Liberté • Égalité • Fraternité
RÉPUBLIQUE FRANÇAISE

MINISTÈRE
DE
L'INTÉRIEUR

Cadre de cohérence technique
du ministère de l'intérieur

Stratégie d'API

cctmi

Table des matières

Versions du document 1.1

Introduction 1.2

Les API 1.3

Choix technologique 1.4

Principes & règles

Principe n°1 2.1

règles 2.1.1

Principe n°2 2.2

règles 2.2.1

Principe n°3 2.3

règles 2.3.1

Principe n°4 2.4

règles 2.4.1

Principe n°5 2.5

règles 2.5.1

Principe n°6 2.6

Versions du document

Version du CCT	Date de modification du document	Auteurs
3.0	Juin 2017	Philippe Bron
3.0.6	février 2021	Philippe Bron, Hassan Driss
3.1.0	Juillet 2023	Hassan Driss
3.1.1	Août 2024	H Driss
3.1.2	Juillet 2025	H Driss
3.1.3	Septembre 2025	H Driss

Introduction

Sous l'impulsion de la Stratégie État Plateforme lancée par la DINUM, de la loi pour une République Numérique et de la loi 3DS, les systèmes d'information des ministères sont amenés à s'ouvrir. De même, la notion de plateforme incite à la mise en commun de données et de services de l'État dans un objectif de construction d'un écosystème d'acteurs publics ou privés. Ces derniers pourront alors les assembler pour construire ou rénover les services à destination des usagers de l'Administration.

La construction de cet État Plateforme repose sur l'utilisation d'interfaces de programmation applicative appelées API (Application Programming Interface). Cette technologie s'appuie sur des standards largement éprouvés par les géants du Web tels que Google, Amazon ou encore Twitter. Une telle plateforme permet la collaboration au sein de l'écosystème, mais également l'expérimentation de nouveaux services publics numériques dans des délais et des coûts réduits.

Ce document se veut un cadre pour la conception et la réalisation d'API. Celui-ci sera enrichi et adapté au rythme de la construction de l'État Plateforme et des retours des utilisateurs. Cette version pose les bases nécessaires aux premières réalisations. Elle n'a pas la prétention d'être exhaustive ni dans les réponses apportées, ni dans les sujets abordés.

Les API

Pour remplir cette promesse, les API mises en place dans le cadre de l'État Plateforme doivent pouvoir être assemblées et interopérer. Le respect des standards est un élément essentiel mais pas suffisant pour atteindre cet objectif. En effet, ils focalisent sur les aspects techniques sans explicitement adresser les besoins fonctionnels et métiers. Ce sont pourtant ces derniers qui donnent un sens à l'utilisation ou à la création d'une API.

C'est pour ces raisons qu'il est essentiel, en complément du respect des standards, de définir une stratégie claire des API répondant aux enjeux métiers. Pour cela, on s'attachera, lors de la conception d'une API, aux aspect fonctionnels et métiers des services fournis :

- format de la donnée facilitant son utilisation quelques soient les contextes
- sollicitation au fil de l'eau (intégration dans les processus des partenaires)
- adaptation des traitements en fonction des contextes des partenaires (plage d'ouverture, pics saisonniers …)
- contrôle des données transmises adaptées aux cadres légaux et aux besoins des partenaires
- …

Choix technologiques

La stratégie État Plateforme met en avant certains standards qui sont également utilisés dans les différentes API construites par des acteurs publics et disponibles sur data.gouv.fr :

- l'architecture REST et API RESTful pour l'appel et l'utilisation des API
- la spécification OpenAPI et le framework swagger pour la documentation des API
- le format JSON pour la structuration des données

Par ailleurs, AsyncAPI, en tant que standard pour la description des APIs asynchrones (évènements, messages, flux de données) peut être considéré comme un complément à cette stratégie.

Retour au sommaire

Chaque principe exposé ici est décliné en un ensemble de règles consultables. Ces principes visent à mettre en place un cadre partagé permettant d'inscrire l'ensemble des projets dans une dynamique API first.

Principe n°1 : une API doit exposer des services métiers et non des composants techniques

Un service métier repose sur l'utilisation d'une ou plusieurs ressources du système d'information. Ces dernières peuvent être de deux types :

- donnée : le service retourne de l'information à celui qui l'appelle;
- traitement : le service déclenche l'exécution d'un processus dans le SI. En fonction du contexte, le résultat peut être :
- l'état du processus : pris en compte, accepté, en cours, finalisé, rejeté …
- le résultat du traitement qui peut se matérialisé par :
- un élément physique : permis de conduire, acte d'état civil …
- un élément numérique

Le délai de retour n'est pas le même entre les ressources de type donnée et celles de type traitement. Les premières retournent l'information quasi instantanément alors que les secondes peuvent nécessiter plusieurs jours voire semaines.

Dans tous les cas, les ressources exposées doivent être documentées en prenant le point de vue fonctionnel du service :

- quel usage puis-je faire de cette API ?
- comment l'utiliser ?
- quelles sont les conditions d'utilisation ?
- inclure des exemples pour accompagner la compréhension.

De même, l'API doit respecter un certain nombre de règles dans sa définition :

- disposer d'une interface bien définie conforme aux règles de nommage
- fournir des données structurées et adaptées au métier faisant abstraction des ressources techniques sous-jacentes
- maîtriser les différentes versions disponibles en garantissant, par exemple, une compatibilité ascendante (cf. **principe 3**)

Afin de s'assurer du respect de ce premier principe, l'approche " Eat your own dog food " est un bon moyen pour :

- détecter les bugs;
- les corriger rapidement;
- améliorer l'utilisabilité de l'API;
- réaliser les premiers tests de charge;
- vérifier la bonne application des règles de sécurité. OWASP décrit ici le top 10 des règles de sécurité d'une API.

