TP11 — Amélioration du CRUD MVC “Créations”

⚠️ Remarques d’amélioration

🔹 1. delete via GET

On le sait déjà, mais le jury peut tiquer :

```
/?controller=creation&action=delete&id=1
```

👉 **Réponse orale à donner** :

> *“Pour un projet pédagogique, j’ai utilisé GET.*
> *En production, j’utiliserais POST avec un token CSRF.”*

🔹 2. created_at en string

Ce n’est **pas une erreur**.

👉 **Réponse orale :**

> *“On pourrait utiliser DateTimeImmutable dans l’entité, mais pour garder un projet accessible et lisible, on a conservé une string.”*

➞ Très bien accepté.

🔹 3. Absence de constructeur dans Creation

✓ C’est **un choix assumé** et cohérent avec l’hydratation
✓ Même avantage pédagogique

1) Suppression en POST + CSRF (au lieu de GET)

🎯 Objectif

- Ne plus supprimer via lien GET
- Utiliser un formulaire POST
- Ajouter un token CSRF simple (maison)

1.0 Modifier le front controller

public/index.php

Tout en haut du fichier :

```php
<?php

declare(strict_types=1);

use App\Core\Router;
use App\Core\Autoloader;

session_start();

require_once dirname(__DIR__) . '/Core/Autoloader.php';

Autoloader::register();

$router = new Router();
$router->routes();
```

La session doit être démarrée dans le front controller (public/index.php), et non dans les controllers ou services.

1.1 Ajouter un mini service CSRF

Core/Csrf.php

```php
<?php

declare(strict_types=1);

namespace App\Core;

final class Csrf
{
    private const KEY = '_csrf_tokens';

    public static function token(string $id): string
    {
```

```
        if (session_status() !== PHP_SESSION_ACTIVE) {
            session_start();
        }

        $token = bin2hex(random_bytes(32));
        $_SESSION[self::KEY][$id] = $token;

        return $token;
    }

    public static function isValid(string $id, ?string $token): bool
    {
        if (session_status() !== PHP_SESSION_ACTIVE) {
            session_start();
        }

        if ($token === null) {
            return false;
        }

        $known = $_SESSION[self::KEY][$id] ?? null;

        return is_string($known) && hash_equals($known, $token);
    }
}
```

1.2 Passer le delete en POST côté vue

Views/creation/show.php (remplacer le lien delete)

```
<?php
use App\Core\Csrf;
/** @var Creation $creation */
?>

<form method="post" action="/?controller=creation&action=delete&id=<?=
$creation->getIdCreation() ?>"
      style="display:inline;">
    <input type="hidden" name="_token" value="<?=
htmlspecialchars(Csrf::token('delete_creation_' . $creation->getIdCreation()))
?>">
    <button type="submit" onclick="return confirm('Supprimer ?');">🗑
Supprimer</button>
</form>
```

✅ On garde l'URL pour id, mais l'action est désormais **POST**.

1.3 Vérifier méthode + token côté Controller

CreationController::delete(int $id)

```php
public function delete(int $id): void
{
    if ($_SERVER['REQUEST_METHOD'] !== 'POST') {
        http_response_code(405);
        echo 'Méthode non autorisée';
        return;
    }

    $token = $_POST['_token'] ?? null;

    if (!Csrf::isValid('delete_creation_' . $id, $token)) {
        http_response_code(403);
        echo 'CSRF invalide';
        return;
    }

    $model = new CreationModel();
    $model->delete($id);

    header('Location: /?controller=creation&action=index');
    exit;
}
```

2) Post/Redirect/Get sur create/edit (déjà OK)

Tu le fais déjà : POST → header Location → exit.
✅ Rien à changer, juste **à verbaliser** comme bonne pratique PRG.

3) Passer createdAt en DateTimeImmutable dans l'entité

🎯 Objectif

- createdAt devient un objet date
- On convertit created_at SQL (string) → DateTimeImmutable dans le setter

3.1 Modifier l'entité Creation

Entities/Creation.php (partie createdAt)

```php
    private ?\DateTimeImmutable $createdAt = null;

    public function getCreatedAt(): ?\DateTimeImmutable
    {
```

```
        return $this->createdAt;
    }

    /**
     * Setter appelé par hydratation : clé SQL "created_at"
     * On accepte string (DB) ou DateTimeImmutable (code)
     */
    public function setCreatedAt(string|\DateTimeImmutable|null $createdAt):
void
    {
        if ($createdAt instanceof \DateTimeImmutable) {
            $this->createdAt = $createdAt;
            return;
        }

        if (is_string($createdAt) && $createdAt !== '') {
            $dt = \DateTimeImmutable::createFromFormat('Y-m-d H:i:s',
$createdAt);
            $this->createdAt = $dt ?: null;
            return;
        }

        $this->createdAt = null;
    }
```

✅ Hydratation toujours automatique : created_at → setCreatedAt().

