BY+ · RÉFÉRENCE · CONVENTIONS

BY+ : conventions

Comment on écrit le code de BY+ : PHP, règles d'architecture, front, traductions, langue, Git, tests et documentation.

Version 1.0 · 17 octobre 2026

Sommaire

1 La langue
Identifiants en anglais, textes et commentaires en français

2 Le code PHP
Pint, nommage, forme d'une brique et d'un module

3 Les règles d'architecture
Socle, modules, ponts, extensions, console, Deptrac

4 La base de données

5 L'API

6 Le front
Composants partagés, formulaires, chargement, confirmations, 375 px

7 Les traductions

8 Git

9 Les tests

10 La documentation

11 Avant chaque fusion

1 La langue

Élément	Langue	Exemples
Identifiants : classes, méthodes, variables, tables, colonnes, routes, droits, clés de traduction, clés de manifeste	Anglais	`CompleteSale`, `pos_checkouts`, `pos.sales.cancel`, `pos::messages.sale_completed`
Commentaires, docblocks, messages de commit, documentation	Français	« Le ticket photographie les prix… »
Textes affichés et messages de l'API	Français et anglais, par clé de traduction	« Annuler une vente » / « Cancel a sale »
Noms des briques	Anglais dans le code, français à l'écran	`Audit` = « Journal », `Templates` = « Modèles de documents »

Aucune phrase en dur dans le code, ni côté API ni côté front : tout texte affiché passe par une clé.

2 Le code PHP

2.1 Formatage et analyse

- **Pint** (configuration Laravel par défaut) : `make pint` formate, `vendor/bin/pint --test` vérifie (CI).
- **PHPStan / Larastan niveau 8** sur `app`, `core`, `modules`, `bridges`, `extensions`, `packages/byplus-console/src` ; les tests Feature et Unit ne sont pas analysés.
- Types partout (paramètres, retours, propriétés), `readonly` pour les dépendances injectées, docblocks de tableaux typés (`@return list<string>`, `array<string, mixed>`).

2.2 La forme d'une brique, d'un module, d'un domaine de la console

Dossier	Ce qu'on y met
`manifest.php`	La fiche d'identité. Brique : `key`, `name`, `description`, `owner`, `tables` (et `documents`, `sync`). Module : en plus `version`, `requires`, `sellable`, `permissions`, `settings`, `schedule`, `provider`. Pont : `key`, `name`, `when`, `provider` (et `sync`)
`{Nom}ServiceProvider.php`	À la racine, avec le manifeste : branchements, enregistrement des droits, des widgets, des flux de synchronisation
`Public/`	Le guichet (la vitrine) : `Contracts/` (interfaces), `Data/` (objets de données en lecture seule, énumérations publiques), `Events/`, `Hooks/` (points d'extension), `Concerns/` (traits offerts), `Exceptions/`. Seule partie que les autres utilisent
`Actions/`	Une classe = une action métier, nommée par un verbe : `CompleteSale`, `IssueCreditNote`, `OpenSaleSession`, `InviteUser`
`Services/`	Logique interne : implémentation des guichets, règles, calculs ; un sous-dossier par famille. Pas de dossier par sujet à la racine
`Providers/`	Branchements vers l'extérieur derrière un contrat du guichet (LDAP, WhatsApp, push), et ce que le module dépose chez le socle (`Providers/Sync`, `Providers/Dashboard`)
`Policies/`	Les droits : classe `{Nom}Rights` avec une constante par droit et `all()` (droit → clé de libellé), pour une brique
`Http/`	`Controllers/` fins (valider, autoriser, appeler l'action, rendre la ressource), `Requests/`, `Resources/`, `Middleware/`
`Models/`, `Jobs/`, `Console/`, `Mail/`, `Imports/`, `Exports/`	Selon besoin : un dossier n'existe que s'il sert
`routes/`	`tenant.php` (routes client), `api.php` (routes centrales)
`database/migrations/`, `lang/{fr,en}/`, `resources/views/`, `tests/`	Tables, traductions, vues Blade (documents par défaut, mails), tests de la brique ou du module

