Laravel : Créer un système de panier e-commerce

Auteur	Domaine	Technologies
Wilo Ahadi	Développement web	Laravel, PHP

Un tutoriel pour mettre en place un système de panier dynamique dans un projet Laravel de vente en ligne (e-commerce).

Sommaire

Introduction

Un système de panier est l'une des fonctionnalités principales à implémenter lors de la création d'un site web e-commerce (commerce électronique) : Le panier permet à l'utilisateur (client) de grouper les produits qu'il désire pour les commander (payer) en une seule facture.

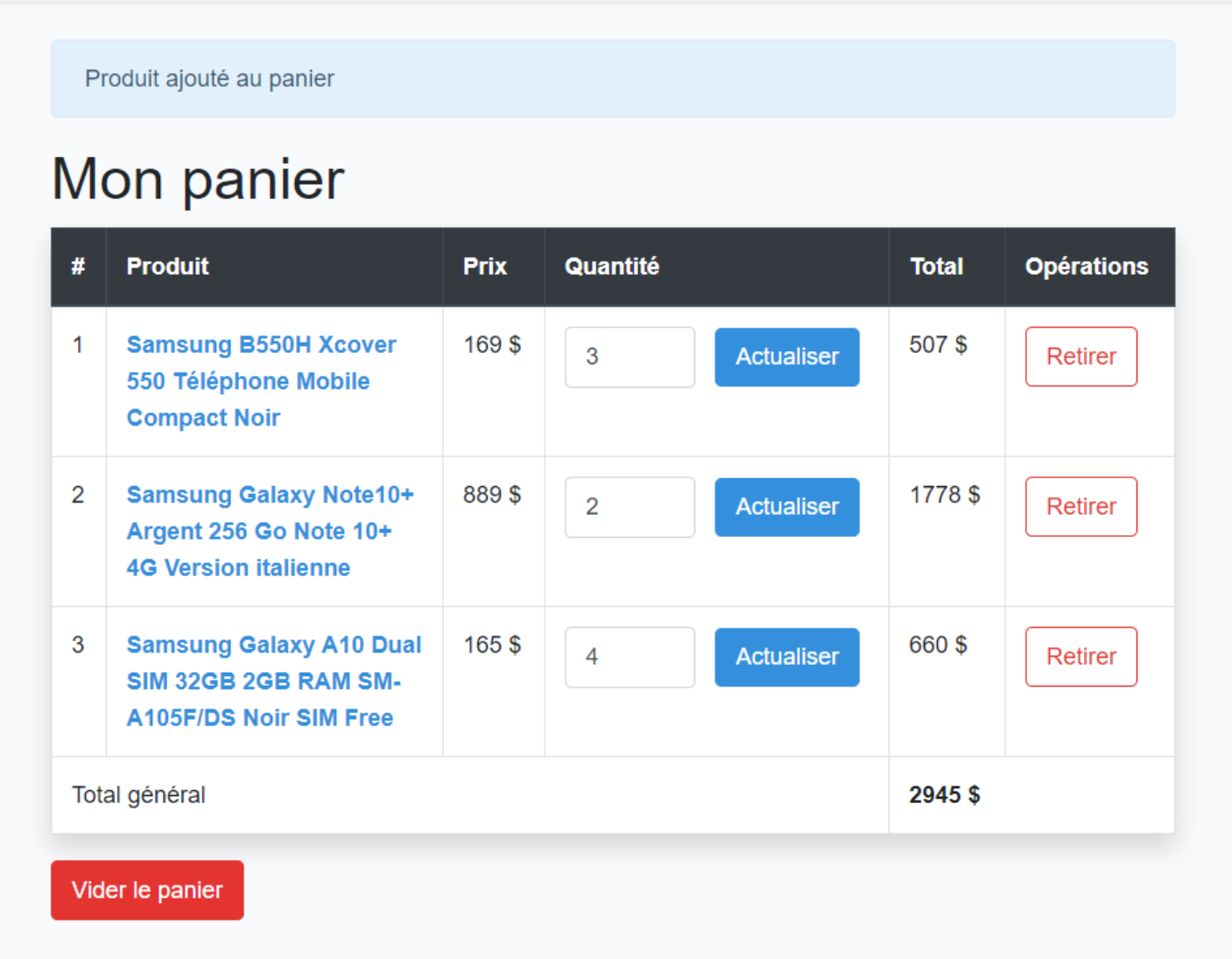
Considérez le terme « produit » ici comme un article, une marchandise, un service ou toute autre chose qu'on peut mettre en vente sur internet.