Enfin, mettre en place une stratégie « API first ». L'objectif est d'analyse le positionnement du projet dans l'écosystème d'API (du MI, de l'État Plateforme et/ou de partenaires). Cette phase doit permettre d'identifier, dès le lancement, les API disponibles utiles au projet et celles qui pourraient venir compléter l'écosystème.

Principe n°2 : veiller au découplage des API

Le respect de cette règle se traduit par la conception de services qui ne portent qu'une responsabilité et n'assurent qu'une seule fonction. Il convient de viser l'excellence du service rendu plutôt qu'une large couverture fonctionnelle. Pour cela, le service doit être idempotent et ne pas conserver de contexte des différents appels qu'il reçoit.

L'application de cette règle permet de construire un catalogue d'API modulaires et maintenables en limitant les dépendances et la redondance entre les différentes ressources exposées. Une politique de gestion des versions (cf. **principe 3**) assure également cette pérennité et cette maîtrise des évolutions.

Ainsi, une API expose un service qui réalise une fonction métier unique, claire avec des périmètres bien définis. Ce service peut être un traitement ou la fourniture de données et le résultat retourné est indépendant du nombre de fois où l'API est appelée.

Principe n°3 : les versions d'une API suivent la politique de gestion de versions de la DTNUM

En respect les **principes 1 et 2**, une API doit maintenir une certaine indépendance avec les systèmes techniques. Pour cette raison, les versions d'une API sont en lien avec des évolutions fonctionnelles ou techniques qui lui sont propres et ne reflètent en rien les évolutions du SI. Le respect de ce principe est facilité par un travail de conception basé sur les besoins métiers et une volonté de réutilisabilité maximale.

Pour des raisons de gestion, le nombre de versions différentes actives en même temps ne peut être infini. Toute nouvelle version doit supporter, autant que faire se peut, les fonctionnalités de la version précédente (on parle de compatibilité ascendante). La mise en place d'une nouvelle version ou la disparition d'une ancienne version doit s'accompagner d'un délai raisonnable permettant aux utilisateurs de prendre en compte ses évolutions.

Une politique de gestion de versions permet de :

- maintenir une adéquation entre besoins métiers et API exposées;
- garantir la pérennité des API;
- accompagner les utilisateurs dans les évolutions

Principe n°4 : sélectionner les API à exposer

Seuls les services répondant à des besoins métiers avérés ont vocation à être exposés, partagés. Le catalogue des API est le reflet des activités du ministère. Il doit donc être construit de manière réfléchie indépendamment des contingences techniques.

L'exposition d'API cohérentes et complémentaires permet :

- un couplage faible entre les API;
- d'éviter la redondance au niveau des API disponibles;
- maîtriser les interactions et limiter le risque d'enchaînement en cascade d'appels d'API

Principe n°5 : mettre en place des éléments de mesure, de supervision et de contrôle

L'exposition d'API implique d'être en capacité de mesurer, contrôler et superviser la fourniture de ces services. Pour cela, un ensemble de métriques techniques et métiers doivent être définis et mis en œuvre par chaque API pour :

- vérifier le respect des engagements (conventions);
- détection d'usage frauduleux;
- prendre des mesures de régulation en cas de non respect des engagements;
- limitation ou coupure de l'accès à l'API;
- mesurer les performances du service *Service Level Objectives*;
- mesurer le niveau du service *Service Level Agreement*;
- détecter les défaillances;
- identifier un besoin d'évolution et en mesurer l'efficacité

Principe n°6 : une API expose de la donnée et des traitements

Bien que dans un premier temps, les API soient principalement utilisées pour l'exposition de données, il ne faut pas les cantonner à ce cas d'usage. En effet, les API peuvent être utilisées pour déclencher des traitements nécessitant des délais de réalisation longs. Dans ces cas de figure, il est nécessaire de mettre en place des mécanismes adaptés à ces temps de traitement longs.

Ces mécanismes doivent permettre à l'émetteur de l'appel à l'API de recevoir le résultat dans un échange différent de son appel. Ils mettent en œuvre les principes de :

- callback : l'utilisateur indique à l'API la localisation du système par lequel le résultat doit lui être transmis. Il s'agit généralement d'une API dédiée à la réception des retours.
- corrélation : l'utilisateur et le fournisseur de l'API s'entendent sur un identifiant unique permettant de corréler le résultat transmis à un précédent appel.

En complément, des mécanismes de suivi de l'avancement de la réalisation des traitements et de reprise en cas d'incident doivent être mis en place pour informer les utilisateurs de l'API.

Les règles définies dans cette partie se veulent une déclinaison opérationnelle des principes définis précédemment. Ces différentes règles ont été mises à jour en reprenant pour partie le guide API publié par le Government Digital Service (GDS) anglais, ainsi que certains éléments référencés dans le guide de design d'API d'Octo.

Principe n°1 : une API doit exposer des services métiers et non des composants techniques

Règle 1.1

Une API a pour vocation de mettre à disposition des données aux plus grand nombre de partenaires légitimes à en faire usage. Le contrôle et le filtrage de l'accès à une API doit donc être adapté à cet objectif de diffusion.

L'utilisation d'adresses IP fixes et/ou d'authentification mutuelle par certificats limite fortement la multiplication des utilisateurs. Ces mécanismes sont donc à réserver dans les contextes pour lesquels ils sont indispensables.