2.3 Nommage

Élément	Règle
Dossiers	PascalCase, même nom côté API et côté front : `modules/Pos`, `app/src/modules/Pos` ; ponts `bridges/PosEntrepot` (API), `src/bridges/pos-entrepot` (front)
Espaces de noms	`Core\{Brique}\…`, `Modules\{Module}\…`, `Bridges\…`, `Extensions\…`, `Platform\{Domaine}\…` (console)
Clé	Minuscules : `pos`, `entrepot`, `pos-entrepot`
Droits	`{clé}.{objet}.{action}` ou `{clé}.{action}` : `pos.sessions.open`, `catalogue.view`
Routes	Adresse `/api/v1/{clé}/…`, nom `{clé}.{objet}.{action}` (`auth.users.store`) ; ressources REST ; paramètres UUID contraints. L'application mobile : `routes/mobile.php` du propriétaire, sous `/api/v1/mobile/{clé}/…` (nom `mobile.{clé}.…`) ; jamais de module « mobile »
Tables	`{préfixe}_{pluriel}` : `pos_checkout_lines`, `core_users`
Événements	Au passé : `SaleCompleted`, `StockMoved`, `ContactArchived`
Commandes Artisan	`byplus:{sujet}:{action}` : `byplus:link:sync`, `byplus:module:enable`

2.4 Les outils du socle à utiliser

- Contrôleurs : étendre `TenantController` (adresses client) ou `CentralController` (console) ; réponses et erreurs au format commun.
- Société : trait `BelongsToCompany` ; contexte : `Context` (jamais de variable globale).
- Journal : guichet `Journal` (`record`, `consulted`) ; fichiers : guichet `Storage` et trait `HasAttachments` ; notifications : `Notifier` ; codes : `OneTimeCodes` ; documents : `Documents::render()`.
- Files : `Queue::Urgent|Notifications|Default|Heavy->assign($this)` ; tout ce qui est long part en file, ce qui est lu souvent est en cache.
- Écouteurs d'un module : étendre `ModuleListener` (`whenActive`).

3 Les règles d'architecture

Qui	Peut utiliser	Ne peut jamais utiliser
Le socle	Le noyau, et le guichet des briques d'un étage plus bas	Un module, un pont, une extension, la console. **Le socle ne nomme jamais un module** : il lit les manifestes (droits, documents, widgets, flux)
Un module	Le socle (guichets et modèles lisibles `User`, `Role`, `Company`, `Office`, `File`, `Tenant`) et le dossier `Public/` des modules qu'il déclare dans `requires`	L'intérieur d'un autre module ; un module non requis ; un pont ; une extension
Un pont	Le socle et le `Public/` de ses modules (`when`) ; il écoute leurs événements, remplit leurs points d'extension	L'intérieur de ses modules ; un autre pont. Il n'a ni droit, ni réglage, ni entrée de menu
Une extension client	Le socle et le `Public/` du module qu'elle complète (`Hooks/`)	L'intérieur du module
La console	Le noyau et les guichets du socle ; le guichet des autres domaines	Les tables d'un client ; le socle n'importe jamais `Platform\`

- **Requis ou pont** : A a-t-il un sens sans B ? Non : A requiert B (vitrine, clé étrangère permise). Oui : un pont `a-b` (identifiant simple sans contrainte). Jamais les deux. Deux niveaux de `requires` au plus, jamais de cycle.
- Trois façons de coopérer, pas d'autre : **événement**, **vitrine** d'un module requis, **pont**. À l'écran : des **emplacements** (`SlotOutlet`).
- Aucun nom de client, aucune liste de modules ou d'offres dans le code ; aucun `if` sur un client : un besoin propre suit l'échelle de personnalisation.
- **Deptrac** (`deptrac.php`) vérifie ces liens et les étages du socle ; une nouvelle brique s'inscrit à son étage, sinon l'analyse échoue. Un écart connu va dans `deptrac.baseline.yaml` (`make deptrac-baseline`) et se résorbe.

4 La base de données

- Chaque table appartient à une brique ou à un module et porte son préfixe ; un module ne lit ni n'écrit jamais dans les tables d'un autre : il passe par la vitrine.
- Clé primaire UUID dans les bases client ; `company_id` sur les objets d'une société ; pays et devise par leur code ; montants `decimal(15,4)` dans les modules, entiers en F CFA dans la console.
- Clé étrangère seulement vers le socle ou un module requis.
- On archive au lieu de supprimer ; ce qui doit rester tel qu'à l'opération est photographié.
- Une migration ne change jamais une table d'un autre propriétaire ; elle a un `down()` ; une migration sans effet gardée pour l'historique le dit dans son commentaire.
- Toutes les bases client ont le même schéma : pas de migration à l'activation d'un module, pas de table propre à un client.

