

# Aplicaciones Demostrativas

Guía para desarrolladores de Software

**Rolosa HyJ S.A. - MICITT**

21 de Agosto de 2014



## **Resumen**

El presente documento es una guía paso a paso para desarrolladores de software.

## Tabla de contenido

Introducción.....	1
Pre-requisitos.....	1
Trust Service.....	2
Creación del Proyecto .....	2
Archivos de Configuración .....	3
Creación de nuevos archivos .....	9
Compilación y despliegue .....	14
Digital Signature Service .....	17
Subir documento a ser firmado .....	17
Petición de firmado.....	18
Descarga del documento firmado .....	19
Identity Provider .....	21
Instalación de SimpleSAMLphp.....	21
Configuración de SimpleSAMLphp.....	22
Uso de SimpleSAMLphp.....	25

# Introducción

---

El presente documento es una guía paso a paso para desarrolladores de software, para permitir agregar las funcionalidades de validación de certificados, autenticación y la funcionalidad de firma digital en sitios Web.

## Pre-requisitos

---

Para proseguir con el presente documento, es necesario disponer de:

- Una Tarjeta Inteligente para Firma Digital de Costa Rica
- Un lector de tarjetas apropiado
- Certificados Digitales de la cadena de la confianza de la Firma Digital de Costa Rica
- Drivers necesarios para la Tarjeta Inteligente, dependiendo el sistema operativo en el que se desee realizar los trabajos.

Los drivers y certificados digitales tienen que ser obtenidos desde el sitio de Soporte de Firma Digital de Costa Rica:

<https://www.soportefirmadigital.com/sfd/default.aspx>

# Trust Service

---

La presente sección detalla como agregar la funcionalidad de validación de certificados a un sitio web.

El Trust Service pone a disposición de aplicaciones cliente para la validación de cadenas de certificados el protocolo XKMS2. El protocolo XKMS2 del Trust Service esta descrito en el documento "**Guía para desarrolladores - eID TrustService**".

La aplicación Web demostrativa del Trust Service descrita en la presente sección está escrita en Java y consume el web service XKMS2. Esta se encuentra en:

**[http://dcfd-mw-applet.googlecode.com/svn/trunk/aplicaciones\\_demostrativas/trust-service/](http://dcfd-mw-applet.googlecode.com/svn/trunk/aplicaciones_demostrativas/trust-service/)**

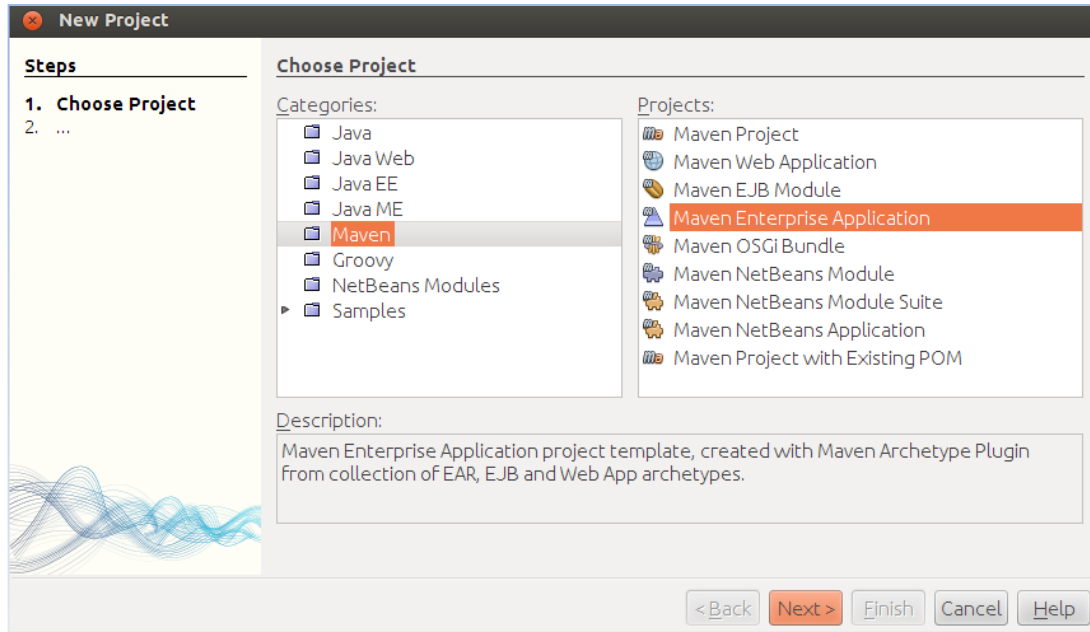
Para esta sección , es necesario conocimientos de Java 6 EE y experiencia con el desarrollo de proyectos de software utilizando Apache Maven 3, utilizando como entorno de desarrollo el IDE NetBeans.

El siguiente software es requerido para trabajar con la aplicación demostrativa en Java:

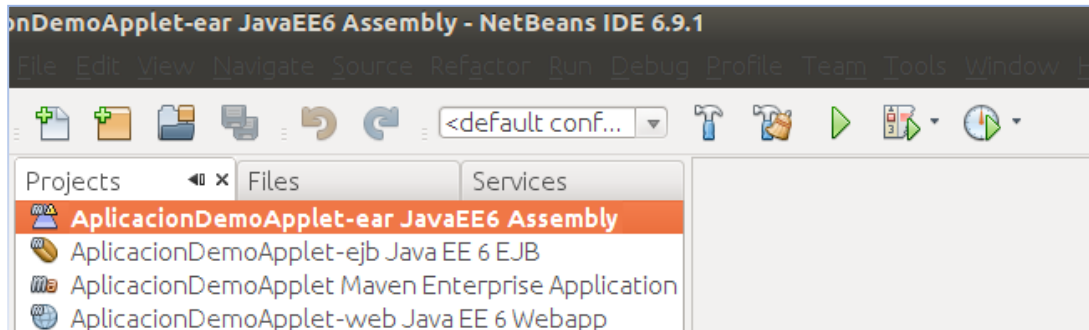
- **Java Runtime Environment 6 update 45**
- **NetBeans 6.9.1 para Java EE**
- **Apache Maven 3**

## Creación del Proyecto

Abrir NetBeans 6.9.1, elegir el menú Nuevo Proyecto, elegir la Categoría "**Maven**", y el Proyecto "**Maven Enterprise Application**" y presionar el botón **Next**:



Especificar un Nombre y ubicación adecuados para el proyecto. Luego especificar la versión adecuada de **Java EE 6** y asegurarse que se crearan los módulos **EJB** y **WebApp**. Hacer click en el botón **Finish** para completar la creación del proyecto. La Creación del Proyecto resulta en 4 sub-proyectos:



## Archivos de Configuración

Es necesario modificar los Archivos de Configuración de los Proyectos (**pom.xml**) para proseguir con la aplicación demostrativa.

