BY+ · SOCLE · MISE EN PLACE

Le socle BY+ : guide de mise en place

Ce qu'il faut pour commencer : les décisions de départ, la forme du code, les briques à construire dans l'ordre, et quand c'est terminé.

S'appuie sur « L'architecture de BY+ », le « Catalogue des modules », « Le socle de BY+ » et la version relue de « Le socle BY+, en clair ».
Octobre 2026 · version de référence pour le démarrage.

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

Objectif de la première phase : un socle minimal et une console NetByUs minimale, sur lesquels on pourra construire le premier module (le module modèle).

Terminé quand : NetByUs crée un client dans la console, lui attribue une licence vide, et un utilisateur de ce client se connecte sur `{client}.byplus.io`. Il voit ses sociétés, et ne peut rien atteindre qu'il n'a pas payé.

Phase	Contenu	Terminé quand
1. Socle minimal (ce document)	Noyau, Tenancy, Registre des modules, Companies, Countries, Currencies, Languages, Units, Sequences, Files, Audit, Auth, Notifications, console NetByUs minimale	Le critère ci-dessus
2. Outils de personnalisation	Customization, Circuits de validation, Dashboard, Subscription	Un module obtient champs personnalisés, validation et widget sans code dans le socle
3. Ouverture	Sync (mobile), Integrations (clés API, webhooks), Helpdesk (support)	Une application mobile se synchronise ; un webhook part signé ; un ticket se traite

Ensuite vient le **module modèle** (Access ou Congés), puis les modules offre par offre (architecture, chapitre 11).

2 Ce qui existe déjà dans le dépôt

Élément	État
Laravel 13, Docker (`make install` , `make doctor`), PostgreSQL, Redis, Mailpit	**En place**
Un client = une base, reconnu par son sous-domaine (`core/Tenancy`)	**En place**
Registre des modules : manifestes, exigences, activation, middleware `module:{clé}` , commandes `byplus:*`	**En place**
API sous `/api/v1` : routes centrales (`routes/api.php`) et routes client (`routes/tenant.php`) séparées	**En place**
Tests : registre, activation, routage (19 tests)	**En place**
Toutes les briques du socle hors Tenancy et Modules ; la console NetByUs	**À faire**

3 Les décisions de départ

Ces décisions permettent de commencer sans attendre. Celles marquées **à confirmer** doivent être validées à la réunion de lancement. Si elles changent, l'impact reste faible tant que peu de code est écrit.

#	Sujet	Décision de départ
D1	Nom des briques	**Anglais dans le code, français à l'écran** : Audit = « Journal », Workflows = « Circuits de validation », Helpdesk = « Support ». Le tableau du § 4 fait foi
D2	Étages	**4 étages (0 à 3)**, voir § 4. Une brique ne s'appuie que sur un étage plus bas. *À confirmer : retirer Sync → Auth pour revenir à 3 (le lien Notifications → Settings disparaît avec la brique Settings)*
D3	Préfixe des tables	`core_` pour toutes les tables du socle (`core_companies`), le préfixe du module pour un module (`pos_sessions`)
D4	Forme d'une brique	**La même que celle d'un module** : un dossier `Public/` (le guichet), tout le reste est privé. **Pas de dossier** `src/` : le chargement du code actuel (`Core\` → `core/`) suffit, sans configuration supplémentaire
D5	Fiche d'identité `brick.php`	**5 informations** : `key` , `name` , `description` , `owner` , `tables` (le préfixe). Pas de `api_version` tant qu'aucun besoin réel n'apparaît (un seul dépôt, une seule livraison)
D6	Accès des modules au socle	**Par le guichet** (`Public/`) pour tout ce qui agit : créer, modifier, envoyer, vérifier un droit. **Lecture directe autorisée** sur quelques modèles stables, pour les clés étrangères et l'affichage (User, Company, File). *À confirmer après le module modèle*
D7	Adresses de l'API des briques	Un `routes.php` par brique, servi sous `/api/v1/{clé}` sur le sous-domaine du client (ex. `/api/v1/companies`). Les adresses de la console sont à part, sur le domaine central
D8	Portes d'entrée	**Selon besoin** : une brique n'a que les dossiers qu'elle utilise
D9	Authentification	**Laravel Sanctum** : jetons pour l'application web et les applications mobiles. **Deux accès séparés** : utilisateurs des clients (base du client) et agents NetByUs (base centrale, console)
D10	Tenancy, Mon abonnement, Sync	**Ce sont des briques**
D11	Tests	*À confirmer* : **PHPUnit** (en place) ou **Pest** (prévu par l'architecture). À trancher avant d'écrire les tests du socle
D12	Recherche globale	**Hors périmètre** de la première phase

