Autenticazione in ambito distribuito

Analisi dei meccanismi di autenticazione in scenari distribuiti

Sommario

- Introduzione
- SSO
- OpenID
  - OpenID Provider
  - JOIDS Java OpenID Server
  - OpenID Authentication
- Spring
  - Spring security

Introduzione

Nel campo della sicurezza informatica, si definisce **autenticazione** (*in greco: αυθεντικός, da 'authentes'='autore'*) il processo tramite il quale un sistema informatico, un computer, un software o un utente, verifica la corretta, o almeno presunta, identità di un altro computer, software o utente che vuole comunicare attraverso
una connessione autorizzandolo ad usufruire dei relativi servizi associati.

Come autenticare

Come autenticare un utente?

- tramite qualcosa che conosce
  - Password, PIN
- tramite qualcosa che ha
  - Tesserino, chiave hardware
- tramite qualcosa che è
  - Identificazione biometrica: Impronte, Retina

Autorizzazioni

- Le autorizzazioni sono i privilegi associati all'utente ovvero "ciò che l'utente può fare".
- Spesso si fa confusione fra autenticazione ed autorizzazione; sono due aspetti fortemente correlati, ma diversi che si possono valutare in momenti differenti.

Web Authentication

1. Click su un link

Click su un link

Web Authentication

1. Click su un link
2. Il server verifica se la risorsa a cui si vuole accedere è protetta

Web Authentication

Devi autenticarti

1. Click su un link
2. Il server verifica che la risorsa a cui si vuole accedere è protetta
3. Se l'utente non è autenticato il server richiederà all'utente di autenticarsi (HTTP response code, redirect to a particular web page)

Web Authentication

1. Click su un link
2. Il server verifica che la risorsa a cui si vuole accedere è protetta
3. Se l'utente non è autenticato il server richiederà all'utente di autenticarsi (HTTP response code, redirect to a particular web page)
4. In base al meccanismo di autenticazione il browser dovrà fornire un meccanismo per identificare l'utente (Form authentication, cookie, X.509)

Identificazione utente

Web Authentication

Invio credenziali

1. Click su un link
2. Il server verifica che la risorsa a cui si vuole accedere è protetta
3. Se l'utente non è autenticato il server richiederà all'utente di autenticarsi (HTTP response code, redirect to a particular web page)
4. In base al meccanismo di autenticazione il browser dovrà fornire un meccanismo per identificare l'utente (Form authentication, cookie, X.509)
5. Le credenziali verranno inviate al server

Web Authentication

1. Click su un link
2. Il server verifica che la risorsa a cui si vuole accedere è protetta
3. Se l'utente non è autenticato il server richiederà all'utente di autenticarsi (HTTP response code, redirect to a particular web page)
4. In base al meccanismo di autenticazione il browser dovrà fornire un meccanismo per identificare l'utente (Form authentication, cookie, X.509)
5. Le credenziali verranno inviate al server
6. Il server verificherà l'identità dell'utente

Autenticazione a molti servizi

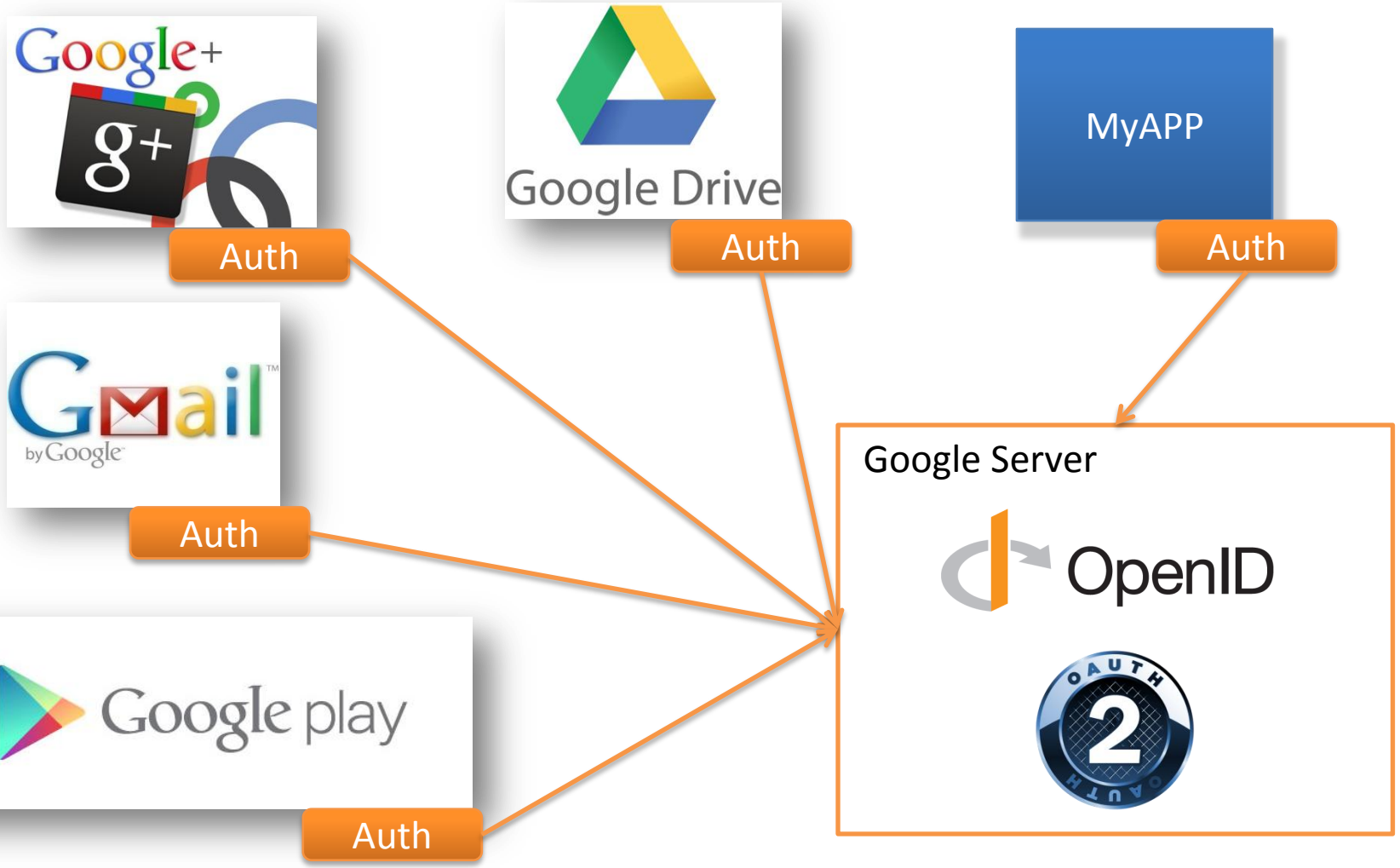
Devo autenticarmi a molti servizi!!!

- Tante password e nomi utente da ricordare …
  - … me li scrivo da qualche parte ☹
- Uso sempre la stessa password …
  - … vorrei avere lo stesso nome utente, ma è già impegnato ☹
  - … potrei usare la stessa password comunicandola a siti non affidabili ☹

SSO Single Sign On

- Metodo per il controllo di accesso che permette di autenticarsi in molti sistemi inserendo le proprie credenziali una volta sola
- L'utente non dovrà reinserire le proprie credenziali in ogni sistema a cui accede

