NORME ST.90 DE L'OMPI

RECOMMANDATIONS RELATIVES AU TRAITEMENT ET À LA COMMUNICATION DES DONNÉES DE PROPRIÉTÉ INTELLECTUELLE AUX API WEB (INTERFACES DE PROGRAMMATION D'APPLICATION)

Version 1.1

Révision approuvée par le Comité des normes de l'OMPI (CWS) à sa dixième session le 25 novembre 2022

TABLE DES MATIÈRES

ANNEXES

NORME ST.90 DE L'OMPI

RECOMMANDATIONS RELATIVES AU TRAITEMENT ET À LA COMMUNICATION DES DONNÉES DE PROPRIÉTÉ INTELLECTUELLE AUX API WEB (INTERFACES DE PROGRAMMATION D'APPLICATION)

Version 1.1

Révision approuvée par le Comité des normes de l'OMPI
à sa dixième session le 25 novembre 2022

INTRODUCTION

1. La présente norme contient des recommandations concernant des interfaces de programmation d'applications (API) afin de faciliter le traitement et l'échange harmonisé de données de propriété intellectuelle sur le Web.

2. La présente norme a pour but :

- d'assurer l'homogénéité en établissant des principes uniformes pour la création de services Web;
- d'améliorer l'interopérabilité des données entre partenaires de services Web;
- d'encourager la réutilisation grâce à un format unifié;
- de promouvoir la flexibilité en matière de nommage de données dans les unités opérationnelles grâce à une politique d'espace de nommage clairement définie dans les ressources XML associées;
- de promouvoir l'échange sécurisé des informations;
- de proposer des procédures opérationnelles internes pertinentes comme services à valeur ajoutée pouvant être utilisés par d'autres organisations; et
- d'intégrer ses procédures opérationnelles internes et de les relier de manière dynamique aux partenaires.

DÉFINITIONS ET TERMINOLOGIE

3. Aux fins de la présente norme, les expressions :

- "HyperText Transfer Protocol (HTTP)" (protocole de transfert hypertexte) est le protocole de niveau application pour les systèmes d'information distribués, collaboratifs et hypermédia. Le HTTP est la pierre angulaire de la communication des données pour le World Wide Web (Toile mondiale). Le HTTP est un protocole de demande/réponse fonctionnant selon le modèle de calcul de type client-serveur;
- "interfaces de programmation d'applications" (API) désigne les éléments de logiciel qui fournissent une interface réutilisable entre différentes applications pouvant aisément interagir pour échanger des données;
- "Representational State Transfer (transfert d'état représentationnel) (REST)" désigne un ensemble de principes architecturaux selon lesquels des données peuvent être transmises sur une interface normalisée, c'est-à-dire le HTTP. REST ne contient pas de couche supplémentaire pour l'échange de messages et est axé sur les règles de conception applicables à la création de services sans états;
- "Simple Object Access Protocol (SOAP) (protocole d'accès aux objets simples)" désigne un protocole d'envoi et de réception de messages entre applications sans que se posent des problèmes d'interopérabilité. SOAP définit une spécification de protocole (ensemble de règles) de communication normalisé pour l'échange de messages en XML. SOAP utilise différents protocoles de transfert, comme le HTTP et le SMTP. Le protocole normalisé HTTP permet au modèle SOAP de franchir plus facilement les serveurs pare-feu et les serveurs proxy (mandataires) sans modification du protocole SOAP;
- "service Web" désigne une méthode de communication entre deux applications ou machines électroniques sur la Toile mondiale (WWW); les services Web sont de deux types : REST et SOAP;
- "API Web RESTful" désigne un ensemble de services Web fondés sur le paradigme architectural REST, avec utilisation en général du format JSON ou XML pour la transmission des données;
- "API Web SOAP" désigne un ensemble de services Web SOAP fondés sur SOAP, avec obligation d'utiliser le format de charge utile XML;
- "Web Services Description Language" (langage de description de services Web) (WSDL) désigne une norme du W3C qui est utilisée avec le protocole SOAP pour fournir une description d'un service Web. Elle comprend les méthodes utilisées par un service Web, ses paramètres et les moyens de le localiser, etc.;
- "RESTful API Modelling Language" (RAML) désigne un langage qui permet aux développeurs de fournir une spécification de leur API;

- "Open API Specification" (OAS) désigne un langage qui permet aux développeurs de fournir une spécification de leur API;
- "contrat de service" (ou contrat de service Web) désigne un document qui présente les fonctions et ressources que le service peut offrir à d'autres logiciels sous la forme d'une API publiée; le terme "documentation API REST" et celui de contrat de service sont utilisés de manière interchangeable pour les API Web s RESTful;
- "prestataire de services" désigne un logiciel de services qui propose un service Web;
- "consommateur de services" désigne le rôle d'exécution joué par un logiciel lorsqu'il accède à un service et le sollicite. C'est en particulier le cas lorsque le logiciel envoie un message à une capacité de service prévue au contrat de service. À la réception de la demande, le service commence le traitement et peut renvoyer ou non au consommateur de services un message de réponse correspondant;
- "camelcase" désigne la convention de nommage caractères bas de casse de type "camel" (par exemple applicantName) ou caractères haut de casse de type "camel" (par exemple ApplicantName);
- la police de caractères kebab est l'une des conventions de nommage constituées uniquement de caractères bas de casse avec des traits d'union "-" séparant les mots, par exemple a-b-c;
- "normes ouvertes" désigne les normes mises à la disposition du grand public et mises au point (ou approuvées) et dont l'application se poursuit dans le cadre d'un processus collaboratif et consensuel. Les "normes ouvertes" facilitent l'interopérabilité et l'échange de données entre différents produits de services et sont destinées à être largement adoptées;
- l'identifiant uniforme de ressources (URI) identifie une ressource, et le localisateur de ressources uniforme (URL) est un sous-ensemble des URI qui comprennent un emplacement réseau;
- "étiquette d'entité (ETag)" désigne un identificateur opaque assigné par un serveur Web à une variante spécifique d'une ressource trouvée à une adresse URL. Si la représentation associée à cette ressource vient à changer, une nouvelle ETag différente est assignée. On peut comparer rapidement les ETags pour déterminer si deux représentations d'une ressource sont identiques;
- "registre de services" désigne un répertoire en réseau qui contient les services disponibles;
- "RMM" s'entend du modèle de maturité de Richardson, qui mesure le degré de maturité de l'API REST à l'aide d'une échelle allant de 0 à 3; et
- "versionnage sémantique" désigne un système de versionnage dans lequel une version est identifiée par le numéro de version MAJOR.MINOR.PATCH, où :

    - version MAJEURE (MAJOR version) lorsque vous apportez des changements API incompatibles,
    - version MINEURE (MINOR version) lorsque vous ajoutez une fonctionnalité d'une manière rétrocompatible, et
    - version CORRECTIVE (PATCH version) en cas de correction de bogues rétrocompatible.

4. Du point de vue de la compatibilité des règles de conception, les mots-clés ci-après doivent être interprétés de la même manière que celle qui est définie au paragraphe 8 de la norme ST.96 de l'OMPI[1], c'est-à-dire comme suit :

- DOIT ou DOIVENT : ce mot, ou le mot "REQUIS" ou "DEVRA ou DEVRONT", signifie que la définition est une condition absolue de la spécification;
- NE DOIT PAS ou NE DOIVENT PAS : ces mots, ou les mots "NE DEVRA PAS ou NE DEVRONT PAS", signifient que la définition est une interdiction absolue de la spécification;
- DEVRAIT ou DEVRAIENT : ce mot, ou l'adjectif "RECOMMANDÉ(S)", signifie qu'il peut y avoir des raisons valables pour ignorer un terme particulier mais que toutes les conséquences doivent être soigneusement évaluées avant de choisir une trajectoire différente;
- NE DEVRAIT PAS ou NE DEVRAIENT PAS : ces termes, ou les termes "NON RECOMMANDÉ(S)", signifient qu'il peut y avoir des raisons valables pour que le comportement particulier soit acceptable ou même utile mais que les conséquences doivent être soigneusement évaluées avant d'appliquer un comportement assorti de cette étiquette; et
- PEUT ou PEUVENT : ce mot, ou l'adjectif "FACULTATIF(S)", signifie qu'un élément est véritablement facultatif et ne constitue qu'une option parmi beaucoup d'autres.

[1] Voir le chapitre consacré aux références.

NOTATIONS

Notations générales

5. Les notations ci-après sont utilisées d'un bout à l'autre du présent document :

- <> : indique un terme utilisé pour décrire un espace qualifié qui, dans la mise en œuvre, sera remplacé par une valeur d'instance spécifique;
- "" : indique que le texte entre guillemets doit être utilisé in extenso dans la mise en œuvre;
- { } : indique que la mise en œuvre est facultative; et
- `Courier New font` : indique les mots-clés ou le code source.

6. Les adresses URL fournies dans la présente norme ne le sont qu'à titre d'exemples et ne sont pas opérationnelles.

Identificateurs de règle

7. Toutes les règles de conception sont normatives. Les règles de conception sont identifiées au moyen d'un préfixe de [XX-nn] ou [XXY-nn].

a) La valeur "XX" est un préfixe qui sert à classer les types de règles comme suit :

- WS pour les règles de conception des API Web SOAP;
- RS pour les règles de conception des API Web RESTful; et
- CS pour les règles de conception des API Web SOAP et RESTful.

b) La valeur "Y" n'est utilisée que pour les règles de conception RESTful et renforce la précision du type de réponse auquel la règle se rapporte :

- "G" définit une règle générale pour les réponses JSON et XML;
- "J" définit indique une règle générale pour la réponse JSON; et
- "X" définit une réponse XML.

c) La valeur "nn" définit le prochain numéro disponible dans l'ordre d'un type de règle donné. Le numéro ne désigne pas la position de la règle, en particulier pour une nouvelle règle. Une nouvelle règle sera placée dans le contexte pertinent. Par exemple, l'identificateur [WS-4] identifie la quatrième règle de conception des API Web SOAP. La règle [WS-4] peut être placée entre les règles [WS-10] et [WS-11] au lieu de suivre [WS-3] s'il s'agit de la position la plus appropriée pour cette règle.

d) L'identificateur de la règle supprimée sera conservé tandis que la règle sera remplacée par le terme "Supprimé".

CHAMP D'APPLICATION

8. La présente norme vise à donner des orientations aux offices de propriété intellectuelle et aux autres organisations devant gérer, stocker, traiter, échanger et diffuser des données de propriété intellectuelle en utilisant les API Web. Grâce à l'utilisation de cette norme, la mise en œuvre des API Web pourrait être simplifiée et accélérée et s'effectuer de manière harmonisée et l'interopérabilité entre les API Web pourrait être améliorée.

9. La norme vise à s'appliquer aux communications entre offices de propriété intellectuelle et déposants ou utilisateurs de données et entre offices de propriété intellectuelle grâce à des connexions entre applications "devices-to-devices" et applications "devices-to-software".

Fig. 1 Champ d'application de la norme

10. Cette norme a pour objectif de fournir un ensemble de règles et conventions de conception pour les API Web RESTful et les API Web SOAP; la liste des données de propriété intellectuelle qui seront échangées ou exposées, et une documentation ou contrat de service API type, qui peut être utilisé à des fins de personnalisation, en décrivant le format de message, la structure de données et le dictionnaire de données au format JSON sur la base de la norme ST.97 de l'OMPI ou XML sur la base de la norme ST.96 de l'OMPI.

11. La présente norme fournit des contrats de service types pour des API Web SOAP utilisant le langage WSDL et pour des API Web RESTful utilisant le langage de modélisation API RESTful (RAML) et la spécification d'API ouverte (OAS). De plus, un contrat de service définit ou mentionne des types de données pour les interfaces (voir la section "convention sur les types de données" plus loin). La présente norme recommande trois types d'interfaces : REST-XML (XSD), REST-JSON et SOAP-XML (XSD).

12. Cette norme exclut ce qui suit :

a) La liaison avec des piles de technologies de mise en œuvre et des produits commerciaux standard spécifiques;
b) La liaison avec des conceptions architecturales spécifiques (par exemple, l'architecture orientée services (AOS) ou l'architecture orientée microservices (AOM));
c) La liaison avec des algorithmes spécifiques, tels que les algorithmes de calcul de l'ETag, c'est-à-dire le calcul d'un identificateur unique d'une version spécifique d'une ressource (utilisée, par exemple, pour la mise en antémémoire).

PRINCIPES DE CONCEPTION DES API WEB

13. Les API Web RESTful et les API Web SOAP se sont avérés capables de répondre aux exigences des grandes organisations et d'assurer le service des petites applications intégrées en production. Au moment de choisir entre RESTful et SOAP, on pourra prendre en considération les aspects ci-après :

- Sécurité, par ex, SOAP comporte la WS-Security, tandis que REST n'indique aucun mécanisme de sécurité;
- Transaction ACID, p. ex., SOAP comporte une spécification WS-AT, tandis que REST n'a pas de spécification pertinente;
- Style architectural, p. ex., le style architectural axé sur les microservices et sans serveur utilise REST tandis que l'architecture orientée services (AOS) utilise les services Web SOAP;
- Flexibilité;
- Contraintes en matière de largeur de bande; et
- Remise garantie, p. ex. SOAP propose WS-RM tandis que REST n'a pas de spécification pertinente.

14. La conception d'une API doit respecter les principes orientés services ci-après :

a) Contrat de service normalisé : la normalisation des contrats de service est le plus important principe de conception car les contrats permettent la gouvernance et une conception de services uniforme. Un contrat de service doit être facile à comprendre et à exécuter. Il est constitué de métadonnées qui décrivent comment le prestataire et le consommateur de services interagiront. Les métadonnées décrivent également les conditions dans lesquelles ces parties sont habilitées à entrer en relations. Il est recommandé de faire figurer dans les contrats de service :

   - Obligations fonctionnelles : la fonctionnalité fournie par le service et les données qu'il remettra, ou, le plus souvent, une combinaison des deux;
   - Obligations non fonctionnelles : des informations sur la responsabilité des prestataires en matière de fourniture de leur fonctionnalité ou de leurs données, ainsi que les responsabilités prévues des consommateurs de ces informations et ce qu'ils devront fournir en échange. Par exemple, la disponibilité d'un consommateur, la sécurité et d'autres considérations liées à la qualité de service.

b) Couplage lâche de service : les clients et les services devraient évoluer indépendamment. L'application de ce principe de conception exige :

   - le versionnage des services – les consommateurs liés à une API Web ne devraient pas prendre le risque de rupture imprévue due à des changements d'API incompatibles; et
   - le contrat de service devrait être indépendant des informations technologiques.

c) Abstraction du service – Les détails d'implémentation du service devraient être masqués. La conception de l'API devrait être indépendante des stratégies reposant sur un serveur. Par exemple, pour le service Web REST, le modèle de ressource d'API devrait être découplé du modèle d'entité dans la couche rémanente;
d) Service sans état – les services devraient être évolutifs;

e) Réusabilité du service – Une API bien conçue devrait fournir des services réutilisables avec des contrats généraux. À cet égard, la présente norme fournit un contrat de service type;
f) Autonomie du service – Les limites fonctionnelles du service devraient être clairement définies;
g) Découvrabilité du service – Les services devraient être découverts et interprétés de façon effective;
h) Composabilité – les services peuvent être utilisés pour composer d'autres services;
i) Utilisation de normes comme base – L'API devrait suivre les normes de l'industrie (comme IETF (Internet Engineering Task Force), ISO et OASIS) chaque fois que cela est possible, en les préférant naturellement aux solutions optimisées localement; et
j) Principe de sélection – Il n'est pas exigé de mettre en œuvre toutes les règles de conception des API. Ces règles devraient être choisies en fonction de la mise en œuvre de chaque cas concret.

15. De plus, les principes ci-après devraient être respectés, notamment en ce qui concerne les API Web RESTful :

a) Antémémorisabilité : les réponses indiquent explicitement leur antémémorisabilité;
b) Identification des ressources dans les demandes : chacune des ressources est identifiée dans les demandes; par exemple, utilisation des URI dans les systèmes Web REST. Les ressources proprement dites sont conceptuellement distinctes des représentations qui sont remises au client;
c) Hypermedia as the engine of application state (hypermédia en tant que moteur de l'état d'application) (HATEOAS); après avoir accédé à un URI initial pour l'application REST – comme dans le cas d'un individu accédant à la page d'accueil d'un site Web –, un client REST devrait pouvoir utiliser de façon dynamique des liens fournis par le serveur pour découvrir toutes les actions disponibles et toutes les ressources dont il a besoin;
d) Manipulation des ressources par l'intermédiaire de leurs représentations – lorsqu'un client détient une représentation d'une ressource, notamment toutes métadonnées jointes, il dispose de suffisamment d'informations pour modifier ou supprimer cette ressource;
e) Messages autodescriptifs – chaque message comprend suffisamment de métadonnées pour décrire comment traiter le contenu du message;
f) Une API Web devrait respecter la sémantique du HTTP, comme les méthodes, les erreurs, etc.;
g) Disponibilité pour le public – concevoir avec l'objectif de rendre le moment venu l'API accessible depuis l'Internet public, même si aucun plan n'a encore été établi à cette fin;
h) Authentification commune – utiliser un mode d'authentification et d'autorisation commun, de préférence basé sur les éléments de sécurité existants, afin de ne pas créer de solution personnalisée pour chaque API;
i) Moindre privilège – l'accès et l'autorisation doivent être accordés aux consommateurs d'API dans la limite de ce dont ils ont besoin pour remplir les fonctions requises;
j) Maximisation de l'entropie – le caractère aléatoire des authentifiants de sécurité doit, pour l'autorisation des API, être maximisé en utilisant les clés API plutôt qu'un nom d'utilisateur et un mot de passe, car ces clés fournissent une surface d'attaque qui résistent mieux aux éventuels pirates; et
k) Performance et sécurité – trouver un compromis entre la performance et la sécurité en ce qui concerne les principales durées de vie et les principaux coûts indirects des services de chiffrement/déchiffrement.

API WEB RESTFUL

16. Une API Web RESTful permet aux systèmes demandeurs d'accéder à des représentations textuelles de ressources Web et de les manipuler à l'aide d'un ensemble uniforme et prédéfini d'opérations sans état.

Éléments de l'URI

17. Les API Web RESTful utilisent les URI pour identifier des ressources. Selon le RFC 3986, la syntaxe d'un URI devrait être définie comme suit :

```
URI = <mécanisme> " ://" <autorité> "/" <chemin d'accès> {"?" requête}

autorité = {userinfo@}hôte{ :port}
```

Par exemple, https://wipo.int/api/v1/patents?sort=id&offset=10

```
          _____/_____/__________/________________/
            |     |        |         |
paramètres : mécanisme autorité chemin d'accès requête
```

18. Le caractère "/" est utilisé dans le chemin d'accès de l'URI pour définir un rapport hiérarchique entre les ressources, mais le chemin ne doit pas s'achever sur un trait oblique car il ne fournit aucune valeur sémantique et peut être déroutant.

[RSG-01] Le caractère "/" DOIT être utilisé dans le chemin d'accès de l'URI pour définir un rapport hiérarchique entre les ressources, mais le chemin NE DOIT PAS s'achever sur un trait oblique.

19. Les URI distinguent les majuscules et les minuscules, sauf pour les parties mécanisme et hôte. Par exemple, bien que `https://wipo.int/api/my-resources/uniqueId` et `https://wipo.INT/api/my-resources/uniqueId` soient identiques, ce n'est pas le cas de `https://wipo.int/api/my-resources/uniqueid`. Pour les noms de ressources, les conventions des polices de caractères kebab et de caractères bas de casse de type "camel" présentent une bonne lisibilité et établissent la correspondance entre les noms de ressources et les entités dans les langages de programmation avec une simple transformation. Pour les paramètres relatifs aux requêtes, les caractères bas de casse de type "camel" devraient être utilisés. Par exemple, `https://wipo.int/api/v1/inventors?firstName=John`. Les noms de ressources et le paramètre de la requête distinguent tous les majuscules et les minuscules. On notera que les noms de ressources et les noms de paramètres de requête peuvent être abrégés.

20. Une API Web RESTful peut avoir des mots arguments :

- dans le paramètre de requête; par exemple, /inventeurs?id=1;
- dans le paramètre d'un segment de chemin d'accès de l'URI, par exemple, /inventeurs/1; et
- dans la charge utile de la requête, comme une partie du corps de la requête au format JSON.

21. Mis à part le cas des types d'arguments susmentionnés, qui font partie de l'URI, un argument peut également faire partie de la charge utile de la requête.

[RSG-02] Les noms des ressources DOIVENT avoir une structure de nommage uniforme.

[RSG-03] Les noms des ressources dans la requête DEVRAIENT utiliser les conventions de nommage des polices de caractères kebab et PEUVENT être abrégés.

[RSG-04] Les paramètres de requête DOIVENT avoir une structure de nommage uniforme.

[RSG-05] Les paramètres de requête DEVRAIENT utiliser la convention des caractères bas de casse de type "camel" et PEUVENT être abrégés.

22. Le point de terminaison d'une API Web doit être conforme au `IETF RFC 3986` (RFC (appel à observations) 3986 de l'IETF) et devrait éviter d'éventuelles collisions avec les URL de page pour le site Web hébergé sur le domaine racine. Une API Web doit avoir un point d'entrée exact pour regrouper toutes les requêtes. En règle générale, il existe deux modes de définition des points de terminaison :

- en tant que premier segment du chemin d'accès de l'URI, par exemple : `https://wipo.int/api/v1/`; et
- en tant que sous-domaine, par exemple : `https://api.wipo.int/v1/`

[RSG-06] La structure d'URL d'une API Web DOIT contenir le mot "`api`" dans l'URI.

23. Les paramètres de matrice indiquent la complexité d'une API comprenant de multiples niveaux de ressources et de sous-ressources. Cela va à l'encontre des principes de conception orientée services définis précédemment. De plus, les paramètres de matrice ne sont pas normalisés car ils s'appliquent à un certain élément du chemin d'accès, alors que les paramètres de requête s'appliquent à l'ensemble de la requête. On trouvera ci-après un exemple de paramètre de matrice : `https://api.wipo.int/v1/path;param1=value1;param2=value2`.

[RSG-07] Les paramètres de matrice NE DOIVENT PAS être utilisés.

Codes d'état

24. Une API Web doit appliquer systématiquement les codes d'état HTTP tels que définis dans les RFC de l'IETF. Les codes d'état HTTP devraient être utilisés parmi ceux de la liste des codes d'état HTTP normalisés (RFC 7807) reproduits à l'annexe V.

[RSG-08] Une API Web DOIT appliquer systématiquement les codes d'état HTTP tels que définis dans les RFC de l'IETF.

[RSG-09] Les codes recommandés à l'annexe V DEVRAIENT être utilisés par une API Web pour classer les erreurs.

Principe de sélection

25. Un contrat de service doit être tolérant à l'égard de paramètres imprévus (dans la requête, en utilisant les paramètres de requête), mais doit mentionner une erreur en cas de valeurs mal formées concernant les paramètres prévus.

[RSG-10] Si l'API détecte des valeurs d'entrée invalides, elle DOIT renvoyer le code d'état HTTP "`400 Bad Request`". La charge utile de l'erreur DOIT indiquer la valeur erronée.

[RSG-11] Si l'API détecte des noms d'arguments syntaxiquement corrects (dans les paramètres de requête) qui ne sont pas prévus, elle DEVRAIT ne pas en tenir compte.

[RSG-12] Si l'API détecte des valeurs valides qui exigent que des fonctionnalités ne soient pas exécutées, elle DOIT renvoyer le code d'état HTTP "`501 Not Implemented`". La charge utile de l'erreur DOIT indiquer la valeur non traitée.

Modèle de ressources

26. Un modèle de données de propriété intellectuelle devrait être divisé en contextes délimités selon une conception pilotée par le domaine. Chaque contexte délimité doit être associé à une ressource. Selon les principes régissant la conception, un modèle de ressources d'une API Web devrait être découplé du modèle de données. Une API Web devrait être modélisée en tant que hiérarchie de ressources afin de tirer parti du caractère hiérarchique de l'URI pour suggérer une structure (association, composition ou agrégation), où chaque nœud est une ressource simple (unique) ou une collection de ressources.

27. Dans ce modèle de ressources hiérarchisé, les nœuds racines sont appelés "nœuds de premier niveau" et toutes les ressources imbriquées "sous-ressources". Les sous-ressources ne doivent servir qu'à suggérer des compositions, c'est-à-dire des ressources qui ne peuvent pas être des ressources de premier niveau; autrement, il existerait de multiples façons d'extraire les mêmes entités. Ces sous-ressources, qui impliquent une association, sont appelées sous-collections. Les autres structures hiérarchiques, à savoir l'association et l'agrégation, devraient être évitées pour ne pas aboutir à des API complexes et à des doublonnements de fonctionnalités.

28. Le point de terminaison détermine toujours le type de réponse. Par exemple, le point de terminaison `https://wipo.int/api/v1/patents` donne toujours des réponses qui concernent des ressources en matière de brevets. Le point de terminaison `https://wipo.int/api/v1/patents/1/inventor` donne toujours des réponses qui concernent des ressources en matière d'inventeurs. Toutefois, le point de terminaison `https://wipo.int/api/v1/inventors` n'est pas autorisé car la ressource en matière d'inventeur ne peut pas être autonome.

29. Seules les ressources de premier niveau, c'est-à-dire avec un niveau au maximum, devraient être utilisées, sauf à rendre très complexe l'implémentation de ces API. Par exemple, `https://wipo.int/api/v1/patents?inventorId=12345` devrait être utilisée au lieu de `https://wipo.int/api/v1/inventors/12345/patents`.

[RSG-13] Une API Web DEVRAIT n'utiliser que des ressources de premier niveau. S'il y a des sous-ressources, elles devraient être des collections et impliquer une association. Une entité devrait être accessible soit comme ressource de premier niveau, soit comme sous-ressource, mais pas les deux.

[RSG-14] Si une ressource peut être autonome, elle DOIT être une ressource de premier niveau; dans le cas contraire, elle doit être une sous-ressource.

[RSG-15] Pour extraire des données imbriquées, les paramètres de requête DOIVENT être utilisés au lieu des chemins d'URL.

30. Il existe des types[2] d'API Web : les API Web CRUD (Create, Read, Update, and Delete (créer, lire, mettre à jour et supprimer)) et les API Web Intent. Les API Web CRUD modélisent les changements apportés à une ressource, c'est-à-dire créer/lire/mettre à jour/supprimer des opérations. Les API Web Intent, elles, modélisent les activités métier, par exemple renouveler/enregistrer/publier. Les opérations CRUD devraient utiliser des noms et les API Web Intent devraient utiliser des verbes pour désigner les ressources. Les API Web CRUD sont les plus courantes, mais les deux types peuvent être combinés; par exemple, le consommateur du service pourrait utiliser une API Web Intent modélisant une activité métier, qui orchestrerait l'exécution d'une ou plusieurs opérations du service d'API Web CRUD. S'il utilise une API Web CRUD, le demandeur de service doit orchestrer la logique métier, mais avec les API Web Intent, c'est le prestataire du service qui le fait. Les API Web CRUD sont moins atomiques que les API Web Intent[3].

– Par exemple, un propriétaire de marques veut renouveler celles dont la durée de protection va bientôt expirer (par exemple, le `aaaa-mm-jj`). Il s'agit d'une combinaison des activités métier ci-après :
  – extraire les marques dont la durée de protection expire le `aaaa-mm-jj`; et
  – renouveler les marques ainsi extraites à l'aide de leur numéro d'enregistrement international.

  Si une API Web CRUD est utilisée, les activités métier précédentes seraient modélisées par un processus non atomique, requérant deux actions telles que les suivantes :

Phase 1 : Extraire toutes les marques au format XML[4] qui appartiennent au propriétaire appelé John Smith et dont la durée de protection expirera, par exemple, le 31-12-2018 :

```
GET /api/v1/trademarks? holderFullName=John%20Smith&expiryDate=2018-12-31. HTTP/1.1
Host: wipo.int
Accept: application/xml
```

L'exemple de réponse HTTP ci-après est obtenu :

```
HTTP/1.1 200 OK
Content-Type: application/xml

<?xml version="1.0" encoding="UTF-8"?>
<tmk:TrademarkBag xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:com="http://www.wipo.int/standards/XMLSchema/ST96/Common"
xmlns:tmk="http://www.wipo.int/standards/XMLSchema/ST96/Trademark"
xsi:schemaLocation="http://www.wipo.int/standards/XMLSchema/ST96/Trademark
TrademarkBag.xsd">
      <tmk:Trademark xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:com="http://www.wipo.int/standards/XMLSchema/ST96/Common"
xmlns:tmk="http://www.wipo.int/standards/XMLSchema/ST96/Trademark"
com:operationCategory="Delete"
xsi:schemaLocation="http://www.wipo.int/standards/XMLSchema/ST96/Trademark
Trademark.xsd">
            ...
            <com:RegistrationNumber>
                  <com:IPOfficeCode>IT</com:IPOfficeCode>

      <com:ST13ApplicationNumber>000000000000001</com:ST13ApplicationNumber>
            </com:RegistrationNumber>
            ...
```

[2] Nous pourrions aussi classer les API selon leur archétype. Voir par exemple : "REST API Design Rulebook : Designing Consistent RESTful Web Service Interfaces".

[3] De plus, une API Intent permet d'appliquer la configuration Command Query Responsibility Segregation (CQRS) (séparation commande-requête). Cette configuration permet d'utiliser deux modèles différents pour mettre à jour et lire l'information. Le fondement logique en est que, pour beaucoup de problèmes, en particulier dans les domaines plus complexes, le fait d'appliquer le même modèle conceptuel pour les commandes et les requêtes débouche sur un modèle plus complexe qui n'est pas bénéfique.

[4] L'exemple au format JSON est sauté car il n'ajoute aucune valeur en l'occurrence.

```
            <com:ExpiryDate>2018-12-31</com:ExpiryDate>
            ...
      </tmk:Trademark>
      ...
</tmk:TrademarkBag>
```

Phase 2 : Présenter une demande de renouvellement de marque pour chaque marque extraite dans la phase précédente (on ne décrit ici que la première demande de renouvellement) :

```
POST /api/v1/trademarks/renewalRequests HTTP/1.1
Host: wipo.int
Accept: application/xml
Content-Type: application/xml
<?xml version="1.0" encoding="UTF-8"?>
<tmk:MadridRenewal xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:com="http://www.wipo.int/standards/XMLSchema/ST96/Common"
xmlns:tmk="http://www.wipo.int/standards/XMLSchema/ST96/Trademark"
xsi:schemaLocation="http://www.wipo.int/standards/XMLSchema/ST96/Trademark
MadridRenewal.xsd">
      ...
      <com:InternationalRegistrationNumber>000000000000001</com:InternationalRegist
rationNumber>
      ...
</tmk:MadridRenewal>
```

– L'exemple précédent pourrait aussi être modélisé avec une demande de service atomique utilisant en API Web Intent, comme suit[5] :

```
POST /api/v1/trademarks/findAndRenew?holderFullName=john%20smith&expiryDate=2018-
12-31
Host: wipo.int
```

31. Ensuite, le type d'API Web devrait imposer des contraintes en ce qui concerne la manière dont les ressources sont nommées afin d'indiquer celle qui est utilisée. On notera que les noms de ressources localisés du fait de besoins opérationnels peuvent être exprimés dans d'autres langues.

[RSG-16] Les noms de ressources DEVRAIENT être des noms pour les API Web CRUD et des verbes pour les API Web Intent.

[RSG-17] Si un nom de ressource est un nom, il DEVRAIT toujours être au pluriel. Les formes de nom irrégulières NE DEVRAIENT PAS être utilisées. Par exemple, /persons devrait être utilisé au lieu de /people.

[RSG-18] Les paramètres des noms de ressources, des segments et des requêtes DOIVENT être composés de mots anglais, écrits selon l'orthographe anglaise figurant dans le dictionnaire Oxford English Dictionary. Les noms de ressources localisés du fait de besoins opérationnels PEUVENT être exprimés dans d'autres langues.

Prise en charge de formats multiples

32. Les consommateurs du service peuvent avoir des besoins différents en ce qui concerne le format auquel le service fournit les données. Le type de média utilisé pour les données devrait être découplé des données elles-mêmes, ce qui permettrait au service de prendre en charge divers types de médias. Par conséquent, une API Web doit prendre en charge la négociation du type de contenu en utilisant l'en-tête requête HTTP `Accept` et l'en-tête réponse HTTP `Content-Type` comme l'exige le RFC 7231 de l'IETF. Par exemple, pour demander des données au format JSON, l'en-tête `Accept` devrait être `Accept: application/json` et, pour des données au format XML, l'en-tête `Accept` devrait être `Accept:`

[5] L'élément `InternationalRegistrationNumber` a été supprimé de la charge utile pour représenter tous les NEI. La norme ST.96 ne devrait pas être utilisée, même sous une forme assouplie, car cet exemple étend les cas d'utilisation autorisés par la norme ST.96.

`application/xml`. De même pour l'en-tête `Content-Type`. En outre, une API Web peut prendre en charge d'autres modalités de négociation du type de contenu, comme le paramètre de requête (par exemple`?format`) ou le suffixe d'URL (par exemple `.json`).

[RSG-19] Une API Web DEVRAIT utiliser, pour la négociation du type de contenu, l'en-tête requête HTTP `Accept` et l'en-tête réponse HTTP `Content-Type`.

33. Les API doivent prendre en charge les requêtes et les réponses aux formats XML et JSON. Pour le XML, les réponses doivent être conformes à une norme de l'OMPI utilisant le XML, comme la norme ST.96, et pour le format JSON, les réponses doivent être conformes à la norme ST.97 de l'OMPI. Une correspondance systématique entre ces deux formats devrait être utilisée.

[RSG-20] Une API Web DOIT prendre en charge la négociation du type de contenu d'une façon conforme au RFC 7231 de l'IETF.

[RSG-21] Le format JSON DOIT être retenu lorsque aucun type de contenu spécifique n'est demandé.

[RSG-22] Une API Web DEVRAIT renvoyer le code d'état "`406 Not Acceptable`" si un format demandé n'est pas pris en charge.

