Introduction

Chaque équipe est responsable du respect de ces standards pendant les développements d'API et chaque membre est encouragé à faire évoluer ces guidelines de manière collaborative, avec les membres de la communauté de pratique API.

Dans le cas d'une évolution de ces standards API, il faut suivre les règles suivantes:

- les API existantes ne doivent pas être systématiquement changées, bien que cela soit conseillé,
- les nouvelles API doivent respecter la version courante des standards.

Ce présent document utilise les termes **DOIT**, **DEVRAIT**, **PEUT** comme mots-clés pour définir le niveaux d'exigence d'une spécification telle que définit dans la RFC 2119 (version française).

Terme	Définition
DOIT	Ce mot, ou les termes "**OBLIGATOIRE**" ou "**DEVRA**", veut dire que la définition est une exigence absolue de la spécification.
NE DOIT PAS	Cette phrase, ou la phrase "**NE DEVRA PAS**", veut dire que la définition est une interdiction absolue de la spécification.

Terme	Définition
DEVRAIT	Ce mot, ou l'adjectif "**RECOMMANDÉ**", veut dire qu'il peut exister des raisons valides dans des circonstances particulières pour ignorer un item particulier, mais les répercussions doivent être comprises et soigneusement évaluées avant de choisir un cours différent.

Terme	Définition
NE DEVRAIT PAS	Cette phrase, ou la phrase “**NON RECOMMANDÉE**” veut dire qu’il peut exister des raisons valides dans des circonstances particulières quand le comportement spécifique est acceptable ou même utile, mais les répercussions complètent devraient être comprise et le cas soigneusement évalué avant d’implémenter tout comportement décrit avec cette étiquette.

Terme	Définition
PEUT	Ce mot ou l'adjectif "**OPTIONNELLE**", signifie qu'un item est vraiment optionnel. Un vendeur peut choisir d'inclure l'item parce qu'une place de marché spécifique l'exige ou parce que le vendeur pressent que cela améliore le produit alors qu'un autre vendeur peut omettre le même item. Une implémentation qui n'inclut pas une option particulière **DOIT** être préparée à inter-opérer avec une autre implémentation qui n'inclut pas l'option, même peut-être avec une fonctionnalité réduite. Dans la même veine, une implémentation qui inclut vraiment une option particulière **DOIT** être préparée à

Terme	Définition

Table des matières

Principes de base

Cette section couvre les principes de base.

API First

Designer l'API avant de commencer l'implémentation technique

La signature de l'API **DOIT** être établie avant de commencer l'implémentation (OpenAPI specification, Stub, etc).

L'objectif d'une telle démarche est de permettre des retours le plus rapidement possible et d'avoir une discipline interne qui se concentre sur:

- la connaissance du domaine exposé ainsi que les fonctionnalités requises
- des entités / ressources business généralisées, i.e., éviter d'avoir des APIs pour des use-case spécifiques
- une réelle séparation entre le QUOI et le COMMENT

La description de l'API est l'unique vérité, et non pas l'implémentation technique de l'API. Si l'outil pour décrire l'API ne supporte pas nativement la description correcte de l'API, il est toujours possible d'écrire une documentation à la main.

L'implémentation d'une API **DOIT** toujours être conforme avec la description de l'API : cela représente le contrat entre l'API et ses consommateurs.

Compatibilité

Ne pas casser la rétrocompatibilité

Les changements d'APIs dans une même version majeur **NE DOIVENT PAS** casser la rétrocompatibilité. Les APIs sont un contrat entre le fournisseur de service et les consommateurs qui ne peut pas être cassé par des décisions unilatérales.

Il y a deux possibilités pour changer une API sans la casser:

- suivre les règles des extensions compatibles
- introduire une nouvelle version d'API tout en supportant les anciennes versions.

Règles pour faire évoluer une API

Chaque modification mineure d'une API DOIT suivre les règles d'extension:

- **NE DOIT PAS** retirer de champs/propriétés,
- **NE DOIT PAS** rendre requis des champs optionnels,
- **NE DOIT PAS** supprimer un endpoint existant,
- Tout ce qui est rajouté **DOIT** être optionnel.

Si l'une des règles ci-dessus ne peut être respectée pour une quelconque raison, un développeur d'API **DOIT** publier une nouvelle version majeure.

