Amazon Store

Multi-Seller Marketplace

BACKEND – NO FRONTEND UI

Product Backlog · Architecture · Data Model

Airhacks Workshop — February 26th, 2026

Stack: Java 25 · Quarkus 3.x · MicroProfile 10 · Hibernate ORM + Panache · SmallRye JWT · PostgreSQL 16

Architecture: BCE / ECB Pattern · Modular Monolith → Microservices

Product Manager / Product Owner: Mohamed

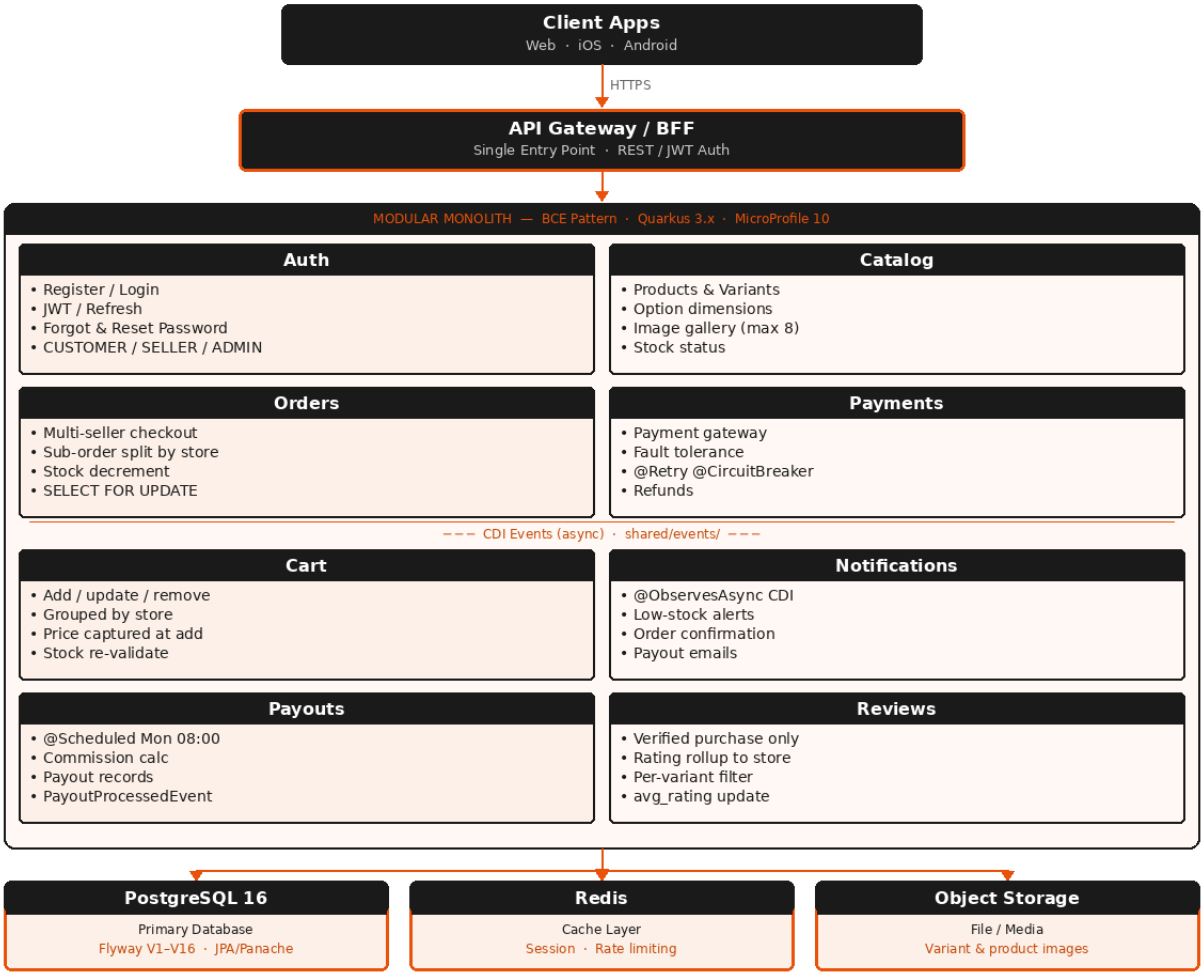
Platform Overview

The Amazon Store is a Multi-Seller Marketplace where independent sellers run branded stores, customers shop across all stores in a single experience, and the platform earns commission on every sale. Products have variants (Size, Colour, Material, Storage, or custom seller-defined options). Each variant is an independently SKU with its own price, stock level, and image gallery. The platform is built as a Modular Monolith using BCE, designed for future microservice extraction.

Product Decisions

Decision	What it means for the build
✅ **Variants: Size, Storage, Material + Custom**	Sellers define their own variant dimensions. Each variant combination = one SKU with its own stock, price, and image gallery.
✅ **Variant Images: Multi-image per variant**	Each variant supports up to 8 images (primary + gallery). Variants inherit product images when no own images are defined.
✅ **Out-of-stock**	In-stock: selectable. OOS: greyed out. All variants OOS: 'Currently unavailable'. Low stock (≤5): 'Only N left in stock'.
✅ **Stock: Auto-decrement + low-stock seller alert**	Stock decrements atomically on order confirmation. Seller receives low-stock notification when variant stock falls to or below the threshold (default: 5 units).
✅ **Checkout: One order, platform splits internally**	Customer pays once. Orders control groups cart items by store_id and creates one SubOrder per seller automatically.
✅ **Seller onboarding: Self-service, live immediately**	POST /api/v1/auth/register/seller creates User (SELLER) + Store (ACTIVE) atomically. JWT issued immediately. No admin approval queue.
✅ **Workshop scope: Store profile, Payouts, Dashboards**	Store branding, commission + weekly payout CRON, seller dashboard, admin marketplace dashboard.

