Intégration Fidélité ↔️ Shop — Parcours utilisateur et plan d'action (v2)

Mise à jour du 19/08/2026 — remplace la version précédente de ce document. Retour équipe intégré : confirmation du modèle 1 tenant core-api = 1 tenant fidelity-api = 1 tenant shop-core-api (Option B).

Convention de statut

Statut	Signification
✅ **Couvert**	Le plan technique existe et s'appuie sur du code déjà vérifié dans les 3 dépôts.
🔧 **Fondation posée**	Le socle est prévu/planifié, mais le détail d'implémentation reste à faire.
❌ **Non couvert**	Nouveau chantier découvert cette semaine, pas encore cadré techniquement.
[~] Décision produit	Point ouvert qui doit être tranché avant de pouvoir estimer/planifier.

Architecture actée : deux interfaces, deux responsabilités

Confirmé cette semaine : la configuration de la fidélité (activation + programmes) vit dans **core-app** (back-office déjà existant du compte BY+). shop-core-api obtient son propre module **Fidelity**, mais avec un rôle différent : savoir si la fidélité est active pour exécuter la vente, pas la configurer.

Interface	Rôle	Mécanisme
core-app (+ core-api)	Activer la fidélité pour le compte. Configurer les programmes de points, codes promo, promotions, bonus.	core-api héberge un nouveau module proxy `Fidelity` qui relaie l'admin connecté vers fidelity-api (routes déjà en `central-auth`, donc directement appelables une fois le domaine du tenant connu).
shop-core-api (+ shop-core-app / POS)	Savoir si la fidélité est active pour ce compte. Exécuter la vente avec fidélité (recherche client, points, codes promo) au checkout.	Le statut d'activation est lu depuis core-api en réutilisant le pattern M2M déjà en place (`RemoteLicenseService`, pk/sk, cache, retry). La vente relaie le Bearer du caissier directement vers fidelity-api.

Vue d'ensemble — statut par cas d'usage

#	Cas d'usage	Statut
1	Activer la fidélité pour mon compte	🔧 **Fondation posée**
2	Configurer un programme de points	❌ **Non couvert**
3	Configurer un code promo	❌ **Non couvert**

4	Configurer une promotion	❌ **Non couvert**
5	Savoir si la fidélité est active côté boutique	✅ **Couvert (pattern confirmé faisable)**
6	Appliquer une promotion à un produit	❌ **Non couvert**
7	Octroyer des points à un client (hors vente)	❌ **Non couvert — bloqué côté fidelity-api**
8	Rechercher un client au checkout	🔧 **Fondation posée**
9	Vente POS avec code promo / points gagnés	🔧 **Fondation posée**
10	Payer avec des points de fidélité au POS	❌ **Non couvert**

Cas d'usage détaillés

UC1 — Activer la fidélité pour mon compte

Acteur : Admin du compte BY+, sur core-app.

Scénario : L'admin ouvre l'écran "Fidélité", renseigne les informations légales demandées (IFU, RCCM, représentant, secteur/pays/ville), valide. Le système crée le tenant fidelity-api en arrière-plan et active la fidélité pour tout le compte une fois le provisioning terminé.

Statut : 🔧 **Fondation posée** — le mécanisme de provisioning (appel `register/company`, polling jusqu'au domaine, écriture sur `tenant_information`) est conçu et vérifié sur le code réel. L'écran a été déplacé de shop-core-api vers **core-app** cette semaine pour rester cohérent avec le rôle de back-office de core-app.

Précision (vérifiée) : core-app a déjà un écran d'activation de modules (`Module/Index.vue`, "Suit Apps") avec une table `module_installs` qui a même déjà un champ `domain`. L'emplacement et l'idée sont donc les bons réflexes — mais ce mécanisme est un simple toggle booléen sur un enum PHP fermé de modules *internes* (CRM, Finance, RH...), sans collecte de champs légaux ni provisioning externe. Il ne peut pas être réutilisé tel quel, seulement comme inspiration UX.

Reste à faire : écran core-app (dédié, pas un ajout dans "Suit Apps"), endpoint d'activation dans un nouveau module `Fidelity` côté core-api, migration `tenant_information` (`fidelity_domain`, `fidelity_activated_at`).

UC2 — Configurer un programme de points

Acteur : Admin/marketing, sur core-app.

Scénario : Depuis core-app, création d'une règle de points (ex. "1 point par 1000 F dépensés"), associable à une ou plusieurs boutiques.

Statut : ❌ **Non couvert** — bonne nouvelle : le CRUD complet existe déjà côté fidelity-api (`RulePointsController`), mais aucune interface ne l'utilise. Il manque un proxy core-api + un module frontend core-app entier (écrans liste/création/édition).

