Rétrospective PoC architecture

Exploration MVP, ValueObject, CQRS...

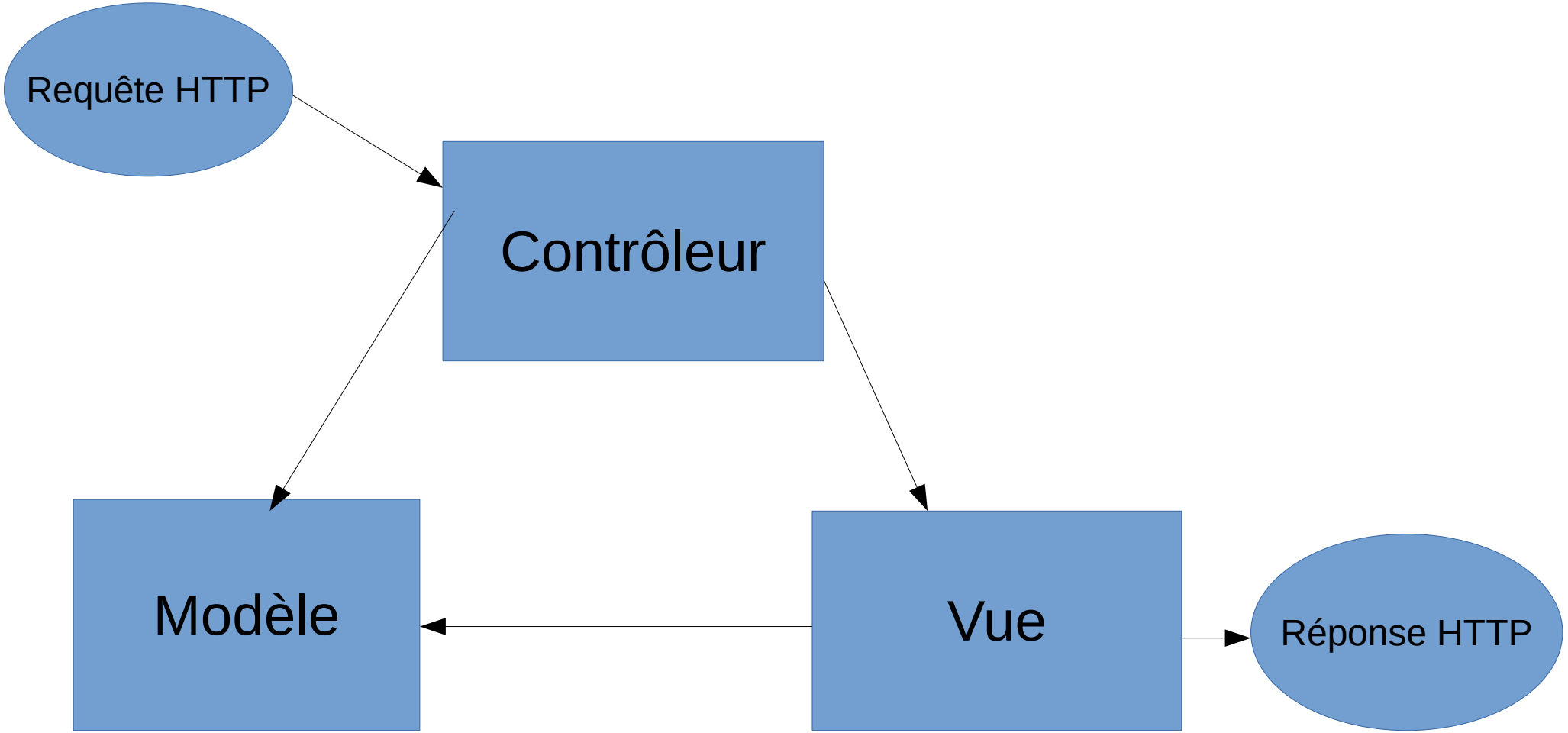
Introduction

- Pas un cours magistral
- Simple aperçu de concepts d’architecture
- PoC application stateless
- Tests de technologies

