2023

# CTTI – Bones pràctiques

Generalitat de Catalunya
**Centre de Telecomunicacions i Tecnologies de la Informació**

# Table of Contents

## FULL DE CONTROL DE LA DOCUMENTACIÓ

| **TÍTOL** | | |
|---|---|---|
| Document de bones pràctiques per a la plataforma d' API Connect. | | |
| **CODI** | **1A EDICIÓ** | **EDICIÓ VIGENT** |
| | Data: 06/11/2023 | EDICIÓ Nº: 1 |

| **CLASSIFICACIÓ** | **TIPUS DE DOCUMENT** | **ESTAT** |
|---|---|---|
| Públic | Document Tècnic **X** | Esborrany **X** |
| Intern **X** | Presentació | En Revisió |
| D' exclusiu ús per | Proposta/Informe | Actualitzable |
| Confidencial | Altres: | Informe Final |
| **NOM DE FITXER** | CTTI_Bones_Practiques_v1.0 | |

| | **NOM / LLOC** | **SIGNATURA / DATA** |
|---|---|---|
| **REALIZADO**<br>(responsable d'actualització i manteniment del document) | | |
| **REVISAT** | | |

- **Històric de versions**

| Data efectiva | Versió | Descripció | Autor | Aprovat |
|---|---|---|---|---|
| 06-11-2023 | 1.0 | Versió inicial- Plantilla | Accenture | Accenture |
| | | | | |

# 1 Introducció

En aquest document, es defineixen els estàndards i les bones pràctiques a l' hora de realitzar desenvolupaments d' interfícies REST sobre la plataforma IBM API Connect.

**REST (Representational State Transfer)** és un tipus d'arquitectura de desenvolupament web stateless dissenyat per funcionar amb l'estàndard HTTP, permetent crear serveis que puguin ser usats per qualsevol dispositiu o client que interpreti aquest protocol. El seu principi fonamental és la definició de recursos, que poden ser manipulats usant un nombre reduït de mètodes. En usar el protocol HTTP, els mètodes es corresponen amb els mètodes HTTP (GET, POST, PUT, DELETE...) mentre que els recursos es mapegen de forma natural a URIs.

Es fa servir aquesta arquitectura perquè ofereix molta facilitat per al descobriment de serveis, proveint endpoints predecibles que són invocats mitjançant peticions HTTP estàndard i molta flexibilitat.

# 2 Bones pràctiques

## 2.1 Mètodes HTTP

En realitzar trucades a una API REST, es fa ús dels **mètodes/verbs HTTP** per interaccionar amb un recurs i realitzar una determinada acció sobre aquest recurs invocat. Aquesta acció pot ser considerada **segura** si no produeix canvis en l'estat del servidor (només els mètodes de lectura són segurs), i pot ser considerada **idempotent** si l'execució repetida d'aquesta acció usant els mateixos paràmetres té el mateix efecte que si es realitza una sola vegada.

A continuació, s'exposen els mètodes HTTP que es faran servir en el disseny de les interfícies REST, els quals fan en la seva majoria una de les quatre operacions bàsiques **CRUD (Create, Read, Update, Delete)**, juntament amb les seves característiques i casos d'ús:

| **Mètode** | **CRUD** | **Assegurança** | **Idempotent** | **Quan usar** |
|---|---|---|---|---|
| GET | Read | Sí | Sí | Per llegir les dades d' un recurs.<br>No s' ha de fer servir per crear, modificar o esborrar un recurs. |
| POST | Create | No | No | Per crear un recurs nou en el servidor.<br>No s' ha de fer servir per modificar o esborrar un recurs. Es pot fer servir per llegir dades d'un recurs si la trucada requereix de molts paràmetres i amb un GET es podria arribar al límit de caràcters d'una URI (**2048**). |
| PATCH | Update | No | Sí | Per modificar un recurs existent de forma parcial.<br>No s'ha de fer servir per crear, llegir o esborrar un recurs. |
| PUT | Update | No | Sí | Per modificar un recurs existent de forma completa.<br>No s'ha de fer servir per crear, llegir o esborrar un recurs. |
| DELETE | Delet | No | Sí | Per esborrar un recurs del servidor.<br>No s' ha de fer servir per crear, llegir o modificar un recurs. Si cridar usant DELETE provoca que, per exemple, un comptador disminueixi, llavors la trucada no seria idempotent. |
| HEAD | | Sí | Sí | Per a l'existència d'un recurs en el servidor i obtenir els mètodes associats al seu estat.<br>No s' ha de fer servir per obtenir el contingut d' un recurs. |

| | | | | |
|---|---|---|---|---|
| OPTIONS | | Sí | Sí | Per descobrir els mètodes d' interacció HTTP possibles amb el servidor de recursos.<br>No s' ha d' agregar aquest verb a la definició Swagger ja que s' activa a nivell de servidor i està implícit. |

## 2.2 Disseny d' URIs

Com s'esmenta en la introducció, REST es basa en la manipulació mitjançant mètodes HTTP d'una sèrie de recursos, els quals seran localitzats dins del servidor mitjançant **URIs (identificadors uniformes de recursos)**. Aquestes URIs han de ser definides de manera que es pot identificar de manera ràpida sobre què s' està realitzant una acció.