Règle 1.2

Une API expose des données qui ont une pertinence métier en dehors du périmètre de l'application qui les héberge.

Une API ne doit pas être utilisée pour les besoins d'intégration d'applications, mais pour délivrer un service qui est indépendant des cas d'usage dans lesquels il est appelé.

Règle 1.3

Une API prend en compte dès la conception sa capacité à passer à l'échelle sur l'ensemble des composants contribuant à son fonctionnement. Cela concerne les infrastructures réseau, l'API management, mais également les serveurs de base de données ou applicatifs traitant la demande transmise via API. Cette règle vise à maintenir dans le temps les niveaux de qualité de service et d'engagement lorsque la demande augmente.

Principe n°2 : développer des API modulaires

Règle 2.1

L'exposition des données se fait indépendamment des contraintes techniques de l'application (format de données, technologie …) en prenant en compte la pertinence métier.

Une API ne doit pas exposer le modèle de données d'une application sauf à ce que ce dernier soit identique aux objets métiers (ce qui est rarement le cas).

Règle 2.2

JSON (JavaScript Object Notation) doit être privilégié pour la manipulation de données (envoi ou réception) au travers d'une API.

L'utilisation d'une autre représentation doit être exceptionnelle et limitée à certains cas tels que :

- l'obligation de se connecter à un système hérité, par exemple, un système qui utilise uniquement XML
- pouvoir bénéficier d'avantages significatifs en respectant des standards largement adoptés dans l'organisation (par exemple, SAML)

En dehors de ces exceptions, nous recommandons de :

- créer des réponses en tant qu'objet JSON et non en tant que tableau (les objets JSON peuvent contenir des tableaux JSON). En effet, les tableaux peuvent limiter la possibilité d'inclure des métadonnées sur les résultats et limiter la capacité de l'API à ajouter des clés de niveau supérieur supplémentaires à l'avenir
- documenter votre objet JSON pour s'assurer qu'il est bien décrit et pour qu'il ne soit pas traité comme un tableau séquentiel. Chaque information est directement accessible sans avoir à parcourir une structure de données.

- éviter les clés d'objet non déterministes telles que celles dérivées des données, car cela peut amener des conflits avec les utilisateurs de l'API
- utiliser une grammaire cohérente pour les clés des objets. Préférer les notations utilisant les types de style *spin-case*, *snake_case*, ou *CamelCase* en vaillant surtout à être cohérent.

Règle 2.2.1

Le format GeoJSON doit être prévilégié dans le cadre d'échange d'information de localisation géographique. Ce format est référencé dans Référentiel Général Interopérabilité (RGI V2).

Règle 2.2.2

La norme Unicode Transformation Format (UTF-8) est à adopter lors du codage de texte ou d'autres représentations textuelles de données. Cette norme d'encodage est référencée dans Référentiel Général Interopérabilité (RGI V2).

Règle 2.2.3

La représentation du format d'une date ou d'une date et d'une heure doit respecter la norme ISO 8601 standard ainsi le traitement en sera internationalement facilité

Ainsi nous pourrions avoir pour une date `2017-08-09` et pour une date et heure `2017-08-09T13:58:07Z` .

Règle 2.3

Configurez les API pour qu'elles répondent aux « demandes » de données plutôt qu'à « envoyer » ou « pousser » des données. Cela garantit que l'utilisateur de l'API ne reçoit que les informations dont il a besoin.

Lors de la réponse, votre API doit répondre à la demande de manière complète et spécifique. Par exemple, une API doit répondre à la demande « Cet utilisateur est-il marié? » avec un booléen. La réponse ne doit pas renvoyer plus de détails que nécessaire et doit s'appuyer sur l'application cliente pour l'interpréter correctement.

Par exemple :

```
{ "married": "true" }
```

Au lieu de :

```
{

"person": {
"name": "Alice Betterland",
"dob": "1999-01-01",
"married": false,

"validFrom":"2011-04-03",

"validTo":""
}

}
```

Règle 2.4

Concevez les champs de données en tenant compte des besoins des utilisateurs. Lors de la conception de vos champs de données, vous devez considérer comment les champs répondront aux besoins des utilisateurs. Avoir un architecte API dans votre équipe peut vous aider à le faire. Vous pouvez également tester régulièrement votre documentation auprès de vos utilisateurs.

Par exemple, si vous avez besoin de recevoir des informations personnelles, avant de décider de la structure de la réponse, vous devrez peut-être déterminer si :

- la conception peut accepter des informations d'identité provenant de région où il n'y a pas de nom ni de prénom
- l'abréviation DdN a du sens ou s'il est préférable d'utiliser le terme `date de naissance`
- DdN a du sens lorsqu'il est combiné avec DdDC (date du décès)

Vous devez également vous assurer de fournir toutes les options pertinentes. Par exemple, le champ « civilStatus » contient probablement plus de 2 états que vous souhaitez enregistrer : `marié` , `non marié` , `divorcé` , `veuf` , `séparé` , `annulé` , etc.

Selon ce que vous décidez, vous pouvez choisir la structure de données suivante comme réponse :

```
{
"person": {
"firstName": "Alice",
"lastName": "Wonderland",
"dob": "1999-01-01",
"civilStatus": "marié",

"validFrom":"2010-03-12",

"validTo":"2011-04-03"
},

"person": {
"firstName": "Alice",
"lastName": "Betterland"
"dob": "1999-01-01",
"civilStatus": "divorcé",

"validFrom":"2011-04-03",

"validTo":""
}

}
```

Règle 2.5

Il convient d'être au plus proche des normes et standards utilisés dans l'industrie. Pour cette raison il est fortement recommandé de créer des API RESTful, qui utilisent des requêtes de verbe HTTP pour manipuler les données.

