



D'une architecture N-Tiers à une architecture Clean Hexagonale

Hands-On Labs

- Céline Gilet -

18 avril 2019

Faisons connaissance

Céline Gilet



Céline Gilet



@celinegilet

- Tribu Software Craftsmanship à OCTO Technology
- Développeuse depuis + de 10 ans
- Conseil & accompagnement sur les pratiques de qualité logicielle
- Formation (Test Driven Development, Clean Code, Legacy Code...)

Objectifs du Hands-On Labs

- Les limites d'un design applicatif basé sur une organisation en couches techniques
- Les principes de l'architecture hexagonale et de la clean architecture pour simplifier la maintenabilité et l'évolutivité
- Une projection sur une base de code
 - Isoler le coeur métier de tout le reste (outils, frameworks, briques d'infrastructure)
 - S'abstraire des accès à une base de données / un système de fichiers / un serveur de mail
- Les apports sur le quotidien de développeur

Table des matières

01

Happy Town -
Installation &
découverte du sujet

02

Maintenabilité et
évolutivité du code

03

Vers une architecture
clean hexagonale

04

Clean Architecture -
Step by Step

05

Hexagonale
Architecture

01

Happy Town
Installation & découverte du sujet

Prise en main du sujet

A la découverte de notre fil rouge : "Happy Town"

Installation du projet (<https://github.com/celinegilet/happy-town>)

Pour accueillir dignement ses nouveaux habitants, le conseil municipal de *HappyTown* a décidé d'offrir un cadeau à tous ses habitants qui soufflent leur première bougie dans la commune.

Le rôle de notre application est donc :

- De sélectionner les habitants éligibles à l'obtention d'un cadeau (ils ont emménagés depuis plus de 1 an)
- Pour chacun des habitants éligibles :
 - Trouver le cadeau approprié en fonction de son âge : il y a des cadeaux différents par tranche d'âge
 - Envoyer un mail annonçant l'attribution du cadeau
- Envoyer un mail récapitulatif au service cadeau de la mairie avec tous les cadeaux attribués de la journée

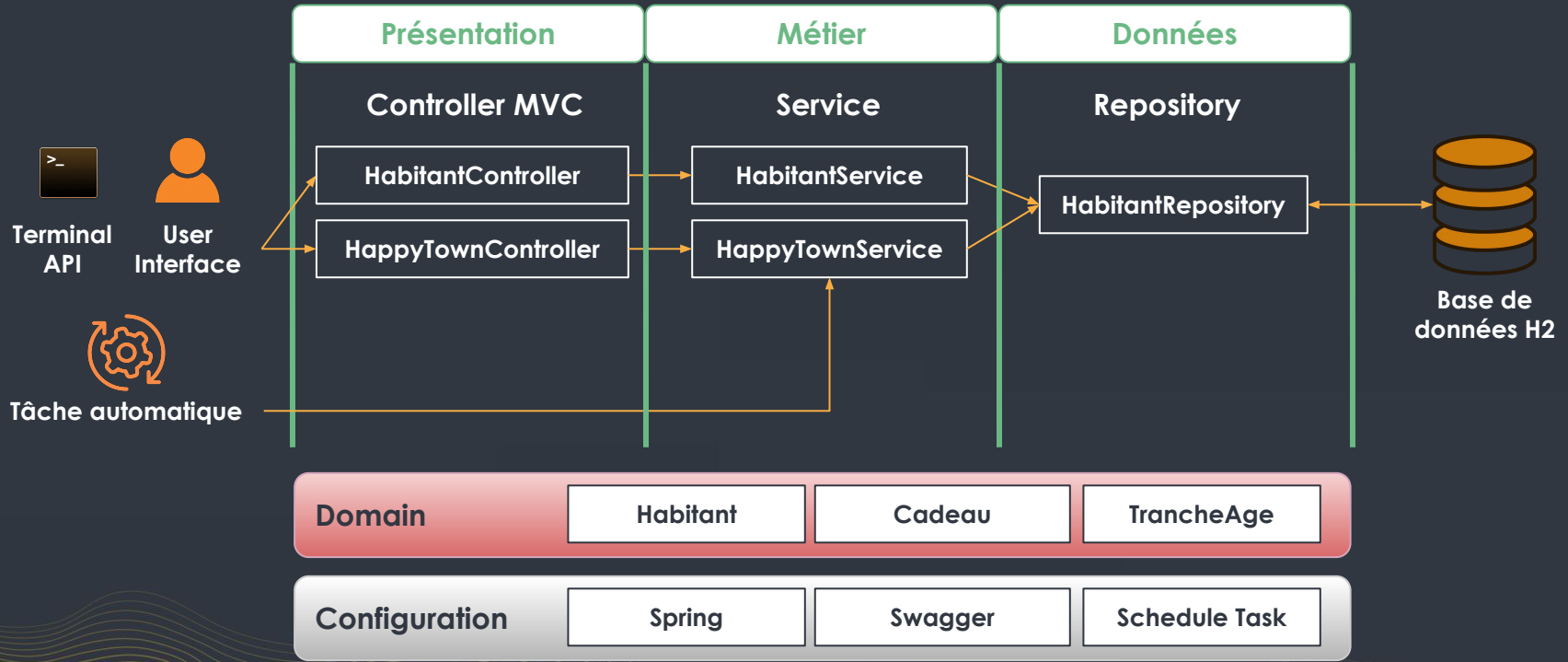
Prise en main du sujet

Une représentation graphique de "Happy Town"



Prise en main du sujet

Schéma de l'architecture actuelle : 3-Tiers



02

Maintenabilité et évolutivité du code

Maintenabilité et évolutivité du code

Revue collective de code



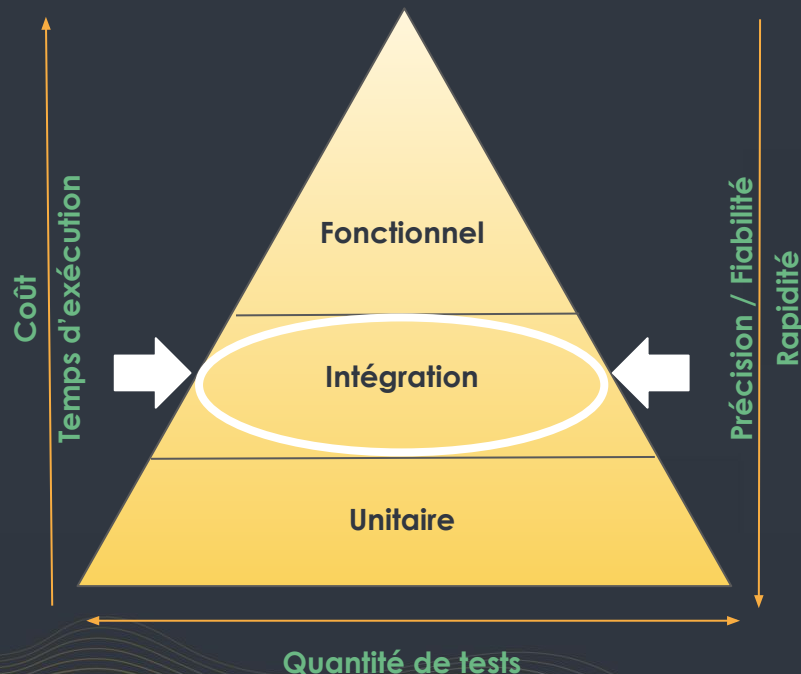
Maintenabilité et évolutivité du code

Limites actuelles

- Du code centré autour des frameworks
- Un découpage et une architecture par responsabilité technique (controller / service / repository)
- Le modèle métier est à la fois le modèle de stockage et le modèle de présentation
- Une perte de la logique métier et des services de l'application
- Un fort couplage et une adhérence aux composants d'infrastructure (serveur de mail, système de fichiers)
- Des difficultés à écrire des tests rapidement qui représentent le métier
- Des tests écrits à posteriori avec une stratégie de couverture de code (absence de TDD / design émergent)
- Évolutions de + en + difficile (en durée et complexité)

Maintenabilité et évolutivité du code

Une pyramide de tests sans base solide



- Une **pyramide** de **tests** dont la base démarre au niveau “Intégration”
- Des **tests** avec un caractère **aléatoire**
- Une nécessité de **démarrer** un **serveur** de **mail** pour tester les **règles métiers**
- Une **stratégie** de **tests** basée sur le fonctionnement de **frameworks**

03

Vers une architecture clean hexagonale

Vers une architecture clean hexagonale

Dans quel but ?

- La valeur d'une application réside dans ses cas d'utilisation et ses services métiers
- Isoler et protéger cette valeur des changements et évolutions techniques
- Le domaine métier d'une application n'évolue pas au même rythme que les éléments connexes (frameworks, base de données, infra...)
- Une prise en compte des aspects techniques à posteriori
- Pas de dispersion de la logique métier dans plusieurs couches
- Des tests ciblés sur une problématique précise : rapidité, fiabilité et robustesse
- Un découpage par responsabilité pour favoriser les évolutions et accélérer le cycle des déploiements
- Des mises à jour de dépendances techniques plus régulières

Vers une architecture clean hexagonale

Clean Architecture / Hexagonale Architecture

Clean Architecture

- Uncle Bob Martin -

L'élément clé d'une application ne réside pas dans sa base de données et les frameworks utilisés. Les use-cases d'une application sont l'élément central

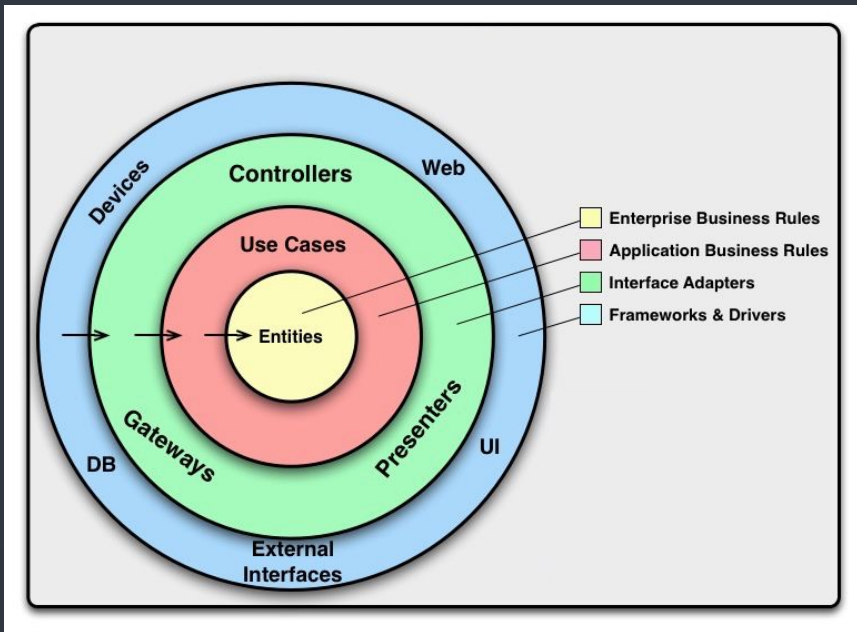
Hexagonale Architecture

- Alistair Cockburn -

Permettre à une application d'être pilotée aussi bien par des utilisateurs que par des programmes, des tests automatisés ou des scripts batchs, et d'être développée et testée en isolation de ses éventuels systèmes d'exécution et bases de données

Vers une architecture clean hexagonale

Clean Architecture



Source :

