Norme API Groupe CA

Conception et utilisation de ressources

Architecture d'entreprise Groupe CA

Niveau	Mise en œuvre SI

Catégorie	Normes d'interopérabilité
Nature	Document
Type	Bonnes pratiques
Libellé	Conception et utilisation de ressources
Version	1.1.0
Date d'émission	04/2018
Rédacteur	CASA/SIG/AEG – Marc Helloco

Nom	RefAE_NormeAPI_BP_Ressources
Version étendue (avec révisions mineures)	1.1.011 (12/02/2021)

Version	Date	Paragraphes Modifiés/ajoutés	Objet
1.0	11/12/2017		Création du document.
1.1	19/02/2018		Revue de la mise en forme

Suivi des validations

Version	Date	Organe de revue / validation	Observations
1.0	19/12/2017	CAEG	
1.1	06/03/2018	CAEG	

Table des matières

1 Identification des ressources

1.1 Généralités

Une ressource peut être tout ce que l'on considère comme suffisamment important pour être géré et représenté en tant que tel.

Pour y faire référence on utilise une représentation de la ressource, qui correspond à l'état courant de la ressource, caractérisé par la valeur de ses différents attributs.

La ressource est décrite par :

- Une URI d'identification (l'anglais est préconisé)
- Des métadonnées spécifiques à la représentation de la ressource :
  - Le type de représentation exprimé au moyen du Type MIME (application/json, application/pdf, text/xml, text/html, image/jpeg, video/mpeg, …),
  - La date de dernière modification,
  - Les données nécessaires au cache.
- La valeur des différents attributs de la ressource :
  - Son identifiant,
  - Les autres attributs.

1.2 Format de l'URI d'identification

1.2.1 Principes généraux

Préconisation 01

Une URI permettant d'identifier une ressource est de la forme :

http(s) : //<host_name>[:<port>]/<api_name>/<api_version>/<resource_name>

<host_name> : le nom d'hôte caractérise le domaine qui héberge la ressource. Le format du nom d'hôte est contraint par des règles de nommages rappelées au chapitre suivant.

<port> : Par défaut le port 443 est associé au protocole HTTPS

<api_name> : le libellé identifiant l'API. L'anglais est préconisé pour nommer une API

<api_version> : la version de l'API.

<resource_name> : le chemin permettant d'identifier la ressource. C'est une chaîne de caractères au format d'un chemin de fichier (Path). L'anglais est préconisé pour nommer une Ressource. Il peut comprendre un segment identifiant un contexte technique. Il est recommandé de suivre les bonnes pratiques décrites en 1.3.

Exemples :

```
https://api.casa.group.gca/struct_ref/v1/offices/ql4052
```

1.2.2 Règles de nommage du nom d'hôte pour une URI Intranet

Il doit être conforme aux règles de nommage groupe, rappelées ci-dessous.

Format du nom d'hôte

Le nom d'hôte (Host Name) doit être conforme aux normes Groupe d'une application qui n'utilise pas le protocole IC04, réservé aux applications en architecture ADSU (voir [NHOT] et [ADSU] pour plus de détail) :

Préconisation 02

<Environnement>-<nomapplication>.<Propriétairedelapplication>.group.gca

Règles de nommage des environnements

<Environnement> caractérise l'environnement dans lequel est déployé le serveur ciblé. Il correspond à l'étape du cycle de développement de la fonction. On distingue les valeurs suivantes :

Étape du développement	Valeur de <Environnement>
Intégration	int ou int<i> (intégration numéro i)
Homologation	hom
Pré-production	pre
Formation	form
Certification	certi
Recette	rct

Pour l'environnement de production, la variable <Environnement> n'est pas renseignée.

Nommage des applications

Aucune règle de nommage n'est définie.

Cependant, dans le cadre d'une URL d'API, il est autorisé de remplacer ou de préfixer avec un « - » le <nomapplication> par « api ». Le nom d'hôte devient alors :

Préconisation 03

<Environnement>-**api.**<Propriétairedelapplication>.group.gca

OU

<Environnement>-**api-<nomapplication>**.<Propriétairedelapplication>.group.gca

Nommage des entités

Pour les Caisses Régionales, l'entité correspond à CA-Technologies et Services (CATS) et non à une Caisse Régionale car les environnements sont mutualisés jusqu'au « host ».

Lorsque cela est nécessaire, le numéro de Caisse Régionale est indiqué dans les paramètres de la requête.

Autres contraintes pour le nom d'hôte

Préconisation 04

Il est préconisé d'utiliser des mots comprenant **12 caractères maximum**, dans le nom d'hôte.

Il est préconisé de se limiter à un **maximum de 5 niveaux de sous domaine**, racine comprise.

Exemple :

```
hom-exemplapp.fakeentite.group.gca
```

1.2.3 Règles groupe de nommage du nom d'hôte pour une URI exposée sur internet

Celui-ci est de la responsabilité de l'Entité propriétaire du nom de domaine. En conséquence, le choix du nom d'hôte doit suivre les procédures de validation prévues par les Entités concernées. Par exemple, pour « credit-agricole.fr », il est attribué par l'unité Département de l'information de la direction de la communication groupe de Crédit Agricole S.A. (CASA/DG/SGL/DCG/DI) auprès de qui une demande doit être faite.

Cependant pour assurer la cohérence des noms d'hôte d'API au sein du Groupe, il est préconisé de respecter le nommage suivant :

Préconisation 05

<Environnement>-**api.**<sous-domaine*>.<domaine de l'Entité>

** le sous-domaine est optionnel*

1.3 Bonnes pratiques de nommage des API, ressources et attributs

1.3.1 Langue utilisée pour le nommage des API, ressources et attributs

Préconisation 06

L'anglais est préconisé pour le nommage des API, des ressources et des attributs des ressources, qu'ils soient exposés en interne Groupe ou en externe

1.3.2 Liste des caractères autorisés pour le nommage

Pour maximiser la compatibilité des noms d'API, ressources et attributs avec l'ensemble des composants logiciels qui auraient à les manipuler (navigateurs, etc.), seules les caractères suivants sont autorisés pour leur nommage :

Préconisation 07

Caractères alphanumériques, « - » et « _ » : **[0-9a-zA-Z_-]**

Il est ainsi conseillé d'éviter tout caractères accentués ou spéciaux.

1.3.3 Casse préconisée pour le nommage des API, ressources et attributs

Pour maximiser la compatibilité avec les navigateurs et pour faire converger le nommage des API du Groupe :

Préconisation 08

Pour le nom des API, des ressources et des attributs, **il est préconisé d'utiliser le format « snake_case ».** Le format « lowerCamelCase », historiquement utilisé dans le Groupe, reste toléré.

- snake_case :

  Par définition, en « snake_case », les données sont, généralement, en minuscule et les mots sont séparés par des "_" (undescore).

  Au niveau du Groupe, nous préconisons de privilégier l'utilisation des minuscules (pour éviter les éventuelles problèmes de casse entre appelant et appelé), bien que l'utilisation de majuscule reste acceptable (par exemple, pour les acronymes, si l'entité considère que cela améliore la lisibilité).

1.3.4 Typage des ressources

Il existe essentiellement deux types de ressources :

- Type unitaire : la ressource représente un objet qui existe individuellement. Par exemple une facture, un billet de transport, un personnage célèbre, …
- Type collection : la collection est elle-même une ressource qui représente la totalité des occurrences d'une ressource individuelle de même nature. Par exemple les salariés d'une entreprise, ses clients, les articles de son catalogue, les comptes d'un client, les mouvements d'un compte, une liste d'aéroports, de vols, ...

  Une collection peut représenter des notions associées à des instances si l'on souhaite par ailleurs les gérer en tant que tel.

  Par exemple les créneaux de disponibilités d'un conseiller peuvent être gérés d'une part comme des ressources associées à un conseiller, et d'autre part en tant que tel, tous conseillers confondus. Ce sont des choix de gestion qui induisent une modélisation.

```
/employees/{identifier}/availability

OU

/availability
```

1.3.5 Autres bonnes pratiques de nommage

Le nommage des API, des ressources et de leurs attributs requiert de partager un certain nombre de bonnes pratiques, exposées ci-dessous :

Préconisation 09

Pour le nommage d'une API, une ressource ou un attribut, on utilisera des noms courts, concrets et explicites et ayant un sens fonctionnel. On n'utilisera pas de verbe.

Exemples

```
GET /potential_client/124578
GET /contracts/2586542124578

DELETE /contracts/2586542124578
```

```
GET /create_potential_client/124578
GET /create_contract/2586542124578

DELETE /create_contract/2586542124578
```

Préconisation 10

Pour le nommage d'une API, une ressource ou un attribut, on s'efforcera de ne pas dépasser 3 mots

Exemples

```
https://api.casa.group.gca/my_api/payment_transaction_id
```

Préconisation 11

Pour le nommage d'une ressource ou d'un attribut, on ne répétera pas un nom déjà présent dans l'objet dans lequel il se trouve

Exemples d'objet JSON

```
"card_holder": {
    "name": "…",
    "address": "…",
    "birthdate": "…"
}
```

```
"card_holder": {
    "card_holder_name": "…",
    "card_holder_address": "…",
    "card_holder_birthdate": "…"
}
```

Préconisation 12

Une ressource de type collection est nommée par un nom au pluriel

Exemples

```
https://api.casa.group.gca/app/locations
https://api.casa.group.gca/app/employees
https://api.casa.group.gca/app/providers
```

Préconisation 13

Une ressource de type unitaire est nommée par un identifiant dans la collection. On conserve dans l'URI la collection à laquelle elle est rattachée.

Exemples

- Uri de la ressource matérialisant l'ensemble des collaborateurs

```
https://api.casa.group.gca/employees
```

- Uri d'une ressource particulière dans la collection (un agent en particulier)

```
https://api.casa.group.gca/employees/001174853
```

Préconisation 14

Pour indiquer dans l'Uri d'une ressource l'existence d'un identifiant à renseigner lors de l'accès, on utilise le pattern {id}.

Exemples

- Uri d'une ressource particulière dans la collection (un agent, un mouvement en particulier)

```
/employees/{id}
/payment_requests/{payment_information_id}/transactions/{instruction_id}/report
```

Préconisation 15

Pour la détermination des identifiants, privilégier les identifiants techniques de type UUID.

Par exemple éviter les identifiants séquentiels qui permettent sans difficulté de récupérer des instances en déroulant une séquence.

Préconisation 16

Lorsqu'un attribut est un identifiant, une bonne pratique sera d'utiliser le suffixe « _id » en fin de mot.

