Architecture générale, fonctionnelle et interfaces logicielles

Application : Suivi FTI EMGI
Projet : Suivi facturation / échéancier / paiement (EMGI)
Date : 2026-02-09
Contenu : Architecture générale (acteurs, extérieur, entrées/sorties), décomposition fonctionnelle (modules), interfaces logicielles détaillées.

---

1. Architecture générale de l'application

Vue de haut niveau : acteurs, interactions avec l'extérieur, entrées et sorties du système.

1.1 Schéma de contexte



1.2 Acteurs et utilisateurs

Acteur	Rôle	Entrées qu'il fournit	Sorties qu'il reçoit
Commercial (équipe EMGI)	Utilisateur principal	Recherche client, saisie facture/paiement/écheancier, choix des filtres (type, cycle), demande d'export	Liste clients, fiche client (factures, paiements, écheanciers, encours), tableau de bord, Top clients, Quantités, fichier CSV
Trésorerie / Direction (si accès)	Consultation, reporting	Consultation des écrans, demande d'export	Mêmes sorties en lecture (selon rôles à définir)
Système (automatique)	—	—	Calculs encours, état facture (payé/en attente), échéances à venir

1.3 Interactions avec l'extérieur

Externe	Type d'interaction	Détail
Utilisateur (navigateur)	Entrée / Sortie	Utilisateur accède à l'URL de l'application ; saisit des données (formulaires) ; consulte des écrans ; déclenche le téléchargement du CSV.
Réseau / Hébergeur	Infra	Accès restreint (VPN, IP, ou auth à définir). Pas d'API exposée vers l'internet public.
Pennylane	Métier (hors appli)	La facturation (document, PDF) est gérée dans Pennylane. Aucune interface technique (API, fichier) entre l'application et Pennylane en MVP. Les factures sont saisies manuellement dans l'appli après émission dans Pennylane.
Fichiers sur disque	Entrée / Sortie	En mode mock : lecture/écriture des fichiers `mock/*.json`. En mode DB : pas d'échange de fichiers métier avec l'extérieur. Export CSV = génération à la volée puis téléchargement navigateur.

