

SÉRAPHIN PARYS

Application Intranet — SP-Vendeurs

DOCUMENTATION TECHNIQUE

Architecture, modèle de données et exploitation

Version 1.1 — Application v11

Public : équipes techniques et mainteneurs

Mai 2026

Sommaire

1. Présentation générale

1.1 Objectifs

L'application SP-Vendeurs centralise la gestion commerciale interne de Séraphin Parys : authentification des vendeurs, gestion du portefeuille clients, consultation du catalogue produits, saisie et suivi des commandes, statistiques en temps réel et synchronisation avec l'application Intranet fournisseurs développée sous WinDev.

1.2 Périmètre fonctionnel

- Authentification basée sur le rôle « vendeur / administrateur ».
- Tableau de bord avec KPI et graphiques calculés en SQL.
- Gestion CRUD complète des clients et lecture du catalogue produits.
- Saisie de commandes via un panier stocké en session, validation transactionnelle.
- Historique des commandes triées chronologiquement, avec calcul automatique du délai de livraison.
- API HTTP exposant l'authentification, la création de client, la création de produit et le réapprovisionnement vers / depuis l'application mobile et l'application fournisseur.

1.3 Acteurs

Acteur	Rôle
Vendeur	Utilisateur principal de l'Intranet, dispose de tous les droits CRUD sauf la création de vendeurs.
Administrateur technique	Mainteneur de l'application, opère sur le serveur, la base et les déploiements.
Application fournisseur (WinDev)	Système tiers qui pousse les mises à jour catalogue et stock via l'API.
Application mobile client	Application Dart / Flutter qui consomme l'API d'authentification et de commande.

2. Architecture applicative

2.1 Vue d'ensemble

L'application suit une architecture MVC artisanale (sans framework) en PHP 8. Chaque domaine fonctionnel est isolé dans un module autoporté contenant son contrôleur, son modèle et ses vues Smarty. Le point d'entrée unique est `index.php`, qui agit comme un routeur.

2.2 Routage par module

Le routeur de premier niveau lit le paramètre HTTP « gestion » pour instancier dynamiquement le contrôleur correspondant et invoquer la méthode passée dans « action ».

```
// index.php (extrait)
session_start();
require_once 'include/configuration.php';
Autoloader::chargerClasse();

if (!isset($_SESSION['login'])) {
    $_REQUEST['gestion'] = 'authentification';
} elseif (!isset($_REQUEST['gestion'])) {
    $_REQUEST['gestion'] = 'accueil';
}

$oRouteur = new $_REQUEST['gestion']($_REQUEST);
$oRouteur->choixAction();
```

Le pattern d'URL est donc `index.php?gestion=<module>&action=<methode>&<parametres>`. Lorsque la session n'est pas authentifiée, le routeur force le module d'authentification.

2.3 Autoloader

La classe Autoloader (`include/autoloader.php`) enregistre un chargement automatique des classes (`spl_autoload_register`). Elle parcourt les répertoires de tous les modules (contrôleur, modèle, vue) et charge le premier fichier `.php` qui correspond au nom de la classe demandée (en l'first).

2.4 Modules

Module	Rôle
mod_authentification	Gère la connexion, la déconnexion et le contrôle des informations de session.
mod_accueil	Tableau de bord : KPI et graphiques (Chart.js).
mod_client	CRUD complet du portefeuille clients et statistiques par client.
mod_produit	Lecture du catalogue produits et fiches statistiques par article.
mod_commande	Panier, validation des commandes et insertion des lignes.
mod_historique	Historique trié des commandes avec délai de livraison calculé.
mod_profil	Fiche personnelle du vendeur connecté, statistiques et changement de mot de passe.

Chaque module respecte la même décomposition : <module>Controleur.php, <module>Table.php (DTO avec hydratation et contrôles), <module>Modele.php (héritage de la classe abstraite Modele) et <module>Vue.php + templates .tpl Smarty associés.

