

# Java Enterprise Edition (JEE)

## Chapitre 06 : Sécurisation d'applications web avec Spring Security

Chèvre Sébastien

- Sémantique des diapositives



À savoir théoriquement (TE)



Sensibilisation et aspect pratiques

## Contenu du module

- Notions de base – sécurité IT
  - **Identity Access Management**
  - **Vulnérabilités**
- Le framework Spring Security
  - **Authentication**
  - **Autorisation**
- Exercices pratiques

## La sécurité logicielle....



- C'est un métier....



## Spring Security

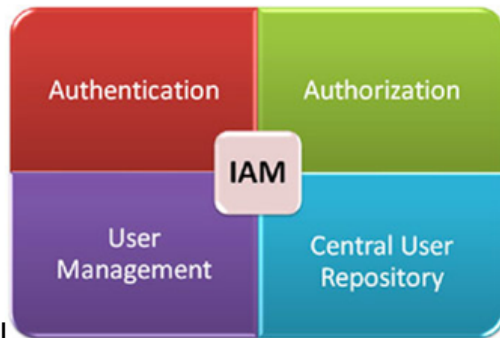
# Identity et Access Management

# Notions de bases – sécurité IT

## *Sujet du cours*



- Le cours va se concentrer sur les aspects de :
  - **IAM – Gestion des accès au sens large**
  - **Sensibilisation vulnérabilités Web**



# Notions de bases – sécurité IT

## *IAM - Identity & Access Management*



### **Role Management**

- Gestion des rôles

### **Authorization**

- Gestion des autorisation

### **Password Management**

- Gestion des mots de passes

### **User Provisionning**

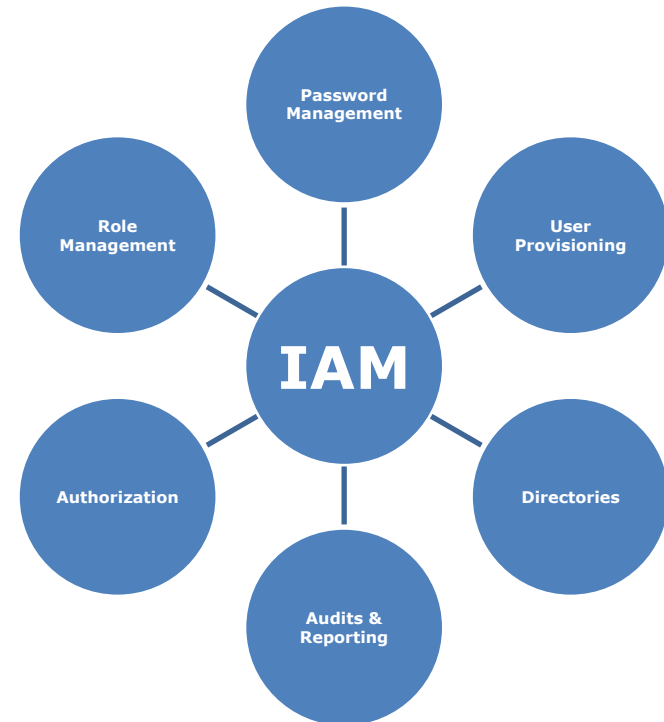
- Gestion des utilisateurs des annuaires

### **Directories**

- Structure des répertoires des annuaires

### **Audit & reporting**

- Gestion de la trace des accès (qui a fait quoi, quand?)



# Notions de bases – sécurité IT

*Identification/Authentication/Autorisation*



- **Identification**
  - *Je suis « Toto »*
- **Authentication**
  - *Je suis « Toto » et mon mot de passe est « Tutu »*
  - Garantie de l'identification
- **Autorisation**
  - *Je suis « Toto » et j'ai accès à des ressources*

