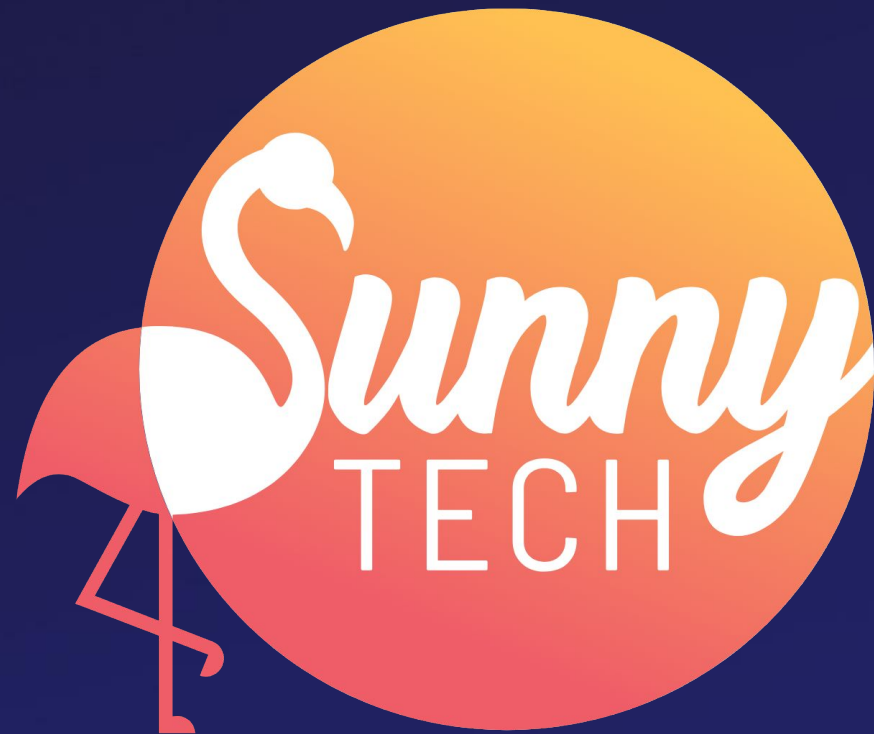


Migrer sa fonctionnalité la plus **critique** sans **régression** ni coupure de **service**

Florian MARIN
Software Engineering Manager



05 Juillet 2024

MERCI
à toutes et tous d'être venu



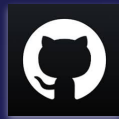
Software Engineering Manager

Teads depuis 2020

Web Platform Team



florian-marin



flomarin88



flomarin



Des questions ? Envie de discuter ?
Mes messages privés sont ouverts



Agenda

Contexte

Fonctionnalité critique qui ne supporte plus notre business

Résultats

Satisfaisants après 2 ans de projet

Démarche

Étapes clés de cette migration

Conclusion

Quelques leçons apprises



Disclaimer



Contexte

Point d'entrée de nos clients

Gérer leurs campagnes de publicités digitales en autonomie

Fonctionnalité **LA plus critique**

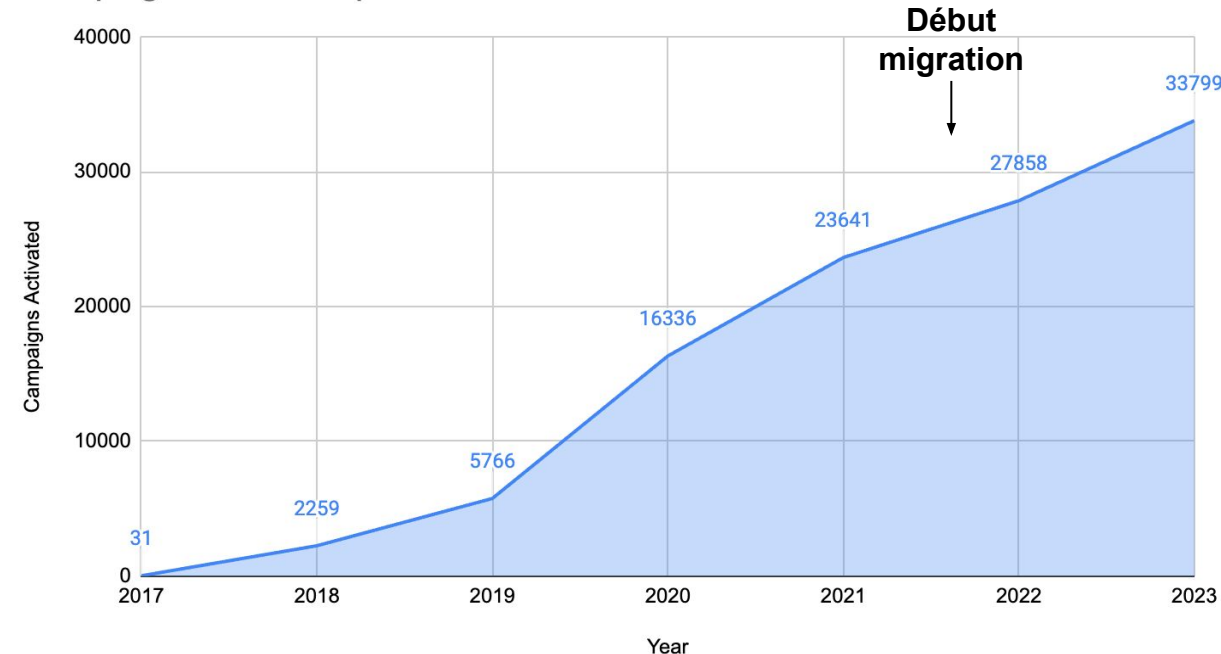
Disponibilité attendue : **24 / 7**

Stratégie de Teads

D'un business model "Agence" vers "SaaS"

