Magazine
Java Authentication and Authorization Service

Óbuda University, Java Enterprise Edition
John von Neumann Faculty of Informatics
Lab 7

Dávid Bedők
2018-02-11
v1.0

Magazine

Protect the data with authentication

Task: based on the *BookStore* blueprint create an Enterprise Application which handles Magazines (CRUD operations), but protect the pages with authentication and bind the operations for roles.

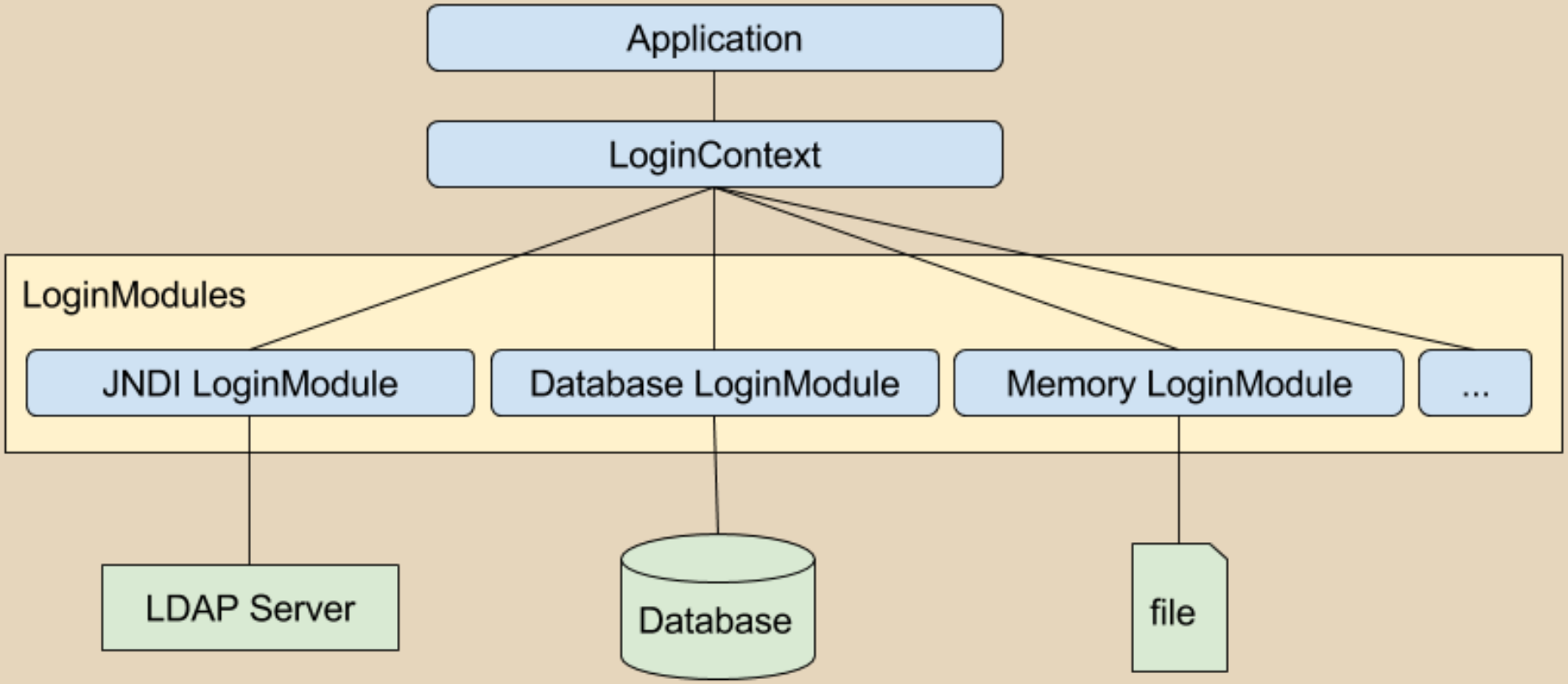
- The style sheet files (`css`) can be reachable to anybody.
- The query operations can be reachable to any authenticated user after a successful login process (all the users have the `maguser` role).
- The operations of data manipulation (create new magazin, edit or delete existing magazin) can be reachable only if the authenticated user has the `magadmin` role.

`[gradle|maven]\jboss\magazine`

Java Authentication and Authorization Service

- **Authentication** can defined as the process to confirm the identity of an user. This can be achieved in a variety of ways, e.g. by entering a password, swiping an ID card, or using a biometrical scanner. The user need not be a human being, but can be a computer process.
- **Authorization** can be defined as the process of deciding whether an authenticated user/system is allowed to access a certain resource or perform a certain action or not. A system may have many logical sections/modules, and not all users might have access to all modules. This is where authorization comes into play. Though the user might have authenticated himself, he might not have sufficient authorization to access certain particular data items.
- **JaaS** provides a framework and an API for the authentication and authorization of users, whether they're human or automated processes[1].

[1] JaaS implements a Java version of the standard **Pluggable Authentication Module** (PAM) framework.



LoginModules may prompt for and verify a user name and password. Others may read and verify a voice or fingerprint sample, or ascertain holding a secure private key, etc.

Realm

A **Realm** is a 'database' of usernames and passwords that identify valid users of a web application (or set of web applications), plus an enumeration of the list of roles associated with each valid user. A particular user can have any number of *roles* associated with their username.

- **JDBCRealm**: Accesses authentication information stored in a relational database, accessed via a `JDBC` driver.
- **DataSourceRealm**: Accesses authentication information stored in a relational database, accessed via a named `JNDI` `JDBC` DataSource.
- **JNDIRealm**: Accesses authentication information stored in an `LDAP` based directory server, accessed via a `JNDI` provider.
- **MemoryRealm**: Accesses authentication information stored in an in-memory object collection, which is initialized from an `XML` document.
- **JAASRealm**: Accesses authentication information through the Java Authentication and Authorization Service (`JaaS`) framework.

