BY+ · CONSOLE NETBYUS · MISE EN PLACE

La console NetByUs : plan de mise en place

Ce que la console doit faire, ce que core-api sait déjà faire (applications, licences, offres, paiement, déploiement, support), ce qu'on en reprend dans la nouvelle API, et l'ordre de construction.

S'appuie sur « L'architecture de BY+ », le « Catalogue des modules », « Le socle BY+ : guide de mise en place », et la relecture du code de core-api.
Code relu : `byplus/api` (branche `new-develop`) ; core-api : `develop`, `by-plus-access-develop`, `Feature/Add/HelperDeskWebhookTicketId` (la plus récente avec la vente de licences), `by-plus-shop-mini-staging`.
L'authentification de la console est branchée après les lots 1 à 3 : connexion jumelée LDAP ou compte du client NetByUs, double authentification obligatoire (chapitre 14).
Octobre 2026 · version à valider.

1 Ce qu'on construit, et quand c'est terminé

> **Objectif.** Construire la console NetByUs dans la nouvelle API (`platform/`, base centrale), pour que **l'abonnement d'un client décide des modules qu'il a actifs** : NetByUs crée le client, lui attribue un abonnement, et BY+ active automatiquement les bons modules. On reprend ce que core-api sait déjà faire, sans recopier ses défauts.

Terminé quand (phase 1) : on crée un client, on lui attribue un abonnement avec POS et Entrepôt pour 10 utilisateurs ; Catalogue et Contacts s'activent seuls (modules requis) ; un utilisateur du client se connecte sur `{client}.byplus.io`, ne voit que ces modules, et ne peut ni créer un 11e utilisateur ni atteindre un module non souscrit. Ces actions passent par les **routes de l'API de la console** (`/api/v1/console/…`) ; leur authentification (LDAP ou comptes du client NetByUs, chapitre 14) sera branchée plus tard.

Phase	Contenu	Terminé quand
1. Console minimale	Clients et déploiement d'une base, catalogue, abonnements, réconciliateur (activation), journal de la console, routes de l'API console	Le critère ci-dessus
2. Vente	Calculateur de prix, commandes, paiement FedaPay / KkiaPay, factures, codes promo, cycle de vie (essai, grâce, suspension), guichets de « Mon abonnement »	Un client paie en ligne et son abonnement s'active seul
3. Exploitation et ouverture	Migrations et sauvegardes suivies, archivage, support et accès support, clés de licence signées, annonces, reprise des clients actuels	Les clients actuels tournent sur la nouvelle API avec un abonnement
Ensuite	Accès à la console : comptes des équipes, connexion jumelée LDAP ou `netbyus`, double authentification obligatoire (chapitre 14), écrans	Un agent inscrit se connecte avec double authentification

À retenir

- **La console ne part pas de zéro.** core-api sait déjà vendre : catalogue d'offres, devis, commande, paiement FedaPay et KkiaPay vérifié par webhook, facture PDF, codes promo, clés API, webhooks, tickets. Mais ce code vend des **licences d'applications externes** (Shop, ECU, Influencer) **à des contacts** du tenant `netbyus`, pas des **abonnements BY+ à des clients**.
- **Le lien central n'existe pas.** Dans core-api, rien ne relie ce qu'un client paie et ce qu'il a d'actif : un client s'active lui-même ses modules (`POST install/modules`), sans licence ni quota.
- **La nouvelle API a déjà la moitié de l'activation** : registre des modules, exigences résolues, table `tenant_modules`, cache, middleware `module:{clé}`. Il manque l'abonnement en amont et le réconciliateur entre les deux.
- **On construit à neuf dans `platform/`**, avec des tables propres en base centrale, en **recopiant les algorithmes éprouvés** (devis, passerelles de paiement, facture PDF, promotions) et en corrigeant leurs défauts (montants en décimal, signature à clé partagée…).

2 Ce que fait la console

La console est le back-office de NetByUs : les écrans où les équipes gèrent **tous** les clients de BY+, comme l'espace d'un opérateur téléphonique. Elle vit sur le domaine central et la base centrale ; elle ne voit jamais les données métier des clients.

Rôle	Ce que ça couvre	Phase

Déployer	Créer un client : sa base, son sous-domaine, son premier administrateur. Suivre les migrations et les sauvegardes de chaque base. Archiver.	1, puis 3
Vendre	Le catalogue (modules publiés, prix, options, promotions) et le calcul du prix d'une offre composée par le client.	1, puis 2
Suivre	L'abonnement de chaque client : ses modules, son nombre d'utilisateurs, ses dates, ses commandes, factures et paiements.	1, puis 2
Activer	Mettre chez chaque client exactement les modules que son abonnement prévoit, avec leurs modules requis et leurs ponts.	1
Assister	Traiter les tickets ; entrer chez un client seulement avec son accord, pour un temps limité.	3

Les trois règles qui font tenir l'ensemble

1. **L'abonnement est la source de vérité.** Il dit ce que le client doit avoir : ses modules choisis, son nombre d'utilisateurs, ses dates. Personne n'active un module à la main : on modifie l'abonnement, le reste suit.
2. **Un réconciliateur applique.** Il calcule l'état voulu (modules choisis + modules requis + ponts, quotas), le compare à l'état réel (`tenant_modules`) et corrige l'écart. Il peut être relancé sans risque ; un passage de nuit rattrape les dérives.
3. **Une seule sortie vers les clients.** L'état est publié dans le cache (Redis) avec un numéro de version. Le BY+ du client ne lit que cela pour ses menus et ses routes : il n'interroge jamais les tables de la console.

