

RETAIL INNOVATIONS LTD

Digital Retail Management Platform

Task 1 – Activity B: Visual Interface and Data Designs

T Level Technical Qualification in Digital Software Development

1. Algorithm Design

Before designing the visual interface, the key algorithms and logic flows were planned as pseudocode and structured flowcharts. This ensures the solution is logically sound before implementation begins. The following algorithms cover the core processes within the platform.

1.1 User Authentication Algorithm

This algorithm handles both login and registration, with role assignment and appropriate UI state changes on success.

ALGORITHM: handleAuth()

PROCEDURE handleAuth()

```
INPUT: email (string), password (string)
VALIDATE: IF email is empty OR password is empty THEN
    DISPLAY toast('Please enter email and password', 'error')
    RETURN
END IF
```

```
IF authMode == 'register' THEN
    CALL localRegister(email, password)
ELSE
    CALL localLogin(email, password)
END IF
```

PROCEDURE localLogin(email, password)

```
IF email == ADMIN_EMAIL AND password == ADMIN_PASSWORD THEN
    SET currentUser = { id: 'admin_1', email: email }
    SET isAdmin = TRUE
    STORE session in localStorage
    CALL enterApp()
    RETURN
END IF
```

```
users = READ localStorage['ri_users']
user = FIND user WHERE user.email == email
IF user NOT FOUND THEN
    DISPLAY toast('User not found', 'error')
    RETURN
END IF
IF user.password != password THEN
    DISPLAY toast('Invalid credentials', 'error')
    RETURN
END IF
SET currentUser, isAdmin = (user.role == 'admin')
STORE session, CALL enterApp()
```

PROCEDURE localRegister(email, password)

```
IF password.length < 8 THEN
    DISPLAY toast('Password too short', 'error')
    RETURN
END IF
IF email already exists in ri_users THEN
    DISPLAY toast('User already exists', 'error')
```

```

    RETURN
  END IF
  newUser = { id: generateId(), email, role: 'customer' }
  APPEND newUser to ri_users
  STORE session, CALL enterApp()
END PROCEDURE

```

1.2 Role-Based UI Rendering Algorithm

This algorithm determines which UI elements are shown based on the authenticated user's role. It runs every time a user logs in.

ALGORITHM: enterApp()

```

PROCEDURE enterApp()
  HIDE authScreen
  SHOW appWrapper

  IF isAdmin == TRUE THEN
    ADD class 'is-admin' to document.body
    // CSS rule: .admin-only { display: block }
    // This reveals: Customers tab, admin action buttons,
    //                dashboard charts, revenue total
    SET ordersSubtitle = 'All customer orders (admin view)'
  ELSE
    REMOVE class 'is-admin' from document.body
    // CSS rule: .admin-only { display: none }
    SET ordersSubtitle = 'Your orders'
  END IF

  UPDATE userBadge with name and role
  CALL loadAllData()
  // loadAllData calls: loadProducts, loadCustomers (admin only),
  //                loadOrders, loadRewards, loadDashboard
END PROCEDURE

```

1.3 Product Filtering Algorithm

This algorithm filters the product catalogue in real time as the user types or selects a category. It operates on the cached product array to avoid repeated database calls.

ALGORITHM: filterProducts()

```

PROCEDURE filterProducts()
  searchTerm = LOWERCASE(getValue('productSearch'))
  category = getValue('productCategoryFilter')

  filtered = []
  FOR EACH product IN productsCache DO
    nameMatch = CONTAINS(LOWERCASE(product.name), searchTerm)
    descMatch = CONTAINS(LOWERCASE(product.description), searchTerm)
    catMatch = (category == '') OR (product.category == category)

    IF (nameMatch OR descMatch) AND catMatch THEN

```

```

        APPEND product TO filtered
    END IF
END FOR

CALL renderProducts(filtered)
UPDATE productCount label to show filtered.length
END PROCEDURE

```

1.4 Loyalty Tier Calculation Algorithm

This algorithm determines a customer's loyalty tier based on their current points balance. It is called whenever loyalty points are updated.

