Clean Architecture & Python/Django

Yann JACOB

Décembre 2017

Principe

Des problèmes rencontrés tous les jours...

- Lire facilement les Use Cases, le domaine métier
- Avoir des tests réellement unitaires (AUCUN appel BDD, fichier, ...)
- Dépendance aux choix techno initiaux

Conséquences

- Temps perdu pour comprendre le métier
- L'abandon des tests (ou peu de tests, ou des problème à l'intégration, ...)
- Difficulté à faire évoluer le code

Principe

Cas le plus courant

- Imbrication entre logique métier, BDD, affichage, appels APIs externes, ...
- Lenteur (charger le framework, lancer une instance jetable BDD, ...)
- Logique métier enfouie, difficile à reproduire dans un test

Idée

Le centre d'une application n'est pas le framework, ni la base de données, mais le domaine / les use case.

Clean Architecture

Origine

- Robert C. Martin (Uncle Bob)
- Exemples en Java

Dans ce talk

- Demo jouet Python / Django (gestion de paniers)
- **Pythonique - PEP20**
- Mon expérience sur des projets réels

Clean Architecture

Comment

- Logique métier = coeur application
- Logique métier isolée de la technique
- Détails techniques autour (BDD, Web, ressources externes, . . .)
- Modèle en 4 couches

Clean Architecture

Modèle en couches

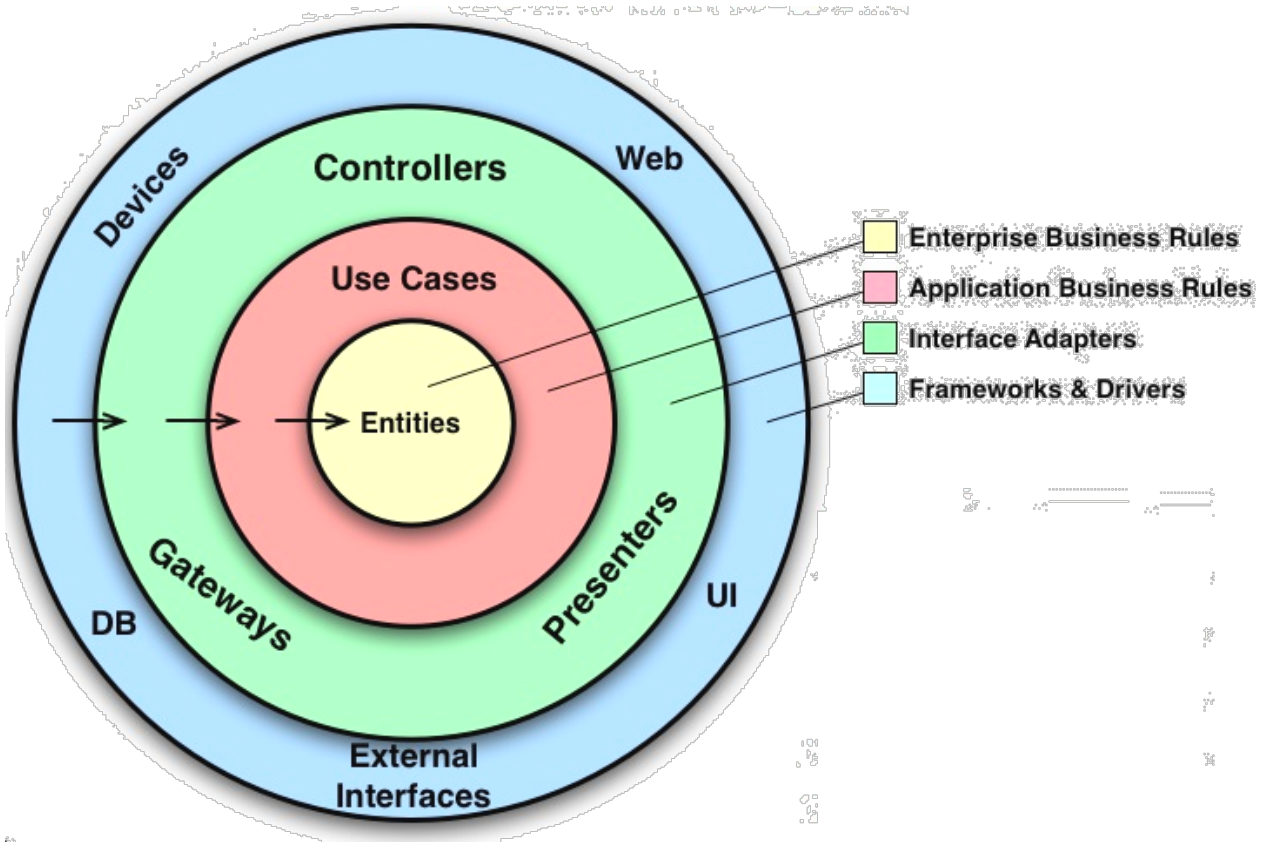

FIGURE : Figure par Robert C. Martin

Couche 1 : Entités

Entités

- Python pur, ni dépendances ni objets externes
- Classe entités du domaine métier
- Fournit explicement toutes les règles métier
- Exception = violation de règle
- Compatible DDD, TDD