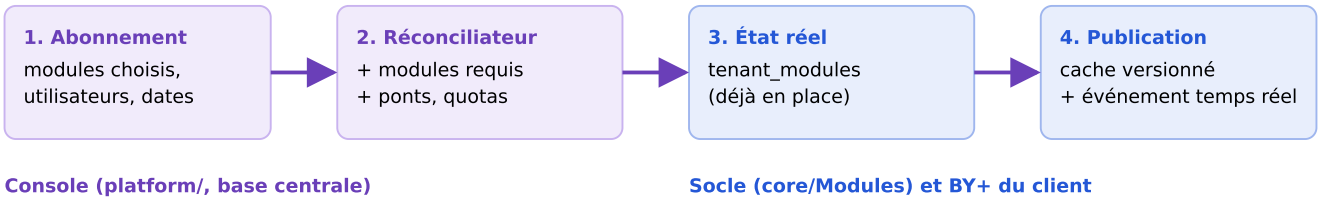


Figure 1 : de l'abonnement aux menus du client. Les étapes 3 et 4 existent déjà en partie dans la nouvelle API.

3 Le point de départ : la nouvelle API

Élément	Ce qu'il apporte à la console	État
Noyau : `Context`, `Actor` (user, system, api_key), `BricksServiceProvider`, `RouteLoader`, `CentralController`	Le cadre des routes centrales et l'acteur de chaque action (le journal de la console en a besoin). Un type d'acteur « agent NetByUs » viendra avec l'accès.	en place
Tenancy : `tenants`, `domains`, base par client, routes client sous `/api/v1`	La création d'une base existe (Stancl : `CreateDatabase` + `MigrateDatabase` à la création d'un client).	en place
Registre des modules : manifestes, `requires`, `sellable`, `ModuleRegistry::resolve()`, `dependentsOf()`	Le résolveur des exigences existe déjà : le réconciliateur l'appelle, sans recopier `ModuleDependencyResolver` de core-api.	en place
`tenant_modules` (base centrale) et `TenantModules` : activer, désactiver, cache par client	L'état réel. Devient écrit **uniquement** par le réconciliateur ; les commandes `byplus:module:enable/disable` restent pour le dépannage.	en place
Middleware `module:{clé}` posé automatiquement sur les routes des modules	Ce que core-api n'a jamais branché : un module non souscrit renvoie 403.	en place
Auth : `core_users` (base du client), Sanctum	Le premier administrateur du client est créé là ; le quota d'utilisateurs s'y applique.	en cours
Docker : PostgreSQL, Redis, MinIO, Mailpit	Redis pour l'état publié, MinIO pour factures PDF et sauvegardes.	en place
`platform/`	Le dossier de la console.	vide

Ce qui manque pour relier abonnement et modules

- **Le cache versionné** : aujourd'hui, la liste des modules d'un client est oubliée puis relue ; il faut un numéro de version incrémenté à chaque changement, publié avec l'état (prévu au guide du socle, § 8.2).
- **L'origine d'une activation** dans `tenant_modules` : choisi, requis, pont, extension. Elle permet de retirer proprement un module requis quand plus rien ne le demande.
- **Les capacités et quotas publiés** (nombre d'utilisateurs, stockage) : lus par Auth et Files au moment de créer un utilisateur ou un fichier.

4 Ce que core-api sait déjà faire

Relu dans le code. « Branche » indique où se trouve la version la plus complète. Sauf mention, ces tables sont dans la **base du tenant** `netbyus` (migrations `tenant`), pas dans la base centrale.

Domaine	Où (branche · module)	Tables	Services et comportement
Gestion des applications BY+	`develop` · Module (base centrale)	`suit_apps`, `modules`, `module_installs` (tenant_id, enabled, domain)	`SuitAppsEnum` / `SuitModulesEnum` codés en dur ; `installModules()` bascule installer/supprimer, sans licence ; `getSeeders()` jamais appelé
Applications externes	`by-plus-access-develop` · Application	`applications`, `application_modules`, `application_packs`, `application_module_dependencies`, `application_permissions`, `application_configs`, `storage_offers`, `storage_subscriptions`	Catalogue des applications vendues (Shop, ECU, Influencer), packs, permissions ; `ModuleDependencyResolver` ; options de stockage souscrites
Licences	id. · License	`licenses` (propriétaire polymorphe : un **contact**), `license_types`, `license_durations`, `license_activations`, `license_modules`, `license_offer` (instantané), `license_usages`, `contact_applications` (type de déploiement `saas` / `dedicated`, `base_url`, secret de webhook)	Statuts : `pending`, `active`, `expired`, `revoked`, `suspended` ; `validate`, `activate`, `deactivate`, `check-option` (quotas) ; jeton de licence `LicenseTokenService` signé en HMAC-SHA256
Offres et prix	`Feature/Add/HelperDesk…` · License	`offer_types`, `service_offers`, `service_offer_prices` (frequency, server partagé / dédié)	`CompositeOfferService` (offre composée de plusieurs offres) ; `LicenseCheckoutService::quote()`
Commande et paiement	id. · License	`checkout_orders` (montants `decimal(10,2)`), `payment_aggregator_configs`	`FedaPayService`, `KkiaPayService` ; webhooks signés (`X-FEDAPAY-SIGNATURE`, `X-KKIAPAY-SECRET`) ; contrôle du montant payé ; facture PDF ; envoi de la licence et de la facture par mail et WhatsApp
Promotions	id. · PromoCode	`promo_codes`, `promo_code_usages`	`PromoCodeService::resolve()` (remise, limites d'usage), rattaché aux influenceurs
Déploiement des clients	`develop` · Central (base centrale)	`tenants`, `domains`, `tenant_information` (raison sociale, IFU, RCCM, pays, devise, logo, responsable), `legal_pages`	`CreateANewTenantJob` : un seul job (172 lignes) : `cache:clear`, client, domaine, base, une quinzaine de seeders, société, agence ; commandes `Migrate`, `Rollback`, `MigrateFresh`, `Seed`, `SaveDataBase`, `DropTenant` ; inscription publique `POST inscription`
Sauvegardes et stockage	`by-plus-shop-mini-staging` · TenantBackup (base centrale)	`backups`, `storages`	Sauvegardes par client, place utilisée (job de recalcul), demandes de sauvegarde et de stockage payantes, webhooks d'approbation
Support	`by-plus-access-develop` · Helpdesk	`helpdesk_tickets`, `helpdesk_ticket_responses`, catégories, affectations, lectures, appels (`helpdesk_calls` , LiveKit)	Source polymorphe du ticket (contrat `HelpdeskTicketSource`) ; notifications Mattermost ; temps de première réponse

