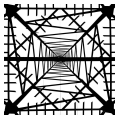
Et si on en finissait avec CRUD ?

Mort au tirant



Julien Vinber
https://www.linkedin.com/in/julienvinber/

AFUP Montpellier - 28 octobre 2020

0. Disclaimer

Disclaimer

J’ai utilisé, j’utilise et j’utiliserais encore CRUD

Disclaimer

Un bon dev peut faire de bonne chose même avec une mauvaise archi

1. Définition

Create : créer

Read : lire

Update : mettre à jour

Delete : supprimer

En architecture logiciel

CRUD n’est pas une vraie architecture.

En général on peut utiliser ce terme quand l’application se calque fortement sur la BDD en utilisant massivement les 4 opérateurs.

1. Avantages

Avantages

- **Rapide**
- **Simple**
- **Souple**
- **Sur**

Exemple avec Symfony

```php
/**
* @ORM\Entity()
*/
class Tag implements \JsonSerializable
{
    /**
     * @ORM\Id
     * @ORM\GeneratedValue
     * @ORM\Column(type="integer")
     */
    private int $id;

    /**
     * @ORM\Column(type="string", unique=true)
     */
    private string $name;
```

Exemple avec Symfony

```php
/**
* @ORM\Entity()
*/
class Post
{

    /**
     * @ORM\Id
     * @ORM\GeneratedValue
     * @ORM\Column(type="integer")
     */
    private $id;

    /**
     * @ORM\Column(type="string")
     * @Assert\NotBlank
     */
    private $title;

    /**
     * @ORM\Column(type="string")
     */
    private $slug;

    /**
     * @ORM\Column(type="string")
     * @Assert\NotBlank(message="post.blank_summary")
     * @Assert\Length(max=255)
     */
    private $summary;

    /**
     * @ORM\Column(type="text")
     * @Assert\NotBlank(message="post.blank_content")
     * @Assert\Length(min=10, minMessage="post.too_short_content")
     */
    private $content;

    /**
     * @ORM\Column(type="datetime")
     */
    private $publishedAt;

    /**
     * @ORM\ManyToOne(targetEntity="App\Entity\User")
     * @ORM\JoinColumn(nullable=false)
     */
    private $author;

    /**
     * @ORM\OneToMany(
     *      targetEntity="Comment",
     *      mappedBy="post",
     *      orphanRemoval=true,
     *      cascade={"persist"}
     * )
     * @ORM\OrderBy({"publishedAt": "DESC"})
     */
    private $comments;

    /**
     * @ORM\ManyToMany(targetEntity="App\Entity\Tag", cascade={"persis
     * @ORM\JoinTable(name="symfony_demo_post_tag")
     * @ORM\OrderBy({"name": "ASC"})
     * @Assert\Count(max="4", maxMessage="post.too_many_tags")
     */
    private $tags;
```

```php
class PostType extends AbstractType
{
    public function buildForm(FormBuilderInterface $builder, array $options): void
    {
        $builder
            ->add('title', null, [
                'attr' => ['autofocus' => true],
                'label' => 'label.title',
            ])
            ->add('summary', TextareaType::class, [
                'help' => 'help.post_summary',
                'label' => 'label.summary',
            ])
            ->add('content', null, [
                'attr' => ['rows' => 20],
                'help' => 'help.post_content',
                'label' => 'label.content',
            ])
            ->add('publishedAt', DateTimePickerType::class, [
                'label' => 'label.published_at',
                'help' => 'help.post_publication',
            ])
            ->add('tags', TagsInputType::class, [
                'label' => 'label.tags',
                'required' => false,
            ])
            ->addEventListener(FormEvents::SUBMIT, function (FormEvent $event) {
                $post = $event->getData();
                if (null !== $postTitle = $post->getTitle()) {
                    $post->setSlug($this->slugger->slug($postTitle)->lower());
                }
            })
        ;
    }
```

```php
/**
* @Route("/admin/post")
* @IsGranted("ROLE_ADMIN")
*/
class BlogController extends AbstractController
{
    /**
     * @Route("/new", methods="GET|POST", name="admin_post_new")
     */
    public function new(Request $request): Response
    {
        $post = new Post();
        $post->setAuthor($this->getUser());

        $form = $this->createForm(PostType::class, $post)
            ->add('saveAndCreateNew', SubmitType::class);

        $form->handleRequest($request);

        if ($form->isSubmitted() && $form->isValid()) {
            $em = $this->getDoctrine()->getManager();
            $em->persist($post);
            $em->flush();
            $this->addFlash('success', 'post.created_successfully');
            if ($form->get('saveAndCreateNew')->isClicked()) {
                return $this->redirectToRoute('admin_post_new');
            }

            return $this->redirectToRoute('admin_post_index');
        }

        return $this->render('admin/blog/new.html.twig', [
            'post' => $post,
            'form' => $form->createView(),
        ]);
    }
```

```php
    /**
     * @Route("/{id<\d+>}", methods="GET", name="admin_post_show")
     */
    public function show(Post $post): Response
    {
        $this->denyAccessUnlessGranted(PostVoter::SHOW, $post, 'Posts can only be shown to their authors.');
        return $this->render('admin/blog/show.html.twig', [
            'post' => $post,
        ]);
    }

    /**
     * @Route("/{id<\d+>}/edit", methods="GET|POST", name="admin_post_edit")
     * @IsGranted("edit", subject="post", message="Posts can only be edited by their authors.")
     */
    public function edit(Request $request, Post $post): Response
    {
        $form = $this->createForm(PostType::class, $post);
        $form->handleRequest($request);

        if ($form->isSubmitted() && $form->isValid()) {
            $this->getDoctrine()->getManager()->flush();

            $this->addFlash('success', 'post.updated_successfully');

            return $this->redirectToRoute('admin_post_edit', ['id' => $post->getId()]);
        }

        return $this->render('admin/blog/edit.html.twig', [
            'post' => $post,
            'form' => $form->createView(),
        ]);
    }
```

```php
    /**
     * @Route("/{id}/delete", methods="POST", name="admin_post_delete")
     * @IsGranted("delete", subject="post")
     */
    public function delete(Request $request, Post $post): Response
    {
        if (!$this->isCsrfTokenValid('delete', $request->request->get('token'))) {
            return $this->redirectToRoute('admin_post_index');
        }
        $post->getTags()->clear();

        $em = $this->getDoctrine()->getManager();
        $em->remove($post);
        $em->flush();

        $this->addFlash('success', 'post.deleted_successfully');

        return $this->redirectToRoute('admin_post_index');
    }
}
```

