RETAIL INNOVATIONS

Task 1B — Design Documentation

DPDD Occupational Specialism — Set Task

UI/UX Design • Algorithm Design • Data Requirements • Test Strategy

Document Version: 1.0
Date: January 2025

Table of Contents

1. UI/UX Design

This section presents the wireframe designs for all five pages of the Retail Innovations web application. Each wireframe is accompanied by a detailed explanation of the design rationale, user experience considerations, and technical implementation approach. The wireframes follow a consistent design system to ensure visual cohesion across the entire application.

1.1 Wireframe of Home Page

The home page wireframe (see accompanying wireframe-home.html file) establishes the visual identity and primary navigation structure for the Retail Innovations platform. The page is designed using a single-column layout with clearly defined sections, following the established F-pattern reading model that users naturally follow when scanning web content.

1.1.1 Explanation of Home Page

The home page serves as the primary entry point for all users and is designed to accomplish three key objectives: introduce the platform, highlight core features, and drive user engagement through clear calls to action.

Navigation Bar: A fixed-position horizontal navigation bar spans the full width of the viewport. It contains the brand logo on the left (establishing brand identity immediately), core navigation links in the centre (Home, Products, Dashboard, Loyalty, About), and a prominent Login/Sign Up CTA button on the right. The navigation uses a contrasting background colour to ensure it remains visible during scrolling. On mobile devices, the navigation collapses into a hamburger menu icon to maintain usability on smaller screens.

Hero Section: A two-column layout occupies the first viewport. The left column contains a compelling headline, descriptive subtext explaining the platform value proposition, and two action buttons (primary: Browse Products, secondary: Learn More). The right column contains a hero image or illustration. This layout follows the inverted pyramid principle, presenting the most important information first. The primary CTA button uses a high-contrast dark colour against the lighter background to draw attention.

Feature Highlights Section: A three-column grid showcases the three core features of the platform: Product Search and Filter, Retail Analytics Dashboard, and Loyalty Programme. Each feature card includes an icon, a heading, and a brief description. This section uses equal-width columns to create visual balance and helps users immediately understand what the platform offers. The cards use subtle background shading to differentiate them from the main page background.

Promotional Banner: A full-width call-to-action section with a contrasting background colour creates visual separation. This section is designed to promote current offers, seasonal campaigns, or sign-up incentives (e.g., earning bonus loyalty points upon registration). The centralised text and single CTA button follow best practices for conversion-focused design.

Trending Products Grid: A four-column product grid displays currently popular items, each showing a product image, name, price, and star rating. This section serves dual purposes: providing social proof through popular items and encouraging product exploration. The grid is dynamically populated from the Supabase database, pulling products sorted by view count or sales volume.

Footer: A four-column footer provides secondary navigation, quick links, support information, and contact details. This is a standard convention that users expect and provides essential information without cluttering the main page content.

1.2 Wireframe of Login Page

The login page wireframe (see wireframe-login.html) implements a split-screen layout that balances branding with functionality. The left panel displays branding imagery while the right panel houses the authentication forms.

1.2.1 Explanation of Login Page

The login page has been designed with security, usability, and accessibility as primary concerns. It handles both authentication (login) and registration (sign up) through a tabbed interface, reducing the number of pages needed and keeping users in context.

Split-Screen Layout: The page is divided into two equal halves. The left side displays a branding illustration or promotional imagery, reinforcing the platform identity. The right side contains the authentication form. This pattern is widely used by modern web applications (e.g., Spotify, Slack) because it provides visual interest while keeping the form focused and uncluttered. On mobile devices, the branding panel is hidden to maximise form visibility.

Tab-Based Authentication: A toggle switch at the top of the form area allows users to switch between Login and Register modes without navigating to a separate page. This reduces friction in the authentication flow. The active tab is visually distinguished using a filled dark background, while the inactive tab uses a lighter style.

Login Form Fields: The login form requires two fields: Email Address and Password. Each field includes a descriptive label, placeholder text, and validation hints below the input. The password field masks input by default with an optional show/hide toggle for accessibility. A Remember Me checkbox is provided for returning users, and a Forgot Password link is positioned on the same row for easy access.

Registration Form Fields: When the Register tab is active, additional fields appear: Full Name, Email Address, Password, Confirm Password, Account Type (Customer or Retailer radio buttons), Terms and Conditions checkbox, and GDPR Consent checkbox. Each field includes client-side validation triggered on blur events, with red border highlighting and error messages for invalid inputs. A password strength meter provides real-time feedback on password security.

Client-Side Validation: JavaScript validation runs on every form field before submission. The validation checks include: email format (regex pattern matching), password strength (minimum 8 characters, requiring at least one uppercase letter, one lowercase letter, one number, and one special character), password match confirmation, and required field completion. The submit button remains disabled until all validation criteria are met, preventing invalid submissions.

Social Login Options: Google and Facebook social login buttons are provided as alternative authentication methods. A visual divider with an OR separator distinguishes social login from traditional form submission. These options use Supabase OAuth integration.

Security Notice: A security information box at the bottom of the form reassures users that their data is encrypted and stored in compliance with GDPR regulations. This builds trust and addresses potential privacy concerns.

1.3 Wireframe of Functional Page 1 (FP1) — Product Search and Browse

The Product Search and Browse page wireframe (see wireframe-fp1-product-search.html) is the primary customer-facing functional page, implementing advanced search, filtering, and product display capabilities.

1.3.1 Explanation of FP1

This page represents the core shopping experience of the platform. It combines powerful search and filtering with an intuitive product grid display, designed to help users efficiently find products that match their specific criteria.

Search Bar: A prominent full-width search bar sits at the top of the content area, allowing keyword-based product search. The search implements real-time suggestions as the user types (debounced at 300ms to reduce unnecessary API calls). Search queries are sent to Supabase using the ilike operator for partial matching against product names, descriptions, and categories. Below the search bar, popular search terms are displayed as clickable suggestions to aid product discovery.

Active Filter Tags: Below the search bar, applied filters are displayed as removable tags (pills). Each tag shows the filter value and an X button for removal. A Clear All option removes all active filters simultaneously. This provides clear visual feedback about which filters are currently active and allows quick modification.

Sidebar Filter Panel: A 280px-wide left sidebar contains multiple filter categories. Category Filter uses checkboxes with result counts in brackets, allowing multi-select. Price Range Filter uses both text inputs and a visual slider, enabling precise budget-based filtering. Rating Filter allows filtering by minimum star rating. Availability Filter shows in-stock versus out-of-stock options. An Apply Filters button sends the combined filter query to Supabase, and a Clear All Filters button resets all selections. On mobile, the sidebar converts to a slide-out drawer activated by a filter button.

Product Grid: Products are displayed in a responsive three-column grid layout. Each product card contains: a product image with optional badges (NEW, SALE), a wishlist heart icon, the category breadcrumb, product name, price (with strikethrough for discounted items), star rating with review count, and a View Details button. The grid layout collapses to two columns on tablet and a single column on mobile. Products are fetched from the Supabase products table with pagination (9 products per page).