Intégrations	id. · ExternalApplications, Webhook	`external_applications` (pk / sk, scopes, usages, configuration des points d'accès), `contact_application_api_keys`, `webhook_subscriptions`	`ApiKeyService` ; `WebhookDispatcherService` envoie sans garder de trace
Distributeurs	id. · Influencer, License	`application_influencers`	`DistributorLicenseController` : vente par des influenceurs. Hors périmètre de la console BY+

Deux systèmes qui ne se parlent pas

	Système A : modules des clients BY+	Système B : licences des applications externes
Base	Base centrale	Base du tenant `netbyus`
Le client est	Un tenant	Un contact (relation polymorphe)
Ce qui est vendu	Rien : pas de prix, pas de licence	Offres par application, mensuelles ou annuelles, serveur partagé ou dédié
Activation	Le client s'active lui-même (`POST install/modules`)	Licence activée après paiement, jeton remis à l'application
Contrôle	`module.access` posé sur aucune route	Clés pk / sk, scopes, signature des webhooks

5 Les constats qui comptent pour la console

Constat	Où	Conséquence pour la console
Montants en décimal et calculs en `float`, arrondis à 2 décimales	`checkout_orders.amount`, `quote()`, `PromoCodeService`, facture PDF	Montants en **entiers** (F CFA, sans décimales) ; prix **figés** à la commande
Jeton de licence signé avec une clé partagée (HMAC)	`LicenseTokenService`	Signature **asymétrique** : une application ou un serveur dédié vérifie la licence hors ligne sans connaître le secret
Pas d'état « essai » ni « grâce » ; aucune tâche d'expiration planifiée	`LicenseStatus`, `routes/console.php`	Cycle de vie complet et tâche quotidienne à écrire
Le titulaire d'une licence est un contact, pas un client	`License::contact()` (morphTo)	L'abonnement est rattaché au **tenant** (un abonnement par client)
Webhooks de paiement vérifiés mais non dédoublonnés	`LicenseCheckoutController`	Journal des événements reçus, unique par référence : un webhook rejoué ne paie pas deux fois
Création d'un client en un seul job, non rejouable, avec `cache:clear` global	`CreateANewTenantJob`	Déploiement en **étapes rejouables**, suivi de l'état, aucun vidage global
Clés de passerelle de paiement en clair en base	`payment_aggregator_configs`	Colonnes **chiffrées** (cast `encrypted`)
Envoi de webhook sans trace ni nouvelle tentative suivie	`WebhookDispatcherService`	Journal des livraisons, nouvelles tentatives
Secrets versionnés dans Git (ex. clé Firebase dans access-api)	Historique des dépôts	Aucun secret dans le code ; clés de la console hors du dossier web

Routes centrales ouvertes (pages légales modifiables sans authentification, inscription publique)	`Central/routes/api.php`	Aucune route d'écriture de la console sans protection ; l'accès sera défini plus tard

6 Où va la console dans la nouvelle API

La console a la **même forme que le socle** : un dossier par domaine, chacun avec sa fiche, ses routes, son guichet (`Public/`) et ses tables. Elle est chargée par le même noyau, mais **uniquement sur les domaines centraux** et sur la **base centrale**.

```
platform/                          LA CONSOLE · namespace Platform\ · base centrale
├── Tenants/       ○ phase 1       clients, domaines, profil, contacts, déploiement d'une base
├── Catalog/       ○ phase 1       modules publiés, prix, options, promotions
├── Subscriptions/ ○ phase 1       abonnement de chaque client, historique
├── Activation/    ○ phase 1       réconciliateur, publication de l'état
├── Audit/         ○ phase 1       journal des actions de la console
├── Orders/        · phase 2       commandes, devis figés
├── Billing/       · phase 2       factures, paiements, passerelles, webhooks
├── Operations/    · phase 3       migrations, sauvegardes, archivage
├── Support/       · phase 3       tickets BY+, accès support accordé par le client
├── Licensing/     · phase 3       clés de licence signées (serveurs dédiés, applications)
├── Announcements/ · phase 3       annonces aux clients
└── Staff/         · ensuite       agents NetByUs, rôles, connexion jumelée, double authentification
```