[RSG-23] Une API Web DEVRAIT rejeter les requêtes contenant des en-têtes de type de contenu imprévus ou manquants en indiquant le code d'état HTTP "`406 Not Acceptable`" ou "`415 Unsupported Media Type`".

[RSG-24] Les requêtes et les réponses (convention de nommage, format du message, structure des données et dictionnaire de données) DEVRAIENT se référer à la norme ST.96 de l'OMPI pour le format XML et à la norme ST.97 de l'OMPI pour le format JSON.

[RSJ-25] Les noms de propriété d'objets JSON DEVRAIENT être fournis en caractères bas de casse de type "camel", par exemple applicantName.

[RSX-26] Les noms de composantes XML DEVRAIENT être fournis en caractères haut de casse de type "camel".

[RSG-27] Une API Web DOIT prendre en charge au moins le XML ou le JSON.

Méthodes HTTP

34. Les méthodes HTTP (ou verbes HTTP) sont un type de fonction fourni par un contrat uniforme de traitement d'identificateurs de ressources et de données. Ces méthodes doivent être utilisées comme prévu au départ en respectant la sémantique normalisée exposée dans les RFC 7231 et 5789 de l'IETF, à savoir :

- `GET` – extraire des données
- `HEAD` – comme `GET` mais sans charge utile de réponse
- `POST` – envoyer de nouvelles données
- `PUT` – mettre à jour
- `PATCH` – faire une mise à jour partielle
- `DELETE` – supprimer des données
- `TRACE` – reprendre
- `OPTIONS` – verbes de requête pris en charge par le serveur pour une adresse URL donnée

35. Le contrat uniforme établit un ensemble de méthodes que les services doivent utiliser et figurant dans une collection ou un inventaire donné. La tunnellisation des méthodes HTTP peut être utile lorsque les en-têtes HTTP sont rejetés par certains pare-feu.

36. Les méthodes HTTP peuvent se conformer au principe de sélection, selon lequel seule la fonctionnalité exigée par le scénario d'usage cible doit être implémentée. Certains mandataires ne prennent en charge que les méthodes `POST` et

GET. Pour surmonter ces limitations, une API Web peut utiliser une méthode POST en ajoutant un en-tête HTTP personnalisé qui "tunnellise" la véritable méthode HTTP.

[RSG-28] Les méthodes HTTP DOIVENT se limiter aux méthodes HTTP normalisées POST, GET, PUT, DELETE, OPTIONS, PATCH, TRACE et HEAD, spécifiées dans les RFC 7231 et 5789 de l'IETF.

[RSG-29] Les méthodes HTTP PEUVENT se conformer au principe de sélection, selon lequel seule la fonctionnalité exigée par le scénario d'usage cible doit être implémentée.

[RSG-30] Certains mandataires ne prennent en charge que les méthodes POST et GET. Pour surmonter ces limitations, une API Web PEUT utiliser une méthode POST en ajoutant un en-tête HTTP personnalisé qui "tunnellise" la véritable méthode HTTP. L'en-tête HTTP personnalisé X-HTTP-Method DEVRAIT être utilisé.

[RSG-31] Si une méthode HTTP n'est pas prise en charge, le code d'état HTTP "405 Method Not Allowed" DEVRAIT être renvoyé.

37. Dans certains cas d'utilisation, des opérations multiples devraient être prises en charge simultanément.

[RSG-32] Une API Web DEVRAIT prendre en charge des opérations par lots (ou opérations en vrac) au lieu de requêtes individuelles multiples afin de réduire le temps d'attente. La même sémantique devrait être utilisée pour les méthodes HTTP et les codes d'état HTTP. La charge utile des réponses DEVRAIT contenir des informations sur toutes les opérations par lots. Si de multiples erreurs se produisent, la charge utile des erreurs DEVRAIT contenir des informations sur toutes les erreurs survenues (dans l'attribut de l'élément détails). Toutes les opérations en vrac DEVRAIENT être exécutées dans une opération atomique.

GET

38. Selon le RFC 2616 de l'IETF, le protocole HTTP ne fixe aucune limite préalable à la longueur d'un URI. D'un autre côté, les serveurs devraient être incités par prudence à ne pas dépendre de longueurs d'URI supérieures à 255 octets, car certaines implémentations client ou mandataire anciennes peuvent ne pas les prendre correctement en charge. Dans le cas où cette limite est dépassée, il est recommandé d'utiliser des requêtes nommées. Selon une autre formule, un ensemble de règles déterminant les modalités de conversion entre une méthode GET et une méthode POST doit être spécifié. Selon le RFC susvisé, une requête GET doit être idempotente, en ce que la réponse sera identique, quel que soit le nombre d'exécutions de la requête.

[RSG-33] Pour un point de terminaison qui apporte une ressource unique, si aucune ressource n'est trouvée, la méthode GET DOIT renvoyer le code d'état "404 Not Found". Les points de terminaison qui renvoient des listes de ressources indiqueront simplement une liste vide.

[RSG-34] Si l'extraction d'une ressource aboutit, la méthode GET DOIT renvoyer 200 OK.

[RSG-35] Une requête GET DOIT être idempotente.

[RSG-36] Lorsque la longueur d'un URI dépasse 255 octets, la méthode POST DEVRAIT être utilisée à la place de GET en raison des limitations de GET. On peut aussi songer à créer des requêtes nommées, si cela est possible.

HEAD

39. Lorsqu'un client a besoin d'informations sur une opération, il peut utiliser HEAD. HEAD obtient l'en-tête HTTP que vous obtiendriez si vous faisiez une requête GET, mais sans le corps de la requête. Le client peut ainsi déterminer les informations sur la mise en antémémoire, ainsi que le type de contenu et le code d'état qui seront renvoyés. Selon le RFC 2616 de l'IETF, une requête HEAD DOIT être idempotente.

[RSG-37] Une requête HEAD DOIT être idempotente.

[RSG-38] Certains mandataires ne prennent en charge que les méthodes `POST` et `GET`. Afin de surmonter ces limitations, une API Web DEVRAIT prendre en charge un en-tête de requête HTTP personnalisé pour surclasser la méthode HTTP.

POST

40. Lorsqu'un client a besoin de créer une ressource, il peut utiliser `POST`. Par exemple, la requête HTTP ci-après envoie une requête comportant une demande de brevet.

– Exemple d'envoi d'une requête comportant une demande de brevet.

Exemple avec des charges utiles XML basées sur la norme ST.96

Le client envoie la requête comportant une demande de brevet au format XML :

```
POST /v1/patents/applications HTTP/1.1
Host: wipo.int
Accept: application/xml
Content-Type: application/xml
<?xml version="1.0" encoding="UTF-8"?>
<pat:ApplicationBody xmlns="http://www.wipo.int/standards/XMLSchema/ST96/Common"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:com="http://www.wipo.int/standards/XMLSchema/ST96/Common"
xmlns:pat="http://www.wipo.int/standards/XMLSchema/ST96/Patent"
com:languageCode="pl" com:receivingOffice="ST" com:st96Version="V5_0"
xsi:schemaLocation="http://www.wipo.int/standards/XMLSchema/ST96/Patent
ApplicationBody_V5_0.xsd">
      ...
</pat:ApplicationBody>
```

La réponse HTTP ci-après indique le dépôt réussi de la demande de brevet :

```
HTTP/1.1 201 Created
Content-Type: application/xml
<?xml version="1.0" encoding="UTF-8"?>
<pat:ApplicationBody xmlns="http://www.wipo.int/standards/XMLSchema/ST96/Common"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:com="http://www.wipo.int/standards/XMLSchema/ST96/Common"
xmlns:pat="http://www.wipo.int/standards/XMLSchema/ST96/Patent"
com:languageCode="pl" com:receivingOffice="ST" com:st96Version="V5_0"
xsi:schemaLocation="http://www.wipo.int/standards/XMLSchema/ST96/Patent
ApplicationBody_V5_0.xsd" applicationBodyStatus="pending">
      ...
</pat:ApplicationBody>
```

Exemple avec des charges utiles JSON sur la base de la norme ST.97

Le client envoie la requête comportant une demande de brevet au format JSON :

```
POST /v1/patents/applications HTTP/1.1
Host: wipo.int
Accept: application/json
Content-Type: application/json
{
```

```
      " applicationBody ": {
            ...
      }
}
```

La réponse HTTP ci-après indique le dépôt réussi de la demande de brevet :

```
HTTP/1.1 200 OK
Content-Type: application/json
{
      " applicationBody ": {
            "applicationBodyStatus" : "pending",
            ...
      }
}
```

[RSG-39] Selon le RFC 2616 de l'IETF, une requête `POST` NE DOIT PAS être idempotente.

[RSG-40] Si la création de la ressource a réussi, l'en-tête HTTP `Location` DEVRAIT contenir un URI (absolu ou relatif) indiquant une ressource créée.

[RSG-41] Si la création de la ressource a réussi, la réponse DEVRAIT contenir le code d'état "`201 Created`".

[RSG-42] Si la création de la ressource a réussi, la charge utile de la réponse DEVRAIT par défaut contenir le corps de la ressource créée, pour permettre au client de l'utiliser sans avoir à faire un nouvel appel HTTP.

PUT

41. Lorsqu'un client a besoin de remplacer entièrement une ressource existante, il peut utiliser `PUT`. Les caractéristiques idempotentes de `PUT` devraient être prises en considération. Une requête `PUT` comporte une sémantique de mise à jour (telle que mentionnée dans le RFC 7231 de l'IETF), et une sémantique d'insertion.

[RSG-43] Une requête `PUT` DOIT être idempotente.

[RSG-44] Si une ressource n'est pas trouvée, `PUT` DOIT renvoyer le code d'état "`404 Not Found`".

[RSG-45] Si la mise à jour d'une ressource a réussi, `PUT` DOIT renvoyer le code d'état "`200 OK`" si la ressource mise à jour est retournée ou un "`204 No Content`" si elle ne l'est pas.

PATCH

42. Lorsqu'un client a besoin d'une mise à jour partielle, il peut utiliser `PATCH`. Les caractéristiques idempotentes de `PATCH` devraient être prises en considération.

– Par exemple, la requête ci-après ne met à jour que la langue d'un brevet dont le numéro est indiqué :

```
PATCH /api/v1/patents/publications/100000000000001 HTTP/1.1
Host: wipo.int
If-Match:456
Content-Type: application/merge-patch+json
{ "languageCode": "en" }
```

43. Selon le RFC 2616 de l'IETF, PATCH ne doit pas être idempotent. Afin de le rendre idempotent, l'API peut suivre la suggestion du RFC 5789 de l'IETF consistant à utiliser un verrouillage optimiste.

[RSG-46] Une requête `PATCH` NE DOIT PAS être idempotente.

[RSG-47] Si une API Web exécute des mises à jour partielles, les caractéristiques idempotentes de `PATCH` DEVRAIENT être prises en considération. Afin de le rendre idempotent, l'API PEUT suivre la suggestion du RFC 5789 de l'IETF consistant à utiliser un verrouillage optimiste.

[RSG-48] Si une ressource n'est pas trouvée, `PATCH` DOIT renvoyer le code d'état "`404 Not Found`".

[RSJ-49] Si une API Web exécute des mises à jour partielles à l'aide de `PATCH`, elle DOIT utiliser le format de Patch conçu pour JSON (*JSON Merge Patch format*) pour décrire l'ensemble de modifications partielles dont il est question dans le RFC 7386 de l'IETF (en utilisant le type de contenu `application/merge-patch+json`).

DELETE

44. Lorsqu'un client a besoin de supprimer une ressource, il peut utiliser `DELETE`. Selon le RFC 2616 de l'IETF, une requête `DELETE` ne doit pas être idempotente.

[RSG-50] Une requête `DELETE` NE DOIT PAS être idempotente.

[RSG-51] Si une ressource n'est pas trouvée, `DELETE` DOIT renvoyer le code d'état "`404 Not Found`".

[RSG-52] Si une ressource est supprimée avec succès, `DELETE` DOIT renvoyer le code d'état "`200 OK`" si la ressource supprimée est retournée ou "`204 No Content`" si elle ne l'est pas.

TRACE

45. Selon le RFC 2616 de l'IETF, la méthode `TRACE` n'est pas assortie d'une sémantique API et est utilisée à des fins de test et de collecte de renseignements diagnostiques, par exemple pour tester une chaîne de mandataires. `TRACE` permet au client de visualiser ce qui est reçu à l'autre bout de la chaîne de requête et utilise ces données. Selon le même RFC, une requête `TRACE` NE DOIT PAS être idempotente.

[RSG-53] Le destinataire final est soit le serveur d'origine, soit le premier mandataire ou la première passerelle à recevoir une valeur de `Max-Forwards` de zéro dans la requête. Une requête `TRACE` NE DOIT PAS inclure un corps.

[RSG-54] Une requête `TRACE` NE DOIT PAS être idempotente.

[RSG-55] La valeur du champ d'en-tête HTTP `Via` DOIT suivre la chaîne de requête.

[RSG-56] Le champ d'en-tête HTTP `Max-Forwards` DOIT être utilisé pour permettre au client de réduire la longueur de la chaîne de requête.

[RSG-57] Si la requête est valide, la réponse DEVRAIT contenir l'intégralité du message de requête dans le corps de la réponse, avec `Content-Type` de "`message/http`".

[RSG-58] Les réponses à `TRACE` NE DOIVENT PAS être mises en antémémoire.

[RSG-59] Le code d'état "`200 OK`" DEVRAIT être renvoyé à `TRACE`.

OPTIONS

46. Lorsqu'un client a besoin d'informations sur une API Web, il peut utiliser `OPTIONS`. La méthode `OPTIONS` n'est pas assortie d'une sémantique API. Une requête `OPTIONS` DOIT être idempotente selon le RFC 2616 de l'IETF, En-têtes HTTP personnalisés.

[RSG-60] Une requête `OPTIONS` DOIT être idempotente.

47. Il est courant pour une API Web utilisant des en-têtes HTTP personnalisés de fournir le préfixe usuel "`X-`", que le RFC 6648 désapprouve et dont il décourage l'utilisation.

[RSG-61] Les en-têtes HTTP personnalisés commençant par le préfixe "`X-`" NE DEVRAIENT PAS être utilisés.

[RSG-62] Les en-têtes HTTP personnalisés NE DEVRAIENT PAS être utilisés pour modifier le comportement des méthodes HTTP, à moins qu'il ne s'agisse de venir à bout d'éventuelles limites techniques (p. ex., voir [RSG-39]).

[RSG-63] La convention de nommage pour les en-têtes HTTP personnalisés est `<organization>-<header name>`, où `<organization>` et `<header>` DEVRAIENT respecter la convention de la police de caractères kebab.

48. En vertu des principes de la conception orientée services, les clients et les services devraient évoluer indépendamment. Le versionnage de services le permet. Les implémentations courantes du versionnage de services sont les suivantes : versionnage d'en-tête (en utilisant un en-tête personnalisé), versionnage de la chaîne d'interrogation (p. ex., `?v=v1`), versionnage par type de média (p. ex. `Accept: application/vnd.v1+json`) et versionnage d'URI (p. ex. `/api/v1/inventors`).

[RSG-64] Une API Web DEVRAIT prendre en charge une méthode unique de versionnage de services en utilisant le versionnage d'URI (p. ex.`/api/v1/inventors`), le versionnage d'en-tête (p ex. `Accept-version: v1`) ou le versionnage par type de média (p. ex. `Accept: application/vnd.v1+json`). Le versionnage de la chaîne d'interrogation NE DEVRAIT PAS être utilisé.

49. En vertu des principes de la conception orientée services, les prestataires et les consommateurs de services devraient également évoluer indépendamment. Le consommateur de services ne devrait pas être affecté par les modifications mineures (rétrocompatibles) apportées par le prestataire. En conséquence, le versionnage de services ne devrait utiliser que les versions majeures. Pour les API non publiées à usage interne (par exemple, aux fins de développement et d'essai), les versions mineures, telles que le versionnage sémantique, peuvent également être utilisées.

[RSG-65] Un système de versionnage et numérotation DEVRAIT être appliqué en ne prenant en considération que le numéro de la version majeure (p. ex. `/v1`).

50. Les identificateurs de points de terminaison de services comprennent des informations qui peuvent évoluer dans le temps. Il peut ne pas être possible de remplacer toutes les références à un point de terminaison obsolète, ce qui peut amener le consommateur du service à ne plus pouvoir interagir avec le point de terminaison de ce service. Le prestataire peut donc renvoyer une réponse de redirection. La redirection peut être temporaire ou permanente. Les codes d'état HTTP ci-après sont disponibles :

	Permanent	**Temporaire**
Permet de remplacer la méthode de requête POST par la méthode GET	301	302
Ne permet pas de remplacer la méthode de requête POST par la méthode GET	308	307

Étant donné que `301` et `302` sont plus génériques, ils sont préférés pour accroître la flexibilité et surmonter toute complexité superflue.

[RSG-66] Les contrats de service API PEUVENT comprendre une fonctionnalité de redirection des points de terminaison. Lorsqu'un consommateur de services essaie d'invoquer un service, une réponse de redirection peut

lui être adressée pour lui dire de renvoyer la requête à un nouveau point de terminaison. Les redirections PEUVENT être temporaires ou permanentes :

- Redirection temporaire – utilisant l'en-tête de réponse HTTP `Location` et le code d'état HTTP "`302 Found`" selon le RFC 7231 de l'IETF; ou
- Redirection permanente – utilisant l'en-tête de réponse HTTP `Location` et le code d'état HTTP "`301 Moved Permanently`" selon le RFC 7238 de l'IETF.

51. Une API étant évolutive, elle passera par un certain nombre de phases importantes : planification et conception, développement, test, déploiement et retrait. Plutôt que de fournir des recommandations concernant les périodes pendant lesquelles une API devrait, de préférence, être maintenue dans une phase donnée, il est préférable que l'organisation ou les prestataires de services publient la stratégie applicable au cycle de vie de leur API. L'annexe VII présente un modèle comprenant les éléments de base qui définissent une stratégie applicable au cycle de vie.

[RSG-67] Des stratégies applicables au cycle de vie des API DEVRAIENT être publiées par les développeurs afin d'aider les utilisateurs à comprendre combien de temps une version doit être maintenue.

Configurations des requêtes de données

Options de pagination

52. La pagination permet à un client d'extraire des données sous forme de pages. L'utilisation de la pagination évite d'inonder le prestataire de services de requêtes coûteuses en ressources selon les principes de conception. Le serveur définit la taille d'une page par défaut si le consommateur ne l'a pas précisée. Les requêtes paginées ne peuvent pas être idempotentes, c'est-à-dire qu'une requête paginée ne donne pas d'aperçu des données.

[RSG-68] Une API Web DEVRAIT prendre en charge la pagination.

[RSG-69] Les requêtes paginées NE PEUVENT PAS être idempotentes.

[RSG-70] Une API Web DOIT utiliser des paramètres de requête pour exécuter la pagination.

[RSG-71] Une API Web NE DOIT PAS utiliser d'en-têtes HTTP pour exécuter la pagination.

[RSG-72] Les paramètres de requête `limit=<number of items to deliver>` et `offset=<number of items to skip>` DEVRAIENT être utilisés, où `limit` est le nombre d'objets à renvoyer (taille de la page) et `skip` le nombre d'objets à sauter (déduction). Si aucune taille de page n'est précisée, une taille par défaut DEVRAIT être définie – globale ou par collection; la déduction par défaut DOIT être zéro "0" :

- Par exemple, voici une adresse URL valide :

```
https://wipo.int/api/v1/patents?limit=10&offset=20
```

[RSG-73] Les valeurs des paramètres `limit` et `offset` DEVRAIENT figurer dans la réponse.

Tri

53. L'extraction des données peut nécessiter leur tri (sens croissant ou décroissant). Un critère de tri multiclé peut aussi être utilisé. Le tri est déterminé par l'utilisation du paramètre de chaîne de requête `sort`. La valeur de ce paramètre est une liste de clés de tri séparées par une virgule et de sens de tri qui peuvent être optionnellement ajoutés à chaque clé de tri, séparés par les deux points ":". Les sens de tri pris en charge sont "`asc`" pour croissant ou "`desc`" pour décroissant. Le client peut indiquer un sens de tri pour chaque clé. Si un sens de tri n'est pas indiqué pour une clé, le serveur en définit un par défaut.

Par exemple :

a) Seules les clés de tri sont indiquées :

`sort=key1,key2`

'`key1`' est la première clé et '`key2`' est la deuxième clé, et les sens de tri sont définis par défaut par le serveur

b) Certains sens de tri sont indiqués :

`sort=key1:asc,key2`

où '`key1`' est la première clé (sens croissant) et '`key2`' est la deuxième clé (sens défini par défaut par le serveur, c'est-à-dire que toute clé de tri sans sens correspondant est définie par défaut).

c) Le sens est indiqué pour chaque clé :

`sort=key1:asc,key2:desc`

où '`key1`' est la première clé (sens croissant) et '`key2`' est la deuxième clé (sens décroissant).

54. Afin de spécifier un tri selon des critères multiattributs, la valeur d'un paramètre de requête peut être une liste de clés de tri séparées par une virgule et de sens de tri, avec '`asc`' pour croissant ou '`desc`' pour décroissant, pouvant être ajoutés à chaque clé de tri, séparés par les deux points ":".

[RSG-74] Une API Web DEVRAIT prendre en charge le tri.

[RSG-75] Afin de spécifier un critère de tri multiattribut, un paramètre de requête DOIT être utilisé. La valeur de ce paramètre est une liste de clés de tri séparées par une virgule et de sens de tri avec '`asc`' pour croissant ou '`desc`' pour décroissant. Ces derniers PEUVENT être ajoutés à chaque clé de tri, séparés par les deux points. Le serveur DOIT définir le sens par défaut si celui-ci n'est pas spécifié pour une clé.

[RSG-76] Une API Web DEVRAIT renvoyer les critères de tri dans la réponse.

Extension

55. Un consommateur de services peut contrôler la quantité de données qu'il reçoit en étendant un champ unique à des objets plus importants. Cela est le plus souvent combiné avec une prise en charge hypermédia. Au lieu de demander simplement l'insertion de l'identifiant d'une entité liée, la personne invoquant le service peut demander l'extension de la représentation intégrale de l'entité dans le cadre des résultats. Les demandes de services peuvent mettre en œuvre une extension pour obtenir toutes les données nécessaires dans une seule requête API :

– Par exemple, si l'hypermédia est pris en charge, la requête HTTP ci-après extrait un brevet et étend le déposant.

Exemple avec des charges utiles JSON sur la base de la norme ST.97

Extraire un brevet à partir de son numéro[6] :

```
GET /api/v1/patents/publications/100000000000001 HTTP/1.1
Host: wipo.int
Accept: application/json
```

[6] Patent/PatentNumber.xsd

La réponse HTTP est la suivante :

```
HTTP/1.1 200 OK
Content-Type: application/json
200 OK
{
      "patentPublication":{
            "languageCode": "en",
            ...
            "bibliographicData": {
                  "st96Version": "V5 0",
                  "applicationIdentification": {
                        "ipOfficeCode": "XX",
                        "applicationNumber": {
                              "applicationNumberText": "13797521"
                        },
                        "inventionSubjectMatterCategory": "Utility",
                        "filingDate": "2013-03-12"
                  },
                  "patentGrantIdentification": {
                        "ipOfficeCode": "XX",
                        "patentNumber": "100000000000001"
                  },
                  ...
            "partyBag": {
                  "applicantBag": {
                        "applicant": {
                              "href": "https://wipo.int/api/v1/link/to/applicants"
                        },
                        ...
                        }
                  }
            },
            ...
      }
}
```

Au lieu de la requête précédente, l'utilisation de la requête HTTP ci-après permet d'extraire l'intégralité des informations concernant le déposant de la demande de brevet portant le numéro 100000000000001 :

```
GET /api/v1/patents/publications?id=100000000000001&expand=applicant HTTP/1.1
Host: wipo.int
Accept: application/json
```

La réponse HTTP est la suivante :

```
HTTP/1.1 200 OK
Content-Type: application/json
200 OK
{
      "patentPublication":{
      "languageCode": "en",
      ...
            "bibliographicData": {
            "st96Version": "V5_0",
            "applicationIdentification": {
                  "ipOfficeCode": "XX",
```

```
                "applicationNumber": {
                      "applicationNumberText": "13797521"
                },
                "inventionSubjectMatterCategory": "Utility",
                "filingDate": "2013-03-12"
          },
                "patentGrantIdentification": {
                "ipOfficeCode": "XX",
                      "patentNumber": "100000000000001"
          },
          ...
          "partyBag": {
                "applicantBag": {
                      "applicant": {
                            "applicantCategory": ...,
                                  {
                                        "sequenceNumber": "001",
                                        "publicationContact": [
                                              {
                                                    "name": {
                                                          "personName": …,
                                        "applicantCategory": "Applicant",
                                  },
                                  {
                                        "sequenceNumber": "002",
                                        "publicationContact": [
                                              {
                                                    "name": {
                                                          "personName": …
                                                    }
                                              }
                                        ],
                                        "applicantCategory": "Applicant",
                                  },
                                  {
                                        "sequenceNumber": "003",
                                        "publicationContact": [
                                              {
                                                    "name": {
                                                          "personName": …
                                                    }
                                              }
                                        ],
                                        "applicantCategory": "Applicant",
                                  },
                      ...
                      }
          }
    },
          ...   }}
```

56. Une API Web peut prendre en charge l'extension du corps du contenu renvoyé.

[RSG-77] Une API Web PEUT prendre en charge l'extension du corps du contenu renvoyé. Le paramètre de requête `expand=<comma-separated list of attributes names>` DEVRAIT être utilisé.

Projection

57. Une API Web devrait prendre en charge la projection du champ, qui contrôle la quantité de données d'une entité qui est renvoyée en réponse à une requête API. La projection de champ peut diminuer le temps de réponse et la taille de la charge utile. Si seulement certains attributs des données extraites sont requis, un paramètre de requête de projection doit être utilisé au lieu des chemins d'URL. Le paramètre de requête devrait être formé comme suit : "`fields="<comma-separated list of attribute names>`. Un paramètre de requête de projection est plus facile à exécuter et peut

extraire des attributs multiples. Si une projection est prise en charge, le schéma XSD/JSON ne devrait pas s'appliquer dans la réponse car celle-ci ne sera pas valide au regard du schéma XSD/JSON originel.

– Par exemple, au message de requête ci-après n'est renvoyé que le nom complet de l'auteur du brevet d'invention requis :

Dans le cas des charges utiles XML sur la base de la norme ST.96

Obtenez le nom complet de l'auteur du brevet d'invention dont l'identifiant est id12345 :

```
GET /api/v1/patents/inventors/id12345?fields=fullName
Host: wipo.int
Accept: application/xml
```

Voici un exemple de message de réponse HTTP:

```
HTTP/1.1 200 OK
Content-Type: application/xml
<?xml version="1.0" encoding="UTF-8"?>
<pat:Inventor xmlns="http://www.wipo.int/standards/XMLSchema/ST96/Common"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:com="http://www.wipo.int/standards/XMLSchema/ST96/Common"
xmlns:pat="http://www.wipo.int/standards/XMLSchema/ST96/Patent"
com:sequenceNumber="String" com:id="ID1"
xsi:schemaLocation="http://www.wipo.int/standards/XMLSchema/ST96/Patent
PatentPublication_V5_0.xsd">
      <Contact>
            <Name>
                  <PersonName>
                        <PersonFullName>John Smith</PersonFullName>
                  </PersonName>
            </Name>
      </Contact>
</pat:Inventor>
```

Dans le cas des charges utiles JSON sur la base de la norme ST.97

Obtenez le nom complet de l'auteur du brevet d'invention dont l'identifiant[7] est id12345 :

```
GET /api/v1/patents/inventors/id12345?fields=fullName
Host: wipo.int
Accept: application/json
```

Voici un exemple de message de réponse HTTP:

```
HTTP/1.1 200 OK
Content-Type: application/json
{
      "inventor": {
            "sequenceNumber": "001",
```

[7] Common/id.xsd

```
            "Contact": [
                  {
                        "name": {
                              "personName": [
                                    {
                                          "personFullName": "John Smith"
                                    }
                              ]
                        }
                  }
            ]
}}
```

[RSG-78] Un paramètre de requête DEVRAIT être utilisé au lieu des chemins d'URL si une API Web prend en charge la projection selon le format : "`fields="<comma-separated list of attribute names>`.

Nombre d'objets

58. Dans certains cas d'utilisation, le nombre d'objets d'une collection peut intéresser le consommateur de l'API. Cela est très courant en cas de combinaison avec la pagination pour connaître le nombre total d'objets de la collection.

– Par exemple, la requête HTTP ci-après extrait au maximum 3 publications de brevets, sautant les 4 premiers résultats, et la réponse devrait également contenir le nombre total de résultats disponibles :

Exemple avec des charges utiles XML basées sur la norme ST.96

```
GET /api/v1/patents/publications?count=true&limit=3&offset=4 HTTP/1.1
Host: wipo.int
Accept: application/xml
```

Voici un exemple de réponse HTTP renvoyée :

```
HTTP/1.1 200 OK
Content-Type: application/xml
<?xml version="1.0" encoding="UTF-8"?>
<pat:PatentPublication xmlns="http://www.wipo.int/standards/XMLSchema/ST96/Common"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:com="http://www.wipo.int/standards/XMLSchema/ST96/Common"
xmlns:pat="http://www.wipo.int/standards/XMLSchema/ST96/Patent"
com:languageCode="de" com:st96Version="V5_0"
xsi:schemaLocation="http://www.wipo.int/standards/XMLSchema/ST96/Patent
PatentPublication_V5_0.xsd">
   ...
</pat:PatentPublication>
<pat:PatentPublication>
   ...
</pat:PatentPublication>
   ...
<pat:PatentPublication>
   ...
</pat:PatentPublication>
<count>100</count>
```

Exemple avec des charges utiles JSON sur la base de la norme ST.97

```
GET /api/v1/patents/publications?count=true&limit=3&offset=4 HTTP/1.1
Host: wipo.int
Accept: application/json
```

Voici un exemple de réponse HTTP renvoyée :

```
HTTP/1.1 200 OK
Content-Type: application/json
{
      "patentPublication": [
            {
            ...
            },
            {
            ...
            },
            {
            ...
            }
      ],
      "count": 100
}
```

59. Une API Web pourrait prendre en charge le renvoi du nombre d'objets d'une collection en l'insérant dans la partie de la réponse qui contient la collection elle-même. Selon une autre solution, ce nombre ferait partie d'une enveloppe de métadonnées, en dehors de la partie principale de la réponse.

[RSG-79] Une API Web DOIT prendre en charge le renvoi du nombre d'objets d'une collection.

[RSG-80] Un paramètre de requête DOIT être utilisé pour prendre en charge le renvoi du nombre d'objets d'une collection.

[RSG-81] Le paramètre de requête `count` DEVRAIT être utilisé pour renvoyer le nombre d'objets d'une collection.

[RSG-82] Une API Web PEUT prendre en charge le renvoi du nombre d'objets d'une collection en l'insérant dans la partie de la réponse qui contient la collection elle-même. Un paramètre de requête DOIT être utilisé.

[RSG-83] Le paramètre de requête `count=true` DEVRAIT être utilisé. S'il n'est pas spécifié, `count` devrait être défini par défaut comme `false`.

[RSG-84] Si une API Web prend en charge la pagination, elle DEVRAIT prendre en charge le renvoi du nombre de la collection (c'est-à-dire le nombre total d'objets de la collection) dans le texte de la réponse.

Expressions de recherche avancée

60. Pour extraire des données avec un petit nombre seulement de critères de recherche, les paramètres de requête suffisent. S'il y a un cas d'utilisation rendant nécessaire la recherche de données à l'aide d'expressions de recherche avancée (avec des critères multiples, des expressions booléennes et des opérateurs de recherche), l'API doit être conçue à l'aide d'un langage de requête plus avancé. Un langage de requête doit être pris en charge par une grammaire de recherche.

61. Le langage de requête contextuel (CQL) est un langage formel pour représenter les interrogations adressées à des systèmes d'extraction d'information comme les moteurs de recherche, les catalogues bibliographiques et les informations sur les collections muséales. Fondé sur la sémantique de Z39.50[8], il a pour objectif de conception que les requêtes doivent pouvoir être lues et écrites et que ce langage est intuitif et conserve l'expression de langages de requête plus avancés. Il s'agit simplement d'une option dont l'utilisation est recommandée, car elle est très utilisée dans l'industrie.

[RSG-85] Lorsqu'une API Web prend en charge des expressions de recherche avancée, un langage de requête, tel que le CQL, DEVRAIT être spécifié.

[RSG-86] Un contrat de service DOIT spécifier la grammaire prise en charge (comme les champs, les fonctions, les mots-clés et les opérateurs).

[RSG-87] Le paramètre de requête "`q`" DOIT être utilisé.

Gestion des erreurs

62. Les réponses-erreurs devraient toujours utiliser le code d'état HTTP approprié choisi dans la liste type des codes d'état HTTP (RFC 7807), reproduite à l'annexe V. Lorsque le requérant prévoit JSON, renvoyer les informations concernant les erreurs sous la forme d'une structure de données commune. À moins que le projet n'en dispose autrement, il est inutile de définir des codes d'erreur pour une application donnée. Dans les environnements de production, les traces de pile et autres informations de débogage ne devraient pas figurer dans le corps de la réponse.

Charge utile d'erreur

63. La gestion des erreurs intervient à deux niveaux : au niveau du protocole (HTTP) et à celui de l'application (charge utile renvoyée). Au niveau du protocole, une API Web renvoie un code d'état HTTP approprié et, au niveau de l'application, une API Web renvoie une charge utile signalant l'erreur de façon suffisamment détaillée (attributs obligatoires et facultatifs).