Exemples d'objet JSON avec un attribut de type identifiant.

```
"contract": {
    "id":          "…id du contrat…",
    "date":        "…",
    "client_id": "…id du client concerné…"
}
```

Préconisation 17

Lorsqu'un attribut est un booléen, une bonne pratique sera d'utiliser des préfixes ou suffixes tel que « is », « are », « has » ou « can ». L'objectif étant que la simple lecture du nom de l'attribut permettre d'en déduire que sa valeur est un booléen

Exemples d'objet JSON avec un attribut booléen

```
"contract": {
    "id":          "…id du contrat…",
    "date":        "…",
    "is_paid": true
}
```

Préconisation 18

Limiter, dans la mesure du possible, l'utilisation du mot « of » dans le nom des API, ressources et attributs

Il sera préférable d'utiliser « birth_date » plutôt que « date_of_birth ».

Préconisation 19

Pour le nommage des ressources correspondant à des objets d'échange identifiés on utilise le nom de l'objet d'échange (s'il est défini en anglais).

Les objets d'échanges sont identifiés dans le cadre des travaux nationaux sur les canevas[1].

La liste des objets d'échanges identifiés peut être consultée à l'adresse suivante : https://ca-sa.ca-mocca.com/site/architecture-entreprise-groupe/Data/Canevasdechange/Pages/Canevasdechange.aspx

S'il n'y a pas la traduction anglaise de l'objet d'échange, privilégiez un terme en anglais.

Préconisation 20

Respecter autant que possible le principe d'opacité des URI.

[1] Le canevas est un ensemble de bonnes pratiques adressant une problématique répétitive sur un thème donné. Il permet d'identifier un ensemble d'objets systématiquement échangés pour un thème donné. Voir [CANV] pour plus de détails.

Le principe d'opacité des URI est une des bonnes pratiques de design signalées par le W3C (http://www.w3.org/DesignIssues/Axioms.html#opaque) : Les URI ne sont pas utilisées pour analyser le contenu et ne changent pas même lorsque l'état de la ressource change.

Dans l'identification des ressources il est nécessaire de trouver le bon compromis entre URI permanentes et URI lisibles :

- URI permanentes : qui ne sont pas sensibles au changement d'état de la ressource, ce qui est structurant pour la gestion de l'état de la ressource.

Préconisation 21

On évite d'utiliser dans les URI les éléments d'état les plus variables (intitulé, nom, titre, …) et l'on privilégie les identifiants techniques (UUID).

Exemples

```
GET /employees/124578 HTTP/1.1
Host: api.casa.group.gca
```

- URI lisibles : on favorise une correspondance intuitive entre l'URI et la ressource, ce qui expose au problème précédent et diminue la pérennité des favoris car l'URI contient des informations sur l'état de la ressource.

Exemples

```
GET /france/town/bruay-en-artois HTTP/1.1
Host: api.casa.group.gca
```

Remarque : la commune de Bruay-en-Artois a été fusionnée en 1987 avec la commune de Labuissière pour donner la commune de Bruay-La-Buissière. Dans ce cas, l'URI est impacté par la modification.

1.3.6 Catalogues Groupes des nomenclatures de ressources et données API

Pour accompagner l'APIsation des SI du Groupe, l'Architecture d'Entreprise Groupe a initié des ateliers de travail avec les Entités pour établir des catalogues décrivant les nomenclatures des ressources et données exposées dans les API afin que toutes les entités du Groupe utilisent un vocabulaire commun dans leurs APIs.

Ces catalogues permettent :

- D'accélérer la phase de conception des projets, ces derniers pouvant s'appuyer sur le catalogue pour définir leurs données et ressources
- D'assurer une cohérence des données et ressources au niveau groupe, le catalogue étant transverse à toutes les entités

- De suivre les standards de marché et du Groupe au plus près (ISO20022, DSP2, réglementaire, Vega, etc.)

Les catalogues sont accessibles sur L'intranet AEG : lien

Exemples de nommage de quelques données du domaine « Personne »

Donnée Format snake-case	Type	Description
id	string	Identifiant de la personne
last_name	string	Nom de la personne
first_name	string	Prénom de la personne La donnée peut être de type collection de string si l'ensemble des prénoms doit être restitué ([« prénom1 », « prénom2 »,« prénom3 »,…])
usual_name	string	Nom usuel de la personne
birth_name	string	Nom de naissance de la personne
title_code	string	Code du titre de la personne (nomenclature CAM0028)

2 Description des échanges

2.1 Principes généraux

2.1.1 Interactions avec les ressources

Les interactions avec les ressources utilisent les "méthodes" HTTP spécifiées dans les RFC 7231 et 5789 qui constituent une interface uniforme pour toutes les ressources :

Méthode	Action standard
GET	Obtenir une représentation d'une ressource.
POST	Création d'une nouvelle ressource non encore identifiée. Ajout de données à la représentation d'une ressource existante.
PUT	Edition incrémentale de la ressource (Remplacement). Mise à jour de toutes les représentations de la ressource avec les données transmises.
PATCH	Mise à jour partielle d'une ressource
DELETE	Suppression de toutes les représentations de la ressource.
HEAD	Vérification de l'existence d'une ressource sans la récupérer et éventuellement découvrir les métadonnées de la ressource.
OPTIONS	Déterminer les verbes disponibles (les actions autorisées) pour une ressource.

Par conséquent, plutôt que de concevoir pour les ressources des méthodes "CRUD" (Create, Read, Update, Delete) d'un service et d'utiliser systématiquement POST pour transmettre les paramètres, la bonne pratique de développement sera d'utiliser les verbes http POST pour créer, GET pour lire, PUT ou PATCH pour modifier, DELETE pour supprimer.

Point d'attention 1 : lors de l'utilisation de la méthode GET, la taille des données applicatives pouvant être transmises est limitée à quelques kilo-octets (car ces données seront positionnées dans la QueryString). En conséquence, en cas de besoin de transmettre des données applicatives de plusieurs kilo-octets (par exemple en cas de transmission d'un jeton JWT applicatif, en plus de l'Access Token), il pourra être envisager, de manière dérogatoire (s'il n'est possible d'adapter la conception de l'API), d'utiliser la méthode HTTP POST, en lieu et place d'un GET, pour s'affranchir de cette limite de taille (les données applicatives seront alors transmises dans le corps de la requête).

Point d'attention 2 : les méthodes PUT et DELETE, qui peuvent dans certains contextes présenter des vulnérabilités, sont quelquefois bloquées par des équipements réseau avec le paramétrage constructeur par défaut.

Le risque a été pris en compte par les travaux sécurité, des mesures ad hoc identifiées (RS-23 – voir [ARAPI]) et prises en compte dans la spécification.

Par conséquent, pour autant que les mesures concernées aient été mises en œuvre, l'hébergeur n'est plus fondé à bloquer ces flux et le paramétrage des équipements peut être modifié de manière à permettre leur utilisation.

Les méthodes http disposent de propriétés qui caractérisent leur comportement vis-à-vis des ressources. Parmi ces propriétés, deux sont remarquables :

- "safe" : indique que la méthode est orientée lecture seule et ne provoque pas de changement d'état de la ressource accédée,
- "idempotente" : indique que de multiples requêtes consécutives utilisant la méthode sur une ressource n'ont pas plus d'effet qu'une seule requête, les requêtes suivant la première étant sans effet ajouté. Répéter n fois le remplacement d'une ressource A par la ressource B n'aura d'effet que lors de la première requête.

Le tableau ci-dessous indique les propriétés associées à chacune des méthodes.

Méthode	Safe	Idempotente
GET	X	X
POST		
PUT		X
PATCH		
DELETE		X
HEAD	X	X
OPTIONS	X	X

2.1.2 Format des données échangées

2.1.2.1 Casse des données contenues dans les URI et le body

Préconisation 22

Pour le nom des données composants les URI ou transmises, soit en paramètre de l'URI, soit dans le body de la requête et de la réponse, il est préconisé d'utiliser le format « snake_case ». Le format « lowerCamelCase », historiquement utilisé dans le Groupe, reste toléré.

- snake_case :

Les données sont en minuscule et les mots sont séparés par des "_" (undescore).

Exemple :

```
GET /contracts?client_id=40100785632
POST/locations {"delivery_date":"2015-01-31”}
```

Il est recommandé d'avoir au moins deux caractères par mot (surtout pour le premier mot).

2.1.2.2 Format des dates

Préconisation 23

Pour les échanges de dates, dates et heures, horodatages, on utilisera les formats décrits dans la norme ISO 8601 et retenus par le W3C (voir [DATE] pour plus de précisions).

Les dates de création et d'expiration du jeton de sécurité JSON WEB TOKEN constituent la seule exception à ce format ISO (voir la norme API Groupe [NAPI]).

Exemples de formats :

```
Année : YYYY (ex. 2015) ;
Année et mois : YYYY-MM (ex. 2015-07) ;
Date complète : YYYY-MM-DD (ex. 2015-07-24) ;
Date, heures et minutes : YYYY-MM-DDThh:mmTZD
    TZD  = time zone designator (Z ou +hh:mm ou -hh:mm)
    (ex. 2015-07-24T19:20+01:00) ;
Date, heures, minutes et secondes : YYYY-MM-DDThh:mm:ssTZD
    (ex. 2015-07-24T19:20:30+01:00) ;
Date, heures, minutes, secondes et fractions de seconde : YYYY-MM-
DDThh:mm:ss.sTZD
    (ex. 2015-07-24T19:20:30.45+01:00).
```

La norme W3C autorise 2 formats pour la représentation du timezone :

- Soit les dates sont exprimées en UTC (Temps Universel Coordonné ~ GMT), avec un indicateur UTC spécial ("Z").
- Soit les dates sont exprimées en heure locale, avec un décalage de fuseau horaire en heures et en minutes.

  Un décalage de "+hh:mm" indique un fuseau horaire local "hh" heures et "mm" en avance sur UTC.

  Un décalage de "-hh:mm" indique un fuseau horaire local "hh" heures et "mm" minutes en retard sur UTC

Exemples : représentation de la date du 31 mars 2016, 10:15:30 du matin à Paris

- En heure locale : 2016-03-31T10:15:30+01:00
- En temps UTC : 2016-03-31T09:15:30Z

Préconisation 24

La règle adoptée par le Groupe pour les échanges est d'utiliser l'heure locale

Pour représenter le timezone, en java, pour les jdk 7 et 8 la classe `SimpleDateFormat` propose un pattern spécifique conforme à ISO8601: le pattern X

(voir https://docs.oracle.com/javase/8/docs/api/java/text/SimpleDateFormat.html)

Exemple :

Format "yyyy-MM-dd'T'HH:mm:ss.SSXXX") : 2016-03-25T17:17:27.27+01:00

Point d'attention : pour le jdk6 il est nécessaire de prévoir un développement spécifique pour retrouver ce format.

