From CRUD to CQRS

Guillaume Mary - www.codefilarete.org

- CRUD
  - Definition
  - Why not use it
- CQRS
  - Definition
  - Embrace it smoothly
- Migration Demo
- Conclusion

CRUD

CRUD: definition

- Create Read Update Delete
- Simple access to database entities exposed as resources through REST

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

Create, read, update and delete - Wikipedia

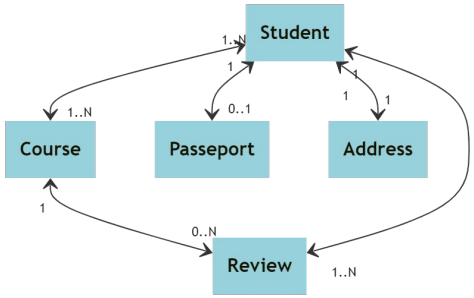
CRUD: usual architecture

Front-End

Components

HTML

Forms



HTTP / JSON

Back-End

StudentController

ReviewController

CourseController

StudentService

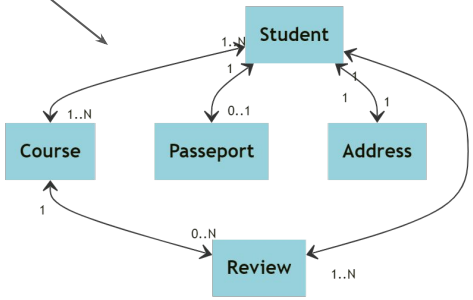
ReviewService

CourseService

StudentRepository

ReviewRepository

CourseRepository



CRUD: what’s the problem ?

- 😰 Pushes unnecessary data to the Front-end, load too many things
  - memory impact
  - network impact
- 😰 Tends to create God classes / God entity graph
- 😰 Tends to create Entity-Oriented Services (with many dependencies)
- 😰 One new feature may impact an existing one
- 😰 Front-End composes the feature

CRUD: let’s see Spring Pet Clinic

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

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

spring HOME OWNERS VETERINARIANS PET TYPES SPECIALTIES

Owners

Name	Address	City	Telephone	Pets
George Franklin	110 W. Liberty St.	Madison	6085551023	Leo
Betty Davis	638 Cardinal Ave.	Sun Prairie	6085551749	Basil
Eduardo Rodriquez	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

