RETAIL INNOVATIONS LTD

Task 2: Solution & Functionality

HTML • CSS • JavaScript • Supabase • CRUD • Security

DPDD Occupational Specialism — Set Task

Document Version 1.0 | January 2025

1. HTML Page Templates Created

The Retail Innovations platform is built as a single-page application (SPA) within one primary HTML file (index.html, 502 lines). This architecture eliminates full page reloads, providing a seamless user experience. The HTML structure uses semantic elements and is organised into clearly labelled sections using HTML comment blocks (e.g., <!-- ==================== SECTION ==================== -->).

The index.html file contains the following logical page templates, each controlled by JavaScript tab navigation:

Template	HTML Lines	Key Elements	Visibility
Authentication Screen	16–76	Supabase config fields, login form, register form, status indicator	Shown by default; hidden after login
Dashboard Panel	112–162	4 KPI stat cards, bar chart container, donut chart container	Default active tab; charts admin-only
Products Panel	165–198	Search bar, category dropdown, product data table, Add Product button	Visible to all; actions admin-only
Customers Panel	201–237	Tier visual cards, search/filter, customer data table	Admin-only (class admin-only on section)
Orders Panel	240–278	Status filter, order data table, customer column admin-only	All users; admin sees all orders
Loyalty Panel	281–298	Rewards data table, Add Reward button admin-only	Visible to all; CRUD admin-only
Product Modal	306–356	Name, SKU, category, price, stock, description, image URL, active checkbox	Overlay triggered by JS
Customer Modal	359–402	Name, email, phone, loyalty points, tier select	Overlay triggered by JS
Order Modal	405–449	Customer select (admin), total, status, payment, notes	Overlay triggered by JS
Reward Modal	452–497	Name, description, type select, value, points, active checkbox	Overlay triggered by JS

The <head> section (lines 1–9) includes the viewport meta tag for responsive rendering, the external CSS link, and the Supabase JavaScript client library loaded from CDN with the async attribute to prevent render blocking. The <script src="app.js"> tag is placed at the bottom of <body> (line 499) to ensure the DOM is fully loaded before JavaScript executes.

2. CSS Responsive for Desktop/Tablet/Mobile

The styles.css file (766 lines) implements a comprehensive responsive design using CSS custom properties, Flexbox, and CSS Grid. The stylesheet defines 25+ CSS custom properties in :root (lines 7–44) for consistent theming across colours, fonts, spacing, borders, shadows, and transitions.

2.1 Desktop Layout (>900px)

- Stats grid: display: grid with grid-template-columns: repeat(auto-fit, minmax(220px, 1fr)) — automatically creates 2–4 columns depending on viewport width (line 589)
- Charts grid: grid-template-columns: 2fr 1fr — bar chart takes two-thirds, donut chart one-third (line 616)
- Loyalty tier cards: display: flex in a single horizontal row (line 643)
- Main content padding: 32px with max-width: 1400px centred (lines 354–358)
- Top bar: sticky positioning with backdrop-filter: blur(16px) for depth effect (lines 195–206)
- Data tables: full-width with comfortable 14px 18px cell padding (line 519)

2.2 Tablet Breakpoint (≤900px, line 746)

- Charts grid collapses: grid-template-columns: 1fr — stacks vertically
- Form rows stack: grid-template-columns: 1fr — inputs go full width
- Loyalty tier cards: flex-wrap: wrap — wraps to 2x2 grid
- Stats grid: grid-template-columns: 1fr 1fr — forced 2-column
- Padding reduces: 32px → 16px throughout
- Section titles: font-size reduces from 2rem to 1.5rem

2.3 Mobile Breakpoint (≤600px, line 756)

- Stats grid: grid-template-columns: 1fr — single column stack
- Toolbar: flex-direction: column — search and filter stack vertically
- Search box: min-width: 100% — takes full available width
- Section headers: flex-direction: column; align-items: flex-start — title and button stack
- User info: display: none — hidden to save space; avatar still visible
- Auth card padding: reduces from 40px 36px to 28px 20px
- Table cell padding: reduces from 14px 18px to 10px 12px

3. Clear User Interface and Navigation

Navigation is implemented as a horizontal tab bar (lines 100–106 of index.html) with five tabs: Dashboard, Products, Customers, Orders, and Loyalty. The Customers tab has the class admin-only, making it invisible to regular users via the CSS rule at line 171: body:not(.is-admin) .admin-only { display: none !important; }.

