BTS SIO — Option SLAM

Documentation Technique de Réalisation

Application web de pilotage des adhésions

Laravel 13 — Nexus / Union Patronale du Var

Candidat	Cyrille COUR
Établissement	Aristée
Organisation	Union Patronale du Var (UPV)
Stack technique	PHP 8.3 / Laravel 13 / MySQL 8.4 / Odoo / Power Automate
Date	Octobre 2026 -> janvier 2027

Sommaire

1. Présentation du projet

1.1 Contexte

Nexus est une application web interne développée pour l'Union Patronale du Var (UPV), organisation professionnelle d'accompagnement d'entreprises adhérentes. Elle est destinée à la direction, qui doit disposer d'une vision consolidée des adhésions sans naviguer entre plusieurs outils.

Les données d'adhésion sont gérées dans l'ERP Odoo, via son module Abonnements. Nexus les récupère automatiquement, les normalise et les stocke dans sa propre base MySQL pour alimenter un tableau de bord de pilotage.

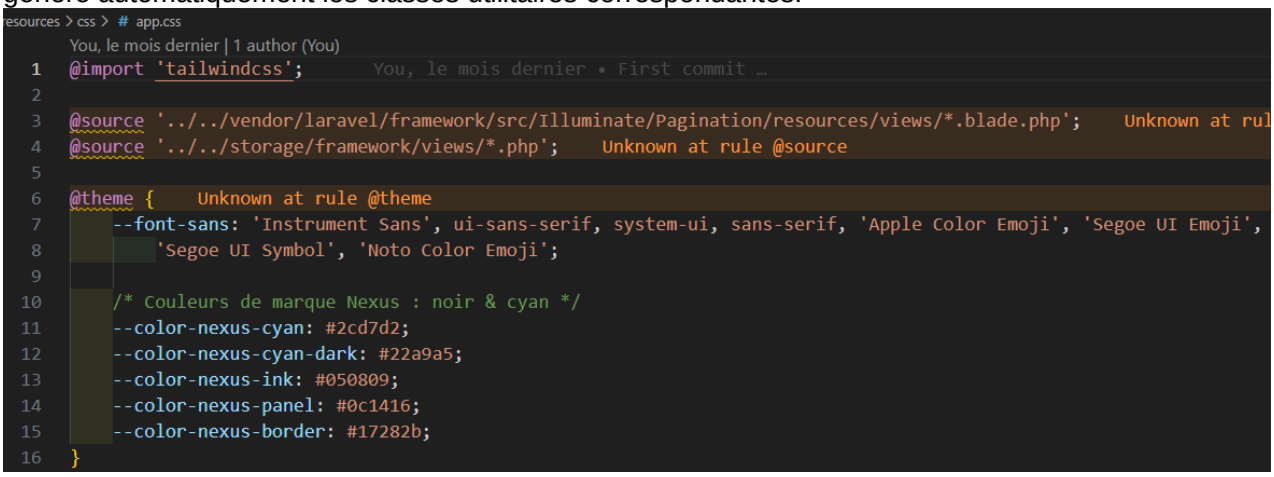
Trois applications de l'organisation (Nexus, Mandats, Papie) exploitent le même référentiel d'entreprises adhérentes, chacune avec sa propre base de données et sa propre API de synchronisation.

1.2 Périmètre fonctionnel

Module	Rôle
Authentification	Connexion par session, deux rôles (administrateur, consultant), gestion des comptes réservée aux administrateurs
Adhérents	Liste paginée des entreprises adhérentes avec recherche multi-colonnes, fiche détaillée de 32 champs
API de synchronisation	Endpoint d'ingestion recevant les abonnements Odoo, appelé quotidiennement par un flux Power Automate
Comptabilité, Juridique, Social	Rubriques préfigurées dans la navigation, contenu à développer

1.3 Identité visuelle

Thème sombre, noir profond et cyan (#2cd7d2), typographie Instrument Sans auto-hébergée. Les couleurs de marque sont déclarées une seule fois via la directive @theme de Tailwind CSS 4, ce qui génère automatiquement les classes utilitaires correspondantes.



2. Stack technique et environnement

2.1 Plateforme

Domaine	Technologie	Rôle
Langage back-end	PHP 8.3.31	Langage de l'application
Framework	Laravel 13	MVC, Eloquent ORM, Blade, migrations, authentification
Base de données	MySQL 8.4	SGBD relationnel (base nexus)
Front-end	Tailwind CSS 4 + Vite	Thème de marque via @theme, build des assets
Environnement de dev	Laragon (Windows 11)	Stack WAMP intégrée, vhost nexus.test
Hébergement production	Evolix / OVH	nexus.upv.org, accès FTP uniquement
ERP source	Odoo (API JSON-RPC)	Source des données d'abonnement
Automatisation	Power Automate	Connecteur Odoo vers Nexus, exécution quotidienne
Test d'API	Hoppscotch	Validation manuelle de l'endpoint de synchronisation
Versionnement	Git	Branche main

2.2 Dépendances back-end

Paquet	Version	Rôle
`laravel/framework`	v13.18.1	Framework MVC complet
`laravel/tinker`	v3.0.2	REPL interactif pour tester modèles et requêtes
`phpunit/phpunit`	12.5.30	Tests unitaires et fonctionnels
`laravel/pint`	v1.29.3	Formatage automatique du code (PSR-12)
`laravel/pail`	v1.2.7	Lecture des logs applicatifs en temps réel

2.3 Dépendances front-end

Paquet	Version	Rôle
`tailwindcss`	4.3.2	Framework CSS utilitaire
`@tailwindcss/vite`	4.3.2	Intégration Tailwind 4 dans Vite
`vite`	8.1.3	Bundler et serveur de développement
`laravel-vite-plugin`	3.1.0	Pont Vite / Laravel, directive @vite, module fonts
`concurrently`	9.2.3	Exécution parallèle des tâches de développement

La police Instrument Sans est servie localement via le module fonts de laravel-vite-plugin. Les fichiers .woff2 sont compilés dans public/build/assets/ : aucune requête vers un CDN tiers, donc aucune fuite d'adresse IP des utilisateurs et aucune connexion externe bloquante au rendu.

2.4 Configuration applicative

Fonction	Driver	Support
Sessions	`database`	Table sessions
Cache	`database`	Tables cache, cache_locks
Files d'attente	`database`	Tables jobs, job_batches, failed_jobs
Envoi d'e-mails	`log`	Écriture dans les logs, aucun envoi réel
Langue	`fr`	APP_LOCALE=fr, repli en

Sessions, cache et files d'attente reposent sur MySQL plutôt que sur Redis : un service de moins à installer et à maintenir, pour une charge d'application interne à faible nombre d'utilisateurs simultanés.