Application devient au centre de la stratégie

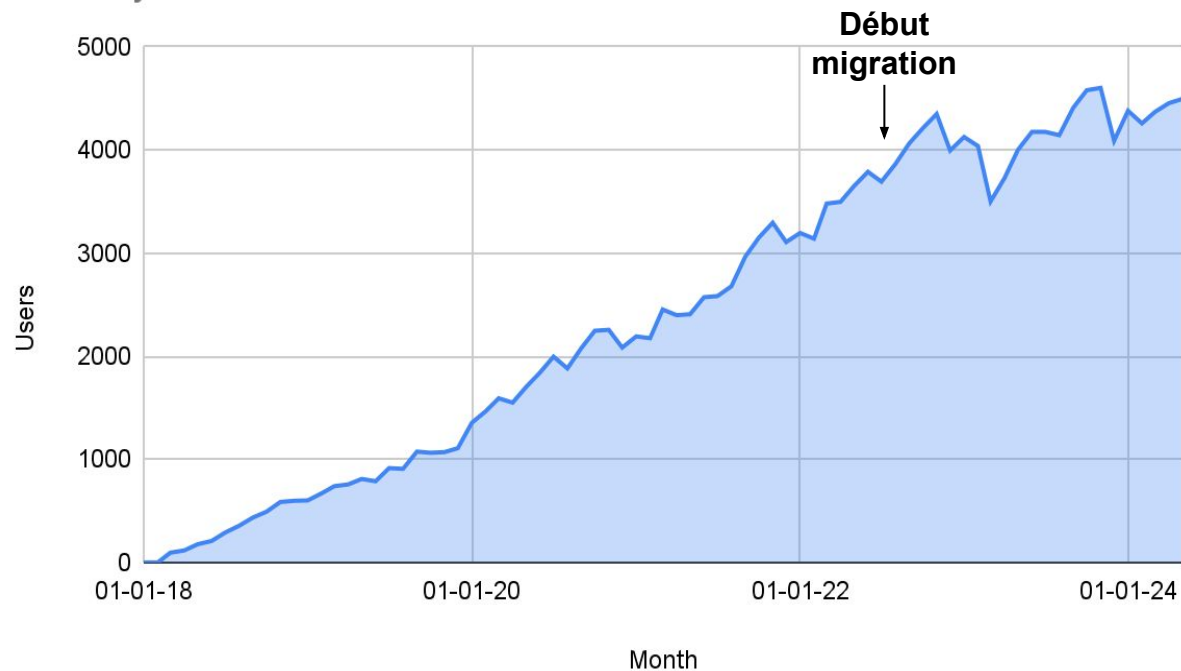
Campaigns Activated per Year



Hausse du trafic qui stresse le système

Et surtout l'organisation, finalement pas prête pour cette nouvelle stratégie

Monthly active users



5%

Taux d'erreur

Des contributeurs de plus en plus nombreux

Dans une base de code en constante évolution

Application initialement développée par une seule équipe, de A à Z.