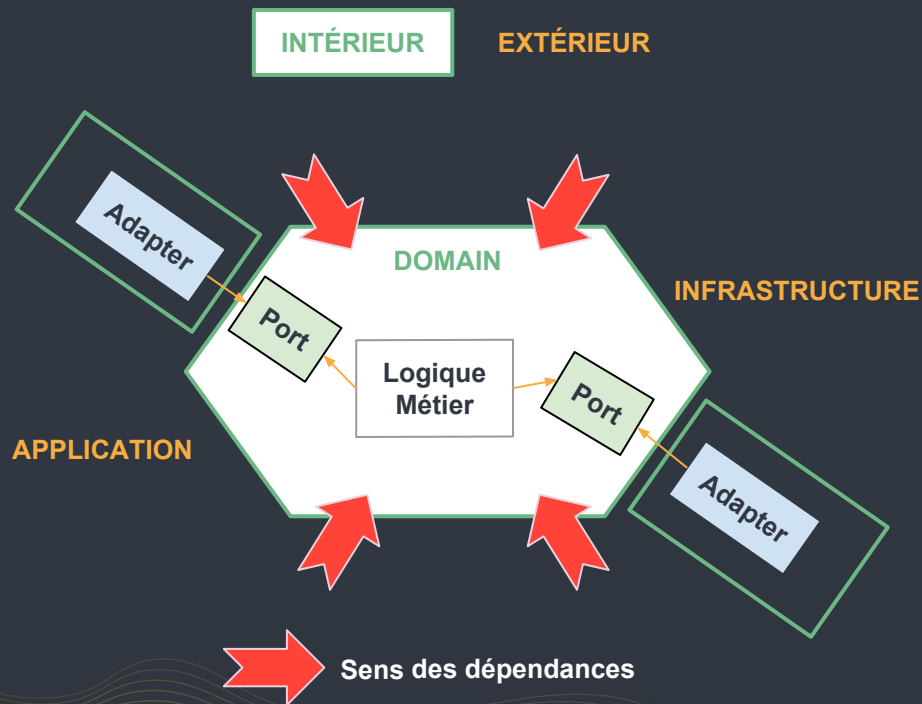
<http://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html>

Principes

- Un centre contenant la **logique métier** sans frameworks ni annotations
 - **Entities** - Objets du domaine
 - **Use Cases** - Services applicatifs
- Des **points d'entrée (Entrypoints)** pour déclencher les **use cases** : API Rest, interface graphique, jobs
- Des **fournisseurs de données (DataProviders)** pour **récupérer** et **stocker** les **données** : BDD, périphériques réseau, fichiers, systèmes externes
- Les **éléments de configuration (Configuration)**

Vers une architecture clean hexagonale

Hexagonale Architecture



Principes

- Intérieur vs Extérieur
- Découpage en 3 zones distinctes
 - Application - les moyens d'interactions pour piloter / déclencher le métier
 - Domain - la logique métier
 - Infrastructure - les besoins et dépendances nécessaires au métier (BDD, Systèmes extérieurs, File System)
- Sens des dépendances uniquement vers l'intérieur : le Domain
- Isolation des frontières par des Ports et Adapters (Interfaces)

04

Clean architecture - Step by Step

Clean Architecture - Step by Step

Cartographie et projection

Déclencheurs

- Un terminal console (CURL endpoint)
- Une interface graphique (swagger-ui.html)
- Une tâche automatisée
- Des tests

Fournisseurs de données

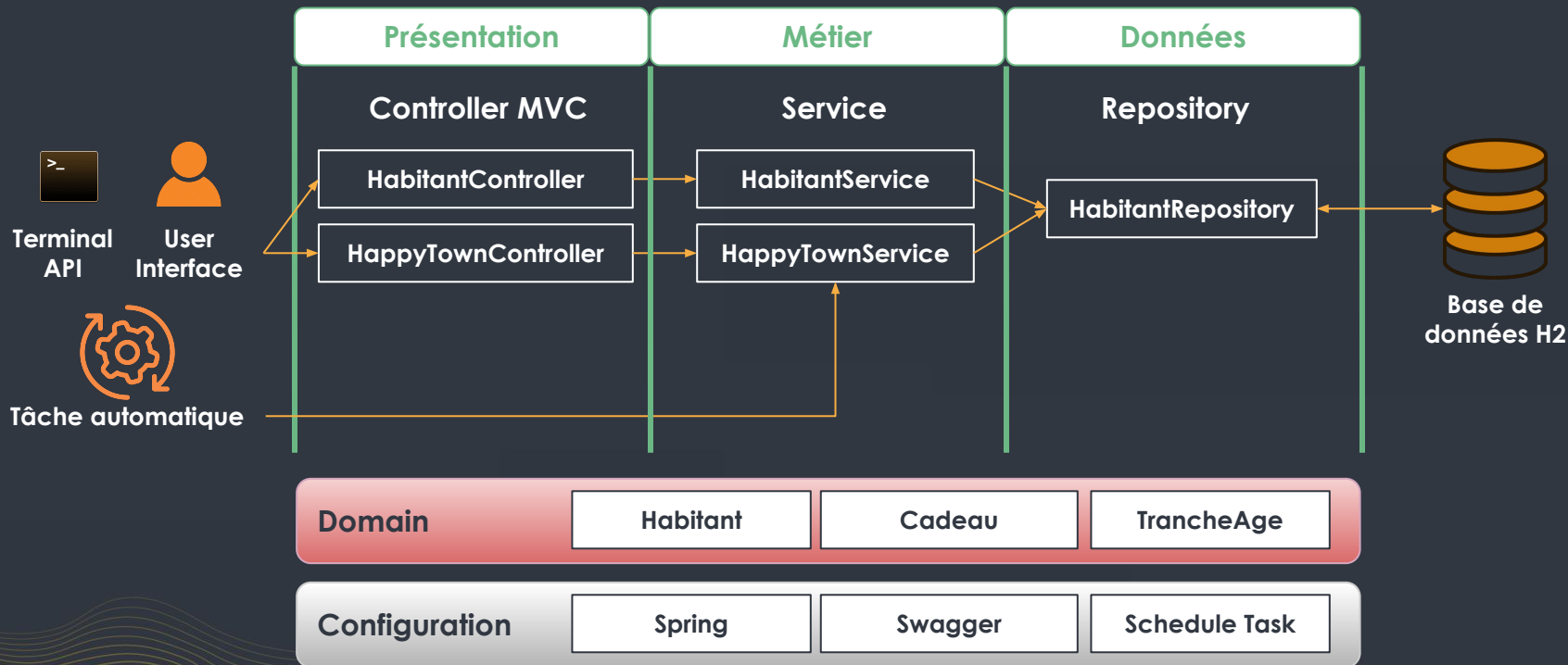
- Base de données H2 contenant les habitants
- Fichier contenant les cadeaux par tranche d'âge
- Serveur de mail

Use-Cases

- Récupérer les informations des habitants de HappyTown
- Attribuer un cadeau aux habitants de HappyTown