Documentation

Documentation générale

Une API **DOIT** faire l'objet d'une documentation sous forme d'une page dans notre Wiki d'entreprise contenant au moins:

- Un descriptif
- L'équipe responsable clairement identifiée
- Lien vers les documentations Swagger

Cette page wiki **DOIT** être ajoutée à l'annuaire.

Nota Bene : La communauté de l'API est tout à fait consciente que cette solution est temporaire. A terme, nous nous attendons à avoir un service registry centralisé, probablement dans un outil d'API Management.

Documentation swagger

Une API **DOIT** exposer une documentation explicite, complète et à jour de ses endpoints et **DEVRAIT** l'exposer sous forme d'un Swagger.

REST

Ressources au lieu de verbe

Les APIs **DOIVENT** être designées autour des ressources, et **NE DOIVENT PAS** représenter des actions. Une API **PEUT** inclure les contrôles hypermedia (HATEOAS).

Niveau de maturité

Idéalement, nous visons le niveau 2 du modèle de maturité de Richardson, mais il est tout à fait possible d'utiliser le niveau 3 (Plus d'informations).

REST est centralisé autour des entités/ressources et l'utilisation des méthodes HTTP standards (tq. GET/POST/PUT/DELETE) en tant qu'opérations. Les URLs ne doivent contenir que des noms, et non pas des verbes.

Par exemple, au lieu d'avoir le verbe annuler dans l'URL, il faut plutôt penser à la ressource annulation.

Utilisation des verbes

Les méthodes Http standard ont une signification, elles sont à utiliser pour déterminer le type d'action à effectuer.

Bien que ces méthodes ne soient pas équivalentes à du CRUD, il est préférable dans notre cas de les utiliser comme tel à des fins de simplification et de ne garder que des créations non idempotentes.

Méthode	Action	Définition	
POST	Non-idempotent	Créer une resource	C
GET	Nullipotent (Safe)	Retourner une/des resource(s)	R
PUT	Idempotent	Modifier une ressource	U
DELETE	Idempotent	Supprimer une resource	D

POST

- Un POST (create, dans notre cas) exécuté avec succès retourne un 201. Le header doit contenir un header `Location` donnant le lien vers la nouvelle entité.
- Dans le cas d'une opération asynchrone, la réponse doit être un 202 contenant un header `Location` permettant de monitorer l'état de l'opération.

GET

- Un GET retournant une ressource exécutée avec succès retourne un 200.
- Un GET retournant plusieurs ressources avec succès retourne un 200 si toutes les ressource sont présentes ou un 206 si une partie des ressources est retournée (paging, top n). Dans ce cas-là, la réponse doit contenir un header `Content-Range`.

PUT

- Un PUT (update, dans notre cas) exécutée avec succès retourne un 200 ou un 204.
- Dans le cas d'une opération asynchrone, la réponse doit être un 202 contenant un header `Location` permettant de monitorer l'état de l'opération.

DELETE

- Un DELETE exécuté avec succès retourne un 200 ou un 204.
- Dans le cas d'une opération asynchrone, la réponse doit être un 202 contenant un header `Location` permettant de monitorer l'état de l'opération.

Message

Cette section couvre la gouvernance sur le format des messages.

Données et représentations

Encodage

Les données **DEVRAIENT** être encodées en UTF-8.

Enumérations

Les données **DEVRAIENT** être représentées sous forme d'énumérations plutôt que sous forme de codes cryptiques. De plus, les positions d'énumérations **DEVRAIENT** être sérialisées sous forme de chaîne de caractères en `camelCase` afin d'éviter les erreurs de mapping.

```
Content-type: application/x.va.validation+json
{
    // Pas d'ambiguité
    "titre":"baron",

    // Risque d'erreur de mapping, risque de dérive en maintenance évolutive
    "titre": 4
}
```

Données et affichage

Lorsqu'une propriété peut être exprimée soit sous forme de données, soit sous forme d'affichage, l'API **DEVRAIT** l'énoncer clairement.

```
Content-type: application/x.va.validation+json
{
    // Par défaut, de la data
    "myDateTime": "1997-09-02T19:20:30.45+01:00",
    // Expliciter lorsqu'il s'agit de display. C'est assez long ?
    "myDateTimeDisplay": "Lundi 2 septembre à 19heures 20 minutes 30 secondes 45 centièmes,

    // Par défaut, de la data
    "myDate": "1985-08-09",
    // Expliciter lorsqu'il s'agit de display et de l'anniversaire à JFR
    "myDateDisplay": "Vendredi 9 août 1985",

    "gender":"M",
    "genderDisplay":"Male"
}
```

Booléens

Le nom des propriétés booléennes **POURRAIT** être préfixé par `Is` ou `Has` afin de rendre la nature du champ plus claire.