2.5 Couche de persistance

La classe abstraite Modele (include/modele.php) encapsule la connexion PDO. Elle expose une méthode executeRequete(\$sql, \$parametres = null) qui choisit automatiquement entre une requête simple (query) et une requête préparée (prepare / execute).

```
abstract class Modele {
    private $cnx;
    protected function executeRequete($sql, $parametre = NULL) {
        if ($parametre == NULL) {
            $resultat = $this->getBD()->query($sql);
        } else {
            $resultat = $this->getBD()->prepare($sql);
            $resultat->execute($parametre);
        }
        return $resultat;
    }
    protected function getBD() {
        if ($this->cnx == NULL) {
            $this->cnx = new PDO('mysql:host='.SERVEUR.';dbname='.BASE, NOM, PASSE,
                [PDO::MYSQL_ATTR_INIT_COMMAND => 'SET NAMES utf8',
                 PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION]);
        }
        return $this->cnx;
    }
}
```

Bonne pratique

Toutes les requêtes utilisant des paramètres provenant de \$_REQUEST passent par des requêtes préparées, ce qui protège l'application contre l'injection SQL standard. La revue de code montre qu'aucune concaténation directe de paramètres utilisateur n'est utilisée dans les requêtes critiques.

3. Stack technique et environnement

3.1 Technologies

Couche	Technologie	Version cible
Langage serveur	PHP	8.1 ou supérieur
Framework serveur	Aucun (architecture MVC artisanale)	—
Templating	Smarty	4.x
Base de données	MySQL via PhpMyAdmin	5.7 / 8.0
Accès BDD	PDO MySQL avec ERRMODE_EXCEPTION	—
Front-end	Bootstrap, jQuery, Chart.js, vector-map	Bootstrap 4
Hébergement	Plesk avec déploiement Git/SSH	—
Versionnage	Git / GitHub	—

3.2 Arborescence des sources

```

sp-vendeurs-11/
├── index.php           # Point d'entrée et routeur
├── include/           # Configuration, autoloader, modèle abstrait
│   ├── configuration.php # Charge parametre, modele, Smarty, autoloader
│   ├── parametre.php    # SERVEUR, NOM, PASSE, BASE
│   ├── autoloader.php   # spl_autoload_register
│   ├── modele.php       # Classe abstraite Modele (PDO)
│   └── libs/            # Bibliothèque Smarty embarquée
├── mod_accueil/       # Tableau de bord
├── mod_authentification/ # Authentification (connexion / déconnexion)
├── mod_client/        # Portefeuille clients
├── mod_commande/      # Panier et nouvelles commandes
├── mod_historique/    # Historique des commandes
├── mod_produit/       # Catalogue produits
├── mod_profil/        # Profil vendeur
├── public/            # Templates partagés, CSS, JS, images
└── templates_c/       # Cache Smarty (templates compilés)

api/                  # Endpoints HTTP exposés à WinDev / mobile
├── wAuth.php          # Authentification mobile
├── wClientHttp.php     # Lecture clients
├── wClientCreerHttp.php # Création client
├── wProduitCreerHttp/  # Création de produit (WinDev)
└── reaprovisionnerProduits.php # Push stock fournisseur

```

3.3 Paramétrage

Les constantes de connexion sont définies dans `include/parametre.php`. En environnement local (Laragon), la base se connecte à `localhost` en utilisateur `root` sans mot de passe, sur la base `seraphinparys`.

```

define('SERVEUR', 'localhost');
define('NOM', 'root');
define('PASSE', '');
define('BASE', 'seraphinparys');

```

Recommandation

Pour les environnements de pré-production et de production, déplacer ces paramètres dans un fichier hors dépôt (par exemple `include/parametre.local.php`) chargé par `configuration.php`, ou utiliser des variables d'environnement Plesk. Ne jamais committer un mot de passe en clair dans le dépôt Git.

4. Authentification et sécurité

4.1 Cinématique d'authentification

- L'utilisateur saisit son login et son mot de passe dans le formulaire généré par AuthentificationVue.
- Le contrôleur instancie AuthentificationTable, qui hydrate ses propriétés et applique les contrôles de saisie.
- Si la saisie est valide, AuthentificationModele::rechercheVendeur lance un SELECT sur la table vendeur avec le login.
- Le mot de passe est comparé après hachage RIPEMD-128 salé (cf. section 4.2). En cas de succès, login, prenomNom et codevendeur sont stockés en session.
- L'utilisateur est redirigé vers index.php (Accueil). Sinon, le message « authentification invalide » s'affiche.

4.2 Stockage des mots de passe

Le mot de passe en base est stocké sous forme hachée avec un sel statique en préfixe et en suffixe, puis passé dans la fonction de hachage RIPEMD-128.

```
$gauche = "ar30&y%";
$droite = "tk!@";
$this->motdepasse = hash('ripemd128', "$gauche$motdepasse$droite");
```

Risque identifié

RIPEMD-128 est un algorithme rapide non conçu pour le stockage de mots de passe. Le sel étant statique, il ne protège pas contre les tables précalculées dédiées à l'application. Pour une montée en sécurité, migrer vers password_hash() avec PASSWORD_BCRYPT ou PASSWORD_ARGON2ID, avec un sel par mot de passe.