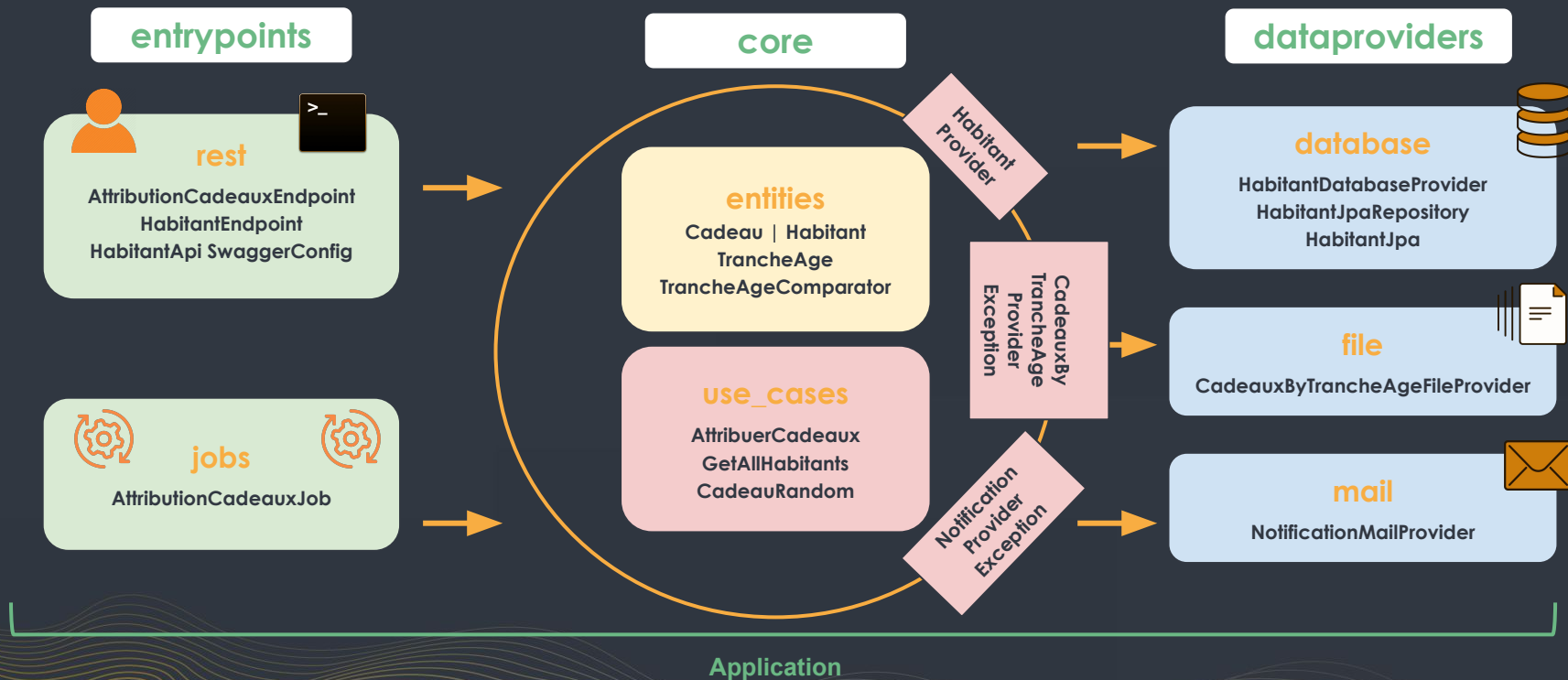
Clean Architecture - Step by Step

Migration de l'architecture actuelle 3-Tiers



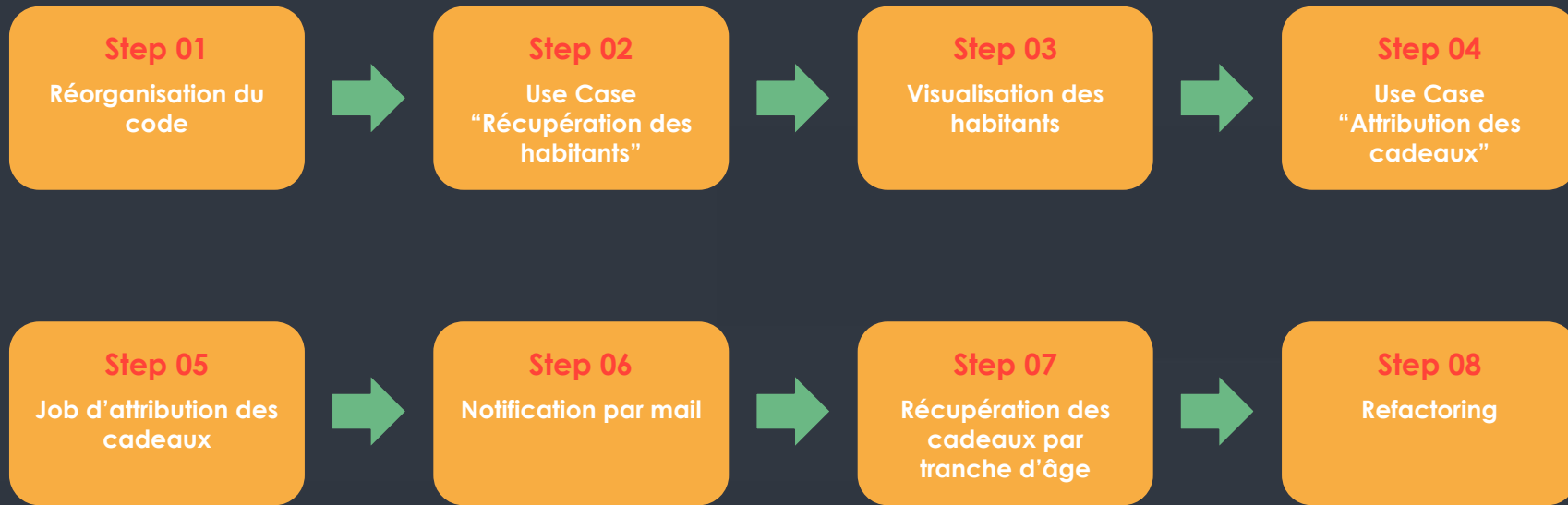
Clean Architecture - Step by Step

Schéma cible



Clean Architecture - Step by Step

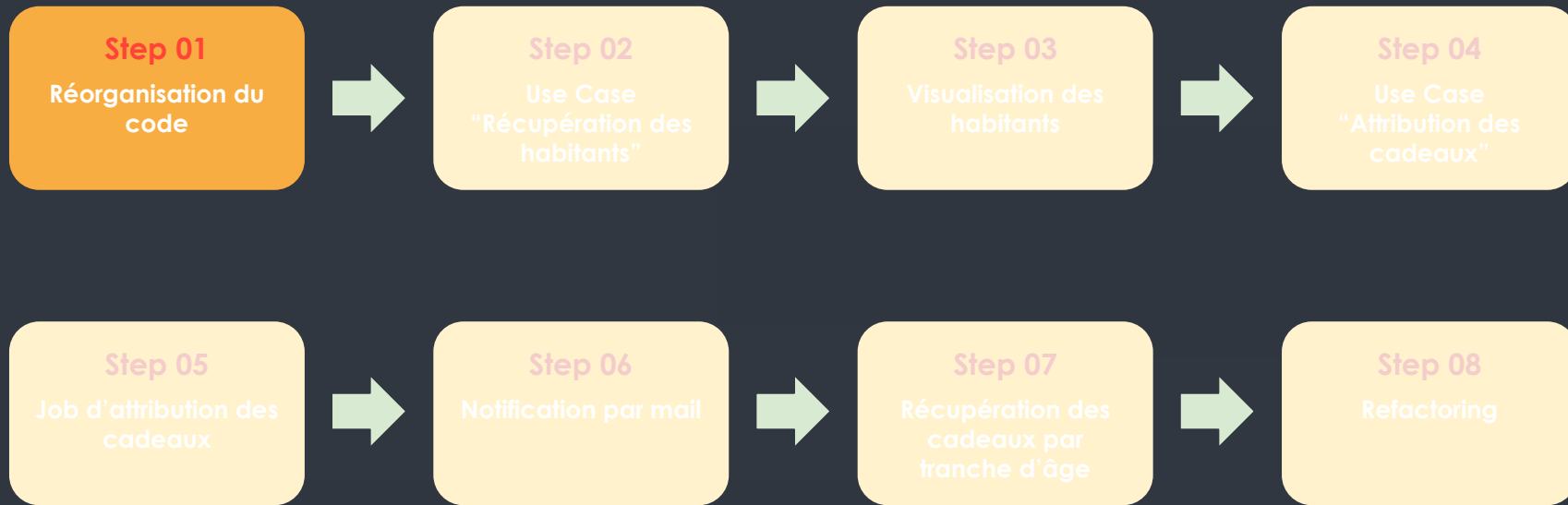
Un programme en 8 étapes



Clean Architecture - Step by Step

Step 01 : Réorganisation du code

Step 01



Clean Architecture - Step by Step

Step 01 : Réorganisation du code

Step 01

entrypoints

core

dataproviders

entities

Cadeau | Habitant
TrancheAge

use_cases

Application

Clean Architecture - Step by Step

Step 01 : Réorganisation du code

Step 01

- ▼ com.happytown
 - ▼ configuration
 - ScheduleTasks
 - SwaggerConfig
 - ▼ controller
 - HabitantController
 - HappyTownController
 - ▼ domain
 - Cadeau
 - Habitant
 - TrancheAge
 - ▼ repository
 - HabitantRepository
 - ▼ service
 - HabitantService
 - HappyTownService
 - TrancheAgeComparator

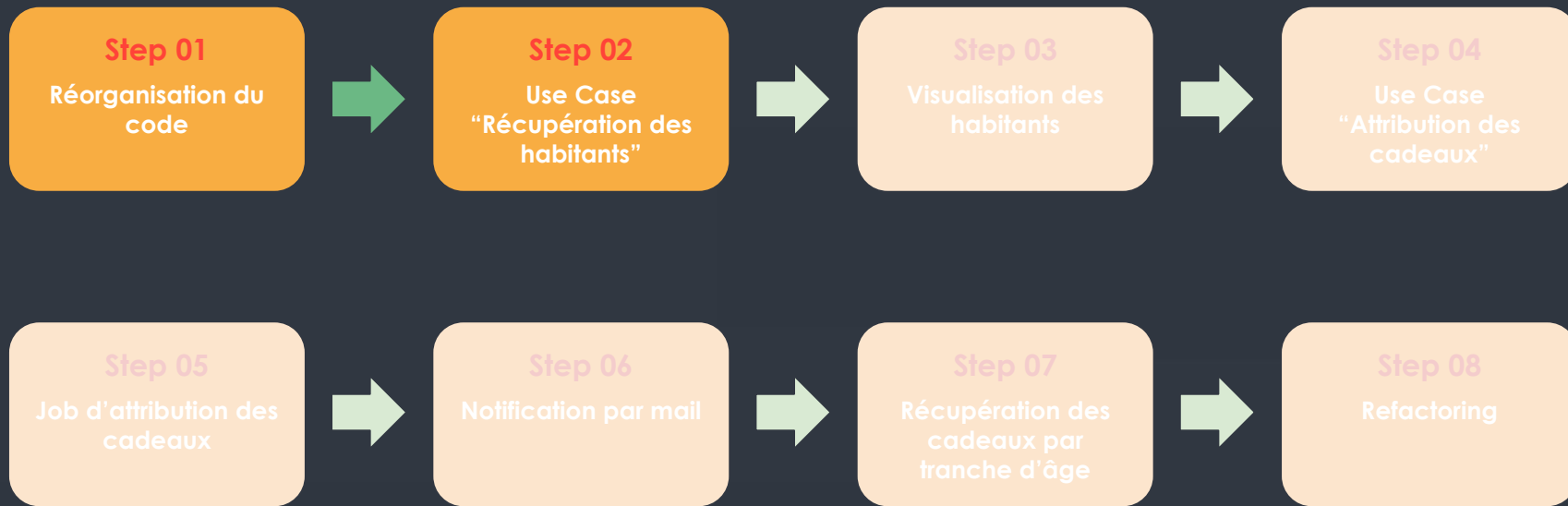
- ❑ **Création de la nouvelle structure de packages**
 - ❑ **core (entities + use_cases)**
 - ❑ **entrypoints**
 - ❑ **dataproviders**
- ❑ **Déplacement des objets de domain vers entities**
- ❑ **Suppression des frameworks / annotations (Lombok, Javax Validation)**

- ▼ com.happytown
 - ▼ configuration
 - ScheduleTasks
 - SwaggerConfig
 - ▼ controller
 - HabitantController
 - HappyTownController
 - ▼ core
 - ▼ entities
 - Cadeau
 - Habitant
 - TrancheAge
 - ▶ use_cases
 - ▶ dataproviders
 - ▶ entrypoints
 - ▼ repository
 - HabitantRepository
 - ▼ service
 - HabitantService
 - HappyTownService
 - TrancheAgeComparator

Clean Architecture - Step by Step

Step 02 : Récupération des habitants

Step 02



Clean Architecture - Step by Step

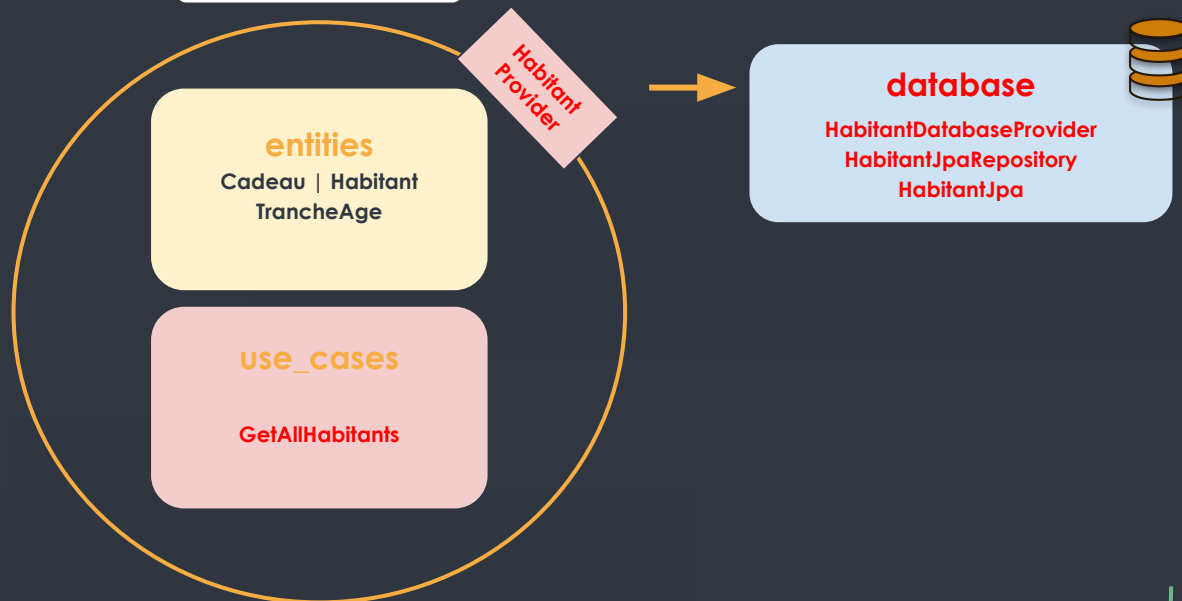
Step 02 : Récupération des habitants

Step 02

entrypoints

core

dataproviders



Application

Clean Architecture - Step by Step

Step 02 : Récupération des habitants

Step 02

- ▼ com.happytown
 - ▼ configuration
 - ScheduleTasks
 - SwaggerConfig
 - ▼ controller
 - HabitantController
 - HappyTownController
 - ▼ core
 - ▼ entities
 - Cadeau
 - Habitant
 - TrancheAge
 - ▶ use_cases
 - ▶ dataproviders
 - ▶ entypoints
 - ▼ repository
 - HabitantRepository
 - ▼ service
 - HabitantService
 - HappyTownService
 - TrancheAgeComparator