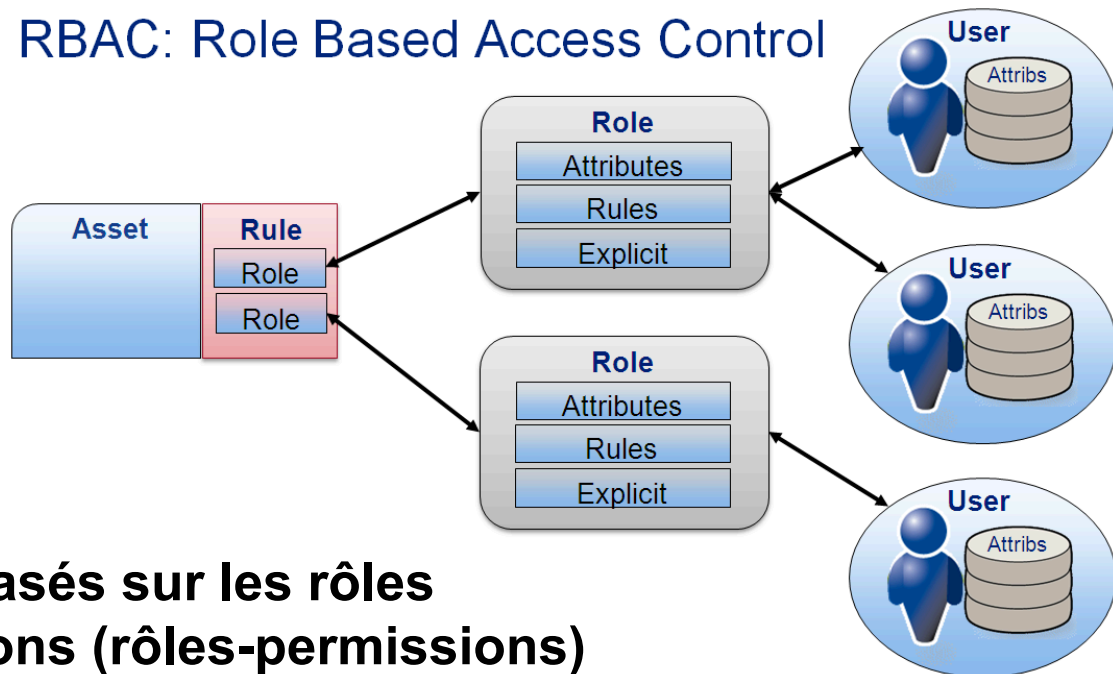


# Notions de bases – sécurité IT

## Modèle d'accès RBAC – Role Based Access Control



### RBAC: Role Based Access Control



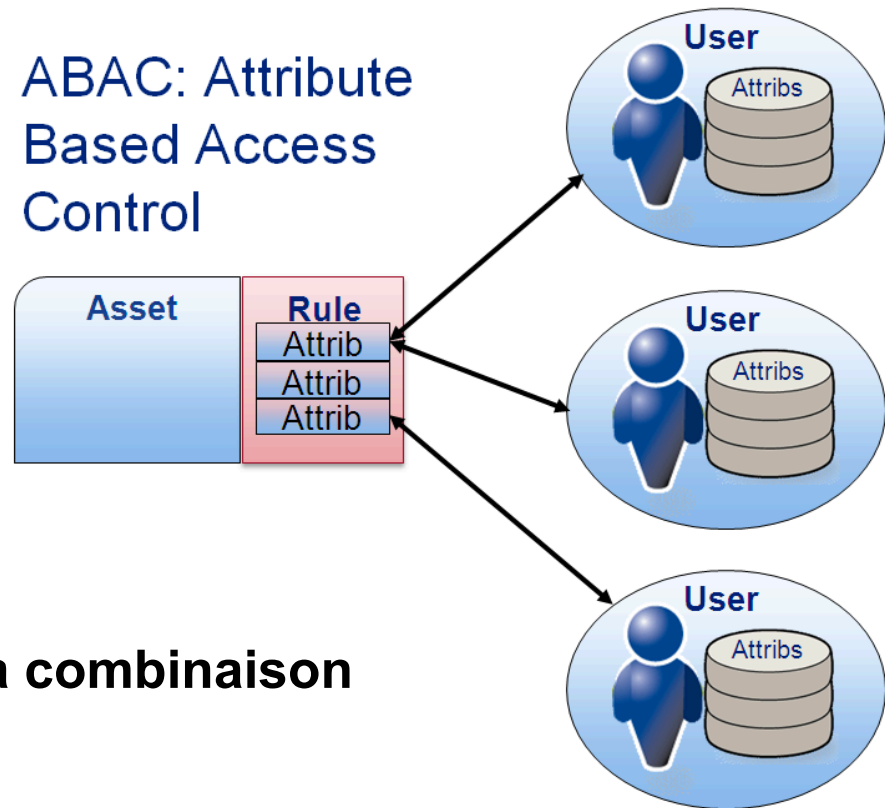
- Les accès sont basés sur les rôles
- En deux dimensions (rôles-permissions)
- Moins flexible à partir d'un certain volume