64. En ce qui concerne les attributs obligatoires et facultatifs pour la gestion des erreurs au niveau de l'application,

a) Les attributs `code` et `message` ci-après sont obligatoires et, tandis que le `message` peut changer à l'avenir, le `code` ne changera pas; il est fixe et renverra toujours à ce problème particulier :

- `code` (nombre entier) – code technique de la situation d'erreur à utiliser à des fins d'appui; et
- `message` (chaîne) – message reçu par l'utilisateur (localisable) décrivant la requête-erreur comme demandé par l'en-tête HTTP `Accept-Language` (voir RSG-114).

b) Les attributs ci-après sont conditionnellement obligatoires :

- `details` – si le traitement des erreurs nécessite l'emboîtement des réponses-erreurs, il doit utiliser le champ de données correspondant. Le champ de données doit contenir un ensemble d'objets JSON qui montre les propriétés des codes et des messages en utilisant la même sémantique que celle qui a été décrite plus haut.

c) Les attributs ci-après sont facultatifs :

- `target` – la structure d'erreur peut contenir un attribut cible qui décrit un élément de données (par exemple, un chemin d'accès à une ressource);
- `status` – doublonner le code d'état HTTP pour le propager le long de la chaîne d'appels ou pour le consigner dans le journal d'assistance sans avoir à ajouter explicitement ce code chaque fois;
- `moreInfo` – ensemble de liens contenant des informations supplémentaires sur la situation d'erreur, par exemple en donnant des conseils à l'utilisateur; et
- `internalMessage` – message technique, par exemple à des fins de connexion.

[8] Voir le chapitre relatif aux références.

65. La gestion des erreurs devrait respecter les normes HTTP (RFC 2616). Une charge utile d'erreur minimale est recommandée :

– Par exemple, les réponses HTTP ci-après sont renvoyées lorsque le numéro d'enregistrement international fourni n'a pas permis de trouver la marque correspondante :

Exemple avec une charge utile XML basée sur la norme ST.96

```
GET /api/v1/trademarks?irn=000000000000001John%20Smith&expiryDate=2018-12-31.
HTTP/1.1
Host: wipo.int
Accept: application/xml
```

Voici un exemple de réponse HTTP renvoyée :

```
HTTP/1.1 404
Content-Type: application/xml
 <?xml version="1.0" encoding="UTF-8"?>
<com:TransactionError xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:com="http://www.wipo.int/standards/XMLSchema/ST96/Common"
xsi:schemaLocation="http://www.wipo.int/standards/XMLSchema/ST96/Common
TransactionError.xsd">
      <com:TransactionErrorCode>TRADEMARK_NOT_FOUND</com:TransactionErrorCode>
      <com:TransactionErrorText>The trademark with the provided International
Registration Number was not found</com:TransactionErrorCode>
</com:TransactionError>
```

Exemple avec une charge utile JSON sur la base de la norme ST.97

```
HTTP/1.1 404
Content-Type: application/json
{

      "transactionError": [
            {
                  "transactionErrorCode": " TRADEMARK_NOT_FOUND ",
            },
            {
                  "transactionErrorText": " The trademark with the provided
International Registration Number was not found
            },
      ]
}


 "
            },
      ]
}
```

[RSG-88] Au niveau du protocole, une API Web DOIT renvoyer un code d'état HTTP approprié choisi dans la liste type des codes d'état HTTP.

[RSJ-89] Au niveau de l'application, une API Web DOIT renvoyer une charge utile signalant l'erreur de façon suffisamment détaillée. Les attributs `code` et `message` sont obligatoires, l'attribut `details` est conditionnellement obligatoire et les attributs `target`, `status`, `moreInfo` et `internalMessage` sont facultatifs.

[RSG-90] Les erreurs NE DOIVENT PAS exposer les données dont la sécurité est primordiale ou les données techniques internes, comme les piles d'appels, dans les réponses-erreurs.

[RSG-91] L'en-tête HTTP `Header: Reason-Phrase` (décrit dans le RFC 2616) NE DOIT PAS être utilisé pour transmettre des réponses-erreurs.

Identifiant de corrélation

66. En règle générale, la consommation d'un service déclenche celle de multiples autres services. Il devrait exister un mécanisme de corrélation de toutes les activations de services dans le même contexte d'exécution. Il pourrait, par exemple, consister à insérer l'identifiant de corrélation dans les messages enregistrés, car cela permet d'identifier de manière unique l'erreur enregistrée. Un nom d'en-tête devrait être utilisé – on utilise couramment, par exemple, un identifiant de requête ou un identifiant de corrélation – car si ce point est pris en considération dans la phase de conception d'une API, la compatibilité ultérieure entre les différentes API et les implémentations nouvelles sera favorisée.

[RSG-92] Chaque erreur enregistrée DEVRAIT avoir son propre identifiant de corrélation. Un en-tête HTTP personnalisé DEVRAIT être utilisé et DEVRAIT être nommé identifiant de corrélation.

Contrat de service

67. REST n'est pas un protocole ni une architecture, mais un style architectural avec des propriétés architecturales et des contraintes architecturales. Il n'existe pas de contrat type officiel pour les API REST. La présente norme désigne la documentation API sous l'appellation de contrat de service REST. Le contrat de service s'appuie sur les trois éléments fondamentaux suivants :

k) Syntaxe de l'identificateur de ressource – comment exprimer la destination ou la provenance des données transférées?
l) Méthodes – quels sont les mécanismes de protocole utilisés pour transférer les données?
m) Types de média – quel type de donnée est transféré? Les différents services REST utilisent ces éléments dans différentes combinaisons pour exposer leurs capacités. La définition d'un ensemble maître de ces éléments à utiliser par une collection (ou inventaire) de services rend ce type de contrat de service "uniforme".

[RSG-93] Un format de contrat de service DOIT comprendre les éléments ci-après :

- Version de l'API;
- Informations sur la sémantique des éléments de l'API;
- Ressources;
- Attributs des ressources;
- Paramètres de requête;
- Méthodes;
- Types de médias;
- Grammaire de recherche (si une telle grammaire est prise en charge);
- Codes d'état HTTP;
- Méthodes HTTP;
- Restrictions et éléments distinctifs; et
- Sécurité (par exemple, schémas privés).

[RSG-94] Un format de contrat de service DEVRAIT comprendre les requêtes et les réponses au schéma XML ou au schéma JSON, ainsi que des exemples d'utilisation de l'API dans les formats pris en charge, c'est-à-dire XML ou JSON.

[RSG-95] Une API REST DOIT fournir la documentation API en tant que contrat de service.

[RSG-96] Une implémentation de l'API Web qui s'écarterait de la présente norme DOIT être explicitement documentée dans le contrat de service. Si une règle dérogeant à la norme n'est pas spécifiée dans le contrat de service, on DOIT présumer que cette norme est appliquée.

[RSG-97] Un contrat de service DOIT permettre de générer le code squelette du client de l'API.

[RSG-98] Un contrat de service DEVRAIT permettre de générer le code squelette du serveur.

68. La documentation de l'API Web peut être écrite, par exemple, en utilisant le langage de modélisation API RESTful (RAML), la spécification d'API ouverte (OAS) et le langage de description de services Web (WSDL). Étant donné que seul le RAML prend pleinement en charge la validation des requêtes et des réponses aux formats XML et JSON (en utilisant les schémas XSD et les schémas JSON), la présente norme recommande le RAML[9].

[RSG-99] Une documentation API Web DEVRAIT être écrite en RAML ou OAS. Les formats de documentation personnalisés NE DEVRAIENT PAS être utilisés.

Pause

69. En vertu des principes de la conception orientée services, l'utilisation des serveurs devrait être limitée dans le temps.

[RSG-100] Un consommateur d'API Web DEVRAIT pouvoir spécifier une pause de serveur pour chaque requête; un en-tête HTTP personnalisé DEVRAIT être utilisé. De plus, une pause de serveur maximale DEVRAIT être utilisée afin de protéger les ressources du serveur contre la surutilisation.

Gestion des états

70. Si le développement respecte les principes REST, la gestion des états doit être prise en charge par le client, et non par le serveur, puisque les API REST sont sans états. Si, par exemple, plusieurs serveurs exécutent une session, la réplication devrait être découragée.

Versionnage des réponses

71. Le fait d'extraire à de multiples reprises le même ensemble de données peut entraîner une consommation de bande passante si l'ensemble de données n'a pas été modifié entre les requêtes. Les données ne devraient être conditionnellement extraites que si elles n'ont pas été modifiées. Cela peut être fait au moyen de la validation des ressources en fonction du contenu ou de la validation des ressources en fonction du temps. S'il utilise le versionnage des réponses, un consommateur de service peut mettre en œuvre un verrouillage optimiste.

[RSG-101] Une API Web DEVRAIT prendre en charge l'extraction conditionnelle de données, afin que seules les données qui sont modifiées soient extraites. La validation des ressources en fonction du contenu DEVRAIT être utilisée car elle est plus précise.

[RSG-102] Afin d'exécuter la validation des ressources en fonction du contenu, l'en-tête HTTP `ETag` DEVRAIT être utilisé dans la réponse pour coder l'état des données. Ensuite, cette valeur DEVRAIT être utilisée dans les requêtes ultérieures dans les en-têtes HTTP conditionnels (comme If-Match ou If-None-Match). Si les données n'ont pas été modifiées depuis que la requête a renvoyé le `ETag`, le serveur DEVRAIT renvoyer le code d'état "`304 Not Modified`" (en l'absence de modifications). Ce mécanisme est spécifié dans les RFC 7231 et 7232 de l'IETF.

[RSG-103] Afin d'exécuter la validation des ressources en fonction du temps, l'en-tête HTTP `Last-Modified` DEVRAIT être utilisé. Ce mécanisme est spécifié dans les RFC 7231 et 7232 de l'IETF.

[9] L'OAS est une spécification. Elle prend également en charge Markdown, ce qui n'est pas le cas du RAML. D'un autre côté, bien que l'OAS et le RAML prennent en charge la validation par le schéma JSON pour les requêtes et les réponses, l'OAS ne prend pas en charge les schémas XSD. À l'avenir, l'OAS pourra donc être recommandée lorsque ses fonctionnalités seront complètes.

[RSG-104] En utilisant le versionnage des réponses, un consommateur de service PEUT mettre en œuvre un verrouillage optimiste.

Mise en antémémoire

72. L'implémentation d'une API Web devrait prendre en charge la mise en antémémoire afin d'économiser de la bande passante, conformément au RFC 7234 de l'IETF.

[RSG-105] Une API Web DOIT prendre en charge la mise en antémémoire des résultats de `GET`; une API Web PEUT prendre en charge la mise en antémémoire des résultats d'autres méthodes HTTP.

[RSG-106] Les en-têtes de réponse HTTP `Cache-Control` et `Expires` DEVRAIENT être utilisés. Ce dernier en-tête PEUT être utilisé pour prendre en charge les anciens clients.

Encadrement du transfert de fichiers

73. Le transfert (c'est-à-dire le téléchargement en aval ou en amont) de gros fichiers présente une forte probabilité d'interruption du réseau ou d'une autre défaillance de transmission. De plus, il consomme une grande quantité de mémoire tant pour le prestataire que pour le consommateur du service. Il est donc recommandé de transférer les fichiers volumineux en plusieurs morceaux en envoyant plusieurs demandes. Cette option indique également l'état d'avancement du téléchargement aval ou amont. Le transfert partiel de gros fichiers devrait reprendre la prise en charge. Le prestataire devrait faire savoir s'il prend en charge le transfert partiel de gros fichiers[10].

74. Ce type de transfert peut se faire de deux manières : la première consiste à utiliser un en-tête `Transfer-Encoding: chunked` et la seconde à utiliser l'en-tête `Content-Length`. Ces en-têtes ne devraient pas être utilisés ensemble. `Content-Length` indique la taille totale du fichier transféré; le destinataire connaîtra donc la longueur du corps et pourra estimer le temps dé téléchargement aval. L'en-tête `Transfer-Encoding: chunked` est utile pour transmettre en continu des données limitées à l'infini, telles que les données audio ou vidéo, mais pas les fichiers. Il est recommandé d'utiliser l'en-tête `Content-Length` pour le téléchargement aval car l'utilisation du serveur est faible par rapport à l'en-tête `Transfer-Encoding: chunked`. Pour un téléchargement amont, l'en-tête `Transfer-Encoding: chunked` est recommandé.

Une API Web devrait faire savoir si elle prend en charge les téléchargements partiels de fichiers en répondant aux requêtes `HEAD` et en envoyant les en-têtes de réponse HTTP : `Accept-Ranges` et `Content-Length`. Le premier devrait indiquer l'unité pouvant être utilisée pour définir une plage et ne devrait jamais être défini comme "aucune". Le second indique la taille totale du fichier à télécharger.

[RSG-107] Une API Web DEVRAIT faire savoir si elle prend en charge les téléchargements partiels de fichiers en répondant aux requêtes `HEAD` et en envoyant les en-têtes de réponse HTTP `Accept-Ranges` et `Content-Length`.

75. Selon le RFC 7232 de l'IETF, une API Web qui prend en charge le téléchargement de gros fichiers devrait prendre en charge les requêtes de téléchargement partiel, c'est-à-dire que :

- le consommateur de service demandant une plage devrait utiliser l'en-tête HTTP `Range`;
- la réponse du prestataire de service devrait contenir les en-têtes HTTP `Content-Range` et `Content-Length`; et
- la réponse du prestataire de service devrait comprendre l'état HTTP "`206 Partial Content`" en cas d'acceptation de la demande de plage. Si une demande de plage dépasse les limites (les valeurs de la plage débordent les limites de la ressource), le serveur répond par l'état "`416 Requested Range Not Satisfiable`". Si la plage demandée n'est pas prise en charge, le serveur renvoie l'état "`200 OK`".

[RSG-108] Une API Web DEVRAIT prendre en charge les téléchargements partiels de fichiers. Les plages à plusieurs parties DEVRAIENT être prises en charge.

[10] Le prestataire de service peut renvoyer l'emplacement du fichier, puis le consommateur de service peut appeler un service d'annuaire pour télécharger le fichier. À la fin, un téléchargement partiel du fichier est requis. Ce paragraphe ne prend pas en considération les protocoles autres que REST, tels que FTP, sFTP ou rsync.

76. Des plages à plusieurs parties peuvent également être demandées si l'en-tête HTTP `Content-Type: multipart/byteranges; boundary=XXXXX` est utilisé. Une demande de plage peut être conditionnelle si elle est associée aux en-têtes HTTP `ETag` ou `If-Range`.

77. Aucun RFC de l'IETF ne porte sur le téléchargement amont de gros fichiers. Aussi la présente norme ne fournit-elle aucune recommandation d'exécution concernant les téléchargements amont de fichiers volumineux.

[RSG-109] Une API Web DEVRAIT faire savoir si elle prend en charge les téléchargements amont partiels de fichiers.

[RSG-110] Une API Web DEVRAIT prendre en charge les téléchargements amont partiels de fichiers. Les plages à plusieurs parties DEVRAIENT être prises en charge.

78. Le RFC 2616 de l'IETF n'impose aucune limite de taille spécifique pour les demandes. Le contrat de service de l'API devrait stipuler la limite maximale des requêtes. De plus, pendant l'exécution, le prestataire de service devrait indiquer au consommateur si la limite maximale autorisée a été dépassée.

[RSG-111] Si la requête a dépassé la limite maximale autorisée, le prestataire de services DEVRAIT renvoyer avec les en-têtes de réponse HTTP l'en-tête HTTP "`413 Request Entity Too Large`". Un en-tête HTTP personnalisé PEUT être utilisé pour indiquer la taille maximale de la réponse.

Gestion des préférences

79. Un prestataire de services peut autoriser un consommateur à configurer des valeurs et à influencer la manière dont le premier traite les requêtes du second. Le RFC 7240 de l'IETF décrit un moyen standard d'implémenter la gestion des préférences.

[RSG-112] Si une API Web prend en charge la gestion des préférences, celle-ci DEVRAIT être implémentée conformément au RFC 7240 de l'IETF, c'est-à-dire que l'en-tête de requête HTTP `Prefer` DEVRAIT être utilisé et l'en-tête de réponse HTTP `Preference-Applied` DEVRAIT être renvoyé (confirmant la requête originelle).

[RSG-113] Si une API Web prend en charge la gestion des préférences, la liste des préférences qui PEUT être constituée en utilisant l'en-tête `Prefer` DOIT être enregistrée dans le contrat de service.

Traduction

80. Un consommateur de service peut demander à recevoir des réponses dans une certaine langue si le prestataire la prend en charge. On trouvera une spécification normalisée pour la prise en charge d'une série de langues naturelles dans le RFC 7231 de l'IETF.

[RSG-114] Si une API Web prend en charge les données localisées, l'en-tête de requête HTTP `Accept-Language` DOIT, conformément au RFC 7231 de l'IETF, être pris en charge pour indiquer la série de langues naturelles qui sont préférées dans la réponse.

Opérations de longue durée

81. Il y a des cas où une API Web peut donner lieu à des opérations de longue durée. Par exemple, la création d'un PDF par le prestataire de services peut prendre plusieurs minutes. Le présent paragraphe recommande un modèle courant d'échange de messages pour implémenter ces cas. Par exemple :

```
// (a)
GET https://wipo.int/api/v1/patents
Accept: application/pdf
…
// (b)
HTTP/1.1 202 Accepted
Location: https://wipo.int/api/v1/queues/12345
```

```
…
// (c1)
GET https://wipo.int/api/v1/queues/12345
…
HTTP/1.1 200 OK
…
// (c2)
GET https://wipo.int/api/v1/queues/12345
HTTP/1.1 303 See Other
Location: https://wipo.int/api/v1/path/to/pdf
…
// (c3)
GET https://wipo.int/api/v1/path/to/pdf
…
```

82. Si une API prend en charge les opérations de longue durée, celles-ci devraient être exécutées de manière asynchrone pour que l'utilisateur n'ait pas à attendre une réponse. La règle ci-après énonce un mode d'implémentation recommandé.

[RSG-115] Si l'API prend en charge les opérations de longue durée, celles-ci DEVRAIENT être asynchrones. Le mode d'implémentation ci-après DEVRAIT être appliqué :

a) le consommateur active le fonctionnement du service;
b) le service renvoie le code d'état "202 Accepted" conformément au RFC 7231 de l'IETF (section 6.3.3), c'est-à-dire que la requête a été acceptée pour traitement, mais celui-ci n'est pas terminé. L'emplacement de la tâche dans la file d'attente qui a été créé est également renvoyé avec l'en-tête HTTP Location; et
c) le consommateur appelle le Location renvoyé pour savoir si la ressource est disponible. Si elle ne l'est pas, la réponse DEVRAIT comporter le code d'état "200 OK", contenir l'état d'avancement de la tâche (par exemple, en attente d'exécution) et PEUT contenir d'autres informations (par exemple, un indicateur de déroulement ou un lien permettant d'annuler ou de supprimer la tâche à l'aide de la méthode HTTP DELETE). Si la ressource est disponible, la réponse DEVRAIT comporter le code d'état "303 See Other", et l'en-tête HTTP Location DEVRAIT contenir l'adresse URL pour extraire les résultats de la tâche.

Modèle de sécurité

Règles générales

83. Dans le champ d'application de la présente norme, la sécurité de l'API implique des attributs de sécurité essentiels qui doivent garantir que les informations accessibles par une API et les API elles-mêmes sont sécurisées tout au long de leur cycle de vie. Ces attributs sont la confidentialité, l'intégrité, la disponibilité, la confiance, la non-répudiation, la compartimentation, l'authentification, l'autorisation et la vérification.

[RSG-116] Confidentialité : les API et les informations sur les API DOIVENT être identifiées, classées et protégées à tout moment contre l'accès et la divulgation non autorisés et contre les écoutes. Les principes du moindre privilège, de l'absence de confiance, du besoin de connaître et du besoin de partager[11] DOIVENT être respectés.

[RSG-117] Intégrité : les API et les informations sur les API DOIVENT être protégées contre toute modification, reproduction, altération et destruction non autorisées. Les informations DOIVENT être modifiées au moyen de transactions et d'interfaces approuvées. Les systèmes DOIVENT être mis à jour en lançant des processus approuvés de gestion de configuration, de gestion des modifications et de gestion des correctifs.

[RSG-118] Disponibilité : les API et les informations sur les API DOIVENT être disponibles pour les utilisateurs autorisés au moment voulu, selon les stipulations des accords de niveau de service, les politiques de contrôle d'accès et les processus opérationnels définis.

[RSG-119] Non-répudiation : Chaque transaction traitée ou action exécutée par les API DOIT veiller au respect de la non-répudiation en mettant en œuvre des processus de vérification, d'autorisation et d'authentification, en sécurisant les chemins d'accès et en mettant en place des services et mécanismes de non-répudiation.

11 https://www.owasp.org/index.php/Security_by_Design_Principles

[RSG-120] Authentification, autorisation, vérification : Les utilisateurs, les systèmes, les API ou les dispositifs associés à des transactions ou actions critiques DOIVENT être authentifiés et autorisés à l'aide de services de contrôle d'accès basés sur des rôles ou des attributs, et maintenir la séparation des tâches. En outre, toutes les actions DOIVENT être enregistrées et le poids de l'authentification doit augmenter avec le risque en matière de sécurité de l'information.

Directives concernant une gestion des API sécurisée et capable de faire face aux menaces

84. Les API devraient être conçues, construites, testées et exécutées en gardant à l'esprit les obligations et les risques en matière de sécurité. Les parades et contrôles appropriés devraient être directement intégrés à la conception, et non ajoutés après coup. Il est recommandé d'utiliser les meilleures pratiques et normes, telles que l'OWASP (Open Web Application Security Project).

[RSG-121] Pendant la mise au point des API, les menaces, les cas d'usage malveillant, les techniques sûres de codage, la sécurité de la couche transport et les tests de sécurité DOIVENT faire l'objet d'une attention particulière, notamment en ce qui concerne :

- Les `PUTs` et les `POSTs` – à savoir : quelles modifications des données internes pourraient être utilisées pour attaquer ou induire en erreur;
- `DELETES` – à savoir : pourraient être utilisé pour supprimer le contenu d'un référentiel de ressources internes;
- Méthodes autorisées d'établissement d'une liste blanche – pour s'assurer que les méthodes HTTP autorisées sont légitimement restreintes tandis que les autres renverraient un code de réponse légitime; et
- Les attaques les plus courantes devraient être prises en considération pendant la phase de la conception qui porte sur la modélisation des menaces de façon que le risque n'augmente pas. Les menaces et les mesures d'atténuation définies dans l'aide-mémoire de l'OWASP Top Ten (OWASP Top Ten Cheat Sheet)[12] DOIVENT être prises en considération.

[RSG-122] Lors de la mise au point des API, les normes et meilleures pratiques énumérées ci-après DEVRAIENT être appliquées :

- Meilleures pratiques de codage sûres : principes de codage sûrs de l'OWASP;
- Sécurité des API REST : aide-mémoire sur la sécurité REST;
- Aseptisation des entrées et protection contre l'injection de code indirecte (XSS) : aide-mémoire XSS de l'OWASP;
- Prévention de l'injection SQL : aide-mémoire de l'OWASP sur l'injection SQL, aide-mémoire de l'OWASP sur le paramétrage; et
- Sécurité de la couche transport : aide-mémoire de l'OWASP sur la protection de la couche transport.

[RSG-123] Les tests de sécurité et l'appréciation de la vulnérabilité DOIVENT être réalisés pour garantir la sécurité des API et leur résistance aux menaces. Cette exigence PEUT être satisfaite en utilisant les tests statiques et dynamiques de sécurité des applications (SAST/DAST), les outils automatisés de gestion des vulnérabilités et les tests de pénétration.

Chiffrement, intégrité et non-répudiation

85. Les services protégés doivent être sécurisés afin de protéger les données d'authentification en transit : par exemple, les mots de passe, les clés API ou les jetons Web JSON. L'intégrité des données transmises et la non-répudiation des actions exécutées devraient également être assurées. Des mécanismes cryptographiques sûrs peuvent garantir la confidentialité, le chiffrement, l'intégrité et la non-répudiation. La parfaite sécurité itérative est l'un des moyens utilisés pour que les clés de session ne puissent pas être prédites.

[RSG-124] Les services protégés DOIVENT ne fournir que des points de terminaison HTTPS qui utilisent le TLS 1.2, ou une version plus récente, avec une suite de chiffres qui comprend le protocole d'échange de clés ECDHE.

[12] https://www.owasp.org/index.php/Top_10-2017_Top_10

[RSG-125] En ce qui concerne les protocoles d'authentification, la parfaite sécurité itérative DEVRAIT être utilisée pour assurer la sécurité du transport. L'utilisation d'algorithmes cryptographiques non sûrs et la rétrocompatibilité avec les protocoles SSL 3 et TLS 1.0/1.1 NE DEVRAIENT PAS être autorisées.

[RSG-126] Pour une sécurité et une confiance maximales, un réseau privé virtuel sous protocole de sécurité IPSec DEVRAIT être installé de site à site pour mieux protéger les informations transmises sur des réseaux non sécurisés.

[RSG-127] L'application consommatrice DEVRAIT valider la chaîne de certificats TLS au moment de demander l'accès à des ressources protégées, notamment en consultant la liste de révocation de certificats.

[RSG-128] Les services protégés DEVRAIENT utiliser uniquement des certificats valides émis par une autorité de certification digne de confiance.

[RSG-129] Les jetons DEVRAIENT être signés à l'aide d'algorithmes de signature sécurisés qui soient conformes à la norme de signature numérique FIPS –186-4. L'algorithme de signature numérique RSA ou l'algorithme ECDSA DEVRAIENT être pris en considération.

Authentification et autorisation

86. L'autorisation s'entend de l'acte consistant à contrôler l'accès à une ressource. L'autorisation concerne non seulement l'application de mesures de contrôle d'accès, mais aussi la définition de ces mesures. Il s'agit notamment des règles et politiques d'accès, qui devraient définir le niveau requis d'accès acceptable tant pour le prestataire de service que pour l'application consommatrice. Le contrôle de l'accès repose sur le fait que le prestataire accorde ou refuse à une application consommatrice ou à un consommateur l'accès à une ressource à un certain degré de détail. Un accès peu détaillé devrait être envisagé au point de requête de l'API ou de la passerelle de l'API, tandis qu'un contrôle détaillé devrait être envisagé au niveau du service dorsal, si possible. Le modèle "contrôle d'accès à base de rôles" (RBAC) ou "contrôle d'accès basé sur les attributs" (ABAC) peut être pris en considération.

87. Si un service est protégé, le standard d'identification Open ID Connect devrait être préféré à OAuth 2.0 car il comble un grand nombre des failles de ce dernier et normalise la manière d'obtenir les données relatives au profil du propriétaire de la ressource : le format et la cryptographie normalisés des jetons Web JSON (JWT). Les autres systèmes de sécurité, comme l'Authentification de base HTTP, qui exige que le client conserve un mot de passe quelque part en texte clair pour l'envoyer avec chaque requête, ne devraient pas être utilisés. De même, la vérification de ce mot de passe serait ralentie car elle devrait accéder au magasin d'authentifiants. OAuth 2.0 ne spécifie pas le jeton de sécurité. Il s'ensuit que le jeton JWT devrait être utilisé plutôt, par exemple, que le SAML 2.0, qui est plus verbeux.

[RSG-130] L'authentification anonyme DOIT n'être utilisée que lorsque les clients et l'application qu'ils utilisent accèdent à des informations ou des fonctionnalités d'un faible niveau de sensibilité qui ne devraient pas exiger une authentification, comme les informations publiques.

[RSG-131] Une authentification par identifiant et mot de passe ou par hachage de mot de passe NE DOIT PAS être autorisée.

[RSG-132] Si un service est protégé, Open ID Connect DEVRAIT être utilisé.

[RSG-133] Lorsqu'un jeton Web JSON (JWT) est utilisé, un secret de JWT DEVRAIT posséder un degré élevé d'entropie afin d'augmenter le facteur de travail d'une attaque par force brute; les jetons TTL et RTTL DEVRAIENT être aussi courts que possible; et les informations sensibles NE DEVRAIENT PAS être stockées dans la charge utile du JWT.

88. En matière de conception de sécurité, on choisit couramment de centraliser l'authentification de l'utilisateur. Elle devrait être stockée dans un fournisseur d'identité (IdP) ou, localement, aux points de terminaison REST.

89. Les services devraient veiller à prévenir toute divulgation d'authentifiants. Les mots de passe, les jetons de sécurité et les clés d'API ne devraient pas apparaître dans l'adresse URL, car ils peuvent être saisis dans les journaux de serveur Web, ce qui confère à ces derniers un intérêt intrinsèque. Par exemple, ce qui suit est incorrect (clé d'API dans l'URL) : https://wipo.int/api/patents?apiKey=a53f435643de32.

[RSG-134] Dans les requêtes `POST/PUT`, les données sensibles DEVRAIENT être transférées dans le corps de la requête ou par des en-têtes de requête.

[RSG-135] Dans les requêtes `GET`, les données sensibles DEVRAIENT être transférées dans un en-tête HTTP.

[RSG-136] Afin de limiter au minimum le temps d'attente et de réduire le couplage entre services protégés, la décision de contrôle d'accès DEVRAIT être prise localement par les points de terminaison REST.

90. Authentification par clés API : les clés d'API devraient être utilisées chaque fois qu'une authentification système à système est exigée et elles devraient être générées de façon automatique et aléatoire. Ce mode d'authentification présente le risque inhérent que la possession d'une copie de la clé API permette de l'utiliser en se faisant passer pour l'application consommatrice légitime. Il s'ensuit que toutes les communications devraient se conformer à [RSG-124], pour protéger la clé en transit. Il incombe au développeur de l'application de protéger comme il convient sa copie de la clé API. Si la clé API est incorporée dans l'application consommatrice, elle peut être décompilée et extraite. Si elle est stockée dans des fichiers en clair, elle peut être volée et réutilisée à des fins malveillantes. Une clé API doit donc être protégée par un magasin d'authentifiants ou un mécanisme de gestion des secrets. Les clés API peuvent être utilisées pour contrôler l'utilisation de services même dans le cas de services publics.

[RSG-137] Les clés API DEVRAIENT être utilisées pour les services protégés et les services publics afin d'empêcher que leur prestataire de services ne soit submergé par des requêtes multiples (attaques par déni de service). S'agissant des services protégés, les clés API PEUVENT être utilisées à des fins de monétisation (plans achetés), d'application de règlements d'utilisation (QoS, qualité de service) et de contrôle.

[RSG-138] Comme l'indique le RFC 7231 de l'IETF, les clés API PEUVENT être combinées avec l'utilisateur ou l'agent de l'en-tête de requête HTTP pour distinguer entre un utilisateur humain et un agent intelligent.

[RSG-139] Le prestataire de service DEVRAIT renvoyer l'état d'utilisation actuelle en même temps que les en-têtes de réponse HTTP. Les données de réponse ci-après PEUVENT être renvoyées :

- limite de débit – limite de débit (par minute) fixée dans le système;
- limite de débit restante – nombre restant de requêtes autorisé pendant la tranche de temps actuelle (-1 indique que la limite a été dépassée); et
- remise à zéro de la limite de débit – durée (en secondes) restant à courir jusqu'à la remise à zéro du compteur de requêtes.

[RSG-140] Le prestataire de service DEVRAIT renvoyer le code d'état "`429 Too Many Requests`" si les requêtes lui sont envoyées trop rapidement.

[RSG-141] Les clés API DOIVENT être révoquées si le client viole l'accord d'utilisation, comme prescrit par l'Office de propriété intellectuelle.

[RSG-142] Les clés API DEVRAIENT être transférées à l'aide d'en-têtes HTTP personnalisés. Elles NE DEVRAIENT PAS l'être à l'aide de paramètres de requête.

[RSG-143] Les clés API DEVRAIENT être générées de façon aléatoire.

91. Si l'utilisation de la cryptographie par clé publique et des certificats de clé publique occasionne des coûts indirects, l'authentification réciproque par certificat n'en devrait pas moins être utilisée lorsqu'une API Web exige une authentification plus forte que celle qu'offrent les clés API, afin de fournir une sécurité supplémentaire. Des certificats sécurisés et fiables doivent être émis par une autorité de certification qui a la confiance des deux parties dans le cadre d'un processus d'instauration de la confiance ou d'une certification croisée. Afin de réduire les risques pour la sécurité des données d'identification qui sont particuliers aux systèmes sensibles et aux actions protégées, une authentification forte peut être utilisée. Les certificats partagés entre le client et le serveur, p. ex. X.509, devraient être utilisés.

[RSG-144] Des certificats sécurisés et fiables DOIVENT être émis par une autorité de certification qui a la confiance des deux parties dans le cadre d'un processus d'instauration de la confiance ou d'une certification croisée.

[RSG-145] Les certificats partagés entre le client et le serveur, p. ex. le X.509, DEVRAIENT être utilisés afin de réduire les risques pour la sécurité des données d'identification qui sont particuliers aux systèmes sensibles et aux actions protégées.

[RSG-146] Pour les services hautement protégés, une authentification bidirectionnelle entre le client et le serveur DEVRAIT utiliser des certificats pour fournir une sécurité supplémentaire.

[RSG-147] Une authentification multifacteurs DEVRAIT être implémentée afin de réduire les risques pour la sécurité des données d'identification qui sont particuliers aux applications présentant un profil de risque élevé, aux systèmes traitant des informations très sensibles ou aux actions protégées.

Disponibilité et protection contre les menaces

92. Dans ce contexte, la disponibilité couvre la protection contre les menaces afin de réduire au minimum le temps d'indisponibilité, en étudiant comment les menaces contre les API exposées peuvent être atténuées en mettant en œuvre les principes de conception de base. La disponibilité implique également de dimensionner les capacités pour répondre à la demande et de garantir la stabilité des environnements d'hébergement, etc. Ces niveaux de disponibilité sont pris en charge par les piles de matériels et de logiciels qui appuient la mise en œuvre des API. En principe, la question de la disponibilité est traitée au regard des normes de continuité des opérations et de reprise des opérations après un sinistre, qui recommandent une approche fondée sur l'évaluation des risques pour définir les exigences de disponibilité.