4 Les briques et leurs étages

Étage	Brique	Rôle	Phase
—	**Kernel** (noyau)	Contexte (client, utilisateur, société, acteur), outils communs. Ce n'est pas une brique	1
0	**Tenancy**	Retrouver le client à partir du sous-domaine, ouvrir sa base	1
1	**Companies**	Sociétés, agences, informations légales, fuseau horaire	1
1	**Countries**	Liste des pays	1
1	**Currencies**	Liste des devises	1
1	**Languages**	Liste des langues	1
1	**Units**	Liste des unités de mesure	1
1	**Sequences**	Numérotations des documents	1
1	**Files**	Stockage des fichiers (S3 / MinIO), pièces jointes, place utilisée	1
1	**Audit**	Journal des actions et des consultations	1
1	**Modules**	Registre des modules : licence, modules actifs, ponts, extensions	1
2	**Notifications**	Dans l'application, mail, push, SMS ; toujours en arrière-plan	1
2	**Auth**	Utilisateurs, rôles, droits, connexion	1
2	**Customization**	Champs personnalisés, listes de valeurs, modèles de documents, libellés	2
2	**Dashboard**	Accueil, widgets fournis par les modules	2
2	**Subscription**	Mon abonnement : montre la licence au client (lecture seule)	2
3	**Circuits de validation**	Validation en plusieurs étapes (dossier `Workflows/` dans le code)	2
3	**Sync**	Synchronisation mobile	3
3	**Integrations**	Clés API et webhooks du client	3
3	**Helpdesk**	Tickets du client vers NetByUs	3

Qui peut utiliser quoi :

- une brique utilise le **noyau** et le **guichet** des briques des étages plus bas ;
- un **module** utilise le socle (selon D6) et le guichet des modules qu'il **requiert** ;
- le **socle n'utilise jamais** un module, un pont ou une extension.

5 L'arborescence cible

`✔` existe · `○` à créer en phase 1 · `·` phases suivantes.

```
core-api/
├── core/                              LE SOCLE · namespace Core\
│   ├── Kernel/                    ○   noyau : Context, Actor, BricksServiceProvider (chargeur)
│   ├── Tenancy/                   ✔   Tenant.php (+ brick.php à ajouter)
│   ├── Modules/                   ✔   registre des modules (+ brick.php, guichet à ajouter)
│   ├── Companies/  Countries/  Currencies/  Languages/          ○   étage 1
│   ├── Units/  Sequences/  Files/  Audit/                       ○   étage 1
│   ├── Notifications/  Auth/                                    ○   étage 2
│   ├── Customization/  Dashboard/  Subscription/  Workflows/    ·   phase 2
│   └── Sync/  Integrations/  Helpdesk/                          ·   phase 3
├── platform/                      ○   console NetByUs (domaine central, base centrale)
├── modules/  bridges/  extensions/ ✔  vides pour l'instant
├── routes/api.php                 ✔   routes centrales /api/v1
├── routes/tenant.php              ✔   routes client communes /api/v1
├── config/byplus.php              ✔   préfixe api/v1, modules
├── database/migrations/           ✔   base centrale (tenants, domains, tenant_modules)
└── deptrac.yaml                   ○   vérification automatique des liens
```

6 Le modèle d'une brique

6.1 Les dossiers

Exemple avec Companies. Une brique a la **même forme qu'un module** : ce qui est dans `Public/` est le guichet, tout le reste est privé.

```
core/Companies/
├── brick.php                      fiche d'identité (5 informations)
├── CompaniesServiceProvider.php   chargé automatiquement par le noyau
├── routes.php                     /api/v1/companies (sous-domaine du client)
│
├── Public/                        LE GUICHET : la seule partie que les autres utilisent
│   ├── Contracts/                 CompanyDirectory, CurrentCompany
│   ├── Events/                    CompanyCreated…
│   ├── Data/                      objets de données en lecture seule
│   └── Concerns/                  BelongsToCompany (trait offert aux modules)
│
├── Actions/                       1 classe = 1 action métier (CreateCompany…)
├── Http/                          Controllers (fins), Requests, Resources
├── Models/                        Company, Office
├── Policies/                      les droits
├── database/migrations/           tables core_companies, core_offices (base du client)
├── lang/                          fr.json, en.json
└── tests/
```

