21/08/20

# Instrucciones para la programación de Sso en Angular

## Antecedentes

Se tiene como antecedentes al desarrollo realizado para Vaadin 14 con los servicios adicionales que se tuvieron que desarrollar para el micro.servicio Oauth2srv.

Los requerimientos del negocio son los siguientes:

#### REQUERIMIENTOS

- Que el usuario que tiene varios PWAs definidos dentro de su browser de Chrome no tenga que hacer un log-in por cada PWA.
- Es suficiente que funcione para Chrome únicamente.
- Los PWAs pueden ser de Vaadin o de Angular.
- Si el usuario ya se conectó y se desea conectar en otros dispositivo u otro browser (e.g., Safari, Firefox, etc) dentro del mismo equipo no lo debe dejar el PWA y debe regresar al panel de log.-in.
- Para terminar su sesión (que mantiene Oauth2srv) se puede hacer de la siguientes maneras:
    - Que el usuario haga un log-out en alguno de los PWAs abiertos.
    - Que se cambie de día.
    - Que un usuario de alta prioridad (i.e. facultad de ADMIN_IAM) resetee la sesión desde el IAM-UI.
    - Resetear el servidor de Oauth2srv.
- Las acciones de log-in y log-out deben de generar un evento (i.e., en el event-logger).

#### DISEÑO CONCEPTUAL

El diseño conceptual es un protocolo entre el browser y PWA con el servidor de tokens OAth2srv. Cada componente mencionado tiene la siguiente responsabilidad:

*(a) Browser Chrome:*

- Almacena una cookie definida en el archivo de application.properties. Debe ser el mismo nombre para ‘todos’ los PWAs. Actualmente el nombre es SSO.

- El valor de la cookie SSO es un valor que debe estar codificado con Base64.
- El valor de la cookie ya codificado es un unique-ID generado por el Oauth2Srv que es la llave de acceso de su HashMap de sesiones que tiene en memoria.
- Chrome al salirse elimina todas las cookies que tiene en memoria. nota: existe una opción en preferencias d Chrome donde se puede deshabilitar el borrado de las cookies para cuando se salga del browser. Se probó dicha funcionalidad y no funcionó sobre Chrome 84.0.4147.15 en Mac OSX.

*(b) PWAs:*

- Cuando se inicia la aplicación y previo a mostrar el panel de log-in (y obviamente antes de crear un `SpringSecurityContext`), se debe validar si existe una sesión abierta del mismo usuario con el Oauth2srv.
- Si se tiene una sesión abierta en el Oauth2srv se debe abrir de manera automática una sesión de `SpringSecurityContext` de manera automática sin el panel de log-in solicitando un token al Oauth2srv. nota: en este caso 'no' se debe de generar un evento para el evento-blogger de log-in.
- Si no se tiene un sesión abierta en el Oauth2srv se procede normalmente a abrir el panel de log-in para preguntar por el nombre y password del usuario, hacer el log-in correspondiente creando el `SpringSecurityContext` y llamar al event-logger para que genere el evento de LOG_IN.
- Al mostrar la primera pantalla de la aplicación del PWAs (i.e., lo que sería el 'home page') se debe de checar si existe la cookie SSO en el browser.
- Si ya existe no se hace nada.
- Sino existe se debe de generar una cookie nueva SSO con el valor que nos del el servidor de OAuth2srv y salvarla en el browser de Chrome.
- Debe existir una opción en el PWA de log-out que haga lo siguiente:
    - Envíe un evento al event-logger de salida de Genesis.
    - Borre la cookie SSO que se tenga en el browser Chrome.
    - Avisarle al servidor OAuth2srv que se terminó la sesión de SSO.

*(c) Servidor Oauth2srv:*

Además de las funciones actuales que tiene de generación y validación de los token se añadieron las siguiente funcionalidad para el manejo de Sso:

- Abrir un sesión de Sso y que regrese el identificador de esa sesión.
- Validar si una sesión esta activa con el unique-id de la cookie. En caso afirmativo regresa el usuario y el password codificado. nota: este es el único servicio de REST que ‘no’ requiere token, ya que aun no ha sido solicitado.
- Terminar una sesión de Sso.
- Consultar las sesiones actuales. nota: este servicio solo lo debe utilizar el IAM-UI.
- Todas las sesiones por cuestiones de seguridad se encuentran en memoria y no son almacenadas de manera persistente. Es decir, en caso de que se re-inicie el servidor OAuth2srv, todas las sesiones de Sso se perderán y los usuarios tendrán que volver a registrarse, si inician nuevamente un PWA; los PWA ya abierto no sufirán ninguna cambio, aunque sus token ya ‘no’ serán válidos.

---

## PWA: Proceso previo a mostrar el panel de log-in.

El proceso previo a mostrar el panel de log-in se muestra en el siguiente diagrama:

---

## PWA: Panel de log-in

De acuerdo al proceso anterior se tienen dos caminos que el usuario puede tomar:

(a) Log-in manual: No sufre cambio y se muestra el panel donde se pregunta el nombre y password del usuario. nota: elpanel debe ser el mismo para todos los PWAs y no debe de incluir el nombre el PWA ya que es un Sso.

(b) Log-in automático: es cuando el proceso anterior tiene el nombre y password del usuario (proveniente del Oauth2srv) y debe entrar de manera automática creando el `SpringSecuirtyContext` correspondiente y también ‘deberá de solicitar un token nuevo para este PWA, con esto se evita el tener un solo token para los PWAs, es decir, ‘cada’ PWA manejara su propio token, con su caducidad, auditoria, correlation-id, etc.

## PWA: Home page.

Para cuando el PWA ya haya entrado al sistema el PWAS deberá de preguntar por la existencia de la cookie SSO. Si no existe deberá de generar una nueva de la siguiente manera:

Nombre las cookie: SSO (definida en su application.properties). Este nombre debe ser único para todo Genesis.

Valor de la cookie se obtiene llamando a Oauth2srv con el siguiente servicio:

```
REST GET /sso/cookie?credentials=CCCCC&userCredentials=UUUUUU
```

nota: el REST debe tener un token válido.

Parámetro de `credentials`:

```
nombreAplicacion:secretPassword
```

Estos son los valores que se utilizan para pedir un token es decir por seguridad debe estar registrada la aplicación y su llave secreta en el servidor Oauth2srv para evitar hackeo.

Parámetro de `userCredentials`:

```
usuarion:password
```

Son los datos con los que el usuario acceso el sistema.

nota: Por seguridad ambos parámetros se encuentran codificado con Base64.

Return value: codificado en Base64 el unique-id de la sesión SSO. Ese debe ser el valor que se almacene en la cookie SSO.

## PWA: Log out.

Para terminar una sesión de Sso de manera ‘normal’ el PWA deberá de incluir una opción de log-out (parte superior derecha del menú, donde se tiene preferencias… y acerca de…).

La opción de log-out debe hacer tres cosas:

1. Llamar el event-logger para generar un evento de salida de Genesis.
2. Borrar el cookie de SSO del browser Chrome.
3. Llamar a Oauth2srv con el siguiente servicio REST:

```
REST GET /sso/logout?uName=nombreUsuario
```

nota: el REST debe tener un token válido.

Parámetro de `uName`:

Nombre del usuario que se está saliendo del PWA.

Return value:

‘exited user:*uname’*

o bien

‘no session’

En ambos casos el PWA no debe hacer nada, solo que valide que recibió un status 200 OK. (el no session puede darse si ya se había terminado la sesión manualmente pro el IAM UI y no es una condición de error).