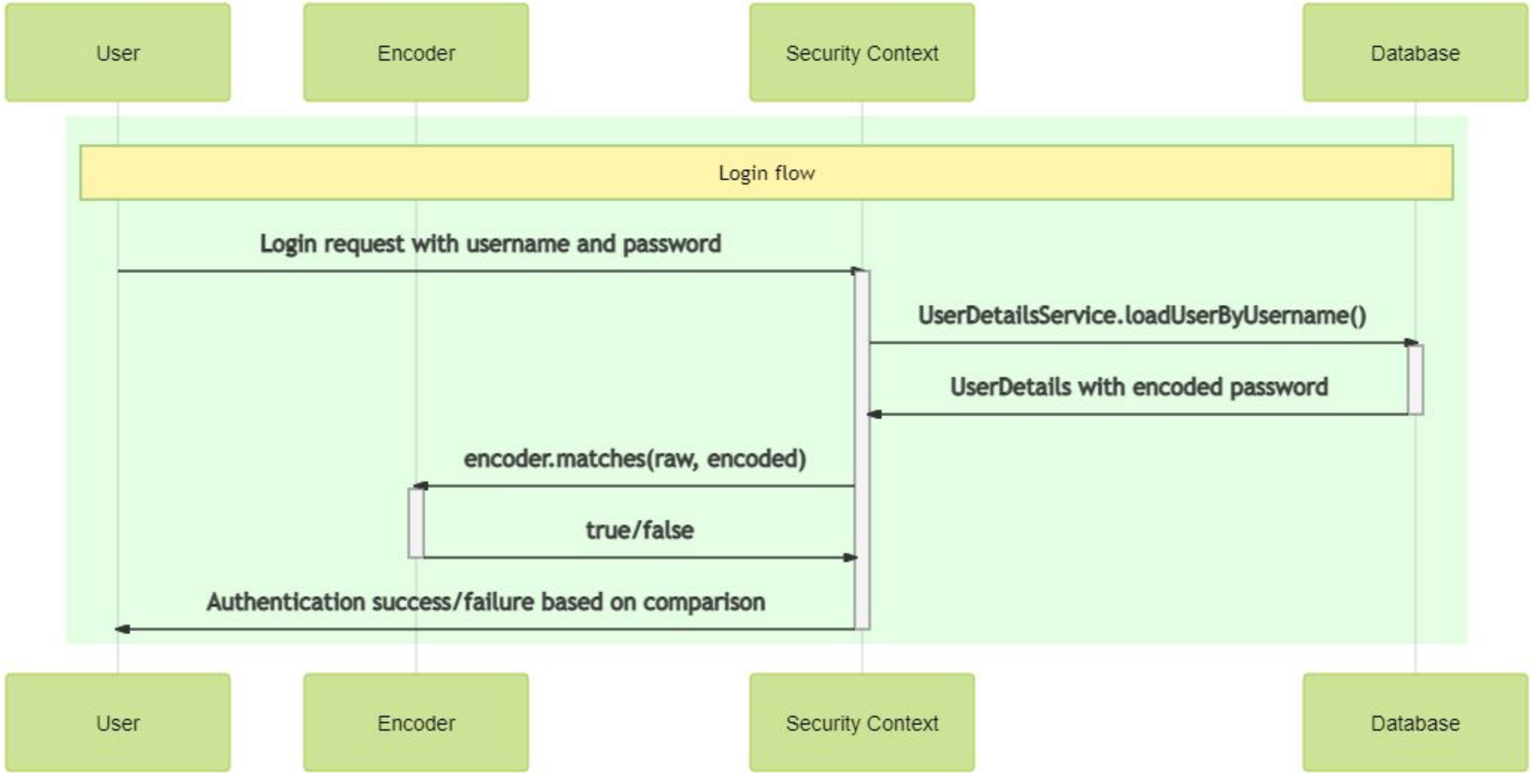
taken from https://auth0.com/blog/how-secure-are-encryption-hashing-encoding-and-obfuscation/

The problem of storing passwords



taken from https://springhow.com/spring-security-password-encoder/

The problem of validating passwords


User
Encoder
Security Context
Database
Login flow
Login request with username and password
UserDetailsService.loadUserByUsername()
UserDetails with encoded password
encoder.matches(raw, encoded)
true/false
Authentication success/failure based on comparison
User
Encoder
Security Context
Database

taken from https://springhow.com/spring-security-password-encoder/

Hash Functions

Hash Functions

- Using one-way function from the password
  - I can calculate the hash from the password
  - I cannot calculate the password
- Comparing the fingerprint is like comparing the message
  - So I can only store the fingerprints

How to create the hash?