Abrir el archivo pom.xml del proyecto **Webapp**, en el bloque de dependencias agregar las siguientes dependencias:

- javaee-api (Para disponer de todo el Java EE API dentro del proyecto)
- AplicacionDemoApplet-ejb (El modulo EJB será utilizado desde el Webapp)
- eid-applet-service (El modulo Applet Service el cual contiene los servicios requeridos como el de Autenticación)
- eid-applet-package (El modulo Applet Package el cual contiene el applet en sí mismo)

La configuración resultante de las dependencias debe lucir similar a:

```
<dependencies>
  <dependency>
    <groupId>javax</groupId>
    <artifactId>javaee-web-api</artifactId>
    <version>6.0</version>
    <scope>provided</scope>
  </dependency>

  <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>3.8.2</version>
    <scope>test</scope>
  </dependency>

  <dependency>
    <groupId>javax</groupId>
    <artifactId>javaee-api</artifactId>
    <version>6.0</version>
    <scope>provided</scope>
  </dependency>

  <dependency>
    <groupId>com.mycompany</groupId>
    <artifactId>AplicacionDemoApplet-ejb</artifactId>
    <version>1.0-SNAPSHOT</version>
  </dependency>

  <dependency>
    <groupId>be.fedict.eid-applet</groupId>
    <artifactId>eid-applet-service</artifactId>
    <version>1.1.2-SNAPSHOT</version>
  </dependency>

  <dependency>
    <groupId>be.fedict.eid-applet</groupId>
    <artifactId>eid-applet-package</artifactId>
    <version>1.1.2-SNAPSHOT</version>
    <scope>provided</scope>
  </dependency>
</dependencies>
```

En el bloque interno **profiles -> profile -> build -> plugins** agregar el siguiente plugin de compilación el cual permitirá la copia del Applet dentro del paquete final compilado:

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-dependency-plugin</artifactId>
  <executions>
    <execution>
      <id>copy</id>
      <phase>process-resources</phase>
      <goals>
        <goal>copy</goal>
      </goals>
      <configuration>
        <artifactItems>
          <artifactItem>
            <groupId>be.fedict.eid-applet</groupId>
            <artifactId>eid-applet-package</artifactId>
            <type>jar</type>
            <outputDirectory>${project.build.directory}/${project.artifactId}</outputDirectory>
          </artifactItem>
        </artifactItems>
      </configuration>
    </execution>
  </executions>
</plugin>
</plugins>
</build>
</profile>
</profiles>
```

Abrir el archivo pom.xml del proyecto **EJB**, en el bloque de dependencias agregar las siguientes dependencias:

- eid-applet-service-spi ( El modulo Applet Service SPI el cual contiene las interfaces necesarias para implementar en como EJB.)
- commons-logging-api (Permite logging)
- bcprov-jdk16 (API de Bouncycastle para Critografia)
- commons-io (Permite controles de Input/Output)

La configuración resultante de las dependencias debe lucir similar a:

```
<dependency>
  <groupId>be.fedict.eid-applet</groupId>
  <artifactId>eid-applet-service-spi</artifactId>
  <version>1.1.2-SNAPSHOT</version>
</dependency>

<dependency>
  <groupId>commons-logging</groupId>
  <artifactId>commons-logging-api</artifactId>
  <version>1.1</version>
</dependency>

<dependency>
  <groupId>org.bouncycastle</groupId>
  <artifactId>bcprov-jdk16</artifactId>
  <version>1.45</version>
</dependency>

<dependency>
  <groupId>commons-io</groupId>
  <artifactId>commons-io</artifactId>
  <version>1.4</version>
</dependency>
</dependencies>
```

En el bloque interno **build -> plugins** agregar el siguiente plugin de compilación el cual permitirá el desempaquetado de de las clases importante del modulo eid-applet-service-spi necesaria para su utilización dentro del modulo EJB.



```

<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-dependency-plugin</artifactId>
  <version>2.8</version>
  <executions>
    <execution>
      <id>unpack</id>
      <phase>generate-resources</phase>
      <goals>
        <goal>unpack</goal>
      </goals>
      <configuration>
        <artifactItems>
          <artifactItem>
            <groupId>be.fedict.eid-applet</groupId>
            <artifactId>eid-applet-service-spi</artifactId>
            <version>1.1.2-SNAPSHOT</version>
            <type>jar</type>
            <overWrite>>false</overWrite>
            <outputDirectory>${project.build.directory}/classes</outputDirectory>
            <includes>**/*.class,**/*.xml</includes>
            <excludes>**/*test.class</excludes>
          </artifactItem>
        </artifactItems>
        <outputDirectory>${project.build.directory}/wars</outputDirectory>
        <overWriteReleases>>false</overWriteReleases>
        <overWriteSnapshots>>true</overWriteSnapshots>
      </configuration>
    </execution>
  </executions>
</plugin>

</plugins>
<finalName>AplicacionDemoApplet-ejb</finalName>
</build>

```

Abrir el archivo pom.xml del proyecto **Assembly**, en el bloque de dependencias agregar las siguientes dependencias:

- eid-applet-service-spi ( El modulo Applet Service SPI el cual contiene las interfaces necesarias para implementar en como EJB.)

La configuración resultante de las dependencias debe lucir similar a:

```

</build>
<dependencies>

  <dependency>
    <groupId>com.mycompany</groupId>
    <artifactId>AplicacionDemoApplet-ejb</artifactId>
    <version>1.0-SNAPSHOT</version>
    <type>ejb</type>
  </dependency>

  <dependency>
    <groupId>com.mycompany</groupId>
    <artifactId>AplicacionDemoApplet-web</artifactId>
    <version>1.0-SNAPSHOT</version>
    <type>war</type>
  </dependency>

