La Clean Architecture

Une solution pour augmenter la maintenabilité de ses systèmes

Mehdi BOISSAT-BRON
26/05/2023

leboncoin

Qui suis-je ?

Mehdi BOISSAT-BRON

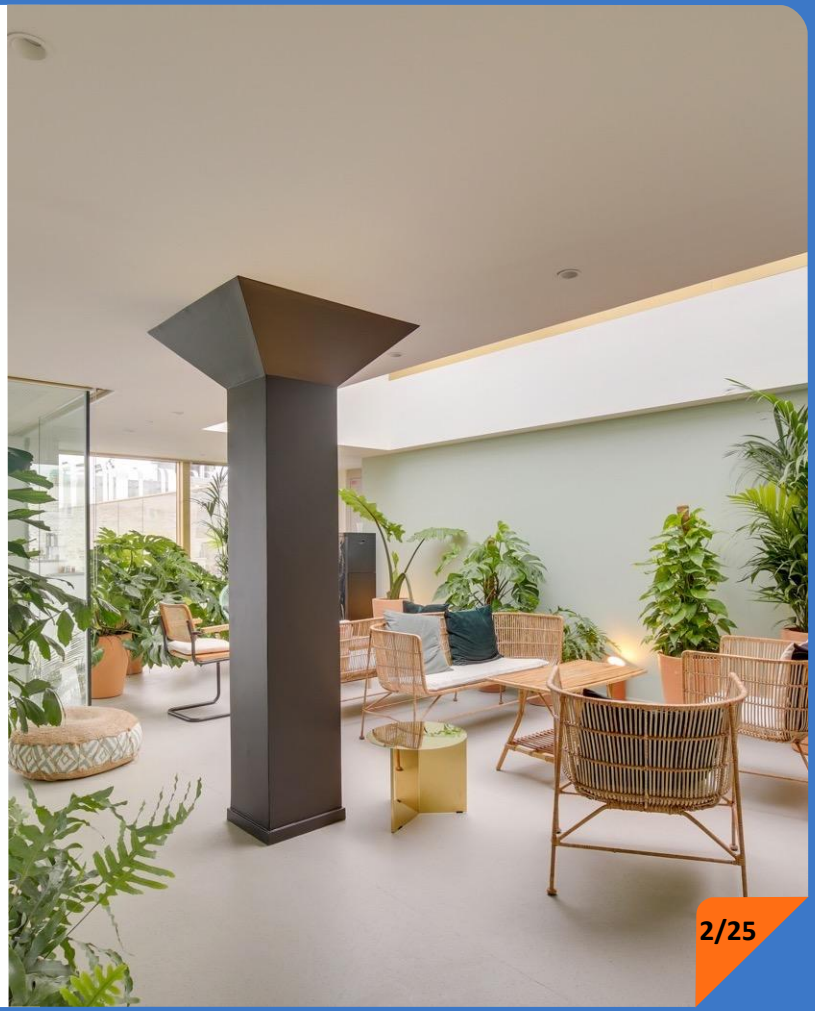
Développeur Backend
Feature Team Import

mbbo.org

linkedin.com/in/mbbo128

twitter.com/mbbo128

github.com/mbbo128



leboncoin, leader de l'économie collaborative

Le 6ème site visité de France

39 millions d'annonces en ligne

32 millions d'utilisateurs uniques par mois

800 000 nouvelles annonces par jour

10 milliards de pages vues par mois

C'est aussi un groupe

~ 1500 Collaborateurs

~ **600** Personnes Product & Tech

~ **60** Feature teams

leboncoin GROUPE · VIDE-DRESSING · Locasun · PAYCAR · Agriaffaires · ravendre alouer · Groupe Argus · PILGO · MachineryZone

SOMMAIRE

01

INTRODUCTION

INTRODUCTION

Contexte Leboncoin

Features Teams pluridisciplinaires
- Import, Contact, Search ...

