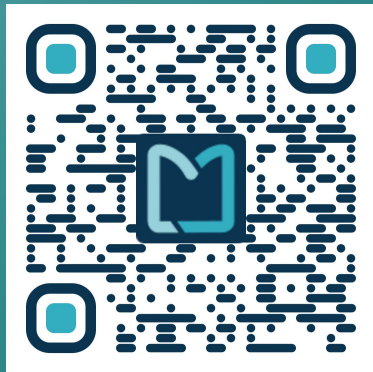


De CRUD à CQRS

Guillaume Mary - www.codefilarete.org



- CRUD
 - Définition
 - Quand l'éviter
- CQRS
 - Définition
 - Mise en oeuvre
- Démo de migration
- Conclusion



CRUD



CRUD: définition

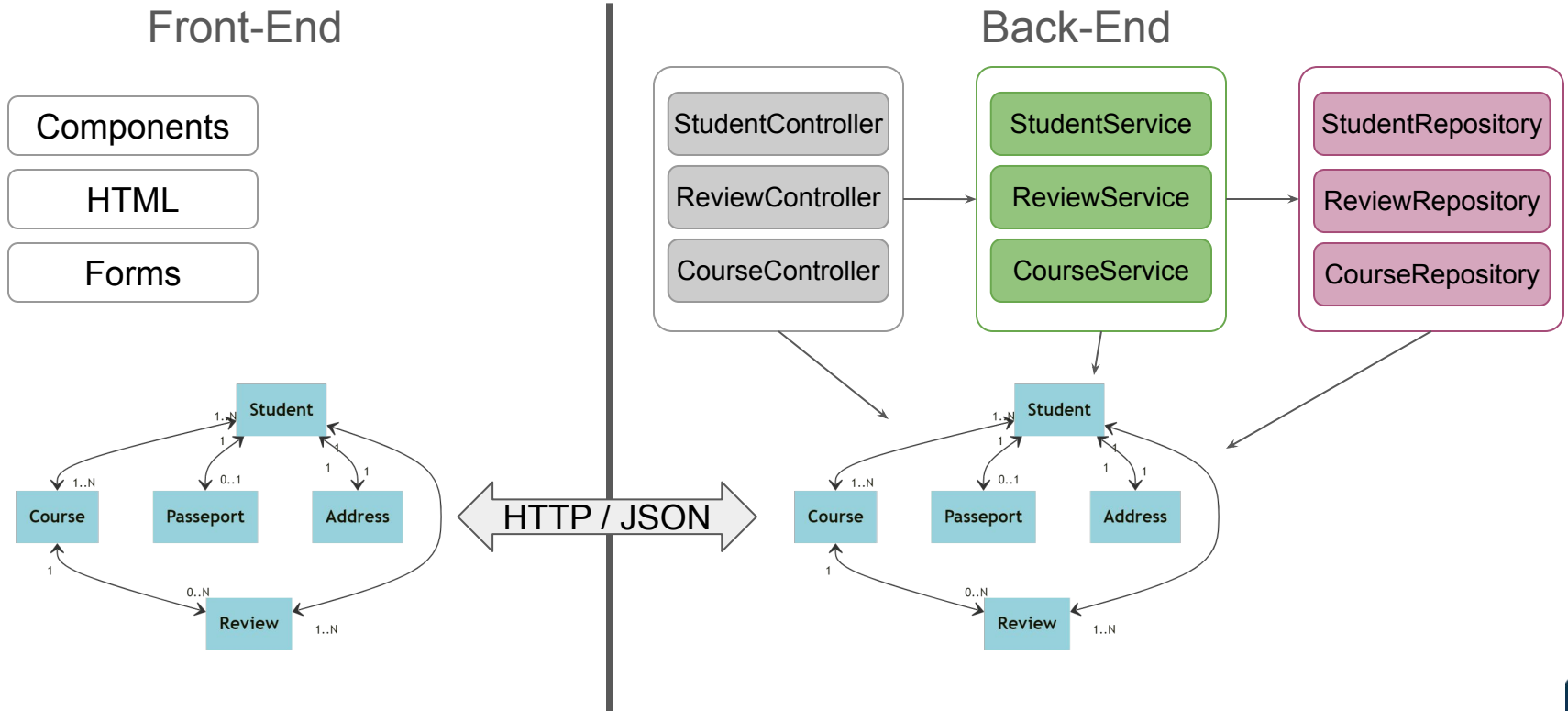
- Create Read Update Delete
- Accès aux entités de la base de données au travers de REST