Puis ownership s'est décentralisé

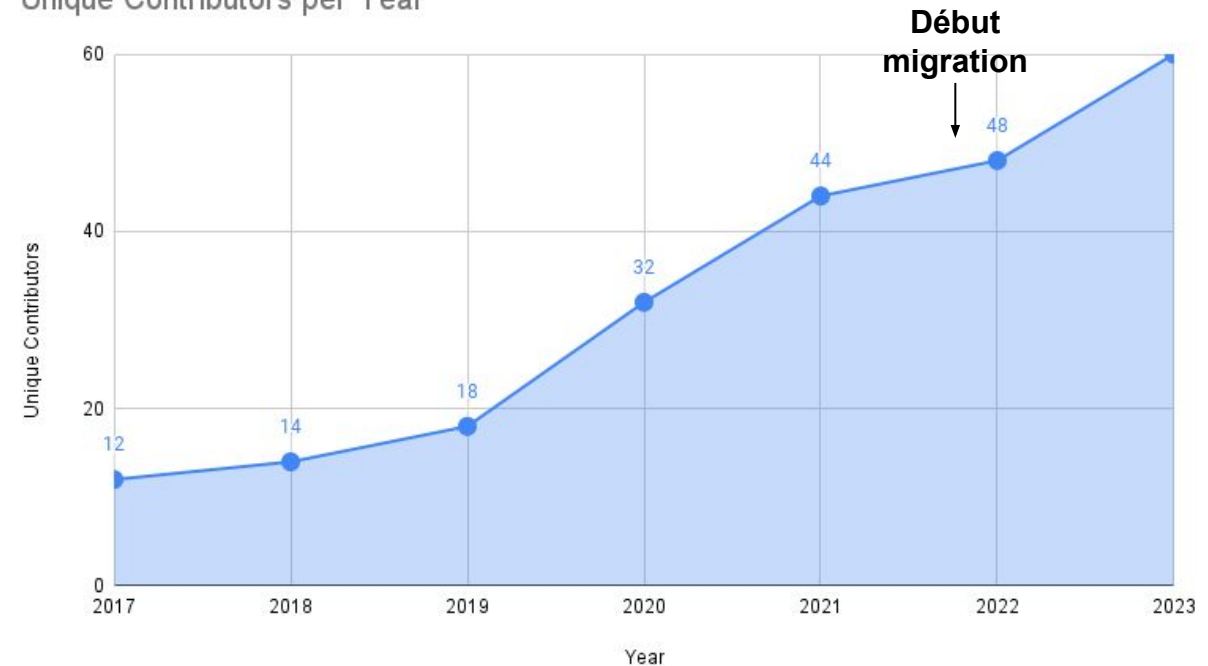
Contributeurs

- 2017: 12 / 1 équipe
- Aujourd'hui: 60 / +10 équipes

Plateforme : Création d'une équipe

- Fin 2021 : 3 personnes (2 Senior, 1 Junior)
- Aujourd'hui : 10 personnes

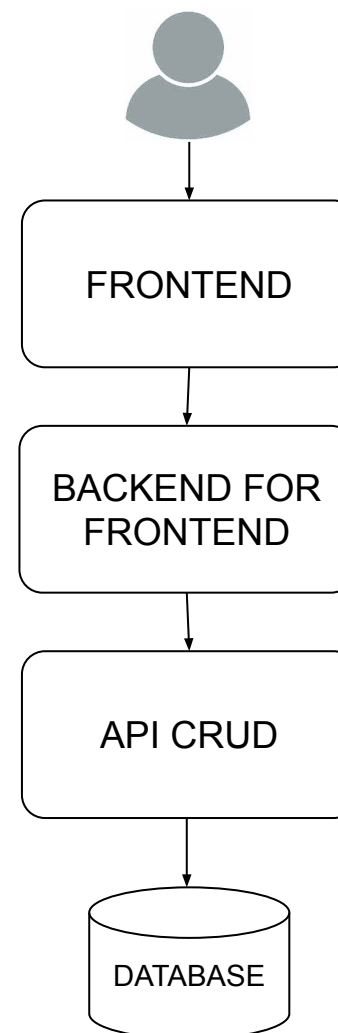
Unique Contributors per Year



Une architecture à faire évoluer

En respectant les contraintes business et techniques de l'entreprise

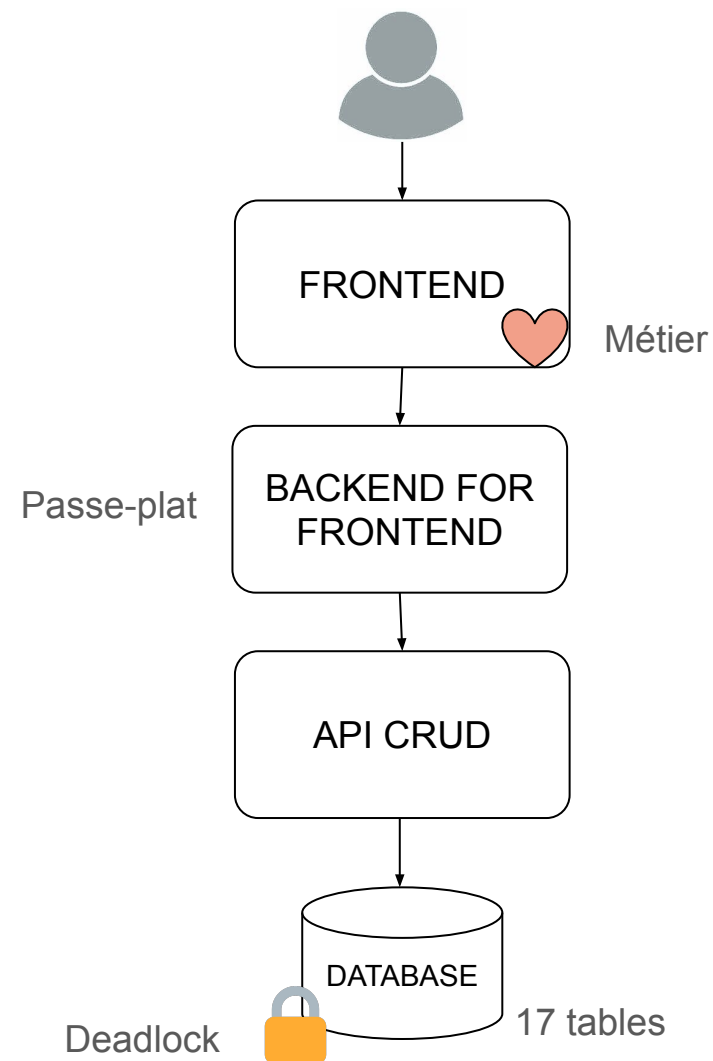
- Continuer à faire évoluer la fonctionnalité
- Conserver le même modèle de base de données
- Eviter les effets Tunnel et Big Bang



Une architecture à faire évoluer

En respectant les contraintes business et techniques de l'entreprise

- Continuer à faire évoluer la fonctionnalité
- Conserver le même modèle de base de données
- Eviter les effets Tunnel et Big Bang