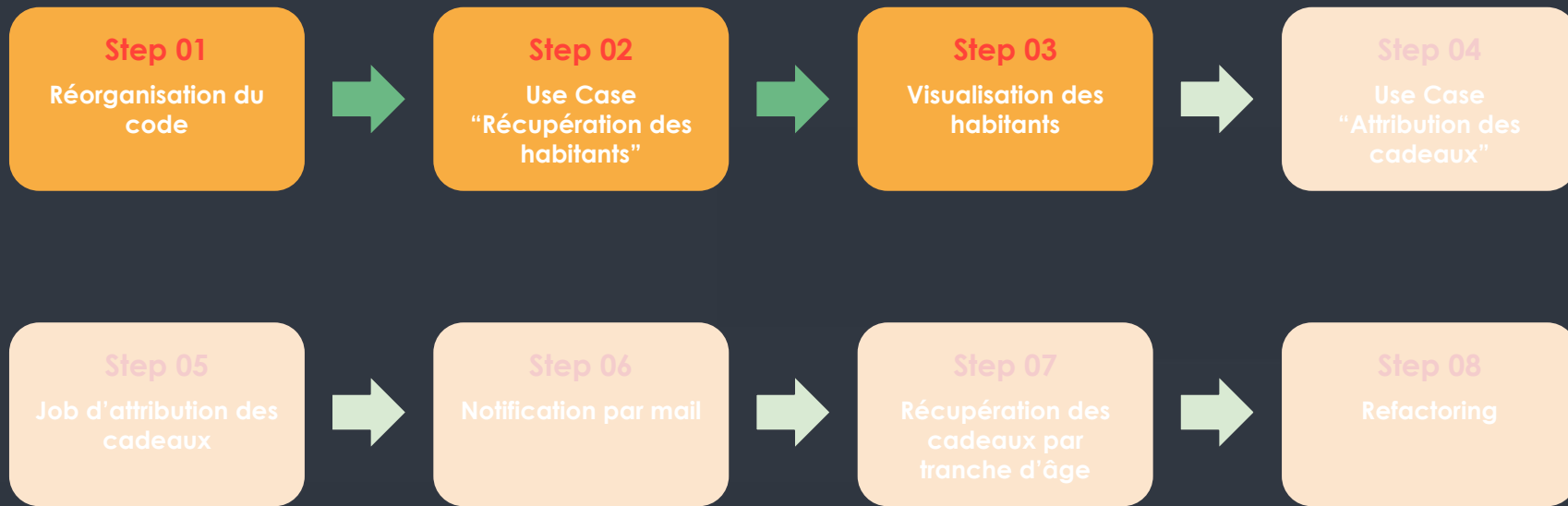
- ❑ Ajout d'un **use case** pour récupérer tous les habitants : **GetAllHabitants**
- ❑ Nécessité d'avoir un **nouveau fournisseur de données** pour les habitants : **HabitantProvider**
- ❑ Implémentation du **fournisseur de données** des habitants en base de données : **HabitantDatabaseProvider**
- ❑ Séparation entre les objets du domaine **métier** : **Habitant** et le modèle de **stockage** : **HabitantJpa**
- ❑ Suppression du package **repository**

- ▼ com.happytown
 - ▼ configuration
 - ScheduleTasks
 - SwaggerConfig
 - ▼ controller
 - HabitantController
 - HappyTownController
 - ▼ core
 - ▼ entities
 - Cadeau
 - Habitant
 - TrancheAge
 - ▼ use_cases
 - GetAllHabitants
 - HabitantProvider
 - ▼ dataproviders.database
 - HabitantDatabaseProvider
 - HabitantJpa
 - HabitantJpaRepository
 - ▶ entypoints
 - ▼ service
 - HabitantService
 - HappyTownService
 - TrancheAgeComparator

Clean Architecture - Step by Step

Step 03 : Visualisation des habitants

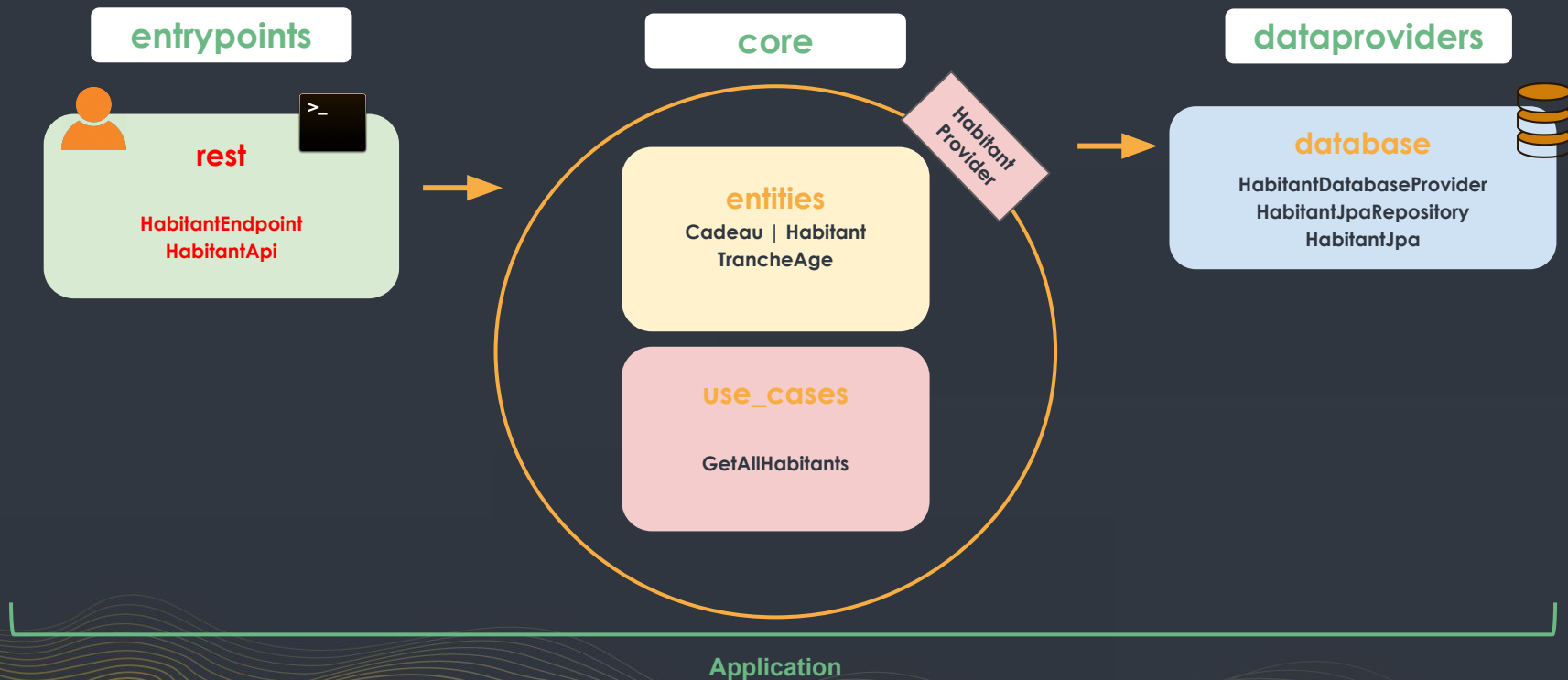
Step 03



Clean Architecture - Step by Step

Step 03 : Visualisation des habitants

Step 03



Clean Architecture - Step by Step

Step 03 : Visualisation des habitants

Step 03

- ▼ com.happytown
 - ▼ configuration
 - ScheduleTasks
 - SwaggerConfig
 - ▼ controller
 - HabitantController
 - HappyTownController
 - ▼ core
 - ▼ entities
 - Cadeau
 - Habitant
 - TrancheAge
 - ▼ use_cases
 - GetAllHabitants
 - HabitantProvider
 - ▼ dataproviders.database
 - HabitantDatabaseProvider
 - HabitantJpa
 - HabitantJpaRepository
 - ▶ endpoints
 - ▼ service
 - HabitantService
 - HappyTownService
 - TrancheAgeComparator

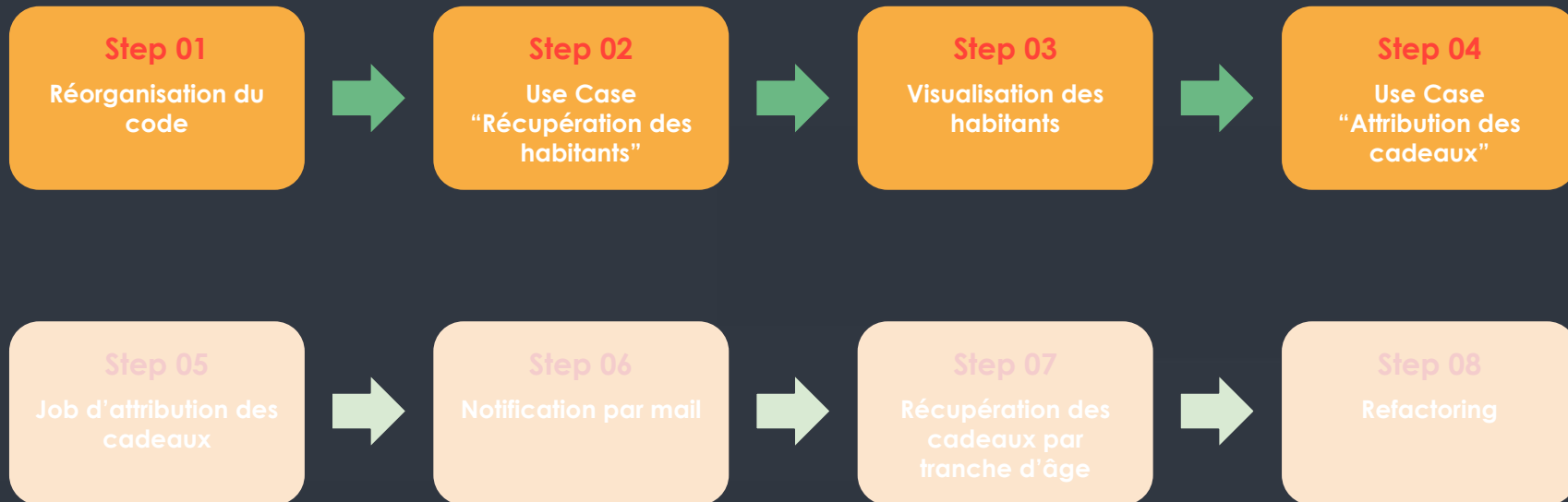
- ❑ **Ajout d'un endpoint rest pour récupérer la liste des habitants : `HabitantEndpoint`**
- ❑ **Séparation entre les objets du domaine métier : `Habitant` et le modèle de présentation : `HabitantApi`**
- ❑ **Suppression de `HabitantController` et `HabitantService`**
- ❑ **L'objet `Habitant` est maintenant agnostique de tout framework technique (présentation, persistance)**

- ▼ com.happytown
 - ▼ configuration
 - ScheduleTasks
 - SwaggerConfig
 - ▼ controller
 - HappyTownController
 - ▼ core
 - ▼ entities
 - Cadeau
 - Habitant
 - TrancheAge
 - ▼ use_cases
 - GetAllHabitants
 - HabitantProvider
 - ▼ dataproviders.database
 - HabitantDatabaseProvider
 - HabitantJpa
 - HabitantJpaRepository
 - ▼ endpoints.rest
 - HabitantApi
 - HabitantEndpoint
 - ▼ service
 - HappyTownService
 - TrancheAgeComparator

