

Développement d'un site e-commerce avec Laravel (Pagination)

Objectifs de la séance

- Manipuler les données d'une base de données avec **Eloquent ORM**.
 - Utiliser des **requêtes SQL natives** dans Laravel.
 - Mettre en place une **pagination** simple et personnalisée pour améliorer la navigation.
-

Partie 1 : Manipulation des données avec Laravel

1.1 Utilisation d'Eloquent ORM

Objectif :

Récupérer les données depuis la base de données et les afficher dans une vue.

- Rôle d'Eloquent ORM

Eloquent est l'ORM de Laravel. Il permet d'interagir avec la base de données à travers des modèles PHP, sans écrire directement du SQL.

- Code du contrôleur

```
public function index()
{
    $articles = Article::all(); // Récupère tous les articles
    return view('articles', ['articles' => $articles]);
}
```

Explication :

- `Article::all()` récupère tous les enregistrements de la table `articles`.
- Les données sont stockées dans la variable `$articles`.
- La vue `articles.blade.php` reçoit ces données.

- Code de la vue (`articles.blade.php`)

```
<table border="2">
<tr>
    <th>Titre</th>
    <th>Contenu</th>
</tr>

@foreach ($articles as $article)
<tr>
    <td>{{ $article->titre }}</td>
```

```

        <td>{{ $article->contenu }}</td>
    </tr>
    @endforeach
</table>

```

Explication :

- `@foreach` permet de parcourir la liste des articles.
 - Chaque article est affiché dans une ligne du tableau.
 - Les données sont affichées avec la syntaxe Blade `{{ }}`.
-

1.2 Utilisation de SQL natif

Objectif :

Manipuler les données à l'aide de requêtes SQL brutes.

- Code du contrôleur

```

use Illuminate\Support\Facades\DB;

public function produitsParCategorie($cat)
{
    $query = "SELECT * FROM produits WHERE categorie = '$cat'";
    $produits = DB::select($query); // Retourne un tableau d'objets
    return view('produits', ['produits' => $produits]);
}

```

Explication :

- `DB::select()` exécute une requête SQL brute.
- Les résultats sont retournés sous forme d'objets PHP.
- Les données sont envoyées à la vue `produits.blade.php`.

⚠ Remarque importante :

Les données sont accessibles comme des objets :

`$item->nom`

et non :

`$item['nom']`

Code de la vue (`produits.blade.php`)

```

<table border="2">
    <tr>
        <th>Nom</th>
        <th>Description</th>
    </tr>
    @foreach ($produits as $item)

```

```

<tr>
    <td>{{ $item->nom }}</td>
    <td>{{ $item->description }}</td>
</tr>
@endforeach
</table>

```

Cas	Type de données	<code>\$item->nom</code>	<code>\$item['nom']</code>
<code>DB::select()</code>	Objet (<code>stdClass</code>)	✓	✗
Eloquent (<code>Model</code>)	Objet (<code>Model</code>)	✓ (recommandé)	✓ (possible)
Tableau PHP	Tableau associatif	✗	✓

Partie 2 : Gestion de la pagination

2.1 Pagination avec Eloquent

Objectif :

Diviser les données en plusieurs pages pour améliorer l'expérience utilisateur.

Code du contrôleur

```
public function index()
{
    $articles = Article::paginate(10); // 10 articles par page
    return view('articles', ['articles' => $articles]);
}
```

Explication :

- `paginate(10)` limite l'affichage à 10 articles par page.
- Laravel gère automatiquement le numéro de page.

Code de la vue

```
<table border="2">
    <tr>
        <th>Titre</th>
        <th>Contenu</th>
    </tr>

    @foreach ($articles as $article)
    <tr>
        <td>{{ $article->titre }}</td>
        <td>{{ $article->contenu }}</td>
    </tr>
    @endforeach
</table>

{{ $articles->links() }}
```

Explication :

- `links()` génère automatiquement les boutons de navigation (pages).
-

Partie 3 : Personnalisation de la pagination

3.1 Crédation d'une pagination personnalisée

Objectif :

Personnaliser l'apparence de la pagination avec Bootstrap.

Étapes

1. Créer le fichier :

```
resources/views/vendor/pagination/custom.blade.php
```

2. Ajouter le code suivant :

```
@if ($paginator->hasPages())
<ul class="pagination">

    @if ($paginator->onFirstPage())
        <li class="page-item disabled">
            <span class="page-link">Précédent</span>
        </li>
    @else
        <li class="page-item">
            <a class="page-link" href="{{ $paginator-
>previousPageUrl() }}">
                Précédent
            </a>
        </li>
    @endif

    @foreach ($elements as $element)
        @if (is_string($element))
            <li class="page-item disabled">
                <span class="page-link">{{ $element }}</span>
            </li>
        @endif

        @if (is_array($element))
            @foreach ($element as $page => $url)
                @if ($page == $paginator->currentPage())
                    <li class="page-item active">
                        <span class="page-link">{{ $page }}</span>
                    </li>
                @else
                    <li class="page-item">
                        <a class="page-link" href="{{ $url }}>{{
$page }}</a>
                    </li>
                @endif
            @endforeach
        @endif
    @endforeach

    @if ($paginator->hasMorePages())
        <li class="page-item">
```

```

        <a class="page-link" href="{{ $paginator->nextPageUrl() }}>
            Suivant
        </a>
    </li>
@else
    <li class="page-item disabled">
        <span class="page-link">Suivant</span>
    </li>
@endif

</ul>
@endif

```

3.2 Utilisation de la pagination personnalisée

Dans la vue principale :

```
{{ $articles->links('vendor.pagination.custom') }}
```

Explication :

- Laravel utilise maintenant la pagination personnalisée au lieu de celle par défaut.

Atelier 8 :

Développer et déployer en ligne la version 3 de l'application e-commerce (store en ligne) en Ajoutant la pagination.

À rendre :

-Rapport avec des captures d'écran

-lien github

-lien vercel

Bienvenue sur notre site e-commerce

Découvrez nos produits de qualité à des prix compétitifs

Nom	Prix	Image
Machine à laver	3000DH	
Four	1500DH	