What should a framework like this do?

In the context of webapplication's authentication

A webapplication has **public** and **protected** resources (e.g.: webpages). You can reach the public items without authentication (e.g.: *login* page by default, images, css files, non-sensitive pages, etc.), but you have to successfully log in if you would like to get the 'protected' resources/pages.

If we visit a protected page the framework checks that we have already authenticated or not. If not we will redirected to the login page automatically. After a successful login process we will redirected back to the protected page (*landing page*).

What should a framework like this do?

In the context of webapplication's authorization

After login certain operations are only available to a certain people who are part of a certain group. We can **bind the operations to roles**, and you can perform a protected operation only if you have the appropriate role (*authorization*).

- We can bind the protected resources to a given role (e.g.: only the users who have the admin role are able to navigate to the *Admin* pages).
- Inside a reachable page we restricted the operations depending on the roles.

We do not need to create an own *LoginModule*. We can choose one of the pre-implemented one and we can configure it via parameters.

- We have users/password/roles and its relations in a PostgreSQL database
- Configuration: set the queries for JBoss how to use our tables for authentication and authorization
- We would like to use an own (custom) login page with (`XHTML`) form based authentication
- We want to configure our protected resources (need authentication) and operations (need specific role)

Auditing all the PostgreSQL queries

On database side

It would be useful if we can monitor the executed SQL queries on the database side because this time the JBoss will execute these queries and we would like to check its activity.

Audit file: `[PG-HOME]/data/pg_log/postgresql-[YYYY-MM-DD]_[random].log` Configuration file: `[PG-HOME]/data/postgresql.conf`

```
1 log_statement = 'all' # none, ddl, mod, all
```

`postgresql.conf`

Restart PostgreSql:

- Among Windows services (Stop/Start)
-
```
1 [PG-HOME]/bin/pg_ctl -D [PG-HOME]/data/ restart
```

`PG_HOME` environment variable:

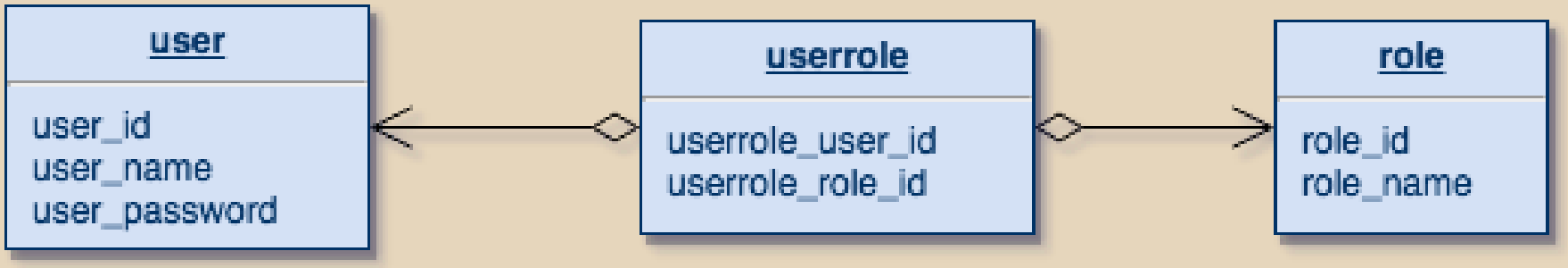
- WINOS: `c:\ProgramFiles\PostgreSQL\[version]`
- MACOS: `/Library/PostgreSQL/[version]`

Other table names could be possible, but in the samples we will authenticate the **user**(s) (with the stored password or hash) and we will use **role**(s) inside the application to restrict the access for some resources or operations.

The adventage of the 'A' sample is its simplicity. Altogether two tables, but if we use a `user_role` field instead of the `user_role_id` foreign key (and add a new index) we will solve the issue with one table. But of course there are some serious disadvantages as well: each user can have only one role, and in case of this database schema we will get a quite complex role handling logic inside the application.

- if an operation can be performed with multiple roles we have to list all these roles all the time (it causes redundancy)
- the access control of the application depends on database schema (this should be avoided).

If we insert a relation table between the `user` and the `role` tables we are able to add muliple roles to a user. This also simplifies the complexity of the application because we can create a new combined role if we have to use the same list of roles in more than one page/situation.