Identifiant

Pour des raisons de sécurité, les identifiants techniques exposés **DEVRAIENT** être non-séquentiels et non-déterministes, par exemple, `UUID` v4 RFC-4122.

Représentation commune des données business

L'API **DEVRAIT** se baser sur la représentation commune des données business. Pour plus d'informations, consulter Représentation communes des données business (Internal link).

Validations métiers

Format des validations métiers

En cas d'un échec de la requête pour des raisons de validation métier, la réponse **DEVRAIT** utiliser un code HTTP 422, **DEVRAIT** avoir un Content-Type clairement défini

`Content-type: application/vnd.va.validation+json`

et **DEVRAIT** retourner une payload avec

```
{
    "validations": [
        {
            // Field translated according the i18n/l10n and displayable to the user
            "display": "The name is required",

            // ValidationCode used to defined the label
            "code":"validationRequired",

            // Concerned field(s)
            "fields":["firstName"],

            // Variable data that are in the message (Validation property)
            "valParams":{}
        },
        {
            // Field translated according the i18n/l10n and displayable to the user
```

```
            "display": "Le npa devrait comporter au moins 42 caractères",

            // ValidationCode used to defined the label
            "code":"validationMinLength",

            // Concerned field(s)
            "fields":["address[0].npa"],

            // Variable data that are in the message (Validation property)
            "valParams":{
                "min": 42
            }
        },
        {
            ...
        }
    ]
}
```

Erreurs métiers

Format des erreurs métiers

Lors de l'échec d'une opération métier, les statuts **DOIVENT** être de l'ordre de 4XX, le `Content-Type` **DEVRAIT** être

```
Content-type: application/vnd.va.error+json
```

et le contenu de la payload **DEVRAIT** être

```
{
    // Technical field
    "message": "This message will not be displayed to the user",

    // i18n/l10n field which can be displayed to the user
    "display": "If this error append again, please call your mama!",

    // Standard error code used by the client to define a specific label to display
    "code":"uniqueErrorCodeForDoesNotWork"
}
```

Exception

Format des exceptions

Sur l'environnement de production, une exception logicielle **DOIT** retourner un code HTTP 500 et **NE DOIT PAS** retourner de stack trace.

Sur les environnements non productifs, la payload retournée **DEVRAIT** ressembler à

```
Content-type: application/vnd.va.exception+json
{
    // Champs techniques habituels
    "message": "object not set to an instance",
    "stackTrace": "...",
    "innerException": {...}
}
```

Payload JSON

Format - négociation de contenu

Les payloads **DEVRAIENT** être retournées au format application/json et **DOIVENT** respecter les conventions de ce format (`camelCase`, etc). Un service **PEUT** traiter d'autres formats (xml, yml) via le header standard Accept.

JSON'ception

Les propriétés contenues dans une payload JSON **NE DOIVENT PAS** contenir elles-mêmes du json ou du xml.

Requête

Cette section couvre les standards liés aux requêtes (i.e. filtre, pagination, tri, asynchronisme, etc).

Asynchronisme

Lors d'une opération conduite de manière asynchrone par le serveur, celui-ci **DOIT** retourner un code HTTP 202 avec un header Location désignant l'emplacement de l'URL de suivi de l'opération. Cet URL pointera sur une ressource de type operations.

`Location: https://VaHappyHi:8081/v2/operations/8156ab4e`

La ressource `operation` **DEVRAIT** contenir l'état actuel de l'opération (`notStarted`, `running`, `succeeded`, `failed`).

- Si l'état est `notStarted` ou `running`, alors le code de retour **DOIT** être 202 et le header location reste le même,
- Si l'état est `notStarted` ou `running`, alors le header Retry-After **DEVRAIT** indiquer le nombre de secondes à attendre avant de vérifier l'état de l'opération,
- Si l'état est `succeeded`, alors le code de retour **DOIT** être 200 et le header location doit désormais retourner l'emplacement de la ressource en question.

Impersonation

L'implémentation de l'impersonation **NE DEVRAIT PAS** être implémentée uniquement au niveau du client, mais **DEVRAIT** être au niveau de l'API. L'impersonation **DEVRAIT** se faire au moyen d'un header custom:

`Va-Impersonate: sio`

L'API **DEVRAIT** logguer le fait que l'action a été effectuée par un utilisateur A impersonant un utilisateur B.