Google Authentication System



SSO Centralizzato

MyWeb App

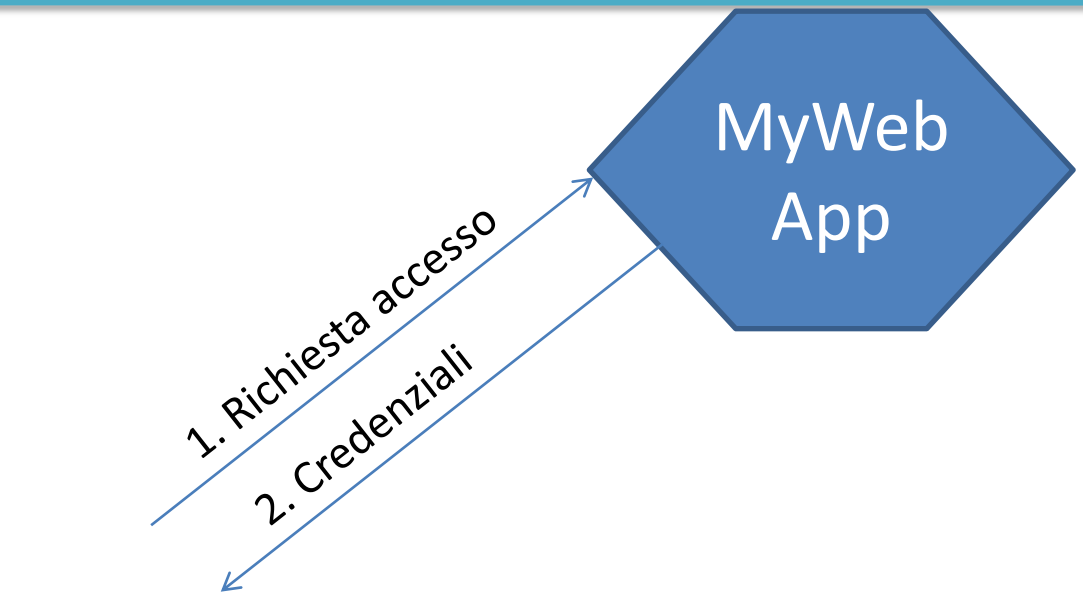
1. Richiesta accesso

1. L'utente vuole accedere ad una zona protetta e non è autenticato

Registro Utenti

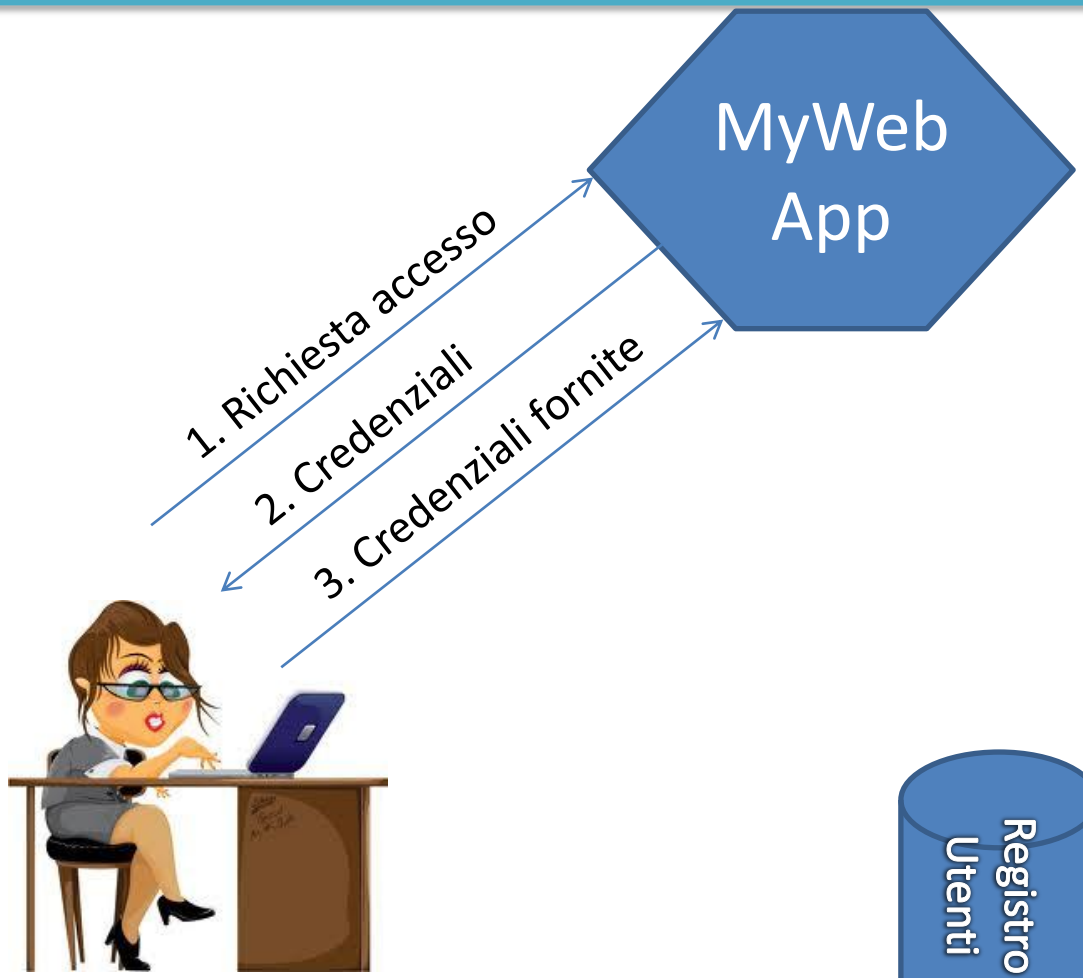
Centralized authentication Server
Es. Microsoft Passport

SSO Centralizzato



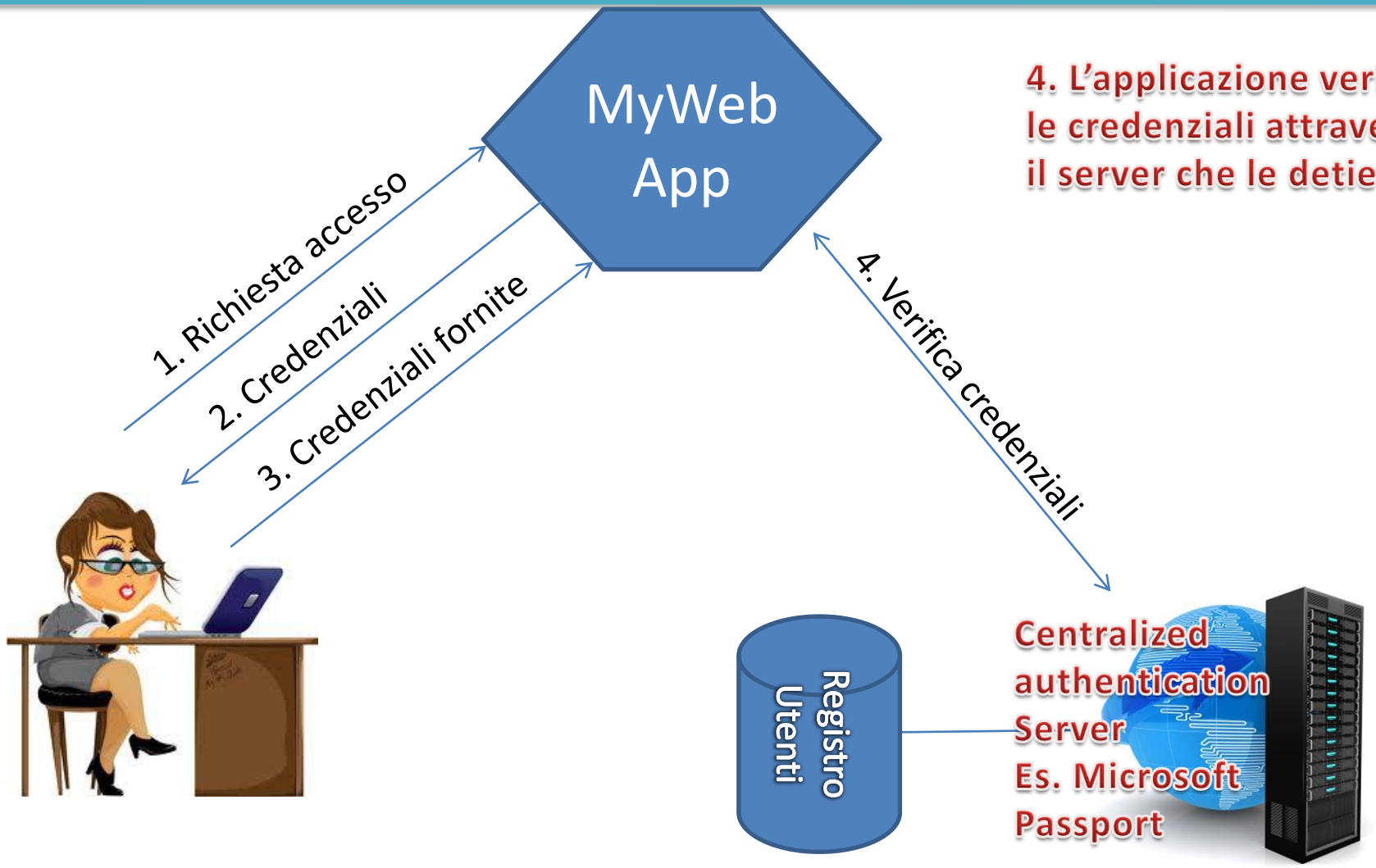
2. L'applicazione web richiederà le credenziali di autenticazione