Sorting Options: A sort dropdown allows users to reorder results by Relevance (default), Price Low to High, Price High to Low, Rating, and Newest First. A grid/list view toggle switches between card grid and list view layouts. These controls sit above the product grid for easy access.

Pagination: A numbered pagination component sits below the product grid, showing the current page, adjacent pages, and first/last page indicators. This is implemented using OFFSET and LIMIT in Supabase queries, with 9 products per page. The total result count is displayed above the grid.

1.4 Wireframe of Functional Page 2 (FP2) — Analytics Dashboard

The Analytics Dashboard wireframe (see wireframe-fp2-analytics.html) is the admin/retailer-facing page providing real-time business intelligence through data visualisation.

1.4.1 Explanation of FP2

The dashboard is designed exclusively for retailer account holders and presents aggregated sales, inventory, and customer data in an intuitive visual format. It follows dashboard design best practices: overview first, then detail on demand.

Dashboard Sidebar Navigation: A 220px-wide dark sidebar provides secondary navigation specific to the admin interface. It includes user profile information (avatar, name, role), and navigation items for Overview, Products, Orders, Customers, Revenue, Loyalty Programme, and Settings. The active section is highlighted with a left border accent and contrasting background. This sidebar pattern separates the admin experience from the customer-facing interface.

Date Range Selector: A row of preset date range buttons (Today, 7 Days, 30 Days, Custom) allows retailers to adjust the reporting period. The Custom option opens a date picker. All dashboard data dynamically updates when the date range changes, using parameterised Supabase queries filtered by the selected date range.

KPI Cards: Four key performance indicator cards display at the top: Total Revenue, Total Orders, Active Customers, and Average Order Value. Each card shows the current period value, the percentage change compared to the previous period (green for positive, red for negative), and a mini sparkline chart showing the trend. These provide immediate insight into business health at a glance. Data is aggregated from the orders and users tables in Supabase.

Revenue Line Chart: A full-width line chart displays revenue over time for the selected date range. The X-axis shows dates, and the Y-axis shows revenue in GBP. Data points are interactive, showing exact values on hover via tooltips. This chart is rendered using Chart.js or a similar JavaScript charting library, with data fetched from Supabase and aggregated by date using JavaScript reduce functions.

Category Doughnut Chart: A doughnut chart displays sales distribution by product category, with each segment representing a category percentage. A legend below identifies categories by colour. This visualisation helps retailers identify their strongest product categories and make informed inventory decisions.

Top Products Table: A sortable table lists the best-selling products with columns for Product Name, Category, Units Sold, and Revenue. A View All link navigates to the detailed products page. Data is queried from the orders table joined with the products table, grouped by product ID.

Inventory Alerts Table: A critical operational table shows products with low stock levels. Each row displays the product name, current stock count, and a colour-coded status badge (Critical: red, under 5 units; Low: orange, 5-15 units; Warning: yellow, 15-25 units; OK: green, over 25 units). This enables proactive inventory management.

Bottom Analytics Row: Three smaller chart panels display Customer Activity (new vs. returning customers bar chart), Loyalty Points Issued (area chart showing points distributed over time), and Page Views (line chart tracking which pages receive the most traffic). These provide secondary insights that complement the primary KPIs.

1.5 Wireframe of Functional Page 3 (FP3) — Loyalty Programme

The Loyalty Programme wireframe (see wireframe-fp3-loyalty.html) is the customer-facing rewards and engagement page, designed to drive retention and repeat purchases.

1.5.1 Explanation of FP3

This page serves as the loyalty hub for registered customers, displaying their points balance, available rewards, transaction history, tier status, and referral programme. It is designed to create a sense of achievement and motivate continued engagement.

Points Summary Banner: A dark-themed banner at the top provides an immediate overview of the user's loyalty status. It includes a circular points display showing the current balance, a welcome message with the user's name (fetched from Supabase auth), their current tier badge (Bronze, Silver, Gold, Platinum), and a progress bar showing advancement toward the next tier with specific points remaining. This gamification element encourages users to continue earning.

Tabbed Navigation: A horizontal tab bar allows users to switch between four views: Available Rewards, Points History, Tier Benefits, and Referrals. This keeps the page organised without requiring separate URLs. The active tab is indicated by an underline accent and bold text. Tab state is managed in JavaScript without page reload.

Rewards Grid: Available rewards are displayed in a three-column grid of reward cards. Each card shows a reward image, title, description, point cost, and a Redeem Now button. Cards are sorted by point cost ascending (most accessible first). If the user does not have sufficient points, the button changes to a disabled Not Enough Points state with a greyed-out appearance. Popular rewards display a badge indicator. A filter dropdown allows sorting by All Rewards, Affordable, or Premium. Rewards data is stored in a rewards table in Supabase.

Points History Table: A detailed transaction log displays all points activity, including Date, Description, Type (colour-coded badges: Purchase in green, Redeem in red, Bonus in orange), Points (positive values in green, negative in red), and Running Balance. The table is paginated and sorted by date descending (newest first). This provides full transparency and helps users track their earning and spending patterns.

Referral Programme: A highlighted referral box at the bottom encourages word-of-mouth marketing. It displays the user's unique referral code (generated on account creation), a Copy Code button for easy sharing, and a description of the referral reward (200 bonus points for both referrer and referee). The referral code is stored in the users table and tracked in the loyalty_transactions table.

1.6 Explanation of Layout Choices

The overall layout strategy for Retail Innovations follows a modular, component-based approach that ensures consistency, scalability, and responsiveness across all pages. The following layout principles have been applied throughout the design:

F-Pattern Layout: Research by the Nielsen Norman Group demonstrates that users scan web pages in an F-shaped pattern, reading horizontally across the top, then vertically down the left side. The homepage and product pages are structured accordingly, with the most important content (navigation, search, hero section) positioned at the top and left of the viewport.

Card-Based Design: Products, features, rewards, and KPI metrics all use a card-based layout. Cards provide clear visual boundaries between content items, are inherently responsive (reflowing from multi-column to single-column on smaller screens), and create a consistent interaction pattern across the application.

Consistent Navigation: A persistent top navigation bar appears on every page, ensuring users always have access to core pages regardless of their current location. The navigation highlights the active page to provide orientation. For the admin dashboard, a secondary sidebar navigation supplements the top bar without replacing it.

Responsive Design Strategy: The layout uses CSS Grid and Flexbox to create fluid, responsive layouts. Breakpoints are set at 768px (tablet) and 480px (mobile). On tablet, multi-column grids collapse from 3-4 columns to 2 columns. On mobile, all content becomes single-column, sidebar filters convert to slide-out drawers, and the navigation collapses to a hamburger menu. Media queries handle these transitions.

Visual Hierarchy: Typography sizing follows a clear hierarchy: page titles at 28-32px, section headings at 20-24px, subheadings at 16-18px, body text at 14px, and metadata/labels at 11-12px. This hierarchy guides users through content in the intended order of importance.

Whitespace and Spacing: Generous padding and margins are used throughout to prevent visual crowding. Section spacing of 30-40px creates clear separation between content blocks. Card padding of 15-25px ensures content does not feel cramped. This aligns with the principle that whitespace improves readability by up to 20%.