UC3 — Configurer un code promo

Acteur : Admin/marketing, sur core-app.

Scénario : Création d'un code promo (montant/pourcentage de remise), ciblage de clients éligibles, activation/désactivation.

Statut : ❌ **Non couvert** — même situation que UC2 : le CRUD (`CouponController`) existe côté fidelity-api, aucune interface ne l'expose encore.

UC4 — Configurer une promotion

Acteur : Admin/marketing, sur core-app.

Scénario : Création d'une promotion et de ses "cibles" (produits/catégories concernés — à confirmer avec l'équipe fidelity-api ce que recouvre exactement une "target").

Statut : ❌ **Non couvert** — CRUD existant (`PromotionController`, `TargetController`) côté fidelity-api, aucune interface. **Décidé :** mapping produit ↔ target via une table de correspondance côté shop-core-api (fidelity-api garde ses propres `Target`, indépendants du

catalogue shop, pour rester capable de fonctionner avec d'autres systèmes). **Nouveau chantier :** synchronisation du catalogue produits shop-core-api → fidelity-api pour peupler ces `Target`.

UC5 — Savoir si la fidélité est active côté boutique

Acteur : Système (shop-core-api), déclenché à la connexion caissier ou à l'ouverture du POS.

Scénario : shop-core-api doit savoir si la fidélité est activée pour désactiver proprement les fonctionnalités fidélité si ce n'est pas le cas (vente normale, sans blocage).

Statut : ✅ **Couvert — pattern confirmé faisable**. Réutilisation directe de `RemoteLicenseService` (déjà utilisé pour les infos de licence : mêmes clés M2M, cache 1h et retry déjà configurés). Pas de nouvelle plomberie à inventer, juste une nouvelle méthode d'appel vers un endpoint "info tenant" de core-api qui existe déjà et sera étendu (UC1).

UC6 — Appliquer une promotion à un produit

Acteur : Gérant boutique, sur shop-core-app (fiche produit).

Scénario : Depuis la fiche produit, associer une promotion existante (créée en UC4) à ce produit.

Statut : ❌ **Non couvert** — dépend du mapping produit↔target non encore défini (cf. UC4). Aucune investigation faite côté catalogue produit shop-core-api pour ce cas précis.

UC7 — Octroyer des points à un client (hors vente)

Acteur : Caissier ou gérant, geste manuel (ex. geste commercial, correction).

Statut : ❌ **Non couvert — bloqué côté fidelity-api.** Les endpoints correspondants (`giveIssuePoints`, `redeem/process`, `point-transactions`) existent dans le code mais sont **commentés / désactivés** dans `Fidelity/routes/tenant.php`. Seule l'association client↔règle de points (éligibilité) fonctionne aujourd'hui, pas l'octroi effectif d'un solde de points.

Décidé : mis en attente pour cette itération, retiré du planning ci-dessous. Relance de l'équipe fidelity-api faite en direct par le porteur du projet (pas un chantier technique pour l'instant).

UC8 — Rechercher un client au checkout

Acteur : Caissier, sur le POS shop-core-app.

Scénario : Recherche d'un client par téléphone/email pendant l'encaissement, pour lui appliquer des avantages fidélité.

Statut : 🔧 **Fondation posée** — l'endpoint `GET get/customer/search` existe côté fidelity-api et est documenté. **Point de vigilance sécurité (aggravé, vérifié dans le code) :** cet endpoint ne filtre par rien du tout — ni tenant, ni société. Comme le client (`Customer`) est stocké en central et partagé entre tous les comptes BY+ (décision actée n°2 ci-dessous), la recherche retourne aujourd'hui des clients de **n'importe quel compte**, pas seulement ceux liés au tenant appelant. À faire corriger côté fidelity-api avant mise en production — prioritaire.

Décidé (identité client) : si le client n'est pas trouvé, pas de création depuis la caisse — il doit s'inscrire lui-même (self-register). Cohérent avec le fait que l'identité client est partagée entre tous les comptes BY+.

UC9 — Vente POS avec code promo appliqué / points gagnés

Acteur : Caissier, sur le POS.

Scénario : Panier calculé → appel `sales/prepare` (avec coupons/bonus sélectionnés) → si validation nécessaire, QR affiché ou OTP saisi → `sales/confirm` → points crédités, remise appliquée, vente conclue même si fidelity-api est indisponible (dégradation).

Statut : 🔧 **Fondation posée** — le flux complet (`sales/prepare → confirm/authorize → status`) est documenté côté fidelity-api. Le point d'ancrage exact dans shop-core-api est identifié (`Pay.php::save()`, avant la création des lignes de paiement). Reste à coder : le module `Fidelity` de shop-core-api, la colonne `fidelity_sale_id`, et le job de résilience

(jamais bloquer une vente si fidelity-api ne répond pas).