# Notions de bases – sécurité IT

## Modèle d'accès ABAC – Attributes Based Access Control



ABAC: Attribute  
Based Access  
Control



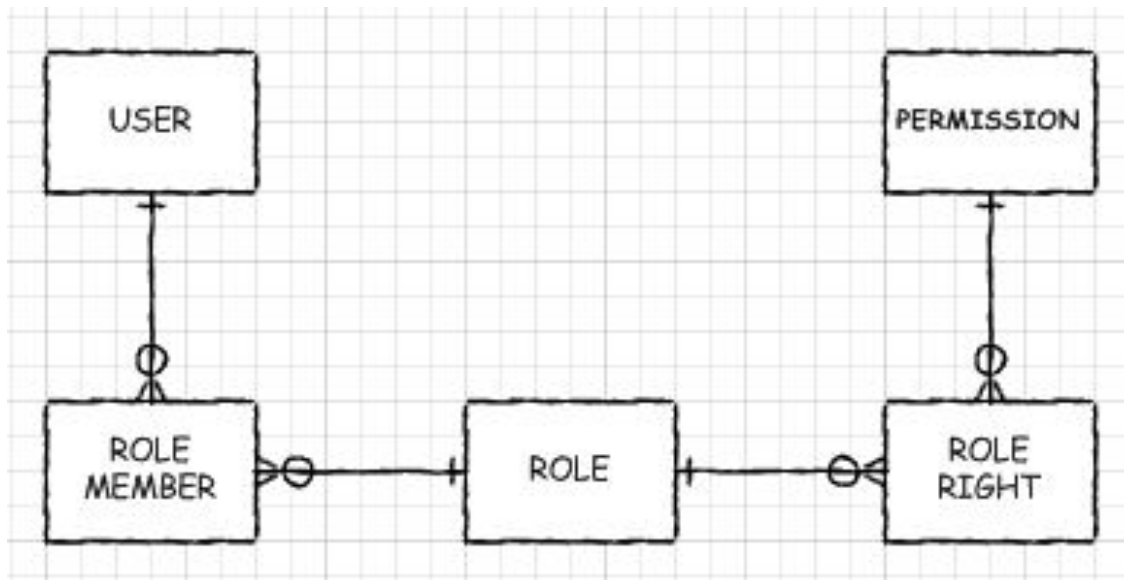
- Les accès sont basés sur la combinaison d'attributs
- Multidimensionnel
- Plus flexible, plus complexe

# Notions de bases – sécurité IT

## Modèle d'accès RBAC modélisation



- Exemple modélisation typique RBAC



# Notions de bases – sécurité IT

## Autorisations et niveaux d'autorisations



- **Niveau Groupe**

- Généralement au niveau d'un ISP (Identity Service Provider)

- **Niveau Rôles**

- Au niveau de l'application

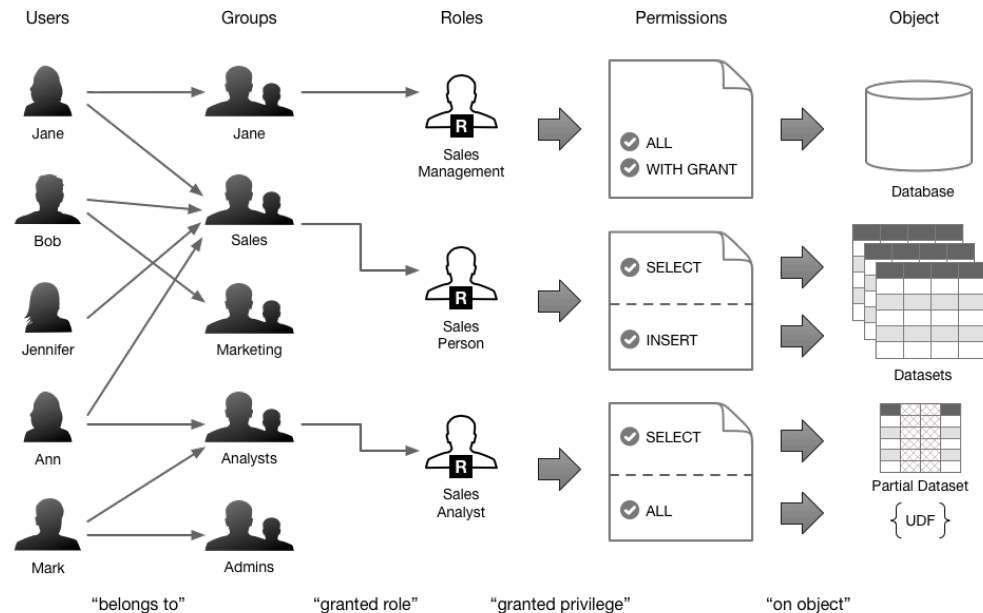
- **Niveau Permissions**

- Au niveau de l'application

- **Niveau ACL**

- Acknowledge Control List

- Au niveau des entités



## Spring Security

# Sensibilisation vulnérabilités Web

# Notions de bases – sécurité IT

## *Sensibilisation vulnérabilité*



- Une vulnérabilité est une **potentielle faille exploitable par un attaquant**
- Aujourd'hui, les attaquants ont recours à **deux méthodes** afin de pouvoir cibler une attaque:
- **Ingénierie technique**  
Implémentation technique permettant d'exploiter une faille (code, scripts, etc...)
- **Ingénierie sociale**  
Eléments nécessaires à l'ingénierie technique afin de pouvoir exploiter une faille (email, mot de passe, n° de compte, informations réseaux sociaux, etc...)