A grans trets, una URI d' una API ha de complir amb els criteris següents:

- Ha de proveir la **localització o direcció del servidor** en el qual es troba desplegada l' API, o del balancejador que precedeix aquest servidor.
- Ha de proveir el **nom de l' API**.
- Pot proveir la **versió de l' API**, però no és obligatori.
- Ha de proveir el **recurs** on s' està executant l' acció definida pel mètode HTTP.

Quant a la nomenclatura i format de l' esmentada URI, s' han de complir les regles generals següents:

- El **separador /** hauria d'indicar una relació de jerarquia i no s'ha de fer servir com a últim carácter.
- La nomenclatura dels recursos hauria de ser en **kebab-case**. Això significa que no s'hauria de fer servir barres baixes (_) sinó guions (-) per unir les zones on hi hauria d'haver espais.
- S'hauria de buscar **evitar lletres majúscules** dins d'una URI, sent d'especial importància evitar-ho en hostnames, basepaths i paths.
- Les **extensions dels fitxers** no han de ser incloses dins la URI, sinó que s' ha de fer servir la capçalera **Content-Type** per especificar el format d' arxiu a consultar.
- Els caràcters **alfanumèrics** haurien de ser usats amb **poca freqüència**, atès que alguns caràcters no poden formar part d' una URI o tenen un significat especial si en formen part.

Aquestes normes són normes generals que s' haurien d' aplicar per a totes les seccions de la URI. Ara es comentarà el cas particular dels recursos, oferint algunes guies per al seu disseny.

### 2.2.1 Disseny de Recursos

El model de recursos d'una API ha d'establir una jerarquia dins de l'API, on cada nivell jeràrquic dins d'aquest model estarà separat de l'anterior mitjançant el **separador /**. Aquesta estructura jeràrquica hauria d' anar sempre d' un recurs de major rellevància a un menor, permetent que, en llegir la URI, es pugui identificar inequívocament el recurs al qual estem accedint.

Per establir aquesta jerarquia, primer s' ha de tenir clara la definició dels possibles arquetips d' un recurs, que són:

- **Document.** Un recurs d' aquest tipus és similar en concepte a una instància d' un objecte o una fila d' una base de dades. Representa una unitat atòmica d' informació.
- **Col·lecció.** Una col·lecció representa un conjunt de documents del mateix tipus.
- **Magatzem.** Un magatzem és un concepte similar a una col·lecció que representa un repositori administrat pel client i que permet al client de l'API emmagatzemar recursos, obtenir-los i decidir quan esborrar-los. Un magatzem mai genera noves URIs, sinó que cada recurs de tipus magatzem té una URI. Un exemple d'aquest tipus de recurs és el d'una llista de reproducció per a un proveïdor de música. La URI podría ser: https://api.example.com/song-management/users/{id}/playlists
- **Controlador.** Un controlador descriu una funció executable, un procediment. En el cas d'un controlador, no es tracta la informació que maneja l'API, sinó les operacions que pots fer sobre ella. Un exemple d'un recurs d'aquest tipus és: https://api.example.com/cart-management/users/{id}/cart/checkout

Tenint clars aquests conceptes, per fer ús d' una bona pràctica, el modelatge de recursos hauria de seguir les següents guies i regles:

- Qualsevol recurs, sigui de l' arquetip que sigui, ha de començar amb un **caràcter alfabètic.**
- Per representar un **document**, s' ha de fer servir un **nom singular**.
- Per representar una **col·lecció**, s'ha de fer servir un **nom en plural**.
- Per representar un **magatzem**, s' ha de fer servir un **nom en plural**.
- Per representar un **controlador**, s' ha de fer servir un **verb**. **No** hauria de ser cap dels verbs associats a les accions **CRUD (Create, Read, Update, Delete)** atès que, per a aquestes accions, el millor és usar el mètode HTTP per representar-les.
- Cada nivell de la jerarquia ha de retornar algun valor, sent aquest valor la resposta natural a la pregunta **"Què és el que espera el consumidor?"**
- S'hauria de buscar, si és possible, que les URIs no s'estenguin més enllà de **quatre nivells jeràrquics** amb el format **/col·lecció/document/col·lecció/document**.
- La nomenclatura de les APIs serà, principalment, en **català**, en ser l' idioma principal usat en CTTI, tot i que també es podrà fer servir l' **espanyol**. Existirà una excepció a aquesta regla i és que, si aquesta API s' ha d' **internacionalitzar**, la nomenclatura de l' API haurà de ser en **anglès**.

