RETAIL INNOVATIONS LTD

Digital Retail Management Platform

Task 3 — Feedback Gathering and Evaluation Report

T Level Technical Qualification in Digital Software Development

Document Contents

§	Section	Activity
B 8	Reflection on Development Process	Activity B
B 9	Overall Evaluation Summary	Activity B

TASK 3 — ACTIVITY A: Feedback Gathering

A1. Feedback Strategy

Feedback on the Retail Innovations Ltd platform was gathered from two distinct audience groups. This two-audience approach is necessary because the platform must function correctly at a technical level and also be usable and commercially valuable to non-developer business users.

Technical feedback came from two fellow learners who reviewed the codebase (app.js v1.4, index.html v1.1, styles.css v1.1, schema.sql v1.3), the database schema, and the overall system architecture. Non-technical feedback came from three participants: a simulated retail manager (admin role), a customer-role test user, and a person with no prior retail software experience.

Feedback was gathered in three stages: (1) participants watched a screencast demonstrating the platform relevant to their audience, (2) they completed a structured questionnaire, and (3) a brief unstructured follow-up discussion captured any additional points not covered by the questionnaire.

A2. Screencast Script — Technical Audience

The following script was used to record a screencast aimed at technical peer reviewers. The recording walks through the codebase structure, database schema, and key implementation decisions. Duration: approximately 12–15 minutes. Screen recording captures both the code editor and the running application simultaneously.

SCREENCAST SCRIPT — TECHNICAL AUDIENCE (CODE REVIEW)

Introduction (0:00 – 0:45)

Developer: Hi, welcome to this technical review of the Retail Innovations Ltd digital platform. I'm going to walk you through the key implementation decisions, the database schema, and how the front-end and back-end work together. After watching this, you'll be asked to complete a short questionnaire — feel free to pause or rewind at any point.

[Show: project folder structure in VS Code — index.html, app.js, styles.css, schema.sql visible]

Developer: The project consists of four files. index.html is the single-page application shell. app.js contains all the JavaScript logic — roughly 800 lines. styles.css handles all visual presentation, and schema.sql is the PostgreSQL schema for Supabase. There is no build pipeline — this was a deliberate choice to keep the project simple and deployable without Node tooling.

Database Schema (0:45 – 3:30)

[Show: schema.sql open in editor, scroll slowly through the table definitions]

Developer: Starting with the database. The schema defines seven tables. I'll highlight a few key design decisions. First, the order_items table. The subtotal column is defined as GENERATED ALWAYS AS quantity times unit_price STORED — this is a computed column. It means the subtotal is always mathematically consistent with its components and can never be manually set to an incorrect value. This removes a whole class of potential data integrity bugs.

[Show: order_items table definition, highlight the GENERATED ALWAYS AS line]

Developer: Second, notice the mix of CASCADE and SET NULL in foreign key definitions. Order items use CASCADE on the orders FK — if an order is deleted, its items go too, which makes sense because orphaned line items are meaningless without their parent order. But orders use SET NULL on the customer FK — if a customer account is deleted, we keep the order record for financial history purposes, just without a customer link.

[Show: scroll to RLS section]

Developer: Now the security layer. Row Level Security is enabled on all seven tables. I created a helper function called is_admin() that queries the user_profiles table for the current authenticated user's role. Every policy that needs to check admin status calls this function — so there's one place to update if the logic ever changes. Without this, each policy would need its own subquery, which is harder to maintain and easier to get wrong.

[Show: is_admin() function, then show one RLS policy using it]

Developer: Here's an example — the policy for inserting products. The WITH CHECK clause calls is_admin() — so even if someone manipulates the front-end JavaScript, the database itself will reject any INSERT from a non-admin user with a 403 permission denied error. The front-end JS is the first line of defence; RLS is the second.

JavaScript Architecture (3:30 – 7:00)

[Show: app.js, scroll to top — constants and Supabase initialisation]

Developer: Moving to the JavaScript. The application has two operating modes — I call this the dual-mode architecture. At startup, it checks whether the Supabase client was successfully initialised. If it was, all data operations go to the cloud database. If not, it falls back to localStorage. This means the app can be demonstrated offline without any infrastructure, but the same codebase serves both scenarios. Every async data function has an if supabase conditional that switches between the two paths.

[Show: a loadProducts() function highlighting the if(supabase) branch]

Developer: Here's loadProducts() as an example. The if supabase branch calls the Supabase client and maps the result into the productsCache array. The else branch reads from localStorage. Both paths produce the same data structure, so renderProducts() doesn't need to know which source the data came from.

[Show: esc() function]

Developer: XSS prevention — the esc() function. Every user-supplied string that gets inserted into the DOM goes through this function first. It replaces the five HTML characters that would allow script injection — ampersand, less-than, greater-than, double quote, and single quote — with their HTML entity equivalents. This is applied consistently across all renderX() functions. I tested this in tests SEC03 and SEC04 by entering a script tag and an img onerror attribute in product name and description fields — both were correctly neutralised.

[Show: calculateLoyaltyTier() function]

Developer: The loyalty tier calculation. This is a pure function — it takes a points integer and returns a tier string. The boundary values are 500 for Silver, 1000 for Gold, and 2000 for Platinum. I specifically tested the boundary values in test CUST12 — 999 returns Silver and 1000 returns Gold, which confirms the comparison operators are correct.

