BY+ · ARCHITECTURE DE RÉFÉRENCE

L'architecture de BY+, expliquée simplement

Un seul logiciel, des modules que chaque client choisit,
une console NetByUs qui décide par des licences signées,
et des règles simples que tout le monde peut comprendre.
C'est la base du redémarrage de BY+.

À lire par toute l'équipe : développeurs, produit, commercial, direction.
Les pages 1 à 3 et le chapitre 10 (personnalisation) s'adressent à tout le monde ; les autres détaillent la technique.
La liste des modules et des ponts est dans le « Catalogue des modules » ; la console, dans « La console NetByUs : plan de mise en place ».
Septembre 2026. Mise à jour du 3 octobre 2026 : plusieurs serveurs (instances), licences signées, console installée en package sur le seul serveur NetByUs (chapitres 4 et 5).

1 BY+ en une page

> **En une phrase.** BY+ est un seul logiciel, écrit une seule fois, utilisé par tous les clients. Chaque client compose son offre, c'est-à-dire l'ensemble des modules qu'il prend, et ne voit, ne charge et ne paie que ceux-là. Ce qu'il a est décidé par NetByUs, dans une licence signée.

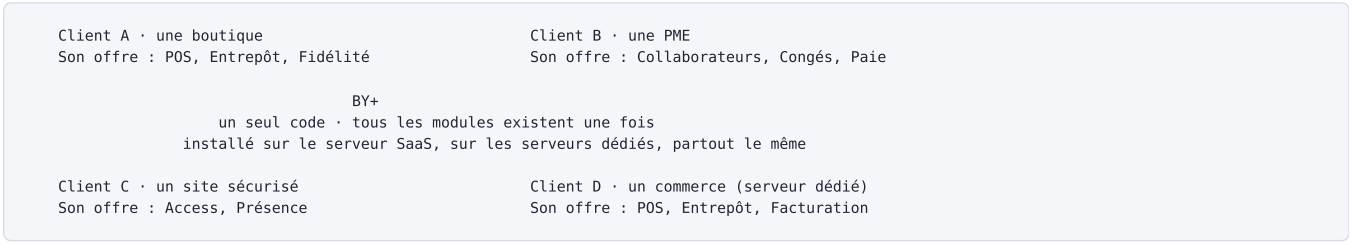


Figure 1 : un seul BY+, quatre clients, quatre combinaisons de modules. Aucun client n'a « sa » version du code, même sur son propre serveur.

Ce que ça change pour chacun

Pour…	Concrètement
Le client	Il ne voit que ce qu'il a pris : menus, écrans et droits. L'application est plus légère et plus rapide. Il peut ajouter un module à tout moment, sans migration ni attente.
Le commercial	Il compose avec le client son offre. Ajouter ou retirer un module est une mise à jour de son abonnement dans la console : aucun développement.
Le produit	Chaque fonctionnalité est un module avec un nom métier, un périmètre clair et un responsable.
Le développeur	Il travaille dans un dossier (son module) avec des règles claires. Une modification dans un module ne peut pas casser un autre module.
NetByUs	Un seul code pour tous les clients ; un correctif est écrit une fois, pour tout le monde. Les clients, abonnements, licences et tickets se gèrent depuis la console NetByUs, quel que soit le serveur du client.

Les 4 mots à connaître

Socle	Module	Pont	Offre
Ce que tout client a toujours : comptes, droits, sociétés, fichiers, notifications.	Une fonctionnalité qu'un client peut utiliser seule : POS, Paie, Access, Contacts…	Ce qui relie deux modules quand un client a les deux. Ex. : une vente au POS fait baisser le stock.	Les modules choisis par un client, dans son abonnement, et livrés à son serveur par une licence signée.

Pour aller plus loin : le module de base (requis par d'autres, jamais vendu seul, ex. Catalogue), l'extension client (chapitre 10), les instances et la console NetByUs (chapitres 4 et 5).

2 L'image à retenir : un smartphone

Pour expliquer BY+ à quelqu'un qui n'est pas développeur, on peut le comparer à un smartphone.

Smartphone	BY+	Pourquoi c'est pareil
Le système du téléphone (accueil, réglages, compte)	Le socle	Toujours là, identique pour tous, et chaque application s'en sert.
Une application (Agenda, Photos…)	Un module	Elle fait une chose, fonctionne seule, et on peut l'installer ou la retirer sans casser le téléphone.
Le bouton « Partager vers… »	Un pont	Il n'apparaît que si les deux applications sont présentes. Aucune ne dépend de l'autre.
Les applications que chacun a installées	Une offre	C'est le choix du client, pas un autre téléphone.
Fond d'écran, sonneries, widgets	La personnalisation	Chacun adapte son téléphone sans que le fabricant réécrive le système (chapitre 10).
L'opérateur et son espace de gestion des abonnés ; la carte SIM	La console NetByUs ; la licence signée	L'opérateur voit tous les abonnés et leurs forfaits ; la SIM porte le forfait dans le téléphone, et l'abonné ne peut pas la réécrire.

> Tous les téléphones d'un même modèle ont le même système. Ce qui les distingue, ce sont les applications installées et les réglages. Il en va de même pour les clients BY+ : même code, modules et réglages différents.

Comment savoir si quelque chose est un module ?

#	Question	Si « non »
1	Un client peut-il s'en servir seul (hors modules qu'il requiert) ?	Requis par plusieurs modules : module de base. Sinon : partie d'un module, ou pont.
2	A-t-il son entrée de menu et ses écrans ?	C'est un morceau du socle ou d'un autre module.
3	A-t-il ses propres droits (voir, créer, valider…) ?	Trop petit : on le regroupe avec un module voisin.
4	Peut-on le retirer sans casser un autre module ?	Tout le monde en a besoin : il va dans le socle. Si d'autres modules le requièrent, le système refuse de le retirer tant qu'ils sont actifs.

