NaturaCorp

Tech4business

Rapport Individuel – Projet NaturaCorp

Livrable 4.3 – Maintenance évolutive

Réalisé par :

SELLIER Luka

Formation : Bachelor 3 Développement WEB

Année universitaire : 2024–2025

Projet d'évolution fonctionnelle Maintenance évolutive de la solution numérique NaturaCorp

Juin 2025

Sommaire

1 Introduction

1.1 Contexte du projet et de la roadmap

Dans le cadre du livrable 4.3 de maintenance évolutive du projet NaturaCorp, j'ai été chargé d'implémenter de nouvelles fonctionnalités visant à améliorer l'expérience utilisateur et à enrichir les capacités d'analyse des données pour les commerciaux et les administrateurs. NaturaCorp est une entreprise spécialisée dans la distribution de produits naturels aux pharmacies, et sa plateforme numérique joue un rôle crucial dans la gestion des relations commerciales et l'analyse des performances.

La roadmap du projet prévoyait plusieurs évolutions fonctionnelles pour cette phase, notamment l'enrichissement du tableau de bord commercial avec des visualisations graphiques et des fonctionnalités d'export, ainsi que l'amélioration de la page "Rapports" pour les administrateurs avec des capacités d'analyse et d'export plus avancées.

1.2 Objectifs de la fonctionnalité choisie

Pour ce livrable, j'ai choisi de me concentrer sur l'implémentation des fonctionnalités suivantes :

- Afficher sous forme de graphique le nombre de ventes du commercial sur une période donnée pour le tableau de bord des commerciaux
- Afficher sous forme de graphique la liste des clients rapportés par le commercial sur une période donnée pour le tableau de bord des commerciaux
- Permettre le téléchargement en CSV séparé de chaque graphique ou tableau de données pour le tableau de bord des commerciaux
- Enrichir la page "Rapports" des administrateurs avec des fonctionnalités d'export CSV pour les tableaux de pharmacies par zone et de performances des commerciaux
- Ajouter des visualisations graphiques pour la répartition des pharmacies et des commandes par zone et par commercial pour la page "Rapports" coté administrateur

Ces fonctionnalités visent à fournir aux utilisateurs des outils d'analyse visuelle plus puissants et à faciliter l'exploitation des données en permettant leur export dans un format standard. L'objectif principal est d'améliorer la prise de décision basée sur les données et de renforcer le suivi des performances commerciales.

2 Analyse préalable

2.1 Enjeux fonctionnels, techniques ou utilisateurs

L'analyse préalable du projet a permis d'identifier plusieurs enjeux majeurs :

— **Enjeux fonctionnels** : Les commerciaux avaient besoin d'une visualisation claire de leurs performances de vente et de leur capacité à acquérir de nouveaux clients. De même, les administrateurs nécessitaient une vue d'ensemble plus détaillée de la répartition des pharmacies par zone et des performances des commerciaux, avec la possibilité d'exporter ces données pour une analyse plus approfondie.

— **Enjeux techniques** : L'implémentation de graphiques interactifs nécessitait l'intégration d'une bibliothèque JavaScript adaptée (Chart.js) et la création d'un contrôleur dédié pour la préparation et la structuration des données. La génération de fichiers CSV devait être optimisée pour gérer efficacement de grands volumes de données sans impact sur les performances du serveur.

— **Enjeux utilisateurs** : L'interface utilisateur devait rester intuitive malgré l'ajout de nouvelles fonctionnalités. Les graphiques devaient être lisibles et informatifs, avec une granularité adaptée à la période sélectionnée. Les exports CSV devaient contenir toutes les informations pertinentes tout en restant faciles à exploiter dans des outils d'analyse externes.

2.2 Contraintes identifiées

Plusieurs contraintes ont été identifiées lors de l'analyse préalable :

— **Contraintes techniques** :
  — Nécessité de maintenir la compatibilité avec l'architecture MVC existante de Laravel
  — Gestion des requêtes SQL complexes pour l'agrégation des données sans impact sur les performances
  — Adaptation de l'interface utilisateur aux différentes tailles d'écran pour garantir la lisibilité des graphiques
  — Résolution des problèmes d'ambiguïté dans les requêtes SQL impliquant plusieurs tables avec des noms de colonnes identiques

— **Contraintes fonctionnelles** :
  — Nécessité d'adapter la granularité des données (jour, semaine, mois) en fonction de la période sélectionnée
  — Cohérence entre les données affichées dans les graphiques et celles exportées en CSV
  — Gestion des périodes de temps (30 derniers jours, 3 derniers mois, etc.) de manière précise et cohérente

— **Contraintes d'intégration** :
  — Respect de la charte graphique existante et de l'expérience utilisateur globale
  — Intégration harmonieuse des nouveaux éléments d'interface dans les pages existantes

3 Conception

3.1 Schéma de base de données

Pour ce projet de maintenance évolutive, aucune modification du schéma de base de données n'a été nécessaire. Les fonctionnalités développées s'appuient sur les structures de données existantes :

- Table `orders` : Stockage des commandes avec les informations de base (date, montant, statut)
- Table `order_items` : Détails des articles commandés (quantité, prix unitaire, remise)
- Table `pharmacies` : Informations sur les pharmacies clientes (nom, adresse, zone, commercial assigné)
- Table `users` : Utilisateurs du système, incluant les commerciaux
- Table `zones` : Zones géographiques pour l'organisation des pharmacies

Les relations existantes entre ces tables ont été exploitées pour générer les données nécessaires aux graphiques et aux exports CSV.