6.2 La fiche d'identité

```
// core/Companies/brick.php
return [
    'key' => 'companies',                // adresses /api/v1/companies, noms de routes companies.*
    'name' => 'Sociétés',                // nom affiché
    'description' => 'Sociétés, agences, informations légales, fuseau horaire',
    'owner' => 'équipe socle',
    'tables' => 'core_',                 // préfixe de toutes ses tables
];
```

Une brique **n'a pas** de `requires`, de `sellable`, de `permissions` dans sa fiche, ni de contrôle d'activation : elle est toujours active. Ses dépendances découlent de son étage, et ses droits sont définis dans `Policies/`.

6.3 Le chargeur du noyau

Le noyau charge les briques comme le registre charge les modules. Il est **à écrire en premier** : `core/Kernel/BricksServiceProvider.php`, déclaré une fois dans `bootstrap/providers.php`. Pour chaque `core/*/brick.php`, il :

1. enregistre le provider de la brique (`{Nom}ServiceProvider`) ;
2. charge son `routes.php` sous `/api/v1/{clé}`, avec les middlewares `api`, client par sous-domaine, et refus du domaine central ;
3. ajoute ses `database/migrations/` aux migrations des bases client ;
4. charge ses traductions `lang/*.json`.

`Kernel/` n'a pas de `brick.php` : il n'est donc pas chargé comme une brique.

7 Le noyau (Kernel)

Élément	Rôle	Contenu de départ
Context	Ce que toute brique peut savoir de la requête en cours	client, utilisateur (ou aucun), société courante, acteur, `can($droit)`
Actor	Qui fait l'action ; jamais vide	`user` (une personne), `system` (une tâche planifiée, un traitement), `api_key` (un logiciel externe)
BricksServiceProvider	Le chargeur des briques (§ 6.3)	—
Outils communs	Ce que toutes les briques partagent	Contrôleur de base de l'API, format des réponses et des erreurs JSON, pagination

Le contexte est **rempli** par Tenancy (le client), Auth (l'utilisateur) et Companies (la société). Il est **lu** par tous les autres. Il est vidé à la fin de chaque requête et de chaque traitement, pour ne jamais mélanger deux clients (compatible Octane).

Il faut **deux contrôleurs de base** : un pour les adresses client (avec client et droits), un pour la console (domaine central, sans client).

8 Les briques de la phase 1, une par une

Les tables sont des **propositions de départ** : à affiner au moment de coder chaque brique.

8.1 Tenancy (étage 0)

- **Fait :** sous-domaine → client → base du client ; refus du domaine central ; 404 pour un sous-domaine inconnu.
- **À faire :**
  - ajouter `brick.php` ;
  - remplir le contexte avec le client ;
  - **retirer la relation** `Tenant::modules()`, qui importe `Core\Modules` (étage 1) depuis l'étage 0. C'est le registre qui lit `tenant_modules` à partir de l'identifiant du client.
- **Guichet :** `ClientResolver` (dans quel client on s'exécute).
- **Terminé quand :** deux appels simultanés de deux clients différents ne se mélangent jamais. Un test le prouve.

8.2 Modules — le registre (étage 1)

- **Fait :** manifestes, exigences, activation et désactivation, middleware `module:{clé}`, cache des modules actifs.
- **À faire :**
  - le cache Redis, avec un numéro de version incrémenté à chaque changement de licence ;
  - la classe de base `ModuleListener`, qui ignore un événement si le module n'est pas actif ;
  - le contrôle au démarrage des traitements en file d'attente ;
  - les tâches planifiées lancées seulement pour les clients qui ont le module ;
  - le chargement des **ponts** (`bridges/*/bridge.php`, actifs si les deux modules le sont) et des **extensions client** (`extensions/*/extension.php`, actives pour leur client seulement) ;
  - les commandes `byplus:make-bridge` et `byplus:make-extension`.
- **Guichet :** `ModuleContext` : `active($module)`, `modules()`, `bridgeActive($pont)`.
- **Terminé quand :** les 5 portes d'entrée sont gardées, un test par porte.