3 La carte : socle, modules, ponts, offres

```
OFFRE D'UN CLIENT : les modules de son abonnement, livrés par sa licence signée
  POS + Entrepôt   Collab. + Congés   Access + Présence   Pipeline + RDV   Projets + Tâches   … au choix

MODULES : extrait du catalogue (33 modules)
  Vendre     Catalogue (base) · POS —pont— Entrepôt · Facturation · Clients · Contacts · Pipeline, RDV, NPS
  Personnes  Collaborateurs · Congés · Paie · Sites · Access —pont— Présence
  Piloter    Tâches · Projets · Dépenses        Animer   Event · Portail client
  Liste complète, exigences et ponts : « Catalogue des modules ».

SOCLE : présent chez tous les clients, jamais vendu séparément
  Comptes et droits · Sociétés · Paramètres · Fichiers · Notifications · Journal · Validations · Tableau de bord
  Personnalisation · Modèles de documents · Mon abonnement · Support · Intégrations · Registre des modules · Lien avec la console · Kit d'interface
```

Figure 2 : chaque étage n'utilise que l'étage du dessous. Les modules facultatifs l'un pour l'autre sont reliés par des ponts.

Les liens autorisés, en une table

Qui	Peut utiliser	Ne peut jamais utiliser
Socle	Rien d'autre que lui-même	Un module, un pont, une extension, la console
Module	Le socle, et la vitrine d'un module qu'il déclare requérir (ex. Paie → Collaborateurs)	L'intérieur d'un autre module ; un module qu'il ne requiert pas ; une extension
Pont	Le socle et la vitrine de ses deux modules	L'intérieur de ses modules
Extension client	Le socle et la vitrine du module qu'elle complète	L'intérieur du module
Console NetByUs	Le noyau et les guichets du socle	Les tables du socle ou d'un module, la base d'un client
Offre	Des données : l'abonnement dans la console, la licence sur le serveur	Du code

La vitrine d'un module est son dossier `Public/` : les quelques classes qu'il accepte de montrer (interfaces, événements, points d'extension, objets de données). Tout le reste est privé.

4 Vue d'ensemble technique



Figure 3 : sur chaque serveur, une API, une base centrale, une base par client, Redis et des workers. La passerelle matérielle d'Access reste un service séparé.

Les instances : un même BY+ sur plusieurs serveurs

Serveur (instance)	Ce qu'il héberge	Domaine
Serveur NetByUs	BY+ pour le client `netbyus`, et le package de la console	`byplus.io`, `netbyus.byplus.io`
Serveur SaaS	BY+ pour tous les clients en SaaS, une base chacun	`{client}.byplus.io`
Serveur dédié (0 à n)	BY+ pour un seul client, chez NetByUs ou chez lui, mis à jour par NetByUs ou par le client	Son propre domaine, ex. `erp.sdp.bj`

La console NetByUs

NetByUs administre BY+ depuis une seule console : clients, offres et tarifs, abonnements et licences, paiement, support, applications externes et leurs licences, sauvegardes, annonces. C'est un **package installé par une commande sur le seul serveur NetByUs** (`netbyus/byplus-console`, `php artisan byplus:console:install`) : les autres serveurs n'en ont jamais le code. La console ne voit pas les données métier des clients. NetByUs reste par ailleurs un client normal de BY+ (`netbyus`) pour ses propres besoins.

La licence signée : ce que chaque serveur applique

- **La console décide :** l'abonnement d'un client devient une licence (modules, quotas, accès, date de fin) **signée** par la clé privée de NetByUs, qui ne quitte jamais le serveur NetByUs.
- **L'instance applique :** chaque serveur vient chercher la licence de ses clients (sortie HTTPS seulement, avec un webhook pour aller vite), la vérifie avec la clé publique et l'applique. Une licence modifiée est refusée : un client ne peut pas s'activer un module seul, même sur son propre serveur.
- **Sans console :** un serveur fonctionne jusqu'à la date de fin de sa licence, puis passe en grâce et en lecture seule. Un renouvellement arrive par le lien ou par un fichier de licence importé.
- **Toutes les versions restent supportées :** le lien avec la console n'évolue que par ajouts.

Ce que le socle de chaque serveur porte pour la console

Brique du socle	Rôle
`core/Licensing`	Vérifie la licence signée (clé publique de NetByUs), la garde, en déduit modules, quotas et accès ; « Ma licence » et import d'un fichier
`core/Link`	Enrôlement (`byplus:link:enroll`), interrogation toutes les 5 minutes et webhook « viens chercher » ; exécute le déploiement d'un nouveau client ; rend compte (version, modules, licences appliquées)
`core/Subscription`	« Mon abonnement » du client (abonnement, compléments, paiements, licences commerciales) : vérifie les droits `subscription.*`, puis demande à la console par le lien. Sur le serveur NetByUs, la console répond sur place

La console, elle, ne sert **aucune route sur le domaine d'un client** : elle tient le registre des clients (où qu'ils soient hébergés), le catalogue des modules connus (son code et ceux que déclarent les instances), et répond aux serveurs par le lien.