Règle	Pourquoi
Chaque domaine a une fiche `platform.php` (clé, nom, préfixe de tables) ; le noyau charge `routes/api.php` sous `/api/v1/console/{clé}`	Même chargeur que les briques (`RouteLoader::central`) ; pas de route client
Toutes les tables sont en base centrale (connexion centrale fixée sur chaque modèle)	La console ne lit jamais une base client, sauf l'accès support accordé (phase 3)
La console écrit l'état des clients **par le guichet du registre** (`Core\Modules\Public`), jamais directement dans les tables du socle	Le socle reste maître de ses tables ; Deptrac le vérifie
Le socle n'importe jamais `Platform\`	Un client lit l'état publié, pas la console
Chaque action est une Action (une classe = une action) appelée par une route de l'API console, sous `/api/v1/console/{clé}` , derrière le garde `console`	Les routes se construisent maintenant ; seul le garde change quand l'accès sera choisi (chapitre 14)
Montants en entiers, dates en UTC, identifiant du client = `tenants.id` (chaîne lisible, ex. `sdp`)	Pas de calcul flottant, pas de colonne de clé en double

> **Noms des tables.** La base centrale de la nouvelle API est vide : on ne reprend pas les noms de core-api (`licenses`, `checkout_orders`…), qui désignent d'autres choses (licences d'applications vendues à des contacts). Chaque domaine de la console a son préfixe : `tenant_`, `catalog_`, `sub_`, `act_`, `ord_`, `bill_`, `ops_`, `sup_`, `lic_`, `ann_`, `aud_`, `staff_`. Exceptions : `tenants` et `domains` (paquet Stancl) et `tenant_modules` (registre du socle), qui gardent leur nom.

7 Le cœur : de l'abonnement aux modules actifs

Ce que fait le réconciliateur, pour un client

1. **Lire l'abonnement** : modules choisis (`sub_subscription_modules` non retirés), état de l'abonnement, quotas.
2. **Calculer l'état voulu** : si l'abonnement ne donne pas accès (suspendu, résilié), rien ; sinon les modules choisis, plus leurs exigences (`ModuleRegistry::resolve()`), plus les ponts dont les deux modules sont présents, plus les extensions du client.
3. **Comparer** à `tenant_modules` et calculer l'écart (à activer, à retirer).
4. **Préparer la base du client** pour chaque module à activer : droits, rôles et données de départ du manifeste (rejouables).
5. **Écrire l'état réel en dernier** dans `tenant_modules` (avec l'origine : choisi, requis, pont, extension) : un module n'est jamais marqué actif sans ses données.
6. **Publier** : incrémenter la version, écrire l'état et les quotas dans le cache, émettre l'événement « état modifié » (menus rechargés).
7. **Tracer** l'exécution dans `act_runs` (déclencheur, état voulu, écart, résultat).

Déclencheur	Exemple
Changement d'abonnement	Module ajouté, retiré, quota modifié, abonnement payé, suspendu ou rétabli
Nouvelle version du code	Un module gagne une exigence ou un pont : passage sur tous les clients après déploiement
Passage de nuit	Rattrape toute dérive (écriture manuelle, échec réseau)
Demande d'un agent	`POST /api/v1/console/tenants/{id}/reconcile`

Ce que lit le BY+ du client

Clé publiée	Lue par
`byplus:tenant:{id}:version`	Le front (rechargement des modules si la version change)
`byplus:tenant:{id}:modules`	Middleware `module:{clé}`, écouteurs, tâches planifiées, `/me/bootstrap`
`byplus:tenant:{id}:quotas`	Auth (nombre d'utilisateurs), Files (stockage)
`byplus:tenant:{id}:access`	Tenancy : `active`, `grace` (bandeau), `suspended` (connexion bloquée sauf abonnement et export)

Si une clé manque (cache vidé), le socle la reconstruit depuis `tenant_modules` et l'abonnement, puis la republie : le cache n'est jamais la seule copie.

8 L'offre d'un client et son prix

L'offre d'un client est **l'ensemble des modules qu'il choisit**, enregistré dans son abonnement. Il n'y a pas d'offres écrites dans le code. Le calcul du prix est fait côté serveur, par **un seul calculateur**, le même pour la page de tarifs, « Mon abonnement », la console et le paiement.

Élément du prix	Proposition	Statut
Prix d'un module	Montant par mois (et par an, avec remise) pour chaque module vendable ; les modules de base (`sellable: false`) sont gratuits	à confirmer
Utilisateurs	Prix par utilisateur et par mois, au-delà d'un nombre inclus	à confirmer
Formules	Facultatives : une formule (Gratuit, Standard, Entreprise) peut fixer un plafond de modules, un nombre d'utilisateurs inclus et des remises. Ce sont des données, pas du code	à trancher
Options	Stockage supplémentaire, porte connectée, installation… quantité × prix unitaire (reprend `storage_offers`)	Proposé
Promotions	Codes promo : pourcentage ou montant, limites d'usage par client (reprend `PromoCodeService` , sans le lien influenceur)	Proposé
Grille de prix	Versionnée (date d'effet) : une nouvelle grille ne change pas le prix d'un abonnement en cours	Proposé
Prorata	Un module ajouté en cours de période est facturé au prorata jusqu'à l'échéance	Proposé

Le calculateur isole ces règles : si le modèle commercial change après l'atelier, seul le calculateur change.

Le cycle de vie d'un abonnement

État	Ce que voit le client	Comment on en sort
Essai	Accès complet	Paiement : Actif. Fin d'essai sans paiement : Suspendu
Actif	Accès complet	Échéance sans renouvellement : En grâce
En grâce	Accès complet, bandeau « renouvelez »	Paiement : Actif. Fin de la grâce : Suspendu
Suspendu	Connexion bloquée, sauf « Mon abonnement » et export	Paiement : Actif. Délai échu : Résilié
Résilié	Aucun accès ; données conservées	Fin de la conservation : Archivé
Archivé	Base sauvegardée puis supprimée	État final, sous double validation

Une tâche quotidienne fait avancer ces états et déclenche le réconciliateur. Les durées (essai, grâce, conservation) sont à fixer en atelier.