1.7 Colour Palette

The colour palette has been selected to convey professionalism, trust, and clarity, while maintaining WCAG 2.1 AA contrast compliance for accessibility.

Colour Name	Hex Code	Usage	Contrast Ratio
Primary Dark	#1B4F72	Navigation bar, primary headings, primary buttons	8.5:1 on white
Primary Medium	#2E75B6	Secondary headings, hover states, active links	4.8:1 on white
Primary Light	#D5E8F0	Table header backgrounds, info cards, subtle highlights	1.2:1 (decorative)
Accent Orange	#E65100	Sale badges,	5.2:1 on white

		promotional banners, warning indicators	
Success Green	#2E7D32	Positive change indicators, in-stock badges, earned points	5.9:1 on white
Error Red	#C62828	Validation errors, out-of-stock, spent points, critical alerts	6.1:1 on white
Neutral Dark	#333333	Body text, primary content	12.6:1 on white
Neutral Medium	#666666	Secondary text, descriptions, metadata	5.7:1 on white
Neutral Light	#F5F5F5	Page backgrounds, alternate table rows	N/A (background)
White	#FFFFFF	Card backgrounds, form inputs, primary surfaces	N/A (background)

The palette was tested using the WebAIM Contrast Checker to verify WCAG 2.1 AA compliance (minimum 4.5:1 contrast ratio for normal text, 3:1 for large text). All primary text colours exceed the minimum requirements. The choice of a blue-based primary palette conveys trust and reliability, which is particularly important for an e-commerce platform handling financial transactions and personal data.

2. Algorithm Design

2.1 Systems Overview

The Retail Innovations platform operates as a client-server web application with the following architecture: the front end is built with HTML, CSS, and JavaScript running in the user's browser; the back end uses Supabase (a hosted PostgreSQL database with built-in authentication and RESTful API). The system does not require a custom server, as Supabase provides the database, authentication, and API layer.

2.1.1 Inputs Identified

Input	Source	Data Type	Validation Required
User email/password	Login/Register forms	String	Format validation, length checks, strength
Search query text	Product search bar	String	Sanitisation, max length 200 chars
Filter selections	Sidebar filter panel	Array of strings/numbers	Whitelist validation against known categories
Sort preference	Sort dropdown	String (enum)	Must match allowed sort values
Product page number	Pagination controls	Integer	Must be positive, within valid range
Loyalty point redemption	Redeem button	Integer (reward ID)	Must exist in rewards table, user has enough points
Date range selection	Dashboard date picker	Date range	Start date must be before end date
Registration data	Registration form	Multiple strings	All fields validated per requirements

2.1.2 Process Description

The system processes follow a request-response cycle: the user interacts with the front-end interface, which triggers JavaScript functions. These functions construct Supabase queries (using the Supabase JavaScript client library), send them to the Supabase API, receive JSON responses, and update the DOM accordingly. All data processing occurs in two locations: client-side JavaScript for UI logic, validation, and data formatting; and server-side PostgreSQL for data storage, retrieval, and aggregation. Supabase Row Level Security (RLS) policies enforce access control at the database level.

2.1.3 Outputs Identified

Output	Destination	Format	Description

Authentication token	Browser localStorage/cookie	JWT string	Session management after successful login
Product list	Product grid DOM	JSON array rendered as HTML	Filtered, sorted, paginated product cards
Dashboard charts	Chart.js canvas elements	Rendered SVG/Canvas	Visual data representations
KPI values	Dashboard KPI cards	Formatted numbers	Aggregated metrics with percentage change
Points balance	Loyalty page banner	Integer	Current user points total
Transaction history	Loyalty history table	JSON array rendered as table rows	Paginated list of point transactions
Error messages	Form validation areas	HTML text	User-friendly error descriptions
Success confirmations	Toast/modal notifications	HTML text	Action completion feedback

2.1.4 KPI Metrics

The following Key Performance Indicators are tracked and displayed on the Analytics Dashboard:

KPI	Calculation Method	Target	Data Source
Total Revenue	SUM(order_total) for date range	Growth of 10% month-on-month	orders table
Total Orders	COUNT(*) from orders for date range	Minimum 50 orders per day	orders table
Active Customers	COUNT(DISTINCT user_id) from orders	20% of registered users active monthly	orders + users tables
Average Order Value	SUM(order_total) / COUNT(*)	Maintain above £30	orders table
Customer Retention Rate	(Returning customers / Total) * 100	Above 40%	orders table
Loyalty Programme Uptake	(Loyalty members / Total users) * 100	Above 60% of users enrolled	users + loyalty tables
Product Search Success	Searches resulting in product view / Total searches	Above 70%	analytics tracking
Page Load Time	Performance API measurement	Under 3 seconds	Client-side monitoring

2.2 Login Process

2.2.1 Login Process Diagram

The login process follows a sequential flow through client-side validation, Supabase authentication, session management, and role-based redirection. The diagram below describes the process in structured text format (a visual flowchart should be created using draw.io or Lucidchart for the final submission):

START → User navigates to Login page → User enters email and password → Client-side validation runs → [VALID?] → No: Display error messages, return to form → Yes: Call supabase.auth.signInWithPassword() → [AUTH SUCCESS?] → No: Display 'Invalid credentials' error → Yes: Receive JWT session token → Store session in browser → Query user role from profiles table → [ROLE?] → Customer: Redirect to Home page → Retailer: Redirect to Dashboard → END

2.2.2 Login Process Explanation

The login process implements a multi-layered security approach. First, client-side JavaScript validates the form inputs before any network request is made, checking that the email matches a valid format (using a regex pattern) and that the password meets minimum length requirements. This prevents unnecessary API calls for obviously invalid inputs.

If client-side validation passes, the application calls supabase.auth.signInWithPassword({ email, password }). Supabase handles the actual authentication, comparing the provided credentials against its internal auth.users table. Password comparison uses bcrypt hashing, meaning plain-text passwords are never stored or transmitted after initial registration.

On successful authentication, Supabase returns a session object containing a JWT (JSON Web Token) access token and a refresh token. The access token is stored in the browser and automatically included in subsequent Supabase API calls for authorisation. The refresh token allows the session to be extended without requiring the user to re-enter credentials.

After authentication, the application queries the profiles table (linked by user ID) to determine the user's role (customer or retailer). Based on this role, the application redirects to the appropriate landing page: customers go to the home page, while retailers are directed to the analytics dashboard. This role-based routing is also enforced on subsequent page loads using route guards in JavaScript.

2.2.3 Login Algorithm Flowchart

The following pseudocode represents the login algorithm that would be depicted in a visual flowchart:

FLOWCHART STRUCTURE:

[Terminal] START

[Process] Display login form

[Input] User enters email and password

[Process] Run clientValidation(email, password)

[Decision] Is input valid?

  [No] → [Process] Show validation errors → Return to input

  [Yes] → [Process] Show loading spinner

[Process] Call supabase.auth.signInWithPassword(email, password)

[Decision] Authentication successful?

  [No] → [Process] Display error: Invalid email or password → Return to input

  [Yes] → [Process] Store session token

[Process] Query profiles table for user role

[Decision] User role = retailer?