Dans ce guide, nous allons voir comment mettre en place un système de panier dynamique dans une application Laravel pour permettre aux utilisateurs de :

- Ajouter un produit au panier

- Retirer un produit du panier
- Afficher le prix total de chaque produit du panier selon sa quantité
- Afficher le prix total de tous les produits du panier
- Retirer tous les produits ou vider le panier

La capture ci-dessous illustre ce que nous allons produire :



Mise en place du système de panier

On ne peut parler de panier sans présenter les produits qui sont la base même du système. Commençons par voir comment présenter produits dans la base de données et sur une vue puis nous verrons comment les gérer au panier dynamique.

1. Les produits du panier

Au niveau de la base de données, les informations (champs) minimales qu'un produit doit présenter pour un système de panier e-commerce sont :

- Un identifiant
- Un nom ou titre

- Un prix

A celles-ci, nous pouvons ajouter une image, une description, une date d'expiration, une catégorie, une couleur, … dépendant de ce qu'on veut présenter.

La migration **database\migrations\…_create_products_table.php** de la table des produits avec les informations minimales peut se présenter de la manière suivante :

```
<?php
// ...
class CreateProductsTable extends Migration
{

    public function up()
    {
        Schema::create('products', function (Blueprint $table) {
            $table->id(); // L'identifiant
            $table->string('name'); // Le nom ou titre
            $table->text("description"); // La description
            $table->double("price"); // Le prix
            $table->timestamps(); // created_at, updated_at
        });
    }
    // ...
}
?>
```

Supposons maintenant avoir une ressource **App\Http\Controllers\ProductController.php** qui s'occupe de la logique de gestion des produits, le modèle **App\Product.php** qui représente un produit de la base données et les routes suivantes au fichier **routes\web.php** :

```
λ php artisan route:list
+--------+-----------+-----------------------+----------------+-----------------------------------------------+--------------+
| Domain | Method    | URI                   | Name           | Action                                        | Middleware   |
+--------+-----------+-----------------------+----------------+-----------------------------------------------+--------------+
|        | GET|HEAD  | /                     |                | Closure                                       | web          |
|        | GET|HEAD  | api/user              |                | Closure                                       | api,auth:api |
|        | GET|HEAD  | product               | product.index  | App\Http\Controllers\ProductController@index  | web          |
|        | POST      | product               | product.store  | App\Http\Controllers\ProductController@store  | web          |
|        | GET|HEAD  | product/create        | product.create | App\Http\Controllers\ProductController@create | web          |
|        | GET|HEAD  | product/{product}     | product.show   | App\Http\Controllers\ProductController@show   | web          |
|        | PUT|PATCH | product/{product}     | product.update | App\Http\Controllers\ProductController@update | web          |
|        | DELETE    | product/{product}     | product.destroy| App\Http\Controllers\ProductController@destroy| web          |
|        | GET|HEAD  | product/{product}/edit| product.edit   | App\Http\Controllers\ProductController@edit   | web          |
+--------+-----------+-----------------------+----------------+-----------------------------------------------+--------------+
```

L'action « show » de la route nommée « product.show » où on récupère un produit de la base de données par son identifiant pour l'afficher sur la vue **resources\views\products\product.blade.php** se décrit de la manière suivante au contrôleur **ProductController.php** :

```
// ...
public function show(Product $product)
{
      return view("products.product", compact("product"));
}
//...
```