8.3 Companies (étage 1)

- **Tables :** `core_companies` (raison sociale, identifiants légaux, adresse, devise, fuseau horaire, logo) et `core_offices` (agences, établissements).
- **Guichet :** `CompanyDirectory` (sociétés accessibles, lire une société), `CurrentCompany` (société courante, fuseau), trait `BelongsToCompany`.
- **API :** `/api/v1/companies` : lister, créer, modifier, archiver.
- **Terminé quand :** un client gère ses sociétés, et la société courante (avec son fuseau) est dans le contexte.

8.4 Countries (étage 1)

- **Données :** la liste des pays (code ISO, nom, indicatif téléphonique). Un fichier livré avec le code, pas une table à saisir.
- **Guichet :** `Countries` (lister, lire un pays par son code).
- **API :** `/api/v1/countries` : lister.
- **Terminé quand :** une brique ou un module lit un pays par le guichet, et la liste est servie par l'API.

8.5 Currencies (étage 1)

- **Données :** la liste des devises (code ISO, nom, symbole, nombre de décimales). Un fichier livré avec le code, pas une table à saisir.
- **Guichet :** `Currencies` (lister, lire une devise par son code).
- **API :** `/api/v1/currencies` : lister.
- **Terminé quand :** une brique ou un module lit une devise par le guichet, et la liste est servie par l'API.

8.6 Languages (étage 1)

- **Données :** la liste des langues (code, nom). Un fichier livré avec le code, pas une table à saisir.
- **Guichet :** `Languages` (lister, lire une langue par son code).
- **API :** `/api/v1/languages` : lister.
- **Terminé quand :** une brique ou un module lit une langue par le guichet, et la liste est servie par l'API.

8.7 Units (étage 1)

- **Données :** la liste des unités de mesure (code, nom, famille : quantité, poids, longueur, volume, durée). Un fichier livré avec le code, pas une table à saisir.
- **Guichet :** `Units` (lister, lire une unité par son code).
- **API :** `/api/v1/units` : lister.
- **Terminé quand :** une brique ou un module lit une unité par le guichet, et la liste est servie par l'API.

8.8 Sequences (étage 1)

- **Table :** `core_sequences` : numérotations (clé, préfixe, modèle, prochain numéro, remise à zéro annuelle ou mensuelle).
- **Guichet :** `Sequencing` (prochain numéro d'une séquence déclarée par un module).
- **Terminé quand :** deux numéros demandés en même temps ne sont jamais identiques, et il n'y a pas de trou. Un test avec des appels simultanés le prouve.

8.9 Files (étage 1)

- **Prérequis :** ajouter **MinIO** (stockage compatible S3) aux conteneurs Docker.

- **Table :** `core_files` (identifiant, nom, type, taille, emplacement, objet auquel il est attaché, déposé par).
- **Guichet :** `Storage` (enregistrer, lire, supprimer, adresse temporaire), `Attachments` (attacher à un objet), trait `HasAttachments`. Un module ne manipule jamais un chemin de fichier : seulement un identifiant.
- **Terminé quand :** un fichier est stocké dans l'espace du client, servi par une adresse temporaire, compté dans la place utilisée, et supprimé avec son objet.

8.10 Audit (étage 1)

- **Table :** `core_audit_entries` (quand, acteur et son identifiant, action, objet concerné, changements, société, adresse IP). **Aucune colonne d'acteur vide.**
- **Guichet :** `Journal` (noter une action, lire l'historique d'un objet), trait `Journalizable`.
- **Terminé quand :** une tâche de nuit est journalisée comme `system`, et une consultation par le support est tracée.

8.11 Notifications (étage 2)

- **Tables :** `core_notifications` (dans l'application, lue ou non), `core_device_tokens` (téléphones pour le push).
- **Canaux :** dans l'application et mail en premier (Mailpit en local), puis push et SMS derrière une interface « fournisseur ».
- **Guichet :** `Notifier` (envoyer à un utilisateur ou à un groupe, à partir d'un modèle).
- **Règle :** l'envoi passe **toujours** en arrière-plan ; un canal en panne n'en bloque pas un autre.
- **Terminé quand :** une notification arrive dans l'application et par mail, sans ralentir la requête qui l'a déclenchée.

