**Exemples d'interacció amb la API**

**Introducció**

En aquesta secció de la memòria seran presentats els diferents exemples implementats en la nostre aplicació web que interactuen amb la API de FamilySearch.

Per cada una de las funcionalitat es proporcionarà una descripció adequada de la seva funció, detalls de la implementació i patrons d'usabilitat i exemples d'utilització si s'escau.

**Identificació i desconnexió amb l'API de FamilySearch**

**Descripció de la funcionalitat**

Les funcionalitats d'identificació i desconnexió amb la API de FamilySearch s'utilitzen exactament pel que el seu nom indica, per permetre als usuaris de l'aplicació web identificar-se o desconnectar-se a FamilySearch des de la pàgina d'un tercer, en aquest cas, nosaltres.

Els usuaris no tenen permisos d'interacció amb l'API a menys que tinguin un compte propi de FamilySearch i s'identifiquin en el sistema. El motiu és que un cop identificats cada usuari rep un 'token' o 'fitxa' personal, que un cop ajuntat a les crides a l'API, fa que aquestes siguin acceptades pel sistema.

En l'aplicació web implementada, la possibilitat d'identificar-se amb FamilySearch apareix quan l'usuari intenta accedir a la secció d'exemples o a un exemple concret. La pàgina en qüestió és relativament simple. L'usuari té o bé l'opció de tornar enredera o identificar-se amb FamilySearch.

La figura [] mostra la pàgina d'identificació.

Un cop els usuaris es troben identificats, tenen la opció de desconnectar-se des de qualsevol pàgina del web mitjançant un botó situat a la part dreta de la barra de navegació.

**Detalls de la implementació**

El client de FamilySearch

Degut a que es va decidir interactuar amb l'API de FamilySearch des de la capa controlador de la pàgina web, cal crear una instància del client cada cop que l'usuari canvia de pàgina.

Això és degut a que cada cop que el navegador canvia de URL, també neteja el conjunt de variables globals declarades en els fitxers Javascript. No obstant, això no vol dir que l'usuari s'hagi d'identificar de nou cada cop que canvia de pàgina.

El fitxer 'client.js' és l'encarregat de crear les instàncies del client i gestionar la concordança d'estat entre la connexió a FamilySearch i la informació emmagatzemada en el nostre back end o servidor. Recordem que el servidor és el que decideix si l'usuari pot accedir i utilitzar els exemples i que per tant esdevé important mantenir aquesta concordança d'estats.

A pesar de que l'usuari només ha d'estar identificat a les pàgines d'exemples, al voler oferir la possibilitat de desconnexió des de qualsevol pàgina de l'aplicació, el fitxer 'client.js' forma part dels documents Javascripts inclosos en el fitxer 'javascripts.html' que és inclòs a totes les pàgines del web.

El client FamilySearch és declarat a cada pàgina mitjançant la funció 'new Familysearch()' proporcionada des del SDK.

*var client = new FamilySearch({  
 client\_id: 'a02j000000E5DXqAAN',  
 redirect\_uri: document.location.protocol + '//' + document.location.host + '/',  
 save\_access\_token: saveCookie,  
 access\_token: token,  
 auto\_expire: true,  
 auto\_signin: true,  
 expire\_callback: function(data) { ... },  
 environment: 'sandbox'  
});*

Els paràmetres inclosos en la creació de la instància del client marquen les següents característiques:

* **client\_id:** Número del client que identifica l'aplicació a la plataforma de desenvolupadors de FamilySearch.
* **redirect\_uri:** URL de redirecció que s'ha registrat a la plataforma de desenvolupadors per l'aplicació.
* **save\_access\_token:** Variable que permet emmagatzemar el 'token' retornat per l'API en una galeta quan un usuari s'identifica i permet crear instàncies del client a totes les pàgines del domini web sense la necessitat d'anar demanant a l'usuari que s'identifiqui. En el nostre cas es tracte d'un booleà que contindrà el valor 'False' quasi sempre, ja que utilitzem un mètode alternatiu per emmagatzemar el 'token' d'accés que compleix amb els requisits de certificació.
* **test\_token:** En cas de que l'usuari ja es trobi identificat amb FamilySearch li passem el 'token' emmagatzemant de forma automàtica. Això ens permet no haver de demanar a l'usuari que s'identifiqui en cada una de les pàgines.
* **auto\_expire:** Booleà que indica si volem que el sistema netegi el 'token' de forma automàtica en cas de que aquest quedi invalidat.
* **auto\_signin:** Booleà que indica si volem que es demani a l'usuari que s'identifiqui cada cop que intenta realitzar una operació sense trobar-se connectat a FamilySearch. En la nostre aplicació, li passem el paràmetre 'True', no obstant, no s'hauria de poder donar el cas en que un usuari pogués llençar una crida contra la API sense trobar-se identificat. L'utilitzem com a mesura de seguretat.
* **expire\_callback:** Funció a executar quan el 'token' expira. Conté quasi el mateix codi que la funció de desconnexió implementada, així que en veurem el codi més endavant.
* **environment:** Entorn de l'API de FamilySearch en el que ens volem connectar: 'production', 'sandbox', etcètera.