2.1.2.1 *Format d'encodage des caractères échangés (Charset UTF-8)*

Préconisation 25

Pour les caractères échangés en entrée et en réponse d'une API, il est préconisé d'utiliser le Charset UTF-8

Il s'agit du format le plus standard pour les échanges Web. Ce format intègre quasiment tous les caractères que l'on rencontre dans toutes les langues.

2.1.3 Négociation de contenu

2.1.3.1 *Cas général*

Préconisation 26

Dans le cas général, le contenu restitué par une requête est déterminé par les principes de la négociation de contenu proactive, tel que décrit dans spécification http (https://tools.ietf.org/html/rfc7231#section-3.4).

L'émetteur de la requête indique dans un header `Accept` les types de contenu souhaités avec les règles de préférence associées.

Le fournisseur peut restituer, si elle existe, la représentation la plus adaptée à la demande. Il indique dans un header `Content-Type` le type de contenu restitué.

Si la ressource n'est disponible dans aucun des formats demandés, le fournisseur **doit** restituer (https://tools.ietf.org/html/rfc7231#section-6.5.6) :

- La liste des représentations disponibles et les identifiants de la ressource ;
- Un code status 406 (Not Acceptable)

Exemple :

```
GET /employees/a0eebc99-9c0b-4ef8-bb6d-6bb9bd380a11 HTTP/1.1
Host: api.casa.group.gca
Accept: application/json; application/xml
```

Si la représentation JSON existe, le serveur la fournit.

```
HTTP/1.1 200 OK
Content-Type: application/json
…
{"id": "a0eebc99-9c0b-4ef8-bb6d-6bb9bd380a11", "name": "SMITH"}
```

Sinon, si la représentation XML existe, le serveur la fournit

```
HTTP/1.1 200 OK
Content-Type: application/XML
…
<client>
    <id>a0eebc99-9c0b-4ef8-bb6d-6bb9bd380a11</id>
    <name>SMITH</name>
</client>
```

Si aucune des représentations disponibles ne correspond à la requête, restitution utilisant une erreur standard (voir 2.9 – Gestion des retours en erreur) :

```
HTTP/1.1 406 Not Acceptable
Content-Type: application/json
{
  "error":
    [{
      "error": " UOM35268CD409520",
      "error_lib": "invalid mediatype",
       "error_description": "no representation for media type accepted",
       "error_uri": "https://appfoo.casa.group.gca/errors/mediatype.html"
    }],
  "available_resources":[
       {"type":"application/html": "id":"H145gg2f5d11xs"}
       {"type":"application/text": "id":"T145gg2f5d11xs"}
              ]
}
```

2.1.3.2 Échanges de données "brutes"

Préconisation 27

Pour les échanges de données textuelles on utilise prioritairement le format JSON (voir [JSON] pour des précisions). Le type de contenu échangé doit être "application/json".

Exemple de Requête

```
GET /employees/a0eebc99-9c0b-4ef8-bb6d-6bb9bd380a11 HTTP/1.1
Host: api.casa.group.gca
Accept: application/json
```

Exemple de Réponse

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

{
     "id": "a0eebc99-9c0b-4ef8-bb6d-6bb9bd380a11",
     "lastName": "Martin",
     "firstName": "Alain"
     …
}
```

D'autres formats pourront être utilisés dans le cadre de la négociation de contenu, par exemple XML pour les données textuelles ou pdf pour les documents.

2.1.3.3 Restitution des liens dans les réponses

Le niveau de maturité REST retenu pour les échanges est minimum 2 (cf. la norme API Groupe [NAPI]) avec possibilité d'utiliser les liens hypermédia si les 2 entités en présence dans l'échange le souhaitent et que la représentation demandée le justifie (ressource textuelle). La représentation restituée contient alors les liens "hypermédia" matérialisant les actions possibles sur la ressource ou des liens vers des ressources liées et permettant une navigation dynamique, ce qui permet aux consommateurs de s'affranchir des modifications des liens.

Ainsi, il peut être nécessaire à un fournisseur de savoir restituer une ressource au niveau 2, sans les liens ou au niveau 3, avec les liens. Il s'agit de deux représentations de la même ressource et elles doivent correspondre à deux types de contenu distincts.

Comme vu au paragraphe précédent, s'agissant d'une ressource textuelle, le contenu au niveau 2 sera échangé en utilisant le type application/json (ou application/xml).

Préconisation 28

Le contenu textuel au niveau 3 sera échangé en utilisant le type application/hal+json (ou application/hal+xml).

Les liens seront alors indiqués en utilisant les conventions du formalisme HyperText Application Langage (HAL) (voir [HAL] pour des précisions).

Le contenu restitué est un objet JSON qui contient, en complément de ses données propres, deux propriétés aux noms réservés, **optionnelles** :

- "`_links`" qui contient les liens vers d'autres ressources,
- "`_embedded`" qui contient des ressources incluses.

Les ressources incluses peuvent elles-mêmes, individuellement, contenir des liens et des ressources incluses, et donc comprendre les balises réservées "_links" et "_embedded".

Exemple de Requête

```
GET /employees/a0eebc99-9c0b-4ef8-bb6d-6bb9bd380a11 HTTP/1.1
Host: api.casa.group.gca
Accept: application/hal+json
```

Exemple de Réponse

```
HTTP/1.1 200 OK
Content-Type: application/hal+json

{
    "id": "a0eebc99-9c0b-4ef8-bb6d-6bb9bd380a11",
    "last_name": "Martin",
    "first_name": "Alain",

    "_links": {
        "self": {"href": "employees/a0eebc99-9c0b-4ef8-bb6d-6bb9bd380a11"},
        "bank_branches": {"href": "/bank_branches/54fb8t4g2c"}
    },
    "_embedded": {
        "poste_fonctionnel": {
            "id": "8820006",
            "shortlib": "charge de clientèle",
            "_links": {
                "self": {"href": "postesFonctionnels/8820006"}
            }
        }
    }
}
```

2.1.4 Compression

La compression des flux est indiquée par l'utilisation des en-têtes `Accept-Encoding` dans la requête et `Content-Encoding` dans la réponse.

Exemple de Requête

```
GET /employees/1000 HTTP/1.1
Host: api.casa.group.gca
Accept: application/json
Accept-Encoding: gzip
```

Exemple de Réponse

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Encoding: gzip

…
```

2.1.5 Négociation de la langue du contenu des réponses du service (API)

En cible, il est acquis que c'est la couche présentation qui assure, si nécessaire, l'internationalisation des messages affichés à l'utilisateur (donc pas l'API). Cependant, il peut s'avérer nécessaire de transmettre un paramètre de langue en entrée de la requête d'appel de l'API pour que celle-ci fasse une réponse dans la

langue demandée. Dans ce cas, il faudra utiliser les en-têtes `Accept-Language` dans la requête et `Content-Language` dans la réponse.

Exemple de Requête

```
GET /employees/1000 HTTP/1.1
Host: api.casa.group.gca
Accept: application/json
Accept-Language: fr,en,en-US
```

Exemple de Réponse

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Language: fr

…
```

Préconisation 29

Si aucune des langues présentes dans le header `Accept-Language` ne peut être servie par le fournisseur de l'API, la langue par défaut du fournisseur de l'API sera utilisée.

A noter qu'il existe également des situations pour lesquelles il est nécessaire de transmettre un paramètre de langue dans les données fonctionnelles de la requête :

- À des fins d'audit pour tracer la langue dans laquelle a été réalisée une opération (et avoir la garantie que le client a bien obtenu les informations dans cette langue) ;
- Pour internationaliser un libellé d'opération ;

S'il est nécessaire de renseigner un paramètre de langue à des fins fonctionnelles lors de l'appel d'une API, il doit être précisé dans les données fonctionnelles du message d'entrée.

De manière similaire, si l'on souhaite que le service restitue la langue dans laquelle le service a été traité, il sera nécessaire de le décrire dans les données fonctionnelles en sorties de l'API.

Les messages d'erreur sont, dans la mesure du possible, restitués dans la langue indiquée en entrée si elle est spécifiée, la langue par défaut restant néanmoins le français ou l'anglais.

L'internationalisation des messages d'erreur peut être gérée par la couche présentation à partir de la valeur de la variable code (voir Gestion des erreurs en 2.9).

2.1.6 Cas particulier des API appelés par des Batch

Lorsqu'une API est destinée à être appelée par un batch, il conviendra de porter une attention particulière à sa conception pour tenir compte de ce mode d'appel.

En effet, pour assurer la meilleure performance du batch et assurer la bonne disponibilité de l'API, il conviendra, en générale, d'éviter les appels en masse de l'API dans un espace de temps réduit. En effet, il sera souvent préférable de concevoir une API pouvant traiter des lots de demandes plutôt que de traiter chaque demande unitairement (pour réduire le nombre d'appel à l'API).

De plus, lorsqu'un appel par batch d'une API est envisagée, il est fortement préconisé de mettre en place des quotas d'appel de l'API (« throttling ») pour pouvoir gérer au mieux sa disponibilité.

2.2 Interrogation d'une ressource : GET

2.2.1 Cas général

Dans le cas général, l'interrogation d'une ressource se fait en utilisant la méthode GET.

Si une représentation de la ressource correspondant à la requête existe, le serveur la restitue et le code status http transmis dans la réponse prend la valeur 200.

Si la ressource n'est pas disponible, il prend la valeur 404 (non trouvé).

Exemple :

- Requête

```
GET /employees/a0eebc99-9c0b-4ef8-bb6d-6bb9bd380a11 HTTP/1.1
Host: api.casa.group.gca
Accept: application/json
```

- Réponse correcte : la ressource est disponible

```
HTTP/1.1 200 OK
Accept: application/json
[d'autres headers]

{
       "id": "a0eebc99-9c0b-4ef8-bb6d-6bb9bd380a11",
       "last_name": "martin",
       "first_name": "alain"
…
}
```

– Réponse en erreur : la ressource n'est pas disponible

```
HTTP/1.1 404 Not Found
[d'autres headers]
```

2.2.2 Représentations multiples

Pour faciliter l'utilisation des ressources plusieurs URI peuvent identifier la même ressource.

Un des URI identifie la représentation référentielle de la ressource, les autres URI sont des facilités d'accès vers la représentation référentielle.

Préconisation 30

Lorsqu'une requête est émise vers une des représentations secondaires, la correspondance vers la représentation référentielle doit être assurée par le fournisseur de la ressource.

Il indique en réponse à la requête :

- **Un code retour spécifique (303 See Other) qui est une redirection**
- **L'URI de la représentation de référence dans un header spécifique (Content-Location).**

Exemples :

– Identifier une ressource par son identifiant ou un label