Ainsi,lors du traitement des demandes, vous devez utiliser les verbes HTTP aux fins spécifiées. L'un des avantages de REST est qu'il offre également un cadre pour la communication des états d'erreur.

De manière plus spécifique, une API RESTful doit respecter les directives suivantes :

- stateless : en ce sens il n'associe pas de contexte à un appel
- utilisation des verbes HTTP :
  - POST pour la création
  - GET pour la consultation/lecture
  - PUT pour la mise à jour
  - DELETE pour la suppression
- utilisation des codes retour HTTP suivants :

Code	Message	Description
102	Processing	Traitement en cours (évite que le client dépasse le temps d'attente limite).
200	OK	Code de retour par défaut en cas de succès
201	Created	Code retour en cas de succès du traitement et de la création d'une nouvelle ressource
202	Accepted	Indique que la requête a bien été prise en compte et sera traitée ultérieurement. Ce mode de fonctionnement est principalement utilisé dans le cas d'échanges asynchrones et nécessite la mise en place d'un mécanisme de Call Back côté client
204	No Content	Indique que la requête a bien été traitée mais qu'il n'y a pas de résultat à retourner. C'est par exemple le cas lors d'action de suppression
206	Partial Content	Indique qu'une partie de la ressource a été transmise. Utilisé en cas de pagination
304	Not Modified	Document non modifié depuis la dernière requête.
400	Bad Request	Code d'erreur générique en cas d'informations non valides fournies au service
401	Unauthorized	Code utilisé par les services nécessitant une autorisation lorsque l'identification fournie n'est pas autorisée à utiliser le service
402	Unprocessable entity	Code de retour générique lorsque la requête ne peut être traitée suite à des paramètres d'entrée non valides
404	Not Found	Code d'erreur en cas d'URI d'entrée inexistante ou d'information non trouvée
405	Method not Allowed	Code d'erreur en cas d'incompatibilité entre une URI et une méthode
406	Not Acceptable	Code de retour lorsque les entêtes ne semblent pas compatibles avec le fonctionnement du service
409	Conflict	La requête ne peut être traitée en l'état actuel. Ce code peut être utilisé afin d'indiquer une détection de doublon lors d'une création ou mise à jour par exemple
429	Too Many Requests	Code de retour lorsque le client émet trop de requêtes dans un délai donné
500	Server Error	Code d'erreur par défaut
503	Service Unavailable	Code de retour lorsque le service n'est pas disponible

Règle 2.6

Vous devez utiliser le protocole HTTPS lors de la création d'API. L'ajout de HTTPS sécurisera les connexions à votre API, préservera la confidentialité des utilisateurs, garantira l'intégrité des données et authentifiera le serveur fournissant l'API.

Sécurisez les API à l'aide de Transport Layer Security (TLS) v1.2. N'utilisez pas Secure Sockets Layer (SSL) ou TLS v1.0. L'obtention de certificats peut être réalisée en utilisant les processus en vigueur dans les différentes organisations.

Assurez-vous que les utilisateurs potentiels de l'API peuvent vérifier vos certificats. Assurez-vous de disposer d'un processus solide pour le renouvellement et la révocation des certificats en temps opportun.

Point à adresser

Votre API peut établir des liaisons avec d'autres données. Vous pouvez rendre votre API plus accessible par programme en renvoyant des URI et en utilisant les normes et spécifications existantes. Pour cela, utilisez des identificateurs de ressources uniformes (URI) pour identifier certaines données :

```
{
  "name": "Bob Person",
  "company": "https://your.api/company/bobscompany";
}
```

Règle 2.7

Lorsque vous fournissez une API Open Data, vous devez permettre aux utilisateurs de télécharger des ensembles de données entiers à moins qu'ils ne contiennent des informations sensibles. Cela offre aux utilisateurs :

- la possibilité d'analyser le jeu de données localement
- l'assistance lors de l'exécution d'une tâche nécessitant l'accès à l'ensemble des données (par exemple, tracer un graphique sur les zones de chalandise scolaire en Angleterre)

Les utilisateurs doivent pouvoir indexer leur copie locale des données en utilisant la technologie de base de données de leur choix, puis effectuer une requête pour répondre à leurs besoins. Cela signifie que les futures indisponibilité de l'API ne les affecteront pas, car ils disposent déjà de toutes les données dont ils ont besoin.

L'utilisation d'une requête API permettant la récupération enregistrement par enregistrement pour effectuer la même action ne serait pas optimale, à la fois pour l'utilisateur et pour l'API. Ceci est dû au fait :

- des limites de débit qui ralentiraient l'accès, ou pourraient même empêcher le téléchargement complet de l'ensemble de données
- si l'ensemble de données est mis à jour en même temps que le téléchargement enregistrement par enregistrement, les utilisateurs peuvent obtenir des enregistrements incohérents

Si vous autorisez un utilisateur à télécharger un ensemble de données complet, vous devez envisager de lui fournir un moyen de le maintenir à jour. Par exemple, vous pouvez diffuser vos données en direct ou les informer que de nouvelles données sont disponibles afin que les consommateurs d'API puissent venir télécharger vos données périodiquement.

Règle 2.8

N'encouragez pas les utilisateurs à mettre à jour de grands ensembles de données en les retéléchargeant, car cette approche est inutile et peu pratique. Au lieu de cela, laissez les utilisateurs télécharger des listes incrémentielles de modifications apportées à un ensemble de données. Cela leur permet de garder leur propre copie locale à jour et leur évite d'avoir à télécharger à nouveau l'ensemble de données à plusieurs reprises.