- Use proven methods to calculate hash
  - bcrypt, argon2…
- Don't make your own algorithm!!!

Hashes with Spring Security

PasswordEncoder

- Define the means of interacting with passwords
- org.springframework.security.crypto.password.PasswordEncoder

```
// Encode a password (return the password hash)
public String encode(CharSequence rawPassword);
// Verify a password (don't do it with encode() because of the salt)
public boolean matches(CharSequence rawPassword, String passwordHash);
// Should i re-encode the password ? (because default algorithm changed for example)
public boolean upgradeEncoding(String passwordHash);
```

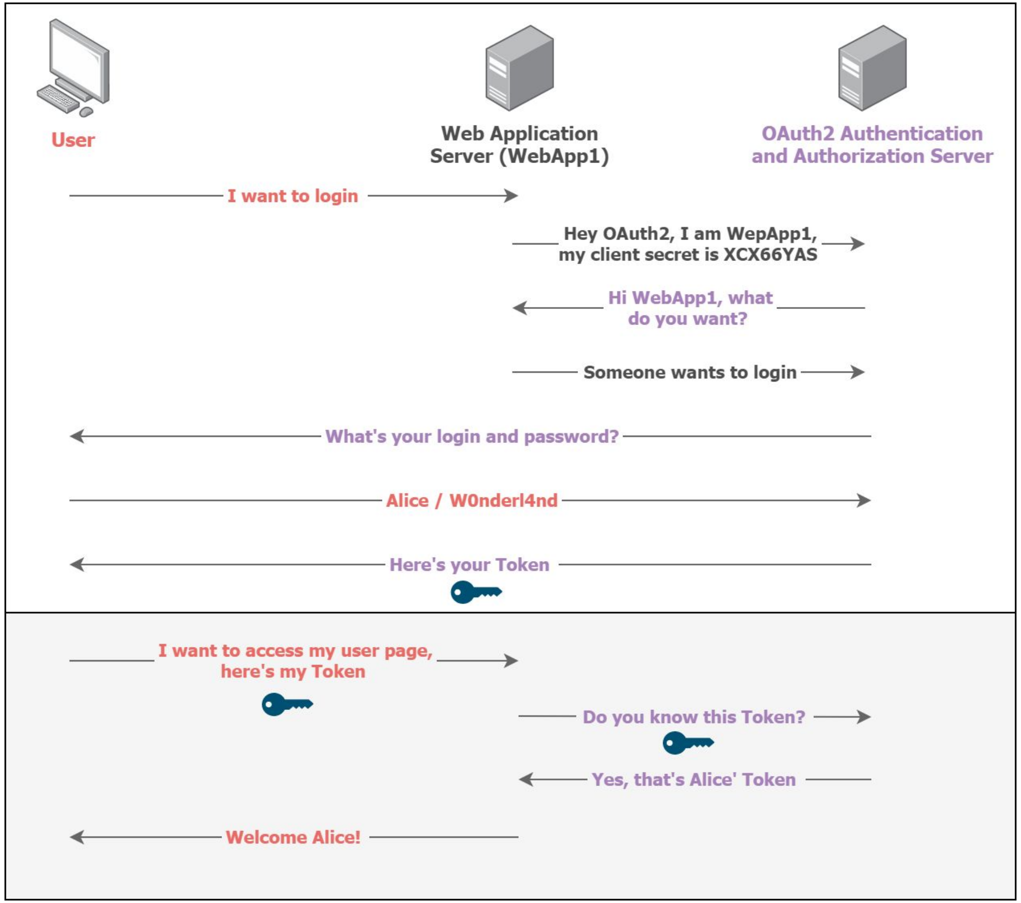
What about the User class?