8.12 Auth (étage 2)

- **Tables (base du client) :** `core_users`, `core_roles`, `core_role_permissions`, `core_user_roles`, `core_user_companies` (sociétés accessibles), et les jetons Sanctum.
- **Droits :** chaque module déclare ses droits dans son manifeste (`permissions`), chaque brique dans ses `Policies`. Auth les rassemble ; elle n'en invente aucun.
- **API :** connexion et déconnexion, mot de passe oublié, `GET /api/v1/me`. Ensuite `GET /api/v1/me/bootstrap` : modules actifs, droits, compteurs.
- **Deux accès séparés :** les agents NetByUs se connectent à la console, avec leurs comptes dans la base centrale. Ils n'ont **aucun** compte dans les bases des clients.
- **À noter :** la migration `users` livrée par Laravel est dans la base **centrale**. Elle est à remplacer (comptes clients → `core_users` dans la base du client ; agents → console).
- **Terminé quand :** un utilisateur se connecte sur son sous-domaine, ne voit que ses sociétés, et chaque adresse vérifie ses droits.

8.13 La console NetByUs minimale (`platform/`)

Ce n'est pas le socle, mais la phase 1 en a besoin pour être testable :

- connexion des agents NetByUs (domaine central) ;

- créer un client : sa base et son sous-domaine ;
- lui attribuer une **licence** (offre + modules en plus, nombre d'utilisateurs, dates). La licence alimente le registre des modules, qui s'occupe de l'activation ;
- créer le premier administrateur du client.

9 L'outillage à mettre en place dès le début

Outil	Rôle	Quand
Deptrac	Refuse automatiquement les liens interdits (étages, socle → module…)	Avant la première brique
Larastan	Détecte les erreurs dans le code sans l'exécuter	Avant la première brique
Pint	Formate le code (déjà installé)	En place
Tests	PHPUnit ou Pest (D11), lancés à chaque modification	En place (PHPUnit)
Pipeline GitLab	Lance Pint, Larastan, Deptrac et les tests à chaque push ; refuse la fusion si l'un échoue	Avant la première brique
MinIO	Stockage des fichiers en local, compatible S3	Avant Files
Sanctum	Jetons de connexion	Avant Auth

Point de départ pour Deptrac : la configuration proposée en annexe de « Le socle BY+, en clair (version relue) ».

10 Les règles pour chaque modification du socle

À relire avant chaque fusion :

- ☐ Rien dans `core/` n'importe `Modules\`, `Bridges\` ou `Extensions\`.
- ☐ La brique n'utilise que le noyau et le guichet (`Public/`) des briques d'un étage plus bas.
- ☐ Toutes ses tables commencent par `core_`. Elle ne lit ni n'écrit dans les tables d'une autre brique : seulement des clés étrangères.
- ☐ Ce qu'elle offre aux modules est dans `Public/` ; tout le reste est privé.
- ☐ Toute action est journalisée avec un acteur (jamais vide).
- ☐ Ce qui est long (envoi, export, import) passe en arrière-plan ; ce qui est lu souvent est en cache.
- ☐ Aucun nom de client, aucune liste de modules ou d'offres écrite dans le code.
- ☐ Les tests passent, Deptrac et Larastan sont verts.

11 La liste de démarrage

Dans cet ordre, chaque étape s'appuyant sur la précédente.

Fondations

- [ ] Valider en réunion de lancement les décisions marquées « à confirmer » (§ 3)
- [ ] Installer Deptrac et Larastan ; créer le pipeline GitLab
- [ ] Écrire le noyau : `Context`, `Actor`, `BricksServiceProvider`, les deux contrôleurs de base
- [ ] Ajouter `brick.php` à Tenancy et à Modules ; retirer `Tenant::modules()`
- [ ] Remplacer la migration `users` centrale de Laravel
- [ ] Ajouter MinIO aux conteneurs ; installer Sanctum

Étage 1

- [ ] Registre des modules complet (cache Redis versionné, 5 portes gardées, ponts, extensions)
- [ ] Companies
- [ ] Countries
- [ ] Currencies
- [ ] Languages
- [ ] Units
- [ ] Sequences
- [ ] Files
- [ ] Audit

Étage 2

- [ ] Notifications
- [ ] Auth, puis `GET /api/v1/me/bootstrap`

Ensuite : le module modèle (Access ou Congés) et `byplus:make-module` mis à jour, puis la phase 2 du socle.