3.2 Représentation

Bien que des maquettes détaillées n'aient pas été nécessaires pour ce projet, voici une représentation conceptuelle des éléments ajoutés :

Structure du tableau de bord commercial

- En-tête avec sélecteur de période (30 derniers jours, 3 derniers mois, 6 derniers mois, dernière année)
- Graphique des ventes avec bouton d'export CSV
- Graphique des clients rapportés avec bouton d'export CSV

Structure de la page Rapports administrateur

- Statistiques synthétiques (zone et commercial les plus performants)
- Tableau des pharmacies par zone avec boutons d'export CSV (global et par zone)
- Tableau des performances des commerciaux avec boutons d'export CSV (global et par commercial)
- Graphiques de répartition des pharmacies et des commandes par zone
- Graphiques de répartition des pharmacies et des commandes par commercial

3.3 Choix techniques argumentés

Plusieurs choix techniques ont été effectués pour garantir la qualité et la maintenabilité de la solution :

— **Chart.js pour les visualisations graphiques** : Cette bibliothèque JavaScript a été choisie pour sa légèreté, sa simplicité d'intégration et sa richesse fonctionnelle. Elle offre une grande variété de types de graphiques (lignes, barres, camemberts) et des fonctionnalités interactives comme les infobulles au survol.

— **Contrôleur dédié pour les exports CSV** : La création d'un contrôleur spécifique (`ExportController.php`) permet de centraliser la logique d'export et de faciliter la maintenance. Cette approche respecte le principe de responsabilité unique et s'intègre parfaitement dans l'architecture MVC de Laravel.

— **Méthode helper pour la gestion des périodes** : L'implémentation d'une méthode `setPeriodDates()` permet de standardiser le calcul des dates de début et de fin en fonction du type de période sélectionné, garantissant ainsi la cohérence entre l'affichage et l'export des données.

— **Granularité adaptative des données** : Le choix d'adapter automatiquement la granularité des données (jour, semaine, mois) en fonction de la durée de la période sélectionnée permet d'optimiser la lisibilité des graphiques et la pertinence des informations affichées.

— **Utilisation de Tailwind CSS** : Le framework CSS déjà présent dans le projet a été exploité pour garantir une intégration visuelle cohérente des nouveaux éléments d'interface.

4 Réalisation

4.1 Développement effectué

Le développement des fonctionnalités a été réalisé selon une approche méthodique structurée. Pour garantir un suivi rigoureux et une organisation optimale du projet, j'ai mis en place :

— **Un dépôt Git** pour le versionnement complet du code, permettant de suivre l'évolution des modifications et de sécuriser le développement

— **Un fichier de contexte** décrivant en détail les objectifs du projet, les contraintes techniques et les fonctionnalités attendues

— **Un journal d'avancées** mis à jour régulièrement pour documenter la progression, les décisions prises et les difficultés rencontrées

— **Une structure de projet claire** organisant le code selon l'architecture MVC de Laravel

Note

Le fichier de contexte et le journal d'avancées sont inclus dans les annexes du Git pour permettre de suivre en détail l'ensemble de la démarche projet. Le code complet du projet sera également sur Git : github.com/Kauluche/4.3.nc.1xprod.com

Cette approche méthodologique a permis d'assurer la qualité du code et la traçabilité des développements tout au long du projet. Voici les principales étapes techniques du développement :

1. **Création du contrôleur d'export** : Développement du fichier `ExportController.php` avec les méthodes nécessaires pour générer les exports CSV des différentes données (ventes, clients, pharmacies par zone, performances des commerciaux).
2. **Ajout des routes d'export** : Configuration des routes dans `routes/web.php` pour accéder aux différentes fonctionnalités d'export CSV.
3. **Modification du contrôleur du tableau de bord** : Enrichissement du `DashboardController.p` avec les méthodes de préparation des données pour les graphiques (`prepareSalesChartData` et `prepareClientsChartData`).
4. **Intégration des graphiques** : Ajout de Chart.js et implémentation des graphiques dans les vues `dashboard.blade.php` et `admin/reports/index.blade.php`.
5. **Correction de bugs** : Résolution d'un problème d'ambiguïté SQL dans les requêtes impliquant plusieurs tables avec des colonnes de même nom.
6. **Amélioration de l'interface utilisateur** : Optimisation de la mise en page et du style des graphiques et boutons d'export pour une meilleure expérience utilisateur.
7. **Ajout de statistiques synthétiques** : Enrichissement de la page des rapports administrateur avec des blocs de statistiques sur les zones et commerciaux les plus performants.

4.2 Captures de code ou extraits clés avec explication

Information complémentaire

Des rapports détaillés sur chacune des fonctionnalités clés développées ont été joints en annexe sur le Git. Ces documents techniques fournissent davantage de détails sur le code et l'implémentation si cela vous intéresse.