SSO Centralizzato

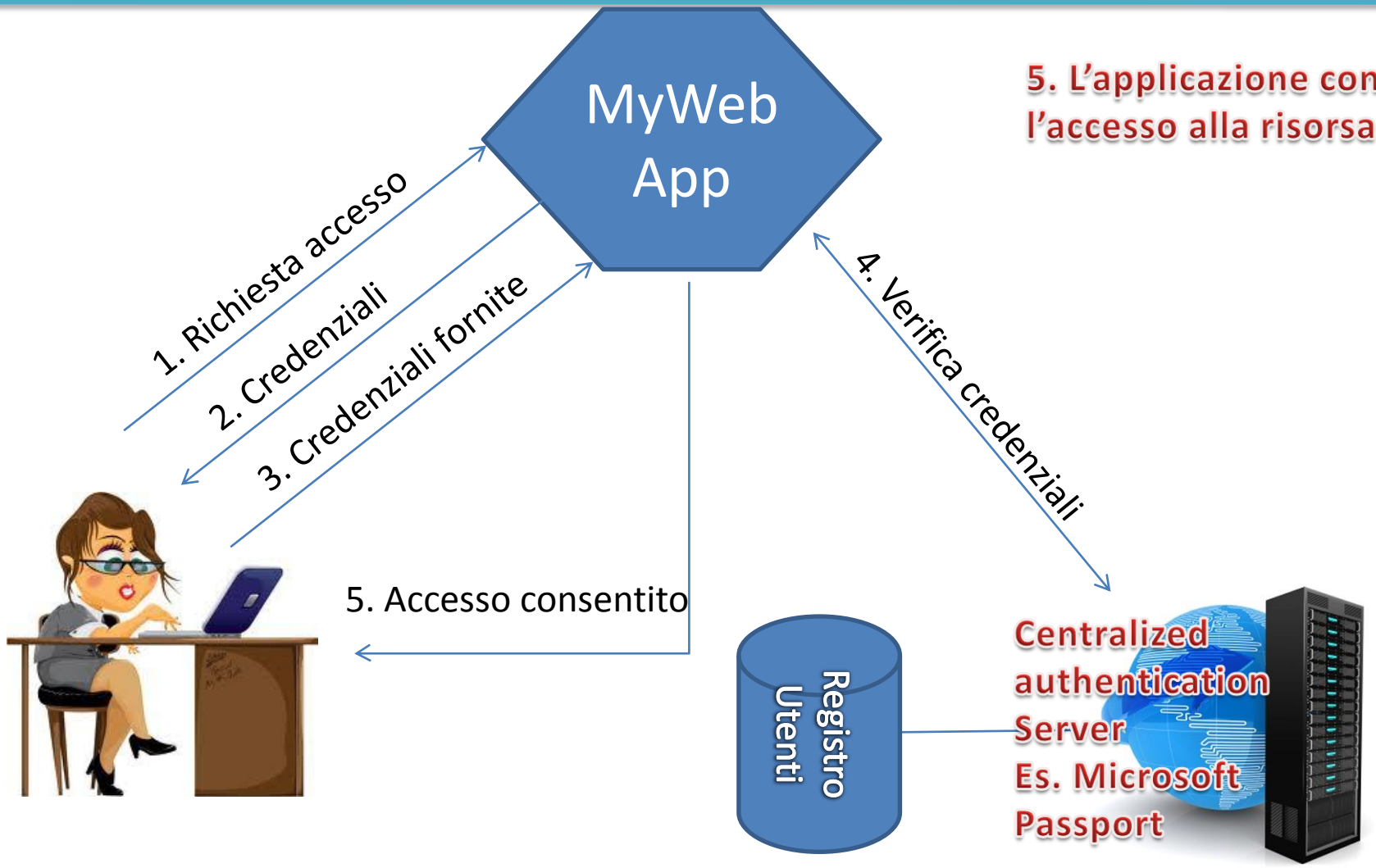


3. L'utente fornisce le credenziali richieste ad esempio username e password

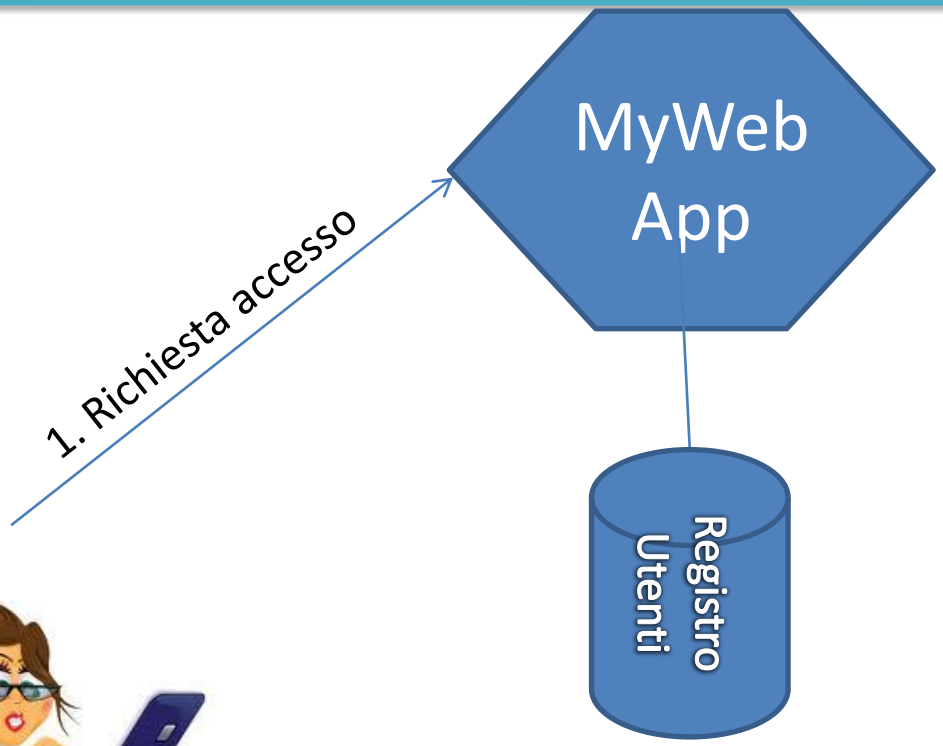
SSO Centralizzato



SSO Centralizzato

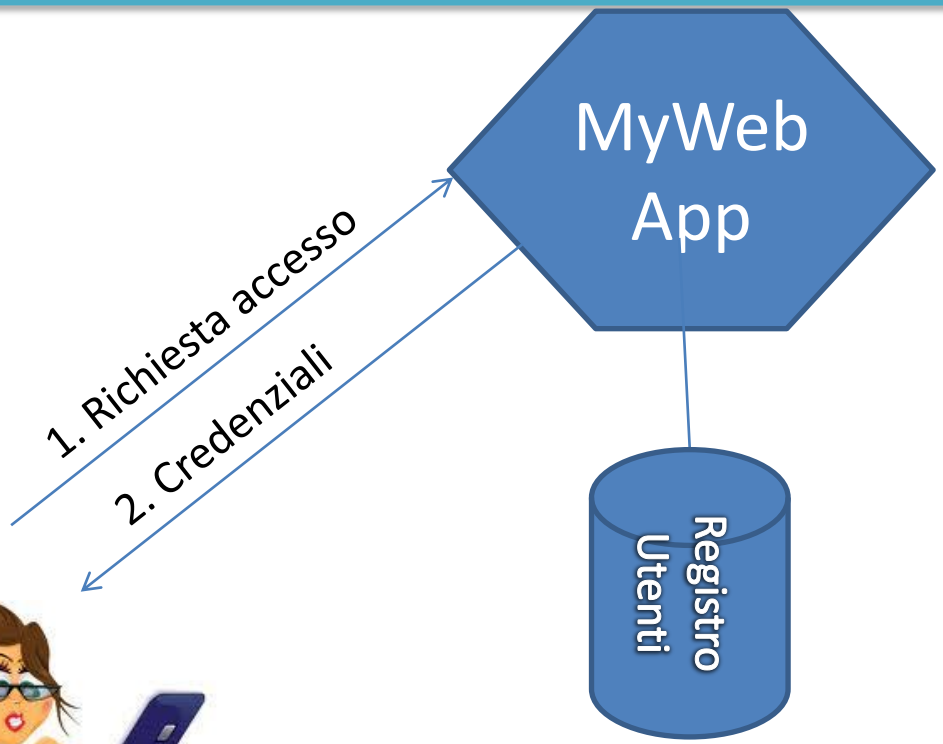


SSO Decentralizzato



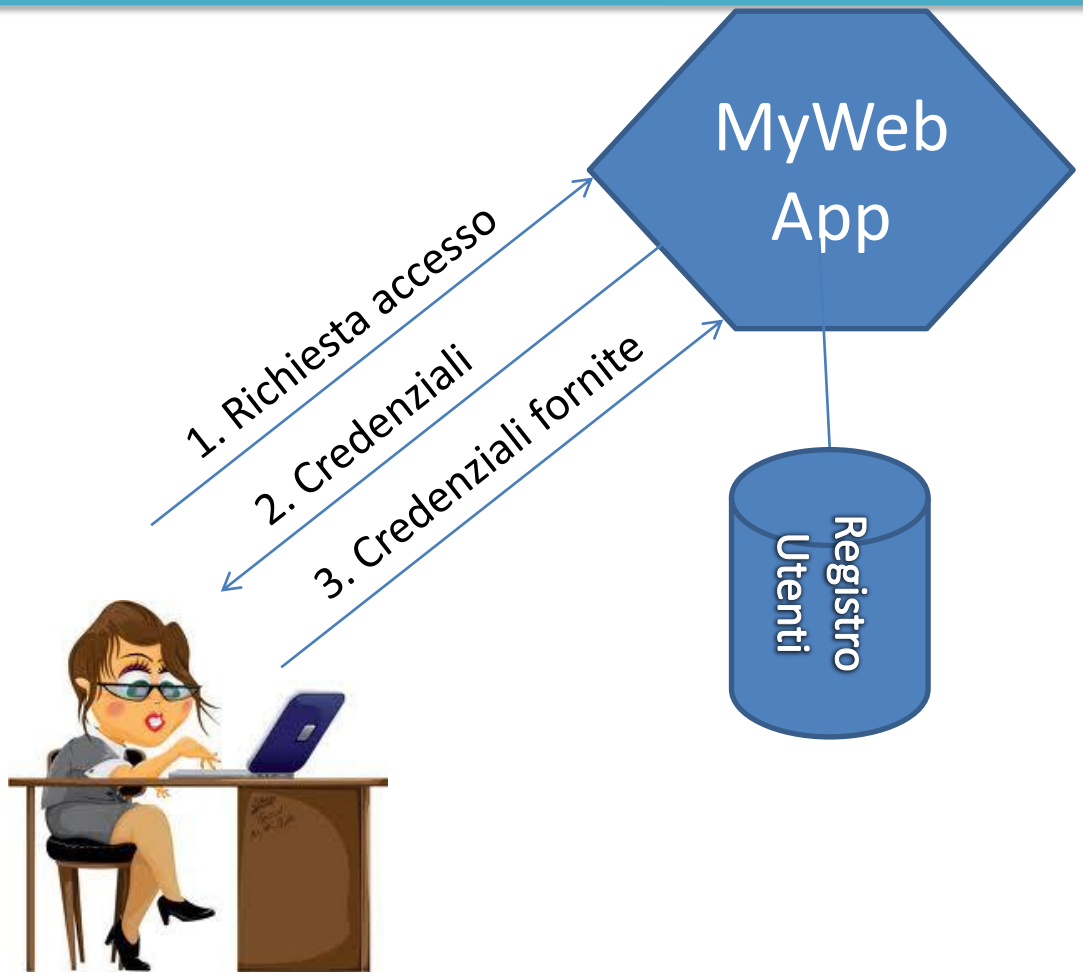
1. L'utente vuole accedere ad una zona protetta e non è autenticato

