API

- REST
  - choix de vision
    - J'ai toujours fait du web
      - ça continue la façon dont je pense les choses
    - SOAP
      - franchement ?
        - Simple ?
      - certainement adapté à d'autres cas d'usages
        - comme pour se tirer dans le pied
        - ou pour d'autres trucs plus sérieux mais qui m'attirent tout autant
  - Pragmatique
    - On oublie la littérature Restful et autres HATEOAS
      - c'est joli
      - c'est attirant
    - on prend les raccourcis assez souvent
    - basé sur mes retours d'expérience et ceux des autres
      - celles de white house
        - https://github.com/WhiteHouse/api-standards
      - ou celles de Vinay Sahni
        - http://www.vinaysahni.com/best-practices-for-a-pragmatic-restful-api#hn
    - bien penser aux obectifs
      - Favoriser les usages
        - simple
        - rapide à essayer
        - rien à apprendre
        - peu de dépendances
      - Ne pas bloquer le futur
        - rester souples
        - pas de "ça tombe en marche" pour l'utilisateur
        - évolutif
      - ne pas contraindre le présent
        - favoriser des usages non prévus
        - simple à maintenir
        - ne pas refaire les mêmes erreurs que tout le monde
        - ne pas réinventer la roue
    - The pessimist complains about the wind;
      the optimist expects it to change;
      the realist adjusts the sails
    - Opinion
- Vos clients sont votre premier problème
  - fainéants ou manque de temps/volonté
    - ne chercheront pas très loin
    - ne lisent pas les docs
    - vont avoir du mal
    - **Faire des exemples**
    - **Proposez un bac à sable**
    - **Faire un SDK**
    - **prévoir un jeu de tests de recette automatisable**
    - **pretty print**
  - pas forcément les mêmes connaissances techniques
    - bloqueront sur des trucs simples
    - leur faciliter la vie
      - **éviter les surprises sur la forme des URL**
        - a - z
          - minuscules seulement
        - 0 - 9
        - tiret (haut)
        - point
          - uniquement pour l'extension
        - slash
          - comme séparateur de "répertoire"
        - ? & et =
          - pour les paramètres
        - pas d'autres caractères
          - même en %xx
            - surtout en %xx
        - pas plus de 76 caractères
          - oui, pour les emails
      - **URL bidouillables**
        - oubliez les URL significatives
          - vous perdriez votre temps
        - mais moindre surprise
          - pouvoir remonter dans la hiérarchie
          - changer un identifiant, une date
          - changer un contexte
          - et que ça fonctionne
          - **Penser les URL comme un système de fichier**
            - ce qu'est presque REST
          - **Paramètres similaires sur toute l'API**
            - même si peu pertinents à certains endroits
            - quitte à trouver plus adéquat
            - et avoir moins de spécifique
            - ce qui est facultatif doit toujours l'être, et inversement
          - **Formats similaires sur toute l'api**
            - format d'entrée = format de sortie
          - **Retours et erreurs similaires sur toute l'api**
      - **évitez les entêtes perso obligatoires**
        - ou alors rendez-là obligatoire
        - et validez-là dès le début
      - vous leur éviterez bien des bugs cachés
  - se basent sur les exemples
    - qu'ils interprètent mal
    - qu'ils vont bidouiller
      - sur des hypothèses qui ne sont pas la théorie de REST, HATEOAS et autres
      - et faire tomber en marche
      - **ce qui leur semble naturel doit fonctionner**
        - même si ce n'est pas votre philosophie
        - **Forme des URL**
    - **Best practice : générateur d'exemples**
  - mais évoluent peu
    - toute une galère pour faire évoluer les clients
      - voir l'exemple github
        - et vous n'avez pas le poids de github
    - si vous évoluez, le code qui tombe en marche ne fonctionnera plus
      - et on vous le reprochera
    - **séparez l'api du domaine web navigation**
      - ils n'évolueront pas à la même vitesse
    - **on en parlera plus tard**
- Vous êtes votre second problème
  - Simplifiez vous la vie
    - **au plus simple**
      - résister à l'envie de croire que vous avez "besoin" de plus
      - il sera toujours temps d'ajouter
        - mais pas de retirer ce qui existe
    - **Structure plate**
      - pas plus de 3 niveaux
      - collection/
      - collection/item
      - collection/filtre
      - collection/item/raccourci
      - **le reste est du paramétrage**
        - éviter le xxx/limit/5/order/date/desc
    - Identifiants
      - **uniques**
      - **texte**
        - pour pouvoir plus tard faire différents cas
      - **prévoyez large**
    - **Ressources simples**
      - **commencez avec du basique sans aggrégat**
        - il sera temps de l'ajouter
        - mais vous ne pourrez pas (facilement) retirer ce qui existe
      - **peu d'options**
    - Nommage
      - **Donnez des noms spécifiques aux collections**
        - fuyez tout ce qui est un minimum générique
        - il sera temps de rediriger plus tard (dans ce sens là)
        - vous risquez le foutoir si votre modèle se complexifie
      - **donnez des noms génériques aux paramètres**
        - pour éviter que deux paramètres similaires sur deux ressources s'appellent différemment
        - permettra aussi de la généricité
      - **singulier, toujours**
        - on peut envisager un pluriel pour les collection et singulier pour les items
        - mais croyez moi, vous finirez par vous posez tout le temps la question
        - et commencer des débats sur le pluriel de certains termes
      - underscore, pas de camel case
    - **favorisez l'obligatoire, fuyez l'optionnel**
    - **uniquement ce qui est nécessaire**
- Les problématiques classiques
  - i18n et l10n
    - **dates au format iso**
      - **avec fuseau horaire**
      - **gmt par défaut**
      - 18 mars 2013, mais que quel fuseau horaire ?