1.4 Synthèse des entrées et sorties

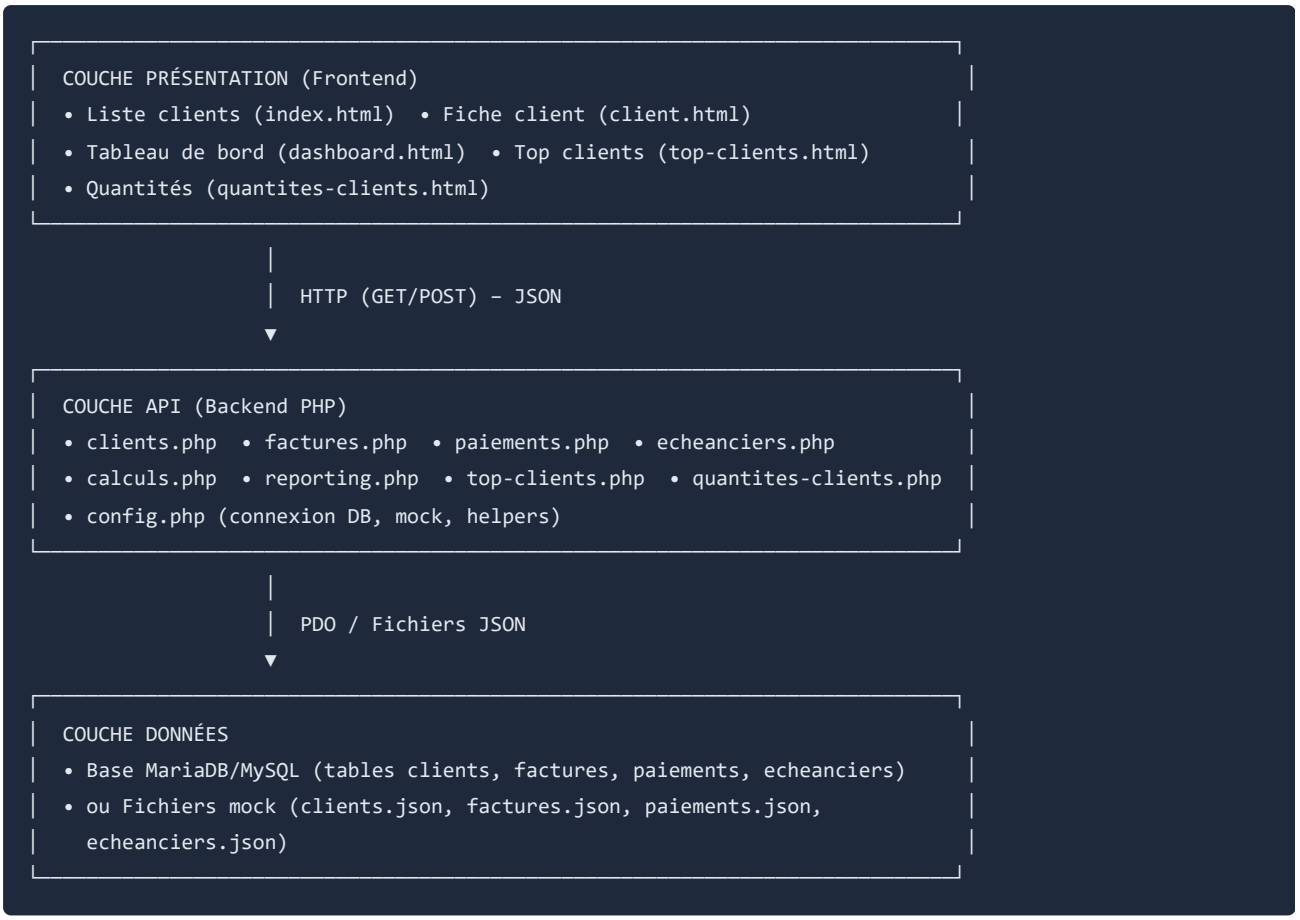
Catégorie	Entrées	Sorties
Utilisateur	Recherche (nom client), création/modification/suppression client, facture, paiement, écheancier (formulaires et boutons), filtres (type, cycle), clic « Exporter CSV »	Écrans HTML (liste clients, fiche client, dashboard, Top clients, Quantités), messages d'erreur, fichier CSV téléchargé
API (côté front)	Paramètres GET (client_id, id, type, cycle, format), body POST/PUT (JSON), DELETE (id en query)	Réponses JSON (tableaux ou objets), codes HTTP (200, 400, 404, 500), en-têtes (Content-Type, Content-Disposition pour CSV)
Base de données	Requêtes SQL (SELECT, INSERT, UPDATE, DELETE) via PDO	Jeu de résultats (lignes)
Fichiers mock	Lecture `mock/*.json`	Écriture `mock/*.json` (après POST)
Configuration	Fichier `.env` (DB_*, MOCK_MODE)	—

---

2. Architecture fonctionnelle – Décomposition fonctionnelle

Décomposition du système en **modules fonctionnels** et responsabilités.

2.1 Vue par couches



2.2 Modules fonctionnels (décomposition)

Module	Responsabilité	Fichiers / composants principaux
Config / Fondations	Connexion DB, mode mock, chargement .env, helpers (sendJson, sendError, loadMock, saveMock)	`api/config.php`, `.env`
Clients	Liste des clients, recherche par nom, détail d'un client, création et suppression client	`api/clients.php`, `index.html`, `client.html`, `js/client-detail.js`
Factures	Liste des factures par client, ajout et modification manuelle d'une facture (suivi), suppression, champs type/cycle/quantités	`api/factures.php`, `client.html`, `js/facture-form.js`, `js/client-detail.js`
Paiements	Liste des paiements par client, enregistrement et suppression d'un paiement (facture, montant, date)	`api/paiements.php`, `client.html`, `js/paiements.js`, `js/client-detail.js`
Écheanciers	Liste des écheanciers (par client ou facture), création et suppression d'un écheancier (facture, nb échéances, date début)	`api/echeanciers.php`, `client.html`, `js/echeancier-form.js`, `js/client-detail.js`
Calculs financiers	Encours par client, encours total, échéances à venir (dates et montants dérivés des écheanciers si présents, sinon date facture et encours ; filtre date ≥ aujourd'hui)	`api/calculs.php`, `api/reporting.php`, `client.html`, `dashboard.html`, `js/client-detail.js`, `js/dashboard.js`
Reporting	Tableau de bord (encours, échéances), export encours CSV, export ensemble des factures (PJ)	`api/reporting.php` (format=json
Top clients	Agrégation par type (alevins/larves) et cycle, quantités et CA, pourcentages	`api/top-clients.php`, `top-clients.html`, `js/top-clients.js`
Quantités par client	Synthèse quantités facturées / livrées par client, filtre par type	`api/quantites-clients.php`, `quantites-clients.html`, `js/quantites-clients.js`

