**Familysearch SDK oficial Javascript**

**Introducció**

Abans de procedir a explicar com funciona l'aplicació web desenvolupada, volíem presentar per sobre en què consisteix el SDK oficial Javascript de FamilySearch.

No explicarem la informació accessible a través d'aquest, perquè en realitat, es tracta gairebé de la mateixa que la presentada en la secció cinc d'aquesta memòria. No obstant això, creiem que pot resultar interessant presentar les principals funcionalitats que aquest SDK ofereix i exposar els diferents motius, pels que aquestes funcionalitats, compensen l’esforç d’estudiar-lo.

L’objectiu del SDK, per resumir-ho d’una forma simple, és facilitar el consum dels recursos accessibles a través de l'API de FamilySearch.

Per aconseguir-ho, el SDK envolta cada funció que realitza una crida a l’API amb una funció pròpia i afegeix funcions de conveniència a la resposta per tal de navegar pels resultats de forma còmoda. De totes maneres, el SDK no emmascara la resposta retornada per l’API i en conseqüència, els recursos JSON o XML retornats per aquesta, són accessibles en cas de voler accedir a una peça d'informació concreta que no hagi estat emmascarada en una funció de conveniència.

En cas que el SDK no estigués preparat per fer front a alguna de les peticions que l’API ofereix, aquest implementa un conjunt de funcions de plomeria que permeten a l'usuari realitzar amb comoditat les típiques crides GET, POST, DEL, etcètera, que es realitzarien contra l’API en cas d’una integració directa, mitjançant el SDK.

Per tant, el SDK ens ofereix una cobertura quasi completa de l’API oficial amb funcions de conveniència i en cas que alguna consulta no fos suportada, sempre es podria realitzar mitjançant la via tradicional a través del SDK.

A continuació, s'expliquen les principals funcionalitats d'aquest SDK i en què consisteixen.

**Crides REST a l’API envoltades per funcions del SDK**

Qualsevol crida que es pretengui realitzar contra l’API de FamilySearch, es realitzarà en el SDK mitjançant una funció Javascript asíncrona que l'emmascara. Això ens permet no haver de preocupar-nos per les URI dels recursos als quals volem accedir o haver d'afegir les capçaleres correctes a cada petició, ja que és el mateix SDK el que s'encarrega de fer-ho i mantenir el conjunt d'URIs als diferents recursos i operacions actualitzat.

Per exemple, si volem accedir a la persona amb identificador: `KW7S-VQJ', ho podríem fer mitjançant la següent funció.

client.getPerson('KW7S-VQJ', {persons:true}).then(function(response) {  
 ...   
});

La variable *client* representa una instància del SDK, l’operació, *getPerson,* l’operació del SDK que volem invocar, l’identificador *KW7S-VQJ* i el JSON *{persons:true},* són els paràmetres a passar a la funció i la variable *response,* és l’objecte que emmagatzemarà la resposta de l’API.

**Implementació basada en Promeses**

Les promeses són els objectes retornats pels blocs de codi Javascript asíncrons, com per exemple, en la funció de l’apartat anterior, la promesa retornada seria la variable *response.* Una promesa es troba sempre en un dels següents tres estats:

* Pendent: Estat inicial de la promesa.
* Satisfeta: L'estat de la promesa representa una operació finalitzada amb èxit.
* Rebutjada: L'estat de la promesa representa una operació fallida.

Tan bon punt una promesa rep l'estat de satisfeta o rebutjada, ja no pot tornar a canviar d'estat.

La gràcia de les promeses és que tan bon punt són resoltes, permeten executar una part del codi definida amb anterioritat (el que seria el cos de la funció de l'apartat anterior) i mentre aquestes no són satisfetes o rebutjades, la resta del codi pot seguir executant-se. Amb altres paraules, el codi no roman bloquejat mentre espera la resposta de la funció contra l’API.

**Implementació pensada per la programació orientada a objectes**

Les funcions de crida del SDK, retornen promeses, que no deixen de ser objectes Javascript. Aquests objectes, disposen de funcions de conveniència que permeten accedir als resultats de la resposta sense haver de navegar per objectes de format XML o JSON.

Per exemple, en la crida de fa dos apartats, podríem obtenir el nom de la Persona cercada, a través de l’objecte *response*, mitjançant la navegació al recurs de la persona i seguidament, demanant-ne el nom. El següent bloc de codi mostra aquest exemple.

client.getPerson('KW7S-VQJ', {persons:true}).then(function(response) {  
 console.log(response.getPrimaryPerson().getDisplayName());  
});

**Model de dades pràcticament idèntic al de FamilySearch**

Una altra característica destacable del SDK és que implementa un model de dades equivalent al de l’API de FamilySearch, però en aquest cas, pensat per ser navegat mitjançant els estàndards dels llenguatges de programació orientada a objectes, en comptes dels enllaços hypermedia.

A la figura [ref] podem observar el model de dades proposat pel SDK i com cada un dels objectes es troba relacionat amb els altres.

