Architecture hexagonale

A quelles questions répond cette fiche ?

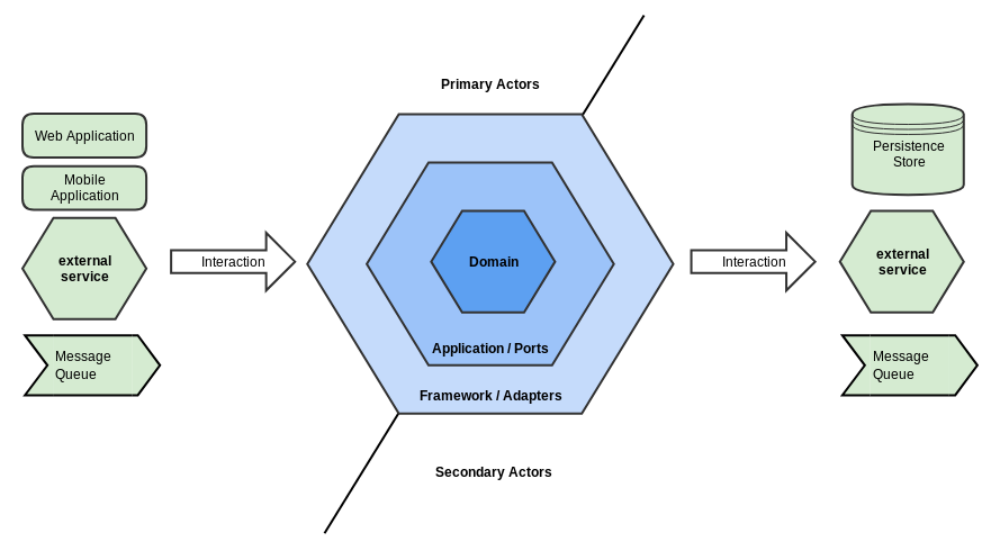
Qu'est-ce que l'architecture hexagonale ?
Quels sont ses différents composants?
Pourquoi utiliser cette architecture ?

Walid

Qu'est-ce que l'architecture hexagonale ?

En 2005, Alistair Cockburn a présenté son approche Ports & Adapters comme une solution pour faire face aux problèmes rencontrés avec les applications traditionnelles à plusieurs niveaux, tels que le couplage et l'enchevêtrement du code. L'objectif était de fournir une isolation entre le domaine et ses consommateurs, et d'améliorer la maintenabilité globale du code du domaine.

Avec l'approche de l'architecture hexagonale, nous isolons la logique principale de l'application de ses interactions avec le monde extérieur. Cette structure permet de prendre en charge plusieurs types de clients (par exemple, navigateur web, applications mobiles, etc.) tout en fournissant une abstraction entre la logique principale de l'application et ses services externes sur lesquels elle dépend. Cette abstraction empêche les détails spécifiques des clients et des services externes de se propager dans le domaine.
Alistair Cockburn a choisi la forme hexagonale comme représentation graphique de l'application, d'où le nom d'architecture hexagonale.



Quels sont ses différents composants?

À l'intérieur de l'hexagone, nous définissons la logique métier centrale de l'application. Historiquement, le schéma ne dicte pas le contenu de l'hexagone, mais par convention, il est souvent représenté par trois anneaux concentriques d'hexagones :

- Cadre (Framework)
- Application
- Domaine

La logique métier du domaine au cœur de l'hexagone est défini sans tenir compte d'une technologie ou d'un framework particulier. En maintenant la pureté de la logique métier, nous évitons de nous coupler à une technologie spécifique.

L'isolation offerte par cette approche assure une grande pérennité à la logique centrale en empêchant tout mécanisme spécifique au framework d'influencer cette logique.
Autour de la logique du domaine se trouve un ensemble de ports qui gèrent tous les accès au modèle de domaine.

Ces ports fournissent une couche d'abstraction autour du domaine et définissent l'API de l'application.
Autour de la couche des ports/de l'application se trouve un ensemble d'adaptateurs qui facilitent l'accès aux ports de l'application depuis le monde extérieur, y compris tout framework de soutien. Ces adaptateurs fournissent une couche d'abstraction entre les acteurs principaux qui agissent sur la logique métier et les ports de l'application qui agissent sur les acteurs secondaires.

Les ports

Les ports sont les interfaces définies par l'hexagone pour interagir avec le monde extérieur. Il y a deux types de ports : les ports permettant d'appeler des fonctions à l'intérieur de l'hexagone et les ports permettant à l'hexagone d'appeler des éléments extérieurs. De plus, il est important de souligner que les ports (d'entrée ou de sortie) sont définis à l'intérieur de l'hexagone.

- **Les ports PRIMAIRES**

Les ports primaires sont la frontière entre le domaine et ses clients externes. Ces ports permettent une séparation logique entre le domaine et la technologie utilisée pour la communication (par exemple, HTTP/REST, bus de messages, etc.). Ils visent à éviter tout couplage entre le domaine et une technologie spécifique.

