V2 — les API à faire

Ce qui vient après la V1 (R1 à R4), côté API, lot par lot : les adresses, les tables, les règles. Dans l'ordre conseillé. Préfixe des adresses : `/api/v1/restaurant`, sauf mention.

> **Avant de commencer** : R1 à R3 sont faits (branche `Feature/api/newdev/RestoPos`) ; R4 (QR de table, appels) doit être fusionné et relu. La V2 réutilise ce qui existe : le jeton du QR (R4), le guichet `PosOrders`, l'envoi aux postes (`StationDispatcher`), les accès (`shopOf()`, `authorizeInShop()`), le temps réel de la salle.

#	Lot	En une ligne	À cadrer ?
V2.1	Commande par le client (panier du QR)	Le client compose sa commande sur la page du QR ; le serveur la valide ; elle part au bar et en cuisine	Non
V2.2	Statistiques et widgets	Couverts, ticket moyen, temps de préparation, annulations, chiffre par serveur, sur le tableau de bord	Non
V2.3	Addition séparée	Une table paie en plusieurs fois : par articles ou par parts	Oui (encaissement)
V2.4	Réservations et horaires	Réserver une table à une heure ; horaires d'ouverture du restaurant	Oui (contacts)
V2.5	Reprise de l'ancien projet	Commande `byplus:import:restaurant` depuis `shop/api`	Oui (ancienne base)
V2.6	Annulation partielle d'une ligne	Annuler une des deux bières, pas la ligne entière	Non

1 V2.1 — Commande par le client (panier du QR)

Le client, sur la page du QR (sans connexion, comme R4), ajoute des plats avec leurs options et envoie sa commande. Elle **n'est pas envoyée aux postes** tant qu'un serveur ne l'a pas validée. Seulement si le restaurant l'a permis : paramètre `self_order` (existe déjà dans `restaurant_settings`, faux par défaut).

Table

- `restaurant_self_orders` : `shop_id`, `table_id`, `order_id` (la commande de la table une fois validée), `status` (`pending`, `accepted`, `rejected`), `items` (JSON : élément de vente ou formule, quantité, options avec `option_group_id`, commentaire), `amount_ttc` (calculé à l'envoi, pour information), `handled_by`, `handled_at`, `reject_reason`, timestamps.

Adresses

POST `/public/tables/{token}/orders`

Client · sans connexion · nombre de requêtes limité · seulement si `self_order`

- Envoi : `{ items: [{ product_sale_element_id | shop_bundle_id, quantity, comment?, children: [{ option_group_id, product_sale_element_id, quantity }] }] }` (même forme qu'une commande de table).
- Vérifié comme à la caisse : article en vente et en ligne dans ce point de vente (`ShopMenu`), options proposées (`LineOptions`), quantités raisonnables (max 20 lignes, 20 par article).
- Une seule commande en attente par table : une deuxième remplace la première tant qu'elle n'est pas validée.
- Prévient les serveurs de la zone (notification, temps réel sur le canal de la salle : `restaurant.self_order.changed`), pastille `restaurant.self_orders_pending` .

GET `/public/tables/{token}/orders/current`

Client · sans connexion

- Où en est sa commande : en attente, acceptée (et ce qui est parti en cuisine), refusée (motif). Rien d'une autre table.

GET `/shops/{shop}/self-orders`

Serveur · `restaurant.orders.take` dans le point de vente

- Les commandes en attente (`?mine=1` : celles des zones du serveur), avec la table, les articles et leur prix.

POST `/self-orders/{selfOrder}/accept` · **POST** `/self-orders/{selfOrder}/reject`

Serveur · `restaurant.orders.take` dans le point de vente

- **Accepter** : ouvre la commande de la table si elle n'en a pas (`OpenTableOrder`), ajoute les articles (`SaveOrderLines`), puis les envoie aux postes (`SendOrderToStations`). Le serveur peut d'abord retirer un article. Sous verrou : deux serveurs n'acceptent pas deux fois.
- **Refuser** : avec un motif, que le client voit.

Le compte client (connexion, inscription : module « Portail client ») n'est **pas** nécessaire : le jeton du QR suffit, comme en R4. À ne faire que si une commande à emporter ou une fidélité le demande plus tard.

2 V2.2 — Statistiques et widgets

Les chiffres du Restaurant sur le tableau de bord d'accueil, comme le Point de vente (`WidgetRegistry` , voir `PosServiceProvider`). Les données existent déjà : commandes (`restaurant_orders` : couverts, serveur, montant payé, ouverture, fermeture) et tickets (`restaurant_tickets` : envoi, prêt, servi, annulations).