## Notions de bases – sécurité IT

### *Sensibilisation vulnérabilité*



- Le temps des hackers au fond d'une cave est révolue
- Les hackers d'aujourd'hui font partie de groupes et utilisent des techniques d'ingénierie sociale basé sur le principal maillon faible en terme de sécurité IT:

# L'humain

# Notions de bases – sécurité IT

## *Sensibilisation vulnérabilité*



## L'UTILISATEUR : LE MAILLON FAIBLE ?

Pour stocker, partager ou enregistrer des données ou des informations d'identification, **les employés privilégient la commodité personnelle** par rapport aux protocoles de sécurité.

En **ignorant les meilleures pratiques** en matière de sécurité des données, les employés de bureau s'**exposent eux-mêmes et leurs employeurs à de grands risques**.



# Notions de bases – sécurité IT

## *Sensibilisation vulnérabilité - faille connues 1*



- Failles de sécurité Web les plus connues:
  - **Injectons SQL**  
*Possibilité d'injecter du code sql dans l'application via des formulaires non sécurisées*
  - **Cross-Site Scripting (XSS)**  
*Possibilité d'injecter du code (script) dans l'application via des paramètres de requêtes ou des formulaires non sécurisées*
  - **Authentification cassé et gestion de session**  
*Utilisation d'informations afin de détourner une authentification et de l'utiliser à des fins malveillantes*

# Notions de bases – sécurité IT

## *Sensibilisation vulnérabilité - faille connues 2*



- Failles de sécurité Web les plus connues:
  - **Références sur des objets non sécurisés**  
*Utilisation d'informations fournies dans des objets (fichiers, images, etc...) non sécurisées a des fins malveillantes*
  - **Configuration de la sécurité non fiable**  
*La configuration de la sécurité n'est pas complète, ou erronée ce qui aboutit a des failles de sécurité exploitables*
  - **Cross-Site Request Forgery**  
*Exécution par un utilisateur authentifié et à son insu d'une action à but malveillant*

# Notions de bases – sécurité IT

*Sensibilisation vulnérabilité – référentiel et normes*



- Normes et référentiels de sécurité:
  - **OWASP**  
*Open Web Application Security Project*
  - **ISO/CEI 27001, ISO/CEI 27034**



## OWASP Top 10 - 2017

A1:2017-Injection
A2:2017-Broken Authentication
A3:2017-Sensitive Data Exposure
A4:2017-XML External Entities (XXE)
A5:2017-Broken Access Control
A6:2017-Security Misconfiguration
A7:2017-Cross-Site Scripting (XSS)
A8:2017-Insecure Deserialization
A9:2017-Using Components with Known Vulnerabilities
A10:2017-Insufficient Logging & Monitoring

# Notions de bases – sécurité IT

## Sensibilisation vulnérabilité – CSRF/XSRF



- Cross Site Request Forgery



## Notions de bases – sécurité IT

*CORS*



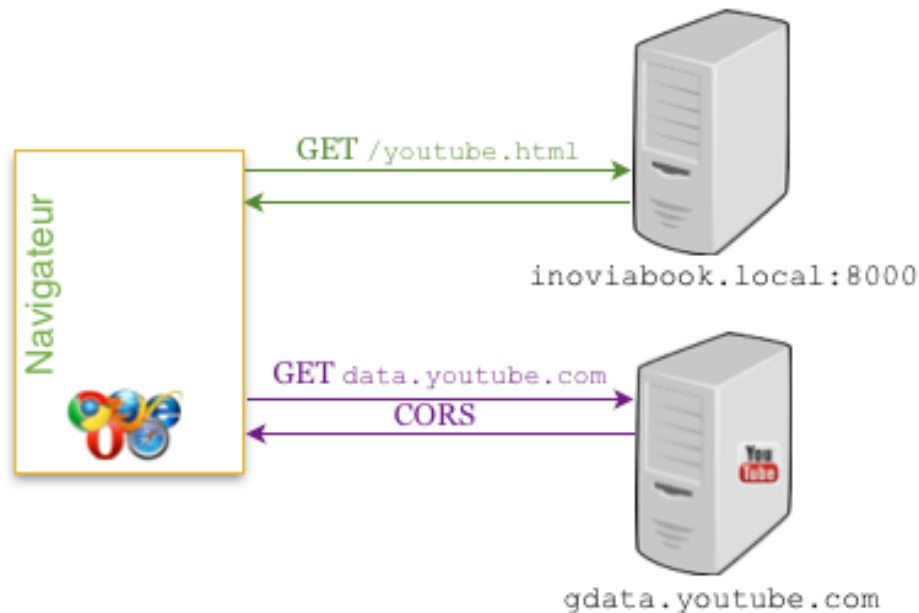
- **Cross Origin Request Sharing**
- **CORS** est un mécanisme robuste et normalisé par le W3C permettant à tout navigateur compatible d'effectuer des **requêtes HTTP *cross-domain***
- Un domaine (ou origine) est la combinaison de l'hôte et du port
- C'est le travail du navigateur !

