



règles du jeu

[CHOIX TECHNIQUES]

Les **choix techniques** sont
effectués et
assumés par la
Feature Team.





règles du jeu

[CHOIX TECHNIQUES]

La Feature Team doit agir de manière **responsable** afin d'identifier les choix qui l'impactent exclusivement et les choix qui impactent l'organisation.

Les **choix qui dépassent le périmètre** de la Feature Team (exemple : licence, langage de programmation peu répandu) doivent être soumis à **validation** par l'organisation ou par le processus de convergence des pairs.





règles du jeu

[BON USAGE]

Le **bon outil** pour le
bon usage est une
source d'économies.





règles du jeu

[BON USAGE]

Un **mauvais outil** imposé à tous est un **risque**.

Le **mauvais usage** d'un bon outil peut avoir des **conséquences** très **néfastes**. Par exemple, les méthodes Agiles mal utilisées sont dangereuses.

Les **outils** doivent être **remis en question**.

Excel est souvent un choix **rationnel** mais ça n'est **pas un outil à tout faire** (CRM, ERP, Datamart, ...)





règles du jeu

[BUILD VS. BUY]

Privilégiez le **Build**
pour le coeur de
métier.

Envisagez le **Buy**
pour le reste, au cas
par cas.





règles du jeu

[BUILD VS. BUY]

Plus un outil porte une fonctionnalité apportant un **caractère différenciant** pour l'organisation, plus il a vocation à être **construit**. Le cœur de métier doit permettre la **spécificité** et **s'adapter souvent et rapidement**. Certains **progiciels** sont **parfois adaptés** à ce besoin. Pour le reste : SaaS, Open Source, Build ou Propriétaire sont à étudier **au cas par cas**.





règles du jeu

[OPEN SOURCE]

Privilégiez l'Open Source.

Les choix alternatifs
doivent être étayés.





règles du jeu

[OPEN SOURCE]

Les solutions propriétaires sont un risque pour l'organisation qui doit être capable de reprendre la maintenance si besoin.

Rares sont les outils propriétaires qui n'ont pas d'**alternatives Open Source**.

L'organisation **bénéficie** de la **Communauté Open Source** et peut lui **reverser ses contributions**.





règles du jeu

[MICRO-SERVICES]



Développez des
services **autonomes**
et **faiblement**
couplés.





règles du jeu

[MICRO-SERVICES]

Le **couplage faible** doit être la norme.

Chaque micro-service dispose d'une **interface clairement définie**.

Cette **interface** détermine le **couplage** entre les **micro-services**.

Le **Domain Driven Design** permet, notamment avec les **Bounded Contexts**, d'anticiper au mieux cette problématique.





règles du jeu

[DONNÉES]

Chaque service
possède son
propre système
de stockage de
données.





règles du jeu

[DONNÉES]

Un **Data Store** n'a vocation à être **couplé** qu'avec un seul **micro-service**.

L'**accès aux données** d'un micro-service à un autre est effectué **exclusivement via son interface**.

Ce design implique la **cohérence à terme** à l'échelle de la plateforme. Elle doit être **appréhendée à tous les niveaux**, y compris UX.





règles du jeu

[PÉRIMÈTRE]

Chaque micro-service doit avoir
un périmètre
fonctionnel
raisonnable, qui
"loge dans la tête".





règles du jeu

[PÉRIMÈTRE]

Un micro-service propose
un nombre raisonnable de fonctionnalités.

N'hésitez pas à découper
un micro-service lorsque
celui-ci commence à
grossir.

Un service de taille
raisonnable permet
d'envisager sereinement la
réécriture, si le besoin se
présente.





règles du jeu

[RÉACTIF]

Le **Reactive
Manifesto** ouvre
la voie vers
la conception
d'architectures
réactives.





règles du jeu

[RÉACTIF]

La programmation **réactive** se concentre sur le flux de données et la propagation du changement. Elle s'appuie sur le pattern "**Observer**" contrairement à l'approche "**Iterator**", plus classique.

Le Reactive Manifesto fixe des axes fondamentaux : **disponibilité** et rapidité, **résilience** aux pannes, **souplesse, élasticité et orientation messages**.





règles du jeu

[ASYNC-FIRST]

Les processus
asynchrones
favorisent le
découplage et la
scalabilité au profit
des **performances**.





règles du jeu

[ASYNC-FIRST]

Les échanges entre applications doivent être **en premier lieu asynchrones**.

