Architecture — dirCops

Application : dirCops — suivi facturation / échéancier / paiement
Mode : POC puis migration vers MODX en production

---

1. Contexte et périmètre

Exigences fonctionnelles : 13 FR en 6 catégories (Client Management, Invoice Tracking, Payment Management, Financial Calculations, Reporting & Analytics, Data Management). Application web MPA, calculs automatiques (encours, échéances), export CSV.

Exigences non fonctionnelles : Sécurité (accès restreint), Performance (< 3s chargement, < 1s interactions).

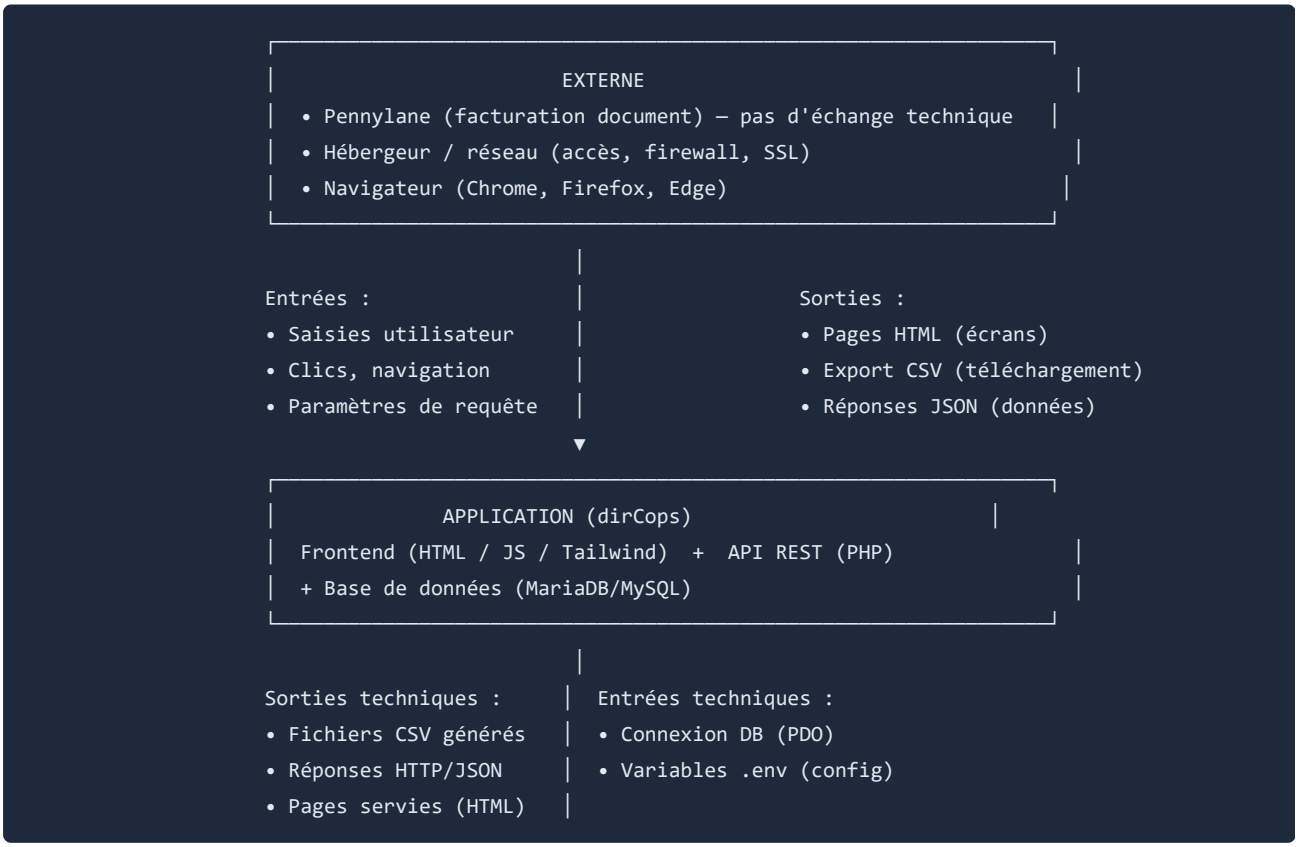
Contraintes : Facturation gérée dans Pennylane ; l'appli assure uniquement le suivi (saisie manuelle des factures). Budget restreint → architecture minimale. Stack : PHP (API REST) + JavaScript + Tailwind + HTML + MariaDB/MySQL. Application interne uniquement.