Fins aquí, tot és relativament simple, però a on s'emmagatzema el 'token' per tal de complir amb les regles de certificació i poder reutilitzar-lo? La resposta és en l'espai local del navegador. Els navegadors moderns permeten escriure en el que és coneix com l'espai local del navegador, el mateix lloc on els navegadors emmagatzemen les imatges i recursos d'una aplicació web per tal de reutilitzar-les en futures crides, evitant descarregar-les de nou i augmentant així la velocitat de navegació.

S'utilitza l'espai local per emmagatzemar el 'token' ja que és un espai on tercers no poden entrar i robar així la identificació d'un usuari.

Al principi del fitxer 'client.js' és comprova si el navegador suporta l'escriptura a l'espai local, en cas afirmatiu es carrega el 'token' emmagtzemat si aquest existeix o es configura el paràmetre que permetrà guardar el 'token' en un galeta, a pesar de violar les normes de certificació.

Aquets dos paràmetres són els que s'utilitzen en la creació del client descrita fa uns moments.

*if (typeof(Storage) !== "undefined") {  
 token = localStorage.token ? localStorage.token : '';  
} else {  
 saveCookie = true;  
}*

La funció d'identificació amb FamilySearch

La funció d'identificació es realitza des de la pàgina 'login.html' de l'aplicació i el controlador que la gestiona està inclosa en el fitxer 'login.js'.

Quan l'usuari selecciona el botó 'Access FamilySearch' mostrat a la figura [], el controlador executa la funció 'getAccessToken()' proporcionada pel SDK de FamilySearch. Aquesta funció obre un pop-up que permet a l'usuari identificar-se i n'espera la resposta de forma asíncrona.

En cas que l'usuari s'identifiqui de forma correcta, s'emmagatzema el 'token' retornat per la funció del SDK a l'espai local d'emmagatzematge del navegador i és llença la funció 'serverLogIn(token)' que s'encarregarà de mantenir la concordança d'estat entre un token vàlid a FamilySearch i el nostre servidor.

*client.getAccessToken().then(function(testTokenValue) {  
 localStorage.token = testTokenValue;  
 serverLogIn(testTokenValue);  
});*

La funció serverLogin està inclosa en el fitxer 'client.js' que recordem està inclòs en totes les pàgines del web i el que fa és realitzar una crida AJAX al servidor, passant com a paràmetre el 'token' retornat per l'API. Un cop el servidor ha processat la petició, la funció definida en el paràmetre 'success', redirecciona l'aplicació a una pàgina indicada pel servidor.

*function serverLogIn(apiToken) {  
 $.ajax({  
 type: "POST",  
 url: "/token/login",  
 data: JSON.stringify({token: apiToken}),  
 dataType: 'json',  
 contentType: 'application/json; charset=UTF-8',  
 success: function(data) { ... }  
 });  
}*

El servidor, per la seva banda, crea una galeta de sessió segura i no modificable pel client amb el 'token' codificat. Mentre aquesta galeta existeixi, el servidor permetrà l'accés de l'usuari a les pàgines d'exemples implementats.

*app.post('/token/login', function(req, res) {  
 req.session.logged = req.session.logged || req.body.token;  
 res.end('{"redirect" : "/examples"}');  
});*

La funció de desconnexió amb FamilySearch

Aquesta funció s'invoca de forma automàtica quan el 'token' expira o quan l'usuari utilitza la funcionalitat de desconnexió situada a la barra de navegació.

La funció de desconnexió amb FamilySearch realitza el procés invers realitzat per les funcions 'getAccessToken()' i 'serverLogIn()' de l'apartat anterior. Aquesta, invalida el 'token' actual configurat en el client, elimina el valor contingut a l'espai local d'emmagatzematge i sincronitza l'estat amb el servidor mitjançant una crida AJAX com la de l'apartat anterior.

*function serverLogOut() {  
 client.invalidateAccessToken();  
 localStorage.removeItem('token');*

*$.ajax({  
 type: "POST",  
 url: "/token/logout",  
 ...  
 });  
}*

Per la seva banda, el servidor elimina la galeta de sessió simplement assigannt-li el valor 'null'.

*app.post('/token/logout', function(req, res) {  
 req.session = null;  
 res.end('{"redirect" : "/"}');  
});*

**Aspectes d'usabilitat considerats**

Les funcions de connexió i desconnexió realitzen molta feina transparent per l'usuari, no obstant, si que s'ha volgut incloure un petit aspecte d'usabiliat per informar a l'usuari de que s'està esperant la seva identificació amb FamilySearch quan aquest prem el botó d'identificació.

Quan el pop up d'identificació amb FamilySearch és mostra, el contingut de la pàgina 'login.html' s'esvaeix lentament i apareix el gif mostrat en la figura [] que indica que s'està esperant una interacció per part de l'usuari. Quan l'usuari acaba el procés d'identificació és redirigit de forma automàtica a la pàgina d'exemples.

[]

Un petit segon exemple d'usabilitat és que el botó de desconnectar-se només es mostra si l'usuari es troba identificat.

**Cerca de persones a l'arbre familiar de FamilySearch**

**Descripció de la funcionalitat**