Les échanges asynchrones permettent **naturellement le couplage faible**, l'**isolation** et le **contrôle des flux (back-pressure)**.

La **communication synchrone** ne doit être envisagée **que lorsque l'action l'impose**.



règles du jeu



[ÉVÉNEMENTS]



Le système
d'informations
doit être orienté
événements.





règles du jeu

[ÉVÉNEMENTS]

Les processus fonctionnels orientés "événements" sont naturellement implémentés de manière asynchrone.

L'orientation événements permet de favoriser la mise en place d'approches telles que Command Query Responsibility Segregation (CQRS) et Event Sourcing.





règles du jeu

[MESSAGE BROKER]



Privilégiez un
**message-broker
simple**, robuste et
performant à un
"tuyau intelligent".





règles du jeu

[MESSAGE BROKER]

Les **ESB** ont montré leurs **limites** : la **maintenance évolutive** est **critique**, aussi bien d'un point de vue **technique** qu'**organisationnel**.

Les **messages brokers** comme **Kafka** offrent une solution **simple, durable et résiliente**.

Des **endpoints intelligents** et des **tuyaux simples** est une architecture qui fonctionne à l'échelle : c'est **Internet**.





règles du jeu

[SYNCHRONISATION]

La **synchronisation totale** du système doit être pensée dès sa **conception**.





règles du jeu

[SYNCHRONISATION]

Si la **synchronisation** entre deux systèmes est assurée par un **flux d'événements**, la **resynchronisation** totale de ces systèmes doit être **prévue dès la conception**.

Un **audit** automatique de la **synchronisation** (exemple : par échantillons) permet de **mesurer** et **détecter** les éventuelles **erreurs** de synchronisation.





règles du jeu

[CENTRALISATION]

La **configuration**
des services est
centralisée, leur
découverte est
assurée par un
annuaire.





règles du jeu

[CENTRALISATION]

La **configuration**
des **micro-services**
est **centralisée** pour
l'ensemble des
environnements.

Un **annuaire** centralisé
permet d'assurer la
découverte dynamique
des **micro-services**.

La **scalabilité** globale du **SI**
dépend de cet **annuaire**.





règles du jeu

[BAC À SABLE]

Les Feature Teams
fournissent un
environnement
"bac à sable".





règles du jeu

[BAC À SABLE]

Les Feature Teams maintiennent un **environnement "bac à sable"** (version actuelle et version à venir) afin de permettre aux autres équipes de **développer à l'échelle**.

Dans **certains cas** non nominaux, des **features** peuvent être **désactivées** dans l'environnement de **développement**.





règles du jeu

[DESIGN FOR FAILURE]



**Votre système
tombera en panne !**

Concevez-le afin
qu'il y soit tolérant.





règles du jeu

[DESIGN FOR FAILURE]

Votre système **tombera en panne**, c'est inévitable. Il doit être conçu pour cela (**Design For Failure**).

Prévoir la **redondance** à tous les niveaux : **matériel** (réseau, disque, etc...), **applicatifs** (plusieurs instances des applications), **zones géographiques**, **providers** (exemple : AWS + OVH).





règles du jeu

[TOOLKITS]

Fournissez des
toolkits, n'imposez
pas de cadres
stricts.





règles du jeu

[TOOLKITS]

Attention aux composants techniques maisons et transverses ! Ils sont contraignants, coûteux et difficiles à maintenir.

Des accélérateurs, toolkits, stacks techniques peuvent être mis en commun, au libre choix des Feature Teams, en évitant une approche dogmatique.





règles du jeu

[CLOUD]

Public, privé ou hybride, le **cloud** (**IaaS** ou **PaaS**) est la norme pour la production.





règles du jeu

[CLOUD]

Les services de type **PaaS** sont à **privilégier**, ils sont **simples** et passent rapidement à l'échelle.

Les services **IaaS** permettent d'adresser les cas demandant une plus grande **souplesse**, mais ils demandent plus de travail opérationnel.