Requêtes interdomaines

93. Certaines requêtes "interdomaines", en particulier les requêtes Ajax, sont interdites par défaut par la politique de sécurité de même origine. En vertu de la politique de même origine, un navigateur Web permet à des scripts d'une première page Web d'accéder à des données d'une deuxième page Web uniquement si les deux pages ont la même origine (c'est-à-dire la même combinaison de plan d'URI, de nom d'hôte et de numéro de port).

94. Le Cross-Origin Resource Sharing (partage de ressources entre origines multiples)(CORS) est une norme W3C qui désigne de façon souple les requêtes interdomaines autorisées. En fournissant des en-têtes HTTP CORS appropriés, une API REST indique au navigateur quels domaines ou origines peuvent réaliser des appels en JavaScript au service REST.

95. Le JSON with padding (JSON avec formatage) (JSONP) est une méthode servant à envoyer des données JSON sans avoir à se préoccuper des questions liées aux requêtes interdomaines. Elle introduit des fonctions de rétroappel pour le chargement de données JSON provenant de domaines différents. Son principe est basé sur le fait que la balise `<script>` HTML n'est pas concernée par la politique de même origine. Tout ce qui est importé par l'intermédiaire de cette balise est immédiatement exécuté dans le contexte global. Au lieu d'utiliser un fichier JavaScript, on peut utiliser une adresse URL pour interroger un service qui renvoie un code JavaScript.

96. Les méthodes couramment employées pour contourner cette restriction sont les suivantes :

- JSONP est un palliatif pour les requêtes interdomaines. Il ne propose aucun mécanisme de détection d'erreur, c'est-à-dire que si un problème survient et que le service tombe en panne ou réponde avec un message d'erreur HTTP, le client n'a aucun moyen de savoir ce qui s'est passé. Et l'application AJAX n'aura plus qu'à "poireauter". De plus, le site qui utilise JSONP fera inconditionnellement confiance au JSON fourni depuis un domaine différent;
- Iframe est une autre solution de contournement pour les requêtes interdomaines. En appliquant la méthode JavaScript `window.postMessage (message, targetOrigin)` à l'objet iframe, il est possible d'envoyer une requête à un site d'un domaine différent. La solution Iframe présente une bonne compatibilité, même dans les navigateurs anciens. De plus, elle ne prend en charge que GET. La source de la page Iframe devrait toujours être vérifiée en raison de problèmes de sécurité; et
- CORS est une méthode normalisée pour un appel à un domaine externe. Elle peut utiliser `XMLHttpRequest` pour envoyer et recevoir des données et son mécanisme de gestion des erreurs est meilleur que celui de JSONP. Elle prend en charge un plus grand nombre de types d'autorisations que JSONP, qui ne prend en charge que les mouchards. Elle prend également en charge les méthodes HTTP, alors que JSONP ne prend en charge que `GET`. D'un autre côté, il n'est pas toujours possible d'exécuter CORS car les navigateurs doivent prendre en charge cette méthode et les consommateurs d'API doivent être inscrits sur sa liste blanche.

[RSG-148] Si l'API REST est publique, l'en-tête HTTP `Access-Control-Allow-Origin` DOIT être réglé sur "*".

[RSG-149] Si l'API REST est protégée, CORS DEVRAIT être utilisé, si possible. À défaut, JSONP PEUT être utilisé comme solution de repli, mais seulement pour les requêtes GET, par exemple, lorsque l'utilisateur utilise un navigateur ancien. Iframe NE DEVRAIT PAS être utilisé.

Modèle de maturité d'API

97. Il est courant de classer une API REST à l'aide d'un modèle de maturité. Il existe différents modèles de ce type, mais la présente norme se réfère au modèle de maturité de Richardson (RMM). Le RMM définit trois niveaux et la présente norme recommande le niveau 2 pour les API REST car le niveau 3 est d'une exécution complexe et nécessite de la part des prestataires de services comme des consommateurs d'importants investissements en matière de conception et de mise en point. En même temps, il ne procure pas un avantage immédiat aux consommateurs de service.

98. Si une API Web implémente le niveau 3 du RMM, un format hypermédia doit être mis en place. L'Hypertext Application Language (HAL)[13] est simple et est compatible avec les réponses JSON et XML. Toutefois, ce n'est qu'un projet de recommandation, à côté d'autres formats hypermédia, comme JSON-LD[14]. JSON-Schema[15] devrait être utilisé car, étant donné qu'il n'existe actuellement pas de spécification concernant le niveau 3 du RMM, ce schéma est considéré comme le plus développé. Les formats hypermédia ci-après ne devraient pas être pris en considération : RFC 5988 de l'IETF et `Collection+JSON`.

99. Il est recommandé que les cas décrits par un schéma fournissent un lien vers un JSON Schema téléchargeable en utilisant la liaison "`describedby`", définie par le Linked Data Protocol 1.0, section 8.1 [W3C.REC-ldp-20150226][16].

Dans le HTTP, ces liens peuvent être attachés à toute réponse à l'aide de l'en-tête `Link` [RFC8288]. Cet en-tête pourrait par exemple être le suivant :

```
Link: <http://example.com/my-hyper-schema#>; rel="describedby"
```

[RSJ-150] Si on utilise des cas décrits par un schéma, l'en-tête `Link` DEVRAIT être utilisé pour fournir un lien vers un schéma JSON téléchargeable SELON LE RFC 8288.

[RSJ-151] Une API Web DEVRAIT implémenter au moins le niveau 2 (Transport Native Properties) du RMM. Le niveau 3 (Hypermédia) PEUT être implémenté pour rendre l'API autodécouvrable.

100. Un format hypermédia personnalisé peut être mis au point. Dans ce cas, un ensemble d'attributs est recommandé. Par exemple :

```
{
  "link": {
     "href": "/patents",
     "rel": "self"
  },
   ...
}
```

[RSJ-152] Pour mettre au point un format hypermédia personnalisé, l'ensemble d'attributs ci-après DEVRAIT être utilisé en les imbriquant dans un lien d'attributs :

- `href` – l'URI cible;
- `rel` – la signification de l'URI cible;
- `self` – l'URI renvoie à la ressource elle-même;
- `next` – l'URI renvoie à la page précédente (s'il est utilisé pendant la pagination);

13 https://tools.ietf.org/html/draft-kelly-json-hal-08t
14 https://www.w3.org/TR/json-ld/
15 https://json-schema.org/specification.html#specification-documents
16 http://json-schema.org/latest/json-schema-core.html#hypermedia

– `previous` – l'URI renvoie à la page suivante (s'il est utilisé pendant la pagination), et
– le nom arbitraire `v` indique la signification personnalisée d'une liaison.

API WEB SOAP

101. La présente norme recommande le style architectural REST comme approche à préférer en matière de conception d'API. Les architectures RESTful sont généralement plus simples à concevoir, à développer et à intégrer que les architectures SOAP. Ces dernières sont traitées ici dans un souci d'exhaustivité; on ne fournit pas d'exemple ni de cas d'utilisation.

102. Une API Web SOAP est une application logicielle identifiée par un URI, dont les interfaces et associations peuvent être définies, décrites et découvertes par des objets XML. De plus, elle prend en charge les interactions directes avec d'autres applications logicielles utilisant des messages en XML par l'intermédiaire de protocoles Internet tels que SOAP et HTTP.

103. Un contrat basé sur SOAP est décrit dans un langage de description de services Web (WSDL), qui est un document type élaboré par le W3C. D'un bout à l'autre du présent document, le "document WSDL valant contrat de service Web" sera simplement désigné par l'abréviation "WSDL".

104. Deux styles peuvent présider à la création de services Web : le contrat en dernier et le contrat en premier. Selon l'approche du contrat en dernier, on commence par le code, et le contrat de services Web est généré en conséquence. Avec l'approche du contrat en premier, on commence par le contrat WSDL et on utilise le code pour exécuter ledit contrat.

Règles générales

105. Le profil du Web Service Interoperability (interopérabilité des services Web) (WS-I) est l'une des normes les plus importantes en matière d'API basées sur SOAP, et il fournit un socle minimal à l'écriture de services Web capables de travailler ensemble. Il donne des orientations sur la manière dont les services sont "exposés" les uns aux autres et sur leur mode de transfert d'informations (ou "messagerie"). Ce profil sert à exécuter des versions spécifiques de certaines des plus importantes normes applicables aux services Web, à savoir notamment WSDL, SOAP et XML. Le fait de se conformer à certains profils implique une conformité à des versions spécifiques de ces normes. WS-I Basic Profile v1.1 donne des conseils pour utiliser XML 1.0, HTTP 1.1, UDDI, SOAP 1.1, WSDL 1.1 et UDDI 2.0. WS-I Basic Profile 2.0 donne des conseils pour utiliser SOAP 1.2, WSDL 1.1, UDDI 2.0, WS-Addressing et MTOM. SOAP 1.2 fournit un modèle de traitement clair et améliore l'interopérabilité. WSDL 2.0 a été mis au point pour résoudre les problèmes d'interopérabilité que présentait WSDL 1.1 en utilisant des associations SOAP 1.2 améliorées.

[WS-01] Tous les WSDL DOIVENT se conformer au WS-I Basic Profile 2.0. Le WSDL 1.2 PEUT être utilisé.

106. Une association WSDL SOAP peut être une association de style d'appel de procédure à distance (RPC) ou une association de style de document. Une association SOAP peut également avoir une utilisation codée ou une utilisation littérale. On a ainsi cinq styles/modèles d'utilisation : RPC/codé, RPC/littéral, document/codé, document/littéral, document/littéral compacté.

[WS-02] Les services DOIVENT suivre l'association de style de document et les modèles d'utilisation littérale (document/littéral ou document/littéral compacté). Lorsqu'il y a des graphiques, le style RPC/code DOIT être utilisé.

[WS-03] Pour les cas d'utilisation exceptionnels, par exemple en cas d'opérations surchargées dans le WSDL, tous les autres styles DEVRAIENT être utilisés.

107. Le WSDL concret devrait être séparé du WSDL abstrait afin de fournir une interface plus modulaire et flexible. Le WSDL abstrait définit les types de données, les messages, l'opération et le type de port. Le WSDL concret définit l'association, le port et le service.

[WS-04] Le WSDL DEVRAIT être divisé en une partie abstraite et une partie concrète.

[WS-05] Tous les types de données DEVRAIENT être définis dans un fichier XSD et importés dans le WSDL abstrait.

[WS-06] Le WSDL concret ne DOIT définir qu'un seul service avec un seul port.

Schémas

108. Les schémas utilisés dans le WSDL doivent être conformes à la norme ST.96 de l'OMPI. À des fins de réutilisation et dans l'optique de la modularité, un schéma doit être un document distinct qui est soit inclus, soit importé dans le WSDL, au lieu d'être défini directement dans ce dernier. Il sera alors possible de modifier la structure XML sans modifier le WSDL.

[WS-07] Le schéma défini dans l'élément `wsdl:types` DOIT être importé d'un fichier de schéma autonome, pour permettre la modularité et la réutilisation.

[WS-08] L'importation d'un schéma externe DOIT être implémentée à l'aide d'une technique `xsd:import`, et non une `xsd:include`.

[WS-09] L'élément `xsd:any` NE DOIT PAS être utilisé pour spécifier un élément racine dans le corps du message.

[WS-10] L'espace de nommage cible pour le WSDL (attribut `targetNamespace` sur `wsdl:definitions`) DOIT être différent de l'espace de nommage cible du schéma (attribut `targetNamespace` sur `xsd:schema`).

[WS-11] Les requêtes et les réponses (convention de nommage, format de message, structure des données et dictionnaire de données) DEVRAIENT suivre la norme ST.96 de l'OMPI.

Nommage et versionnage

109. Des conventions de nommage appropriées devraient également être appliquées pour nommer les services et les éléments de WSDL. Ces conventions de nommage devraient suivre celles qui sont implémentées dans la norme ST.96 de l'OMPI.

[WS-12] Les services DOIVENT être nommés en caractères haut de casse de type "camel" et avoir un suffixe "Service", par exemple `https://wipo.int/PatentsService`.

[WS-13] Les éléments WSDL que sont le message, la partie, le type de port, l'opération, les données d'entrée, les données de sortie et l'association DEVRAIENT être nommés en caractères haut de casse de type "camel".

[WS-14] Les noms des messages de requête DEVRAIENT avoir un suffixe "Request".

[WS-15] Les noms des messages de réponse DEVRAIENT avoir un suffixe "Response".

[WS-16] Les noms d'opération DEVRAIENT suivre le format `<Verb><Object>{<Qualifier>}`, où `<Verb>` indique l'opération (de préférence Get, Create, Update ou Delete, le cas échéant) sur l'`<Object>` de l'opération, éventuellement suivie d'un `<Qualifier>` de l'`<Object>`.

110. Tous les noms d'opération auront au moins deux parties. Une troisième partie facultative peut être incluse pour clarifier ou préciser le but de l'opération. Les trois parties sont : **<**`Verb> <Object> <Qualifier – Optional>`. Chaque partie est décrite en détail ci-après.

Verbe – chaque nom d'opération commencera par un verbe. On trouvera ci-après des exemples de verbes couramment utilisés.

Verbe	Description	Exemple
Get	Extraire un objet unique	`GetBibData`
Create	Extraire un nouvel objet	`CreateBibData`
Update	Mettre à jour un objet	`UpdateBibData`

Delete	Supprimer un objet	`DeleteCustomer`

Objet – Un nom suivant un verbe sera une description succincte et non ambiguë de la fonction de l'opération. Il s'agit de donner sans ambiguïté aux consommateurs une meilleure idée de ce qu'accomplit l'opération. Étant donné que les différents centres de coût ne définissent pas certaines entités de la même manière, l'objet peut être un champ composite dont le premier nœud est le centre de coût et le deuxième l'entité, par exemple `PatentCustomer`.

Qualificatif – L'attribut (facultatif) qualificatif de l'objet a pour objectif de préciser davantage le domaine d'activité, par exemple `GetCustomerList`. `Get` indique l'opération devant être exécutée par le client et `List` précise qu'il s'agit d'obtenir une liste de clients, et non pas seulement un client, comme dans `GetCustomer`.

111. En vertu des principes de la conception orientée services, les prestataires et les consommateurs de services devraient évoluer indépendamment. Le consommateur de services ne devrait pas être affecté par les modifications mineures (rétrocompatibles) apportées par le prestataire. En conséquence, le versionnage de services ne devrait utiliser que les versions majeures. Pour les API à usage interne (par exemple, aux fins de développement et d'essai), les versions mineures, telles que le versionnage sémantique, peuvent également être utilisées.

[WS-17] Le nom du fichier WSDL DEVRAIT se conformer à la structure suivante : `<service name>_V<major version number>`

[WS-18] L'espace de nommage du fichier WSDL DEVRAIT contenir la version du service; p. ex. `https://wipo.int/PatentsService/V1"`

112. La description du service et de ses opérations est fournie sous la forme de documentation WSDL.

[WS-19] L'élément `wsdl:documentation` DEVRAIT être utilisé dans WSDL avec la description du service (en tant que premier enfant de `wsdl:definitions` dans le WSDL) et de ses opérations.

Conception du contrat de services Web

113. Un contrat de services Web devrait incorporer une interface technique comprenant un langage de description de services (WSDL), les définitions de schéma XML et les descriptions de la politique sur les services Web**,** ainsi qu'une interface non technique comprenant un ou plusieurs documents de description de services.

114. Le WSDL, qui fait partie intégrante du "contrat de service", doit être mis au point avant tout développement de code. Le WSDL ne devrait jamais être autogénéré à partir du code. La devise est la suivante : "Le contrat d'abord", et NON "Le code d'abord". Tous les contrats de services Web doivent être conformes au Web Service Interoperability Basic Profile (WS-I BP). Tout projet qui s'autogénère à partir du code pourra être modifié pour garantir la conformité à ces normes.

Joindre les politiques aux définitions WSDL

115. Les contrats de services Web peuvent être complétés par des politiques de sécurité qui expriment des contraintes, exigences et qualités supplémentaires, lesquelles se rapportent généralement aux comportements des services. Les politiques de sécurité peuvent être lisibles par l'utilisateur et être incorporées dans un accord de niveau de service complémentaire, ou lisibles par machine et traitées au moment de l'exécution. Les politiques lisibles par machine sont définies à l'aide du langage des politiques de service Web et des spécifications connexes.

[WS-20] Les expressions des politiques DOIVENT être isolées dans un document de définition des politiques de services Web distinct, auquel il est renvoyé dans le document WSDL par l'intermédiaire de l'élément `wsp:PolicyReference`.

[WS-21] Les politiques globales ou par domaine DEVRAIENT être isolées et appliquées à plusieurs services.

[WS-22] Les points d'attache des politiques DEVRAIENT être conformes au WSDL 1.1 ou à une version plus récente, de préférence la version 2.0, aux éléments de points d'attache et aux sujets politiques correspondants (service, point de terminaison, opération et message).

SOAP – Web Service Security (Sécurité des services Web)

116. Web Services Security (WSS) : SOAP Message Security (Sécurité des messages SOAP) est un ensemble d'améliorations de la messagerie SOAP qui assure l'intégrité et la confidentialité des messages. WSS : SOAP Message Security est extensible et est compatible avec divers modèles de sécurité et techniques de chiffrement. WSS : SOAP Message Security fournit trois principaux mécanismes qui peuvent être utilisés de façon indépendante ou simultanée :

- La capacité d'envoyer des jetons de sécurité en tant qu'éléments d'un message et d'associer ces jetons au contenu du message;
- La capacité de protéger le contenu d'un message contre toute modification non autorisée et non détectée (intégrité du message); et
- La capacité de protéger le contenu d'un message contre toute divulgation non autorisée (confidentialité du message).

WSS : SOAP Message Security peut être utilisé avec d'autres extensions de service Web et protocoles d'application pour satisfaire à diverses exigences en matière de sécurité.

[WS-23] Les services Web qui utilisent les messages SOAP DEVRAIENT être protégés conformément aux recommandations de WSS : SOAP Standard.

FORMATS DE TYPES DE DONNÉES

117. La présente norme recommande que les formats de types de données primitifs comme l'heure, la date et la langue soient conformes aux recommandations de la norme ST.96 et de la norme ST.97 de l'OMPI, qui sont utilisées respectivement pour les requêtes et les réponses XML et JSON ainsi que pour les paramètres de requête.

[CS-01] Les objets correspondant à l'heure DOIVENT être formatés comme spécifié dans le RFC 3339 de l'IETF (c'est un profil de l'ISO 8601).

[CS-02] Les informations sur le fuseau horaire DEVRAIENT être utilisées comme spécifiées dans le RFC 3339 de l'IETF. Par exemple : `20:54:21+00:00`

[CS-03] Les objets correspondant à la date DOIVENT être formatés comme spécifié dans le RFC 3339 de l'IETF (c'est un profil de l'ISO 8601). Par exemple : `2018-10-19`

[CS-04] Les objets correspondant à la date et à l'heure (c'est-à-dire l'heure système) DOIVENT être formatés comme spécifié dans le RFC 3339 de l'IETF (c'est un profil de l'ISO 8601).

[CS-05] Le fuseau horaire pertinent DEVRAIT être utilisé comme spécifié dans le RFC 3339 de l'IETF. Par exemple : `2017-02-14T20:54:21+00:00`

[CS-06] L'ISO 4217-Alpha (codes monnaie à trois lettres) DOIT être utilisée pour les codes monnaie. La précision de la valeur (c'est-à-dire le nombre de chiffres après la décimale) PEUT varier en fonction des exigences opérationnelles.

[CS-07] La norme ST.3 de l'OMPI (codes à deux lettres) DOIT être utilisée pour la représentation des offices de propriété intellectuelle, États, autres entités et organisations, ainsi que des pays/organisations prioritaires et désignés.

[CS-08] L'ISO 3166-1-Alpha-2 Éléments de code (codes de pays à deux lettres) DOIT être utilisée pour la représentation des noms de pays, des dépendances et autres zones d'intérêt géopolitique particulier, sur la base des listes de noms de pays provenant des Nations Unies.

[CS-09] La norme ISO 639-1 (codes de langue à deux lettres) DOIT être utilisée pour les codes de langue.

[CS-10] Les unités de mesure DEVRAIENT utiliser les unités de mesure définies dans le Code unifié des unités de mesure (basé sur les définitions de l'ISO 80000). Par exemple, pour la mesure de poids à l'aide de kilogrammes (kg).

[CSJ-11] Les caractères utilisés dans les énumérations DOIVENT se limiter aux caractères suivants : {a-z, A-Z, 0-9, point (.), virgule (,), espaces ( ), tiret (-) et caractère de soulignement (_).

[CSJ-12] Les termes de représentation figurant dans l'annexe VI DOIVENT être utilisés pour les noms de propriétés atomiques.

[CSJ-13] Les acronymes et abréviations placés au début de noms de propriétés DOIVENT être constitués de caractères bas de casse. Toutes les autres valeurs d'une énumération, tous les autres acronymes et toutes les autres valeurs d'abréviations DOIVENT être constitués de caractères haut de casse.

CONFORMITÉ

118. La présente norme est conçue comme un ensemble de règles et conventions de conception qui peuvent être disposées en couches au-dessus d'API de service Web existantes ou nouvelles pour créer des fonctionnalités communes. Tous les services ne prennent pas en charge toutes les conventions définies dans la norme, en raison de contraintes opérationnelles (par exemple, la qualité de service peut ne pas être exigée) ou techniques (par exemple, OAuth 2.0 peut déjà être utilisé).

119. La présente norme définit deux niveaux de conformité : les niveaux de conformité A et AA. On notera que les règles contenant PEUT/PEUVENT ne sont pas considérées comme importantes lorsqu'il s'agit de déterminer la conformité.

120. Les API de service Web sont encouragées à prendre en charge un nombre de fonctionnalités supplémentaires au-delà de leur niveau de conformité qui soit aussi important que celui que réclame leur scénario nominal.

121. Deux niveaux de conformité sont définis :

- **Niveau A :** Pour le niveau de conformité A, l'API indique que les règles générales de conception obligatoires (RSG), que la présente norme signale par le verbe "DOIT/DOIVENT", sont suivies. En outre, les règles applicables au type de réponse renvoyée doivent également être respectées. En d'autres termes, les sous-niveaux de conformité ci-après sont indiqués :
    - Niveau AJ : le renvoi d'une réponse JSON ST.97 doit respecter toutes les règles générales (RSG) signalées par le verbe DOIT/DOIVENT ainsi que toutes les règles JSON spécifiques (RSJ) signalées par le verbe DOIT/DOIVENT;
    - Niveau AX : le renvoi d'une instance XML ST.96 doit respecter toutes les règles générales (RSG) signalées par le verbe DOIT/DOIVENT ainsi que toutes les règles XML spécifiques (RSX) signalées par le verbe DOIT/DOIVENT; et
    - Niveau A : le renvoi d'une réponse JSON ou XML doit respecter toutes les règles générales (RSG) signalées par le verbe DOIT/DOIVENT ainsi que toutes les règles JSON spécifiques (RSJ) signalées par le verbe DOIT/DOIVENT et toutes les règles XML spécifiques (RSX) signalées par le verbe DOIT/DOIVENT.

- **Niveau AA :** Pour le niveau de conformité AA, l'API indique que toutes les règles de conception recommandées que la présente norme signale par le verbe "DEVRAIT/DEVRAIENT" sont suivies. Comme pour le niveau A, des sous-niveaux sont prévus en fonction du type de réponse :

    - Niveau AAJ : niveau de conformité AJ ainsi que les règles recommandées DEVRAIT/DEVRAIENT applicables à une réponse JSON; et

    - Niveau AAX : niveau de conformité AX ainsi que les règles recommandées DEVRAIT/DEVRAIENT applicables à une réponse XML.

122. On trouvera à l'annexe I la matrice de traçabilité entre les règles de conception et les niveaux de conformité.

RÉFÉRENCES

Normes de l'OMPI

Norme ST.3 de l'OMPI	Codes à deux lettres pour la représentation des États, autres entités et organisations
Norme ST.96 de l'OMPI	Utilisation du XML dans le traitement de l'information en matière de propriété intellectuelle
Norme ST.97 de l'OMPI	Recommandation relative au traitement des données de propriété intellectuelle au format JSON

Normes et conventions

RFC 2119 de l'IETF	Mots-clés à utiliser dans les appels à observations pour indiquer les niveaux de prescription – www.ietf.org/rfc/rfc2119.txt
IEFT RFC 3339	Date and Time on the Internet: Timestamps – www.ietf.org/rfc/rfc3339.txt
IEFT RFC 3986	Uniform Resource Identifier (URI): Generic Syntax – www.ietf.org/rfc/rfc3986.txt
IEFT RFC 5789	PATCH Method for HTTP – https://tools.ietf.org/rfc/rfc5789.txt
IEFT RFC 5988	Web Linking – https://tools.ietf.org/rfc/rfc5988.txt
IEFT RFC 6648	Deprecating the "X-" Prefix and Similar Constructs in Application Protocols – https://tools.ietf.org/rfc/rfc6648.txt
IEFT RFC 6750	The OAuth 2.0 Authorization Framework: Bearer Token Usage – https://tools.ietf.org/rfc/rfc6750.txt
IEFT RFC 7231	Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content – www.ietf.org/rfc/rfc7231.txt
IEFT RFC 7232	Hypertext Transfer Protocol (HTTP/1.1) – Conditional Requests www.ietf.org/rfc/rfc7232.txt
IEFT RFC 7234	Hypertext Transfer Protocol (HTTP/1.1) – Caching www.ietf.org/rfc/rfc7234.txt
IEFT RFC 7386	JSON Merge Patch – www.ietf.org/rfc/rfc7386.txt.
IEFT RFC 7240	Prefer Header for HTTP – https://tools.ietf.org/rfc/rfc7240.txt
IEFT RFC 7519	JSON Web Token – www.ietf.org/rfc/rfc7519.txt
IEFT RFC 7540	Hypertext Transfer Protocol Version 2 (HTTP/2) – https://tools.ietf.org/html/rfc7540
IEFT BCP-47	Tags for Identifying Languages – https://tools.ietf.org/rfc/bcp/bcp47.txt.
ISO 639-1	Language codes – https://en.wikipedia.org/wiki/List_of_ISO_639-1_codes
ISO 3166-1 alpha-2	Two-letter acronyms for country codes – https://fr.wikipedia.org/wiki/ISO_3166-1_alpha-2
ISO 3166-1 alpha-3	Three-letter acronyms for country codes – https://en.wikipedia.org/wiki/ISO_3166-1_alpha-3
ISO 4217	Codes des monnaies – www.iso.org/iso/home/standards/currency_codes.htm
ISO 8601	Date and Time Formats – https://fr.wikipedia.org/wiki/ISO_8601

OData https://www.odata.org/

OASIS OData Metadata Service Entity Model – http://docs.oasis-open.org/odata/odata/v4.0/os/models/MetadataService.edmx.

OASIS OData JSON Format Version 4.0. Texte établi par Ralf Handl, Michael Pizzo et Mark Biamonte. Dernière version – http://docs.oasis-open.org/odata/odata-json-format/v4.0/odata-json-format-v4.0.html.

OASIS OData Atom Format Version 4.0. Texte établi par Martin Zurmuehl, Michael Pizzo et Ralf Handl. Dernière version – http://docs.oasis-open.org/odata/odata-atom-format/v4.0/odata-atom-format-v4.0.html.

OASIS OData "OData Version 4.0

- Part 1: Protocol– http://docs.oasis-open.org/odata/odata/v4.0/os/part1-protocol/odata-v4.0-os-part1-protocol.html.
- Part 2: URL Conventions – http://docs.oasis-open.org/odata/odata/v4.0/os/part2-url-conventions/odata-v4.0-os-part2-url-conventions.html.
- Part 3: Common Schema Definition Language (CSDL) – http://docs.oasis-open.org/odata/odata/v4.0/os/part3-csdl/odata-v4.0-os-part3-csdl.html.

OASIS ABNF components: OData ABNF Construction Rules Version 4.0 and OData ABNF Test Cases – http://docs.oasis-open.org/odata/odata/v4.0/os/abnf/

OASIS Vocabulary components: OData Core Vocabulary, OData Measures Vocabulary and OData Capabilities Vocabulary – http://docs.oasis-open.org/odata/odata/v4.0/os/vocabularies/

OASIS XML schemas:

OData EDMX XML Schema and OData EDM XML Schema– http://docs.oasis-open.org/odata/odata/v4.0/os/schemas/

OASIS SAML 2.0 http://docs.oasis-open.org/security/saml/Post2.0/sstc-saml-tech-overview-2.0.html

RAML
(ReSTful API Modeling Language) http://raml.org

OpenAPI Initiative www.openapis.org

Richardson's
REST API Maturity Model https://martinfowler.com/articles/richardsonMaturityModel.html

HAL http://stateless.co/hal_specification.html

JSON-LD https://json-ld.org

Collection+JSON Document Format http://amundsen.com/media-types/collection/format/

BadgerFish http://badgerfish.ning.com/

Semantic Versioning https://semver.org/

REST https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm

CQL https://fr.wikipedia.org/wiki/Contextual_Query_Language

Z39.50 https://www.loc.gov/z3950/agency/Z39-50-2003.pdf

WS-I Basic Profile 2.0	http://ws-i.org/profiles/basicprofile-2.0-2010-11-09.html
W3C SOAP 1.2 Part 1:	Messaging Framework – https://www.w3.org/TR/soap12-part1/
W3C SOAP 1.2 Part 2:	Adjuncts https://www.w3.org/TR/soap12-part§
W3C WSDL Version 2.0 Part 1: Core Language	https://www.w3.org/TR/wsdl20/
W3C CORS	https://www.w3.org/TR/cors/
W3C Matric Parameters	https://www.w3.org/DesignIssues/MatrixURIs.html

API REST des offices de propriété intellectuelle

OEB – Services brevets ouverts (OPS) v 3.2	https://developers.epo.org
Office des brevets des États-Unis d'Amérique (USPTO) – PatentsView	http://www.patentsview.org/api/doc.html
OMPI – ePCTv1.1	https://pct.wipo.int/
Office de l'Union européenne pour la propriété intellectuelle (EUIPO) – TMview, Designview, TMclass	http://www.tm-xml.org/TM-XML/TM-XML_xml/TM-XML_TM-Search.xml

API REST du secteur privé et directives en matière de conception

Facebook – https://developers.facebook.com/docs/graph-api/reference	
GitHub	https://developer.github.com/v3
Google APIs Design Guide	https://cloud.google.com/apis/design/
Azure	https://docs.microsoft.com/en-us/rest/api/
OpenAPI	https://swagger.io/docs/specification/about/
OData	http://www.odata.org/documentation/
JSON API	http://jsonapi.org/format/
Microsoft API Design	https://docs.microsoft.com/en-us/azure/architecture/best-practices/api-design
JIRA REST API	https://developer.atlassian.com/server/jira/platform/jira-rest-api-examples
Confluence REST API	https://developer.atlassian.com/server/confluence/
Ebay API	https://developer.ebay.com/api-docs/static/ebay-rest-landing.html
Oracle REST Data Services	http://www.oracle.com/technetwork/developer-tools/rest-data-services/overview/index.html
PayPal REST API	https://developer.paypal.com/docs/api/overview/

Data on the Web Best Practices	https://www.w3.org/TR/dwbp/#intro
SAP Guidelines for Future REST API Harmonization	https://d.dam.sap.com/m/xAUymP/54014_GB_54014_enUS.pdf
GitHub API	https://developer.github.com/v3/
Zalando	https://github.com/zalando/ReSTful-api-guidelines
Dropbox	https://www.dropbox.com/developers
Twitter	https://developer.twitter.com/en/docs

Divers

CQRS	https://martinfowler.com/bliki/CQRS.html
UIT	https://www.itu.int/en/ITU-T/ipr/Pages/open.aspx
OWASP Rest Security Cheat Sheet	https://www.owasp.org/index.php/REST_Security_Cheat_Sheet
DDD	https://martinfowler.com/bliki/BoundedContext.html
REST Principles	https://fr.wikipedia.org/wiki/Representational_state_transfer
Principe ouvert/fermé	https://fr.wikipedia.org/wiki/Open/closed_principle
Which style of WSDL should I use?	https://www.ibm.com/developerworks/library/ws-whichwsdl/

Gouvernement de la Nouvelle-Zélande

API Standard and Guidelines	https://www.ict.govt.nz/guidance-and-resources/standards-compliance/api-standard-and-guidelines/

Cross site scripting prevention cheat sheet
https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html

OWASP Cheat Sheet Series	https://cheatsheetseries.owasp.org/
Digital Signature Standard (DSS)	https://nvlpubs.nist.gov/nistpubs/fips/nist.fips.186-4.pdf

SOAP Message Security 1.0, OASIS Standard 200401 http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0.pdf

SOA Principles of Service Design, Thomas Erl (2008)

[L'annexe I suit]

ANNEXE I

LISTE DES RÈGLES ET CONVENTIONS DE CONCEPTION DES SERVICES WEB RESTFUL

Version 1.1

Révision approuvée par le Comité des normes de l'OMPI (CWS)
à sa dixième session le 25 novembre 2022

Les tableaux ci-après synthétisent les règles et conventions de conception de services et recensent les prescriptions de base en matière de conformité pour déterminer les niveaux de conformité que l'implémentation des API de services Web permet d'atteindre. On trouvera ci-dessous des indications sur les tableaux.

- Le Tableau 1 récapitule les règles à respecter pour atteindre le niveau de conformité AJ (pour une réponse JSON);
- Le Tableau 2 récapitule les règles de conception à respecter pour atteindre le niveau de conformité AX (pour une réponse XML);
- Le Tableau 3 récapitule les règles de conception à respecter pour atteindre le niveau de conformité AAJ (pour une réponse JSON); et
- Le Tableau 4 récapitule les règles de conception à respecter pour atteindre le niveau de conformité AAX (pour une réponse XML).

[Avertissement : Afin d'atteindre un niveau de conformité A, il suffit de suivre les règles figurant dans les tableaux 1 et 2. Afin d'atteindre un niveau de conformité AA, il suffit de suivre les règles figurant dans les tableaux 3 et 4. La troisième lettre indique le type de réponse fournie.]