2.5 Technologies écartées

Technologie	Motif
Laravel Breeze / Jetstream / Fortify	Authentification écrite à la main pour n'embarquer que le nécessaire (environ 55 lignes de contrôleur)
Alpine.js / Vue / React / Livewire	Aucun besoin d'interactivité complexe ; le menu mobile est réalisé en CSS pur
SQLite	Ne permet pas d'accéder à la table corporation partagée en MySQL
Laravel Sanctum / Passport	Un jeton statique partagé suffit pour un unique webhook serveur-à-serveur
Module Odoo sur mesure	Éviterait les appels chaînés, mais nécessiterait un accès développeur à l'instance Odoo

3. Architecture de l'application

3.1 Modèle architectural

Nexus est une application monolithique rendue côté serveur. Les contrôleurs retournent des vues Blade compilées en HTML : aucun framework front (Vue, React, Inertia, Livewire), aucun JavaScript applicatif. Le rendu serveur assure une protection CSRF native et des pages immédiatement fonctionnelles.

Le seul point d'entrée API, /api/sync/abonnements, n'est pas consommé par le front de Nexus : c'est un webhook d'ingestion appelé par Power Automate, hors de toute session utilisateur.

3.2 Routage

```php
return Application::configure(basePath: dirname(__DIR__))
    ->withRouting(
        web: __DIR__.'/../routes/web.php',
        api: __DIR__.'/../routes/api.php',
        commands: __DIR__.'/../routes/console.php',
        health: '/up',
    )
```

Le paramètre api n'est pas déclaré par défaut dans le squelette Laravel. Sans cette ligne, les routes définies dans routes/api.php sont ignorées, sans message d'erreur. Sa déclaration active le préfixe /api et le groupe de middleware stateless.

Les routes web sont organisées en groupes de middleware imbriqués : guest pour la connexion, auth pour les pages authentifiées, auth + admin pour la gestion des comptes.

3.3 Chaîne de traitement des données

```
Odoo (ERP)
   |  search_read : sale.order, sale.order.line, product.template
   v
Power Automate (flux planifie quotidien)
   |  POST JSON + en-tete X-Sync-Token
   v
Nexus : SyncController::syncAbonnements()
   |  normalisation, resolution des relations, upsert
   v
MySQL : table abonnements  <->  table corporation (lecture seule)
   v
Vues Blade : tableau de bord de direction
```

3.4 Coquille applicative

Toutes les pages authentifiées étendent un layout unique comportant une barre latérale fixe. La navigation est pilotée par un tableau PHP où chaque entrée porte son nom de route, son libellé et le tracé SVG de son icône : ajouter une rubrique revient à ajouter une ligne.

Fichier : `resources/views/layouts/app-shell.blade.php`

```php
$nav = [
    ['route' => 'dashboard',    'label' => 'Tableau de bord', 'icon' => '...'],
    ['route' => 'adherents',    'label' => 'Adhérents',       'icon' => '...'],
    ['route' => 'comptabilite', 'label' => 'Comptabilité',    'icon' => '...'],
];
```

L'onglet actif est mis en évidence par request()->routeIs($item['route']). Le menu mobile est obtenu par une case à cocher masquée et la variante peer de Tailwind (peer-checked:translate-x-0), sans JavaScript.

4. Base de données

4.1 Table corporation (table héritée)

La table corporation recense les entreprises adhérentes. Elle n'est pas créée par une migration du projet : elle préexiste dans la base nexus, importée depuis l'application Mandats qui l'alimente depuis Odoo. Nexus la consomme en lecture seule.

Elle s'écarte des conventions Laravel sur trois points, qui conditionnent toute la suite :

- nom au singulier : corporation, et non corporations ;
- clé primaire id de type INT signé, et non BIGINT UNSIGNED ;
- aucune colonne created_at ni updated_at.

Les 32 colonnes exploitées se répartissent en cinq groupes :

Groupe	Colonnes
Identité	`name, nom_commercial, siret, forme_juridique, naf, secteur_activite, effectif`
Adresse	`address, address_comp, zip, city, territoire`
Contact	`telephone, mobile, email, site_web, assistante_sociale`
Adhésion	`est_adherent, est_membre, date_adhesion, reducbox, service_social, gsc`
Système	`id, id_pm, id_adh, odoo_id, last_synced_at, remove`

La colonne odoo_id est la charnière de l'intégration : elle fait correspondre un partenaire Odoo à une entreprise du référentiel local.

4.2 Table abonnements

```php
use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

You, il y a 3 semaines | 1 author (You)
return new class () extends Migration {
    public function up(): void
    {
        Schema::create('abonnements', function (Blueprint $table) {
            $table->id();

            // corporation.id est un `int` SIGNÉ (pas unsigned),
            // table nommée `corporation` au singulier
            $table->integer('corporation_id');
            $table->foreign('corporation_id')
                ->references('id')->on('corporation')
                ->cascadeOnDelete();

            $table->unsignedBigInteger('odoo_subscription_id')->unique();
            $table->string('reference')->nullable();

            $table->date('prochaine_facture')->nullable();
            $table->decimal('montant_recurrent', 10, 2)->default(0);
            $table->enum('periodicite', ['mensuel', 'trimestriel', 'annuel'])->nullable();
            $table->string('statut')->nullable();
            $table->string('type_adhesion')->nullable();

            $table->json('raw_payload')->nullable();
            $table->timestamp('last_synced_at')->nullable();

            $table->timestamps();

            $table->index(['periodicite', 'type_adhesion', 'statut']);
            $table->index('prochaine_facture');
        });
    }

    public function down(): void
    {
        Schema::dropIfExists('abonnements');
    }
};
```

Colonne	Type	Rôle
`corporation_id`	int signé	Clé étrangère vers corporation.id
`odoo_subscription_id`	bigint unsigned, UNIQUE	Identifiant Odoo, clé d'idempotence de la synchronisation
`reference`	varchar(255)	Référence lisible de la commande Odoo (ex. S04335)
`prochaine_facture`	date	Date de prochaine facturation
`montant_recurrent`	decimal(10,2)	Montant normalisé en base mensuelle
`periodicite`	enum	mensuel, trimestriel ou annuel
`statut`	varchar(255)	Code subscription_state d'Odoo, conservé brut
`type_adhesion`	varchar(255)	Catégorie du produit facturé, valeur dérivée
`raw_payload`	json	Copie intégrale du payload Odoo reçu
`last_synced_at`	timestamp	Horodatage de la dernière synchronisation