CRUD	HTTP	SQL
Create	PUT / POST	INSERT
Read	GET	SELECT
Update	PUT / PATCH	UPDATE
Delete	DELETE	DELETE

[Create, read, update and delete - Wikipedia](#)



CRUD: architecture classique



CRUD: quel est le problème ?



Pousse trop de données au Front-end, charge trop de choses

- impact sur la mémoire
- impact sur le réseau



A tendance à créer des God classes / God entity graph



A tendance à créer des Services orientés Entités (avec trop de dépendance)



L'ajout d'une fonctionnalité peut en impacter une autre



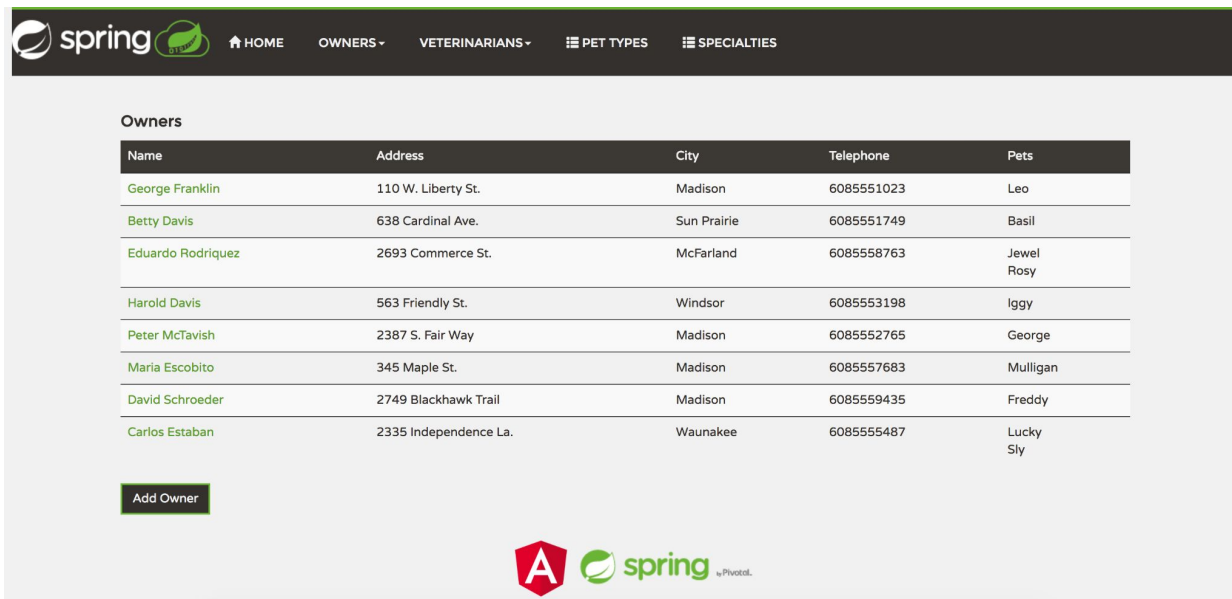
Le Front-End compose la fonctionnalité par des appels REST



CRUD: exemple avec Spring Pet Clinic

Back-end: <https://github.com/spring-petclinic/spring-petclinic-rest>

Front-end: <https://github.com/spring-petclinic/spring-petclinic-angular>



The screenshot displays the Spring Pet Clinic web application. At the top is a navigation bar with the Spring logo and menu items: HOME, OWNERS, VETERINARIANS, PET TYPES, and SPECIALTIES. Below the navigation bar, the 'Owners' section is active, showing a table of pet owners. The table has five columns: Name, Address, City, Telephone, and Pets. There are eight rows of data. Below the table is a button labeled 'Add Owner'. At the bottom of the page, there is a footer with the Angular logo, the Spring logo, and the text 'aPivotal®'.