Tableau 1 : Tableau de conformité pour une réponse JSON

ID de la règle	Description de la règle	Renvoi et observation
[RSG-01]	Le caractère "/" DOIT être utilisé dans le chemin d'accès de l'URI pour définir un rapport hiérarchique entre les ressources, mais le chemin NE DOIT PAS s'achever sur un trait oblique car il ne fournit aucune valeur sémantique et peut être déroutant.	AJ, AX, AAJ, AAX
[RSG-02]	Les noms des ressources DOIVENT avoir une structure de nommage uniforme.	AJ, AX, AAJ, AAX
[RSG-04]	Les paramètres de requête DOIVENT avoir une structure de nommage uniforme.	AJ, AX
[RSG-06]	La structure d'URL d'une API Web DOIT contenir le mot "api" dans l'URI.	AJ, AX, AAJ, AAX
[RSG-07]	Les paramètres de matrice NE DOIVENT PAS être utilisés.	AJ, AX, AAJ, AAX
[RSG-08]	Une API Web DOIT appliquer systématiquement les codes d'état HTTP tels que définis dans les RFC de l'IETF.	AJ, AX, AAJ, AAX

[RSG-10]	Si l'API détecte des valeurs d'entrée invalides, elle DOIT renvoyer le code d'état HTTP "400 Bad Request". La charge utile de l'erreur DOIT indiquer la valeur erronée.	AJ, AX, AAJ, AAX
[RSG-12]	Si l'API détecte des valeurs valides qui exigent que des fonctionnalités ne soient pas exécutées, elle DOIT renvoyer le code d'état HTTP "501 Not Implemented". La charge utile de l'erreur DOIT indiquer la valeur non traitée.	AJ, AX, AAJ, AAX
[RSG-14]	Si une ressource peut être autonome, elle DOIT être une ressource de premier niveau; dans le cas contraire, elle doit être une sous-ressource.	AJ, AX, AAJ, AAX
[RSG-15]	Pour extraire des données imbriquées, les paramètres de requête DOIVENT être utilisés au lieu des chemins d'URL.	AJ, AX, AAJ, AAX
[RSG-18]	Les paramètres des noms de ressources, des segments et des requêtes DOIVENT être composés de mots anglais, écrits selon l'orthographe anglaise figurant dans le dictionnaire Oxford English Dictionary. Les noms de ressources localisés du fait de besoins opérationnels PEUVENT être exprimés dans d'autres langues.	AJ, AX, AAJ, AAX
[RSG-20]	Une API Web DOIT prendre en charge la négociation du type de contenu d'une façon conforme au RFC 7231 de l'IETF.	AJ, AX, AAJ, AAX
[RSG-21]	Le format JSON DOIT être présumé lorsque aucun type de contenu spécifique n'est demandé.	AJ, AX, AAJ, AAX
[RSG-27]	Une API Web DOIT prendre en charge au moins le XML ou le JSON.	AJ, AX, AAJ, AAX
[RSG-28]	Les méthodes HTTP DOIVENT se limiter aux méthodes HTTP normalisées POST, GET, PUT, DELETE, OPTIONS, PATCH, TRACE et HEAD, spécifiées dans les RFC 7231 et 5789 de l'IETF.	AJ, AX, AAJ, AAX
[RSG-33]	Pour un point de terminaison qui apporte une ressource unique, si aucune ressource n'est trouvée, la méthode GET DOIT renvoyer le code d'état "404 Not Found". Les points de terminaison qui renvoient des listes de ressources indiqueront simplement une liste vide.	AJ, AX, AAJ, AAX
[RSG-34]	Si l'extraction d'une ressource aboutit, la méthode GET DOIT renvoyer 200 OK.	AJ, AX, AAJ, AAX
[RSG-35]	Une requête GET DOIT être idempotente.	AJ, AX, AAJ, AAX
[RSG-37]	Une requête HEAD DOIT être idempotente	AJ, AX, AAJ, AAX
[RSG-39]	Selon le RFC 2616 de l'IETF, une requête POST NE DOIT PAS être idempotente	AJ, AX, AAJ, AAX
[RSG-43]	Une requête PUT DOIT être idempotente	AJ, AX, AAJ, AAX
[RSG-44]	Si une ressource n'est pas trouvée, PUT DOIT renvoyer le code d'état "404 Not Found".	AJ, AX, AAJ, AAX

[RSG-45]	Si la mise à jour d'une ressource a réussi, PUT DOIT renvoyer le code d'état "200 OK" si la ressource mise à jour est retournée ou un "204 No Content" si elle ne l'est pas.	AJ, AX, AAJ, AAX
[RSG-46]	Une requête PATCH NE DOIT PAS être idempotente	AJ, AX, AAJ, AAX
[RSG-48]	Si une ressource n'est pas trouvée, PATCH DOIT renvoyer le code d'état "404 Not Found".	AJ, AX, AAJ, AAX
[RSJ-49]	Si une API Web exécute des mises à jour partielles à l'aide de PATCH, elle DOIT utiliser le format de Patch conçu pour JSON (*JSON Merge Patch format*) pour décrire l'ensemble de modifications partielles dont il est question dans le RFC 7386 de l'IETF (en utilisant le type de contenu `application/merge-patch+json`).	AJ, AAJ
[RSG-50]	Une requête DELETE NE DOIT PAS être idempotente	AJ, AX, AAJ, AAX
[RSG-51]	Si une ressource n'est pas trouvée, DELETE DOIT renvoyer le code d'état "404 Not Found".	AJ, AX, AAJ, AAX
[RSG-52]	Si une ressource est supprimée avec succès, DELETE DOIT renvoyer le code d'état "200 OK" si la ressource supprimée est retournée ou "204 No Content" si elle ne l'est pas.	AJ, AX, AAJ, AAX
[RSG-53]	Le destinataire final est soit le serveur d'origine, soit le premier mandataire ou la première passerelle à recevoir une valeur de Max-Forwards de zéro dans la requête. Une requête TRACE NE DOIT PAS inclure un corps.	AJ, AX, AAJ, AAX
[RSG-54]	Une requête TRACE NE DOIT PAS être idempotente	AJ, AX, AAJ, AAX
[RSG-55]	La valeur du champ d'en-tête HTTP Via DOIT suivre la chaîne de requête.	AJ, AX, AAJ, AAX
[RSG-56]	Le champ d'en-tête HTTP Max-Forwards DOIT être utilisé pour permettre au client de réduire la longueur de la chaîne de requête.	AJ, AX, AAJ, AAX
[RSG-58]	Les réponses à TRACE NE DOIVENT PAS être mises en antémémoire	AJ, AX, AAJ, AAX
[RSG-60]	Une requête OPTIONS DOIT être idempotente	AJ, AX, AAJ, AAX
[RSG-70]	Une API Web DOIT utiliser des paramètres de requête pour exécuter la pagination	AJ, AX, AAJ, AAX
[RSG-71]	Une API Web NE DOIT PAS utiliser d'en-têtes HTTP pour exécuter la pagination	AJ, AX, AAJ, AAX
[RSG-75]	Afin de spécifier un critère de tri multiattribut, un paramètre de requête DOIT être utilisé. La valeur de ce paramètre est une liste de clés de tri séparées par une virgule et de sens de tri avec "`asc`" pour croissant ou "`desc`" pour décroissant. Ces derniers PEUVENT être ajoutés à chaque clé de tri, séparés par les deux points. Le serveur DOIT définir le sens par défaut si celui-ci n'est pas spécifié pour une clé.	AJ, AX, AAJ, AAX

[RSG-76]	Une API Web DEVRAIT renvoyer les critères de tri dans la réponse	AJ, AX, AAJ, AAX
[RSG-79]	Une API Web DOIT prendre en charge le renvoi du nombre d'objets d'une collection.	AJ, AX, AAJ, AAX
[RSG-80]	Un paramètre de requête DOIT être utilisé pour prendre en charge le renvoi du nombre d'objets d'une collection.	AJ, AX, AAJ, AAX
[RSG-82]	Une API Web PEUT prendre en charge le renvoi du nombre d'objets d'une collection en l'insérant dans la partie de la réponse qui contient la collection elle-même. Un paramètre de requête DOIT être utilisé.	AJ, AX, AAJ, AAX
[RSG-86]	Un contrat de service DOIT spécifier la grammaire prise en charge (comme les champs, les fonctions, les mots-clés et les opérateurs).	AJ, AX, AAJ, AAX
[RSG-87]	Le paramètre de requête "q" DOIT être utilisé	AJ, AX, AAJ, AAX
[RSG-88]	Au niveau du protocole, une API Web DOIT renvoyer un code d'état HTTP approprié choisi dans la liste type des codes d'état HTTP.	AJ, AX, AAJ, AAX
[RSJ-89]	Au niveau de l'application, une API Web DOIT renvoyer une charge utile signalant l'erreur de façon suffisamment détaillée. Les attributs code et message sont obligatoires, l'attribut details est conditionnellement obligatoire et les attributs target, status, `moreInfo` et `internalMessage` sont facultatifs.	AJ, AX, AAJ, AAX
[RSG-90]	Les erreurs NE DOIVENT PAS exposer les données dont la sécurité est primordiale ou les données techniques internes, comme les piles d'appels, dans les réponses-erreurs.	AJ, AX, AAJ, AAX
[RSG-91]	L'en-tête HTTP `Header: Reason-Phrase` (décrit dans le RFC 2616) NE DOIT PAS être utilisé pour transmettre des réponses-erreurs.	AJ, AX, AAJ, AAX
[RSG-93]	Un format de contrat de service DOIT comprendre les éléments ci-après : – Version de l'API; – Informations sur la sémantique des éléments de l'API; – Ressources; – Attributs des ressources; – Paramètres de requête; – Méthodes; – Types de médias; – Grammaire de recherche (si une telle grammaire est prise en charge); – Codes d'état HTTP; – Méthodes HTTP; – Restrictions et éléments distinctifs; et – Sécurité (le cas échéant).	AJ, AX, AAJ, AAX
[RSG-95]	Une API REST DOIT fournir la documentation API en tant que contrat de service.	AJ, AX, AAJ, AAX

[RSG-96]	Une implémentation de l'API Web qui s'écarterait de la présente norme DOIT être explicitement documentée dans le contrat de service. Si une règle dérogeant à la norme n'est pas spécifiée dans le contrat de service, on DOIT présumer que cette norme est appliquée.	AJ, AX, AAJ, AAX
[RSG-97]	Un contrat de service DOIT permettre de générer le code squelette du client de l'API.	AJ, AX, AAJ, AAX
[RSG-105]	Une API Web DOIT prendre en charge la mise en antémémoire des résultats de GET; une API Web PEUT prendre en charge la mise en antémémoire des résultats d'autres méthodes HTTP.	AJ, AX, AAJ
[RSG-113]	Si une API Web prend en charge la gestion des préférences, la liste des préférences qui PEUT être constituée en utilisant l'en-tête `Prefer` DOIT être enregistrée dans le contrat de service.	AAJ, AAX, AJ, AX
[RSG-114]	Si une API Web prend en charge les données localisées, l'en-tête de requête HTTP `Accept-Language` DOIT, conformément au RFC 7231 de l'IETF, être pris en charge pour indiquer la série de langues naturelles qui sont préférées dans la réponse.	AJ, AX, AAJ, AAX
[RSG-116]	Confidentialité : les API et les informations sur les API DOIVENT être identifiées, classées et protégées à tout moment contre l'accès et la divulgation non autorisés et contre les écoutes. Les principes du moindre privilège, de l'absence de confiance, du besoin de connaître et du besoin de partager[i] DOIVENT être respectés.	AJ, AX, AAJ, AAX
[RSG-117]	Intégrité : les API et les informations sur les API DOIVENT être protégées contre toute modification, reproduction, altération et destruction non autorisées. Les informations DOIVENT être modifiées au moyen de transactions et d'interfaces approuvées. Les systèmes DOIVENT être mis à jour en lançant des processus approuvés de gestion de configuration, de gestion des modifications et de gestion des correctifs.	AJ, AX, AAJ, AAX
[RSG-118]	Disponibilité : les API et les informations sur les API DOIVENT être disponibles pour les utilisateurs autorisés au moment voulu, selon les stipulations des accords de niveau de service, les politiques de contrôle d'accès et les processus opérationnels définis.	AJ, AX, AAJ, AAX
[RSG-119]	Non-répudiation : Chaque transaction traitée ou action exécutée par les API DOIT veiller au respect de la non-répudiation en mettant en œuvre des processus de vérification, d'autorisation et d'authentification, en sécurisant les chemins d'accès et en mettant en place des services et mécanismes de non-répudiation.	AJ, AX, AAJ, AAX
[RSG-120]	Authentification, autorisation, vérification : Les utilisateurs, les systèmes, les API ou les dispositifs associés à des transactions ou actions critiques DOIVENT être authentifiés et autorisés à l'aide de services de contrôle d'accès basés sur des rôles ou des attributs, et maintenir la séparation des tâches. En outre, toutes les actions DOIVENT être enregistrées et le poids de l'authentification doit augmenter avec le risque en matière de sécurité de l'information.	AJ, AX, AAJ, AAX
[RSG-121]	Pendant la mise au point des API, les menaces, les cas d'usage malveillant, les techniques sûres de codage, la sécurité de la couche transport et les tests de sécurité DOIVENT faire l'objet d'une attention particulière, notamment en ce qui concerne :	AJ, AX, AAJ, AAX

	– Les `PUTs` et les `POSTs` – à savoir : quelles modifications des données internes pourraient être utilisées pour attaquer ou induire en erreur; – `DELETES` – à savoir : pourraient être utilisé pour supprimer le contenu d'un référentiel de ressources internes; – Méthodes autorisées d'établissement d'une liste blanche – pour s'assurer que les méthodes HTTP autorisées sont légitimement restreintes tandis que les autres renverraient un code de réponse légitime; et – Les attaques les plus courantes devraient être prises en considération pendant la phase de la conception qui porte sur la modélisation des menaces de façon que le risque n'augmente pas. Les menaces et les mesures d'atténuation définies dans l'aide-mémoire de l'OWASP Top Ten (OWASP Top Ten Cheat Sheet) DOIVENT être prises en considération.	
[RSG-122]	Lors de la mise au point des API, les normes et meilleures pratiques énumérées ci-après DEVRAIENT être appliquées : – Meilleures pratiques de codage sûres : principes de codage sûrs de l'OWASP; – Sécurité des API REST : aide-mémoire sur la sécurité REST; – Aseptisation des entrées et protection contre l'injection de code indirecte (XSS) : aide-mémoire XSS de l'OWASP; – Prévention de l'injection SQL : aide-mémoire de l'OWASP sur l'injection SQL, aide-mémoire de l'OWASP sur le paramétrage; et – Sécurité de la couche transport : aide-mémoire de l'OWASP sur la protection de la couche transport.	AJ, AX, AAX, AAJ
[RSG-123]	Les tests de sécurité et l'appréciation de la vulnérabilité DOIVENT être réalisés pour garantir la sécurité des API et leur résistance aux menaces. Cette exigence PEUT être satisfaite en utilisant les tests statiques et dynamiques de sécurité des applications (SAST/DAST), les outils automatisés de gestion des vulnérabilités et les tests de pénétration.	AJ, AX, AAX, AAJ
[RSG-124]	Les services protégés DOIVENT ne fournir que des points de terminaison HTTPS qui utilisent le TLS 1.2, ou une version plus récente, avec une suite de chiffres qui comprend le protocole d'échange de clés ECDHE.	AJ, AX, AAJ, AAX
[RSG-130]	L'authentification anonyme DOIT n'être utilisée que lorsque les clients et l'application qu'ils utilisent accèdent à des informations ou des fonctionnalités d'un faible niveau de sensibilité qui ne devraient pas exiger une authentification, comme les informations publiques.	AJ, AX, AAJ, AAX
[RSG-131]	Une authentification par identifiant et mot de passe ou par hachage de mot de passe NE DOIT PAS être autorisée.	AJ, AX, AAJ, AAX
[RSG-141]	Les clés API DOIVENT être révoquées si le client viole l'accord d'utilisation, comme prescrit par l'Office de propriété intellectuelle.	AJ, AX, AAJ, AAX
[RSG-144]	Des certificats sécurisés et fiables DOIVENT être émis par une autorité de certification qui a la confiance des deux parties dans le cadre d'un processus d'instauration de la confiance ou d'une certification croisée.	AJ, AX, AAJ, AAX

[RSG-145]	Les certificats partagés entre le client et le serveur, p. ex. le X.509, DEVRAIENT être utilisés afin de réduire les risques pour la sécurité des données d'identification qui sont particuliers aux systèmes sensibles et aux actions protégées.	AJ, AX, AAJ, AAX
[RSG-148]	Si l'API REST est publique, l'en-tête HTTP Access-Control-Allow-Origin DOIT être réglé sur "*".	AJ, AX, AAJ, AAX

Tableau 2 : Tableau de conformité pour une réponse XML

ID de la règle	Description de la règle	Renvoi et observation
[RSG-01]	Le caractère "/" DOIT être utilisé dans le chemin de l'URI pour définir un rapport hiérarchique entre les ressources, mais le chemin NE DOIT PAS s'achever sur un trait oblique car il ne fournit aucune valeur sémantique et peut être déroutant.	AJ, AX, AAJ, AAX
[RSG-02]	Les noms des ressources DOIVENT avoir une structure de nommage uniforme.	AJ, AX, AAJ, AAX
[RSG-04]	Les paramètres de requête DOIVENT avoir une structure de nommage uniforme.	AJ, AX
[RSG-06]	La structure d'URL d'une API Web DOIT contenir le mot "`api`" dans l'URI.	AJ, AX, AAJ, AAX
[RSG-07]	Les paramètres de matrice NE DOIVENT PAS être utilisés.	AJ, AX, AAJ, AAX
[RSG-08]	Une API Web DOIT appliquer systématiquement les codes d'état HTTP tels que définis dans les RFC de l'IETF.	AJ, AX, AAJ, AAX
[RSG-10]	Si l'API détecte des valeurs d'entrée invalides, elle DOIT renvoyer le code d'état HTTP "`400 Bad Request`". La charge utile de l'erreur DOIT indiquer la valeur erronée.	AJ, AX, AAJ, AAX
[RSG-12]	Si l'API détecte des valeurs valides qui exigent que des fonctionnalités ne soient pas exécutées, elle DOIT renvoyer le code d'état HTTP "`501 Not Implemented`". La charge utile de l'erreur DOIT indiquer la valeur non traitée.	AJ, AX, AAJ, AAX
[RSG-14]	Si une ressource peut être autonome, elle DOIT être une ressource de premier niveau; dans le cas contraire, elle doit être une sous-ressource.	AJ, AX, AAJ, AAX
[RSG-15]	Pour extraire des données imbriquées, les paramètres de requête DOIVENT être utilisés au lieu des chemins d'URL.	AJ, AX, AAJ, AAX
[RSG-18]	Les paramètres des noms de ressources, des segments et des requêtes DOIVENT être composés de mots anglais, écrits selon l'orthographe anglaise figurant dans le dictionnaire Oxford English Dictionary. Les noms de ressources localisés du fait de besoins opérationnels PEUVENT être exprimés dans d'autres langues.	AJ, AX, AAJ, AAX
[RSG-20]	Une API Web DOIT prendre en charge la négociation du type de contenu d'une façon conforme au RFC 7231 de l'IETF.	AJ, AX, AAJ, AAX

[RSG-21]	Le format JSON DOIT être présumé lorsque aucun type de contenu spécifique n'est demandé.	AJ, AX, AAJ, AAX
[RSG-27]	Une API Web DOIT prendre en charge au moins le XML ou le JSON.	AJ, AX, AAJ, AAX
[RSG-28]	Les méthodes HTTP DOIVENT se limiter aux méthodes HTTP standard `POST`, `GET`, `PUT`, `DELETE`, `OPTIONS`, `PATCH`, `TRACE` et `HEAD`, spécifiées dans les RFC 7231 et 5789 de l'IETF.	AJ, AX, AAJ, AAX
[RSG-33]	Pour un point de terminaison qui apporte une ressource unique, si aucune ressource n'est trouvée, la méthode `GET` DOIT renvoyer le code d'état "`404 Not Found`". Les points de terminaison qui renvoient des listes de ressources indiqueront simplement une liste vide.	AJ, AX, AAJ, AAX
[RSG-34]	Si l'extraction d'une ressource aboutit, la méthode `GET` DOIT renvoyer "`200 OK`".	AJ, AX, AAJ, AAX
[RSG-35]	Une requête `GET` DOIT être idempotente.	AJ, AX, AAJ, AAX
[RSG-37]	Une requête `HEAD` DOIT être idempotente	AJ, AX, AAJ, AAX
[RSG-39]	Selon le RFC 2616 de l'IETF, une requête `POST` NE DOIT PAS être idempotente	AJ, AX, AAJ, AAX
[RSG-43]	Une requête `PUT` DOIT être idempotente	AJ, AX, AAJ, AAX
[RSG-44]	Si une ressource n'est pas trouvée, `PUT` DOIT renvoyer le code d'état "`404 Not Found`".	AJ, AX, AAJ, AAX
[RSG-45]	Si la mise à jour d'une ressource a réussi, `PUT` DOIT renvoyer le code d'état "`200 OK`" si la ressource mise à jour est retournée ou un "204 No Content" si elle ne l'est pas.	AJ, AX, AAJ, AAX
[RSG-46]	Une requête `PATCH` NE DOIT PAS être idempotente	AJ, AX, AAJ, AAX
[RSG-48]	Si une ressource n'est pas trouvée, `PATCH` DOIT renvoyer le code d'état "`404 Not Found`".	AJ, AX, AAJ, AAX
[RSG-50]	Une requête `DELETE` NE DOIT PAS être idempotente	AJ, AX, AAJ, AAX
[RSG-51]	Si une ressource n'est pas trouvée, `DELETE` DOIT renvoyer le code d'état "`404 Not Found`".	AJ, AX, AAJ, AAX
[RSG-52]	Si une ressource est supprimée avec succès, `DELETE` DOIT renvoyer le code d'état "`200 OK`" si la ressource supprimée est retournée ou "`204 No Content`" si elle ne l'est pas.	AJ, AX, AAJ, AAX
[RSG-53]	Le destinataire final est soit le serveur d'origine, soit le premier mandataire ou la première passerelle à recevoir une valeur de Max-Forwards de zéro dans la requête. Une requête TRACE NE DOIT PAS inclure un corps.	AJ, AX, AAJ, AAX

[RSG-54]	Une requête `TRACE` NE DOIT PAS être idempotente	AJ, AX, AAJ, AAX
[RSG-55]	La valeur du champ d'en-tête HTTP `Via` DOIT suivre la chaîne de requête.	AJ, AX, AAJ, AAX
[RSG-56]	Le champ d'en-tête HTTP Max-Forwards DOIT être utilisé pour permettre au client de réduire la longueur de la chaîne de requête.	AJ, AX, AAJ, AAX
[RSG-58]	Les réponses à `TRACE` NE DOIVENT PAS être mises en antémémoire.	AJ, AX, AAJ, AAX
[RSG-60]	Une requête `OPTIONS` DOIT être idempotente.	AJ, AX, AAJ, AAX
[RSG-70]	Une API Web DOIT utiliser des paramètres de requête pour exécuter la pagination.	AJ, AX, AAJ, AAX
[RSG-71]	Une API Web NE DOIT PAS utiliser d'en-têtes HTTP pour exécuter la pagination.	AJ, AX, AAJ, AAX
[RSG-75]	Afin de spécifier un critère de tri multiattribut, un paramètre de requête DOIT être utilisé. La valeur de ce paramètre est une liste de clés de tri séparées par une virgule et de sens de tri avec "`asc`" pour croissant ou "`desc`" pour décroissant. Ces derniers PEUVENT être ajoutés à chaque clé de tri, séparés par les deux points. Le serveur DOIT définir le sens par défaut si celui-ci n'est pas spécifié pour une clé.	AJ, AX, AAJ, AAX
[RSG-76]	Une API Web DEVRAIT renvoyer les critères de tri dans la réponse.	AJ, AX, AAJ, AAX
[RSG-79]	Une API Web DOIT prendre en charge le renvoi du nombre d'objets d'une collection.	AJ, AX, AAJ, AAX
[RSG-80]	Un paramètre de requête DOIT être utilisé pour prendre en charge le renvoi du nombre d'objets d'une collection.	AJ, AX, AAJ, AAX
[RSG-82]	Une API Web PEUT prendre en charge le renvoi du nombre d'objets d'une collection en l'insérant dans la partie de la réponse qui contient la collection elle-même. Un paramètre de requête DOIT être utilisé.	AJ, AX, AAJ, AAX
[RSG-86]	Un contrat de service DOIT spécifier la grammaire prise en charge (comme les champs, les fonctions, les mots-clés et les opérateurs).	AJ, AX, AAJ, AAX
[RSG-87]	Le paramètre de requête "`q`" DOIT être utilisé.	AJ, AX, AAJ, AAX
[RSG-88]	Au niveau du protocole, une API Web DOIT renvoyer un code d'état HTTP approprié choisi dans la liste type des codes d'état HTTP.	AJ, AX, AAJ, AAX
[RSJ-89]	Au niveau de l'application, une API Web DOIT renvoyer une charge utile signalant l'erreur de façon suffisamment détaillée. Les attributs `code` et `message` sont obligatoires, l'attribut `details` est conditionnellement obligatoire et les attributs `target`, `status`, `moreInfo` et `internalMessage` sont facultatifs.	AJ, AX, AAJ, AAX

[RSG-90]	Les erreurs NE DOIVENT PAS exposer les données dont la sécurité est primordiale ou les données techniques internes, comme les piles d'appels, dans les réponses-erreurs.	AJ, AX, AAJ, AAX
[RSG-91]	L'en-tête HTTP `Header : Reason-Phrase` (décrit dans le RFC 2616) NE DOIT PAS être utilisé pour transmettre des réponses-erreurs.	AJ, AX, AAJ, AAX
[RSG-93]	Un format de contrat de service DOIT comprendre les éléments ci-après : – Version de l'API; – Informations sur la sémantique des éléments de l'API; – Ressources; – Attributs des ressources; – Paramètres de requête; – Méthodes; – Types de médias; – Grammaire de recherche (si une telle grammaire est prise en charge); – Codes d'état HTTP; – Méthodes HTTP; – Restrictions et éléments distinctifs; et – Sécurité (le cas échéant).	AJ, AX, AAJ, AAX
[RSG-95]	Une API REST DOIT fournir la documentation API en tant que contrat de service	AJ, AX, AAJ, AAX
[RSG-96]	Une implémentation de l'API Web qui s'écarterait de la présente norme DOIT être explicitement documentée dans le contrat de service. Si une règle dérogeant à la norme n'est pas spécifiée dans le contrat de service, on DOIT présumer que cette norme est appliquée.	AJ, AX, AAJ, AAX
[RSG-97]	Un contrat de service DOIT permettre de générer le code squelette du client de l'API.	AJ, AX, AAJ, AAX
[RSG-105]	Une API Web DOIT prendre en charge la mise en antémémoire des résultats de GET; une API Web PEUT prendre en charge la mise en antémémoire des résultats d'autres méthodes HTTP.	AJ, AX, AAJ
[RSG-113]	Si une API Web prend en charge la gestion des préférences, la liste des préférences qui PEUT être constituée en utilisant l'en-tête `Prefer` DOIT être enregistrée dans le contrat de service.	AAJ, AAX, AJ, AX
[RSG-114]	Si une API Web prend en charge les données localisées, l'en-tête de requête HTTP `Accept-Language` DOIT, conformément au RFC 7231 de l'IETF, être pris en charge pour indiquer la série de langues naturelles qui sont préférées dans la réponse.	AAJ, AAX, AJ, AX
[RSG-116]	Confidentialité : les API et les informations sur les API DOIVENT être identifiées, classées et protégées à tout moment contre l'accès et la divulgation non autorisés et contre les écoutes. Les principes du moindre privilège, de l'absence de confiance, du besoin de connaître et du besoin de partager[ii] DOIVENT être respectés.	AAJ, AAX, AJ, AX
[RSG-117]	Intégrité : les API et les informations sur les API DOIVENT être protégées contre toute modification, reproduction, altération et destruction non autorisées. Les informations DOIVENT être modifiées au moyen de	AAJ, AAX, AJ, AX

	transactions et d'interfaces approuvées. Les systèmes DOIVENT être mis à jour en lançant des processus approuvés de gestion de configuration, de gestion des modifications et de gestion des correctifs.	
[RSG-118]	Disponibilité : les API et les informations sur les API DOIVENT être disponibles pour les utilisateurs autorisés au moment voulu, selon les stipulations des accords de niveau de service, les politiques de contrôle d'accès et les processus opérationnels définis.	AAJ, AAX, AJ, AX
[RSG-119]	Non-répudiation : Chaque transaction traitée ou action exécutée par les API DOIT veiller au respect de la non-répudiation en mettant en œuvre des processus de vérification, d'autorisation et d'authentification, en sécurisant les chemins d'accès et en mettant en place des services et mécanismes de non-répudiation.	AAJ, AAX, AJ, AX
[RSG-120]	Authentification, autorisation, vérification : Les utilisateurs, les systèmes, les API ou les dispositifs associés à des transactions ou actions critiques DOIVENT être authentifiés et autorisés à l'aide de services de contrôle d'accès basés sur des rôles ou des attributs, et maintenir la séparation des tâches. En outre, toutes les actions DOIVENT être enregistrées et le poids de l'authentification doit augmenter avec le risque en matière de sécurité de l'information.	AAJ, AAX, AJ, AX
[RSG-121]	Pendant la mise au point des API, les menaces, les cas d'usage malveillant, les techniques sûres de codage, la sécurité de la couche transport et les tests de sécurité DOIVENT faire l'objet d'une attention particulière, notamment en ce qui concerne : – Les `PUTs` et les `POSTs` – à savoir : quelles modifications des données internes pourraient être utilisées pour attaquer ou induire en erreur; – `DELETES` – à savoir : pourraient être utilisé pour supprimer le contenu d'un référentiel de ressources internes; – Méthodes autorisées d'établissement d'une liste blanche – pour s'assurer que les méthodes HTTP autorisées sont légitimement restreintes tandis que les autres renverraient un code de réponse légitime; et – Les attaques les plus courantes devraient être prises en considération pendant la phase de la conception qui porte sur la modélisation des menaces de façon que le risque n'augmente pas. Les menaces et les mesures d'atténuation définies dans l'aide-mémoire de l'OWASP Top Ten (OWASP Top Ten Cheat Sheet) DOIVENT être prises en considération.	AAJ, AAX, AJ, AX
[RSG-122]	Lors de la mise au point des API, les normes et meilleures pratiques énumérées ci-après DEVRAIENT être appliquées : – Meilleures pratiques de codage sûres : principes de codage sûrs de l'OWASP; – Sécurité des API REST : aide-mémoire sur la sécurité REST; – Aseptisation des entrées et protection contre l'injection de code indirecte (XSS) : aide-mémoire XSS de l'OWASP; – Prévention de l'injection SQL : aide-mémoire de l'OWASP sur l'injection SQL, aide-mémoire de l'OWASP sur le paramétrage; et	AJ, AX, AAX, AAJ

	– Sécurité de la couche transport : aide-mémoire de l'OWASP sur la protection de la couche transport.	
[RSG-123]	Les tests de sécurité et l'appréciation de la vulnérabilité DOIVENT être réalisés pour garantir la sécurité des API et leur résistance aux menaces. Cette exigence PEUT être satisfaite en utilisant les tests statiques et dynamiques de sécurité des applications (SAST/DAST), les outils automatisés de gestion des vulnérabilités et les tests de pénétration.	AJ, AX, AAJ, AAX
[RSG-124]	Les services protégés DOIVENT ne fournir que des points de terminaison HTTPS qui utilisent le TLS 1.2, ou une version plus récente, avec une suite de chiffres qui comprend le protocole d'échange de clés ECDHE.	AJ, AX, AAJ, AAX
[RSG-130]	L'authentification anonyme DOIT n'être utilisée que lorsque les clients et l'application qu'ils utilisent accèdent à des informations ou des fonctionnalités d'un faible niveau de sensibilité qui ne devraient pas exiger une authentification, comme les informations publiques.	AJ, AX, AAJ, AAX
[RSG-131]	Une authentification par identifiant et mot de passe ou par hachage de mot de passe NE DOIT PAS être autorisée.	AJ, AX, AAJ, AAX
[RSG-141]	Les clés API DOIVENT être révoquées si le client viole l'accord d'utilisation, comme prescrit par l'Office de propriété intellectuelle.	AJ, AX, AAJ, AAX
[RSG-144]	Des certificats sécurisés et fiables DOIVENT être émis par une autorité de certification qui a la confiance des deux parties dans le cadre d'un processus d'instauration de la confiance ou d'une certification croisée.	AJ, AX, AAJ, AAX
[RSG-145]	Les certificats partagés entre le client et le serveur, p. ex. le X.509, DEVRAIENT être utilisés afin de réduire les risques pour la sécurité des données d'identification qui sont particuliers aux systèmes sensibles et aux actions protégées.	AJ, AX, AAJ, AAX
[RSG-148]	Si l'API REST est publique, l'en-tête HTTP Access-Control-Allow-Origin DOIT être réglé sur "*".	AJ, AX, AAJ, AAX