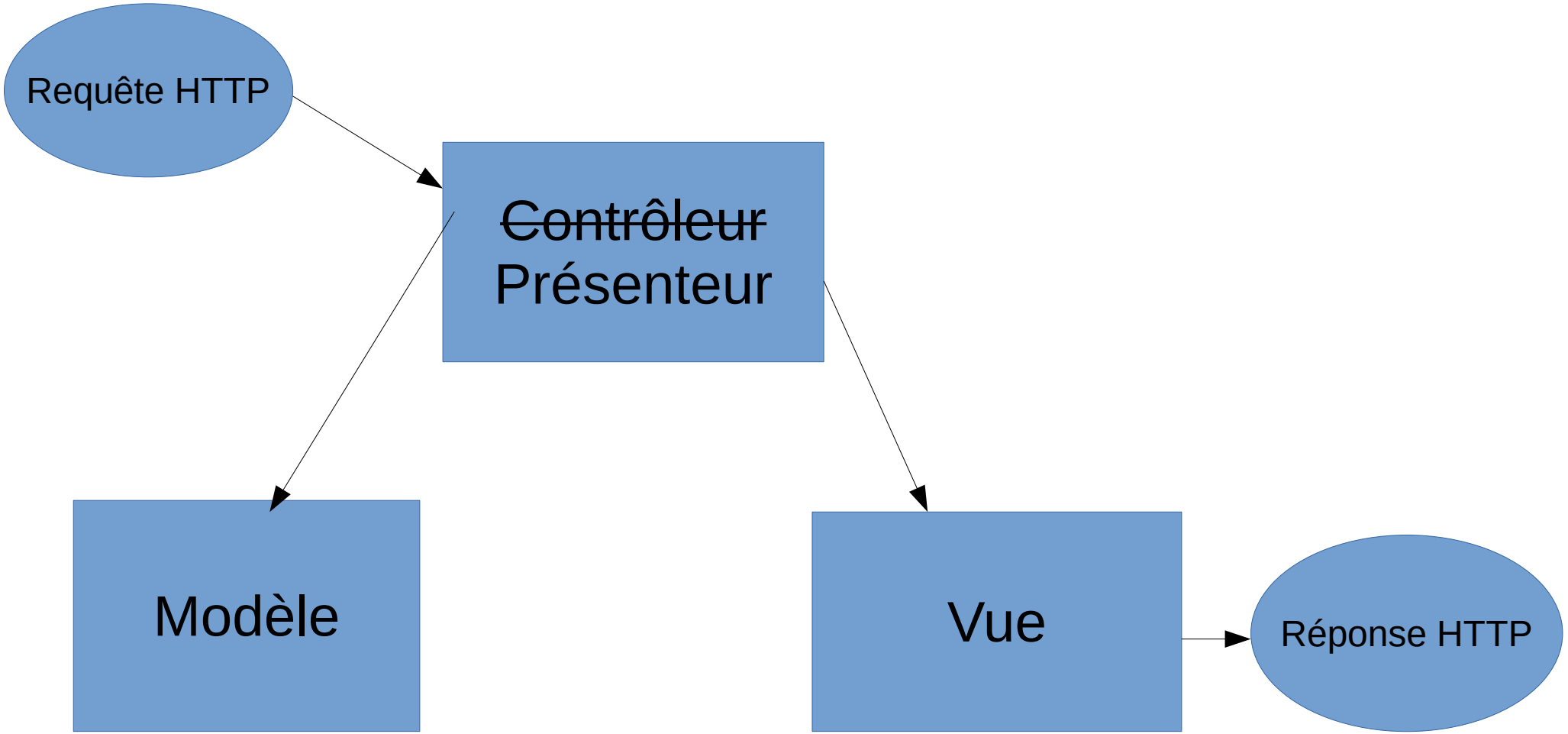
Sommaire

- Architecture MVP
- ValueObject
- Immutabilité et flow typesafe
- CQRS
- Architecture hexagonale
- Workerman et SSE

Architecture MVC



Architecture MVP



Architecture MVP

- Le présenteur a le rôle du Contrôleur
- Le lien entre vue et modèle est supprimé
- Le présenteur formate la donnée du modèle pour la vue
- Données sont formatées avant d’être passé à la vue
- La vue n’a pas accès aux entités

Exemple MVC/MVP

- Affiche les messages d’un chat
- Un liste de messages est donc chargée
- La liste des auteurs aussi

Exemple MVC - Contrôleur

```php
/**
 * @param ShowChatRequest $request
 */
Vincent QUATREVIEUX *
public function handle(object $request): object
{
    $messages = $this->repository->all();
    $usersIds = [];

    foreach ($messages as $message) {
        if ($message->userId !== null) {
            $usersIds[$message->userId->value] = $message->userId->value;
        }
    }

    $users = $this->userRepository->findAllById(array_values($usersIds));

    return $this->render('chat/show', [
        'messages' => $messages,
        'users' => $users,
        'me' => $request->user,
    ]);
}
```