Un cloud privé n'est pas un environnement de virtualisation traditionnel, il s'appuie sur du **commodity hardware**.





règles du jeu

[INFRASTRUCTURE]

Les Feature Teams
ne gèrent pas
l'infrastructure,
elle est **fournie
et maintenue par
l'organisation.**





règles du jeu

[INFRASTRUCTURE]

Les problèmes d'infrastructure ne sont **pas du ressort des Feature Teams**. L'infrastructure doit leur être **fournie et maintenue** par un **service transverse**.





règles du jeu

[CONTENEURS]

Les **conteneurs**
offrent la souplesse
nécessaire à
un outillage
hétérogène.





règles du jeu

[CONTENEURS]

Les conteneurs offrent la **souplesse** nécessaire aux Feature Teams pour leur permettre un **outillage hétérogène** dans un **contexte homogène**.





règles du jeu

[ENVIRONNEMENTS]

L'utilisation de
conteneurs permet
de s'affranchir
des problèmes
d'**environnements**
techniques.





règles du jeu

[ENVIRONNEMENTS]

Les conteneurs (exemple : **Docker**) permettent de **s'affranchir** des différences d'environnement.

Le processus de **déploiement** doit être **agnostique** à l'environnement.

Certains composants comme les bases de données ne doivent pas être déployés dans des conteneurs.

Leur déploiement est malgré tout automatisé.





règles du jeu

[MÉTRIQUES]

Les mesures
doivent être
centralisées et
accessibles à tous.





règles du jeu

[MÉTRIQUES]

Les **métriques** sont **accessibles** à tous avec différents niveaux de granularité : vue détaillé pour la Feature Team concernée, agrégations pour les autres membres de l'organisation.

L'accès aux métriques **n'implique pas l'accès aux données unitaires**, celui-ci doit être contrôlé pour préserver la confidentialité. **Tous les environnements** sont concernés.





règles du jeu

[QUALITÉ]

La **qualité** logicielle
est un **facteur clé**.





règles du jeu

[QUALITÉ]

Les **revues de code** sont **systématiques**. Elles sont effectuées par des membres de la Feature Team ou d'autres membres de l'organisation, dans le cadre de l'**amélioration continue**.

Ça n'est pas vous qu'on **audite** mais **votre code** : "You are not your code !". La **qualimétrie** peut être en partie automatisée, mais rien ne vaut "**l'oeil neuf**".





règles du jeu

[TEST AUTOMATISÉS]

Le **testing automatisé** est
un prérequis
non négociable
au déploiement
continu.





règles du jeu

[TEST AUTOMATISÉS]

Le **testing** automatisé permet d'assurer la **qualité** du produit **dans le temps**.

Il est un **prérequis** au déploiement continu, il permet **changements et déploiements fréquents**.

Le **déploiement en production** devient un événement **anecdotique** !





règles du jeu

[NIVEAUX DE TESTS]

Des **tests à tous les niveaux** : unitaires, intégration, fonctionnels, résilience, performance.





règles du jeu

[NIVEAUX DE TESTS]

Les tests **d'intégration** et **fonctionnels** sont les plus importants, ce sont eux qui **garantissent** le bon **fonctionnement** du **produit**.

Les tests **unitaires** sont adaptés au **développement**.
Les tests de **performance** permettent de mesurer la performance **dans le temps**.

Les tests de **résilience** permettent d'anticiper les **pannes**.





règles du jeu

[COUVERTURE]

La **couverture** est le principal indicateur objectif de qualité des tests.





règles du jeu

[COUVERTURE]

La **couverture** de code par les tests est une **bonne métrique** de la qualité du code.

C'est une **condition nécessaire** mais **pas suffisante**, la couverture d'une **mauvaise stratégie** de tests peut être élevée sans être garante de la bonne qualité du code.





règles du jeu

[SÉCURITÉ]

La **sécurité** est un **processus**, elle ne doit pas être traitée en réaction aux problèmes.





règles du jeu

[SÉCURITÉ]

**Des experts sécurité
peuvent être intégrés
directement aux Feature
Teams si nécessaire.**

**Des experts sécurité
sont disponibles dans
l'organisation pour
auditer, sensibiliser et
transmettre.**





time-to-market

[MVP]



Réalisez votre
**Minimum Viable
Product** pour
éprouver votre
concept.





time-to-market

[MVP]

Le **périmètre** de votre MVP doit être **réduit** tout en permettant de marketer votre produit.