Backend Monorepo Golang
- Codes communs pour le monitoring, log ...
- Codes spécifiques pour chaque équipe

Microservices
- De quelques-uns jusqu'à 20+ par équipe

➔ **L'organisation du code est à définir par les équipes en fonction de leurs besoins**

INTRODUCTION

Contexte Feature Team Import

- **Assurer la maintenabilité** avec une croissance du code
- **Faciliter le partage** de fonctionnalités entre nos microservices
- **Anticiper des chantiers techniques**
  Découpage base de données, http vs gRPC, redis / elasticsearch

➔ **Compte tenu de ces besoins nous avons opté pour utiliser la clean architecture sur les nouveaux microservices**

02

PROBLÉMATIQUE

PROBLÉMATIQUE

Est-ce qu'adopter cette architecture a satisfait ces besoins ?

03

RAPPELS

RAPPEL

Sur la clean architecture

Robert C. Martin (auteur de clean code)

- Article de blog, 2012
- Livre écrit en 2017

Architectures similaires :

- Hexagonal architecture
- Onion Architecture
- Entity-control-boundary (ECB)

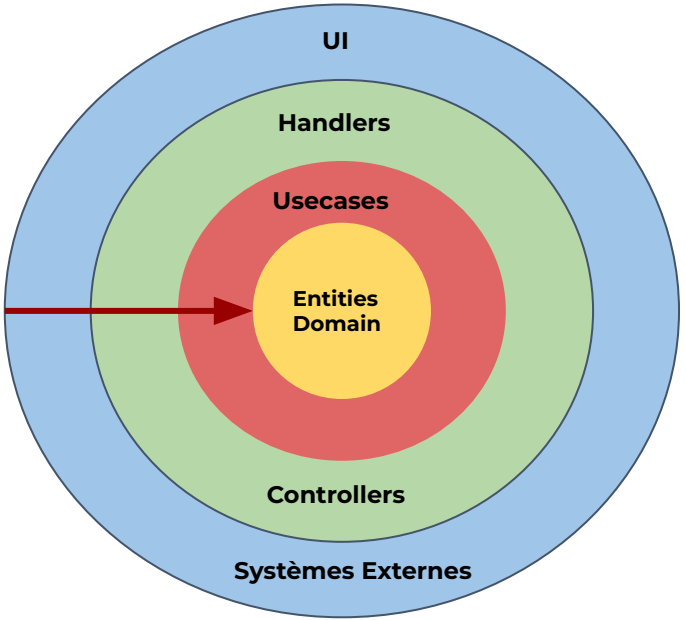
RAPPEL

C’est quoi ?