5 L'API

- Sous `/api/v1`, en JSON. Réponses des contrôleurs client : `{ success, message, data, meta }` ; erreurs : `{ success: false, message, errors }` ; console : sans `success`.
- Listes paginées (`?per_page=`, `?q=`). Pour une même ressource : une **collection de choix** (l'essentiel, ouverte à qui voit l'objet parent) et un **tableau de gestion** `/datatable` (tout, protégé par le droit).
- Chaque action vérifie son droit ; une suppression d'un objet utilisé est un archivage ; un « supprimer un utilisateur » le désactive.
- Le front n'envoie jamais un montant qui fait foi : l'API recalcule.

6 Le front

6.1 Les règles

#	Règle
1	**Le noyau n'importe jamais un module, un pont ou une extension** (`src/core` → `@/modules/*` interdit). ESLint.
2	**D'un autre module, un module n'importe que la vitrine** (`@/modules/{Dossier}/public`), et seulement s'il le déclare dans `requires` ; ses propres fichiers en import relatif. ESLint et `boundaries.spec.ts`.
3	**Deux modules facultatifs l'un pour l'autre passent par un pont** (`src/bridges/{a-b}/manifest.ts` , `when`). Personne n'importe un pont.
4	**Les écrans se chargent à la demande** (`() => import(...)`), y compris ce qu'on dépose dans un emplacement.
5	**Le client vient toujours du sous-domaine**, jamais d'un paramètre ou d'un choix à l'écran.
6	**Un composant utile à plusieurs modules va dans `src/core/ui`** ; il est ajouté à la page « Composants » dans la même fusion.
7	**Chaque vue fonctionne à 375 px**, sur tablette et sur ordinateur, sans défilement horizontal de la page : grilles Bootstrap (`col-12 col-md-6`), tableaux dans `table-responsive` , boutons et menus utilisables au doigt. Elle est vérifiée en largeur mobile avant d'être livrée.
8	**Tout champ a un texte d'exemple** (`placeholder`) et passe par les composants de `src/core/ui/components/form` .
9	**Une page qui attend l'API montre un squelette** (`src/core/ui/components/skeleton`), jamais un écran vide ni une roue seule.
10	**Toute confirmation passe par `confirmAction`** (`src/core/ui/alert.ts` , SweetAlert2), **jamais `window.confirm`** .
11	**Les actions d'une ligne de tableau sont grises** (`btn-secondary`), en icône seule avec un titre (`title`).
12	**Les droits** : `meta.permission` sur une route, `permission` sur une entrée de menu, `session.can(...)` sur un bouton. C'est un confort : l'API vérifie toujours.

6.2 Les composants partagés (`src/core/ui`)

Composant	Usage
`form/FormInput` , `FormTextarea` , `FormSelect` (Select2), `FormPhone` (indicatif international), `FormAmount` , `FormNumber` , `FormDate` , `FormDateRange` , `FormCheckbox` , `FormSwitch` , `FormRadio` , `FormTags` , `ImageInput` , `FileDrop` , `OtpInput` , `AsyncPicker` , `FieldActionButton` , `RequiredText`	Tous les formulaires

Composant	Usage
`skeleton/SkeletonBlock`, `SkeletonTable`, `SkeletonForm`, `SkeletonCards`, `GlobalLoader`, `RouteProgress`, `ProgressBar`	Chargement
`PageHeader`, `DetailLayout`, `PropertyGrid`, `DataTable`, `ViewToggle`, `EmptyState`, `StatusBadge`, `StepperNav`, `BaseModal`, `DropdownMenu`, `BackButton`, `InfoAlert`, `ApexChart`, `StatRanking`, `AddToDashboardButton`, `SlotOutlet`	Mise en page, listes, fiches, graphiques, emplacements
`format.ts`, `download.ts`, `alert.ts`	Montants, nombres, dates, tailles ; fichiers de l'API avec le jeton ; confirmations. Aucun module n'écrit son propre formatage

6.3 Le style

- Template Metronic de core-app (`public/assets/metronic/` , code tiers, jamais modifié) ; classes Metronic et Bootstrap ; icônes Font Awesome et Bootstrap Icons. Thème d'un client : `src/core/ui/themes/{client}.css` .
- Prettier : sans point-virgule, guillemets simples, 100 colonnes (`npm run format`).
- TypeScript strict (`vue-tsc`), Composition API (`<script setup lang="ts">`), stores Pinia propres au module.