Tableau 3 : Tableau de conformité pour le niveau AAJ

ID de la règle	Description de la règle	Renvoi et observation
[RSG-01]	Le caractère "/" DOIT être utilisé dans le chemin d'accès de l'URI pour définir un rapport hiérarchique entre les ressources, mais le chemin NE DOIT PAS s'achever sur un trait oblique car il ne fournit aucune valeur sémantique et peut être déroutant.	AAJ, AAX, AX, AJ
[RSG-02]	Les noms des ressources DOIVENT avoir une structure de nommage uniforme.	AAJ, AAX, AX, AJ
[RSG-03]	Les noms des ressources dans la requête DEVRAIENT utiliser les conventions de nommage des polices de caractères kebab et PEUVENT être abrégés.	AAJ, AAX
[RSG-05]	Les paramètres de requête DEVRAIENT utiliser la convention des caractères bas de casse de type "camel" et PEUVENT être abrégés.	AAJ, AAX
[RSG-06]	La structure d'URL d'une API Web DOIT contenir le mot "api" dans l'URI.	AAJ, AAX, AX, AJ
[RSG-07]	Les paramètres de matrice NE DOIVENT PAS être utilisés.	AAJ, AAX, AX, AJ
[RSG-08]	Une API Web DOIT appliquer systématiquement les codes d'état HTTP tels que définis dans les RFC de l'IETF.	AAJ, AAX, AX, AJ
[RSG-09]	Les codes recommandés à l'annexe V DEVRAIENT être utilisés par une API Web pour classer les erreurs.	AAX, AAJ
[RSG-10]	Si l'API détecte des valeurs d'entrée invalides, elle DOIT renvoyer le code d'état HTTP "`400 Bad Request`". La charge utile de l'erreur DOIT indiquer la valeur erronée.	AAJ, AAX, AX, AJ
[RSG-11]	Si l'API détecte des noms d'arguments syntaxiquement corrects (dans les paramètres de requête) qui ne sont pas prévus, elle DEVRAIT ne pas en tenir compte.	AAJ, AAX
[RSG-12]	Si l'API détecte des valeurs valides qui exigent que des fonctionnalités ne soient pas exécutées, elle DOIT renvoyer le code d'état HTTP "`501 Not Implemented`". La charge utile de l'erreur DOIT indiquer la valeur non traitée.	AAJ, AAX, AX, AJ
[RSG-13]	Une API Web DEVRAIT n'utiliser que des ressources de premier niveau. S'il y a des sous-ressources, elles devraient être des collections et impliquer une association. Une entité devrait être accessible soit comme ressource de premier niveau, soit comme sous-ressource, mais pas les deux.	AAJ, AAX
[RSG-14]	Si une ressource peut être autonome, elle DOIT être une ressource de premier niveau; dans le cas contraire, elle doit être une sous-ressource.	AAJ, AAX, AX, AJ
[RSG-15]	Pour extraire des données imbriquées, les paramètres de requête DOIVENT être utilisés au lieu des chemins d'URL.	AAJ, AAX, AX, AJ
[RSG-16]	Les noms de ressources DEVRAIENT être des noms pour les API Web CRUD et des verbes pour les API Web Intent.	AAJ, AAX

[RSG-17]	Si un nom de ressource est un nom, il DEVRAIT toujours être au pluriel. Les formes de nom irrégulières NE DEVRAIENT PAS être utilisées. Par exemple, `/persons` devrait être utilisé au lieu de `/people`	AAJ, AAX
[RSG-18]	Les paramètres des noms de ressources, des segments et des requêtes DOIVENT être composés de mots anglais, écrits selon l'orthographe anglaise figurant dans le dictionnaire Oxford English Dictionary. Les noms de ressources localisés du fait de besoins opérationnels PEUVENT être exprimés dans d'autres langues.	AAJ, AAX, AX, AJ
[RSG-19]	Une API Web DEVRAIT utiliser, pour la négociation du type de contenu, l'en-tête requête HTTP `Accept` et l'en-tête réponse HTTP `Content-Type`.	AAJ, AAX
[RSG-20]	Une API Web DOIT prendre en charge la négociation du type de contenu d'une façon conforme au RFC 7231 de l'IETF.	AAJ, AAX, AX, AJ
[RSG-21]	Le format JSON DOIT être présumé lorsque aucun type de contenu spécifique n'est demandé.	AAJ, AAX, AX, AJ
[RSG-22]	Une API Web DEVRAIT renvoyer le code d'état "`406 Not Acceptable`" si un format demandé n'est pas pris en charge.	AAJ, AAX
[RSG-23]	Une API Web DEVRAIT rejeter les requêtes contenant des en-têtes de type de contenu imprévus ou manquants en indiquant le code d'état HTTP "`406 Not Acceptable`" ou "`415 Unsupported Media Type`".	AAJ, AAX
[RSG-24]	Les requêtes et les réponses (convention de nommage, format du message, structure des données et dictionnaire de données) DEVRAIENT se référer à la norme ST.96 de l'OMPI pour le format XML et à la norme ST.97 de l'OMPI pour le format JSON.	AAX, AAJ
[RSJ-25]	Les noms de propriété d'objets JSON DEVRAIENT être fournis en caractères bas de casse de type "camel", par exemple applicantName.	AAJ
[RSG-27]	Une API Web DOIT prendre en charge au moins le XML ou le JSON.	AAJ, AAX, AX, AJ
[RSG-28]	Les méthodes HTTP DOIVENT se limiter aux méthodes HTTP normalisées `POST`, `GET`, `PUT`, `DELETE`, `OPTIONS`, `PATCH`, `TRACE` et `HEAD`, spécifiées dans les RFC 7231 et 5789 de l'IETF.	AAJ, AAX, AX, AJ
[RSG-29]	Les méthodes HTTP PEUVENT se conformer au principe de sélection, selon lequel seule la fonctionnalité exigée par le scénario d'usage cible doit être implémentée.	AAJ, AAX
[RSG-30]	Certains mandataires ne prennent en charge que les méthodes `POST` et `GET`. Pour surmonter ces limitations, une API Web PEUT utiliser une méthode `POST` en ajoutant un en-tête HTTP personnalisé qui "tunnellise" la véritable méthode HTTP. L'en-tête HTTP personnalisé `X-HTTP-Method` DEVRAIT être utilisé.	AAJ, AAX
[RSG-31]	Si une méthode HTTP n'est pas prise en charge, le code d'état HTTP "`405 Method Not Allowed`" DEVRAIT être renvoyé.	AAJ, AAX
[RSG-32]	Une API Web DEVRAIT prendre en charge des opérations par lots (ou opérations en vrac) au lieu de requêtes individuelles multiples afin de réduire le temps d'attente. La même sémantique devrait être utilisée pour les méthodes HTTP et les codes d'état HTTP. La charge utile des réponses DEVRAIT contenir des informations sur toutes les opérations	AAJ, AAX

	par lots. Si de multiples erreurs se produisent, la charge utile des erreurs DEVRAIT contenir des informations sur toutes les erreurs survenues (dans l'attribut de l'élément détails). Toutes les opérations en vrac DEVRAIENT être exécutées dans une opération atomique.	
[RSG-33]	Pour un point de terminaison qui apporte une ressource unique, si aucune ressource n'est trouvée, la méthode `GET` DOIT renvoyer le code d'état "`404 Not Found`". Les points de terminaison qui renvoient des listes de ressources indiqueront simplement une liste vide.	AAJ, AAX, AX, AJ
[RSG-34]	Si l'extraction d'une ressource aboutit, la méthode `GET` DOIT renvoyer "`200 OK`".	AAJ, AAX, AX, AJ
[RSG-35]	Une requête `GET` DOIT être idempotente.	AAJ, AAX, AX, AJ
[RSG-36]	Lorsque la longueur d'un URI dépasse 255 octets, la méthode `POST` DEVRAIT être utilisée à la place de `GET` en raison des limitations de `GET`. On peut aussi songer à créer des requêtes nommées, si cela est possible.	AAJ, AAX
[RSG-37]	Une requête `HEAD` DOIT être idempotente	AAJ, AAX, AX, AJ
[RSG-38]	Certains mandataires ne prennent en charge que les méthodes `POST` et `GET`. Afin de surmonter ces limitations, une API Web DEVRAIT prendre en charge un en-tête de requête HTTP personnalisé pour surclasser la méthode HTTP.	AAJ, AAX
[RSG-39]	Selon le RFC 2616 de l'IETF, une requête `POST` NE DOIT PAS être idempotente.	AAJ, AAX, AX, AJ
[RSG-40]	Si la création de la ressource a réussi, l'en-tête HTTP Location DEVRAIT contenir un URI (absolu ou relatif) indiquant une ressource créée.	AAJ, AAX
[RSG-41]	Si la création de la ressource a réussi, la réponse DEVRAIT contenir le code d'état "`201 Created`".	AAJ, AAX
[RSG-42]	Si la création de la ressource a réussi, la charge utile de la réponse DEVRAIT par défaut contenir le corps de la ressource créée, pour permettre au client de l'utiliser sans avoir à faire un nouvel appel HTTP.	AAJ, AAX
[RSG-43]	Une requête `PUT` DOIT être idempotente.	AAJ, AAX, AX, AJ
[RSG-44]	Si une ressource n'est pas trouvée, `PUT` DOIT renvoyer le code d'état "`404 Not Found`".	AAJ, AAX, AX, AJ
[RSG-45]	Si la mise à jour d'une ressource a réussi, `PUT` DOIT renvoyer le code d'état "`200 OK`" si la ressource mise à jour est retournée ou un "`204 No Content`" si elle ne l'est pas.	AAJ, AAX, AX, AJ
[RSG-46]	Une requête `PATCH` NE DOIT PAS être idempotente.	AAJ, AAX, AX, AJ
[RSG-47]	Si une API Web exécute des mises à jour partielles, les caractéristiques idempotentes de `PATCH` DEVRAIENT être prises en considération. Afin de le rendre idempotent, l'API PEUT suivre la suggestion du RFC 5789 de l'IETF consistant à utiliser un verrouillage optimiste.	AAJ, AAX

[RSG-48]	Si une ressource n'est pas trouvée, `PATCH` DOIT renvoyer le code d'état "`404 Not Found`".	AAJ, AAX, AX, AJ
[RSJ-49]	Si une API Web exécute des mises à jour partielles à l'aide de `PATCH`, elle DOIT utiliser le format de Patch conçu pour JSON (*JSON Merge Patch format*) pour décrire l'ensemble de modifications partielles dont il est question dans le RFC 7386 de l'IETF (en utilisant le type de contenu application/merge-patch+json).	AAJ, AJ
[RSG-50]	Une requête `DELETE` NE DOIT PAS être idempotente	AAJ, AAX, AX, AJ
[RSG-51]	Si une ressource n'est pas trouvée, `DELETE` DOIT renvoyer le code d'état "404 Not Found".	AAJ, AAX, AX, AJ
[RSG-52]	Si une ressource est supprimée avec succès, `DELETE` DOIT renvoyer le code d'état "`200 OK`" si la ressource supprimée est retournée ou "`204 No Content`" si elle ne l'est pas.	AAJ, AAX, AX, AJ
[RSG-53]	Le destinataire final est soit le serveur d'origine, soit le premier mandataire ou la première passerelle à recevoir une valeur de Max-Forwards de zéro dans la requête. Une requête `TRACE` NE DOIT PAS inclure un corps.	AAJ, AAX, AX, AJ
[RSG-54]	Une requête `TRACE` NE DOIT PAS être idempotente	AAJ, AAX, AX, AJ
[RSG-55]	La valeur du champ d'en-tête HTTP `Via` DOIT suivre la chaîne de requête.	AAJ, AAX, AX, AJ
[RSG-56]	Le champ d'en-tête HTTP Max-Forwards DOIT être utilisé pour permettre au client de réduire la longueur de la chaîne de requête.	AAJ, AAX, AX, AJ
[RSG-57]	Si la requête est valide, la réponse DEVRAIT contenir l'intégralité du message de requête dans le corps de la réponse, avec Content-Type de "message/http".	AAJ, AAX
[RSG-58]	Les réponses à `TRACE` NE DOIVENT PAS être mises en antémémoire.	AAJ, AAX, AX, AJ
[RSG-59]	Le code d'état "`200 OK`" DEVRAIT être renvoyé à `TRACE`.	AAJ, AAX
[RSG-60]	Une requête `OPTIONS` DOIT être idempotente.	AAJ, AAX, AX, AJ
[RSG-61]	Les en-têtes HTTP personnalisés commençant par le préfixe "X-" NE DEVRAIENT PAS être utilisés.	AAJ, AAX
[RSG-62]	Les en-têtes HTTP personnalisés NE DEVRAIENT PAS être utilisés pour modifier le comportement des méthodes HTTP, à moins qu'il ne s'agisse de venir à bout d'éventuelles limites techniques (p. ex., voir [RSG-39]).	AAJ, AAX
[RSG-63]	La convention de nommage pour les en-têtes HTTP personnalisés est `<organization>-<header name>`, où `<organization>` et `<header>` DEVRAIENT respecter la convention de la police de caractères kebab.	AAJ, AAX
[RSG-64]	Une API Web DEVRAIT prendre en charge une méthode unique de versionnage de services en utilisant le versionnage d'URI (p. ex. `/api/v1/inventors`), le versionnage d'en-tête (p. ex. `Accept-version: v1`) ou le versionnage par type de média (p. ex. `Accept:`	AAJ, AAX

	`application/vnd.v1+json`). Le versionnage de la chaîne d'interrogation NE DEVRAIT PAS être utilisé.	
[RSG-65]	Un système de versionnage et numérotation DEVRAIT être appliqué en ne prenant en considération que le numéro de la version majeure (par exemple `/v1`).	AAJ, AAX
[RSG-66]	Les contrats de service API PEUVENT comprendre une fonctionnalité de redirection des points de terminaison. Lorsqu'un consommateur de services essaie d'invoquer un service, une réponse de redirection peut lui être adressée pour lui dire de renvoyer la requête à un nouveau point de terminaison. Les redirections PEUVENT être temporaires ou permanentes : – Redirection temporaire – utilisant l'en-tête de réponse HTTP Location et le code d'état HTTP "`302 Found`" selon le RFC 7231 de l'IETF; ou – Redirection permanente – utilisant l'en-tête de réponse HTTP Location et le code d'état HTTP "`301 Moved Permanently`"" selon le RFC 7238 de l'IETF.	AAJ, AAX
[RSG-67]	Des stratégies applicables au cycle de vie des API DEVRAIENT être publiées par les développeurs afin d'aider les utilisateurs à comprendre combien de temps une version doit être maintenue.	AAJ, AAX
[RSG-68]	Une API Web DEVRAIT prendre en charge la pagination.	AAJ, AAX
[RSG-69]	Les requêtes paginées NE PEUVENT PAS être idempotentes.	AAJ, AAX
[RSG-70]	Une API Web DOIT utiliser des paramètres de requête pour exécuter la pagination.	AAJ, AAX, AX, AJ
[RSG-71]	Une API Web NE DOIT PAS utiliser d'en-têtes HTTP pour exécuter la pagination.	AAJ, AAX, AX, AJ
[RSG-72]	Les paramètres de requête `limit=<number of items to deliver>` et `offset=<number of items to skip>` DEVRAIENT être utilisés, où limit est le nombre d'objets à renvoyer (taille de la page) et skip le nombre d'objets à sauter (déduction). Si aucune taille de page n'est précisée, une taille par défaut DEVRAIT être définie – globale ou par collection; la déduction par défaut DOIT être zéro "0". Par exemple, voici une adresse URL valide : https://wipo.int/api/v1/patents?limit=10&offset=20	AAJ, AAX
[RSG-73]	Les valeurs des paramètres limit et offset DEVRAIENT figurer dans la réponse.	AAJ, AAX
[RSG-74]	Une API Web DEVRAIT prendre en charge le tri.	AAJ, AAX
[RSG-75]	Afin de spécifier un critère de tri multiattribut, un paramètre de requête DOIT être utilisé. La valeur de ce paramètre est une liste de clés de tri séparées par une virgule et de sens de tri avec "asc" pour croissant ou "desc" pour décroissant. Ces derniers PEUVENT être ajoutés à chaque clé de tri, séparés par les deux points. Le serveur DOIT définir le sens par défaut si celui-ci n'est pas spécifié pour une clé.	AAJ, AAX, AX, AJ
[RSG-76]	Une API Web DEVRAIT renvoyer les critères de tri dans la réponse.	AAJ, AAX, AX, AJ

[RSG-77]	Une API Web PEUT prendre en charge l'extension du corps du contenu renvoyé. Le paramètre de requête `expand=<comma-separated list of attributes names>` DEVRAIT être utilisé.	AAJ, AAX
[RSG-78]	Un paramètre de requête DEVRAIT être utilisé au lieu des chemins d'URL si une API Web prend en charge la projection selon le format : `"fields="<comma-separated list of attribute names>`.	AAJ, AAX
[RSG-79]	Une API Web DOIT prendre en charge le renvoi du nombre d'objets d'une collection.	AAJ, AAX, AX, AJ
[RSG-80]	Un paramètre de requête DOIT être utilisé pour prendre en charge le renvoi du nombre d'objets d'une collection.	AAJ, AAX, AX, AJ
[RSG-81]	Le paramètre de requête count DEVRAIT être utilisé pour renvoyer le nombre d'objets d'une collection.	AAJ, AAX
[RSG-82]	Une API Web PEUT prendre en charge le renvoi du nombre d'objets d'une collection en l'insérant dans la partie de la réponse qui contient la collection elle-même. Un paramètre de requête DOIT être utilisé.	AAJ, AAX, AX, AJ
[RSG-83]	Le paramètre de requête `count=true` DEVRAIT être utilisé. S'il n'est pas spécifié, count devrait être défini par défaut comme false.	AAJ, AAX
[RSG-84]	Si une API Web prend en charge la pagination, elle DEVRAIT prendre en charge le renvoi du nombre de la collection (c'est-à-dire le nombre total d'objets de la collection) dans le texte de la réponse.	AAJ, AAX
[RSG-85]	Lorsqu'une API Web prend en charge des expressions de recherche avancée, un langage de requête, tel que le CQL, DEVRAIT être spécifié.	AAJ, AAX
[RSG-86]	Un contrat de service DOIT spécifier la grammaire prise en charge (comme les champs, les fonctions, les mots-clés et les opérateurs).	AAJ, AAX, AX, AJ
[RSG-87]	Le paramètre de requête "`q`" DOIT être utilisé.	AAJ, AAX, AX, AJ
[RSG-88]	Au niveau du protocole, une API Web DOIT renvoyer un code d'état HTTP approprié choisi dans la liste type des codes d'état HTTP.	AAJ, AAX, AX, AJ
[RSJ-89]	Au niveau de l'application, une API Web DOIT renvoyer une charge utile signalant l'erreur de façon suffisamment détaillée. Les attributs `code` et `message` sont obligatoires, l'attribut `details` est conditionnellement obligatoire et les attributs `target`, `status`, `moreInfo` et `internalMessage` sont facultatifs.	AAJ, AAX, AX, AJ
[RSG-90]	Les erreurs NE DOIVENT PAS exposer les données dont la sécurité est primordiale ou les données techniques internes, comme les piles d'appels, dans les réponses-erreurs.	AAJ, AAX, AX, AJ
[RSG-91]	L'en-tête HTTP Header: `Reason-Phrase` (décrit dans le RFC 2616) NE DOIT PAS être utilisé pour transmettre des réponses-erreurs.	AAJ, AAX, AX, AJ
[RSG-92]	Chaque erreur enregistrée DEVRAIT avoir son propre identifiant de corrélation. Un en-tête HTTP personnalisé DEVRAIT être utilisé et DEVRAIT être nommé identifiant de corrélation.	AAJ, AAX
[RSG-93]	Un format de contrat de service DOIT comprendre les éléments ci-après :	AAJ, AAX, AX, AJ

	– Version de l'API; – Informations sur la sémantique des éléments de l'API; – Ressources; – Attributs des ressources; – Paramètres de requête; – Méthodes; – Types de médias; – Grammaire de recherche (si une telle grammaire est prise en charge); – Codes d'état HTTP; – Méthodes HTTP; – Restrictions et éléments distinctifs; et – Sécurité (le cas échéant).	
[RSG-94]	Un format de contrat de service DEVRAIT comprendre les requêtes et les réponses au schéma XML ou au schéma JSON, ainsi que des exemples d'utilisation de l'API dans les formats pris en charge, c'est-à-dire XML ou JSON.	AAJ, AAX
[RSG-95]	Une API REST DOIT fournir la documentation API en tant que contrat de service.	AAJ, AAX, AX, AJ
[RSG-96]	Une implémentation de l'API Web qui s'écarterait de la présente norme DOIT être explicitement documentée dans le contrat de service. Si une règle dérogeant à la norme n'est pas spécifiée dans le contrat de service, on DOIT présumer que cette norme est appliquée.	AAJ, AAX, AX, AJ
[RSG-97]	Un contrat de service DOIT permettre de générer le code squelette du client de l'API.	AAJ, AAX, AX, AJ
[RSG-98]	Un contrat de service DEVRAIT permettre de générer le code squelette du serveur.	AAJ, AAX
[RSG-99]	Une documentation API Web DEVRAIT être écrite en RAML ou OAS. Les formats de documentation personnalisés NE DEVRAIENT PAS être utilisés.	AAJ, AAX
[RSG-100]	Un consommateur d'API Web DEVRAIT pouvoir spécifier une pause de serveur pour chaque requête; un en-tête HTTP personnalisé DEVRAIT être utilisé. De plus, une pause de serveur maximale DEVRAIT être utilisée afin de protéger les ressources du serveur contre la surutilisation.	AAJ, AAX
[RSG-101]	Une API Web DEVRAIT prendre en charge l'extraction conditionnelle de données, afin que seules les données qui sont modifiées soient extraites. La validation des ressources en fonction du contenu DEVRAIT être utilisée car elle est plus précise.	AAJ, AAX
[RSG-102]	Afin d'exécuter la validation des ressources en fonction du contenu, l'en-tête HTTP `ETag` DEVRAIT être utilisé dans la réponse pour coder l'état des données. Ensuite, cette valeur DEVRAIT être utilisée dans les requêtes ultérieures dans les en-têtes HTTP conditionnels (comme If-Match ou If-None-Match). Si les données n'ont pas été modifiées depuis que la requête a renvoyé le `ETag`, le serveur DEVRAIT renvoyer le code d'état "`304 Not Modified`" (en l'absence de modifications). Ce mécanisme est spécifié dans les RFC 7231 et 7232 de l'IETF.	AAJ, AAX

[RSG-103]	Afin d'exécuter la validation des ressources en fonction du temps, l'en-tête HTTP `Last-Modified` DEVRAIT être utilisé. Ce mécanisme est spécifié dans les RFC 7231 et 7232 de l'IETF.	AAJ, AAX
[RSG-104]	En utilisant le versionnage des réponses, un consommateur de service PEUT mettre en œuvre un verrouillage optimiste.	AAJ, AAX
[RSG-105]	Une API Web DOIT prendre en charge la mise en antémémoire des résultats de `GET`; une API Web PEUT prendre en charge la mise en antémémoire des résultats d'autres méthodes HTTP.	AAJ, AJ, AX
[RSG-106]	Les en-têtes de réponse HTTP `Cache-Control` et `Expires` DEVRAIENT être utilisés. Ce dernier en-tête PEUT être utilisé pour prendre en charge les anciens clients.	AAJ, AAX
[RSG-107]	Une API Web DEVRAIT faire savoir si elle prend en charge les téléchargements partiels de fichiers en répondant aux requêtes `HEAD` et en envoyant les en-têtes de réponse HTTP `Accept-Ranges` et `Content-Length`.	AAJ, AAX
[RSG-108]	Une API Web DEVRAIT prendre en charge les téléchargements partiels de fichiers. Les plages à plusieurs parties DEVRAIENT être prises en charge.	AAJ, AAX
[RSG-109]	Une API Web DEVRAIT faire savoir si elle prend en charge les téléchargements amont partiels de fichiers.	AAJ, AAX
[RSG-110]	Une API Web DEVRAIT prendre en charge les téléchargements amont partiels de fichiers. Les plages à plusieurs parties DEVRAIENT être prises en charge.	AAJ, AAX
[RSG-111]	Si la requête a dépassé la limite maximale autorisée, le prestataire de service DEVRAIT renvoyer avec les en-têtes de réponse HTTP l'en-tête HTTP "`413 Request Entity Too Large`". Un en-tête HTTP personnalisé PEUT être utilisé pour indiquer la taille maximale de la réponse.	AAJ, AAX
[RSG-112]	Si une API Web prend en charge la gestion des préférences, celle-ci DEVRAIT être implémentée conformément au RFC 7240 de l'IETF, c'est-à-dire que l'en-tête de requête HTTP `Prefer` DEVRAIT être utilisé et l'en-tête de réponse HTTP `Preference-Applied` DEVRAIT être renvoyé (confirmant la requête originelle).	AAJ, AAX
[RSG-113]	Si une API Web prend en charge la gestion des préférences, la liste des préférences qui PEUT être constituée en utilisant l'en-tête `Prefer` DOIT être enregistrée dans le contrat de service.	AAJ, AAX, AJ, AX
[RSG-114]	Si une API Web prend en charge les données localisées, l'en-tête de requête HTTP `Accept-Language` DOIT, conformément au RFC 7231 de l'IETF, être pris en charge pour indiquer la série de langues naturelles qui sont préférées dans la réponse.	AAJ, AAX, AJ, AX
[RSG-115]	Si l'API prend en charge les opérations de longue durée, celles-ci DEVRAIENT être asynchrones. Le mode d'implémentation ci-après DEVRAIT être appliqué : a. Le consommateur active le fonctionnement du service; b. Le service renvoie le code d'état "202 Accepted" conformément au RFC 7231 de l'IETF (section 6.3.3), c'est-à-dire que la requête a été acceptée pour traitement, mais	AAJ, AAX

	celui-ci n'est pas terminé. L'emplacement de la tâche dans la file d'attente qui a été créé est également renvoyé avec l'en-tête HTTP Location; et c. Le consommateur appelle le Location renvoyé pour savoir si la ressource est disponible. Si elle ne l'est pas, la réponse DEVRAIT comporter le code d'état "200 OK", contenir l'état d'avancement de la tâche (par exemple, en attente d'exécution) et PEUT contenir d'autres informations (par exemple, un indicateur de déroulement ou un lien permettant d'annuler ou de supprimer la tâche à l'aide de la méthode HTTP DELETE). Si la ressource est disponible, la réponse DEVRAIT comporter le code d'état "303 See Other", et l'en-tête HTTP Location DEVRAIT contenir l'adresse URL pour extraire les résultats de la tâche.	
[RSG-116]	Confidentialité : les API et les informations sur les API DOIVENT être identifiées, classées et protégées à tout moment contre l'accès et la divulgation non autorisés et contre les écoutes. Les principes du moindre privilège, de l'absence de confiance, du besoin de connaître et du besoin de partager[iii] DOIVENT être respectés.	AAJ, AAX, AJ, AX
[RSG-117]	Intégrité : les API et les informations sur les API DOIVENT être protégées contre toute modification, reproduction, altération et destruction non autorisées. Les informations DOIVENT être modifiées au moyen de transactions et d'interfaces approuvées. Les systèmes DOIVENT être mis à jour en lançant des processus approuvés de gestion de configuration, de gestion des modifications et de gestion des correctifs.	AAJ, AAX, AJ, AX
[RSG-118]	Disponibilité : les API et les informations sur les API DOIVENT être disponibles pour les utilisateurs autorisés au moment voulu, selon les stipulations des accords de niveau de service, les politiques de contrôle d'accès et les processus opérationnels définis.	AAJ, AAX, AJ, AX
[RSG-119]	Non-répudiation : Chaque transaction traitée ou action exécutée par les API DOIT veiller au respect de la non-répudiation en mettant en œuvre des processus de vérification, d'autorisation et d'authentification, en sécurisant les chemins d'accès et en mettant en place des services et mécanismes de non-répudiation.	AAJ, AAX, AJ, AX
[RSG-120]	Authentification, autorisation, vérification : Les utilisateurs, les systèmes, les API ou les dispositifs associés à des transactions ou actions critiques DOIVENT être authentifiés et autorisés à l'aide de services de contrôle d'accès basés sur des rôles ou des attributs, et maintenir la séparation des tâches. En outre, toutes les actions DOIVENT être enregistrées et le poids de l'authentification doit augmenter avec le risque en matière de sécurité de l'information.	AAJ, AAX, AJ, AX
[RSG-121]	Pendant la mise au point des API, les menaces, les cas d'usage malveillant, les techniques sûres de codage, la sécurité de la couche transport et les tests de sécurité DOIVENT faire l'objet d'une attention particulière, notamment en ce qui concerne : – Les `PUTs` et les `POSTs` – à savoir : quelles modifications des données internes pourraient être utilisées pour attaquer ou induire en erreur; – `DELETES` – à savoir : pourraient être utilisé pour supprimer le contenu d'un référentiel de ressources internes; – Méthodes autorisées d'établissement d'une liste blanche – pour s'assurer que les méthodes HTTP autorisées sont légitimement restreintes tandis que les autres renverraient un code de réponse légitime; et	AAJ, AAX, AJ, AX

	– Les attaques les plus courantes devraient être prises en considération pendant la phase de la conception qui porte sur la modélisation des menaces de façon que le risque n'augmente pas. Les menaces et les mesures d'atténuation définies dans l'aide-mémoire de l'OWASP Top Ten (OWASP Top Ten Cheat Sheet) DOIVENT être prises en considération.	
[RSG-122]	Lors de la mise au point des API, les normes et meilleures pratiques énumérées ci-après DEVRAIENT être appliquées : – Meilleures pratiques de codage sûres : principes de codage sûrs de l'OWASP; – Sécurité des API REST : aide-mémoire sur la sécurité REST; – Aseptisation des entrées et protection contre l'injection de code indirecte (XSS) : aide-mémoire XSS de l'OWASP; – Prévention de l'injection SQL : aide-mémoire de l'OWASP sur l'injection SQL, aide-mémoire de l'OWASP sur le paramétrage; et – Sécurité de la couche transport : aide-mémoire de l'OWASP sur la protection de la couche transport.	AAJ, AAX, AJ, AX
[RSG-123]	Les tests de sécurité et l'appréciation de la vulnérabilité DOIVENT être réalisés pour garantir la sécurité des API et leur résistance aux menaces. Cette exigence PEUT être satisfaite en utilisant les tests statiques et dynamiques de sécurité des applications (SAST/DAST), les outils automatisés de gestion des vulnérabilités et les tests de pénétration.	AAJ, AAX, AJ, AX
[RSG-124]	Les services protégés DOIVENT ne fournir que des points de terminaison HTTPS qui utilisent le TLS 1.2, ou une version plus récente, avec une suite de chiffres qui comprend le protocole d'échange de clés ECDHE.	AAJ, AAX, AJ, AX
[RSG-125]	En ce qui concerne les protocoles d'authentification, la parfaite sécurité itérative DEVRAIT être utilisée pour assurer la sécurité du transport. L'utilisation d'algorithmes cryptographiques non sûrs et la rétrocompatibilité avec les protocoles SSL 3 et TLS 1.0/1.1 NE DEVRAIENT PAS être autorisées.	AAX, AAJ
[RSG-126]	Pour une sécurité et une confiance maximales, un réseau privé virtuel sous protocole de sécurité IPSec DEVRAIT être installé de site à site pour mieux protéger les informations transmises sur des réseaux non sécurisés.	AAX, AAJ
[RSG-127]	L'application consommatrice DEVRAIT valider la chaîne de certificats TLS au moment de demander l'accès à des ressources protégées, notamment en consultant la liste de révocation de certificats.	AAX, AAJ
[RSG-128]	Les services protégés DEVRAIENT utiliser uniquement des certificats valides émis par une autorité de certification digne de confiance.	AAX, AAJ
[RSG-129]	Les jetons DEVRAIENT être signés à l'aide d'algorithmes de signature sécurisés qui soient conformes à la norme de signature numérique FIPS –186-4. L'algorithme de signature numérique RSA ou l'algorithme ECDSA DEVRAIENT être pris en considération.	AAX, AAJ
[RSG-130]	L'authentification anonyme DOIT n'être utilisée que lorsque les clients et l'application qu'ils utilisent accèdent à des informations ou des	AAJ, AAX, AJ, AX

	fonctionnalités d'un faible niveau de sensibilité qui ne devraient pas exiger une authentification, comme les informations publiques.	
[RSG-131]	Une authentification par identifiant et mot de passe ou par hachage de mot de passe NE DOIT PAS être autorisée.	AAJ, AAX, AJ, AX
[RSG-132]	Si un service est protégé, Open ID Connect DEVRAIT être utilisé.	AAX, AAJ
[RSG-133]	Lorsqu'un jeton Web JSON (JWT) est utilisé, un secret de JWT DEVRAIT posséder un degré élevé d'entropie afin d'augmenter le facteur de travail d'une attaque par force brute; les jetons TTL et RTTL DEVRAIENT être aussi courts que possible; et les informations sensibles NE DEVRAIENT PAS être stockées dans la charge utile du JWT.	AAX, AAJ
[RSG-134]	Dans les requêtes POST/PUT, les données sensibles DEVRAIENT être transférées dans le corps de la requête ou par des en-têtes de requête.	AAX, AAJ
[RSG-135]	Dans les requêtes GET, les données sensibles DEVRAIENT être transférées dans un en-tête http.	AAX, AAJ
[RSG-136]	Afin de limiter au minimum le temps d'attente et de réduire le couplage entre services protégés, la décision de contrôle d'accès DEVRAIT être prise localement par les points de terminaison REST.	AAX, AAJ
[RSG-137]	Les clés API DEVRAIENT être utilisées pour les services protégés et les services publics afin d'empêcher que leur prestataire de services ne soit submergé par des requêtes multiples (attaques par déni de service). S'agissant des services protégés, les clés API PEUVENT être utilisées à des fins de monétisation (plans achetés), d'application de règlements d'utilisation (QoS, qualité de service) et de contrôle.	AAX, AAJ
[RSG-138]	Comme l'indique le RFC 7231 de l'IETF, les clés API PEUVENT être combinées avec l'utilisateur ou l'agent de l'en-tête de requête HTTP pour distinguer entre un utilisateur humain et un agent intelligent.	AAX, AAJ
[RSG-139]	Le prestataire de service DEVRAIT renvoyer l'état d'utilisation actuelle en même temps que les en-têtes de réponse HTTP. Les données de réponse ci-après PEUVENT être renvoyées : – limite de débit – limite de débit (par minute) fixée dans le système; – limite de débit restante – nombre restant de requêtes autorisé pendant la tranche de temps actuelle (-1 indique que la limite a été dépassée); et – remise à zéro de la limite de débit – durée (en secondes) restant à courir jusqu'à la remise à zéro du compteur de requêtes.	AAX, AAJ
[RSG-140]	Le prestataire de service DEVRAIT renvoyer le code d'état "`429 Too Many Requests`" si les requêtes lui sont envoyées trop rapidement.	AAX, AAJ
[RSG-141]	Les clés API DOIVENT être révoquées si le client viole l'accord d'utilisation, comme prescrit par l'Office de propriété intellectuelle.	AAJ, AAX, AJ, AX
[RSG-142]	Les clés API DEVRAIENT être transférées à l'aide d'en-têtes HTTP personnalisés. Elles NE DEVRAIENT PAS l'être à l'aide de paramètres de requête.	AAX, AAJ