SSO Decentralizzato



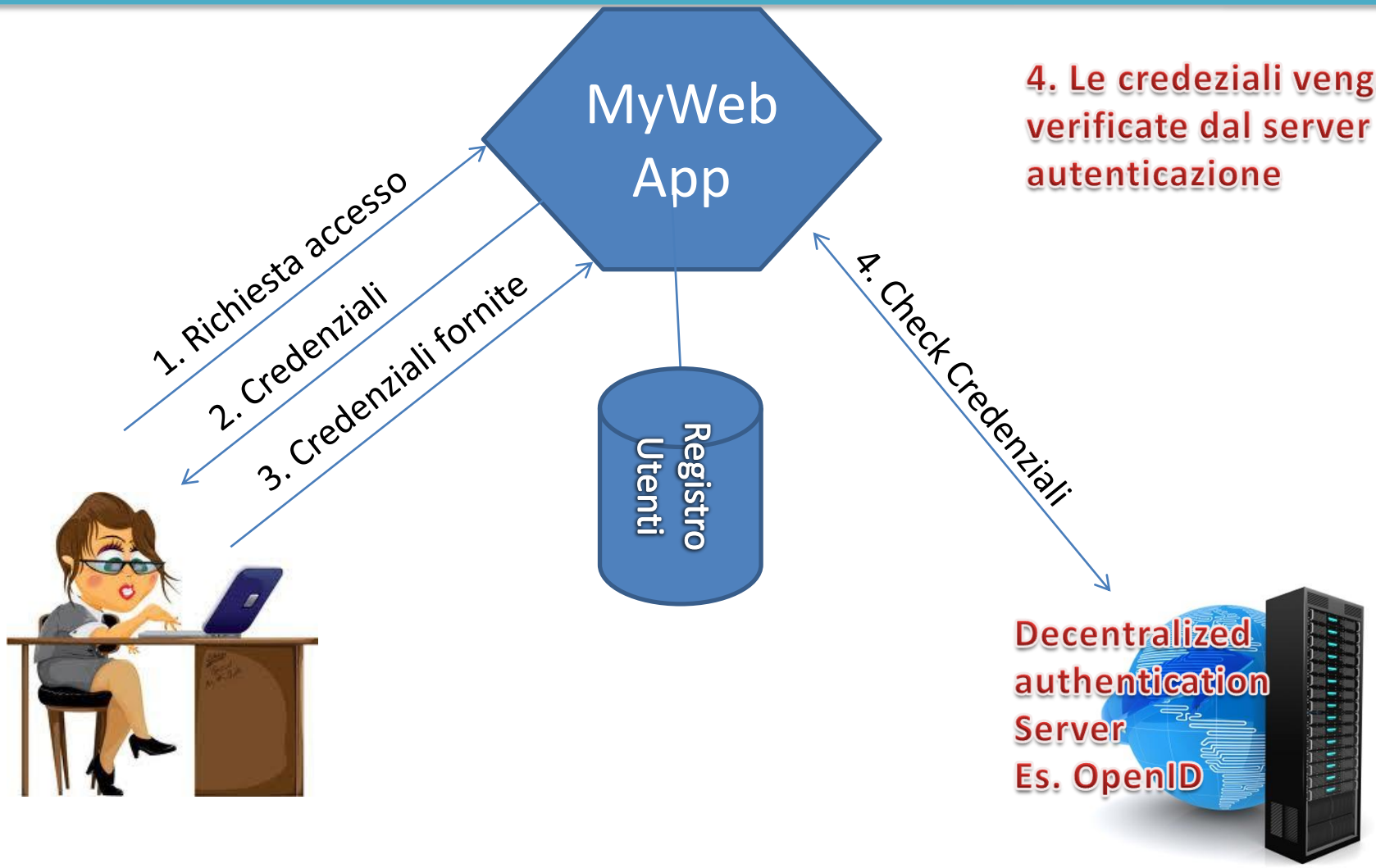
2. L'applicazione web richiederà le credenziali di autenticazione.

SSO Decentralizzato

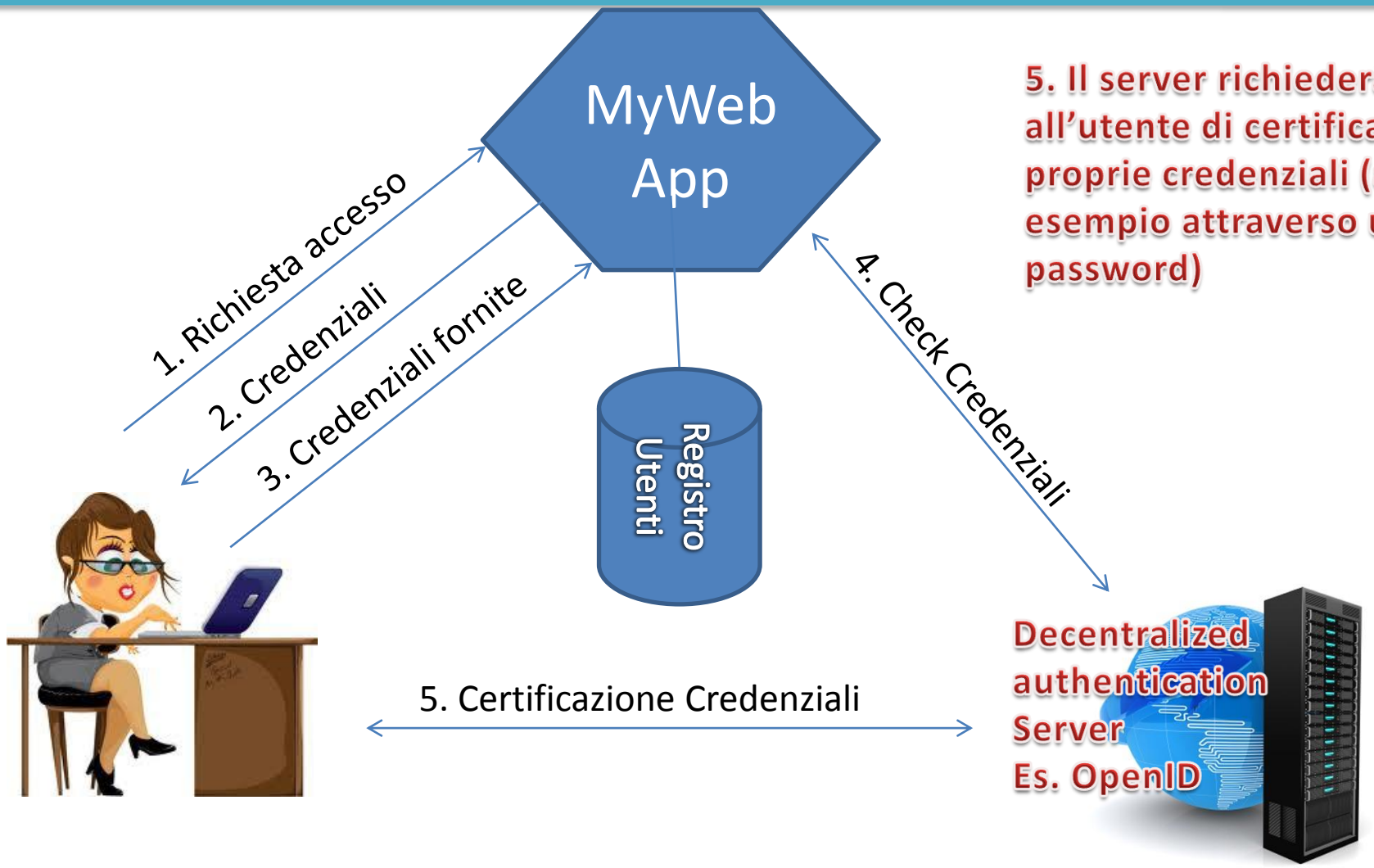


3. L'utente fornisce le proprie credenziale, senza però fornire la password. L'applicazione web dovrà occuparsi inoltre di individuare lo UserAgent dell'utente.