System Architecture



Roles & Permissions

Role	Who	Can Do	Cannot Do
CUSTOMER	Registered buyer	Browse all stores, search products, manage cart (multi-store), checkout, pay, track orders per seller, review purchased products, reset forgotten password	See other customers' data, manage products, access seller financial
SELLER	Self-registered shop owner	Manage own Store profile & branding, CRUD own products & variants, view own SubOrders, ship orders, view commission & payouts, view seller dashboard, reset forgotten password	See other sellers' products or orders. Modify own commission rate. Access platform-wide metrics.
PLATFORM_ ADMIN	Platform operator	View all stores/orders/products, set commission rates, suspend/reinstate sellers, resolve disputes, access admin dashboard and full GMV metrics	Place orders on behalf of customers, alter payout history without audit trail

Data Model

All Tables

Table	Key Columns	Notes
users	id UUID PK, email UNIQUE, password_hash, password_reset_token VARCHAR(255), password_reset_expires_at TIMESTAMP, first_name, last_name, role DEFAULT 'CUSTOMER', active BOOL, created_at	password_reset_token: short-lived random token (expires 1h). Cleared after use. Hashed before storage.
stores	id UUID PK, seller_id FK→users UNIQUE, name, slug UNIQUE, description, logo_url, banner_url, status DEFAULT 'ACTIVE', commission_rate NUMERIC(5,2) DEFAULT 10.00, avg_rating, created_at	Self-service: created ACTIVE on seller register. PLATFORM_ADMIN can suspend.
categories	id UUID PK, name, slug UNIQUE, parent_id FK→categories, created_at	Hierarchical. Parent_id NULL = top-level.
products	id UUID PK, store_id FK→stores NOT NULL, category_id FK→categories, name, description, base_price NUMERIC(10,2), avg_rating, review_count, status DEFAULT 'ACTIVE', created_at	No stock_quantity on product — stock lives exclusively on variants.
product_variant_options	id UUID PK, product_id FK→products, name VARCHAR(100), display_order INT DEFAULT 0, created_at	Seller-defined option dimensions: 'Colour', 'Size', 'Material', 'Storage'. Fully flexible.

Table	Key Columns	Notes
product_variant_option_values	id UUID PK, option_id FK→product_variant_options, value VARCHAR(100), display_order INT, created_at. UNIQUE(option_id, value)	The selectable values: 'Red', 'Large', '128GB', 'Cotton'.
product_variants	id UUID PK, product_id FK→products, sku VARCHAR(100) UNIQUE, price NUMERIC(10,2), stock_quantity INT DEFAULT 0, low_stock_threshold INT DEFAULT 5, image_url TEXT (legacy), status DEFAULT 'ACTIVE', created_at	The buyable SKU. Own price, stock, and image gallery. INACTIVE = hidden from customers.
product_variant_values	id UUID PK, variant_id FK→product_variants, option_value_id FK→product_variant_option_values. UNIQUE(variant_id, option_value_id)	Junction: links a variant to its selected option values.
product_images	id UUID PK, product_id FK→products, image_url TEXT, display_order INT, is_primary BOOL DEFAULT false, alt_text VARCHAR(255), created_at. UNIQUE(product_id, display_order)	Fallback gallery when selected variant has no own images. Max 8.
variant_images	id UUID PK, variant_id FK→product_variants, image_url TEXT, display_order INT, is_primary BOOL DEFAULT false, alt_text VARCHAR(255), created_at. UNIQUE(variant_id, display_order). CHECK(display_order < 8)	Gallery per variant. Max 8 enforced. is_primary drives the swatch image.
cart_items	id UUID PK, customer_id	References variant_id not product_id. Price captured at add-time.

Table	Key Columns	Notes
	FK→users, variant_id FK→product_variants, store_id FK→stores, quantity INT, unit_price NUMERIC(10,2), created_at. UNIQUE(customer_id, variant_id)	
orders	id UUID PK, customer_id FK→users, status DEFAULT 'PENDING', total_amount NUMERIC(10,2), payment_id FK→payments, created_at	One per checkout regardless of seller count.
sub_orders	id UUID PK, order_id FK→orders, store_id FK→stores, status DEFAULT 'PENDING', subtotal, commission, seller_net, tracking_number, shipped_at, delivered_at, created_at	One per seller within an order. Created automatically by Orders control.
order_items	id UUID PK, order_id FK→orders, sub_order_id FK→sub_orders, variant_id FK→product_variants, store_id FK→stores, quantity INT, unit_price NUMERIC(10,2), subtotal NUMERIC(10,2)	References variant_id. Price captured at purchase time.
payments	id UUID PK, order_id FK→orders UNIQUE, amount NUMERIC(10,2), currency DEFAULT 'GBP', status, provider_ref, created_at	One payment per order.
payouts	id UUID PK, store_id FK→stores, period_start DATE, period_end DATE, gross_amount, commission_amount, net_amount, status DEFAULT 'PENDING',	Weekly. Created by Payouts @Scheduled every Monday 08:00 UTC.

Table	Key Columns	Notes
	paid_at, created_at	
store_metrics	id UUID PK, store_id FK→stores, date DATE, total_orders INT, total_revenue NUMERIC(10,2), total_commission NUMERIC(10,2), avg_order_value NUMERIC(10,2). UNIQUE(store_id, date)	Daily snapshot for dashboard queries.
addresses	id UUID PK, user_id FK→users, line1, line2, city, postcode, country, is_default BOOL, created_at	Reusable shipping addresses. Default pre-filled at checkout.
reviews	id UUID PK, product_id FK→products, variant_id FK→product_variants, customer_id FK→users, sub_order_id FK→sub_orders, rating INT CHECK(1-5), body TEXT, created_at. UNIQUE(customer_id, sub_order_id, variant_id)	Verified purchase only. Rolls up to product avg_rating and store avg_rating.

Flyway Migrations

File	Purpose
V1__users.sql	CREATE TABLE users. Includes password_reset_token and password_reset_expires_at. INDEX on email, password_reset_token.
V2__stores.sql	CREATE TABLE stores. INDEX on slug, seller_id, status.
V3__categories.sql	CREATE TABLE categories (self-referencing parent_id).
V4__products.sql	CREATE TABLE products. No stock_quantity. INDEX on store_id, category_id, status.
V5__product_variants.sql	CREATE TABLE product_variant_options, product_variant_option_values, product_variants, product_variant_values. UNIQUE on sku, UNIQUE(option_id,value), UNIQUE(variant_id,option_value_id).
V6__cart_items.sql	CREATE TABLE cart_items. References variant_id NOT product_id. UNIQUE(customer_id, variant_id).
V7__orders.sql	CREATE TABLE orders. INDEX on customer_id, status.

