

# Szövetségi alapon történő felhasználó azonosítás Huntéka könyvtári alkalmazásokban

## Könyvtári alkalmazás használati esetei

### Könyvtári és föderatív azonosságok

A könyvtári alkalmazás olvasói nyilvántartásában szereplő és a föderatív azonosítás szolgáltatójánál lévő azonosságokat a könyvtári alkalmazásban célszerűen össze kell kapcsolni. Ez az identitás összekapcsolás mindig feltételezi a felhasználó aktív közreműködését, e nélkül az identitások összekapcsolása nem jöhet létre.

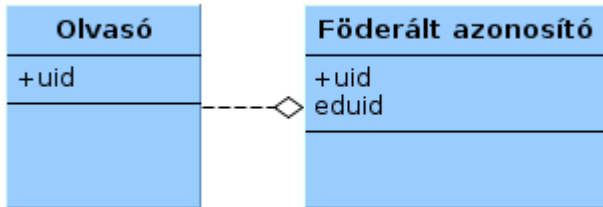
Egy föderatív azonosítással elérhető, SP-vel védett Online könyvtári beiratkozás szolgáltatásnál ez egy tájékoztató szöveg megjelenése és tudomásul vétele után történik meg, a már hagyományos módon beiratkozott, az alkalmazás olvasói nyilvántartásában már szereplő olvasók "könyvtári identitását" viszont külön eljárással kell összekapcsolni az IdP által azonosítottakkal.

Ha a könyvtári alkalmazásban már létezik webes azonosítás, akkor célszerű egy olyan komponenst beilleszteni, ahol a felhasználó mind a könyvtári azonosságát (könyvtári azonosító - jelszó), mind a föderatív azonosságát tudja bizonyítani, és a komponens összeköti a két azonosítót a kapcsoló táblában (account linking). A tájékoztató szöveg megjelenítése és annak elfogadása ekkor történhet meg.

Az MTA SZTAKI könyvtára esetében a felhasználók nem tudtak bejelentkezni a webes felületen, az identitás összekapcsolást segítő kulcs az email cím lehetett, ez alapján az összerendelések nagy tömegét elvégezhettük, a maradék azonosítót pedig kézzel, további adatok összehasonlításával, vagy közvetlenül az olvasó megkeresésével lehetett elvégezni.

### Identitások összekapcsolása

A könyvtári rendszert fel kell készíteni a szövetségi azonosítás során használt azonosítójokkal érkező felhasználók fogadására. Ehhez szükség van egy kapcsoló táblára, ahol a könyvtári azonosítót és a föderált azonosítókat tároljuk.ú



Bejelentkezés során bármely eduid-vel azonosítja a felhasználót a szóbanforgó Home IdP, az alkalmazás kikeresi a megfelelő könyvtári alkalmazásbeli azonosítót, és a továbbiakban az alkalmazás ezt használja a munkamenet során.

## Beiratkozás

Ha a felhasználó olyan azonosítóval jön a Home IdP-től, ami nincs egyetlen alkalmazás azonosítóhoz sem kapcsolva, a beiratkozás folyamatát kell végrehajtani.

A beiratkozás folyamán létre kell hozni az olvasó táblában az új olvasót, és a Home IdP-től kapott személyes adatokkal fel kell tölteni az olvasó rekordot.

A "Föderált azonosító" kapcsolótáblában létre kell hozni a könyvtári azonosítót és a föderált azonosítót összerendelő rekordot.

## Kiiratkozás

Felhasználó kiiratását az OIOSAML.java nem támogatja, az azonosító lejáratásával lehet a felhasználó jogosultságait korlátozni. Tervezésnél meg kell fontolni, hogy egy-egy IdP által azonosított felhasználó beiratásánál mennyi az az idő, amíg az olvasót aktívnek szeretnénk tudni. Megvizsgálva néhány könyvtár Használati szabályzatát a megoldás megfelel az általános gyakorlatnak. A könyvtárak beiratkozáskor kiállított olvasójegyet bizonyos időre szóló érvényességgel (általában 1 év) állítják ki, ha az olvasó ennek elteltével nem hosszabbítja meg olvasójegye érvényességét, az lejár.

## Azonosítás és attribútumok átadása

A LoginFilter által védett URL-re irányuló kéréskor a SAML Assertion és az általa hordozott attribútumok a dk.itst.oiosaml.sp.UserAssertion objektumból lekérdezhetőek. Ezen objektum csak a védett URL-eken érhető el a dk.itst.oiosaml.sp.UserAssertionHolder.get() metódus meghívásával.