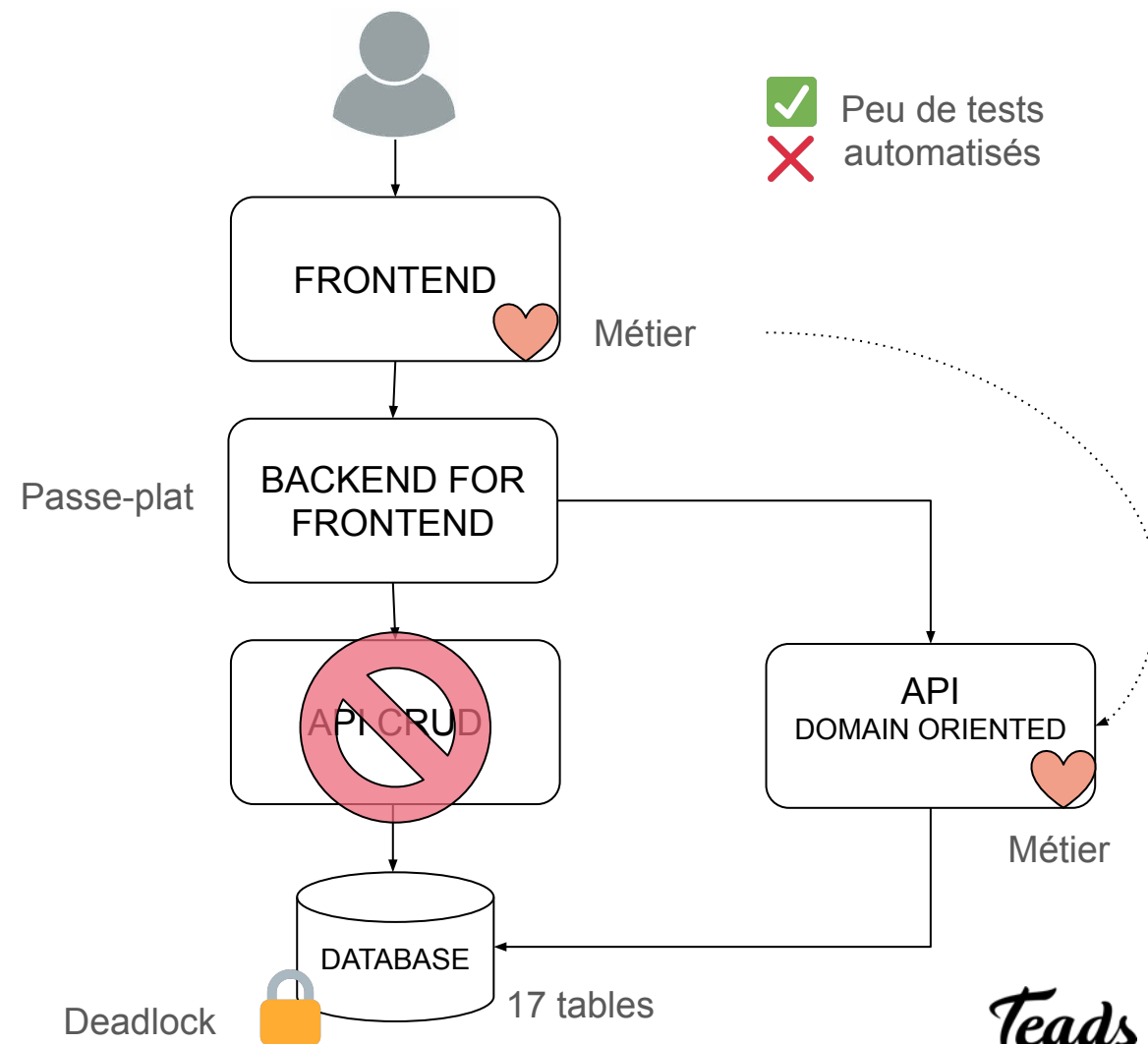


✓ Peu de tests automatisés
✗

Une architecture à faire évoluer

En respectant les contraintes business et techniques de l'entreprise

- Continuer à faire évoluer la fonctionnalité
- Conserver le même modèle de base de données
- Eviter les effets Tunnel et Big Bang





Résultats

Des améliorations significatives

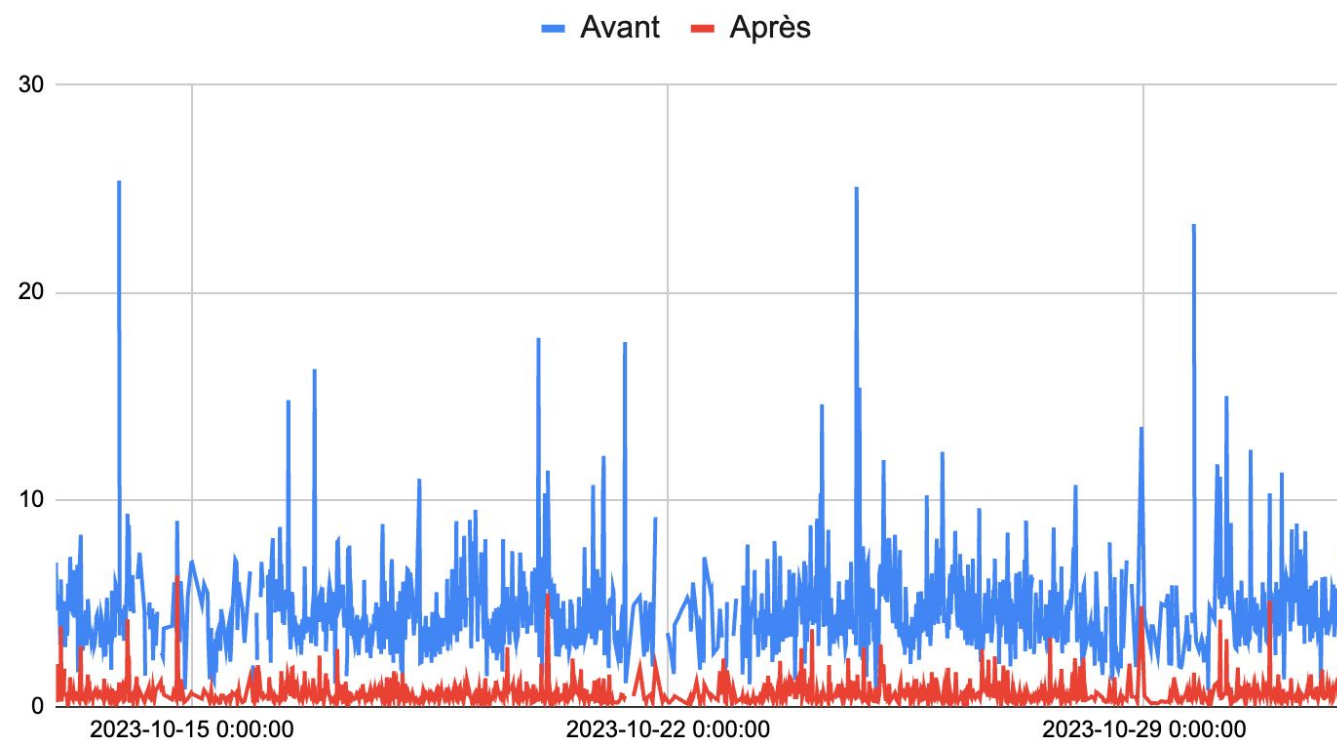
Et la migration n'est pas encore totalement terminée

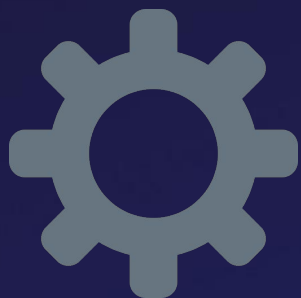


0,15%

Taux d'erreur

Comparaison des latences (Avant / Après)





Démarche

Rédiger une vision de l'architecture finale

Avant même d'écrire une première ligne de code

Design Document @Teads

- Contexte / Problème(s) à résoudre
- Solutions étudiées et sélectionnée avec argumentaires
- Impacts et dépendances
- Stratégie de tests
- Stratégie de déploiement
- Macro Planning
- Coûts



Validation inter équipes

Public => Engagement fort

Monitorer lors toutes les phases

"If you can't measure it, you can't improve it"