Les choix techniques

Sujet	Choix	Pourquoi
Forme générale	Monolithe modulaire : une application découpée en modules ; la console à part, en package	Simple à développer, déployer et déboguer. Pas de réseau entre modules ; une transaction SQL suffit quand deux modules agissent ensemble
Back-end	Laravel 13, PHP 8.4+	Déjà maîtrisé par l'équipe ; écosystème complet

Sujet	Choix	Pourquoi
Clients	Une base par client (stancl/tenancy), identifiée par son domaine	Isolation forte, sauvegarde ou restauration d'un client seul
Serveurs	Plusieurs instances du même code, pilotées par une seule console par licences signées (Ed25519)	SaaS et serveurs dédiés sans version différente, sans perte de contrôle
Base de données	PostgreSQL	Robuste, déjà en place
Cache et files	Redis + Horizon (files urgent, notifications, défaut, lourd), Supervisor en production	Ce qui est lu souvent ne touche pas la base ; un traitement long ne retarde jamais un code de connexion
Front-end	Vue 3 + Vite + Pinia + TypeScript	Continuité avec l'existant ; stores chargés à la demande
Interface	Un kit d'interface interne (dans le socle), le template Metronic de core-app	Une seule manière de faire un tableau, un formulaire ou une carte
Traductions	Fichiers PHP à clés par brique, module et domaine (API) ; JSON par module (front) ; langue par l'en-tête `X-localization`	Conforme à Laravel ; mails dans la langue de l'écran
Fichiers, temps réel	S3 (MinIO) · Laravel Reverb	Standards, hébergeables par nous
Qualité	PHPUnit, Vitest · Pint, Larastan, ESLint · Deptrac · Telescope en développement	Deptrac vérifie les liens autorisés entre socle, modules, ponts, extensions et console

> **Ce qu'on ne fait pas :** pas de microservices ; pas de code spécifique à un client dans un module ; pas de branche Git par produit ni par client ; pas de version du code par serveur ; pas de schéma de base différent selon les clients. Chacun de ces choix complique le système pendant des années pour un gain que nous n'avons pas.

5 Organisation du code

Deux dépôts, un pour l'API et un pour l'application web, et un package privé pour la console. Un module porte le même nom côté API et côté web : quand on cherche « le POS », on sait où regarder.

```
byplus/
├── api/                      ← Laravel (dépôt core-api), installé sur chaque instance
│   ├── core/                 ← LE SOCLE : Tenancy, Auth, Companies, Files, Notifications, Modules,
│   │                           Link (lien avec la console), Kernel, ...
│   ├── modules/
│   │   ├── catalogue/        ← module de base
│   │   ├── pos/              ← UN MODULE (voir page suivante)
│   │   └── ...
│   ├── bridges/
│   │   └── pos-entrepot/     ← UN PONT
│   ├── extensions/
│   │   └── sdp-primes/       ← UNE EXTENSION CLIENT (chapitre 10)
│   ├── packages/byplus-console/ ← la console (package, serveur NetByUs seulement)
│   └── docs/                 ← architecture, catalogue, socle, console, déploiement
│
├── app/                      ← application des clients (Vue 3, dépôt core-app)
│   └── src/
│       ├── core/             ← shell, connexion, kit d'interface, i18n, droits
│       ├── modules/          ← mêmes noms que côté API
│       ├── bridges/
│       └── extensions/
│
└── netbyus/byplus-console    ← PACKAGE PRIVÉ de la console (API + écrans), serveur NetByUs seulement
```

Anatomie d'un module (et d'une brique du socle, et d'un domaine de la console)

```
api/modules/pos/                                    app/src/modules/Pos/
├── manifest.php      ← carte d'identité            ├── manifest.ts      ← carte d'identité
├── PosServiceProvider.php                          ├── routes.ts
├── Public/           ← VITRINE (seule partie       ├── menu.ts
│   ├── Events/          visible des autres)        ├── store.ts         ← Pinia
│   ├── Contracts/                                  ├── views/
│   ├── Data/                                       ├── components/      ← privés au module
│   └── Hooks/        ← points d'extension          └── lang/fr.json, en.json
├── Actions/          ← 1 classe = 1 action métier
├── Services/         ← logique interne             Tout composant utile à plusieurs modules
├── Providers/        ← branchements vers l'extérieur va dans le kit d'interface du socle
├── Http/             ← contrôleurs fins            (app/src/core/ui).
├── Models/
├── Jobs/ · Console/ · Listeners/
├── routes/tenant.php
├── database/migrations/, seeders/   ← tables pos_*
├── lang/fr/messages.php, lang/en/messages.php
└── tests/
```

La carte d'identité (manifeste)

```php
// api/modules/pos/manifest.php
return [
    'key'         => 'pos',
    'name'        => 'POS',
    'version'     => '1.0.0',
    'requires'    => ['catalogue'],              // modules indispensables (2 niveaux max)
    'permissions' => ['pos.vendre', 'pos.cloturer', 'pos.parametrer'],
    'roles'       => ['caissier' => ['pos.vendre']],
    'settings'    => ['arrondi' => null, 'ticket_logo' => true], // réglages par client
    'seeders'     => [MoyensPaiementParDefaut::class],
    'schedule'    => [ClotureSessionsOubliees::class => 'daily'],
    'documents'   => [                           // documents que le client peut personnaliser (chapitre 10)
        'pos.invoice' => ['name' => 'pos::documents.invoice', 'format' => 'a4', 'view' => 'pos::documents.invoice'],
    ],
];

// api/modules/catalogue/manifest.php (extrait)
'sellable' => false,     // module de base : activé par POS, Entrepôt, Facturation
```

L'offre d'un client n'est pas dans le code : c'est son abonnement dans la console, livré à son serveur par sa licence signée, qui alimente la table `tenant_modules` de l'instance.

6 Comment les modules travaillent ensemble

Il n'existe que trois façons de faire coopérer des modules. Toute autre façon est interdite et bloquée par Deptrac.