**Captura d'errors**

Els errors de connexió amb l’API són fàcilment tractables gràcies a la naturalesa de les crides asíncrones implementades pel SDK. Només cal afegir la clàusula `.catch' al codi de la petició, tal com es mostra en el següent bloc de codi. La variable *error* és un altre objecte Javascript que conté informació sobre l’error.

client.getPerson('KW7S-VQJ', {persons:true}).then(function(response) {  
 // Tractar les dades retornades per l’API  
})  
.catch(function(error) {  
 // Codi per gestionar l'error  
});

**Intentar de nou peticions GET fallides**

Els errors de connexió amb l’API són fàcilment tractables gràcies a la naturalesa de les crides asíncrones implementades pel SDK. Només cal afegir la clàusula `.catch' al codi de la petició, tal com es mostra en el següent bloc de codi. La variable *error* és un altre objecte Javascript que conté informació sobre l’error.

**Gestió automàtica del Throttling**

Com s'exposava en la quarta secció de la memòria, quan es parlava d'algunes de les funcionalitats extres que l’API de FamilySearch oferia, va ser presentada la funcionalitat de *Throttling,* que evitava que un usuari realitzes `masses' peticions, durant un cert període de temps.

En cas que l'usuari del SDK sigui bloquejat durant un període de temps per aquesta causa, el mateix SDK, s'encarregarà de gestionar la duració del bloqueig, evitant rellançar la crida fins que aquesta pugui tornar a ser llençada.

**Autentificació mitjançant un Pop-up**

Mentre es realitzi des d'un navegador, l'autentificació d’una aplicació amb FamilySearch, pot ser gestionada mitjançant un pop-up a la pàgina oficial de l’organització.

D'aquesta forma, no ens hem de preocupar de crear una pàgina de redirecció específica pel protocol OAuth. L'únic requisit, és registrar un URL vàlid, en el mateix domini i port que l’aplicació web, que serà cridada un cop es finalitzi el procés d'identificació a la banda de FamilySearch.

**Autentificació automàtica**

Existeix l’opció d'activar el protocol d'identificació de forma automàtica, en cas que un usuari intenti realitzar una operació contra l’API sense identificar-se primer, aquesta acció serà demanada abans d’executar la crida. Pot esdevenir útil, ja que les connexions expiren al cap d'un temps d'inactivitat.

**Emmagatzematge del Token en una cookie**

Existeix l'opció d'emmagatzemar el token d'identificació (retornat per l’operació d'identificació amb FamilySearch) en una cookie.

Resulta útil de cara a utilitzar el SDK des de la capa del controlador, ja que d’aquesta forma no resulta necessari crear una instància del client a cada pàgina diferent de l’aplicació web o implementar una aplicació web basada en una sola pàgina.

Malauradament, viola una de les condicions per la certificació de les aplicacions, així que no s'acaba d'entendre perquè apareix en un SDK oficial, mes enllà d'ajuda durant el desenvolupament.

Per tal d'intentar superar aquesta restricció, en el nostre projecte, hem emmagatzemat el valor del Token en l’espai local del navegador.

**Disparador automàtic de la funció d'expiració**

Per tal de mantenir la coherència entre l'aplicació web i l'estat de la identificació, recordem que aquesta pot expirar per inactivitat, existeix la possibilitat d'implementar una funció que s'executi quan el token expira.

D'aquesta forma, podem garantir que en tot moment existeix una concordança d'estat entre el client i el servidor i que per tant, no correm el risc que l'usuari pugui realitzar operacions no permeses en expirar el token d’identificació.

**Utilitzable des de diferents plataformes**

El SDK és utilitzable tant des del client com el servidor. En altres paraules, pot córrer tant mitjançant la tecnologia Javascript en la capa del controlador, com mitjançant Node.js a la capa del back-end.

**Suport en el desenvolupament**

Una de les característiques principals del SDK és que es troba relativament ben documentat. Inclòs m'atreviria a dir, que de cara a obtenir una idea general de com funciona l’API de FamilySearch i quina informació conté, realitza una millor feina que la documentació oficial de l’API. Encara que sigui de forma involuntària, és un punt a destacar.

Al mateix temps, existeixen diferents grups d'ajuda per aquells desenvolupadors que desitgin utilitzar l'API de FamilySearch i que són visitats de forma regular pels desenvolupadors dels SDK oficials i l’API de FamilySearch.

Des d'aquesta memòria, es recomana passar pel grup de Google `FamilySearch Developer Network', ja que resulta l'emplaçament ideal per qualsevol dubte que puguem tenir i un centre d'informació ideal sobre el funcionament de l’API i l'estat en què es troba a cada moment.

**Conclusió**

Com s'ha pogut observar en les seccions anteriors, són molts els beneficis resultats d'utilitzar un SDK oficial, en comptes de realitzar una implementació directa contra l’API. Per altra banda, el preu a pagar és pràcticament nul, més enllà d'haver d'estudiar el funcionament del SDK, però resulta un esforç rendible.