[RSG-143]	Les clés API DEVRAIENT être générées de façon aléatoire.	AAX, AAJ
[RSG-144]	Des certificats sécurisés et fiables DOIVENT être émis par une autorité de certification qui a la confiance des deux parties dans le cadre d'un processus d'instauration de la confiance ou d'une certification croisée.	AAJ, AAX, AJ, AX
[RSG-145]	Les certificats partagés entre le client et le serveur, par exemple le X.509, DEVRAIENT être utilisés afin de réduire les risques pour la sécurité des données d'identification qui sont particuliers aux systèmes sensibles et aux actions protégées.	AAJ, AAX, AJ, AX
[RSG-146]	Pour les services hautement protégés, une authentification bidirectionnelle entre le client et le serveur DEVRAIT utiliser des certificats pour fournir une sécurité supplémentaire.	AAX, AAJ
[RSG-147]	Une authentification multifacteurs DEVRAIT être implémentée afin de réduire les risques pour la sécurité des données d'identification qui sont particuliers aux applications présentant un profil de risque élevé, aux systèmes traitant des informations très sensibles ou aux actions protégées.	AAX, AAJ
[RSG-148]	Si l'API REST est publique, l'en-tête HTTP Access-Control-Allow-Origin DOIT être réglé sur "*".	AAJ, AAX, AJ, AX
[RSG-149]	Si l'API REST est protégée, CORS DEVRAIT être utilisé, si possible. À défaut, JSONP PEUT être utilisé comme solution de repli, mais seulement pour les requêtes GET, par exemple, lorsque l'utilisateur utilise un navigateur ancien. Iframe NE DEVRAIT PAS être utilisé.	AAX, AAJ
[RSJ-150]	Si on utilise des cas décrits par un schéma, l'en-tête Link DEVRAIT être utilisé pour fournir un lien vers un schéma JSON téléchargeable SELON LE RFC 8288.	AAJ
[RSJ-151]	Une API Web DEVRAIT implémenter au moins le niveau 2 (Transport Native Properties) du RMM. Le niveau 3 (Hypermédia) PEUT être implémenté pour rendre l'API autodécouvrable.	AAJ
[RSJ-152]	Pour mettre au point un format hypermédia personnalisé, l'ensemble d'attributs ci-après DEVRAIT être utilisé en les imbriquant dans un lien d'attributs : – `href` – l'URI cible; – `rel` – la signification de l'URI cible; – `self` – l'URI renvoie à la ressource elle-même; – `next` –l'URI renvoie à la page précédente (s'il est utilisé pendant la pagination); – `previous` – l'URI renvoie à la page suivante (s'il est utilisé pendant la pagination), et – le nom arbitraire `v` indique la signification personnalisée d'une liaison.	AAJ

Tableau 4 : Niveau de conformité AAX

ID de la règle	Description de la règle	Renvoi et observation
[RSG-01]	Le caractère "/" DOIT être utilisé dans le chemin d'accès de l'URI pour définir un rapport hiérarchique entre les ressources, mais le chemin NE DOIT PAS s'achever sur un trait oblique car il ne fournit aucune valeur sémantique et peut être déroutant.	AAJ, AAX
[RSG-02]	Les noms des ressources DOIVENT avoir une structure de nommage uniforme.	AAJ, AAX, AJ, AX
[RSG-03]	Les noms des ressources dans la requête DEVRAIENT utiliser les conventions de nommage des polices de caractères kebab et PEUVENT être abrégés.	AAJ, AAX
[RSG-05]	Les paramètres de requête DEVRAIENT utiliser la convention des caractères bas de casse de type "camel" et PEUVENT être abrégés.	AAJ, AAX
[RSG-06]	La structure d'URL d'une API Web DOIT contenir le mot "api" dans l'URI.	AAJ, AAX, AX, AJ
[RSG-07]	Les paramètres de matrice NE DOIVENT PAS être utilisés.	AAJ, AAX, AX, AJ
[RSG-08]	Une API Web DOIT appliquer systématiquement les codes d'état HTTP tels que définis dans les RFC de l'IETF.	AAJ, AAX, AX, AJ
[RSG-09]	Les codes recommandés à l'annexe V DEVRAIENT être utilisés par une API Web pour classer les erreurs.	AAX, AAJ
[RSG-10]	Si l'API détecte des valeurs d'entrée invalides, elle DOIT renvoyer le code d'état HTTP "`400 Bad Request`". La charge utile de l'erreur DOIT indiquer la valeur erronée.	AAJ, AAX, AX, AJ
[RSG-11]	Si l'API détecte des noms d'arguments syntaxiquement corrects (dans les paramètres de requête) qui ne sont pas prévus, elle DEVRAIT ne pas en tenir compte.	AAJ, AAX
[RSG-12]	Si l'API détecte des valeurs valides qui exigent que des fonctionnalités ne soient pas exécutées, elle DOIT renvoyer le code d'état HTTP "`501 Not Implemented`". La charge utile de l'erreur DOIT indiquer la valeur non traitée.	AAJ, AAX, AX, AJ
[RSG-13]	Une API Web DEVRAIT n'utiliser que des ressources de premier niveau. S'il y a des sous-ressources, elles devraient être des collections et impliquer une association. Une entité devrait être accessible soit comme ressource de premier niveau, soit comme sous-ressource, mais pas les deux.	AAJ, AAX
[RSG-14]	Si une ressource peut être autonome, elle DOIT être une ressource de premier niveau; dans le cas contraire, elle doit être une sous-ressource.	AAJ, AAX, AX, AJ
[RSG-15]	Pour extraire des données imbriquées, les paramètres de requête DOIVENT être utilisés au lieu des chemins d'URL.	AAJ, AAX, AX, AJ
[RSG-16]	Les noms de ressources DEVRAIENT être des noms pour les API Web CRUD et des verbes pour les API Web Intent.	AAJ, AAX
[RSG-17]	Si un nom de ressource est un nom, il DEVRAIT toujours être au pluriel. Les formes de nom irrégulières NE DEVRAIENT PAS être utilisées. Par exemple, `/persons` devrait être utilisé au lieu de `/people`.	AAJ, AAX
[RSG-18]	Les paramètres des noms de ressources, des segments et des requêtes DOIVENT être composés de mots anglais, écrits selon l'orthographe anglaise figurant dans le dictionnaire Oxford English Dictionary. Les noms de ressources	AAJ, AAX, AX, AJ

	localisés du fait de besoins opérationnels PEUVENT être exprimés dans d'autres langues.	
[RSG-19]	Une API Web DEVRAIT utiliser, pour la négociation du type de contenu, l'en-tête requête HTTP `Accept` et l'en-tête réponse HTTP `Content-Type`.	AAJ, AAX
[RSG-20]	Une API Web DOIT prendre en charge la négociation du type de contenu d'une façon conforme au RFC 7231 de l'IETF.	AAJ, AAX, AX, AJ
[RSG-21]	Le format JSON DOIT être présumé lorsque aucun type de contenu spécifique n'est demandé.	AAJ, AAX, AX, AJ
[RSG-22]	Une API Web DEVRAIT renvoyer le code d'état "`406 Not Acceptable`" si un format demandé n'est pas pris en charge.	AAJ, AAX
[RSG-23]	Une API Web DEVRAIT rejeter les requêtes contenant des en-têtes de type de contenu imprévus ou manquants en indiquant le code d'état HTTP "`406 Not Acceptable`" ou "`415 Unsupported Media Type`".	AAJ, AAX
[RSG-24]	Les requêtes et les réponses (convention de nommage, format du message, structure des données et dictionnaire de données) DEVRAIENT se référer à la norme ST.96 de l'OMPI pour le format XML et à la norme ST.97 de l'OMPI pour le format JSON.	AAX
[RSX-26]	Les noms de composantes XML DEVRAIENT être fournis en caractères haut de casse de type "camel" conformément à la norme ST.96 de l'OMPI.	AAX
[RSG-27]	Une API Web DOIT prendre en charge au moins le XML ou le JSON.	AAJ, AAX, AX, AJ
[RSG-28]	Les méthodes HTTP DOIVENT se limiter aux méthodes HTTP normalisées `POST`, `GET`, `PUT`, `DELETE`, `OPTIONS`, `PATCH`, `TRACE` et `HEAD`, spécifiées dans les RFC 7231 et 5789 de l'IETF.	AAJ, AAX, AX, AJ
[RSG-29]	Les méthodes HTTP PEUVENT se conformer au principe de sélection, selon lequel seule la fonctionnalité exigée par le scénario d'usage cible doit être implémentée.	AAJ, AAX
[RSG-30]	Certains mandataires ne prennent en charge que les méthodes `POST` et `GET`. Pour surmonter ces limitations, une API Web PEUT utiliser une méthode `POST` en ajoutant un en-tête HTTP personnalisé qui "tunnellise" la véritable méthode HTTP. L'en-tête HTTP personnalisé `X-HTTP-Method` DEVRAIT être utilisé.	AAJ, AAX
[RSG-31]	Si une méthode HTTP n'est pas prise en charge, le code d'état HTTP "`405 Method Not Allowed`" DEVRAIT être renvoyé.	AAJ, AAX
[RSG-32]	Une API Web DEVRAIT prendre en charge des opérations par lots (ou opérations en vrac) au lieu de requêtes individuelles multiples afin de réduire le temps d'attente. La même sémantique devrait être utilisée pour les méthodes HTTP et les codes d'état HTTP. La charge utile des réponses DEVRAIT contenir des informations sur toutes les opérations par lots. Si de multiples erreurs se produisent, la charge utile des erreurs DEVRAIT contenir des informations sur toutes les erreurs survenues (dans l'attribut de l'élément détails). Toutes les opérations en vrac DEVRAIENT être exécutées dans une opération atomique.	AAJ, AAX
[RSG-33]	Pour un point de terminaison qui apporte une ressource unique, si aucune ressource n'est trouvée, la méthode `GET` DOIT renvoyer le code d'état "`404 Not Found`". Les points de terminaison qui renvoient des listes de ressources indiqueront simplement une liste vide.	AAJ, AAX, AX, AJ

[RSG-34]	Si l'extraction d'une ressource aboutit, la méthode `GET` DOIT renvoyer `200 OK`.	AAJ, AAX, AX, AJ
[RSG-35]	Une requête `GET` DOIT être idempotente.	AAJ, AAX, AX, AJ
[RSG-36]	Lorsque la longueur d'un URI dépasse 255 octets, la méthode `POST` DEVRAIT être utilisée à la place de `GET` en raison des limitations de `GET`. On peut aussi songer à créer des requêtes nommées, si cela est possible.	AAJ, AAX
[RSG-37]	Une requête `HEAD` DOIT être idempotente.	AAJ, AAX, AX, AJ
[RSG-38]	Certains mandataires ne prennent en charge que les méthodes `POST` et `GET`. Afin de surmonter ces limitations, une API Web DEVRAIT prendre en charge un en-tête de requête HTTP personnalisé pour surclasser la méthode HTTP.	AAJ, AAX
[RSG-39]	Selon le RFC 2616 de l'IETF, une requête `POST` NE DOIT PAS être idempotente.	AAJ, AAX, AX, AJ
[RSG-40]	Si la création de la ressource a réussi, l'en-tête HTTP Location DEVRAIT contenir un URI (absolu ou relatif) indiquant une ressource créée.	AAJ, AAX
[RSG-41]	Si la création de la ressource a réussi, la réponse DEVRAIT contenir le code d'état "`201 Created`".	AAJ, AAX
[RSG-42]	Si la création de la ressource a réussi, la charge utile de la réponse DEVRAIT par défaut contenir le corps de la ressource créée, pour permettre au client de l'utiliser sans avoir à faire un nouvel appel HTTP.	AAJ, AAX
[RSG-43]	Une requête `PUT` DOIT être idempotente.	AAJ, AAX, AX, AJ
[RSG-44]	Si une ressource n'est pas trouvée, `PUT` DOIT renvoyer le code d'état "`404 Not Found`".	AAJ, AAX, AX, AJ
[RSG-45]	Si la mise à jour d'une ressource a réussi, `PUT` DOIT renvoyer le code d'état "`200 OK`" si la ressource mise à jour est retournée ou un "`204 No Content`" si elle ne l'est pas.	AAJ, AAX, AX, AJ
[RSG-46]	Une requête `PATCH` NE DOIT PAS être idempotente.	AAJ, AAX, AX, AJ
[RSG-47]	Si une API Web exécute des mises à jour partielles, les caractéristiques idempotentes de `PATCH` DEVRAIENT être prises en considération. Afin de le rendre idempotent, l'API PEUT suivre la suggestion du RFC 5789 de l'IETF consistant à utiliser un verrouillage optimiste.	AAJ, AAX
[RSG-48]	Si une ressource n'est pas trouvée, `PATCH` DOIT renvoyer le code d'état "`404 Not Found`".	AAJ, AAX, AX, AJ
[RSG-50]	Une requête `DELETE` NE DOIT PAS être idempotente.	AAJ, AAX, AX, AJ
[RSG-51]	Si une ressource n'est pas trouvée, `DELETE` DOIT renvoyer le code d'état "`404 Not Found`".	AAJ, AAX, AX, AJ
[RSG-52]	Si une ressource est supprimée avec succès, `DELETE` DOIT renvoyer le code d'état "`200 OK`" si la ressource supprimée est retournée ou "`204 No Content`" si elle ne l'est pas.	AAJ, AAX, AX, AJ
[RSG-53]	Le destinataire final est soit le serveur d'origine, soit le premier mandataire ou la première passerelle à recevoir une valeur de Max-Forwards de zéro dans la requête. Une requête `TRACE` NE DOIT PAS inclure un corps.	AAJ, AAX, AX, AJ

[RSG-54]	Une requête `TRACE` NE DOIT PAS être idempotente	AAJ, AAX, AX, AJ
[RSG-55]	La valeur du champ d'en-tête HTTP `Via` DOIT suivre la chaîne de requête.	AAJ, AAX, AX, AJ
[RSG-56]	Le champ d'en-tête HTTP `Max-Forwards` DOIT être utilisé pour permettre au client de réduire la longueur de la chaîne de requête.	AAJ, AAX, AX, AJ
[RSG-57]	Si la requête est valide, la réponse DEVRAIT contenir l'intégralité du message de requête dans le corps de la réponse, avec Content-Type de "`message/http`".	AAJ, AAX
[RSG-58]	Les réponses à `TRACE` NE DOIVENT PAS être mises en antémémoire.	AAJ, AAX, AX, AJ
[RSG-59]	Le code d'état "`200 OK`" DEVRAIT être renvoyé à `TRACE`.	AAJ, AAX
[RSG-60]	Une requête `OPTIONS` DOIT être idempotente.	AAJ, AAX, AX, AJ
[RSG-61]	Les en-têtes HTTP personnalisés commençant par le préfixe "X-" NE DEVRAIENT PAS être utilisés.	AAJ, AAX
[RSG-62]	Les en-têtes HTTP personnalisés NE DEVRAIENT PAS être utilisés pour modifier le comportement des méthodes HTTP, à moins qu'il ne s'agisse de venir à bout d'éventuelles limites techniques (par exemple, voir [RSG-39]).	AAJ, AAX
[RSG-63]	La convention de nommage pour les en-têtes HTTP personnalisés est `<organization>-<header name>`, où `<organization>` et `<header>` DEVRAIENT respecter la convention de la police de caractères kebab.	AAJ, AAX
[RSG-64]	Une API Web DEVRAIT prendre en charge une méthode unique de versionnage de services en utilisant le versionnage d'URI (p. ex. `/api/v1/inventors`), le versionnage d'en-tête (p. ex. `Accept-version: v1`) ou le versionnage par type de média (p. ex. `Accept: application/vnd.v1+json`). Le versionnage de la chaîne d'interrogation NE DEVRAIT PAS être utilisé.	AAJ, AAX
[RSG-65]	Un système de versionnage et numérotation DEVRAIT être appliqué en ne prenant en considération que le numéro de la version majeure (par exemple `/v1`).	AAJ, AAX
[RSG-66]	Les contrats de service API PEUVENT comprendre une fonctionnalité de redirection des points de terminaison. Lorsqu'un consommateur de services essaie d'invoquer un service, une réponse de redirection peut lui être adressée pour lui dire de renvoyer la requête à un nouveau point de terminaison. Les redirections PEUVENT être temporaires ou permanentes : – Redirection temporaire – utilisant l'en-tête de réponse HTTP Location et le code d'état HTTP "302 Found" selon le RFC 7231 de l'IETF; ou – Redirection permanente – utilisant l'en-tête de réponse HTTP Location et le code d'état HTTP "301 Moved Permanently"" selon le RFC 7238 de l'IETF.	AAJ, AAX
[RSG-67]	Des stratégies applicables au cycle de vie des API DEVRAIENT être publiées par les développeurs afin d'aider les utilisateurs à comprendre combien de temps une version doit être maintenue.	AAJ, AAX
[RSG-68]	Une API Web DEVRAIT prendre en charge la pagination.	AAJ, AAX
[RSG-69]	Les requêtes paginées NE PEUVENT PAS être idempotentes.	AAJ, AAX

[RSG-70]	Une API Web DOIT utiliser des paramètres de requête pour exécuter la pagination.	AAJ, AAX, AX, AJ
[RSG-71]	Une API Web NE DOIT PAS utiliser d'en-têtes HTTP pour exécuter la pagination.	AAJ, AAX, AX, AJ
[RSG-72]	Les paramètres de requête `limit=<number of items to deliver>` et `offset=<number of items to skip>` DEVRAIENT être utilisés, où limit est le nombre d'objets à renvoyer (taille de la page) et skip le nombre d'objets à sauter (déduction). Si aucune taille de page n'est précisée, une taille par défaut DEVRAIT être définie – globale ou par collection; la déduction par défaut DOIT être zéro "0". Par exemple, voici une adresse URL valide : https://wipo.int/api/v1/patents?limit=10&offset=20	AAJ, AAX
[RSG-73]	Les valeurs des paramètres limit et offset DEVRAIENT figurer dans la réponse.	AAJ, AAX
[RSG-74]	Une API Web DEVRAIT prendre en charge le tri.	AAJ, AAX
[RSG-75]	Afin de spécifier un critère de tri multiattribut, un paramètre de requête DOIT être utilisé. La valeur de ce paramètre est une liste de clés de tri séparées par une virgule et de sens de tri avec "`asc`" pour croissant ou "`desc`" pour décroissant. Ces derniers PEUVENT être ajoutés à chaque clé de tri, séparés par les deux points. Le serveur DOIT définir le sens par défaut si celui-ci n'est pas spécifié pour une clé.	AAJ, AAX, AX, AJ
[RSG-76]	Une API Web DEVRAIT renvoyer les critères de tri dans la réponse.	AAJ, AAX, AX, AJ
[RSG-77]	Une API Web PEUT prendre en charge l'extension du corps du contenu renvoyé. Le paramètre de requête `expand=<comma-separated list of attributes names>` DEVRAIT être utilisé.	AAJ, AAX
[RSG-78]	Un paramètre de requête DEVRAIT être utilisé au lieu des chemins d'URL si une API Web prend en charge la projection selon le format : "`fields="<comma-separated list of attribute names>`.	AAJ, AAX
[RSG-79]	Une API Web DOIT prendre en charge le renvoi du nombre d'objets d'une collection.	AAJ, AAX, AX, AJ
[RSG-80]	Un paramètre de requête DOIT être utilisé pour prendre en charge le renvoi du nombre d'objets d'une collection.	AAJ, AAX, AX, AJ
[RSG-81]	Le paramètre de requête count DEVRAIT être utilisé pour renvoyer le nombre d'objets d'une collection.	AAJ, AAX
[RSG-82]	Une API Web PEUT prendre en charge le renvoi du nombre d'objets d'une collection en l'insérant dans la partie de la réponse qui contient la collection elle-même. Un paramètre de requête DOIT être utilisé.	AAJ, AAX, AX, AJ
[RSG-83]	Le paramètre de requête `count=true` DEVRAIT être utilisé. S'il n'est pas spécifié, count devrait être défini par défaut comme false.	AAJ, AAX
[RSG-84]	Si une API Web prend en charge la pagination, elle DEVRAIT prendre en charge le renvoi du nombre de la collection (c'est-à-dire le nombre total d'objets de la collection) dans le texte de la réponse.	AAJ, AAX
[RSG-85]	Lorsqu'une API Web prend en charge des expressions de recherche avancée, un langage de requête, tel que le CQL, DEVRAIT être spécifié.	AAJ, AAX

[RSG-86]	Un contrat de service DOIT spécifier la grammaire prise en charge (comme les champs, les fonctions, les mots-clés et les opérateurs).	AAJ, AAX, AX, AJ
[RSG-87]	Le paramètre de requête "`q`" DOIT être utilisé.	AAJ, AAX, AX, AJ
[RSG-88]	Au niveau du protocole, une API Web DOIT renvoyer un code d'état HTTP approprié choisi dans la liste type des codes d'état HTTP.	AAJ, AAX, AX, AJ
[RSJ-89]	Au niveau de l'application, une API Web DOIT renvoyer une charge utile signalant l'erreur de façon suffisamment détaillée. Les attributs `code` et `message` sont obligatoires, l'attribut `details` est conditionnellement obligatoire et les attributs `target`, `status`, `moreInfo` et `internalMessage` sont facultatifs.	AAJ, AAX, AX, AJ
[RSG-90]	Les erreurs NE DOIVENT PAS exposer les données dont la sécurité est primordiale ou les données techniques internes, comme les piles d'appels, dans les réponses-erreurs.	AAJ, AAX, AX, AJ
[RSG-91]	L'en-tête HTTP `Header: Reason-Phrase` (décrit dans le RFC 2616) NE DOIT PAS être utilisé pour transmettre des réponses-erreurs.	AAJ, AAX, AX, AJ
[RSG-92]	Chaque erreur enregistrée DEVRAIT avoir son propre identifiant de corrélation. Un en-tête HTTP personnalisé DEVRAIT être utilisé et DEVRAIT être nommé identifiant de corrélation.	AAJ, AAX
[RSG-93]	Un format de contrat de service DOIT comprendre les éléments ci-après : – Version de l'API; – Informations sur la sémantique des éléments de l'API; – Ressources; – Attributs des ressources; – Paramètres de requête; – Méthodes; – Types de médias; – Grammaire de recherche (si une telle grammaire est prise en charge); – Codes d'état HTTP; – Méthodes HTTP; – Restrictions et éléments distinctifs; et – Sécurité (le cas échéant).	AAJ, AAX, AX, AJ
[RSG-94]	Un format de contrat de service DEVRAIT comprendre les requêtes et les réponses au schéma XML ou au schéma JSON, ainsi que des exemples d'utilisation de l'API dans les formats pris en charge, c'est-à-dire XML ou JSON.	AAJ, AAX
[RSG-95]	Une API REST DOIT fournir la documentation API en tant que contrat de service.	AAJ, AAX, AX, AJ
[RSG-96]	Une implémentation de l'API Web qui s'écarterait de la présente norme DOIT être explicitement documentée dans le contrat de service. Si une règle dérogeant à la norme n'est pas spécifiée dans le contrat de service, on DOIT présumer que cette norme est appliquée.	AAJ, AAX, AX, AJ
[RSG-97]	Un contrat de service DOIT permettre de générer le code squelette du client de l'API.	AAJ, AAX, AX, AJ
[RSG-98]	Un contrat de service DEVRAIT permettre de générer le code squelette du serveur.	AAJ, AAX

[RSG-99]	Une documentation API Web DEVRAIT être écrite en RAML ou OAS. Les formats de documentation personnalisés NE DEVRAIENT PAS être utilisés.	AAJ, AAX
[RSG-100]	Un consommateur d'API Web DEVRAIT pouvoir spécifier une pause de serveur pour chaque requête; un en-tête HTTP personnalisé DEVRAIT être utilisé. De plus, une pause de serveur maximale DEVRAIT être utilisée afin de protéger les ressources du serveur contre la surutilisation.	AAJ, AAX
[RSG-101]	Une API Web DEVRAIT prendre en charge l'extraction conditionnelle de données, afin que seules les données qui sont modifiées soient extraites. La validation des ressources en fonction du contenu DEVRAIT être utilisée car elle est plus précise.	AAJ, AAX
[RSG-102]	Afin d'exécuter la validation des ressources en fonction du contenu, l'en-tête HTTP `ETag` DEVRAIT être utilisé dans la réponse pour coder l'état des données. Ensuite, cette valeur DEVRAIT être utilisée dans les requêtes ultérieures dans les en-têtes HTTP conditionnels (comme If-Match ou If-None-Match). Si les données n'ont pas été modifiées depuis que la requête a renvoyé le `ETag`, le serveur DEVRAIT renvoyer le code d'état "`304 Not Modified`" (en l'absence de modifications). Ce mécanisme est spécifié dans les RFC 7231 et 7232 de l'IETF.	AAJ, AAX
[RSG-103]	Afin d'exécuter la validation des ressources en fonction du temps, l'en-tête HTTP `Last-Modified` DEVRAIT être utilisé. Ce mécanisme est spécifié dans les RFC 7231 et 7232 de l'IETF.	AAJ, AAX
[RSG-104]	En utilisant le versionnage des réponses, un consommateur de service PEUT mettre en œuvre un verrouillage optimiste.	AAJ, AAX
[RSG-106]	Les en-têtes de réponse HTTP `Cache-Control` et `Expires` DEVRAIENT être utilisés. Ce dernier en-tête PEUT être utilisé pour prendre en charge les anciens clients.	AAJ, AAX
[RSG-107]	Une API Web DEVRAIT faire savoir si elle prend en charge les téléchargements partiels de fichiers en répondant aux requêtes HEAD et en envoyant les en-têtes de réponse HTTP `Accept-Ranges` et `Content-Length`.	AAJ, AAX
[RSG-108]	Une API Web DEVRAIT prendre en charge les téléchargements partiels de fichiers. Les plages à plusieurs parties DEVRAIENT être prises en charge.	AAJ, AAX
[RSG-109]	Une API Web DEVRAIT faire savoir si elle prend en charge les téléchargements amont partiels de fichiers.	AAJ, AAX
[RSG-110]	Une API Web DEVRAIT prendre en charge les téléchargements amont partiels de fichiers. Les plages à plusieurs parties DEVRAIENT être prises en charge.	AAJ, AAX
[RSG-111]	Si la requête a dépassé la limite maximale autorisée, le prestataire de service DEVRAIT renvoyer avec les en-têtes de réponse HTTP l'en-tête HTTP "`413 Request Entity Too Large`". Un en-tête HTTP personnalisé PEUT être utilisé pour indiquer la taille maximale de la réponse.	AAJ, AAX
[RSG-112]	Si une API Web prend en charge la gestion des préférences, celle-ci DEVRAIT être implémentée conformément au RFC 7240 de l'IETF, c'est-à-dire que l'en-tête de requête HTTP `Prefer` DEVRAIT être utilisé et l'en-tête de réponse HTTP `Preference-Applied` DEVRAIT être renvoyé (confirmant la requête originelle).	AAJ, AAX
[RSG-113]	Si une API Web prend en charge la gestion des préférences, la liste des préférences qui PEUT être constituée en utilisant l'en-tête `Prefer` DOIT être enregistrée dans le contrat de service.	AAJ, AAX, AJ, AX
[RSG-114]	Si une API Web prend en charge les données localisées, l'en-tête de requête HTTP `Accept-Language` DOIT, conformément au RFC 7231 de l'IETF, être pris	AAJ, AAX, AJ, AX

	en charge pour indiquer la série de langues naturelles qui sont préférées dans la réponse.	
[RSG-115]	Si l'API prend en charge les opérations de longue durée, celles-ci DEVRAIENT être asynchrones. Le mode d'implémentation ci-après DEVRAIT être appliqué : a) Le consommateur active le fonctionnement du service; b) Le service renvoie le code d'état "202 Accepted" conformément au RFC 7231 de l'IETF (section 6.3.3), c'est-à-dire que la requête a été acceptée pour traitement, mais celui-ci n'est pas terminé. L'emplacement de la tâche dans la file d'attente qui a été créé est également renvoyé avec l'en-tête HTTP Location; et c) Le consommateur appelle le Location renvoyé pour savoir si la ressource est disponible. Si elle ne l'est pas, la réponse DEVRAIT comporter le code d'état "200 OK", contenir l'état d'avancement de la tâche (par exemple, en attente d'exécution) et PEUT contenir d'autres informations (par exemple, un indicateur de déroulement ou un lien permettant d'annuler ou de supprimer la tâche à l'aide de la méthode HTTP DELETE). Si la ressource est disponible, la réponse DEVRAIT comporter le code d'état "303 See Other", et l'en-tête HTTP Location DEVRAIT contenir l'adresse URL pour extraire les résultats de la tâche.	AAJ, AAX
[RSG-116]	Confidentialité : les API et les informations sur les API DOIVENT être identifiées, classées et protégées à tout moment contre l'accès et la divulgation non autorisés et contre les écoutes. Les principes du moindre privilège, de l'absence de confiance, du besoin de connaître et du besoin de partager[iv] DOIVENT être respectés.	AAJ, AAX, AJ, AX
[RSG-117]	Intégrité : les API et les informations sur les API DOIVENT être protégées contre toute modification, reproduction, altération et destruction non autorisées. Les informations DOIVENT être modifiées au moyen de transactions et d'interfaces approuvées. Les systèmes DOIVENT être mis à jour en lançant des processus approuvés de gestion de configuration, de gestion des modifications et de gestion des correctifs.	AAJ, AAX, AJ, AX
[RSG-118]	Disponibilité : les API et les informations sur les API DOIVENT être disponibles pour les utilisateurs autorisés au moment voulu, selon les stipulations des accords de niveau de service, les politiques de contrôle d'accès et les processus opérationnels définis.	AAJ, AAX, AJ, AX
[RSG-119]	Non-répudiation : Chaque transaction traitée ou action exécutée par les API DOIT veiller au respect de la non-répudiation en mettant en œuvre des processus de vérification, d'autorisation et d'authentification, en sécurisant les chemins d'accès et en mettant en place des services et mécanismes de non-répudiation.	AAJ, AAX, AJ, AX
[RSG-120]	Authentification, autorisation, vérification : Les utilisateurs, les systèmes, les API ou les dispositifs associés à des transactions ou actions critiques DOIVENT être authentifiés et autorisés à l'aide de services de contrôle d'accès basés sur des rôles ou des attributs, et maintenir la séparation des tâches. En outre, toutes les actions DOIVENT être enregistrées et le poids de l'authentification doit augmenter avec le risque en matière de sécurité de l'information.	AAJ, AAX, AJ, AX
[RSG-121]	Pendant la mise au point des API, les menaces, les cas d'usage malveillant, les techniques sûres de codage, la sécurité de la couche transport et les tests de sécurité DOIVENT faire l'objet d'une attention particulière, notamment en ce qui concerne : – Les `PUTs` et les `POSTs` – à savoir : quelles modifications des données internes pourraient être utilisées pour attaquer ou induire en erreur; – `DELETES` – à savoir : pourraient être utilisé pour supprimer le contenu d'un référentiel de ressources internes;	AAJ, AAX, AJ, AX

	– Méthodes autorisées d'établissement d'une liste blanche – pour s'assurer que les méthodes HTTP autorisées sont légitimement restreintes tandis que les autres renverraient un code de réponse légitime; et – Les attaques les plus courantes devraient être prises en considération pendant la phase de la conception qui porte sur la modélisation des menaces de façon que le risque n'augmente pas. Les menaces et les mesures d'atténuation définies dans l'aide-mémoire de l'OWASP Top Ten (OWASP Top Ten Cheat Sheet) DOIVENT être prises en considération.	
[RSG-122]	Lors de la mise au point des API, les normes et meilleures pratiques énumérées ci-après DEVRAIENT être appliquées : – Meilleures pratiques de codage sûres : principes de codage sûrs de l'OWASP; – Sécurité des API REST : aide-mémoire sur la sécurité REST; – Aseptisation des entrées et protection contre l'injection de code indirecte (XSS) : aide-mémoire XSS de l'OWASP; – Prévention de l'injection SQL : aide-mémoire de l'OWASP sur l'injection SQL, aide-mémoire de l'OWASP sur le paramétrage; et – Sécurité de la couche transport : aide-mémoire de l'OWASP sur la protection de la couche transport.	AAJ, AAX, AJ, AX
[RSG-123]	Les tests de sécurité et l'appréciation de la vulnérabilité DOIVENT être réalisés pour garantir la sécurité des API et leur résistance aux menaces. Cette exigence PEUT être satisfaite en utilisant les tests statiques et dynamiques de sécurité des applications (SAST/DAST), les outils automatisés de gestion des vulnérabilités et les tests de pénétration.	AAJ, AAX, AJ, AX
[RSG-124]	Les services protégés DOIVENT ne fournir que des points de terminaison HTTPS qui utilisent le TLS 1.2, ou une version plus récente, avec une suite de chiffres qui comprend le protocole d'échange de clés ECDHE.	AAJ, AAX, AJ, AX
[RSG-125]	En ce qui concerne les protocoles d'authentification, la parfaite sécurité itérative DEVRAIT être utilisée pour assurer la sécurité du transport. L'utilisation d'algorithmes cryptographiques non sûrs et la rétrocompatibilité avec les protocoles SSL 3 et TLS 1.0/1.1 NE DEVRAIENT PAS être autorisées.	AAX, AAJ
[RSG-126]	Pour une sécurité et une confiance maximales, un réseau privé virtuel sous protocole de sécurité IPSec DEVRAIT être installé de site à site pour mieux protéger les informations transmises sur des réseaux non sécurisés.	AAX, AAJ
[RSG-127]	L'application consommatrice DEVRAIT valider la chaîne de certificats TLS au moment de demander l'accès à des ressources protégées, notamment en consultant la liste de révocation de certificats.	AAX, AAJ
[RSG-128]	Les services protégés DEVRAIENT utiliser uniquement des certificats valides émis par une autorité de certification digne de confiance.	AAX, AAJ
[RSG-129]	Les jetons DEVRAIENT être signés à l'aide d'algorithmes de signature sécurisés qui soient conformes à la norme de signature numérique FIPS –186-4. L'algorithme de signature numérique RSA ou l'algorithme ECDSA DEVRAIENT être pris en considération.	AAX, AAJ
[RSG-130]	L'authentification anonyme DOIT n'être utilisée que lorsque les clients et l'application qu'ils utilisent accèdent à des informations ou des fonctionnalités d'un faible niveau de sensibilité qui ne devraient pas exiger une authentification, comme les informations publiques.	AAJ, AAX, AJ, AX