File	Purpose
V8__sub_orders.sql	CREATE TABLE sub_orders. INDEX on order_id, store_id, status.
V9__order_items.sql	CREATE TABLE order_items. References variant_id. INDEX on order_id, sub_order_id.
V10__payments.sql	CREATE TABLE payments. INDEX on order_id, status.
V11__payouts.sql	CREATE TABLE payouts. INDEX on store_id, status.
V12__store_metrics.sql	CREATE TABLE store_metrics. UNIQUE(store_id, date).
V13__addresses.sql	CREATE TABLE addresses. INDEX on user_id.
V14__reviews.sql	CREATE TABLE reviews. UNIQUE(customer_id, sub_order_id, variant_id). INDEX on product_id.
V15__product_images.sql	CREATE TABLE product_images. UNIQUE(product_id, display_order). INDEX on product_id, is_primary.
V16__variant_images.sql	CREATE TABLE variant_images. UNIQUE(variant_id, display_order). CHECK(display_order < 8). INDEX on variant_id, is_primary.

Product Backlog

Each user story has full acceptance criteria covering the happy path, edge cases, validation rules, and error responses.

Iteration 1 — Foundation

ID	User Story	Acceptance Criteria	Priority
US01	As a developer, I want a Quarkus project scaffold with BCE package structure so all modules follow a consistent architecture from day one.	*GIVEN a fresh checkout WHEN 'mvn quarkus:dev' is run THEN the application starts in under 2 seconds with no errors. ✅ BCE packages (boundary / control / entity) exist for every module. ✅ AGENTS.md is committed to the root of the repo. ✅ /q/health returns HTTP 200 with status UP. ✅ No business logic exists yet — scaffold only.*	**Must Have**
US02	As a developer, I want PostgreSQL with Flyway migrations V1–V16 so the schema is version-controlled from the first commit.	*GIVEN the application starts WHEN Flyway runs on startup THEN all 16 migration scripts execute without errors. ✅ All 19 tables created. ✅ /q/health/ready shows DB check = UP. ✅ If a migration fails, application refuses to start with a clear error. ✅ Re-running migrations on a seeded DB is idempotent.*	**Must Have**
US03	As an operator, I want MicroProfile Health endpoints so load balancers can detect readiness and liveness.	*GIVEN the application is running WHEN GET /q/health is called THEN HTTP 200 is returned with overall status UP. ✅ GET /q/health/live returns 200 when JVM is healthy. ✅ GET /q/health/ready returns 200 only when DB connection is UP. ✅ GET /q/health/ready returns 503 if DB is unreachable. ✅ Response body is valid JSON with 'status' and 'checks' array.*	**Must Have**
US04	As a developer, I want OpenAPI auto-generated from JAX-RS annotations so the API is self-documenting.	*GIVEN the application is running WHEN GET /q/swagger-ui is opened THEN all API endpoints are listed and grouped by @Tag. ✅ Every resource method has @Operation(summary=...) annotation. ✅ Every resource class has @Tag(name=...) annotation. ✅ Request and response schemas are visible in the UI. ✅ GET /q/openapi returns valid OpenAPI 3.x YAML/JSON.*	**Must Have**

Iteration 2 — Authentication & Registration

ID	User Story	Acceptance Criteria	Priority
US05	As a new customer, I want to register with my name, email and password so I can create an account.	*GIVEN a valid registration payload { firstName, lastName, email, password } WHEN POST /api/v1/auth/register is called THEN HTTP 201 is returned with a signed JWT (15-min access token) and a refresh token. ✅User created with role=CUSTOMER and active=true. ✅Password hashed with bcrypt — never stored in plaintext. ✅ JWT 'groups' claim contains ['CUSTOMER']. ✅ Duplicate email → HTTP 409 'Email already registered'. ✅Missing required fields → HTTP 400 with field-level error message from boundary validation. ✅Password under 8 characters → HTTP 400.*	**Must Have**
US06	As a seller, I want to self-register my store so I can start selling immediately without waiting for approval.	*GIVEN a valid payload { firstName, lastName, email, password, storeName, storeDescription } WHEN POST /api/v1/auth/register/seller is called THEN HTTP 201 is returned with a JWT issued immediately. ✅User (role=SELLER) and Store (status=ACTIVE) created in a single @Transactional boundary call — both succeed or both roll back. ✅Store slug auto-generated from storeName (lowercase, hyphenated). ✅If slug already exists, a numeric suffix is appended (e.g. sport-zone-2). ✅JWT 'groups' claim contains ['SELLER']. ✅Duplicate email → HTTP 409. ✅Store immediately visible on GET /api/v1/stores/{slug}.*	**Must Have**
US07	As a registered user, I want to log in so I receive a JWT access token (15 min) and a refresh token (7 days).	*GIVEN valid credentials { email, password } WHEN POST /api/v1/auth/login is called THEN HTTP 200 is returned with { accessToken, refreshToken, expiresIn }. ✅Access token expires in 900 seconds (15 min). ✅Refresh token expires in 604800 seconds (7 days). ✅Wrong password → HTTP 401 'Invalid credentials' (do not distinguish email vs password). ✅Unknown email → HTTP 401 'Invalid credentials' (no user enumeration). ✅Inactive account (active=false) → HTTP 403 'Account suspended'. ✅ Suspended store seller → HTTP 403 'Store suspended'.*	**Must Have**
US08	As a logged-in user, I want to refresh my access token so I stay logged in without re-entering credentials.	*GIVEN a valid refresh token WHEN POST /api/v1/auth/refresh is called THEN HTTP 200 is returned with a new access token and a rotated refresh token. ✅Old refresh token invalidated immediately on use (rotation). ✅Expired refresh token → HTTP 401 'Refresh token expired'. ✅ Already-used (rotated) token → HTTP 401 'Invalid refresh token'. ✅Malformed token → HTTP 401.*	**Must Have**
US09	As a user who forgot their password, I want to request a password reset link sent to my email so I can regain access to my account.	*GIVEN a registered email address WHEN POST /api/v1/auth/forgot-password with { email } is called THEN HTTP 200 is returned with message 'If this email is registered, a reset link has been sent' (same message whether email exists or not — no user enumeration). ✅A cryptographically random token (min 32 bytes, URL-safe) is generated and stored as a hash in users.password_reset_token with a 1-hour expiry in users.password_reset_expires_at. ✅An email is sent to the address containing a reset link: https://[domain]/reset-password?token=[raw-token]. ✅If the email is not registered, no email is sent but HTTP 200 is still returned. ✅If a previous token*	**Must Have**

