RETAIL INNOVATIONS LTD

Task 2: Documentation

Rationale • README • Schema • Changelog • Version Control

DPDD Occupational Specialism — Set Task

Document Version 1.0 | January 2025

1. Rationale for Choice of Languages/Tools

Each technology in the stack was selected based on specific criteria relevant to the project requirements:

Technology	Choice	Alternatives Considered	Rationale for Selection
Markup	HTML5	Pug, JSX, Handlebars	Industry standard with universal browser support. Semantic elements (<header>, <nav>, <main>, <section>) improve accessibility without any compilation step. Native form validation attributes (type, required, min, max) reduce JavaScript validation code.
Styling	CSS3 with Custom Properties	Sass/SCSS, Tailwind CSS, Bootstrap	No build step required — runs directly in browsers. CSS custom properties enable runtime theming (25+ variables). Grid and Flexbox provide responsive layouts without utility class bloat. Native animations and transitions eliminate JavaScript animation overhead.
Scripting	Vanilla JavaScript (ES6+)	React, Vue, Angular, jQuery	No framework dependencies means no build tools, no package.json, no node_modules. Faster initial load (no framework bundle). Full control over DOM manipulation. async/await for clean async code. Demonstrates core JavaScript competency rather than framework abstraction.
Database	Supabase (PostgreSQL)	Firebase, MongoDB, custom Express server	Free tier sufficient for development. Built-in

			authentication (bcrypt, JWT, email). RESTful API with auto-generated JavaScript client. Row Level Security for database-level access control. Real-time capabilities. Eliminates need for custom server code entirely.
Typography	Google Fonts (3 families)	System fonts, Adobe Fonts, self-hosted	Free and open-source (SIL License). Fast CDN delivery with display=swap. Three complementary families: Playfair Display (serif for headings), DM Sans (sans-serif for body), JetBrains Mono (monospace for data). Creates professional visual hierarchy.
Architecture	Single Page Application	Multi-page, SSR (Next.js)	No page reloads for instant navigation. Reduced server requests (data fetched via API). Tab-based navigation fits retail dashboard UX. Entire frontend in standard HTML/CSS/JS without compilation.

The deliberate choice of vanilla JavaScript over a framework was driven by three factors: assessment transparency (assessors can read every line without framework knowledge), deployment simplicity (works by opening index.html — no build process), and educational value (demonstrates understanding of core web technologies rather than framework-specific patterns).

2. README.md Completed and Clear

The project README.md file is structured as follows:

2.1 Project Overview

- Project title: Retail Innovations Ltd — Digital Retail Platform
- Brief description: A web-based retail management platform with product search, analytics dashboard, and loyalty programme
- Technology stack: HTML5, CSS3, JavaScript (ES6+), Supabase (PostgreSQL)

2.2 Features

- Product catalogue with real-time search and category filtering
- Retail analytics dashboard with KPI cards and visual charts
- Customer management with 4-tier loyalty programme
- Order management with status tracking
- Role-based access: Admin (full CRUD) and Customer (read + own orders)
- Responsive design: desktop, tablet, and mobile
- Supabase authentication with login/register

2.3 Setup Instructions

Step-by-step guide included in the README:

1. Create a free Supabase project at supabase.com
2. Open the SQL Editor in the Supabase Dashboard
3. Copy and paste the entire schema.sql file contents
4. Click Run to create all tables, triggers, and seed data
5. Copy the Project URL and anon public key from Settings > API
6. Open index.html in a browser
7. Enter the Supabase URL and anon key in the configuration form
8. Register a new account or login

2.4 Admin Setup

Instructions for promoting a user to admin role after registration:

```
UPDATE user_profiles SET role = 'admin'
WHERE email = 'your-email@example.com';
```

2.5 Folder Structure

index.html	Main application file — all HTML markup and page templates
styles.css	Complete stylesheet — dark theme, responsive, animations
app.js	Application logic — auth, CRUD, navigation, data caching
schema.sql	Database schema — tables, constraints, RLS, seed data
README.md	Project documentation

2.6 Known Limitations

- Customer-facing point redemption not yet implemented
- No product detail page — products shown in table format only
- No shopping cart or checkout flow
- Supabase configuration required on each visit (no environment variable deployment)

3. Supabase Schema Documented

The schema.sql file (199 lines) is comprehensively documented:

- 12 numbered sections with clear ASCII headers describing each component
- Every table includes column names, types, constraints, and their purpose
- Foreign key relationships documented with ON DELETE behaviour (CASCADE vs SET NULL)
- Index purposes explained (performance optimisation for frequently queried columns)
- Trigger functions include inline comments explaining their logic
- RLS policy names are descriptive English sentences (e.g., 'Users see own profile', 'Admins manage all orders')
- Seed data section includes realistic product and reward records
- Admin setup section provides step-by-step SQL commands with comments

4. Changelog Maintained

Version	Date	Type	Changes
0.1.0	Week 1	feat	Initial project: index.html structure, CSS variable system, dark theme base, Supabase CDN link
0.2.0	Week 1	feat	Auth system: login, register, logout, session check, role detection, profile loading
0.2.1	Week 1	fix	Profile race condition — added fallback defaults when trigger hasn't fired
0.3.0	Week 2	feat	Products CRUD: create, read, update, delete with search bar and category filter
0.3.1	Week 2	fix	Category filter not populating after product changes
0.4.0	Week 2	feat	Customers CRUD: management with loyalty tier tracking, tier visual cards
0.5.0	Week 3	feat	Orders CRUD: creation, role-based visibility, status management, customer join
0.5.1	Week 3	fix	Order modal admin fields visible to customers — added admin-only class
0.6.0	Week 3	feat	Loyalty rewards CRUD: type/value/points system, active/inactive toggle
0.7.0	Week 4	feat	Dashboard: KPI stat cards, CSS bar chart, CSS donut chart, responsive grid
0.7.1	Week 4	fix	Donut chart division by

			zero with no customers
0.8.0	Week 4	feat	Security: RLS policies (17), XSS prevention (esc function), admin role enforcement
0.8.1	Week 4	fix	Customer status dropdown exploit — disabled for non-admin
0.9.0	Week 5	feat	Responsive: 900px tablet breakpoint, 600px mobile breakpoint, touch tested
0.9.1	Week 5	fix	Modal form data retention between edit/create — explicit field clearing
1.0.0	Week 5	release	Final testing, documentation, code comments, README, changelog complete

5. Version Control Used Correctly (Git Commits)

5.1 Commit Practices

- Regular commits at logical completion points — each feature, bug fix, or refactoring is a separate commit
- Descriptive commit messages following conventional format: type: description
- Examples of commit messages used:

```
feat: add product search and category filter functionality
feat: implement login/register with Supabase Auth
feat: add CSS responsive breakpoints for tablet and mobile
fix: prevent XSS via HTML escaping utility function
fix: disable order status dropdown for non-admin users
docs: add comprehensive section comments to all files
style: add loyalty tier visual cards with metallic colours
refactor: extract parallel data loading into loadAllData()
```

5.2 Repository Structure

- Main branch used for stable, tested releases
- .gitignore file excludes: node_modules/, .env files, IDE configuration (.vscode/, .idea/)
- All source files tracked: index.html, styles.css, app.js, schema.sql, README.md
- No sensitive data (API keys, passwords) committed to the repository

5.3 Commit Frequency

Commits were made at each significant development milestone, averaging 2–3 commits per development session. The commit history provides a clear narrative of how the project was built incrementally, from initial structure through feature development, bug fixes, and final documentation.