spring by Pivotal

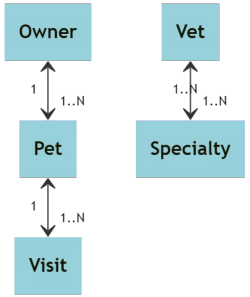
CRUD: let’s see Spring Pet Clinic

Front-End

Components

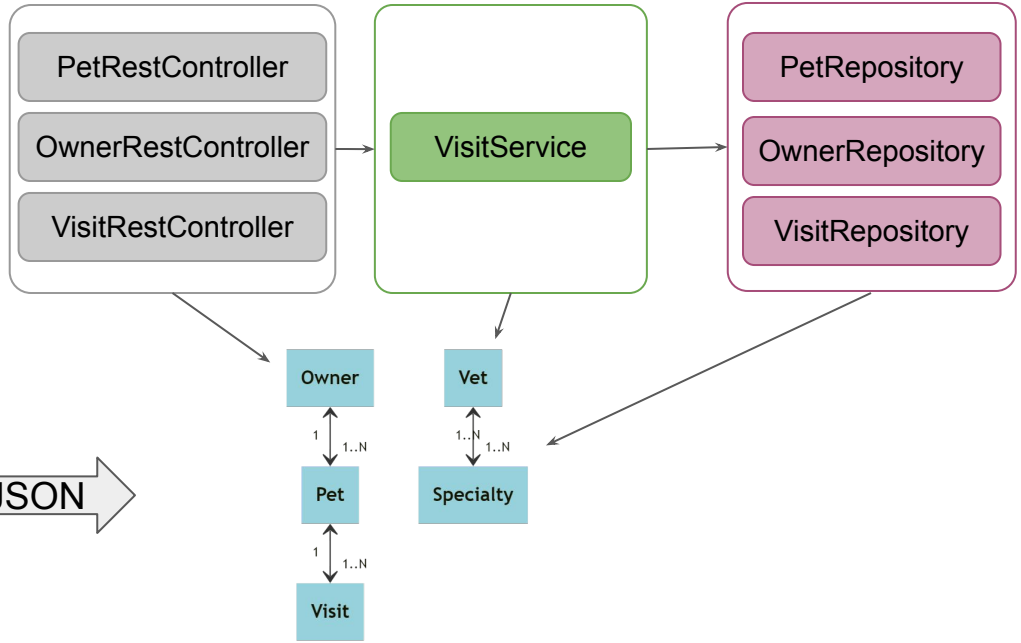
HTML

Forms



HTTP / JSON

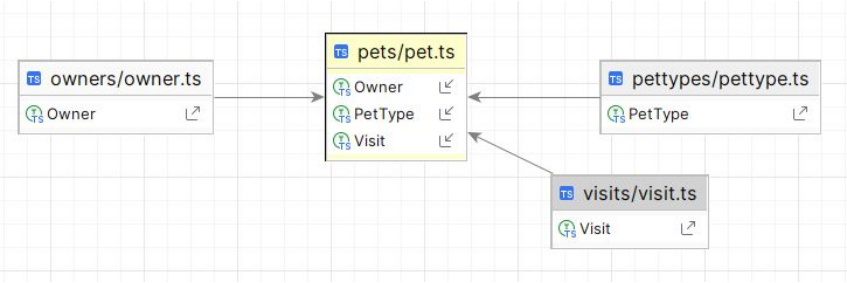
Back-End



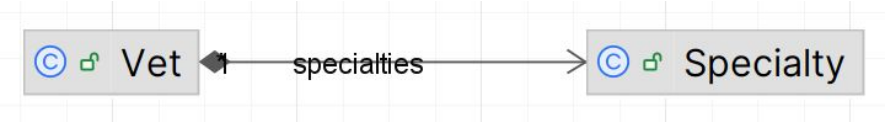
CRUD: let’s see Spring Pet Clinic

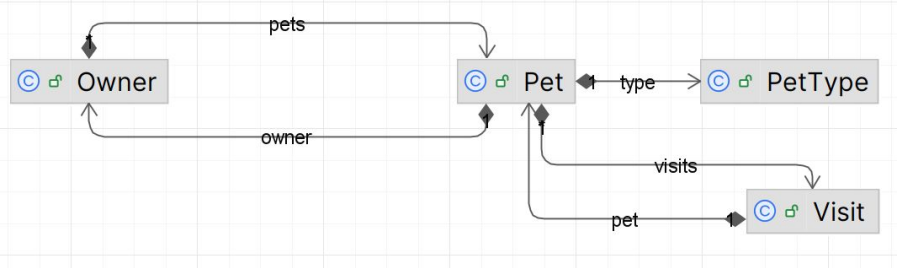
🤔 Front-end mimics Back-end, same complexity on both sides

Front-end model



Back-end model





CRUD: let’s see Spring Pet Clinic

🤔 Front-end model contains useless data for some features

pet-list.component.html

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

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: let’s see Spring Pet Clinic