Voici quelques extraits de code clés qui illustrent les aspects importants du développement :

Listing 1 – Méthode prepareSalesChartData dans DashboardController.php

```
private function prepareSalesChartData($user, Carbon $startDate, Carbon $
{
    $pharmacyIds = $user->pharmacies()->pluck('id')->toArray();

    $salesData = [];
    $labels = [];
    $interval = 'month'; // Par defaut, intervalle mensuel

    // Determiner l'intervalle approprie en fonction de la duree
    $diffInDays = $startDate->diffInDays($endDate);

    if ($diffInDays <= 31) {
        // Periode courte (moins d'un mois) : afficher par jour
        $interval = 'day';
    } else if ($diffInDays <= 90) {
        // Periode moyenne (1-3 mois) : afficher par semaine
        $interval = 'week';
    }

    // Generation des donnees selon l'intervalle approprie
    $currentDate = clone $startDate;

    while ($currentDate <= $endDate) {
        // Code pour generer les donnees selon l'intervalle...
        // ...
    }

    return [
        'labels' => $labels,
        'data' => $salesData
    ];
}
```

Cet extrait montre comment la granularité des données est adaptée automatiquement en fonction de la durée de la période sélectionnée, ce qui permet d'optimiser la lisibilité des graphiques.

Listing 2 – Méthode d'export CSV dans ExportController.php

```
public function exportCommercialSales(Request $request)
{
    $user = Auth::user();

    // Utilisation de la methode helper pour definir les dates
    list($startDate, $endDate) = $this->setPeriodDates($request);

    $pharmacyIds = $user->pharmacies()->pluck('id')->toArray();

    // Requete pour recuperer les ventes
    $sales = Order::whereIn('pharmacy_id', $pharmacyIds)
        ->whereDate('orders.created_at', '>=', $startDate)
        ->whereDate('orders.created_at', '<=', $endDate)
        ->join('order_items', 'orders.id', '=', 'order_items.order_id')
        ->join('pharmacies', 'orders.pharmacy_id', '=', 'pharmacies.id')
        ->select(
            'orders.id',
            'pharmacies.name as pharmacy_name',
            'orders.created_at',
            DB::raw('SUM(order_items.quantity * order_items.unit_price *
        )
        ->groupBy('orders.id', 'pharmacies.name', 'orders.created_at')
        ->orderBy('orders.created_at', 'desc')
        ->get();

    // Generation du CSV
    $filename = 'ventes_' . $startDate->format('Y-m-d') . '_' . $endDate-

    $headers = [
        'Content-Type' => 'text/csv; charset=UTF-8',
        'Content-Disposition' => 'attachment; filename="' . $filename . '
    ];

    $callback = function() use ($sales) {
        $file = fopen('php://output', 'w');

        // En-tetes CSV
        fputcsv($file, ['ID', 'Pharmacie', 'Date', 'Montant total (EUR)']

        // Donnees
        foreach ($sales as $sale) {
            fputcsv($file, [
                $sale->id,
                $sale->pharmacy_name,
                $sale->created_at->format('d/m/Y'),
                number_format($sale->total_amount, 2, ',', ' ')
            ]);
        }
```

```
            fclose($file);
        };

        return response()->stream($callback, 200, $headers);
    }
```

Cet extrait illustre la méthode d'export des ventes d'un commercial au format CSV, avec la récupération des données pertinentes et leur formatage approprié.

4.3 Difficultés rencontrées et gestion

Plusieurs difficultés ont été rencontrées lors du développement et ont été résolues de manière méthodique :

- **Problème d'ambiguïté SQL** : Une erreur "Integrity constraint violation : 1052 Column 'created_at' in where clause is ambiguous" a été identifiée dans le `DashboardController.` Cette erreur était due à l'ambiguïté de la colonne `created_at` présente dans les tables `orders` et `order_items`. La solution a consisté à spécifier explicitement la table dans les clauses WHERE (`orders.created_at` au lieu de `created_at`).
- **Problème d'affichage des graphiques comprimés** : Les graphiques apparaissaient comprimés et peu lisibles sur certaines tailles d'écran. Ce problème a été résolu en modifiant les conteneurs de graphiques pour qu'ils prennent 100% de la largeur et en ajoutant des classes CSS personnalisées pour les conteneurs et les éléments canvas. Un script JavaScript a également été implémenté pour ajuster dynamiquement la taille des graphiques au chargement et au redimensionnement.
- **Incohérence dans le calcul des périodes** : Une incohérence a été détectée dans le calcul de la période "3 derniers mois" qui couvrait en réalité 4 mois. La correction a consisté à modifier `subMonths(3)` par `subMonths(2)` pour couvrir exactement 3 mois (mois courant + 2 mois précédents).
- **Synchronisation entre affichage et export** : Pour garantir que les données exportées correspondent exactement à celles affichées dans les graphiques, une méthode helper `setPeriodDates()` a été créée pour calculer les dates de début et de fin en fonction du type de période. Les liens d'export CSV ont également été mis à jour pour transmettre les paramètres de période actuellement sélectionnée.