Couche 1 : Entités (modules/entities/product)

Entités : Produit

- Le nom doit être des lettres puis des chiffres
- Prix strictement positif
- Nombre de produits réservés (dans les paniers) et disponibles
- Conversion monétaire

Couche 1 : Entités (modules/entities/cart)

Entités : Panier

- Liste de produits achetés avec le nombre (positif !)
- Conversion monétaire / valeur totale
- Exercice : 25 euros de montant minimum de panier

Couche 2 : Cas d'Usage

Cas d'Usage

- Python pur, ni dépendances ni objets externes
- Fonctions cas d'usage métiers explicites
- Cas d'usage trivaux shuntés dans la démo (getters/setters/etc.)

Couche 2 : Cas d'Usage (modules/usecases/fill_cart)

Cas d'Usage : remplir panier

Trois choses à faire :

- 1) Retirer le nombre de la liste des produits disponibles
- 2) Ajout le nombre à la la listes produits réservés
- 3) Ajouter dans le panier

Couche 2 : Cas d'Usage (modules/usecases/fill_available_products)

Cas d'Usage : autres

- Ajouter des produits disponibles
- Exercice : détruire un panier (penser à rendre au stock)

Couche 3 : Adapteurs

Présentation

- Le liant entre les couches métier, et les briques techniques
- Présentation : fournit les différentes vues des entités
- Répertoires : sélectionne la source de données selon l'objets
- Controlleurs divers et variés

Couche 3 : Présentation (modules/presentation)

Présentation

- Différentes présentations pour une donnée (JSON, XML, . . .)
- Choisir quel champs afficher (nb_available et nb_reserved sont cachés)

Couche 4 : Couches Techniques

Couches Techniques

- Les interfaces externes à l'application
- Notamment Django intervient ici
- Base de données, interface Web, API HTTP, ...
- Note : Django peut servir sans avoir besoin de BDD

Couche 4 : Base de données (postgres_api)

Base de données (postgres_api)

- postgres_api/models : le modèle
- postgres_api/queries : requêtes pour interagir avec la BDD
- Les appels retournent toujours des entités
- Jamais d'objet BDD hors de ce répertoire
- Testé avec UnitTest de Django

Couche 4 : Appel API externe (http_api)

Couche 4 : Appel API externe (http_api)

- Appel à une API REST externe pour le taux de conversion euro-dollar en temps réel
- Mettre euro-dollar-rate.json dans localhost pour simuler un appel HTTP externe
- Définit la stratégie d'appels HTTP, de gestion des erreurs
- Outillage de test à définir, adapté au cas

Couche 4 : Vue Web

Couche 4 : Vue Web

- /Web pour la config
- /Webapps pour les applis
- /Webapps/API : example Django Rest Framework / Swagger
- Un site web pourrait se greffer facilement en plus de l'API

Couche 4 : Vue Web (webapps/API/views)

Couche 4 : Vue Web(webapps/API/views)

- wrapper exception métier - messages d'erreurs
- Le comportement REST attendu est défini ici
- Les méthodes font le liant entre entités, use cases, présentation, BDD, API

Conclusion

Les forces

- Domaine et règles métiers évidents
- Tester sans prise de tête (quasiment tout en pre-commit !)
- La stack Django fonctionne bien
- Migration MongoDB $\rightarrow$ PostGres triviale
- Stratégies d'appel HTTP, de présentation REST, de la gestion d'erreurs, d'optimsation des appels BDD, . . . groupées à 1 endroit

Conclusion

Les limitations

- Surtout utile pour les projets moyens/gros avec de la logique métier
- Objets passe plat, beaucoup d'objets (shuntez les coquilles vides !)
- Quelques hacks sur Django (faux projet postgres_api, structuration inhabituelle)
- Incompactible avec d'autres frameworks

Conclusion

Perspectives

- Extension de Django créeant le bon squelette from scratch ?
- Explorer de façon plus complète les concepts théoriques et adapter à python