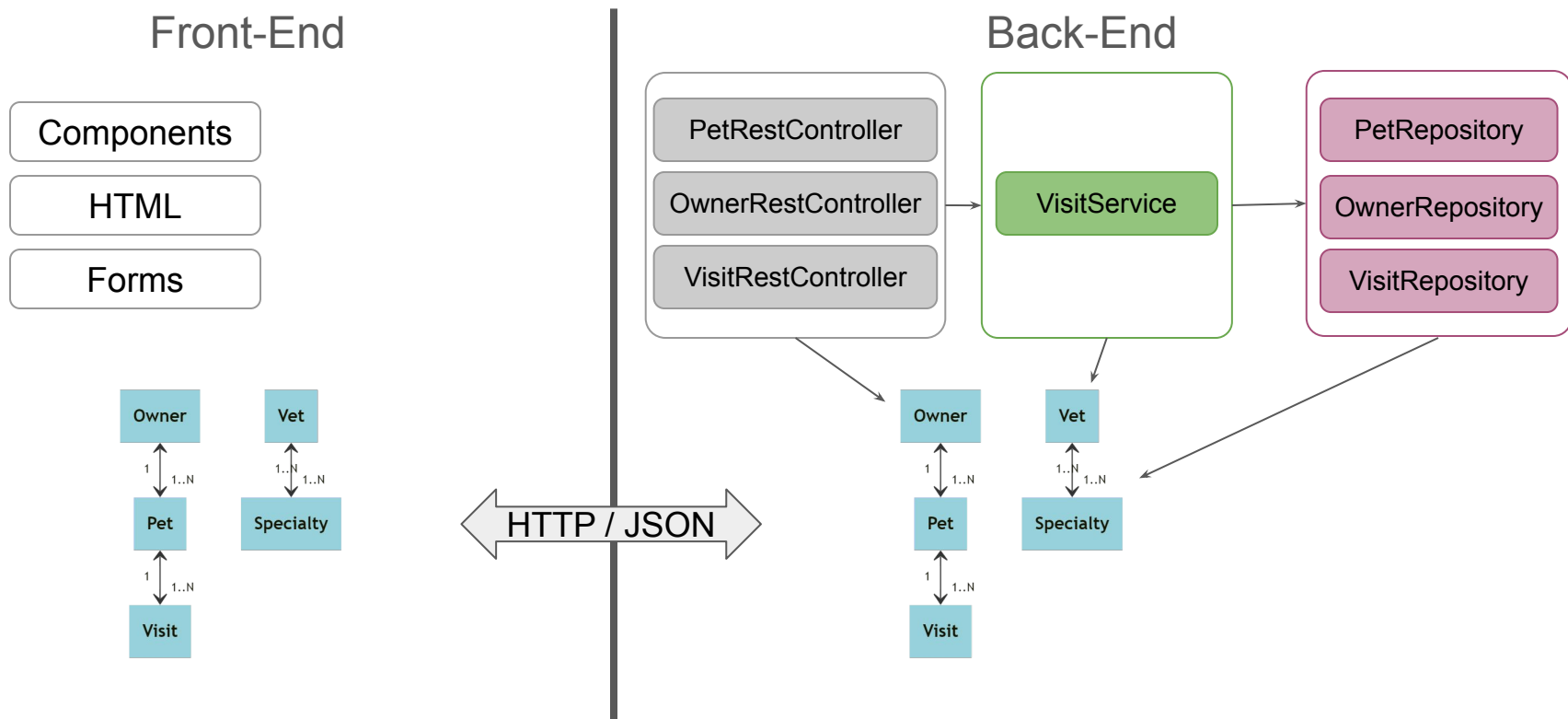
Name	Address	City	Telephone	Pets
George Franklin	110 W. Liberty St.	Madison	6085551023	Leo
Betty Davis	638 Cardinal Ave.	Sun Prairie	6085551749	Basil
Eduardo Rodriguez	2693 Commerce St.	McFarland	6085558763	Jewel Rosy
Harold Davis	563 Friendly St.	Windsor	6085553198	Iggy
Peter McTavish	2387 S. Fair Way	Madison	6085552765	George
Maria Escobito	345 Maple St.	Madison	6085557683	Mulligan
David Schroeder	2749 Blackhawk Trail	Madison	6085559435	Freddy
Carlos Estaban	2335 Independence La.	Waunakee	6085555487	Lucky Sly