Il n'existe pas de norme recommandée pour ce modèle. Les utilisateurs peuvent donc essayer différentes approches telles que:

- l'encodage des données dans les flux Atom / RSS
- en utilisant des modèles, tels que les flux d'événements utilisés par des produits tels que Apache Kafka
- en se basant sur la spécification asyncAPI qui est particulièrement utile pour les cas d'utilisation où les réponses ne sont pas immédiates ou où les interactions doivent être traitées de manière asynchrone.

- utiliser des registres de données ouverts

Règle 2.9

Lorsque vous publiez des données en masse, rendez ces données disponibles aux formats CSV et JSON. Cela garantit que les utilisateurs peuvent utiliser un large éventail d'outils, y compris des logiciels standard, pour importer et analyser ces données.

Publiez des données en masse sur data.gouv.fr et assurez-vous qu'il existe un lien bien en vue vers votre API.

Règles 2.10

Les noms de domaine des API doivent suivre les recommandations suivantes :

- utiliser des noms plutôt que des verbes
- être bref, simple et compréhensible
- être intuitif pour un humain en évitant, si possible, les termes techniques ou spécialisés
- utiliser des tirets plutôt que des underscores pour séparer les mots lorsque que le nom est composé de plusieurs mots. Par exemple `api-name.interieur.gouv.fr`

Règle 2.11

Les noms de domaine utilisés dans la construction d'une API doivent se limiter à trois sous-domaines en production. L'objectif est de maintenir intuitive la compréhension de l'URL à utiliser pour respectivement :

- utiliser l'API : `https://api.{nomressource}.interieur.gouv.fr`
- récupérer un jeton d'authentification : `https://oauth2.{nomressource}.interieur.gouv.fr`
- consulter la documentation de l'API : `https://developpeurs.{nomressource}.interieur.gouv.fr`

Dans le cas, par exemple, d'interrogation du Système d'Immatriculation des véhicules on aurait :

- `https://api.immatriculation.interieur.gouv.fr`
- `https://oauth2.immatriculation.interieur.gouv.fr`
- `https://developpeurs.immatriculation.interieur.gouv.fr`

Règle 2.12

Eviter les espaces de nommage (namespaces) en privilégiant un nom de domaine par API tout comme pour les services numériques. Cela permet de mélanger les API et facilite la gestion des versions.

Règle 2.13

Les ressources visées étant des collections ou une instance parmi une collection, il est donc préférables de toujours utiliser le pluriel pour ces collections.

Cette règle peut se traduire par :

- *retourne une collection de ressources (toutes les immatriculations)*
  - **GET** `https://api-name/v1/immatriculations`
- *retourne une ressource unique (les informations pour l'immatriculation fournie)*
  - **GET** `https://api-name/v1/immatriculations/AB-123-XZ`

Règle 2.14

L'objet de cette règle est donc de définir les conventions devant être respectées dans la construction des URI d'une API.

Pour mémoire, le principe de fondateur des API RESTful est de manipuler des ressources identifiées au travers d'une URI (*Uniform Resource Identifier*). Afin de faciliter la réutilisation des API, il est donc important d'accéder de manière homogène aux URI utilisés dans les services.

racine du service

La racine de l'URI débute par l'URL (*Uniform Resource Locator*) de base du serveur web exposant l'API tel que défini par la Règle 2.3.

version de l'API

L'élément suivant indique le numéro de version de l'API. Celui-ci commence par v et est suivi d'un numéro.

En fonction de la politique de gestion des versions, le numéro de version pourra contenir des versions mineures en étant codé sur 2 digits.

classement

Si la ressource visée est organisée suivant un classement permettant de différencier des ressources ayant le même nom, ce critère de classement figure après le numéro de version. Ce classement peut être une date, un département ou tout autre élément permettant de regrouper des ressources.

Si il existe plusieurs niveaux d'imbrication des classements, ceux-ci sont insérés dans l'URI en partant du grain le plus gros.
Par exemple, une déclaration fiscale est organisée autour d'une année de déclaration, une préfecture autour d'un département.

Cette règle s'applique également aux ressources nécessitant d'autres ressources pour construire l'URI. C'est le cas notamment lorsqu'il y a une imbrication fonctionnelle entre des objets métiers. Nous pouvons prendre le cas des bibliothèques et des livres à titre d'illustration :

- la liste de toutes les bibliothèques :
  - **GET** `https://api-name/v1/bibliotheques`
- la liste de tous les livres :
  - **GET** `https://api-name/v1/livres`
- la liste de tous les livres de la BNF :
  - **GET** `https://api-name/v1/bibliotheques/bnf/livres`
- la liste des bibliothèques où on peut trouver le livre « Dom Juan » :
  - **GET** `https://api-name/v1/livres/domjuan/bibliotheques`

nom de la ressource

Enfin, le nom désignant la ressource manipulée au travers de l'API est le dernier élément de la chaîne constituant l'URI.

paramètres de la requête

L'utilisation des paramètres de la requête permettent la pagination, le filtrage, le tri sur les données d'une ressource.

Pagination

Il est nécessaire de prévoir des fonctionnalités de pagination de ressources afin de maitriser la quantité d'informations qui sera présentée à chaque appel.