## 2.3 Codis HTTP

Quan es fa una petició HTTP, es retorna sempre una resposta, la qual vindrà amb un codi HTTP de resposta de tres xifres. Aquest codi indica què ha ocorregut amb la petició, ja que cada codi té un significat concret, tot i que es poden agrupar els codis en cinc grans grups en funció de la centena a la qual pertanyin:

- **Codis 1xx.** Codis associats a respostes informatives sobre l' estat de la petició.
- **Codis 2xx.** Codis associats a peticions rebudes, processades i respostes correctamente.
- **Codis 3xx.** Codis que indiquen que es necessita realitzar una acció addicional com pot ser una redirecció.
- **Codis 4xx.** Codis associats a errors produïts pel client.
- **Codis 5xx.** Codis associats a errors del costat del servidor.

En el disseny de les respostes d' una API REST, s' han de tenir en compte aquests codis i definir els diversos tipus de resposta usant el codi adequat en cada cas. Per exemple, no s' ha de retornar una resposta d' un error de la petició per part del client usant un codi 2xx.

El següent llistat mostra els principals codis de resposta HTTP juntament amb el seu significat i quan s' haurien de fer servir:

| Codi | Descripció | Significat | Quan usar |
|---|---|---|---|
| 200 | OK | La petició ha tingut èxit. | Quan s' obtingui una resposta correcta per part del servidor. Es pot fer servir per a tots els mètodes. |
| 201 | Created | La petició ha tingut èxit i s'ha creat un nou recurs. | Quan es genera un nou recurs en un servidor mitjançant una trucada **POST**. |
| 202 | Accepted | La petició ha estat acceptada i està sent processada, però no completada. | Quan l'operació és asíncrona. Normalment, un API retorna un 202 juntament amb una URI de localització on es pot revisar l'estat de l'operació. |
| 204 | No Content | La petició ha estat exitosa, no es retorna contingut. | Quan es realitza una trucada **DELETE**. Es pot retornar bé un 200 amb contingut, o un 204 sense contingut. |

| | | | |
|---|---|---|---|
| 400 | Bad request | La petició és invàlida per algun motiu. | Quan es produeix un error de sintaxi o de validació de la petició. |
| 401 | Unauthorized | La petició requereix d' autenticació o l' autenticació proveïda és incorrecta. | Quan el header **Authorization** ve buit o el seu valor és incorrecte. S' ha de fer servir en tots els recursos que requereixin autenticació. |
| 403 | Forbidden | La petició no està autoritzada per accedir al recurs. | S' ha de fer servir en tots els recursos que requereixin autorització. |
| 404 | Not Found | No es troba el recurs darrere de la URI. | Quan no es troba el recurs que es busca en la petició. |
| 405 | Method not allowed | El mètode de la petició no és permès. | Quan el mètode HTTP usat no coincideix amb els permesos per a aquest recurs. |
| 408 | Request Timeout | La petició va trigar massa a processar. | Quan el servidor no rep una resposta completa des del client durant el període de timeout configurat. |
| 429 | Too many requests | El límit de peticions va ser excedit. | Quan s' estableixen polítiques de control de trànsit i s' excedeixen els límits establerts. |
| 500 | Internal Server Error | Hi ha hagut un error intern en el servidor i no es va poder processar la resposta. | Quan s' esdevingui un error intern en el servidor inesperat i no es reconegui. És el codi d' error genèric per a aquests errors. |
| 502 | Bad Gateway | El servidor va obtenir una resposta incorrecta mentre treballava com Gateway per obtenir la resposta. | Quan no es pugui accedir al backend. |
| 503 | Service Unavailable | El servidor no està llest per processar la petició. | Quan passa un problema de rendiment o es troba en manteniment i el servidor no pot processar en aquell moment la resposta. |
| 504 | Gateway Timeout | La petició no pot ser processada en aquell moment. | Quan el backend no respon al servidor en el període de temps configurat. |

## 2.4 Filtratge i ordenació

El primer que s' ha de tenir en compte quan es vagi a considerar algun tipus de filtratge o ordenació és que la URL s' ha de mantenir el més petita i simple possible, evitant sobrecarregar-la amb nombrosos paràmetres i evitant arribar al **límit** màxim de caràcters, **2048**.