5 Tests et validation

5.1 Scénarios de tests mis en place

Pour garantir le bon fonctionnement des fonctionnalités développées, plusieurs scénarios de tests ont été mis en place :

1. **Test des graphiques du tableau de bord commercial** :
   - Vérification de l'affichage correct des graphiques pour différentes périodes (30 derniers jours, 3 derniers mois, 6 derniers mois, dernière année)
   - Contrôle de la cohérence des données affichées avec les données présentes en base
   - Test de l'adaptation automatique de la granularité des données selon la période sélectionnée
   - Vérification des infobulles au survol des points du graphique
2. **Test des exports CSV** :
   - Vérification de la génération correcte des fichiers CSV pour chaque type d'export
   - Contrôle de la cohérence des données exportées avec celles affichées dans l'interface
   - Test des en-têtes et du formatage des données dans les fichiers CSV
   - Vérification de la gestion des caractères spéciaux et des accents
3. **Test des graphiques de la page Rapports administrateur** :
   - Vérification de l'affichage correct des graphiques circulaires et à barres
   - Contrôle de la cohérence des données affichées avec les tableaux correspondants
   - Test des légendes et des infobulles au survol

5.2 Résultats observés

Les tests ont permis de valider le bon fonctionnement des fonctionnalités développées et d'identifier quelques points d'amélioration qui ont été corrigés :

- **Graphiques du tableau de bord commercial** :
  - Les graphiques s'affichent correctement pour toutes les périodes testées
  - La granularité s'adapte automatiquement selon la période (jour, semaine, mois)
  - Les données affichées correspondent bien aux données en base
  - Les infobulles au survol affichent correctement les valeurs précises
- **Exports CSV** :
  - Les fichiers CSV sont générés correctement pour tous les types d'export
  - Les données exportées sont cohérentes avec celles affichées dans l'interface
  - Les en-têtes et le formatage des données sont corrects
  - Les caractères spéciaux et les accents sont bien gérés grâce à l'encodage UTF-8
- **Graphiques de la page Rapports administrateur** :
  - Les graphiques circulaires et à barres s'affichent correctement
  - Les données affichées sont cohérentes avec les tableaux correspondants
  - Les légendes et les infobulles au survol fonctionnent correctement

5.3 Outils utilisés

Plusieurs outils ont été utilisés pour faciliter le développement, les tests et la validation des fonctionnalités :

— **Git** : Système de contrôle de version pour suivre les modifications et gérer le code source

— **Chrome DevTools** : Pour inspecter et déboguer le code HTML, CSS et JavaScript, ainsi que pour tester la responsivité sur différentes tailles d'écran

— **Laravel Debugbar** : Extension Laravel pour surveiller les requêtes SQL, les temps de chargement et la consommation de mémoire

— **Lighthouse** : Pour évaluer les performances, l'accessibilité et les bonnes pratiques de développement web

— **Excel** : Pour vérifier la lisibilité et la cohérence des fichiers CSV exportés

6 Procédure de mise à jour

6.1 Liste des fichiers modifiés

Voici la liste complète des fichiers créés ou modifiés dans le cadre de cette maintenance évolutive :

— **Fichiers créés :**
  — `app/Http/Controllers/ExportController.php` : Nouveau contrôleur pour gérer tous les exports CSV

— **Fichiers modifiés :**
  — `app/Http/Controllers/Admin/ReportController.php` : Ajout des statistiques synthétiques et des données pour les graphiques
  — `app/Http/Controllers/DashboardController.php` : Préparation des données pour les graphiques et correction du bug SQL
  — `resources/views/admin/reports/index.blade.php` : Ajout des graphiques, des statistiques et des boutons d'export
  — `resources/views/dashboard.blade.php` : Ajout des graphiques et des boutons d'export CSV
  — `routes/web.php` : Ajout des routes pour les exports CSV

6.2 Étapes nécessaires pour intégrer la fonctionnalité dans le projet existant

Pour intégrer ces fonctionnalités dans le projet existant, suivez ces étapes :

1. **Sauvegarde préalable** : Effectuez une sauvegarde complète du projet et de la base de données avant toute modification.
2. **Mise à jour des fichiers** : Copiez les fichiers créés et modifiés dans leurs emplacements respectifs dans la structure du projet.
3. **Vérification des routes** : Assurez-vous que les nouvelles routes définies dans `routes/web.php` ne créent pas de conflits avec des routes existantes.
4. **Intégration de Chart.js** : Si Chart.js n'est pas déjà intégré dans le projet, ajoutez le script CDN dans les vues concernées ou installez-le via npm.
5. **Vérification des permissions** : Assurez-vous que les utilisateurs ont les permissions nécessaires pour accéder aux nouvelles fonctionnalités (notamment les exports CSV).
6. **Nettoyage du cache** : Exécutez les commandes Laravel pour nettoyer le cache des routes, des vues et de la configuration.
7. **Tests** : Exécutez les scénarios de tests décrits précédemment pour vérifier le bon fonctionnement des fonctionnalités.

6.3 Commandes à exécuter si des dépendances ont été ajoutées

Aucune dépendance PHP supplémentaire n'a été ajoutée pour ce projet. Cependant, pour garantir le bon fonctionnement de l'application après la mise à jour, exécutez les commandes suivantes :

```
# Nettoyer le cache des routes
php artisan route:cache

# Nettoyer le cache des vues
php artisan view:clear

# Nettoyer le cache de configuration
php artisan config:clear

# Optimiser l'autoloader de Composer
composer dump-autoload -o
```

Si vous choisissez d'installer Chart.js via npm plutôt que d'utiliser le CDN, exécutez également ces commandes :

```
# Installer Chart.js
npm install chart.js

# Compiler les assets
npm run dev
```

6.4 Précisions sur l'installation ou la mise à jour éventuelle de la base de données

Cette maintenance évolutive n'a pas nécessité de modifications du schéma de la base de données. Aucune migration ou mise à jour de la base de données n'est donc requise.