# Notions de bases – sécurité IT

## *CORS - Exemple*



- Une page affiche la liste de vidéo YouTube via un script Ajax



# Notions de bases – sécurité IT

## CORS - Exemple



- La requête contient un champ **origin** dans les header
- Pour des raisons de sécurité, seul le navigateur est responsable de la valorisation de cet entête HTTP

### ▼ Request Headers

```
:authority: gdata.youtube.com
:method: GET
:path: /feeds/api/standardfeeds/most_popular?v=2&alt=json
:scheme: https
accept: application/json, text/javascript, */*; q=0.01
accept-encoding: gzip, deflate, sdch
accept-language: fr-FR,fr;q=0.8,en-US;q=0.6,en;q=0.4
cache-control: no-cache
origin: http://inoviabook.local:8000
pragma: no-cache
referer: http://inoviabook.local:8000/youtube.html
user-agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_4)
```

- La réponse du serveur contient l'entête **access-Control-Allow-Origin** permettant d'indiquer au client qu'il est bien habilité à obtenir une réponse valide
- La requête échouerait si l'entête n'était pas présent ou s'il contenait une valeur ne concordant pas avec la valeur de l'entête **origin**.

### ▼ Response Headers

```
access-control-allow-origin: http://inoviabook.local:8000
access-control-expose-headers: Cache-Control,Content-Encoding,
cache-control: private, max-age=300, no-transform
content-encoding: gzip
content-type: application/json; charset=UTF-8
date: Wed, 15 Apr 2015 14:58:58 GMT
etag: W/"A04HSXg7cCp7I2A9XRRaEU0."
expires: Wed, 15 Apr 2015 14:58:58 GMT
```

# Notions de bases – sécurité IT

## *CORS – Pre Flight Request*



- Le mécanisme Pre Flight est activé dans ces conditions:
  - La requête utilise une méthode HTTP différente de **GET, POST ou HEAD**
  - la requête contient un entête HTTP **Content-Type** dont la valeur est différente de :
    - text/plain
    - multipart/form-data
    - application/x-www-form-urlencoded



# Notions de bases – sécurité IT

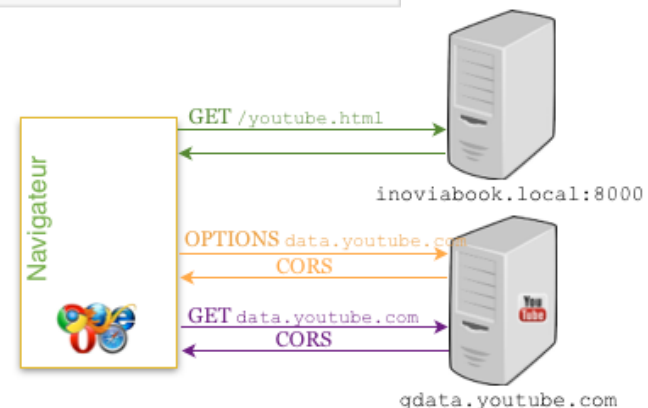
## CORS – Pre Flight Request



- La requête **GET** est envoyé via le script
- Le navigateur envoie une requête **OPTIONS**
  - *Il demande la permission au serveur*

```
OPTIONS /feeds/api/standardfeeds/most_popular?v=2&alt=json HTTP/1.1
Host: gdata.youtube.com
origin: http://inoviabook.local:8000
access-control-request-method: GET
access-control-request-headers: accept, content-type
...
```

- Le serveur retourne la réponse



# Notions de bases – sécurité IT

## *Sensibilisation vulnérabilité*



- Bonnes pratiques
  - N'implémenter pas de sécurité « maison »
  - Utilisez des frameworks
  - En cas de configuration « custom » → impératif de maîtriser les concepts
  - Effectuez des tests de sécurité (pen tests, scannage de failles, etc...)
    - → **OWASP**

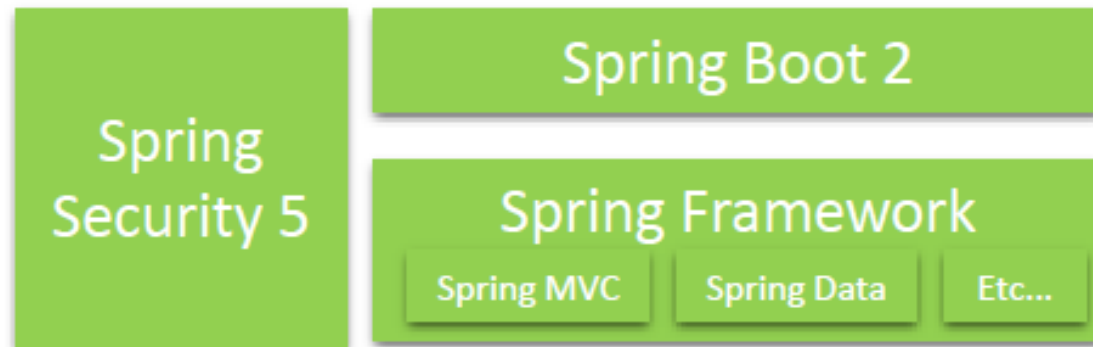
## Spring Security

# Spring Security

# Spring Security



- ***Spring Security*** est une extension de *Spring* qui fournit des services d'authentification, d'autorisation et de sécurité pour les applications d'entreprise..