Pour ce faire, il est utile de préciser :

- du point de vue du consommateur (ou réutilisateur):
  - la pagination souhaitée au moment de l'appel à la ressource
- du point de vue du producteur de l'API :
  - le nombre d'éléments maximum renvoyé pour la ressource;
  - l'interval entre la première page et la dernière page renvoyée;
  - le nombre d'éléments total de la ressource.

Pour cela, la règle est traduite comme suit en prenant comme exemple une liste d'associations avec une pagination du premier au dixième élément.

> ℹ Un exemple inspiré du guide du design des API rédigé par Octo.

```
GET https://api-name/v1/associations?range=1-10

< HTTP/1.1 200 OK
< Accept-Range: 50
< Content-Range: 1-10/600
```

Tri

La fonctionnalité de tri comme de *filtrage* ou de *recherche* impacte la fonctionnalité de pagination. Pour répondre à cette fonctionalité 2 paramètres son utilisés `sort` et `desc` :

- sort : indique les attributs métiers à trier. Si plusieurs attributs sont concernés, ils doivent être séparés par une virgule `sort=theme,titre`
- desc: indique le sens du tri descendant pour les attributs métiers souhaités. Par défaut le tri est ascendant.

```
GET https://api-name/v1/associations?range=1-70&sort=id_association&desc=id_association

< HTTP/1.1 200 OK
< Accept-Range: 50
< Content-Range: 1-50/600
```

Filtrage

Le filtrage permet de limiter le nombre d'élément renvoyé pour une ressource donnée, en spécifiant des attributs et leurs valeurs attendus. Il est possible de filtrer une collection sur plusieurs attributs simultanément, et de permettre plusieurs valeurs pour un même attribut filtré en utilisant comme séparateur la virgule.

```
    GET  "https://api-name/communes?codeDepartement=13,15&format=json&geometry=centre&fields=nom,code,codeDepartement,departem
ent"
```

L'exemple cité permet de filtrer non seulement l'ensemble des occurences souhaité en utilisant les attributs et valeurs choisis (ex: *code_departement*), mais également en précisant uniquement les attributs souhaités via le paramètre `fields` (ex: *nom,code,codeDepartement,departement*).

> l'usage du paramètre `fields` permet, à l'issue de l'appel, de ne renvoyer que les informations nécessaires que sont *nom*, *code*, *codeDepartement* et *departement*.

Recherche globale

Il s'agit de permettre une recherche approchante sur un ensemble d'attributs, de champs liés à la ressource et qui seront les plus pertinents sur le plan métier. Pour se faire la notation Google est utilisée.

Cette règle est traduite comme suit :

```
    GET `https://api-name/v1/associations/search?q=asso`
```

Le nombre d'occurrence d'informations pouvant être important, il convient d'en limiter le nombre retourné avec une valeur par défaut (ex: `limit=5`) modifiable jusqu'à une valeur maximale que vous considérez adapter au contexte :

```
    GET `https://api-name/v1/associations/search?q=asso&limit=5`
```

Règle 2.15

Vous devez fournir une documentation permettant de faciliter l'appropriation et l'usage de votre API et permettre à vos consommateurs de démarrer rapidement. Pour cela, vous devez :

- utiliser la Spécification OpenAPI 3 le cas échéant pour générer la documentation (recommandé par l'Open Standards Board)

- générer une documentation suivant le formalisme Swagger
- fournir un exemple de code pour illustrer comment appeler l'API et pour indiquer aux utilisateurs les réponses auxquelles ils peuvent s'attendre

Vous devez également inclure dans la documentation :

- les informations contextuelles / générales - ce que fait l'API, à qui elle s'adresse et suivant quelles circonstances
- les règles contractuelle et des données - dans quelles circonstances les données sont-elles disponibles / non disponibles
- les scénarios d'erreur - pré-conditions et résultats - y compris les codes d'erreur et les messages
- les détails sur le service de test - comment l'utiliser et comment simuler les différents scénarios de réussite et d'erreur
- tous les détails des paramètres de requête et de réponse, y compris la signification, le type de données et toute autre contrainte. Donnez des exemples de valeurs valides.
- les règles relatives au traitement des informations, à la gestion des incidents et à la gestion des risques
- la méthode d'authentification en place (et son impact sur l'interopérabilité des services, l'authentification unique(SSO) et la limitation du débit (throttling)
- toutes les règles d'autorisation, par exemple, l'utilisation d'OAuth 2.0 et spécifiquement les scopes requis pour cette API
- les modifications de conception (récentes et prévues) et les informations de version
- la disponibilité, la latence, la propriété, la politiques de dépréciation et l'état de la capacité
- l'approche permettant la compatibilité descendante de l'API
- des conseils sur la configuration de l'API pour s'assurer que toutes les exigences sont suivies
- le coût d'utilisation, le cas échéant

Vous devez toujours vous assurer que votre documentation est claires et communiquer lorsque des modifications sont apportées.

Principe n°3 : les versions d'une API suivent la politique de gestion de versions de la DTNUM

Règle 3.1

Une API peut supporter au plus deux versions en même temps. Dès la mise en œuvre de la nouvelle version que l'on nommera « Vn+1 », la version « Vn » ne doit pas être disponible au-delà de 18 mois. Ce délai doit permettre une migration de la version « Vn » vers la version « Vn+1 » en toute quiétude.

Règle 3.2

Une feuille de route est associée à chaque API. Celle-ci doit comporter a minima les informations suivantes :

- la liste des futures versions prévues ainsi que les dates de mise en service associées
- la liste des nouvelles fonctions ainsi que celles qui ne seront plus supportées
- les évolutions prévues dans chaque version en précisant les impacts éventuels sur la version précédente
- le point de contact pour toute question relative à cette feuille de route

Cette information doit faire l'objet d'une diffusion auprès des utilisateurs. Des canaux classiques comme la publication sur la page de présentation de l'API ou de mailing sont à prévoir. Il est également possible d'indiquer les éléments qui ne seront plus supportés en utilisant l'entête `Deprecation` et `Sunset` (RFC 8594) des réponses HTTP.