Clean Architecture - Step by Step

Step 04 : Attribution des cadeaux

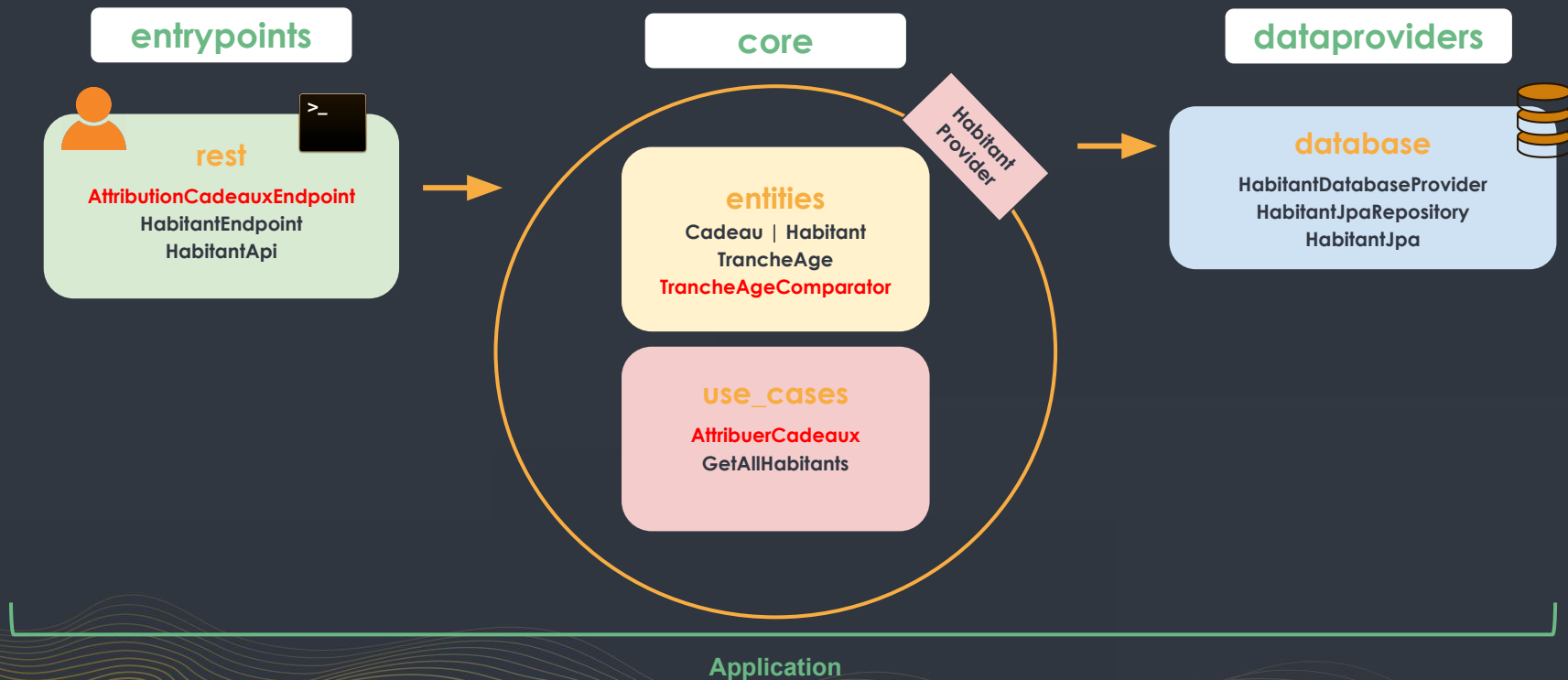
Step 04



Clean Architecture - Step by Step

Step 04 : Attribution des cadeaux

Step 04



Clean Architecture - Step by Step

Step 04 : Attribution des cadeaux

Step 04

```
▼ com.happytown
  ▼ configuration
    ○ ScheduleTasks
    ○ SwaggerConfig
  ▼ controller
    ○ HappyTownController
  ▼ core
    ▼ entities
      ○ Cadeau
      ○ Habitant
      ○ TrancheAge
    ▼ use_cases
      ○ GetAllHabitants
      ⓘ HabitantProvider
  ▼ dataproviders.database
    ○ HabitantDatabaseProvider
    ○ HabitantJpa
    ⓘ HabitantJpaRepository
  ▼ entrypoints.rest
    ○ HabitantApi
    ○ HabitantEndpoint
  ▼ service
    ○ HappyTownService
    ○ TrancheAgeComparator
```

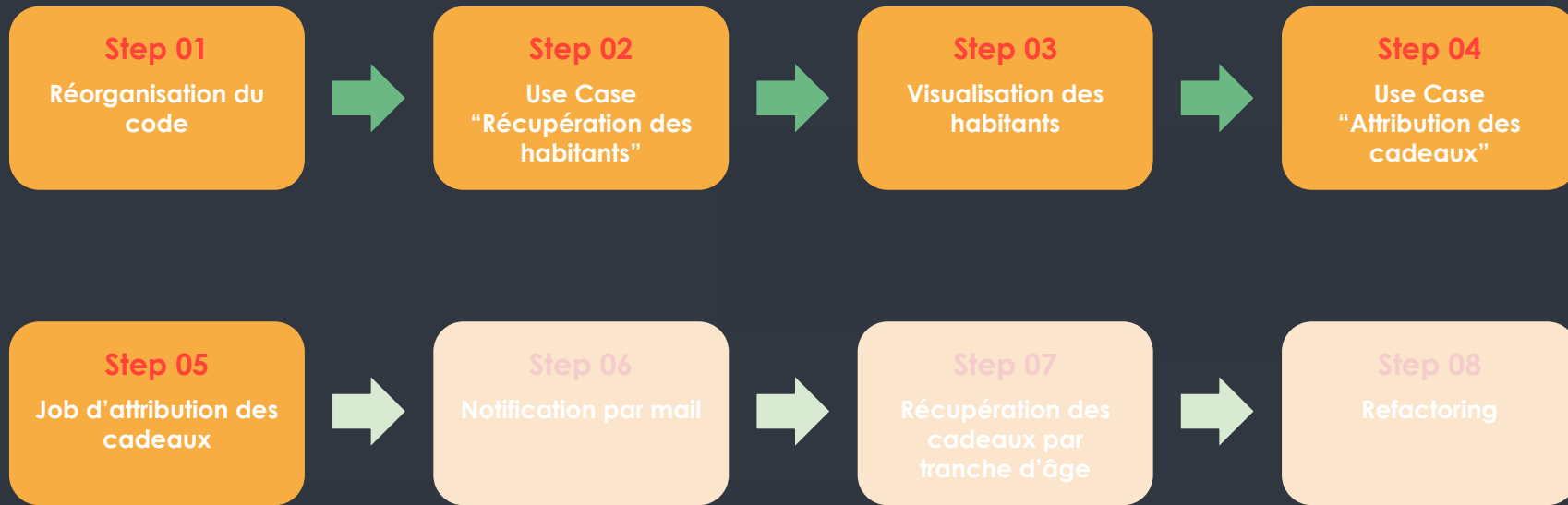
- ❑ **Ajout d'un use case pour attribuer les cadeaux : **AttribuerCadeaux****
- ❑ **Ajout d'un entypoint rest pour attribuer les cadeaux : **AttributionCadeauxEndpoint****
- ❑ **Suppression de HappyTownService**
- ❑ **Le comparateur des tranches d'âge **TrancheAgeComparator** rejoint le package des **entites****
- ❑ **Suppression du package de **service****

```
▼ com.happytown
  ▼ configuration
    ○ ScheduleTasks
    ○ SwaggerConfig
  ▼ core
    ▼ entities
      ○ Cadeau
      ○ Habitant
      ○ TrancheAge
      ○ TrancheAgeComparator
    ▼ use_cases
      ○ AttribuerCadeaux
      ○ GetAllHabitants
      ⓘ HabitantProvider
  ▼ dataproviders.database
    ○ HabitantDatabaseProvider
    ○ HabitantJpa
    ⓘ HabitantJpaRepository
  ▼ entrypoints.rest
    ○ AttributionCadeauxEndpoint
    ○ HabitantApi
    ○ HabitantEndpoint
```

Clean Architecture - Step by Step

Step 05 : Jobb d'attribution des cadeaux

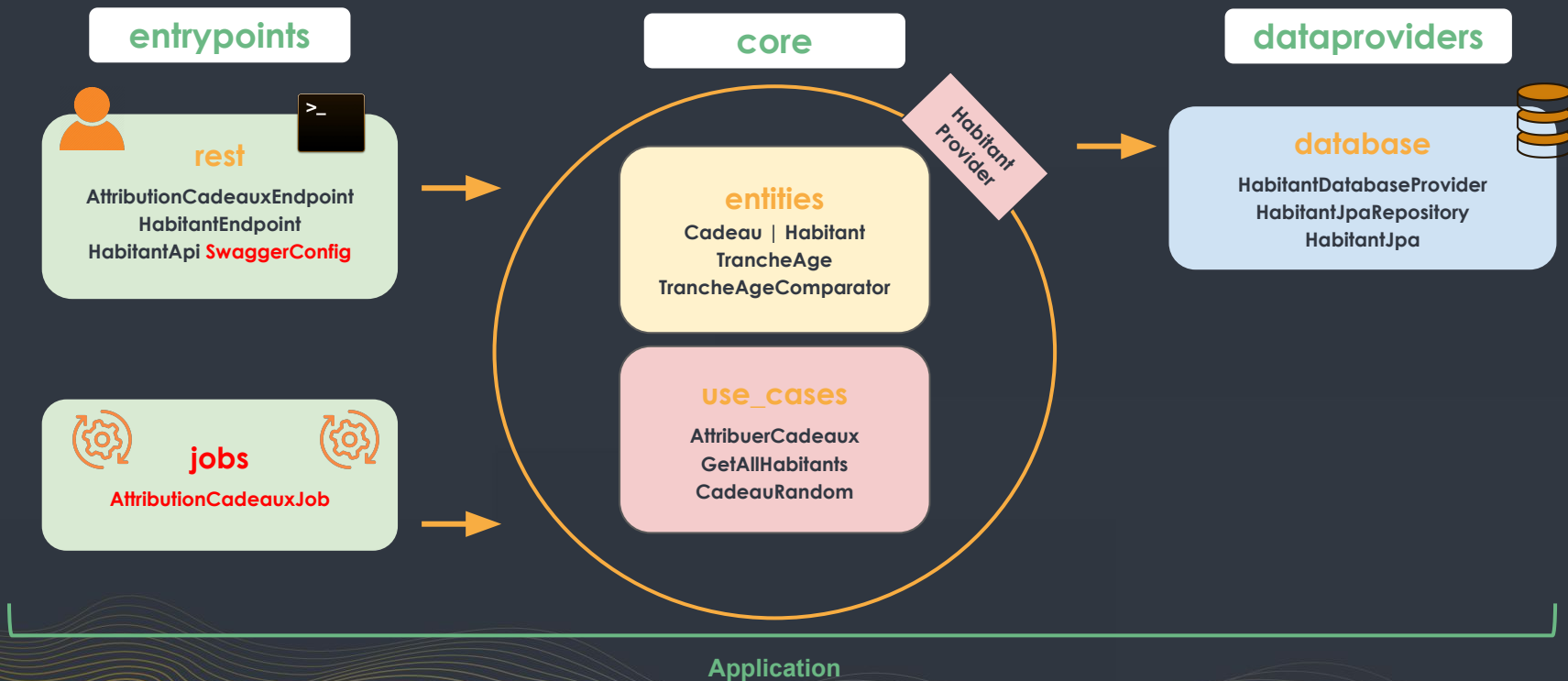
Step 05



Clean Architecture - Step by Step

Step 05 : Job d'attribution des cadeaux

Step 05



Clean Architecture - Step by Step

Step 05 : Job d'attribution des cadeaux

Step 05

- ▼ com.happytown
 - ▼ configuration
 - ScheduleTasks
 - SwaggerConfig
 - ▼ core
 - ▼ entities
 - Cadeau
 - Habitant
 - TrancheAge
 - TrancheAgeComparator
 - ▼ use_cases
 - AttribuerCadeaux
 - GetAllHabitants
 - HabitantProvider
 - ▼ dataproviders.database
 - HabitantDatabaseProvider
 - HabitantJpa
 - HabitantJpaRepository
 - ▼ entrypoints.rest
 - AttributionCadeauxEndpoint
 - HabitantApi
 - HabitantEndpoint

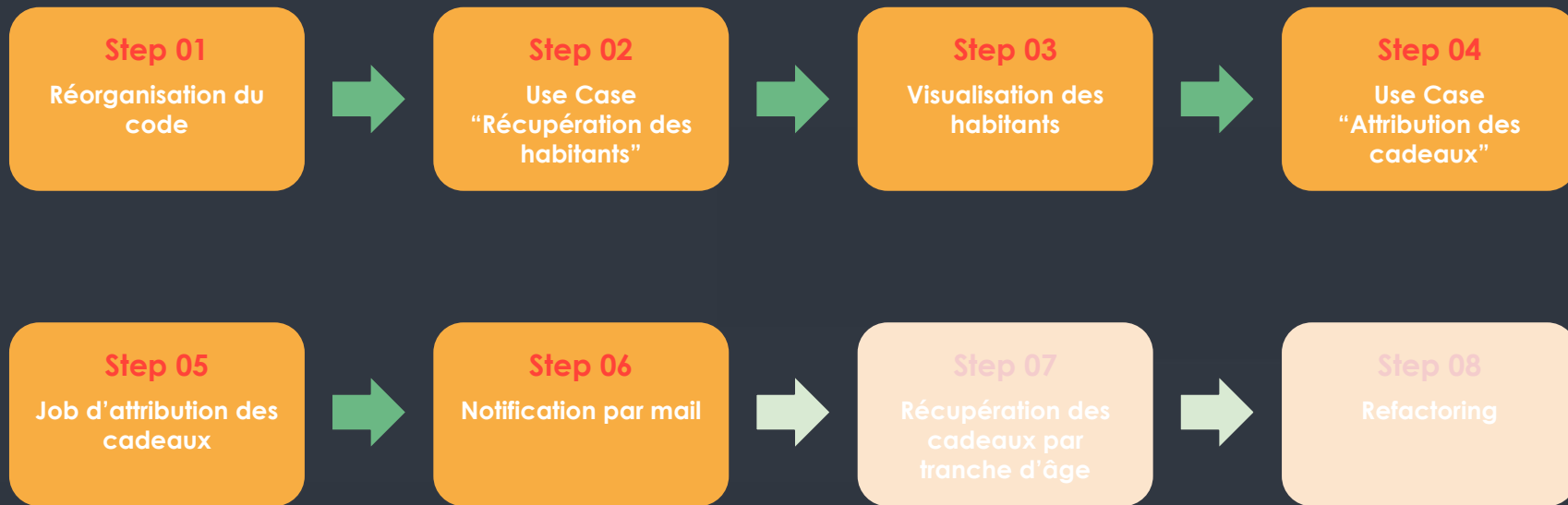
- ❑ **Ajout d'un entripoint jobs** pour la tâche automatique d'attribution de cadeaux : **AttributionCadeauxJob**
- ❑ **Déplacement** de la classe de configuration **SwaggerConfig** dans le package **rest** des **entrypoints**
- ❑ **Suppression** du package de **configuration**
- ❑ **Mise en place d'un mécanisme de vérification** du respect de la **clean architecture (sens des dépendances)** : **tools/check-cleanArchi.sh**