JSON Patch

La modification d'un objet peut se faire via une requête http `PUT`. De plus, l'utilisation du `PATCH` est possible via la description d'opérations telles que décrites par la RFC-6902 (**JavaScript Object Notation (JSON) Patch**).

On **DEVRAIT** se limiter aux opérations add, remove, replace. Les autres opérations décrites dans la RFC ne DEVRAIENT pas être utilisées.

```
si un objet est
{ firstName:"Albert", contactDetails: { phoneNumbers: [] } };

et que l'on applique la suite d'opérations suivantes:
[
  { op:"replace", path:"/firstName", value:"Joachim" },
  { op:"add", path:"/lastName", value:"Wester" },
 { op:"add", path:"/contactDetails/phoneNumbers/0", value: { number:"555-123" } }
];

L'objet DOIT être transformé en
{ firstName:"Joachim", lastName:"Wester", contactDetails: { phoneNumbers: [{number:"555-123"}]
```

Attention, il a été constaté que le swagger peut ne pas être généré correctement. Dans ce cas, il **DOIT** contenir une description textuelle décrivant qu'il s'agit d'une opération json-patch et quel type d'objet elle reçoit.

Localisation

La langue désirée **DEVRAIT** être définie en utilisant le header `Accept-Language`.

A noter que le contenu de la payload JSON ainsi que les paramètres transmis dans l'URL **DOIVENT** être formatés selon le standard JSON.

Exemple

HTTP Request

```
GET /contracts HTTP/1.1
Accept-Language: fr-ch, de-ch
```

HTTP Response

```
HTTP/1.1 200 OK
Content-Type: [...]
Content-Language: fr-ch

[...]
```

Pagination

L'accès à des listes de données **DOIT** supporter la pagination pour une meilleure expérience du côté du consommateur. Cette affirmation est vraie pour toutes les listes qui sont potentiellement plus grandes que quelques centaines d'enregistrements.

Il existe deux types de techniques d'itération:

- Offset/Limit-based,
- Cursor-based.

Il est important de prendre en compte la façon dont est utilisée la pagination c/o les consommateurs. Il semblerait que l'accès direct à une page spécifique est bien moins utilisé que la navigation via des liens de type *page suivante/page précédente*. De ce fait, il vaut mieux favoriser la pagination de type *cursor-based*.

Nomenclature

Cette section couvre les standards liés aux nomenclatures des ressources, URI, etc.

Règles générales

Règles générales de nommage

Le développement d'API **DOIT** se faire en anglais, **NE DOIT PAS** contenir d'acronyme et **DOIT** utiliser le `camelCase` (sauf indication contraire).

Lexique

Les noms **DOIVENT** provenir du Glossaire métier (Internal link), et dans un deuxième temps, se baser sur le référentiel de l'AFA (Specific Insurance Link).

URI

Chaque URI DOIT suivre les Règles générales de nommage, à part pour le `camelCase`. A la place, un tiret `-` **DEVRAIT** être utilisé pour délimiter les mots composés. De plus, une URI **NE DOIT PAS** terminer avec un slash `/`.

Exemples

```
// Retourne toutes les personnes
GET https://MyHappyApi:8081/v2/people
// Retourne la personne d8a0f1ed
GET https://MyHappyApi:8081/v2/people/d8a0f1ed

// Retourne la liste des sous resources home in one de la personne d8a0f1ed
GET https://MyHappyApi:8081/v2/people/d8a0f1ed/home-in-one
// Retourne la sous resource home in one 587d038d de la personne d8a0f1ed
GET https://MyHappyApi:8081/v2/people/d8a0f1ed/home-in-one/587d038d

// Retourne la configuration actuelle
GET https://MyHappyApi:8081/v2/configuration
// Retourne la configuration de la personne d8a0f1ed
GET https://MyHappyApi:8081/v2/people/d8a0f1ed/configuration
```

Versioning

La version de l'API **DEVRAIT** être spécifiée dans le segment d'url suivant le root du serveur et **DOIT** représenter le premier digit – *majeur* – de la version sémantique.

`https://MyHappyApi:8081/v2/...`

De plus, sur les environnements de développement non-productif, la dernière version **POURRAIT** être accessible par le segment latest, i.e.,

`https://MyHappyApi:8081/latest/...`

Protocole

Cette section traite des problématiques au niveau du protocole et de ses standards.