ALGORITHM: calculateLoyaltyTier(points)

```

FUNCTION calculateLoyaltyTier(points) RETURNS string
    IF points >= 2000 THEN
        RETURN 'Platinum'
    ELSE IF points >= 1000 THEN
        RETURN 'Gold'
    ELSE IF points >= 500 THEN
        RETURN 'Silver'
    ELSE
        RETURN 'Bronze'
    END IF
END FUNCTION

// Called when saving a customer:
PROCEDURE saveCustomer()
    loyalty_points = parseInt(pointsInput.value)
    IF loyalty_points < 0 THEN
        DISPLAY toast('Points cannot be negative', 'error')
        RETURN
    END IF
    loyalty_tier = calculateLoyaltyTier(loyalty_points)
    payload.loyalty_tier = loyalty_tier
    // tier is stored in DB, not recalculated each read
END PROCEDURE

```

1.5 Dashboard Analytics Algorithm

This algorithm processes the in-memory product and customer caches to generate the chart data structures used by the dashboard.

ALGORITHM: renderBarChart() and renderDonutChart()

```

PROCEDURE renderBarChart()
    categories = {} // empty dictionary
    FOR EACH product IN productsCache DO
        IF categories[product.category] EXISTS THEN
            categories[product.category] = categories[product.category] + 1
        ELSE
            categories[product.category] = 1
        END IF
    END FOR

```

```

END FOR

entries = SORT categories BY count DESCENDING
entries = TAKE first 7 entries
maxCount = MAX(entry.count for entry in entries)

FOR EACH [category, count] IN entries DO
  barHeight = (count / maxCount) * 100 // percentage
  RENDER bar column with height barHeight
  RENDER label below bar
END FOR

PROCEDURE renderDonutChart()
  tiers = { Bronze: 0, Silver: 0, Gold: 0, Platinum: 0 }
  FOR EACH customer IN customersCache DO
    tiers[customer.loyalty_tier] = tiers[customer.loyalty_tier] + 1
  END FOR

  total = customersCache.length
  degree = 0
  conicParts = []
  FOR EACH [tier, count] IN tiers DO
    sweep = (count / total) * 360
    conicParts.APPEND(tierColor + ' ' + degree + 'deg ' + (degree+sweep) +
'deg')
    degree = degree + sweep
  END FOR
  APPLY conic-gradient(conicParts) to donutRing element
END PROCEDURE

```

1.6 Save/Update Entity Algorithm (Generic CRUD Pattern)

All create/update operations follow the same pattern. This is shown using the saveProduct procedure as a representative example.

ALGORITHM: saveProduct()

```

PROCEDURE saveProduct()
  id = getValue('productEditId')

  // 1. GATHER AND VALIDATE INPUTS
  name = TRIM(getValue('prodName'))
  price = PARSE_FLOAT(getValue('prodPrice'))
  stock = PARSE_INT(getValue('prodStock'))
  IF name == '' OR price < 0 OR stock < 0 THEN
    DISPLAY toast('Please fill all required fields', 'error')
    RETURN
  END IF

  payload = { name, price, stock, category, sku, description, is_active }

  // 2. DETERMINE OPERATION MODE
  IF supabase IS CONNECTED THEN
    IF id != '' THEN

```

```

        CALL supabase.update(payload).where(id == id)
    ELSE
        CALL supabase.insert(payload)
    END IF
    IF error THEN DISPLAY toast(error.message, 'error') RETURN
    DISPLAY toast('Product saved!', 'success')
ELSE // local storage mode
    IF id != '' THEN
        FIND product in productsCache WHERE product.id == id
        MERGE payload into found product
    ELSE
        payload.id = generateId('p')
        PREPEND payload to productsCache
    END IF
    WRITE productsCache to localStorage['ri_products']
END IF

// 3. UPDATE UI
CLOSE modal('productModal')
CALL renderProducts(productsCache)
CALL loadDashboard()
END PROCEDURE

```

1.7 Toast Notification Algorithm

The toast system provides feedback on all user actions. Toasts appear, remain visible for 3.5 seconds, then animate out and are removed from the DOM.