Sur le papier c’est GENIAL…

Sur le PAPIER c’est génial…

En vrais

Bombe à retardement

2. Inconvénients

Inconvénients

- **Rapide**
- Simple
- Souple
- Sur

On délègue les responsabilités

Inconvénients

- Rapide
- **Simple**
- Souple
- Sur

Car c’est difficile de faire des choses avancé ?

Inconvénients

- Rapide
- Simple
- **Souple**
- Sur

On vire les contraintes et cela passe.

Inconvénients

- Rapide
- Simple
- Souple
- **Sur**

Dans la vie on ne peut jamais tout avoir.

Historique

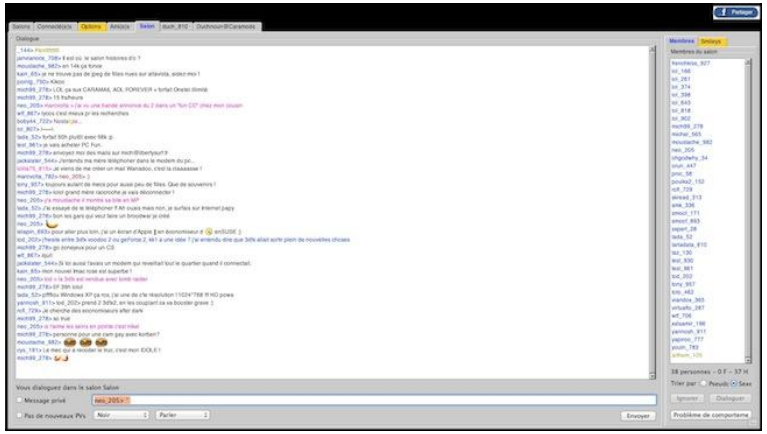
Avant (il y as longtemps) :

- Lourd
- Cycle en V
- Plein de controle
- Des administrateur de base de données
- Merise : Sainte base de données priée pour nous.