9 Le déploiement d'un client

Remplace `CreateANewTenantJob` . Chaque étape est **rejouable** et son résultat est noté : en cas d'échec, on relance à partir de l'étape en échec.

#	Étape	Ce qui se passe
1	Enregistrer le client	Ligne `tenants` (état `provisioning`), profil (raison sociale, pays, devise), contacts
2	Créer la base	Base PostgreSQL du client (Stancl, qui note son nom dans `tenants.data`)

3	Jouer les migrations	Socle et tous les modules ; résultat noté dans `tenant_provisioning_steps`
4	Préparer le socle	Données de départ du socle (pays, devises, unités sont des fichiers livrés avec le code : rien à copier) ; première société
5	Créer le premier administrateur	Compte dans `core_users` de la base du client ; lien d'invitation envoyé par mail
6	Rattacher le domaine	`{client}.byplus.io` dans `domains` (marqué principal)
7	Attribuer l'abonnement	Essai ou abonnement donné par NetByUs ; le réconciliateur active les modules
8	Ouvrir	État `ready` ; le domaine répond

Exploitation des bases (phase 3)

Besoin	Repris de	Ajout
Migrer toutes les bases	`tenants:migrate` (Stancl, déjà dans la nouvelle API)	Suivi par client (`ops_migration_runs`) ; reprise des clients en échec
Sauvegarder	`SaveDataBase` (core-api), TenantBackup (branche shop)	Sauvegardes chiffrées dans MinIO, suivies (`ops_backups`), restauration testée
Place utilisée	`RecalculateTenantStorageUsageJob` (branche shop)	Comparée au quota publié
Archiver	`DropTenant`	Sauvegarde vérifiée avant suppression ; double validation
Serveur dédié	`contact_applications.deployment_type` (`saas` / `dedicated`)	Type de déploiement sur le client ; licence signée vérifiable hors ligne (domaine Licensing)

10 Les tables de la console

Toutes en base centrale. Montants en entiers (F CFA), `created_at` / `updated_at` partout, `tenant_id` = chaîne (`tenants.id`). Colonnes principales seulement : à affiner au moment de coder chaque domaine. existe = déjà dans la nouvelle API.

Domaine	Table	Colonnes principales	Repris de
Tenants	`tenants` existe	id (chaîne), data, + status (provisioning, ready, archived), + deployment (saas, dedicated)	Stancl ; colonnes déclarées dans `getCustomColumns()`
	`domains` existe	domain (unique), tenant_id, + is_primary	Stancl
	`tenant_profiles`	tenant_id, raison sociale, IFU, RCCM, pays, devise, adresse, logo	`tenant_information`
	`tenant_contacts`	tenant_id, nom, email, téléphone, rôle (facturation, technique, décideur)	colonnes `responsable_*`
	`tenant_provisioning_steps`	tenant_id, étape, statut, erreur, tentatives, fin	—
Catalog	`catalog_modules`	module_key, nom affiché, catégorie, description courte, publié, ordre, éligible à l'essai	`modules` / `suit_apps` ; exigences lues dans les manifestes
	`catalog_price_lists`	version, devise, effective_from, effective_to, publiée	—
	`catalog_prices`	price_list_id, cible (module, utilisateur, option, formule), cible_key, fréquence, montant	`service_offer_prices`
	`catalog_options`	clé, nom, unité, récurrente, module_key (si liée)	`storage_offers`
	`catalog_promotions` · `_usages`	code, type, valeur, limites, dates · tenant_id, remise appliquée	`promo_codes` · `promo_code_usages`
Subscriptions	`sub_subscriptions`	tenant_id (unique), status, frequency, price_list_id, starts_at, ends_at, trial_ends_at, grace_ends_at, users_limit, auto_renew	`licenses` (statuts)
	`sub_subscription_modules`	subscription_id, module_key, prix figé, added_at, removal_at, removed_at	`license_modules`
	`sub_subscription_options`	subscription_id, option_key, quantité, prix figé	`storage_subscriptions`
	`sub_events`	subscription_id, type, avant (json), après (json), acteur	événements non historisés
Activation	`tenant_modules` existe	tenant_id, module, activated_at, + origin (choisi, requis, pont, extension)	`module_installs`
	`act_runs`	tenant_id, déclencheur, état voulu, écart, statut, erreur, début, fin	—
Orders	`ord_orders`	tenant_id, statut, devis figé (json), sous-total, remise, total, source (console, client, page publique), expires_at	`checkout_orders`
	`ord_order_lines`	order_id, type, cible, quantité, prix unitaire, montant, période	—

Billing	`bill_invoices` · `_lines`	numéro (unique), tenant_id, order_id, période, totaux, état, échéance, fichier PDF	`LicenseInvoicePdfService`
	`bill_payments`	invoice_id, passerelle, référence (unique), montant, état, reçu le	colonnes de paiement de `checkout_orders`
	`bill_gateway_events`	passerelle, référence d'événement (unique), type, contenu, traité le	—
	`bill_gateway_configs`	passerelle, active, clés (chiffrées), secret de webhook, sandbox	`payment_aggregator_configs`
Audit	`aud_logs`	acteur (type, id), action, cible, tenant_id, avant, après, ip	—
Operations	`ops_migration_runs` · `_tenants`	version, lancé par, début, fin, statut · tenant_id, statut, erreur	`Migrate`
	`ops_backups`	tenant_id, emplacement, taille, statut, fait le, vérifié le	TenantBackup
Support	`sup_tickets` · `sup_messages`	tenant_id, module_key, auteur, priorité, statut, première réponse · contenu, interne, pièces jointes	Helpdesk
	`sup_access_grants` · `_sessions`	tenant_id, accordé par, ticket, début, fin, révoqué · agent, début, fin, ip	—
Licensing	`lic_signing_keys`	identifiant, clé publique, clé privée (chiffrée), active, rotation	`LicenseTokenService`
Announcements	`ann_announcements` · `ann_reads`	titre, corps, type, cible, début, fin · tenant_id, utilisateur, lu le	—
Staff	`staff_users`	email, nom, fournisseur (ldap, netbyus, local), identifiant externe, actif, secret TOTP (chiffré), codes de récupération (hachés), ajouté par, dernière connexion	—
	`staff_roles` · `staff_user_roles`	clé (super_admin, commercial, support, finance, ops), nom · agent, rôle	—