Ha csak a login funkciót védjük SAML SP-vel, érdemes a user session-be átmenteni a szükséges attribútumokat, ha azokat más funkciókkal is el kell érni.

Attribútum mappingről igény szerint a fejlesztés során kell gondoskodnunk, az OIOSAML.java nem támogat ilyen funkciót.

Példa az eduPersonPrincipalName attribútum kinyeréséről:

```
UserAssertion ua = UserAssertionHolder.get();  
String eppn = ua.getAttribute("urn:oid:1.3.6.1.4.1.5923.1.1.1.6 ");
```

A rendszerfejlesztéshez kapcsolódó dokumentáció az alábbi URL-en található:

[<https://svn.softwareborsen.dk/oiosaml.java/sp/trunk/docs/developers.html>]

## Attribútumok frissítése

Hogy a könyvtári rendszerben nyilvántartott olvasók adatai naprakészek legyenek, a Home IdP-től származó attribútumokat nem csak a beiratkozáskor, hanem a mindennapi használatkor is igénybe vehetjük.

Az IdP-től kapott attribútumokat a könyvtári alkalmazás ellenőrzi, és ha az adott attribútum még nem szerepel a nyilvántartásában, azt rögzíti.

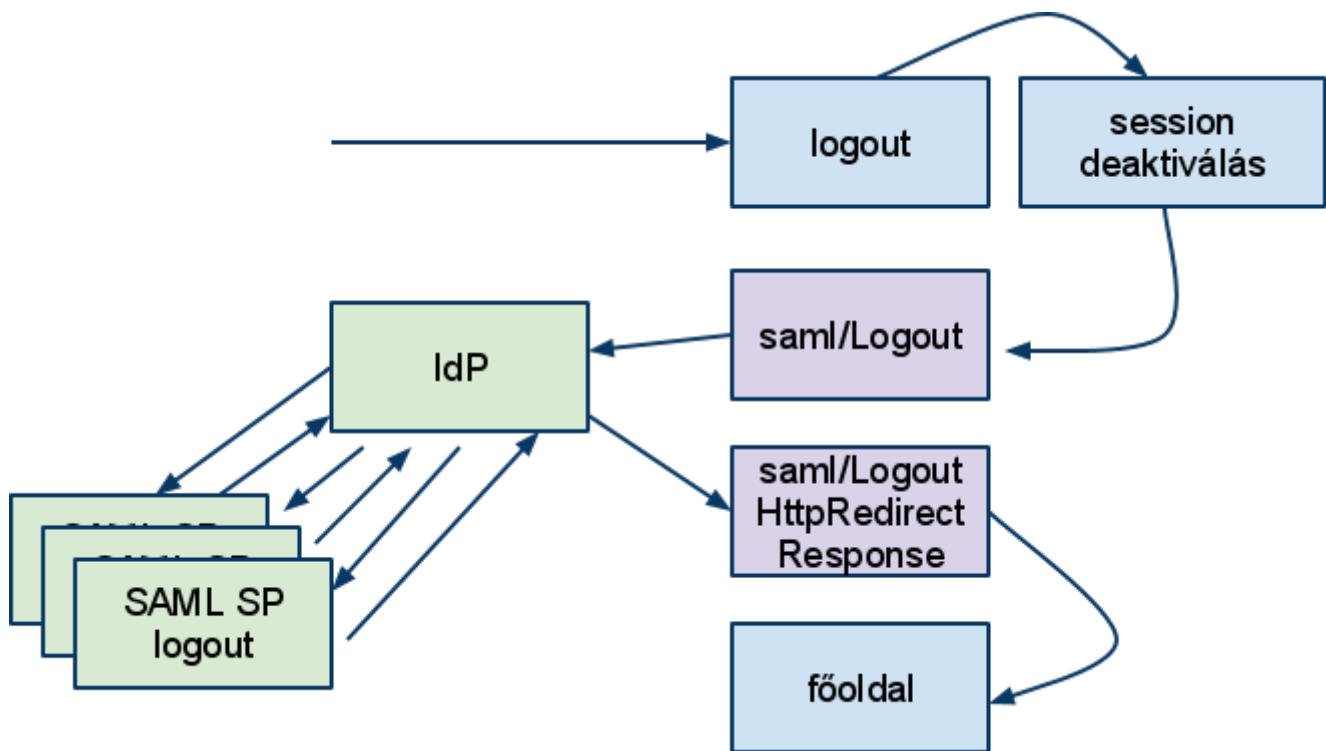
Az alkalmazás saját szabály-rendszere döntheti el, hogy a meglévő értékeket lecseréli az Home IdP-től kapottal, vagy csak hozzáadja új értéként a már meglévőhöz.