4.3 Gestion de la session

Une session PHP est démarrée à chaque requête (session_start). Les variables \$_SESSION['login'], \$_SESSION['prenomNom'] et \$_SESSION['codevendeur'] permettent d'identifier l'utilisateur, d'afficher son nom dans le header et de pré-remplir le code vendeur lors de la saisie de commandes.

4.4 Déconnexion

La méthode deconnecter du module authentification détruit la session côté serveur (session_destroy) et vide manuellement \$_SESSION avant de rediriger vers index.php.

4.5 Sécurité applicative

- Toutes les requêtes en base utilisent PDO::prepare puis execute avec des paramètres positionnels (?). Aucune injection SQL via les flux contrôlés.
- Les contrôles de saisie sont centralisés dans les classes Table (ex. ClientTable, AuthentificationTable) via les setters qui contrôlent la validité avant d'autoriser l'écriture en base.
- Le mode d'erreur PDO est positionné en ERRMODE_EXCEPTION : tout problème SQL remonte sous forme d'exception, ce qui facilite la détection des anomalies.

- L'application ne propose pas de CSRF token : à ajouter en priorité si elle est exposée hors du réseau interne.

5. Modèle de données

5.1 Schéma logique

La base seraphinparys est organisée autour de cinq entités principales : vendeur, client, produit, commande et ligne_commande. Le schéma est de type opérationnel relationnel, sans dénormalisation.

5.2 Tables principales

Table vendeur

Colonne	Type	Description
codev	INT (PK)	Identifiant du vendeur.
prenom	VARCHAR(35)	Prénom du vendeur.
nom	VARCHAR(35)	Nom du vendeur.
adresse	VARCHAR(50)	Adresse postale.
cp	VARCHAR(5)	Code postal.
ville	VARCHAR(25)	Ville.
telephone	VARCHAR(25)	Téléphone principal.
login	VARCHAR(255)	Login utilisé pour l'authentification.
motdepasse	VARCHAR(255)	Hash RIPEMD-128 salé.

Table client

Colonne	Type	Description
codec	INT (PK)	Identifiant unique du client.
nom	VARCHAR(35)	Nom et prénom (souvent combinés).
adresse	VARCHAR(50)	Adresse postale.
cp	VARCHAR(5)	Code postal.
ville	VARCHAR(25)	Ville.
telephone	VARCHAR(25)	Téléphone.
genre	VARCHAR(1)	M ou F (avatar par défaut).

Table produit

Colonne	Type	Description
reference	INT (PK)	Référence imposée par SP.
designation	VARCHAR(30)	Nom commercial.
quantite	INT	Conditionnement (poids ou nb de pièces).

Colonne	Type	Description
descriptif	VARCHAR(1)	G (gramme) ou P (pièce).
prix_unitaire_HT	DOUBLE	Prix unitaire hors taxes.
stock	SMALLINT	Stock courant.
poids_piece	INT	Poids unitaire pour les articles à la pièce.

Table commande

Colonne	Type	Description
numero	INT (PK)	Identifiant unique de la commande.
codec	INT (FK)	Référence vers client.
codev	INT (FK)	Référence vers vendeur.
date_commande	DATETIME	Date de saisie.
date_livraison	DATETIME	Date de livraison prévue.
total_ht	DOUBLE	Total hors taxes.
total_tva	DOUBLE	Total TVA.
etat	TINYINT	0 = stock non actualisé, 1 = stock MAJ.
source	TINYINT	0 = Intranet vendeur, 1 = Application mobile.

Table ligne_commande

Colonne	Type	Description
numero	INT (FK)	Référence vers commande.numero.
numero_ligne	SMALLINT	Numéro de ligne dans la commande.
reference	INT (FK)	Référence vers produit.reference.
quantite_demandee	SMALLINT	Quantité commandée.

5.3 Relations principales

- vendeur (1) — (N) commande : un vendeur peut saisir plusieurs commandes.
- client (1) — (N) commande : un client peut avoir plusieurs commandes.
- commande (1) — (N) ligne_commande : composition de la commande.
- produit (1) — (N) ligne_commande : un produit peut apparaître dans plusieurs commandes.