Au total 38 tables. Phase 1 : Tenants, Catalog (modules), Subscriptions (sans les options), Activation, Audit, soit 12 tables dont 3 existent déjà. Les 3 tables Staff viennent avec l'accès à la console (lot 10, chapitre 14).

11 Ce qu'on reprend de core-api, et comment

« Recopier » : reprendre l'algorithme dans la nouvelle API en l'adaptant. « Concept » : reprendre l'idée, réécrire. « Non » : ne pas reprendre.

Existant	Cible	Mode	Changement
`SuitAppsEnum`, `SuitModulesEnum`	manifestes des modules	non	Le catalogue vient des manifestes ; `catalog_modules` n'en garde que l'affichage commercial
`ModuleController::installModules()`	—	non	Plus d'auto-activation : on passe par l'abonnement
`ModuleDependencyResolver`	`ModuleRegistry::resolve()`	déjà fait	Le registre résout déjà les exigences et détecte les cycles
`LicenseStatus`, `LicenseService` (validate, activate, check-option)	Subscriptions	concept	Ajout essai et grâce ; titulaire = tenant ; quotas publiés dans le cache
`license_offer` (instantané de l'offre)	prix figés	concept	Prix figés par ligne (module, utilisateurs, option) au lieu d'un instantané json
`LicenseCheckoutService::quote()`, `CompositeOfferService`	calculateur (Catalog)	recopier	Entiers, prix par module et par utilisateur, prorata, grille versionnée
`checkout_orders`, `initiateSubscription()`	Orders	recopier	Lignes de commande, devis figé, client déduit du domaine
`FedaPayService`, `KkiaPayService`, webhooks	Billing	recopier	Dédoublonnage (`bill_gateway_events`), montants entiers, on ne croit jamais le navigateur
`payment_aggregator_configs`	`bill_gateway_configs`	recopier	Clés chiffrées
`LicenseInvoicePdfService`, mails et WhatsApp de facture	Billing	recopier	Numérotation continue, PDF dans MinIO, envoi par la brique Notifications
`PromoCodeService`	Catalog	recopier	Sans lien influenceur ; limite par client
`CreateANewTenantJob`	Tenants (déploiement)	concept	Étapes rejouables et suivies ; plus de `cache:clear` ni de seeders métier
`tenant_information`	`tenant_profiles`	recopier	Colonnes reprises ; responsable → `tenant_contacts`
`Migrate`, `SaveDataBase`, `DropTenant`, TenantBackup	Operations	concept	Suivi par client, sauvegardes dans MinIO, double validation
Helpdesk (tickets, réponses, appels)	Support	concept	Source = tenant et module ; accès support accordé et limité dans le temps
`ApiKeyService`, ExternalApplications	brique Integrations du socle	concept	Clés API d'un client : côté socle (phase 3 du socle), pas dans la console
`WebhookDispatcherService`	brique Integrations du socle	concept	Journal des livraisons, nouvelles tentatives
`LicenseTokenService` (HMAC)	Licensing	concept	Signature asymétrique (Ed25519), rotation des clés
Influencer, distributeurs, applications externes (Shop, ECU)	—	non	Restent dans core-api tant qu'elles y sont vendues (décision D7)

12 Les décisions de départ

#	Sujet	Décision de départ
D1	Où construire	Dans la nouvelle API, `platform/`, base centrale. Rien n'est construit dans core-api
D2	Réutilisation	Recopier les algorithmes (devis, passerelles, PDF, promotions), pas le code tel quel ni les tables
D3	Noms des tables	Préfixe par domaine de la console ; seules `tenants`, `domains` et `tenant_modules` gardent leur nom
D4	Identifiant du client	`tenants.id`, chaîne lisible (ex. `sdp`) ; tous les `tenant_id` sont des chaînes
D5	Montants	Entiers en F CFA ; prix figés à la commande ; grille de prix versionnée
D6	Avant l'accès à la console	Les routes de l'API console sont construites dès maintenant derrière un garde `console` ; en local et en test, ce garde accepte un agent de développement. LDAP ou comptes NetByUs viendront ensuite (chapitre 14)
D7	Applications externes	à confirmer Shop, ECU, Influencer restent vendues par core-api (tenant `netbyus`) ; la console BY+ ne les reprend pas en phase 1–3
D8	Modèle de prix	à confirmer prix par module et par utilisateur ; formules facultatives (chapitre 8)
D9	Durées du cycle de vie	à confirmer essai, grâce, conservation avant archivage
D10	Clients actuels	à confirmer chaque client actuel reçoit un abonnement avec les modules qu'il utilise (lus dans `module_installs`), au prix et à la durée décidés
D11	Inscription publique	à confirmer une page publique peut-elle créer un client (essai) sans intervention de NetByUs ?
D12	Jeton de licence	à confirmer une paire de clés pour la plateforme (proposé) ou une par client
D13	Identité des agents	décidé méthode jumelée : LDAP ou compte du client `netbyus` (chapitre 14)
D14	Droits sur la console	décidé inscription explicite dans `staff_users`, jamais par un administrateur de `netbyus`
D15	Double authentification	décidé obligatoire pour tous les agents

13 Le plan de construction

Lot 1 · Fondations de la console — phase 1

- Le noyau charge `platform/*/platform.php` sur les domaines centraux (même principe que les briques) ; Deptrac : le socle n'importe pas `Platform\`.
- Garde `console` (interface d'authentification) et agent de développement pour le local et les tests ; type d'acteur « agent NetByUs » dans le noyau.
- Audit de la console : `aud_logs`, acteur obligatoire sur chaque route.
- Passer le cache à Redis partout (déjà fait dans Docker).