```
api.casa.group.gca/clients/01234567-89ab-cdef-0123-456789abcdef
api.casa.group.gca/clients/martin
```

– Identifier une ressource par différents labels

```
/employees/skywalkerluke
/employees/lukeskywalker
```

– Requête vers la ressource référentielle

```
GET /employees/01234567-89ab-cdef-0123-456789abcdef HTTP/1.1
Host: api.casa.group.gca
Accept: application/json
```

– Réponse

```
HTTP/1.1 200 OK
[d'autres headers]

{
  "employee": {
         "id": "01234567-89ab-cdef-0123-456789abcdef",
         "last_name": "martin",
         "first_name": "alain"
     }
}
```

– Requête vers une ressource alternative

```
GET /employees/martin HTTP/1.1
Host: api.casa.group.gca
Accept: application/json
```

– Réponse

```
HTTP/1.1 303 See Other
Content-Location : https://api.casa.group.gca/employees/01234567-89ab-cdef-
0123-456789abcdef
```

2.2.3 Requêtes avec QueryString

La QueryString associée à une requête GET peut être utilisée pour répondre à différents besoins :

- Réduire le volume des données restituées par filtrage ;
- Trier les données restituées

2.2.3.1 Filtrage des données restituées

Le filtrage peut être réalisé selon deux méthodes :

- Limiter le nombre d'occurrences restituées dans la collection,
- Limiter le nombre d'attributs restitués pour une occurrence.

Préconisation 31

Pour limiter le nombre d'occurrences restituées dans la collection :

Soit, on précise dans la requête la valeur recherchée pour certains attributs en indiquant dans la query string les attributs, séparés par un caractère "**&**" et la ou les valeurs, séparées par une virgule.

Soit, on indique dans la requête que l'on souhaite limiter le nombre d'occurrence en réponse. Pour cela on utilisera dans la query string le mot-clé "**limit**", suivi de "**=**" et d'un nombre.

Préconisation 32

Pour limiter le nombre d'attributs restitués pour une occurrence :

On indique dans la requête que l'on souhaite ne restituer que certains attributs en utilisant dans la query string le mot-clé "**fields**", suivi de "=" et de la liste des attributs à restituer, séparés par une virgule.

Les deux méthodes peuvent être utilisées conjointement.

Exemples

- Limiter le nombre d'occurrences restituées en indiquant la valeur de certains attributs

```
GET /resources?uom=87800,88200&vip=1 HTTP/1.1
Host: api.casa.group.gca
Accept: application/json
```

- Limiter le nombre d'occurrences restituées en indiquant la limite maximum

```
GET /resources?limit=100 HTTP/1.1
Host: api.casa.group.gca
Accept: application/json
```

- Limiter le nombre d'attributs restitués

```
GET /clients/1001?fields=first_name,last_name HTTP/1.1
Host: api.casa.group.gca
Accept: application/json
```

- Utilisation conjointe des deux méthodes

```
GET /resources?uom=87800,88200&vip=1&fields=first_name,last_name HTTP/1.1
Host: api.casa.group.gca
Accept: application/json
```

2.2.3.2 Tri des données restituées

Préconisation 33

Il est possible, lorsqu'une requête concerne une liste d'occurrences, de demander que les données restituées soient triées. On l'indique en utilisant les mots clés **`sort=`** et **`:desc`** dans la query string.

`"sort="`: permet d'indiquer les arguments de tri, séparés par une virgule. Les arguments de tri sont des attributs des ressources.

Par défaut le tri est ascendant (croissant).

`":desc"` : permet d'indiquer sur un attribut que l'on souhaite un tri descendant (décroissant).

Ces mots-clés peuvent être utilisés avec des mots clé de filtrage.

Exemple :

- Requête de tri ascendant sur le premier attribut restitué, descendant sur le deuxième et ascendant de nouveau sur le troisième.

```
GET
/clients?sort=last_name,birth_date:desc,first_name&uom=87800&vip=1&fields=las
t_name,first_name,birth_date,dep HTTP/1.1
Host: api.casa.group.gca
Accept: application/json
```

2.2.4 Recherches

Préconisation 34

Les valeurs renseignées pour les attributs de filtrage peuvent contenir le caractère « * » en début, en fin ou les deux, permettant une sélection sur une valeur partielle :

- « * » en fin de valeur signifie que la valeur commence par la chaine qui précède « * »
- « * » en début de valeur signifie que la valeur se termine par la chaine qui suit « * »
- « * » en début et fin de valeur signifie que la valeur contient la chaîne comprise entre les caractères « * »

Exemples :

- Rechercher les occurrences de collaborateurs dont le nom commence par "Du".

```
GET /employees?name=Du* HTTP/1.1
```

- Rechercher les occurrences de mouvements d'un compte qui se terminent par "MSA".

```
GET /clients/a0eebc99-9c0b-4ef8-bb6d-
6bb9bd380a11/accounts/98562458002/transactions?shortlib=*MSA HTTP/1.1
```

– Rechercher les occurrences de collaborateurs dont la date de naissance contient 12.

```
GET /employees?birth_date=*12* HTTP/1.1
```

2.2.5 Pagination des listes de données restituées

2.2.5.1 Principes généraux

Préconisation 35

Lorsqu'une requête restitue une liste d'occurrences, le nombre maximum d'occurrences restituées est limité et la pagination doit être gérée.

Le fournisseur de la requête définit dans le contrat d'interface le nombre maximum d'occurrences restituées pour une requête.

Lors de l'accès à la ressource, le consommateur indique, dans la limite du maximum défini dans le contrat, le nombre d'occurrences souhaité.

Le fournisseur les lui restitue (ou moins si fin de liste) ainsi que des données de gestion de la pagination

Si le consommateur n'indique pas le nombre d'occurrences souhaité et que la liste comprend plus d'éléments que le maximum prévu, le fournisseur lui restitue le nombre d'éléments maximum indiqué dans le contrat ainsi que les données de gestion de la pagination.

Le comportement est identique si le consommateur indique un nombre d'occurrences supérieur au maximum prévu. Cela n'est pas considéré comme une erreur.

Préconisation 36

Le nombre maximum de niveaux d'imbrication des listes est de 4 (4 niveaux de listes imbriqués dans la liste principale).

Pour gérer la pagination, deux modes de gestion peuvent être proposés par un fournisseur de données :

- Pagination par page : le consommateur de la requête renseigne un numéro de page et le fournisseur restitue les données correspondantes, dans la limite du nombre demandé ou du maximum prévu par défaut. Ce mode de pagination permet une navigation libre dans les pages d'une liste.
- Pagination par index de repositionnement : le consommateur de la requête indique une valeur permettant le positionnement vers la première occurrence à restituer et le fournisseur restitue les données correspondantes, dans la limite du nombre demandé ou du maximum prévu par défaut. Ce mode de pagination est plus particulièrement adapté à la lecture séquentielle d'une liste à partir du début.

Préconisation 37

Le mode de gestion par page est privilégié. Le second mode est proposé pour tenir compte des existants utilisant la technologie relationnelle.

La réponse peut contenir également des liens de navigation.

Préconisation 38

Le formalisme utilisé pour décrire les liens de pagination doit être celui décrit au chapitre 2.1.3.3 (restitution des liens hypermédia dans les réponses). La restitution des liens est facultative au niveau de maturité 2.

Une réponse à une requête de liste est considérée comme une réponse correcte, même si des données supplémentaires sont disponibles ou si la fin de liste est atteinte.

Préconisation 39

Les codes retours associés à la réponse à une requête de pagination sont les codes standards d'une requête GET :

- **Retour correct =** `200 Ok`
- **Aucune donnée restituée =** `404 Not found`

2.2.5.2 Pagination par page

On utilise les données suivantes :

Dans les requêtes

Des attributs spécifiques de la query string

- "`count`" : indique le nombre d'occurrences souhaitées. Donnée facultative. Par défaut on restitue le maximum spécifié dans le contrat d'interface.
- "`page`" : Numéro de la page que l'on souhaite obtenir, sur la base de la quantité souhaitée. Donnée facultative. Par défaut la requête se fait en début de liste.

Dans les réponses

On utilise une propriété "`paging`", obligatoire, qui regroupe les attributs suivants :

- "`count`" : Nombre d'occurrences restituées. Donnée obligatoire.
- "`total`" : Nombre total d'occurrences de la liste. Donnée obligatoire.
- "`last_page`" : Numéro de la dernière page, sur la base de la quantité souhaitée. Donnée facultative.

Pour restituer dans les réponses des liens de navigation, on utilise un objet nommé "_links" qui regroupe les liens de navigation suivants :

- "`next_page`" : vers la page suivante. Ce lien n'est pas valorisé si la fin de la liste est atteinte.
- "`previous_page`" : vers la page précédente. Ce lien n'est pas valorisé dans la réponse à une requête qui part du début de la liste.
- "`last_page`" : vers la dernière page.
- "`first_page`" : vers la première page.

Il est également possible de restituer un lien vers une page intermédiaire de la liste en utilisant l'attribut de forme générique suivant :

- "pagexx": { "href": "<uri>+<?page=>+<xx>+<&count=>+<yy>" }

  xx : entier > 1 et < "dernière page de la liste"
  yy : entier > 1 et < "maximum indiqué dans le contrat d'interface "

Si l'on souhaite restituer plusieurs liens vers des pages intermédiaires, ils doivent êtres distincts et ordonnés en ordre croissant. De plus, la cohérence doit être assurée entre index et liens en valeur et ordre.

Exemples

– Exemples de requête

```
GET /clients?count=20

GET /clients?page=5&count=20
```

– Exemple de restitution des attributs de gestion de liste

```
"paging": {
    "count": "20",
    "total": "180",
    "last_page": "9"
    }
```

– Exemple de restitution des liens de pagination.

```
{
    "_links": {
    "next_page": { "href": "/clients?page=9&count=15" },
    "previous_page": { "href": "/clients?page=7&count=15" },

    "first_page": { "href": "/clients?count=15" },
    "page09": { "href": "/clients?page=9&count=15" },
    "page10": { "href": "/clients?page=10&count=15" },
    "page11": { "href": "/clients?page=11&count=15" },
    "last_page": { "href": "/clients?page=60&count=15" }
    }
}
```

Exemple global 1 : Utilisation du niveau de maturité 2 de REST

Le type de contenu échangé est application/json.

Le contenu restitué est un objet json qui comprend :

- Un tableau d'objets json (une entrée par occurrence de la collection),
- Un objet "paging" (obligatoire) qui contient les données de pagination,
- Un objet "_links" (facultatif) qui contient les liens de pagination,
- D'autres données éventuellement.

- Première requête pour obtenir la liste des 20 premiers mouvements d'un compte sur le premier trimestre 2015 :

```
GET /partners/982-cr85-
g24a4d/accounts/45674125811/transactions?period=T12015&count=20
Host: api.casa.group.gca
Accept: application/json
```

- Réponse :
  La propriété "previousPage" n'est pas valorisée.