ALGORITHM: showToast(message, type)

```

PROCEDURE showToast(message, type)
    // type is one of: 'success', 'error', 'info'
    icons = { success: '✓', error: 'x', info: 'i' }

    toast = CREATE new div element
    SET toast.className = 'toast ' + type
    SET toast.innerHTML = icon + message

    APPEND toast to toastContainer

    WAIT 3500 milliseconds
    ADD class 'leaving' to toast // triggers CSS fade-out
    WAIT 300 milliseconds
    REMOVE toast from DOM
END PROCEDURE

```

2. Interface Style Guide

A consistent style guide is applied across all screens to ensure the platform feels cohesive and professional. Consistency reduces cognitive load and helps users navigate confidently between different sections of the platform.

Element	Specification	Justification
Primary Colour	#1F3864 (Deep Blue)	Creates a professional, trustworthy appearance appropriate for a retail business. Used for headings, navigation bar, and primary buttons.
Accent Colour	#2E5FAC (Mid Blue)	Used for secondary headings, active tab indicators, and interactive elements. Creates clear visual hierarchy without overwhelming contrast.
Background	#F0F2F5 (Light Grey)	Provides a neutral, non-fatiguing background. White content cards sit on top, creating depth and clear content zones.
Success / Error / Info	#28A745 / #DC3545 / #007BFF	Standard traffic-light colour coding for toast notifications. Universally understood by users without requiring text labels alone.
Primary Font	Arial, sans-serif	Universal font with excellent legibility at all sizes. System font avoids external loading latency. WCAG-compliant at all sizes used.
Font Sizes	32px (H1), 26px (H2), 22px (body), 18px (code/small)	Sufficient size hierarchy to guide users through content. 22px body text exceeds WCAG 2.1 minimum for readability.
Button Style	Rounded corners (6px), padding 8px 16px, hover with darkened colour	Rounded buttons are widely associated with interactive elements in modern UI. Hover state provides affordance for desktop users.
Spacing	8px base unit; sections separated by 24px	Consistent spacing creates visual rhythm. 8px grid aligns with Material Design conventions familiar to most web users.
Icons / Status Badges	Text-based badges with background colour (e.g. green 'Active', orange 'pending')	Colour-coded badges communicate status at a glance. Text is retained inside badges to remain accessible when colour alone is insufficient.
Navigation	Top bar + horizontal tab bar. Active tab underlined in blue.	Horizontal tab navigation is a well-established convention for web applications. The active indicator makes the user's current location unambiguous.
Responsive Design	Mobile: single column, stacked cards. Desktop: multi-column grid. Nav collapses on small screens.	Retail customers access services on mobile devices. Responsive design ensures equal functionality on all screen sizes.

3. Wireframe Designs

All designs were planned using annotated wireframe sketches before implementation. Each design documents the layout, hierarchy of elements, and key user interactions. The wireframes are presented below as structured descriptions, as Miro was used for the visual versions.

3.1 Authentication Screen

WIREFRAME: Login / Register Screen

[RI] Retail Innovations
Sign in to your account

Email
[_____]

Password
[_____]

[Sign In (primary btn)]

Don't have an account? [Create one]

• Signed out

Design Decisions — Authentication Screen

- Centred card layout with brand logo at top creates a professional first impression and follows established convention for web application login pages.
- Toggle between Login and Register modes reduces the number of screens, keeping the authentication flow in one place and reducing confusion.
- Register mode reveals additional fields (Full Name, Confirm Password) inline rather than navigating away, maintaining context.
- Status indicator at the bottom (green dot when connected, grey when not) provides technical transparency for admin users checking Supabase connectivity.
- Enter key triggers authentication form submission, supporting keyboard-only users (accessibility requirement).
- Animated background pattern is decorative only and adds no semantic content, so it does not require alt text.