Une représentation : sous la forme de cercles concentriques

Une règle :
Dependency Rule

Langage agnostique : iOS, Android, Backend, Frontend



04

MISE EN PLACE

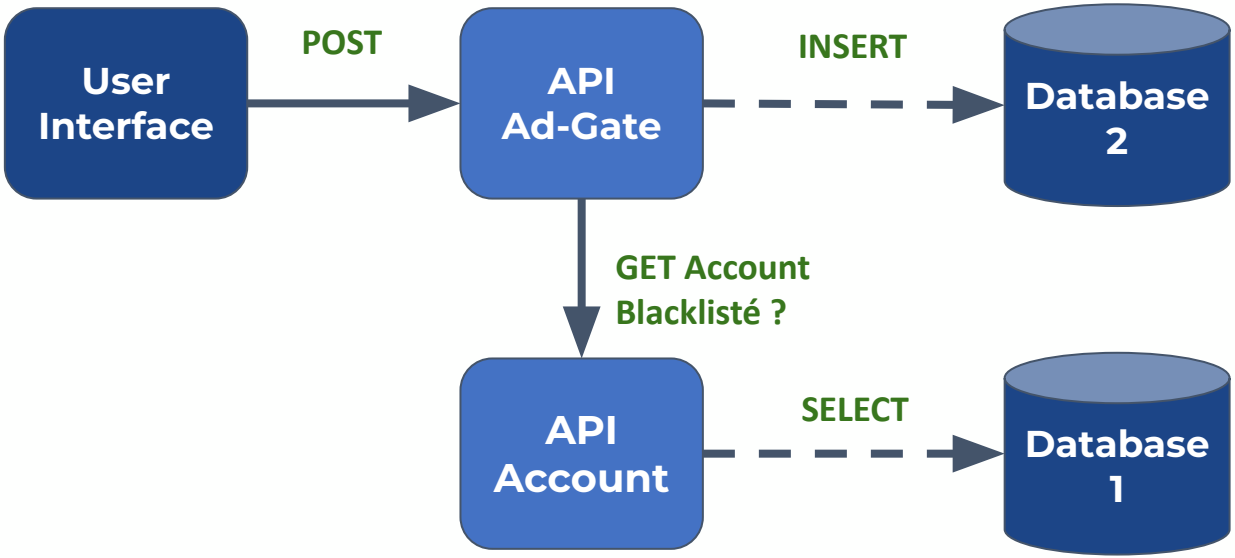
MISE EN PLACE

Points de contrôle

- **Définition d'une arborescence** de fichiers qui match la clean architecture
  Nous avons fait des réunions techniques pour se mettre d'accord sur l'arboresence
- **Le domain est accessible** partout dans l'application
- **Centralisation du code métier** dans le usecase
- **Respect de la dependency rule** via l'utilisation d'interface

MISE EN PLACE

Schéma cas d'étude



MISE EN PLACE

Package Handler

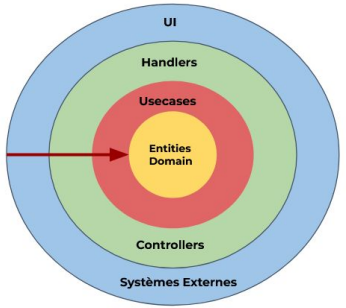
```
CreateAd() {
     var classifiedAd domain.ClassifiedAd

     // User Input Validation
     err = c.Validate(classifiedAd)
     if err != null { // return Bad Request }

     // Check the account is not blacklisted
     isBlacklisted = accountAPI.CheckAccountBlacklisted(classifiedAd.AccountID)
     if isBlacklisted  == true { // return Not Found }

     // Store Ad in database
     err = repository.StoreAd(classifiedAd);
     if err != null { // return Internal Server Error }

    return JSON(http.StatusOK, classifiedAd)
}
```

MISE EN PLACE

Package Handler

```
CreateAd() {
    var classifiedAd domain.ClassifiedAd

    // User Input Validation
    err = c.Validate(classifiedAd)
    if err != null { // return Bad Request }

    // Check the account is not blacklisted
    isBlacklisted = accountAPI.CheckAccountBlacklisted(classifiedAd.AccountID)
    if isBlacklisted  == true { // return Not Found }

    // Store Ad in database
    err = repository.StoreAd(classifiedAd);
    if err != null { // return Internal Server Error }

    return JSON(http.StatusOK, classifiedAd)
}
```

❌ Le code métier doit être dans le usecase (rouge)

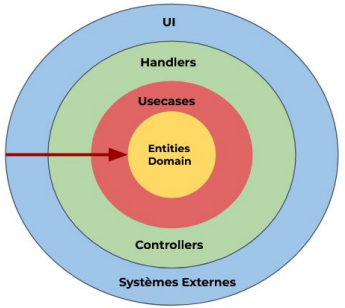


MISE EN PLACE

Package Handler

```
CreateAd() {
    var classifiedAd domain.ClassifiedAd

    // User Input Validation
    err = c.Validate(classifiedAd)
    if err != null { // return Bad Request }

     // Check account and store the Ad in database
    err = usecaseStore.CheckAccountStoreAd(classifiedAd)

    if err == domain.ErrBlacklisted {
      // return Not Found
    }

    // return Internal Server Error

    return JSON(http.StatusOK, classifiedAd)
}
```

✅ Le code métier est bien dans un usecase (rouge)