## Kijelentkezés

### SP által kezdeményezett SLO

Az alkalmazásból történő kijelentkezést a session-változók törlésével kell megoldani, és biztosítani kell a Single Logout folyamat elindítását. Az OIOSAML.java esetében ezt a saml/Logout URL-re történő redirect-tel lehet kezdeményezni.

```
HttpSession session = req.getSession(false);  
res.sendRedirect("saml/Logout");
```



## IdP által kezdeményezett SLO

Ha a Single Logout-ot a felhasználó egy másik alkalmazásnál kezdeményezte, és kérte, hogy valamennyi aktív alkalmazásból léptesse ki a rendszer, az IdP a könyvtári alkalmazásnak egy Logout Request-et küld a böngésző átirányításának segítségével. A kérésről a SAML SP-n kívül az alkalmazásnak is tudnia kell, hogy a felhasználó session-jén a kilépést érvényre juttassa. Ezt a SAML SP SLO endpoint-jai elé tett filter segítségével lehet a legegyszerűbben végrehajtani.

```
LogoutFilter.java
```

```
import java.io.IOException;
import javax.servlet.Filter;
import javax.servlet.FilterChain;
import javax.servlet.FilterConfig;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpSession;

import org.apache.log4j.Logger;

/**
 * Servlet Filter implementation class LogoutFilter
 */
```

```

public class LogoutFilter implements Filter {

    /**
     * Default constructor.
     */
    private static final Logger log = Logger.getLogger(LogoutFilter.class);

    public LogoutFilter() {
    }

    /**
     * @see Filter#destroy()
     */
    public void destroy() {
    }

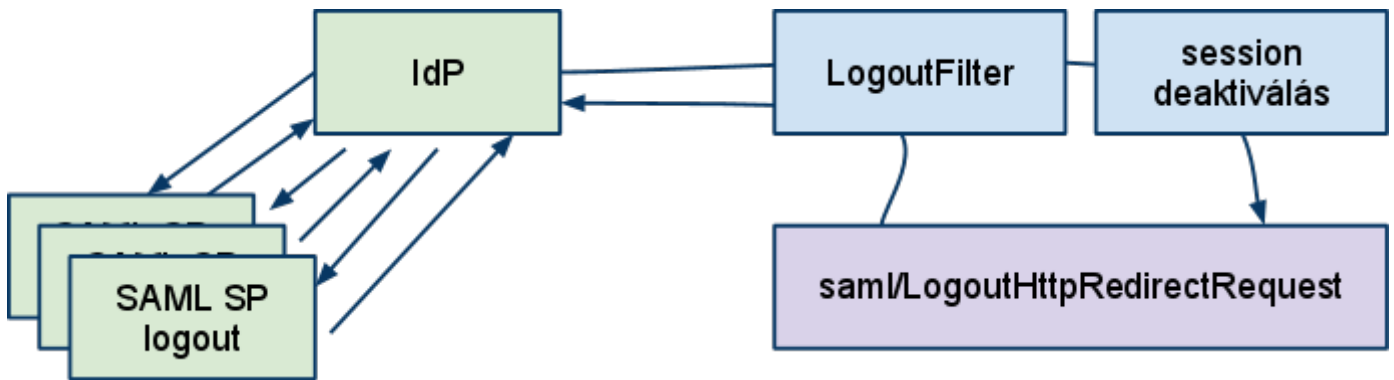
    /**
     * @see Filter#doFilter(ServletRequest, ServletResponse, FilterChain)
     */
    public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain)
    throws IOException, ServletException {
        chain.doFilter(request, response);
        log.debug("Enter into LoginFilter");
        HttpServletRequest servletRequest = ((HttpServletRequest) request);
        HttpSession session = servletRequest.getSession(false);
        if (session != null){
            log.debug("Remove application related session attributes.");
            session.removeAttribute("aai_auth");
        }
    }

    /**
     * @see Filter#init(FilterConfig)
     */
    public void init(FilterConfig fConfig) throws ServletException {
    }
}

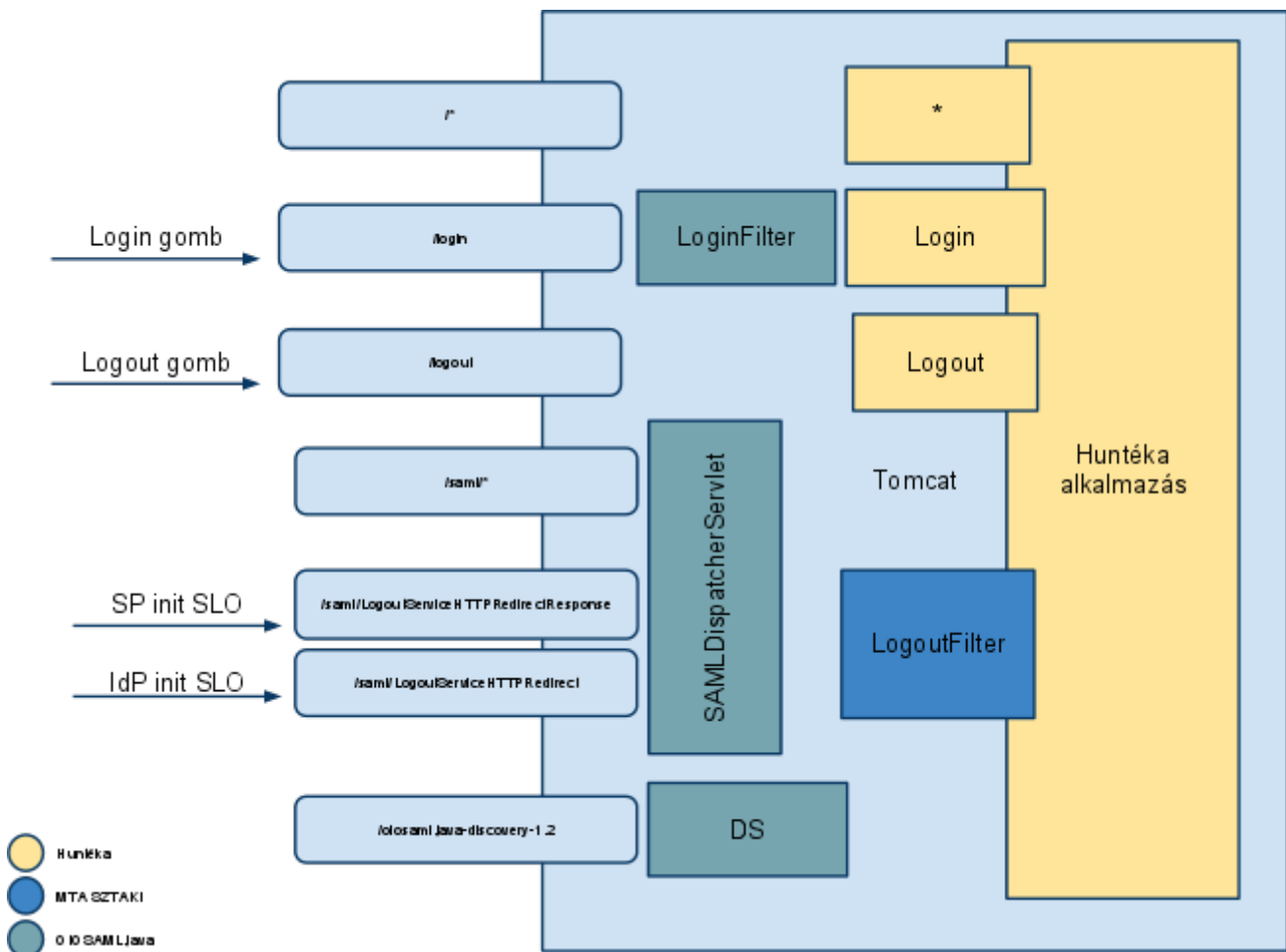
```