Règle 3.3

Lors de la publication d'une nouvelle version d'une API, il faut minimiser au maximum les impacts pour les utilisateurs actuels afin de ne pas générer de surcoût pour leur permettre de prendre en compte cette nouvelle version.

Pour cela, il faut privilégier :

- des modifications rétrocompatibles lorsque cela est possible - précisez aux parsers d'ignorer les propriétés qu'ils ne s'attendent pas ou ne comprennent pas pour garantir que les modifications sont rétrocompatibles (cela permet d'ajouter des champs pour mettre à jour les fonctionnalités sans nécessiter de modifications de l'application cliente)

- la mise à disposition d'un nouvel endpoint pour des changements importants
- la documentation des endpoints obsolètes

De nouveaux endpoints ne sont pas toujours nécessaires pour la fourniture de nouvelles fonctionnalités si ils permettent de maintenir la compatibilité descendante.

Règle 3.4

Lorsqu'il n'est pas possible de maintenir une compatibilité avec les versions précédentes, les évolutions de l'API doivent être publiées en prenant en compte :

- l'incrément du numéro de version qui doit apparaître dans l'URL en commençant pas `/v1/`
- le support les endpoints des anciennes versions conformément à la règle 3.1
- l'information des utilisateurs pour leur indiquer comment valider les données. En leur indiquant, par exemple, les champ qui ne seront plus présents afin qu'ils puissent s'assurer que leurs règles de validation traiteront ce champ de manière optionnelle
- la fourniture d'un nouvel objet lorsqu'il est nécessaire de modifier la structure d'un objet complexe. Cela peut être le cas lorsque l'on souhaite combiner les données provenant de plusieurs objets. Dans ce cas le nouvel objet sera exposé au travers d'un nouvel endpoint
  - combiner les données d'un utilisateur `/v1/users/123` et de son compte `/v1/accounts/123` dans un nouvel objet `/v1/consolidated-account/123`

Règle 3.5

Afin de permettre aux consommateurs de l'API de tester leur application dans son usage de l'API, celle-ci doit proposer un service de test. En fonction des cas de figure, cette API de test pourra prendre en compte différentes contraintes :

- si l'API est en lecture seule sur des ressources, il n'est pas indispensable de prévoir une API de test
- si l'API a un comportement complexe ou avec état, envisagez de fournir un service de test qui imite autant que possible le service en direct, mais gardez à l'esprit le coût de cette opération
- si l'API nécessite une autorisation, l'API de test doit inclure ce mécanisme ou offrir plusieurs niveaux de service de test en fonction des niveaux d'autorisation possibles

Pour que ce service de test soit le plus efficace possible, il est conseillé de réaliser une enquête auprès des consommateurs qui pourront ainsi préciser leurs attentes.

Principe n°4 : sélectionner les API à exposer

Règle 4.1

Pour ce qui est des API permettant d’exposer des données sous responsabilité du ministère de l’intérieur, l’identification de ces données de références se fait en collaboration avec l’administrateur ministériel des données (*AMD*).

Règle 4.2

Pour ce qui concerne les API exposant des traitements, l’identification et la validation des API devant être mises en place se font en collaboration avec les directions métiers visées par le règlement portant ce traitement.

Principe n°5 : mettre en place des éléments de mesure, de supervision et de contrôle

Règle 5.1

Si votre API sert des données personnelles ou sensibles, vous devez journaliser la date à laquelle les données sont fournies et à qui. Cela contribura à apporter une réponse aux exigences du règlement général sur la protection des données (RGPD), à répondre aux demandes d'accès des personnes concernées et à détecter les fraudes ou les abus.

Règle 5.2

Privilégiez un accès sans contrôle si vous souhaitez donner un accès sans entrave à votre API et que vous n'avez pas besoin d'identifier vos utilisateurs. C'est le cas par exemple lorsque vous fournissez des données ouvertes. Cependant,il faut garder à l'esprit le risque d'attaques par déni de service.

Attention, le libre accès ne signifie pas que vous ne pouvez pas limiter l'usage de votre API.

Règle 5.3

L'authentification des consommateurs est requise lorsque l'on souhaite :

- limiter le débit (ou throttling)
- auditer l'usage
- facturer les consommateurs
- contrôler l'accès en fonction d'autorisations accordées aux consommateurs

En fonction du ou des objectifs, les exigences de sécurité de la solution d'authentification seront différentes.

Par exemple, si vous devez identifier les utilisateurs uniquement à des fins de limitation de débit, vous n'aurez peut-être pas besoin d'actualiser les jetons utilisateur très souvent, car un jeton entre de mauvaises mains ne menacera probablement pas votre service.

Utilisez l'autorisation au niveau de données applicatives si vous souhaitez contrôler quelles applications peuvent accéder à votre API, mais pas quels utilisateurs finaux spécifiques. Cela convient si vous souhaitez utiliser la fonctionnalité de limitation de débit, d'audit ou de facturation. Cependant, elle ne sera sans doute pas adaptée aux API contenant des données personnelles ou sensibles à moins que vous ne fassiez vraiment confiance à vos consommateurs. Si il s'agit, par exemple, d'un autre service gouvernemental.