```
HTTP/1.1 200 OK
Content-Type: application/json
{
  "transactions" : [
      {"id": "r7rb44r6(55", "date": "2015-01-01", "amount": "100,00",
"direction": "D"},
…
…   la liste des mouvements
…
      {"id": "r7rb44r8ach", "date": "2014-03-31", "amount": "150,00",
"direction" : "C"}
    ],
   "paging": {
       "count": "20",
       "total": "180",
       "last_page": "9"
            },
   "_links": {
       "next_page": { "href": "/transactions?period=T12015&page=2&count=20"
},
       "first_page": { "href": "/transactions?period=T12015&count=20" },
       "page03": {"href": "/transactions?period=T12015&page=3&count=20" },
       "page04": {"href": "/transactions?period=T12015&page=4&count=20" },
       "page05": {"href": "/transactions?period=T12015&page=5&count=20" },
       "last_page": { "href": "/transactions?period=T12015&page=9&count=20" }
            }
}
```

- Requête pour obtenir les 20 mouvements suivants :

```
GET /partners/982-cr85-
g24a4d/accounts/45674125811/transactions?period=T12015&page=2&count=20
Host: api.casa.group.gca
Accept: application/json
```

- Réponse :

```
HTTP/1.1 200 OK
Content-Type: application/json
{
    "transactions": [
      {"id": "r7rb64r89bh", "date": "2015-03-31", "amount": "100,00",
"direction": "D"},
…
… la liste des mouvements
…
      {"id": "r7rb44r8ach", "date": "2015-03-31", "amount": "150,00",
"direction": "C"}
                  ],
    "paging": {
      "count": "20",
      "total": "180",
      "last_page": "9"
          },
    "_links": {
      "next_page": { "href": "/transactions?period=T12015&page=3&count=20"
},
      "previous_page": { "href": "/transactions?period=T12015&count=20" },
      "first_page": { "href": "/transactions?period=T12015&count=20" },
      "page03": {"href": "/transactions?period=T12015&page=3&count=20" },
      "page04": {"href": "/transactions?period=T12015&page=4&count=20" },
      "page05": {"href": "/transactions?period=T12015&page=5&count=20" },
      "last_page": { "href": "/transactions?period=T12015&page=9&count=20" }
} }
```

Exemple global 2 : Utilisation du niveau de maturité 3 de REST (voir 4.1.3.3)

Le type de contenu échangé est application/hal+json,

Le contenu restitué est un objet json qui comprend :

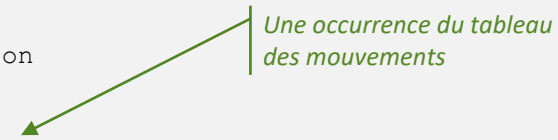
- Un objet "`paging`" (obligatoire) qui contient les données de pagination,
- Un objet "`_links`" (facultatif) qui contient les liens de pagination,
- Un objet "`_embedded`" qui contient les occurrences de la liste, dans un tableau.
  Pour chaque occurrence on pourra retrouver un objet "`_links`" avec une relation "`self`" a minima et d'autres liens vers des ressources en relation (le compte par exemple s'il s'agit de mouvements).
- D'autres données éventuellement.

- Première requête pour obtenir la liste des 20 premiers mouvements d'un compte sur le premier trimestre 2015 :

```
GET /partners/982-cr85-
g24a4d/accounts/45674125811/transactions?period=T12015&count=20
Host: api.casa.group.gca
Accept: application/hal+json
```

— Réponse :

La propriété "previousPage" n'est pas valorisée.




```
HTTP/1.1 200 OK
Content-Type: application/hal+json
{
     "_embedded": [
       {"_links": {
          "self": { "href": "/transactions/r7-*q35r6(94" },
          "account": { "href": "/accounts/45674125811" },
          "client": { "href": "/partners/982cr-*85g24" }},
        "id": "r7rb44r6(55", "date": "2014-01-01", "amount": "100,00",
"direction" : "D"
          },
…
… la liste des mouvements
…
        {"_links": {
          "self": { "href": "/transactions/r9g4t35r6+98" },
          "account": { "href": "/accounts/45674125811" },
          "client": { "href": "/partners/982cr-*85g24" }},
        "id": "r7rb44r8ach", "date": "2014-03-31", "amount": "150,00",
"direction": "C"
          }
                  ],

     "_links": {
       "next_page": { "href": "/transactions?period=T12015&page=5&count=20"
},
       "first_page": { "href": "/transactions?period=T12015&count=20" },
       "page03": {"href": "/transactions?period=T12015&page=3&count=20" },
       "page04": {"href": "/transactions?period=T12015&page=4&count=20" },
       "page05": {"href": "/transactions?period=T12015&page=5&count=20" },
       "last_page": { "href": "/transactions?period=T12015&page=9&count=20" }
             },
     "paging": { "count": "20", "total": "180", "last_page": "9" }
}
```

— Requête pour obtenir les 20 mouvements suivants :

```
GET /partners/982-cr85-g24a4d/accounts/45674125811/transactions?period=T12015
&page=2&count=20
Host: api.casa.group.gca
Accept: application/hal+json
```

— Réponse :

```
HTTP/1.1 200 OK
Content-Type: application/hal+json
{
     "_embedded" : [
     {"_links": {
        "self": { "href": "/transactions/r7-*q35r6(94" },
        "account": { "href": "/accounts/45674125811" },
        "client": { "href": "/partners/982cr-*85g24" }
               },
     "id": "r7rb44r6(55", "date": "2014-01-01", "amount": "100,00",
"direction": "D"
     },
…
… la liste des mouvements
…
     {"_links": {
        "self": { "href": "/transactions/r9g4t35r6+98" },
        "account": { "href": "/accounts/45674125811" },
        "client": { "href": "/partners/982cr-*85g24" }
               },
     "id" : "r7rb44r8ach", "date" : "2014-03-31", "amount" : "150,00",
"direction" : "C"
     }
                 ],

     "_links": {
        "next_page": { "href": "/transactions?period=T12015&page=3&count=20"
},
        "previous_page": { "href": "/transactions?period=T12015&count=20" },
        "first_page": { "href": "/transactions?period=T12015&count=20" },
        "page04": {"href": "/transactions?period=T12015&page=4&count=20" },
        "page05": {"href": "/transactions?period=T12015&page=5&count=20" },
        "page06": {"href": "/transactions?period=T12015&page=6&count=20" },
        "last_page": { "href": "/transactions?period=T12015&page=9&count=20" }
               },

     "paging": {
        "count": "20",
        "total": "180",
        "last_page": "9"
               }
}
```

2.2.5.3 Pagination par index de repositionnement

Cette méthode permet le repositionnement de l'algorithme de recherche lors des requêtes de navigation vers la page suivante ou la précédente. Elle ne permet pas de gérer la navigation vers des pages intermédiaires.

On utilise les données suivantes :

Dans les requêtes :

- "`count`" : indique le nombre d'occurrences souhaitées. Donnée facultative. Par défaut on restitue le maximum spécifié dans le contrat d'interface.

- "`start_index`" : indique la clé à partir de laquelle la lecture des instances restituées doit être faite. Donnée facultative.

Dans une requête de suite, la valeur de `start_index` correspond à la valeur restituée par la requête précédente dans l'attribut `next_set_start_index` (voir ci-dessous).

Dans les réponses :

Un objet nommé "`paging`", obligatoire, regroupe les données suivantes :

- "`count`" : Nombre d'occurrences restituées. Donnée obligatoire.
- "`total`" : Nombre total d'occurrences de la liste. Donnée obligatoire.
- "`next_set_start_index`" : indique la clé à partir de laquelle la lecture des instances restituées doit être faite pour restituer la suite de la liste. Donnée obligatoire. Cette donnée n'est pas valorisée si la fin de la liste est atteinte ;
- "`previous_set_start_index`" indique la clé à partir de laquelle la lecture des instances restituées doit être faite pour restituer la page précédente de la liste. Donnée facultative. Cette donnée n'est pas valorisée dans la réponse à une requête qui part du début de la liste ;

Pour restituer dans les réponses des liens de navigation, on utilise un objet nommé "`_links`" qui regroupe les liens de navigation suivants :

- "`next_set`" : vers la page suivante. Ce lien n'est pas valorisé si la fin de la liste est atteinte.
- "`previous_set`" : vers la page précédente. Ce lien n'est pas valorisé dans la réponse à une requête qui part du début de la liste.

Exemples

— Exemple de requête

```
GET /clients?count=20

GET /clients?start_index=982cr-*85g24&count=20&count=20
```

— Exemple de restitution des attributs de gestion de liste

```
    "paging": {
      "count": "20",
      "total": "180",
      "next_set_start_index": "982cr-*85g24",
      "previous_set_start_index": "r7-*q35r6(55"
}
```

- Exemple de restitution des liens de pagination

```
{
     "_links": {
        "next_set": {"href": "/clients?start_index=982cr-*85g24&count=15"},
        "previous_set": {"href": "/clients?start_index=r7-*q35r655&count=15"}
             }
}
```

Exemple global 1 : Utilisation du niveau de maturité 2 de REST

Le type de contenu échangé est application/json,

Le contenu restitué est un objet json qui comprend :

- Un tableau d'objets json (une entrée par occurrence de la collection),
- Un objet "`paging`" qui contient les données de pagination,
- Un objet "`_links`" qui contient les liens de pagination,
- D'autres données éventuellement.

- Première requête pour obtenir la liste des 20 premiers mouvements d'un compte sur le premier trimestre 2015 :

```
GET /partners/982-cr85-g24a4d/accounts/45674125811/transactions?period=T12015
&count=20
Host: api.casa.group.gca
Accept: application/json
```

— Réponse :

Les propriétés "previous_set_start_index" et "previous_set" ne sont pas valorisées.