4.3 Décisions de conception

Décision	Motif
odoo_subscription_id en contrainte UNIQUE	Garantit l'idempotence au niveau du SGBD : un lot rejoué ne peut pas créer de doublon, indépendamment de la logique applicative
decimal(10,2) plutôt que float	Type exact, sans erreur d'arrondi binaire, obligatoire sur des montants financiers
Colonne raw_payload en JSON	Permet d'auditer la donnée reçue, de rejouer une transformation si le mapping évolue, et d'exploiter plus tard des champs sans colonne dédiée
Index composite (periodicite, type_adhesion, statut)	Couvre le filtrage et l'agrégation prévus par le tableau de bord
Index simple sur prochaine_facture	Couvre les tris et les plages de dates (échéances à venir)
cascadeOnDelete()	La suppression d'une entreprise supprime ses abonnements : aucun orphelin possible

5. Modèles Eloquent

5.1 Corporation

```php
class Corporation extends Model
{
    protected $table = 'corporation';

    public $timestamps = false;
}
```

Chacune des deux propriétés corrige une convention Laravel incompatible avec la table héritée. Sans $table, Eloquent pluralise le nom du modèle et cherche une table corporations inexistante. Sans $timestamps = false, tout appel à save() tente d'écrire dans des colonnes created_at et updated_at absentes.

Aucun $fillable n'est déclaré : le modèle n'est utilisé qu'en lecture, et cette absence bloque par défaut toute affectation de masse, ce qui protège la table externe d'une écriture accidentelle.

5.2 Abonnement

```php
class Abonnement extends Model
{
    protected $fillable = [
        'corporation_id',
        'odoo_subscription_id',
        'reference',
        'prochaine_facture',
        'montant_recurrent',
        'periodicite',
        'statut',
        'type_adhesion',
        'raw_payload',
        'last_synced_at',
    ];

    protected $casts = [      You, il y a 3 semaines • accueil de API a
        'prochaine_facture' => 'date',
        'montant_recurrent' => 'decimal:2',
        'raw_payload'       => 'array',
        'last_synced_at'    => 'datetime',
    ];

    public function corporation(): BelongsTo
    {
        return $this->belongsTo(Corporation::class, 'corporation_id');
    }
}
```

Les casts assurent la conversion entre types SQL et types PHP dans les deux sens. Le cast raw_payload en array permet au contrôleur d'écrire directement un tableau PHP, Eloquent se chargeant de la sérialisation JSON. Les casts date et datetime retournent des instances Carbon, ce qui donne accès au formatage localisé et à l'arithmétique de dates.

La clé étrangère est déclarée explicitement dans la relation belongsTo, alors que la convention l'aurait déduite : le modèle lié ne respectant déjà pas les conventions, aucune déduction implicite n'est laissée au framework.

5.3 User

```php
#[Fillable(['name', 'email', 'password', 'role'])]
#[Hidden(['password', 'remember_token'])]
class User extends Authenticatable
{
    /** @use HasFactory<UserFactory> */
    use HasFactory, Notifiable;

    /**
     * Get the attributes that should be cast.
     *
     * @return array<string, string>
     */
    protected function casts(): array
    {
        return [
            'email_verified_at' => 'datetime',
            'password' => 'hashed',
            'role' => UserRole::class,
        ];
    }

    /**
     * L'utilisateur possède-t-il le rôle Administrateur ?
     */
    public function isAdmin(): bool
    {
        return $this->role === UserRole::Admin;
    }
}       You, le mois dernier • First commit …
```

Le cast role vers l'enum UserRole transforme la chaîne stockée en base en instance typée : la comparaison porte sur un type et non sur une chaîne, la validation utilise Rule::enum(UserRole::class), et l'affichage passe par la méthode label() de l'enum. Le cast password en hashed garantit qu'aucun mot de passe ne peut être stocké en clair, même si un appel à Hash::make() était omis dans un contrôleur.

6. Contrôleurs et sécurité applicative

6.1 Authentification

```php
public function store(Request $request): RedirectResponse
{
    $credentials = $request->validate([
        'email' => ['required', 'string', 'email'],
        'password' => ['required', 'string'],
    ]);

    if (! Auth::attempt($credentials, $request->boolean('remember'))) {
        throw ValidationException::withMessages([
            'email' => 'Ces identifiants ne correspondent à aucun compte.',
        ]);
    }

    $request->session()->regenerate();

    return redirect()->intended(route('dashboard'));
}
```

La régénération de session après connexion protège contre la fixation de session. Le message d'erreur générique empêche l'énumération de comptes en ne révélant pas lequel des deux champs est erroné. À la déconnexion, la session est invalidée et le jeton CSRF régénéré.

6.2 Gestion des comptes

L'inscription publique a été supprimée : sur un outil interne exposant des données d'entreprises adhérentes, un formulaire d'inscription ouvert constituait une surface d'attaque injustifiée. Les comptes sont créés depuis l'écran Paramètres, réservé aux administrateurs par le middleware EnsureUserIsAdmin.

Cette restriction impose deux garde-fous : un administrateur ne peut ni se retirer son propre rôle, ni supprimer son propre compte, faute de quoi un administrateur unique rendrait la gestion des comptes définitivement inaccessible.

```php
public function updateRole(Request $request, User $user): RedirectResponse
{
    $validated = $request->validate([
        'role' => ['required', Rule::enum(UserRole::class)],
    ]);

    // On empêche un admin de se retirer à lui-même ses droits
    // (pour éviter de se verrouiller hors de la gestion).
    if ($user->is($request->user()) && $validated['role'] !== UserRole::Admin->value) {
        return back()->withErrors([
            'role' => "Vous ne pouvez pas retirer votre propre rôle d'administrateur.",
        ]);
    }

    $user->update(['role' => $validated['role']]);

    return back()->with('status', 'role-mis-a-jour');
}
```

Un compte administrateur d'amorçage est créé par le DatabaseSeeder à l'aide de firstOrCreate, ce qui rend le seeder rejouable sans violer la contrainte d'unicité sur l'e-mail.

6.3 Restriction au niveau des données

```php
public function index(): View
{
    // La liste des comptes n'est utile qu'aux administrateurs.
    $users = auth()->user()->isAdmin()    Undefined method 'user'.
        ? User::orderBy('name')->get()
        : collect();

    return view('settings.index', [
        'users' => $users,
    ]);
```

Un consultant reçoit une collection vide : la vue ne peut pas exposer une donnée qu'elle n'a pas reçue. Masquer la section uniquement dans le template aurait laissé transiter les données jusqu'au rendu.