Règle 5.4

Il existe deux grandes familles de consommateurs d'une API :

- anonymes, ne disposant pas d'identifiant et dont l'usage de l'API ne peut être imputé à un acteur. Dans ce cas, seul un contrôle global peut être assuré (cf. Règle 5.4).
- enregistré, disposant d'un identifiant unique permettant de leur imputer l'usage d'une API. Dans ce cas, un contrat d'usage a été validé lors de l'obtention de l'identifiant. Un contrôle particulier peut alors être mis en place.

Bien qu'il y ait une forte affinité entre consommateurs anonymes et API ouvertes, cette association n'est pas exclusive. Il peut être nécessaire d'obtenir un élément d'identification pour utiliser une API ouverte. Dans ce cas, le contrôle ne portera pas sur l'éligibilité de l'acteur, mais sur l'usage des ressources.

De manière générale, les ressources mises en oeuvre pour un usage anonyme doivent être limité et donc se refléter dans le contrat d'usage avec des niveaux de qualité de service et de volume restreints.

Règle 5.5

Utilisez l'autorisation au niveau de l'utilisateur si vous souhaitez contrôler les utilisateurs finaux qui peuvent accéder à votre API. Cela convient pour traiter des données personnelles ou sensibles.

OpenID Connect (OIDC), qui s'appuie sur OAuth2, avec son utilisation de JSON Web Token (JWT), est utilisé par France Connect et peut convenir dans certains cas. Il faudra néanmoins prévoir un dispositif OIDC interne pour permettre à des utilisateurs inconnus des Fournisseurs d'Identité d'utiliser l'API.

Règle 5.6

L'usage d'une API doit être identifié par un élément unique valide pour une période de temps fixée. Ce dernier doit permettre le suivi de l'usage et la vérification du respect des clauses du contrat. Il peut reposer sur une information spécifique ou être une combinaison d'éléments permettant cette unicité. L'usage des jetons et des autorisations répond à ce besoin et suit les bonnes pratiques suivantes :

- choisissez une fréquence d'actualisation et une période d'expiration appropriées pour vos jetons d'accès utilisateur - le fait de ne pas actualiser régulièrement les jetons d'accès peut entraîner des vulnérabilités
- autorisez vos utilisateurs à révoquer leur autorité
- invalidez vous-même un jeton d'accès et forcez une réémission s'il y a une raison de soupçonner qu'un jeton a été compromis
- assurez-vous que les jetons que vous fournissez disposent des autorisations les plus étroites possibles (la réduction des autorisations signifie qu'il y a un risque beaucoup plus faible pour votre API si les jetons sont perdus par les utilisateurs ou compromis)

Règle 5.7

Le contrôle de l'usage d'une API peut se faire suivant deux niveaux :

- à un niveau global afin de préserver les ressources mises en oeuvre pour la délivrance de l'API. Ce niveau s'applique globalement à tous les consommateurs et repose généralement sur des éléments techniques pour le comptage de l'usage. Ce contrôle permet de répartir les ressources sur l'ensemble des demandes en limitant l'usage des gros consommateurs.
- à un niveau contractuel afin de s'assurer du respect des conditions d'utilisation par les consommateurs. Ce niveau repose sur des éléments métiers pointés dans le contrat d'usage. Il permet d'être fin dans la gestion des ressources et ainsi garantir le niveau de qualité de service annoncé.

Règle 5.8

Surveillez l'usage de votre API pour détecter tout comportement inhabituel, tout comme vous surveilleriez de près tout site Web. Recherchez les changements d'adresses IP ou d'utilisateurs utilisant des API à des moments inhabituels de la journée.

La fourniture d'une API ne doit pas se limiter à la prise en compte des composants permettant son exposition et sa consommation. L'ensemble des composants contribuant aux traitements des demandes soumises par l'API doivent être pris en compte dans cette surveillance.

Cette chaîne de composants entre également dans l'élaboration du contrat d'usage unissant le fournisseur d'API à ses consommateurs.

Règle 5.9

Un périmètre de données ou de services (scope) doit être défini pour un contrat d'usage.

L'identification d'un contrat d'usage ne doit pas reposer sur des éléments techniques (adresse IP) ou métier (user ID).

Règle 5.10

Il existe différentes stratégies permettant de garantir le niveau de qualité de service et de disponibilité de votre API ainsi que son passage à l'échelle.

Pour les API d'accès aux données ouvertes pouvant être mises en cache, un réseau de diffusion de contenu (CDN) bien configuré peut fournir une évolutivité suffisante.

Pour les API qui ne présentent pas ces caractéristiques, vous devez définir des quota d'usage pour vos utilisateurs en termes de capacité et de taux disponibles. Commencez petit, en fonction des besoins des utilisateurs, et répondez aux demandes d'augmentation de la capacité en vous assurant que votre API peut respecter les quotas que vous avez définis. Assurez-vous que les utilisateurs peuvent tester votre API complète jusqu'aux quotas que vous avez définis.

Appliquez les quotas que vous avez définis, même lorsque vous avez une capacité excédentaire. Cela garantit que vos utilisateurs bénéficieront d'une expérience cohérente lorsque vos ressources viendront à diminuer, et qu'ils concevront et construiront leurs applications pour gérer votre quota d'API.

Comme pour les services destinés aux utilisateurs, vous devez tester la capacité de vos API dans un environnement représentatif pour vous assurer de pouvoir répondre à la demande.

Lorsque l'API fournit des informations personnelles ou privées, vous, en tant que responsable du traitement, devez fournir des délais d'expiration suffisants pour toutes les informations mises en cache dans votre réseau de distribution (CDN).