3.2 Main Application Layout

WIREFRAME: Main App Shell (Desktop)

[RI] Retail Innovations	[Avatar] Jane	[Out]	← Top Bar		
[Dashboard]	[Products]	[Customers*]	[Orders]	[Loyalty]	← Nav Tabs (* admin)
[Active Section Panel]					

Design Decisions — Navigation

- Top bar persists across all sections, providing consistent access to brand identity, user information, and sign-out — all expected locations per convention.
- Tab-based navigation is a well-established convention for dashboard applications (e.g. Google Analytics, Shopify). Users can jump directly to any section without back/forward navigation.
- The Customers tab is admin-only and hidden via CSS when the body lacks the 'is-admin' class. This ensures role separation is enforced visually as well as functionally.
- Active tab is underlined to provide clear location indication without requiring colour alone (accessible to colour-blind users).

3.3 Dashboard Panel

WIREFRAME: Dashboard (Admin View)

Dashboard [↻ Refresh] Retail analytics at a glance			
Products 10 Active	Customers* 24 Registered	Orders 6 All ord.	Revenue* £1,247.50 Total revenue
Sales by Category (Bar) ■ ■ ■ ■ ■ ■ ■ ■ ■ ■		Customer Tiers (Donut) ● Donut ring Legend: Br Si Go Pl	


Design Decisions — Dashboard

- The four stat cards are arranged in a responsive grid. On mobile, they stack vertically. On desktop, they appear in a row. This is achieved with CSS Grid.
- Customers card and Revenue card are admin-only. Regular customers see their personal order count and spend instead, maintaining relevance without overwhelming non-admin users.
- The bar chart is rendered using pure CSS and JavaScript (no external library) for performance. Each bar's height is proportional to the category count, normalised to 100%.
- The donut chart uses a CSS conic-gradient to avoid the need for an SVG or Canvas library. The centre shows the total customer count.
- Charts are admin-only, as customer tier distribution and revenue data are commercially sensitive.

3.4 Products Panel

WIREFRAME: Products Panel

Products [+ Add Product (admin)] Browse our product catalogue [🔍 Search products...] [Category ▼]				
Product (img+nm)	Category	Price	Stock	Status [Active]

Org Green Tea 	Beverages	£12.99	150	[Active] [Edit][X]*
10 products shown		* = admin only		

Design Decisions — Products Panel

- Search and category filter are placed in a toolbar immediately above the data table, which is the expected position for filtering controls in web applications (e.g. eBay, Amazon).
- Real-time filtering via JavaScript (filterProducts()) updates the table on every keystroke without a page reload, providing instant feedback.
- Product images are displayed as small thumbnails (40x40px) beside the product name, aiding visual identification without taking excessive space.
- Low stock indicator (orange badge when stock < 10) draws admin attention to items needing restocking.
- The Actions column (Edit/Delete) is admin-only, rendered via JavaScript based on the isAdmin flag. The column header is present in the HTML but hidden for customers.
- The Add Product button is styled differently from the navigation tabs to indicate it is an action (primary button colour) rather than navigation.

3.5 Customer Management Panel

WIREFRAME: Customers Panel (Admin Only)

<div>Customers [+ Add Customer]</div> <div>Customer management & loyalty tracking</div> <div> [🏆 Bronze] [🥈 Silver] [🥇 Gold] [💎 Pl] </div> <div> [0-499 pts] [500-999pts] [1000-1999] [2000+] </div> <div> [🔍 Search customers...] [Tier ▼] </div>				
Customer	Email	Pts	Tier	Spent
Jane D	jane@..	1200	[Gold]	£450.00
Actions:			[Edit]	[Delete]

Design Decisions — Customers Panel

- The loyalty tier visual at the top of the panel provides immediate context for the tier system before any data is displayed. This is more informative than a legend tucked away elsewhere.
- Tier cards use emoji icons (🏆 🥈 🥇 💎) which are universally recognisable representations of ranking and are available without icon libraries.
- The tier filter dropdown allows admin staff to quickly identify all customers in a specific tier, supporting targeted marketing and reward management.
- Colour-coded tier badges in the table (bronze/silver/gold/platinum CSS classes) provide at-a-glance tier identification without requiring column sorting.