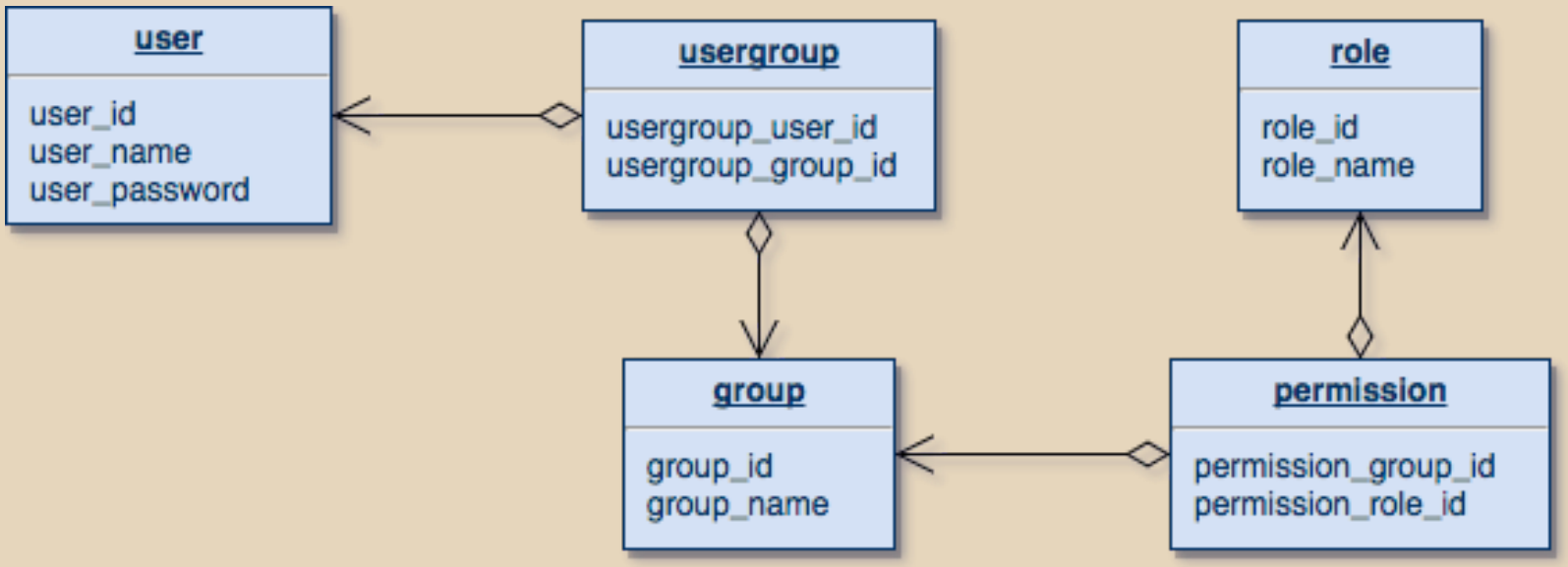


With time at an imagined company when a new *manager* arrives we will know which N roles we have to bind to the user, and which roles are needed in case of a *salesman*.
It is easy to make administration mistake in this sample, and this is one of its disadvantage. What if a new system or component is introduced at the company? It has some new roles which the new employees will get automatically but the old ones perhaps does not. It could cause some chaos.

Database

Sample 'C'

To come up with a solution for the issue of the Sample 'B' let introduce the group of the roles. In that case we will store the job roles in the database level, and beside the application the administration will be easier as well. But in that model it is difficult to handle the individual job roles, we have to create a new *group* even if it contains only one single user.



For comparison of the illustrated models I did not change the name of the tables, and it is also important to note that at the level of the application we use the items of the `role` table no matter what it is called. But in everyday life we call the `group` as *role*, and the `permission` could be a better name for the `role` table.

Database

Schema creation


```
1  CREATE TABLE magazinecategory ( [..] );
2  CREATE TABLE magazine ( [..] );
3
4  CREATE TABLE appuser (
5    appuser_id SERIAL NOT NULL,
6    appuser_name CHARACTER VARYING(100) NOT NULL,
7    appuser_password CHARACTER VARYING(100) NOT NULL,
8    CONSTRAINT PK_APPUSER_ID PRIMARY KEY (appuser_id)
9  );
10 CREATE UNIQUE INDEX UI_APPUSER_NAME ON appuser USING
11
12 CREATE TABLE role (
13   role_id SERIAL NOT NULL,
14   role_name CHARACTER VARYING(100) NOT NULL,
15   CONSTRAINT PK_ROLE_ID PRIMARY KEY (role_id)
16 );
17
18 CREATE TABLE userrole (
19   userrole_id SERIAL NOT NULL,
20   userrole_appuser_id INTEGER NOT NULL,
21   userrole_role_id INTEGER NOT NULL,
22   CONSTRAINT PK_USERROLE_ID PRIMARY KEY (userrole_id),
23   CONSTRAINT FK_USERROLE_USER FOREIGN KEY (userrole_appuser_id)
24     REFERENCES appuser (appuser_id) MATCH SIMPLE ON UPDATE RESTRICT ON DELETE RESTRICT,
25   CONSTRAINT FK_USERROLE_ROLE FOREIGN KEY (userrole_role_id)
26     REFERENCES role (role_id) MATCH SIMPLE ON UPDATE RESTRICT ON DELETE RESTRICT
27 );
```

We are going to choose the **B sample** which is good enough to be a live example but it's complexity will not distract attention. We avoid the `user` table name because of the *user* keyword, so the table name of the user will be `appuser`.
In the first round we will store *plaintext* passwords.

Database

Initial data


```
1  INSERT INTO role (role_id, role_name) VALUES (0, 'maguser');
2  INSERT INTO role (role_id, role_name) VALUES (1, 'magadmin');
3
4  SELECT SETVAL('role_role_id_seq', COALESCE(MAX(role_id), 1) ) FROM role;
5
6  INSERT INTO appuser (appuser_id, appuser_name, appuser_password) VALUES (0, 'alice',
       'a123');
7  INSERT INTO appuser (appuser_id, appuser_name, appuser_password) VALUES (1, 'bob',
       'b123');
8  INSERT INTO appuser (appuser_id, appuser_name, appuser_password) VALUES (2, 'charlie',
       'c123');
9
10 SELECT SETVAL('appuser_appuser_id_seq', COALESCE(MAX(appuser_id), 1) ) FROM appuser;
11
12 INSERT INTO userrole (userrole_appuser_id, userrole_role_id) VALUES (0, 0);
13 INSERT INTO userrole (userrole_appuser_id, userrole_role_id) VALUES (0, 1);
14 INSERT INTO userrole (userrole_appuser_id, userrole_role_id) VALUES (1, 0);
15 INSERT INTO userrole (userrole_appuser_id, userrole_role_id) VALUES (2, 0);
16
17 [..]
```

All three users have the **maguser** role, but *alice* has got the **megadmin** role as well.