2.3 Dépendances entre modules

- **Clients** : indépendant (sauf config pour DB/mock).
- **Factures** : dépend de **Clients** (client_id).
- **Paiements** : dépend de **Factures** (facture_id).
- **Écheanciers** : dépend de **Factures** (facture_id).
- **Calculs** : dépend de **Clients**, **Factures**, **Paiements** (lecture seule).
- **Reporting** : dépend de **Calculs** (ou recopie logique encours/échéances en mock).
- **Top clients** : dépend de **Factures** (et clients pour noms).
- **Quantités clients** : dépend de **Factures** (et clients).

---

3. Interfaces logicielles

Détail de chaque **interface** : acteurs, protocole, données, sens du flux, erreurs.

3.1 Interface Utilisateur ↔ Application (I1)

Nom : Interface Utilisateur – Application Web
Type : Homme – Machine (navigateur Web).

Caractéristique	Détail
Acteurs	Utilisateur (commercial) côté client ; serveur HTTP (PHP) côté application.
Support	Requêtes HTTP (GET pour affichage des pages et des données, POST via formulaires ou fetch).
Entrées (utilisateur → appli)	URL (navigation), champs de formulaire (recherche, n° facture, montant, date, etc.), clics (boutons, liens).
Sorties (appli → utilisateur)	Pages HTML (avec Tailwind CSS), données affichées (listes, tableaux), messages d'erreur (zone dédiée ou alert), fichier CSV (téléchargement).
Contraintes	Navigateurs supportés : Chrome, Firefox, Edge (dernières versions). Pas de SPA : rechargement de page après certaines actions (ex. après ajout facture/paiement/ écheancier).

Points d'entrée principaux :

URL / action	Rôle
`index.html`	Liste des clients, recherche par nom.
`client.html?id=<id>`	Fiche client (factures, paiements, écheanciers, formulaires).
`dashboard.html`	Tableau de bord (encours, échéances, lien export CSV).
`top-clients.html`	Top clients (filtres type, cycle).
`quantites-clients.html`	Quantités par client (filtre type).
Lien « Exporter CSV »	Télécharge `api/reporting.php?format=csv`.

3.2 Interface Frontend ↔ API (I2)

Nom : Interface Frontend – API REST
Type : Logicielle (JavaScript ↔ PHP).

Caractéristique	Détail
Acteurs	Scripts JS (utils.js, client-detail.js, facture-form.js, etc.) ; scripts PHP (api/*.php).
Protocole	HTTP/HTTPS. Méthodes : GET (lecture), POST (création), PUT (modification), DELETE (suppression).
Format des requêtes	GET : paramètres en query string (?client_id=1, ?id=1, ?type=alevins&cycle=25-26, ?format=csv). POST / PUT : body JSON (Content-Type: application/json). DELETE : paramètre `id` en query string (ex. ?id=1).
Format des réponses	JSON (Content-Type: application/json), sauf export CSV (Content-Type: text/csv; charset=utf-8, avec Content-Disposition: attachment).
Gestion d'erreurs	Réponses 4xx/5xx avec body `{ "error": "message" }`. Côté front : `apiGet`, `apiPost`, `apiPut`, `apiDelete` (utils.js) lisent le JSON et lèvent une exception en cas de !res.ok.

Détail des appels (par écran / fonctionnalité) :

