

# Authentification et contrôle d'accès (ACA)

## Plan

- Authentification et contrôle d'accès vs sécurité
- Les niveaux de contrôle d'accès
- ACA gérée par la base de données
- ACA gérée par le serveur applicatif
- ACA gérée par l'application

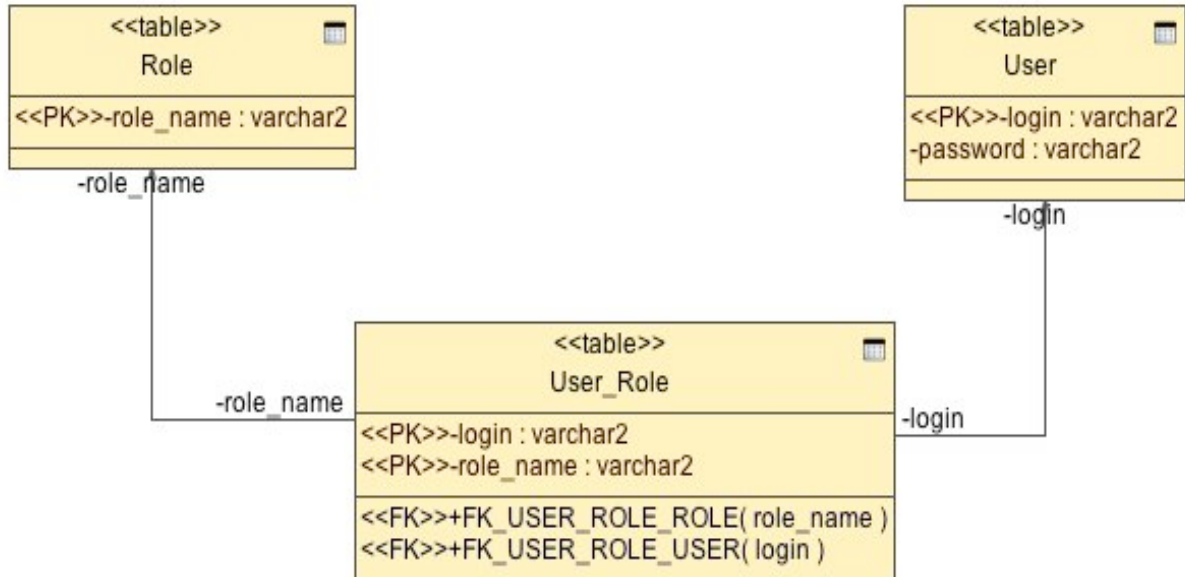
# ACA vs sécurité

- sécurité englobe l'ACA
- Sécurité doit garantir aux composants « physiques » :
  - la disponibilité
    - attaque
    - montée en charge
  - l'intégrité
  - ...

# Authentification et contrôle d'accès

- Authentification :
  - identifier la personne qui se connecte au SI
  - déterminer ces droits d'accès
  - déterminer ces rôles
- Contrôle d'accès :
  - garantir un accès aux ressources basé sur l'indentification et les droits d'accès

# Exemple de modèle d'authentification



## Exemple de contrôle d'accès

- Seule les personnes ayant le droits d'accès **CREATE TABLE** peuvent créer des tables dans la base de données
- Seule les personnes ayant le role **Admin** peuvent accéder à la resource `infoEtudiant.jsp`

# Les niveaux de contrôle d'accès

- Contrôle d'accès aux ressources :
  - tables/classes
  - traitements
  - objets
  - interfaces
- Quelques primitives d'accès aux ressources :
  - ajouter
  - mettre à jours
  - supprimer
  - lire
  - lister

## ACA gérée par la base de données

- Oracle fournit une gestion des droits assez fine au niveau :
  - des ressources :
    - Table
    - Objet (raffinement via view)
  - des primitives :
    - INSERT, DELETE, DROP, CREATE,...
- Nécessite d'utiliser la gestion d'authentification d'oracle
- Gère que les ressources qui sont dans oracle
- authentification peu paramétrable
- nécessite une connexion par utilisateur

# ACA gérée par le serveur applicatif

- Serveur app offre des mécanismes pour l'authentification et le contrôle d'accès.
- authentification peu paramétrable
- Contrôle d'accès assez simple :
  - basé sur les rôles :
  - binaire (accès ou refus)
  - granularité : répertoire, page, image,...