  <dependency>
    <groupId>be.fedict.eid-applet</groupId>
    <artifactId>eid-applet-service-spi</artifactId>
    <version>1.1.2-SNAPSHOT</version>
  </dependency>
</dependencies>
</project>

```

En el bloque interno **build -> plugins** modificar el plugin maven-ear-plugin agregando el parámetro de configuración **<defaultLibBundleDir/>** como se muestra a continuación, cambiar lo existente:

```

<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-ear-plugin</artifactId>
  <version>2.4</version>
  <configuration>
    <version>6</version>
  </configuration>
</plugin>

</plugins>
<finalName>AplicacionDemoApplet-ear</finalName>
</build>

```

Por:

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-ear-plugin</artifactId>
  <version>2.4</version>
  <configuration>
    <version>6</version>
    <defaultLibBundleDir>lib</defaultLibBundleDir>
  </configuration>
</plugin>

</plugins>
<finalName>AplicacionDemoApplet-ear</finalName>
</build>
```

## Creación de nuevos archivos

Se necesita crear el archivo HTML que servirá de contenedor del Applet, para esto en el proyecto **Webapp**, haciendo click derecho agregaremos un nuevo archivo JSP. El nombre de archivo que elegiremos será "**authenticate.jsp**", solamente hacer click en el botón Finish.

Como se había mencionado anteriormente, este archivo será el contenedor del Applet, para lo cual se deberá agregar el código que se muestra a continuación. Como se puede observar el parámetro **TargetPage** : "**authenticate-result.jsp**" hace referencia a un archivo jsp que mostrará el resultado de la autenticación.

El parámetro AppletService : "**applet-service-authn**" hace referencia al web-service que será invocado para realizar la autenticación. La implementación de este servicio será mostrada más adelante.

```

authenticate.jsp x
1  <!--
2      Document : authenticate
3      Author  : jubarran
4  -->
5
6  <%@page contentType="text/html" pageEncoding="UTF-8"%>
7  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
8      "http://www.w3.org/TR/html4/loose.dtd">
9
10 <html>
11 <head>
12     <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
13     <title>JSP Page</title>
14 </head>
15 <body>
16
17 <h1>Autenticación - eID Applet Demo</h1>
18 <script src="https://www.java.com/js/deployJava.js"></script>
19 <script>
20     var attributes = {
21         code : 'be.fedict.eid.applet.Applet.class',
22         archive : 'eid-applet-package-1.1.2-SNAPSHOT.jar',
23         width : 600,
24         height : 400
25     };
26     var parameters = {
27         TargetPage : 'authenticate-result.jsp',
28         AppletService : 'applet-service-authn',
29         BackgroundColor : '#cccccc',
30         Language : 'es'
31     };
32     var version = '1.6';
33     deployJava.runApplet(attributes, parameters, version);
34 </script>
35
36 </body>
37 </html>
38

```

Ahora procedemos a crear el archivo **"authenticate-result.jsp"** dentro del proyecto **Webapp**. El archivo **"authenticate-result.jsp"** deberá lucir similar al siguiente código. Como se puede observar en las líneas 17 y 20, utilizando tags de JSP se pueden obtener los atributos **"eid.identifier"** y **"AuthenticationResult"** respectivamente de la sesión. Estos atributos serán establecidos durante la ejecución de la autenticación que esta implementada para el web-service.

```

authenticate-result.jsp
1  <%--
2      Document : authenticate-result
3      Author  : jubarran
4  --%>
5
6  <%page contentType="text/html" pageEncoding="UTF-8"%>
7  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
8      "http://www.w3.org/TR/html4/loose.dtd">
9
10 <html>
11 <head>
12     <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
13     <title>JSP Page</title>
14 </head>
15 <body>
16     <h1>Página con el resultado de la Autenticación</h1>
17     <p>Usuario: <%=session.getAttribute("eid.identifier")%>
18     </p>
19     <p>Resultado de la
20         Autenticación: <%=session.getAttribute("AuthenticationResult")%>
21     </p>
22
23     <a href="authenticate.jsp">Autenticar nuevamente</a>
24     |
25     <a href=".">Página de Inicio</a>
26 </body>
27 </html>
28

```

Ahora procedemos a crear la implementación del servicio de Autenticación que utilizaremos para la presente aplicación. Para lo cual se necesita crear una nueva clase Java en el proyecto **EJB**, la cual nombraremos **AuthenticationServiceOCSPBean**.