Misez sur les **Early Adopters** et récoltez un maximum de **feedback**.
Votre MVP est déployé et exploitable en **production**.





time-to-market

[FAIL-FAST]



Échouer vite c'est
apprendre vite.





time-to-market

[FAIL-FAST]

Éprouvez rapidement
la solution (quelques
semaines), récoltez
le **feedback** de vos
utilisateurs et apprenez de
vos erreurs.

N'ayez pas peur de tout
changer.

Ne l'oubliez pas, **vous allez**
échouer !





time-to-market

[KISS]



Kee**P** **I**t **S**imple and
Stupid.*

*Pourquoi faire compliqué
quand on peut faire simple ?





time-to-market

[KISS]

Évitez l'over-engineering,
si une maquette "papier"
ou un Google Form suffit
pour éprouver votre
concept, n'allez pas plus
loin.

Restez simple ! À la fois
sur le plan technique et sur
le plan fonctionnel.





time-to-market

[PRODUCTIVITÉ]

Spécifiez moins,
développez plus.





time-to-market

[PRODUCTIVITÉ]

Limitez vos spécifications
au strict nécessaire,
concentrez vous sur le
"quoi" plutôt que sur le
"comment".

Le produit doit être le plus
auto-documenté possible.
La documentation doit être
versionnée au même titre
que le code.





time-to-market

[SAAS]

Étudiez
systématiquement
les solutions **SaaS**.





time-to-market

[SAAS]

Les solutions SaaS sont **pérennes et économiques**. Dans certains cas, le SaaS permet d'**accélérer** la mise en oeuvre d'un **MVP**.

Pensez la vision **économique à terme** vis à vis des alternatives en terme de coût total (TCO : Total Cost of Ownership) et non uniquement en terme de coût de licence.





time-to-market

[COEUR DE MÉTIER]

Le **coeur de métier** ne doit pas être un frein à la construction de nouveaux services et applications.





time-to-market

[COEUR DE MÉTIER]

Le rythme d'évolution et de livraison du coeur de métier doit être **compatible avec l'agilité des services** qui le consomment.

Le coeur de métier doit **exposer des services**.

Le coeur de métier doit adopter un principe **Event-Driven**, il rend compte des actes de gestion sous la forme d'événements.





time-to-market

[DÉPLOIEMENT CONTINU]

Le **déploiement en production** est un non-événement.





time-to-market

[DÉPLOIEMENT CONTINU]

Misez sur le **déploiement continu** afin d'adapter le **déploiement** en **production** aux contraintes et besoins **business** et non l'inverse.

Les **déploiements** à travers les environnements, jusqu'en **production**, doivent être **automatisés** et **fréquents**.





time-to-market

[BÊTA PERPÉTUELLE]

L'approche **bêta
perpétuelle** permet
d'impliquer vos
utilisateurs dans
le processus de
développement.





time-to-market

[BÊTA PERPÉTUELLE]

N'hésitez pas à avoir recours au principe de bêta perpétuelle dans laquelle **les utilisateurs participent au développement.**

Le terme de bêta perpétuelle désigne une application développée en flux tendu, en **constante évolution**, et non pas un produit inachevé.





user experience

[PERCEPTION]



L'**expérience perçue**
par l'utilisateur est
fondamentale.
L'**ergonomie** n'est
pas négociable.





user experience

[PERCEPTION]

Ne négligez pas le travail des **designers UX**, il est fondamental dans le développement d'une application.

Intégrez le **feedback** de vos utilisateurs, celui-ci est essentiel.





user experience

[PERFORMANCE]

Misez sur des
interfaces
performantes, tant
pour les usages
internes que pour
les usages externes.





user experience

[PERFORMANCE]

Les interfaces sont
tournées vers l'**efficacité**.
La **performance**
d'une interface
permet de **gagner du
temps**, d'augmenter
la **satisfaction** des
utilisateurs et donc de
ménager leur frustration.





user experience

[**MOBILE FIRST**]

Adoptez une
stratégie **Mobile
First.**





user experience

[MOBILE FIRST]

Les terminaux mobiles
représentent **la part la plus
importante du marché.**

Penser mobile, c'est
penser à l'**essentiel.**

Le **Responsive Design** est
la norme, c'est une source
d'économies (**MVP**).