  [Yes] → [Process] Redirect to /dashboard

  [No] → [Process] Redirect to /home

[Terminal] END

2.2.4 Login Algorithm Pseudocode

```
FUNCTION handleLogin(email, password):
    errors = validateLoginForm(email, password)
    IF errors IS NOT EMPTY THEN
        displayValidationErrors(errors)
        RETURN
    END IF

    showLoadingSpinner()

    TRY
        response = AWAIT supabase.auth.signInWithPassword({
            email: email,
            password: password
        })

        IF response.error THEN
            hideLoadingSpinner()
            displayError('Invalid email or password. Please try again.')
            incrementFailedAttempts()
            IF failedAttempts >= 5 THEN
                lockAccount(email)
                displayError('Account locked. Please try again in 15 minutes.')
            END IF
            RETURN
        END IF

        session = response.data.session
        user = response.data.user

        profileData = AWAIT supabase
            .from('profiles')
            .select('role, display_name')
            .eq('id', user.id)
            .single()

        IF profileData.data.role == 'retailer' THEN
            REDIRECT TO '/dashboard'
        ELSE
            REDIRECT TO '/home'
        END IF

    CATCH error
        hideLoadingSpinner()
        displayError('A network error occurred. Please check your connection.')
    END TRY
END FUNCTION

FUNCTION validateLoginForm(email, password):
    errors = []
    IF email IS EMPTY THEN
        errors.push('Email is required')
    ELSE IF NOT matchesPattern(email, EMAIL_REGEX) THEN
        errors.push('Please enter a valid email address')
    END IF
    IF password IS EMPTY THEN
        errors.push('Password is required')
    ELSE IF length(password) < 8 THEN
```

```
        errors.push('Password must be at least 8 characters')
    END IF
    RETURN errors
END FUNCTION
```

2.3 Sub-Process 1 — Product Search and Filtering

2.3.1 Sub-Process 1 Diagram

PROCESS FLOW: User enters search query and/or selects filters → JavaScript builds Supabase query with all active parameters → Query sent to Supabase → Results returned as JSON array → JavaScript renders product cards in grid → Update result count and pagination → User can modify search/filters (loop back to start)

2.3.2 Sub-Process 1 Explanation

The product search and filtering process is the most complex client-side algorithm in the application. It must handle multiple simultaneous filter parameters, construct efficient database queries, and render results performantly.

When a user types in the search bar, a debounce function delays the query by 300 milliseconds after the user stops typing. This prevents excessive API calls during rapid typing. The debounced function then constructs a Supabase query using the .ilike() method for case-insensitive partial matching against the product name and description columns.

Filter parameters are accumulated in a JavaScript object that tracks all active selections: selected categories (array of strings), price minimum and maximum (numbers), minimum rating (number), and availability status (boolean). When any filter changes, the buildQuery() function constructs a new Supabase query that chains all active filters:

```
FUNCTION buildProductQuery(searchTerm, filters, sortBy, page):
    query = supabase.from('products').select('*', { count: 'exact' })
    IF searchTerm IS NOT EMPTY THEN
        query = query.or('name.ilike.%' + searchTerm + '%,description.ilike.%' + searchTerm + '%')
    END IF
    IF filters.categories IS NOT EMPTY THEN
        query = query.in('category', filters.categories)
    END IF
    IF filters.minPrice IS SET THEN
        query = query.gte('price', filters.minPrice)
    END IF
    IF filters.maxPrice IS SET THEN
        query = query.lte('price', filters.maxPrice)
    END IF
    IF filters.minRating IS SET THEN
        query = query.gte('rating', filters.minRating)
    END IF
    IF filters.inStock IS TRUE THEN
        query = query.gt('stock_quantity', 0)
    END IF
    query = query.order(sortBy.column, { ascending: sortBy.ascending })
    query = query.range((page - 1) * 9, page * 9 - 1)
    RETURN AWAIT query
END FUNCTION
```

2.4 Sub-Process 2 — Dashboard Data Aggregation

2.4.1 Sub-Process 2 Flowchart and Explanation

FLOWCHART:

[Start] → Admin loads dashboard page

[Process] Verify user role = retailer via profiles table

[Decision] Is user authorised?

  [No] → Redirect to home page

  [Yes] → [Process] Read selected date range

[Process] Execute parallel Supabase queries:

  Query 1: SELECT SUM(order_total), COUNT(*) FROM orders WHERE date BETWEEN start AND end

  Query 2: SELECT COUNT(DISTINCT user_id) FROM orders WHERE date BETWEEN start AND end

  Query 3: SELECT category, SUM(quantity) FROM order_items GROUP BY category

  Query 4: SELECT product_name, SUM(quantity) as units FROM order_items GROUP BY product_name ORDER BY units DESC LIMIT 5

  Query 5: SELECT * FROM products WHERE stock_quantity < 25 ORDER BY stock_quantity ASC

[Process] JavaScript processes returned data:

  Calculate average order value (total revenue / total orders)

  Calculate percentage changes vs previous period

  Format currency values

  Prepare chart data arrays

[Process] Render:

  Update KPI card values and change indicators

  Draw revenue line chart via Chart.js

  Draw category doughnut chart

  Populate top products table

  Populate inventory alerts table

[End]

The dashboard uses Promise.all() to execute all five Supabase queries in parallel, reducing the total load time compared to sequential execution. Each query targets specific data needed by a dashboard component. The JavaScript processing layer handles all calculations that cannot be efficiently done at the database level, such as percentage change calculations and data formatting for chart libraries.

The date range comparison works by querying two periods: the selected range and the equivalent previous range (e.g., if viewing the last 7 days, the comparison is the 7 days before that). This enables the percentage change calculation for each KPI.

2.5 Sub-Process 3 — Loyalty Point Redemption

2.5.1 Sub-Process 3 Pseudocode and Explanation

```
FUNCTION redeemReward(userId, rewardId):
    // Step 1: Fetch reward details
    reward = AWAIT supabase.from('rewards').select('*').eq('id', rewardId).single()
    IF reward.error OR reward.data IS NULL THEN
        displayError('Reward not found.')
        RETURN false
    END IF

    // Step 2: Fetch user's current points balance
    userProfile = AWAIT supabase.from('profiles').select('loyalty_points').eq('id', userId).single()
    currentPoints = userProfile.data.loyalty_points

    // Step 3: Validate sufficient balance
    IF currentPoints < reward.data.point_cost THEN
        displayError('Insufficient points. You need ' + (reward.data.point_cost - currentPoints) + ' more points.')
        RETURN false
    END IF

    // Step 4: Deduct points from user profile
    newBalance = currentPoints - reward.data.point_cost
    updateResult = AWAIT supabase.from('profiles')
        .update({ loyalty_points: newBalance })
        .eq('id', userId)

    IF updateResult.error THEN
        displayError('Failed to process redemption. Please try again.')
        RETURN false
    END IF

    // Step 5: Create transaction record
    AWAIT supabase.from('loyalty_transactions').insert({
        user_id: userId,
        type: 'redeem',
        points: -reward.data.point_cost,
        description: 'Redeemed: ' + reward.data.name,
        reward_id: rewardId,
        balance_after: newBalance,
        created_at: new Date().toISOString()
    })

    // Step 6: Create user voucher/reward record
    AWAIT supabase.from('user_rewards').insert({
        user_id: userId,
        reward_id: rewardId,
        status: 'active',
        expires_at: calculateExpiry(reward.data.validity_days),
        redeemed_at: new Date().toISOString()
    })

    // Step 7: Update UI
    updatePointsDisplay(newBalance)
    updateProgressBar(newBalance)
    displaySuccess('Reward redeemed successfully! Check your email for details.')
    refreshTransactionHistory()
    RETURN true
END FUNCTION
```

The loyalty point redemption process is a critical transactional operation that must maintain data integrity. The algorithm follows a defensive programming approach, validating at each step before proceeding to the next. The points deduction and transaction record creation should ideally be atomic (wrapped in a database transaction), but since Supabase client-side operations do not directly support multi-statement transactions, the application performs these sequentially with error checking at each step. If the points deduction succeeds but the transaction record fails, a reconciliation process can detect and correct the discrepancy.