HTTP

Protocole HTTP

Toutes les API **DOIVENT** supporter le protocole HTTP et sa sémantique.

Codes HTTP

Quelques règles pour l'utilisation des codes HTTP, le développeur d'API

- **NE DOIT PAS** inventer des nouveaux codes HTTP ou dériver de leur sens originel,
- **DOIT** fournir une documentation de qualité lors de l'utilisation de codes HTTP non-listés ci-dessous.

2XX Success

La requêtes a été traitée avec succès.

Code	Definition
200 OK	Succès de la requête
201 Created	Resource créée avec succès
202 Accepted	Requête acceptée mais non complétée (process asynchrone…)
204 No content	Succès de la requête, réponse vide
206 Partial	Résultat partiel (voir pagination)

4XX Client Errors

La requête contenait une erreur de la part de l'appelant.

Code	Definition
400 Bad request	La requête n'est pas valide (syntaxe, taille,...)
401 Unauthorized	Le client n'est pas authentifié
403 Forbidden	Le client ne possède pas des droits nécessaires
404 Not found	La ressource demandée n'existe pas
416 Range Not Satisfiable	Range Not Satisfiable
418 I'm a teapot	Une requête de café a été envoyé à une théière
422 Business validation	Un échec de la requête est du à une erreur de validation métier

Remarque: dans le cas d'une collection vide, le résultat se doit d'être un 200 retournant un tableau vide. Le 404 n'est pas approprié puisque, bien que vide, la collection existe.

5XX Server Errors

Le serveur n'a pas pu traiter la requête.

Code	Definition
500 Internal server error	Une exception inattendue s'est produite

TLS

Une API utilisant le protocole HTTP **DEVRAIT** utiliser l'HTTPS.

Exploitation

Cette section couvre les standards liés à l'exploitation.

Environnements

Une API déployée en production **DOIT** systématiquement être déployé en amont sur un environnement de qualité assurance (autrement nommé UAT).

S'il est nécessaire d'avoir d'autres environnements, un développeur d'API **DEVRAIT** se baser sur les DNS Names Convention (Internal Link) existantes.

Monitoring

Monitor l'utilisation de l'API

L'équipe en charge d'une API utilisée en production **DEVRAIT** s'assurer qu'elle est monitorée.

Health check

Une API **DEVRAIT** fournir un endpoint de health check sous la forme de

```
{
  "name": "Va.Api.Business.MyAwesomeProduct",
  "status": "up",
  "dependencies": {
    "Va.Api.Tech.Dependency1": {
      "depth": 1,
      "status": "up"
    },
    "Va.Api.Tech.SubDependency": {
```

```
      "depth": 2,
      "status": "up"
    }
  }
}
```

De plus, la chaîne d'intégration continue **POURRAIT** l'utiliser afin de s'assurer que le déploiement s'est bien déroulé.

Informations

Dans les environnements non production, une API **DEVRAIT** fournir un endpoint exposant les informations de dépendance à ses librairies Vaudoise.

```
{
  "product": "Va.XCut.Back.Actuators.Core",
  "version": "1.0.0.13490",
  "libraries": [
    {
      "name": "Va.XCut.Api.Template.Application",
      "product": "Va.XCut.Api.Template",
      "version": "0.0.0.13490",
      "informationalVersion": "0.0.0",
      "configuration": "Debug"
    },
    {
      "name": "Va.XCut.Back.Logger.Std",
      "product": "Va.XCut.Back.Logger.Std",
      "version": "1.0.0.13490",
      "informationalVersion": "1.0.0-Beta01",
      "configuration": "Debug"
  ]
}
```

Metriques

Dans les environnements non production, une API **DEVRAIT** fournir un endpoint exposant les unformations systeme basiques du serveur l'hébergeant.

```
{
  "machineDomain": "VAUDOISE",
  "machineName": "DEVABCDEF",
  "machineOS": "Microsoft Windows 10.0.10240 ",
  "machineProcessorCount": 8,
  "environmentName": ".NET Core 4.6.26606.02",
  "environmentArchitecture": "x64",
```

```
  "serviceName": "Va.XCut.Api.Template.Application",
  "serviceProcessId": 8752,
  "serviceStartTime": "2018-07-05T07:29:44.4771925+02:00",
  "serviceMemory": 92827648,
  "serviceThreads": 21
}
```