🤔 Back-end loads too much data for some features

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

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

```
@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: what’s the target ?

👍 Fine for “small” business / simple model

- 👽 few entities
- 💔 not so many relations
- 💾 little data (overall through entity joins)

CQRS

CQRS: a bit of literature

📖 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]

👍 A Command modifies the state of the system

👍 A Query only retrieves data

🙁 More a design pattern than an architecture one (method level)

🙁 Forbids Object creation action : can't return a value

CQRS: a bit of literature

📖 Command and Query Responsibility Segregation (CQRS)

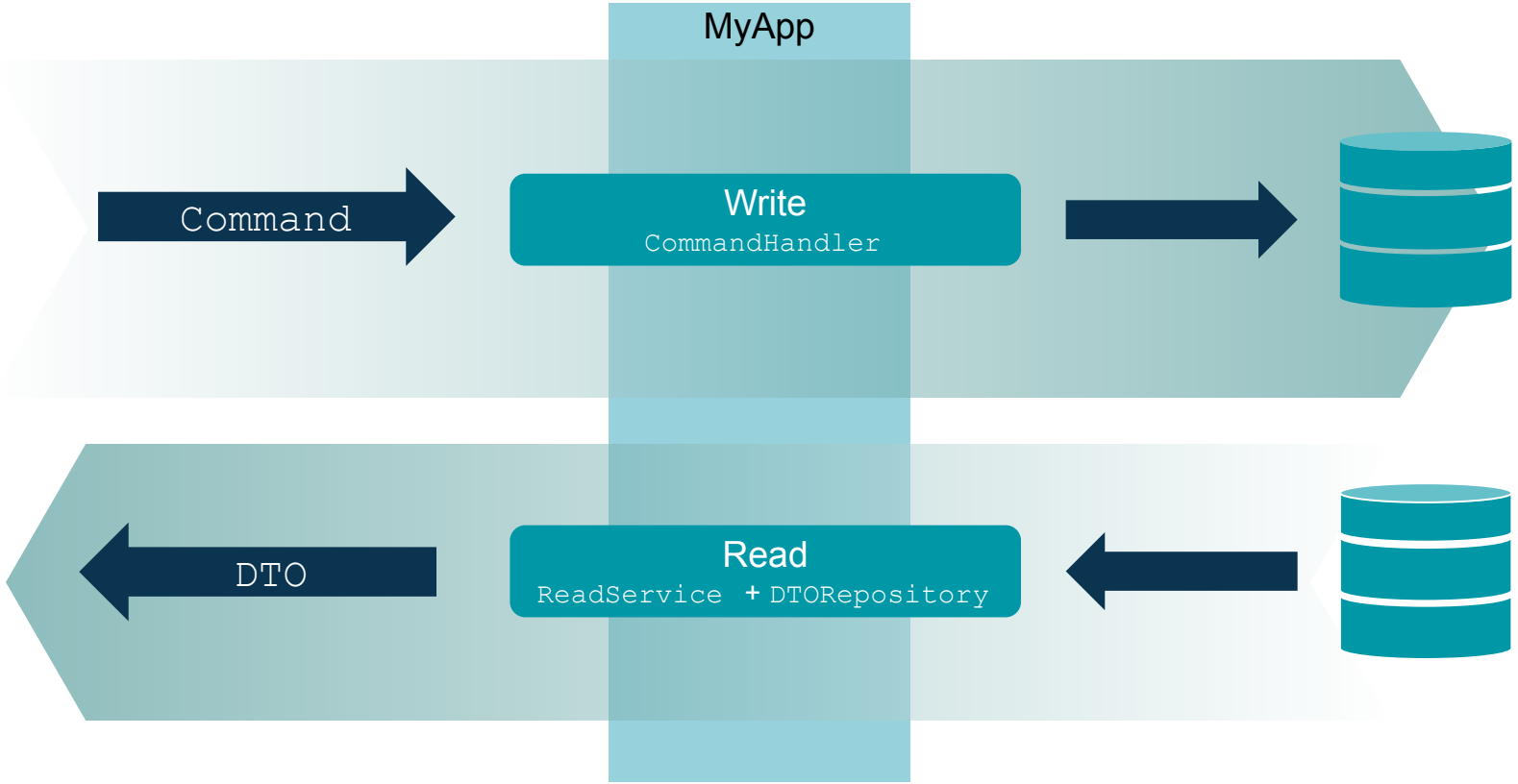
“A system architecture that extends the idea behind command–query separation (CQS) to the level of services”

Command Query Responsibility Segregation - Wikipedia

🙁 Still forbids Object creation action : can’t return value

👍 Let’s forget about literature and allow id returnal on entity creation

CQRS: overview



CQRS: overview

- Command : a write “channel”
  - a `Command` only contains what is necessary to it execution
    - no need of all children to add one,
    - no need of an entire entity to delete it
  - it is consumed by a `CommandHandler`
    - only one method `execute(Command)`
    - same as a method of a Service in CRUD
  - `AddItemToBasketCommand` + `AddItemToBasketCommandHandler`

- Query : a read “channel”
  - only retrieves what’s necessary to caller
  - Ex : a lookup screen doesn’t load all entity fields

On-demand endpoint (not REST)

Migrating from CRUD to CQRS

Beware of callers impacts (Front-end)

we could have thought that starting from query channel is easier…

… but Commands still require all data !

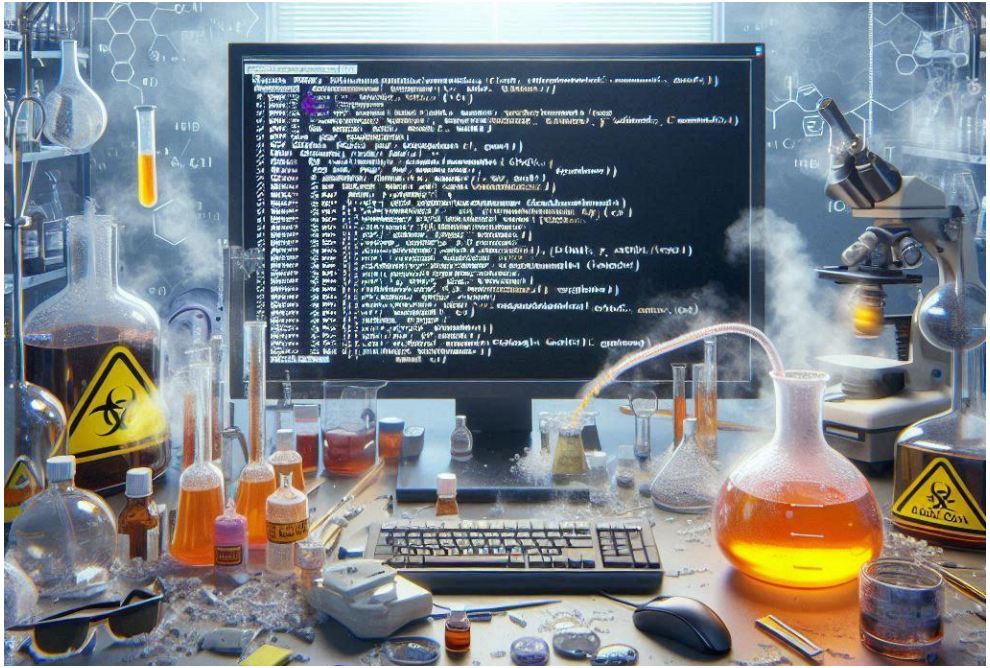
1. Start to implement Commands in Back-end, consume only what's necessary
2. Fix Front-end Commands