Role-Based Access Control (7:00 – 9:30)

[Show: enterApp() function in app.js]

Developer: Role-based access control is implemented at two levels. At the UI level, the enterApp() function sets an is-admin CSS class on the document body when the logged-in user is admin. In the stylesheet, any element with the admin-only class is hidden by default — but when the body has is-admin, those elements become visible. This means a single CSS rule controls the visibility of every admin element simultaneously.

[Show: CSS file — .admin-only and body.is-admin .admin-only rules]

Developer: At the JavaScript level, the openCustomerModal, openProductModal, and openRewardModal functions each check the isAdmin flag before doing anything. If a non-admin somehow calls these functions — for example by typing directly into the browser console — they'll get a toast notification saying admin access is required and the function returns early. And of course, even if they got past that, the Supabase RLS policies would reject any database write.

Testing and Iteration (9:30 – 11:30)

[Show: testing log document briefly, then switch back to app.js]

Developer: During testing, six failures were identified and corrected. The most instructive was the bar chart crash in DASH08. Math.max applied to an empty array returns negative infinity in JavaScript, which then caused the chart height calculation to produce NaN values and the chart to throw a TypeError. The fix was to include 1 as a fallback argument to Math.max, guaranteeing the result is always at least 1 even if the array is empty.

[Show: renderBarChart() — the Math.max line]

Developer: The other failures were mostly around validation — saveProduct, saveCustomer, and saveOrder all initially accepted empty or negative values. These were

fixed by adding early-return validation blocks before any database call is made. The principle is: validate client-side first to give the user immediate feedback, but never rely on client-side validation alone — the database CHECK constraints and RLS are there as a second layer.

Closing (11:30 – 12:00)

Developer: That covers the main technical decisions. The questionnaire that follows asks about code quality, naming conventions, the database design, security, and what you'd prioritise as improvements. I'd particularly welcome your view on the hardcoded admin credentials in the local login function — I know this is a security concern for production and I'm interested in your thoughts on the best approach to replace it. Thanks for reviewing.

[End recording. Questionnaire distributed immediately after.]

A3. Screencast Script — Non-Technical Audience

The following script was used to record a screencast aimed at non-technical business users. The recording demonstrates the platform's features through the lens of real retail tasks. Duration: approximately 8–10 minutes. Screen recording shows only the running application, not the code.

SCREENCAST SCRIPT — NON-TECHNICAL AUDIENCE (PLATFORM DEMONSTRATION)

Introduction (0:00 – 0:30)

Developer: Hi — this short video demonstrates the Retail Innovations digital platform. I'll show you how to sign in, find and manage products, view customer information, track orders, and access the loyalty programme. After watching, you'll be asked to answer a few questions about what you saw. There's no right or wrong — your honest reaction is what I'm after.

Signing In (0:30 – 1:30)

[Show: login screen]

Developer: This is the sign-in page. You'd type your email address and password here and click Sign In. If you're new, clicking 'Create one' here reveals a registration form — it asks for your email, a password, and your name. Once you're registered, you don't need to register again. The platform remembers you even if you close the browser tab.

[Log in as admin user — show app load with all tabs visible]

Developer: I've signed in as an administrator. You can see my name in the top right corner. As an admin, I can see all five sections across the top — Dashboard, Products, Customers, Orders, and Loyalty. Regular customers would see fewer options, which I'll show you later.

Dashboard (1:30 – 3:00)

[Show: Dashboard with stat cards and charts]

Developer: This is the Dashboard — the first thing an admin sees after signing in. The four boxes at the top give a quick overview: how many products are in the catalogue, how many registered customers there are, the total number of orders, and the total revenue. These update automatically whenever new data is added.

[Point to bar chart]

Developer: Below that, there are two charts. This one on the left shows our product catalogue broken down by category — the taller the bar, the more products in that category. Sports is our biggest category at the moment. On the right is a donut chart showing how our customers are distributed across our four loyalty tiers — Bronze, Silver, Gold, and Platinum. Most of our customers are in the Bronze tier, with a smaller number reaching the higher tiers.

Products (3:00 – 5:00)

[Click Products tab — show full product table]

Developer: The Products section shows our full catalogue. At the moment we have ten products across eight categories. You can search for a specific product by typing here — watch what happens when I type 'tea'.

[Type 'tea' in search box — show filter to 1 result]

Developer: Instantly filters down to just Organic Green Tea. I can also filter by category using this dropdown — if I select Sports, I'll see only the running shoes and yoga mat.

[Select Sports — show 2 results. Then clear filters.]

Developer: As an admin, I can add a new product using this blue button here, edit any existing product by clicking Edit on its row, or remove a product by clicking Delete — which asks for confirmation first so nothing gets deleted by accident.

[Click Edit on a product — show pre-populated modal. Point out fields.]

Developer: The edit form opens with all the current details already filled in. I can change the price, the stock level, or any other detail. If I accidentally leave a required field empty, the platform tells me clearly what's missing rather than just failing silently.

Customers (5:00 – 6:30)

[Click Customers tab]

Developer: The Customers section is only visible to administrators — regular shoppers don't see this. At the top, there are four cards explaining the loyalty tier system. Bronze is for customers with zero to four hundred and ninety-nine points. Silver from five hundred, Gold from a thousand, and Platinum from two thousand points.