## Spring Security



*L'ajout des dépendances de **Spring Security** amènent **automatiquement** les protections suivantes:*

- Protections de bases des principales vulnérabilités web
- Protection des headers http
- Génération automatique de formulaire de login

# Spring Security

## *Notions de base – Building blocks*



- **Principal**
  - User that performs the action
- **Authentication**
  - Confirming truth of credentials
- **Authorization**
  - Define access policy for principal
- **GrantedAuthority**
  - Application permission granted to a principal
- **SecurityContext**
  - Hold the authentication and other security information
- **SecurityContextHolder**
  - Provides access to SecurityContext
- **AuthenticationManager**
  - Controller in the authentication process
- **AuthenticationProvider**
  - Interface that maps to a data store which stores your user data.
- **Authentication Object**
  - Object is created upon authentication, which holds the login credentials.
- **UserDetails**
  - Data object which contains the user credentials, but also the Roles of the user.
- **UserDetailsService**
  - Collects the user credentials, authorities(roles) and build an UserDetails object.

# Spring Security

## Architecture



### AuthenticationManager

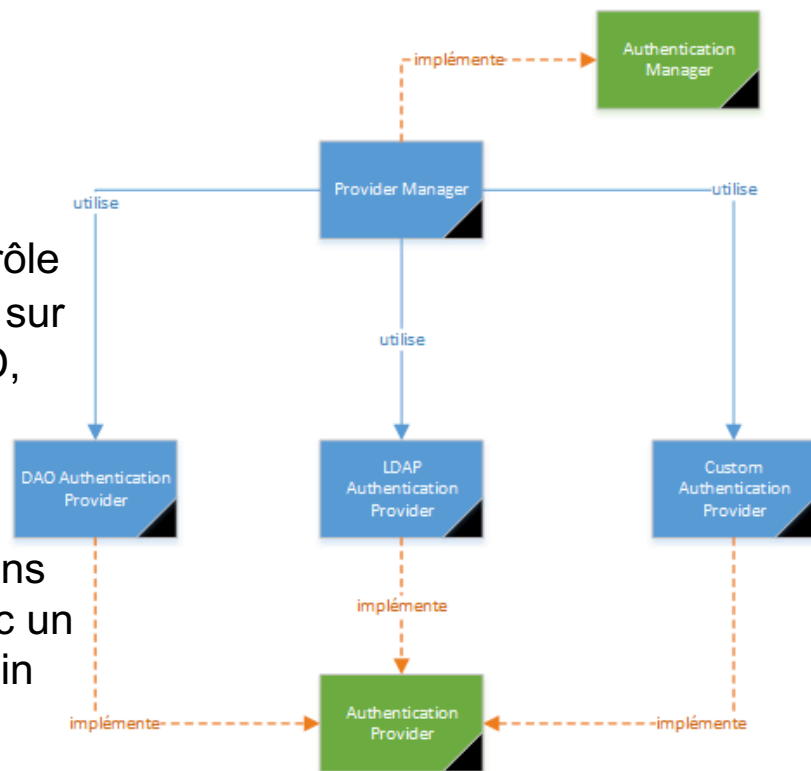
C'est le point d'entrée du processus d'authentification de Spring Security.  
Méthode ***authenticate()*** à implémenter

### ProviderManager

Implémentation de l'interface ***AuthenticationManager***, son rôle est de réaliser une authentification basée sur un ou plusieurs « providers » (LDAP, BDD, etc.)