# ACA gérée par l'application

- L'authentification est entièrement définissable
- Permet de gérer tout les niveaux de contrôle d'accès
- Grande liberté
- Nécessite de tout codé
- Peut vite engendrer un modèle de contrôle d'accès compliquer et peu flexible.

# ACA dans Tomcat

- Règles de contrôle d'accès
- Authentification

## Règles de contrôle d'accès

```
<security-constraint>
  <web-resource-collection>
    <web-resource-name>Protected Area</web-resource-name>
    <url-pattern>/person</url-pattern>
    <http-method>DELETE</http-method>
    <http-method>GET</http-method>
    <http-method>POST</http-method>
    <http-method>PUT</http-method>
  </web-resource-collection>
  <auth-constraint>
    <role-name>Person</role-name>
    <role-name>Admin</role-name>
  </auth-constraint>
  <user-data-constraint>
    <transport-guarantee>NONE</transport-guarantee>
  </user-data-constraint>
</security-constraint>
```

# Authentication

```
<login-config>  
  <auth-method>BASIC</auth-method>  
  <realm-name>Myapp authentication</realm-name>  
</login-config>
```

- Ou

```
<login-config>  
<auth-method>FORM</auth-method>  
  <form-login-config>  
    <form-login-page>/login.jsp</form-login-page>  
    <form-error-page>/errorLogin.jsp</form-error-page>  
  </form-login-config>  
</login-config>
```

## Authentication (suite)

- Realm :
  - MemoryRealm
  - JDBCRealm
  - DataSourceRealm
  - JAASRealm

# MemoryRealm

- \$CATALINA\_HOME/conf/server.xml
- ou dans META-INF/context.xml

```
<context path="/MyApp">
  <Realm className="org.apache.catalina.realm.MemoryRealm" />
</context>
```
- \$CATALINA\_HOME/conf/tomcat-users.xml

```
<tomcat-users>
  <role rolename="tomcat"/>
  <role rolename="role1"/>
  <role rolename="manager"/>
  <role rolename="admin"/>
  <user username="tomcat" password="tomcat" roles="tomcat"/>
  <user username="both" password="tomcat" roles="tomcat,role1"/>
  <user username="role1" password="tomcat" roles="role1"/>
  <user username="admin" password="tomAdmin"
    roles="admin,manager"/>
</tomcat-users>
```

# JDBCRealm

```
<Context path="/MyApp">
  <Realm className="org.apache.catalina.realm.JDBCRealm" debug="99"
    driverName="oracle.jdbc.driver.OracleDriver"
    connectionURL="jdbc:oracle:thin:@cuisuna.unige.ch:1521:cuiprd"
    connectionName= »bda2004_x" connectionPassword= " bda_x"
    userTable= »User" userNameCol="login" userCredCol="password"
    userRoleTable= »User_Role" roleNameCol="role_name"/>
</Context>
```

# DataSourceRealm

```
<Context path="/MyApp">  
  <Realm className="org.apache.catalina.realm.DataSourceRealm"  
    debug="99"  
    dataSourceName="jdbc/authority"  
    userTable="users" userNameCol="user_name"  
    userCredCol="user_pass"  
    userRoleTable="user_roles" roleNameCol="role_name"/>  
</Context>
```

## MemoryRealm vs JDBCRealm

- MemoryRealm :
  - gestion des utilisateurs et rôle difficile car nécessite un redémarrage de tomcat.
  - pas de gestion dynamique
- JDBCRealm
  - permet une gestion plus souple des utilisateurs et rôle
  - peut de flexible au niveau de l'authentification

# JAASRealm

- Baser sur JAAS de java (Java Authentication and Authorization Service (JAAS))
- Offre des possibilités très poussées pour la gestion de l'authentification.
- Permet d'interfacer l'authentification avec plusieurs sources (NIS, JDBC,...).
- Puissant mais très complexe.

## Realm dans les jsp

- Obtenir le login après une Realm authentification :  
`request.getRemoteUser();`
- Savoir si l'utilisateur a un rôle donné :  
`boolean request.isUserInRole(String role);`