6.4 Synthèse des mesures de sécurité

Mesure	Mise en œuvre
Hachage des mots de passe	bcrypt coût 12, cast password en hashed sur le modèle
Politique de mot de passe	Password::defaults() et règle confirmed
Protection CSRF	Middleware web, directive @csrf, balise meta csrf-token
Protection XSS	Échappement automatique de Blade, aucun affichage non échappé
Injection SQL	Query builder et Eloquent exclusivement, paramètres liés
Fixation de session	session()->regenerate() après connexion
Énumération de comptes	Message d'erreur de connexion générique
Contrôle d'accès	Groupes de middleware auth et admin, réponse 403
Anti-verrouillage	Retrait de son propre rôle admin et auto-suppression interdits
Changement de mot de passe	Règle current_password obligatoire
Affectation de masse	#[Fillable] sur User, $fillable sur Abonnement, aucun sur Corporation
Fuite de données sensibles	#[Hidden] sur password et remember_token
Authentification API	En-tête X-Sync-Token, refus si le jeton n'est pas configuré
Secrets	SYNC_TOKEN et identifiants en .env, exclu du versionnement

7. Intégration Odoo

7.1 Données récupérées

Les adhésions sont gérées dans le module Abonnements d'Odoo, construit sur le modèle sale.order. Les champs retenus ont été relevés en mode développeur sur l'instance de production.

Champ affiché	Nom technique	Modèle	Usage dans Nexus
Client	`partner_id`	`sale.order`	Résolution de l'entreprise adhérente
Référence	`name`	`sale.order`	Référence lisible de l'abonnement
Date de la prochaine facture	`next_invoice_date`	`sale.order`	Échéance de facturation
Récurrent total	`recurring_total`	`sale.order`	Montant récurrent, à normaliser
Plan récurrent	`plan_id`	`sale.order`	Périodicité du plan
Statut d'abonnement	`subscription_state`	`sale.order`	Statut, conservé brut
Prix unitaire de ligne	`price_subtotal`	`sale.order.line`	Identification de la ligne payante
Catégorie de produits	`categ_id`	`product.template`	Type d'adhésion

Les champs Carte Odyssée et Service Social, ainsi que le vendeur et les activités, ont été écartés : ils ne portent pas d'information de revenu. Ils restent néanmoins accessibles dans la colonne raw_payload.

7.2 Détermination du type d'adhésion

Le type d'adhésion (convention, effectif, masse salariale) n'est pas un champ de l'abonnement. C'est la catégorie de produit du produit vendu dans la ligne de commande dont le montant est strictement positif.

Une commande d'abonnement comporte en effet plusieurs lignes : une cotisation payante et des prestations incluses à 0 euro (adhésion syndicale, service social, carte Odyssée). Seule la ligne payante détermine le type réel.

Cette règle impose de chaîner trois interrogations d'Odoo : sale.order pour l'abonnement, sale.order.line pour ses lignes, product.template pour la catégorie du produit vendu.

7.3 Correspondance des entreprises

Le champ Client d'un abonnement est un many2one vers res.partner. Le contrôle en mode développeur confirme que les clients concernés sont de type Société, sans société apparentée : la correspondance est directe, sans remontée par un parent_id.

```
sale.order.partner_id  ->  res.partner (Société)  ->  corporation.odoo_id
```

7.4 Endpoint d'ingestion

```php
Route::post('/sync/abonnements', [SyncController::class, 'syncAbonnements'])
    ->name('sync.abonnements');
```

L'endpoint reçoit un unique payload JSON regroupant les trois tableaux issus d'Odoo : orders, lines et products. L'authentification reprend le mécanisme déjà en place sur les applications Mandats et Papie : un jeton statique transmis en en-tête, comparé à une valeur de configuration.

```php
class SyncController extends Controller
{
    public function syncAbonnements(Request $request): JsonResponse
    {
        // ── Vérification du token ──────────────────────────
        $token      = $request->header('X-Sync-Token');
        $validToken = config('services.sync.token');

        if (empty($validToken) || $token !== $validToken) {
            return response()->json(['error' => 'Unauthorized'], 401);
        }

        $data = $request->json()->all();

        if (!isset($data['orders']) || !is_array($data['orders'])) {
            return response()->json(['error' => 'Invalid payload: missing orders'], 400);
        }
```

Le jeton est lu depuis la configuration, elle-même alimentée par la variable d'environnement SYNC_TOKEN. Un jeton distinct est utilisé pour chaque environnement.

```php
'sync' => [
'token' => env('SYNC_TOKEN'),
],
```

7.5 Normalisation du format Odoo

Odoo, héritant de son protocole XML-RPC, sérialise ses données selon des conventions propres : un champ vide vaut false et non null, une relation many2one est un tableau de deux éléments contenant l'identifiant et le libellé. Trois fonctions encapsulent cette traduction.

```php
// ── Helpers ──────────────────────────────────────────────
$val   = fn ($v) => ($v === false || $v === null) ? null : $v;
$label = fn ($v) => (is_array($v) && isset($v[1])) ? $v[1] : null;
$refId = fn ($v) => (is_array($v) && isset($v[0])) ? $v[0] : null; // many2one → id
($v) => (is_array($v) && isset($v[0])) ? $v[0] : null;
```

7.6 Reconstitution de la jointure

L'API d'Odoo interroge un seul modèle par appel et ne réalise aucune jointure. Celle-ci est reconstituée côté Laravel par deux tableaux associatifs, ce qui ramène chaque résolution à un accès direct plutôt qu'à un parcours imbriqué.

```php
// ── Index : product_id → catégorie (label) ──────────────
$productCategMap = [];
foreach ($products as $product) {
    $productCategMap[$product['id']] = $label($product['categ_id'] ?? null);
}

// ── Index : order_id → type_adhesion (ligne au prix > 0) ─
$typeAdhesionParOrder = [];
foreach ($lines as $line) {
    $orderId = $refId($line['order_id'] ?? null);
    $montant = (float) ($line['price_subtotal'] ?? 0);

    if ($orderId && $montant > 0) {
        $productId = $refId($line['product_id'] ?? null);
        $typeAdhesionParOrder[$orderId] = $productCategMap[$productId] ?? null;
    }
}
```