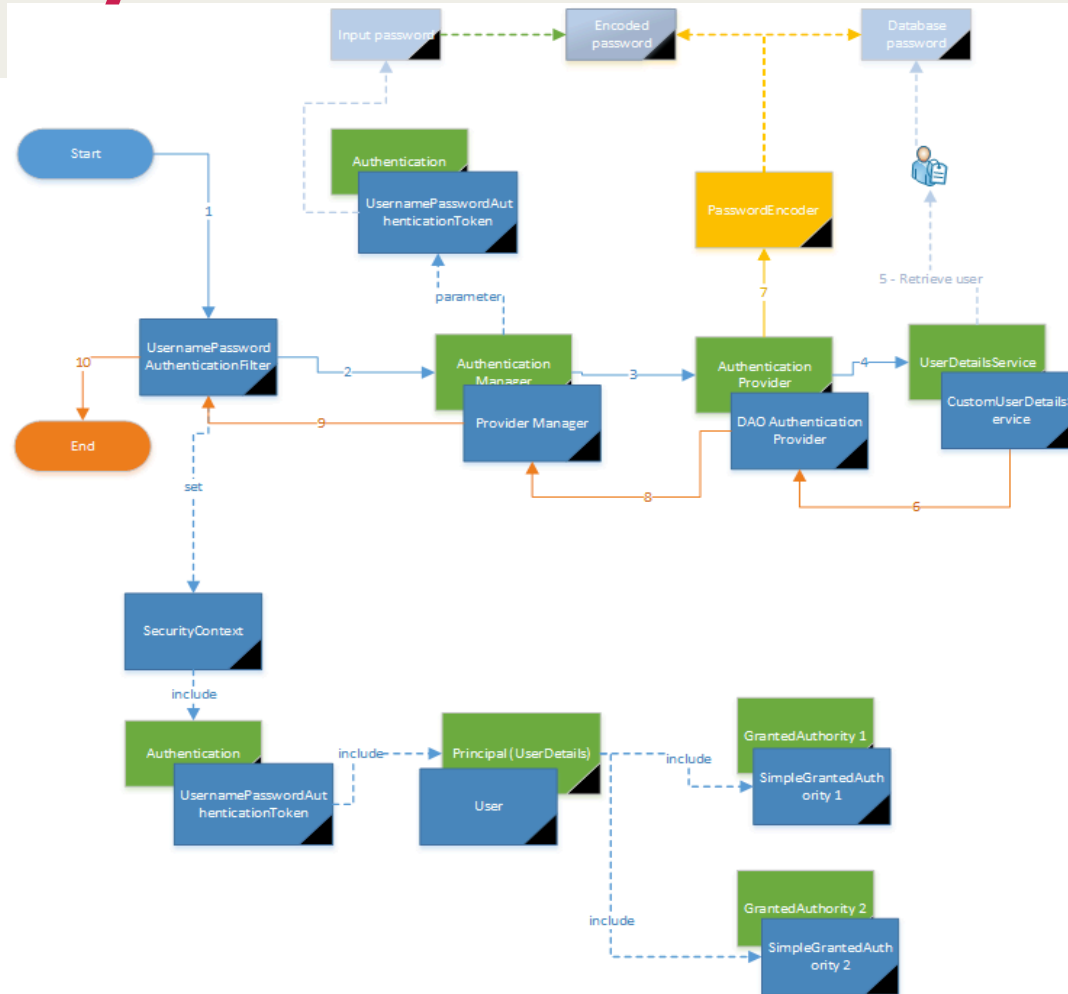
### Authentication Provider

Interface dont les multiples implémentations définissent l'authentification concrète avec un tiers (LDAP, BDD, etc..) ou localement (« in memory » par exemple)



# Spring Security

## "The big Picture"





# Spring Security

## *AuthenticationManagerBuilder*



- **In-Memory Authentication**
  - *User/passwd sont écrits en “dur” dans le code*
- **JDBC Authentication**
  - *User/Passwd sont dans une base de donnée*
- **LDAP Authentication**
  - *User/Passwd sont dans un LDAP*
- **UserDetailsService**
  - *User/Passwd sont fournis par une entité User et l’implementation d’un service (contrat)*
- **AuthenticationProvider**
  - *Personnalisé*

# Spring Security

## *Encryption du mot de passe*



- Un mot de passe **DOIT** être encrypté
- Spring supporte plusieurs mécanismes d'encryptage
- Dans ce cours nous utiliserons **Bcrypt**
- **Eviter les encodeurs dépréciés**
  - *MD5PasswordEncoder*
  - *ShaPasswordEncoder*

```
@Autowired
BCryptPasswordEncoder bCryptPasswordEncoder;

@Bean
public BCryptPasswordEncoder bCryptPasswordEncoder() {
    return new BCryptPasswordEncoder();
}
```

## Spring Security

# Exercices pratique sécurisation application Spring

# Spring Security

Wiki Github:

<https://github.com/sebChevre/cours-JEE-2019-2020/wiki/S%C3%A9curisation-de-l'application-beershop>

- Exercices pratiques
  - Mise en pratique JVM et Garbage Collector
  - Première Servlet JEE
  - Ajout requêtes BeerShop
  - Bean, Component, Controller, IOC et profiles
  - Spring Data JPA
  - Sécurisation beershop