- ▼ com.happytown
 - ▼ core
 - ▼ entities
 - Cadeau
 - Habitant
 - TrancheAge
 - TrancheAgeComparator
 - ▼ use_cases
 - AttribuerCadeaux
 - GetAllHabitants
 - HabitantProvider
 - ▼ dataproviders.database
 - HabitantDatabaseProvider
 - HabitantJpa
 - HabitantJpaRepository
 - ▼ entrypoints
 - ▼ jobs
 - AttributionCadeauxJob
 - ▼ rest
 - AttributionCadeauxEndpoint
 - HabitantApi
 - HabitantEndpoint
 - SwaggerConfig

Clean Architecture - Step by Step

Step 06 : Notification par mail

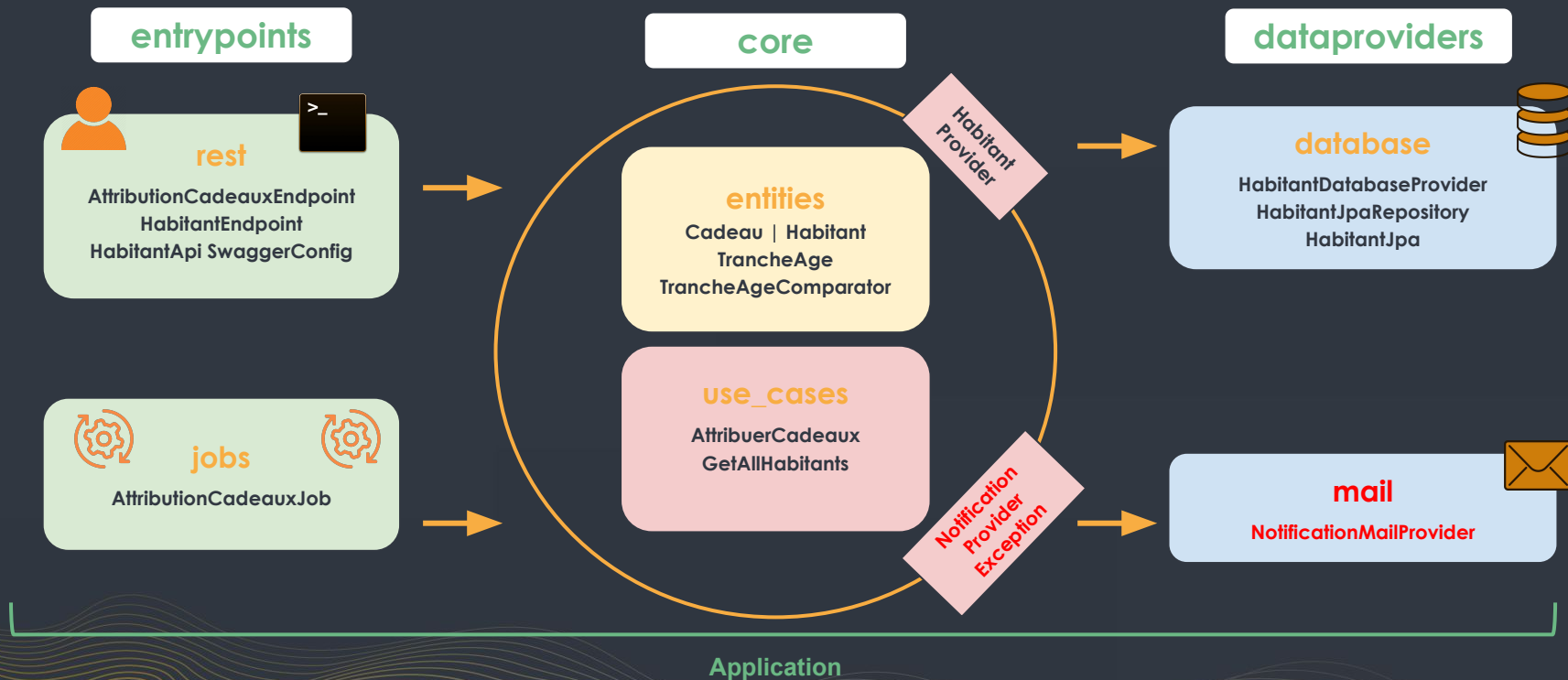
Step 06



Clean Architecture - Step by Step

Step 06 : Notification par mail

Step 06

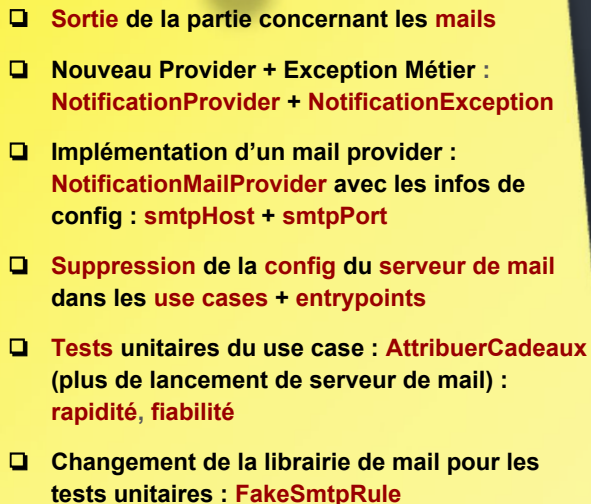


Clean Architecture - Step by Step

Step 06 : Notification par mail

Step 06

```
▼ com.happytown
  ▼ core
    ▼ entities
      C Cadeau
      C Habitant
      C TrancheAge
      C TrancheAgeComparator
    ▼ use_cases
      C AttribuerCadeaux
      C GetAllHabitants
      I HabitantProvider
    ▼ dataproviders.database
      C HabitantDatabaseProvider
      C HabitantJpa
      I HabitantJpaRepository
    ▼ endpoints
      ▼ jobs
        C AttributionCadeauxJob
      ▼ rest
        C AttributionCadeauxEndpoint
        C HabitantApi
        C HabitantEndpoint
        C SwaggerConfig
```

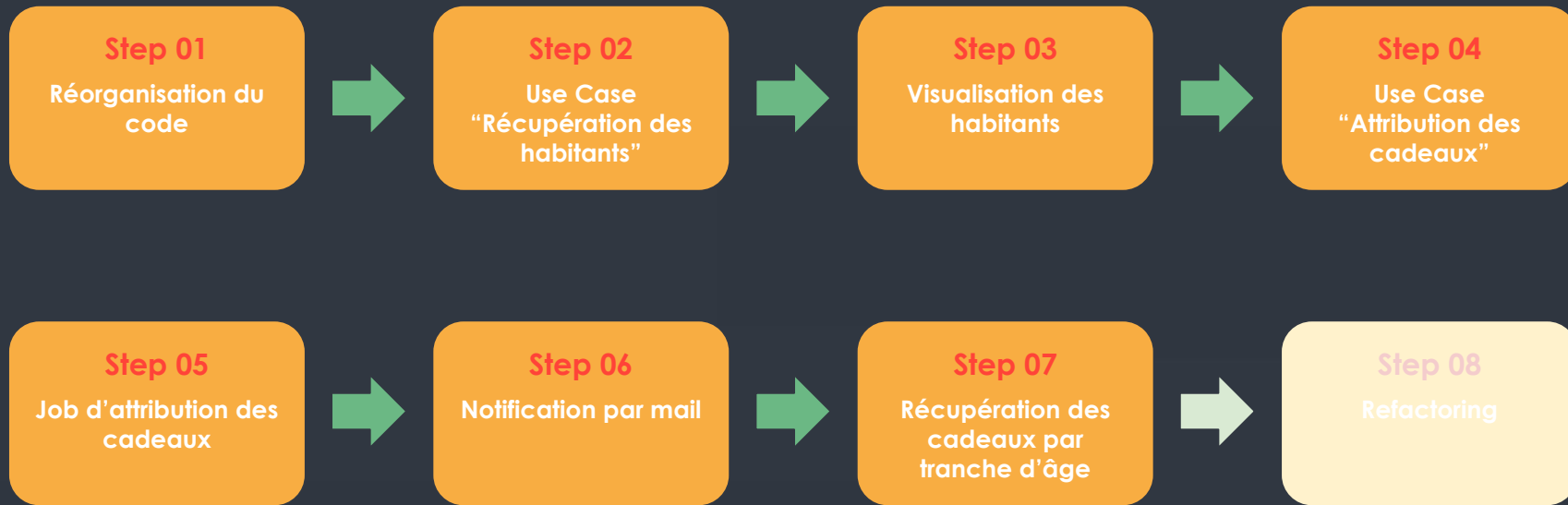
- 
- ❑ **Sortie de la partie concernant les mails**
 - ❑ **Nouveau Provider + Exception Métier : NotificationProvider + NotificationException**
 - ❑ **Implémentation d'un mail provider : NotificationMailProvider avec les infos de config : smtpHost + smtpPort**
 - ❑ **Suppression de la config du serveur de mail dans les use cases + entypoints**
 - ❑ **Tests unitaires du use case : AttribuerCadeaux (plus de lancement de serveur de mail) : rapidité, fiabilité**
 - ❑ **Changement de la librairie de mail pour les tests unitaires : FakeSmtpRule**

```
▼ com.happytown
  ▼ core
    ▼ entities
      C Cadeau
      C Habitant
      C TrancheAge
      C TrancheAgeComparator
    ▼ use_cases
      C AttribuerCadeaux
      C GetAllHabitants
      I HabitantProvider
      NotificationException
      I NotificationProvider
    ▼ dataproviders
      ▼ database
        C HabitantDatabaseProvider
        C HabitantJpa
        I HabitantJpaRepository
      ▼ mail
        C NotificationMailProvider
    ▼ endpoints
      ▼ jobs
        C AttributionCadeauxJob
      ▼ rest
        C AttributionCadeauxEndpoint
        C HabitantApi
        C HabitantEndpoint
        C SwaggerConfig
```