Quan es vulgui filtrar algun recurs en base a un o diversos paràmetres, si la petició compta amb menys de set paràmetres en total, s' ha d' establir un paràmetre que s' ha d' establir per cada paràmetre de filtratge necessari per a les peticions **GET** a aquest recurs. Si el **nombre de paràmetres totals** a poder usar **supera els set**, s' ha d' establir el filtratge com a camps del payload, convertint l' anomenada GET en una anomenada **POST**. Un exemple de filtratge sería el següent:

*HTTP GET https://preproduccio.ctti.apim.intranet.gencat.cat/ctti/privat-pre/gsit/api/modalitats/exportar?usuari=user1*

En el cas que es vulgui ordenar la resposta d'una API, es recomana l'ús d'un paràmetre que s'ha anomenat **ordre**, **orden** o **orderby**, en funció de l'idioma de l'API. El backend de l' API haurà de suportar la capacitat d' ordenació dels resultats de la petició. Un exemple d' ordenació de la resposta seria:

*HTTP GET https://preproduccio.ctti.apim.intranet.gencat.cat/ctti/privat-pre/gsit/api/modalitats/exportar?ordre=expiracio*

*HTTP GET https://preproduccio.ctti.apim.intranet.gencat.cat/ctti/privat-pre/gsit/api/modalitades/exportar?orden=expiracion*

*HTTP GET https://preproduccio.ctti.apim.intranet.gencat.cat/ctti/privat-pre/gsit/api/modalities/export?orderby=expiration*

## 2.5 Paginació

Si una API respon amb una llista d'entitats molt llarga, és recomanable establir algun tipus de paginació, per retornar només un subconjunt d'aquesta llista i evitar possibles sobrecàrregues en realitzar peticions que retornin respostes molt grans. Per establir la paginació, s'haurà de definir a l'API els següents paràmetres que hi ha opcionals:

- **offset**, que indicarà la pàgina (objecte de la llista) des d'on es començarà a comptar.
- **compte, cuenta o count,** en funció de l'idioma de l'API, que indicarà el nombre de pàgines (objectes de la llista) que s'han de retornar.
- **limit o limite,** en funció de l' idioma de l' API, que indicarà el nombre de paràmetres a retornar per cada pàgina.

La resposta hauria d' incloure com a metadades els enllaços següents:

- Enllaç a la pàgina anterior, si existeix
- Enlace a la página siguiente, si existe
- Enllaç a la primera pàgina, si hi ha
- Enllaç a l'última pàgina, si hi ha
- Enllaç a la pàgina actual
- Nombre total de pàgines

Per establir la paginació a l' API, el backend haurà de suportar aquesta paginació.

**Ejemplo:**

*HTTP GET https://preproduccio.ctti.apim.intranet.gencat.cat/ctti/privat-pre/ca/cercador?offset=1&compte=1&limit=10*

*{*

*"enllaços": {*

*"primer": "https://preproduccio.ctti.apim.intranet.gencat.cat/ctti/privat-pre/ca/cercador?offset=0”.*

*"anterior": "https://preproduccio.ctti.apim.intranet.gencat.cat/ctti/privat-pre/ca/cercador?offset=0&compte=1&limit=10",*

*"actual": "https://preproduccio.ctti.apim.intranet.gencat.cat/ctti/privat-pre/ca/cercador?offset=1”.*

*"següent": "https://preproduccio.ctti.apim.intranet.gencat.cat/ctti/privat-pre/ca/cercador?offset=2&compte=1&limit=10",*

*"últim": "https://preproduccio.ctti.apim.intranet.gencat.cat/ctti/privat-pre/ca/cercador?offset=199”.*

*"registresTotals": "200"*

*}*

*}*

## 2.6 Resposta parcial

En el cas que un client només necessiti un subconjunt de camps de la resposta general de l'API, més gran, i es vulgui evitar rebre tots els camps, s'hauria d'establir en la definició de l'API un paràmetre que s'ha anomenat **camps**, **campos** o **fideldset**, en funció de l'idioma de l'API, on s'indiquin els valors dels camps que el client vol rebre. El valor d' aquest paràmetre serà una llista de camps separats per comes, on el punt es farà servir per indicar que el camp és el subcamp d' un altre.