Terminé quand : un domaine vide de `platform/` est chargé et testé ; une action de test est journalisée avec son acteur.

Lot 2 · Tenants : créer et déployer un client — phase 1

- Colonnes `status` et `deployment` sur `tenants`, `is_primary` sur `domains` ; `tenant_profiles`, `tenant_contacts`, `tenant_provisioning_steps`.
- Déploiement en 8 étapes rejouables (chapitre 9) ; premier administrateur dans `core_users`.
- Routes : `POST /api/v1/console/tenants` (créer), `POST …/tenants/{id}/provision` (reprendre à l'étape en échec), `GET …/tenants`, `GET …/tenants/{id}`.

Terminé quand : un client est créé de bout en bout ; une étape en échec se relance sans recréer les précédentes.

Lot 3 · Catalog, Subscriptions, Activation — phase 1

- `catalog_modules` synchronisé depuis les manifestes à chaque déploiement ; seuls les modules `sellable` sont proposés.
- Abonnement : `sub_subscriptions`, `sub_subscription_modules`, `sub_events` ; routes `PUT /api/v1/console/tenants/{id}/subscription` (modules, utilisateurs, dates) et `GET …/subscription/events`.
- Réconciliateur et `act_runs` ; colonne `origin` sur `tenant_modules` ; écriture par le guichet du registre.
- Publication : cache versionné, quotas, état d'accès, événement temps réel ; `POST /api/v1/console/tenants/{id}/reconcile` et passage de nuit (tâche planifiée).
- Quota d'utilisateurs appliqué par Auth (création refusée au-delà).

Terminé quand : le critère du chapitre 1 est vérifié par un test de bout en bout (client créé, abonnement attribué, modules requis activés, 11e utilisateur refusé, module non souscrit en 403).

Lot 4 · Calculateur, grilles, promotions — phase 2

- `catalog_price_lists`, `catalog_prices`, `catalog_options`, `catalog_promotions` ; calculateur unique en entiers, prorata.

Terminé quand : le même devis sort pour la console, « Mon abonnement » et le paiement ; une nouvelle grille ne change pas un abonnement en cours.

Lot 5 · Commandes et paiement — phase 2

- `ord_orders`, `ord_order_lines` ; passerelles FedaPay et KkiaPay recopiées ; webhooks dédoublonnés ; factures et PDF.
- Guichets pour la brique Subscription du socle (« Mon abonnement ») : voir, composer, payer. Le client est déduit du domaine, jamais d'un paramètre.

Terminé quand : un client ajoute un module, paie, et le module s'active seul ; un webhook rejoué ne crée ni second paiement ni seconde activation.

Lot 6 · Cycle de vie — phase 2

- Tâche quotidienne : essai, grâce, suspension, résiliation ; bandeau de grâce ; connexion bloquée en suspension sauf abonnement et export.

Terminé quand : un abonnement non renouvelé passe en grâce puis en suspension, et un paiement rétablit tout sans intervention.

Lot 7 · Exploitation — phase 3

- Migrations suivies par client, sauvegardes MinIO et restauration testée, place utilisée, archivage sous double validation.

Terminé quand : une migration générale montre les clients en échec et les reprend ; une sauvegarde est restaurée sur une base de test.

Lot 8 · Support, licences signées, annonces — phase 3

- Tickets BY+ (source = client et module), accès support accordé par l'administrateur du client, tracé des deux côtés.
- Clés de licence signées (Ed25519) pour les serveurs dédiés et les applications ; annonces.

Terminé quand : un ticket se traite avec un accès accordé qui s'éteint seul ; une licence se vérifie hors ligne.

Lot 9 · Reprise des clients actuels — phase 3

- Pour chaque client de core-api : profil (depuis `tenant_information`), abonnement avec les modules de `module_installs` (décision D10), données importées module par module (architecture, étape 5).

Terminé quand : le dernier client tourne sur la nouvelle API avec un abonnement ; `netbyus` est un client ordinaire.

Lot 10 · Staff et accès à la console — ensuite (après les lots 1 à 3)

- `staff_users`, `staff_roles`, `staff_user_roles` ; inscription d'un agent par un super-administrateur uniquement.
- Pilotes d'identité `ldap` et `netbyus` derrière le garde `console` ; compte de secours `local`.
- Double authentification (TOTP) obligatoire, codes de récupération ; jetons courts limités à `/api/v1/console/…`.
- Brique Auth du socle : connexion LDAP configurable par client (profite à `netbyus`).

Terminé quand : un agent inscrit se connecte par LDAP ou par son compte `netbyus` avec double authentification ; un compte valide mais non inscrit est refusé ; un administrateur de `netbyus` ne peut pas s'inscrire.

Première étape concrète

> Lot 1 puis lot 3 sur un client créé à la main : `sub_subscriptions` et `sub_subscription_modules`, la colonne `origin` de `tenant_modules`, le réconciliateur, et un test qui prouve qu'un module ajouté à l'abonnement s'active avec ses modules requis, et qu'un module retiré renvoie 403. Le reste se construit autour de cette chaîne.