user experience

[OMNI-CANAL]

Adaptez-vous aux
usages, l'**omni-**
canal est la norme.





user experience

[OMNI-CANAL]

L'approche omni-canal permet d'offrir à l'utilisateur une **expérience unifiée** (exemple : Netflix). Les différents **canaux** sont **synchronisés et cohérents** (contrairement aux traitements par batchs). Tous les acteurs (clients, conseillers) accèdent aux mêmes informations.





user experience

[SELF-DATA]

Les **utilisateurs**
sont **propriétaires**
de leurs données et
de leur parcours.





user experience

[SELF-DATA]

Laissez aux **individus**, à tout moment, le **contrôle** sur leurs **données** personnelles.

Établissez un **climat de confiance** en permettant aux utilisateurs traçabilité et contrôle en temps-réel. Les **sous-systèmes** doivent répondre aux mêmes exigences.





user experience

[CRM/SFA]



La relation client
doit être unifiée et
contextualisée
grâce à un **CRM / SFA**
souple, fédérateur
et orienté
événements.





user experience

[CRM/SFA]

Optez pour un **CRM** qui gère à la fois la relation client et l'animation de la force de vente (**SFA** : **Sales Force Automation**).

Le **CRM** doit être **ouvert** aux nouvelles opportunités.

Le **CRM** produit des **événements** correspondant aux actes de gestion pour s'inscrire dans la logique **Event-Driven** de la plateforme.





user experience

[BIG DATA]

La plate-forme
Big Data permet
de **centraliser** et
traiter les données
de l'utilisateur pour
servir au mieux son
parcours.





user experience

[BIG DATA]

Centralisez les données du **Groupe Maif**, des **partenaires** et des **fournisseurs** dans une logique de **parcours**.
La "Data Preparation" et les traitements permettent de **consolider** les données.
Les équipes **Big Data** **collaborent** avec les Feature Teams pour assurer la **gouvernance** des données.





user experience

[POSTE DE TRAVAIL]

Le poste de travail
est adapté et
adaptable aux
usages et **canaux**
modernes.





user experience

[POSTE DE TRAVAIL]

Adoptez la **fédération d'identité** pour une expérience unifiée.

Un **portail** permet d'offrir une **vision d'ensemble**, il ne remplace pas les applications.

Le poste de travail doit être **mobile, multi-canal et standard** afin de permettre l'ouverture dans le cadre de l'**entreprise étendue**.





user experience

[COLLABORATEURS]

N'oubliez pas que
vos **collaborateurs**
utilisent à la
maison des
applications
modernes à l'UX
aboutie.





user experience

[COLLABORATEURS]

Traitez tous vos
utilisateurs comme des
"clients" : internautes,
gestionnaires,
opérationnels,
développeurs, etc ...

Ne sous-estimez pas
l'effort d'UX à mettre
en oeuvre pour les
applications de gestion à
usage interne.





user experience

[TOUT MESURER]



Tout ce qui peut être
mesuré doit l'être.

**Sans mesure, tout
n'est qu'opinion.**





user experience

[TOUT MESURER]

Pensez les métriques lors du **développement** de l'application. Les **logs** doivent avoir une dimension **métier** autant **que technique**.

Ne négligez pas les **métriques de performances**, elles sont fondamentales.

La Feature Team assure l'**exploitation** : charge à elle de rendre l'**application exploitable**.





user experience

[DÉGRADATION]



**Envisagez la
dégradation** plutôt
que l'interruption du
service en cas de
panne.





user experience

[DÉGRADATION]

En cas de panne d'un des sous-systèmes, une **version dégradée** du service doit être **envisagée** en premier lieu plutôt qu'une interruption. Grâce aux **Circuit Breakers**, isolez une panne afin d'éviter son **impact** et sa **propagation** sur l'ensemble du **système**.





user experience

[A/B TESTING]

L'A/B Testing

vous permet de
gagner du temps en
laissant le **feedback**
trancher.





user experience

[A/B TESTING]

Plutôt que de trancher arbitrairement entre deux solutions, n'hésitez pas à mettre en place l'**A/B testing**.

