TP6 - Walkthrough complet

Form Requests, recherche et pagination, pas à pas

Ce guide suit le travail effectué dans mini-crm. Chaque étape indique le fichier, le rôle de Laravel, la raison du changement et la manière de le vérifier.

1. État du projet avant le TP6

- TP3 : tables SQLite, modèles Eloquent, seeders et affichage des listes.
- TP4 : CRUD Client et Product complet avec Route::resource().
- TP5 : validation des formulaires, messages français et conservation des valeurs avec old().

Avant TP6, les méthodes store() et update() appelaient validatedData() dans chaque contrôleur. Le sujet pédagogique illustre un ancien code avec $request->all(), mais ce n'était pas le code réel du projet : les valeurs étaient déjà validées au TP5. Ce détail compte pour comprendre ce que le TP6 modifie réellement.

2. Lire le sujet et décider de l'architecture

Le travail comporte quatre Form Requests (création et modification pour les deux ressources), une recherche Client dans trois colonnes, une recherche Product par nom et prix pour le bonus, puis une pagination de dix lignes par page. Le CRUD et la base ne sont pas recréés.

Pourquoi sortir la validation du contrôleur ?

Le contrôleur coordonne les étapes HTTP. Les règles, l'autorisation et les messages forment un ensemble cohérent dans une classe Form Request. L'injection de dépendances fait exécuter la validation avant d'entrer dans la méthode du contrôleur.

3. Créer les deux Form Requests Client

```
php artisan make:request StoreClientRequest
php artisan make:request UpdateClientRequest
```

Ces commandes génèrent normalement app/Http/Requests. Les deux fichiers ont été créés avec le contenu équivalent. StoreClientRequest est utilisé par store() ; UpdateClientRequest par update().

authorize()

```
public function authorize(): bool
{
    return true;
}
```

Le TP n'a pas encore de gestion des droits. true permet de poursuivre vers rules(). Si authorize() renvoyait false, Laravel répondrait 403 avant la validation.

rules() et messages()

```
'name'  => ['required', 'string', 'max:100'],
'email' => ['required', 'email', Rule::unique('clients', 'email')],
'phone' => ['nullable', 'string', 'max:20'],
'city'  => ['nullable', 'string', 'max:100'],
```

Ces règles reproduisent le TP5. messages() garde les explications françaises déjà visibles sous les champs. L'unicité de l'email va au-delà du minimum écrit dans le TP6, car la base possède déjà une contrainte UNIQUE.

La particularité de UpdateClientRequest

```
$client = $this->route('client');
Rule::unique('clients', 'email')->ignore($client)
```

Le client lié à l'URL peut conserver son propre email. Sans ignore($client), la modification d'une ville échouerait si l'email restait inchangé.

4. Simplifier ClientController

```
public function store(StoreClientRequest $request)
{
    Client::create($request->validated());
    return redirect()->route('clients.index');
}

public function update(UpdateClientRequest $request, Client $client)
{
    $client->update($request->validated());
    return redirect()->route('clients.index');
}
```

La méthode privée validatedData() du TP5 a été retirée du contrôleur : les règles sont dans les Form Requests. validated() retourne les valeurs autorisées par ces règles ; all() retournerait également les autres entrées envoyées. Les formulaires Blade TP5 et leurs messages restent utilisables.

5. Comprendre l'injection de dépendances

La signature store(StoreClientRequest $request) déclare ce dont la méthode a besoin. Laravel construit l'objet, vérifie authorize() puis rules(), et appelle store() seulement si la validation réussit. La signature update(UpdateClientRequest $request, Client $client) ajoute le Client trouvé par route model binding à partir de /clients/{client}.

Cas invalide

POST /clients avec un nom vide et un email incorrect ne rentre pas dans ClientController@store : Laravel redirige vers le formulaire avec les erreurs et les anciennes valeurs. Les tests vérifient que le comportement TP5 est conservé.

Correction d'un ancien détail du projet

Le contrôleur de base Controller.php contenait une méthode index() vide, souvenir d'un ancien exercice. Sa signature sans paramètre empêchait ClientController::index(Request $request). Elle a été retirée ; chaque contrôleur possède maintenant son index() adapté.

6. Ajouter la recherche Client

Dans ClientController@index(Request $request), search est lu depuis l'URL. Une valeur non textuelle est traitée comme vide et les espaces externes sont retirés. La requête part de Client::query(), ce qui permet d'ajouter des conditions seulement si nécessaire.

```
$searchInput = $request->query('search', '');
$search = is_string($searchInput) ? trim($searchInput) : '';
$query = Client::query();

if ($search !== '') {
    $query->where(function ($query) use ($search) {
        $query->where('name', 'like', '%'.$search.'%')
              ->orWhere('email', 'like', '%'.$search.'%')
              ->orWhere('city', 'like', '%'.$search.'%');
    });
}
```

La fonction autour des trois conditions crée un groupe logique : nom OU email OU ville. Par exemple search=Paris peut correspondre à city. Sans search, aucun where n'est ajouté et tous les clients sont éligibles. Laravel lie la valeur comme paramètre SQL.

7. Formulaire GET dans la vue

```
<form method="GET" action="{{ route('clients.index') }}">
  <input type="search" name="search" value="{{ $search }}">
  <button type="submit">Rechercher</button>
</form>
```

GET est le verbe adapté à une consultation. L'utilisateur voit /clients?search=jean dans l'URL. La valeur reste dans le champ lors du nouvel affichage. Un lien Effacer revient à /clients sans filtre.

