Rapport Final du Projet ERPNextClone

1. Introduction

Ce document détaille le projet ERPNextClone, un système simplifié de planification des ressources d'entreprise (ERP) développé en PHP natif avec une architecture MVC. L'objectif principal était de créer une application web modulaire permettant de gérer des fonctionnalités clés d'un ERP, telles que la gestion des produits, des tiers (fournisseurs, clients), les cycles d'approvisionnement et de vente, la gestion des stocks, et la génération de rapports.

Les fonctionnalités clés implémentées incluent :

- Gestion des Produits : Création, modification, suppression et listage des produits, y compris la gestion des prix d'achat et de vente.
- Gestion des Tiers : Modules distincts pour les Fournisseurs et les Clients, avec CRUD pour chacun.
- Cycle d'Approvisionnement :
    - Commandes Fournisseurs : Création et suivi des commandes.
    - Réceptions de Livraisons : Enregistrement des produits reçus, mise à jour automatique du stock et du statut des commandes fournisseurs.
    - Factures Fournisseurs : Enregistrement et suivi des factures des fournisseurs.
- Cycle de Vente :
    - Ventes : Enregistrement des ventes (paiement immédiat ou différé), gestion des clients (enregistrés ou occasionnels), décrémentation automatique du stock.
    - Suivi des Paiements : Enregistrement des paiements pour les ventes différées.
- Gestion des Stocks :
    - Suivi précis des mouvements de stock (entrées, sorties, ajustements).
    - Maintien d'un cache de la quantité en stock au niveau du produit, synchronisé avec les mouvements.
    - Historique des mouvements par produit et possibilité d'ajustements manuels.
- Rapports : Génération de rapports pour l'état du stock, les entrées/sorties de stock, les ventes et les achats, avec options de filtrage.
- Interface Utilisateur : Une interface web responsive avec une navigation claire et des formulaires pour interagir avec le système.

Le projet met l'accent sur une structure de code claire, la séparation des préoccupations (MVC), et l'utilisation de PHP natif sans frameworks externes majeurs pour la logique backend.

2. Architecture Logicielle

2.1. Architecture MVC

Le projet est structuré selon le motif de conception Modèle-Vue-Contrôleur (MVC) :

- **Modèles (`app/models/`) :** Représentent la logique métier et l'interaction avec la base de données. Chaque entité principale (Produit, Client, Vente, etc.) possède son propre modèle qui hérite d'une classe `Model` de base. Ils sont responsables de la validation des données, de l'exécution des requêtes SQL (via la classe `Database`), et du retour des données au contrôleur.
- **Vues (`app/views/`) :** Sont responsables de la présentation des données à l'utilisateur. Elles sont principalement composées de fichiers PHP contenant du HTML et des boucles/conditions PHP simples pour afficher les données passées par les contrôleurs. Un système de layout principal (`layouts/main.php`) est utilisé pour maintenir une structure de page cohérente.
- **Contrôleurs (`app/controllers/`) :** Agissent comme intermédiaires entre les Modèles et les Vues. Ils reçoivent les requêtes de l'utilisateur (via le routeur), interagissent avec les modèles pour récupérer ou modifier des données, puis sélectionnent et passent les données à la vue appropriée pour l'affichage. Chaque module fonctionnel a généralement son propre contrôleur (ex: `ProductsController`, `SaleController`).

2.2. Composants `core/`

Le dossier `core/` contient les classes fondamentales de l'application :

- **`Database.php` :** Une classe Singleton responsable de la connexion à la base de données (PostgreSQL via PDO) et de l'exécution des requêtes SQL (SELECT, INSERT, UPDATE, DELETE). Elle est utilisée par tous les modèles pour interagir avec la base de données.
- **`Controller.php` :** Une classe de base pour tous les contrôleurs de l'application. Elle fournit des méthodes utilitaires communes, comme `loadModel()` pour instancier un modèle spécifique et `renderView()` pour charger une vue et lui transmettre des données (y compris l'intégration dans le layout principal).
- **`Model.php` :** Une classe de base pour tous les modèles. Elle reçoit une instance de la classe `Database` dans son constructeur, la rendant disponible à tous les modèles enfants pour leurs opérations sur la base de données.