Widget (clé `restaurant.*`)	Contenu
`restaurant.key_figures`	Sur la période : couverts, tables encaissées, chiffre, ticket moyen par couvert, temps moyen d'une table (ouverture → encaissement).
`restaurant.sales_by_waiter`	Chiffre, tables et couverts par serveur.
`restaurant.preparation_times`	Temps moyen de préparation par poste (envoi → prêt), tickets en retard (délais du restaurant).
`restaurant.cancellations`	Articles annulés après envoi : nombre, motifs les plus fréquents, par serveur.

Widget (clé `restaurant.*`)	Contenu
`restaurant.peak_hours`	Couverts par heure et par jour de la semaine.

- Réglage de chaque widget : un restaurant ou tous ceux visibles, la période (comme les widgets de vente).
- Droit : `restaurant.floor.view`, ou un droit de statistiques à ajouter au manifeste (`restaurant.statistics.view`) — à décider.
- Requêtes groupées (une par widget), pas de calcul par commande.

3 V2.3 — Addition séparée (à cadrer)

Une table paie en plusieurs fois. Deux façons : **par articles** (chacun paie ce qu'il a pris) ou **par parts** (3 parts égales).

- **Par articles** : déplacer les lignes choisies vers un nouveau bon (`PosOrders::moveLines` , déjà là), que la caisse encaisse ; la commande de table garde plusieurs bons. Il faut que `restaurant_orders` puisse porter plusieurs bons (une table `restaurant_order_checkouts`) et que la table ne se libère qu'au dernier encaissement.
- **Par parts** : touche la facture (une facture avec plusieurs paiements partiels dans le temps, ou plusieurs factures d'un même bon) : **à décider avec le Point de vente et la normalisation DGI** avant d'écrire du code.

Adresses envisagées : `POST /shops/{shop}/orders/{order}/split` (`{ line_ids }` → un nouveau bon), `GET /shops/{shop}/orders/{order}/bills` (les bons de la table et ce qui reste à payer).

4 V2.4 — Réservations et horaires (à cadrer)

- Table `restaurant_reservations` : `shop_id` , `table_id` (facultative), `contact_id` (si lié aux contacts : pont `restaurant-contacts`) ou nom et téléphone, `covers` , `starts_at` , `duration_minutes` , `status` (`booked` , `seated` , `cancelled` , `no_show`), `note` .
- Adresses : `GET·POST /shops/{shop}/reservations` , `PUT·DELETE /reservations/{reservation}` , `POST /reservations/{reservation}/seat` (ouvre la table avec les couverts).
- La table passe `reserved` (état déjà prévu) avant l'heure ; le plan de salle l'affiche.
- Horaires d'ouverture : `restaurant_opening_hours` (jour, ouverture, fermeture) ; `GET·PUT /shops/{shop}/opening-hours` ; la page du QR peut dire « fermé ».
- À décider : réservation par le client lui-même (page publique) ou seulement par le personnel.

5 V2.5 — Reprise de l'ancien projet (à cadrer)

- Commande `php artisan byplus:import:restaurant --tenant=… --source=…` , sur le modèle des repreneurs du Point de vente (`byplus:import:data`).
- Reprend : salles et tables, plats vers le Catalogue (s'ils n'y sont pas), accompagnements et suppléments vers `restaurant_option_*` , postes (`cooking_place` BAR / KITCHEN de l'ancien projet) et routage.
- Ne reprend pas : les commandes passées (elles restent dans l'ancien système), sauf décision contraire.
- Essai à blanc (`--dry-run`), journal de ce qui est laissé de côté, rejouable sans doublon.

6 V2.6 — Annulation partielle d'une ligne

POST **`/shops/{shop}/orders/{order}/lines/{line}/cancel`** (existe : ajouter `quantity`)

Serveur · `restaurant.orders.cancel`

- `{ reason, quantity? }` : sans quantité, toute la ligne (comme aujourd'hui) ; avec, seulement ce nombre.
- Demande au Point de vente un guichet pour réduire la quantité d'une ligne de bon (`PosOrders::reduceLine` , options comprises au prorata), puis un ticket « ANNULÉ » avec la quantité annulée.

Pas en V2

- L'application du serveur hors ligne, les fiches techniques (ingrédients) : « plus tard » dans le document.
- La caisse mobile (application) : on n'y touche pas.

Pour chaque lot : tests (droits, autre point de vente, concurrence), Pint, PHPStan, Deptrac, `tenants:migrate` sur la vraie base, et la doc (`pos-restauration` § 9 et 13, App Flow § 7) mise à jour.