Datasource configuration

JBoss

```
<subsystem xmlns="urn:jboss:domain:datasources:1.2">
    <datasources>
        <datasource jndi-name="java:jboss/datasources/magazineds"
             pool-name="MagazineDSPool" enabled="true" use-java-context="true">
            <connection-url>jdbc:postgresql://localhost:5432/magazinedb</connection-url>
            <driver>postgresql</driver>
            <security>
                <user-name>magazine_user</user-name>
                <password>123topSECret321</password>
            </security>
            <validation>
                <check-valid-connection-sql>SELECT 1</check-valid-connection-sql>
                <validate-on-match>true</validate-on-match>
                <background-validation>false</background-validation>
            </validation>
            <statement>
                <share-prepared-statements>false</share-prepared-statements>
            </statement>
        </datasource>
    </datasources>
</subsystem>
```

standalone.xml

The JNDI name of the *datasource* is `java:jboss/datasources/magazineds`. We will use that value not only in the persistence layer of the application but during the *Security Domain* configuration as well.

Security Domain and Login Module configuration

JBoss

```
1  <subsystem xmlns="urn:jboss:domain:security:1.2">
2      <security-domains>
3          <security-domain name="mag-security-db-domain">
4              <authentication>
5                  <login-module code="Database" flag="required">
6                      <module-option name="dsJndiName"
                           value="java:jboss/datasources/magazineds"/>
7                      <module-option name="principalsQuery" value="SELECT
                           appuser_password FROM appuser WHERE appuser_name = ?"/>
8                      <module-option name="rolesQuery" value="SELECT role_name, 'Roles'
                           FROM userrole INNER JOIN appuser ON (appuser_id =
                           userrole_appuser_id) INNER JOIN role ON (role_id =
                           userrole_role_id) WHERE appuser_name = ?"/>
9                  </login-module>
10             </authentication>
11         </security-domain>
12     </security-domains>
13 </subsystem>
```

standalone.xml

The JNDI name of the *Security Domain* will be java:/jaas/**mag-security-db-domain**. Behind the code attribute of the *LoginModule* there is a class. We can use the constant (e.g. UserRoles or Database) or the full qualified name of the class (e.g. org.jboss.security.auth.spi.UsersRolesLoginModule or org.jboss.security.auth.spi.DatabaseServerLoginModule).

SQL configuration

Principals and *Roles* queries

Pricipals query: The prepared statement query equivalent to: `select Password from Principals where PrincipalID=?`. If not specified this is the exact prepared statement that will be used.

```
1 SELECT appuser_password
2 FROM appuser
3 WHERE appuser_name = ?
```

Roles query: The prepared statement query equivalent to: `select Role, RoleGroup from Roles where PrincipalID=?`. If not specified this is the exact prepared statement that will be used. Value of `RoleGroup` column always has to be `Roles` (with capital 'R'). This is specific to JBoss.

```
1 SELECT
2   role_name,
3   'Roles'
4 FROM userrole
5   INNER JOIN appuser ON
6     (appuser_id = userrole_appuser_id)
7   INNER JOIN role ON
8     (role_id = userrole_role_id)
9 WHERE appuser_name = ?
```

Control Flag

Fine tuning the Security Domain's modules

A *Security Domain* can contain multiple *Login Modules*. The dependency between the modules is controlled by the **Control Flag**.

```
1 <login-module code="..." flag="required">...</login-module>
```

- `required`: The login module is required to succeed for the authentication to be successful. If any required module fails, the authentication will fail. The remaining login modules in the stack will be called regardless of the outcome of the authentication.
- `requisite`: The login module is required to succeed. If it succeeds, authentication continues down the login stack. If it fails, control immediately returns to the application.
- `sufficient`: The login module is not required to succeed. If it does succeed, control immediately returns to the application. If it fails, authentication continues down the login stack.
- `optional`: The login module is not required to succeed. Authentication still continues to proceed down the login stack regardless of whether the login module succeeds or fails.

It does not a good practice to store the plaintext password in the database. Of course this is not only ethical question but it may cause series safety issues. Storing the hash of the password is one of the common solution. We need hash functions to create a hash (eg. SHA family, MD5, etc.). The hash will be good for verifying but in theory these functions are irreversible (we cannot find a password for a known hash). In case of JBoss we can turn on these hash functions with the `hashUserPassword` property of the *Login Module*. We also have to modify the `initial-data.sql` to store the hash instead of the *plaintext* password (e.g. MD5-hex hash of the 123 is 202cb962ac59075b964b07152d234b70).

```
1 <login-module code="Database" flag="required">
2     <module-option name="dsJndiName"
          value="java:jboss/datasources/magazineds"/>
3     <module-option name="principalsQuery" value="[..]"/>
4     <module-option name="rolesQuery" value="[..]"/>
5     <module-option name="hashAlgorithm" value="MD5"/>
6     <module-option name="hashEncoding" value="hex"/>
7     <module-option name="hashUserPassword" value="true"/>
8 </login-module>
```

standalone.xml

Realm and logging configuration

JBoss

```
<management>
    <security-realms>
        <security-realm name="mag-realm">
            <authentication>
                <jaas name="mag-security-db-domain"/>
            </authentication>
        </security-realm>
    </security-realms>
</management>
[..]

<profile>
    [..]
    <subsystem xmlns="urn:jboss:domain:logging:1.5">
        [..]
        <logger category="org.jboss.security">
            <level name="TRACE"/>
        </logger>
        [..]
    </subsystem>
</profile>
```

Primarily we need **Realm** in case of BASIC authentication method (we are going to examine but will not use it). The configuration of the *Realm*s is very similar than the *Login Module* (but it has its own XSD), but we have the opportunity to refer an existing *Login Module*, and this is true from the opposite direction as well (but not the same time of course..).

If we would like to get some detailed information about the authentication and authorization steps we can turn on the debug logs for the presented package (do not use that config in a *production* system).

standalone.xml

Preparing the webapplication for JaaS

- Create a **Login** page
  - An XHTML form with a name and a password fields.
- Create a **LoginError** page
  - Use it when the authentication failed.
  - In general case we redirect the user to the Login page but before that we set an error message (or there is a hyperlink to navigate back to the login page).
- Create a **Logout** page
  - To log out the application (and empty/invalidate the related session).
  - In general case we redirect the users to the LoginPage after a 'successful' logout.
- Create a **Error** page(s)
  - In case of all other *Security* reason (we bind these pages to HTTP error codes).