7 Les traductions

	API	Front
Fichiers	`{brique ou module}/lang/fr/*.php` et `lang/en/*.php` : `messages.php`, `rights.php`, `notifications.php`, `documents.php` … Pas de JSON. À la racine, seulement `lang/fr/validation.php`	`src/core/i18n/lang/{fr,en}.json` et `src/modules/{Nom}/lang/{fr,en}.json`, chargés avec le module
Clés	Courtes, namespacées par la clé du propriétaire : `__('auth::messages.profile_saved')`, `kernel::` pour le noyau. Jamais une phrase comme clé	Rangées sous la clé du module : `t('pos.menu.shops')` ; le caractère `@` s'écrit `{'@'}`
Langue	En-tête `X-localization`, sinon `Accept-Language` ; `fr` par défaut. Les mails déclenchés par une personne partent dans la langue de son écran ; les autres dans celle du destinataire	Choix de la personne, sinon langue du compte, du navigateur, puis le français ; envoyée à chaque appel
Contrôle	Chaque clé existe en `fr` et en `en`	`i18n.spec.ts` vérifie que fr et en ont les mêmes clés

Libellés des droits : le manifeste relie la clé du droit à sa clé de libellé (`'pos.sales.cancel' => 'pos::rights.sales_cancel'`) ; titre du groupe : `{module}::rights.title`. Libellés des énumérations et des documents : dans les traductions, jamais dans le code.

8 Git

- **Branche de travail et de déploiement :** `new-develop`, pour l'API (core-api) et le front (core-app). Une seule branche principale : jamais de branche par client, par produit ou par serveur.
- Le travail se fait sur une branche `Feature/…`, fusionnée dans `new-develop` par merge request ; le pipeline doit passer avant la fusion.
- **Les commits sont faits par l'équipe**, sous son nom. Messages en français, préfixés par la partie touchée : « Brique Auth : guichets DeviceSessions et Accounts », « Brique Sync : … », « Caisse : … », « Console : … ».
- Aucun secret, aucun `.env`, aucune sauvegarde de base dans Git.

9 Les tests

- **API** : PHPUnit, dans le dossier `tests/` de chaque brique, module, pont et domaine de la console ; `tests/` à la racine pour le routage, le chargement des briques, le contexte et les portes de la console.
- **Test d'activation** de chaque module : client neuf, activation (avec ses modules requis), ses écrans répondent, ceux des autres sont refusés. Un pont : trois tests (A seul, B seul, A et B avec le pont).
- Chaque lot est vérifié sur une vraie base client PostgreSQL avant d'être fusionné. Les tests ne dépendent pas du `.env` local (adresses fixées dans `phpunit.xml`).
- **Front** : Vitest (`npm test`) ; tests du noyau dans `src/core/__tests__`, des modules dans leur dossier `__tests__`.
- Tout le contrôle en une commande : `make check` (API : Pint, PHPStan, Deptrac, tests) ; `npm run lint`, `npm run type-check`, `npm test` (front).

10 La documentation

- Source en HTML dans `api/docs/sources/`, PDF du même nom dans `api/docs/`, produit par Chrome sans en-tête ni pied de page :

  ```
  google-chrome --headless --no-pdf-header-footer \
    --print-to-pdf=api/docs/{nom}.pdf api/docs/sources/{nom}.html
  ```

- Même gabarit pour tous les documents : CSS d'impression A4, page de titre avec date et version, sommaire, chapitres numérotés, tableaux, badges d'état (« Fait », « À faire »).
- **Un seul document par sujet** : on met à jour le document existant au lieu d'en créer un autre. Un document décrit l'état réel et les règles en vigueur.
- Le suivi des tâches de la console (chapitre 18 de `console-mise-en-place`) et le plan d'implémentation sont mis à jour dans la même livraison que la tâche.
- Les fichiers `README.md` de chaque dépôt décrivent le démarrage et les commandes ; `install.md` décrit l'installation des serveurs.

11 Avant chaque fusion

#	Vérification
1	Rien dans `core/` n'importe `Modules\`, `Bridges\`, `Extensions\` ou `Platform\` ; un module n'utilise que le socle et le `Public/` des modules qu'il requiert.
2	Toutes les tables portent le préfixe de leur propriétaire ; clés étrangères seulement vers le socle ou un module requis.
3	Chaque adresse vérifie son droit ; chaque action importante est journalisée avec un acteur.
4	Ce qui est long part en file ; ce qui est lu souvent est en cache ; les listes sont paginées.
5	Aucun secret en clair, aucun nom de client, aucune liste de modules dans le code.
6	Tout texte affiché existe en français et en anglais, par clé.
7	Chaque vue nouvelle ou modifiée a été vérifiée à 375 px ; champs avec placeholder ; squelette ; `confirmAction`.
8	`make check` vert ; `npm run lint`, `npm run type-check`, `npm test` verts.
9	Documentation et plan mis à jour, PDF régénéré.