0. Pré-requisito no banco

A view que o Elytron usa:

```sql
CREATE OR REPLACE VIEW vw_usuario_roles AS
SELECT
    u.login  AS username,
    a.acesso AS role_name,
    u.senha  AS senha
FROM aut_usuario u
JOIN aut_usuario_acesso ua ON ua.usuario_id = u.id
JOIN aut_acesso a          ON a.id = ua.acesso_id;
```

Regras:

- `username` → login do usuário
- `senha` → senha em texto claro (no teu caso está `clear-password`)
- `role_name` → nome da role (ex: ADMIN, USER etc.)

1. Datasource que o Elytron vai usar

Trecho do `standalone.xml` (subsystem datasources):

```xml
<subsystem xmlns="urn:jboss:domain:datasources:7.0">
    <datasources>

        <!-- ... ExampleDS ... -->

        <datasource jndi-name="java:jboss/datasources/talentiDS"
                    pool-name="talentiDS"
                    enabled="true"
                    use-java-context="true"
                    statistics-enabled="${wildfly.datasources.statistics-enabled:${wildfly.statistics-
enabled:false}}">

            <connection-url>jdbc:postgresql://localhost:5433/talenti-pro</connection-url>
            <driver>postgresql</driver>

            <pool>
                <min-pool-size>10</min-pool-size>
                <max-pool-size>50</max-pool-size>
                <prefill>true</prefill>
                <flush-strategy>IdleConnections</flush-strategy>
            </pool>

            <security>
                <user-name>postgres</user-name>
                <password>admin</password>
            </security>

            <validation>
                <check-valid-connection-sql>SELECT 1</check-valid-connection-sql>
                <validate-on-match>true</validate-on-match>
                <background-validation>true</background-validation>
                <background-validation-millis>30000</background-validation-millis>
            </validation>

            <timeout>
                <blocking-timeout-millis>5000</blocking-timeout-millis>
                <idle-timeout-minutes>3</idle-timeout-minutes>
                <query-timeout>30</query-timeout>
                <use-try-lock>30000</use-try-lock>
            </timeout>

            <statement>
                <track-statements>NOWARN</track-statements>
                <prepared-statement-cache-size>100</prepared-statement-cache-size>
                <share-prepared-statements>true</share-prepared-statements>
            </statement>
        </datasource>

        <drivers>
            <driver name="postgresql" module="org.postgresql">
                <driver-class>org.postgresql.Driver</driver-class>
            </driver>
        </drivers>
    </datasources>
</subsystem>
```

Pontos importantes:

- **JNDI**: `java:jboss/datasources/talentiDS`
- **Nome do datasource (pool-name)**: `talentiDS`
- O Elytron **usa o pool-name** no `data-source="talentiDS"`.

2. Elytron – Realms (incluindo o JDBC Realm)

Trecho do subsystem Elytron:

xml

```xml
<subsystem xmlns="urn:wildfly:elytron:16.0" final-providers="combined-providers" disallowed-
providers="OracleUcrypto">
    ...
    <security-realms>

        <!-- Realm local (para $local) -->
        <identity-realm name="local" identity="$local"/>

        <!-- Realm baseado em arquivos (aplicação legacy) -->
        <properties-realm name="ApplicationRealm">
            <users-properties path="application-users.properties"
                              relative-to="jboss.server.config.dir"
                              digest-realm-name="ApplicationRealm"/>
            <groups-properties path="application-roles.properties"
                               relative-to="jboss.server.config.dir"/>
        </properties-realm>

        <!-- Realm de gerenciamento (console admin) -->
        <properties-realm name="ManagementRealm">
            <users-properties path="mgmt-users.properties"
                              relative-to="jboss.server.config.dir"
                              digest-realm-name="ManagementRealm"/>
            <groups-properties path="mgmt-groups.properties"
                               relative-to="jboss.server.config.dir"/>
        </properties-realm>

        <!-- *** SEU REALM JDBC *** -->
        <jdbc-realm name="talenti-jdbc-realm">

            <principal-query
                    sql="SELECT senha, role_name FROM vw_usuario_roles WHERE username = ?"
                    data-source="talentiDS">

                <!-- primeira coluna da query = senha -->
                <clear-password-mapper password-index="1"/>

                <!-- segunda coluna = role_name, mapeada para atributo 'groups' -->
                <attribute-mapping>
                    <attribute index="2" to="groups"/>
                </attribute-mapping>

            </principal-query>

        </jdbc-realm>

    </security-realms>
    ...
</subsystem>
```

Entendendo:

- `principal-query` busca: `senha` (coluna 1) e `role_name` (coluna 2).
- `clear-password-mapper password-index="1"` → senha **em texto claro** na coluna 1.
- `attribute index="2" to="groups"` → joga a coluna 2 no atributo `groups`.
- Mais pra frente, o role-decoder vai pegar `groups` e transformar em roles.