Et le code source de la vue **resources\views\products\product.blade.php** :

```
@extends("layouts.app")
@section("content")
<div class="container">
 <div class="row justify-content-center">
  <div class="col-md-6">
   <h1>{{ $product->name }}</h1>
   <h3 class="text-success" >{{ $product->price }} $</h3>
   <div class="mb-3" >{!! nl2br($product->description) !!}</div>
   <div class="bg-white mt-3 p-3 border text-center" >
    <p>Commander <strong>{{ $product->name }}</strong></p>
    <form method="POST" action="#" class="form-inline d-inline-block" >
     {{ csrf_field() }}
     <input type="number" name="quantity" placeholder="Quantité ?" class="form-contr
     <button type="submit" class="btn btn-warning" >+ Ajouter au panier</button>
    </form>
   </div>
  </div>
 </div>
</div>
@endsection
```

Ce qui affiche les informations (nom, prix, description) du produit sélectionné avec un formulaire pour ajouter le produit au panier en entrant la quantité :

Nous reviendrons sur ce formulaire d'ajout du produit au panier après avoir vu les routes de gestion du panier au point suivant.

L'arborescence actuelle du projet pour les fichiers dont nous avons parlé est :

- app\Http\Controllers**ProductController.php**
- app**Product.php**
- database\migrations**..._create_products_table.php**
- resources\views\products**product.blade.php**
- routes**web.php**

2. Les routes de gestion du panier

Pour gérer les informations du panier, nous allons travailler avec les routes comme nous allons transmettre ou traiter les données de page en page. Ajoutons les routes suivantes au fichier **routes\web.php** :

```
// Les routes de gestion du panier
Route::get('basket', "BasketController@show")->name('basket.show');
Route::post('basket/add/{product}', "BasketController@add")->name('basket.add');
Route::get('basket/remove/{product}', "BasketController@remove")->name('basket.remov
Route::get('basket/empty', "BasketController@empty")->name('basket.empty');
```

Les routes nommées :

- « basket.show » pour afficher le panier
- « basket.add » pour ajouter ou mettre à jour un produit du panier
- « basket.remove » pour retirer un produit du panier
- « basket.empty » pour vider les produits du panier

Les actions de ces routes sont gérées au contrôleur **App\Http\Controllers\BasketController.php** que nous pouvons générer en exécutant la commande *artisan* suivante :

```
php artisan make:controller BasketController
```

Nous allons compléter les méthodes de ce contrôleur en avançant mais nous pouvons déjà insérer l'action du formulaire d'ajout d'un produit au panier sur la vue **resources\views\products\product.blade.php** :

```
<form method="POST" action="{{ route('basket.add', $product) }}" class="form-inline
 {{ csrf_field() }}
 <input type="number" name="quantity" placeholder="Quantité ?" class="form-control m
 <button type="submit" class="btn btn-warning" >+ Ajouter au panier</button>
</form>
```

L'arborescence du projet devient :

- app\Http\Controllers**BasketController.php**
- app\Http\Controllers**ProductController.php**
- app**Product.php**
- database\migrations**..._create_products_table.php**
- resources\views\products**product.blade.php**
- routes**web.php**

3. Le panier dynamique

Pour mettre en place le système de panier dynamique, travaillons avec la **session HTTP** en implémentant le **pattern Repository** pour la gestion des informations du panier.

Le choix se porte sur la session HTTP pour accéder aux informations du panier sur toutes les pages du site web et les traiter simplement; et sur le pattern Repository pour créer une abstraction sur la gestion des données : séparer la logique de gestion des produits et leur persistance dans le panier par la session HTTP.

Le pattern Repository permettra aussi au besoin d'implémenter un autre support que la session HTTP pour persister les informations du panier, tels qu'une base de données.