5.4 Requêtes notables

Calcul du chiffre d'affaires mensuel

```
SELECT DATE_FORMAT(date_commande, '%Y-%m') AS mois,
       SUM(total_ht) AS total_ht
FROM commande
```

```
GROUP BY DATE_FORMAT(date_commande, '%Y-%m')  
ORDER BY mois;
```

Top 5 clients par chiffre d'affaires HT

```
SELECT c.nom AS client,  
       SUM(cmd.total_ht) AS ca  
FROM commande cmd  
JOIN client c ON cmd.codec = c.codec  
GROUP BY c.codec  
ORDER BY ca DESC  
LIMIT 5;
```

Classement d'un produit par ventes

```
SELECT classement FROM (  
  SELECT p.reference,  
         SUM(lc.quantite_demandee) AS total_vendu,  
         RANK() OVER (ORDER BY SUM(lc.quantite_demandee) DESC) AS classement  
  FROM ligne_commande lc  
  JOIN produit p ON lc.reference = p.reference  
  GROUP BY p.reference, p.designation  
) c WHERE reference = ?;
```

6. API et intégrations externes

6.1 Détection de la source d'appel

Le contrôleur Commande détecte automatiquement si la requête provient de l'Intranet vendeur ou de l'application mobile via la méthode privée `detectSourceBool()` : header HTTP X-App-Source, puis heuristique sur le User-Agent (okhttp, cfnetwork, android, iphone, dart, flutter), et enfin URL hôte. Cette source est persistée dans le champ `commande.source`.

6.2 Endpoints exposés

Endpoint	Méthode	Rôle
<code>api/wAuth.php</code>	POST	Authentifie un utilisateur mobile (login + mot de passe).
<code>api/wClientHttp.php</code>	GET	Renvoie la liste des clients pour intégration externe.
<code>api/wClientCreerHttp.php</code>	POST	Crée un client depuis l'application mobile.
<code>api/wProduitCreerHttp/wProduitCreerHttp.php</code>	POST	Crée un produit depuis l'application fournisseur WinDev.
<code>api/reaprovisionnerProduits.php</code>	GET	Liste les produits dont le stock est inférieur à 50 unités.

6.3 Convention d'appel

Pour des raisons historiques, plusieurs endpoints utilisent un paramètre de garde `wDemande` (ex. `wDemande=aNot24get` pour `wAuth.php`, `wDemande=Not24get` pour `reaprovisionnerProduits.php`). Cette valeur agit comme une clé partagée triviale.

Recommandation

Remplacer ce garde-clé statique par un mécanisme d'authentification serveur-à-serveur (clé d'API en header, JWT signé ou OAuth2 `client_credentials` selon le contexte). Activer également HTTPS en production.

6.4 Format de réponse

Les endpoints renvoient soit du texte brut (codes de retour internes comme « 1 », « champs_manquants », « login_invalide ») soit, pour les listes, du JSON sérialisé depuis le résultat PDO. Les codes HTTP utilisés sont 200, 400 (entrée invalide) et 401 (authentification refusée).

7. Cycle de saisie d'une commande

7.1 Diagramme synthétique

- L'utilisateur ouvre « Passer une commande » → `CommandeControleur::form_ajouter`.
- Pour chaque article ajouté, le panier est sérialisé en JSON dans `$_SESSION['panier_json']`.
- Les opérations de panier (modifier / supprimer une ligne) sont gérées par `form_modifier` et `form_supprimer`, qui réécrivent le JSON en session.
- À la validation, `CommandeControleur::form_valider_commande` charge les détails de chaque produit (`getProduitByReference`) et calcule le total HT.
- `CommandeModele::insererCommande` insère la commande et retourne le numéro (`lastInsertId`).
- `CommandeModele::insererLigneCommande` ajoute chaque article (référence + quantité).
- `CommandeModele::mettreAJourStock` décrémente le stock du produit.
- L'utilisateur est renvoyé sur l'historique, la nouvelle commande figure en tête.

7.2 Points d'attention

- La séquence (insertion commande + lignes + mise à jour stock) n'est pas encapsulée dans une transaction SQL. En cas d'échec partiel, la base peut rester incohérente.
- Le calcul de la TVA n'est pas explicité dans l'insertion (la colonne `total_tva` est alimentée mais sans formule centrale).
- Le champ `etat` (0/1) permettrait de mettre en place un flux de validation différée du stock (à activer dans une évolution future).