Ce pattern consiste à présenter **deux versions différentes** d'une même application et à choisir l'une d'entre elles sur la base de **mesures objectives** de l'activité des utilisateurs.





humain

[FEATURE TEAM]



L'équipe est
organisée autour
d'un **produit** ou d'un
service rendu.





humain

[FEATURE TEAM]

Les équipes sont des **Feature Teams**, organisées autour d'un ensemble fonctionnel cohérent, et composées de l'ensemble des **compétences** nécessaires à cet ensemble. Par exemple : Expert Métier + Développeur Web + Développeur Java + Architecte + DBA + Opérationnel. La **responsabilité est collective**, la Feature Team jouit du pouvoir nécessaire à cette responsabilité.





humain

[2-PIZZA TEAM]

Limitez la **taille des
Feature Teams** (de 5
à 12 personnes).





humain

[2-PIZZA TEAM]

Limitez la taille d'une Feature Team : **entre 5 et 12 personnes.**

En dessous de 5, elle est trop sensible aux événements extérieurs et manque de créativité. Au dessus de 12, elle perd en productivité.

Le terme "**2-Pizza Team**" indique que la taille de la Feature Team ne doit pas dépasser le nombre de personnes que l'on peut nourrir avec deux pizzas.





humain

[ARTISAN LOGICIEL]



Misez sur des
personnes
polyvalentes qui
savent faire et qui
aiment faire.





humain

[ARTISAN LOGICIEL]

Le plus important est la **culture du développement**, l'**évolutivité** et la **faculté d'adaptation**.

Recrutez des **artisans logiciels (software craftsmen) et développeurs full-stack**, ils apportent une vraie plus-value par leur savoir faire et leur vision d'ensemble. Néanmoins, les développeurs mobiles - par exemple - sont généralement des **développeurs spécialisés**.





humain

[RECRUTEMENT]

Soyez **attractifs**
pour recruter les
meilleurs.





humain

[RECRUTEMENT]

Proposez des modes de fonctionnement adaptés aux collaborateurs :

mobilité, home working, CYOD (Choose Your Own Device).

Laissez du temps pour l'expérimentation et faites en sorte que cela soit **organisé dans le temps de travail.**





humain

[VEILLE]

L'organisation doit
être un moteur de
veille

La veille fait partie
du métier.





humain

[VEILLE]

L'organisation doit être un moteur de **veille** en mettant en place des dispositifs tels que la **formation continue** ou les **Universités d'entreprise**. N'hésitez pas à les combiner avec d'autres moyens plus **informels** tels que : **Coding Dojos, Brown Bag Lunchs, Conférences** externes.





humain

[CO-CONSTRUCTION]

Cassez les barrières
entre les métiers,
misez sur la
convergence des
objectifs.





humain

[CO-CONSTRUCTION]

Pour casser les barrières entre les métiers, il ne suffit pas de regrouper les gens autour d'un produit commun dans un lieu commun.

Les démarches Agiles permettent de supprimer ces barrières afin d'assurer la **convergence des objectifs**.

Ces pratiques font partie intégrante des clés du succès, l'organisation en est garante.





humain

[DEVOPS]

Les pratiques
DevOps permettent
de faire tomber les
murs entre Build et
Run.





humain

[DEVOPS]

Adoptez **DevOps** pour faire converger **Dev** et **Ops** vers un objectif commun : **servir l'organisation.** Les métiers restent **différents** ! DevOps ne veut pas dire qu'une même personne effectue les tâches de Dev et d'Ops. **Développeurs** et **Opérationnels** sont amenés à **collaborer** afin de bénéficier des **compétences** de chacun et améliorer l'**empathie.**





humain

[DOULEUR]



Les **tâches pénibles**
sont effectuées **par**
la Feature Team.
L'automatisation en
découle.





humain

[DOULEUR]

Dans une organisation traditionnelle, le **manque de compréhension** entre les équipes est généralement lié à la distance et au **manque de communication**.