L'arborescence finale du projet que nous allons obtenir ou les fichiers que nous allons éditer pour le système sont :

- app\Http\Controllers**BasketController.php**
- app\Http\Providers**BasketServiceProvider.php**
- app\Http\Repositories**BasketInterfaceRepository.php**
- app\Http\Repositories**BasketSessionRepository.php**
- app\Http\Controllers**ProductController.php**
- app**Product.php**
- config**app.php**
- database\migrations**..._create_products_table.php**
- resources\views\basket**show.blade.php**
- resources\views\products**product.blade.php**
- routes**web.php**

Décrivons ou complétons ces fichiers :

3.1. L’interface "BasketInterfaceRepository"

L’interface **App\Http\Repositories\BasketInterfaceRepository.php** définit les méthodes qui doivent être décrites dans une classe qui implémente la gestion du panier, sans décrire le fonctionnement de ces méthodes :

```
<?php

namespace App\Repositories;

use App\Product;

interface BasketInterfaceRepository {

 // Afficher le panier
 public function show();

 // Ajouter un produit au panier
 public function add(Product $product, $quantity);

 // Retirer un produit du panier
 public function remove(Product $product);

 // Vider le panier
 public function empty();

}

?>
```

3.2. La classe "BasketSessionRepository"

La classe **App\Http\Repositories\BasketSessionRepository.php** décrit la logique de gestion du panier par la session HTTP, suivant les méthodes définies à l'interface **App\Http\Repositories\BasketInterfaceRepository.php** qu'elle implémente :

```php
<?php

namespace App\Repositories;

use App\Product;

class BasketSessionRepository implements BasketInterfaceRepository  {

 # Afficher le panier
 public function show () {
  return view("basket.show"); // resources\views\basket\show.blade.php
 }

 # Ajouter/Mettre à jour un produit du panier
 public function add (Product $product, $quantity) {
  $basket = session()->get("basket"); // On récupère le panier en session

  // Les informations du produit à ajouter
  $product_details = [
   'name' => $product->name,
   'price' => $product->price,
   'quantity' => $quantity
  ];

  $basket[$product->id] = $product_details; // On ajoute ou on met à jour le produit
  session()->put("basket", $basket); // On enregistre le panier
 }

 # Retirer un produit du panier
 public function remove (Product $product) {
  $basket = session()->get("basket"); // On récupère le panier en session
  unset($basket[$product->id]); // On supprime le produit du tableau $basket
  session()->put("basket", $basket); // On enregistre le panier
 }

 # Vider le panier
 public function empty () {
  session()->forget("basket"); // On supprime le panier en session
 }

}

?>
```

3.3. Le provider "BasketServiceProvider"

Le service provider **App\Providers\BasketServiceProvider.php** indique à l'application quel implémentation d'interface de gestion de panier utiliser. Générons-le par la commande *artisan* suivante :

```
php artisan make:provider BasketServiceProvider
```

Lions (binding) dans la method `register()` du provider l'interface **BasketInterfaceRepository** avec **BasketSessionRepository** :

```
<?php

namespace App\Providers;

use Illuminate\Support\ServiceProvider;

use App\Repositories\BasketInterfaceRepository;
use App\Repositories\BasketSessionRepository;

class BasketServiceProvider extends ServiceProvider
{

    public function register()
    {
        $this->app->bind(BasketInterfaceRepository::class, BasketSessionRepository::
    }

    // ...
}
```

Inscrivons maintenant le provider **BasketServiceProvider** au tableau `providers` du fichier de configuration **config\app.php** :

```
// ...
'providers' => [
        // ...,
        // Le service provider du panier
        App\Providers\BasketServiceProvider::class
],
// ...
```

3.4. Le contrôleur "BasketController"

Complétons le contrôleur **App\Http\Controllers\BasketController.php** avec les actions des routes du panier :

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

use App\Repositories\BasketInterfaceRepository;

use App\Product;

class BasketController extends Controller
```

```
{

 protected $basketRepository; // L'instance BasketSessionRepository

    public function __construct (BasketInterfaceRepository $basketRepository) {
     $this->basketRepository = $basketRepository;
    }

    # Affichage du panier
    public function show () {
     return view("basket.show"); // resources\views\basket\show.blade.php
    }