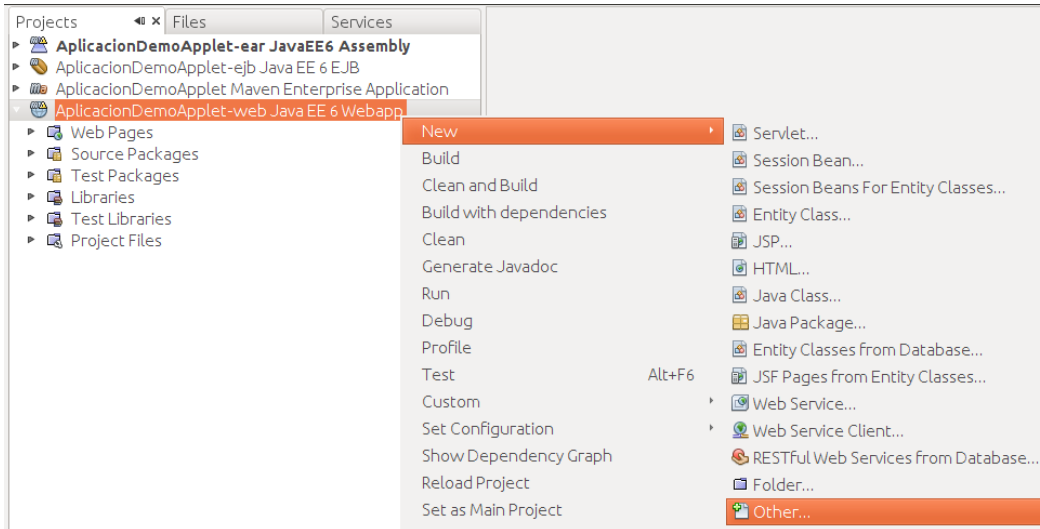
La implementación de la clase **AuthenticationServiceOCSPBean** se muestra a continuación. Como se puede observar, la clase implementa la interfaz **AuthenticationService** la cual está definida en el módulo **eid-applet-service-spi**. El package **be.fedict.trust.client.XKMS2Client** contiene la clase **XKMS2Client** que consume el web service que nos interesa. Se puede apreciar que se pasa el URI del web service como parámetro al constructor de la clase **XKMS2Client** y luego la función **validate()** recibe la cadena de certificados de confianza que el Applet proveerá para su validación.

```

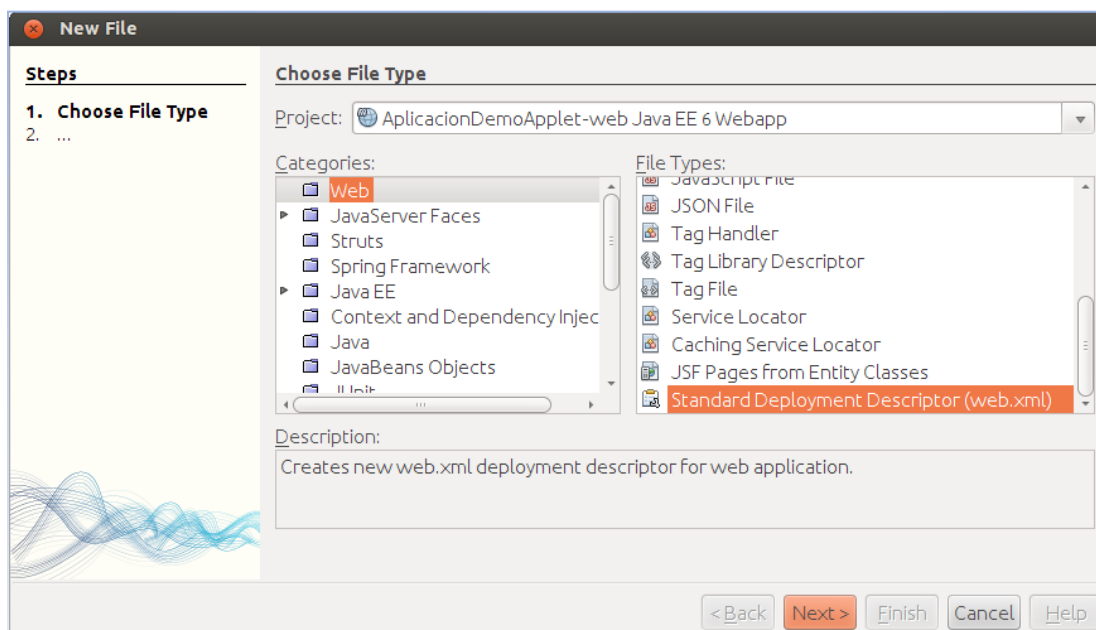
1 package ejb;
2
3 import java.security.cert.X509Certificate;
4 import javax.ejb.LocalBean;
5 import javax.ejb.Stateless;
6 import javax.servlet.http.HttpSession;
7 import be.fedict.eid.applet.service.spi.AuthenticationService;
8 import be.fedict.trust.client.XKMS2Client;
9 import be.fedict.trust.client.exception.ValidationFailedException;
10 import java.util.List;
11
12 /**
13  * @author jubarran
14  */
15 @LocalBean
16 @Stateless
17 public class AuthenticationServiceOCSPBean implements AuthenticationService {
18
19     private HttpSession session;
20
21     @Override
22     public void validateCertificateChain(List<X509Certificate> certificateChain)
23         throws SecurityException {
24
25         for(X509Certificate c : certificateChain) {
26             System.out.println(c.getSubjectDN().toString());
27         }
28
29         try {
30             String xkmsUrl = new String("http://alpha.rolosa.com:7386/eid-trust-service-ws/xkms2");
31             XKMS2Client xkms2Client = new XKMS2Client(xkmsUrl );
32             xkms2Client.setLogging(true);
33             xkms2Client.validate(certificateChain);
34             System.out.println("OK");
35             session.setAttribute("AuthenticationResult", "Certificado Validado");
36         } catch (ValidationFailedException e) {
37             e.printStackTrace();
38             session.setAttribute("AuthenticationResult", "Error en la validacion");
39         } catch (Exception e) {
40             e.printStackTrace();
41         }
42     }
43
44     public void setHttpSessionObject(Object sessionObject) {
45         session = (HttpSession) sessionObject;
46     }
47 }

```

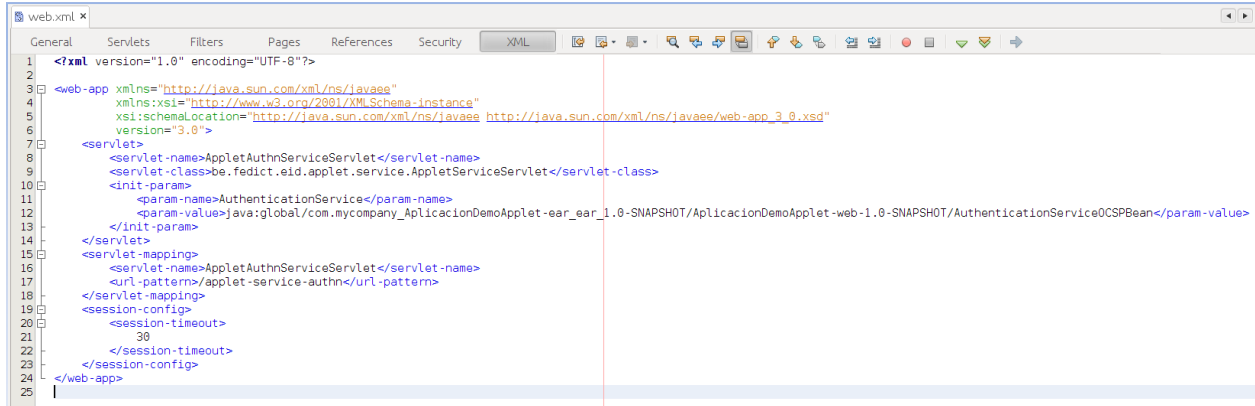
Ahora es necesario crear el archivo que definirá la relación entre el web-service y la implementación real como ejb que se encuentra dentro del proyecto **EJB**. Para lo cual hacer click derecho en el proyecto **Webapp** y agregar "Other":



Elegir la categoría **Web** y el tipo de archivo "**Standard Deployment Descriptor (web.xml)**"



