

## DDD sans douleur

Optimiser les coûts d'une architecture complexe avec API Platform



L'abécédaire du digital pour  
les métiers de la santé en libéral

## David Merle

Lead Dev chez CBA Informatique Libérale



@\_Merle\_David\_



mdavid-dev

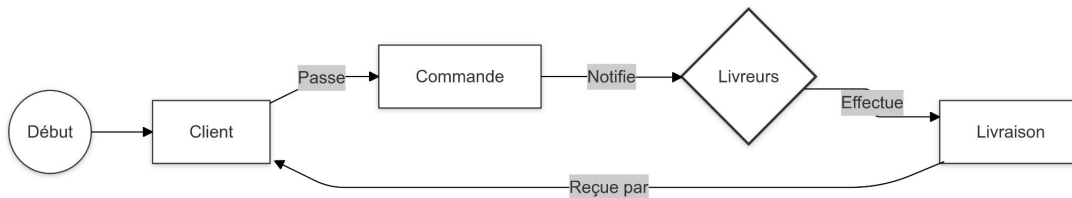
# Introduction



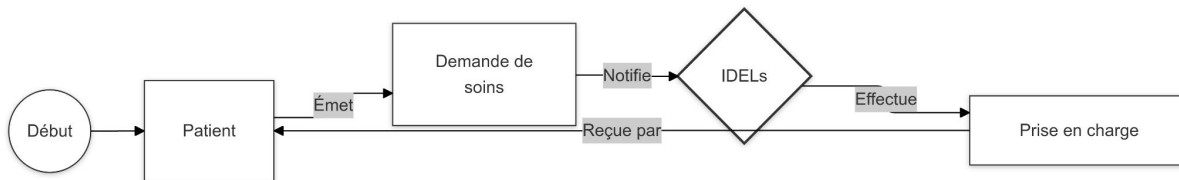
# Contexte: Le cas Opaline

- Leader du marché français 33k IDELs clients

- Uber Eats :



- Opaline Santé:



- Opaline: 70-90k visiteurs uniques/mois
- Couverture nationale + DOM

# Le défi technique

- Contexte métier parfois complexe
- Besoin de flexibilité pour les évolutions
- Maintenir une dette technique maîtrisée
- Préserver la vélocité de développement sur le long terme

# Aperçu de la présentation

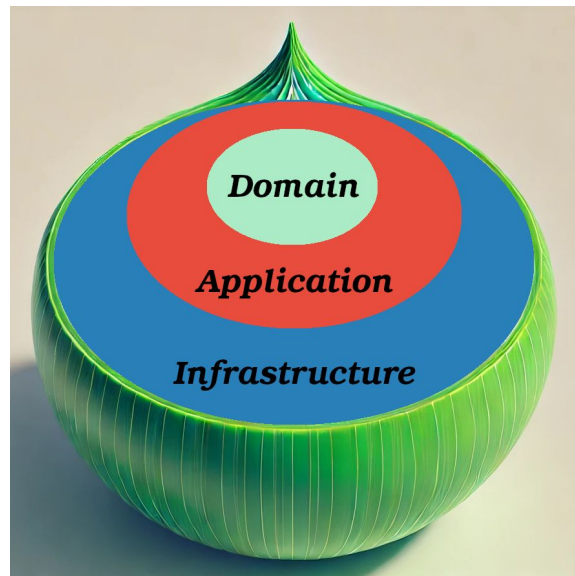
- Introduction à la DDD
- Solutions mises en place:
  - MakerBundle DDD
  - Infrastructure de tests
- Retours d'expériences et bonnes pratiques

# La DDD dans notre contexte



# Introduction à la DDD

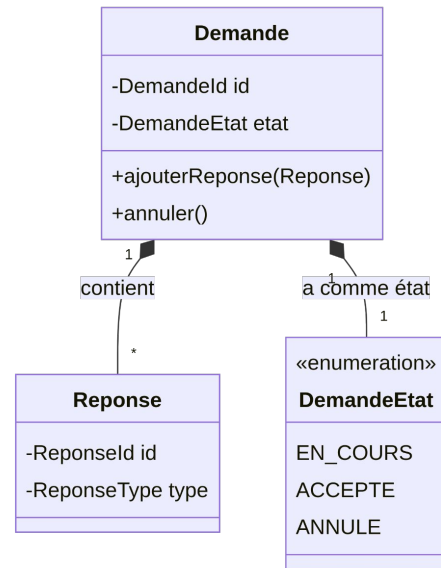
- DDD ou Domain-Driven Design
  - Le métier au coeur de l'application
- “La DDD n’est pas normative”
- Architecture en couches (Oignon)
- Pour aller plus loin sur les principes de la DDD?
  - Présentations de Mathias ARLAUD et Robin CHALAS