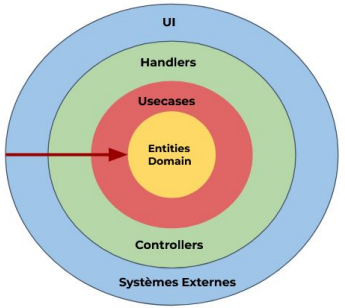


MISE EN PLACE

Package Handler

```
CreateAd() {
    var classifiedAd domain.ClassifiedAd

    // User Input Validation
    err = c.Validate(classifiedAd)
    if err != null { // return Bad Request }

     // Check account and store the Ad in database
    err = usecaseStore.CheckAccountStoreAd(classifiedAd)

    if err == domain.ErrBlacklisted {
      // return Not Found
    }

    // return Internal Server Error

    return JSON(http.StatusOK, classifiedAd)
}
```

✅ Le code métier est bien dans un usecase (rouge)



MISE EN PLACE

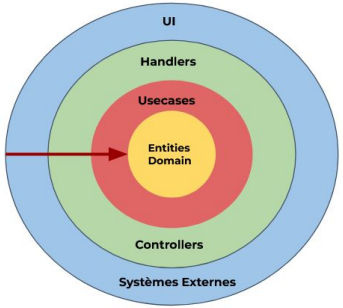
Package Usecase

```
CheckAccountStoreAd(classifiedAd domain.ClassifiedAd) error {

    // Check the account is not blacklisted
    isBlacklisted = accountAPI.CheckAccountBlacklisted(classifiedAd.AccountID)
    if isBlacklisted  == true { // return Not Found }

    // Store Ad in database
    err = repository.StoreAd(classifiedAd);
    if err != null { // return Internal Server Error }

  return null
}
```

MISE EN PLACE

Package Usecase

```
CheckAccountStoreAd(classifiedAd domain.ClassifiedAd) error {

    // Check the account is not blacklisted
    isBlacklisted = accountAPI.CheckAccountBlacklisted(classifiedAd.AccountID)
    if isBlacklisted  == true { // return Not Found }

    // Store Ad in database
    err = repository.StoreAd(classifiedAd);
    if err != null { // return Internal Server Error }

  return null
}
```



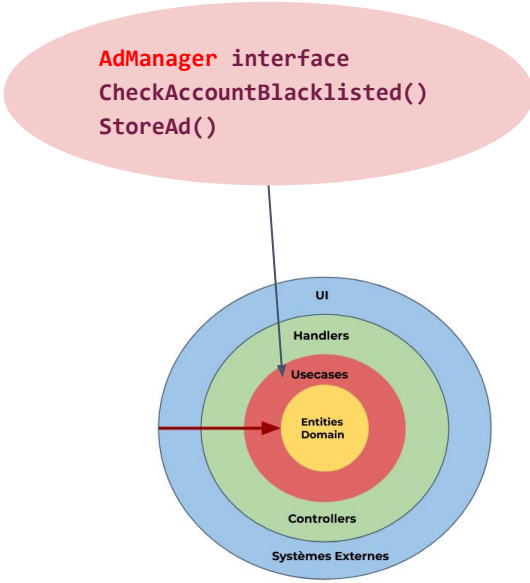
❌ **Non respect de la dependency rule**

Le usecase (rouge) utilise une API externe et un repository qui sont dans la couche verte

MISE EN PLACE

Package Usecase

```
CheckAccountStoreAd(classifiedAd domain.ClassifiedAd) error {

    // Check the account is not blacklisted
    isBlacklisted = AdManager.CheckAccountBlacklisted(classifiedAd.AccountID)
    if isBlacklisted  == true { // return Not Found }

    // Store Ad in database
    err = AdManager.StoreAd(classifiedAd);
    if err != null { // return Internal Server Error }

  return null
}
```



✅ Avec l'interface présente dans la couche usecase (rouge), on supprime la dépendance entre le usecase et le handler

05

BÉNÉFICES et INCONVÉNIENTS

BÉNÉFICES et INCONVÉNIENTS

✅ Quels sont les avantages ?

Augmentation de la maintenabilité du code, couverture de tests plus élevée

Facilite le partage de fonctionnalités via les usecases

Remplacement des composants facilité

Permet de travailler à plusieurs sur une même user story

❌ Quels sont les inconvénients ?

Pas applicable à tous les cas de figure (code externe)

Difficile à mettre en place sur du code legacy

➔ **Nous sommes très satisfait de l'adoption de la clean architecture**

Merci

Avez-vous des questions ?