- The **User.withDefaultPasswordEncoder()** encodes when calling *password(String)*
- Uses the algorithm recommended by Spring

Access Control

Authorization

Spring Security in an OAuth2 Scenario


User
Web Application Server (WebApp1)
OAuth2 Authentication and Authorization Server
I want to login
Hey OAuth2, I am WepApp1, my client secret is XCX66YAS
Hi WebApp1, what do you want?
Someone wants to login
What's your login and password?
Alice / W0nderl4nd
Here's your Token
I want to access my user page, here's my Token
Do you know this Token?
Yes, that's Alice' Token
Welcome Alice!

Spring security in a web application

Synopsis

- A user visits a page protected by access control
- If she is not yet authenticated:
  - Either she is redirected to a particular page (login, do not enter ...)
  - Or she receives an HTTP Status code (403 Access denied)
  - The user can then make an authentication attempt (by providing a login / password for example ...) via HTTP post or an HTTP header
  - The server checks the information provided
    - Valid information: return to the page initially requested
    - Invalid information: return back to the information request
- Otherwise (she is already authenticated)
  - If she has the necessary rights: she accesses the requested page
  - Otherwise: 403 Access Denied

Protected resources?

Access control to what?

- HTTP requests (*Mapping, Method*)
- Method call (Not just controllers)
  - Access control at the business level

Redefine the default configuration

The *WebSecurityConfigurerAdapter*

- Abstract class - Must be inherited
- [WebSecurityConfigurerAdapter]

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

        // Override to configure HttpSecurity
        @Override
        protected void configure(HttpSecurity http) {
                // Add your configuration here
        }

        // You can also override this one to redefine authentication...
         protected void configure(AuthenticationManagerBuilder auth) { }
}
```

Access control by HTTP request

HttpSecurity

- Allows configuration of access control by HTTP requests
- org.springframework.security.config.annotation.web.builders.HttpSecurity

```
// To customize authorization based on HTTP requests
public ExpressionInterceptUrlRegistry authorizeRequest() throws Exception;

// Allow a form based authentication and configure it
public FormLoginConfigurer<HttpSecurity> formLogin() throws Exception;

// Configure HTTP based authentication
public HttpBasicConfigurer<HttpSecurity> httpBasic() throws Exception;