Add Owner

Angular Spring aPivotal®



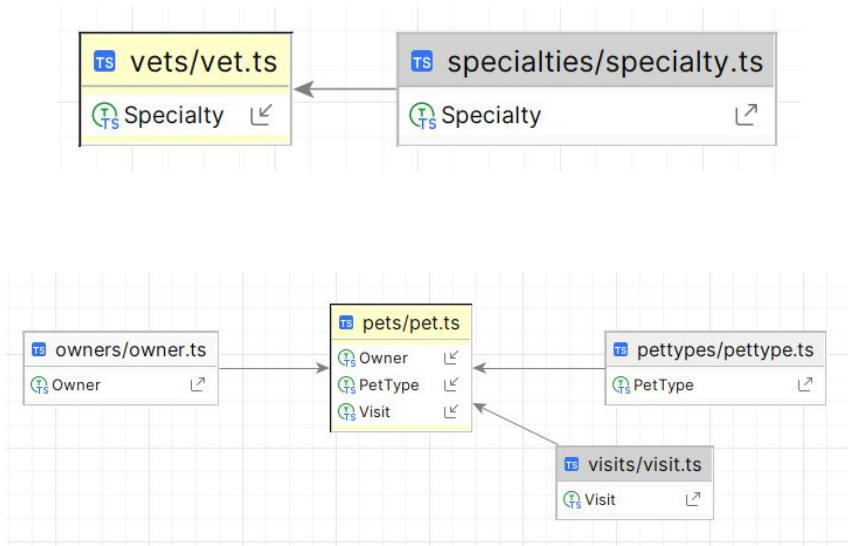
CRUD: exemple avec Spring Pet Clinic



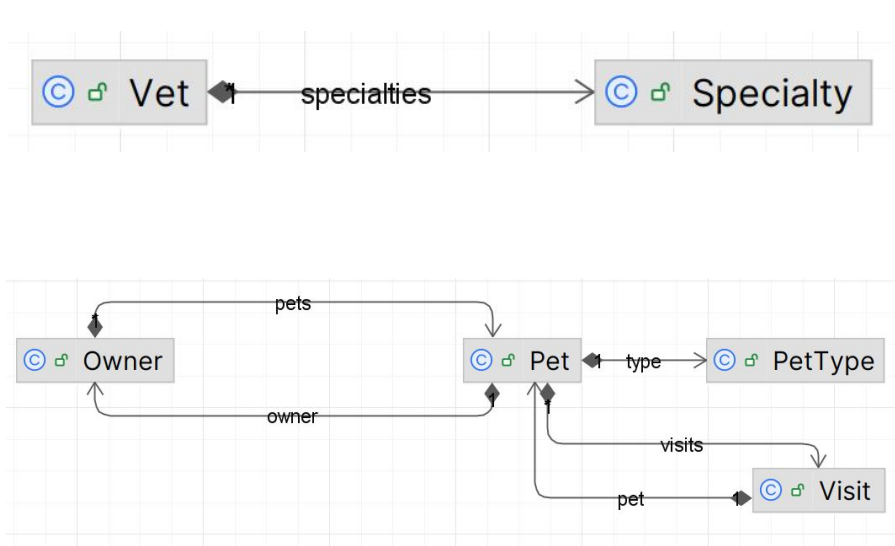
CRUD: exemple avec Spring Pet Clinic

🤔 Le Front-end imite le Back-end, même complexité des 2 côtés

Front-end model



Back-end model



CRUD: exemple avec Spring Pet Clinic

🤔 Le modèle Front-end a trop champs pour certaines fonctionnalités

pet-list.component.html

```
<dl class="dl-horizontal">
  <dt>Name</dt>
```

owner-list.component.html

```
<tbody>
<tr *ngFor="let owner of owners">
<td class="ownerFullName"><a routerLink="/owners/{{owner.id}}" routerLinkActive="active"
                                (click)="onSelect(owner)">{{ owner.firstName }} {{ owner.lastName }}</a></td>
<td>{{ owner.address }}</td>
<td>{{ owner.city }}</td>
<td>{{ owner.telephone }}</td>
<td>
<tr *ngFor="let pet of owner.pets">
  {{ pet.name }}
</tr>
</td>
</tr>
</tbody>
```



CRUD: exemple avec Spring Pet Clinic

🤔 Le Back-end trop de données pour certaines fonctionnalités

```
@Entity
@Table(name = "owners")
public class Owner extends Person {

    @OneToMany(cascade = CascadeType.ALL, mappedBy = "owner", fetch =
```

```
FetchType.EAGER)
private Set<Visit> visits;

@Entity
@Table(name = "pets")