[Point to tier cards then to customer table]

Developer: Below that is the full customer list. You can see each customer's name, email, how many loyalty points they've accumulated, what tier they're in — shown with these coloured labels — and their total spend. The tier is automatically updated whenever the points change, so you never have to manually upgrade someone to Gold — it happens on its own.

Orders (6:30 – 7:30)

[Click Orders tab]

Developer: The Orders section gives you a full view of every order placed. Each one shows the customer name, the order total, the current status — shown here as a coloured label — and the date. Status can be updated as the order moves through your process: from Pending, to Processing, to Shipped, to Delivered.

[Click Edit on an order — show status dropdown]

Developer: Clicking Edit lets you update the status. This is how you'd mark an order as shipped once it's left the warehouse, for example.

Loyalty Programme (7:30 – 8:30)

[Click Loyalty tab]

Developer: Finally, the Loyalty Programme section. This is visible to both admin users and customers. It shows the rewards available — what they are, how many points are required to claim them, and whether they're currently active. Administrators can add new rewards or adjust existing ones. Customers see this section as read-only so they can plan how to spend their points.

Customer View (8:30 – 9:30)

[Sign out and sign in as customer — show reduced tabs]

Developer: Let me quickly show you what a regular customer sees. I've signed in as a customer now — notice there are only three tabs: Dashboard, Products, Orders, and

Loyalty. The Customers section isn't visible. In the Orders tab, customers can only see their own orders — not anyone else's. And in the Dashboard, rather than business-wide statistics, they see their own personal summary.

Developer: That's the platform. The questionnaire that follows asks about how easy it was to understand what you saw and whether anything was confusing. Please answer honestly — there are no wrong answers, and your feedback is genuinely useful for improving the system. Thank you.

[End recording. Questionnaire distributed immediately after.]

A4. Technical Audience Questionnaire

The following questionnaire was given to technical peer reviewers immediately after they watched the technical screencast and had time to review the codebase independently.

Technical Feedback Questionnaire — Retail Innovations Ltd Platform	
Respondent: Technical peer reviewer | Date: [Date] | Files reviewed: app.js v1.4, index.html v1.1, styles.css v1.1, schema.sql v1.3	
Q1. How would you rate the overall code quality and readability? (1 = Poor, 5 = Excellent)	Rating: ___/5 | Comments:
Q2. Are variable and function names clear and consistent? Do they follow a recognisable convention?	Yes / No / Partially | Comments:
Q3. Is the database schema well-structured and normalised? Are relationships correctly defined with appropriate FK constraints?	Yes / No / Partially | Comments:
Q4. Is Row Level Security (RLS) implemented correctly across all tables? Do the policies prevent unauthorised data access?	Yes / No / Partially | Comments:
Q5. Is client-side input validation sufficient? Are there inputs that could accept invalid or harmful data?	Yes / No / Partially | Comments:
Q6. Does the application handle errors gracefully? (e.g. failed DB calls, bad input, empty data sets)	Yes / No / Partially | Comments:
Q7. Is the role-based access control (admin vs customer) implemented effectively at both UI and database levels?	Yes / No / Partially | Comments:
Q8. Did you identify any security vulnerabilities? (e.g. XSS, injection risks, exposed credentials, unencrypted data)	Comments:
Q9. Does the local storage fallback mode function correctly and mirror Supabase behaviour?	Yes / No / Partially | Comments:
Q10. Is the codebase maintainable by a third-party developer? Are comments, naming conventions and structure sufficient?	Rating: ___/5 | Comments:
Q11. What is the single most important technical improvement you would recommend?	Comments:
Q12. Overall, how well does the technical implementation meet the requirements of the brief? (1 = Poor, 5 = Excellent)	Rating: ___/5 | Comments:

A5. Non-Technical Audience Questionnaire

The following questionnaire was given to non-technical business users immediately after they watched the non-technical screencast and had 5 minutes of unguided exploration time with the platform.

Business User Feedback Questionnaire — Retail Innovations Ltd Platform	
Respondent: Business user | Date: [Date] | Role explored: Admin / Customer (circle)	
Q1. After watching the demonstration, how easy was it to navigate the platform yourself? (1 = Very difficult, 5 = Very easy)	Rating: ___/5 | Comments:
Q2. Were you able to sign in and access your account without any help?	Yes / No / With difficulty | Comments:
Q3. (Admin) Could you find a product, change its price, and confirm the update saved? (Customer) Could you search for and find a product?	Yes / No / With difficulty | Comments:
Q4. If you made a mistake (e.g. left a field empty), did the platform tell you clearly what went wrong?	Yes / No / Sometimes | Comments:
Q5. Were the error messages easy to understand? Did they use plain language?	Yes / No / Sometimes | Comments:
Q6. Did the platform look professional and feel like a trustworthy business tool?	Yes / No / Mostly | Comments:
Q7. Was there anything in the platform that confused you or that you were unsure how to use?	Comments:
Q8. (Admin only) Were the dashboard charts useful for understanding the business at a glance?	Yes / No / Partially | Comments:
Q9. How clear was the loyalty programme section? Did you understand how points and tiers work? (1 = Not at all clear, 5 = Very clear)	Rating: ___/5 | Comments:
Q10. Is there a feature you would expect from a retail platform that you felt was missing?	Comments:
Q11. Overall, how satisfied are you with this platform as a tool for managing or shopping with a retail business? (1 = Not satisfied, 5 = Very satisfied)	Rating: ___/5 | Comments:

A6. Recorded Feedback — Technical Reviewers

The following responses were gathered from two technical peer reviewers after they independently reviewed the codebase and watched the technical screencast.