ID	User Story	Acceptance Criteria	Priority
		exists and has not expired, it is overwritten with the new token. ✅ Rate-limited: max 3 requests per email per hour → HTTP 429 if exceeded.	
US10	As a user who received a password reset email, I want to set a new password using the link so I can regain access.	*GIVEN a valid reset token WHEN POST /api/v1/auth/reset-password with { token, newPassword, confirmPassword } is called THEN HTTP 200 is returned and the user's password is updated. ✅ token must match the hashed value in users.password_reset_token → HTTP 400 'Invalid or expired reset token' if not. ✅ Token must not be expired (password_reset_expires_at > now) → HTTP 400 'Invalid or expired reset token'. ✅ newPassword must be at least 8 characters → HTTP 400. ✅ newPassword and confirmPassword must match → HTTP 400 'Passwords do not match'. ✅ On success: password is hashed with bcrypt and saved, password_reset_token and password_reset_expires_at are cleared (set to null), all existing refresh tokens for this user are invalidated. ✅ Token can only be used once — reuse → HTTP 400 'Invalid or expired reset token'. ✅ Unknown or already-cleared token → HTTP 400 (same message — no enumeration).*	**Must Have**

Iteration 3 — Store Profile & Branding

ID	User Story	Acceptance Criteria	Priority
US11	As a seller, I want to update my store profile (name, description, logo, banner) so my storefront looks professional.	*GIVEN an authenticated SELLER WHEN PUT /api/v1/stores/me with { name, description, logoUrl, bannerUrl } is called THEN HTTP 200 is returned with the updated store object. ✅ store_id derived from JWT subject — seller cannot update another seller's store. ✅ If name changes, slug is regenerated. ✅ If new slug conflicts, HTTP 409. ✅ logoUrl and bannerUrl are optional; omitting them does not clear existing values. ✅ Non-SELLER JWT → HTTP 403. ✅ Response includes updated slug.*	**Must Have**
US12	As a customer, I want to browse a seller's public storefront so I can see all their products and rating.	*GIVEN a store slug WHEN GET /api/v1/stores/{slug} is called (no auth required) THEN HTTP 200 is returned with store profile and paginated ACTIVE products. ✅ Response includes: name, description, logoUrl, bannerUrl, avg_rating. ✅ commissionRate hidden from customers — admin only. ✅ Only products with status=ACTIVE are returned. ✅ Pagination: ?page=0&size=20 (default size 20). ✅ Suspended store → HTTP 404. ✅ Unknown slug → HTTP 404. ✅ Products include primary image and stockStatus per variant.*	**Must Have**
US13	As a Platform Admin, I want to set the commission rate per store so the platform earns the correct percentage.	*GIVEN an authenticated PLATFORM_ADMIN WHEN PUT /api/v1/admin/stores/{id}/commission with { commissionRate: 12.50 } is called THEN HTTP 200 is returned and the store's commission_rate is updated. ✅ Rate must be between 0.00 and 100.00 inclusive — outside range → HTTP 400. ✅ Change applies to all future SubOrders only — existing sub_orders are not recalculated. ✅ Non-PLATFORM_ADMIN JWT → HTTP 403. ✅ Unknown store id → HTTP 404.*	**Must Have**
US14	As a Platform Admin, I want to	*GIVEN an authenticated PLATFORM_ADMIN WHEN*	**Must Have**

ID	User Story	Acceptance Criteria	Priority
	suspend or reinstate a seller store so I can remove bad actors from the marketplace.	*PUT /api/v1/admin/stores/{id}/suspend is called THEN HTTP 200 is returned and store status becomes SUSPENDED. ✅ All products of the suspended store set to status=INACTIVE. ✅ Suspended store no longer appears in public browsing or search. ✅ Seller JWT still works for login but store endpoints return 403. ✅ Audit log entry created with: admin userId, storeId, action, timestamp. ✅ WHEN PUT /api/v1/admin/stores/{id}/reinstate is called — store status returns to ACTIVE. ✅ Products that were ACTIVE before suspension are restored to ACTIVE. ✅ Products already INACTIVE before suspension remain INACTIVE. ✅ Non-PLATFORM_ADMIN JWT → HTTP 403.*	

Iteration 4 — Product Catalog & Variants

ID	User Story	Acceptance Criteria	Priority
US15	As a seller, I want to create a product with base details so I have a catalogue entry to attach variants to.	*GIVEN an authenticated SELLER WHEN POST /api/v1/products with { name, description, basePrice, categoryId } is called THEN HTTP 201 is returned with the new product including its id. ✅ store_id taken from JWT subject — seller cannot create products for another store. ✅ Product created with status=ACTIVE. ✅ No stock_quantity field on product — stock lives only on variants. ✅ categoryId must reference a valid existing category → HTTP 404 otherwise. ✅ basePrice must be > 0 → HTTP 400 otherwise. ✅ name and description are required → HTTP 400 if missing.*	**Must Have**
US16	As a seller, I want to define variant options (e.g. Colour, Size) and their values so customers can select exactly what they want.	*GIVEN an authenticated SELLER who owns the product WHEN POST /api/v1/products/{id}/options with { name: 'Colour' } is called THEN HTTP 201 is returned with the new option id. ✅ Ownership check: product.store.sellerId must equal JWT subject → HTTP 403 otherwise. ✅ Unknown product id → HTTP 404. ✅ WHEN POST /api/v1/products/{id}/options/{optId}/values with { value: 'Red' } is called → HTTP 201 returned with the new option value id. ✅ Duplicate value within same option → HTTP 409 'Value already exists for this option'. ✅ Unknown option id → HTTP 404. ✅ Value cannot be blank → HTTP 400.*	**Must Have**
US17	As a seller, I want to create variants with their own SKU, price, stock and image so each combination is independently buyable.	*GIVEN an authenticated SELLER who owns the product WHEN POST /api/v1/products/{id}/variants with { sku, price, stockQuantity, lowStockThreshold, imageUrl, optionValueIds } is called THEN HTTP 201 is returned with the new variant. ✅ SKU must be unique platform-wide → HTTP 409 if duplicate. ✅ optionValueIds must include exactly one value per option dimension defined on the product. ✅ Duplicate option-value combination → HTTP 409. ✅ price must be > 0 → HTTP 400. ✅ stockQuantity must be ≥ 0 → HTTP 400. ✅ lowStockThreshold defaults to 5 if not provided. ✅ Ownership check → HTTP 403.*	**Must Have**
US17A	As a seller, I want to upload multiple images per variant so customers see the product from all angles.	*GIVEN an authenticated SELLER who owns the variant WHEN POST /api/v1/products/{id}/variants/{varId}/images with { imageUrl, altText, isPrimary } is called THEN HTTP*	**Must Have**