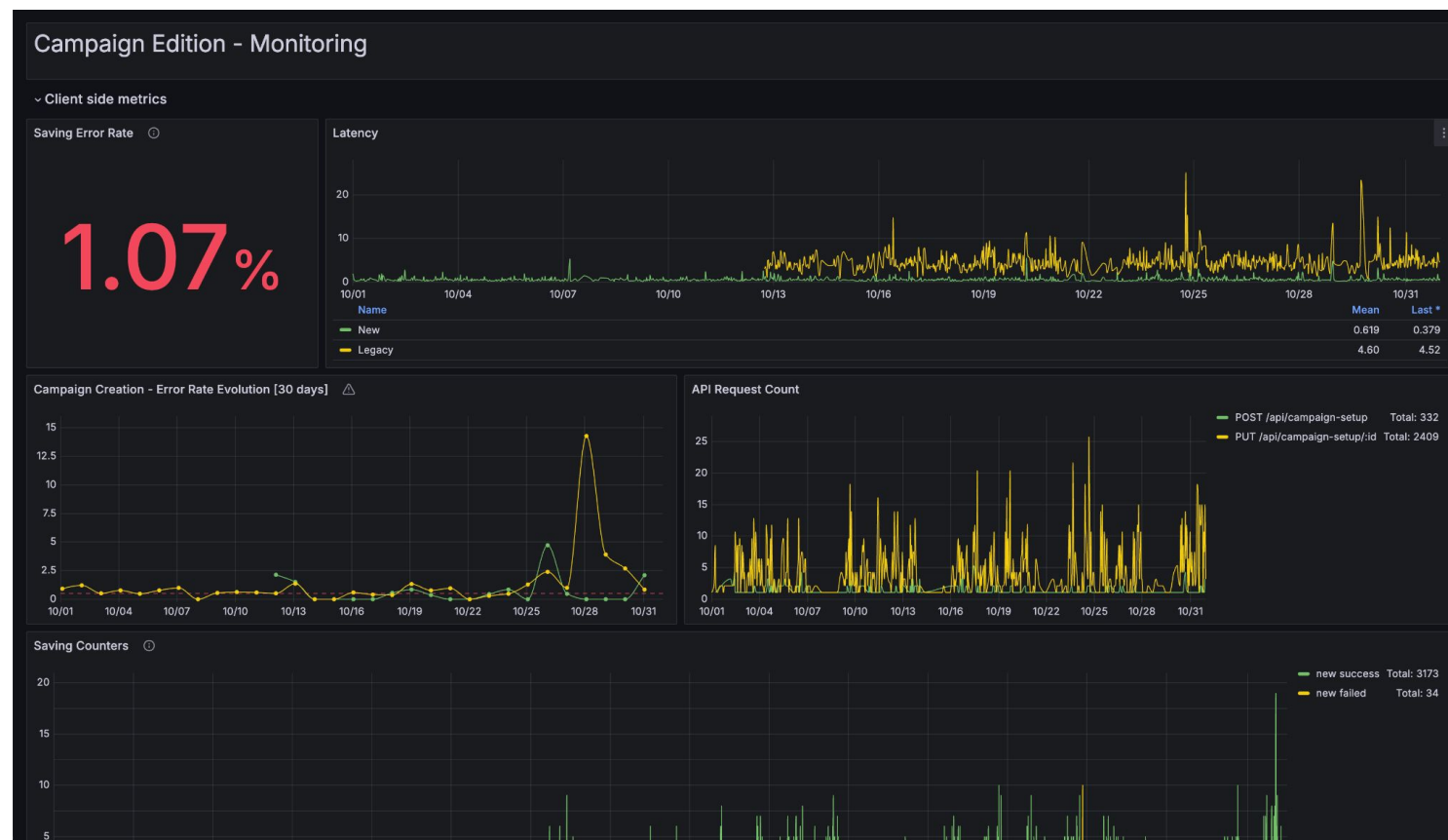
Dashboard spécifique de supervision

Metriques

- Taux d'erreur
- Latence
- Nombre d'appels API legacy

Alerte contextualisée à chaque erreur

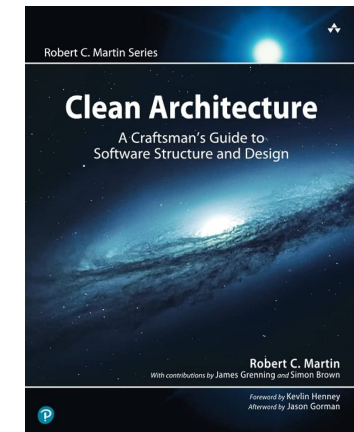
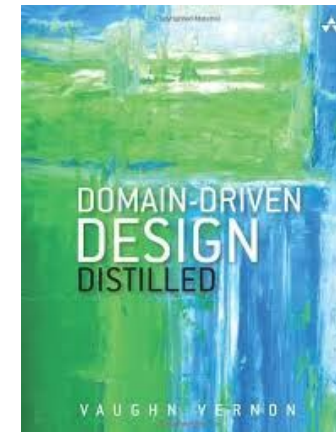
Outillez vous



Définir un nouveau “domaine”

Supporté par une Clean Architecture

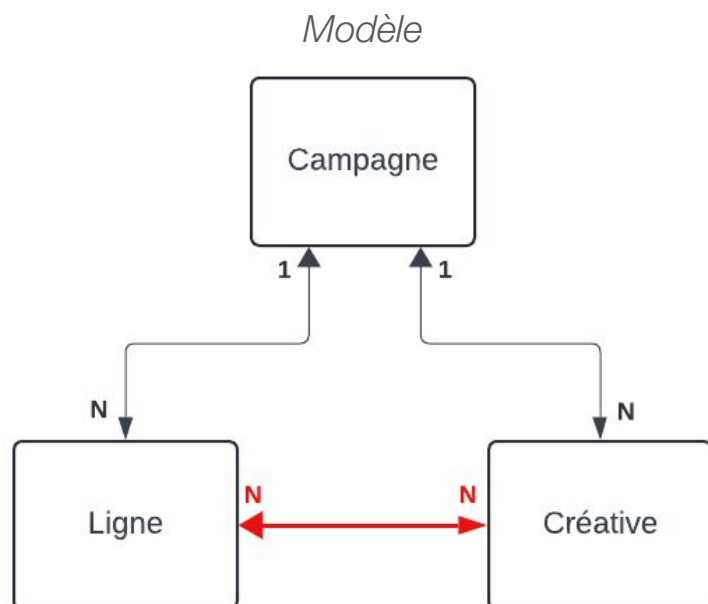
- Mettre le métier au coeur du code et de l'architecture
- Gérer entièrement le cycle de vie d'une configuration de campagne
- S'abstraire des couches d'infrastructure (DB) et de communication (GRPC)
- Définir un contrat API orienté “use case”