Mécanisme	Quand l'utiliser	Exemple
Événement	Prévenir que quelque chose s'est passé, sans savoir qui est intéressé	`VenteEffectuee` , `CongeValide` , `ContactFusionne`
Vitrine	Utiliser un module qu'on requiert (déclaré dans `requires`) : jamais de pont dans ce cas	Paie appelle `Collaborateurs\Public\Contracts\Annuaire` ; Facturation lit les clients dans Contacts
Pont	Faire travailler ensemble deux modules facultatifs l'un pour l'autre	`pos-entrepot` : chaque vente crée une sortie de stock

Et à l'écran ? Des emplacements

Le socle définit des emplacements nommés dans l'interface : onglets d'une fiche, widgets du tableau de bord, actions d'une ligne. Un module, un pont ou une extension y dépose ses composants. Contacts ne connaît ni la facturation ni les projets : Facturation requiert Contacts, c'est elle qui ajoute l'onglet « Factures » ; Projets ne requiert pas Contacts, c'est le pont `projets-contacts` qui ajoute l'onglet « Projets », seulement chez les clients qui ont les deux.

```
// app/src/modules/facturation/manifest.ts (extrait) : requiert Contacts, pas de pont
slots: { 'contact.onglets': () => import('./OngletFactures.vue') },

// app/src/bridges/projets-contacts/manifest.ts : facultatifs l'un pour l'autre
export default {
  when: ['projets', 'contacts'],
  slots: { 'contact.onglets': () => import('./OngletProjets.vue') },
}
```

Requis ou pont : que faire, point par point

Deux modules qui doivent travailler ensemble sont dans l'un de ces deux cas, jamais les deux. La question qui tranche : **A a-t-il encore un sens chez un client qui n'a pas B ?** Non : A requiert B. Oui : ils sont facultatifs l'un pour l'autre, et un pont les relie.

Sujet	A requiert B (ex. Paie → Collaborateurs)	A et B reliés par un pont (ex. POS et Entrepôt)
Déclaration	`requires: ['b']` dans le manifeste de A. B ne sait rien de A	Un dossier `bridges/a-b` avec `when: ['a', 'b']` . Ni A ni B ne déclarent l'autre
Code	A appelle la vitrine de B (`B\Public\Contracts`). Jamais l'intérieur de B, jamais l'inverse	Le pont appelle la vitrine de A et celle de B, ou écoute leurs événements. A et B ne s'appellent pas
Données	Clé étrangère de A vers une table de B permise	Aucune clé étrangère entre A et B. Le module qui porte le lien garde un simple identifiant (ex. `projets_projets.contact_id` , sans contrainte) ; c'est le pont qui le remplit et le lit
Écran et menu	A dépose ses onglets et actions dans les emplacements de B. Le menu de A s'affiche ; B a son entrée s'il a des écrans, même activé seulement comme requis	Le pont dépose ses composants dans les emplacements de A et de B. **Un pont n'a jamais d'entrée de menu** : il enrichit les écrans de ses deux modules
Vente	On vend A. B s'ajoute de lui-même à la licence, sans ligne de prix à part ; un module de base (`sellable: false`) n'est jamais vendu seul. Dans une offre, il suffit de cocher A	Le pont ne se vend pas et n'apparaît ni au catalogue ni dans une offre. Il est offert à qui a les deux modules
Activation	B est activé avant A, marqué « requis »	Le pont s'active tout seul dès que A et B sont actifs chez le client
Retrait	Retirer B est refusé tant que A est actif. Retirer A laisse B s'il a été choisi ; s'il n'était là que comme requis, il part avec A	Retirer A ou B coupe le pont ; l'autre module continue seul. Les données du lien restent
Test	Le test d'activation de A active aussi B	Trois tests : A seul, B seul, puis A et B avec le pont

Les cas particuliers

Cas	Que faire
A requiert B **et** B requiert A	Interdit. Soit c'est un seul module (les fusionner), soit ils partagent quelque chose : le sortir dans un module de base que tous deux requièrent
A requiert B, qui requiert C, qui requiert D	Deux niveaux au plus. Au-delà, le découpage est trop fin : regrouper, ou faire descendre la partie commune dans un module de base
A requiert B, mais des clients veulent A sans B	Le lien devient facultatif : retirer B de `requires` , déplacer dans un pont `a-b` le code de A qui appelle B, remplacer la clé étrangère par un identifiant simple. A doit alors marcher seul
Un pont devient indispensable (personne n'a A sans B)	Le chemin inverse : A requiert B, le code du pont rentre dans A, le pont disparaît

Cas	Que faire
A requiert B, et A a aussi un lien facultatif avec C	Les deux à la fois : `requires: ['b']` dans A, et un pont `a-c`. Jamais un pont entre A et B
Trois modules facultatifs travaillent ensemble	Un pont par paire tant que c'est possible. Un pont à trois (`when` à trois modules) seulement si la fonction n'a de sens qu'avec les trois
Le pont a besoin d'un réglage ou d'un droit	Il le range dans celui de ses deux modules qui porte la fonction ; un pont ne déclare ni droit ni réglage à lui
On hésite entre événement et pont	A prévient sans attendre de réponse et ne sait pas qui écoute : événement, écouté par le pont. Il faut une réponse ou un écran commun : pont

Deptrac bloque le reste : un module qui appelle un module non requis, un pont qui touche l'intérieur d'un module, un module qui appelle un pont.