El contenido del archivo web.xml debe contener la definición de servlet como se muestra a continuación:



```

1 <?xml version="1.0" encoding="UTF-8"?>
2
3 <web-app xmlns="http://java.sun.com/xml/ns/javaee"
4          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5          xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
6          version="3.0">
7
8   <servlet>
9     <servlet-name>AppletAuthnServiceServlet</servlet-name>
10    <servlet-class>be.fedict.eld.applet.service.AppletServiceServlet</servlet-class>
11
12    <init-param>
13      <param-name>AuthenticationService</param-name>
14      <param-value>java:global/com.mycompany_AplicacionDemoApplet-ear_ear_1.0-SNAPSHOT/AplicacionDemoApplet-web-1.0-SNAPSHOT/AuthenticationServiceOCSPBean</param-value>
15    </init-param>
16  </servlet>
17  <servlet-mapping>
18    <servlet-name>AppletAuthnServiceServlet</servlet-name>
19    <url-pattern>/applet-service-authn/</url-pattern>
20  </servlet-mapping>
21  <session-config>
22    <session-timeout>
23      30
24    </session-timeout>
25  </session-config>
26 </web-app>

```

Es importante resaltar que el valor del parámetro "**AuthenticationService**" corresponde al identificador JNDI para el Java Bean creado e implementado en la clase **AuthenticationServiceOCSPBean**. Este identificador utiliza el **path java:global** para poder ser compatible con los servidores como GlassFish3.

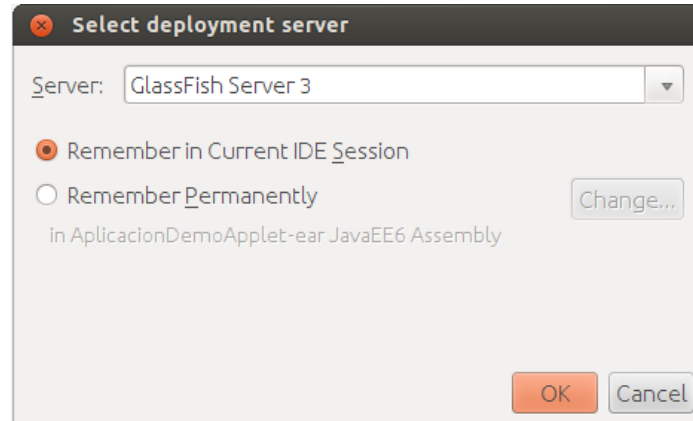
## Compilación y despliegue

Una vez completada la modificación de los archivos de configuración y la creación de archivos nuevos, es posible compilar la aplicación. Para este cometido hacer click derecho en el proyecto **Application** y elegir el menú "**Clean and Build**"

No habiendo ningún error de compilación, ya es posible realizar el despliegue de la aplicación en el servidor de aplicaciones, para esto hacer click derecho en el proyecto **Assembly** y elegir el menú "**Run**"

Es necesario especificar el servidor de aplicaciones sobre el cual se hará el despliegue de la aplicación. NetBeans 6.9.1 viene incorporado con el servidor GlassFish 3 con el que podemos trabajar sin ningún inconveniente:

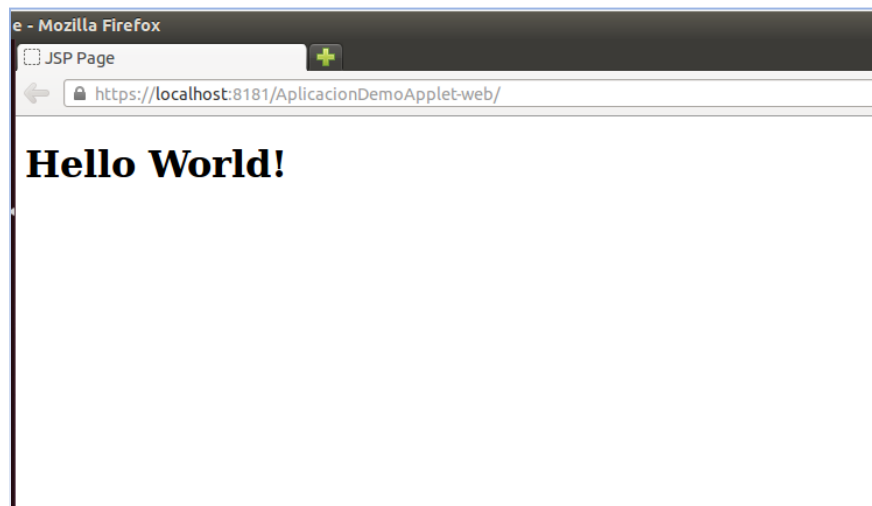




No habiendo errores en el despliegue ya podemos utilizar la aplicación recién creada. Asegurarse de tener un lector de tarjetas inteligente y una tarjeta correctamente instalados y conectados a la computadora. En una ventana de navegador, dirigirse a:

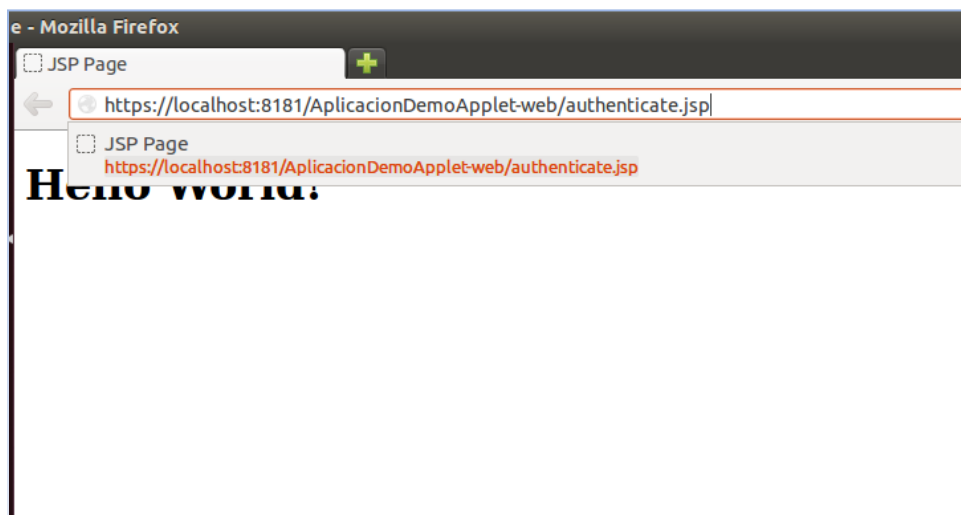
**<https://localhost:8181/AplicacionDemoApplet-web/>**

El resultado esperado se muestra en la siguiente Figura:



Ahora podemos dirigirnos a la página de Autenticación, para lo cual ir a:

**<https://localhost:8181/AplicacionDemoApplet-web/authenticate.jsp>**



La página cargará el Applet y nos otorgará una advertencia con respecto al certificado de seguridad, puesto que esta es una aplicación demostrativa, podemos confiar en el mismo.