// To customize the logout behaviour
public LogoutConfigurer<HttpSecurity> logout() throws Exception;
```

Definition of access control

ExpressionInterceptUrlRegistry

- Allows you to specify URLs
- ExpressionInterceptUrlRegistry and AuthorizedUrl

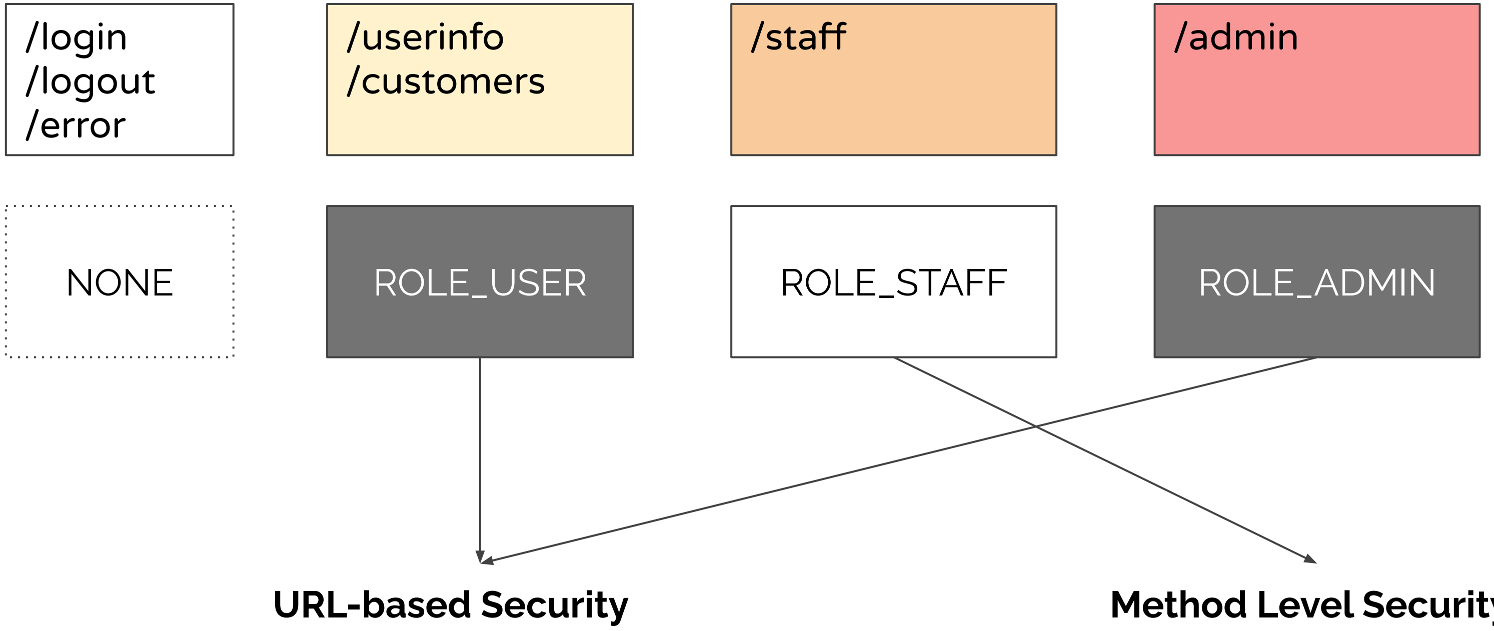
```
http.authorizeRequest() // Get the ExpressionInterceptUrlRegistry
        // Specifying the URLs involved
        .antMatchers(String path...)   // One or several path patterns
        // Or
        .anyRequest()                  // Any request -- evaluated in the specified order
        // Then specifiying the access control (ExpressionUrlAuthorizationConfigurer.AuthorizedUrl)
        .hasRole(String aRole) // Only for user having this particular role
        // Or
        .hasAnyRole(String roles...) // For user of any of the specified roles
        // Or
        .permitAll() // No access control
        // Or
        authenticated() // Any authenticatd user
        // Can be followed by another antMatchers/anyRequest
```

Example of access control

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

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http
            .authorizeRequests()                              // Define access control
                .antMatchers("/").permitAll()
                .antMatchers("/admin").hasRole("ADMIN")
                .antMatchers("/private").hasAnyRole("USER", "ADMIN")
                .anyRequest().authenticated()
            .and()
                .httpBasic() // Add http basic auth with default configuration
            .and()
                .formLogin() // Add a login form with default configuration
            ;
    }
}
```

Configure your Access Control (Webinar Sample)



Note: In our sample, all authenticated Users must have a Role.

WILD CODE SCHOOL

CSRF attacks

Spring security protections

A CSRF attack?

What is it?

- Make a request to an authenticated user
- Cross-Site Request Forgery

How do you protect yourself from it?

- Make sure that the requests (POST request for example) correspond to requests made by the user (via a form generated previously)
- Solution: add a unique and difficult to predict (random) token to the questions (form) and check their presence in the answers

Spring security - CSRF Protection

Creating and Adding a Token

- Creating a view The “th: action” present in the form automatically triggers:
  - The production of a token
  - Adding the token via a hidden form field

Verifying a form

- Upon receipt of a POST request:
  - Spring security automatically checks for the presence of a previously issued and unexpired csrf token

Bibliography

- Spring Security Reference
- Spring Guide : Securing a Web Application
- Spring Security Architecture
- Spring Boot Security auto-configuration
- Intro to Spring Security Expressions