7.7 Persistance

Chaque abonnement est recherché par son identifiant Odoo puis créé ou mis à jour. L'écriture est effectuée manuellement plutôt que par updateOrCreate, afin de compter séparément les créations et les mises à jour.

```php
        // — Résolution corporation via partner_id → odoo_id —
        $partnerId = $refId($item['partner_id'] ?? null);
        $corp      = $partnerId ? Corporation::where('odoo_id', $partnerId)->first() : null;

        if (!$corp) {
            Log::warning("Sync abonnements: aucune corporation pour partner_id={$partnerId}, order=" . ($item['name'] ?? '?'));
            $skipped++;
            continue;
        }

        $abo   = Abonnement::where('odoo_subscription_id', $odooId)->first();
        $isNew = false;

        if (!$abo) {
            $abo   = new Abonnement();
            $isNew = true;      You, il y a 3 semaines • accueil de API abonnement
            $created++;
        } else {
            $updated++;
        }

        $prochaineFacture = null;
        if (!empty($item['next_invoice_date'])) {
            try {
                $prochaineFacture = new \DateTime($item['next_invoice_date']);
            } catch (\Exception $e) {
                $prochaineFacture = null;
            }
        }

        $periodicite = $this->mapPeriodicite($label($item['plan_id'] ?? null));

        $abo->fill([
            'corporation_id'       => $corp->id,
            'odoo_subscription_id' => $odooId,
            'reference'            => $val($item['name'] ?? null),
            'prochaine_facture'    => $prochaineFacture,
            'montant_recurrent'    => $this->toMonthly((float) ($item['recurring_total'] ?? 0), $periodicite),
            'periodicite'          => $periodicite,
            'statut'               => $val($item['subscription_state'] ?? null),
            'type_adhesion'        => $typeAdhesionParOrder[$odooId] ?? null,
            'raw_payload'          => $item,
            'last_synced_at'       => now(),
        ]);

        $abo->save();
    }
```

Un abonnement dont le partenaire Odoo n'a pas de correspondance locale est ignoré, comptabilisé et journalisé, sans interrompre le lot : un référentiel partiellement désynchronisé ne doit pas faire échouer la synchronisation entière.

La réponse renvoie un compte rendu exploitable par le système appelant, qui peut détecter un nombre anormal d'enregistrements ignorés sans consulter les logs du serveur.

```php
    return response()->json([
        'success' => true,
        'created' => $created,
        'updated' => $updated,
        'skipped' => $skipped,
        'total'   => $created + $updated,
    ]);
}       You, il y a 3 semaines • accueil d
```

7.8 Normalisation des montants et des périodicités

Le champ recurring_total exprime le montant dans l'unité de la périodicité du plan : un abonnement annuel à 3 600 euros et un abonnement mensuel à 300 euros représentent le même revenu mensuel, mais seraient stockés bruts comme 3600 et 300. Toute somme agrégée serait alors dénuée de sens, alors que le revenu récurrent est précisément l'indicateur attendu du tableau de bord.

Deux méthodes privées assurent la normalisation. Le libellé du plan étant saisi par des utilisateurs, sa reconnaissance repose sur une recherche de sous-chaîne plutôt que sur une égalité stricte.

```php
    private function mapPeriodicite(?string $planLabel): ?string
    {
        $label = strtolower($planLabel ?? '');

        return match (true) {
            str_contains($label, 'mensuel')     => 'mensuel',
            str_contains($label, 'trimestriel') => 'trimestriel',
            str_contains($label, 'annuel')      => 'annuel',
            default => null,
        };
    }

    private function toMonthly(float $recurringTotal, ?string $periodicite): float
    {
        return match ($periodicite) {
            'trimestriel' => round($recurringTotal / 3, 2),
            'annuel'      => round($recurringTotal / 12, 2),
            default       => $recurringTotal, // mensuel, ou inconnu → pas de conversion
        };
    }
}
```

Périodicité	recurring_total Odoo	Calcul	montant_recurrent
mensuel	300	aucune conversion	300,00
trimestriel	900	900 / 3	300,00
annuel	3 600	3 600 / 12	300,00

7.9 Validation de l'endpoint

L'endpoint a été validé par appels manuels avec Hoppscotch avant tout branchement du flux automatisé.

Test	Requête	Résultat attendu
Jeton invalide	En-tête X-Sync-Token erroné	401 Unauthorized
Lot vide	Corps : orders vide	200, compteurs à zéro
Cas réel	Un abonnement, une ligne payante, une ligne à 0 euro	200, created à 1, type_adhesion résolu
Rejeu du même lot	Payload identique renvoyé	200, updated à 1, aucun doublon

Le contrôle en base après chaque appel a confirmé la résolution de corporation_id à partir de partner_id, la sélection de la catégorie de la ligne payante, et la conversion du montant en base mensuelle.

7.10 Flux Power Automate

Le connecteur est un flux unique, planifié quotidiennement, pointant vers l'environnement de production. Il enchaîne huit actions sans boucle imbriquée : les extractions sont effectuées à plat, la jointure étant reconstituée côté Laravel.

Étape	Action	Rôle
1	Recurrence	Déclenchement quotidien
2	HTTP — Get Orders	search_read sur sale.order
3	Select — Extract Order IDs	Extraction des identifiants d'abonnements
4	HTTP — Get Lines	search_read sur sale.order.line filtré sur ces identifiants
5	Select — Extract Product IDs	Extraction des identifiants de produits
6	HTTP — Get Products	search_read sur product.template, champ categ_id
7	Compose — Build Payload	Assemblage des trois tableaux
8	HTTP — Send to Nexus	POST vers /api/sync/abonnements

Étape 2 — Get Orders

```
{
  "jsonrpc": "2.0",
  "method": "call",
  "params": {
    "service": "object",
    "method": "execute_kw",
    "args": [
      "david-styleo-upv-main-15205613",
      20,
      "TOKEN",
      "sale.order",
      "search_read",
      [
        [
          [
            "subscription_state",
            "!=",
            false
          ]
        ]
      ],
      {
        "fields": [
          "name",
          "partner_id",
          "next_invoice_date",
          "recurring_total",
          "plan_id",
          "subscription_state"
        ],
        "limit": 2000
      }
    ]
  }
}
```

Le domaine filtre sur subscription_state différent de false : ce champ n'est renseigné que sur les commandes qui sont réellement des abonnements. Tous les statuts sont récupérés, y compris les abonnements résiliés, afin de permettre des statistiques sur les adhésions terminées ; le filtrage d'affichage est assuré côté application.

Étapes 3 et 5 — Actions Select

Les actions Select transforment un tableau d'objets en tableau de valeurs simples. Le champ Carte est basculé en mode Texte et reçoit une expression préfixée par l'arobase.

```
Début : @body('HTTP_"Get_Orders"')?['result']
Carte  : @item()?['id']

Début : @body('HTTP_"Get_Lines"')?['result']
Carte  : @item()?['product_id'][0]
```

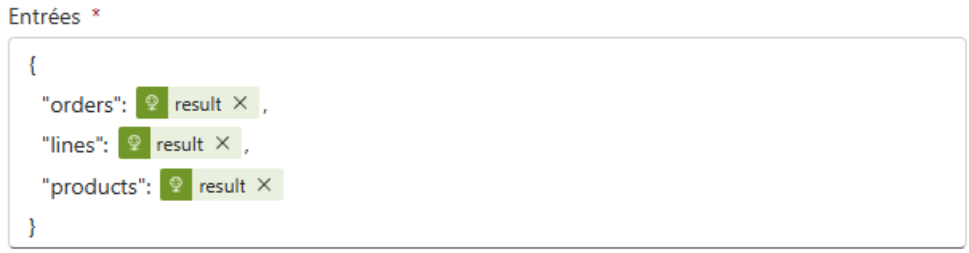
Le suffixe [0] extrait l'identifiant d'une relation many2one, dont le second élément est le libellé.