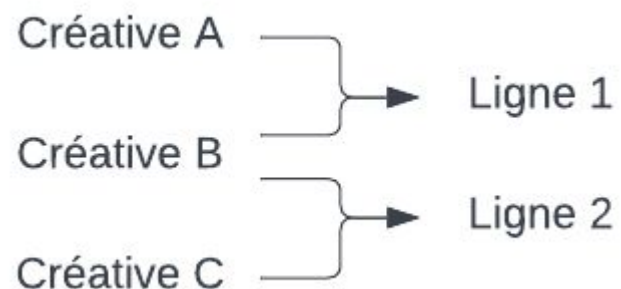
Résoudre les problèmes un par un

Mode itératif qui permet de garder la motivation et de souffler de temps en temps

Exemple



Exemple simplifié





4 calls API en //

Chaque call fait une écriture dans la table "Ligne"

Deadlock

Résolution

-  Séquentiel: Impact les performances et si un call échoue ?
-  Batch : Endpoint GRPC *AssignCreativeToLineItems*

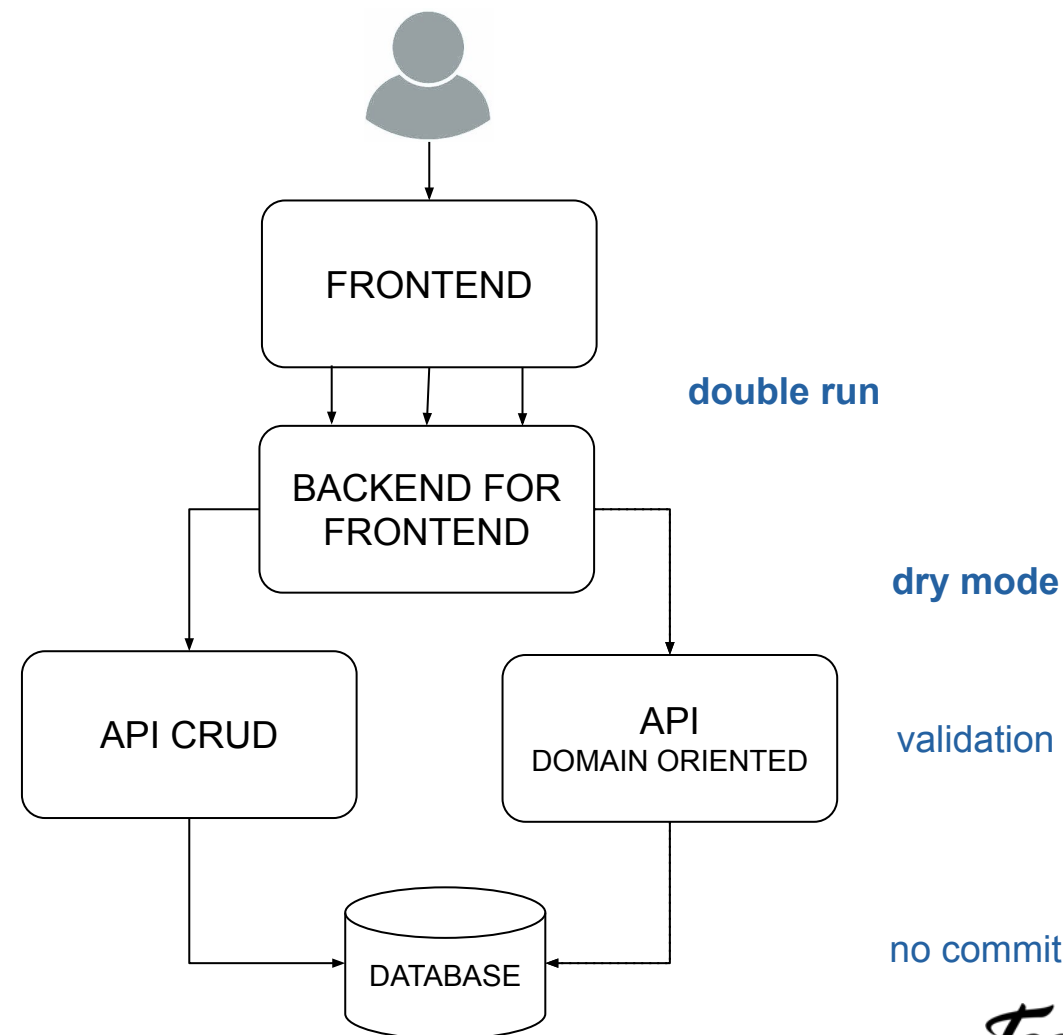
Expérimenter l'architecture cible

Double Run en "dry" mode