Exemple MVC - Vue

```
<div id="chat">
    <h1>Chat</h1>

    <section
        class="messages"
        data-source="<?= $renderer->url(ShowChatRequest::ajax()) ?>"
        data-events="<?= $renderer->url(new SubscribeEventRequest()) ?>"
        data-pulling-delay="60000"
    >
        <?php foreach($this->messages as $message): ?>
            <div class="message <?= $this->me && $message->userId?->id->value === $this->me->id->value ? 'from-me' : 'from-other' ?>">
                <span class="author">
                    <?php $user = $message->userId ? $this->users[$message->userId->value] ?? null : null; ?>
                    <?= $user?->html() ?? 'Deleted' ?>
                </span>
                <span class="date"><?= $message->createdAt->format('d/m/Y H:i:s') ?></span>
                <span class="content"><?= $message->message->html() ?></span>
            </div>
        <?php endforeach; ?>
    </section>

    <?php if ($renderer->hasAccess($sendMessage = new SendMessageRequest())): ?>
        <section class="input">
            <form action="<?= $renderer->url($sendMessage) ?>" method="post">
                <input type="text" name="message" placeholder="Message" autofocus />
                <input type="submit" value="Envoyer" />
            </form>
        </section>
    <?php endif ?>
</div>
```

Exemple MVP - Présenteur

Exemple MVP – Modèle de vue

```php
5 usages  Vincent Quatrevieux
class ShowChatResponse
{
    1 usage  Vincent Quatrevieux
    public function __construct(
        /**
         * @var ChatMessageWithUser[]
         */
        public readonly array $messages,
        public readonly bool $ajax = false,
    ) {
    }
}
```

```php
2 usages  Vincent QUATREVIEUX
class ChatMessageWithUser
{
    1 usage  Vincent QUATREVIEUX
    public function __construct(
        public readonly ChatMessageId $id,
        public readonly MessageContent $message,
        public readonly DateTimeImmutable $createdAt,
        public readonly bool $isMine,
        public readonly ?Pseudo $pseudo,
    ) {
    }
}
```

Exemple MVP - Vue

```php
<div id="chat">
    <h1>Chat</h1>

    <section
        class="messages"
        data-source="<?= $renderer->url(ShowChatRequest::ajax()) ?>"
        data-events="<?= $renderer->url(new SubscribeEventRequest()) ?>"
        data-pulling-delay="60000"
    >
        <?php foreach($this->messages as $message): ?>
            <div class="message <?= $message->isMine ? 'from-me' : 'from-other' ?>">
                <span class="author"><?= $message->pseudo?->html() ?? 'Deleted' ?></span>
                <span class="date"><?= $message->createdAt->format('d/m/Y H:i:s') ?></span>
                <span class="content"><?= $message->message->html() ?></span>
            </div>
        <?php endforeach; ?>
    </section>

    <?php if ($renderer->hasAccess($sendMessage = new SendMessageRequest())): ?>
        <section class="input">
            <form action="<?= $renderer->url($sendMessage) ?>" method="post">
                <input type="text" name="message" placeholder="Message" autofocus />
                <input type="submit" value="Envoyer" />
            </form>
        </section>
    <?php endif ?>
</div>
```

Avantages MVP

- Vue plus simple : les données sont préformatées pour la vue
- Indépendance métier et vue
- Évolutivité : une modification du modèle n’implique pas de modification de la vue
- Cohérence de la donnée grâce au typage

Inconvénients MVP

- Plus de classes :au moins une classe de réponse par contrôleur
- Performance : donnée dupliquée
- Plus de code dans le contrôleur

ValueObject

- Un objet immutable
- Contient une valeur primitive
- Valeur validée au constructeur
- Toutes les propriétés des entités doivent être un value object
- Ajoute une sémantique à la valeur

ValueObject - Exemple

```php
final class UserId implements ValueObjectInterface, JsonSerializable
{
    Vincent Quatrevieux *
    public function __construct(
        public readonly int $value,
    )
    {
        if ($value < 1) {
            throw new InvalidValueException(self::class, 'User id must be greater than 0');
        }
    }

    no usages  Vincent Quatrevieux
    public function jsonSerialize(): int
    {
        return $this->value;
    }

    Vincent Quatrevieux
    public function value(): int
    {
        return $this->value;
    }

    Vincent Quatrevieux *
    public function __toString(): string
    {
        return (string) $this->value;
    }

    Vincent Quatrevieux *
    public static function from(mixed $value): static
    {
        if (!is_int($value)) {
            throw new InvalidPrimitiveTypeError(self::class, 'int', $value);
        }

        return new static($value);
    }

    Vincent Quatrevieux *
    public static function tryFrom(mixed $value): ?static
    {
        if (!is_int($value)) {
            return null;
        }

        try {
            return new static($value);
        } catch (ValueObjectException) {
            return null;
        }
    }
}
```