Étape 4 — Get Lines

```
"sale.order.line",
"search_read",
[[["order_id", "in", "@body('Extract_Order_IDs')"]]],
{
  "fields": ["order_id", "product_id", "price_subtotal"],
  "limit": 10000
}
```

Une expression placée seule entre guillemets comme valeur JSON est résolue par Power Automate en la valeur réelle de l'expression, ici un tableau d'identifiants, et non en chaîne de caractères.

Étape 7 — Build Payload



L'action Compose assemble les trois résultats au format attendu par l'endpoint. Le suffixe ?['result'] est nécessaire : le corps d'une réponse JSON-RPC encapsule les données dans une propriété result.

Étape 8 — Send to Nexus

Paramètre	Valeur
URI	https://nexus.upv.org/api/sync/abonnements
Méthode	POST
En-tête	Content-Type: application/json
En-tête	X-Sync-Token: <jeton de production>
Corps	Sortie de l'action Build Payload

Un seul flux est maintenu, connecté à la production. Les environnements local et de pré-production sont alimentés par export et import de la base, ce qui évite de multiplier les flux à maintenir.

8. Problèmes rencontrés et solutions

8.1 Incompatibilité de types sur la clé étrangère

La création de la table abonnements devait référencer corporation.id. La forme idiomatique de Laravel échoue sur deux points : foreignId() génère une colonne BIGINT UNSIGNED alors que corporation.id est un INT signé, et constrained() déduit le nom de la table en pluralisant la colonne, soit corporations au lieu de corporation.

MySQL exige un typage strictement identique entre les deux colonnes d'une contrainte de clé étrangère et renvoie successivement les erreurs 1824 puis 3780.

```
// corporation.id est un `int` SIGNÉ (pas unsigned),
// table nommée `corporation` au singulier
$table->integer('corporation_id');
$table->foreign('corporation_id')
    ->references('id')->on('corporation')
    ->cascadeOnDelete();
```

Le commentaire dans la migration documente l'écart, afin qu'une reprise ultérieure ne rétablisse pas la forme idiomatique et ne reproduise pas l'erreur.

8.2 Modèle Eloquent et table non conventionnelle

Deux erreurs successives à l'exécution ont mis en évidence les conventions implicites d'Eloquent : SQLSTATE[42S02] sur une table nexus.corporations inexistante, puis SQLSTATE[42S22] sur une colonne created_at absente. La surcharge explicite des propriétés $table et $timestamps résout les deux.

8.3 Routes API non chargées

Les routes déclarées dans routes/api.php renvoyaient une erreur 404. Sur Laravel 11 et suivants, le fichier n'est pas chargé par défaut : le paramètre api doit être déclaré explicitement dans withRouting(), faute de quoi le fichier est ignoré sans avertissement.

8.4 Rejet des lots légitimement vides

Le contrôle initial du payload utilisait empty(), qui retourne true pour un tableau vide comme pour une clé absente. Une synchronisation sans nouvel abonnement, cas nominal, recevait donc une réponse 400. Le contrôle a été remplacé par isset(), qui teste uniquement la présence de la clé.

```
// Avant :
if (empty($data['orders']) || ! is_array($data['orders'])) { ... }

// Après :
if (! isset($data['orders']) || ! is_array($data['orders'])) { ... }
```

8.5 Extraction erronée du statut

Le statut était extrait par $label(), fonction destinée aux relations many2one. Or subscription_state est une chaîne simple : $label() retourne null sur une chaîne, et seul le repli corrigeait le résultat. L'extraction a été ramenée à $val().

```
// Avant :
'statut' => $label($item['subscription_state'] ?? null)
                ?? $val($item['subscription_state'] ?? null),

// Après :
'statut' => $val($item['subscription_state'] ?? null),
```

8.6 Montants non comparables

La première version lisait le champ recurring_monthly, qui s'est révélé indisponible sur le modèle sale.order : il appartient au modèle de reporting sale.subscription.report. Le champ effectivement disponible, recurring_total, exprime le montant dans l'unité de la périodicité du plan. La méthode toMonthly() ramène l'ensemble à une base mensuelle comparable, sans dépendre d'un modèle Odoo supplémentaire.

8.7 Configuration du serveur de production

L'hébergement Evolix/OVH ne donne pas d'accès terminal : les commandes artisan ne peuvent pas être exécutées à distance et le vidage du cache s'effectue en supprimant manuellement les fichiers de bootstrap/cache/ par FTP. Trois points ont été corrigés lors de la mise en ligne.

Symptôme	Cause	Correction
Erreur 500 à l'accès au site	Directive Options -MultiViews -Indexes non autorisée par AllowOverride	Directive commentée dans public/.htaccess
Assets chargés en HTTP sur une page HTTPS	Le serveur ne transmet aucun en-tête HTTPS : HTTPS, X-Forwarded-Proto et X-Forwarded-Ssl sont absents, SERVER_PORT vaut 80	URL::forceScheme('https') dans AppServiceProvider
Échec du flux Power Automate sur une erreur SSL	Certificat non encore généré pour le sous-domaine	Génération du certificat pour nexus.upv.org

```
    public function boot(): void
    {
        \Illuminate\Support\Facades\URL::forceScheme('https');
    }
}
```

8.8 Perte des critères de recherche à la pagination

Les liens de pagination générés par Laravel ne contiennent que le numéro de page : un utilisateur ayant filtré la liste des adhérents retombait sur la liste complète en changeant de page. L'appel à withQueryString() sur le paginateur reporte les paramètres de l'URL courante sur les liens générés.

8.9 Collision des messages d'erreur entre formulaires

L'écran Paramètres présente trois formulaires distincts, dont deux comportent un champ password. Avec le sac d'erreurs par défaut, une erreur de validation s'affichait sous tous les champs de ce nom. La validation utilise désormais validateWithBag() avec un sac nommé par formulaire, et l'affichage cible ce sac.

```
public function updatePassword(Request $request): RedirectResponse
{
    $validated = $request->validateWithBag('updatePassword', [
        'current_password' => ['required', 'current_password'],
        'password' => ['required', 'confirmed', Password::defaults()],
    ]);
```

9. Modules comptabilité

9.1 Module Comptabilité — abonnements

La page Comptabilité, jusque-là une simple vue statique déclarée par Route::view, devient un module complet branché sur les données Odoo synchronisées.