SSO Decentralizzato

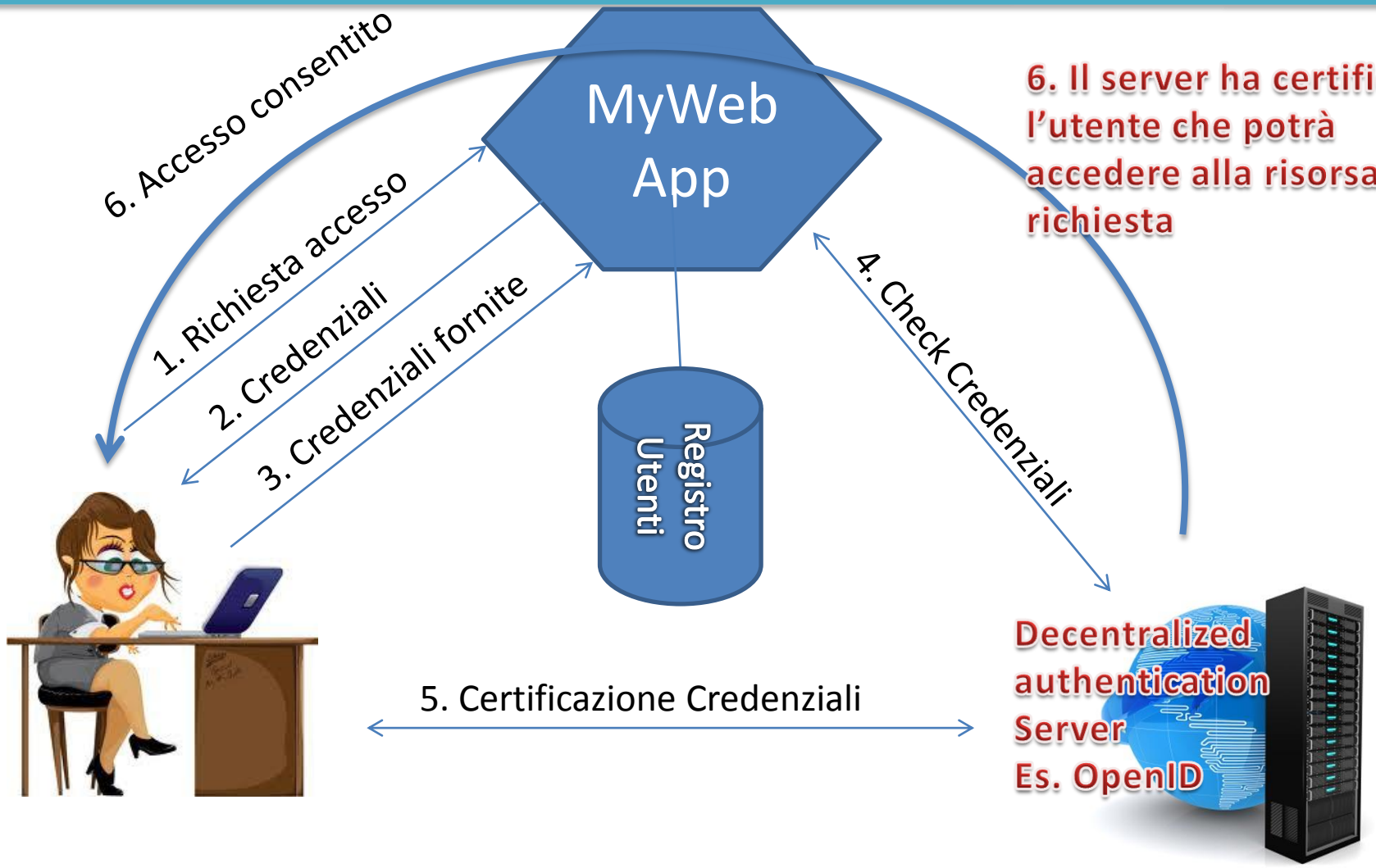


SSO Decentralizzato



5. Il server richiederà all'utente di certificare le proprie credenziali (ad esempio attraverso una password)

SSO Decentralizzato



meccanismo
decentralizzato per il
Single Sign On

Cos'è?

- Un OpenID non è altro che una URI

  https://username.myopenid.com

- Permette di utilizzare un account esistente per accedere a diverse applicazioni WEB (e non), **senza la necessità di creare una nuova password**.

Storia

OpenID è stato creato nell'estate 2005 con lo scopo di risolvere un problema non facilmente risolto da altre tecnologie sul mercato.

È stato sviluppato da un gruppo Open Source.

È uno standard libero ovvero chiunque può usare un OpenID o diventare un OpenID provider.

The OpenID Foundation

- Nasce con lo scopo di **promuovere**, **proteggere** e **nutrire** l'OpenID community
- Organizzazione non-profit fondata a giugno 2007.
- Sostenuta da individui e da diverse compagnie.

OpenID

OIDF Organization

OpenID

Benefici

- Accelerate Sign Up Process at Your Favorite Websites
- Reduce Frustration Associated with Maintaining Multiple Usernames and Passwords
- Gain Greater Control Over Your Online Identity
- Minimize Password Security Risks

OpenID

Specification

Specifiche complete

- OpenID Authentication 2.0
- OpenID Attribute Exchange 1.0
- OpenID Provider Authentication Policy Extension 1.0
- OpenID Simple Registration Extension 1.0
- Yadis Discovery Protocol

OpenID Provider

Diverse implementazioni:

- PHP (Community-ID, Prairie, PHP OpenID Server, …)
- Ruby (local-openid, …)
- Pyton (DjangoID, …)
- Perl (Packetizer OpenID Server, …)
- .Net (DotNetOpenId, …)
- Altri (OpenLink Data Spaces (ODS), …)

Java OpenID Provider

- NetMesh InfoGrid LID Java.
- Atlassian's Crowd.
- WSO2 Identity Solution.
- JOIDS(Java OpenID Server).
- OpenASelect Server.

Java OpenID Server

Server OpenID sviluppato tramite OpenID4Java, Spring, Hibernate e Velocity

- Provider di OpenID nei seguenti formati
  - http(s)://username.example.com
  - http(s)://example.com/username
- Multi-domain support.
- Multi-user support.
- Account management.
- Multilingual support.

Java OpenID server

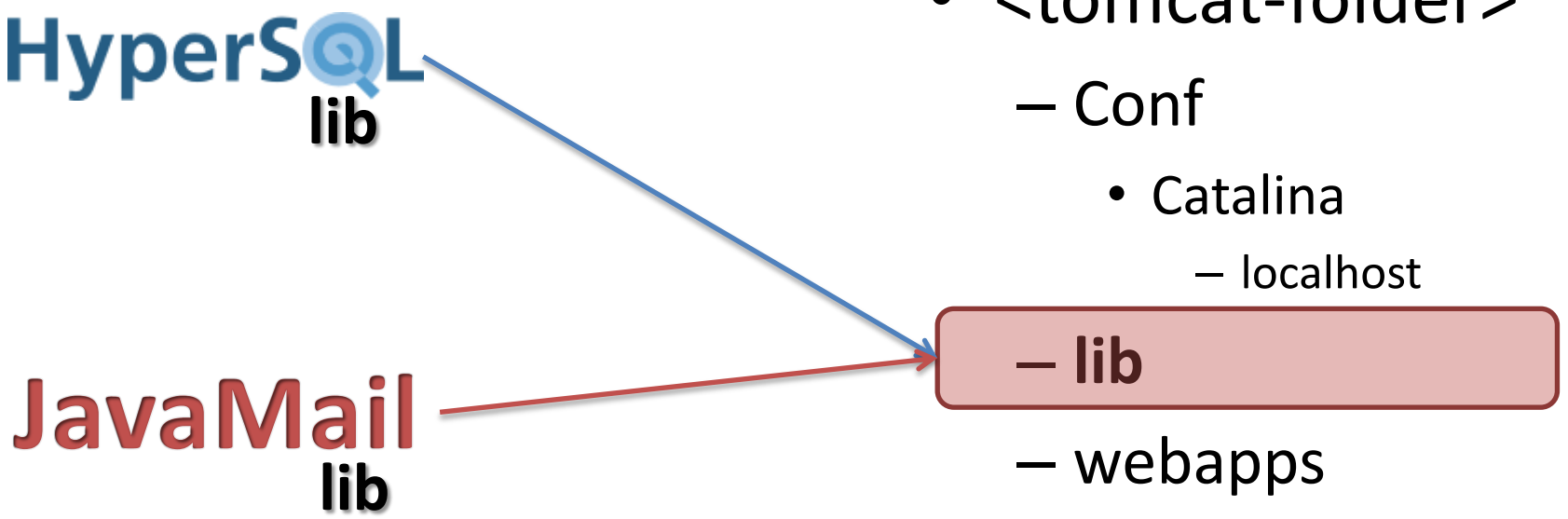
http://localhost:9080/joids

Ingredienti

- Tomcat Server
- HyperSQLDB library
- JavaMail
- joids.war
  - http://code.google.com/p/openid-server/

Preparazione

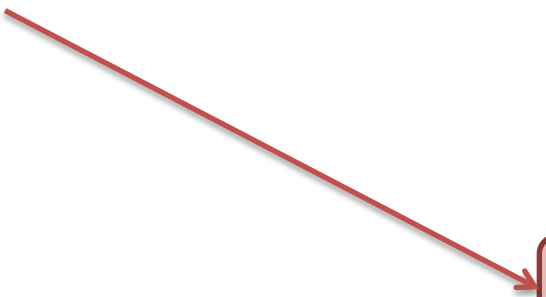
1. Scaricare e copiare le librerie necessarie nella cartella lib di tomcat



Preparazione

2. Copiare il war scaricato (nel caso il nome non sia joids.war rinominare il file)

joids.war



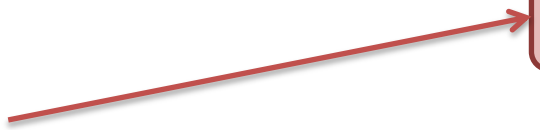
- <tomcat-folder>
  - conf
    - Catalina
      - localhost
  - **lib**
  - webapps

Preparazione

3. Configurare joids.xml e copiarlo dentro localhost.
joids.xml non è altro che il context.xml di joids.war presente in META-INF

- <tomcat-folder>
  - conf
    - Catalina
      - localhost
  - **lib**
  - webapps

joids.xml



OpenID

Esempio autenticazione

http://localhost:8080/auth

OpenID Authentication 2.0

December 5, 2007

Abstract

OpenID Authentication provides a **way to prove that an end user controls an Identifier**.

… … end user can prove their Identity to a Relying Party **without having to leave their current Web page**.

OpenID Authentication uses **only standard HTTP(S) requests and responses**, … …

Terminologia

- OP OpenID provider:
  - Fornitore di OpenIDs (JOIDS)
- RP Relying Party
  - Applicazione che vuole utilizzare un OpenID (auth)
- UserAgent
  - Identifica il browser dell'utente
- User-Supplied Identifier
  - Identificatore fornito dall'utente (tipicamente la URI che è detenuta dall'utente)

Caratteristiche

- **Protocol Messages (4.1)**
  - The OpenID Authentication protocol messages are mappings of plain-text keys to plain-text values. 4

Key-Value Form encoded:

```
mode:error
error:This is an example message
```

x-www-urlencoded, as in a HTTP POST body or in a URL's query string ([RFC3986] section 3):

```
openid.mode=error&openid.error=This%20is%20an%20example%20message
```

Processo di autenticazione

Il processo di autenticazione si compone di diverse parti

- **Initiation**
- **Normalization**
- **Discovery**
- **Associations** (optional if stateful authentication)
- **Authentication request**
- **Relying Party verify**

Initiation

- Prima fase del processo di autenticazione.
- L'utente presenta il proprio UserAgent e il proprio OpenID (User-Supplied Identifier) all'RP, ovvero, all'applicazione web alle quale si vuole autenticare.
  - ES: localhost:9080/joids/test
    da Chrome
- L'RP dovrà comunicare con l'OP

Normalization

Il processo di normalizzazione è a carico dell'RP.