ValueObject - Entité

```php
16 usages  Vincent QUATREVIEUX
final class ChatMessage
{
    2 usages  Vincent QUATREVIEUX
    public function __construct(
        public readonly ChatMessageId $id,
        public readonly MessageContent $message,
        public readonly ?UserId $userId,
        public readonly DateTimeImmutable $createdAt,
    ) {
    }
}
```

ValueObject - Utilité

- Cohérence des données garantie (ex : un email est forcément un email)
- Le type reflète la donnée (ex : un UserId vient de l'entité User)
- Impossibilité de jongler entre 2 données incompatibles
- Helper method directement sur le value object

ValueObject - Utilisation

- Ajoute contrainte fonctionnelle et pas métier
- La valeur doit être validée avant de créer le VO
- Pas de try/catch (VO ne remplace pas un form)
- Pas de VO sur les form
- Faut-il en utiliser niveau paramètre repository ? (ex : findById prend UserId ou int ?)

ValueObject - Inconvénients

- Prise en compte sur l'ORM
- Impact sur les performances
- Fait planter l'application si db corrompue
- Plus de code pour déclarer les VO et convertir les primitives en VO

Entités immutables

- Toutes les propriétés sont readonly
- Pas d'objet « partiels » : toutes les propriétés non optionnelles ont une valeur
- Toutes modifications retournent une nouvelle instance
- Les données « incomplètes » ou décrivant un état doivent avoir leur propre classe
- Méthodes pour passer d'un « état » à un autre

Immutabilité – Création User

- Classe UserCreation avec les données sans id
- Méthode created qui prend id en paramètre
- Méthode created appelée par repository après insert
- User représente un utilisateur valide en base

Immutabilité - Authentification

- Classe AuthenticatedUser qui représente le user authentifié
- Méthode User::authenticate()
- AuthenticatedUser garde que les données pertinentes

Immutability - Modification

- Classe ModifiedUser qui représente un User modifié par encore sauvegardé
- Méthodes withXXX qui créent une nouvelle instance avec nouvelle valeur
- Ces méthodes gardent un trace des propriétés modifiées
- Repository lit cette trace pour faire l'update + appelle save() qui retourne un User valide

Immutabilité – Avantages

- Limite les effets de bords
- Typage empêche entité invalide dans métier
- Flow clair grâce aux nom de classes
- Entités plus simple (pas besoin de getter ou setter grâce à readonly)

Immutabilité - Inconvénients

- Léger impact sur les performances
- Compatibilité avec ORM et form
- Maintenir les différentes classes de flow : leurs propriétés doivent rester cohérentes entres elles

CQRS - Introduction

- Séparation entre lecture (query) et écriture (command)
- Nécessite pas un bus
- Simple séparation des interface pour lire et écrire suffit
- Permet d’avoir un backend différent pour l’écriture et lecture (ex : elasticsearch en lecture)

CQRS - Query

- Pas de nécessité à faire des classes de query
- Simple interface avec liste des méthodes de lecture
- Implémentation du dépôt de lecture peut être séparé de celui d'écriture (non obligatoire)
- Utiliser UNIQUEMENT l'interface niveau métier et contrôleurs

CQRS - Command

- Toutes les écritures doivent passer par une commande
- Une interface pour écriture sur dépôt doit être créée
- Une classe par commande
- Seul les handlers ont accès au dépôt d'écriture
- Données doivent être validées avant de créer la commande
- Handler doit pouvoir retourner un résultat (cas contrainte unique)

CQRS - Avantages

- Évolution vers SQL+NoSQL simple
- Actions métiers visibles et claire avec commands
- Gestion de la donnée cachée au métier
- Simplification du contrôleur / limite de besoin de classes de « service »
- Facile à migrer vers ce modèle

CQRS - Inconvénients

- Nouvelle façon de faire
- Commandes peuvent rendre la navigation dans le code plus complexe
- Gestion des erreurs en asynchrone