**Ejemplo:** *?camps=nom,descripcio,dades.direccion*

**NOTA:** perquè existeixi aquesta capacitat, el backend haurà de ser capaç de proveir aquestes respostes parcials.

## 2.7 Gestió d' errors

És molt important realitzar una gestió d'errors adequada perquè tant el client consumidor com el proveïdor de l'API puguin tenir clar què va anar malament amb la petició en concret, ajudant el client a aprendre com invocar correctament l'API i el proveïdor a detectar si el funcionament del seu API és l'adequat o existeix algun tipus de bug o, fins i tot, a detectar possibles intents d'accés maliciosos. És per això que s' han de seguir les bones pràctiques següents a l' hora de gestionar els errors:

- S' han d' utilitzar els **codis HTTP estàndard** i utilitzar-se de manera correcta, tal com s' especifica en el **punt 2.3**. Amb un ús adequat d' aquests codis, serà possible identificar si l' error correspon a un error de servidor o de client.
- S' ha de retornar un **missatge d' error clar** i que serveixi d' ajuda per identificar la causa i solució del problema.
- S' hauria de comptar amb una **estructura d' error estandarditzada** per gestionar la majoria dels errors d' una forma uniforme. Aquesta estructura hauria de comptar com a **mínim** amb un **codi d' error**, una **breu descripció de l' error** i **un detall més ampli i específic de l' error**.
- **No** s'haurien de retornar ni **stacktraces ni detalls** que puguin ser **confidencials o informació sensible** en el missatge d'error.
- En l'acoblament d'una API, hauria d'existir almenys un **default catch** que permeti gestionar tots els errors. Si no hi ha cap catch, no es farà una gestió d' errors, sinó que es retornaran els missatges per defecte.
- Si es vol realitzar una **gestió personalitzada** d'un tipus d'error en concret, s'hi haurà d'afegir el **catch** corresponent (*JavaScriptError*, *ConnectionError*, *RuntimeError*...)

## 2.8 Seguridad

La seguretat és un punt fonamental en qualsevol plataforma i tecnologia i sempre s'ha de buscar mitigar els riscos i vulnerabilitats associats buscant seguir els estàndards i bones pràctiques.

Atès que les APIs són normalment el component més exposat d'una xarxa i exposen lògica de negoci i fins i tot dades sensibles com **informació d'identificació personal (PII)**, s'han convertit en un objectiu d'atac. Els majors riscos de seguretat de les APIs es detallen en el projecte **OWASP API Security Project**, detallat concretament en https://owasp.org/www-project-api-security/

Entre elles s' inclouen **vulnerabilitats** per mala autenticació, autorització, absència de control del trànsit, exposició de dades innecessàries, possibilitat d' atacs d' injecció, etc.

És per això que totes les APIs haurien de seguir les següents consideracions i bones pràctiques de seguretat:

- Totes les APIs haurien de comptar amb mecanismes d' autenticació i autorització, essent la recomanació l' ús dels estàndards **OAuth2.0**, **OIDC** o **tokens JWT**.
- S'han de mantenir les APIs darrere d'un firewall, publicades en un API Gateway fent servir i requerint el protocol **HTTPS** per encriptar el trànsit, que permeti obtenir protecció bàsica a certs atacs com escaneig per a les amenaces basades en firma o atacs basats en injecció.
- La capçalera **HTTP Strict Transport Security** és un reforç de seguretat especificat per una aplicació web a un navegador. S'utilitza amb navegadors i el seu rebut provoca que un navegador previngui que existeixin comunicacions enviades al domini especificat usant HTTP, totes les comunicacions s'enviaran com a HTTPS al servidor web. Prevé diversos **atacs man-in-the-middle**. Com a bona pràctica, informar al navegador la capçalera Strict-Transport-Security és **més segur** que configurar una redirecció del trànsit HTTP a HTTPS en el servidor web al qual s'accedeix. Dins de la plataforma d' API Connect, s' ha generat una extensió post-response **Response-Headers-Validation**, la qual, si es configura en una API, permet gestionar les capçaleres de seguretat retornades per una API. El seu funcionament és tal que, si la capçalera Strict-Transport-Security no existeix, es generarà i retornarà al servidor web invocant.
- És bona pràctica establir polítiques de **throttling** que limitin el trànsit per l'API, impedint que els servidors de backend se sobrecarreguin i siguin fruit d'atacs de denegació de servei per part d'agents maliciosos.