## Spring Security

# Autorisation

# Autorisation

## *Sécurisation thymeleaf-produit*



- **Role**

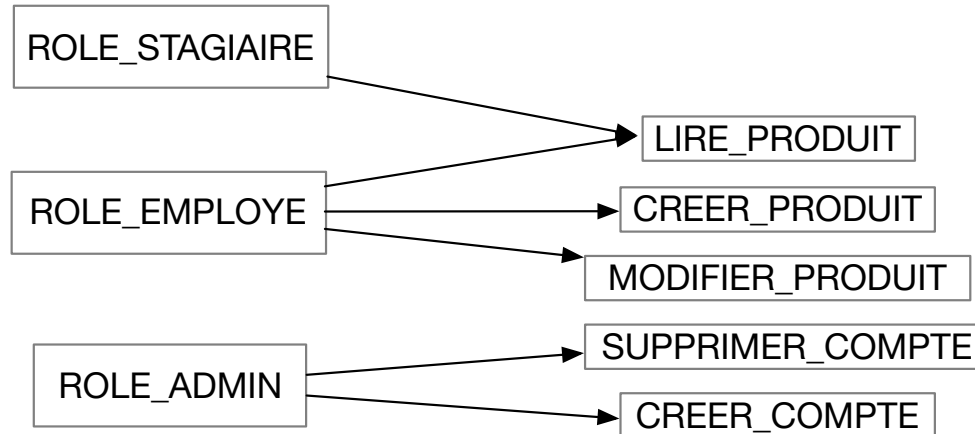
- Chaine de caractère libre de sens
- Utilisé pour définir un rôle applicatif
- Préfixé par ROLE\_
- Ex: ROLE\_ADMIN, ROLE\_STAGIARE
- Souvent utilisé en tant que container de permission

- **Permission/granted authority/privilège**

- Chaine de caractère libre de sens
- Utilisé pour définir une permission bas-niveau
  - P.ex. les permission sont définis au niveau des controlleurs, et la permission peut être utilisé dans plusieurs rôles

# Autorisation

## Bonnes pratiques



- Grouper des permission dans des rôles permet un niveau plus fin de permission
- Cela laisse la possibilité au client de créer ses propres rôles qu'il composera avec le catalogue des permissions applicative

# Autorisation

## Mise en place



- Activation des mécanismes d'autorisation via annotations:

```
@EnableGlobalMethodSecurity(  
    prePostEnabled = true,  
    securedEnabled = true,  
    jsr250Enabled = true)
```

- **prePostEnabled**: activation des annotation pre/post
- **securedEnabled**: utilisation de la balise @Secured
- **jsr250Enabled**: @RoleAllowed



# Autorisation

## *Roles et authority*



- Autorisation au niveau des API
  - @PreAuthorize("hasRole('ROLE\_ADMIN')")
  - @PreAuthorize("hasAuthority('LIRE\_PRODUIT')")
  - @Annotation([Expression Spel])
    - Spring expression language
  - **Peut être utilisée sur n'importe quelle méthode!**
- Autorisation via la configuration globale
  - antMatchers().hasRole(({ "ROLE\_ADMIN", "ROLE\_STAGIAIRE" })
  - antMatchers().hasAuthority("LIRE\_PRODUIT")

# Autorisation

## *Roles et authority*



- **@Secured, @RolesAllowed**

- Permet de spécifier un rôle ou une collections de rôles
- Ne supporte pas les expressions Spel
- @RolesAllowed conforme à la norme JSR250
- Identique en terme de fonctionnalité

```
@RolesAllowed({ "ROLE_VIEWER", "ROLE_EDITOR" })
public boolean isValidUsername2(String username) {
    //...
}

@Secured("ROLE_VIEWER")
public String getUsername() {
    SecurityContext securityContext = SecurityContextHolder.getContext();
    return securityContext.getAuthentication().getName();
}
```

# Autorisation

## Exemples



```
@PreAuthorize("hasRole('ROLE_VIEWER') or hasRole('ROLE_EDITOR')»)

@PreAuthorize("#username == authentication.principal.username")
public String getMyRoles(String username) {
    //...
}

@Secured("ROLE_VIEWER")
public String getUsername() {
    SecurityContext securityContext = SecurityContextHolder.getContext();
    return securityContext.getAuthentication().getName();
}
```

- Dans le cas d'un évaluateur de permissions custom:

```
@PreAuthorize("hasPermission(returnObject, 'read')")
```

## Aller plus loin



- OWASP
  - [https://www.owasp.org/index.php/Main\\_Page](https://www.owasp.org/index.php/Main_Page)
- Formulaire de login/logout custom
  - <https://docs.spring.io/spring-security/site/docs/current/guides/html5/form-javaconfig.html>
- Intégration avec thymeleaf
  - <https://www.thymeleaf.org/doc/articles/springsecurity.html>
- JSR 250
  - [https://en.wikipedia.org/wiki/JSR\\_250](https://en.wikipedia.org/wiki/JSR_250)
- Spring Expression Language (Spel) et Contrôle d'accès
  - <https://www.baeldung.com/spring-expression-language>
  - <https://docs.spring.io/spring/docs/4.2.x/spring-framework-reference/html/expressions.html>
  - <https://docs.spring.io/spring-security/site/docs/3.0.x/reference/el-access.html>
  - <https://www.baeldung.com/spring-security-create-new-custom-security-expression>