[RSG-131]	Une authentification par identifiant et mot de passe ou par hachage de mot de passe NE DOIT PAS être autorisée.	AAJ, AAX, AJ, AX
[RSG-132]	Si un service est protégé, Open ID Connect DEVRAIT être utilisé.	AAX, AAJ
[RSG-133]	Lorsqu'un jeton Web JSON (JWT) est utilisé, un secret de JWT DEVRAIT posséder un degré élevé d'entropie afin d'augmenter le facteur de travail d'une attaque par force brute; les jetons TTL et RTTL DEVRAIENT être aussi courts que possible; et les informations sensibles NE DEVRAIENT PAS être stockées dans la charge utile du JWT.	AAX, AAJ
[RSG-134]	Dans les requêtes `POST/PUT`, les données sensibles DEVRAIENT être transférées dans le corps de la requête ou par des en-têtes de requête.	AAX, AAJ
[RSG-135]	Dans les requêtes `GET`, les données sensibles DEVRAIENT être transférées dans un en-tête http.	AAX, AAJ
[RSG-136]	Afin de limiter au minimum le temps d'attente et de réduire le couplage entre services protégés, la décision de contrôle d'accès DEVRAIT être prise localement par les points de terminaison REST.	AAX, AAJ
[RSG-137]	Les clés API DEVRAIENT être utilisées pour les services protégés et les services publics afin d'empêcher que leur prestataire de services ne soit submergé par des requêtes multiples (attaques par déni de service). S'agissant des services protégés, les clés API PEUVENT être utilisées à des fins de monétisation (plans achetés), d'application de règlements d'utilisation (QoS, qualité de service) et de contrôle.	AAX, AAJ
[RSG-138]	Comme l'indique le RFC 7231 de l'IETF, les clés API PEUVENT être combinées avec l'utilisateur ou l'agent de l'en-tête de requête HTTP pour distinguer entre un utilisateur humain et un agent intelligent.	AAX, AAJ
[RSG-139]	Le prestataire de service DEVRAIT renvoyer l'état d'utilisation actuelle en même temps que les en-têtes de réponse HTTP. Les données de réponse ci-après PEUVENT être renvoyées : – limite de débit – limite de débit (par minute) fixée dans le système; – limite de débit restante – nombre restant de requêtes autorisé pendant la tranche de temps actuelle (-1 indique que la limite a été dépassée); et – remise à zéro de la limite de débit – durée (en secondes) restant à courir jusqu'à la remise à zéro du compteur de requêtes.	AAX, AAJ
[RSG-140]	Le prestataire de service DEVRAIT renvoyer le code d'état "`429 Too Many Requests`" si les requêtes lui sont envoyées trop rapidement.	AAX, AAJ
[RSG-141]	Les clés API DOIVENT être révoquées si le client viole l'accord d'utilisation, comme prescrit par l'Office de propriété intellectuelle.	AAJ, AAX, AJ, AX
[RSG-142]	Les clés API DEVRAIENT être transférées à l'aide d'en-têtes HTTP personnalisés. Elles NE DEVRAIENT PAS l'être à l'aide de paramètres de requête.	AAX, AAJ
[RSG-143]	Les clés API DEVRAIENT être générées de façon aléatoire.	AAX, AAJ
[RSG-144]	Des certificats sécurisés et fiables DOIVENT être émis par une autorité de certification qui a la confiance des deux parties dans le cadre d'un processus d'instauration de la confiance ou d'une certification croisée.	AAJ, AAX, AJ, AX

[RSG-145]	Les certificats partagés entre le client et le serveur, p. ex. le X.509, DEVRAIENT être utilisés afin de réduire les risques pour la sécurité des données d'identification qui sont particuliers aux systèmes sensibles et aux actions protégées.	AAJ, AAX, AJ, AX
[RSG-146]	Pour les services hautement protégés, une authentification bidirectionnelle entre le client et le serveur DEVRAIT utiliser des certificats pour fournir une sécurité supplémentaire.	AAX, AAJ
[RSG-147]	Une authentification multifacteurs DEVRAIT être implémentée afin de réduire les risques pour la sécurité des données d'identification qui sont particuliers aux applications présentant un profil de risque élevé, aux systèmes traitant des informations très sensibles ou aux actions protégées.	AAX, AAJ
[RSG-148]	Si l'API REST est publique, l'en-tête HTTP `Access-Control-Allow-Origin` DOIT être réglé sur "*".	AAJ, AAX, AJ, AX
[RSG-149]	Si l'API REST est protégée, CORS DEVRAIT être utilisé, si possible. À défaut, JSONP PEUT être utilisé comme solution de repli, mais seulement pour les requêtes GET, par exemple, lorsque l'utilisateur utilise un navigateur ancien. Iframe NE DEVRAIT PAS être utilisé.	AAX, AAJ

[L'annexe II suit]

ANNEXE II

VOCABULAIRE DE PROPRIÉTÉ INTELLECTUELLE REST

Version 1.1

Révision approuvée par le Comité des normes de l'OMPI (CWS)
à sa dixième session le 25 novembre 2022

1. Le Vocabulaire de propriété intellectuelle ci-après est présenté dans le Tableau 5 à titre d'exemples de /paramètres de base de requête de service RESTful. Selon leurs besoins opérationnels, les offices de propriété intellectuelle devront probablement élaborer des requêtes plus complexes et des charges utiles de réponse plus variées. Les paramètres de ce tableau sont des exemples d'éléments de la norme ST.96 en caractères bas de casse de type "camel", utilisé pour une réponse JSON. Le dictionnaire des données de propriété intellectuelle et les schémas XML de propriété intellectuelle de la norme ST.96 peuvent être consultés dans leur intégralité sur le site : https://www.wipo.int/standards/fr/st96/v5-0/.

[Avertissement : à l'avenir, il est prévu qu'un lien vers une liste plus complète du vocabulaire de propriété intellectuelle REST XML et JSON sera fourni, qui sera tenue à jour de manière dynamique et permanente dans la mesure où les éléments et le vocabulaire de la propriété intellectuelle continuent d'évoluer.]

Tableau 5 : Vocabulaire concernant les API en caractères bas de casse de type "camel" : exemples reprenant les éléments XSD de la norme ST.96

Domaine(s) d'activité	Nom(s) de la ressource	Nom du paramètre	Description
TOUS	/marques /brevets /dessins et modèles	`st13ApplicationNumber`	Le numéro de la demande de titre de propriété intellectuelle déposée, au format de la norme ST.13 de l'OMPI qui est une chaîne de plusieurs valeurs comprenant le numéro de la demande nationale, le type de propriété intellectuelle et le pays ou organisation.
TOUS	/marques /brevets /dessins et modèles	`applicationNumber`	Le numéro de la demande de titre de propriété intellectuelle déposée, au format de l'office national.
PLUSIEURS	/marques /dessins et modèles	`internationalRegistrationNumber`	Le numéro d'enregistrement international du droit de propriété intellectuelle. Pour les marques, cela concerne le système de Madrid. Pour les dessins et modèles industriels, cela concerne le système de La Haye.
TOUS	/marques /brevets /dessins et modèles	`AvailableDocument`	Entrée du document unique correspondant aux critères de recherche fournis à DocList API
TOUS	/marques /brevets /dessins et modèles	`SortingCriteria`	Critère de tri utilisé par le DocList API
TOUS	/marques /brevets /dessins et modèles	`ReceivingOfficeCode`	L'office de propriété intellectuelle, au format de la norme ST.2 de l'OMPI.

TOUS	/marques /brevets /dessins et modèles	`ReceivingOfficeDate`	La date de réception par l'office de propriété intellectuelle
Marques	/marques	`RegistrationDate`	La date d'enregistrement par l'office de propriété intellectuelle
		`ApplicationDate`	La date de la demande
		`MarkCurrentStatusCode`	Code de la situation juridique actuelle de la demande
		`MarkCurrentStatusDate`	Date de la situation juridique actuelle de la demande
Brevets	/brevets	`FilingDate`	La date de dépôt de la demande
		`GrantPublicationDate`	La date de publication de la délivrance
		`FileReferenceIdentifier`	Numéro de référence du déposant
		`ApplicationBodyStatus`	Situation du corps de la demande
		`StatusEventData`	Données relatives à la situation juridique d'une demande de brevet
		`KeyEventCode`	Code indiquant un événement de haut niveau qui porte sur les situations les plus générales et importantes à l'intérieur d'une catégorie
Dessins et modèles industriels	/dessins et modèles	`ApplicationDate`	La date de dépôt de la demande
		`designApplicationCurrentStatus`	Catégorie de la situation juridique actuelle de la demande de dessin ou modèle industriels
		`designApplicationCurrentStatusDate`	Date de la situation juridique actuelle de la demande de dessin ou modèle industriels

2. Les paramètres techniques de requête définis dans le tableau 6 ci-après devraient s'appliquer à tous les services API REST.

Tableau 6 : Vocabulaire technique concernant les API

Paramètre de requête/chemin	Valeur de paramètre Type de données	Contrainte	Format	Description	Règle de conception
format	chaîne		`type/subtype; parameter=value` selon le RFC 7231, 3.1.1.1. Type de média	Utilisé pour la négociation du type de contenu (préférer un en-tête requête HTTP)	[RSG-19]
v	chaîne		`v%` où % est un nombre entier positif	Utilisé pour le versionnage de services (préférer l'indication de la version en tant que segment du chemin de l'URL)	[RSG-64]
limit	nombre entier	positif	`limit=10`	La taille de la page utilisée pour la pagination	[RSG-73]
offset	nombre entier	positif; 0 par défaut	`offset=5`	La déduction utilisée pour la pagination	[RSG-73]

sort	liste de chaînes séparées par une virgule	Valeurs possibles : – asc – desc	`sort=key1:asc,key2:desc`	Critère de tri multiattribut	[RSG-74] – [RSG-76]
expand	liste de chaînes séparées par une virgule		`expand=key1,key2`	Utilisé pour l'extension du corps du contenu renvoyé	[RSG-77]
coût	booléenne	Faux par défaut	`count=true`	Renvoie le nombre d'objets d'une collection (éventuellement dans la partie de la réponse qui contient la collection elle-même)	[RSG-81]
apiKey	chaîne		`apiKey=abcdef12345`	Utilisé pour indiquer une clé API Web (un en-tête HTTP devrait être préféré)	[RSG-137] – [RSG-138]

[L'annexe III suit]

ANNEXE III

DIRECTIVES CONCERNANT LA CONCEPTION D'API WEB RESTFUL ET LE CONTRAT DE SERVICE TYPE

Version 1.1

Révision approuvée par le Comité des normes de l'OMPI (CWS)
à sa dixième session le 25 novembre 2022

1. L'annexe III fournit deux modèles d'exemple de spécifications relatives aux API qui répondent à la norme et ont pour ambition de donner des orientations aux offices de propriété intellectuelle souhaitant mettre en place des services Web conformément à la présente norme. Les détails concernant les deux modèles sont exposés ci-dessous et dans les appendices A et B.

2. Il convient d'observer que les modèles d'exemple ont été établis selon une approche hybride combinant les approches du contrat en premier et du code en premier.

Modèle d'exemple DocList

3. Le premier des modèles d'exemple, DocList, a été inspiré par l'ensemble de services Web Open Portal Dossier (OPD) des offices IP5[17]. Le DocList API répertorie les documents de brevet associés à au moins un numéro de demande ou de publication.

Modèle d'exemple de situation juridique des brevets

4. Le deuxième des modèles d'exemple est l'API de la situation juridique des brevets, qui fournit soit l'historique des événements relatifs à la situation juridique d'un numéro de demande donné, soit des informations sur l'un de ces événements.

[Les appendices A et B de l'annexe III suivent]

[17] Les offices IP5 sont l'Administration nationale chinoise de la propriété intellectuelle (CNIPA), l'Office européen des brevets (OEB), l'Office des brevets du Japon (JPO), l'Office coréen de la propriété intellectuelle (KIPO) et l'Office des brevets des États-Unis d'Amérique (USPTO).

APPENDICE A

MODELE D'EXEMPLE DOCLIST

1. L'appendice A fournit un lien vers un fichier compressé comprenant le cahier des charges qui énonce les formats de requête et de réponse, la spécification du YAML et les éléments XSD développés.

2. L'appendice A est disponible à l'adresse suivante :
https://www.wipo.int/standards/en/st90/annex-iii_appendix_a_V1_0.zip

APPENDICE B

MODELE D'EXEMPLE DE LA SITUATION JURIDIQUE D'UN BREVET

1. L'appendice B fournit un lien vers un fichier compressé comprenant la spécification de l'API en RAML, les données relatives aux exemples et les listes d'énumération ST.96 de l'OMPI.

2. L'appendice B est disponible à l'adresse suivante :
https://www.wipo.int/standards/en/st90/annex-iii_appendix_b_V1_0.zip

[L'annexe IV suit]

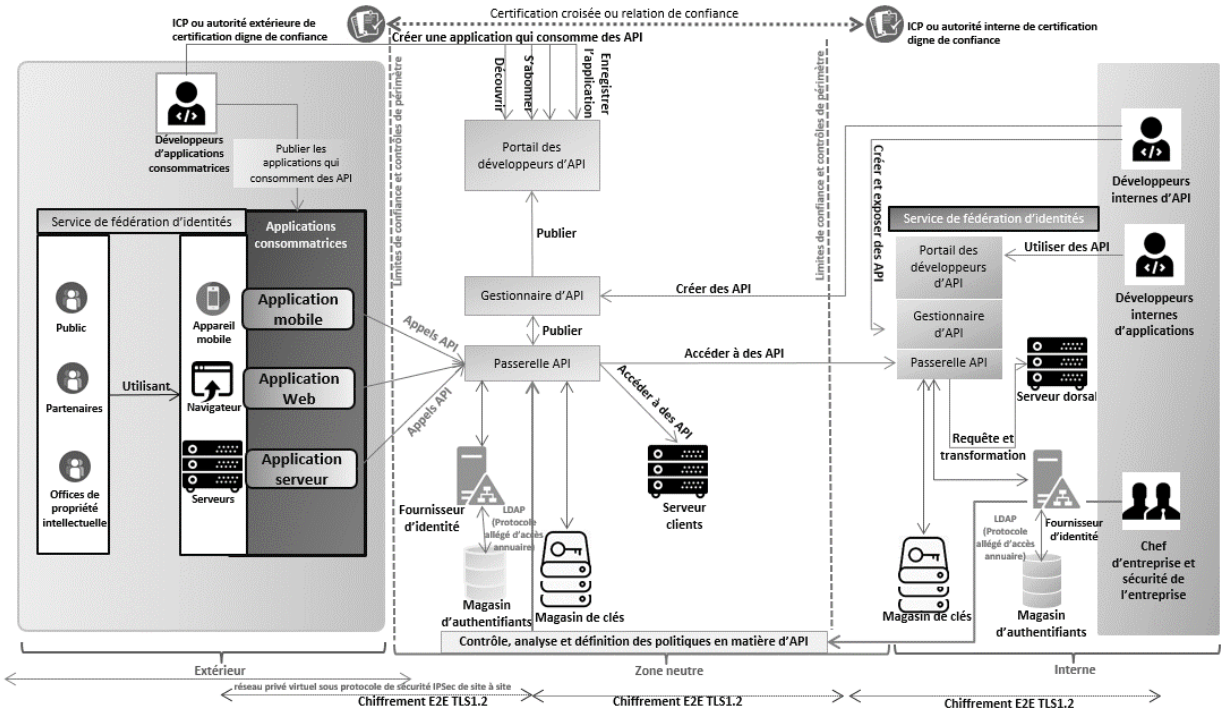
ANNEXE IV

MEILLEURES PRATIQUES EN MATIÈRE D'ARCHITECTURE DE SÉCURITÉ DE HAUT NIVEAU

Version 1.1

Révision approuvée par le Comité des normes de l'OMPI (CWS)
à sa dixième session le 25 novembre 2022

1. L'architecture de sécurité définit les services et les mécanismes qui devraient être mis en place pour appliquer des politiques et règles définies tout en offrant un cadre à la poursuite de la normalisation et de l'automatisation de la sécurité. Les services et mécanismes essentiels de ce cadre de sécurité des API (le portail des développeurs, le gestionnaire d'API et la passerelle API) regroupent les fonctionnalités. Ces fonctions peuvent être remplies par des applications discrètes et le développement de codes personnalisés en utilisant des produits disponibles sur le marché ou en tirant parti des technologies existantes qui peuvent être configurées pour fournir ces fonctions ou services. Certaines des fonctionnalités peuvent se recouper partiellement ou être combinées dans un ou plusieurs produits selon le fournisseur utilisé.



2. L'architecture de sécurité recommandée DEVRAIT comporter les services et mécanismes de sécurité des API ci-après :

- Un portail API Web fournissant des fonctionnalités comme la découverte des API, l'analyse des API, l'accès aux spécifications et à la description, notamment les accords de niveau de service, ainsi qu'au réseau social et à la foire aux questions;
- Un gestionnaire d'API Web assurant une administration et une gouvernance centralisée des API pour les catalogues d'API, la gestion de l'enregistrement et l'intégration des différentes communautés de développeurs d'API, la gestion du cycle de vie des API, l'application de profils de sécurité prédéfinis et la gestion du cycle de vie des politiques de sécurité;
- Une passerelle API Web fournissant des moyens d'automatisation de la sécurité, à savoir notamment une protection centralisée contre les menaces, une authentification centralisée des API, une autorisation, la journalisation, l'application des politiques de sécurité, le chiffrement des messages, le contrôle et l'analyse;

- Un service de contrôle et d'analyse des API Web fournissant des fonctionnalités comme le contrôle et l'analyse poussés des services d'API, l'utilisation de profils pour la définition d'une situation de référence en matière de sécurité, et les changements d'utilisation et de besoins;
- Un magasin d'authentifiants fournissant des moyens de sécurisation du stockage des clés API, des secrets, des certificats, etc.;
- Une autorité de certification digne de confiance chargée d'émettre des certificats sécurisés et de favoriser l'instauration de la confiance entre les différents offices;
- Un système de gestion de l'information et des événements de sécurité (SIEM) permettant la corrélation des journaux de sécurité et l'analyse et le contrôle poussés de la sécurité;
- Un fournisseur d'identité assurant la gestion des identités stockées dans les annuaires LDAP et permettant l'authentification; et
- Un produit de balayage d'applications Web qui effectue régulièrement des analyses de sécurité et établit d'autres analyses à partir de données de référence dignes de confiance telles que celles du projet OWASP Top 10.

[L'annexe V suit]

ANNEXE V

CODES D'ÉTAT HTTP

Version 1.1

Révision approuvée par le Comité des normes de l'OMPI (CWS)
à sa dixième session le 25 novembre 2022

1. Il importe de mettre les réponses en cohérence avec le code d'état HTTP approprié et de suivre les codes HTTP normalisés. Outre le code d'état approprié, le corps de votre réponse HTTP devrait contenir une description utile et concise de l'erreur. Les réponses devraient être claires et précises de manière que les consommateurs puissent en tirer rapidement une conclusion lorsqu'ils utilisent l'API.

2. L'ensemble de codes d'état HTTP est défini sur la base du RFC7231. Les codes d'état énumérés ci-dessous devraient être utilisés par une API, s'il y a lieu.

3. Les catégories de codes ci-après sont définies en fonction des catégories de réponse :

- 1xx : Informations – communique des informations au niveau du protocole de transfert;
- 2xx : Succès – indique que la requête du client a bien été acceptée;
- 3xx : Redirection – indique que le client doit prendre une mesure supplémentaire pour mener à bien sa requête;
- 4xx : Erreur client – Cette catégorie de code d'état d'erreur pointe du doigt le client; et
- 5xx : Erreur serveur – Le serveur assume la responsabilité de ces codes d'état d'erreur.

4. Le tableau ci-après regroupe les codes d'état HTTP et renvoie aux RFC de l'IETF correspondants.

Valeur	Description	Référence
100	Continue	[RFC7231, Section 6.2.1]
101	Switching Protocols	[RFC7231, Section 6.2.2]
102	Processing	[RFC2518]
103	Early Hints	[RFC8297]
104-199	Unassigned	
200	OK	[RFC7231, Section 6.3.1]
201	Created	[RFC7231, Section 6.3.2]
202	Accepted	[RFC7231, Section 6.3.3]
203	Non-Authoritative Information	[RFC7231, Section 6.3.4]
204	No Content	[RFC7231, Section 6.3.5]
205	Reset Content	[RFC7231, Section 6.3.6]
206	Partial Content	[RFC7233, Section 4.1]
207	Multi-Status	[RFC4918]
208	Already Reported	[RFC5842]
209-225	Unassigned	
226	IM Used	[RFC3229]
227-299	Unassigned	
300	Multiple Choices	[RFC7231, Section 6.4.1]
301	Moved Permanently	[RFC7231, Section 6.4.2]

302	Found	[RFC7231, Section 6.4.3]
303	See Other	[RFC7231, Section 6.4.4]
304	Not Modified	[RFC7232, Section 4.1]
305	Use Proxy	[RFC7231, Section 6.4.5]
306	(Unused)	[RFC7231, Section 6.4.6]
307	Temporary Redirect	[RFC7231, Section 6.4.7]
308	Permanent Redirect	[RFC7538]
309-399	Unassigned	
400	Bad Request	[RFC7231, Section 6.5.1]
401	Unauthorized	[RFC7235, Section 3.1]
402	Payment Required	[RFC7231, Section 6.5.2]
403	Forbidden	[RFC7231, Section 6.5.3]
404	Not Found	[RFC7231, Section 6.5.4]
405	Method Not Allowed	[RFC7231, Section 6.5.5]
406	Not Acceptable	[RFC7231, Section 6.5.6]
407	Proxy Authentication Required	[RFC7235, Section 3.2]
408	Request Timeout	[RFC7231, Section 6.5.7]
409	Conflict	[RFC7231, Section 6.5.8]
410	Gone	[RFC7231, Section 6.5.9]
411	Length Required	[RFC7231, Section 6.5.10]
412	Precondition Failed	[RFC7232, Section 4.2][RFC8144, Section 3.2]
413	Payload Too Large	[RFC7231, Section 6.5.11]
414	URI Too Long	[RFC7231, Section 6.5.12]
415	Unsupported Media Type	[RFC7231, Section 6.5.13][RFC7694, Section 3]
416	Range Not Satisfiable	[RFC7233, Section 4.4]
417	Expectation Failed	[RFC7231, Section 6.5.14]
418-420	Unassigned	
421	Misdirected Request	[RFC7540, Section 9.1.2]
422	Unprocessable Entity	[RFC4918]
423	Locked	[RFC4918]
424	Failed Dependency	[RFC4918]
425	Unassigned	
426	Upgrade Required	[RFC7231, Section 6.5.15]
427	Unassigned	
428	Precondition Required	[RFC6585]
429	Too Many Requests	[RFC6585]
430	Unassigned	
431	Request Header Fields Too Large	[RFC6585]
432-450	Unassigned	
451	Unavailable For Legal Reasons	[RFC7725]

452-499	Unassigned	
500	Internal Server Error	[RFC7231, Section 6.6.1]
501	Not Implemented	[RFC7231, Section 6.6.2]
502	Bad Gateway	[RFC7231, Section 6.6.3]
503	Service Unavailable	[RFC7231, Section 6.6.4]
504	Gateway Timeout	[RFC7231, Section 6.6.5]
505	HTTP Version Not Supported	[RFC7231, Section 6.6.6]
506	Variant Also Negotiates	[RFC2295]
507	Insufficient Storage	[RFC4918]
508	Loop Detected	[RFC5842]
509	Unassigned	
510	Not Extended	[RFC2774]
511	Network Authentication Required	[RFC6585]
512-599	Unassigned	

[L'annexe VI suit]

ANNEXE VI

TERMES DE REPRÉSENTATION

Version 1.1

Révision approuvée par le Comité des normes de l'OMPI (CWS)
à sa dixième session le 25 novembre 2022

Terme	Définition	Type de données
Montant	Une valeur monétaire.	`Nombre`
Catégorie	Dans un système de classification, division ou sous-ensemble spécifiquement défini dans lequel tous les éléments partagent le même concept de taxonomie.	`Chaîne`
Code	Une combinaison d'un ou plusieurs chiffres, lettres ou caractères spéciaux, qui est utilisée pour une signification spéciale. Représente des valeurs limites, déterminées à l'avance, ou en format libre.	`Chaîne`
Date	La notion d'un moment spécifique, désignée par l'année, le mois et le jour.	`Chaîne`
Répertoire	Toujours précédé de PATH (chemin d'accès)	`Chaîne`
Document	Un CLOB désigne un "Character Large Object", à savoir un type de données spécifique pour la quasi-totalité des bases de données. Pour dire les choses simplement, un CLOB renvoie à un texte stocké à l'extérieur du tableau dans un bloc dédié. Utilisé pour les documents en XML. Est composé d'informations textuelles échangées au sujet de l'enregistrement international de marques. Les balises XML identifient les données auxquelles se rapportent ces informations. L'équipe de développement TIS-Madrid peut définir l'attribut XML_DOC comme un CLOB, renvoyant à des données balisées stockées à l'extérieur du tableau dans un bloc dédié.	`Chaîne`
Identificateur	Une combinaison d'un ou plusieurs nombres entiers, lettres, caractères spéciaux qui identifie de manière unique une instance spécifique d'un objet commercial mais qui peut ne pas avoir une signification facile à définir.	`Chaîne`
Indicateur	Un signal de la présence, de l'absence ou de l'exigence de quelque chose. Les valeurs recommandées sont "Y", "N", et "?", le cas échéant.	`Booléen`
Mesure	Une mesure est une valeur numérique déterminée en mesurant un objet avec l'unité de mesure donnée. `MeasureType` est utilisé pour représenter un type de dimension physique comme la température, la longueur, la vitesse, la largeur, le poids, le volume et la latitude d'un objet. En termes plus précis, `MeasureType` devrait être utilisé pour mesurer les propriétés intrinsèques ou physiques d'un objet considéré comme un tout.	`Nombre`

Terme	Définition	Type de données
Nom	La désignation d'un objet exprimé en un mot ou en une phrase.	`Chaîne`
Nombre	Une série de chiffres ou de caractères alphanumériques désignant l'étiquette, la valeur, la quantité ou l'identification.	`Nombre, chaîne`
Pour cent	Un chiffre qui représente la partie d'un tout qui sera divisé par 100.	`Nombre`
Quantité	Une quantité est un nombre compté d'unités non monétaires, y compris éventuellement des fractions. `Quantity` est utilisé pour représenter un nombre compté de choses. `Quantity` devrait être utilisé pour des propriétés simples d'un objet considéré comme un composite, une collection ou un conteneur afin de quantifier ou de compter ses éléments. `Quantity` devrait toujours exprimer un nombre compté de choses et la propriété sera totale, expédiée, chargée, stockée. `QuantityType` devrait être utilisé pour les composantes nécessitant des informations sur les unités; et `xsd:nonNegativeInteger` devrait être utilisé pour les composantes comptables qui ne nécessitent pas d'informations sur les unités.	`Nombre`
Taux	Une quantité ou un montant mesuré en fonction d'une autre quantité ou d'un autre montant.	`Nombre`
Texte	Une chaîne de caractères non formatés, généralement sous la forme de mots (comprends les abréviations et commentaires).	`Chaîne`
Temps	La désignation d'un moment chronologique précis dans une période.	`Date`
Date et heure	La date et l'heure d'un événement.	`Date`
URI	L'Uniform Resource Identifier (identifiant uniforme de ressources) qui identifie l'endroit où se trouve le fichier.	`Chaîne`

[L'annexe VII suit]

ANNEXE VII

Publication concernant la gestion du cycle de vie des API

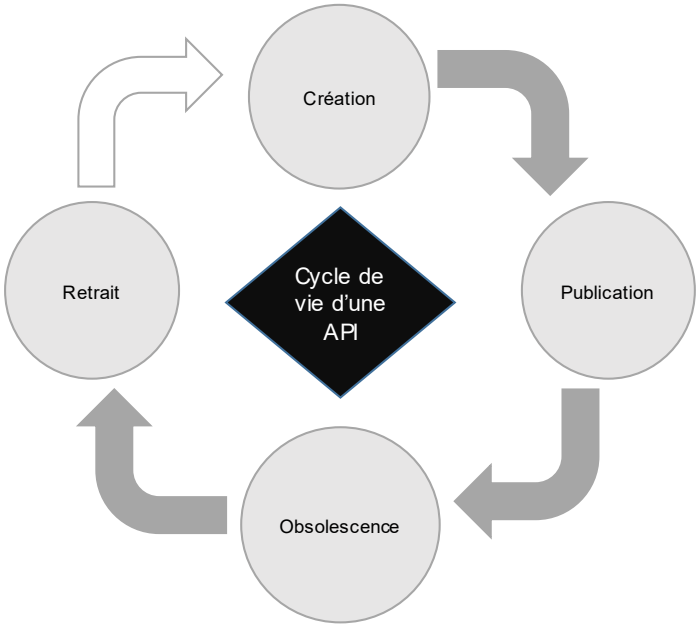
Version 1.1

Révision approuvée par le Comité des normes de l'OMPI (CWS)
à sa dixième session le 25 novembre 2022

1. La présente annexe décrit brièvement la gestion du cycle de vie des API et suggère les informations essentielles qui devraient être publiées par un office de propriété intellectuelle dans un document de politique afin d'aider les consommateurs d'API à comprendre comment utiliser au mieux ces API.

2. La gestion du cycle de vie des API constitue un aspect essentiel d'une stratégie en matière d'API, dans la mesure où elle fournit le cadre dans lequel s'inscrit une API depuis sa création jusqu'à son retrait. Elle est à la fois utile sur le plan interne pour les développeurs et les équipes opérationnelles et, à l'extérieur, pour les consommateurs d'API. En ce qui concerne les développeurs internes, la stratégie contribue à mettre en place une structure et à définir les attentes suscitées par la mise au point d'une API, tandis que, pour les équipes opérationnelles, elle aide à comprendre les besoins de support. Pour ce qui est des consommateurs d'API tant internes qu'externes, elle propose un contrat informel d'attentes relatives à l'utilisation d'une API particulière. Cela apparaîtra clairement dans la description de chaque étape du cycle de vie ci-dessous.

3. Les cycles de vie publiés des API peuvent se décomposer en processus simples en quatre étapes ou en processus complexes comportant 10 étapes ou plus. Toutefois, dans la plupart des cas, les cycles de vie comportant davantage d'étapes sont considérés comme des versions plus détaillées de ceux comptant moins de phases. Le présent document se concentrera sur le processus de base en quatre étapes, qui est nécessaire à la compréhension du cycle de vie d'une API : Créé -> Publié -> Obsolète -> Retiré. Tout document publié sur le cycle de vie d'une API devrait, à tout le moins, comprendre une description de ces quatre étapes, lesquelles sont gérées par un office de propriété intellectuelle.



Création

4. La création d'une API consiste essentiellement en la conception, l'implémentation et la documentation de l'API. L'élément capital à prendre en considération au cours de la phase de création est la finalité de l'API et la structure générale nécessaire pour la rendre aussi "résistante à l'avenir" que possible. Idéalement, l'API devrait répondre à un ensemble de

normes internes et externes, telles que les recommandations contenues dans la norme actuelle. Si l'API doit être monétisée, il conviendrait, à ce stade, d'établir la stratégie de monétisation.

Publication

5. Dès qu'une API est créée, elle doit être publiée. Elle devrait évoluer en suivant une stratégie de versionnage type et une documentation devrait être fournie et contenir la spécification de l'API et un échantillon de requêtes et de réponses (voir [RSG-64] et [RSG-65]. Une fois publiée, l'API est consommée par des applications. Il est à noter que des corrections et des améliorations peuvent être apportées au cours de la phase de publication.

Obsolescence

6. À un moment donné, une API n'est plus utile. Soit elle a été remplacée par une version plus récente, soit elle n'est plus pertinente en raison d'un facteur externe ou interne. Les consommateurs d'API devraient être contactés et il convient de préparer la suppression de l'API du catalogue. À ce stade, il est probable que seuls les bogues importants seront encore corrigés.

Retrait

7. Il s'agit de la phase où l'API est mise hors service. Cela doit inclure la désactivation de l'accès à l'API et sa suppression de la plateforme API. Il convient d'examiner la question de savoir si un "support étendu" sera proposé ou si le retrait sera retardé dans certains cas.

8. Les deux dernières étapes de l'obsolescence et du retrait sont celles qui doivent être le plus documentées en termes de gestion du cycle de vie. Il est essentiel que les consommateurs d'API comprennent ce que l'on attend d'eux lorsqu'ils commencent à utiliser une API afin d'éviter les désillusions ou les difficultés lorsque l'on cherche à supprimer une API du catalogue. Cela devrait inclure, par exemple, des versions majeures et mineures et des délais pour la notification des modifications. À haut niveau, on observe deux approches en matière d'obsolescence et de retrait des API : soit la conservation d'un nombre prédéfini de versions, soit la conservation de versions anciennes pendant une période donnée. Il est également possible de combiner ces deux approches, mais soit le nombre de versions anciennes qui doivent être supportées, soit la durée de conservation de ces versions anciennes, doivent être clairement mentionnés dans le document publié relatif au cycle de vie.

[Fin de l'annexe VII et de la norme]