Limites de l'analyse : migrations, modèles, routes et services relus ; le détail des contrôleurs et le front n'ont pas été relus. Les branches de core-api divergent : la vente de licences complète n'existe que sur des branches de fonctionnalité (la plus récente : `Feature/Add/HelperDeskWebhookTicketId`, 1er septembre 2026).

14 L'accès à la console

L'authentification est branchée après les lots 1 à 3, mais elle conditionne la sécurité de toute la plateforme : la console agit sur **tous** les clients. Deux approches ont été étudiées ; elles sont **retenues ensemble**, encadrées par trois conditions.

	A · LDAP (annuaire de NetByUs)	B · Utilisateurs du client `netbyus`
Principe	L'agent saisit son identifiant d'annuaire ; l'API vérifie le mot de passe auprès du serveur LDAP, puis délivre un jeton Sanctum	L'agent se connecte avec son compte BY+ du client `netbyus` (table `core_users` de sa base) ; certains comptes sont autorisés dans la console
Avantages	Une seule identité pour tout NetByUs ; un départ est coupé partout en un geste ; politique de mots de passe de l'entreprise ; groupes de l'annuaire	Rien de nouveau à installer ; mêmes écrans de connexion et de mot de passe oublié que BY+
Risques	Le serveur LDAP doit être joignable par l'API (réseau, VPN) et disponible ; pas de double authentification native	**Un client gouverne la plateforme** : qui prend la main sur le tenant `netbyus` (mot de passe volé, administrateur du client, accès support) prend la main sur tous les clients. La base centrale dépend d'une base client (migration, suspension, panne de `netbyus` = console fermée)
Prérequis	NetByUs dispose d'un annuaire (Active Directory, OpenLDAP…) ; paquet `LdapRecord-Laravel`	Aucun
Retenu	oui pilote `ldap`	oui pilote `netbyus`, comme preuve d'identité seulement : les trois conditions ci-dessous neutralisent le risque

Décision : une méthode jumelée

> Les deux approches sont retenues **ensemble**. L'agent prouve son identité par **LDAP** ou par son **compte BY+ du client** `netbyus` (qui peut lui-même se connecter par LDAP si l'annuaire est configuré pour ce client). La console applique ensuite **trois conditions** et exige la **double authentification**. On sépare toujours « qui êtes-vous » (identité) de « que pouvez-vous faire » (droits, toujours en base centrale).

Les trois conditions (toutes obligatoires)

1. **L'agent est inscrit explicitement** dans la table centrale `staff_users`, active, avec son rôle de console. Un compte LDAP ou `netbyus` valide ne suffit pas.
2. **Un administrateur du client** `netbyus` **ne peut pas s'y ajouter lui-même** : l'inscription se fait uniquement depuis la console, par un super-administrateur, et elle est journalisée. Rien dans la base du client `netbyus` ne donne un droit sur la console.
3. **La double authentification est obligatoire** pour tous les agents (code à usage unique, TOTP), même si l'identité vient de LDAP ou de `netbyus`.

Le parcours de connexion

#	Étape	Ce qui se passe
1	Identifiant et mot de passe	Sur le domaine central, route de connexion de la console
2	Preuve d'identité	Pilote `ldap` (liaison à l'annuaire) ou pilote `netbyus` (vérification par le guichet de la brique Auth du client `netbyus`, qui peut elle-même s'appuyer sur LDAP)

3	Condition 1	L'identité trouvée correspond à une ligne active de `staff_users` (même fournisseur, même identifiant externe) ; sinon refus
4	Condition 3	Code de double authentification vérifié
5	Jeton	Jeton Sanctum à durée courte, valable uniquement pour les routes `/api/v1/console/…` ; acteur « agent NetByUs » dans le contexte
6	Trace	Connexion réussie ou refusée journalisée (`aud_logs`) ; côté `netbyus` , la connexion apparaît aussi dans son propre journal

Ce que ça implique dans le code

Élément	Contenu
`staff_users` (Staff)	email, nom, fournisseur (`ldap` , `netbyus` , `local`), identifiant externe (DN ou uid LDAP, id du compte `netbyus`), actif, secret TOTP chiffré, codes de récupération hachés, ajouté par, dernière connexion
`staff_roles` , `staff_user_roles`	Rôles de console : super-administrateur, commercial, support, finance, ops
Pilotes d'identité	Une interface (`ConsoleIdentityProvider`) et deux implémentations : `LdapIdentityProvider` (paquet `LdapRecord-Laravel`), `NetByUsIdentityProvider` (guichet de la brique Auth, jamais une lecture directe de la base du client)
Brique Auth du socle	Connexion par LDAP **configurable par client** (adresse, base DN, compte de liaison chiffré, correspondance des attributs) : profite à `netbyus` et à tout client qui le demande
Compte de secours	Un compte `local` (mot de passe et TOTP), à usage exceptionnel et tracé, pour le jour où LDAP et `netbyus` sont indisponibles
Révocation	Compte LDAP désactivé, compte `netbyus` désactivé ou ligne `staff_users` inactive : la connexion échoue et les jetons en cours sont révoqués

#	Décision	Retenu
D13	Identité par LDAP	oui pilote `ldap` de la console, et LDAP configurable par client dans la brique Auth
D14	Identité par les comptes `netbyus`	oui pilote `netbyus` , aux trois conditions
D15	Double authentification	obligatoire pour tous les agents

Le branchement reste prévu après les lots 1 à 3 : les routes sont construites derrière le garde `console` , et le lot « Staff » (comptes, pilotes, double authentification) s'insère sans les modifier.