public class Pet extends NamedEntity {

    @OneToMany(cascade = CascadeType.ALL, mappedBy = "pet", fetch = FetchType.EAGER)
```

```
private Set<Visit> visits;

@Entity
@Table(name = "vets")
public class Vet extends Person {

    @ManyToMany(fetch = FetchType.EAGER)
    private Set<Specialty> specialties;
```



CRUD: pour quel projet ?

👍 Très bien pour des “petits” projets / modèles simples

👾 quelques entités

💔 sans trop de relation

💾 peu de données (surtout au travers des jointures)



CQRS



CQRS: un peu de bibliographie

📖 Command and Query Separation (CQS)

*“Every method should either be a **command** that performs an action, or a **query** that returns data to the caller, **but not both**”*

[Command–query separation - Wikipedia](#)



- 👍 Une Command modifie l'état du système
- 👍 Une Query ne fait que remonter des données

- 😞 Plutôt un Design Pattern qu'un Pattern d'Architecture (car au niveau méthode)
- 😞 Interdit la création d'objet car ne peut pas renvoyer de valeur



CQRS: un peu de bibliographie

📖 Command and Query Responsibility Segregation (CQRS)

“A system architecture that extends the idea behind command–query separation (CQS) to the level of services”

[Command Query Responsibility Segregation - Wikipedia](#)