```
HTTP/1.1 200 OK
Content-Type: application/json
{
     "transactions": [
       {"id": "r7rb44r6(55", "date": "2015-01-01", "amount": "100,00",
"direction": "D"},
…
…    la liste des mouvements
…

       {"id": "r7rb44r8ach", "date": "2015-03-31", "amount": "150,00",
"direction": "C"}
                  ],

     "paging": {
       "count": "20",
       "total": "180",
       "next_set_start_index": "982cr-485g24"
              },

     "_links": {
"next_set": { "href": "/transactions?period=T12015&start_id=982cr-
485g24&count=20" }
              }
}
```

— Requête pour obtenir les 20 mouvements suivants :

```
GET /partners/982-cr85-g24a4d/accounts/45674125811/transactions?period=T12015
&start_index=982cr-485g24&count=20
Host: api.casa.group.gca
Accept: application/json
```

- Réponse :

```
HTTP/1.1 200 OK
Content-Type: application/json
{
    "transactions": [
      {"id": "982cr-485g24", "date": "2015-03-31", "amount": "100,00",
"direction": "D"},
…
…   la liste des mouvements
…

      {"id": "r7rb44r8ach", "date": "2015-03-31", "amount": "150,00",
"direction": "C"}
                ],

    "paging": {
      "count": "20",
      "total": "180",
      "next_set_start_index": "982cr-485g24",
      "previous_set_start_index": "r7-*q35r6(55"
            },

    "_links": {
"next_set": { "href": "/transactions?period=T12015&start_id=982cr-
485g24&count=20" },
"previous_set": { "href": "/transactions?period=T12015&start_id=r7-
*q35r6(55&count=20" }
            }
}
```

Exemple global 2 : Utilisation du niveau de maturité 3 de REST (voir 2.1.3.3)

Le type de contenu échangé est application/hal+json,

Le contenu restitué est un objet json qui comprend :

- Un objet "`paging`" (obligatoire) qui contient les données de pagination,
- Un objet "`_links`" (facultatif) qui contient les liens de pagination,
- Un objet "`_embedded`" qui contient les occurrences de la liste, dans un tableau.
  Pour chaque occurrence on pourra retrouver un objet "`_links`" avec une relation "`self`" a minima et d'autres liens vers des ressources en relation (le compte par exemple s'il s'agit de mouvements).
- D'autres données éventuellement.

- Première requête pour obtenir la liste des 20 premiers mouvements d'un compte sur le premier trimestre 2015 :

```
GET /partners/982-cr85-g24a4d/accounts/45674125811/transactions?period=T12015
&count=20
Host: api.casa.group.gca
Accept: application/hal+json
```

— Réponse :

Les propriétés "previousSetStartIndex" et "previousSet" ne sont pas valorisées.

```
HTTP/1.1 200 OK
Content-Type: application/hal+json
{
     "_embedded" : [
       {"_links": {
          "self": { "href": "/transactions/r7-*q35r6(94" },
          "account": { "href": "/accounts/45674125811" },
          "client": { "href": "/partners/982cr-*85g24" }
                  },
"id": "r7rb44r6(55", "date": "2014-01-01", "amount": "100,00", "direction":
"D"
          },
…
…    la liste des mouvements
…
       {"_links": {
          "self": { "href": "/transactions/r9g4t35r6+98" },
          "account": { "href": "/accounts/45674125811" },
          "client": { "href": "/partners/982cr-*85g24" }
                  },
"id" : "r7rb44r8ach", "date" : "2014-03-31", "amount" : "150,00", "direction"
: "C"
          }
                  ],

     "_links": {
"next_set": { "href": "/transactions?period=T12015&start_index=982cr-
*85g24&count=20" }
              }

     "paging": {
       "count": "20",
       "total": "180",
       "next_set_start_index": "982cr-*85g24",
              }
}
```

— Requête pour obtenir les 20 mouvements suivants :

```
GET /partners/982-cr85-g24a4d/accounts/45674125811/transactions?period=T12015
&start_index=982cr-485g24&count=20
Host: api.casa.group.gca
Accept: application/hal+json
```

— Réponse :

```
HTTP/1.1 200 OK
Content-Type: application/hal+json
{
    "_embedded" : [
        {"_links": {
             "self": { "href": "/transactions/r7-*q35r6(94" },
             "account": { "href": "/accounts/45674125811" },
             "client": { "href": "/partners/982cr-*85g24" }
                 },
"id": "r7rb44r6(55", "date": "2014-01-01", "amount": "100,00", "direction":
"D"
         },
…
…   la liste des mouvements
…
        {"_links": {
             "self": { "href": "/transactions/r9g4t35r6+98" },
             "account": { "href": "/accounts/45674125811" },
             "client": { "href": "/partners/982cr-*85g24" }
               },
"id" : "r7rb44r8ach", "date" : "2014-03-31", "amount" : "150,00", "direction"
: "C"
         }
                ],

    "_links": {
"next_set": { "href": "/transactions?period=T12015&start_index=982cr-
*85g24&count=20" },
"previousSet": { "href": "/transactions?period=T12015&start_index=r7-
*q35r6(55&count=20" }
             }

    "paging": {
       "count": "20",
       "total": "180",
       "next_set_start_index": "982cr-*85g24",
       "previous_set_start_index": "r7-*q35r6(55"
             }
}
```

2.2.6 Gestion de cache

Sera traité dans une version ultérieure.

2.3 Création d'une nouvelle ressource : POST

La création d'une nouvelle ressource se fait en utilisant la méthode http POST. Le comportement standard est le suivant :

- L'émetteur envoie sur la collection une requête qui contient les propriétés valorisées de la ressource **à l'exception de l'identifiant** car il doit être calculé par le serveur cible.
- Le fournisseur calcule un identifiant pour la ressource et la créée. Dans la réponse il restitue :

- Un code retour http 201 qui indique que la ressource a été créée avec succès,
- Un header `Location` qui contient l'URI à laquelle la ressource peut être récupérée,
- Dans le body de la réponse les propriétés valorisées de la ressource, identifiant compris.

▪ En cas d'erreur (voir 2.9 pour la gestion des retours en erreurs), le fournisseur restitue :

- Un code retour adapté : `400 Bad Request` si les contrôles sur les valeurs des propriétés ne sont pas satisfaits ;
- Un message d'erreur dont le format est décrit en 2.9.

— Exemple de requête :

```
POST /agendas/982-cr85-g24a4d/meetings
Host: api.casa.group.gca
Accept: application/json
Accept-Encoding: gzip
Content-Type: application/json

{
    "date": "2015-01-07",
    "start": "1400",
    "end": "1450",
    "subject": "Simulation prêt habitat",
    "partner": {"id": "89ab-cdef-0123-456789abcdef"}
}
```

— Exemple de réponse correcte :

```
HTTP/1.1 201 Created
Location: http://api.casa.group.gca/meetings/m426-789ab-cdef
Content-Type: application/json
Content-Encoding: gzip

{
    "id": "m426-789ab-cdef",
    "employee": {"ident": "982-cr85-g24a4d"}
    "date": "2015-01-07",
    "start": "1400",
    "end": "1450",
    "subject": "Simulation prêt habitat",
    "partner": {"id": "89ab-cdef-0123-456789abcdef"}
}
```

– Exemple de réponse en erreur :

```
HTTP/1.1 400 Bad Request
Content-Type: Application/json

[{
    "error": " UOM48247BD874120",
    "error_lib": "Identifiant incorrect",
    "error_description": "L'dentifiant ne doit pas être renseigné",
    "error_uri": "https://ca-xx.group.gca/monapp/businesserr/BD874120",
    "errors_value_list": ["id"]
}]
```

2.4 Mise à jour ou création d'une ressource : PUT

La méthode http PUT peut être utilisée avec deux finalités :

- La mise à jour globale d'une ressource,
- La création d'une ressource si la valeur de l'identifiant est connue a priori.

Le comportement standard est le suivant :

- L'émetteur envoie sur la ressource une requête qui contient la totalité des propriétés valorisées.
- Le fournisseur met à jour la ressource si elle existe ou la créée si elle n'existe pas. Dans la réponse il restitue :
  - Si la ressource a été mise à jour avec succès un code retour http `200 Ok` ou `204 No Content`.

Préconisation 40

Le choix de restituer un code 200 ou 204 est à l'initiative du fournisseur de la ressource. Si l'on souhaite ne restituer aucun contenu, le code status http restitué doit être 204.

Si l'on restitue un code 200, il doit être accompagné d'un contenu constitué de la représentation de la ressource modifiée.

Après mise à jour, la représentation peut constituer une nouvelle version de la ressource (voir https://tools.ietf.org/html/rfc7231#section-4.3.4 pour plus de détails).

  - Si la ressource a été créée avec succès un code retour http `201 Created` accompagné, dans le body de la réponse, de la représentation de la ressource créée, identifiant compris.
- En cas d'erreur, le fournisseur restitue :
  - Un code retour adapté :

    `400 Bad Request` si les contrôles sur les valeurs des propriétés ne sont pas satisfaits ;
  - Un message d'erreur dont le format est décrit en 2.9.

- Exemple de requête :

```
PUT /meetings/m426-789ab-cdef
Host: api.casa.group.gca
Accept: application/json
Content-Type: application/json
Accept-Encoding: gzip
{
     "idt": "m426-789ab-cdef",
     "employee": {"id": "rf8c-edc9-hbgn25"}
     "date": "2015-01-07",
     "start": "1430",
     "end": "1530",
     "subject": "Simulation prêt habitat",
     "partner": {"id": "89ab-cdef-0123-456789abcdef"}
}
```

- Exemple de réponse :

  Création

```
HTTP/1.1 201 Created
Content-Type: application/json
Content-Encoding: gzip

{
     "id": "m426-789ab-cdef",
     "employee": {"id": "982-cr85-g24a4d"}
     "date": "2015-01-07",
     "start": "1400",
     "end": "1450",
     "objet": "Simulation prêt habitat",
     "partner": {"id": "89ab-cdef-0123-456789abcdef"}
}
```

  Mise à jour

  - Option 1

```
HTTP/1.1 200 Ok
Content-Type: application/json
Content-Encoding: gzip

{
     "id": "m426-789ab-cdef",
     "employee": {"id": "982-cr85-g24a4d"}
     "date": "2015-01-07",
     "start": "1400",
     "end": "1450",
     "subject": "Simulation prêt habitat",
     "partner": {"id": "89ab-cdef-0123-456789abcdef"}
}
```

¬ Option 2

```
HTTP/1.1 204 No Content
```

2.5 Mise à jour partielle d'une ressource : PATCH

Pour appliquer des mises à jour à une ressource sans la remplacer totalement, on utilise la méthode PATCH.

Les modifications à apporter sont transmises dans le corps de la requête. Elles doivent être renseignées en respectant la structure de la ressource (chemin et nommage des objets et des attributs) pour éviter toute ambigüité sur le nommage des données à mettre à jour. Elles sont transmises en utilisant un type de contenu particulier (patch document).

Le comportement standard est le suivant :

- L'émetteur envoie sur la ressource une requête qui contient les différentes mises à jour à appliquer,
- Le fournisseur met à jour la ressource si elle existe. Dans la réponse il restitue :
  - Si la ressource a été mise à jour avec succès un code retour http `200 Ok` ou `204 No Content`.

Le choix de restituer un code 200 ou 204 est à l'initiative du fournisseur de la ressource. Si l'on souhaite ne restituer aucun contenu, le code status http restitué doit être 204.

Si l'on restitue un code 200, il doit être accompagné d'un contenu constitué de la représentation de la ressource modifiée.

Après mise à jour, la représentation peut constituer une nouvelle version de la ressource (voir https://tools.ietf.org/html/rfc5789#section-2 pour plus de détails).

- En cas d'erreur, le fournisseur restitue :
  - Un code retour adapté :

    `400 Bad Request` si les contrôles sur les valeurs des propriétés ne sont pas satisfaits ;
  - Un message d'erreur dont le format est décrit en 2.9.

¬ Exemple de requête :