ID	User Story	Acceptance Criteria	Priority
		201 is returned with the created image record. ✅ Maximum 8 images per variant enforced → HTTP 400 'Max 8 images per variant' if exceeded. ✅ If isPrimary=true, any existing primary image for this variant is cleared first. ✅ display_order is auto-assigned as count of existing images. ✅ imageUrl is required → HTTP 400 if missing. ✅ Ownership check → HTTP 403. ✅ Unknown variant id → HTTP 404.	
US17B	As a seller, I want to reorder my variant images so the most appealing photo appears first.	*GIVEN an authenticated SELLER who owns the variant WHEN PUT /api/v1/products/{id}/variants/{varId}/images/reorder with { imageIds: [uuid1, uuid2, ...] } is called THEN HTTP 200 is returned with the re-ordered image list. ✅ display_order reassigned 0, 1, 2... matching the order of imageIds. ✅ All imageIds must belong to this variant → HTTP 400 if any are foreign. ✅ Count of imageIds must equal count of existing images → HTTP 400 if mismatch. ✅ Primary flag preserved. ✅ Ownership check → HTTP 403.*	**Must Have**
US17C	As a seller, I want to add product-level fallback images that show when a selected variant has no own images.	*GIVEN an authenticated SELLER who owns the product WHEN POST /api/v1/products/{id}/images with { imageUrl, altText, isPrimary } is called THEN HTTP 201 is returned. ✅ Maximum 8 images per product enforced. ✅ Same isPrimary promotion logic as variant images. ✅ These images are only shown when the selected variant has zero entries in variant_images. ✅ Ownership check → HTTP 403. ✅ Unknown product id → HTTP 404.*	**Must Have**
US18	As a seller, I want to update variant stock, price or status so I can manage my inventory.	*GIVEN an authenticated SELLER who owns the product WHEN PUT /api/v1/products/{id}/variants/{varId} with { price, imageUrl, status } is called THEN HTTP 200 is returned with the updated variant. ✅ price must be > 0 if provided → HTTP 400. ✅ status must be ACTIVE or INACTIVE → HTTP 400 for other values. ✅ Setting status=INACTIVE hides the variant from all customer-facing endpoints immediately. ✅ Ownership check → HTTP 403. ✅ WHEN PATCH /api/v1/products/{id}/variants/{varId}/stock with { quantity: 50 } is called — quantity is ADDED to existing stock_quantity (top-up, not replacement). ✅ quantity must be > 0 → HTTP 400. ✅ Response returns updated stockQuantity (seller-facing only).*	**Must Have**
US19	As a customer, I want to view a product detail page with all variant options so I can select exactly what I want to buy.	*GIVEN a product id (no auth required) WHEN GET /api/v1/products/{id} is called THEN HTTP 200 is returned with product details and the full variant matrix. ✅ Response includes: product name, description, basePrice, avg_rating, review_count, category. ✅ All variant option dimensions (e.g. Colour, Size) listed. ✅ All ACTIVE variants returned with: sku, price, stockStatus, lowStockMessage, images[], selectedOptions. ✅ INACTIVE variants excluded entirely. ✅ stockQuantity NEVER included in the customer-facing response. ✅ stockStatus: qty=0 → OUT_OF_STOCK; qty≤threshold → LOW_STOCK; else IN_STOCK. ✅ Unknown product id → HTTP 404. ✅ Product from SUSPENDED store → HTTP 404.*	**Must Have**
US19A	As a customer, when I select a variant I want the image gallery to update to show that variant's images.	*GIVEN a variant id (no auth required) WHEN GET /api/v1/products/{id}/variants/{varId}/images is called THEN HTTP 200 is returned with the resolved image gallery. ✅ Resolution order: (1) variant_images for this variant, (2) product_images fallback, (3) legacy variant.imageUrl as single-item list, (4) empty list. ✅*	**Must Have**

ID	User Story	Acceptance Criteria	Priority
		Images returned ordered by display_order ascending. ✅ isPrimary flag set on the correct image. ✅ INACTIVE variant → HTTP 404. ✅ Unknown variant id → HTTP 404.	
US20	As a customer, I want out-of-stock variants shown but greyed out so I know what exists without confusion.	*GIVEN a product with variants of varying stock levels WHEN GET /api/v1/products/{id} is called THEN out-of-stock variants are included in the response with stockStatus=OUT_OF_STOCK. ✅ Variants with stockQuantity=0 and status=ACTIVE returned with stockStatus=OUT_OF_STOCK. ✅ lowStockMessage is null for OUT_OF_STOCK variants. ✅ INACTIVE variants are NOT returned — hidden entirely. ✅ If ALL ACTIVE variants are OUT_OF_STOCK, the product-level field 'currentlyUnavailable': true is included. ✅ Customer UI expected to disable 'Add to Cart' for OUT_OF_STOCK variants based on stockStatus.*	**Must Have**
US21	As a customer, I want to see a low stock warning on variants so I know when to act fast.	*GIVEN a variant with stockQuantity ≤ lowStockThreshold WHEN GET /api/v1/products/{id} is called THEN that variant has stockStatus=LOW_STOCK and a lowStockMessage in the response. ✅ lowStockMessage = 'Only N left in stock' where N = actual stockQuantity. ✅ Default lowStockThreshold is 5 if not set by seller. ✅ stockQuantity > threshold → stockStatus=IN_STOCK, lowStockMessage=null. ✅ stockQuantity = 0 → stockStatus=OUT_OF_STOCK (takes precedence). ✅ Raw stockQuantity never exposed — only lowStockMessage with the count.*	**Must Have**
US22	As a customer, I want to browse and search products across all stores with filters and pagination.	*GIVEN no authentication required WHEN GET /api/v1/products or GET /api/v1/products/search?q=shoe&minPrice=20&maxPrice=100&sort=price_asc is called THEN HTTP 200 is returned with a paginated list of matching products. ✅ Only ACTIVE products from ACTIVE (non-suspended) stores returned. ✅ Only products with at least one ACTIVE variant with stockQuantity > 0 shown by default. ✅ Text search (q) matches product name and description (case-insensitive). ✅ minPrice / maxPrice filter against the lowest variant price for that product. ✅ sort supports: price_asc, price_desc, newest, rating. ✅ Pagination: ?page=0&size=20. Response includes totalElements, totalPages. ✅ Each result includes primary image and minimum variant price.*	**Must Have**