Les fonctionnalités développées s'appuient uniquement sur les structures de données existantes et les relations déjà établies entre les tables `orders`, `order_items`, `pharmacies`, `users` et `zones`.

7 Bilan personnel

7.1 Apports de cette épreuve

Cette épreuve de maintenance évolutive m'a apporté de nombreux enseignements et compétences :

- **Approfondissement de Laravel** : J'ai pu approfondir ma maîtrise du framework Laravel, notamment dans la gestion des requêtes SQL complexes, l'optimisation des performances et la structuration du code selon le pattern MVC.
- **Intégration de bibliothèques JavaScript** : L'utilisation de Chart.js m'a permis de développer mes compétences en intégration de bibliothèques JavaScript tierces et en manipulation de données pour les visualisations graphiques.
- **Gestion des exports de données** : J'ai acquis une expérience précieuse dans la génération de fichiers CSV à partir de données complexes, avec une attention particulière au formatage et à l'encodage.
- **Débogage et résolution de problèmes** : La résolution des bugs d'ambiguïté SQL et d'affichage des graphiques m'a permis de développer mes compétences en débogage et en analyse de problèmes techniques.
- **Documentation et communication technique** : La tenue du journal des avancées et la rédaction de ce rapport m'ont permis de renforcer mes compétences en documentation et en communication technique.

7.2 Retour sur la méthodologie utilisée

Pour mener à bien ce projet, j'ai adopté une méthodologie structurée qui s'est avérée efficace :

- **Initialisation d'un dépôt Git** : Dès le début du projet, j'ai mis en place un système de versionnement pour suivre les modifications et faciliter les retours en arrière si nécessaire.
- **Approche incrémentale** : J'ai développé les fonctionnalités une par une, en commençant par la création du contrôleur d'export, puis l'ajout des routes, la modification des contrôleurs existants et enfin l'intégration des graphiques.
- **Tests réguliers** : Après chaque étape de développement, j'ai effectué des tests pour vérifier le bon fonctionnement des fonctionnalités et identifier rapidement les problèmes éventuels.
- **Documentation continue** : La tenue du journal des avancées m'a permis de documenter chaque étape du développement, facilitant ainsi la rédaction du rapport final et la tracçabilité des décisions prises.
- **Refactoring et optimisation** : Après avoir implémenté les fonctionnalités de base, j'ai consacré du temps à l'optimisation du code et à l'amélioration de l'interface utilisateur.

Cette méthodologie m'a permis de livrer un produit de qualité dans les délais impartis, tout en maintenant une documentation claire et complète.

7.3 Recommandations éventuelles

Suite à cette expérience, je peux formuler quelques recommandations pour améliorer le projet NaturaCorp :

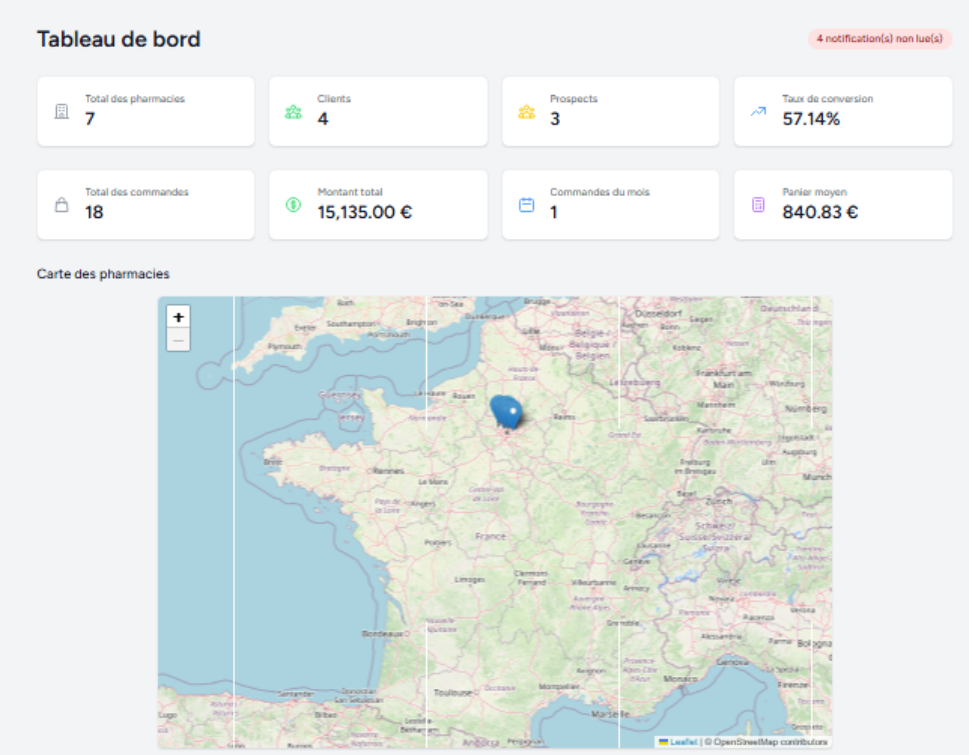
- **Standardisation des exports** : Il serait bénéfique de créer une bibliothèque ou un service dédié aux exports de données pour standardiser le format et faciliter l'ajout de nouveaux types d'exports (PDF, Excel, etc.).
- **Intégration d'un système de cache** : Pour améliorer les performances, notamment pour les graphiques et les tableaux qui nécessitent des requêtes SQL complexes, l'implémentation d'un système de cache serait pertinente.
- **Tests automatisés** : La mise en place de tests automatisés (tests unitaires, tests d'intégration) permettrait de garantir la stabilité des fonctionnalités lors des futures évolutions.
- **Amélioration de l'UX** : L'ajout de filtres supplémentaires pour les graphiques (par produit, par catégorie, etc.) et la possibilité de personnaliser les périodes d'analyse amélioreraient l'expérience utilisateur.
- **Documentation technique** : La création d'une documentation technique détaillée pour les développeurs faciliterait la maintenance et l'évolution future du projet.