- **Les ports SECONDAIRES**

Du côté sortant du domaine se trouvent les ports secondaires qui permettent au domaine d'interfacer son trafic sortant avec le monde extérieur. Ces ports constituent l'interface avec laquelle le domaine communique avec le monde extérieur. Ils ont également pour objectif de désolidariser le domaine de toute technologie spécifique.

Les adaptateurs

Les adaptateurs fournissent le mécanisme par lequel le monde extérieur communique avec les Ports de l'hexagone. Contrairement aux Ports, les adaptateurs sont spécifiques à la technologie. Chaque adaptateur est responsable de faire le lien entre l'API de port abstraite et le monde réel, en effectuant toutes les traductions nécessaires.

- **Les adaptateurs PRIMAIRES**

Du côté de l'adaptateur primaire, chaque adaptateur doit traduire la requête spécifique à la technologie du côté client en une commande appropriée indépendante de la technologie. Des exemples courants d'adaptateurs primaires incluent des adaptateurs HTTP/REST qui extraient les données de la requête HTTP spécifique à la technologie, ou un adaptateur client pour un bus de messages.

- **Les adaptateurs SECONDAIRES**

L'adaptateur secondaire est un adaptateur sortant qui fournit une implémentation concrète de l'interface du port secondaire. Dans ce cas, le message sortant du domaine qui est indépendant de la technologie est converti en une forme pouvant être utilisée pour communiquer avec le monde extérieur.

Le domaine

La logique fonctionnelle, ou le domaine, doit être contenue à l'intérieur de l'hexagone. L'implémentation de ce domaine ne doit pas faire référence directement à des éléments externes. L'objectif est d'isoler le domaine de l'extérieur et d'établir des interfaces d'entrée et de sortie claires. L'hexagone symbolise la séparation stricte entre le code métier et le code technique.
Ainsi, le domaine doit exprimer ses besoins, tels que les éléments de configuration ou les données d'entrée, à l'aide d'interfaces sans faire d'appels directs vers l'extérieur. Ce sont des objets externes qui fournissent au domaine les données dont il a besoin pour effectuer les traitements fonctionnels.
Idéalement, les technologies utilisées à l'intérieur de l'hexagone ne doivent pas avoir une influence excessive sur l'implémentation, de manière à ce que le code métier ne dépende pas de la technologie. L'objectif est de réduire l'impact des changements technologiques, même majeurs, sur l'implémentation de la logique fonctionnelle à l'intérieur de l'hexagone. Ainsi, le domaine est préservé.

Pourquoi utiliser cette architecture ?

La testabilité

Lorsqu'Alistair Cockburn a développé l'architecture hexagonale en 2005, l'un de ses objectifs était de créer une application testable. Cette architecture répond à ce besoin en permettant d'isoler le modèle et de définir clairement toutes ses entrées et sorties. De plus, l'injection de dépendances permet de contrôler les objets utilisés par le modèle pour interagir avec l'extérieur. Ainsi, il est facile d'injecter des "mocks" d'adaptateurs et de tester le domaine de manière exhaustive.

Protection du domaine et de la valeur métier

L'un des objectifs de l'architecture hexagonale est de permettre au domaine de se détacher des couches techniques, car celles-ci peuvent évoluer rapidement. Les adaptateurs servent d'intermédiaires entre le modèle et le monde extérieur. Ainsi, ce sont les adaptateurs qui sont modifiés si un élément du monde extérieur le demande, tandis que le domaine n'est modifié que si la logique métier l'exige. Cela permet de préserver l'intégrité de l'hexagone, en le préservant des changements techniques.

Exemple d'utilisation chez l'un de nos clients

Changement du moteur de base de données sans toucher au domaine

Au début du développement de notre application on a choisi MySQL comme moteur de base de données, vu que le modèle des données était fortement relationnel.
Après un certain temps, le modèle des données a changé et ça a commencé à tendre vers un schéma plutôt non relationnel. Donc il fallait faire la bascule vers un autre moteur qui est MongoDB.
Grâce à l'architecture hexagonale, cette transition se fait de la manière la plus naturelle et la plus transparente possible.

En commençant par implémenter un autre adapter pour MongoDB (le seul composant de l'application responsable de communiquer avec une base MongoDB). Et puis instancié notre service en lui passant le nouveau adapter. Sans oublier bien sûr la migration des données d'une base à une autre.

```python
from abc import ABC, abstractmethod

# Abstract UserRepositoryPort
class UserRepositoryPort(ABC):
    @abstractmethod
    def get_user_by_id(self, user_id):
        pass
```

```python
# MongoDB Adapter
class MongoDBAdapter(UserRepositoryPort):
    def __init__(self, connection_string):
        self.connection_string = connection_string

    def get_user_by_id(self, user_id):
        # MongoDB-specific code to retrieve user data
        # ...

# MySQL Adapter
class MySQLAdapter(UserRepositoryPort):
    def __init__(self, connection_string):
        self.connection_string = connection_string

    def get_user_by_id(self, user_id):
        # MySQL-specific code to retrieve user data
        # ...
```

```python
# User Service
class UserService:
    def __init__(self, repository):
        self.repository = repository

    def get_user_by_id(self, user_id):
        user_data = self.repository.get_user_by_id(user_id)
        return user_data
```