El Applet procederá con el procedimiento de validación y pedirá el Código PIN de la Tarjeta. Una vez completada la Autenticación el Applet nos redirigirá a la página `authenticate-result.jsp` mostrando el resultado de la autenticación.

# Digital Signature Service

La presente sección detalla la como agregar la funcionalidad de firma digital a un sitio web. Los detalles del protocolo del DSS son descritos en el documento "**Guía para desarrollo de Software - eID Digital Signature Service**".

La ejecución básica del protocolo del DSS consiste de tres mensajes de petición/respuesta.

- La aplicación web sube el documento para ser firmado al DSS
- La petición real de firmado desde la aplicación web hacia el DSS
- La aplicación web descarga el documento firmado del DSS

La aplicación para la que se desea agregar la funcionalidad de firma digital debe de realizar el procesamiento, desde el punto de vista de cliente, del protocolo de DSS.

La aplicación Web demostrativa descrita en la presente sección está escrita en PHP y se encuentra en:

[http://dcfd-mw-applet.googlecode.com/svn/trunk/aplicaciones\\_demostrativas/dss](http://dcfd-mw-applet.googlecode.com/svn/trunk/aplicaciones_demostrativas/dss)

## Subir documento a ser firmado

La aplicación Web puede encontrar múltiples formas para subir el documento a ser firmado. A continuación se muestra un formulario básico para lograr ese fin:

```
<form action="upload.php" method="POST" enctype="multipart/form-data">
  <p> Seleccione el archivo a firmar:<br><br><br>
    <input type="file" name="the_file" />
    <input type="submit" value="Subir">
  </p>
</form>
```

El archivo de destino **upload.php**, persigue codificar en base64 el contenido del archivo subido y lo mantiene en sesión. A su vez presenta un enlace a la página donde se realiza la petición real de firmado desde la aplicación web hacia el DSS.

```
<?php

session_start();

if (isset($_FILES['the_file'])) {

    $file_name = $_FILES['the_file']['name'];

    $file_size = $_FILES['the_file']['size'];
    $file_type = $_FILES['the_file']['type'];
    $file_tmp = $_FILES['the_file']['tmp_name'];

    $type = pathinfo($file_tmp, PATHINFO_EXTENSION);
    $data = file_get_contents($file_tmp);
    $base64 = base64_encode($data);

    echo 'Archivo: ' . $file_name . '<br>';
    echo 'Tipo: ' . $file_type . '<br>';
    echo 'Tamano: ' . round(($file_size / 1024), 1) . ' [KB]<br>';
    echo '<a href="sign.php">Firmar</a><br>';

    $_SESSION['the_file'] = $base64;
    $_SESSION['file_name'] = $file_name;
    $_SESSION['file_type'] = $file_type;

?>
```

## Petición de firmado

La petición de firmado a ser enviada a través de HTTP POST requiere los parámetros mínimos que se detallan a continuación:

- URL del Digital Signature Service, al que se realizará la petición.
- Documento codificado en base64
- Página de la aplicación web de destino para re-direccionar luego del proceso de firmado
- Lenguaje que será utilizado para utilización del Applet en el DSS
- Tipo de contenido MIME del archivo a firmar

A continuación se muestra el contenido del archivo **sign.php** el que permite realizar la solicitud de firmado:

```
<?php
session_start();

if(isset($_SESSION['the_file'])) {
    ?>
    <form id="dss-request-form" method="post" action="https://alpha.rolosa.com:8443/eid-dss/protocol/simple">
        <input type="hidden" name="SignatureRequest" value="<?php print $_SESSION['the_file'];?>" />
        <input type="hidden" name="target" value="http://localhost/tests/php-eID-test-DSS/landing.php" />
        <input type="hidden" name="language" value="es" />
        <input type="hidden" name="ContentType" value="<?php print $_SESSION['file_type']; ?>" />
        <input type="hidden" name="RelayState" value="foo123" />
        <input type="submit" value="Submit" style="visibility: hidden;"/>
    </form>
    <script type="text/javascript">
        document.getElementById('dss-request-form').submit();
    </script>
    <?php
}
?>
```

## Descarga del documento firmado

Una vez realizada la firma del documento, se realiza una redirección a la página de destino de la aplicación Web destinada a recibir la respuesta. Un mensaje HTTP POST es enviado de vuelta desde el DSS que contiene la respuesta del firmado, la información puede ser obtenida en **\$\_POST['SignatureStatus']** (para indicar si la firma fue exitosa o no) y en **\$\_POST['SignatureResponse']** el contenido del archivo firmado digitalmente, codificado en base64.

El archivo **landing.php** demuestra todo este detalle:

```
<?php
session_start();
?>
<html>
  <head>
    <title>eID DSS Test</title>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8"/>
  </head>
  <body>
    <h1>Aplicación demostrativa - eID DSS</h1><br>
    <?php
    if ($_POST['SignatureStatus'] == "OK") {
      $_SESSION["SignatureResponse"] = $_POST["SignatureResponse"];
      print 'Firma Exitosa<br>';
      print '<a href=file.php>Descargar Archivo</a><br>';
    } else {
      print 'Firma Corrupta<br><br>';
    }
    ?>
    <a href="upload.php">Realizar Firma Digital</a><br/>
  </body>
</html>
```

El archivo **file.php** muestra como se realiza la decodificación de los datos recibidos para poder descargar el archivo firmado digitalmente.

```
<?php
session_start();
header('Content-Type: '.$_SESSION['file_type']);
header('Content-Disposition: inline; filename="'.$_SESSION['file_name'].'.txt');
echo base64_decode($_SESSION["SignatureResponse"]);
?>
```

# Identity Provider

---

La presente sección detalla la como agregar la funcionalidad de autenticación a un sitio web. Los detalles de los protocolos del IDP son descritos en el documento "**Guía para desarrolladores - eID IDP**".

El eID Identity Provider soporta los siguientes protocolos para autenticación:

- SAML v2.0
- OpenID v2.0
- WS-Federation v1.1

La aplicación para la que se desea agregar la funcionalidad de autenticación debe de realizar el procesamiento, desde el punto de vista de cliente, de alguno de los protocolos mencionados.

