TP5 - Walkthrough de A à Z

Sécuriser et fiabiliser le CRUD Laravel

Ce guide documente toutes les modifications réalisées dans mini-crm pour le TP5. Il indique pourquoi chaque fichier a changé, comment Laravel traite une saisie et comment reproduire les vérifications.

1. Comprendre le point de départ

- Le TP4 possédait déjà ClientController et ProductController, les routes Resource, les vues CRUD et des validations simples.
- La base TP3 possédait **clients.city obligatoire**, aucun **clients.phone**, et un prix produit entier. Ces choix empêchaient certaines règles TP5 de fonctionner réellement.
- Les enregistrements existants ont été conservés. Une copie **database/database.before-tp5.sqlite** a été créée avant la migration.

2. Distinguer navigateur et serveur

Un champ HTML required ou type=email signale une erreur dans le navigateur. Il ne protège pas la base : un client HTTP peut envoyer directement POST /clients. La méthode $request->validate(...) applique les règles côté Laravel, avant tout appel Eloquent.

Flux validé

Formulaire → Request → Controller → validate() → (erreurs + retour au formulaire) ou (Eloquent → SQLite → redirect()).

3. Adapter la base de données

Fichier : database/migrations/2026_10_07_000000_adapt_clients_and_products_for_tp5.php. Une nouvelle migration transforme le schéma déjà installé, ce qui évite de réécrire l'historique TP3 et de perdre ses données.

```
Schema::table('clients', function (Blueprint $table) {
    $table->string('phone', 20)->nullable();
    $table->string('city', 100)->nullable()->change();
});

Schema::table('products', function (Blueprint $table) {
    $table->decimal('price', 10, 2)->change();
});
```

- phone peut être NULL et sa longueur maximale est 20.
- city peut être NULL ; son maximum est aligné sur les 100 caractères demandés.
- price accepte désormais deux décimales. Une validation numeric avec une colonne INTEGER aurait une incohérence pour 15,50 €.

Commande appliquée : **php artisan migrate --force**. Le statut de la migration TP5 est Ran. La copie avant migration reste disponible pour récupérer les données si nécessaire.

4. Autoriser la sauvegarde du téléphone

Dans app/Models/Client.php, phone a été ajouté à $fillable. Sans cela, Client::create() et update() ignoreraient cette valeur ou produiraient une erreur d'affectation de masse selon la configuration. Le $fillable Produit existant couvre déjà name, price et stock.

5. Valider les clients

Fichier : app/Http/Controllers/ClientController.php. store() et update() appellent une même méthode privée validatedData(). Cela garantit des règles identiques à la création et à la modification.

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

Le nom ne peut pas dépasser 100 caractères. L'email doit être valide et unique, ce qui correspond aussi à la contrainte SQL. Pendant update(), ignore($client) permet au client de garder son propre email. Les champs phone et city peuvent être absents ou vides.

Une fois validate() réussi, seules les valeurs validées sont transmises au Model :

```
Client::create($this->validatedData($request));
$client->update($this->validatedData($request, $client));
```

Si validate() échoue, l'exécution s'arrête avant create() ou update(). Laravel redirige automatiquement avec les erreurs et les anciennes valeurs. Les messages français personnalisés incluent « Le nom du client est obligatoire. » et « L’adresse e-mail n’est pas valide. »

Précision sur le téléphone

Le sujet propose au début d'essayer des lettres dans le téléphone, mais impose ensuite nullable|string|max:20. Une chaîne peut contenir des lettres : **poste ABC** est donc accepté. Pour refuser les lettres, il faudrait une règle supplémentaire (regex), absente des critères du TP5.

6. Rendre les erreurs visibles pour les clients

Fichiers : resources/views/clients/create.blade.php et edit.blade.php. Les deux formulaires possèdent maintenant trois mécanismes complémentaires.

- En haut, $errors->any() affiche une liste complète, parcourue avec $errors->all().
- Sous chaque champ, @error('name'), @error('email'), @error('phone') et @error('city') affichent le message précis.
- La valeur du formulaire est récupérée avec old('champ'). Pour l'édition, old('champ', $client->champ) garde la saisie après une erreur ou montre la donnée stockée au premier affichage.

```
<input name="city" value="{{ old('city', $client->city) }}">
@error('city')
    <div role="alert">{{ $message }}</div>
@enderror
```

Le caractère obligatoire a été retiré de l'attribut HTML du champ ville, car le TP5 la rend facultative. Les vues clients/index et clients/show affichent aussi phone et un tiret quand phone ou city est vide.