```
PATCH /meetings/m426-789ab-cdef
Host: api.casa.group.gca
Content-Type: application/json-patch
Accept-Encoding: gzip
{
    "end": "1545",
}
```

- Exemple de réponse :
  - Option 1

```
HTTP/1.1 200 Ok
Content-Type: application/json
Content-Encoding: gzip

{
    "id": "m426-789ab-cdef",
    "employee": {"id": "982-cr85-g24a4d"}
    "date": "2015-01-07",
    "start": "1400",
    "end": "1545",
    "subject": "Simulation prêt habitat",
    "partner": {"id": "89ab-cdef-0123-456789abcdef"}
}
```

  - Option 2

```
HTTP/1.1 204 No Content
```

2.6 Suppression d'une ressource : DELETE

Pour supprimer une ressource, on utilise la méthode DELETE. Le comportement standard est le suivant :

- L'émetteur envoie une requête sur la ressource.
- Le fournisseur supprime la ressource si elle existe. Dans la réponse il restitue :
  - Un code retour http `204 No Content`.
- En cas d'erreur, le fournisseur restitue :
  - Un code retour adapté : `404 Not Found` si la ressource n'existe pas ;
  - Un message d'erreur dont le format est décrit en 2.9.

- Exemple :

```
DELETE /meetings/m426-789ab-cdef
Host: api.casa.group.gca
```

2.7 Vérification de l'existence d'une ressource : HEAD

La méthode HEAD est utilisée pour :

- Retrouver rapidement des informations concernant une ressource, un serveur, un fichier ;
- Vérifier qu'un URI est actif, qu'un lien fonctionne ;

- Vérifier la date de dernière modification, l'empreinte, la longueur d'une ressource ;
- Mettre en œuvre des statistiques de comptage ;
- …

L'implémentation des requêtes HEAD est indispensable car elles sont utilisées de manière masquée par les navigateurs lorsque l'on met en œuvre des dispositifs de gestion de cache.

Le comportement standard est le suivant :

- L'émetteur envoie une requête sur la ressource.
- Le fournisseur restitue une réponse identique à celle d'un GET mais sans la représentation (pas de body) :
  - Un code retour http `200 Ok`,
  - Les en-têtes identiques à celles qui seront restituées par une requête GET sur la ressource. Par exemple :

Le type de contenu	(Content-Type)
Les éléments concernant la fraîcheur	(Last-Modified, ETag)
La taille de la ressource	(Content-Length)

— Exemple de requête :

```
HEAD /meetings/m426-789ab-cdef
Host: api.casa.group.gca
Accept: application/json
```

— Exemple de réponse :

```
HTTP/1.1 200 OK
Content-Type: text/html; charset=UTF-8
Date: Fri, 09 Jan 2014 15:36:11 GMT
ETag: "783456-3g8-5fg56b2970ec1"
Last-Modified: Thu, 25 Dec 2014 23:59:58 GMT
Content-Length: 900
```

2.8 Découverte des actions autorisées sur une ressource : OPTIONS

La méthode OPTIONS peut être utilisée sur une ressource pour en vérifier l'existence et connaître les actions autorisées.

L'implémentation des requêtes OPTIONS est indispensable car elles sont utilisées de manière masquée par les navigateurs lorsque l'on met en œuvre des dispositifs permettant l'envoi de requêtes "cross domain" (voir [CORS] pour plus de détails).

Le comportement standard est le suivant :

- L'émetteur envoie une requête sur la ressource.

- Le fournisseur restitue dans la réponse :
  - Un code retour http `200 Ok` si la ressource existe,
  - Un en-tête `Allow` qui indique les méthodes autorisées sur la ressource.

— Exemple de requête :

```
OPTIONS /meetings/m426-789ab-cdef
Host: api.casa.group.gca
Accept: application/json
```

— Exemple de réponse :

```
HTTP/1.1 200 OK
Allow: GET,HEAD,PUT,PATCH,OPTIONS
Content-Type: text/html; charset=UTF-8
Content-Length: 0
```

Le comportement dans le cadre de CORS est le suivant :

- Une requête OPTIONS est générée par le navigateur avant la requête effective lorsque :
  - La requête utilise un verbe http différent de GET, HEAD et POST
  - La requête utilise un en-tête spécifique (ex. `X-API-Key:`)
  - Le MIME type du contenu (body) transmis avec la requête est différent de "text/plain", "application/x-www-form-urlencoded", "multipart/form-data".
- Le fournisseur restitue dans la réponse :
  - Un code retour http `200 Ok` si la ressource existe,
  - Un en-tête `Access-Control-Allow-Methods:` qui indique les méthodes autorisées sur la ressource.

— Exemple de requête :

```
OPTIONS https://api.casa.group.gca/products
Origin: https://casa.group.gca
Access-Control-Request-Method: POST
Access-Control-Request-Headers: Api-Key
```

— Exemple de réponse :

```
HTTP/1.1 204 No Content
Access-Control-Allow-Methods: GET, POST, OPTIONS
Access-Control-Max-Age: 86400
Access-Control-Allow-Origin: https://casa.group.gca
Content-Length: 0
```

2.9 Gestion des retours en erreur

2.9.1 Erreurs d'authentification

Voir §5.3.4.5 du document « Norme API Groupe - Spécifications détaillées ».

2.9.2 Erreurs côté serveur

Lorsqu'une requête se termine en erreur pour une cause impliquant le serveur ciblé (erreur généralement technique), l'applicatif n'a en général pas la possibilité de restituer un message d'erreur spécifique.

Préconisation 41

Les erreurs serveur sont restituées dans le body de la réponse sous la forme d'un code, permettant d'affiner le code http, et d'un libellé. Ils sont tous deux obligatoires mais libres dans le contenu.

Les erreurs serveur sont associées à un code retour http 500.

Les erreurs de nature technique détectées lors du déroulement d'une requête (serveur applicatif indisponible, base de données indisponible, moniteur transactionnel indisponible, exception détectée) entrent dans le cadre des erreurs serveur.

– Exemple :

```
GET /clients/568255-66542-569225 HTTP/1.1
Host: api.casa.group.gca

HTTP/1.1 500 Server Error
{
    "error": "UOM35268CD500230",
    "error_description": "Base de donnees Host non disponible"
}
```

2.9.3 Erreurs suite aux données transmises par le client

Préconisation 42

Conformément à la spécification http, les réponses en erreur consécutives à des contrôles sur les données métier transmises par l'émetteur de la requête doivent s'accompagner d'un code status de la forme 4xx.

On distingue :

- Les erreurs liées au format des données, qui nécessitent une modification des données avant de pouvoir soumettre de nouveau la requête. Elles seront associées à un code retour `400 Bad Request` (ou à un code erreur 4xx plus précis comme par exemple le code erreur 415 « Unsupported Media Type » lorsque le format des données du corps de la requête n'est pas supporté par l'API).
- Les erreurs liées à des règles de gestion non satisfaites mais qui pourraient être soumises de nouveau sans modification si une action indirecte vient modifier le contexte de la ressource (solde

insuffisant, domiciliation externe à créer, …). Elles seront associées à un code retour `409 Conflict`.

Préconisation 43

Les erreurs client sont restituées dans le body de la réponse sous la forme d'un tableau d'erreurs.

Chaque erreur est décrite par les propriétés suivantes :

Donnée	O/F	Signification
error	O	**Code erreur** Code fonctionnel de l'erreur qui affine le code http global restitué dans la réponse.
error_lib	O	**Libellé court** de l'erreur
error_description	F	**Description de l'erreur** Paramètre optionnel présent lors d'une réponse en erreur. Il indique la description de l'erreur. Cet attribut a pour objectif de fournir aux développeurs une description suffisamment explicite, mais n'est pas destinée à être affichée aux utilisateurs finaux
error_uri	F	**URI de la page Web décrivant l'erreur** Paramètre optionnel présent lors d'une réponse en erreur. Il indique l'URL de la page Web de description de l'erreur.
errors_value_list	F	Liste des propriétés transmises dans la requête et qui ont généré l'erreur, sous forme d'un tableau.

Remarque : Les API ont la possibilité d'ajouter des attributs supplémentaires (pour retourner des données fonctionnelles, par exemple).

– Exemple :