WILD CODE SCHOOL

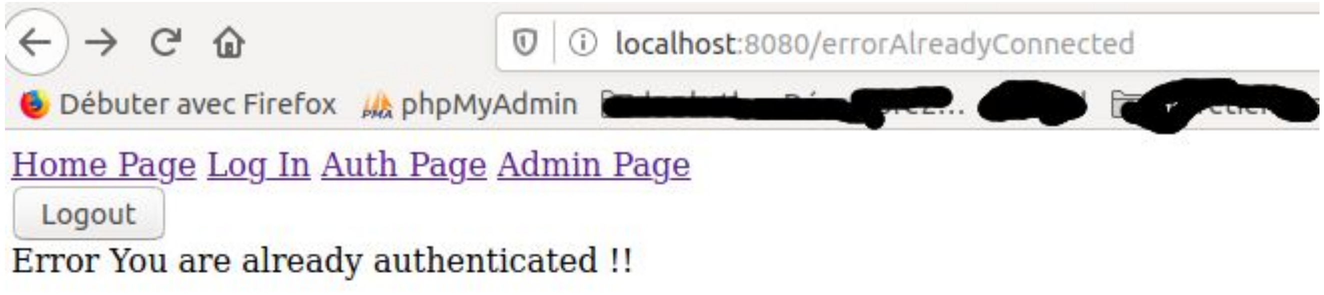
Workshop / Tutorial

Step 1: Setup Spring security project

Creation of a Spring Boot project with dependencies:

- Spring Web
- Thymeleaf
- Spring Data JPA
- Mysql Driver
- Spring Boot DevTools
- Spring Security

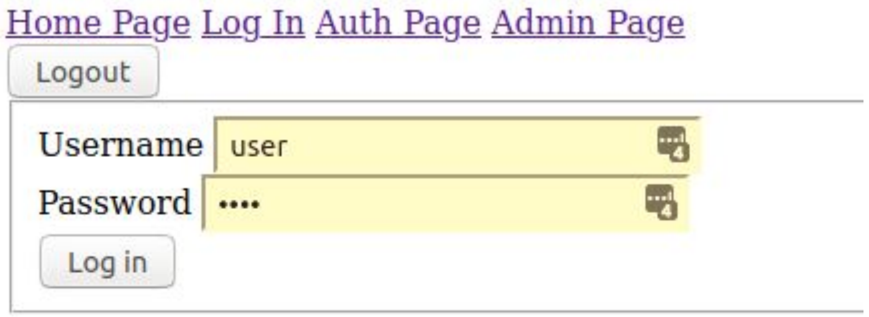
Step 2: Requirements


localhost:8080/errorAlreadyConnected
Débuter avec Firefox
phpMyAdmin
[redacted]
Home Page Log In Auth Page Admin Page
Logout

Error You are already authenticated !!

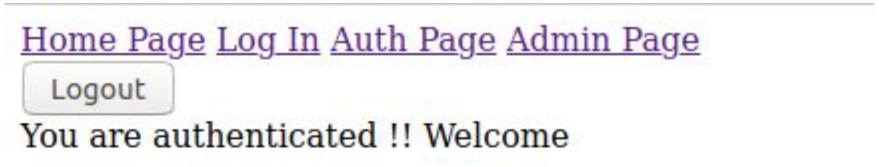
Home Page and Log In: Accessible by everyone.
Authenticated Page: Accessible by any authenticated user
Admin Page : Accessible only through admin. If the user is already logged in and tries to log in a second time they are returned to an error page.

Step 3: Login Page


Home Page Log In Auth Page Admin Page

If the password is correct, the user is redirected to the "auth" page, otherwise they are redirected to the authentication error page.

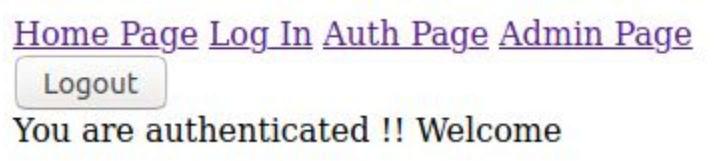
Step 4: Authentication Page


Home Page Log In Auth Page Admin Page

You are authenticated !! Welcome

This page can only be accessed by a logged in user.

Step 5: Admin Page


Home Page Log In Auth Page Admin Page

You are authenticated !! Welcome

This page can only be accessed by a logged in user who has the role of admin.

The Logout button to log out.

Step 6: Setup Demo Environment

Creation of the database:

- name: spring_security_demo
- username : springsercurityadmin
- password : TRFjh24$@2019

Persistence of users and their information.

The password must be encrypted in the DB: (the **BCryptPasswordEncoder** class)

Step 7:

Creating the **User** class that implements the **UserDetails** interface.

Redefining methods (*getAuthorities(), getRoles (), getPassword (), isAccountNonExpired (), ...*)

The definition of possible roles: USER and ADMINISTRATOR in the **RoleEnum** enumeration

Repository:

- Spring Security