3. Elytron – SecurityDomain específico da aplicação

Ainda no Elytron:

```xml
<security-domains>

    <!-- Domains padrão do servidor -->
    <security-domain name="ManagementDomain" default-realm="ManagementRealm" permission-mapper="default-permission-mapper">
        <realm name="ManagementRealm" role-decoder="groups-to-roles"/>
        <realm name="local" role-mapper="super-user-mapper"/>
    </security-domain>

    <security-domain name="ApplicationDomain" default-realm="ApplicationRealm" permission-mapper="default-permission-mapper">
        <realm name="ApplicationRealm" role-decoder="groups-to-roles"/>
        <realm name="local"/>
    </security-domain>

    <!-- *** DOMAIN DO TALENTI *** -->
    <security-domain name="TalentiDomain"
                     default-realm="talenti-jdbc-realm"
                     permission-mapper="default-permission-mapper">

        <realm name="talenti-jdbc-realm" role-decoder="groups-to-roles"/>

    </security-domain>

</security-domains>
```

- `TalentiDomain` é o **security-domain Elytron** da tua aplicação.
- Usa `talenti-jdbc-realm` como realm padrão.
- Usa `role-decoder="groups-to-roles"`, que olha pro atributo `groups` (preenchido lá na query).

Mappers usados (já prontos no XML):

```xml
<mappers>
    <simple-permission-mapper name="default-permission-mapper" mapping-mode="first">
        <permission-mapping>
            <principal name="anonymous"/>
            <permission-set name="default-permissions"/>
        </permission-mapping>
        <permission-mapping match-all="true">
            <permission-set name="login-permission"/>
            <permission-set name="default-permissions"/>
        </permission-mapping>
    </simple-permission-mapper>

    <constant-realm-mapper name="local" realm-name="local"/>

    <!-- IMPORTANTE: decodifica o atributo 'groups' em roles -->
    <simple-role-decoder name="groups-to-roles" attribute="groups"/>

    <constant-role-mapper name="super-user-mapper">
        <role name="SuperUser"/>
    </constant-role-mapper>
</mappers>
```

4. Elytron – HttpAuthenticationFactory do Talenti

Ainda no Elytron:

```xml
<http>
    <http-authentication-factory name="management-http-authentication"
                                 security-domain="ManagementDomain"
                                 http-server-mechanism-factory="global">
        ...
    </http-authentication-factory>

    <http-authentication-factory name="application-http-authentication"
                                 security-domain="ApplicationDomain"
                                 http-server-mechanism-factory="global">
        ...
    </http-authentication-factory>

    <!-- *** PARA O TALENTI *** -->
    <http-authentication-factory name="talenti-http-authentication"
                                 security-domain="TalentiDomain"
                                 http-server-mechanism-factory="global">

        <mechanism-configuration>
            <mechanism mechanism-name="BASIC">
                <mechanism-realm realm-name="TalentiRealm"/>
            </mechanism>
        </mechanism-configuration>

    </http-authentication-factory>

    <provider-http-server-mechanism-factory name="global"/>
</http>
```

- Essa factory define:
    - que o domínio de segurança é o `TalentiDomain` (que usa teu JDBC Realm);
    - que o mecanismo HTTP é **BASIC** (popup do navegador);
    - e o nome que aparece no challenge é `TalentiRealm`.

Para usar **FORM login personalizado**, depois você poderia trocar o `mechanism-name` para `FORM` e ajustar o `web.xml`, mas do jeito que está, funciona com BASIC.