- User-Supplied Identifier (localhost:9080/joids/test)
  - URI fornita dall'utente all'RP

- Claimed Identifier (http://localhost:9080/joids/test)
  - URI presentata all'OP dall'RP

Normalization

User's Input	Identifier	Type	Discussion
example.com	http://example.com/	URL	A URI with a missing scheme is normalized to a http URI
http://example.com	http://example.com/	URL	An empty path component is normalized to a slash
https://example.com/	https://example.com/	URL	https URIs remain https URIs
http://example.com/user	http://example.com/user	URL	No trailing slash is added to non-empty path components
http://example.com/user/	http://example.com/user/	URL	Trailing slashes are preserved on non-empty path components
http://example.com/	http://example.com/	URL	Trailing slashes are preserved when the path is empty
=example	=example	XRI	Normalized XRIs start with a global context symbol
xri://=example	=example	XRI	Normalized XRIs start with a global context symbol

Discovery

- RP dice…
  - Ho un id normalizzato…
  - … cosa me ne faccio?

Discovery

Discovery

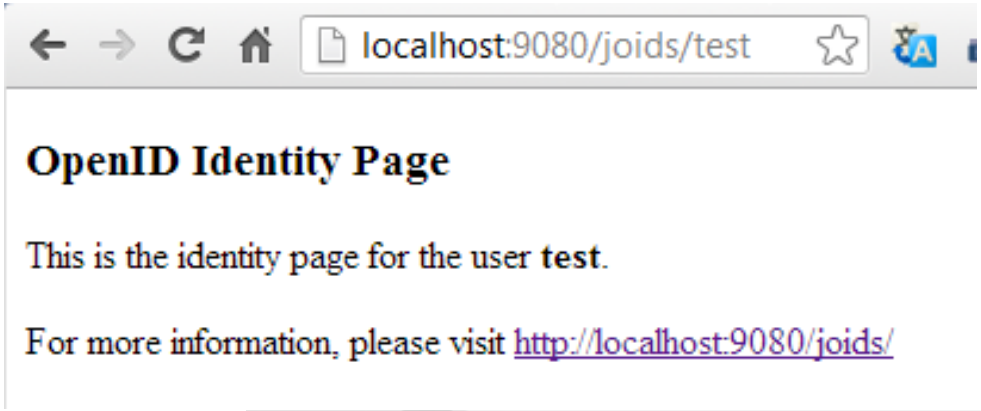
- Individuazione dei servizi legati ad un identificativo.
- Processo di discovery per URL
  1. YADIS Discovery
     - Nel caso YADIS abbia successo viene fornito un XSRDS eXtensible Resource Descriptor Sequence con le informazioni sui servizi di autenticazione
  2. Se YADIS Fallisce HTML Discovery

YADIS Spec 1.0

"The purpose of the Yadis protocol is to enable a Relying Party to obtain a Yadis Resource Descriptor that **describes the services available** using a **Yadis ID**."

- Lo YadisID è un OpenID, un XRI, un LID…?
- Quali sono i servizi disponibili per questo ID?
- Ci sono altri servizi disponibili?

HTML Discovery



Pagina associata al ClaimIdentifier
http://localhost:9080/joids/test

Riferimenti all'OP

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
3          "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
4  <html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
5  <head>
6  <link rel="openid.server" href="http://localhost:9080/joids/server" />
7  <link rel="openid2.provider" href="http://localhost:9080/joids/server" />
8  <title>Identity Endpoint For test</title>
9  </head>
10 <body>
11 <h3>OpenID Identity Page</h3>
12 <p>This is the identity page for the user <strong>test</strong>.
13 </p>
14 <p>For more information, please visit <a
15         href="http://localhost:9080/joids/">http://localhost:9080/joids/</
   a></p>
16 </body>
```

RP Discovery result

```
152  [DEBUG] [http-bio-8080-exec-4 10:23:59] (org.openid4java.discovery.html.CyberNekoDOMHtmlParser:setResult:148)
      Found OpenID2 endpoint: http://localhost:9080/joids/server
153  [DEBUG] [http-bio-8080-exec-4 10:23:59] (org.openid4java.discovery.html.CyberNekoDOMHtmlParser:setResult:148)
      Found OpenID2 endpoint: http://localhost:9080/joids/server
154  [DEBUG] [http-bio-8080-exec-4 10:23:59] (org.openid4java.discovery.html.CyberNekoDOMHtmlParser:parseHtml:65)
     HTML discovery result:
155  ClaimedID:http://localhost:9080/joids/test
156  OpenID2-endpoint:http://localhost:9080/joids/server
157  OpenID1-endpoint:http://localhost:9080/joids/server
```

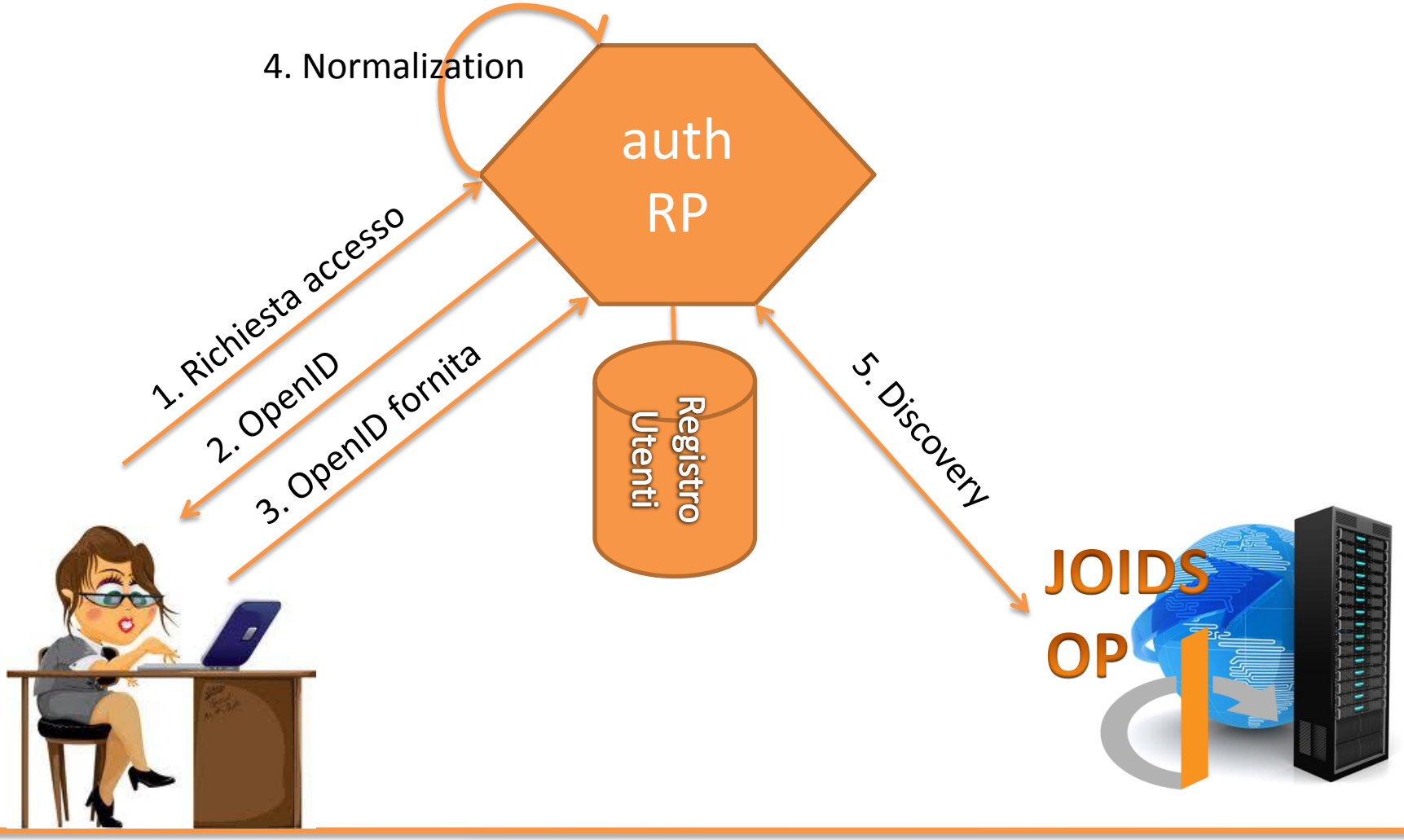
HTML discovery result:

ClaimedID: http://localhost:9080/joids/test

OpenID2-endpoint: http://localhost:9080/joids/server

OpenID1-endpoint: http://localhost:9080/joids/server

A che punto siamo?



Associations

- Comportamento previsto in maniera opzionale.
- L'RP e l'OP si scambiano un "segreto" tramite **Diffie-Hellman Key Exchange.**
- La chiave scambiata sarà utilizzata per firmare le successive chiamate al server.
- Nel caso non sia possibile questo processo si passerà alla modalità **Stateless Mode**.

Associations

- Association_type
  - HMAC-SHA1 - 160 bit key length;
  - HMAC-SHA256 - 256 bit key (RECOMMENDED).
- Session_type
  - no-encryption
  - DH-SHA1
  - DH-SHA256

Associations