Ces recommandations visent à améliorer la qualité, la maintenabilité et l'extensibilité du projet NaturaCorp pour les futures phases de développement.

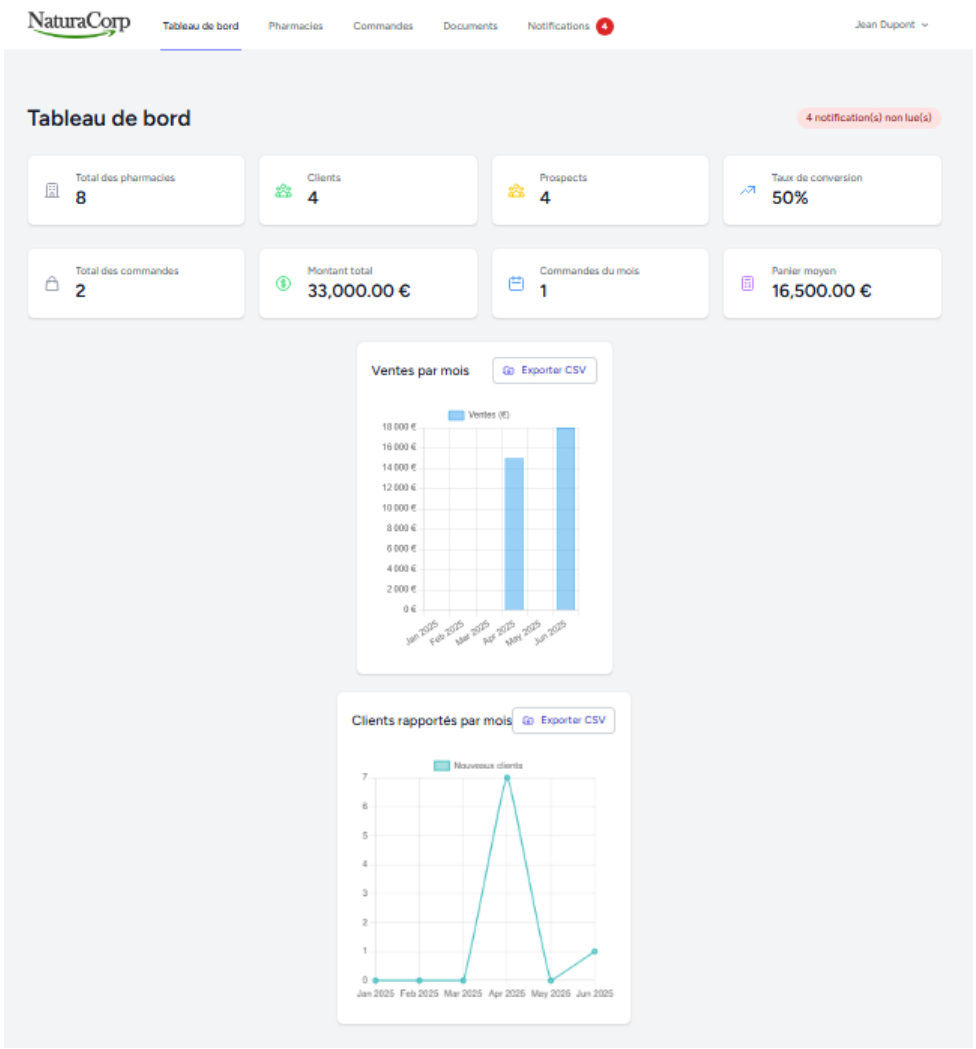
Fin du rapport