3.6 Add/Edit Product Modal

WIREFRAME: Product Modal (Add/Edit)

Add Product	
Product Name *	[_____]
Description	[_____]
Price (£) *	[_____]
Stock Quantity *	[_____]
Category *	[Category ▼]
SKU *	[_____]
Image URL	[_____]
Active?	[] Checkbox
[Cancel]	[Save Product]

(* = required field)

Design Decisions — Modal Forms

- Modals are used for data entry rather than separate pages to maintain context — the user can see the table behind the modal and does not lose their position.
- Required fields are marked with asterisks and validated before submission, providing clear feedback without allowing invalid data to reach the database.
- The modal closes on overlay click or Cancel button press, following standard modal behaviour that users expect from any modern web application.
- The Save button is labelled with a specific action ('Save Product', 'Save Customer') rather than a generic 'Submit', making the consequence of clicking clear.
- Edit mode pre-populates all fields with the existing record's data, preventing the user from having to re-enter unchanged information.

4. Data Requirements

4.1 Entity Relationship Diagram (ERD) — Description

The database is designed as a normalised relational model with seven tables. All tables satisfy Third Normal Form (3NF) — no partial dependencies or transitive dependencies exist. The relationships are described below:

- user_profiles (1) — (1) auth.users: One profile per authenticated user. The id in user_profiles is a foreign key referencing Supabase's built-in auth.users table.
- customers (1) — (0..1) auth.users: A customer record may or may not be linked to an authenticated user account. A Supabase user may have one customer record.
- orders (M) — (1) customers: A customer can place many orders. Each order belongs to one customer.
- order_items (M) — (1) orders: An order contains one or more order items. Each item belongs to one order (CASCADE delete).
- order_items (M) — (1) products: Each order item references one product. Products can appear in many order items.
- loyalty_transactions (M) — (1) customers: A customer can have many loyalty point transactions (earned/redeemed).
- loyalty_transactions (M) — (1) orders: Each transaction may be linked to an order (nullable — some transactions may be manual adjustments).
- loyalty_rewards is a standalone lookup table — it defines the available rewards customers can redeem but has no foreign key relationships to other tables in the current schema.

4.2 Data Dictionary

The following data dictionary documents all fields across all seven tables. Naming conventions follow lower_snake_case throughout. Primary keys use the suffix _id. Foreign keys mirror the name of the referenced primary key. Boolean fields use the is_ prefix.

Table: user_profiles

Field Name	Data Type	Constraints	Validation Rules	User Feedback / Notes
id	UUID (PK)	PRIMARY KEY, REFERENCE S auth.users ON DELETE CASCADE	Auto-generated (Supabase Auth)	N/A — system generated
email	TEXT	NOT NULL	Must be valid email format	Populated from auth.users on account creation. Unique via auth system.
full_name	TEXT	NULLABLE	Alphanumeric, max 100 chars recommended	Optional. Falls back to email prefix in UI if empty.
role	TEXT	DEFAULT 'customer', CHECK (role IN	Must be 'customer' or 'admin'	Set to 'customer' on registration. Changed to 'admin' via SQL Editor by system administrator.

Field Name	Data Type	Constraints	Validation Rules	User Feedback / Notes
		('customer', 'admin'))		
created_at	TIMESTAMPTZ	DEFAULT now()	Auto-populated	N/A — system generated

Table: products

Field Name	Data Type	Constraints	Validation Rules	User Feedback / Notes
id	UUID (PK)	PRIMARY KEY, DEFAULT gen_random_uuid()	Auto-generated	N/A — system generated
name	TEXT	NOT NULL	Required, max 255 chars	'Product name is required.' Shown on save validation failure.
description	TEXT	NULLABLE	Optional, plain text	No validation. Accepts any text.
price	NUMERIC(10,2)	NOT NULL, CHECK (price >= 0)	Required, numeric, >= 0	'Price must be a positive number.' Prevents negative price entry.
category	TEXT	NOT NULL	Required, selected from dropdown	Dropdown populated dynamically from existing categories. New categories can be typed.
image_url	TEXT	NULLABLE	Must be valid URL if provided	No forced validation — invalid URLs result in broken thumbnail (handled gracefully in UI).
stock_quantity	INTEGER	NOT NULL, DEFAULT 0, CHECK (stock_quantity >= 0)	Required, integer >= 0	'Stock cannot be negative.' Enforced at DB and UI level.
sku	TEXT	UNIQUE NOT NULL	Required, unique across all products	'SKU already exists.' Error shown if duplicate SKU is entered.
is_active	BOOLEAN	DEFAULT TRUE	TRUE or FALSE	Checkbox in product form. Inactive products remain in DB but are visually marked Inactive.
created_at / updated_at	TIMESTAMPTZ	DEFAULT now() / Trigger	Auto-populated	updated_at maintained by trg_products_updated trigger.