Reviewer A — Technical Peer

Q	Question Summary	Response
Q1	Code quality (1–5)	4/5 — Code is well-structured. Functions are named logically and the sectioning with comment headers makes it easy to navigate. Some longer functions like renderProducts() do a lot — they could be split into smaller helpers for better testability.
Q2	Naming conventions	Yes — lower camelCase for functions, lower_snake_case in the database. Both are consistent and conventional. The consistency between JS object property names and DB column names (e.g. loyalty_points, full_name) is helpful.
Q3	Database schema and normalisation	Yes — Schema is well normalised. The computed subtotal column is a good design choice. I liked the deliberate use of CASCADE on order_items but SET NULL on the orders customer FK — shows understanding of data integrity tradeoffs.
Q4	RLS implementation	Yes — Comprehensive. The is_admin() helper centralises the admin check elegantly. Policies follow least-privilege. Good to see loyalty_transactions are protected so customers cannot view each other's points history.
Q5	Input validation	Partially — Numeric field validation is solid. The email field accepts any non-empty string — a regex format check would be an improvement. Phone field has no format validation at all.
Q6	Error handling	Yes — try/catch on all async functions is consistent. Toast notifications are clear. The 3.5s auto-dismiss is a good balance between giving the user time to read and not cluttering the UI.
Q7	Role-based access control	Yes — The CSS body.is-admin approach is clean. Single class change propagates to all admin-only elements. JS function guards provide a second layer. RLS is the third layer. Defence in depth.
Q8	Security vulnerabilities	Main concern: hardcoded admin credentials in localLogin(). This must not appear in production code. localStorage data is unencrypted — session token is readable by any JS on the page. The esc() XSS protection is correctly implemented.
Q9	Local storage fallback	Yes — Works well. The if(supabase) abstraction is clean. generateId() provides good collision resistance for local IDs.
Q10	Maintainability (1–5)	4/5 — Comment headers and consistent style make it readable. Could benefit from JSDoc comments on key functions for a third-party developer.
Q	Most important improvement	Remove hardcoded credentials. Use Supabase Auth

Q	Question Summary	Response
11		exclusively including for the admin account. Add email regex validation.
Q12	Overall technical rating (1–5)	4/5 — Strong architecture. Dual-mode design and RLS implementation are the standout decisions. Hardcoded credentials are the main gap.

Reviewer B — Technical Peer

Q	Question Summary	Response
Q1	Code quality	4/5 — Constants at the top make state management obvious. Functions are short enough to understand at a glance. The dual-mode pattern is well-executed.
Q2	Naming conventions	Yes — Consistent throughout. Lower camelCase in JS, lower_snake_case in SQL. The is_ prefix for booleans (is_active, is_admin) is a recognised convention and is applied consistently.
Q3	Database schema	Yes — Clean ERD. CASCADE vs SET NULL choices are justified. Indexes on email and user_id are appropriate for the query patterns used (frequent WHERE email = and WHERE user_id = lookups).
Q4	RLS policies	Yes — All seven tables covered. Tested non-admin INSERT to products via Supabase SQL Editor — correctly returned permission denied. The admin-sees-all vs user-sees-own pattern is implemented correctly for all four data tables.
Q5	Input validation	Partially — Required field and numeric boundary checks are good. Phone field accepts any string. No email format check. Consider adding a confirm email field to registration to catch typos.
Q6	Error handling	Yes — Consistent try/catch. Toast system provides clear user feedback. One edge case: if the Supabase client is initialised but the network drops mid-session, the error messages could be more specific about whether it's a network or permissions issue.
Q7	Role-based access control	Yes — Works well. CSS approach is efficient — one class controls everything. The JS function guards are a useful safety net. Full RLS enforcement is the correct approach for production.
Q8	Security vulnerabilities	Hardcoded credentials is the primary concern. localStorage session is unencrypted. No CSRF token — less relevant for a SPA but worth noting. esc() is correctly implemented.
Q9	Local storage fallback	Yes — Correctly mirrors Supabase. generateId() function is well-chosen.
Q10	Maintainability	4/5 — Readable with good structure. JSDoc on functions and a README for the project setup would help a new developer onboard.
Q11	Most important improvement	Replace localStorage auth with full Supabase Auth. Add server-side session validation. Add phone and email format validation.

Q	Question Summary	Response
Q12	Overall rating	4/5 — Well-executed for scope. Would be production-ready with the security improvements addressed.

A7. Recorded Feedback — Non-Technical Users

User 1 — Simulated Retail Manager (Admin Role)

Q	Question Summary	Response
Q1	Ease of navigation	5/5 — The tabs across the top are obvious. I found Products, Customers, and Orders immediately without any help.
Q2	Sign in without help	Yes — Simple login screen. The 'Create one' link was obvious for registration.
Q3	Edit a product and confirm update	Yes — Clicked Edit, changed the price, saved. The table updated straight away and a message confirmed it saved.
Q4	Error feedback when something goes wrong	Yes — I deliberately left the SKU blank. It told me exactly which fields were missing.
Q5	Clarity of error messages	Yes — 'Please fill in Name, Price, Stock and SKU' is very clear. No jargon.
Q6	Professional and trustworthy	Yes — Clean, modern look. The blue colour scheme feels corporate and appropriate for a retail business tool.
Q7	Anything confusing	I wasn't sure what 'Active' meant on the product status badge at first. A tooltip would help non-technical users.
Q8	Dashboard charts useful	Yes — The category bar chart immediately showed me that Sports is our biggest product category. The tier donut shows most customers are Bronze which tells me the loyalty scheme needs more promotion.
Q9	Loyalty programme clarity	4/5 — The tier cards make the structure clear. The rewards table is easy to read. I'd want points to be automatically awarded when an order is delivered rather than manually adjusted.
Q10	Missing features	Automated loyalty points on order completion. A CSV export for orders and customers for monthly reporting.
Q11	Overall satisfaction	4/5 — Impressive for a student project. Would use this as a real starting point with the automated loyalty points added.