UC10 — Payer avec des points de fidélité au POS

Acteur : Caissier, sur le POS — client choisit de payer (tout ou partie) avec ses points.

Scénario : Le point de fidélité apparaît comme moyen de paiement dans la liste habituelle du POS ; sélectionné, il déclenche la validation client (OTP/QR) avant d'être accepté.

Statut : ❌ **Non couvert** — techniquement faisable : les moyens de paiement (`PaymentType`) sont déjà configurables en base côté shop-core-api (pas un enum figé), donc ajouter "Points de fidélité" comme option est trivial côté configuration. Mais le comportement associé (déclencher la validation OTP/QR au lieu de juste enregistrer un montant) doit être codé spécifiquement dans le flux de paiement — ce n'est pas juste une donnée de configuration. Pas encore cadré techniquement.

Répartition de charge par projet — ordre de livraison recommandé

#	Projet	Tâche	Bloque
1	fidelity-api	Scoper `searchCustomers` et `sales/status` par tenant (sécurité, prioritaire — cf. UC8 ; aggravé : la recherche actuelle n'a aucun filtre, pas même par tenant)	Tout le reste : sans ça, le modèle retenu (Option B) est cassé dans les faits
2	fidelity-api	Endpoint `provision/user` (ID imposé) + `society_id` sur `users`/`shop_users`	Sync Shop/User/ShopUser côté shop-core-api
3	core-api	Migration `tenant_information` + endpoint activation (UC1) + module proxy config (UC2-UC4)	core-app
4	core-app	Écran activation (UC1) + module Fidelity complet (UC2, UC3, UC4)	—
5	shop-core-api	Module Fidelity : lecture du statut d'activation (UC5) + job de backfill à l'activation (Shop/User/ShopUser existants) + sync continue	Étapes suivantes
6	shop-core-api	Sync catalogue produits → fidelity-api (peuple les `Target`, nécessaire pour UC4/UC6)	UC6
7	shop-core-api	Checkout : recherche client + `sales/prepare/confirm` (UC8, UC9)	UC10
8	shop-core-api	Points comme moyen de paiement (UC10)	—

—	—	UC7 (octroi manuel de points) retiré du planning — en attente de retour de l'équipe fidelity-api	Non bloquant pour le reste

Points de vigilance

Point	Risque
Recherche client sans aucun filtre tenant/société (fidelity-api, vérifié dans le code : `Customer::query()` sans clause de scope)	Fuite de données entre tous les comptes BY+ utilisant fidelity-api, pas seulement entre sociétés d'un même tenant — plus grave qu'un simple manque de scope société
`sales/status` sans vérification de propriété	Une boutique pourrait consulter/interférer avec la validation d'une autre boutique
Pas de clé d'idempotence serveur sur `sales/prepare`	Retry réseau = risque de double vente / double octroi de points
Module Webhook de core-api pas encore mergé sur la branche fidelity	Le pattern de retry (base pour la sync Shop/User/ShopUser) n'est pas directement disponible sans portage — prévoir un Job Laravel classique en attendant

Décisions actées (19/08/2026)

#	Décision
1	**Mapping promotions ↔ produits (UC4, UC6)** : table de correspondance côté shop-core-api, pas de référence directe produit dans fidelity-api — fidelity-api doit rester capable de fonctionner avec d'autres systèmes que shop (voire seul), il gère donc ses propres `Target`. **Nouveau chantier identifié :** synchroniser le catalogue produits shop-core-api → fidelity-api pour peupler ces `Target` (même famille que la sync Shop/User/ShopUser).
2	**Identité client partagée entre tenants** : confirmé dans le code — le modèle `Customer` de fidelity-api utilise une connexion **centrale** (pas une table par tenant), donc un client ne s'inscrit qu'une fois et est reconnu par tous les comptes BY+ utilisant la fidélité. Chaque business garde son propre programme de points (mécanisme tenant-scopé, inchangé). **Pas de création de client depuis la caisse** si introuvable — inscription uniquement par le client lui-même.
3	**Synchro ShopUser → fidelity-api** : immédiate (event/job), mais seulement si le module Fidelity est activé pour le compte. À l'activation (UC1), un job de **backfill** synchronise tous les Shop/User/ShopUser déjà existants (pas seulement les nouveaux).
4	`core-api /events/sale-confirmed` : **ne sera pas construit** — aucun consommateur CRM identifié.
5	**Résilience** : queue + job de retry automatique (infra déjà en place dans shop-core-api).
6	**Octroi manuel de points (UC7)** : mis en attente pour cette itération. Rappel : relance de l'équipe fidelity-api à faire pour savoir si ces endpoints seront finis/réactivés.