Côté front : qui a le droit d'importer quoi

Le front suit les mêmes étages que l'API. Tout tient en une règle : **on n'importe jamais un fichier d'un autre module, seulement sa vitrine**, et seulement si on a le droit de le connaître.

```
app/src/modules/Collaborateurs/
├── manifest.ts      ← carte d'identité : clé, routes, menu, emplacements, requires
├── public/          ← LA VITRINE du module : le seul dossier que les autres peuvent importer
│   └── index.ts        types, fonctions de lecture, composants prévus pour être réutilisés
├── views/           ← écrans (privés)
├── components/      ← composants (privés)
└── api.ts, store.ts ← appels à l'API et état (privés)

app/src/bridges/pos-entrepot/
├── manifest.ts      ← when: ['pos', 'entrepot'] + ce qu'il dépose dans les emplacements
└── components/      ← ses composants (privés)
```

Qui importe	Peut importer	Ne peut pas importer
Le socle (`@/core`)	Lui-même	Un module, un pont, une extension
Un module	`@/core`, ses propres fichiers (imports relatifs), et `@/modules/B/public` pour un module B qu'il déclare dans `requires`	Tout autre fichier de B (`@/modules/B/views/…`) ; la vitrine d'un module non requis ; un pont ; une extension
Un pont	`@/core`, ses propres fichiers, la vitrine de chacun de ses modules (`when`)	L'intérieur de ses modules ; un autre pont
Une extension	`@/core` et la vitrine des modules qu'elle étend	L'intérieur d'un module ; un pont

Personne n'importe un pont ni un module entier. Le socle les charge lui-même, à la demande, d'après les modules actifs du client : un module absent de son offre n'est jamais téléchargé, et un pont n'est chargé que si tous ses modules le sont.

Trois situations, trois réponses

Besoin	Comment l'écrire
Paie affiche le nom d'un collaborateur (Paie requiert Collaborateurs)	`import { useCollaborateur } from '@/modules/Collaborateurs/public'`. Rien d'autre de Collaborateurs
Facturation ajoute un onglet « Factures » à la fiche contact (elle requiert Contacts)	Facturation ne modifie pas la fiche : elle dépose son onglet dans l'emplacement `contact.onglets`, depuis son `manifest.ts`. Contacts ne sait pas qui remplit son emplacement
La caisse montre le stock d'un article (POS et Entrepôt, facultatifs l'un pour l'autre)	POS déclare un emplacement `pos.article.infos`. Le pont `pos-entrepot` y dépose son composant, qui lit le stock par la vitrine d'Entrepôt. POS n'importe rien d'Entrepôt ; sans Entrepôt, l'emplacement reste vide

Exemple 1 : Paie requiert Collaborateurs

```
// app/src/modules/Collaborateurs/public/index.ts : LA VITRINE, tout ce que les autres voient
export type { Collaborateur } from '../types'
export { useCollaborateur } from '../composables/useCollaborateur'   // lire un collaborateur
export { default as ChoixCollaborateur } from '../components/ChoixCollaborateur.vue'

// app/src/modules/Paie/manifest.ts
export default defineModule({
  key: 'paie',
  name: 'Paie',
  requires: ['collaborateurs'],                     // comme manifest.php côté API
  routes: [{ path: 'bulletins', name: 'paie.bulletins', component: () => import('./views/Bulletins.vue') }],
  menu: [{ label: 'paie.menu.bulletins', route: 'paie.bulletins', icon: 'fas fa-money-check' }],
  // Paie ajoute son onglet à la fiche d'un collaborateur, sans toucher au module Collaborateurs
  slots: { 'collaborateur.onglets': () => import('./components/OngletBulletins.vue') },
})

// app/src/modules/Paie/views/Bulletins.vue
import { session } from '@/core/session'                                  // le socle : permis
// la vitrine d'un module requis : permis
import { ChoixCollaborateur, useCollaborateur } from '@/modules/Collaborateurs/public'
import LigneBulletin from '../components/LigneBulletin.vue'               // son propre fichier : import relatif

import Fiche from '@/modules/Collaborateurs/views/Fiche.vue'              // INTERDIT : l'intérieur d'un autre module
import { useStock } from '@/modules/Entrepot/public'                      // INTERDIT : Entrepôt n'est pas dans requires
```

Exemple 2 : POS et Entrepôt, reliés par un pont

```
// app/src/modules/Pos/views/Caisse.vue : POS prévoit un emplacement, sans savoir qui le remplira
<SlotOutlet name="pos.article.infos" :article="article" />

// app/src/modules/Entrepot/public/index.ts : la vitrine d'Entrepôt
export { useStock } from '../composables/useStock'

// app/src/bridges/pos-entrepot/manifest.ts : chargé seulement si le client a POS ET Entrepôt
export default defineBridge({
  key: 'pos-entrepot',
  when: ['pos', 'entrepot'],
  slots: { 'pos.article.infos': () => import('./components/StockEnCaisse.vue') },
})

// app/src/bridges/pos-entrepot/components/StockEnCaisse.vue
import { useStock } from '@/modules/Entrepot/public'    // la vitrine d'un de ses deux modules : permis
const props = defineProps<{ article: { id: string } }>()
const stock = useStock(props.article.id)

// Dans POS : aucun import d'Entrepôt, nulle part. Chez un client sans Entrepôt, le pont n'est pas
// téléchargé et l'emplacement « pos.article.infos » reste vide : la caisse marche pareil.
```

Ce qui évite de tout mélanger

1. **La vitrine reste petite.** On y met ce qu'un autre module a une vraie raison d'utiliser : des types, une lecture (« le collaborateur n° X »), un composant de choix (« choisir un contact »). Jamais un écran entier, jamais le store.
2. **Un composant utile à plusieurs modules sans lien métier** (tableau, champ, fenêtre) va dans `@/core/ui`, pas dans une vitrine.
3. **Dans un module, les imports sont relatifs** (`./components/…`). `@/modules/…` ne sert qu'à atteindre la vitrine d'un autre module : un import qui commence ainsi se repère à la relecture.
4. **Les écrans se chargent à la demande** (`() => import('./views/…')`), y compris ce qu'un module ou un pont dépose dans un emplacement.
5. **Les traductions suivent le même découpage** : chaque module et chaque pont a ses fichiers, avec ses clés préfixées par sa clé.