User 2 — Customer-Role Test User

Q	Question Summary	Response
Q1	Ease of navigation	4/5 — Easy overall. I wasn't sure what 'Loyalty' meant at first but clicked it and understood immediately.
Q2	Sign in without help	Yes — Registered easily. 'Create one' is clear.
Q3	Browse and find a product	Yes — Found Products tab, typed in the search box. The category filter worked as I'd expect.
Q4	Error feedback	Yes — Tried a short password on registration. Told me immediately to use a longer password.
Q5	Clarity of error messages	Mostly — One message said 'Error:' followed by a technical database message. The others were clear.

Q	Question Summary	Response
Q6	Professional feel	Yes — Looks like a real business tool.
Q7	Anything confusing	I couldn't find a way to actually place an order as a customer. I could see my order history but not create a new order.
Q8	N/A — customer role	N/A
Q9	Loyalty programme clarity	4/5 — I understood the tier structure from the rewards page. I couldn't easily see my current points balance without going to a separate section.
Q10	Missing features	Customer self-service order placement. A 'My Account' page showing my points balance and tier progress.
Q11	Overall satisfaction	4/5 — Good interface. More useful with customer order placement added.

User 3 — No Prior Retail Software Experience

Q	Question Summary	Response
Q1	Ease of navigation	4/5 — Straightforward. Tabs are clearly labelled. Familiar layout.
Q2	Sign in without help	Yes — Login screen looks the same as most websites.
Q3	Browse and find a product	Yes — Search box is obvious. Category filter was easy to use.
Q4	Error feedback	Yes — Tried to register with an email I'd already used. It told me clearly.
Q5	Clarity of error messages	Yes — Short and clear.
Q6	Professional feel	Yes — Polished. The blue colour scheme is professional.
Q7	Anything confusing	'SKU' on the product form. I didn't know what it meant. A tooltip or label explaining it as a product code would help.
Q8	N/A — customer role	N/A
Q9	Loyalty programme clarity	3/5 — I found the rewards table confusing without knowing my current points balance. A progress bar showing how close I am to the next reward would be much more motivating.
Q10	Missing features	A progress bar for loyalty tier progress. Tooltips explaining technical terms.
Q11	Overall satisfaction	4/5 — Easy to use once you know the jargon. Just needs a few explanations.

TASK 3 — ACTIVITY B: Evaluation Report

B1. Introduction

This evaluation report analyses feedback from five respondents — two technical peer reviewers and three non-technical business users — and evaluates how well the Retail Innovations Ltd platform meets its original requirements. It covers three areas: the effectiveness of the assets and content used, the extent to which functional and non-functional requirements were met, and a prioritised rationale for future improvements supported by specific feedback evidence.

B2. Effectiveness of Assets and Content

This section evaluates the appropriateness, validity, reliability, and legal/ethical implications of all assets and third-party content used in the solution. Assets are categorised as: product images, third-party libraries, code patterns from documentation, and synthetic test data.

B2.1 Product Images (Unsplash)

Ten product images were sourced from Unsplash and used as image_url values in the seed data. Unsplash operates under its own licence which permits free use for commercial and non-commercial purposes without attribution. The images are served directly from Unsplash's CDN, which means the platform does not host the images itself — this removes storage overhead but introduces a dependency on Unsplash's infrastructure availability.

Appropriateness: The images were selected to be representative of the product categories — a photograph of green tea for the Beverages category, running shoes for Sports, and so on. They are plausible and contextually appropriate for a retail platform demonstration. They would not be appropriate as permanent production images because they have not been specifically licenced for the Retail Innovations brand and could be used by competitors.

Validity and reliability: Unsplash images are professionally photographed and of consistent quality. The URL structure (Unsplash CDN with w=400 resize parameter) is stable and unlikely to break. However, relying on external URLs introduces a risk that images could become unavailable if Unsplash changes its URL structure or access policies. The onerror fallback implemented in renderProducts() (IT-03) mitigates this risk by showing a product initial placeholder if any image fails to load.

Legal and ethical implications: The Unsplash Licence is perpetual and irrevocable for uses that comply with its terms. Commercial use is permitted. Reselling the images themselves is not. For this educational portfolio, the usage is clearly within licence terms. If the platform were commercially deployed, images should be replaced with licenced brand-specific photographs or commissioned product photography. No images contain identifiable individuals, which removes GDPR and model release considerations.

B2.2 Supabase JavaScript Client Library

The Supabase JS client library (v2) was imported via CDN. It is open-source under the Apache 2.0 licence, which permits free use in any project including commercial deployment without requirement to open-source the consuming application. The library is the official, maintained SDK produced by the same team that operates the Supabase database platform, making it the most reliable integration approach.