```
HTTP/1.1 409 Conflict
Content-Type: Application/json

[{
    "error": " UOM35268CD409520",
    "error_lib": "Montant supérieur au plafond",
    "error_description": "Le montant du virement dépasse le plafond de 2000 €
",
    "error_uri": "https://ca-xx.group.gca/monapp/businesserr/E409520",
    "errors_value_list": ["operation_amount"]
}, {
    "error": " UOM35268CD409630",
    "error_lib": "Erreur RIB",
    "error_description": "Un RIB externe doit être créé en back-office avant
de pouvoir être utilisé",
    "error_uri": "https://ca-xx.group.gca/monapp/businesserr/E409630",
    "errors_value_list": ["bic"]
}]
```

2.10 Test de santé des API (Health Check)

Préconisation 44

Comme pour tout développement applicatif, il est préconisé de mettre en œuvre un « health check » applicatif sur les API. L'objectif sera de tester de manière plus approfondi la santé de l'application (accès à la base de donnée, à des services externes, etc.) par rapport au « health check » dit technique (accès réseau, démarrage serveurs, etc.)

Ce « health check » applicatif aura pour objectif d'effectuer les tests de santé applicatifs suivants :

- L'état de santé des accès aux services d'infrastructure externes depuis l'API
- L'état de santé du serveur : espace disque, etc.
- L'état de santé des traitements applicatifs spécifiques
- Etc.

Préconisation 45

Il est préconisé que l'API dispose d'une ressource « health » qui permettra sa surveillance par les outils d'exploitation. L'URI de ce test de santé (« Health check endpoint ») sera « /health ».

La ressource « health » devra être accessible directement sur le back-end de l'API, sans passer par l'APIM. L'accès via l'APIM pourra être envisagé, si besoin et si l'analyse de risque associée le permet.

L'accès à ce point d'entrée (via le back-end directement ou via l'APIM) devra être soumis à contrôle d'accès s'il retourne des informations jugées confidentielles par l'entité.

Les appelants de ce test de santé seront les services d'infrastructure tel que les répartiteurs de charge (« load balancer »), les services de surveillance, les registres de services pour les microservices, etc. Ces appelants seront amenés à interroger périodiquement le test de santé.

Pour harmoniser les mises en œuvre du Groupe et faciliter le travail des équipes d'exploitation, nous recommandons d'appliquer les bonnes pratiques suivantes pour la mise en œuvre des réponses aux tests de santé applicatifs (issues du draft de l'IETF : Health Check Response Format for HTTP APIs).

La réponse au test de santé sera **au format JSON** avec les attributs :

- « **status** » (obligatoire) prenant les valeurs ci-après :
  - « status » : « pass » ou « ok » ou « up » si le test de santé est un succès (API en bonne santé)
    - Code HTTP de la requête de réponse compris dans l'intervalle : 2xx-3xx
  - « status » : « fail » ou « error » ou « down » si le test de santé est un échec (l'API ne fonctionne pas)
    - Code HTTP de la requête de réponse compris dans l'intervalle : 4xx-5xx
  - « status » : « warn » si le test de santé est un succès mais que l'API fonctionne de manière dégradé
    - Code HTTP de la requête de réponse compris dans l'intervalle : 2xx-3xx

- « version » (optionnel) indiquant la version publique majeur de l'API de test de santé.
- « releaseId » (optionnel) indiquant la version détaillée de l'implémentation de l'API (par exemple au format x.x.x)
- « notes » (optionnel) : tableau de notes, remarques, relatif à l'état de santé actuel.
- « output » (optionnel) : sortie d'erreur brut. Utile en cas de « status » égale à « fail » ou « warn ».
- « checks » (optionnel) : objet JSON détaillant l'état de santé des composants sous-jacent à l'API. La description de cet objet « checks » est décrit dans le document IETF : Health Check Response Format for HTTP APIs
- « serviceId » (optionnel) : identifiant unique de l'API de test de santé pour une application donnée
- « description » (optionnel) : description de l'API de test de santé
- Etc. (autres paramètres décrit dans le document IETF relatif au HealthCheck)

Remarque : les attributs « notes », « output », « checks » et « description » peuvent contenir des informations confidentielles nécessitant un contrôle d'accès (non compatible avec les outils de surveillance CAGIP).

Exemple de réponse simple d'un test de santé (adapté aux outils de surveillance CAGIP de l'APIM) :

```
HTTP/1.1 200 OK
Content-Type: application/json
{
     "status": "pass",
     "version": "1"
}
```

Autre exemple de réponse d'un test de santé :

Attention, cet exemple retourne des informations détaillées sur l'état de de santé de l'API. En conséquence, son accès devra être soumis à habilitation si l'entité juge ses informations confidentielles.

```
HTTP/1.1 200 OK
Content-Type: application/json
{
     "status": "pass",
     "version": "1",
     "releaseId": "1.2.2",
     "notes": [""],
     "output": "",
     "serviceId": "f03e522f-1f44-4062-9b55-9587f91c9c41",
     "description": "health of authz service",
     "checks": {
           "cassandra:responseTime": [
           {
              "componentId": "dfd6cf2b-1b6e-4412-a0b8-f6f7797a60d2",
              "componentType": "datastore",
              "observedValue": 250,
              "observedUnit": "ms",
              "status": "pass",
              "time": "2018-01-17T03:36:48Z"
           }
        ],
        "cpu:utilization": [
           {
              "componentId": "6fd416e0-8920-410f-9c7b-c479000f7227",
              "node": 1,
              "componentType": "system",
              "observedValue": 85,
              "observedUnit": "percent",
              "status": "warn",
              "time": "2018-01-17T03:36:48Z"
           },
           {
              "componentId": "6fd416e0-8920-410f-9c7b-c479000f7227",
              "node": 2,
              "componentType": "system",
              "observedValue": 85,
              "observedUnit": "percent",
              "status": "warn",
              "time": "2018-01-17T03:36:48Z"
           }
        ]

     }
}
```

Important : à noter, cependant, qu'à ce stade (octobre 2020), les outils CAGIP de surveillance des API demande l'exposition d'une ressource « health » sur l'APIM et sur le back end avec la possibilité d'accéder à cette ressource de manière anonyme, c'est-à-dire sans authentification de l'appelant et donc sans contrôle d'accès. Ainsi, si la ressource health est destinée aux outils de surveillance de CAGIP, il faudra voir avec CAGIP la pertinence de l'exposer sur l'APIM, et si cela s'impose, il ne faudra pas que la réponse contienne d'informations confidentielles (par contre, si besoin, ces informations pourront être tracées côté serveur pour un accès ultérieur, via ELK, par exemple). D'ailleurs, l'outil de surveillance CAGIP n'exploitera que le code retour de la requête « /health » (code retour 200 si ok) pour connaître l'état de santé de l'API.

3 Annexes

3.1 Documents de référence

Acronyme	Document de référence
ADSU	Architecture de Développement Support de l'Urbanisme. Architecture applicative historique, s'appuyant sur http et XML et un protocole de sécurité des flux inter partenaires nommé IC04. Les spécifications peuvent être consultées à l'adresse suivante : https://ca-sa.ca-mocca.com/site/architecture-entreprise-groupe/CadredAEG/Normesdintegration/ADSU/Pages/ADSU.aspx
ARAPI	Analyse de risque MESARI des API Document sous l'égide du CPST (SECAPI) qui présente les risques associés aux échanges API et y associe une liste de mesures de sécurité à mettre en œuvre pour s'en prémunir : https://secapi.adsi.credit-agricole.fr/lib/exe/fetch.php?media=prod-distrib:norme_api:analyse_de_risques_api_v1.2.pdf (accessible aussi sur l'Intranet SECAPI)
CANV	Travaux concernant les canevas d'échanges dans le référentiel de l'AEG : https://ca-sa.ca-mocca.com/site/architecture-entreprise-groupe/Data/Canevasdechange/Pages/Canevasdechange.aspx
CAT_API	Catalogues des ressources et données API. Les catalogues API fournissent un ensemble de ressources et données réutilisables dans les API exposées par le Groupe. Ces catalogues sont organisés par domaines fonctionnels https://ca-sa.ca-mocca.com/site/architecture-entreprise-groupe/API/Pages/CataloguesAPI.aspx
CORS	Cross Origin Ressource Sharing : spécification permettant côté client l'accès à une ressource depuis un domaine différent de son domaine d'origine. http://www.w3.org/TR/cors/ Les navigateurs compatibles CORS (c'est le cas de tous les navigateurs récents) sont : http://caniuse.com/#feat=cors
CSP	Content Security Policy : spécification permettant de déclarer un ensemble de restrictions à appliquer sur une ressource Web. https://www.w3.org/TR/CSP11/

	Pour la mise en oeuvre, on peut se reporter au tutorial disponible à l'adresse : http://www.html5rocks.com/en/tutorials/security/content-security-policy/
DATE	Sous-ensemble des formats de dates décrits dans le standard ISO 8601 retenu par le W3C afin de limiter les erreurs et la complexité de gestion : http://www.w3.org/TR/NOTE-datetime
HAL	Hypertext Application Language est un standard qui définit des conventions pour exprimer les contrôles hypermédias, comme des liens, avec JSON. Il est en cours de validation à l'IETF et la version de travail des spécifications est disponible. https://tools.ietf.org/html/draft-kelly-json-hal-08
HEALTH	Health Check Response Format for HTTP APIs https://tools.ietf.org/html/draft-inadarei-api-health-check-04
HSTS	http Strict Transport Security : spécification d'un mécanisme de politique de sécurité proposé pour HTTP, permettant à un serveur web de déclarer à un agent utilisateur (comme un navigateur web) compatible qu'il doit interagir avec lui en utilisant une connexion sécurisée (comme HTTPS). http://tools.ietf.org/html/rfc6797
IDHAB	Document de spécification des principes retenus pour la gestion des Identités et des habilitations : RefAEG_IdHab_Specifications_v2_0 disponible avec le présent document sur le site intranet de l'AEG à l'adresse : https://ca-sa.ca-mocca.com/site/architecture-entreprise-groupe/Lists/Documents/Fichiers/NNI/RefAEG_IdHab_Specifications_v2_2.zip
JOSE	JSON Object Signing and Encryption : ensemble de spécification décrivant une technologie utilisable pour signer et chiffrer des données structurées en utilisant le formalisme JSON. https://tools.ietf.org/html/rfc7165
JSON	Format d'échange de données textuelles standard, spécifié par la RFC 7159 de l'IETF https://tools.ietf.org/html/rfc7159
JWA	Catalogue des algorithmes utilisable dans le cadre des spécifications JSON Web Encryption (JWE) et JSON Web Signature (JWS): https://tools.ietf.org/html/rfc7518
JWE	Décrit la structure d'un jeton chiffré JSON Web Encryption (JWE) : https://tools.ietf.org/html/rfc7516

JWS	Décrit la structure d'un jeton signé JSON Web Signature (JWS) : https://tools.ietf.org/html/rfc7515
JWT	JSON Web Token (JWT) décrit un moyen de représenter des propriétés devant être transférées entre deux parties. Les propriétés sont codées sous la forme d'un objet JSON utilisé comme contenu utile d'un jeton signé JSON Web Signature (JWS) ou d'un jeton chiffré JSON Web Encryption (JWE) https://tools.ietf.org/html/rfc7519
NAPI	Document décrivant les normes API Groupe. Il peut être téléchargé sur le site de l'AEG à l'adresse suivante : https://ca-sa.ca-mocca.com/site/architecture-entreprise-groupe/CadredAEG/Normesdintegration/Pages/NormeAPI.aspx
NHOT	Synthèse_GT_Nommage_URL_v1.3.doc Ce document est du ressort de la fonction Pilotage de Infrastructures et des Operations dans la direction IIG. Il devrait être publié dans leur Intranet. Par défaut, il est joint aux spécifications NNI REST dans l'archive à l'adresse suivante : https://ca-sa.ca-mocca.com/site/architecture-entreprise-groupe/CadredAEG/Normesdintegration/Pages/IntegrationServices.aspx
REC	Référentiel des échanges : https://ca-sa.ca-mocca.com/site/architecture-entreprise-groupe/CadredAEG/ReferentieldesEchanges/Pages/ReferentieldesEchanges.aspx
REST	Style d'architecture décrit par Roy Fielding dans le document disponible à l'adresse https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm
RMM	Modèle de maturité dans l'adoption des principes de REST, comprenant plusieurs niveaux allant de l'adoption minimum jusqu'à l'adoption complète, décrit par Leonard Richardson https://martinfowler.com/articles/richardsonMaturityModel.html