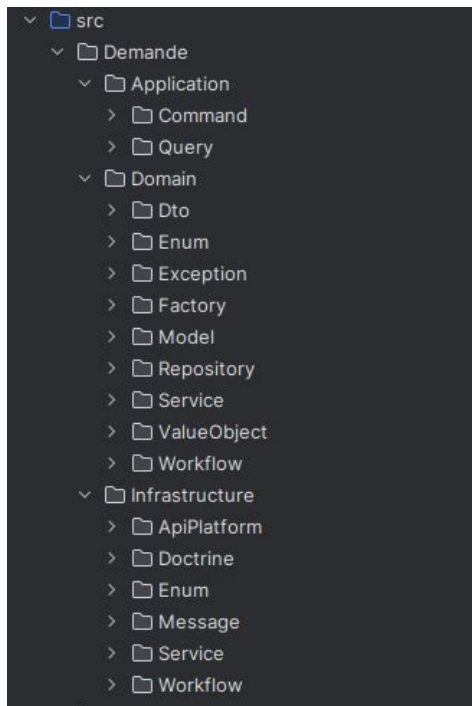


# Les patterns clés utilisés

- Aggregates
- Séparation Entity/Resource pour l'API
- Value Objects dans les Entities

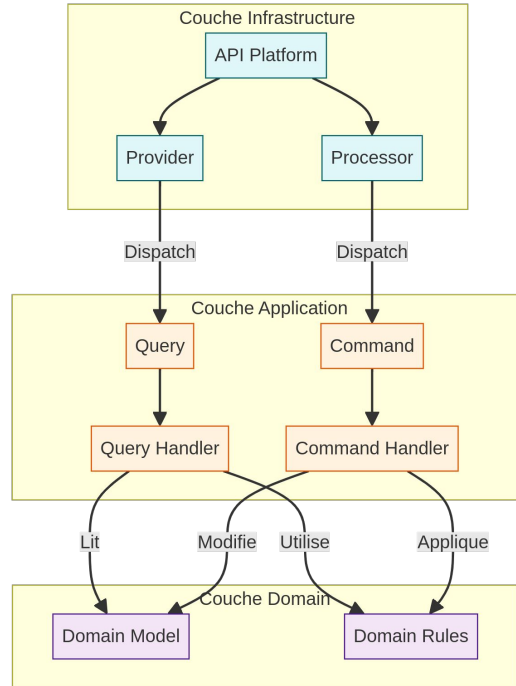


# Structure concrète

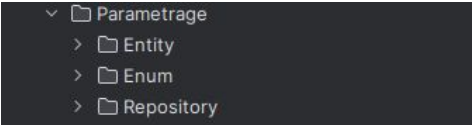


- Application / Domain / Infrastructure
- Flux de communication
  - Base de CQRS (Command Query Responsibility Segregation)
  - Infrastructure (Provider/Processor) -> Dispatch -> Application  
(Command/Query) -> Domain

# Structure concrète



# Structure concrète



```
▼ Parametrage
  > Entity
  > Enum
  > Repository
```

- La DDD n'est pas toujours nécessaire et on peut faire de la RAD (Rapid application development).

# Les défis humains

- Les résistances initiales
  - Nombre important de fichiers
  - Découpage et développement initial conséquent
- Les solutions
  - MakerBundle DDD et accompagnement de l'équipe
- Les moments de validation
  - Facilité des changements fonctionnels
  - Tests unitaires simplifiés
  - Peu d'anomalies

# MakerBundle DDD



# MakerBundle DDD?

- Philosophie: s'appuie sur MakerBundle Classique
- Objectif: gagner du temps sur les parties redondantes
- Focus sur le code métier
- Adaptabilité: DDD vs RAD selon la complexité