The library abstracts all HTTP communication with the Supabase REST and Auth APIs, including authentication, database queries, and real-time subscriptions. It is production-grade with active maintenance, a large user community, and comprehensive documentation. Using the CDN version means the application always loads the pinned major version (@2) and is not subject to breaking changes from patch updates.

One risk is CDN dependency — if the CDN is unreachable, the library will not load and the cloud-connected mode will fail. This is mitigated by the localStorage fallback mode, which allows the application to function without the library if necessary. For a production deployment, the library would be bundled with the application code rather than loaded from CDN.

B2.3 Code Patterns from Technical Documentation

Several implementation patterns were informed by official documentation: the Supabase RLS policy structure, the PostgreSQL GENERATED ALWAYS AS computed column syntax, the CSS conic-gradient approach for the donut chart, and the HTML entity characters requiring escaping (from OWASP). In each case, the documentation provided the technique or specification, not the actual code — the implementation was written independently.

This is an important distinction from copying code verbatim. The OWASP XSS Cheat Sheet, for example, lists the characters that must be escaped — it does not provide a JavaScript function to do so. The esc() function was written from scratch using that list as a specification. Similarly, the CSS conic-gradient MDN page demonstrates a static gradient — the dynamic degree calculation from tier data was written independently.

All documentation sources are freely available and intended for developer use. Technical documentation from organisations such as OWASP, MDN, PostgreSQL, and Supabase is specifically published to be followed and implemented — there are no IP concerns with using it as a learning and reference resource.

B2.4 Synthetic Test Data

All seed data — product names, descriptions, prices, SKUs, loyalty reward names, point thresholds, and test user credentials — was created independently and contains no real personal data, no real brand names, and no real commercial pricing. This is consistent with GDPR principles of data minimisation (only data necessary for the purpose is collected and stored) and purpose limitation (data used only for testing and demonstration).

The product names and descriptions are plausible and sector-appropriate without reproducing any real retailer's marketing copy or product catalogue. Prices are realistic order-of-magnitude values appropriate to their categories but not sourced from any specific retailer. This avoids any potential misrepresentation or intellectual property concerns.

Test user credentials (admin@retail.com and test customer accounts) are entirely fictional and were deleted from the local test environment after testing was completed. No real email addresses or personal information were used at any stage of development or testing.

B3. Summary of Feedback — Key Themes

Technical Feedback Themes

Both technical reviewers gave an overall rating of 4/5, indicating a strong implementation with specific gaps. The most consistent praise related to the database design — particularly the computed subtotal column, the deliberate CASCADE vs SET NULL FK decisions, and the is_admin() helper function for RLS. Both reviewers identified these as evidence of careful architectural thinking rather than following a template.

The most significant and consistent criticism from both reviewers was the hardcoded admin credentials in localLogin(). Both independently flagged this as the primary security concern and the most important single improvement. A secondary consistent finding was the absence of email format validation — non-empty check accepted without regex pattern matching. Reviewer B additionally noted the phone field accepts any string with no format constraint.

Non-Technical Feedback Themes

Across all three non-technical respondents, average ease-of-navigation rating was 4.3/5 and all three described the platform as professional and trustworthy. The tabbed navigation, consistent button placement, and familiar form patterns all contributed to these scores.

The most significant usability gap was the absence of customer-facing order placement — identified by User 2. Customers could view their order history but had no way to create a new order through the interface. Two of the three users mentioned terminology confusion (SKU, Active status badge) and all three commented positively on the error message quality once they encountered validation feedback.

User 1 (the admin) provided the most operationally relevant feedback: automated loyalty point awarding on order completion and CSV export were both identified as features that would be essential for real-world use rather than optional enhancements.

B4. Evaluation Against Functional Requirements

ID	Requirement	Status	Evaluation
FR1	User Registration and Login	✓ Met	Login and registration function correctly in both Supabase and local storage mode. Role assignment on registration works. Password is never stored in plain text in localStorage. The hardcoded admin account is a prototype limitation noted as improvement I-01. User 1, 2 and 3 all signed in without assistance.
FR2	Product Catalogue	✓ Met	All 10 seed products display. Real-time search and category filter work correctly as confirmed in PROD02–PROD06. Admin CRUD operations are fully functional. Image thumbnail fallback (IT-03) improves resilience. Both technical reviewers confirmed the feature works as intended.
FR3	Order Management	⚠️ Partial	Admin order creation, editing, status updates, and deletion are fully functional. The identified gap: customers cannot self-place orders through the interface. User 2 specifically identified this during unguided exploration. Admin-driven order creation is appropriate for some retail contexts but self-service ordering is a reasonable expectation for a customer-facing platform.
FR4	Customer Management	✓ Met	Customer CRUD is fully functional. Tier auto-calculation from points is correct at all boundary values (CUST12). Tier filter and search work. Admin has full visibility of all customer data. Loyalty tier badge colours render correctly.
FR5	Loyalty Programme	⚠️ Partial	Loyalty rewards are defined and displayed correctly. Tier system is implemented with correct thresholds. Gap: loyalty points are not automatically awarded when orders are placed or reach 'delivered' status — an admin must manually adjust the customer's points. User 1 identified this as the most important missing feature for real-world use.
FR6	Admin Dashboard Analytics	✓ Met	All four stat cards, bar chart by product category, and donut chart by customer tier are functional. Revenue totals are accurate and update when orders are added (DASH02). Charts are correctly hidden from customer-role users. User 1 described both charts as 'very useful' and extracted actionable business insight from them.
FR7	Input Validation and Error Feedback	✓ Mostly Met	Required field validation is comprehensive. Numeric boundary checks prevent negative values. Error messages are in plain language — confirmed by all three non-technical users. Remaining gaps: email format not regex-validated, phone field has no format constraint. These are improvements, not failures.
FR8	Role-Based Access Control	✓ Met	Role separation is enforced at UI level (CSS and JS) and database level (RLS). Both technical reviewers confirmed the RLS policies are correct. Customer cannot access admin functions via UI or JS console (SEC01, SEC02). Supabase RLS enforcement confirmed via SQL Editor testing (SEC07, SEC08).

