Fidelity-api — deux cas d'intégration, tâches à faire

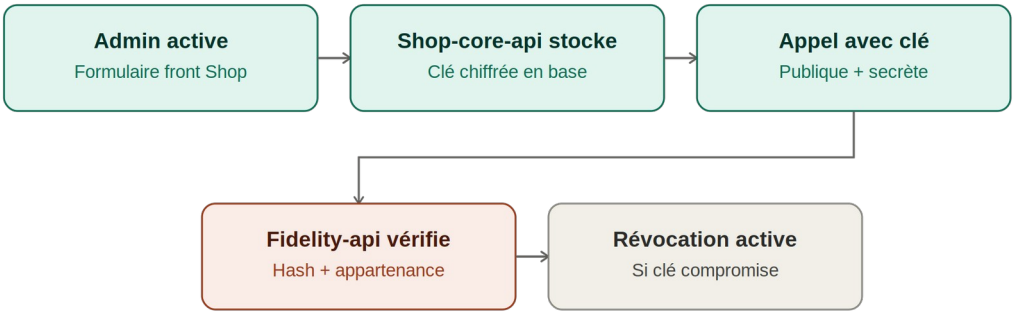
Application Shop (backend) et application tierce avec widget — droits laissés de côté pour l'instant

0. Correspondance avec les tables existantes

Le schéma de base existant utilise déjà les noms réels ci-dessous. On les reprend tels quels plutôt que d'utiliser des noms génériques.

☐	Tâche	Détail
☐	Company	Existe déjà — c'est l'entité à laquelle rattacher la paire de clés (via society_id, déjà utilisé par Shops, promotions, coupons, rule_bonus, rule_points)
☐	Users / shop_users	Existent déjà — copie locale des caissiers côté fidelity-api, utilisée pour vérifier l'appartenance boutique/caissier
☐	api_keys (à créer)	Nouvelle table, liée à Company — porte la clé publique et le hash de la clé secrète

1. Cas 1 — Application Shop (backend-à-backend)



La clé publique/secrète est longue durée et révocable activement. Elle authentifie chaque appel serveur-à-serveur entre shop-core-api et fidelity-api.

1.1 Côté fidelity-api

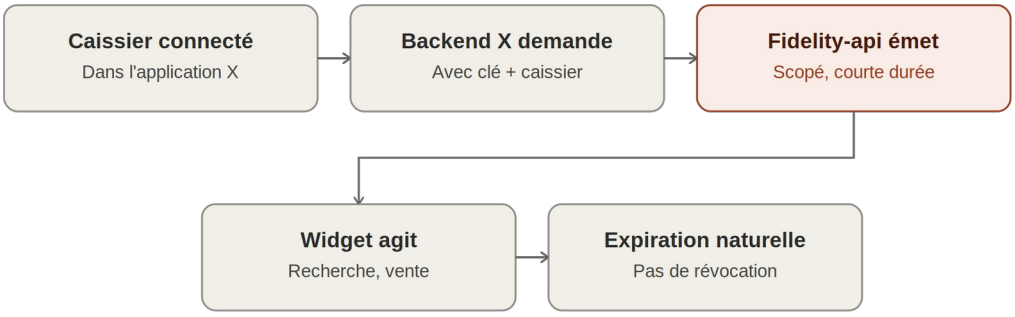
☐	Tâche	Détail
☐	Ajouter shop_tenant_id sur Company	Colonne unique — fait la correspondance avec le tenant shop-core-api d'origine, absente aujourd'hui
☐	Créer la table api_keys	company_id (FK vers Company), public_key, secret_hash, revoked_at, timestamps
☐	Endpoint d'activation	Reçoit shop_tenant_id + nom du tenant fidélité, crée la Company si besoin, génère et renvoie la paire de clés une seule fois
☐	Endpoint de statut de clé	Vérifie que la clé n'est pas révoquée
☐	Endpoint de révocation	Marque la clé comme révoquée pour une Company donnée
☐	Endpoint de rotation des clés	Génère une nouvelle paire, révoque l'ancienne
☐	Middleware de vérification de clés	Vérifie clé publique + secrète sur chaque requête entrante (hors activation)
☐	Vérification d'appartenance via Users/shop_users	Déjà en place pour sales/status — à connecter systématiquement à ce nouveau middleware pour toutes les routes recevant un user_id
☐	Retirer la dépendance à CENTRAL_URL	Plus d'introspection centrale via core-api

☐	Tâche	Détail
☐	Supprimer AuthenticateWithCentral et les routes central-auth	Ancienne étape D, abandonnée

1.2 Côté shop-api

☐	Tâche	Détail
☐	Créer la table fidelity_configs	public_key, secret_key chiffrée, activated_at, status_checked_at, dernier résultat de statut mis en cache
☐	FidelityKeyService — activation	Appelle l'endpoint d'activation avec l'ID de tenant + nom du tenant fidélité, stocke la paire reçue
☐	FidelityKeyService — vérification de statut	Appelle l'endpoint de statut, cache court (quelques dizaines de secondes)
☐	Invalidation immédiate du cache sur 401/403	Rafraîchir le statut tout de suite plutôt que d'attendre l'expiration du cache
☐	FidelityService — en-têtes de clé	Clé publique + secrète sur tous les appels fonctionnels (recherche client, ventes, sync)
☐	FidelityService — transmission user_id/shop_id	En clair dans les requêtes, après authentification du user par shop-core-api lui-même
☐	Retirer l'appel à REMOTE_LICENSE_API_URL dans isActivated()	Remplacé par l'appel direct au statut de clé chez fidelity-api
☐	Route et contrôleur admin d'activation	Réservé à un admin authentifié sur le tenant
☐	Champ du formulaire : nom du tenant fidélité	Seul champ métier nécessaire
☐	Empêcher une double activation	Vérifier qu'aucune paire de clés n'existe déjà avant d'en générer une nouvelle

2. Cas 2 — Application X avec widget (caissier)



Le widget tourne dans le navigateur du caissier de l'application X : il ne peut jamais recevoir la clé secrète de X. Le backend de X échange sa clé contre un token court et scopé, que le widget utilise pour agir.

2.1 Côté fidelity-api

☐	Tâche	Détail
☐	Endpoint d'émission de token widget	Reçoit la clé de l'application (publique+secrète) + l'identité du caissier et de la boutique ; renvoie un token court
☐	Choisir le mécanisme de token	Recommandé : JWT signé par fidelity-api, vérifiable sans état — pas de table de sessions à gérer

☐	Tâche	Détail
☐	Middleware de vérification des routes widget	Accepte le token (recherche client, vente) — distinct du middleware clé publique/secrète de l'étape 1
☐	Durée d'expiration du token	Courte — de l'ordre d'une session de travail caissier
☐	Aucune révocation active à prévoir	Le token expire naturellement ; décision déjà actée

2.2 Le widget (fourni par fidelity, embarqué chez X)

☐	Tâche	Détail
☐	Utiliser le token pour ses appels	Recherche client, vente avec points
☐	Gérer l'expiration proprement	Signaler à l'application hôte qu'un nouveau token est nécessaire, sans jamais gérer lui-même la clé

3. Ordre recommandé

Le cas 1 est la fondation : la table api_keys et le mécanisme de vérification de clé doivent exister avant le cas 2, puisque l'émission d'un token widget dépend d'une clé d'application déjà valide. Construire et stabiliser le cas 1 en premier, puis ajouter l'émission de token pour le cas 2 par-dessus.

4. Hors scope pour l'instant

- La gestion des droits par type d'accès (scopes sur les clés, restrictions du token widget) — traitée dans un document séparé, à reprendre plus tard.