Périmètre utilisateurs : Un seul type traité : **commercial / admin** (accès complet). Les autres rôles (lecteur, stagiaire, etc.) sont en option.

---

2. Vue d'ensemble et architecture fonctionnelle

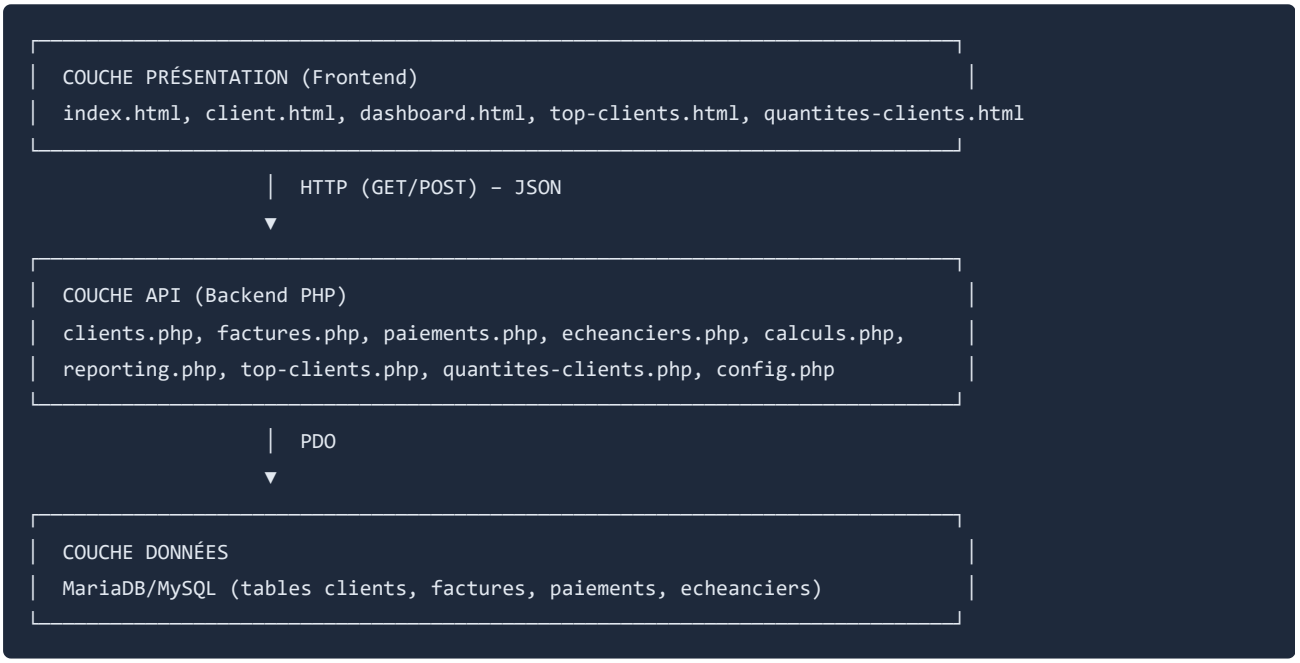
2.1 Schéma de contexte



2.2 Acteurs

Acteur	Rôle	Entrées fournies	Sorties reçues
Commercial (équipe EMGI)	Utilisateur principal	Recherche client, saisie facture/paiement/écheancier, filtres (type, cycle), demande d'export	Liste clients, fiche client, tableau de bord, Top clients, Quantités, fichier CSV
Trésorerie / Direction (si accès)	Consultation	Consultation écrans, export	Mêmes sorties en lecture
Système (automatique)	—	—	Calculs encours, état facture (payé/en attente), échéances