    # Ajout d'un produit au panier
    public function add (Product $product, Request $request) {

     // Validation de la requête
     $this->validate($request, [
      "quantity" => "numeric|min:1"
     ]);

     // Ajout/Mise à jour du produit au panier avec sa quantité
     $this->basketRepository->add($product, $request->quantity);

     // Redirection vers le panier avec un message
     return redirect()->route("basket.show")->withMessage("Produit ajouté au panier"
    }

    // Suppression d'un produit du panier
    public function remove (Product $product) {

     // Suppression du produit du panier par son identifiant
     $this->basketRepository->remove($product);

     // Redirection vers le panier
     return back()->withMessage("Produit retiré du panier");
    }

    // Vider la panier
    public function empty () {

     // Suppression des informations du panier en session
     $this->basketRepository->empty();

     // Redirection vers le panier
     return back()->withMessage("Panier vidé");

    }
```

```
}
```

Enfin, le code source de la vue **resources\views\basket\show.blade.php** qui affiche le panier et permet de le gérer :

```
@extends("layouts.app")
@section("content")
<div class="container">

 @if (session()->has('message'))
 <div class="alert alert-info">{{ session('message') }}</div>
 @endif

 @if (session()->has("basket"))
 <h1>Mon panier</h1>
 <div class="table-responsive shadow mb-3">
  <table class="table table-bordered table-hover bg-white mb-0">
   <thead class="thead-dark" >
    <tr>
     <th>#</th>
     <th>Produit</th>
     <th>Prix</th>
     <th>Quantité</th>
     <th>Total</th>
     <th>Opérations</th>
    </tr>
   </thead>
   <tbody>
    <!-- Initialisation du total général à 0 -->
    @php $total = 0 @endphp

    <!-- On parcourt les produits du panier en session : session('basket') -->
    @foreach (session("basket") as $key => $item)

     <!-- On incrémente le total général par le total de chaque produit du panier --
     @php $total += $item['price'] * $item['quantity'] @endphp
     <tr>
      <td>{{ $loop->iteration }}</td>
      <td>
       <strong><a href="{{ route('product.show', $key) }}" title="Afficher le produi
      </td>
      <td>{{ $item['price'] }} $</td>
      <td>
       <!-- Le formulaire de mise à jour de la quantité -->
       <form method="POST" action="{{ route('basket.add', $key) }}" class="form-inli
       {{ csrf_field() }}
        <input type="number" name="quantity" placeholder="Quantité ?" value="{{ $ite
        <input type="submit" class="btn btn-primary" value="Actualiser" />
       </form>
      </td>
      <td>
```

```
      <!-- Le total du produit = prix * quantité -->
      {{ $item['price'] * $item['quantity'] }} $
     </td>
     <td>
      <!-- Le Lien pour retirer un produit du panier -->
      <a href="{{ route('basket.remove', $key) }}" class="btn btn-outline-danger" t
     </td>
    </tr>
   @endforeach
   <tr colspan="2" >
    <td colspan="4" >Total général</td>
    <td colspan="2">
     <!-- On affiche total général -->
     <strong>{{ $total }} $</strong>
    </td>
   </tr>
  </tbody>

 </table>
</div>

<!-- Lien pour vider le panier -->
<a class="btn btn-danger" href="{{ route('basket.empty') }}" title="Retirer tous le

@else
<div class="alert alert-info">Aucun produit au panier</div>
@endif

</div>
@endsection
```

Nous pouvons aussi enrichir le formulaire de la vue **resources\views\products\product.blade.php** en affichant la quantité du produit s'il se trouve au panier :

```
<input type="number" name="quantity" placeholder="Quantité ?" class="form-control mr
```

Packages Laravel de gestion de panier

Pour vous inspirer ou voir d'autres approches, voici quelques packages de gestion de panier e-commerce qu'on trouve sur GitHub :

- darryldecode/laravelshoppingcart
- Lenius/laravel-basket
- DivineOmega/laravel-extendable-basket