Login servlet

mag-weblayer project

```
1  package hu.qwaevisz.magazine.weblayer.servlet;
2  [..]
3  @WebServlet("/Login")
4  public class LoginServlet extends HttpServlet implements
       LoginAttribute {
5
6    @Override
7    protected void doGet(HttpServletRequest request,
         HttpServletResponse response) throws ServletException,
         IOException {
8      request.setAttribute(ATTR_USERNAME, "");
9      request.setAttribute(ATTR_ERROR, "");
10     RequestDispatcher view =
           request.getRequestDispatcher("login.jsp");
11     view.forward(request, response);
12   }
13
14
15 }
```

In the ATTR_USERNAME and the ATTR_ERROR attributes we (may) store the given username and the possible error message. The login.jsp will process them if the authentication failed.

LoginServlet.java

Login page

mag-weblayer project

```
1  <%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8" %>
2  <%@ page import="hu.qwaevisz.magazine.weblayer.common.LoginAttribute" %>
3  [..]
4  <link rel="stylesheet" type="text/css" href="style/page.css" />
5  [..]
6  <body>
7    <%
8      String userName = (String) request.getAttribute(LoginAttribute.ATTR_USERNAME);
9      String errorMessage = (String) request.getAttribute(LoginAttribute.ATTR_ERROR);
10   %>
11   <form action="j_security_check" method="post">
12     <fieldset>
13       <legend>Login</legend>
14       <div class="error"><%= errorMessage %></div>
15       <div>
16         <label for="username">Username: </label>
17         <input type="text" name="j_username" id="username" value="<%= userName %>" />
18       </div>
19       <div>
20         <label for="password">Password: </label>
21         <input type="password" name="j_password" id="password" />
22       </div>
23       <in
24     </fie
25   </form>
26 </body>
27 </html>
```

Public resource!

If there is error message (errorMessage) or previously there was given the name of user (userName), these fields will be shown on the login page (after unsuccessful authentication). The j_* prefixes (servlet url/action and the name and password fields of the form) are imposed by the API, we have to use them in that way.

login.jsp

Login error servlet

mag-weblayer project

```
1  package hu.qwaevisz.magazine.weblayer.servlet;
2  [..]
3  @WebServlet("/LoginError")
4  public class LoginErrorServlet extends HttpServlet implements
       LoginAttribute {
5    @Override
6    protected void doPost(HttpServletRequest request,
         HttpServletResponse response) throws ServletException,
         IOException {
7      final String userName = request.getParameter("j_username");
8
9      request.setAttribute(ATTR_USERNAME, userName);
10     request.setAttribute(ATTR_ERROR, "Login failed");
11
12     RequestDispatcher view =
           request.getRequestDispatcher(Page.LOGIN.getJspName());
13     view.forward(request, response);
14   }
15
16 }
```

In case of error the `j_security_check` servlet will chuck the request to the login error page (at this time: /LoginError). We can read the form's fields from the request (in that example we read the username to store it for the reloaded login page).

LoginErrorServlet.java

Logout servlet

mag-weblayer project

```
1  @WebServlet("/Logout")
2  public class LogoutServlet extends HttpServlet
3    @Override
4    protected void doGet(HttpServletRequest request,
         HttpServletResponse response) throws ServletException,
         IOException {
5      response.setHeader("Cache-Control", "no-cache, no-store");
6      response.setHeader("Pragma", "no-cache");
7      response.setHeader("Expires", new
           java.util.Date().toString());
8      if (request.getSession(false) != null) {
9        request.getSession(false).invalidate();
10     }
11     if (request.getSession() != null) {
12       request.getSession().invalidate();
13     }
14     request.logout();
15     response.sendRedirect("list.jsp");
16   }
17 }
```

This is kind of *best practice* which deals some previously old fault of the application servers.

The **redirect** at the end of that servlet could be any protected resource. If the logout was successful the protected page loading will cause another redirect to the login page.

LogoutServlet.java

Error page(s)

mag-weblayer project

```java
@WebServlet("/Error")
public class ErrorServlet extends HttpServlet {
  @Override
  protected void doGet(HttpServletRequest request,
      HttpServletResponse response) throws ServletException,
      IOException {
    final RequestDispatcher view =
        request.getRequestDispatcher("error.jsp");
    view.forward(request, response);
  }
}
```

ErrorServlet.java

```jsp
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8" %>
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<link rel="stylesheet" type="text/css" href="style/page.css" />
<title>:: Error ::</title>
</head>
<body>Error</body>
</html>
```

We can define even multiple error pages in case of *Security* issues.

error.jsp

Webapplication's configuration

Protected resources

```
<web-app [..]>

  <security-constraint>
    <display-name>Magazine protected security constraint</display-name>
    <web-resource-collection>
      <web-resource-name>Protected Area</web-resource-name>
      <url-pattern>/*</url-pattern>
      <http-method>GET</http-method>
      <http-method>POST</http-method>
      <http-method>PUT</http-method>
      <http-method>DELETE</http-method>
      <http-method>HEAD</http-method>
      <http-method>OPTIONS</http-method>
      <http-method>TRACE</http-method>
    </web-resource-collection>
    <auth-constraint>
      <role-name>maguser</role-name>
      <role-name>magadmin</role-name>
    </auth-constraint>
    <user-data-constraint>
      <transport-guarantee>NONE</transport-guarantee>
    </user-data-constraint>
  </security-constraint>
  [..]
</web-app>
```

Each webpage (in case of any HTTP method) will be under protected area, except those what we will listed later. These pages can be reachable with `maguser` or `magadmin` roles. The database contains these roles.

