**Introducció a la API de FamilySearch**

**Introducció**

Tota la informació disponible i necessària per tal de poder començar a treballar sobre l’API de FamilySearch és pot trobar en el portal de desenvolupadors de la pàgina web.

Aquesta secció de la pàgina web esta formada per diferent seccions, però l’estructura no acaba de resultar del tot clara per una persona que vulgui començar de cero. Si ens enfoquem més en la documentació disponible que no pas en la estructura proposada per l’organització, podem veure que la informació es distribueix en certa forma en els següents grups:

* **Requisits tècnics:** Conjunt d’informació necessària per comprendre l’estructura de l’API, els formats de dades que maneja i els passos necessaris per tal de començar a interactuar amb aquesta.
* **Recursos disponibles i rutes d’accés:** Informació detallada sobre cada recurs accessible a través de l’API. En concret disposa dels detalls de com accedir aquest recurs, les operacions que es poden realitzar sobre ell, la informació que conté i quines són les connexions amb altres recursos.
* **Capacitats extres de l’API:** Aquest recull d’articles conceptualitza característiques de l’API com poden ser els recursos de *catching, localització o throttling.*
* **Evolució i canvis produïts a l’API:** Informació semi ordenada de com l’API s’ha vist evolucionada al llarg del temps i un recull dels canvis produïts a l’API, procés de certificació, material de documentació i eines de desenvolupament.
* **Eines de desenvolupament:** Recull d’entorns de desenvolupament i eines extres que poden facilitar la feina del programador.

**Arquitectura de l’API i formats de dades**

**Que és una API**

**Arquiectura**

L’arquitectura que l’API de FamilySearch utilitza és una arquitectura REST. Les sigles REST representen en anglès: *Representational State Transfer”.*

Les arquitectures REST es caracteritzen per estar orientades els recursos més que en les accions que es poden realitzar sobre ells i com a peculiaritat implementen sis restriccions; a saber, *uniform interface, stateless, client-server, cacheable, layered system, code on demand.* Aquestes sis restriccions son les que defineixen les bases de les arquitectures REST.

És diu que una arquitectura REST és orientada als recursos perquè estan construïdes al voltant d’objectes en comptes d’accions. Per exemple, a l’API de FamilySearch es parla de Persones i Esdeveniments en comptes de llegir persones o escriure esdeveniments.

Les representacions expliquen com els recursos són tractats per l’API i de quina forma s’han de realitzar les comunicacions entre el servidor i el client. Els formats més freqüents són JSON i XML. FamilySearch dona suport a ambdós formats.

Per posar un exemple reduït que il·lustri el que estem explicant, podem descriure de la següent forma el que podria ser una operació contra la API de FamilySearch especificant quin element seria considerat el Recurs, quin el Servei i quin la Representació.

* **Recurs:** Persona
* **Servei:** Informació de la persona (GET)
* **Representació:** Nom, cognoms, esdeveniments relacionats amb la seva vida, etcètera, en format XML o JSON.

Com també havíem comentat, l’arquitectura REST es caracteritza per sis restriccions que el sistema imposa.

**Interfície uniforme (uniform interface)**

Defineix la interfície entre el client i el servidor. Que ve a significar que utilitzem els protocol de comunicació HTTP o HTTPS i les URI (Identificador de Recurs Uniforme, també conegudes com “URL” en camps diferents), per aconseguir accés als recursos i operacions de l’API.

Els verbs permesos pels protocols de comunicació web HTTP i HTTPS són: GET, PUT, POST, DELETE, OPTIONS and HEAD.

Per exemple, per fer la petició de lectura sobre el recurs d’una Persona a l’API de FamilySearch executaríem la següent crida HTTP/HTTPS:

GET /platform/genealogies/persons/2:2:PPPJ-MYZ7

**Sense Estat (stateless)**

El servidor no emmagatzema informació del client. Això implica que cada petició a l’API ha de contenir tota la informació necessària per poder processar la petició.

En altres paraules, l’estat és emmagatzemat en el client.

Hi ha exemples com el Oauth V2, que no són realment RESTful perque guarden informació del client.

**Client–Servidor (client-server)**

Hem de comprendre que el sistema client-servidor és un sistema desconnectat. Per tant hem de tenir sempre en compte que el client mai tindrà accés directe a les bases de dades que emmagatzemen la informació, sinó que ha de passar sempre per l’intermediari que és l’API.

El protocol de comunicació, o la interfície de comunicació que ja hem descrit fa un moment fa les funcions d’enllaç del sistema.

**Cacheable**

Fa referència a que les respostes del servidor poden ser guardades o emmagatzemades. Existeixen tres nivells de ‘cache’:

* **Implícit:** Si és el client el que decideix quan de temps guardarà els paràmetres o informació és tracte de implícit caching.
* **Explícit:** Si és el servidor el que mana i posa les regles, parlem de explícit caching.
* **Negociat:** Quan el client i el servidor negocien i arriben a un acord, es tracte de caching negociat.

**Sistema per capes (Layered System)**

Aquest principi es basa en que el client no pot assumir que tindrà direcció directa amb el servidor. És a dir, existeixen intermediaris en forma de software i hardware entre el client i el servidor.

Això facilita l’escalabilitat del sistema, perquè el client no s’ha de provocar de comunicar-se amb elements o tecnologies concretes que es poden veure subjectes a canvis degut a canvis de necessitat per part del servidor.

**Codi sota petició (code on demand)**

De forma excepcional el servidor pot donar accés al client a la lògica del funcionament del servidor. Alguns exemples poden ser els ‘Java Applets’ o ‘blocs de codi JavasScript.

**Conclusió**

En conclusió, seguir les regles marcades per una arquitectura REST ens permet assegurar els següents beneficis: escalabilitat, simplicitat, modificabilitat, visibilitat, portabilitat i fiabilitat.