The initNavigation() function (app.js lines 262–285) adds click event listeners to all .nav-tab elements. When clicked, it removes the active class from all tabs and panels, then adds active to the clicked tab and its corresponding section panel (matched via data-tab attribute → panel-{tab} ID pattern). A fadeSlideIn CSS animation (lines 367–370) provides smooth visual transitions.

The active tab indicator is a 2px bottom border in the accent colour (#3EEBBE), implemented with a ::after pseudo-element (lines 339–348). The top bar (lines 82–97) provides persistent context: brand logo (RI gradient icon), user avatar (first letter of name), display name, role badge (golden for admin), and Sign Out button. This ensures the user always knows who they are and can exit at any time.

4. All Required Pages Built in HTML

The Retail Innovations brief (Brief 05) specifies three key features. Each has been implemented as a functional page within the SPA:

Requirement 1 — Enhanced Product Search and Filter: The Products panel (lines 165–198) provides a search bar with real-time filtering by product name or SKU, and a dynamic category dropdown populated from the database. Products display in a data table with thumbnail images, name, SKU, category, price, stock quantity, and active/inactive status badges.

Requirement 2 — Retail Analytics Dashboard: The Dashboard panel (lines 112–162) displays four KPI stat cards (Total Products, Total Customers, Orders, Revenue) and two visualisation charts: a bar chart showing product distribution by category and a donut chart showing customer loyalty tier distribution. These charts are built with pure CSS and JavaScript — no external charting libraries.

Requirement 3 — Standard Loyalty Programmes: The Loyalty panel (lines 281–298) displays available rewards in a structured table. The Customers panel (lines 201–237) includes a visual tier system (Bronze/Silver/Gold/Platinum) with point ranges. The database supports loyalty_rewards and loyalty_transactions tables for complete programme management.

Additionally, the authentication page (lines 16–76) handles both login and registration with real-time Supabase connection status, and all four modal dialogs provide Create/Edit functionality for Products, Customers, Orders, and Rewards.

5. JavaScript Form Validation

Client-side validation is implemented in app.js for every form in the application. Each validation triggers a descriptive toast notification explaining the error:

5.1 Authentication Validation

Email & Password Presence (lines 104–107): if (!email || !password) triggers toast 'Please enter email and password.' Both fields must contain values before submission.

Password Length (line 138): if (password.length < 6) triggers toast 'Password must be at least 6 characters.' Enforces minimum security standard.

Password Match (line 134): if (password !== confirm) triggers toast 'Passwords do not match.' Prevents mistyped passwords during registration.

Duplicate Email (line 154): if (data.user && !data.user.identities?.length) triggers toast 'An account with this email already exists.' Detects duplicates from Supabase response.

Supabase Connection (lines 96–99): if (!supabase) triggers toast 'Connect to Supabase first.' Prevents auth attempts before connection is established.

5.2 Product Form Validation

Required Fields (line 420): if (!payload.name || !payload.sku || !payload.category) triggers toast 'Fill in Name, SKU, and Category.' Three fields are mandatory.

Price Parsing (line 414): parseFloat(value) || 0 ensures price is always a valid number, defaulting to 0 if invalid.

Stock Parsing (line 415): parseInt(value) || 0 ensures stock is always a valid integer.

5.3 Customer Form Validation

Required Fields (line 510): if (!payload.full_name || !payload.email) triggers toast 'Fill in Name and Email.' Both are mandatory.

Points Default (line 507): parseInt(value) || 0 provides safe default for loyalty points.