`web.xml`

Webapplication's configuration

Public resources

```
<web-app [..]>
  [..]
  <security-constraint>
    <display-name>Magazine public security
        constraint</display-name>
    <web-resource-collection>
      <web-resource-name>Public</web-resource-name>
      <url-pattern>/style/*</url-pattern>
      <url-pattern>/Logout</url-pattern>
      <http-method>GET</http-method>
    </web-resource-collection>
  </security-constraint>
  [..]
</web-app>
```

web.xml

The CSS files and the Logout servlet can be reachable to anyone, without authentication. The *Login page* is reachable by default.

Webapplication's configuration

FORM auth-method

```
1  <web-app [..]>
2    [..]
3    <login-config>
4      <auth-method>FORM</auth-method>
5      <form-login-config>
6        <form-login-page>/Login</form-login-page>
7        <form-error-page>/LoginError</form-error-page
8      </form-login-config>
9    </login-config>
10
11   <security-role>
12     <description>Generic user</description>
13     <role-name>maguser</role-name>
14   </security-role>
15
16   <security-role>
17     <description>Administrator</description>
18     <role-name>magadmin</role-name>
19   </security-role>
20
21   <error-page>
22     <error-code>403</error-code>
23     <location>/Error</location>
24   </error-page>
25
26 </web-app>
```

The authentication method could be various. If there is a custom login form there value will be FORM (other values: BASIC, DIGEST and CLIENT-CERT).

The config of the form authentication method contain the path of the Login ((form-login-page)) and the LoginError page (form-error-page). In case of a JSP, the path is the same as the name of the JSP (this is the servlet path).

We have to list the available roles (security-role *elements*).

Error page definition in case of HTTP 403 (*Forbidden*) *status code*.

web.xml

BASIC authentication

The browser will pop up a built-in login window (javascript) where we can add the username and the password. In case of BASIC *auth method* we can define a **realm** in the web.xml, but in JBoss we can use the jboss-web.xml as well for the same purpose (to set the JNDI name of the *Security Domain*).

```
1 <web-app [..]>
2   [..]
3   <login-config>
4     <auth-method>BASIC</auth-method>
5     <realm-name>mag-realm</realm-name>
6   </login-config>
7   [..]
8 </web-app>
```

Warning! This web.xml is **NEM** part of the *Magazine* project.

web.xml

The is the JNDI name of the JBoss *Security Domain* what we set in the standalone.xml.

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <jboss-web>
3     <security-domain>java:/jaas/mag-security-db-domain</security-doma
4 </jboss-web>
```

jboss-web.xml

JBoss EAS and webapplication

relation of the JaaS configurations

- The descriptor of the webapplication (`web.xml`) may refer to the `security-realm` (`mag-realm`) of the JBoss (`standalone.xml`).
- The `security-realm` refers the `security-domain` (`mag-security-db-domain`) inside the `standalone.xml`.
- The JBoss specific descriptor of the webapplication (`jboss-web.xml`) refers the `JNDI` name of the `security-domain` of JBoss (`mag-security-db-domain`) (in case of `FORM` auth method).
- The `mag-security-db-domain` contains the configuration of the `Database` *login module*. We reach the database through this because this refers the `JNDI` name of the *DataSource*.

Role-Based Access Control

1st Restrict resources

Via the *deployment descriptor* of the webapplication (`web.xml`) define the public and protected images, (X)HTML or JSP files (etc.). Connect the resources, HTTP methods and roles to each other.

2nd Restrict GUI pages

Some part of the JSP/JSF pages can be displayed/enabled only for a user with a given role. At this level the authorization is only a UX question, the security is not relevant (the user can modify the sent down (X)HTML pages in the browser).

3rd Authorize user actions on servlet side

We have to verify the authenticated user's roles on the server side before perform an operation and send data to the EJB container. This is important because of the protection of the below EJB layer. Do not create transaction or EJB context if the operation does not allow for the authentication user (avoid resource wasting, increase the robustness of the application, etc.).

4th Authorize EJB operations

In case of a multi layer monolitic application the EJB operations can be called not only from the *frontend*, so we cannot trust on the GUI at all time. At most time at this level the authroization may fail because of a programming error or bug (of course there are some exceptional cases like session timeout, etc.).

Webapplication's authorization

Role-Based Access Control

If a page/servet is reachable for an authenticated user or the resource is public by default, the next step is the linking of the operations and roles. If we restrict some operations with visibility on GUI side, always double check the roles at the servlet side.

```
1 protected void doGet(HttpServletRequest request,
      HttpServletResponse response) throws ServletException,
      IOException {
2   if ( request.isUserInRole("magadmin") ) {
3     [..]
4   }
5   [..]
6 }
```

```
1 <% if (request.isUserInRole("magadmin")) { %>
2   <div>
3     <a href="Magazine?reference=-1&edit=1">Create</a> a brand new
          magazine.
4   </div>
5 <% } %>
```

EJB module configuration

mag-ejbservice project

This is very similar as the JBoss specific descriptor of the webapplication (jboss-web.xml). If we would like to use the Role-Based Access Control (RBAC) in the ejb modules we have to set the name of the *Security Domain* in the jboss-ejb3.xml descriptor (here we have to use the name and not the JNDI name!).

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <jboss:ejb-jar [..]>
3     <assembly-descriptor>
4         <s:security>
5             <ejb-name>*</ejb-name>
6             <s:security-domain>mag-security-db-domain</s:security-do
7         </s:security>
8     </assembly-descriptor>
9 </jboss:ejb-jar>
```

jboss-ejb3.xml

Note: If we do not want to deal with the authentication in the ejb modules we just leave this configuration empty (or do not use that file if it does not contain anything else).