Appelant	Méthode	Endpoint	Paramètres / Body	Rôle
index.html (liste clients)	GET	`/api/clients.php`	`q` (optionnel, recherche)	Liste ou recherche clients.
index.html (ajout client)	POST	`/api/clients.php`	Body : nom	Création client.
index.html (suppression client)	DELETE	`/api/clients.php`	`id`	Suppression client (cascade factures, paiements, écheanciers).
client.html (détail)	GET	`/api/clients.php`	`id`	Détail d'un client.
client.html (détail)	GET	`/api/factures.php`	`client_id`	Factures du client (avec encours, état).
client.html (détail)	GET	`/api/paiements.php`	`client_id`	Paiements du client.
client.html (détail)	GET	`/api/calculs.php`	`client_id`	Encours du client.
client.html (détail)	GET	`/api/echeanciers.php`	`client_id`	Écheanciers du client.
client.html (ajout facture)	POST	`/api/factures.php`	Body : client_id, n_facture, n_vente, date_facture, type_produit, cycle, quantites, qte_livree, montant_ht	Création facture.
client.html (modification facture)	PUT	`/api/factures.php`	Body : id, client_id, n_facture, n_vente, date_facture, type_produit, cycle, quantites, qte_livree, montant_ht	Modification facture.
client.html (suppression facture)	DELETE	`/api/factures.php`	`id`	Suppression facture (cascade paiements, écheancier).
client.html (ajout paiement)	POST	`/api/paiements.php`	Body : facture_id, montant, date_paiement	Création paiement.
client.html (suppression paiement)	DELETE	`/api/paiements.php`	`id`	Suppression paiement.

client.html (ajout écheancier)	POST	`/api/echeanciers.php`	Body : facture_id, nb_echeances, date_debut_echeancier	Création écheancier.
client.html (suppression écheancier)	DELETE	`/api/echeanciers.php`	`id`	Suppression écheancier.
dashboard.html	GET	`/api/reporting.php`	(aucun ou format=json)	Données tableau de bord.
dashboard.html (export encours)	GET	`/api/reporting.php`	`format=csv`	Téléchargement CSV encours par client.
dashboard.html (export factures PJ)	GET	`/api/reporting.php`	`format=factures_csv`	Téléchargement CSV ensemble des factures (suivi-factures-YYYY-MM-DD.csv).
top-clients.html	GET	`/api/top-clients.php`	`type` , `cycle` (optionnel)	Données Top clients.
quantites-clients.html	GET	`/api/quantites-clients.php`	`type` (optionnel)	Quantités par client.

Convention côté front : base des appels = `/api` (constante `API_BASE` dans utils.js). Pas d'API-to-API : le front est le seul client de l'API.

3.3 Interface API ↔ Base de données (I3)

Nom : Interface API – Base de données
Type : Logicielle (PHP PDO – MariaDB/MySQL).

Caractéristique	Détail
Acteurs	Scripts PHP (api/*.php) ; SGBD MariaDB ou MySQL.
Protocole / pilote	PDO (PHP Data Objects), pilote `pdo_mysql` . Connexion centralisée dans `config.php` (getPdo()).
Entrées (API → DB)	Requêtes SQL (SELECT, INSERT, UPDATE, DELETE). Paramètres bindés (requêtes préparées) pour éviter les injections.
Sorties (DB → API)	Jeu de résultats (tableaux associatifs, FETCH_ASSOC). lastInsertId() après INSERT.
Contraintes	Utilisée uniquement si `MOCK_MODE` est désactivé (.env). Charset utf8mb4.

Détail par script API :

Script API	Opérations DB	Tables concernées
clients.php	SELECT (liste ou par id) ; INSERT ; DELETE	clients
factures.php	SELECT par client_id ; INSERT ; PUT (UPDATE) ; DELETE	factures
paiements.php	SELECT par client_id ; INSERT ; DELETE	paiements
echeanciers.php	SELECT par client_id ou facture_id ; INSERT ; DELETE ; vérification unicité (SELECT)	echeanciers
calculs.php	SELECT clients, factures, LEFT JOIN / agrégation paiements	clients, factures, paiements
reporting.php	SELECT + agrégations (encours, échéances)	clients, factures, paiements
top-clients.php	SELECT + GROUP BY type_produit, cycle	factures, clients
quantites-clients.php	SELECT + agrégations quantites, qte_livree	factures, clients

Configuration : `config.php` lit `.env` (DB_HOST, DB_NAME, DB_USER, DB_PASS, DB_CHARSET) et construit le DSN. Pas de pool de connexions ; une connexion PDO partagée par requête (singleton dans getPdo()).

3.4 Interface API ↔ Fichiers mock (I4)

Nom : Interface API – Fichiers mock
Type : Logicielle (PHP – système de fichiers).