Les **membres d'une Feature Team** sont **co-responsables et solidaires** faces à toutes les tâches. La **douleur** est un facteur clé de l'**amélioration continue**.





humain

[CDS]

Les centres de
services sont
difficiles à concilier
avec l'**engagement
collectif.**





humain

[CDS]

Les Feature Teams sont construites autour de principes qui s'appuient fortement sur la **collaboration** et l'**engagement collectif**. Les centres de services tendent vers la rationalisation et le regroupement de l'informatique par métier, ce qui est contraire à cette notion d'engagement collectif.





humain

[VALIDATION]

L'organisation a un
rôle de validation,
sans être
dogmatique.





humain

[VALIDATION]

Veillez à ce que l'organisation conserve son **rôle de validation** sur les outils et les usages. En particulier sur les **outils qui touchent le patrimoine** (exemple : gestion du code source).

Fournissez aux Feature Teams les **moyens d'étayer** leurs choix.

Ne soyez pas dogmatiques et veillez à **encourager l'expérimentation**.





humain

[TRANSVERSALITÉ]



Les Feature Teams
sont amenées à
communiquer et
à partager leurs
expériences et
compétences.





humain

[TRANSVERSALITÉ]

Ne créez pas de barrières
entre les **Feature Teams**.

Mettez en place une
organisation et l'**agilité**
nécessaire afin que
les Feature Teams
communiquent entre
elles et partagent
leurs compétences et
expériences.

L'organisation de la
transversalité chez **Spotify**
(Tribus, Chapters et
Guildes) est un **exemple**
éloquent.





interopérabilité

[API POUR TOUS]



Des **APIs pour
tous les usages** :
internes, clients et
partenaires, publics.





interopérabilité

[API POUR TOUS]

Ouvrez votre organisation
à des nouveaux usages
et nouveaux clients grâce
aux **APIs publiques**.

Dans le cadre
des **partenariats**
commerciaux, **clients**
comme **fournisseurs**,
les APIs sont le format
d'échange standard.
Les **APIs** ont également
vocation à être utilisées
pour les **usages internes** à
l'organisation.





interopérabilité

[SELF SERVICE]



Utiliser une API
doit être **simple** et
rapide.





interopérabilité

[SELF SERVICE]

L'utilisation des APIs doit être la plus simple possible. Pensez à **l'expérience développeur**. La meilleure solution pour valider l'adéquation avec le besoin est de **tester l'API rapidement** : quelques minutes doivent suffire ! La plateforme doit proposer une **interface graphique** permettant de tester l'API simplement.





interopérabilité

[API MANAGEMENT]

Les utilisations
des APIs doivent
être **contrôlées** et
maîtrisées.





interopérabilité

[API MANAGEMENT]

Mettez en place
une solution d'API
Management pour
gérer **quotas, throttling,
authentification et
logging.**

Collectez des métriques
afin de gérer **monitoring,
filtering et reporting.**





interopérabilité

[EXIGENCES]

Fixez des **exigences**
aux **systèmes et**
services extérieurs
intégrés à la
plateforme.





interopérabilité

[EXIGENCES]

Exigez que les **systèmes externes** répondent aux **mêmes exigences** que les **systèmes internes**.

Les systèmes externes doivent publier des **événements** et permettre le **monitoring** technique. Dans le cas où les données des systèmes extérieurs doivent être intégrées, la **synchronisation** totale doit être **possible**.





interopérabilité

[**MULTI-TENANT**]

L'architecture doit
être pensée **multi-
tenant.**





interopérabilité

[MULTI-TENANT]

Même si la marque blanche n'est pas envisagée à la base, mettez en place une architecture multi-tenant. Votre **application** initiale est le premier **tenant**. Pensez la **multi-instanciation fonctionnelle** du système dès le départ.





interopérabilité

[PARAMÉTRAGE]

Les systèmes
doivent être
**nativement
paramétrables.**





interopérabilité

[PARAMÉTRAGE]

Langues, devises, règles métiers, profils de sécurité doivent être simple à paramétrer. Attention à l'**hyper-généricité**, elle est souvent inutile et **source de coût**. Le **paramétrage** doit être **évolutif** et rapide en fonction des besoins.





interopérabilité

[FEATURE FLIPPING]



Créez des
systèmes souples
et génériques en
utilisant le **feature
flipping**.





interopérabilité

[FEATURE FLIPPING]

Le **feature flipping** consiste à concevoir une application comme un ensemble de **fonctionnalités** qui peuvent être **activées** ou **désactivées** à chaud, en **production**.

Dans une application **multi-tenant**, le feature flipping permet de **personnaliser** les tenants. Le feature flipping **simplifie l'A/B testing**.