Garde-fous en place dans le front : ESLint refuse tout import d'un autre module hors de sa vitrine, d'un pont ou d'une extension ; le test d'architecture (`src/core/__tests__/boundaries.spec.ts`) vérifie en plus que la vitrine importée est celle d'un module déclaré dans `requires` (ou dans le `when` du pont), imports relatifs compris. Le socle fournit `defineBridge`, l'activation des ponts et le composant d'emplacement `SlotOutlet`. Les dossiers des modules sont en PascalCase, comme dans l'API.

7 Chaque client ne charge que ses modules

> **Le principe.** Sur le serveur, le code de tous les modules est chargé une seule fois et reste en mémoire. À chaque requête, on vérifie en une lecture Redis que le client a le module. Dans le navigateur, on ne télécharge réellement que les modules du client : c'est là que se voit le gain.

```
1. Connexion : le navigateur apprend ce que le client possède
   Shell (léger)  ──  GET /me/bootstrap (modules actifs, droits, compteurs, 1 seul appel)
                  ──  télécharge uniquement les modules actifs : pos.js entrepot.js catalogue.js contacts.js

2. Chaque appel API : sdp.byplus.io/api/v1/pos/ventes
   Quel client ? sdp → base sdp (Redis) · A-t-il « pos » ? module:pos (Redis) · Action métier · Sinon refus (403)

Licence reçue → modules activés ou retirés → version changée dans Redis → le navigateur recharge ses modules.
```

Figure 5 : le navigateur charge seulement ce qu'il faut ; le serveur vérifie sans interroger la base pour savoir qui a quoi.

La même vérification partout

Porte d'entrée	Vérification (fournie par le socle)
Routes API	Le socle enveloppe automatiquement les routes du module dans le middleware `module:pos`
Tâches planifiées	Le socle ne lance la tâche du manifeste que pour les clients qui ont le module
Écouteurs d'événements	Ignorent l'événement si le module n'est pas actif
Tâches en file	Revérifient le module au démarrage (le client a pu le retirer entre-temps)
Écrans	Le navigateur n'a même pas le code des modules absents

La performance dès la conception

Règle	Effet
Config, routes et événements en cache ; opcache ; aucun état global	L'application démarre presque instantanément ; Octane pourra être activé sans réécriture
Client, modules actifs et droits en Redis	Zéro requête SQL pour savoir « qui a quoi »
Un seul appel de démarrage (`/me/bootstrap`)	Pas de cascade d'appels à l'ouverture
Un fichier JS par module ; bibliothèques lourdes chargées dans l'écran qui les utilise	Le premier chargement reste léger
Tâches longues toujours en file, rangées par nature (urgent, courant, lourd)	Les requêtes restent rapides ; un code de connexion n'attend jamais
Pagination obligatoire, index, chargement paresseux interdit	Pas de requêtes N+1 cachées
Budget de taille du shell et suivi du temps de réponse en CI	Toute régression est vue avant la mise en production

8 Les données

Où	Contient	Ne contient jamais
Base de la console (serveur NetByUs)	Clients, instances, abonnements, licences émises, catalogue et prix, commandes, factures, tickets, journal de la console	De donnée métier d'un client
Base centrale d'une instance	Ses clients et domaines, la licence reçue, les modules actifs qui en sont déduits, l'état publié	De donnée métier ; de quoi créer une licence
Base d'un client	Tables du socle et de tous les modules, qu'il les ait ou non ; ses réglages et personnalisations	De donnée d'un autre client

Les règles des tables

1. Chaque table appartient à un seul module et porte son préfixe : `pos_sessions`, `entrepot_mouvements`.
2. Toutes les tables existent chez tous les clients : toutes les bases ont le même schéma, et une mise à jour se fait en une seule commande. Activer un module est instantané : ses tables sont déjà là, vides.
3. Une clé étrangère ne pointe que vers le socle ou vers un module requis. Ex. `facturation_factures.contact_id` est permis (Facturation requiert Contacts) ; le client d'un projet passe par le pont `projets-contacts`.
4. Un module ne lit ni n'écrit jamais dans les tables d'un autre module. Il passe par la vitrine.
5. Les données restent quand on retire un module. Leur suppression est une action séparée, explicite, précédée d'un export.

Activer un module chez un client

Le commercial ajoute le module à l'abonnement du client dans la console. La console en fait une licence signée ; l'instance du client la reçoit, la vérifie et active le module. Les modules requis (ex. Catalogue, Contacts) s'ajoutent d'eux-mêmes, et les ponts s'activent tout seuls quand leurs deux modules sont présents.

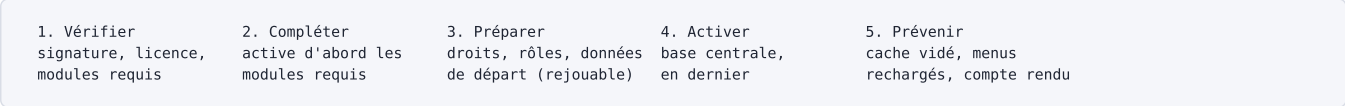


Figure 6 : aucune migration à l'activation. Chaque étape peut être relancée sans risque en cas d'erreur.

9 Les traductions

- **API :** chaque brique, module et domaine de la console a son dossier `lang/fr` et `lang/en`, avec des fichiers PHP à clés (comme Laravel) : `__('auth::messages.profile_saved')`. Jamais une phrase comme clé.
- **Front :** chaque module a `lang/fr.json` et `lang/en.json`, chargés avec lui.
- **Langue :** l'application envoie l'en-tête `X-localization` ; l'API répond, et écrit les mails déclenchés par la personne, dans cette langue.