    - noms, prénoms
      - **champ libre**
        - oui, sérieusement
        - de toutes façons ça ne fonctionnera pas
    - **éviter de localiser trop les URL**
    - **tous les messages humains doivent prévoir plusieurs langues**
    - **codage UTF8**
  - identifiants
    - **ils changent**
      - oui, même un numéro de sécu
      - surtout avec les temporaires
        - mais pas que
    - **peuvent être absents**
    - **ils peuvent être multiples**
      - un enfant, deux n° possibles
      - l'email comme clef unique ? argh
    - **attention aux tailles**
      - exemple twitter
    - **toujours se baser sur des codes, des identifiants**
      - et pas des messages
      - exemple des erreurs
      - **mais ajouter des messages textes en plus pour l'humain**
        - qui ne lira pas la doc mais google
  - pagination
    - **Ajouter des limites**
      - refuser une requête sans paramètre de pagination
      - ne pas mettre de paramètre implicite
    - **paginer avec since ou from, pas avec des pages**
      - que se passe-t-il si votre collection change entre deux pages ?
  - versionnement
    - théorie
      - vous ne changerez pas de version
      - garder la compatibilité
      - prendre en entrée des content-type et des accept différents
    - pratique
      - vous aurez besoin de versionner
        - parce que votre métier évolue
        - parce que votre vision évolue
        - parce que sinon ça vous demande trop d'effort
          - ou de complexité
        - et parce que vous vous planterez au début
          - plusieurs fois
      - et ça va être une galère pour faire mettre à jour
      - vous devrez maintenir plusieurs versions
    - solution
      - garder la compatibilité autant que possible
        - enrichir les services
        - ajouter des ressources
        - retarder la nouvelle version le plus longtemps possible
      - **pas de version mineure**
      - **uniquement des versions majeures**
        - v1, v2, v3
      - pistes
        - dans l'entête
          - complexe
          - ne sera pas utilisé
          - pas assez visible pour l'utilisateur
            - qui au mieux recopiera
        - dans le domaine
          - intéressant
            - mais peut être un peu inutile
        - en fin de ressource
          - peu pratique
          - pas de "racine" commune
        - en paramètre
          - oublié trop souvent, peu pratique
        - **dans la racine de la ressource**
      - **à indiquer dès la v1**
        - pour ne pas risquer d'avoir une version "générique"
      - au pire vous pourrez ignorer si vous ne l'utilisez pas
  - sécurité
    - **http basic par défaut**
    - authentification
      - **oauth si besoin**
        - uniquement si besoin
      - possibilité de token
        - plusieurs token
        - gérables par l'utilisateur
        - //!\ complexe
    - **éviter tout ce qui est custom**
    - abus, spam, filtres
      - **limiter la fréquence des requêtes**
        - limite par IP
        - limite par app
          - **toujours imposer une API_key**
      - graphe, logs, analyse
        - le plus difficile est de détecter ce qu'il se passe
      - **interdire / bloquer tout service qui utilise des mot de passe tiers**
        - utiliser oauth
      - **limiter la profondeur**
        - nombre de résultat
        - complexité des requêtes
    - **ssl**
      - **ne pas prévoir de quoi ignorer le ssl dans le sdk**
  - retours d'erreur
    - **codes standards HTTP**
      - ne pas utiliser le message associé au code
        - peut être modifié par les intermédiaires
    - ajouter un contenu décrivant l'erreur
      - code perso
        - **incrémental, sur toute l'API**
          - avec une doc publique quelque part
          - sera plus simple pour vous aussi
      - message machine
      - message humain
        - différent du message machine
        - attention à la localication
  - format
    - type
      - **json**
        - hype
          - si votre cible attend du json
          - ou si clients js
        - pas de liens
        - pas d'extension
        - évolution pas si pratique
        - ne pas écarter trop vite
      - **xml**
        - xlink
        - attributs / métadonnées
        - espaces de nom
        - pas vraiment plus lent
        - pas bcp plus complexe
        - pas mal de formats existants
      - **html**
        - avec micro-formats micro-data
        - idéal car aussi navigable
        - en pratique complexe à produire
        - et complexe à relire
      - **pratique**
        - privilégier le format natif de vos données
        - éviter le CSV
        - entrée en form-url-encoded
    - **structuration**
      - hash et non donnée directe
      - toujours des identifiants
      - ajouter des liens quand possible
      - ajouter des descriptions
      - différenciez ce qui est pour la machine et pour l'utilisateur
    - **pas d'enveloppe**
  - **performance**
    - limite de profondeur
    - limite de nombre de résultats
    - grip
    - limites de fréquences
      - annoncées
  - accès
    - GET
    - POST
    - PUT
    - PATCH
    - DELETE
    - LOCK ?
    - possibilité de surcharger la méthode via paramètre
    - json-patch ?
    - paramétrage
      - minimum de paramètres
      - communs à toutes les ressources
  - cache
    - **pensez au cache**
      - ajouter le cache control
      - si la ressource change, l'url aussi
    - if-modified-since
    - etag
  - documentation
    - sdk
    - doc simple type apiary
    - ne sera pas lue
    - et ne sera pas à jour
    - ni complète
  - hypermedia
    - exemple pour la pagination
    - embed / expand
- générique
  - faire peu
  - simple
  - standard
  - mais pragmatique
  - enrichir uniquement dans un second temps
  - éviter les versions