Iteration 5 — Shopping Cart

ID	User Story	Acceptance Criteria	Priority
US23	As a customer, I want to add a specific product variant to my cart so I buy exactly what I selected.	*GIVEN an authenticated CUSTOMER WHEN POST /api/v1/cart/items with { variantId, quantity } is called THEN HTTP 201 is returned and the item is in the cart. ✅ Variant must exist and have status=ACTIVE → HTTP 404 otherwise. ✅ Variant must belong to an ACTIVE (non-suspended) store → HTTP 400 otherwise. ✅ stockQuantity must be ≥ requested quantity → HTTP 400 'Insufficient stock' otherwise. ✅ unit_price captured at add-to-cart time from variant.price (price may change later). ✅ If the same variant already exists in the cart, quantity is incremented (not duplicated). ✅ quantity must be ≥ 1 → HTTP 400.*	**Must Have**

ID	User Story	Acceptance Criteria	Priority
US24	As a customer, I want to view my cart grouped by store with a total per store and grand total.	*GIVEN an authenticated CUSTOMER WHEN GET /api/v1/cart is called THEN HTTP 200 is returned with cart contents grouped by store. ✅Each group shows: storeName, storeSlug, list of items, subtotal for that store. ✅Each item shows: productName, variantOptions (e.g. 'Blue / Size 9'), unitPrice, quantity, lineSubtotal, stockStatus (re-checked live). ✅If a variant has gone INACTIVE or OOS since it was added, it is flagged with a warning — not silently removed. ✅ Grand total = sum of all store subtotals. ✅Empty cart → HTTP 200 with empty array.*	**Must Have**
US25	As a customer, I want to update the quantity or remove items from my cart.	*GIVEN an authenticated CUSTOMER WHEN PUT /api/v1/cart/items/{id} with { quantity: 3 } is called THEN HTTP 200 is returned with the updated cart item. ✅ quantity must be ≥ 1 → HTTP 400. ✅quantity must not exceed current stockQuantity → HTTP 400 'Insufficient stock'. ✅Setting quantity = 0 removes the item (same as DELETE). ✅Customer can only update their own cart items → HTTP 403 for others. ✅WHEN DELETE /api/v1/cart/items/{id} is called → HTTP 204 returned. ✅Unknown cart item id → HTTP 404.*	**Must Have**

Iteration 6 — Multi-Seller Orders

ID	User Story	Acceptance Criteria	Priority
US26	As a customer, I want to checkout my entire cart in one payment regardless of how many sellers I'm buying from.	*GIVEN an authenticated CUSTOMER with items in cart WHEN POST /api/v1/orders with { shippingAddressId } is called THEN HTTP 201 is returned with the created order. ✅Each variant locked with PESSIMISTIC_WRITE (SELECT FOR UPDATE) before decrement. ✅If any variant has insufficient stock, entire order fails with HTTP 400 — no partial orders. ✅ Stock decremented BEFORE payment is attempted. ✅ One payment charge for the full cart total. ✅Orders control groups items by store_id and creates one SubOrder per seller. ✅commission = subtotal × store.commissionRate / 100; seller_net = subtotal − commission. ✅LowStockEvent fired for each variant whose stock falls to or below lowStockThreshold after decrement. ✅SubOrderCreatedEvent fired for each seller (async). ✅OrderConfirmedEvent fired for the customer (async). ✅Cart is cleared on success. ✅If payment fails, stock decrements are rolled back (all inside @Transactional boundary).*	**Must Have**
US27	As a seller, I want to see only my sub-orders so I can fulfil my items without seeing other sellers' data.	*GIVEN an authenticated SELLER WHEN GET /api/v1/seller/orders is called THEN HTTP 200 is returned with SubOrders belonging only to this seller's store. ✅SubOrders from other stores never returned — filtered by store.sellerId = JWT subject. ✅Each sub-order includes: subOrderId, status, customerName, shippingAddress, list of items. ✅Each item shows: productName, variantOptions (e.g. 'Red / Size 10'), quantity, unitPrice, lineSubtotal. ✅Pagination: ?page=0&size=20. ✅Optional filter: ?status=PENDING.*	**Must Have**
US28	As a seller, I want to mark a sub-order as shipped with a tracking number so the customer is notified.	*GIVEN an authenticated SELLER WHEN PUT /api/v1/seller/orders/{subOrderId}/ship with { trackingNumber } is called THEN HTTP 200 is returned and sub-order status is SHIPPED. ✅Seller can only ship their own sub-orders → HTTP 403 for*	**Must Have**