3.2 Afficher la date dans les vues (optionnel)

Exemple dans `show.php`

```
<?php if ($creation->getCreatedAt() instanceof DateTimeImmutable): ?>
    <p><small>Créée le : <?= htmlspecialchars($creation->getCreatedAt()-
>format('d/m/Y H:i')) ?></small></p>
<?php endif; ?>
```

Pitch oral “post-jury” en 15 secondes (à apprendre)

> *« Après le TP pédagogique, j’ai amélioré la sécurité : suppression via POST et token CSRF, et j’ai typé createdAt en DateTimeImmutable avec conversion lors de l’hydratation. »*

Dans **NOTRE cours**, on a bien ajouté un **"cas spécial dates" dans Entity::hydrate()**. C'est écrit noir sur blanc (page 32–33) : conversion created_at (et autres *_at) en DateTimeImmutable avant d'appeler le setter.

```php
if (
    in_array($key, ['created_at', 'updated_at', 'deleted_at'], true)
    && is_string($value)
    && $value !== ''
) {
    $value = new DateTimeImmutable($value);
}
```

✅ 1) Pourquoi on l'avait mis dans hydrate() dans le cours

Parce que pédagogiquement c'est très efficace :

- PDO renvoie **des strings** (même pour les dates)
- Nous, on veut **un vrai type date** dans l'entité
- Et on veut que l'hydratation "magique" marche tout de suite, sans charger le setter

C'est exactement le message du cours :
« PDO renvoie des strings. L'entité veut des types propres. »

➞ Donc le choix "conversion dans hydrate()" est **cohérent** avec le TP d'hydratation.

✅ 2) Pourquoi j'ai proposé ensuite "conversion dans le setter"

Parce que là tu es passé en mode **"améliorations post-jury / propreté architecture"** :

- Entity est censée rester **générique**
- Les règles de typage (date, int, etc.) sont plutôt **du métier de l'entité concrète**
- Et tu évites de coder des règles "SQL-ish" dans une classe abstraite

➞ C'est plus "clean" architecturalement.

Option : garder le constructeur (date par défaut) ET garder l'hydratation

À condition de cadrer 3 règles simples :

1. **Le constructeur donne un état valide par défaut**
2. **L'hydratation (DB) a le droit d'écraser ce défaut** *uniquement si la DB fournit une valeur*
3. **created_at** est **NOT NULL** en BDD

1) Entité Creation : constructeur + setter

```php
    private \DateTimeImmutable $createdAt;

    public function __construct()
    {
        $this->createdAt = new \DateTimeImmutable();
    }

    public function getCreatedAt(): \DateTimeImmutable
    {
        return $this->createdAt;
    }

    public function setCreatedAt(\DateTimeImmutable|string $d): void
    {
        if (is_string($d)) {
            $d = new \DateTimeImmutable($d);
        }
        $this->createdAt = $d;
    }
```

2) Entity::hydrate()

On laisse générique :

```php
    protected function hydrate(array $data): void
    {
        foreach ($data as $key => $value) {
            $method = 'set' . str_replace(' ', '', ucwords(str_replace(['-',
'_'], ' ', $key)));
            if (!method_exists($this, $method)) {
                continue;
            }
            $this->$method($value);
        }
    }
```

3) CreationModel::insert() : laisser la date venir de l'entité (pas du modèle)

```php
    public function insert(Creation $creation): Creation
    {
        $sql = 'INSERT INTO creation (title, description, created_at, picture)
                VALUES (:title, :description, :created_at, :picture)';

        $stmt = $this->pdo->prepare($sql);
        $stmt->execute([
            'title' => $creation->getTitle(),
            'description' => $creation->getDescription(),
            'created_at' => $creation->getCreatedAt()->format('Y-m-d H:i:s'),
            'picture' => $creation->getPicture(),
        ]);

        $id = (int) $this->pdo->lastInsertId();

        return $this->find($id) ?? throw new \RuntimeException('Relecture
impossible');
    }
```

En conclusion

- **constructeur** : garantit un createdAt non-null
- **setter** : accepte la DB et écrase seulement si valeur réelle
- **model** : utilise getCreatedAt() au lieu de "fabriquer" une date
- **hydrate** : redevient neutre

🎉 Possibilité de factoriser ?