```
226 [DEBUG] [http-bio-8080-exec-4 10:24:00] (org.openid4java.message.AssociationRequest:createAssociationRequest:140)
    Created association request:
227 openid.ns:http://specs.openid.net/auth/2.0
228 openid.mode:associate
229 openid.session_type:DH-SHA256
230 openid.assoc_type:HMAC-SHA256
231 openid.dh_consumer_public:AMjLbby7N16yM6sm+8RoKI+UX0JfsL06tW8SaMKvtoneqIWsjkZnnhrGxx14mX1Kfbm6WHACjIv0BmyozY5g3nxVtBd/rJ
    gutjMQeZ+LiKarzZXAoy7wLxYwv5HoG5NTTrRMUl+YlsLalFPLBBxEoq8A2Cz9ClEF7ou2iRFtMc++
```

RP

```
34 2013-01-15 10:24:00,863 DEBUG [org.openid4java.message.AssociationResponse:107] -
35 Created association response:
36 ns:http://specs.openid.net/auth/2.0
37 session_type:DH-SHA256
38 assoc_type:HMAC-SHA256
39 assoc_handle:1358241840393-0
40 expires_in:1799
41 dh_server_public:GzZWc1p+xLCQ8sIxyIXTWiga9aCR1Jdey5cah4K31+1HwCmMcnvYC/yZcbBsyFeGYhE4oUuTmu6ghL70zJErAeiWczj3ch
42 enc_mac_key:0j4MPF1IoSuIv28wlT2keUeQhSdlEN1vMiaKK9myAlw=
```

OP

```
265 [DEBUG] [http-bio-8080-exec-4 10:24:00] (org.openid4java.message.ParameterList:createFromKeyValueForm:198)
    Creating parameter list from key-value form:
266 ns:http://specs.openid.net/auth/2.0
267 session_type:DH-SHA256
268 assoc_type:HMAC-SHA256
269 assoc_handle:1358241840393-0
270 expires_in:1799
271 dh_server_public:GzZWc1p+xLCQ8sIxyIXTWiga9aCR1Jdey5cah4K31+1HwCmMcnvYC,
    hWm8bx79P/8+NOYgkcM9UB3XyhzCeW4kK6qBlh666U/AaTwuUN32TZc3kPE7u9UQjrRi3WI3
272 enc_mac_key:0j4MPF1IoSuIv28wlT2keUeQhSdlEN1vMiaKK9myAlw=
```

RP

```
...ion_type:DH-SHA256
assoc_type:HMAC-SHA256
assoc_handle:1358241840393-0
expires_in:1799
...erver_public:GzZWc1p...
```

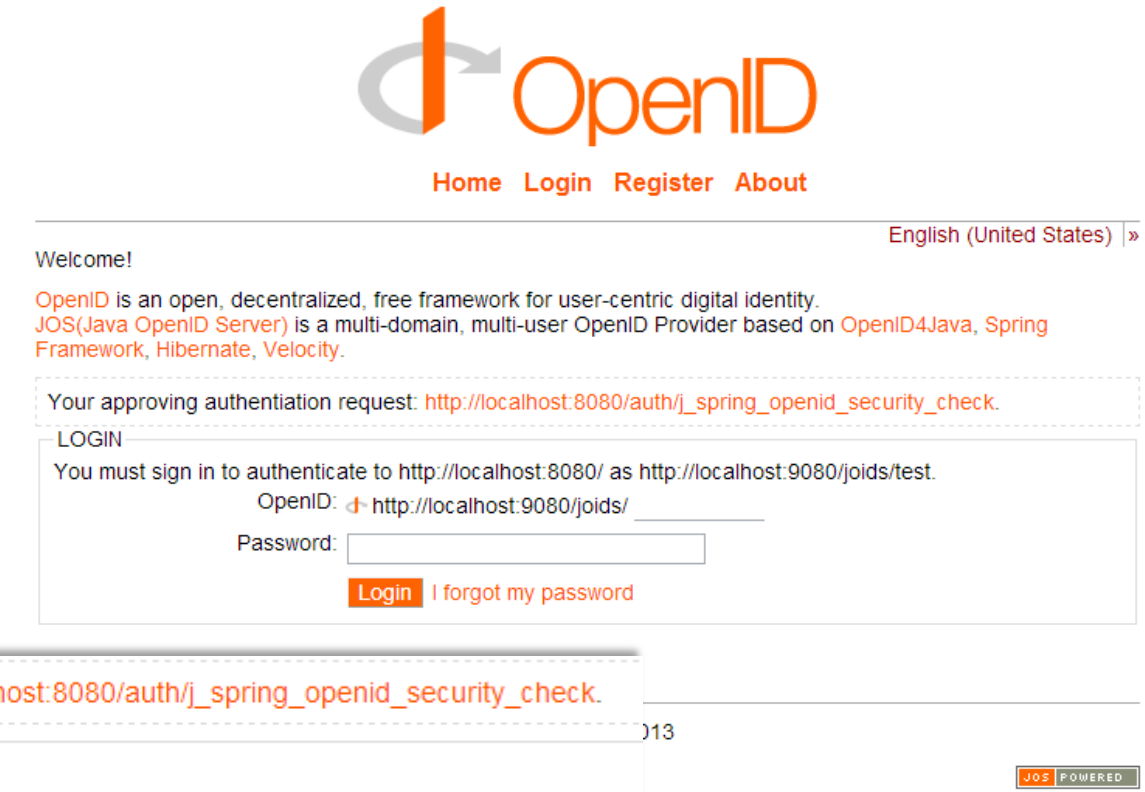
Authetication Request

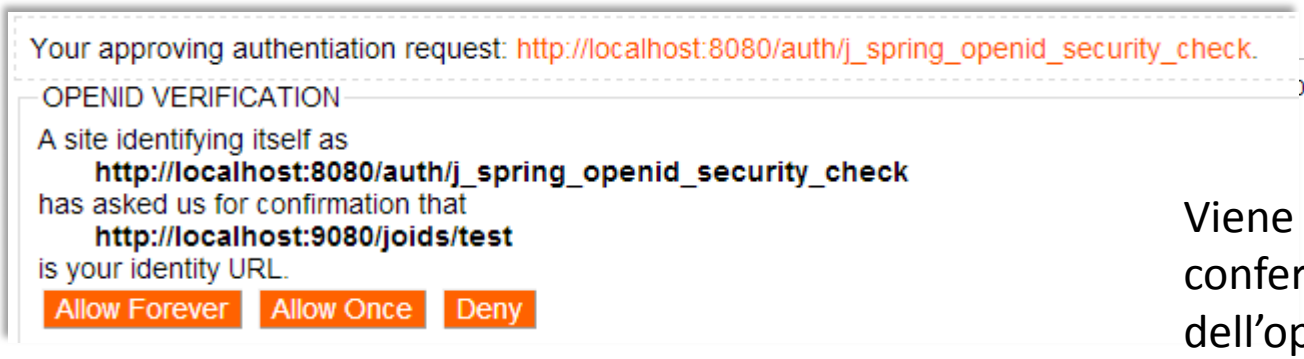
- L'RP effettua richiesta di autenticazione per l'OpenId Test

```
373 [DEBUG] [http-bio-8080-exec-4 10:24:00] (org.openid4java.message.AuthRequest:createAuthRequest:103
    Created auth request:
374 openid.ns:http://specs.openid.net/auth/2.0
375 openid.claimed_id:http://localhost:9080/joids/test
376 openid.identity:http://localhost:9080/joids/test
377 openid.return_to:http://localhost:8080/auth/j_spring_openid_security_check
378 openid.realm:http://localhost:8080/
379 openid.assoc_handle:1358241840393-0
380 openid.mode:checkid_setup
```

Authetication + Allow

L'utente viene rediretto nella pagine di login di JOIDS





Viene richiesta conferma di utilizzo dell'openid

Positive auth response

- L'OP conferma l'identità e firma la consegna