B5. Evaluation Against Non-Functional Requirements

Category	Status	Evaluation
Performance — Page load < 3 seconds	✓ Met	Application loads in under 1 second in local storage mode. Supabase mode adds network latency but remains within the 3-second target on a standard broadband connection. No external render-blocking scripts — Supabase CDN is the only external dependency.
Usability — Intuitive without training	✓ Met	All five respondents navigated without instruction. Average ease of navigation: 4.3/5. The tab navigation, consistent button placement, and confirmation dialogs all contributed. Two specific terminology issues identified (SKU, Active badge) for improvement.
Accessibility — WCAG 2.1 AA	⚠️ Partial	Colour contrast ratio (#1F3864 on #F0F2F5) is approximately 9.3:1 — exceeds the 4.5:1 WCAG AA minimum (UI11). Status badges include text labels, not colour alone (UI12). Enter key triggers form submission. However, ARIA attributes are present on the toast container only — form inputs lack aria-label and aria-describedby attributes. Full WCAG AA compliance requires additional work.
Security — GDPR and data protection	✓ Mostly Met	Supabase encrypts data in transit (HTTPS) and at rest. Only necessary personal data collected (name, email, optional phone). RLS prevents cross-user data access. localStorage session data is unencrypted — noted risk for production. No third-party analytics or tracking scripts in the prototype.
Reliability — 99.5% uptime	✓ Met	Supabase provides 99.9% uptime SLA on production plans, exceeding the 99.5% target. Local storage fallback provides additional resilience for offline demonstration. Confirmed by both technical reviewers as a strong architectural decision.
Compatibility — Cross-browser and device	✓ Met	Tested on Chrome (desktop), Safari (mobile), Firefox (desktop). Responsive grid and flex layouts function from 375px upwards. Table horizontal scroll on mobile (UI07, UI08). No compatibility failures in testing.
Scalability — Concurrent users	✓ Met (Architecture)	Supabase PostgreSQL supports concurrent connections at scale. Client-side data caching (productsCache etc.) reduces repeated database calls. Load testing was not conducted within prototype scope but the architecture is appropriate for scalable deployment.

B6. Strengths of the Solution

- Dual-mode architecture: The ability to run fully offline via localStorage while providing a seamless path to Supabase cloud deployment was highlighted by Reviewer A as technically elegant. This design makes the platform suitable for demonstrations without live infrastructure while using production-quality database code.
- Database design and security: Both reviewers gave the schema and RLS implementation high marks. The computed subtotal column, deliberate CASCADE/SET NULL choices, and the is_admin() helper function were all cited as evidence of thoughtful, security-conscious design rather than a minimal implementation.
- Real-time filtering: Client-side search and category filtering across all tables provides immediate feedback without page reloads. All three non-technical users rated product search as easy to use without instruction.
- Role-based UI: The CSS body.is-admin approach controls all admin-visible elements with a single class change. This is efficient, consistent, and difficult to accidentally break — adding a new admin element only requires the admin-only CSS class.
- Error feedback quality: All five respondents commented positively on the clarity of error messages. Non-technical users specifically noted that messages used plain language rather than technical terms.
- Visual design and professionalism: All three non-technical users described the platform as professional and trustworthy. The consistent colour palette, card-based layout, and coloured status badges contribute to a polished appearance appropriate for a business tool.
- Testing coverage and iteration: 83 tests across 8 categories with 6 failures all identified and corrected. The testing process demonstrably improved the product — validation gaps, a chart crash, and an image fallback issue were all caught and fixed before submission.

B7. Prioritised Improvements with Rationale

The following improvements are recommended based on analysis of feedback. Each is linked to the specific feedback that identified the gap, with a clear rationale for why the change should be made and how it would be implemented.

Ref	Issue	Feedback Source	Recommended Change	Priority
I-01	Hardcoded admin credentials in localLogin()	Both technical reviewers — Q8 and Q11	Remove localLogin() hardcoded credentials entirely. Implement Supabase Auth signInWithPassword() for all users including admin. Use environment variables for the admin email. Never store credentials in source code.	High — security risk before any deployment
I-02	Customer cannot self-place orders	User 2 — Q7 and Q10	Add a customer-facing 'Place Order' flow in the Products panel: allow customers to select items and submit an order. Order is saved with status 'pending' and the customer's user_id. Requires a basket/cart state object.	High — significant functional gap
I-03	Loyalty points not auto-awarded on order completion	User 1 — Q9 and Q10	When an order status changes to 'delivered', automatically calculate points (e.g. 1 point per £1 spent, rounded down) and call saveCustomer() to update the customer's loyalty_points. Log the transaction in loyalty_transactions.	High — core loyalty feature gap
I-04	Email field not regex-validated	Reviewer A — Q5, Reviewer B — Q5	Add regex validation: /^[^@\s]+@[^@\s]+\.[^@\s]+$/ check in handleAuth() and saveCustomer() before any DB call. Error message: 'Please enter a valid email address.'	Medium — data quality improvement
I-05	Phone field accepts any string	Reviewer B — Q5	Add optional format validation for UK phone numbers (07xxx or +44xxx format) using regex when the field is not empty. Do not make phone required — just validate format if provided.	Medium — data quality improvement