Remove magazine inside the EJB layer

mag-ejblayer project

```
1  @PermitAll
2  @Stateless(mappedName = "ejb/magazineFacade")
3  public class MagazineFacadeImpl implements MagazineFacade {
4
5    [..]
6    @Override
7    @RolesAllowed("magadmin")
8    public void removeMagazine(String reference) throws
         FacadeException {
9      try {
10       this.service.delete(reference);
11     } catch (final PersistenceServiceException e) {
12       LOGGER.error(e, e);
13       throw new FacadeException(e.getLocalizedMessage());
14     }
15   }
16
17
18
19 }
```

@DenyAll: not reachable for any roles
@PermitAll: reachable for any roles
@RolesAllowed("magadmin") or @RolesAllowed("magadmin", "maguser"): reachable for the given role(s)
At class level: applies to all methods

MagazineFacadeImpl.java

RedHat JBoss vs. Oracle WebLogic

The same concept, small differences

	JBoss 6.4	WebLogic 12.2.1
web module container specific dd[2]	`weblogic-ejb-jar.xml`	`weblogic.xml`
ejb module container specific dd	`jboss-ejb3.xml`	`jboss-web.xml`
security role name's known limitations		In case of hyphens the Admin Console breaks the visualization
security role	**application role** and **security role** are the same	**application role** and **security role** are not the same
security realm's known limitations	Any number of security realms can exist at the same time.	Any number of security realms can exist, but only one can be active at the same time.
conceptual deviations	**Login Module** (with *Control Flags*)	**Authentication Provider** (with *Control Flags*)
reference from application	JNDI or short name of the Security Domain	no reference (the active one will be used)

[2] deployment descriptor

Application Role vs. Security Role

WebLogic

The **Application Role** is logically separated from the **Security Role** (what we have already known in case of JBoss). For one *Application Role* may bind one or more *User* or one or more *Security Role*. It is obligatory to set the binding rule configuration in the container specific *deployment descriptor* (this is the bridge between the application and the *Security Realm*).

```
1 <security-role-assignment>
2   <role-name>ROLE_A</role-name>
3   <principal-name>PRINCIPAL_1</principal-name>
4   <principal-name>PRINCIPAL_2</principal-name>
5 </security-role-assignment>
```

- `ROLE_A`: **Application Role**, use it in the `web.xml` and the Java code
- `PRINCIPAL_1` és `PRINCIPAL_2`: **Security Role** (recommended) or **Security User** (configure these via the *Security Realm*)
- In case of WEB module use the `weblogic.xml`, in case of EJB module use the `weblogic-ejb-jar.xml` *deployment descriptor*.

Login Module vs. Authentication Provider

JBoss vs. WebLogic

Login Module

- RedHat JBoss
- Multiple *Login Module(s)* can be found under the **Security Domain**.
- A **Security Realm** can refer to a **Security Domain** (and visa-versa).
- Use the **Security Realm** mostly in case of remote access.

Authentication Provider

- Oracle WebLogic
- Multiple *Authentication Provider(s)* can be found under the **Security Realm**.
- The *Security Domain* appellation does not exist in case of WebLogic.

We are going the add a new *Authentication Provider* to the default `myrealm` *Security Realm*. This new *Authentication Provider* will authenticate via our database. Set the *Control Flag* of the original and the new *provider* to `SUFFICIENT` (original value if `REQUIRED`).

WebLogic DataSource configuration

Via Server Administration Console

- Add PostgreSQL JDBC library into the domain's classpath
  - Copy JDBC driver (e.g.: `postgresql-9.4-1201.jdbc41.jar`) here: `[MW\_HOME]\user\_projects\domains\mydomain\lib\`
  - Restart server instance
- WebLogic Server Administration Console
  - Services | Data Sources | Configuration tab
    - New → Generic Data Source
      * Name: **magazineds**
      * JNDI name: `jdbc/datasource/magazineds`
      * Target: "myserver"
      * Database type: PostgreSQL
    - Next, Next..
      * Database name: **magazinedb**
      * Host name: **localhost**
      * Port: **5432** (default)
      * Database user name: **magazine_user**
      * Password: **123topSECret321**
    - It is worth testing the connection through the administration console

Repetition..

WebLogic Security Realm configuration

Via Server Administration Console

- Security Realms → Select `myrealm`
  - Providers tab → New
    - Name: **magazineauthenticator**
    - Type: **SQLAuthenticator**
  - Select `magazineauthenticator`
    - Configuration tab | Common tab
      - Control Flag: **SUFFICIENT**
      - Save
    - Configuration tab | Provider Specific tab
      - Plaintext Passwords Enabled (checked)
      - Data Source Name: **magazineds**
      - Password Style Retained (checked)
      - Descriptions Supported (unchecked)
      - SQL Queries... → next slide

SQL queries I

Configuration tab | Provider Specific tab

SQL Get Users Password (JBoss: **principalsQuery**)	`SELECT appuser_password FROM appuser WHERE appuser_name =?`
SQL Set User Password	`UPDATE appuser SET appuser_password =? WHERE appuser_name =?`
SQL User Exists	`SELECT appuser_name FROM appuser WHERE appuser_name =?`
SQL List Users	`SELECT appuser_name FROM appuser WHERE appuser_name LIKE?`
SQL Create User	`INSERT INTO appuser (appuser_id, appuser_name, appuser_password) VALUES (?,?,?)`
SQL Remove User	`DELETE FROM appuser WHERE appuser_name =?`
SQL List Groups	`SELECT role_name FROM role WHERE role_name LIKE?`
SQL Group Exists	`SELECT role_name FROM role WHERE role_name =?`
SQL Create Group	`INSERT INTO role (role_id, role_name) VALUES (?,?)`
SQL Remove Group	`DELETE FROM role WHERE role_name =?`
SQL Is Member	`SELECT appuser_name FROM userrole INNER JOIN appuser ON ( appuser_id = userrole_appuser_id ) INNER JOIN role ON ( role_id = userrole_role_id ) WHERE appuser_name =? AND role_name =?`