Table: customers

Field Name	Data Type	Constraints	Validation Rules	User Feedback / Notes
id	UUID (PK)	PRIMARY KEY, DEFAULT gen_random_uuid()	Auto-generated	N/A
user_id	UUID (FK)	REFERENCE S auth.users ON DELETE SET NULL, NULLABLE	Must match existing auth user or be NULL	Null for customers added manually by admin without an account.
email	TEXT	UNIQUE NOT NULL	Required, valid email format	'A customer with this email already exists.' Unique constraint prevents duplicate records.
full_name	TEXT	NOT NULL	Required, max 100 chars	'Full name is required.'
phone	TEXT	NULLABLE	Optional, no format enforced	Optional field. No mask enforced in this version — future improvement.
loyalty_points	INTEGER	DEFAULT 0, CHECK (loyalty_points >= 0)	Integer >= 0	'Points cannot be negative.' Tier auto-calculated from this value.
loyalty_tier	TEXT	DEFAULT 'Bronze', CHECK (tier IN ('Bronze', 'Silver', 'Gold', 'Platinum'))	Must be valid tier name	Auto-set by application logic based on loyalty_points. Cannot be set to invalid tier.
total_spent	NUMERIC(12,2)	DEFAULT 0.00	Numeric >= 0	Updated when orders are placed. Displayed in customer table.

Table: orders

Field Name	Data Type	Constraints	Validation Rules	User Feedback / Notes
id	UUID (PK)	PRIMARY KEY, DEFAULT gen_random_uuid()	Auto-generated	N/A
customer_id	UUID (FK)	REFERENCE S customers ON DELETE SET NULL	Must match existing customer or be NULL	Set NULL on delete to preserve order history even if customer is removed.

Field Name	Data Type	Constraints	Validation Rules	User Feedback / Notes
user_id	UUID (FK)	REFERENCE S auth.users ON DELETE SET NULL	Must match auth user or be NULL	Enables Row Level Security policy 'Users see own orders'.
total_amount	NUMERIC(10,2)	NOT NULL, CHECK (total_amount >= 0)	Required, numeric >= 0	'Order total must be a positive number.'
status	TEXT	DEFAULT 'pending', CHECK (status IN ('pending','processing','shipped','delivered','cancelled'))	Must be valid status	Dropdown in order form restricts input to valid values. Invalid status rejected by DB constraint.
payment_method	TEXT	NULLABLE	Optional, text	Descriptive field. No format enforcement in this prototype.
notes	TEXT	NULLABLE	Optional, plain text	Admin notes field. Not shown to customers.

Table: order_items

Field Name	Data Type	Constraints	Validation Rules	User Feedback / Notes
id	UUID (PK)	PRIMARY KEY, DEFAULT gen_random_uuid()	Auto-generated	N/A
order_id	UUID (FK)	REFERENCE S orders ON DELETE CASCADE	Required, must reference existing order	CASCADE delete ensures items are removed when their order is deleted.
product_id	UUID (FK)	REFERENCE S products ON DELETE SET NULL, NULLABLE	Must reference existing product or NULL	Set NULL if product is deleted, preserving order history.
quantity	INTEGER	NOT NULL, CHECK (quantity > 0)	Required, integer > 0	'Quantity must be at least 1.' Prevents zero-quantity line items.
unit_price	NUMERIC(10,2)	NOT NULL, CHECK (unit_price >= 0)	Required, numeric >= 0	Snapshot of price at time of purchase. Unaffected by future product price changes.
subtotal	NUMERIC(10,2)	GENERATED	Computed	Automatically calculated.