Clean Architecture - Step by Step

Step 07 : Récupération des cadeaux par tranche d'âge

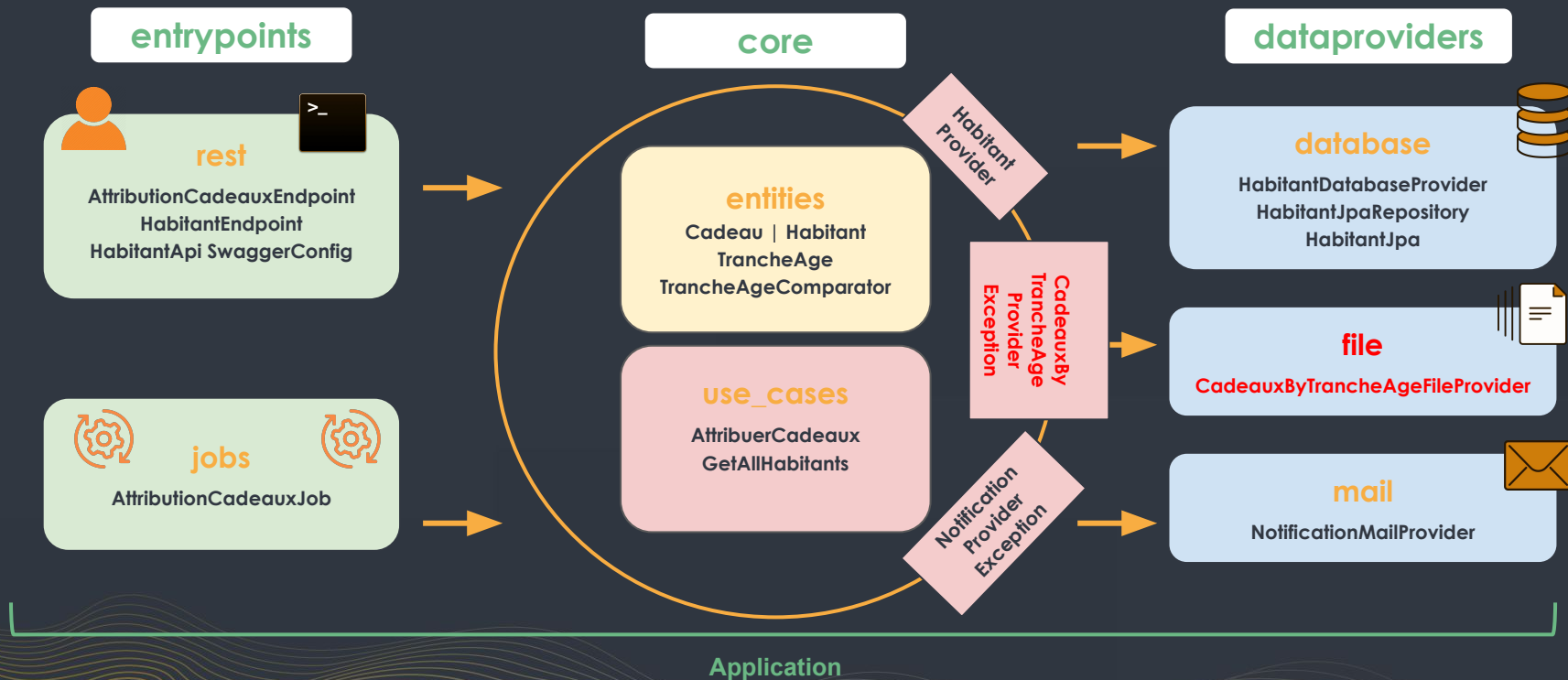
Step 07



Clean Architecture - Step by Step

Step 07 : Récupération des cadeaux par tranche d'âge

Step 07

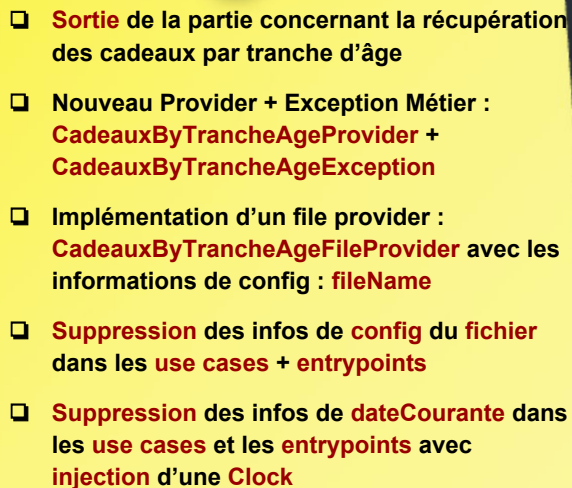


Clean Architecture - Step by Step

Step 07 : Récupération des cadeaux par tranche d'âge

Step 07

```
▼ com.happytown
  ▼ core
    ▼ entities
      C Cadeau
      C Habitant
      C TrancheAge
      C TrancheAgeComparator
    ▼ use_cases
      C AttribuerCadeaux
      C GetAllHabitants
      I HabitantProvider
      ⚡ NotificationException
      I NotificationProvider
    ▼ dataproviders
      ▼ database
        C HabitantDatabaseProvider
        C HabitantJpa
        I HabitantJpaRepository
      ▼ mail
        C NotificationMailProvider
    ▼ entrypoints
      ▼ jobs
        C AttributionCadeauxJob
      ▼ rest
        C AttributionCadeauxEndpoint
        C HabitantApi
        C HabitantEndpoint
        C SwaggerConfig
```

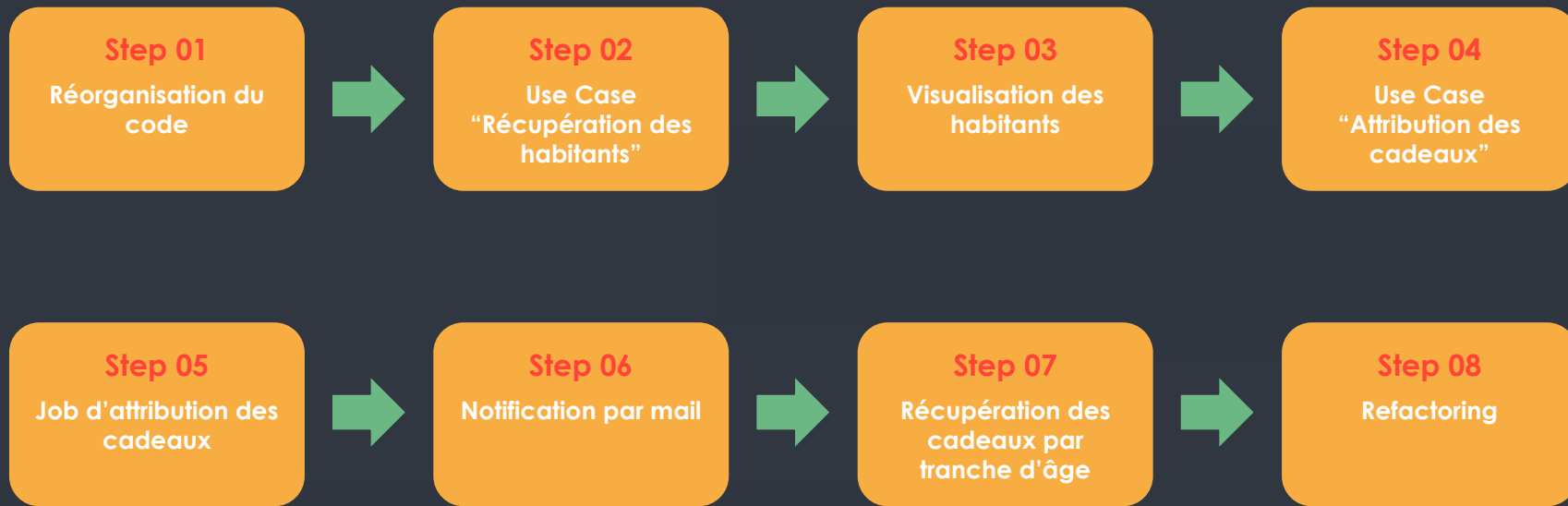
- 
- ❑ **Sortie** de la partie concernant la récupération des cadeaux par tranche d'âge
 - ❑ Nouveau Provider + Exception Métier : **CadeauxByTrancheAgeProvider + CadeauxByTrancheAgeException**
 - ❑ Implémentation d'un file provider : **CadeauxByTrancheAgeFileProvider** avec les informations de config : **fileName**
 - ❑ **Suppression** des infos de **config** du **fichier** dans les **use cases + entrypoints**
 - ❑ **Suppression** des infos de **dateCourante** dans les **use cases et les entrypoints** avec **injection d'une Clock**

```
▼ com.happytown
  ▼ core
    ▼ entities
      C Cadeau
      C Habitant
      C TrancheAge
      C TrancheAgeComparator
    ▼ use_cases
      C AttribuerCadeaux
      C CadeauxByTrancheAgeException
      I CadeauxByTrancheAgeProvider
      C GetAllHabitants
      I HabitantProvider
      ⚡ NotificationException
      I NotificationProvider
    ▼ dataproviders
      ▼ database
        C HabitantDatabaseProvider
        C HabitantJpa
        I HabitantJpaRepository
      ▼ file
        C CadeauxByTrancheAgeFileProvider
      ▼ mail
        C NotificationMailProvider
    ▼ entrypoints
      ▼ jobs
        C AttributionCadeauxJob
      ▼ rest
        C AttributionCadeauxEndpoint
        C HabitantApi
        C HabitantEndpoint
        C SwaggerConfig
```