- És bona pràctica que els equips proveïdors d'APIs realitzin un **assessment de riscos** de totes les seves APIs publicades. Haurien de poder identificar els sistemes i dades afectats si una API ha estat compromesa, i detallar els plans i controls requerits per reduir els riscos a un nivell acceptable.
- S'ha de buscar retornar en les respostes de les APIs només la **informació necessària**. Aquest enfocament s' hauria de tenir en compte a l' hora de dissenyar les APIs, atès que les respostes haurien de contenir la mínima informació necessària per complir amb la petició.
- És bona pràctica establir un **registre i documentació completa** de les APIs per poder ajudar a complir requeriments d' auditoria i compliance.
- És bona pràctica realitzar **revistes periòdiques de seguretat**.
- S' ha de realitzar un **monitoratge i auditoria** contínua de les peticions i la infraestructura de la plataforma, incloent en els missatges d' error el detall necessari, sempre evitant exposar informació sensible en aquests logs.
- Sempre que sigui possible, implementar **CORS**. Per a això, s'ha implementat una extensió anomenada **Cors-Validation**, amb la qual es configurarà CORS de forma automàtica per a qualsevol API.
- Sempre s' han de generar **tokens d' accés amb scopes** per limitar el poder d' accés d' un atacant maliciós en cas que pugui robar una credencial vàlida.
- És bona pràctica usar **algoritmes de signat asimètrics** per realitzar el signat dels tokens JWT i **evitar l'ús d'algoritmes simètrics**. Entre els algoritmes que es consideren recomanats, hi ha **RS256**, **PS256**, **ES256** o **EdDSA**.

## 2.9 Versionat

El versionat és un aspecte important en el disseny d'APIs. Cada API ha de tenir una versió associada i és important realitzar una bona estratègia de versionat de les APIs quan els desenvolupaments van evolucionant en el temps. Les recomanacions a l'hora de dissenyar aquesta estratègia són:

- Es realitzarà el model de "Semantic versioning" que segueix el patró Major.Minor.Patch
    - **Patch.** Es refereix principalment a la correcció d' errors compatibles amb versions anteriors.
    - **Minor.** Es modifica quan apareixen noves funcionalitats compatibles amb versions anteriors.
    - **Major.** Indica que l'API ha patit un gran canvi i que, a més, aquest canvi pot ocasionar errors per a la gent que la utilitza ja que és incompatible amb versions anteriors.
- Tot canvi realitzat en qualsevol API que ja estigui publicat en producció comportarà un canvi de versió incrementant un dels tres indicadors anteriors.
- El llistat de **canvis compatibles** és:
    - Canvi de l' **obligatorietat** d' un camp d' **obligatori** a **opcional**.
    - Addició d' un camp d' **entrada opcional**.
    - Addició d' un camp de **sortida** que no canviï el comportament d' altres camps.
    - **Relaxació** de les **limitacions** d' un camp.
        - Longitud **màxima** del camp o arrelament **augmenta**.
        - Longitud **mínima** del camp o arrelament **disminueix**.
    - Addició d' un **nou codi d' error**.
    - Addició de nova funcionalitat: **nous recursos o mètodes** a recursos existents.
    - Canvi de la **nomenclatura** d' un camp **opcional**.
- El llistat de **canvis incompatibles** és:
    - Canvi de l' **obligatorietat** d' un camp d' **opcional** a **obligatori**.
    - Addició d' un camp d' **entrada obligatori**.
    - **Major** limitació del tipus de dada del camp:
        - Longitud **màxima** del camp o arrelament **disminueix**.
        - Longitud **mínima** del camp o arrelament **augmenta**.
    - Canvi del **tipus de dada** d' un camp.
    - Canvi de la **nomenclatura** o **mètodes** d' un recurs.
    - Canvi en el **format** de la URL.
    - Canvi de la **nomenclatura** o **eliminació** d' un camp .
- Quan es realitzi un versionat en una API pertanyent a un producte, la versió del producte haurà de canviar també.
    - Un canvi de tipus **Patch** en una de les APIs del producte comportarà un canvi de tipus **Patch** en el producte/productes que la conté. Si en el mateix canvi de producte es corregeixen diversos

errors en més d' una API, només es pujarà una versió del producte, ja que els ajustos s' han realitzat en el mateix procés d' actualització del producte.

- Un canvi de tipus **Minor** en una de les APIs del producte comportarà un canvi de tipus **Minor** en el producte/productes que la conté. Si en el mateix canvi de producte es corregeixen en més d'una API canvis de tipus Minor, només es pujarà una versió al producte ja que els canvis s'han realitzat dins de la mateixa pujada del producte. També es considera un canvi de tipus Minor la inclusió d' una nova API en el Producte. En aquest cas l'API seria 1.0.0 i el Producte passaria a ser 1.1.0.
- Un canvi de tipus **Major** en una de les APIs del producte comportarà un canvi de tipus **Major** en el producte/productes que la conté. Si en el mateix canvi de producte es corregeixen en més d'una API canvis de tipus Major, només es pujarà una versió al producte ja que els canvis s'han realitzat dins de la mateixa pujada del producte.