Aucun impact pour l'utilisateur

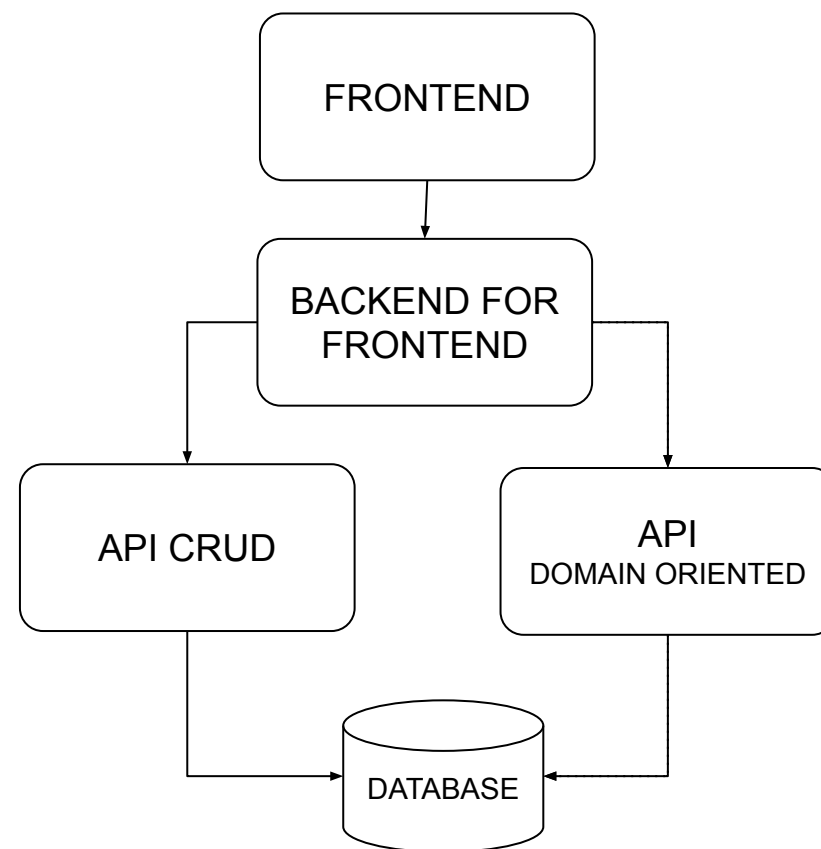
Prendre le temps d'explorer les uses cases

Monitorer les résultats



Adopter une stratégie de tests complète

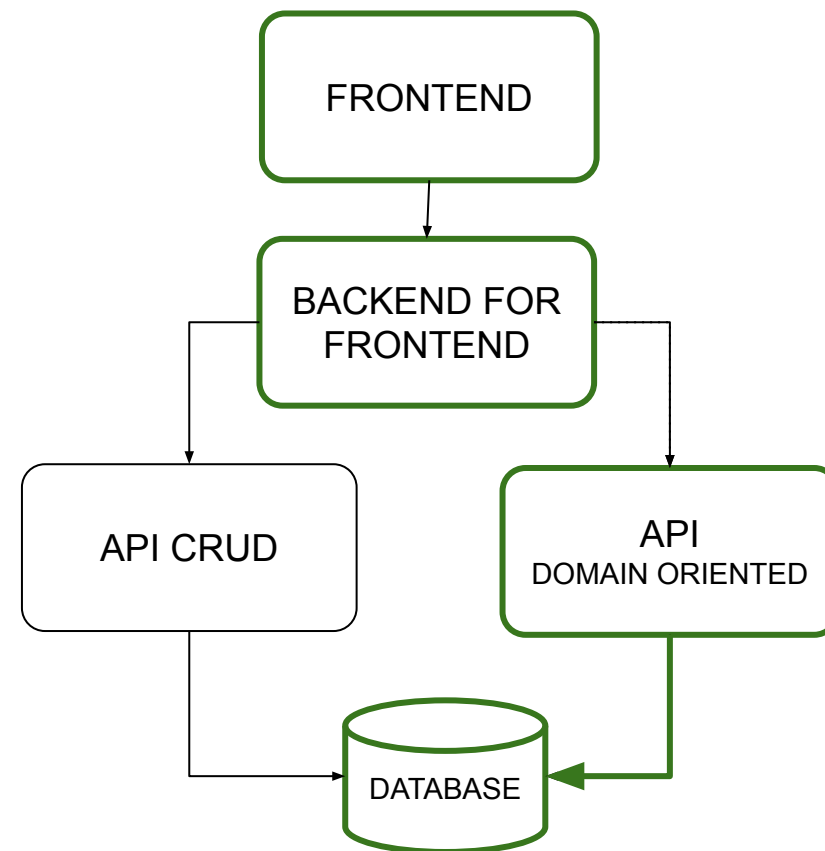
Récupérer de la connaissance métier et éviter les régressions



Adopter une stratégie de tests complète

Récupérer de la connaissance métier et éviter les régressions

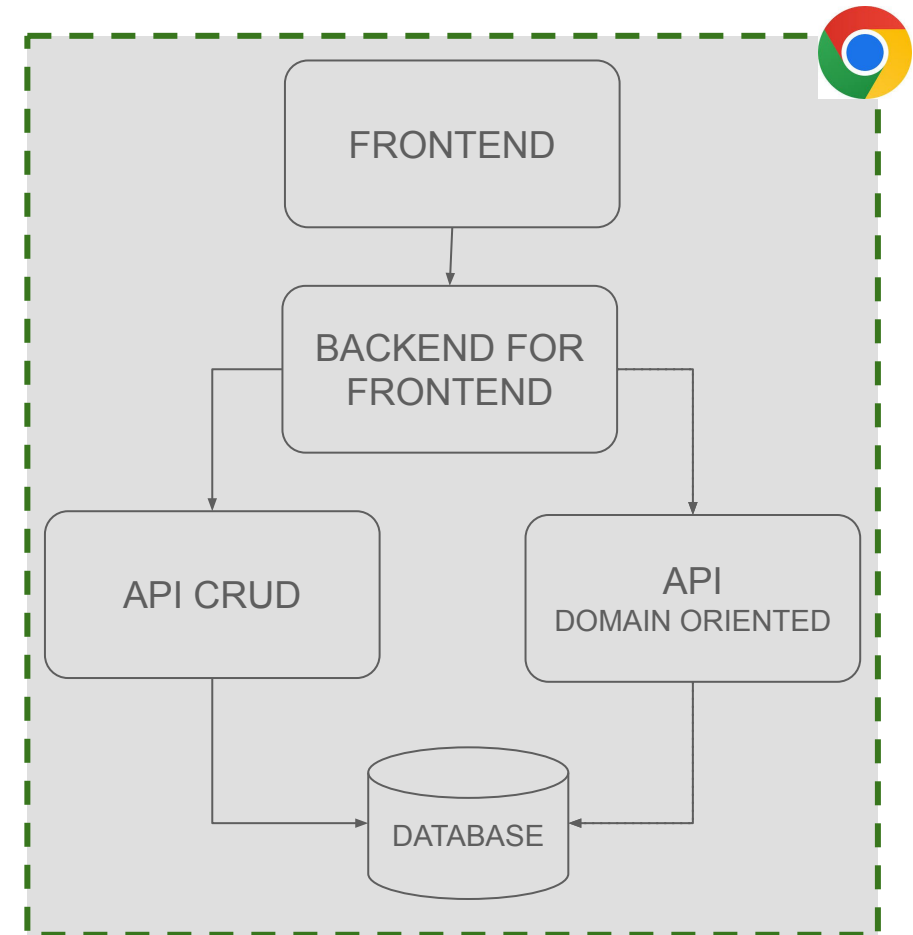
- Unitaires / Intégrations évidemment mais pas suffisants



