StoreFlow — Developer Starter Kit

1. Purpose of This Starter Kit

This Developer Starter Kit explains HOW to turn the StoreFlow A1.x specification docs into a working Laravel + Vue application. It is written for an experienced developer or an AI assistant like Claude that already has access to:

- A1.1 – Architecture & System Design (Parts 1–3)
- A1.2 – Data Model & SQL Specification (Parts 1–5)
- A1.3 – Authentication, Authorization & Security Deep Dive
- A1.4 – Shipping Engine & Fulfilment Workflows (Parts 1–3)
- A1.5 – Dashboard, Storefront, UX & Accessibility Deep Dive
- A1.6 – API Specification & Endpoint Contract

The aim is to provide a clear implementation sequence, project structure, and key decisions so there is no ambiguity when generating code.

2. High-Level Tech Stack Recap

Backend:

• PHP 8.2+

• Laravel 10+

• MySQL 8

• Redis (queues + cache, in production)

• Laravel WebSockets (for real-time updates)

• Stripe Connect (prepared but not yet wired)

Frontend:

• Vue 3 (Composition API or Options API, but be consistent)

• Inertia.js (for dashboard rendering)

• Tailwind CSS

• Vite bundler

Local Dev Environment (User-specific):

• XAMPP for local MySQL + Apache (or use Laravel's artisan serve)

• Node.js + npm/yarn for frontend build

• No Docker in dev (per user decision)

3. Project Initialization – Step-by-Step

1) Create Laravel Project:

laravel new storeflow

or

composer create-project laravel/laravel storeflow

2) Configure .env:

• DB_CONNECTION=mysql

• DB_HOST, DB_PORT, DB_DATABASE, DB_USERNAME, DB_PASSWORD (use XAMPP values)

• QUEUE_CONNECTION=database for early dev (Redis later)

• SESSION_DRIVER=file or database for dev

• APP_ENV=local, APP_DEBUG=true

3) Install Frontend Stack:

• php artisan breeze:install vue (or inertia stack) – or install Inertia + Vue manually

• npm install

• npm run dev

4) Basic Git Structure:

• Initialize git repo

• Commit initial Laravel skeleton

• Create branches for major modules (auth, products, orders, shipping, etc.)

4. Folder Structure & Namespacing

Use standard Laravel structure with domain-oriented organization inside app/:

app/

Models/

Merchant.php

Store.php

User.php

Customer.php

Product.php

CustomizationGroup.php

CustomizationOption.php

Order.php

OrderItem.php

OrderItemOption.php

ShippingZone.php

ShippingMethod.php

ShippingRate.php

LoyaltyConfig.php

LoyaltyAccount.php

AuditLog.php

Http/

Controllers/

Dashboard/

DashboardController.php

OrdersController.php

ProductsController.php

ShippingController.php

LoyaltyController.php

CustomersController.php

AuditLogsController.php

Storefront/

StorefrontController.php

CheckoutController.php

TrackingController.php

Auth/

MerchantAuthController.php

CustomerAuthController.php

Middleware/

TenantMiddleware.php

RoleMiddleware.php

Services/

Orders/

OrderService.php

Shipping/

ShippingEngine.php

Products/

ProductService.php

Loyalty/

LoyaltyService.php

Policies/

OrderPolicy.php

ProductPolicy.php

StorePolicy.php

UserPolicy.php

Events/

OrderCreated.php

OrderStatusUpdated.php

ShippingStatusUpdated.php

Listeners/

BroadcastOrderEvents.php

WriteAuditLog.php

resources/

js/

Pages/

Dashboard/

Storefront/

Components/

css/ (Tailwind)

5. Migration & Model Implementation Strategy

Follow this order to avoid foreign key issues:

1) merchants, stores, users, store_users

2) customers

3) products, customization_groups, customization_options

4) orders, order_items, order_item_options

5) shipping_zones, shipping_methods, shipping_rates

6) loyalty_config, loyalty_accounts

7) audit_logs

For each table:

• Use A1.2 Data Model & SQL Specification for exact columns and types.

• Implement migrations with foreign keys and appropriate onDelete rules.

• Implement Eloquent models with:

- fillable or guarded attributes

- relationships (belongsTo, hasMany, etc.)

- casts for JSON fields (e.g., meta_json, reward_json)

- scopes for merchant + store filtering where helpful.

Example:

• Order model:

- belongsTo Merchant, Store, Customer

- hasMany OrderItems

• Product model:

- belongsTo Merchant (and optionally Store)

- hasMany CustomizationGroups

6. Implementing Tenancy & Scoping

Tenancy is critical and should be implemented early.

Steps:

1) Add merchant_id and store_id fields to all relevant tables per A1.2.

2) Implement TenantMiddleware:

• Reads merchant_id and store_id from session (set post-login).

• Ensures store belongs to merchant.

• Attaches them to request context (e.g., request()->attributes).

3) In Controllers:

• Never accept merchant_id or store_id directly from request payload.

• Always use currently-bound merchant/store from middleware.

4) In Models or Repositories:

• Add scoped queries like:

->where('merchant_id', $currentMerchantId)

• Optionally use global scopes for multi-tenant filtering, but document them clearly.