```
133 2013-01-15 10:28:06,827 DEBUG [org.openid4java.message.AuthSuccess:106] -
        Created positive auth response:
134 openid.ns:http://specs.openid.net/auth/2.0
135 openid.op_endpoint:http://localhost:9080/joids/server
136 openid.claimed_id:http://localhost:9080/joids/test
137 openid.response_nonce:2013-01-15T09:28:06Z0
138 openid.mode:id_res
139 openid.identity:http://localhost:9080/joids/test
140 openid.return_to:http://localhost:8080/auth/j_spring_openid_security_check
141 openid.assoc_handle:1358241840393-0
142 openid.
        signed:op_endpoint,claimed_id,identity,return_to,response_nonce,assoc_handle
143 openid.sig:
144
145 2013-01-15 10:28:06,834 DEBUG [org.openid4java.association.Association:267] -
        Calculated signature: GuAdArUatgDt2H2CKejDfM8Rdm6o9THF+ULAXMukLrs=
146 2013-01-15 10:28:06,836 DEBUG [org.openid4java.message.AuthSuccess:364 ] -
        Added signature: GuAdArUatgDt2H2CKejDfM8Rdm6o9THF+ULAXMukLrs=
```

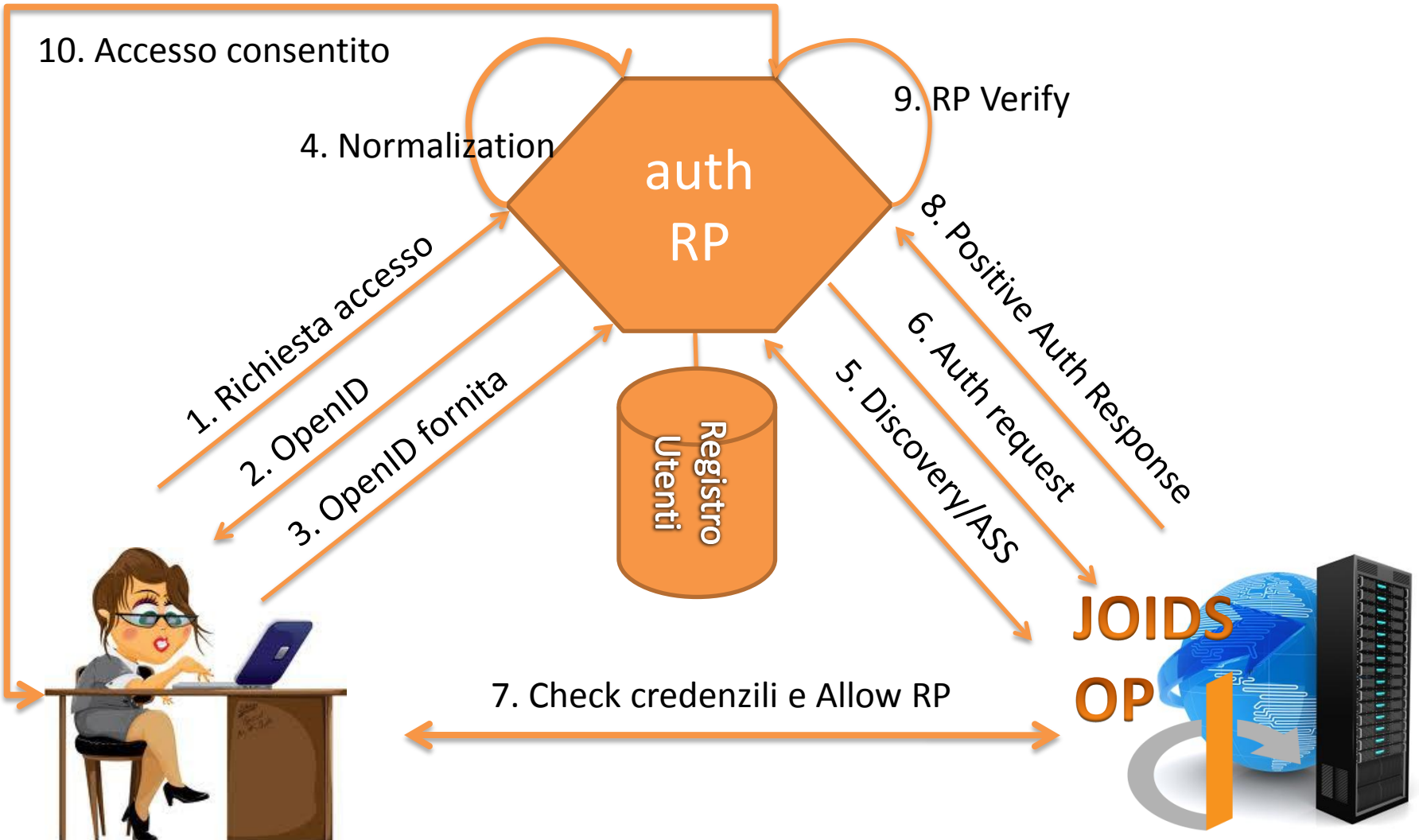
Relying Party verify

Verifying Assertions

1. URL di ritorno corretta (“openid.return_to”)
2. Le informazioni mandate dall’OP corrispondono con quelle individuate dall’RP nella fase di discovery
3. Verifica del nonce (“openid.response_nonce”)
4. Correttezza della firma

Se TUTTE le quattro condizioni sono verificate l’utente è autenticato.

Riassumendo…



Applicazione Auth

Dettagli Implementativi

Spring

- Framework per la produzione di applicazioni distribuite
- Si compone di vari moduli noi vedremo
  - Spring Framework (anche Spring Core) 3.1
  - Spring MVC
  - Spring Security 3.1

Spring Web App

HTTP REQUEST

Spring DispatcherServlet

Spring SecurityFilterChain

Spring MVC Controller (@Controller)

BL (@Service, @Component, @Repository)

Servlet container Ex. Tomcat

Spring security

- Il modulo di spring che si occupa di sicurezza è la spring security
- Basa il suo funzionamento sul pattern CHAIN-OF-RESPOSIBILITY
  - È possibile connettere più meccanismi di autenticazione già implementati nel framework
  - È possibile aggiungere meccanismi senza danneggiare i vecchi limitando di conseguenza la possibilità di produrre falle nella sicurezza

Filter chain

Alias	Filter Class	Namespace Element or Attribute
CHANNEL_FILTER	ChannelProcessingFilter	http/intercept-url@requires-channel
CONCURRENT_SESSION_FILTER	ConcurrentSessionFilter	session-management/concurrency-control
SECURITY_CONTEXT_FILTER	SecurityContextPersistenceFilter	http
LOGOUT_FILTER	LogoutFilter	http/logout
X509_FILTER	X509AuthenticationFilter	http/x509
PRE_AUTH_FILTER	AstractPreAuthenticatedProcessingFilter Subclasses	N/A
CAS_FILTER	CasAuthenticationFilter	N/A
FORM_LOGIN_FILTER	UsernamePasswordAuthenticationFilter	http/form-login
BASIC_AUTH_FILTER	BasicAuthenticationFilter	http/http-basic
SERVLET_API_SUPPORT_FILTER	SecurityContextHolderAwareRequestFilter	http/@servlet-api-provision
JAAS_API_SUPPORT_FILTER	JaasApiIntegrationFilter	http/@jaas-api-provision
REMEMBER_ME_FILTER	RememberMeAuthenticationFilter	http/remember-me
ANONYMOUS_FILTER	AnonymousAuthenticationFilter	http/anonymous
SESSION_MANAGEMENT_FILTER	SessionManagementFilter	session-management
EXCEPTION_TRANSLATION_FILTER	ExceptionTranslationFilter	http
FILTER_SECURITY_INTERCEPTOR	FilterSecurityInterceptor	http
SWITCH_USER_FILTER	SwitchUserFilter	N/A

Vediamo in opera

http://localhost:8080/auth

application-context-security

```
<http auto-config="true">
   <intercept-url pattern="/sec/admin/**"
                        access="ROLE_ADMIN"/>
   <intercept-url pattern="/sec/**"
                        access="ROLE_USER,ROLE_ADMIN"/>
   <intercept-url pattern="/**"
                        access="IS_AUTHENTICATED_ANONYMOUSLY" />
   <openid-login login-page="/login.html"
        authentication-failure-url=
        "/login.html?login_error=true" />
   <access-denied-handler error-page="/403.html"/>
   <form-login authentication-failure
               url="/login.html?login_error=true"/>
</http>
```

OpenID login form

```
<form action=
   '${contextPath}/j_spring_openid_security_check'
   method='POST'>
   <input type='text'
        name='openid_identifier'
        placeholder="OpenID url"/>
   <button type="submit">Sign in</button>
</form>

<form action='${contextPath}/j_spring_security_check'
   method='POST'>
        …
</form>
```

Authentication manager

- Serve a definire il contenitore del registro utenti associato al nostro RP
- Per l'autenticazione username-password la password è cifrata con sha-256

```
<authentication-manager>
   <authentication-provider user-service-ref="userDetailsService" >
       <password-encoder hash="sha-256"/>
   </authentication-provider>
</authentication-manager>
```

Registro utenti

```
<user-service id="userDetailsService" >
    <!-- Google account -->
   <user
        name="https://www.google.com/accounts/o8/id?
                id=AItOawmxL69eQEJ4rFB1apucJbvoAzHRdEfW57M"
        disabled="false"
        authorities="ROLE_ADMIN"/> <!-- gutente.test@gmail.com -->

   <!-- JOIDS account -->
   <user name="http://localhost:9080/joids/test" disabled="false"
   authorities="ROLE_USER, ROLE_ADMIN"/>

   <!-- Username password auth password = test -->
   <user name="user"
        password="9f86d081884c7d659a2feaa0c5
        5ad015a3bf4f1b2b0b822cd15d6c15b0f00a08" authorities="ROLE_USER"/>
   <user name="admin"
        password="9f86d081884c7d659a2feaa0c55ad
        015a3bf4f1b2b0b822cd15d6c15b0f00a08" authorities="ROLE_ADMIN"/>
</user-service>
```

Controller

```
@Controller
public class DefaultController {
    @Autowired
    private BusinessInterface businessInterface;
    @RequestMapping(method = RequestMethod.POST, value="/sec/sec.html")
    public ModelAndView adminAction (
          @RequestParam(value="ping") String pingMessage){
          ModelAndView mav = new ModelAndView("sec/sec");
          String result=null;
          try{
            result = businessInterface.adminOperatio(pingMessage);
          }catch(Exception e){result = "Can't perform admin operation";}
          mav.addObject("aresult", result);
          return mav;
    }
}
```

Business service

```
@Service("businessInterface")
public class BuisnessInterfaceImpl
   implements BusinessInterface {

   @Secured(value="ROLE_ADMIN")
   @Override
   public String adminOperation(String ping) {
       return "Admin operation performed.
              Ping param: " + ping;
   }

}
```

Progetto e presentazione

https://github.com/cristinalombardo/auth

Riferimenti

OpenID

- http://openid.net/
- http://openid.net/specs/openid-authentication-2_0.html
- http://code.google.com/p/openid4java/
- http://code.google.com/p/openid-server/
- http://infogrid.org/trac/wiki/Yadis
- https://developers.google.com/accounts/docs/OpenID?hl=en

Spring e grafica

- http://static.springsource.org/spring-security/site/docs/3.1.x/reference/springsecurity-single.html
- http://www.springsource.org/
- http://twitter.github.com/bootstrap/