SQL queries II

Configuration tab | Provider Specific tab

SQLSQL List Member Groups (JBoss: **rolesQuery** without `RoleGroup`)	`SELECT role_name FROM userrole INNER JOIN appuser ON ( appuser_id = userrole_appuser_id ) INNER JOIN role ON ( role_id = userrole_role_id ) WHERE appuser_name =?`
SQL List Group Members	`SELECT appuser_name FROM userrole INNER JOIN appuser ON ( appuser_id = userrole_appuser_id ) INNER JOIN role ON ( role_id = userrole_role_id ) WHERE appuser_name LIKE? AND role_name =?`
SQL Remove Group Memberships	`DELETE FROM userrole WHERE userrole_role_id = ( SELECT role_id FROM role WHERE role_name =? ) AND userrole_appuser_id = ( SELECT appuser_id FROM appuser WHERE appuser_name =? )`
SQL Add Member To Group	`INSERT INTO userrole (userrole_appuser_id, userrole_role_id) VALUES (?,?)`
SQL Remove Member From Group	`DELETE FROM userrole WHERE userrole_role_id = ( SELECT role_id FROM role WHERE role_name =? ) AND userrole_appuser_id = ( SELECT appuser_id FROM appuser WHERE appuser_name =? )`
SQL Remove Group Member	`DELETE FROM userrole WHERE userrole_role_id = ( SELECT role_id FROM role WHERE role_name =? )`
Description specific	We turned off the usage of the *description*(s).

Alice successful authentication

SQL log

```
1  SELECT appuser_password
2  FROM appuser
3  WHERE appuser_name = 'alice';
4
5  SELECT role_name
6  FROM userrole
7    INNER JOIN appuser ON ( appuser_id = userrole_appuser_id )
8    INNER JOIN role ON ( role_id = userrole_role_id )
9  WHERE appuser_name = 'alice';
10
11 SELECT role_name
12 FROM userrole
13   INNER JOIN appuser ON ( appuser_id = userrole_appuser_id )
14   INNER JOIN role ON ( role_id = userrole_role_id )
15 WHERE appuser_name = 'maguser';
16
17 SELECT role_name
18 FROM userrole
19   INNER JOIN appuser ON ( appuser_id = userrole_appuser_id )
20   INNER JOIN role ON ( role_id = userrole_role_id )
21 WHERE appuser_name = 'magadmin';
```

It runs for the **principal names** (`weblogic.xml`) because this could be *Security Role* and *Security User* as well.

Verify WebLogic Security Realm

Via Server Administration Console

Server restart

- ▷ Security Realms → Select `myrealm`
  - Users and Groups tab
    - Users tab

Name	Provider
alice	`magazineauthenticator`
bob	`magazineauthenticator`
charlie	`magazineauthenticator`
weblogic	`DefaultAuthenticator`

    - Groups tab (after select alice)

Name	Provider
magadmin	`magazineauthenticator`
maguser	`magazineauthenticator`
...	`DefaultAuthenticator`

Magazine project

Source code modifications in case of WebLogic

- Replace Log4j to **JDK Logging**
- Replace Hibernate dependency to **Eclipselink**
- Remove the `request.logout()` on `LogoutServlet`
  - It causes `NullPointerException` (known issue)
- Use `mag-user` and `mag-admin` *Application Roles* instead of `maguser` and `magadmin`
  - e.g.: `list.jsp`
    ```
    <% if (request.isUserInRole("mag-admin")) { %>
    ```
  - e.g.: `MagazineFacadeImpl`
    ```
    @RolesAllowed("mag-admin")
    ```
- `jboss-web.xml` instead of `weblogic.xml`
- `jboss-ejb3.xml` instead of `weblogic-ejb-jar.xml`

Webapplication configuration

mag-weblayer project

```
1  <web-app version="3.0" xmlns="http://java.sun.com/xml/ns/javaee"
2    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
         http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd">
4    [..]
5    <security-constraint>
6      <display-name>Magazine protected security constraint</display-name>
7      [..]
8      <auth-constraint>
9        <role-name>mag-user</role-name>
10       <role-name>mag-admin</role-name>
11     </auth-constraint>
12     [..]
13   </security-constraint>
14   [..]
15   <security-role>
16     <description>Generic user</description>
17     <role-name>mag-user</role-name>
18   </security-role>
19   <security-role>
20     <description>Administrator</description>
21     <role-name>mag-admin</role-name>
22   </security-role>
23   [..]
24 </web-app>
```

Application Role

web.xml

WEB module configuration in case of WebLogic

mag-weblayer project

```
1  <?xml version='1.0' encoding='UTF-8'?>
2  <weblogic-web-app xmlns="http://www.bea.com/ns/weblogic/90"
3    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
4
5    <security-role-assignment>
6      <role-name>mag-user</role-name>
7      <principal-name>maguser</principal-name>
8    </security-role-assignment>
9    <security-role-assignment>
10     <role-name>mag-admin</role-name>
11     <principal-name>magadmin</principal-name>
12   </security-role-assignment>
13
14 </weblogic-web-app>
```

weblogic.xml

Application Role: mag-user and mag-admin
Security Role: maguser and magadmin (these are in the database)

EJB module configuration in case of WebLogic

mag-ejblayer project

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <weblogic-ejb-jar xmlns="http://www.bea.com/ns/weblogic/90"
       xmlns:j2ee="http://java.sun.com/xml/ns/j2ee"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3    xsi:schemaLocation="http://www.bea.com/ns/weblogic/90
         http://www.bea.com/ns/weblogic/90/weblogic-ejb-jar.xsd">
4
5    <security-role-assignment>
6      <role-name>mag-user</role-name>
7      <principal-name>maguser</principal-name>
8    </security-role-assignment>
9    <security-role-assignment>
10     <role-name>mag-admin</role-name>
11     <principal-name>magadmin</principal-name>
12   </security-role-assignment>
13
14 </weblogic-ejb-jar>
```

The only difference from the `weblogic.xml` is the root tag.

`weblogic-ejb-jar.xml`