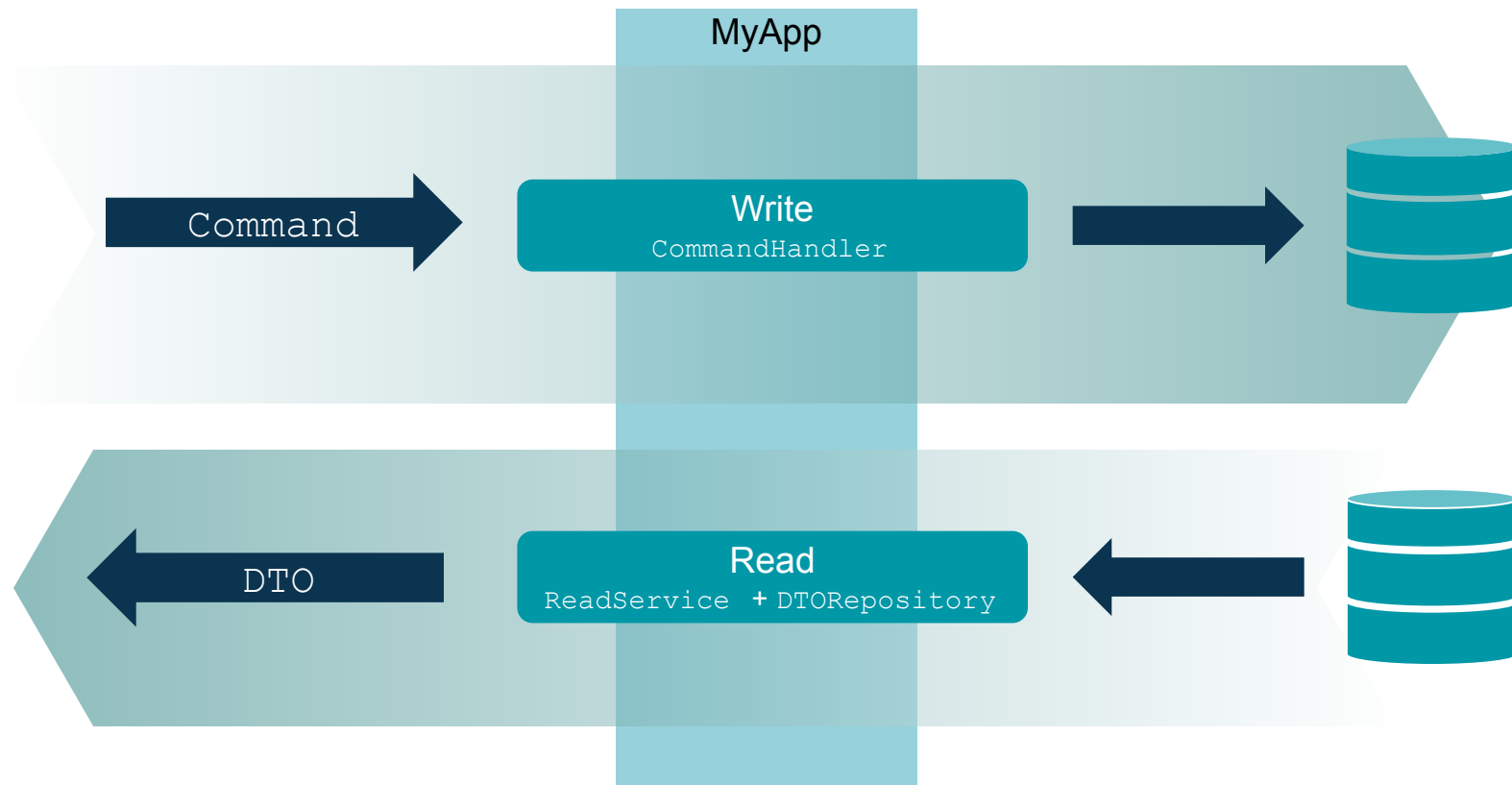


😞 Ne permet toujours pas la création d'objet

👍 Oublions la littérature et autorisons-nous à renvoyer l'id d'une entité créée



CQRS: vue d'ensemble



CQRS: vue d'ensemble

➤ Command : un “canal” d'écriture

- une `Command` ne contient que ce qui est nécessaire à son exécution
 - pas besoin de tous les enfants pour en créer un,
 - pas besoin de toutes une entité pour la supprimer
- est consommée par un `CommandHandler`
 - une seule méthode `execute(Command)`
 - comme une des méthodes d'un Service en CRUD
- `AddItemToBasketCommand` + `AddItemToBasketCommandHandler`

➤ Query : un “canal” de lecture

- ne renvoie que ce qui est nécessaire à l'appelant
- Ex : un écran de recherche n'a pas besoin de tous les champs d'une entité

Endpoint à la demande (plus REST)



Migrating from CRUD to CQRS



Faire attention aux impacts sur l'appelant (Front-end)

💡 on pourrait penser qu'il est plus simple de commencer par la lecture...

💡 ... mais les **Commands** ont toujours besoin de toutes les données !

1. Commencer par implémenter les Commands en Back-end, ne consommer que ce qui est nécessaire
2. Adapter les Commands côté Front-end
3. Remonter le flux pour supprimer les données superflux dans les modèles des formulaires
4. Adapter les requêtes côté Front-end et Back-end