8 Annexes

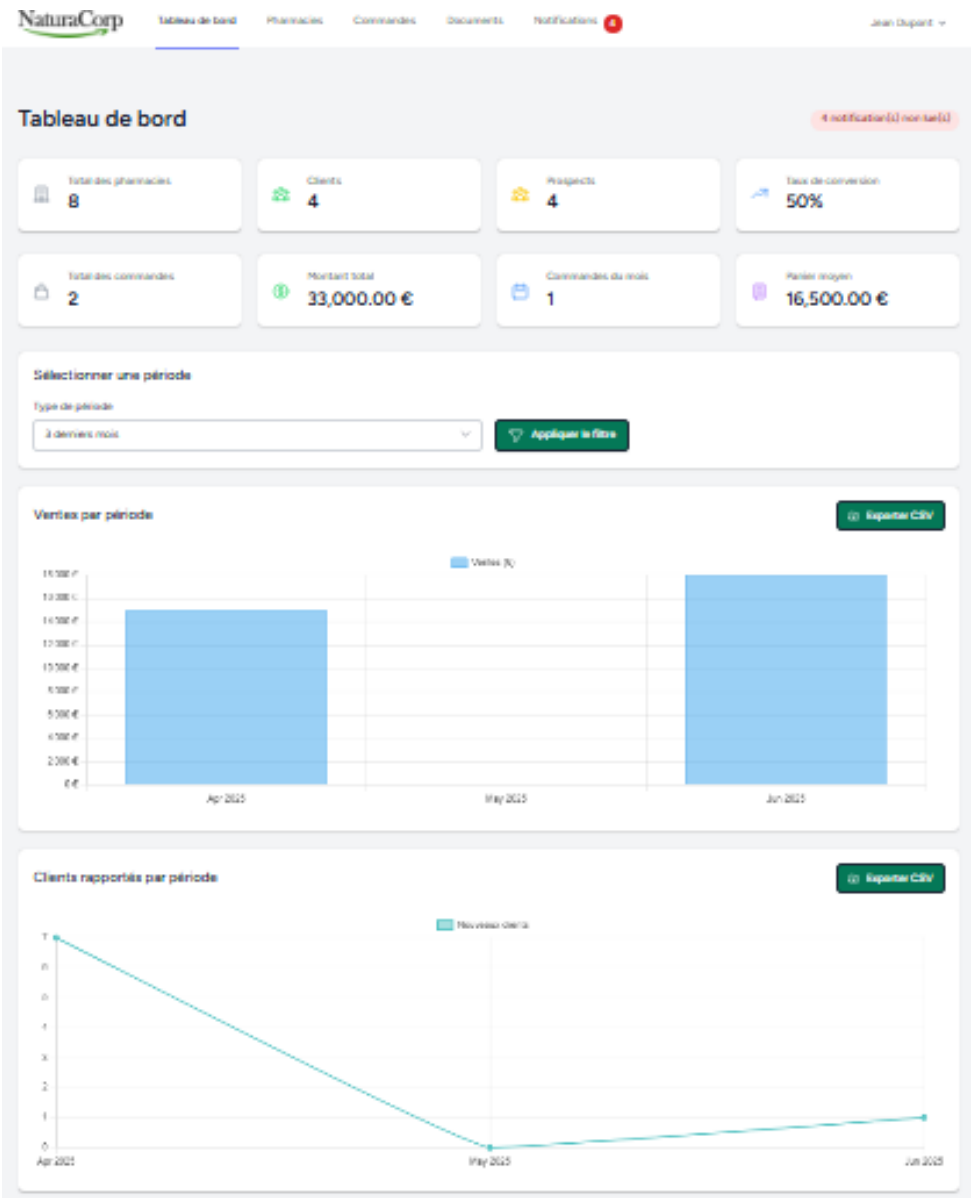
Annexe 1 : 1.0 - Base Tableau de bord Commerciaux



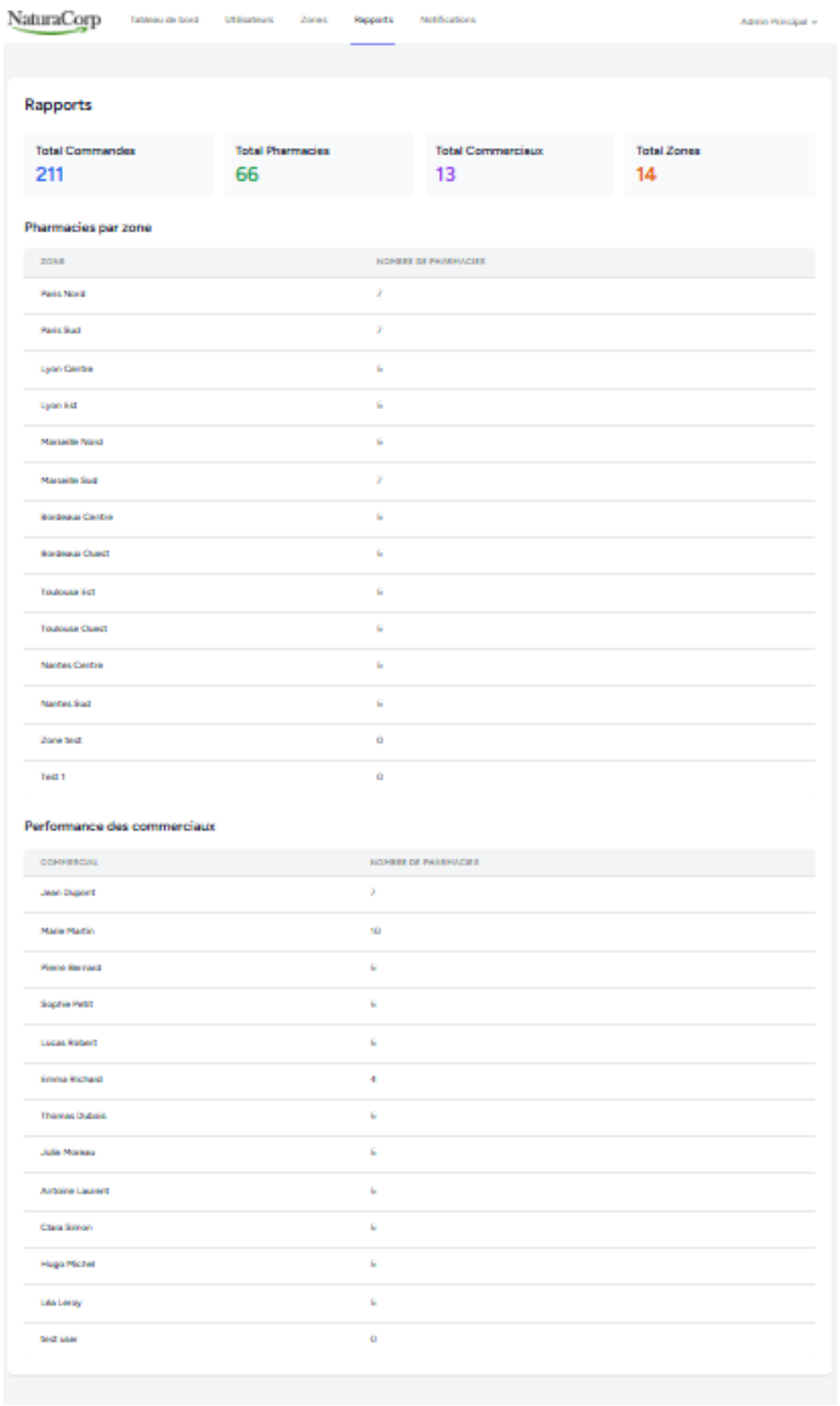
Annexe 2 : 1.0 - Mauvaise disposition des graphiques