Pourquoi afficher l'erreur sous le champ ?

L'utilisateur relie immédiatement le message au champ concerné. Il peut corriger sans chercher dans la liste globale ni refaire les autres saisies.

7. Valider les produits

Fichier : app/Http/Controllers/ProductController.php. La même méthode validatedData() est utilisée dans store() et update().

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

name est obligatoire et limité à 150 caractères. price accepte un nombre entier ou décimal non négatif. stock exige un entier non négatif : 3 est accepté, 3,5 ou « dix » est refusé. Des messages français ont été ajoutés pour les principales erreurs.

Les vues products/create.blade.php et edit.blade.php affichent la liste globale, chaque erreur locale et old(). Le champ price utilise step=0.01, cohérent avec la colonne DECIMAL(10,2). Les contraintes HTML aident la saisie ; la validation serveur reste l'autorité.

8. La création d'un client, étape par étape

Étape	Action de Laravel
1. GET /clients/create	La route clients.create appelle ClientController@create.
2. Vue	Blade affiche le formulaire avec @csrf et les champs name, email, phone, city.
3. POST /clients	La route clients.store appelle ClientController@store avec Request.
4. Validation	Laravel vérifie les règles ; en cas d'échec, retour avec erreurs et old().
5. Écriture	Si tout est valide, Client::create() écrit dans SQLite.
6. Redirection	redirect()->route('clients.index') affiche la liste mise à jour.

9. La modification, étape par étape

GET /clients/{client}/edit charge le Client grâce au route model binding et préremplit le formulaire. Le navigateur soumet un POST avec @method('PUT') et @csrf. Laravel l'associe à clients.update. update() valide les nouvelles valeurs, puis appelle $client->update(). Si la validation échoue, le client en base garde ses anciennes valeurs et le formulaire récupère la saisie avec old().

Le même cycle s'applique à /products/{product}/edit, avec les règles name, price et stock. Les méthodes show() et destroy() du TP4 restent en place.

10. Tests réalisés

Les tests automatisés utilisent SQLite en mémoire et RefreshDatabase. Ils ne modifient pas la base personnelle database/database.sqlite. La suite complète passe : **7 tests, 7 réussis**.

Cas testé	Résultat
Client : nom vide	Erreur personnalisée ; aucun enregistrement créé.
Client : email absent/invalide/déjà utilisé	Erreur ; aucune écriture.
Client : nom > 100 / phone > 20 / city > 100	Erreur sur le bon champ.
Client : phone et city vides	Création acceptée avec valeurs NULL.
Client : mise à jour invalide	Erreur ; anciennes données préservées.
Produit : nom vide ou > 150	Erreur.
Produit : prix négatif ou texte	Erreur price.
Produit : stock négatif ou texte	Erreur stock.
Produit : prix 15,50 et stock 10	Création acceptée ; prix décimal conservé.
Formulaires après erreur	Messages et anciennes valeurs visibles.

11. Refaire la démonstration toi-même

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

Ouvre ensuite http://127.0.0.1:8000/clients/create. Laisse le nom vide et écris « bonjour » dans l'email. Le navigateur peut bloquer cet envoi à cause de required et type=email ; pour observer la validation serveur au navigateur, remplis les champs ou utilise les tests automatisés qui envoient directement les requêtes. Essaie aussi un nom de 101 caractères. Vérifie le message sous le champ et la conservation de la ville.

Sur /clients/{id}/edit, garde l'email actuel et modifie la ville : l'unicité doit laisser passer la mise à jour. Teste ensuite une erreur : les données enregistrées doivent rester inchangées.

Sur /products/create, essaie un prix négatif, un stock négatif, puis un prix 15.50 et un stock 10. Le dernier envoi doit créer le produit. Après une démonstration manuelle, tu peux supprimer uniquement les enregistrements d'essai via l'interface CRUD.

12. Réponse à la question du sujet

L'erreur sous le champ indique immédiatement où agir, tandis que la liste globale donne une vue d'ensemble. Les valeurs old() évitent de ressaisir les champs corrects. Sur un échec, Laravel ne crée ni ne modifie l'enregistrement.

13. Points à retenir

- Définir les mêmes règles pour store() et update().
- Passer uniquement les valeurs validées à create() et update().
- Faire correspondre schéma SQL, validation serveur et formulaires HTML.
- Montrer les erreurs et conserver la saisie.
- Tester les refus et vérifier qu'aucune donnée invalide n'est écrite.