2.3 Vue par couches



2.4 Modules fonctionnels

Module	Responsabilité	Fichiers principaux
Config	Connexion DB, .env, helpers (sendJson, sendError)	`api/config.php`, `.env`
Clients	Liste, recherche, détail, création, suppression	`api/clients.php`, `index.html`, `client.html`, `js/client-detail.js`
Factures	Liste par client, ajout/modification/suppression (suivi)	`api/factures.php`, `client.html`, `js/facture-form.js`
Paiements	Liste, enregistrement, suppression	`api/paiements.php`, `client.html`, `js/paiements.js`
Écheanciers	Liste, création, suppression	`api/echeanciers.php`, `client.html`, `js/echeancier-form.js`
Calculs	Encours, échéances à venir	`api/calculs.php`, `api/reporting.php`, `dashboard.html`, `js/dashboard.js`
Reporting	Tableau de bord, export CSV (encours, factures PJ)	`api/reporting.php`, `dashboard.html`
Top clients	Agrégation type/cycle, quantités, CA	`api/top-clients.php`, `top-clients.html`, `js/top-clients.js`
Quantités par client	Synthèse quantités facturées/livrées	`api/quantites-clients.php`, `quantites-clients.html`, `js/quantites-clients.js`

Dépendances : Clients indépendant ; Factures → Clients ; Paiements → Factures ; Écheanciers → Factures ; Calculs → Clients, Factures, Paiements ; Reporting → Calculs ; Top clients, Quantités → Factures, Clients.

---

3. Décisions techniques

3.1 Stack POC

- **Backend :** PHP (API REST)
- **Frontend :** JavaScript vanilla + Tailwind CSS + HTML (MPA)
- **Base de données :** MariaDB/MySQL
- **Authentification :** Sessions PHP simples ou auth basique ; pas de JWT/OAuth

3.2 Base de données

Tables : `clients`, `factures`, `paiements`, `echeanciers`
Relations : factures → client, paiements → facture, echeanciers → facture
Champs factures : n_facture, n_vente, date_facture, type_produit (alevins/larves), cycle, quantites, qte_livree, montant_ht

Champs échéanciers : facture_id, nb_echeances, date_debut_echeancier
État payé/en attente : dérivé des paiements

3.3 API REST

Endpoints : `/api/clients`, `/api/factures`, `/api/paiements`, `/api/echeanciers`, `/api/calculs`, `/api/reporting`, `/api/top-clients`, `/api/quantites-clients`
Format : JSON ; erreurs `{error: "message"}` avec codes HTTP 200, 400, 404, 500

3.4 Frontend

Pages : `index.html`, `client.html`, `dashboard.html`, `top-clients.html`, `quantites-clients.html`
State : JavaScript vanilla, variables globales ou localStorage
Pas de SPA : rechargement de page après certaines actions

3.5 Infra

Dev : `php -S` ou Apache/Nginx
Déploiement : Hébergement PHP + MariaDB ; `sql/schema.sql` une fois ; `.env` en production

---

4. Patterns et conventions

4.1 Nommage

Contexte	Convention	Exemple
Tables et colonnes	snake_case, pluriel tables	`clients`, `client_id`, `n_facture`
API (endpoints, params, JSON)	snake_case	`?client_id=1`, `{client_id: 1}`
Code PHP/JS	camelCase	`getClientData()`, `$clientId`
Fichiers	minuscules, tirets	`client-detail.js`

4.2 Formats API

- **Succès :** réponse directe `{...}` (pas de wrapper `data`)
- **Erreur :** `{error: "message"}` + code HTTP approprié
- **Dates :** ISO (`2026-02-09T10:00:00Z`)

4.3 Règles agents IA

- snake_case en DB et JSON API ; camelCase en PHP/JS
- Respecter la structure des endpoints et du projet
- Retourner les erreurs au format `{error: "message"}`
- **Anti-patterns :** pas de wrapper `{data: ...}` ; pas de mélange camelCase/snake_case dans le même contexte

---

5. Interfaces logicielles

5.1 I1 — Utilisateur ↔ Application

Points d'entrée : `index.html` (liste clients), `client.html?id=` (fiche client), `dashboard.html`, `top-clients.html`, `quantites-clients.html`, lien « Exporter CSV » → `api/reporting.php?format=csv`.