The process also updates the user interface in real-time after a successful redemption, refreshing the points balance display, progress bar, and transaction history without requiring a full page reload. This provides immediate feedback and confirms the action was successful.

3. Data Requirements

3.1 Supabase Table Design

The database schema consists of seven primary tables designed to support all application functionality. Each table is described below with its columns, data types, constraints, and relationships.

3.1.1 profiles Table

Column Name	Data Type	Constraints	Description
id	UUID	PRIMARY KEY, REFERENCES auth.users(id)	Links to Supabase auth user
display_name	VARCHAR(100)	NOT NULL	User's display name
email	VARCHAR(255)	NOT NULL, UNIQUE	User's email address
role	VARCHAR(20)	NOT NULL, DEFAULT 'customer'	User role: 'customer' or 'retailer'
loyalty_points	INTEGER	NOT NULL, DEFAULT 0	Current loyalty points balance
loyalty_tier	VARCHAR(20)	DEFAULT 'bronze'	Current tier: bronze/silver/gold/platinum
referral_code	VARCHAR(20)	UNIQUE	Unique referral code for loyalty programme
avatar_url	TEXT	NULLABLE	URL to profile avatar image
created_at	TIMESTAMPTZ	DEFAULT NOW()	Account creation timestamp
updated_at	TIMESTAMPTZ	DEFAULT NOW()	Last profile update timestamp

3.1.2 products Table

Column Name	Data Type	Constraints	Description
id	SERIAL	PRIMARY KEY	Auto-incrementing product ID
name	VARCHAR(200)	NOT NULL	Product name
description	TEXT	NOT NULL	Full product description
price	DECIMAL(10,2)	NOT NULL, CHECK(price > 0)	Product price in GBP

original_price	DECIMAL(10,2)	NULLABLE	Original price before discount
category	VARCHAR(50)	NOT NULL	Product category
subcategory	VARCHAR(50)	NULLABLE	Product subcategory
image_url	TEXT	NULLABLE	URL to product image
stock_quantity	INTEGER	NOT NULL, DEFAULT 0	Current stock level
rating	DECIMAL(2,1)	DEFAULT 0, CHECK(rating BETWEEN 0 AND 5)	Average customer rating
review_count	INTEGER	DEFAULT 0	Number of reviews
is_featured	BOOLEAN	DEFAULT FALSE	Whether product appears in trending
created_at	TIMESTAMPTZ	DEFAULT NOW()	When product was added
updated_at	TIMESTAMPTZ	DEFAULT NOW()	Last product update

3.1.3 orders Table

Column Name	Data Type	Constraints	Description
id	SERIAL	PRIMARY KEY	Auto-incrementing order ID
user_id	UUID	NOT NULL, REFERENCES profiles(id)	Customer who placed the order
order_total	DECIMAL(10,2)	NOT NULL	Total order amount in GBP
status	VARCHAR(20)	DEFAULT 'pending'	Order status: pending/confirmed/shipped/delivered
points_earned	INTEGER	DEFAULT 0	Loyalty points earned from order
created_at	TIMESTAMPTZ	DEFAULT NOW()	Order timestamp

3.1.4 order_items Table

Column Name	Data Type	Constraints	Description
id	SERIAL	PRIMARY KEY	Auto-incrementing item ID

order_id	INTEGER	NOT NULL, REFERENCES orders(id)	Parent order
product_id	INTEGER	NOT NULL, REFERENCES products(id)	Product purchased
quantity	INTEGER	NOT NULL, CHECK(quantity > 0)	Quantity purchased
unit_price	DECIMAL(10,2)	NOT NULL	Price per unit at time of purchase

3.1.5 rewards Table

Column Name	Data Type	Constraints	Description
id	SERIAL	PRIMARY KEY	Auto-incrementing reward ID
name	VARCHAR(100)	NOT NULL	Reward name
description	TEXT	NOT NULL	Reward description
point_cost	INTEGER	NOT NULL, CHECK(point_cost > 0)	Points required to redeem
image_url	TEXT	NULLABLE	Reward image URL
validity_days	INTEGER	DEFAULT 30	Days reward remains valid after redemption
is_active	BOOLEAN	DEFAULT TRUE	Whether reward is currently available
created_at	TIMESTAMPTZ	DEFAULT NOW()	When reward was created

3.1.6 loyalty_transactions Table

Column Name	Data Type	Constraints	Description
id	SERIAL	PRIMARY KEY	Auto-incrementing transaction ID
user_id	UUID	NOT NULL, REFERENCES profiles(id)	User involved
type	VARCHAR(20)	NOT NULL	Transaction type: purchase/redeem/bonus/referral
points	INTEGER	NOT NULL	Points amount

			(positive=earned, negative=spent)
description	VARCHAR(255)	NOT NULL	Human-readable description
reward_id	INTEGER	NULLABLE, REFERENCES rewards(id)	Related reward if redemption
order_id	INTEGER	NULLABLE, REFERENCES orders(id)	Related order if purchase
balance_after	INTEGER	NOT NULL	Running balance after this transaction
created_at	TIMESTAMPTZ	DEFAULT NOW()	Transaction timestamp

3.1.7 user_rewards Table

Column Name	Data Type	Constraints	Description
id	SERIAL	PRIMARY KEY	Auto-incrementing ID
user_id	UUID	NOT NULL, REFERENCES profiles(id)	User who redeemed
reward_id	INTEGER	NOT NULL, REFERENCES rewards(id)	Redeemed reward
status	VARCHAR(20)	DEFAULT 'active'	Status: active/used/expired
redeemed_at	TIMESTAMPTZ	DEFAULT NOW()	Redemption timestamp
expires_at	TIMESTAMPTZ	NOT NULL	Expiry date
used_at	TIMESTAMPTZ	NULLABLE	When reward was used