web.xml

```
...
<filter>
  <filter-name>LogoutFilter</filter-name>
  <filter-class>LogoutFilter</filter-class>
</filter>
<filter-mapping>
  <filter-name>LogoutFilter</filter-name>
  <url-pattern>/saml/LogoutServiceHTTPRedirect</url-pattern>
  <url-pattern>/saml/LogoutServiceHTTPRedirectResponse</url-pattern>
</filter-mapping>
...
```



## Rendszer architektúra



# OIOSAML.java, mint SP komponens

## Telepítés

Az oiosaml.java letöltése után a következő lépésekkel telepíthetjük alkalmazásunkhoz az oiosaml.java SP komponenst:

1. Másoljuk a lib/\* fileokat a könyvtári alkalmazás WEB-INF/lib könyvtárába
2. Egészítsük ki az alkalmazás web.xml leírófile-ját a következőkkel:

```
<context-param>
  <param-name>oiosaml-j.home</param-name>
  <param-value>/path/to/oiosaml.home</param-value>
</context-param>

<servlet>
  <servlet-name>SAMLDispatcherServlet</servlet-name>
  <servlet-class>dk.itst.oiosaml.sp.service.DispatcherServlet</servlet-class>
</servlet>
```

```
<servlet-mapping>
  <servlet-name>SAMLDispatcherServlet</servlet-name>
  <url-pattern>/saml/*</url-pattern>
</servlet-mapping>

<filter>
  <filter-name>LoginFilter</filter-name>
  <filter-class>dk.itst.oiosaml.sp.service.SPFilter</filter-class>
</filter>
<filter-mapping>
  <filter-name>LoginFilter</filter-name>
  <url-pattern>/protected/*</url-pattern>
</filter-mapping>
```

A SAMLDispatcherServlet servlet végzi el a SAML-lal kapcsolatos kéréseket, ajánlott a /saml/\* URL alá definiálni.

A LoginFilter filter által védett URL-ek (példánkban /protected/\*) csak SAML-os azonosítással érhetőek el, ezen URL-ek alatt érhetőek el különböző objektumokban TODO a SAML Assertion-ben átadott attribútumértékek.

Telepítés után a az OIOSAML.java ellenőrzi konfigurációját, és ha a rendszer nincs helyesen konfigurálva, elindul az autoconfig folyamat, ahol a rendszert a webes felületről konfigurálhatjuk. A konfigurációs folyamat végén az SP aláírt meta-adata letölthető.

## Konfiguráció

Az oiosaml.java konfigurációja az oiosaml-sp.properties file-ban található.

Telepítés után a következő módosításokat érdemes megtenni a fileban:

```
oiosaml-sp.assurancelevel=0
```

Az elfogadott azonosítási szintet állíthatjuk itt be. Mivel a föderációban illet egyelőre nem használunk, és egyes IdP-k nem támogatják (simpleSAMLphp) ezt az attribútumot, állítsuk 0-ra.

```
oiosaml-sp.uri.home=../
```

Az alkalmazásunk kezdő lapja. SLO után ide irányítódik át a felhasználó böngészője.

A konfigurációs beállítások teljes dokumentációja az alábbi URL-en található:

<https://svn.softwareborsen.dk/oiosaml.java/sp/trunk/docs/configuration.html>

## Meta-adat kezelés

A SAML SP komponens működéséhez elengedhetetlen, hogy a vele bizalmi szövetségben lévő IdP-k meta-adatait ismerje.

Az OIOSAML.java meta-adatformatumának a legfőbb jellemzője, hogy az egyes entitásokat egyenként külön file-ban kell eltárolni. Az szóban forgó file-okat a metadata/IdP alkönyvtárba kell helyezni XML formátumban, a név-konvenció csak annyit követel meg, hogy .xml-re végződjön a file neve.

Fontos tudni, hogy az OIOSAML.java csak induláskor tölti be a meta-adatokat, így ha azokat változtatjuk, az alkalmazást újra kell indítani, hogy a változtatások érvényre jussanak.

## Discovery Service

Ha az SP több IdP-vel is bizalmi szövetséget alkot, szükség van Discovery Service-re is, hogy a felhasználó eldönthesse, melyik IdP-nél kívánja magát azonosítani.

Az OIOSAML.java tartalmaz egy discovery service servlet-et, ami az IdP metadatáiból állít össze egy weboldalt, ahol a felhasználó kiválaszthatja a saját IdP-jét.

Telepítése az `oiosaml.java-discovery..war` file egyszerű *deploy-olásával* történik, *beállításra az `oiosaml-sp.properties` file `oiosaml-sp.discovery` változók használhatóak.*

A discovery service-hez kapcsolódó dokumentáció az alábbi URL-en található:

<https://svn.softwareborsen.dk/oiosaml.java/sp/trunk/docs/discovery.html>

## Metadata frissítés

A meta-adatok frissítése kapcsán meg kell oldani a HREF letölthető teljes meta-adatának entitásonként külön file-ba történő szétvágását.

Ezt a feladatot a `mdsigner-j-cli-2.0-rc2-jar-with-dependencies.jar` nevű program végzi el.

Működéséhez szükséges a metadata aláíró root certificate, ezzel lehet ellenőrizni a metadata hitelességét.

A programot célszerű rendszeresen, pl. naponta futtatni, egy külön folyamatban leválogatni a releváns IdP-ket, ellenőrizni a módosulásokat, módosulás esetén frissíteni a SAML SP metadata könyvtárát és újraindítani a webalkalmazást.

---

Változat #1

document-uploader hozta létre 2025-08-07 12:05:23 CEST

document-uploader frissítette 2025-08-07 12:05:23 CEST