10 Personnaliser pour un client

> **La règle de base.** On ne modifie jamais le code d'un module pour un seul client : pas de `if ($client === 'sdp')`, pas de copie du module, pas de branche. Une demande passe par une échelle de 5 niveaux, et on s'arrête au premier qui suffit.

Niveau	Ce qu'on fait	Qui	Code ?
1. Réglage	Le module a déjà l'option (déclarée dans `settings`) : on l'active pour ce client	Support ou client	Non
2. Configuration	Outils du socle : champs personnalisés, listes de valeurs, circuits de validation, modèles de documents (brique Templates, voir « Personnaliser un document » ci-dessous), rôles et droits, libellés	Client ou intégrateur	Non
3. Option pour tous	Le besoin peut servir à d'autres : nouveau réglage du module, désactivé par défaut	Équipe du module	Oui, dans le module
4. Extension client	Besoin vraiment propre au client : un petit module à part, qui se branche sans modifier le module	Équipe intégration	Oui, hors du module
5. Intégration	Le besoin relève d'un autre logiciel du client : API ou webhooks	Client ou intégrateur	Non, dans BY+

> La question qui tranche entre 3 et 4 : « un autre client pourrait-il le vouloir ? » Oui ou peut-être : niveau 3. Non, vraiment spécifique : niveau 4. Une fonctionnalité entièrement nouvelle passe le test des 4 questions : vendable à plusieurs clients, c'est un nouveau module ; sinon, une extension client.

Personnaliser un document

Le client met son logo, ses couleurs, ses textes et ses mentions sur les documents que BY+ produit pour lui : le mail d'invitation d'un nouvel utilisateur, le ticket de caisse, la facture, l'avoir, le mail qui accompagne un ticket... C'est de la configuration (niveau 2) : aucun code n'est écrit pour un client.

Règle	Ce qu'elle veut dire
Le propriétaire déclare ses documents	Une brique ou un module déclare les documents qu'il produit dans son manifeste (clé `documents`) : un nom traduit, un format, ses variables et son modèle par défaut. Auth déclare `auth.invitation`, POS déclare `pos.receipt`, `pos.invoice`... Le socle ne connaît la liste d'aucun module : il lit les manifestes, comme pour les droits.
Un modèle par document et par société	La brique Templates garde, pour chaque société, au plus un modèle actif par document, avec ses versions. Sans modèle, le document part avec le modèle par défaut de son propriétaire : rien ne casse, rien n'est à recopier.
Trois formats	`email` (mail mis en page), `receipt` (ticket de caisse, 80 mm) et `a4` (facture, avoir, bon). Le format est fixé par le document ; le client ne le choisit pas.
Ce qui est obligatoire reste imprimé	Un mail se compose librement à partir de blocs. Un ticket ou une facture garde la structure de son propriétaire : le client règle l'en-tête, le logo, les couleurs, les mentions et le pied de page, jamais les montants, les taxes ni le code de normalisation DGI.
Construire n'est pas envoyer	Templates rend le document (HTML ou PDF) ; c'est le propriétaire qui l'envoie, l'imprime ou le joint, par la brique Notifications. Un module inactif chez le client n'a aucun document proposé.

Comment fonctionne une extension client

Elle a la même forme qu'un module, vit dans `extensions/` et n'est activée que pour son client (par sa licence). Elle se branche par les mécanismes prévus : événements, emplacements d'écran, et points d'extension déclarés par le module dans `Public/Hooks/`.

```
// api/extensions/sdp-primes/manifest.php
return [
    'key'     => 'sdp-primes',
    'client'  => 'sdp',                      // ne peut être activée que pour ce client
    'extends' => ['paie'],                   // module qu'elle complète
    'bind'    => [
        \Modules\Paie\Public\Hooks\RegleDePrime::class => SdpRegleDePrime::class,
    ],
    'owner'   => 'équipe intégration',
];
```

Les 5 règles des extensions

1. Le module ne connaît jamais l'extension. Deptrac interdit tout lien de `modules/` vers `extensions/`.
2. Il manque un point d'extension ? L'équipe du module l'ajoute proprement, de façon générique. On ne contourne jamais le module.
3. Chaque extension a un responsable, des tests et un contrat : développement facturé, maintenance annuelle, mise à jour quand le module évolue.
4. Règle des 3 clients : dès que trois clients demandent la même chose, elle devient un réglage du module et l'extension est supprimée.
5. Revue annuelle : on garde les extensions utiles, on supprime les autres.

Demande du client	Réponse
« Je veux un champ Matricule interne sur mes contacts »	2 : champ personnalisé
« Mes congés doivent être validés par 2 managers »	2 : circuit de validation
« Je veux mon logo et mes mentions sur les factures »	2 : modèle de document
« Je veux arrondir les prix au POS à 5 F CFA »	3 : nouvelle option « arrondi » dans POS
« Ma prime d'ancienneté suit notre accord d'entreprise »	4 : extension `sdp-primes`
« Envoyez mes ventes à mon logiciel comptable »	5 : webhook ou connecteur
« Je veux gérer mes tournées de livraison »	Test des 4 questions : nouveau module, ou extension

11 Les 10 règles d'or

Cette page s'affiche au mur. Les règles marquées ⚙ sont vérifiées automatiquement en CI.