Field Name	Data Type	Constraints	Validation Rules	User Feedback / Notes
	0,2)	ALWAYS AS (quantity * unit_price) STORED	column — read only	Cannot be set manually.

Table: loyalty_rewards

Field Name	Data Type	Constraints	Validation Rules	User Feedback / Notes
id	UUID (PK)	DEFAULT gen_random_ uuid()	Auto-generated	N/A
name	TEXT	NOT NULL	Required, max 100 chars	'Reward name is required.'
points_required	INTEGER	NOT NULL, CHECK (points_required > 0)	Required, integer > 0	'Points required must be greater than zero.'
reward_type	TEXT	NOT NULL, CHECK (reward_type IN ('discount_percent', 'discount_fixed', 'free_product', 'free_shipping'))	Must be valid type	Dropdown restricts input to valid options. DB constraint enforces same.
reward_value	NUMERIC(10,2)	NOT NULL	Numeric >= 0. 0 for free_shipping type.	Stored as percentage for discount_percent (e.g. 10 = 10%) and as pound amount for discount_fixed.
is_active	BOOLEAN	DEFAULT TRUE	TRUE or FALSE	Inactive rewards remain in DB but are not offered to customers.

Table: loyalty_transactions

Field Name	Data Type	Constraints	Validation Rules	User Feedback / Notes
id	UUID (PK)	DEFAULT gen_random_ uuid()	Auto-generated	N/A
customer_id	UUID (FK)	REFERENCE S customers ON DELETE CASCADE	Required, must reference existing customer	Cascade delete removes transaction history when a customer is deleted.

Field Name	Data Type	Constraints	Validation Rules	User Feedback / Notes
points_change	INTEGER	NOT NULL	Non-zero integer. Positive = earned, Negative = redeemed.	Positive values represent points earned. Negative values represent redemption.
reason	TEXT	NOT NULL	Required, descriptive text	Human-readable description e.g. 'Order #ABC purchase' or 'Free Shipping reward redeemed'.
order_id	UUID (FK)	REFERENCE S orders ON DELETE SET NULL, NULLABLE	Optional FK to orders	Links the transaction to the triggering order. NULL for manual adjustments.
created_at	TIMESTAMPZ	DEFAULT now()	Auto-populated	Provides full audit trail of all points activity.

4.3 Naming Conventions

All field and table names follow lower_snake_case throughout the schema. This convention is:

- Consistent: the same style is used across all seven tables without exception.
- Descriptive: field names are unambiguous (e.g. loyalty_points rather than pts or points, stock_quantity rather than qty).
- Predictable: primary keys always use the _id suffix; foreign keys mirror the name of the referenced primary key (e.g. order_id in order_items references id in orders).
- Boolean fields use the is_ prefix (is_active, is_premium) to immediately communicate true/false semantics.
- Timestamp fields use created_at and updated_at consistently across all tables that need audit trails.

4.4 Error-Handling Strategy

The solution implements a two-layer approach to error handling:

Client-Side Validation (JavaScript)

- Required field checks run on form submission before any database call. Empty required fields trigger an immediate toast error and halt submission.
- Numeric fields (price, stock, loyalty_points, points_required) are parsed using parseFloat/parseInt, and negative or NaN values trigger validation errors.
- The esc() utility function escapes all user-generated content rendered to the DOM, preventing cross-site scripting.

Server-Side / Database Enforcement

- All numeric constraints (price \geq 0, stock_quantity \geq 0, loyalty_points \geq 0, quantity $>$ 0) are enforced by PostgreSQL CHECK constraints, providing a second validation layer independent of the front-end.

- UNIQUE constraints on email (customers) and sku (products) prevent duplicate records even if client-side checks are bypassed.
- CHECK constraints on role, loyalty_tier, status, and reward_type enforce enumerated values, preventing invalid states from being stored.
- The order_items.subtotal computed column eliminates the risk of subtotal calculation errors by deriving the value from quantity and unit_price at the database level.
- Input masks: date fields use TIMESTAMPTZ which enforces ISO 8601 format at the database level. SKU fields are typed as TEXT but validated for non-emptiness at the UI level.