5.4 Order Form Validation

Total Required (line 632): if (!payload.total_amount) triggers toast 'Enter the order total.' Prevents zero-value orders.

Status Lock (line 606): document.getElementById('orderStatus').disabled = !isAdmin prevents customers from changing order status.

5.5 Reward Form Validation

Required Fields (line 717): if (!payload.name || !payload.points_required) triggers toast 'Fill in Name and Points Required.'

Admin Check (line 693): if (!isAdmin) triggers toast 'Admin access required.' Prevents non-admin access to reward creation.

6. Supabase Database Created and Schema Correct

The database schema (schema.sql, 199 lines) defines seven tables in Supabase (hosted PostgreSQL), with comprehensive constraints, indexes, triggers, functions, and Row Level Security policies.

6.1 Table Definitions

Table	Primary Key	Key Columns	Constraints	Indexes
user_profiles	UUID (refs auth.users CASCADE)	email TEXT NOT NULL, full_name TEXT, role TEXT	CHECK role IN ('customer','admin')	idx_profiles_role
products	UUID (gen_random_uuid())	name, description, price, category, sku, stock_quantity, image_url, is_active	price CHECK >= 0, stock CHECK >= 0, sku UNIQUE NOT NULL	idx_products_category
customers	UUID (gen_random_uuid())	user_id FK, email UNIQUE, full_name, phone, loyalty_points, loyalty_tier, total_spent	points CHECK >= 0, tier CHECK IN 4 values	idx_customers_email, idx_customers_user
orders	UUID (gen_random_uuid())	customer_id FK, user_id FK, total_amount, status, payment_method, notes	total CHECK >= 0, status CHECK IN 5 values	idx_orders_customer, idx_orders_user, idx_orders_status
order_items	UUID (gen_random_uuid())	order_id FK CASCADE, product_id FK, quantity, unit_price	quantity CHECK > 0, unit_price CHECK >= 0, subtotal GENERATED	None
loyalty_rewards	UUID (gen_random_uuid())	name, description, points_required, reward_type, reward_value, is_active	points CHECK > 0, type CHECK IN 4 values	None
loyalty_transactions	UUID (gen_random_uuid())	customer_id FK CASCADE, points_change, reason, order_id FK	None (all nullable except customer_id, points, reason)	None

6.2 Triggers and Functions

handle_new_user() (lines 21–32): AFTER INSERT trigger on auth.users. Automatically creates a user_profiles row when someone registers via Supabase Auth. Uses COALESCE to extract full_name and role from raw_user_meta_data, defaulting to empty string and 'customer'. Defined as SECURITY DEFINER to access auth schema.

update_updated_at() (lines 122–127): BEFORE UPDATE trigger applied to products, customers, and orders tables. Automatically sets the updated_at column to now() whenever a row is modified. Ensures timestamp accuracy without client-side involvement.

is_admin() (lines 135–141): Helper function that returns BOOLEAN by checking if auth.uid() has role='admin' in user_profiles. Defined as SECURITY DEFINER so RLS policies can safely call it without granting users direct SELECT on profiles.

6.3 Row Level Security (RLS)

All seven tables have RLS enabled (lines 147–153). 17 policies are defined:

- user_profiles: Users see and update own profile; admins see all
- products: Anyone can SELECT (public catalogue); only admins can INSERT, UPDATE, DELETE
- customers: Admins see all; users see only their own (matched by user_id = auth.uid())
- orders: Admins see all + manage all; users see only own orders, can only update while status = 'pending'
- order_items: Visible if user owns the parent order or is admin; admin manages all
- loyalty_rewards: Public read; admin write
- loyalty_transactions: Visible if user owns the parent customer record or is admin

6.4 Seed Data

The schema includes seed data (lines 170–190): 10 products across 8 categories (Beverages, Bakery, Electronics, Homeware, Sports, Office, Food, Accessories) with realistic names, descriptions, prices, stock quantities, SKUs, and Unsplash image URLs. 4 loyalty rewards are seeded with varying point requirements (200–2000) and types (discount_percent, discount_fixed, free_shipping). ON CONFLICT (sku) DO NOTHING prevents duplication on re-runs.