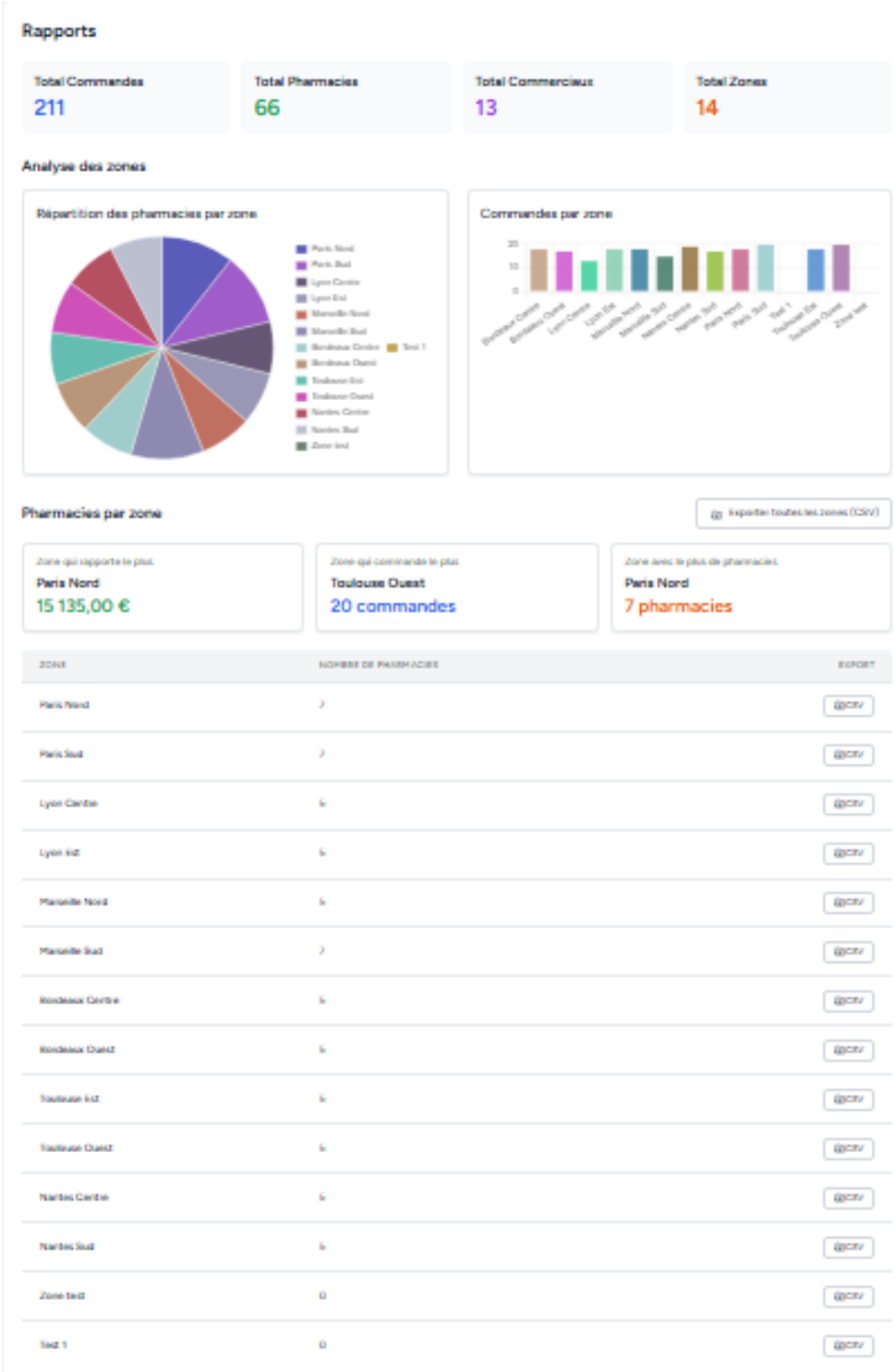
Annexe 3 : 1.5 - Graphiques Comerciaux Terminés



Annexe 4 : 2.0 - Base de la page rapport Admin



Annexe 5 : 2.5 - Stats Admin pour zone



Annexe 6 : 2.5 - Stats admin pour commerciaux