ID	User Story	Acceptance Criteria	Priority
		foreign sub-orders. ✅ Sub-order must be in PENDING status to ship → HTTP 409 if already SHIPPED or DELIVERED. ✅ trackingNumber stored and returned in customer order view. ✅ shipped_at timestamp recorded. ✅ SubOrderShippedEvent fired asynchronously (customer email notification). ✅ Unknown subOrderId → HTTP 404.	
US29	As a customer, I want to view my order with per-seller shipment status and tracking so I know when to expect each delivery.	*GIVEN an authenticated CUSTOMER WHEN GET /api/v1/orders/{id} is called THEN HTTP 200 is returned with the full order detail. ✅ Customer can only view their own orders → HTTP 403 for foreign orders. ✅ Response includes: orderId, status, totalAmount, createdAt, paymentStatus. ✅ sub_orders array includes per seller: storeName, status, trackingNumber, shippedAt, items. ✅ Each item includes: productName, variantOptions, quantity, unitPrice. ✅ Unknown order id → HTTP 404.*	**Must Have**
US30	As a customer, I want to cancel an order before any seller ships it so I receive a full refund.	*GIVEN an authenticated CUSTOMER WHEN PUT /api/v1/orders/{id}/cancel is called THEN HTTP 200 is returned if cancellation is allowed. ✅ Cancellation only allowed if ALL sub-orders are still in PENDING status. ✅ If any sub-order is SHIPPED or DELIVERED → HTTP 409 'Order cannot be cancelled — one or more items already shipped'. ✅ On success: all sub-orders set to CANCELLED, order set to CANCELLED. ✅ Stock restored per variant (stockQuantity += original quantity). ✅ Full payment refund triggered via Payments control. ✅ Customer can only cancel their own orders → HTTP 403.*	**Should Have**

Iteration 7 — Commission & Payouts

ID	User Story	Acceptance Criteria	Priority
US31	As a seller, I want to see the commission breakdown on each sub-order so I understand my net earnings.	*GIVEN an authenticated SELLER WHEN GET /api/v1/seller/orders/{subOrderId} is called THEN HTTP 200 is returned with full financial breakdown. ✅ Response includes: subtotal, commissionRate (%), commissionAmount, sellerNet. ✅ Formula verified: commissionAmount = subtotal × commissionRate / 100; sellerNet = subtotal − commissionAmount. ✅ Seller can only view their own sub-orders → HTTP 403 for others. ✅ Unknown subOrderId → HTTP 404.*	**Must Have**
US32	As a seller, I want to receive an automatic weekly payout for all my delivered sub-orders so I am paid without manual intervention.	*GIVEN it is Monday 08:00 UTC WHEN the @Scheduled Payouts job runs THEN one Payout record is created per store with DELIVERED, unpaid sub-orders. ✅ Only DELIVERED sub-orders with no existing payout are included. ✅ Payout fields: storeId, periodStart, periodEnd, grossAmount, commissionAmount, netAmount, status=PENDING. ✅ netAmount = sum of seller_net across all qualifying sub-orders. ✅ PayoutProcessedEvent fired for each payout → Notifications sends seller an email with full breakdown. ✅ Sub-orders included in the payout are marked as paid. ✅ If no qualifying sub-orders for a store, no payout record is created.*	**Must Have**
US33	As a seller, I want to view my payout history so I can	*GIVEN an authenticated SELLER WHEN GET /api/v1/seller/payouts is called THEN HTTP 200 is returned with paginated payout history. ✅ Results*	**Must Have**

ID	User Story	Acceptance Criteria	Priority
	reconcile my earnings.	*sorted by period_end DESC (most recent first). ✅Each payout shows: periodStart, periodEnd, grossAmount, commissionAmount, netAmount, status, paidAt. ✅ Seller can only see their own store's payouts → HTTP 403 for others. ✅Pagination: ?page=0&size=20.*	
US34	As a seller, I want to be alerted when a variant's stock falls low so I can restock before selling out.	*GIVEN an order is confirmed and stock is decremented WHEN a variant's stockQuantity falls to or below lowStockThreshold THEN a LowStockEvent is fired and the seller receives an email alert. ✅Alert fired inside the order @Transactional boundary after decrement. ✅ Email subject: 'Low stock alert: [SKU]'. ✅Email body includes: SKU, variant options, remaining stock count, link to seller dashboard. ✅Default threshold is 5; seller can configure per variant. ✅Alert fires on every qualifying order — not de-duplicated. ✅Alert fires even if stock hits exactly the threshold (≤, not <).*	**Must Have**

Iteration 8 — Dashboards

ID	User Story	Acceptance Criteria	Priority
US35	As a seller, I want a dashboard with my key metrics so I can understand my store's performance at a glance.	*GIVEN an authenticated SELLER WHEN GET /api/v1/seller/dashboard is called THEN HTTP 200 is returned with aggregated metrics. ✅revenueToday: sum of seller_net from DELIVERED sub-orders today (UTC). ✅revenueMTD: revenue from 1st of current month to now. ✅revenueYTD: revenue from 1st of current year to now. ✅ordersToday / ordersMTD: count of sub-orders in PENDING, SHIPPED, DELIVERED. ✅ pendingOrders: count of sub-orders in PENDING status. ✅avgOrderValue: revenueMTD / ordersMTD (null if no orders). ✅pendingPayoutAmount: sum of seller_net for DELIVERED unpaid sub-orders. ✅top5Variants: top 5 variants by revenue MTD (variantSku, productName, revenue). ✅storeRating: current store avg_rating. ✅commissionRate: current rate for this store.*	**Should Have**
US36	As a Platform Admin, I want a marketplace dashboard showing GMV, commission, top stores and platform health.	*GIVEN an authenticated PLATFORM_ADMIN WHEN GET /api/v1/admin/dashboard is called THEN HTTP 200 is returned with platform-wide aggregated metrics. ✅gmvToday / gmvMTD / gmvYTD: gross merchandise value (sum of order totals). ✅ commissionMTD: total commission earned by platform this month. ✅ totalActiveSellers: count of stores with status=ACTIVE. ✅ordersToday: count of orders created today. ✅ pendingSubOrders: count of sub-orders in PENDING status platform-wide. ✅ top10Stores: top 10 stores by GMV MTD (storeName, gmv, commissionEarned). ✅newCustomersToday: count of users with role=CUSTOMER registered today.*	**Should Have**