7. JavaScript CRUD Linked to Supabase

The app.js file (798 lines) implements complete Create, Read, Update, Delete operations for all four primary entities. All operations use the Supabase JavaScript client and follow a consistent pattern: try-catch with toast notifications, cache update, and UI re-render.

7.1 Products CRUD

READ — loadProducts() (line 333): supabase.from('products').select('*').order('created_at', { ascending: false }). Stores results in productsCache array. Calls renderProducts() and populateCategoryFilter().

CREATE — saveProduct() (line 408): Builds payload from form fields. If no edit ID exists: supabase.from('products').insert(payload).select(). Validates Name, SKU, Category required.

UPDATE — saveProduct() (line 422): If edit ID exists: supabase.from('products').update(payload).eq('id', id).select(). Same function handles both create and update.

DELETE — deleteProduct() (line 432): window.confirm() prompt, then supabase.from('products').delete().eq('id', id). Refreshes products and dashboard.

7.2 Customers CRUD (Admin Only)

READ — loadCustomers() (line 446): Guarded by if (!isAdmin) return. supabase.from('customers').select('*').order('joined_at', { ascending: false }). Renders into customersCache.

CREATE/UPDATE — saveCustomer() (line 501): Validates full_name and email. INSERT or UPDATE based on customerEditId. Refreshes customer list and dashboard.

DELETE — deleteCustomer() (line 522): Confirm dialog with customer name. Deletes and refreshes.

7.3 Orders CRUD

READ — loadOrders() (line 538): supabase.from('orders').select('*, customers(full_name)'). Uses a JOIN to get customer names. If !isAdmin, adds .eq('user_id', currentUser.id) as additional filter alongside RLS.

CREATE — saveOrder() (line 613): Sets user_id = currentUser.id on new orders. Admin can assign customer_id via dropdown. Validates total > 0.

UPDATE — saveOrder() (line 635): Updates total, status, payment, notes. Non-admin status dropdown is disabled.

DELETE — deleteOrder() (line 645): Admin-only delete with confirmation.

7.4 Rewards CRUD

READ — loadRewards() (line 659): supabase.from('loyalty_rewards').select('*').order('points_required', { ascending: true }). Ordered by accessibility.

CREATE/UPDATE — saveReward() (line 707): Validates name and points_required. INSERT or UPDATE based on rewardEditId.

DELETE — deleteReward() (line 729): Confirm with reward name. Admin-only.

7.5 Parallel Data Loading

loadAllData() (line 318): Uses Promise.all([loadProducts(), isAdmin ? loadCustomers() : Promise.resolve(), loadOrders(), loadRewards()]) to fetch all data in parallel, followed by loadDashboard(). This minimises total load time by executing queries concurrently.

8. Login Functionality Working

Authentication is implemented as a three-stage flow using Supabase Auth:

8.1 Stage 1 — Supabase Connection (lines 28–58)

The user enters their Supabase project URL and anon key into the configuration form (index.html lines 26–37). initSupabaseFromConfig() validates both fields are present, checks the Supabase library has loaded (typeof window.supabase !== 'undefined'), creates the client with window.supabase.createClient(url, key), updates the status dot from red to green (.connected class), hides the config form, shows the login form, and calls checkExistingSession() to auto-login if a session exists.

8.2 Stage 2 — Login/Register (lines 95–172)

Login: handleLogin() calls supabase.auth.signInWithPassword({ email, password }). On success, sets currentUser and calls loadUserProfile() then enterApp().

Register: handleRegister() validates password length and match, then calls supabase.auth.signUp() with options.data containing full_name and role:'customer'. Handles both auto-confirmed (immediate login) and email-confirmation-required flows.

Mode Toggle: toggleAuthMode() (line 77) switches between login and register by toggling UI elements: subtitle text, submit button text, register fields visibility, and toggle link text.