2.3. Structure des Dossiers

L'arborescence du projet est organisée comme suit :

```
/
|-- app/
|   |-- controllers/  // Contient tous les contrôleurs
|   |-- models/       // Contient tous les modèles de données
|   |-- views/        // Contient toutes les vues, organisées par module
|       |-- layouts/      // Layouts HTML principaux (ex: main.php)
|       |-- products/
|       |-- suppliers/
|       |-- clients/
|       |-- procurement/
|           |-- purchase_orders/
|           |-- deliveries/
|           |-- supplier_invoices/
|       |-- sales/
|       |-- stock/
|       |-- reports/
|       |-- errors/       // Vues pour les erreurs (404, 403, etc.)
|-- core/               // Classes de base (Database, Controller, Model)
|-- config/             // Fichiers de configuration
|   |-- database.php    // Configuration de la base de données
|-- public/             // Dossier racine web, accessible publiquement
|   |-- css/            // Fichiers CSS
|       |-- style.css
|   |-- js/             // Fichiers JavaScript (si nécessaire)
|   |-- images/         // Images
|   |-- index.php       // Point d'entrée unique de l'application (Routeur)
|-- docs/               // Documentation du projet
|   |-- database_schema.sql // Script SQL complet de la base de données
|   |-- sequence_diagrams/  // Diagrammes de séquence (textuels)
|   |-- Rapport_Final.md    // Ce rapport
|-- .htaccess           // (Optionnel, pour la réécriture d'URL avec Apache)
```

3. Schéma de la Base de Données

Le schéma complet de la base de données PostgreSQL est défini dans le fichier `docs/database_schema.sql`.

Principales tables et relations :

- **`products` :** Stocke les informations sur les produits (nom, description, prix d'achat/vente, unité de mesure, quantité en stock (cache)). `parent_id` permet une structure hiérarchique simple.
- **`suppliers` :** Informations sur les fournisseurs.
- **`clients` :** Informations sur les clients, avec un type (`connu`, `occasionnel`).

- **`purchase_orders` :** En-têtes des commandes fournisseurs (lie `suppliers`).
- **`purchase_order_items` :** Lignes des commandes fournisseurs (lie `purchase_orders` et `products`).
- **`deliveries` :** En-têtes des réceptions de livraison (peut lier `purchase_orders` et/ou `suppliers`).
- **`delivery_items` :** Lignes des réceptions (lie `deliveries`, `products`, et optionnellement `purchase_order_items`).
- **`supplier_invoices` :** Factures des fournisseurs (lie `suppliers`, et optionnellement `deliveries` ou `purchase_orders`).
- **`sales` :** En-têtes des ventes (peut lier `clients` ou utiliser `client_name_occasional`).
- **`sale_items` :** Lignes des ventes (lie `sales` et `products`).
- **`stock_movements` :** Journal de tous les mouvements de stock (lie `products`, et optionnellement le document source via `related_document_id` et `related_document_type`).

Des triggers PostgreSQL sont utilisés pour mettre à jour automatiquement les champs `updated_at` sur les tables concernées. Des contraintes de clé étrangère assurent l'intégrité référentielle (ex: `ON DELETE RESTRICT`, `ON DELETE SET NULL`, `ON DELETE CASCADE` selon le cas). Des index sont créés sur les clés étrangères et les champs fréquemment utilisés dans les recherches pour améliorer les performances.

4. Description des Modules Fonctionnels

4.1. Module Produits (`ProductsController`, `Product`)

- **Fonctionnalités :** CRUD complet pour les produits. Gestion du nom, description, catégorie parente, unité de mesure, prix d'achat et de vente. Le stock initial est enregistré lors de la création.
- **Modèles :** `Product.php`
- **Contrôleurs :** `ProductsController.php`
- **Vues :** `app/views/products/` (index, show, create, edit)
- **Logique métier :** Le stock (`quantity_in_stock`) est un cache mis à jour via la méthode `Product->updateStock()`, qui enregistre également un `StockMovement`. La modification directe du stock via le formulaire d'édition du produit est empêchée ; elle doit passer par des ajustements de stock dédiés.