ID	User Story	Acceptance Criteria	Priority
		✅ *Non-PLATFORM_ADMIN → HTTP 403.*	

Iteration 9 — AI & Intelligence

ID	User Story	Acceptance Criteria	Priority
US37	As a customer, I want an AI-powered product search chatbot so I can describe what I want in natural language and get relevant results.	*GIVEN an authenticated or guest CUSTOMER WHEN POST /api/v1/ai/chat with { message: 'I need a waterproof jacket for hiking under £80' } is called THEN HTTP 200 is returned with a conversational response and a list of matching products. ✅ LLM (e.g. Claude API) interprets intent and extracts filters: category, maxPrice, keywords. ✅ Filters passed to existing product search endpoint. ✅ Response includes: natural language reply + products[]. ✅ Conversation history maintained per session (last 10 turns). ✅ If no products match, LLM suggests alternatives or broader search. ✅ Profanity or off-topic queries responded to gracefully.*	**Should Have**
US38	As a customer, I want AI-powered product recommendations on the home page and product detail page so I discover products I am likely to buy.	*GIVEN an authenticated CUSTOMER WHEN GET /api/v1/ai/recommendations is called THEN HTTP 200 is returned with up to 12 recommended products. ✅ Recommendations based on: order history, browsed products (session), cart contents, avg_rating. ✅ New customers receive trending / high-rated products as cold-start fallback. ✅ On product detail page: GET /api/v1/ai/recommendations?productId={id} returns 'Customers also bought' and 'Similar items'. ✅ Recommendations exclude products already in cart or previously purchased. ✅ Response cached in Redis (TTL 30 min per customer). ✅ PLATFORM_ADMIN can see recommendation model stats via admin dashboard.*	**Should Have**
US39	As a seller, I want an AI assistant to help me write compelling product titles and descriptions so I can list products faster and improve conversion.	*GIVEN an authenticated SELLER WHEN POST /api/v1/ai/seller/generate-description with { productName, category, keyFeatures: ['waterproof', 'lightweight'] } is called THEN HTTP 200 is returned with { title, description, bulletPoints[] }. ✅ LLM generates SEO-friendly title (max 80 chars), description (max 500 chars), and up to 5 bullet points. ✅ Seller can regenerate (up to 5 times per product). ✅ Seller can edit before saving — generated content is a suggestion only, not auto-saved. ✅ Rate-limited: 20 generations per seller per day → HTTP 429 if exceeded. ✅ Ownership check: seller can only generate for their own products.*	**Should Have**
US40	As a seller, I want AI-powered	*GIVEN an authenticated SELLER WHEN*	**Should Have**

ID	User Story	Acceptance Criteria	Priority
	pricing suggestions based on similar products in the marketplace so I can price competitively.	*GET /api/v1/ai/seller/pricing-suggest?productId={id} is called THEN HTTP 200 is returned with a pricing recommendation. ✅ Response includes: suggestedPrice, priceRange { min, max }, rationale (e.g. 'Similar items sell for £45–£65; your current price of £80 is above market'). ✅ Comparison based on: same category, similar name tokens, ACTIVE variants with stock. ✅ Competitor store names are NOT revealed — only aggregate price data shown. ✅ Recommendation is advisory only — seller decides. ✅ Refreshed daily; cached in Redis (TTL 24h per product).*	
US41	As a seller, I want AI to predict which of my variants are likely to go out of stock in the next 7 days so I can restock proactively.	*GIVEN an authenticated SELLER WHEN GET /api/v1/ai/seller/stock-forecast is called THEN HTTP 200 is returned with a list of at-risk variants. ✅ Model uses: current stock_quantity, average daily sales velocity (last 30 days), day-of-week trends. ✅ Each at-risk variant shows: sku, variantOptions, currentStock, forecastedStockoutDate, recommendedRestockQuantity. ✅ Variants with < 14 days of stock at current velocity are flagged. ✅ Seller with < 30 days of sales history returns a message: 'Not enough data yet — check back after 30 days of sales.' ✅ Results refreshed nightly by a @Scheduled job.*	**Should Have**
US42	As a Platform Admin, I want an AI-generated weekly marketplace health report so I can spot trends and anomalies without manually analysing data.	*GIVEN it is Monday 08:00 UTC WHEN the @Scheduled AI report job runs THEN a MarketplaceHealthReport is generated and emailed to all PLATFORM_ADMINs. ✅ Report includes: GMV trend (this week vs last week vs 4-week avg), top 5 rising stores, top 5 declining stores, unusual order cancellation spikes, new seller signup trend, AI-generated executive summary paragraph (2–3 sentences). ✅ LLM generates the executive summary from structured data — no hallucination risk as input is numeric. ✅ Report also available on demand: GET /api/v1/admin/ai/health-report?week=2026-W08. ✅ Non-PLATFORM_ADMIN → HTTP 403.*	**Should Have**
US43	As a customer, I want an AI-powered review summary on each product page so I can quickly understand overall sentiment without reading all reviews.	*GIVEN a product with ≥ 5 reviews WHEN GET /api/v1/products/{id} is called THEN the response includes an aiReviewSummary field. ✅ Summary generated by LLM from the latest 50 review bodies: 2–3 sentences covering overall sentiment, top praised aspects, most common complaints. ✅ Summary regenerated when review count increases by 10 or rating changes by ≥ 0.3. ✅ Cached in Redis (TTL 12h). ✅ Products with < 5 reviews return aiReviewSummary: null. ✅ Summary includes: overallSentiment (POSITIVE / MIXED / NEGATIVE), highlights[], concerns[].*	**Should Have**

Build Order Summary

#	Iteration	Feature	Key Stories
1	1 — Foundation	Quarkus scaffold, BCE, V1–V16 migrations, Health, OpenAPI	US01 to US04
2	2 — Auth	Customer register, Seller register, Login, Refresh, Forgot Password, Reset Password	US05 to US10
3	3 — Stores	Store profile, Public storefront, Admin suspend & commission	US11 to US14
4	4 — Catalog + Images	Products, Variants, Options, Image gallery, Stock status, Browse & Search	US15 to US22
5	5 — Cart	Add variant to cart, View grouped by store, Update/remove	US23 to US25
6	6 — Orders	Multi-seller checkout, Stock decrement, Sub-order split, Ship, Track, Cancel	US26 to US30
7	7 — Payouts	Commission, Weekly CRON, Payout history, Low-stock alerts	US31 to US34
8	8 — Dashboards	Seller dashboard + Admin marketplace dashboard	US35, US36
★	9 — AI & Intelligence	AI chatbot search, Recommendations, Description generator, Pricing suggest, Stock forecast, Health report, Review summary	US37–US43

Workshop target (7 hours): Iterations 1–7 deliver a complete multi-seller marketplace with password recovery, full variant support, image galleries, atomic stock management, multi-seller checkout, automatic sub-order splitting, low-stock alerts, commission calculation, weekly payouts and dashboards. Iterations 9 (AI Intelligence) is post-workshop stretch goals.