Caractéristique	Détail
Acteurs	Scripts PHP (api/*.php) ; répertoire `app/mock/`.
Protocole	Lecture/écriture de fichiers JSON. Fonctions centralisées dans `config.php` : loadMock($name), saveMock($name, $data), mockPath($name).
Entrées (API → fichiers)	saveMock : écriture de tableaux PHP sérialisés en JSON (JSON_UNESCAPED_UNICODE | JSON_PRETTY_PRINT).
Sorties (fichiers → API)	loadMock : lecture et décodage JSON en tableaux PHP. Fichiers : clients.json, factures.json, paiements.json, echeanciers.json.
Contraintes	Utilisée uniquement si `MOCK_MODE` est activé (pas de .env ou MOCK_MODE=1). En lecture, fichier absent ou invalide → tableau vide.

Détail par ressource :

Fichier mock	Utilisé par	Opérations
clients.json	clients.php	GET : lecture, filtre (recherche, id).
factures.json	factures.php, calculs.php (mock), reporting.php (mock)	GET : lecture, filtre client_id. POST : lecture, ajout ligne, saveMock.
paiements.json	paiements.php, factures.php (calcul encours), calculs.php, reporting.php	GET : lecture, filtre client_id. POST : lecture, ajout ligne, saveMock.
echeanciers.json	echeanciers.php	GET : lecture, filtre client_id ou facture_id. POST : lecture, vérification unicité facture_id, ajout, saveMock.

Pas d'interface entre mock et base de données : un seul mode actif à la fois (soit mock, soit DB).

3.5 Interface Application ↔ Extérieur (sorties utilisateur) (I5)

Nom : Sorties vers l'utilisateur (export, téléchargement)
Type : Logicielle (API → navigateur).

Caractéristique	Détail
Acteurs	api/reporting.php ; navigateur (utilisateur).
Protocole	HTTP GET. Paramètres : `format=csv` (encours par client) ou `format=factures_csv` (ensemble des factures, fichier PJ).
Sorties	Corps de réponse : contenu CSV. (1) Encours par client (`format=csv`). (2) Ensemble des factures pour PJ (`format=factures_csv` , fichier suivi-factures-YYYY-MM-DD.csv). En-têtes : Content-Type: text/csv; charset=utf-8, Content-Disposition: attachment; filename="...".
Contraintes	Séparateur et encodage à définir (ex. point-virgule, UTF-8 BOM pour Excel). Pas d'authentification spécifique sur cet endpoint en POC.

Détail :

- L'utilisateur clique sur « Exporter encours CSV » (`api/reporting.php?format=csv`) ou « Exporter l'ensemble des factures (PJ) » (`api/reporting.php?format=factures_csv`) ; le front peut appeler l'URL en fetch et déclencher le téléchargement via un blob.
- Aucune autre sortie fichier vers l'extérieur en MVP (pas d'envoi d'email, pas d'API exposée à un système tiers).

3.6 Interface (future) Application ↔ Pennylane (I6)

Nom : Interface Application – Pennylane (hors MVP)
Type : Logicielle (API ou fichier).

Caractéristique	Détail
Statut	Non réalisée en MVP. Facturation gérée dans Pennylane ; saisie manuelle dans l'appli.
Évolution possible	Import ou synchronisation (API Pennylane ou export CSV/import) : factures, éventuellement paiements. À préciser (fréquence, périmètre, authentification).
Impact sur les interfaces	Nouvelle entrée de données (lecture depuis Pennylane) ; possiblement mise à jour des tables factures (et paiements).

---

4. Récapitulatif des interfaces

Id	Interface	Sens principal	Protocole / support
I1	Utilisateur ↔ Application	Bidirectionnel (saisies, écrans, export)	HTTP, navigateur, HTML/CSV
I2	Frontend ↔ API	Front → API (GET/POST), API → Front (JSON/CSV)	HTTP, JSON
I3	API ↔ Base de données	API → DB (SQL), DB → API (résultats)	PDO, MySQL/MariaDB
I4	API ↔ Fichiers mock	Lecture/écriture JSON	Fichiers .json, loadMock/saveMock
I5	Application → Utilisateur (export)	API → navigateur (fichier CSV)	HTTP, CSV
I6	Application ↔ Pennylane	— (hors MVP)	À définir (API ou fichier)

Ce document complète l'Architecture Decision Document (architecture.md) et le schéma de chiffrage (schema-chiffrage-projet.md). À mettre à jour en cas d'ajout d'écrans, d'endpoints ou d'intégrations externes.