5) Use Policies to enforce:

• user->merchant_id === model->merchant_id

• For store-specific entities: user is owner OR user is attached to the store.

7. Authentication & Roles Implementation

Merchant Auth:

• Use Laravel's default 'web' guard with users table.

• Implement login controller (MerchantAuthController) using username + password.

• On login success:

- Identify merchant_id from user.

- If user has 1 store: set store_id in session.

- If multiple: redirect to store-selection page and then set store_id.

Roles:

• user.role in ['owner','manager','staff'] handles merchant-level role.

• store_users table handles store-specific roles for managers/staff.

• Policies check both user.role and store_users mapping.

Customer Auth:

• Separate 'customer' guard, using customers table.

• Optional at first; guest checkout is default.

• For now, only wire minimal login/register logic according to A1.3.

Rate Limiting:

• Use Laravel's throttle middleware on login routes (see A1.3).

8. Building the Storefront

Goal: Implement a minimal but functional storefront following A1.5.

Steps:

1) Routes:

• GET /s/{store_slug} – storefront home

• GET /s/{store_slug}/p/{product} – product detail (if needed)

• GET /s/{store_slug}/cart – cart page

• POST /api/v1/stores/{id}/shipping/quote – shipping API

• POST /api/v1/stores/{id}/checkout – checkout API

• GET /o/{public_id} – order tracking page

2) Theme Handling:

• Theme selection stored in stores.theme_key.

• In Blade/Vue layouts, include CSS bundle based on theme_key.

3) Cart Implementation:

• Use local storage or session for cart data.

• On checkout, send final items and shipping selection to API.

4) Order Tracking:

• Implement tracking page that subscribes to customer.{public_id}.order channel.

• Use A1.4 & A1.6 tracking endpoint contract for data.

9. Building the Dashboard (Vue + Inertia)

Goal: Implement Operations and Management zones per A1.5.

Steps:

1) Set up Inertia:

• Install Laravel + Vue + Inertia stack.

• Configure Inertia middleware and basic layout component.

2) Operations Page:

• Route: GET /dashboard

• Controller: DashboardController@index returns:

- active orders grouped by status

• Vue Page: Dashboard/Operations.vue

- columns for pending/accepted/in_progress/ready/packing/shipped

- each order card opens a modal for details & actions

• Status updates:

- PATCH /dashboard/orders/{id}/status (see A1.6)

- On success, update UI immediately.

3) Management Sections:

• Products, Shipping, Loyalty, Customers, Audit Logs.

• Each section uses minimal table + modals pattern.

• Use Inertia partial reloads for CRUD responses.

4) Reuse Components:

• Create reusable order card, order detail modal, product form, zone/method/rate forms, etc.

10. WebSockets & Real-Time Wiring

Real-time integration follows A1.1 and A1.4.

Steps:

1) Install laravel-websockets or Pusher-compatible stack.

2) Configure broadcast driver (local: log or pusher-like, prod: websockets).

3) Implement events:

• OrderCreated

• OrderStatusUpdated

• ShippingStatusUpdated

4) Implement broadcast channels:

• store.{store_id}.orders (auth: merchant user with store access)

• customer.{public_id}.order (auth: check public_id belongs to order)

5) Frontend:

• Install Laravel Echo.

• For dashboard Operations page:

- Subscribe to store.{store_id}.orders

- On event, update or insert orders into active list.

• For order tracking page:

- Subscribe to customer.{public_id}.order and update status display.

Ensure:

• Real-time failures do not affect core order persistence.

• Broadcasting happens after DB commit.

11. Stripe Connect Preparation (But Not Yet Active)

Per requirements, payment is treated as always successful in v1, but the system must be ready for Stripe Connect Standard integration.

Preparation Steps:

1) Add columns in merchants table:

• stripe_account_id VARCHAR(255) NULL

2) In Order model:

• payment_status ENUM, payment_reference field ready.

3) Implement placeholder services:

• PaymentService with methods:

- createTestPaymentIntent(order)

- markAsPaid(order)

• For now, these simply set payment_status = 'paid'.

4) Separate configuration:

• Put STRIPE_KEY and STRIPE_SECRET in .env but do not call them yet.

5) Document where Stripe will be wired later (checkout, webhooks, payouts).

12. Testing Strategy & What Claude Should Generate First

Recommended Testing Layers:

• Feature tests:

- Checkout flow (public)

- Order lifecycle (dashboard)

- Shipping quote resolution (API)

• Unit tests:

- ShippingEngine

- OrderService (status transitions)

- Tenancy scoping functions

What Claude should generate FIRST:

1) Migrations + Models for all entities (A1.2)

2) TenantMiddleware + base auth (A1.3 & A1.1)

3) Basic Storefront checkout + shipping quote endpoints (A1.4 & A1.6)

4) Basic Dashboard Operations page with manual status update (A1.5 & A1.6)

5) ShippingEngine service (A1.4)

6) WebSocket events and frontend subscriptions (A1.4)

After that, layer in:

• Products & customizations management

• Shipping configuration UI

• Loyalty & audit log UIs

• Customer login/registration (optional)

This Starter Kit should be read alongside the A1.x PDFs. Claude can be instructed to load specific PDFs (architecture, data model, API spec) and generate code module-by-module according to the sequence described here.