3. Go upstream and remove extra data from screen Form Models
4. Fix queries in Front-end and Back-end

Extra data loaded by repositories is not fixed (requires aggregates)

Migrating from CRUD to CQRS

DEMO Time !!

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



CQRS : going further with separate DataSources ?

What for ?

🚀 Optimize Read (with specific indexes, add ElasticSearch, NoSQL vs SQL)

Brings its own complexity

🔄 Data Synchronization
🧩 Schema management
🛠️ Different data access framework
✈️ Deployment

CQRS : going further with Event Sourcing ?

What for ?

- 📧 Event persistence
- 📜 For huge traceability / audit

Brings its own complexity

- 💿 Require another storage (database, event store, …)
- 🗜️ History compaction

⇒ **should be considered with caution** (business value ?)

CQRS : going further with Command Bus ?

What for ?

- 📧 Asynchronous Command execution
- 🌗 High decoupling
- 🚀 High performance
- 👨‍👩‍👧‍👦 Better for multiple CommandHandler

Brings its own complexity

- 🤯 How to send result back to caller ?
- 🤝 Transactions management

CQRS : going further with with an all-in-one framework

Axon Framework

Axon Framework

Conclusion

Conclusion

Pros

- 😀 **Maintenance** is far better than CRUD after 6 months (50 entities)
- 😀 Benefit on tasks and **Git branches: no conflict** on features and bug fixes
- 😀 **Easier testability**
- 😀 **New feature development made easier**
- 😀 **Front-end and Back-end people discussion**
- 😀 Cleaner Front-end
- 😀 Clearer Business actions (aka documentation)
- 😀 Chainable commands

Cons

- 😤 **Back-end API coupled to Front-end**, quite normal since the project is no more a “bag of resources”
- 😤 **Partially solve memory footprint** in the Back-end: Commands still use one monolithic model

Conclusion

- Costly for small projects (?)
- Not mandatory :
  - Event Sourcing
  - Separate DataSources
  - Command Bus
- Beware of CQRS/ES frameworks (Axon Framework)

from-CRUD-to-CQRS-in-practice

Questions