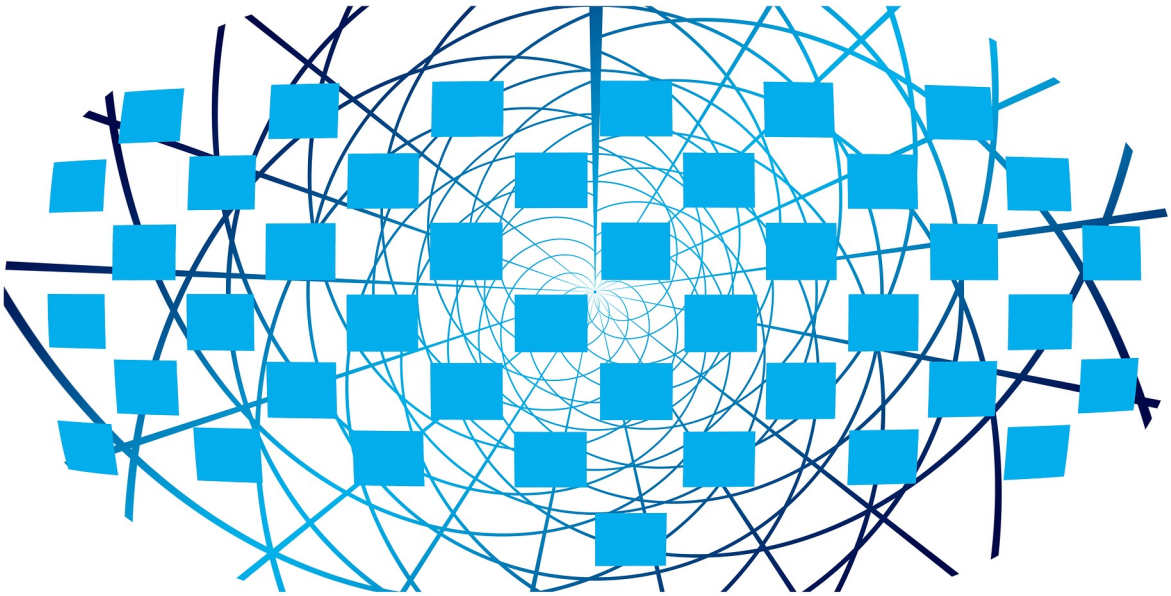
Liberté • Égalité • Fraternité
RÉPUBLIQUE FRANÇAISE

MINISTÈRE DE L'INTÉRIEUR

dsic
MINISTÈRE DE L'INTÉRIEUR

Stratégie API de la DSIC



Version 1.0

du 30 juin 2017

Table des matières

Introduction

Sous l'impulsion de la Stratégie État Plateforme lancée par la DINSIC et de la loi pour une République Numérique, 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 sujet 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 mets en avant certains standards qui sont également utilisés dans les différentes API construites par des acteurs publics et disponibles sur http://api.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

Principes

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

Principe #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

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 #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 #3 : les versions d'une API suivent la politique de gestion de versions de la DSIC

En respect des 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 #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 #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 Agreement*)
- détecter les défaillances
- identifier un besoin d'évolution et en mesurer l'efficacité

Principe #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 utilisateur de l'API.

Règles d'architecture

Les règles définies dans cette partie se veulent une déclinaison opérationnelle des principes définis dans la stratégie API de la DSIC. Il s'agit d'une première itération qui sera amenée à évoluer et à être enrichie au fil des retours d'expérience des projets.

Principe #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.

Principe #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

Le développement des API respecte les principes RESTful suivant :

- 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
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
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
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
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 #2.3

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 #2.4

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……/v1/immatriculations*
- *retourne une ressource unique (les informations pour l'immatriculation fournie)*
  - *GET https://api……/v1/immatriculations/AB-123-XZ*

Règle #2.5

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

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.

Principe #3 : les versions d'une API suivent la politique de gestion de versions de la DSIC

Règle #3.1

Une API peut supporter au plus deux versions en même temps. Dés 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
- 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

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

Règle #5.1

Les informations transmises lors de l'usage d'une API doivent permettre l'identification d'un contexte métier conforme à cette utilisation.