Normalisation des montants

La logique de conversion, qui vivait en privé dans SyncController (cf. 7.8), remonte dans le modèle App\Models\Abonnement sous forme de méthodes statiques réutilisables : normaliserPeriodicite() mappe le libellé de plan Odoo vers mensuel, trimestriel ou annuel, et versMensuel() ramène tout montant récurrent à sa base mensuelle.

Correction apportée à cette occasion : Odoo renvoie ces libellés de plan aussi bien en français qu'en anglais (« Trimestriel », mais aussi « Monthly », « Yearly ») selon la saisie de l'utilisateur Odoo. La version initiale ne reconnaissait que les libellés français, ce qui laissait la périodicité à null et le montant jamais ramené au mois sur une partie du parc d'abonnements.

montant_recurrent est désormais toujours stocké en base mensuelle : c'est la règle centrale du module, elle rend les abonnements additionnables entre eux quelle que soit leur périodicité de facturation. Le montant trimestriel et le montant annuel s'en déduisent par simple facteur (×3, ×12) via les accesseurs montant_trimestriel et montant_annuel, plutôt que d'être stockés séparément.

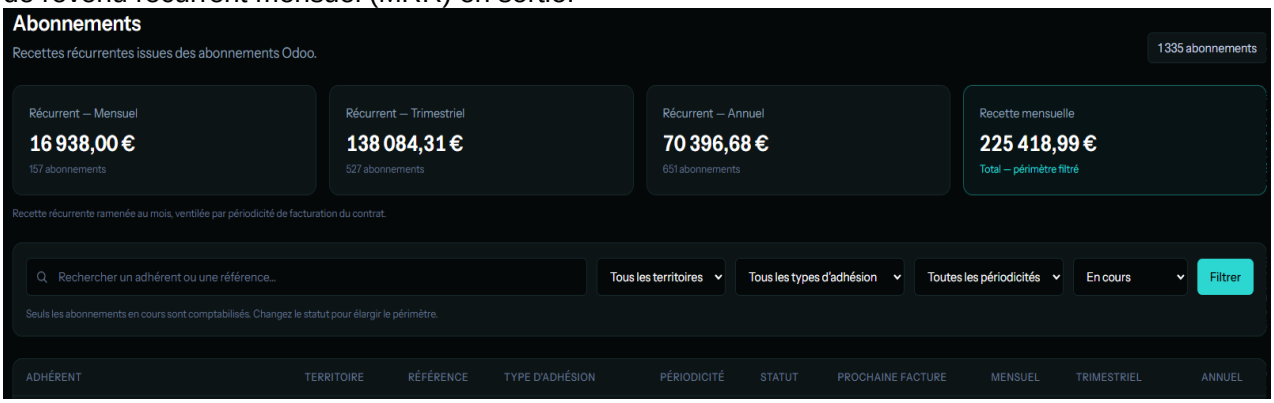
Le modèle reçoit également trois constantes de référence : PERIODICITES, STATUTS — les libellés lisibles des codes subscription_state d'Odoo, par exemple 1_draft pour Brouillon ou 6_churn pour Résilié — et STATUT_EN_COURS, utilisée comme filtre par défaut de la page.

Commande de rattrapage

La correction du bug de reconnaissance des libellés anglais ne nécessite aucune resynchronisation depuis Odoo : la commande artisan App\Console\Commands\RecalculerAbonnements rejoue la périodicité et le montant mensuel de chaque abonnement depuis raw_payload, le payload Odoo brut déjà conservé en base.

```
php artisan abonnements:recalculer --dry-run   # simulation
php artisan abonnements:recalculer             # application
```

Le traitement se fait par lots de 500 (chunkById), avec affichage des écarts ligne par ligne et un total de revenu récurrent mensuel (MRR) en sortie.



Page Comptabilité

AbonnementController et la vue pages/comptabilite.blade.php assurent l'affichage : liste paginée à 50 lignes par page des abonnements, chaque ligne renvoyant vers la fiche de l'adhérent concerné.

Fonctionnalité	Comportement
Filtres	Territoire, type d'adhésion, périodicité, statut, plus une recherche libre sur le nom d'adhérent, le nom commercial et la référence. Les listes se soumettent au changement, la recherche au bouton. Le bouton Réinitialiser ne s'affiche que si un filtre est actif.
Statut par défaut	« En cours » (3_progress) : un contrat résilié ou en brouillon ne rapporte rien, il ne doit pas gonfler les recettes affichées. Le filtre reste ouvrable pour consulter l'ensemble du parc.
Tri	Dix colonnes triables, cycle à trois temps (croissant, décroissant, retour à l'ordre par défaut). Une liste blanche des clés de tri autorisées empêche l'injection d'une clé arbitraire dans orderBy. Le tri survit aux changements de filtre via des champs cachés ; la pagination revient en page 1.
Cartes de recettes	Ventilation du récurrent mensuel par périodicité de facturation, plus un total. Les trois cartes s'additionnent pour donner le total, elles ne le multiplient pas : c'est la lecture d'Odoo, sans extrapolation. Une ligne « indéterminée » explicite l'écart si des contrats ont une périodicité non reconnue, plutôt que de le masquer.
Périmètre des totaux	Les totaux portent sur tout le périmètre filtré, pas sur la seule page affichée à l'écran.
Colonnes Mensuel / Trimestriel / Annuel	Le montant réellement facturé — celui de la périodicité du contrat — ressort en blanc, les deux autres en gris : ce ne sont que des équivalences calculées.

Côté performance, une unique jointure SQL sur corporation sert à la fois au filtre territoire, à la recherche et au tri, à la place de trois sous-requêtes whereHas indépendantes.

9.2 API de synchronisation — personnes morales

Nouvel endpoint POST /api/sync/corporations (SyncCorporationController), protégé par le même en-tête X-Sync-Token que /api/sync/abonnements.

Il applique une stratégie de rapprochement en cascade, pour éviter les doublons lorsque les données Odoo sont incomplètes :

Priorité	Critère	Comportement
1	odoo_id	Recherche prioritaire, absolue.
2	SIRET valide	Non vide et non composé uniquement de zéros ; en cas de doublon, la fiche la plus ancienne l'emporte.
3	Nom exact	Utilisé si le SIRET est invalide ; comparaison insensible à la casse.
4	Aucune correspondance	Création d'une nouvelle fiche.