Adopter une stratégie de tests complète

Récupérer de la connaissance métier et éviter les régressions

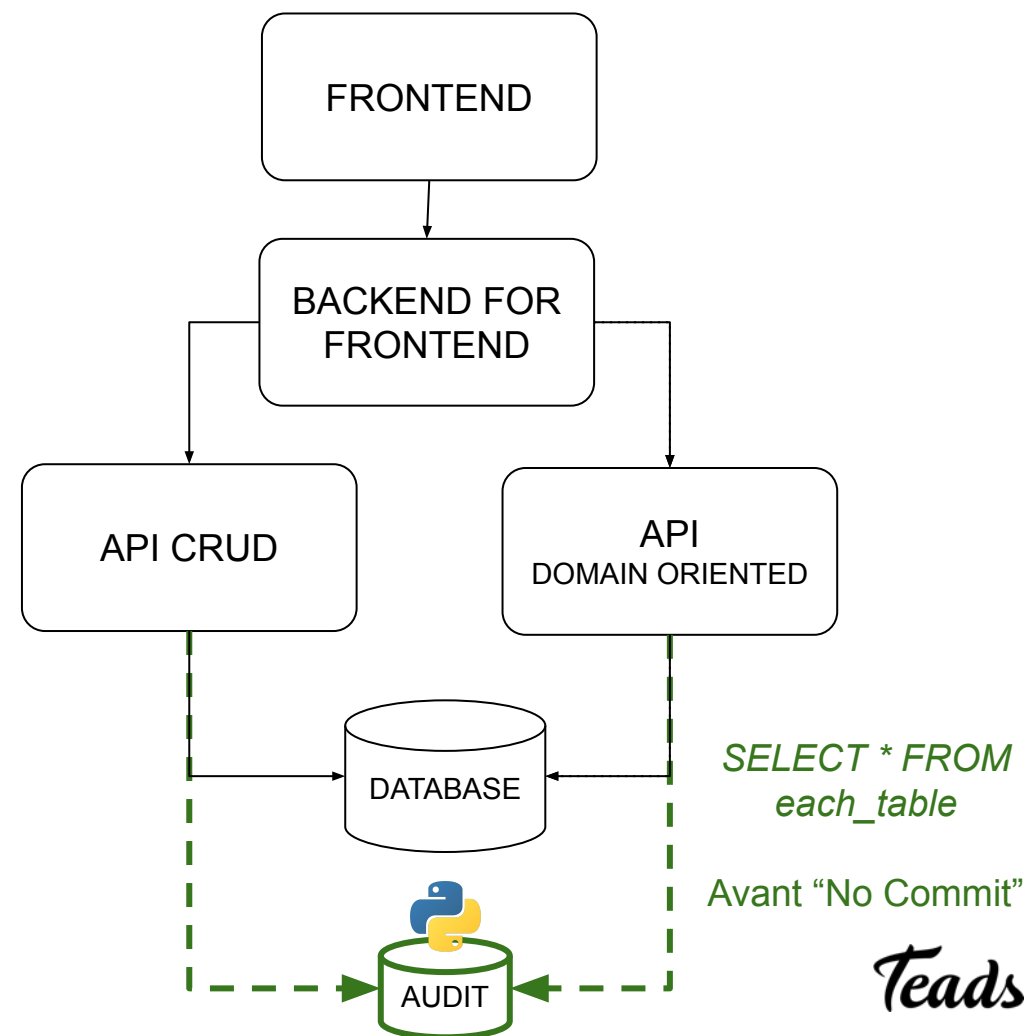
- Unitaires / Intégrations évidemment mais pas suffisants
- End to End dans la CI
 - Uniquement quelques scénario critiques



Adopter une stratégie de tests complète

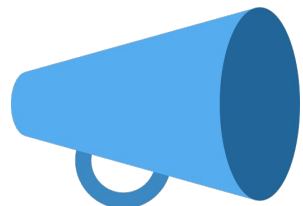
Récupérer de la connaissance métier et éviter les régressions

- Unitaires / Intégrations évidemment mais pas suffisants
- End to End dans la CI
 - Uniquement quelques scénario critiques
- **Production** directement 🤖
 - Comparaison quotidienne des *audits*



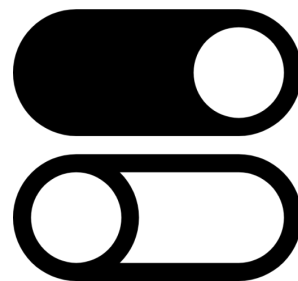
Déployer progressivement

Un plan préparé, communiqué et à suivre à la lettre



Communication

Interne et Clients



Feature Flag

Ouverture hebdomadaire



Suivi Incidents

Tout noter

Conclusion

Quelques leçons apprises

- Mesurer dès le début et en continu
- Définir la vision technique finale et se faire challenger par le reste des équipes
- Automatiser et diversifier les tests
- Déployer progressivement, en interne puis externe
- (Sur)Communiquer
- Anticiper en allant chercher les feedbacks et les informations sur les fonctionnalités à venir
- Soyez patients voire pessimistes parfois

MERCI
pour votre attention et vos
feedbacks

Des questions ?