3.2 SQL Code (CREATE Table / INSERT Examples)

Below are the SQL statements for creating the core tables and inserting sample data:

```
-- CREATE TABLE: profiles
CREATE TABLE profiles (
   id UUID PRIMARY KEY REFERENCES auth.users(id) ON DELETE CASCADE,
   display_name VARCHAR(100) NOT NULL,
   email VARCHAR(255) NOT NULL UNIQUE,
   role VARCHAR(20) NOT NULL DEFAULT 'customer' CHECK (role IN ('customer', 'retailer')),
   loyalty_points INTEGER NOT NULL DEFAULT 0 CHECK (loyalty_points >= 0),
   loyalty_tier VARCHAR(20) DEFAULT 'bronze' CHECK (loyalty_tier IN ('bronze', 'silver', 'gold',
'platinum')),
   referral_code VARCHAR(20) UNIQUE,
   avatar_url TEXT,
   created_at TIMESTAMPTZ DEFAULT NOW(),
   updated_at TIMESTAMPTZ DEFAULT NOW()
);

-- CREATE TABLE: products
CREATE TABLE products (
   id SERIAL PRIMARY KEY,
   name VARCHAR(200) NOT NULL,
   description TEXT NOT NULL,
   price DECIMAL(10,2) NOT NULL CHECK (price > 0),
   original_price DECIMAL(10,2),
   category VARCHAR(50) NOT NULL,
   subcategory VARCHAR(50),
   image_url TEXT,
   stock_quantity INTEGER NOT NULL DEFAULT 0 CHECK (stock_quantity >= 0),
   rating DECIMAL(2,1) DEFAULT 0 CHECK (rating BETWEEN 0 AND 5),
   review_count INTEGER DEFAULT 0,
   is_featured BOOLEAN DEFAULT FALSE,
   created_at TIMESTAMPTZ DEFAULT NOW(),
   updated_at TIMESTAMPTZ DEFAULT NOW()
);
```

```
-- INSERT sample products

INSERT INTO products (name, description, price, original_price, category, subcategory,
stock_quantity, rating, review_count, is_featured) VALUES

('Wireless Bluetooth Headphones', 'Premium noise-cancelling headphones with 30-hour battery
life', 49.99, 69.99, 'Electronics', 'Audio', 150, 4.3, 128, TRUE),

('Smart Fitness Watch', 'Advanced health tracking with heart rate monitor and GPS', 129.99,
NULL, 'Electronics', 'Wearables', 85, 4.8, 256, TRUE),

('Running Shoes Pro', 'Lightweight performance running shoes with cushioned sole', 89.99,
NULL, 'Sports', 'Footwear', 200, 4.1, 89, FALSE),

('Organic Cotton T-Shirt', 'Sustainable 100% organic cotton, available in multiple colours', 24.99,
NULL, 'Clothing', 'Tops', 500, 4.5, 312, FALSE),

('Smart Home Speaker', 'Voice-controlled speaker with built-in virtual assistant', 34.99, 54.99,
'Electronics', 'Smart Home', 75, 3.6, 42, TRUE);
```

3.3 Data Dictionary

The data dictionary provides a comprehensive reference for all database fields, their formats, and validation rules:

Table	Field	Type	Size	Required	Validation Rule
profiles	id	UUID	36 chars	Yes	Auto-generated by Supabase auth
profiles	display_name	VARCHAR	100 chars	Yes	Min 2 chars, letters and spaces only
profiles	email	VARCHAR	255 chars	Yes	Valid email format (RFC 5322)
profiles	role	VARCHAR	20 chars	Yes	Must be 'customer' or 'retailer'
profiles	loyalty_points	INTEGER	4 bytes	Yes	Must be >= 0
profiles	loyalty_tier	VARCHAR	20 chars	No	Must be bronze/silver/gold/platinum
products	name	VARCHAR	200 chars	Yes	Min 3 chars, max 200 chars
products	price	DECIMAL	10,2	Yes	Must be > 0, max 99999999.99
products	category	VARCHAR	50 chars	Yes	Must match predefined categories
products	rating	DECIMAL	2,1	No	Must be between 0.0 and 5.0
products	stock_quantity	INTEGER	4 bytes	Yes	Must be >= 0
orders	order_total	DECIMAL	10,2	Yes	Must be > 0
orders	status	VARCHAR	20 chars	No	Must be pending/confirmed/shipped/delivered
loyalty_transactions	type	VARCHAR	20 chars	Yes	Must be purchase/redeem/bonus/referral

loyalty_transactions	points	INTEGER	4 bytes	Yes	Non-zero integer
rewards	point_cost	INTEGER	4 bytes	Yes	Must be > 0

3.4 Data Flow Diagram

The data flow diagram describes how data moves through the system from external entities to data stores via processes. The following describes a Level 1 DFD in text format (a visual diagram should be created using draw.io or Lucidchart for submission):

External Entities:

E1: Customer (unauthenticated visitor or logged-in customer)

E2: Retailer (authenticated admin user)

Processes:

P1: Authentication System — Handles login, registration, session management

P2: Product Search Engine — Processes search queries and filter parameters

P3: Order Processing — Handles order creation and loyalty point calculation

P4: Analytics Aggregation — Computes dashboard metrics from raw data

P5: Loyalty Management — Manages point earning, redemption, and tier upgrades

Data Stores:

D1: profiles — User account data

D2: products — Product catalogue

D3: orders / order_items — Transaction records

D4: loyalty_transactions — Points activity log

D5: rewards / user_rewards — Reward catalogue and user redemptions

Data Flows:

E1 → P1: Login credentials (email, password)

P1 → D1: Store/retrieve user profile data

P1 → E1: Authentication token, session data

E1 → P2: Search query, filter parameters

P2 → D2: Query product data

D2 → P2: Product results

P2 → E1: Filtered product list

E1 → P3: Order details

P3 → D3: Store order and order items

P3 → P5: Trigger point earning

P5 → D1: Update loyalty points balance

P5 → D4: Create transaction record

E1 → P5: Redemption request

P5 → D5: Create user reward record

E2 → P4: Date range, dashboard view request

P4 → D3: Query order data

P4 → D2: Query product data

P4 → D4: Query loyalty data

P4 → E2: Aggregated metrics, chart data

3.5 Static/Dynamic Model Diagram

The static model describes the class structure and relationships. The dynamic model describes object interactions over time.

3.5.1 Static Model (Class Diagram Description)

The system has six main classes with the following relationships:

User (abstract): id, email, displayName, role. Methods: login(), logout(), updateProfile()

Customer (extends User): loyaltyPoints, loyaltyTier, referralCode. Methods: earnPoints(), redeemReward(), viewHistory()

Retailer (extends User): storeName. Methods: viewDashboard(), manageProducts()

Product: id, name, description, price, category, stockQuantity, rating. Methods: search(), filter(), updateStock()

Order: id, userId, orderTotal, status. Methods: create(), calculatePoints(), updateStatus()

OrderItem: id, orderId, productId, quantity, unitPrice

Reward: id, name, pointCost, validityDays. Methods: redeem(), checkAvailability()

LoyaltyTransaction: id, userId, type, points, balanceAfter. Methods: create(), getHistory()

3.5.2 Dynamic Model (Sequence Diagram for Product Purchase)

The following sequence describes the interaction flow when a customer makes a purchase:

1. Customer → ProductPage: selectProduct(productId)
2. ProductPage → Supabase: query products WHERE id = productId
3. Supabase → ProductPage: return product details
4. ProductPage → Customer: display product with Add to Cart button
5. Customer → OrderSystem: addToCart(productId, quantity)
6. Customer → OrderSystem: checkout()
7. OrderSystem → Supabase: INSERT INTO orders (user_id, order_total, status)
8. OrderSystem → Supabase: INSERT INTO order_items (order_id, product_id, quantity, unit_price)
9. OrderSystem → LoyaltySystem: calculatePoints(orderTotal)
10. LoyaltySystem → Supabase: UPDATE profiles SET loyalty_points = loyalty_points + earned
11. LoyaltySystem → Supabase: INSERT INTO loyalty_transactions
12. LoyaltySystem → LoyaltySystem: checkTierUpgrade(newBalance)
13. OrderSystem → Customer: display order confirmation

3.6 Entity Relationship Diagram (ERD) and Normalisation

The Entity Relationship Diagram represents the database schema relationships. The following describes the ERD in text format (a visual ERD should be created using dbdiagram.io or draw.io for submission):

Entities and Relationships:

profiles (1) ——< (M) orders: One user can have many orders

orders (1) ——< (M) order_items: One order can have many items

products (1) ——< (M) order_items: One product can appear in many order items

profiles (1) ——< (M) loyalty_transactions: One user can have many transactions

rewards (1) ——< (M) loyalty_transactions: One reward can be referenced in many transactions

profiles (1) ——< (M) user_rewards: One user can redeem many rewards

rewards (1) ——< (M) user_rewards: One reward can be redeemed by many users

orders (1) ——< (M) loyalty_transactions: One order can generate one loyalty transaction

Normalisation:

First Normal Form (1NF): All tables contain only atomic (indivisible) values. There are no repeating groups or arrays stored in any column. Each row is uniquely identifiable by its primary key.

Second Normal Form (2NF): All non-key attributes are fully functionally dependent on the entire primary key. For example, in the order_items table, the unit_price is stored per line item rather than being derived from the products table, because prices may change over time and we need to preserve the price at the time of purchase.

Third Normal Form (3NF): No transitive dependencies exist. For instance, the loyalty_tier in profiles could theoretically be derived from loyalty_points, but storing it explicitly avoids complex recalculation on every query and allows for manual tier adjustments (e.g., promotional upgrades). The balance_after field in loyalty_transactions could also be calculated from transaction history but is stored for performance optimisation in displaying the transaction log.

3.7 Error Handling Plan

The error handling plan addresses both SQL-level errors (database constraints, query failures) and JavaScript-level errors (network issues, runtime exceptions, validation failures).

3.7.1 SQL Error Handling

Error Type	Cause	Handling Strategy	User Message
Unique constraint violation	Duplicate email on registration	Check error code 23505	This email is already registered. Please login instead.
Foreign key violation	Reference to non-existent record	Check error code 23503	The requested item could not be found.
Check constraint violation	Invalid data (e.g., negative price)	Check error code 23514	The submitted data is invalid. Please check your input.
Not null violation	Missing required field	Check error code 23502	Please fill in all required fields.
Connection timeout	Network or server issue	Retry with exponential backoff (3 attempts)	Unable to connect. Please check your internet connection.
Row Level Security denial	Unauthorised data access	Supabase returns empty result	You do not have permission to access this resource.

3.7.2 JavaScript Error Handling

Error Type	Cause	Handling Strategy	User Message
TypeError	Accessing property of null/undefined	Try-catch blocks, optional chaining (?.)	Something went wrong. Please refresh the page.
Network error	Fetch API failure	Try-catch with retry logic	Connection lost. Please check your internet.
Validation error	Invalid form input	Prevent form submission, highlight field	Specific field-level error message
Authentication error	Expired session token	Redirect to login page	Your session has expired. Please log in again.
JSON parse error	Unexpected API response	Try-catch around JSON.parse()	An unexpected error occurred. Please try again.
Rate limit error	Too many API requests	Implement request throttling	Please wait a moment before trying again.

All errors are logged to the browser console with full stack traces for debugging purposes. In production, a structured error logging service would capture these for monitoring. User-facing error messages are always friendly and actionable, never exposing technical details or database structure.

3.8 Variable Lists and Descriptions

The following tables document the key JavaScript variables used across the application, organised by functional area:

3.8.1 Authentication Variables

Variable Name	Type	Scope	Description
supabase	Object	Global	Supabase client instance initialised with project URL and anon key
currentUser	Object|null	Global	Currently authenticated user object from Supabase auth
userProfile	Object|null	Global	User profile data from profiles table
isAuthenticated	Boolean	Global	Whether a valid session exists
loginEmail	String	Local (login.js)	Email input value from login form
loginPassword	String	Local (login.js)	Password input value from login form
failedAttempts	Integer	Local (login.js)	Counter for failed login attempts
sessionToken	String	Session	JWT access token for API authorisation

3.8.2 Product Search Variables

Variable Name	Type	Scope	Description
searchQuery	String	Local (search.js)	Current search bar input text
activeFilters	Object	Local (search.js)	Object containing all active filter parameters
activeFilters.categories	Array	Local	Selected category filter values
activeFilters.minPrice	Number	Local	Minimum price filter value
activeFilters.maxPrice	Number	Local	Maximum price filter value
activeFilters.minRating	Number	Local	Minimum rating filter value

activeFilters.inStock	Boolean	Local	Whether to filter by in-stock only
currentSort	Object	Local (search.js)	Current sort column and direction
currentPage	Integer	Local (search.js)	Current pagination page number
totalResults	Integer	Local (search.js)	Total number of matching products
productsPerPage	Constant (9)	Local (search.js)	Number of products displayed per page
debounceTimer	Timer ID	Local (search.js)	setTimeout ID for search debouncing

3.8.3 Dashboard Variables

Variable Name	Type	Scope	Description
dateRange	Object	Local (dashboard.js)	Selected date range {start, end}
kpiData	Object	Local (dashboard.js)	Aggregated KPI values for current period
previousKpiData	Object	Local (dashboard.js)	KPI values for comparison period
revenueChartData	Array	Local (dashboard.js)	Array of {date, revenue} for line chart
categoryChartData	Array	Local (dashboard.js)	Array of {category, total} for doughnut chart
topProducts	Array	Local (dashboard.js)	Array of top-selling product objects
inventoryAlerts	Array	Local (dashboard.js)	Array of low-stock product objects
revenueChart	Chart instance	Local (dashboard.js)	Chart.js instance for revenue chart
categoryChart	Chart instance	Local (dashboard.js)	Chart.js instance for category chart

3.8.4 Loyalty Variables

Variable Name	Type	Scope	Description
userPoints	Integer	Local (loyalty.js)	User's current loyalty

			points balance
userTier	String	Local (loyalty.js)	User's current loyalty tier
availableRewards	Array	Local (loyalty.js)	Array of reward objects from Supabase
transactionHistory	Array	Local (loyalty.js)	Array of loyalty transaction records
historyPage	Integer	Local (loyalty.js)	Current page of transaction history
referralCode	String	Local (loyalty.js)	User's unique referral code
activeTab	String	Local (loyalty.js)	Currently selected tab on loyalty page
tierThresholds	Object	Constant	{bronze: 0, silver: 2000, gold: 5000, platinum: 10000}

4. Test Strategy

4.1 Overview

The test strategy for Retail Innovations follows a systematic, multi-layered approach to quality assurance. Testing begins with individual units (functions and components), progresses to integration testing (ensuring components work together), and culminates in full system and acceptance testing. The strategy is designed to catch defects as early as possible in the development cycle, where they are cheapest and easiest to fix.