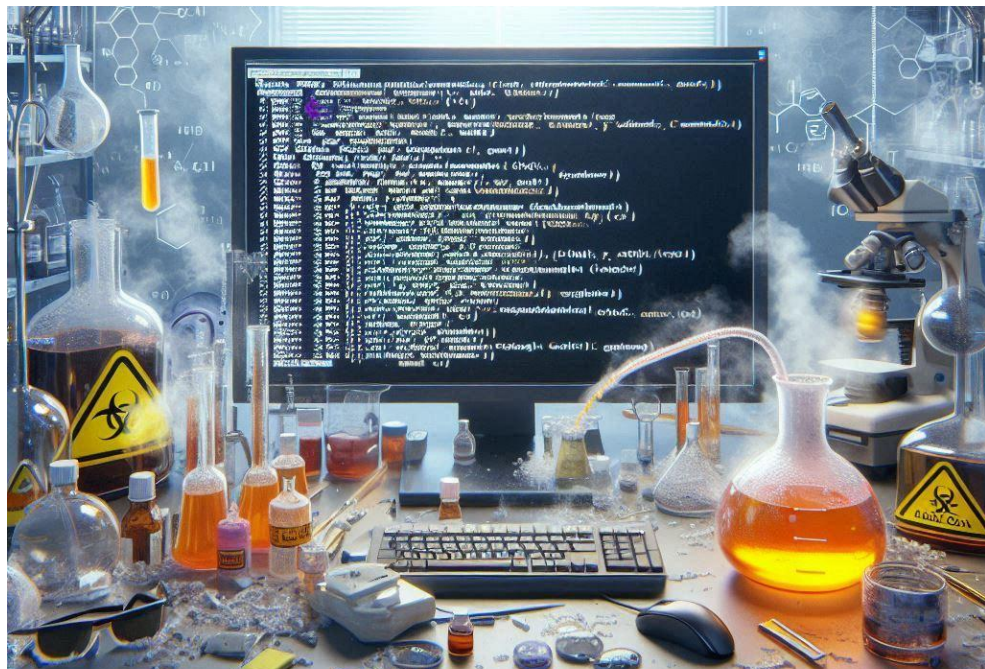
Les données superflux remontées par les Repository n'est pas traité (nécessite la notion d'agrégats)



Migrating from CRUD to CQRS

DEMO Time !!

<https://github.com/tircis/spring-petclinic-rest>



CQRS : aller plus loin en séparant les DataSources ?

Pour quoi ?



Optimiser la lecture (index dédié, ajouter ElasticSearch, NoSQL vs SQL)

Ajoute sa propre complexité



Synchronization des données



Gestion du schéma



Frameworks différents pour accéder aux données



Déploiement



CQRS : aller plus loin avec Event Sourcing ?

Pour quoi ?



Persistence des événements



Pour un besoin de traçabilité et d'audit

Ajoute sa propre complexité



Nécessite un nouveau stockage (base de données, event store, ...)



Compression de l'historique

⇒ **Doit être évalué avec précaution** (valeur pour le métier ?)



CQRS : aller plus loin avec un Bus de Command ?

Pour quoi ?



Exécution des Command en asynchrone



Fort découplage



Haute performance



Mieux quand on a de multiple CommandHandler

Ajoute sa propre complexité



Comment renvoyer des résultats à l'appelant ?



Gestion des transactions



CQRS : aller plus loin avec un framework tout-en-un

AxonIQ [Axon Framework](#)



[Axon Framework](#)



Conclusion



Conclusion

Pros

- 😊 **Maintenance** améliorée après 6 mois (50 entités)
- 😊 Bénéfice sur les tâches et **les branches Git: pas de conflit** sur les fonctionnalités et les corrections de bug
- 😊 **Développement de nouvelles fonctionnalités simplifiées**
- 😊 **Meilleure testabilité**
- 😊 **Interaction des devs Front-end and Back-end**
- 😊 Front-end plus propre
- 😊 Action métier plus claires (aka documentation)
- 😊 Commandes chaînables

Cons

- 😞 **Back-end API couplé au Front-end**, normal puisque le projet n'est plus un "sac à ressource"
- 😞 **Problème de mémoire partiellement traité** côté Back-end: les Commands utilisent toujours le modèle monolithique



Conclusion



Coûteux pour les petits projets (?)



Éléments non obligatoires :

- **Event Sourcing**
- **Séparation des DataSources**
- **Command Bus**



Attention aux frameworks CQRS/ES (Axon Framework)



[from-CRUD-to-CQRS-in-practice](#)



Questions