4.2. Module Fournisseurs (`SuppliersController`, `Supplier`)

- **Fonctionnalités :** CRUD complet pour les fournisseurs (nom, contact, email, téléphone, adresse).
- **Modèles :** `Supplier.php`
- **Contrôleurs :** `SuppliersController.php`
- **Vues :** `app/views/suppliers/`
- **Logique métier :** Gestion standard des informations des fournisseurs. L'email doit être unique.

4.3. Module Clients (`ClientsController`, `Client`)

- **Fonctionnalités :** CRUD complet pour les clients (nom, type de client, email, téléphone, adresse).

- **Modèles :** `Client.php`
- **Contrôleurs :** `ClientsController.php`
- **Vues :** `app/views/clients/`
- **Logique métier :** Distinction entre clients "connus" (enregistrés) et "occasionnels" (nom saisi directement lors d'une vente). L'email doit être unique si fourni.

4.4. Module Approvisionnement

Ce module est divisé en Commandes Fournisseurs, Réceptions, et Factures Fournisseurs.

- **Commandes Fournisseurs (`PurchaseorderController`, `PurchaseOrder`, `PurchaseOrderItem`)**
  - **Fonctionnalités :** Création, affichage, modification (limitée), et annulation des commandes fournisseurs. Ajout dynamique de lignes de produits aux commandes.
  - **Vues :** `app/views/procurement/purchase_orders/`
  - **Logique métier :** Calcul automatique du montant total de la commande. Statuts de commande (`pending`, `partially_received`, `received`, `cancelled`).
- **Réceptions de Livraison (`DeliveryController`, `Delivery`, `DeliveryItem`)**
  - **Fonctionnalités :** Enregistrement des livraisons, qu'elles soient liées à une commande fournisseur ou directes/gratuites. Mise à jour du stock des produits lors de la réception.
  - **Vues :** `app/views/procurement/deliveries/`
  - **Logique métier :** La validation d'une réception met à jour `products.quantity_in_stock` et crée des mouvements de stock (`in_delivery`). Le statut de la commande fournisseur associée est mis à jour (`partially_received`, `received`). Gestion des réceptions partielles.
- **Factures Fournisseurs (`SupplierinvoiceController`, `SupplierInvoice`)**
  - **Fonctionnalités :** Enregistrement et suivi des factures fournisseurs. Liaison possible à des livraisons ou des commandes. Gestion des statuts de paiement (`unpaid`, `paid`, `partially_paid`).
  - **Vues :** `app/views/procurement/supplier_invoices/`
  - **Logique métier :** Principalement le suivi des échéances et des paiements. Une alerte peut être suggérée si on tente de facturer une livraison "gratuite".

4.5. Module Ventes (`SaleController`, `Sale`, `SaleItem`)

- **Fonctionnalités :** Enregistrement des ventes avec paiement immédiat ou différé. Sélection de clients enregistrés ou saisie de clients occasionnels. Ajout dynamique de lignes de produits. Décrémentation du stock lors de la vente. Enregistrement des paiements pour les ventes différées.
- **Vues :** `app/views/sales/` (index, show, create_immediate, create_deferred, record_payment)
- **Logique métier :** Vérification de la disponibilité du stock avant de valider une vente. Mise à jour de `products.quantity_in_stock` et création de mouvements de stock (`out_sale`). Gestion des statuts de paiement.

4.6. Module Gestion des Stocks (`StockController`, `StockMovement`, `Product`)

- **Fonctionnalités :** Historique des mouvements de stock par produit (avec filtres de date). Vue d'ensemble des stocks actuels. Création d'ajustements manuels de stock (entrées ou sorties justifiées).
- **Modèles :** `StockMovement.php` est central. `Product.php` contient la méthode `updateStock()` qui met à jour le cache `quantity_in_stock` ET crée un `StockMovement`.
- **Contrôleurs :** `StockController.php`
- **Vues :** `app/views/stock/` (index, history, create_adjustment)
- **Logique métier :** Tous les changements de stock (livraisons, ventes, ajustements) génèrent des enregistrements immuables dans `stock_movements`. `products.quantity_in_stock` sert de cache pour des lectures rapides. La page d'historique permet de comparer le stock cache et le stock calculé à partir des mouvements pour vérifier la cohérence.