Clean Architecture - Step by Step

Step 08 : Refactoring

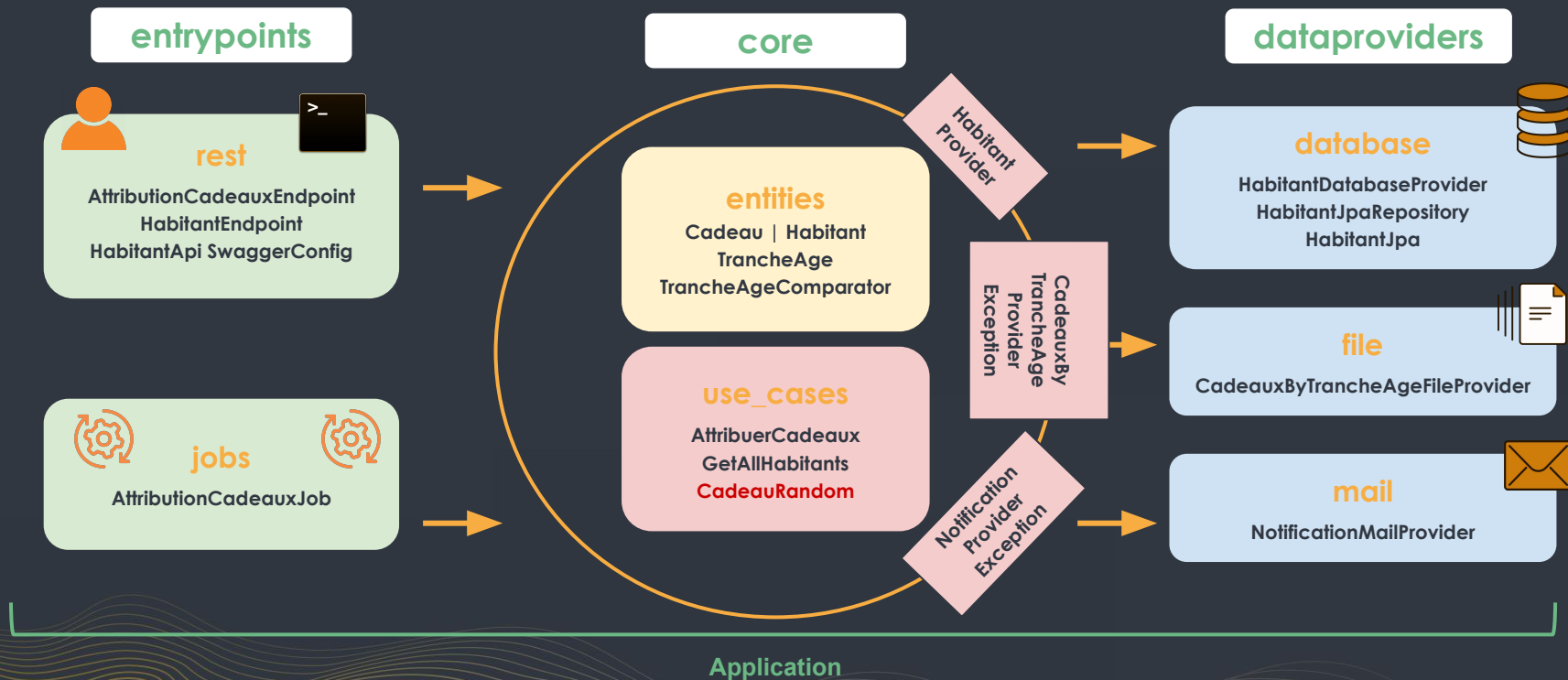
Step 08



Clean Architecture - Step by Step

Step 08 : Refactoring

Step 08

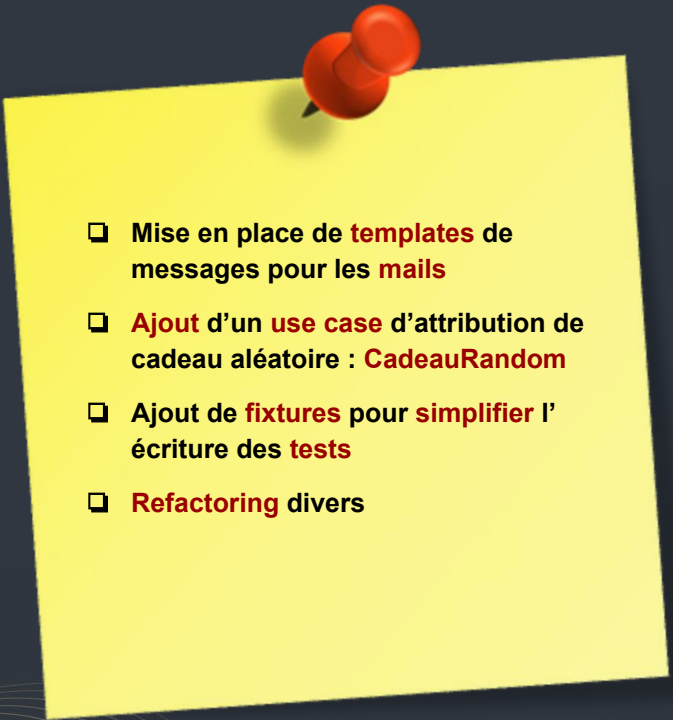


Clean Architecture - Step by Step

Step 08 : Refactoring

Step 08

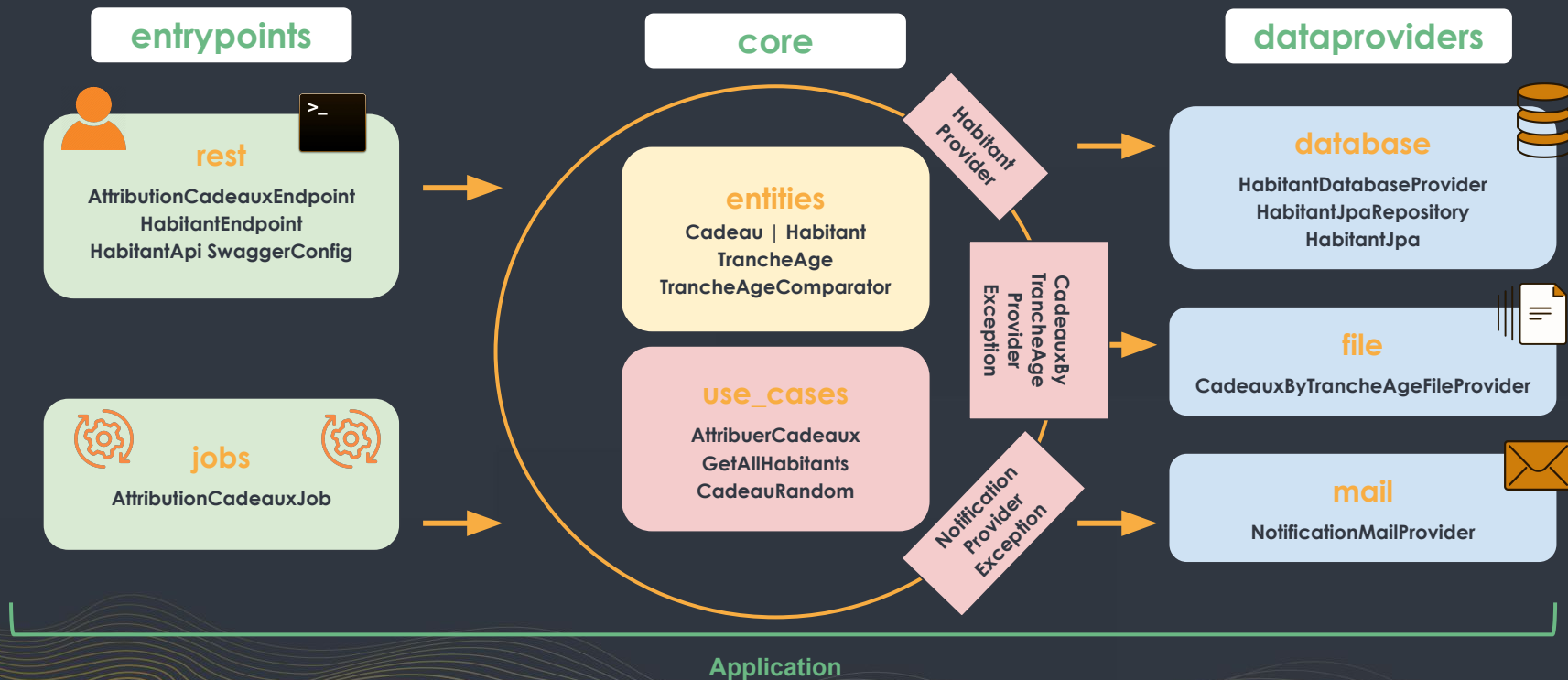
```
▼ com.happytown
  ▼ core
    ▼ entities
      Cadeau
      Habitant
      TrancheAge
      TrancheAgeComparator
    ▼ use_cases
      AttribuerCadeaux
      CadeauxByTrancheAgeException
      CadeauxByTrancheAgeProvider
      GetAllHabitants
      HabitantProvider
      NotificationException
      NotificationProvider
  ▼ dataproviders
    ▼ database
      HabitantDatabaseProvider
      HabitantJpa
      HabitantJpaRepository
    ▼ file
      CadeauxByTrancheAgeFileProvider
    ▼ mail
      NotificationMailProvider
  ▼ endpoints
    ▼ jobs
      AttributionCadeauxJob
    ▼ rest
      AttributionCadeauxEndpoint
      HabitantApi
      HabitantEndpoint
      SwaggerConfig
```

- 
- ❑ Mise en place de **templates de messages** pour les **mails**
 - ❑ Ajout d'un **use case** d'attribution de cadeau aléatoire : **CadeauRandom**
 - ❑ Ajout de **fixtures** pour **simplifier l'écriture des tests**
 - ❑ **Refactoring divers**

```
▼ com.happytown
  ▼ core
    ▼ entities
      Cadeau
      Habitant
      TrancheAge
      TrancheAgeComparator
    ▼ use_cases
      AttribuerCadeaux
      CadeauRandom
      CadeauxByTrancheAgeException
      CadeauxByTrancheAgeProvider
      GetAllHabitants
      HabitantProvider
      NotificationException
      NotificationProvider
  ▼ dataproviders
    ▼ database
      HabitantDatabaseProvider
      HabitantJpa
      HabitantJpaRepository
    ▼ file
      CadeauxByTrancheAgeFileProvider
    ▼ mail
      NotificationMailProvider
  ▼ endpoints
    ▼ jobs
      AttributionCadeauxJob
    ▼ rest
      AttributionCadeauxEndpoint
      HabitantApi
      HabitantEndpoint
      SwaggerConfig
```


Clean Architecture

Schéma final

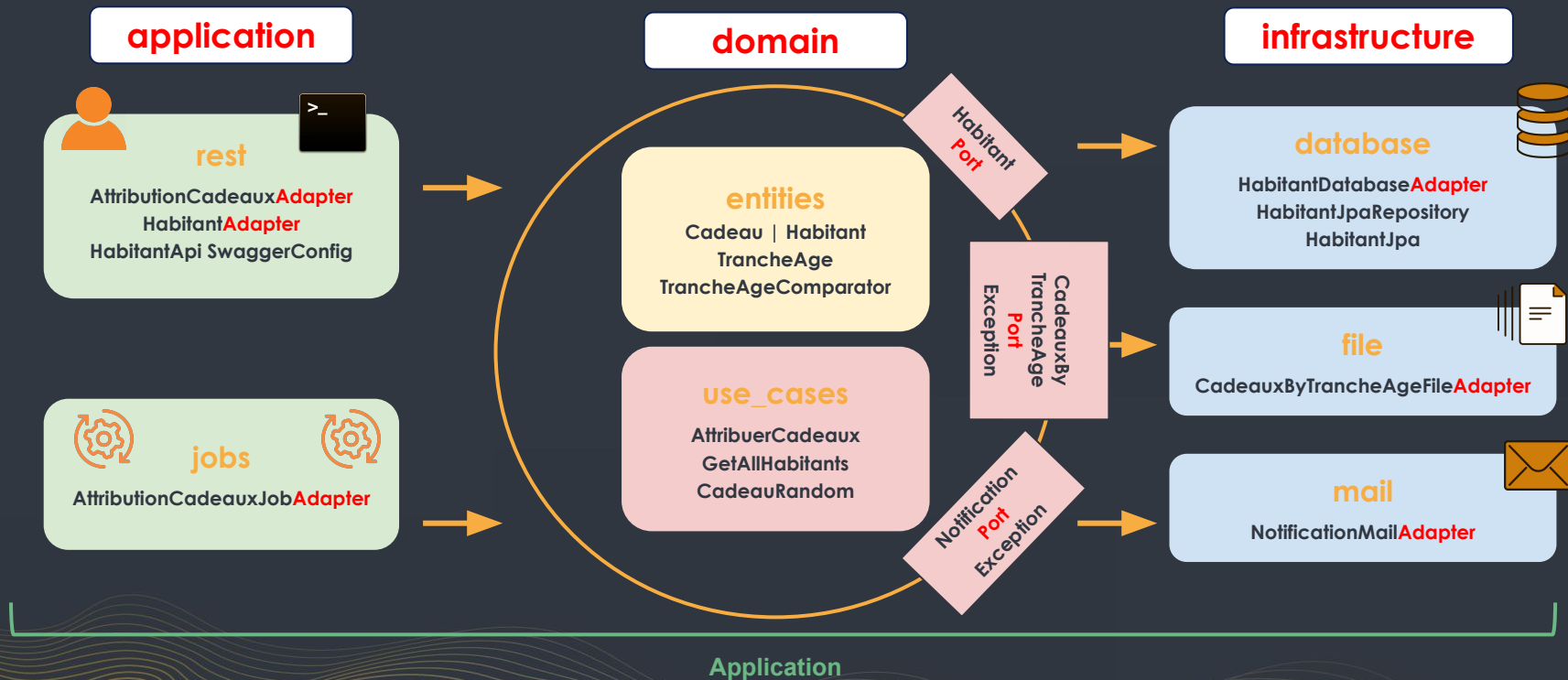


05

Hexagonale Architecture

Hexagonale Architecture

Schéma final



Debrief et conclusions

- ✓ Isolation et protection du métier
- ✓ Totale indépendance vis à vis des frameworks avec un métier clair et explicite
- ✓ Meilleure testabilité orientée sur le comportement métier
- ✓ Une pyramide de tests saine pour les évolutions et la maintenance
- ✓ Séparation claire des problèmes

Discuter en équipe de votre choix d'architecture : clean architecture, hexagonale architecture ou un mix des deux ou une autre...

L'important est de redonner sa place au métier et de sortir des architectures à découpage technique

DEVOXX™ France

MERCI ! :)

- Céline Gilet -

@celinegilet

#DevoxxFR

Références

- <https://github.com/celinegilet/happy-town>
- <https://blog.octo.com/architecture-hexagonale-trois-principes-et-un-exemple-dimplmentation/>
- <https://github.com/damienbeaufils/clean-architecture-demo>
- <https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html>
- <https://github.com/mattia-battiston/clean-architecture-example>