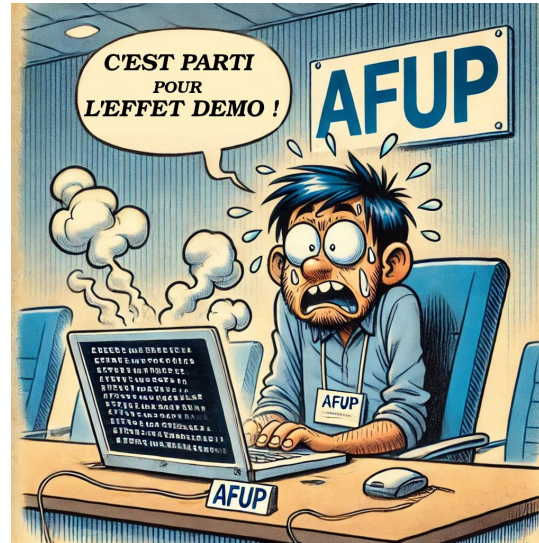
# Vue d'ensemble des fonctionnalités

- Feature complète
  - Structure de dossiers DDD ou RAD
  - Configuration doctrine.yaml
  - Configuration api\_platform.yaml
- Composants individuels
  - Couche Application: Command/Query
  - Couche Infrastructure: Provider/Processor



# Démonstration

- Live coding



# Infrastructure de tests



# Entities InMemory: La base

- Concept:
  - Jeux de données prêts à l'emploi
  - Cas concrets du métier
- Exemple: Pour une entité Demande
  - DemandeEnCoursInMemory
  - DemandeAccepteeInMemory
  - DemandeRefuseeInMemory

# Entities InMemory: La base

```
final readonly class DemandeInMemory
{
    public static function demandeEnCours(): Demande
    {
        return new Demande(
            etat: new DemandeEtat( value: DemandeEtatEnum::EN_COURS),
        );
    }

    public static function demandeAcceptee(): Demande
    {
        return new Demande(
            etat: new DemandeEtat( value: DemandeEtatEnum::ACCEPTE),
        );
    }

    public static function demandeRefusee(): Demande
    {
        return new Demande(
            etat: new DemandeEtat( value: DemandeEtatEnum::REFUSE),
        );
    }
}
```

# Tests Unitaires

- Test du Domaine
- Utilisation direct des Entities InMemory
- Focus sur la logique métier

# Tests Unitaires

```
class DemandeEtatTest extends TestCase
{
    private Demande $demandeEnCours;
    private Demande $demandeAcceptee;
    private Demande $demandeRefusee;

    // TODO: ici écrire les différents tests que l'on veut réaliser

    protected function setUp(): void
    {
        parent::setUp();
        // Ici on définit des jeux de données que l'on va pouvoir réutiliser sur chaque test
        $this->demandeEnCours = DemandeInMemory::DemandeEnCours();
        $this->demandeAcceptee = DemandeInMemory::DemandeAcceptee();
        $this->demandeRefusee = DemandeInMemory::DemandeRefusee();
    }
}
```

# Tests d'Intégration

- Test de l'API
- Base de données dédiée
- Reset entre chaque test
- Réutilisation des mêmes Entities InMemory
- Maîtrise totale du contexte

# Extension de l'ApiTestCase d'API Platform

- Extension de l'ApiTestCase natif
- Fonctionnalités ajoutées :
  - Gestion BDD de test
  - Méthodes HTTP simplifiées
  - Assertions personnalisées



```

final class DemandeApiTest extends CbaApiTestCase
{
    private Demande $demandeEnCours;
    private Demande $demandeAcceptee;
    private Demande $demandeRefusee;

    public function testGetCollection(): void
    {
        $this->generateDatabaseRecords(
            entities: [
                $this->demandeEnCours,
                $this->demandeAcceptee,
                $this->demandeRefusee
            ]
        );

        $demandes = $this->get(
            url: '/api/demandes',
            token: $this->getToken()
        );

        $this->assertCount( expectedCount: 3, $demandes->toArray()['hydra:member']);
    }

    protected function setUp(): void
    {
        parent::setUp();
        $this->demandeEnCours = DemandeInMemory::demandeEnCours();
        $this->demandeAcceptee = DemandeInMemory::demandeAcceptee();
        $this->demandeRefusee = DemandeInMemory::demandeRefusee();
    }
}

```

En conclusion...



# Points clés

- Coût initial vs Bénéfices long terme
- API Platform comme accélérateur
  - Intégration de la DDD facilitée
  - MakerBundle DDD
  - Infrastructure de tests

# Les leçons du terrain

- Écouter l'équipe
- Transformer les frustrations en solutions
- Exemples concrets :
  - Trop de fichiers → MakerBundle DDD
  - Tests complexes → Infrastructure de tests

# Pour aller plus loin

- Philosophie applicable à d'autres architectures :
  - Hexagonale
  - Clean Architecture
- Conseils pratiques :
  - Automatiser les tâches répétitives
  - Adapter l'architecture au besoin
  - Outiller son équipe/projet

“Merci !”