4.7. Module Rapports (`ReportController`)

- **Fonctionnalités :** Fournit une interface pour accéder à divers rapports : état du stock actuel (avec filtre de stock bas), entrées de stock (par période/produit), sorties de stock (par période/produit), rapport des ventes (par période/client/statut paiement), rapport des achats (par période/fournisseur/statut PO).
- **Modèles :** Utilise les modèles existants et leurs nouvelles méthodes de filtrage (ex: `StockMovement->getMovementsByTypeAndDateRange()`, `Sale->getSalesByDateRangeAndFilters()`, etc.).
- **Vues :** `app/views/reports/` (index, et une vue par rapport).
- **Logique métier :** Agrégation et filtrage de données pour fournir des informations utiles à la décision.

5. Interface Utilisateur (UI/UX)

L'interface utilisateur a été développée avec HTML, CSS natif, et un peu de JavaScript natif pour les fonctionnalités dynamiques (comme l'ajout de lignes dans les formulaires).

- **CSS (`public/css/style.css`) :** Un fichier CSS central définit l'apparence globale. Il inclut :
    - Une réinitialisation de base et des styles pour le corps de la page.
    - Une palette de couleurs (principalement des nuances de bleu pour les actions primaires, des gris pour le texte et l'arrière-plan, et des couleurs sémantiques pour les alertes et statuts).
    - Des polices de caractères lisibles (`Segoe UI` ou alternatives sans-serif).
    - Des styles unifiés pour les titres, paragraphes, listes, liens, boutons, champs de formulaire, et tableaux.
    - L'utilisation de Flexbox pour la mise en page principale (en-tête, contenu, pied de page) et pour certains alignements internes.
    - Des media queries pour une responsivité de base sur les écrans de bureau, tablettes et mobiles. Les tableaux larges deviennent déroulants horizontalement sur petits écrans.
- **Layout Principal (`app/views/layouts/main.php`) :** Utilise une structure HTML sémantique (`<header>`, `<nav>`, `<main>`, `<footer>`). La navigation principale est responsive et utilise des menus déroulants pour l'accès aux sous-modules.
- **Améliorations des Vues :**
    - **Formulaires :** Les groupes de champs sont organisés avec `<fieldset>` et `<legend>`. Les labels sont alignés. Les boutons sont stylisés de manière cohérente. Les messages d'erreur de validation sont affichés clairement.

    - **Tableaux :** Amélioration de la lisibilité (padding, bordures). Les tableaux sont responsives grâce à un conteneur permettant le défilement horizontal.
    - **JavaScript :** Les scripts pour l'ajout dynamique de lignes dans les formulaires de commande/vente/livraison ont été conservés et fonctionnent avec la nouvelle interface. Des confirmations (`window.confirm`) ont été ajoutées pour les actions de suppression critiques.
- **Accessibilité (Principes) :** Bien que non exhaustive, une attention a été portée au contraste des couleurs et à la possibilité de navigation au clavier pour les éléments interactifs.
- **Convivialité :** Effort pour maintenir une terminologie cohérente. Les messages de statut (succès, erreur, avertissement) sont stylisés pour être bien visibles.

6. Instructions d'Installation et de Configuration

1. **Prérequis :**
    - Serveur web (Apache, Nginx) avec support PHP (version 7.4+ recommandée).
    - PostgreSQL (version 10+ recommandée).
    - Accès `php-pdo` et `php-pgsql` activé dans la configuration PHP.
2. **Installation :**
    - Cloner ou télécharger les fichiers du projet dans le répertoire racine de votre serveur web (ex: `htdocs`, `www`).
    - S'assurer que le dossier `public` est le DocumentRoot de votre configuration de serveur web, ou que les requêtes sont redirigées vers `public/index.php`.
3. **Configuration de la Base de Données :**
    - Créez une base de données PostgreSQL (ex: `erpnextclone_db`).
    - Ouvrez le fichier `config/database.php`.
    - Modifiez les valeurs suivantes pour correspondre à votre configuration PostgreSQL :

```
return [
    'driver' => 'pgsql',
    'host' => 'localhost',       // Ou l'IP de votre serveur DB
    'port' => '5432',            // Port PostgreSQL
    'database' => 'mydatabase',  // Nom de votre base de données
    'username' => 'user',        // Utilisateur PostgreSQL
    'password' => 'password',    // Mot de passe
    'charset' => 'utf8',
    'prefix' => '',
    'schema' => 'public',
];
```

- Exécutez le script SQL `docs/database_schema.sql` dans votre base de données pour créer toutes les tables, triggers et index nécessaires. Vous pouvez utiliser un outil comme pgAdmin ou la commande `psql`.