8.3 Stage 3 — Session and Role Management (lines 192–256)

Profile Loading: loadUserProfile() queries user_profiles for the current user's row using .eq('id', currentUser.id).single(). Sets isAdmin = currentProfile.role === 'admin'. Falls back to defaults if profile not yet created.

App Entry: enterApp() hides auth screen (classList.add('hidden')), shows app wrapper, sets body class is-admin if applicable, updates user badge (avatar initial, name, role), adjusts UI labels for role context (e.g., 'Your orders' vs 'All customer orders'), and calls loadAllData().

Logout: handleLogout() (line 174) calls supabase.auth.signOut(), resets all state variables, removes is-admin class, shows auth screen, hides app.

9. Backend Code for FP1 — Product Search and Filter

Functional Page 1 implements the Enhanced Product Search and Filter requirement:

9.1 Data Loading

loadProducts() (line 333): Fetches all products from Supabase with .select('*').order('created_at', { ascending: false }). All products are stored in the productsCache global array, enabling instant client-side filtering without additional network requests.

9.2 Real-Time Search

filterProducts() (line 382): Triggered by the oninput event on the search input (index.html line 176). Reads the search value and category filter, then applies a .filter() on productsCache:

```
productsCache.filter(p =>
    (!s || p.name.toLowerCase().includes(s) ||
p.sku.toLowerCase().includes(s)) &&
    (!c || p.category === c)
)
```

The search matches against both product name and SKU (case-insensitive). The category filter uses exact match. An empty search or 'All Categories' shows everything. Results are immediately re-rendered via renderProducts().

9.3 Dynamic Category Filter

populateCategoryFilter() (line 376): Extracts unique categories from products using [...new Set(productsCache.map(p => p.category))].sort(). Generates <option> elements dynamically, ensuring the dropdown always reflects actual database categories.

9.4 Product Display

renderProducts() (line 344): Renders a data table with product thumbnail (Unsplash image with onerror fallback), name, SKU, category, formatted price (£X.XX), stock quantity, active/inactive badge, and admin Edit/Delete buttons. The esc() function sanitises all user content to prevent XSS. Empty state shows a box emoji and 'No products found' message.

10. Backend Code for FP2 — Analytics Dashboard

Functional Page 2 implements the Retail Analytics Dashboard requirement:

10.1 KPI Calculations

loadDashboard() (line 743): Calculates four KPIs from the cached data arrays:

```
statProducts = productsCache.length
statCustomers = customersCache.length
statOrders = ordersCache.length
revenue = ordersCache.reduce((s, o) => s + Number(o.total_amount || 0), 0)
```

Revenue is formatted with £ symbol and toLocaleString('en-GB', { minimumFractionDigits: 2 }) for proper UK currency formatting with thousand separators. For non-admin users, the labels adjust to 'Your orders' and 'Your spend' instead of totals.

10.2 Bar Chart — Sales by Category

renderBarChart() (line 757): Groups products by category using an accumulator object, sorts entries by count descending, takes the top 7 categories, and renders CSS-based proportional bars:

```
const max = Math.max(...entries.map(e => e[1]), 1);
style="height:${(count/max)*100}%"
```

Each bar uses a gradient from var(--color-accent) to transparent, with border-radius for rounded tops. Labels show the first 6 characters of each category name. A hover effect increases brightness. The chart container has a fixed 220px height.

10.3 Donut Chart — Customer Tiers

renderDonutChart() (line 772): Counts customers per loyalty tier (Bronze, Silver, Gold, Platinum), calculates proportional angles (360° total), and generates a CSS conic-gradient:

```
conic-gradient(#CD7F32 0deg 90deg, #C0C0C0 90deg 180deg, ...)
```

A centre overlay div displays the total customer count. A legend renders coloured dots with tier names and counts. Each tier uses its metallic colour: Bronze (#CD7F32), Silver (#C0C0C0), Gold (#FFD700), Platinum (#A0C8FF).