La aplicación Web demostrativa descrita en la presente sección está escrita en PHP y se encuentra en:

**[http://dcfd-mw-applet.googlecode.com/svn/trunk/aplicaciones\\_demostrativas/idp](http://dcfd-mw-applet.googlecode.com/svn/trunk/aplicaciones_demostrativas/idp)**

Esta aplicación utiliza el protocolo SAML v2.0 para interactuar con el Identity Provider. Se utiliza una implementación Open Source (**SimpleSAMLphp** - <https://simplesamlphp.org/>), puesto que el objetivo de esta aplicación NO es el de implementar el protocolo mas al contrario solo se utiliza dicho protocolo para la autenticación con el IDP.

Se ha tomado como punto de partida la aplicación demostrativa del DSS descrita en la sección anterior.

## Instalación de SimpleSAMLphp

Para proceder con la instalación de SimpleSAMLphp se deben seguir los pasos que se indican en:

**<https://simplesamlphp.org/docs/stable/simplesamlphp-install>**

Una vez instalado, (suponiendo que se ha instalado en localhost) verificar que la instalación ha sido correcta accediendo a:

**<http://localhost/simplesaml>**

## Configuración de SimpleSAMLphp

Para la configuración es necesario editar el archivo `/var/simplesamlphp/metadata/saml20-idp-remote.php`, y especificar el contenido de la variable **\$metadata**.

El IDP especifica la información de Metadata en el siguiente sitio:

**<https://alpha.rolosa.com:8443/eid-idp/>**

La configuración de Metadata que nos interesa es **SAML2 Browser Post Authentication Metadata** cuyo contenido XML debe ser convertido al formato que SimpleSAMLphp aceptará. Para esto es necesario acceder a:

**<http://localhost/simplesaml/admin/metadata-converter.php>**

y pegar el contenido del archivo:

**<https://alpha.rolosa.com:8443/eid-idp/endpoints/saml2/post/metadata/auth-metadata.xml>**

, dentro del Metadata parser como se puede ver a continuación:



Metadata parser

https://localhost/simplesaml/admin/metadata-converter.php?

Google

Metadata parser

XML metadata

```
<?xml version="1.0" encoding="UTF-8"?><md:EntityDescriptor xmlns:md="urn:oasis:names:tc:SAML:2.0:metadata"
entityID="Default"><ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
<ds:SignedInfo>
<ds:CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
<ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1" />
<ds:Reference URI="">
<ds:Transforms>
<ds:Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature" />
<ds:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
</ds:Transforms>
<ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
<ds:DigestValue>4zXlSn3pBSEYLJFLvQKWoVYngDs=</ds:DigestValue>
</ds:Reference>
</ds:SignedInfo>
<ds:SignatureValue>
EtRgboRWLXwuUBa0o/+u6dVGMXAU/OeKASnjcWSjkZSvogLMVxrFCqzzwjy1kYbgQCpga3HEfKZ
Ymw9A92ZR8mqjyqlkFKXEmfHphDpAEWqpU+1MwuaIvtqCV2mh8UNDsN9j71Dm/ukVJShunG9jc7
yR0wC+RNoX8z8n1af/w=
</ds:SignatureValue>
<ds:KeyInfo><ds:X509Data>
<ds:X509Certificate>MIICXzCCAciaAwIBAQIEU+qW+iANBqkqhkiG9w0BAQUFADBOMQswCQYDVQQGEwJDUjERMABGA1UE
```

Parse

Converted metadata

saml20-idp-remote

```
$metadata['Default'] = array (
    'entityid' => 'Default',
    'contacts' =>
        array (
        ),
    'metadata-set' => 'saml20-idp-remote',
    'SingleSignOnService' =>
        array (
            0 =>
                array (
                    'Binding' => 'urn:oasis:names:tc:SAML:2.0:bindings:HTTP-POST',
```

La información convertida es la que se utilizará para la configuración del SimpleSAMLphp. Editar el archivo:

```
/var/simplesamlphp/metadata/saml20-idp-remote.php
```

, y pegar la información de Metadata convertida cómo se muestra a continuación:

```
*saml20-idp-remote.php (/var/simplesamlphp/metadata) - gedit
Open Save Undo
*saml20-idp-remote.php
$metadata['Default'] = array (
    'entityid' => 'Default',
    'contacts' =>
        array (
        ),
    'metadata-set' => 'saml20-idp-remote',
    'SingleSignOnService' =>
        array (
            0 =>
                array (
                    'Binding' => 'urn:oasis:names:tc:SAML:2.0:bindings:HTTP-POST',
                    'Location' => 'https://alpha.rolosa.com:8443/oidp/protocol/saml2/post/auth',
                ),
            ),
    'SingleLogoutService' =>
        array (
        ),
    'ArtifactResolutionService' =>
        array (
        ),
    'keys' =>
        array (
            0 =>
                array (
                    'encryption' => false,
                    'signing' => true,
                    'type' => 'X509Certificate',
                    'X509Certificate' => '
MIICXzCCAcigAwIBAgIEU+gW+jANBgkqhkiG9w0BAQUFADB0MQswCQYDVQGEWJDUjERMA8GA1UE
CBMIU2FuIEpvc2UxETAPBgNVBACTCFNhb3NlMRQwEgYDVQKEwtSb2xvc2EgUy5BLjEUMBIG
A1UECXMUM9sb3NhIFMuQ54xEzARBGNVBAMTckLEUCB5b2xvc2EwHhcNMTQwODEwNjAyWHcN
MTQxMTA5MDEwNjAyWjB0MQswCQYDVQGEWJDUjERMA8GA1UECBMIU2FuIEpvc2UxETAPBgNVBAC
TCFNhb3NlMRQwEgYDVQKEwtSb2xvc2EgUy5BLjEUMBIGA1UECXMUM9sb3NhIFMuQ54xEzAR
BgNVBAMTckLEUCB5b2xvc2EwZ8wDQYJKoZIhvcNAQEBBQADgY0AMIGJAoGBAK1aihcxHj8qVSc
v9P+duMuBefR2oJ8HzA+5NRwPxxISG+bQKYX811Sn+hIwLC6BR0h0ervglxEUYTzMfWMSvG9SRJu
aCGf+sJUdw5sPVY7jIH1UL28EJgE+HGvrcPSIxc8sa+ee8as3PmwtWDRSzc7mYYOQZOrKQJXtRk
OrUJAgMBAAEwDQYJKoZIhvcNAQEFBQADgYEAQmWuAuUd5sotaur/o33sAvE6/4xX3HAKmcDJg6J
txIfK6LigZQSpU5dCB0vGdrr0gxQf71DzN+U3CodD8KECcsc6Jc00H/GkEPBmqPtSmKV4uIdYnk
fZkrdLQX/4RGeHpaLeSUwNyQ6sZJWzLAM0XMe6BwcgokE9ySQVEHk1E=',
                ),
            ),
        );
```