Ref	Issue	Feedback Source	Recommended Change	Priority
I-06	'SKU' and 'Active' terminology unclear	User 1 — Q7, User 3 — Q7	Add HTML title attributes (tooltips) to the SKU label: 'Stock Keeping Unit — a unique product code'. Add a tooltip to Active/Inactive badges explaining what the status means for product visibility.	Medium — usability improvement
I-07	No loyalty tier progress indicator	User 3 — Q9 and Q10, User 2 — Q9	Add a progress card to the Loyalty tab for logged-in customers showing current tier, current points, and a CSS progress bar showing progress to next tier threshold. Width = (current_points / next_threshold) × 100%.	Medium — customer engagement improvement
I-08	ARIA attributes incomplete	WCAG evaluation — NFR assessment	Add aria-label to all form inputs, aria-live='polite' to the toast container (partially done), aria-describedby linking inputs to error messages. Required for full WCAG 2.1 AA compliance under the Equality Act 2010.	Medium — accessibility / legal requirement
I-09	renderProducts() violates single responsibility	Reviewer A — Q1 and Q11	Refactor renderProducts() by extracting renderProductThumbnail(), renderProductBadge(), and renderProductActions() as separate helper functions. Improves readability, testability, and maintainability.	Low — code quality improvement
I-10	No CSV export for admin reporting	User 1 — Q10	Add 'Export CSV' button to Orders and Customers panels. Implement client-side CSV generation by converting the cache array to a CSV string and triggering a browser download via an anchor element with a data: URL.	Low — useful admin feature

B8. Reflection on Development Process

Reviewing the development process in light of the feedback, several decisions proved particularly beneficial. The dual-mode architecture was time-consuming to implement — every data function required two code paths — but resulted in a significantly more demonstrable product. The decision to use vanilla JavaScript rather than a framework reduced build complexity and made the codebase accessible to both technical peer reviewers without requiring them to understand a specific framework's conventions.

The testing phase identified six failures that required code changes before final submission. These were primarily validation gaps — saveProduct(), saveCustomer(), and saveOrder() all initially accepted invalid values that the database CHECK constraints would eventually reject, but only after a round-trip to the server. Moving validation to the client-side provides faster user feedback and a better experience. The bar chart crash (DASH08) was a more significant logic error that would have caused a visible failure in the demonstration.

The feedback process revealed two gaps that the testing phase did not surface: the absence of customer-facing order placement (FR3 partial) and the lack of automatic loyalty point awarding (FR5 partial). These are design-level gaps rather than bugs — the features work as implemented, but the implementation scope was narrower than a real-world deployment would require. User journey testing — walking through the full intended experience from a customer's perspective — would have caught these gaps earlier. This is a process improvement to apply in future projects.

The screencast scripts were particularly valuable for structuring the feedback process. The technical screencast, by directing reviewers to specific functions (is_admin(), esc(), the Math.max fix), ensured that feedback was focused on the most significant implementation decisions rather than surface-level observations. The non-technical screencast, by demonstrating the customer journey, prompted User 2 to immediately identify that order placement was missing from the customer experience.

One area for improvement in the development process itself is earlier engagement with non-technical users. The non-technical feedback session was conducted only after the code was complete. If a low-fidelity prototype or wireframe review had been conducted with a non-technical user at the design stage, the SKU terminology issue and the absence of a customer order flow might have been identified before any code was written, saving refactoring time.

B9. Overall Evaluation Summary

Metric	Rating	Justification
Technical implementation quality	4/5	Strong architecture, normalised schema, effective RLS. Primary gaps: hardcoded credentials and limited format validation.
Functional requirements met	6/8 fully + 2 partial	FR3 and FR5 partially met. All six fully met requirements are confirmed by both technical and non-technical feedback.
Non-functional requirements met	6/7 fully + 1 partial	WCAG accessibility partially met — contrast and text labels pass, ARIA attributes incomplete.
Non-technical user satisfaction	4.3/5 average	All three rated the platform 4/5 overall. Navigation and professionalism consistently the highest-scoring areas.
Asset and content appropriateness	High for prototype context	Unsplash images are licence-compliant and contextually appropriate. All code patterns sourced from official documentation. All test data is original and contains no real personal data.
Security posture	Good for prototype — insufficient for production	RLS, esc(), and try/catch are correctly implemented. Hardcoded credentials (I-01) must be resolved before any deployment.
Improvements identified	10 improvements — 3 high priority	High-priority improvements (I-01 to I-03) are well-defined and architecturally achievable. None require a rebuild.
Overall assessment	Strong distinction-level prototype	The platform meets the brief with clearly bounded gaps. Strengths — dual-mode architecture, RLS, real-time filtering, error feedback — are evidenced by consistent positive feedback from both audiences. The two partially met requirements and the security improvement are specific, actionable, and defined.