5. Undertow – ligar o domínio da app ao Elytron

No subsystem Undertow:

```xml
<subsystem xmlns="urn:jboss:domain:undertow:13.0"
           default-server="default-server"
           default-virtual-host="default-host"
           default-servlet-container="default"
           default-security-domain="other"
           ...>

    ...

    <application-security-domains>
        <application-security-domain name="other" security-domain="ApplicationDomain"/>

        <!-- *** TALENTI APP *** -->
        <application-security-domain name="talenti-app" security-domain="TalentiDomain"/>
    </application-security-domains>
</subsystem>
```

Aqui você faz a "ponte":

- Nome lógico da app: `talenti-app`
- Security-domain Elytron: `TalentiDomain`

Esse nome `talenti-app` é o que você vai usar dentro do `.war` (no `wildfly-web.xml`).

6. Dentro da aplicação – wildfly-web.xml

No `WEB-INF/wildfly-web.xml` do teu `talenti-pro.war`:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<jboss-web>
    <security-domain>talenti-app</security-domain>
</jboss-web>
```

- Isso diz para o WildFly:
  "essa aplicação web usa o **application-security-domain** chamado `talenti-app`",
  que por sua vez está mapeado para o `TalentiDomain` (que usa o teu JDBC Realm).

7. Dentro da aplicação – web.xml (autenticação e URLs protegidas)

Como o teu Elytron está configurado com **BASIC**, a configuração mínima no `web.xml` é assim:

```xml
<web-app ...>

    <!-- LOGIN CONFIG: BASIC + Elytron -->
    <login-config>
        <auth-method>BASIC</auth-method>
        <realm-name>TalentiRealm</realm-name>
    </login-config>

    <!-- ROLES QUE A APP CONHECE -->
    <security-role>
        <role-name>ADMIN</role-name>
    </security-role>
    <security-role>
        <role-name>USER</role-name>
    </security-role>

    <!-- ÁREA PÚBLICA (sem auth) -->
    <security-constraint>
        <web-resource-collection>
            <web-resource-name>Publico</web-resource-name>
            <url-pattern>/public/*</url-pattern>
            <url-pattern>/resources/*</url-pattern>
            <url-pattern>/javax.faces.resource/*</url-pattern>
        </web-resource-collection>
        <!-- sem auth-constraint => público -->
    </security-constraint>

    <!-- ÁREA PROTEGIDA -->
    <security-constraint>
        <web-resource-collection>
            <web-resource-name>Protegido</web-resource-name>
            <url-pattern>/*</url-pattern>
        </web-resource-collection>
        <auth-constraint>
            <role-name>ADMIN</role-name>
            <role-name>USER</role-name>
        </auth-constraint>
    </security-constraint>

</web-app>
```

Com isso:

- qualquer URL que não esteja sob `/public/*` etc. exigirá login BASIC;
- o Elytron vai autenticar no banco (`vw_usuario_roles`);
- as roles `ADMIN` e `USER` vêm da coluna `role_name`.

Se depois você quiser **FORM + login.xhtml**, dá pra adaptar (trocar `BASIC` por `FORM`, ajustar http-auth-factory, etc.), mas o cenário atual (BASIC) já está redondinho.

8. Como usar na aplicação (JSF/CDI)

Depois de tudo isso, **dentro da tua app** você não autentica mais "na mão"; só consome:

Usuário logado:

```java
String usuario = FacesContext.getCurrentInstance()
        .getExternalContext()
        .getUserPrincipal()
        .getName();
```

Checar role:

```java
boolean admin = FacesContext.getCurrentInstance()
        .getExternalContext()
        .isUserInRole("ADMIN");
```

Anotação em EJB/Rest/Bean:

```java
@RolesAllowed("ADMIN")
public void salvar() { ... }
```

Resumão do fluxo

1. Navegador acessa URL protegida (`/*`).
2. Undertow vê que a app usa `talenti-app` → `TalentiDomain`.
3. `talenti-http-authentication` faz BASIC auth contra o `TalentiDomain`.
4. `TalentiDomain` usa o `talenti-jdbc-realm`.
5. Realm executa a query na `vw_usuario_roles` via datasource `talentiDS`.
6. Elytron compara senha, carrega roles → atributo `groups`.
7. `groups-to-roles` vira roles de aplicação.
8. `web.xml` decide se o user pode ou não acessar a URL.

Se quiser, no próximo passo posso te montar uma versão de `web.xml` usando **FORM + login.xhtml bonitinho** mantendo o mesmo Elytron/JDBC que você já configurou.