Luego de tener configurada la información de Metadata, es necesario especificar cuál es la fuente de autenticación que SimpleSAMLphp utilizará. Para esto editar el archivo:

**/var/simplesamlphp/config/authsources.php**

como se muestra a continuación:

```
authsources.php (/var/simplesamlphp/config) - gedit
Open Save Undo
authsources.php
<?php
$config = array(
    // This is a authentication source which handles admin authentication.
    'admin' => array(
        // The default is to use core:AdminPassword, but it can be replaced with
        // any authentication source.
        'core:AdminPassword',
    ),

    // An authentication source which can authenticate against both SAML 2.0
    // and Shibboleth 1.3 IdPs.
    'default-sp' => array(
        'saml:SP',

        // The entity ID of this SP.
        // Can be NULL/unset, in which case an entity ID is generated based on the meta
        'entityID' => 'Default',

        // The entity ID of the IdP this should SP should contact.
        // Can be NULL/unset, in which case the user will be shown a list of available
        'idp' => 'Default',

        // The URL to the discovery service.
        // Can be NULL/unset, in which case a builtin discovery service will be used.
        'discoURL' => NULL,
    ),
);
```

Una vez realizada la configuración ya es posible utilizar la instalación en la aplicación que se desea.

## Uso de SimpleSAMLphp

Para la utilización de la librería SimpleSAMLphp, basta con insertar, como primeras líneas del archivo, el siguiente código:

```
<?php
require_once('/var/simplesamlphp/lib/_autoload.php');
$as = new SimpleSAML_Auth_Simple('default-sp');
$as->requireAuth();
?>
```

La implementación del protocolo SAMLv 2.0 dentro de la librería SimpleSAMLphp encapsula el manejo de la sesión con la llamada a la función **requireAuth()**. Esto quiere decir que si no se ha iniciado una sesión, la librería SimpleSAMLphp direccionará al usuario a la página del IDP que permitirá realizar la autenticación correspondiente, para que luego de una autenticación exitosa, se realice el direccionamiento de vuelta a la página del sitio inicial y recién poder mostrar el contenido el usuario.

El archivo **index.php**, cómo se puede apreciar a continuación, es sólo un contenido HTML simple y nos presenta un enlace hacia el archivo **login.php**.

```
<html>
  <head>
    <title>eID IDP Test</title>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8"/>
  </head>
  <body>
    <h1>Aplicación demostrativa - eID IDP - eID DSS</h1><br>
    <br /><a href="login.php">Iniciar Sesión</a>
  </body>
</html>
```

El archivo **login.php** nos muestra la utilización real de SimpleSAMLphp, donde se ve la instanciación de un objeto de la clase **SimpleSAML\_Auth\_Simple** y su posterior llamada a la función **requireAuth()**.

```
<?php
require_once('/var/simplesamlphp/lib/_autoload.php');
$as = new SimpleSAML_Auth_Simple('default-sp');
$as->requireAuth();
?>
<html>
  <head>
    <title>eID IDP Test </title>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8"/>
  </head>
  <body>
    <h1>Aplicación demostrativa - eID IDP - eID DSS</h1><br>
    <?php
    $attributes = $as->getAttributes();
    print ("Bienvenido: " . $attributes["http://schemas.xmlsoap.org/ws/2005/05/identity/claims/name"][0] . "<br />");
    print ("Atributos provistos por eID IDP: <br />");

    print ('<table border="1">');
    foreach ($attributes as $key => $value) {
      print ("<tr><td><strong>" . $key . "</strong></td><td>" . $value[0] . "</td></tr>");
    }
    print ("</table><br><br><br>");

    print ('<a href="upload.php">Realizar Firma Digital</a><br>');

    print ('<hr><a href="logout.php">Cerrar Sesión</a>');
    ?>
  </body>
</html>
```

Como se puede observar en el archivo **login.php**, una vez que la autenticación ha sido exitosa, se mostrarán todos los atributos provistos por el IDP y ya es posible proseguir con la firma digital.

Se debe agregar las líneas de llamada a la librería para cada archivo del sitio web en el que se desee controlar la autenticación, como se puede ver a continuación, el archivo `ha` sido extendido para que sea controlado por autenticación:

```
<?php
require_once('/var/simplesamlphp/lib/_autoload.php');
$as = new SimpleSAML_Auth_Simple('default-sp');
$as->requireAuth();
?>
<html>
    <head>
        <title>eID IDP Test</title>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8"/>
    </head>
    <body>
        <h1>Aplicación demostrativa - eID IDP - eID DSS</h1><br>
        <?php
        if (isset($_FILES['the_file'])) {

            $file_name = $_FILES['the_file']['name'];

            $file_size = $_FILES['the_file']['size'];
            $file_type = $_FILES['the_file']['type'];
            $file_tmp = $_FILES['the_file']['tmp_name'];

            $type = pathinfo($file_tmp, PATHINFO_EXTENSION);
            $data = file_get_contents($file_tmp);
            $base64 = base64_encode($data);

            echo 'Archivo: ' . $file_name . '<br>';
            echo 'Tipo: ' . $file_type . '<br>';
            echo 'Tamano: ' . round(($file_size / 1024), 1) . ' [KB]<br>';
            echo '<a href="sign.php">Firmar</a><br>';

            //echo "Contenido del archivo en Base64 es ".$base64;

            $_SESSION['the_file'] = $base64;
            $_SESSION['file_name'] = $file_name;
            $_SESSION['file_type'] = $file_type;
        }
        <br>

        <?php
    } else {
        ?>
```

Para terminar la sesión, basta con realizar la destrucción de la sesión de la siguiente manera:

```
<?php
session_start();
$_SESSION = array();
session_destroy();
session_unset();
header('Location: index.php');
?>
```