Historique

L'arrivée du web :

- Petit
- Peux d’enjeux
- Sans conséquence
- Vite fait, ~~bien fait~~
- MySQL en MyISAM



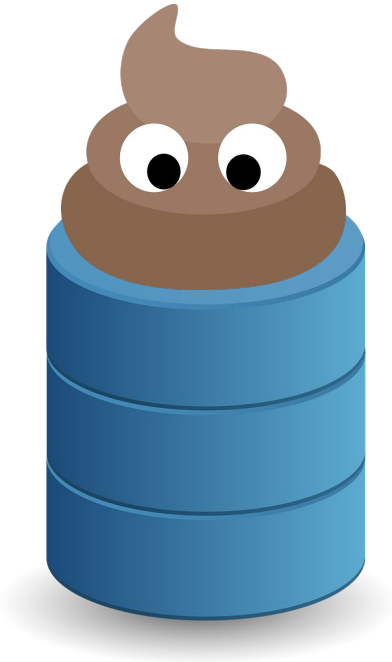
Historique

Aujourd'hui on a pris le pire des deux

- Application lourde
- Sainte BDD
- Agile
  - On fait, puis on réfléchit
  - Fait simple on améliorera après (ou pas)
- C'est les dev qui font des modif au besoin sur la BDD

  => Structure de BDD qui ne ressemble à plus rien.

- Un seul mots d'ordre : rapide (CRUD c'est bien;)



3. Solutions

Tuons les BDD et CRUD

Aujourd’hui BDD d’état

=> vérité à un temps T

Tuons les BDD et CRUD

Pas d’historique

Il faut l'ajouter, mais partiel.

Tuons les BDD et CRUD

Un état n’est qu’une représentation à un temps T d’un système.

Au moindre problème la base part en live.

Tuons les BDD et CRUD

Solution

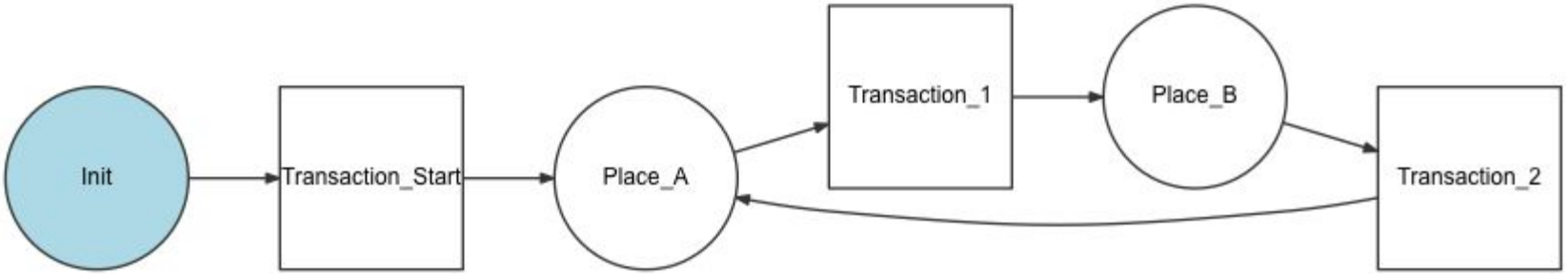
L’action ou évènement est une vérité absolue.

L’état n’est que la résultante de ces évènements.

Workflow

Workflow

Introduction de la notion d'événement (transaction)



Event Sourcing

Pas garantie à 100%

Event Sourcing

Une seule vérité : l’event

=> on enregistre les event

Event Sourcing

Un event est défini par :

- Contraintes d'activations
- Contraintes sur les data
- …
- Nos règles métier

Est uniquement par les règles métier de l’event

Event Sourcing

L’état

- C’est un calcul à partir des évents
- Ce n’est qu’une projection parmi d’autres
- On peut créer des états adaptés à notre besoin