8. Paginer les clients

```
$clients = $query->orderBy('id')
    ->paginate(10)
    ->withQueryString();
```

paginate(10) demande à SQLite au plus dix lignes pour la page affichée, plus une requête de comptage. orderBy('id') assure un ordre prévisible entre pages. withQueryString() conserve search dans les URL suivantes. Dans clients/index.blade.php, $clients->links() affiche les liens et $clients->total() affiche le nombre total de résultats, pas seulement ceux de la page.

Exemple de résultat

URL	Effet
/clients	Tous les clients, dix au maximum par page.
/clients?search=jean	Clients dont le nom, l'email ou la ville contient jean.
/clients?search=jean&page=2	Résultats suivants, avec le filtre jean conservé.

Le TP demandait d'avoir plus de dix clients pour tester la page 2. Les tests créent douze clients uniquement dans SQLite en mémoire ; ils vérifient dix lignes en page 1 et deux en page 2. La base réelle n'a pas été remplie d'exemples artificiels.

9. Créer les Form Requests Product

```
php artisan make:request StoreProductRequest
php artisan make:request UpdateProductRequest
```

Les deux classes ont authorize() = true. Leurs règles et messages reprennent le TP5 : name est une chaîne obligatoire de 150 caractères au plus ; price est numérique et non négatif ; stock est un entier non négatif.

```
'name'  => ['required', 'string', 'max:150'],
'price' => ['required', 'numeric', 'min:0'],
'stock' => ['required', 'integer', 'min:0'],
```

ProductController@store(StoreProductRequest $request) et @update(UpdateProductRequest $request, Product $product) utilisent validated(). La méthode privée de validation a été retirée. Les vues de création et de modification du TP5 continuent d'afficher les mêmes erreurs.

10. Recherche et pagination Product

ProductController@index construit Product::query(), filtre le nom avec LIKE et ajoute le prix pour le challenge bonus. Les deux conditions sont groupées par where(function ...). Ensuite la requête est triée par id, paginée par dix et complétée avec withQueryString(). La vue products/index.blade.php reçoit $products et $search, affiche un formulaire GET puis $products->links().

```
$query->where(function ($query) use ($search) {
    $query->where('name', 'like', '%'.$search.'%')
          ->orWhere('price', 'like', '%'.$search.'%');
});
```

Dans SQLite, LIKE convertit la valeur de prix en texte pour ce type de recherche. Une recherche « 15 » trouve donc un prix 15,50. Ce bonus sert à retrouver un prix contenant le texte saisi ; pour une recherche financière exacte, on utiliserait une comparaison numérique.

11. Repères TP6 dans le code

Les commentaires `TP6` marquent les nouvelles responsabilités dans les quatre Form Requests, les deux contrôleurs, les vues index, le layout et le contrôleur de base. Le fichier REPERES_TP6.md relie chaque fichier à son exercice. Les commentaires sont invisibles dans le navigateur et ne changent pas les résultats.

12. Vérifications automatisées

La suite Laravel comprend maintenant **11 tests réussis et 131 assertions**. Les tests existants vérifient toujours le CRUD TP4 et la validation TP5. Tp6SearchPaginationTest ajoute les cas propres au TP6.

Test	Ce qui est démontré
Clients par nom	Jean est trouvé ; Marie n'apparaît pas.
Clients par email et ville	gmail et Bordeaux retrouvent la bonne personne.
Clients sans filtre	Toutes les lignes sont éligibles.
Clients en deux pages	12 résultats : 10 sur la première, 2 sur la deuxième ; search est conservé.
Produits par nom	12 claviers : dix puis deux ; filtre conservé.
Bonus par prix	Une recherche « 15 » retrouve la Souris à 15,50.
Form Requests	Données invalides refusées ; modification avec email inchangé acceptée.

La configuration phpunit.xml utilise SQLite en mémoire : ces vérifications ne changent pas database/database.sqlite. Aucune migration de schéma n'était nécessaire pour le TP6.

13. Refaire le TP toi-même dans le terminal

```
cd C:\Users\Junior\Desktop\BUT2\BUT2\mini-crm
php artisan route:list
php artisan test
php artisan serve
```

Ouvre http://127.0.0.1:8000/clients puis cherche Jean, Paris ou un fragment d'email. Pour voir la pagination dans ta base personnelle, il faut y avoir au moins onze clients. Les trois clients initiaux du TP3 n'affichent pas encore de page 2 ; le test automatisé la démontre avec douze clients temporaires.

Ouvre http://127.0.0.1:8000/products puis cherche Clavier ou un fragment de prix comme 15. Il faut au moins onze produits dans ta base personnelle pour voir une deuxième page ; les cinq exemples TP3 ne suffisent pas. Les tests le vérifient avec des données temporaires.

14. Réponses rapides à expliquer au professeur

- Un contrôleur n'a pas à contenir toute la validation : les Form Requests isolent cette responsabilité.
- authorize() renvoie true ici parce que le TP n'implémente pas encore les droits d'accès.
- validated() fournit seulement les valeurs acceptées par rules(), contrairement à all().
- Laravel injecte les objets demandés dans la signature d'une méthode et lie le Model à l'identifiant d'URL.
- GET rend la recherche visible dans l'URL ; withQueryString() la garde pendant la pagination.
- total() compte les résultats complets ; count() compte seulement la page courante.

Conclusion

Le CRUD Client et Product dispose maintenant de Form Requests, d'une recherche et d'une pagination cohérentes. Les validations TP5 et le CRUD TP4 restent fonctionnels, avec des repères TP6 pour retrouver facilement chaque changement.