Évolution recommandée

Encadrer la séquence par `$bd->beginTransaction()` / `$bd->commit()` (et `$bd->rollBack()` en cas d'erreur) afin de garantir l'atomicité des opérations.

8. Déploiement et exploitation

8.1 Environnement local

- Stack Laragon (Apache + PHP 8 + MySQL) sous Windows.
- L'application se trouve dans C:\laragon\www\sp-vendeurs avec un dossier par version (sp-vendeurs-01 à sp-vendeurs-11).
- Accès local : <http://localhost/sp-vendeurs/sp-vendeurs-11/index.php>.

8.2 Environnement de production

- Hébergement Plesk avec accès SSH.
- Déploiement par récupération depuis GitHub (git pull) ou via le hook Plesk Git.
- Les sessions PHP utilisent le stockage par défaut (fichiers).

8.3 Procédure de déploiement type

```
# Sur le serveur Plesk
ssh user@serveur-plesk
cd /var/www/vhosts/<domaine>/httpdocs
git fetch --all
git checkout main
git pull --ff-only
# Vider le cache Smarty
rm -rf templates_c/*
# Vérifier les droits
chown -R www-data:psacln .
```

8.4 Sauvegarde et reprise

- Le code source est sauvegardé via Git (origine GitHub privé).
- La base de données est exportée régulièrement via PhpMyAdmin (mysqldump > seraphinparys_YYYYMMDD.sql).
- En cas de sinistre, restaurer la dernière sauvegarde SQL puis redéployer la branche stable.

8.5 Métriques actuelles

Métrique	Valeur
Nombre de vendeurs	2
Nombre de clients	Environ 30
Volume de commandes	Environ 50 transactions
Volume catalogue	Environ une trentaine de références
Charge serveur	Faible — adapté à un serveur mutualisé

9. Limitations connues et pistes d'évolution

9.1 Limitations

- Aucune transaction SQL n'encadre la création de commande, ce qui peut conduire à un état incohérent en cas de panne.
- Stockage du mot de passe en RIPEMD-128 salé statique (cf. section 4.2).
- Pas de protection CSRF sur les formulaires sensibles.
- Identifiants de connexion à la base codés en dur dans parametre.php.
- Pas de système de tickets ni de suivi formel des incidents.
- Templates Smarty compilés en place dans templates_c (à exclure du dépôt Git).

9.2 Pistes d'évolution

- Migration des mots de passe vers password_hash / password_verify (BCRYPT ou ARGON2ID).
- Ajout d'un token CSRF généré par session pour les formulaires POST.
- Externalisation de la configuration sensible (fichier .env ou variables Plesk).
- Mise en place d'une transaction PDO autour de la validation de commande.
- Introduction d'un mécanisme de logging applicatif (Monolog) pour tracer connexions et erreurs.
- Refonte progressive du front avec un framework léger (Bootstrap 5, ou réécriture en composants Vue.js / React selon l'évolution mobile).
- Mise en place de tests automatisés (PHPUnit) sur les contrôleurs et modèles critiques.
- Documentation OpenAPI / Swagger pour les endpoints exposés à l'application mobile et au fournisseur.

10. Annexes

10.1 Glossaire technique

Terme	Définition
MVC	Modèle-Vue-Contrôleur : pattern qui sépare la donnée (modèle), la présentation (vue) et l'orchestration (contrôleur).
DTO / Table	Objet de transfert de données qui hydrate ses propriétés à partir d'un tableau associatif et porte les contrôles de saisie (classes <code><module>Table.php</code>).
Smarty	Moteur de template PHP utilisé pour la couche vue.
PDO	PHP Data Objects, couche d'abstraction d'accès aux bases relationnelles.
RIPEMD-128	Fonction de hachage cryptographique 128 bits utilisée pour stocker les mots de passe.
Plesk	Panneau d'administration serveur utilisé pour héberger l'application.

10.2 Conventions de nommage

- Classes en PascalCase ; fichiers en camelCase (la première lettre est passée en lcfirsr par l'autoloader).
- Méthodes en camelCase. Les contrôleurs préfixent par `form_` les méthodes qui rendent un formulaire.
- Tables SQL en minuscules (client, commande, ligne_commande, produit, vendeur).

10.3 Contacts et références

L'équipe technique peut être contactée via le canal interne de Séraphin Parys. Le code source est hébergé sur GitHub (dépôt privé). Pour toute évolution ou anomalie, ouvrir une issue ou contacter directement le mainteneur en charge.