All tests will be documented in a testing log with the following information: test ID, test description, input data, expected output, actual output, pass/fail status, and remedial action taken for failures. Screenshots will be captured as evidence for each test.

4.2 Interrelation of Components

The Retail Innovations platform has several tightly interrelated components that must be tested both individually and in combination:

The authentication system (login/register) underpins all other functionality. The product search system depends on correctly populated product data in Supabase. The analytics dashboard aggregates data from orders, products, and loyalty transactions. The loyalty programme depends on the order system (for earning points) and the rewards system (for spending points). The user profiles table links authentication to role-based access control.

Because of these interdependencies, integration testing is critical. A failure in the authentication system would cascade into failures across all protected pages. A data integrity issue in the orders table would produce incorrect analytics and loyalty point calculations.

4.3 Data Flow Impacts

Data flows through the system in predictable paths, and errors at any point can propagate downstream. For example, if a product price is incorrectly stored (data integrity issue), order totals would be calculated incorrectly, which would in turn cause incorrect loyalty point awards, which would further cause inaccurate analytics on the dashboard. Testing must verify data integrity at each transformation point.

Key data flow paths to test include: registration data flowing from the form to auth.users and then to profiles; product search queries flowing from the UI to Supabase and results flowing back; order creation triggering point calculation and transaction recording; and dashboard queries aggregating data correctly across multiple tables.

4.4 Transition Between Unit and Integration Tests

Unit tests focus on individual JavaScript functions in isolation: does the email validation regex correctly identify valid and invalid emails? Does the price formatting function correctly convert numbers to currency strings? Does the points calculation function return the correct value?

Once individual functions pass unit tests, integration tests verify that these functions work correctly together: does the login form correctly call the validation function, then the Supabase auth function, then redirect based on role? Does the product search correctly combine search text with filter parameters and sort options into a single working query?

The transition occurs when all unit tests pass. Integration tests then use the verified functions in realistic user workflows to identify interface and communication issues between components.

4.5 Functional Tests

Functional tests verify that each feature works as specified in the requirements:

Test Area	Test Description	Expected Outcome
Login	Valid credentials submitted	User authenticated, redirected to correct page based on role
Login	Invalid credentials submitted	Error message displayed, user remains on login page
Registration	Valid registration data submitted	Account created, user redirected, welcome email implied
Product Search	Search term entered	Matching products displayed with correct count
Product Filter	Category filter applied	Only products in selected category shown
Product Sort	Sort by price ascending selected	Products ordered from lowest to highest price
Dashboard KPIs	Date range changed	All KPI values update to reflect new date range
Loyalty Redeem	User redeems reward with sufficient points	Points deducted, transaction recorded, reward issued
Loyalty Redeem	User attempts redemption with insufficient points	Error message shown, no points deducted

4.6 Non-Functional Tests

Category	Test Description	Acceptance Criteria
Performance	Page load time measurement	All pages load within 3 seconds on standard connection
Performance	Search response time	Search results return within 2 seconds
Responsiveness	Mobile device rendering	All pages usable on 375px width viewport
Responsiveness	Tablet rendering	All pages correctly display at 768px width
Accessibility	Screen reader compatibility	All interactive elements have ARIA labels
Accessibility	Keyboard navigation	All functions accessible via keyboard only
Accessibility	Colour contrast	All text meets WCAG 2.1 AA

		contrast requirements
Security	SQL injection attempt	Malicious input sanitised, no data breach
Security	XSS attack attempt	Script tags escaped, not executed
Compatibility	Cross-browser testing	Functions correctly in Chrome, Firefox, Safari, Edge

4.7 Front End Tests

Front-end tests focus on the visual and interactive elements of the user interface:

CSS rendering tests verify that layouts display correctly at all breakpoints (desktop 1200px, tablet 768px, mobile 480px). Form validation tests confirm that error messages appear in the correct position with the correct styling when invalid data is entered. Navigation tests verify that all links point to the correct pages and that the active page is correctly highlighted. Interactive elements (dropdowns, modals, tabs) are tested for correct open/close behaviour and keyboard accessibility.

4.8 Order of Tests

Tests will be executed in the following order, progressing from most fundamental to most complex:

1. Unit tests: Individual JavaScript functions (validation, formatting, calculations)
2. Database tests: Supabase table creation, constraints, RLS policies
3. Authentication tests: Login, registration, session management
4. CRUD operation tests: Create, Read, Update, Delete for each entity
5. Integration tests: Multi-component workflows (search + filter + sort + paginate)
6. Front-end tests: Responsive design, navigation, visual rendering
7. Non-functional tests: Performance, accessibility, security
8. System tests: Full end-to-end user journeys
9. Acceptance tests: Testing against original client requirements

4.9 Acceptance Testing

Acceptance testing verifies the solution against the original client requirements specified in the brief. Each requirement from the Retail Innovations brief will be mapped to specific test cases. The client (or a representative) will execute these tests and sign off on each requirement. Acceptance criteria include: all three functional features are fully operational, the user interface is intuitive and accessible, data is stored and retrieved correctly, and the system handles errors gracefully.

4.10 Alpha Testing

Alpha testing is conducted by the developer in a controlled environment before any external users interact with the system. During alpha testing, all features are tested systematically using the test cases defined above. Bugs discovered during alpha testing are logged, fixed, and re-tested. The alpha phase continues until all critical and high-priority bugs are resolved and all functional tests pass.

4.11 Beta Testing

Beta testing involves a small group of external users (3-5 people representing both customer and retailer personas) testing the application in a realistic environment. Beta testers receive specific tasks to complete (e.g., register an account, search for a product, redeem loyalty points) and provide feedback through a structured survey. Beta testing identifies usability issues, confusing workflows, and edge cases that developer testing may have missed. Feedback is documented and prioritised for implementation.

4.12 Black Box Testing

Black box testing treats the application as an opaque system, testing inputs and outputs without knowledge of internal implementation. Testers interact with the user interface exactly as a real user would, entering data into forms and verifying the displayed results. This approach is used for all functional tests and acceptance tests. Test data includes normal data (typical valid inputs), boundary data (values at the edge of valid ranges), erroneous data (deliberately invalid inputs), and extreme data (very large or very small values).

4.13 White Box/Structural Testing

White box testing examines the internal code structure to verify that all code paths are executed correctly. This includes testing every branch of conditional statements (if/else), verifying that loops terminate correctly, testing error handling paths (catch blocks), and ensuring that all functions return expected values for all input combinations. Code coverage analysis will be used to identify untested code paths. Particular attention is paid to the product query builder function (which has many possible filter combinations) and the loyalty point calculation logic (which must handle various edge cases).