- No és recomanable que hi hagi **més de dues** versions majors d'APIs i productes coexistint. La versió antiga haurà de coexistir fins que tots els consumidors actuals hagin migrat la seva subscripció a la nova.
- Atès que no s' inclourà la versió de les APIs en l' endpoint, perquè un usuari invoqui a la nova versió, s' haurà de migrar la seva subscripció de la versió antiga a la nova versió. Quan la subscripció hagi estat migrada correctament i invoqui el mateix endpoint al qual invocava prèviament, farà ús de la nova versió en lloc de l' antiga. Aquesta migració es pot realitzar de diferents formes, en funció del tipus de canvi realitzat en el producte i del cas d' ús:
  - Canvis de tipus **Patch**. Atès que els canvis Patch són canvis **retrocompatibles** per solucionar errors de les versions prèvies, és bona pràctica migrar totes les subscripcions del producte antic a la nova versió, retirant el producte anterior. Això s'aconsegueix mitjançant l'opció de **Replace** de la plataforma.
    - L' opció de **Replace** realitza els següents passos de forma automàtica:
      - El producte nou (1.0.1) assumeix l'estat del producte antic (1.0.0).
      - Les propietats de visibilitat i subscripció del producte antic (1.0.0) passen al producte nou (1.0.1).
      - Els subscriptors del producte antic (1.0.0) es migren al producte nou (1.0.1). **Nota:** No es pot migrar automàticament d' un pla gratuït a un de pagament.
      - El producte antic (1.0.0) es queda en estat retirat.
    - Per realitzar el **Replace**, s' han de realitzar els passos següents:
      - Primer, s'ha de publicar el nou producte amb el canvi Patch (exemple: versió 1.0.1) al catàleg i espai corresponent. Això s'aconseguirà usant el pipeline de **PUBLISH** generada pel SIC
      - Després, s' ha de llançar el pipeline de **REPLACE**, indicant en la pestanya input de la consola els paràmetres necessaris, els quals són:
        - ***CURRENT_PRODUCT_VERSION***: Aquí s'indica la versió antiga del producte (exemple: versió 1.0.0).
        - ***NEW_PRODUCT_VERSION***: Aquí s'indica la nova versió del producte (exemple: versió 1.0.1).
      - En cas que es realitzi un desplegament manual, l' opció de Replace apareix en els productes desplegats en un catàleg.
  - Canvis de tipus **Minor**. Tot i que els canvis de tipus Minor són **retrocompatibles** com els Patch, solen ser per afegir noves funcionalitats. És per això que aquí hi ha dues possibilitats vàlides: fer ús del **Replace** com en els tipus Patch, o usar la funcionalitat de **Supersede**.
    Per decidir quina de les dues usar, s' ha de considerar si es vol actualitzar els subscriptors de forma automàtica a aquesta nova versió amb noves funcionalitats o si s' aniran actualitzant ells de forma progressiva i manual quan decideixin implementar aquestes noves funcionalitats.
    - Si es decideix usar l' estratègia d' actualitzar **automàticament** aquests subscriptors, aleshores es farà ús de l' opció **Replace**, els passos de la qual són els mateixos que per als canvis de tipus Patch.
    - Si es decideix usar l' estratègia d' actualització manual **progressiva**, s' ha de fer servir l' opció de **Supersede**, que realitza els següents passos de forma automàtica:
      - Les propietats de visibilitat i subscripció del producte antic (1.0.0) passen al producte nou (1.1.0)
      - El producte antic (1.0.0) es depreca, **no** es podran realitzar noves subscripcions a aquest producte.

- Els subscriptors del producte antic (1.0.0) reben al Developer Portal un missatge amb l'opció de migració de subscripció. En polsar-hi, es migrarà la seva subscripció del producte antic (1.0.0) al nou (1.1.0).
- Un cop el producte antic (1.0.0) no compti amb subscripcions, s'haurà de retirar.