4. **Permissions (si nécessaire) :**
   - Assurez-vous que le serveur web a les permissions d'écriture pour les dossiers qui pourraient en avoir besoin (aucun pour l'instant dans ce projet, mais pertinent pour des fonctionnalités futures comme l'upload de fichiers).
5. **Accès à l'application :**
   - Ouvrez votre navigateur web et accédez à l'URL configurée pour le projet.

7. Conclusion

Le projet ERPNextClone a réussi à mettre en place une base solide pour un système ERP simplifié, couvrant les modules essentiels de gestion de produits, tiers, approvisionnement, ventes, stock et rapports. L'architecture MVC et les composants `core` fournissent une structure organisée pour le développement et la maintenance.

Fonctionnalités Réalisées :

- CRUD complet pour les entités de base.
- Flux de travail pour les achats (PO -> Livraison -> Facture Fournisseur) et les ventes (Vente -> Paiement).
- Gestion des stocks avec traçabilité des mouvements et mise à jour en temps réel du cache de quantité.
- Un ensemble de rapports de base pour l'analyse des données.
- Une interface utilisateur responsive et améliorée.

Pistes d'Améliorations Futures Possibles :

- **Authentification et Gestion des Utilisateurs :** Implémenter un système de connexion avec rôles et permissions.
- **Tableau de Bord :** Créer une page d'accueil (`#`) avec des indicateurs clés et des raccourcis.
- **Gestion des Catégories de Produits :** Permettre de classer les produits par catégories.
- **Fractionnement et Assemblage de Produits :**
  - **Fractionnement :** Compléter la fonctionnalité de fractionnement des produits. Bien que la structure `parent_id` existe et que les types de mouvements de stock `split_in`/`split_out` soient prévus, il manque une interface utilisateur dédiée (par exemple, une action "Fractionner ce produit" sur la page d'un produit) et la logique applicative dans le `ProductsController` ou un service dédié pour :
    - Demander à l'utilisateur le produit parent à fractionner et les produits enfants résultants avec leurs quantités.
    - Valider que les produits enfants existent ou les créer à la volée si nécessaire.
    - Générer un mouvement de stock `split_out` pour diminuer le stock du produit parent.
    - Générer des mouvements de stock `split_in` pour augmenter le stock des produits enfants.
    - Assurer que ces opérations sont atomiques (transaction de base de données).

    - **Assemblage (Kits/Nomenclatures) :** Gérer les produits composés (kits) où la vente d'un produit principal entraîne la sortie de stock de ses composants, ou l'assemblage d'un kit augmente son stock et diminue celui des composants.
- **Gestion des Devis/Propositions Commerciales :** Avant le cycle de vente.
- **Comptabilité :** Intégration plus poussée avec un plan comptable, génération d'écritures, suivi des balances (actuellement, seul le statut de paiement des factures/ventes est géré).
- **Export de Données :** Fonctionnalités d'export en CSV/Excel pour les rapports et les listes.
- **Tests Unitaires et d'Intégration :** Mettre en place des tests automatisés pour assurer la robustesse.
- **API :** Développer une API REST pour permettre des intégrations externes.
- **Améliorations UI/UX Avancées :** Utilisation de bibliothèques JavaScript pour des composants plus riches (modales, sélecteurs avancés, graphiques pour les rapports).
- **Internationalisation (i18n) et Localisation (l10n) :** Permettre la traduction de l'interface et la gestion de formats locaux (dates, devises).

Ce projet constitue une excellente fondation et peut être étendu de manière significative pour répondre à des besoins plus complexes.

Fin du Rapport.