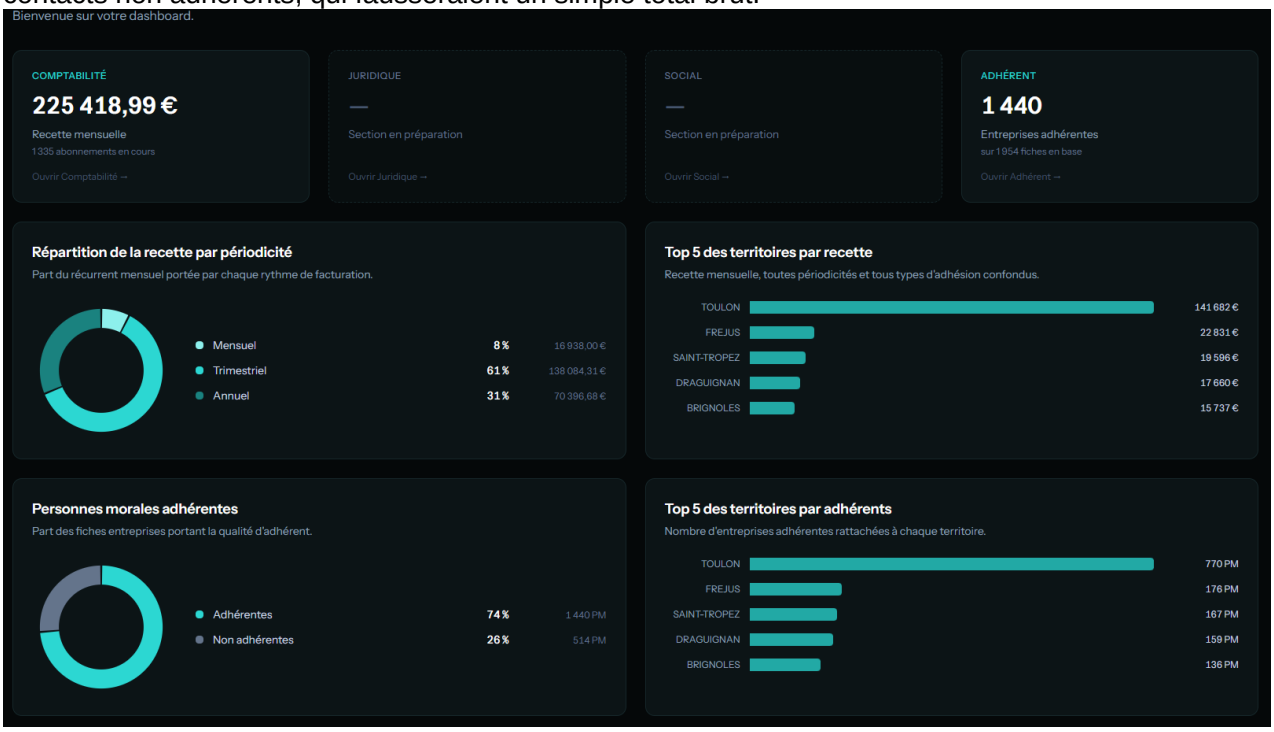
Le mapping couvre l'ensemble des champs x_studio_* d'Odoo : nom commercial, forme juridique, NAF, secteur d'activité, effectif, GSC, territoire, assistante sociale, contacts, ReducBox, service social et date d'adhésion, en plus de last_synced_at. La réponse JSON reprend le même format que l'endpoint des abonnements : {created, updated, total}.

Le modèle Corporation gagne à cette occasion son $fillable complet, ses $casts (booléens, dates, entier) et la relation abonnements().

9.3 Tableau de bord — refonte et graphiques

DashboardController cesse d'être une coquille vide et alimente désormais réellement la page d'accueil, avec un bloc d'en-tête par module (Comptabilité, Juridique, Social, Adhérent), chacun cliquable vers sa page. Les modules pas encore développés conservent la même carte, dans une tenue discrète à bordure pointillée, afin que la grille reste homogène et lisible.

Le bloc Comptabilité affiche la recette mensuelle et le nombre d'abonnements en cours ; le bloc Adhérent affiche le nombre d'entreprises adhérentes rapporté au total des fiches en base. Ce décompte est volontairement séparé du total : la table corporation conserve d'anciennes fiches et des contacts non adhérents, qui fausseraient un simple total brut.



Quatre graphiques

Les graphiques sont tracés en SVG/CSS pur, sans librairie : des formes aussi simples ne justifient pas une dépendance de plusieurs centaines de kilo-octets sur un projet qui n'embarque aujourd'hui aucun JavaScript applicatif (cf. 3.1). Deux composants Blade réutilisables portent l'ensemble :

- <x-graphique-donut> : arcs tracés en stroke-dasharray / stroke-dashoffset sur un même cercle, séparés par un espace de la couleur du fond plutôt qu'un contour — un contour serait de l'encre qui ne porte aucune donnée. Légende et balise <title> accessible pour chaque part ;
- <x-graphique-barres> : barres horizontales à série unique, valeurs posées en bout de barre, ce qui rend un axe gradué superflu.

Graphique	Contenu
Donut	Répartition de la recette par périodicité
Barres	Top 5 des territoires par recette
Donut	Part des personnes morales adhérentes
Barres	Top 5 des territoires par nombre d'adhérents

Périmètre commun : tous les chiffres de la partie comptable partent des abonnements en cours, comme le bloc du haut de page — une part de donut doit toujours pouvoir se rapporter à la recette déjà affichée à l'écran.

Palette

Deux paliers sont ajoutés dans app.css (--color-nexus-cyan-light, --color-nexus-cyan-deep) pour former, avec --color-nexus-cyan déjà en place, une rampe ordinale d'une seule teinte, validée sur le fond nexus-panel : luminosité monotone, écarts d'au moins 0,06, palier le plus sombre à un contraste de 4,03:1.

Cas	Choix de couleur	Raison
Périodicité (mensuel → trimestriel → annuel)	Rampe monochrome	C'est une échelle ordonnée, pas des étiquettes interchangeables : l'ordre se lit directement dans la couleur.
Adhérentes / non adhérentes	Cyan pour la part, gris neutre pour le reste	Ce n'est pas une opposition de même rang mais une part et son reste.
Ensemble de la page	Une seule rampe	Une couleur nouvelle par carte laisserait croire à des correspondances qui n'existent pas entre les graphiques.

9.4 Correctifs et confort

Sujet	Détail
Icônes de la barre latérale	Remplacées par des tracés Heroicons multi-chemins : le tableau icon du layout (cf. 3.4) accepte désormais plusieurs <path> par icône, trait affiné à 1,5.