11. Backend Code for FP3 — Loyalty Programme

Functional Page 3 implements the Standard Loyalty Programme requirement:

11.1 Rewards Display

loadRewards() (line 659): Fetches all rewards from loyalty_rewards ordered by points_required ascending (most accessible rewards first). Stored in rewardsCache.

renderRewards() (line 668): Maps reward_type codes to human-readable labels using a lookup object: { discount_percent: 'Discount (%)', discount_fixed: 'Discount (£)', free_product: 'Free Product', free_shipping: 'Free Shipping' }. Values display as percentages or currency as appropriate. Points use toLocaleString() with 'pts' suffix.

11.2 Rewards CRUD

openRewardModal() (line 692): Admin guard check. Pre-populates all fields if editing, clears for creation. Opens modal overlay.

saveReward() (line 707): Constructs payload with name, description, reward_type, reward_value, points_required, is_active. Validates name and points. Submits via INSERT or UPDATE.

deleteReward() (line 729): Confirmation with reward name. Deletes from loyalty_rewards. Refreshes display.

11.3 Tier System

The Customers panel (index.html lines 209–213) displays four loyalty tier cards with visual hierarchy:

Tier	Points Range	Emoji	CSS Class	Background Colour
Bronze	0–499 pts	🥉	.tier-card.bronze	#CD7F32 at 6% opacity
Silver	500–999 pts	🥈	.tier-card.silver	#C0C0C0 at 6% opacity
Gold	1000–1999 pts	🥇	.tier-card.gold	#FFD700 at 6% opacity
Platinum	2000+ pts	💎	.tier-card.platinum	#A0C8FF at 6% opacity

11.4 Database Support

The loyalty_transactions table records every point change (earn or redeem) with a reason description and optional order_id link. The customers table tracks loyalty_points (current balance), loyalty_tier (current tier), and total_spent (lifetime value). CHECK constraints ensure points cannot go below zero.

12. Security Measures Implemented

12.1 Row Level Security (RLS)

All seven tables have RLS enabled via ALTER TABLE ... ENABLE ROW LEVEL SECURITY (schema.sql lines 147–153). 17 individual policies enforce data isolation at the database level. Even if client-side JavaScript were manipulated, the PostgreSQL policies would reject unauthorised operations. The is_admin() function is SECURITY DEFINER, meaning it executes with the privileges of the function creator, not the calling user.

12.2 XSS Prevention

esc() function (app.js line 790): Every user-supplied string is passed through this function before insertion into innerHTML. It creates a temporary DOM element, sets its textContent (which auto-escapes HTML entities), then reads innerHTML. This converts < to &lt;, > to &gt;, etc., preventing script injection.

```
function esc(text) {
    if (!text) return '';
    const d = document.createElement('div');
    d.textContent = text;
    return d.innerHTML;
}
```

12.3 Authentication Security

Password Hashing: Supabase Auth handles password hashing using bcrypt. Passwords are never stored, transmitted, or visible in plain text.

JWT Sessions: After authentication, Supabase issues a JSON Web Token. The token is stored in the browser and sent with every API request for verification.

Email Validation: Supabase validates email format and can require email confirmation before allowing login.

12.4 Safe Key Usage

The Supabase anon key is entered by the user at runtime via the configuration form (index.html lines 26–37) rather than being hardcoded in source code. This means the key is not exposed in the Git repository. The anon key only grants permissions defined by the RLS policies — it cannot bypass them.

12.5 Client-Server Role Enforcement

The isAdmin flag (app.js line 10) controls UI visibility, but all access control is also enforced server-side through RLS. This dual-layer approach means: if a customer modified the JavaScript to set isAdmin = true, the Supabase API would still reject admin operations because the database policies check auth.uid() against user_profiles.role, not the client-side variable.

12.6 Confirmation Dialogs

All destructive operations (deleteProduct, deleteCustomer, deleteOrder, deleteReward) require explicit window.confirm() confirmation, displaying the item name to prevent accidental deletion.