▪ Per realitzar el **Supersede**, s' han de fer els passos següents:

- Primer, s'ha de publicar el nou producte amb el canvi Minor (exemple: versió 1.1.0) en el catàleg i espai corresponent. Això s' aconseguirà usant el pipeline de **PUBLISH** generada pel SIC.
- Després, s' ha de llançar el pipeline de **SUPERSEDE**, indicant en la pestanya input de la consola els paràmetres necessaris, els quals són:
  - ***CURRENT_PRODUCT_VERSION***: Aquí s'indica la versió antiga del producte (exemple: versió 1.0.0).
  - ***NEW_PRODUCT_VERSION***: Aquí s'indica la nova versió del producte (exemple: versió 1.1.0).
- En cas que es realcés un desplegament manual, l' opció de Supersede apareix en els productes desplegats en un catàleg.

o Canvis de tipus **Major**. Els canvis de tipus Major **no són retrocompatibles**, amb la qual cosa hi ha dues possibilitats vàlides de migració: fer ús de Supersede o fer una publicació normal del producte.

Per decidir quina de les dues usar, s' ha de considerar si es vol permetre a nous consumidors subscriure' s a la versió antiga del producte o no. Això pot ocórrer si, per exemple, les APIs de la versió nova fan ús d'altres backends (**Back2**), perquè CTTI ha decidit canviar de tecnologia o versió els backends antics (**Back1**), i aquests **Back2** no tenen encara implementada tota la funcionalitat dels **Back1** o van a exposar funcionalitat diferent, amb la qual cosa, en funció del cas d'ús del consumidor, voldrà subscriure's a una o altra versió.

- Si es decideix **no permetre** als nous consumidors subscriure's a la versió antiga (1.0.0), s'ha de fer servir l'opció de **Supersede**, que deprecarà aquesta versió antiga i generarà el missatge de la migració de subscripció al portal a cada consumidor.
- Si es **permet** als nous consumidors subscriure's a la versió antiga (1.0.0), s'ha de simplement **publicar** el producte amb una nova versió (2.0.0). Cada consumidor haurà, quan decideixi canviar de versió, eliminar de forma manual la seva subscripció a la versió antiga (1.0.0) i subscriure's a la nova (2.0.0). No ha d' estar subscrit a ambdues alhora amb una mateixa aplicació.

## 2.10 Documentació

És bona pràctica en el desenvolupament d' APIs la generació d' una documentació extensa de les APIs per tal de facilitar als desenvolupadors i consumidors el testeig i integració amb aquestes APIs. Aquesta documentació es troba normalment en els portals de desenvolupament, els quals la generaran de manera automàtica en base a la definició de l' esmentada API, que haurà de ser un fitxer YAML en el format **OpenAPI**. És per això que aquesta definició hauria de ser emplenada amb la màxima informació possible.

Una bona definició d' una API REST en IBM API Connect haurà de comptar amb les seccions següents:

- **Informació general.** Aquesta secció inclou dades d' informació general sobre l' API. És bona pràctica incloure almenys el **nom**, **versió**, **descripció**, **urls de servei/basePath** i **dades de contacte**. Al seu torn, en aquesta secció es podrà incloure els **tags** de l' API si se' n fan ús o enllaços a documentació externa relacionada i llicències d' ús.
  - A **OpenAPI 2.0/Swagger**, aquesta secció estarà formada per les etiquetes **info**, **host**, **basePath**, **schemes** i **tags**.
  - A **OpenAPI 3.0**, aquesta secció estarà formada per les etiquetes **info**, **externalDocs**, **servers** i **tags**.
- **Paths.** En aquesta secció s' especifiquen tots els recursos i mètodes d' accés de l' API. És bona pràctica incloure per a cada operació un **resum**, **una descripció, un esquema complet de tots els paràmetres d' entrada i totes les possibles respostes** amb els seus respectius esquemes complets en funció del codi HTTP que es podrà retornar. Alhora, és recomanable establir un exemple per a cada entrada i sortida.
  - A **OpenAPI 2.0/Swagger**, aquesta secció estarà formada per l' etiqueta **paths**.

  - A **OpenAPI 3.0**, aquesta secció estarà formada per l' etiqueta **paths**.
- **Definicions.** És bona pràctica definir en aquesta secció **esquemes** que hagin de ser reutilitzats, podent ser referenciats en la resta de la definició, per tal d' evitar duplicació de codi i reduir el temps d' escriptura. Aquests esquemes podran ser de paràmetres de la petició d' entrada, de la petició de sortida i de definicions de seguretat.
  - A **OpenAPI 2.0/Swagger**, aquesta secció estarà formada per les etiquetes **definitions** i **securityDefinitions**.
  - A **OpenAPI 3.0**, aquesta secció estarà formada per l' etiqueta **components**.
- **Seccions pròpies d' IBM API Connect.** IBM API Connect compta amb extensions sobre el format OpenAPI on es podrà definir la configuració necessària d'API Connect, com les polítiques de l'acoblament.