Oui, **on peut factoriser** — et on a même **deux duplications** différentes :

1. **duplication des templates** create.php / edit.php (quasi le même <form>)
2. **duplication côté controller** (même validation + mêmes champs récupérés, deux fois)

1) Factoriser les vues : un `_form.php` commun

`create.php` et `edit.php` ont **exactement les mêmes champs** (title, description, picture), seules changent les libellés et le lien “retour/annuler”.

👉 Faire un partial :

Views/creation/_form.php

```
<?php
/** @var string $title */
/** @var ?string $error */
/** @var array $old */
/** @var string $submitLabel */
/** @var string $cancelUrl */
?>
<h1><?= htmlspecialchars($title) ?></h1>

<?php if (!empty($error)): ?>
    <p style="color:red;"><?= htmlspecialchars($error) ?></p>
<?php endif; ?>

<form method="post">
    <p>
        <input name="title" placeholder="Titre" value="<?=
htmlspecialchars($old['title'] ?? '') ?>">
    </p>

    <p>
        <textarea name="description" placeholder="Description"><?=
htmlspecialchars($old['description'] ?? '') ?></textarea>
    </p>

    <p>
        <input name="picture" placeholder="Image (optionnel)" value="<?=
htmlspecialchars($old['picture'] ?? '') ?>">
    </p>

    <button type="submit"><?= htmlspecialchars($submitLabel) ?></button>
```

```
</form>

<p><a href="<?= htmlspecialchars($cancelUrl) ?>">Annuler</a></p>
```

Puis :

Views/creation/create.php

```
<?php
$submitLabel = 'Créer';
$cancelUrl = '/?controller=creation&action=index';
require __DIR__ . '/_form.php';
```

Views/creation/edit.php

```
<?php
$submitLabel = 'Enregistrer';
$cancelUrl = '/?controller=creation&action=show&id=' . $creation-
>getIdCreation();
require __DIR__ . '/_form.php';
```

2) Factoriser create/edit dans le Controller : une méthode “handleForm()”

Les méthodes create() et edit() répètent :

- lecture + trim des champs
- validation “titre + description obligatoires”
- préparation de $old
- puis insert/update

👉 On peut faire deux méthodes privées :

```
    private function getPostedData(): array
    {
        $title = trim($_POST['title'] ?? '');
        $description = trim($_POST['description'] ?? '');
        $picture = trim($_POST['picture'] ?? '');
        $picture = $picture === '' ? null : $picture;

        return [$title, $description, $picture];
    }

    private function buildEntity(string $title, string $description, ?string
$picture): Creation
    {
        $c = new Creation();
```

```
        $c->setTitle($title);
        $c->setDescription($description);
        $c->setPicture($picture);
        return $c;
    }
```

La méthode create() devient

```
    public function create(): void
    {
        if ($_SERVER['REQUEST_METHOD'] === 'POST') {
            [$title, $description, $picture] = $this->getPostedData();

            if ($title === '' || $description === '') {
                $this->render('creation/create', [
                    'title' => 'Créer une création',
                    'error' => 'Titre et description obligatoires',
                    'old' => [
                        'title' => $title,
                        'description' => $description,
                        'picture' => $picture ?? '',
                    ],
                ]);
                return;
            }

            $creation = $this->buildEntity($title, $description, $picture);

            $model = new CreationModel();
            $created = $model->insert($creation);

            header('Location: /?controller=creation&action=show&id=' .
$created->getIdCreation());
            exit;
        }

        $this->render('creation/create', [
            'title' => 'Créer une création',
            'old' => ['title' => '', 'description' => '', 'picture' => ''],
        ]);
    }
```

Et edit ()

```
    public function edit(int $id): void
    {
        $model = new CreationModel();
        $creation = $model->find($id);

        if ($creation === null) {
            http_response_code(404);
            echo '404 - Création introuvable';
            return;
        }

        if ($_SERVER['REQUEST_METHOD'] === 'POST') {
            [$title, $description, $picture] = $this->getPostedData();

            if ($title === '' || $description === '') {
                $this->render('creation/edit', [
                    'title' => 'Modifier la création',
                    'error' => 'Titre et description obligatoires',
                    'creation' => $creation, // pour l'id
                    'old' => ['title' => $title, 'description' =>
$description, 'picture' => $picture ?? ''],
                ]);
                return;
            }

            $toUpdate = $this->buildEntity($title, $description, $picture);

            $toUpdate = $model->update($id, $toUpdate);

            header('Location: /?controller=creation&action=show&id=' . $id);
            exit;
        }

        $this->render('creation/edit', [
            'title' => 'Modifier une création',
            'creation' => $creation,
            'old' => [
                'title' => $creation->getTitle(),
                'description' => $creation->getDescription(),
                'picture' => $creation->getPicture() ?? '',
            ],
        ]);
    }
```