#	Règle
1	**Un seul code, une seule branche principale.** Jamais de branche ni de version par produit, par client ou par serveur.
2	**Un client = un abonnement + des réglages.** Seule la console NetByUs décide de ce qu'il a, par une licence signée ; le socle ne connaît jamais la console. Une demande propre à un client suit l'échelle de personnalisation ; jamais de code client dans un module. ⚙
3	**Un module = une fonctionnalité utilisable seule**, validée par le test des 4 questions. Même nom dans `api/modules` et `app/src/modules`.
4	**Un module ne touche jamais l'intérieur d'un autre module.** Il ne peut utiliser que la vitrine d'un module qu'il requiert. ⚙
5	**Requis → vitrine ; facultatifs l'un pour l'autre → événement ou pont.** ⚙
6	**Chaque table appartient à un module et porte son préfixe.** Clé étrangère seulement vers le socle ou un module requis. ⚙
7	**Tout ce qu'un module expose vérifie qu'il est actif** : routes, tâches, écouteurs, jobs. Le gabarit le fait pour vous. ⚙
8	**Ce qui est lu souvent est en cache ; ce qui est long part en file.**
9	**Le navigateur ne charge que les modules du client.** Un composant partagé va dans le kit d'interface du socle. ⚙
10	**Chaque module a son test d'activation** : client neuf, activation du module (et de ses modules requis), puis ses écrans répondent et ceux des autres modules sont refusés. ⚙

Situations courantes

Situation	Réponse
« On veut vendre une nouvelle fonctionnalité »	Test des 4 questions → nouveau module (dossier API + dossier web + manifeste) → on l'ajoute au catalogue de la console
« Un client prend un module de plus »	On l'ajoute à son abonnement dans la console ; sa licence signée part vers son serveur. Aucun développement
« Un client veut son propre serveur »	On installe BY+ sur son serveur, on l'enrôle dans la console, on y déploie son client ; ses modules arrivent par sa licence
« Ce client veut que ça marche un peu autrement »	Échelle de personnalisation : réglage, configuration, option pour tous, puis extension client
« Le module A a besoin d'une info du module B »	A requiert B ? → vitrine de B. Sinon → pont A-B, ou événement
« Faut-il déclarer B comme requis, ou faire un pont ? »	A a-t-il un sens sans B ? Non → requis. Oui → pont. Tableau « Requis ou pont » du chapitre 6
« Plusieurs modules ont besoin de ceci, mais personne ne l'achète seul »	Module de base (`sellable: false`), comme Catalogue ou Formulaires
« Tous les clients ont besoin de cette fonctionnalité »	Elle va dans le socle (décision d'architecture, notée dans `docs/`)

12 Redémarrer : dans quel ordre construire

Étape	Contenu	Terminé quand
1 · Le catalogue	Atelier produit, commercial et technique sur le « Catalogue des modules » : modules, exigences, ponts, offres	Le catalogue v1 est validé par produit et commercial
2 · Le socle et la console en cours	Base par client, connexion (base ou LDAP, double authentification), droits, sociétés, fichiers, notifications, journal ; registre des modules ; files d'attente ; console NetByUs (clients, abonnements, licences signées, instances) ; shell web, `/me/bootstrap`, kit d'interface ; CI avec tous les garde-fous. Détail et suivi des tâches : « La console NetByUs : plan de mise en place »	NetByUs crée un client dans la console, sur une instance séparée, lui attribue un abonnement, et le client s'y connecte
3 · Le module modèle	Un premier module complet de bout en bout (ex. Access ou Congés) ; gabarit `byplus:make-module` (et `make-bridge`, `make-extension`)	Un développeur qui ne connaît pas le projet crée un module en une journée
4 · Les modules, par priorité	Les plus demandés d'abord, avec leurs modules requis ; les ponts quand leurs deux modules existent. L'ancien BY+ sert de référence métier, pas de base à copier	Chaque module passe son test et est validé par un client pilote
5 · Les clients existants	Pour chaque module, un import depuis l'ancien BY+ ; pour chaque client, un abonnement dans la console ; bascule client par client ; reprise des ventes d'applications externes dans la console	Le dernier client est passé et l'ancien BY+ est arrêté
6 · La console en package	`netbyus/byplus-console` et `byplus:console:install`	La console s'installe en deux commandes sur le seul serveur NetByUs

13 Glossaire

Mot	Définition
Socle	Ce que tout client a toujours ; jamais vendu séparément
Module	Une fonctionnalité utilisable seule ; même nom dans l'API et le web
Module de base	Requis par plusieurs modules, jamais vendu seul (ex. Catalogue, Formulaires)
Manifeste	La carte d'identité d'un module (`manifest.php`)
Vitrine	Le dossier `Public/` d'un module : la seule partie que les autres peuvent utiliser
Pont	Relie deux modules facultatifs l'un pour l'autre ; actif si le client a les deux
Offre	Les modules choisis par un client, enregistrés dans son abonnement
Abonnement	Dans la console : les modules, utilisateurs et dates d'un client ; la source de vérité
Licence	L'abonnement d'un client mis en forme et **signé** par NetByUs, livré à son serveur ; seule NetByUs peut en créer
Instance	Un serveur BY+ : le serveur NetByUs, le serveur SaaS, ou un serveur dédié
Serveur dédié	Une instance pour un seul client, avec son propre domaine, pilotée par la console
Lien	La brique `Link` du socle : chaque instance vient chercher ses licences et rend compte à la console
Console NetByUs	Le back-office de NetByUs pour gérer clients, abonnements, licences, instances et tickets ; un package installé sur le seul serveur NetByUs
Réglage	Une option d'un module, activable client par client, sans code
Document personnalisable	Un document produit par une brique ou un module (mail, ticket, facture), déclaré dans son manifeste, dont chaque société peut changer la mise en forme sans code
Point d'extension	Une interface déclarée par un module dans `Public/Hooks/`, que l'on peut remplacer pour un client
Extension client	Un petit module propre à un seul client, qui se branche sur un module sans le modifier
Emplacement	Un endroit nommé de l'interface où un module, un pont ou une extension ajoute un composant
Shell	La coquille légère de l'application web, toujours chargée ; elle charge ensuite les modules du client