5.2 I2 — Frontend ↔ API

Protocole : GET (lecture), POST (création), PUT (modification), DELETE (suppression). Body JSON pour POST/PUT.
Erreurs : 4xx/5xx avec `{error: "message"}` ; front : `apiGet`, `apiPost`, etc. dans utils.js.

Appels principaux :

Appelant	Méthode	Endpoint	Rôle
index.html	GET/POST/DELETE	`/api/clients.php`	Liste, recherche, création, suppression client
client.html	GET	`/api/clients`, `/api/factures`, `/api/paiements`, `/api/calculs`, `/api/echeanciers`	Détail client
client.html	POST/PUT/DELETE	`/api/factures.php`, `/api/paiements.php`, `/api/echeanciers.php`	Création, modification, suppression
dashboard.html	GET	`/api/reporting.php`	Tableau de bord ; `format=csv` ou `format=factures_csv` pour export
top-clients.html	GET	`/api/top-clients.php`	Données Top clients (params type, cycle)
quantites-clients.html	GET	`/api/quantites-clients.php`	Quantités par client (param type)

5.3 I3 — API ↔ Base de données

Protocole : PDO, pilote pdo_mysql. Connexion dans `config.php` (getPdo()).
Opérations : SELECT, INSERT, UPDATE, DELETE ; requêtes préparées. Charset utf8mb4.

5.4 I4 — Export CSV

Endpoint : `api/reporting.php` avec `format=csv` (encours par client) ou `format=factures_csv` (ensemble factures PJ). En-têtes Content-Type text/csv, Content-Disposition attachment.

5.5 I5 — Pennylane (hors MVP)

Non réalisé en MVP. Facturation dans Pennylane ; saisie manuelle dans l'appli. Évolution possible : import ou sync factures/paiements.

5.6 Récapitulatif

Id	Interface	Sens	Protocole
I1	Utilisateur ↔ Application	Bidirectionnel	HTTP, navigateur, HTML/CSV
I2	Frontend ↔ API	Front ↔ API	HTTP, JSON
I3	API ↔ Base de données	API ↔ DB	PDO, MySQL/MariaDB
I4	Application → Utilisateur (export)	API → navigateur	HTTP, CSV
I5	Application ↔ Pennylane	— (hors MVP)	À définir

---

6. Structure projet et validation

6.1 Arborescence

```
dircops/
├── README.md
├── .env.example
├── .env
├── index.html
├── client.html
├── dashboard.html
├── top-clients.html
├── quantites-clients.html
├── api/
│   ├── config.php
│   ├── clients.php
│   ├── factures.php
│   ├── paiements.php
│   ├── calculs.php
│   ├── reporting.php
│   ├── top-clients.php
│   └── quantites-clients.php
├── js/
│   ├── utils.js
│   ├── client-detail.js
│   ├── paiements.js
│   ├── dashboard.js
│   ├── facture-form.js
│   ├── top-clients.js
│   └── quantites-clients.js
├── css/
└── sql/
    └── schema.sql
```

6.2 Flux de données

- Liste clients : `index.html` → `api/clients.php` → DB

- Détail client : `client.html` → API (clients, factures, paiements, calculs, écheanciers) → DB
- Dashboard : `dashboard.html` → `api/calculs.php`, `api/reporting.php` → DB
- Paiement : `client.html` + `js/paiements.js` → `api/paiements.php` → DB → recalcul API

6.3 Validation

Statut : READY FOR IMPLEMENTATION

Points forts : Stack simple ; séparation nette API / frontend / DB ; patterns explicites.
À préciser en implémentation : Schéma SQL détaillé, format export CSV.
Évolutions futures : Auth détaillée, migration MODX.

6.4 Priorité d'implémentation

1. `sql/schema.sql`
2. `api/config.php`
3. Endpoints de base : `clients.php`, `factures.php`, `paiements.php`, `echeanciers.php`

Document de référence pour toute question d'architecture. À mettre à jour en cas d'ajout d'écrans, d'endpoints ou d'intégrations externes.