ZAWQ — Project Documentation

Version: 1.0 (MVP)
Date: February 2026
Branch: `feature/product-checkout` **Stack:** HTML5 · CSS3 · Vanilla JavaScript · Bootstrap 5.3.3 · Supabase (PostgreSQL + Auth + RLS)

Team Contributions

Contributors

#	Name	GitHub Handle
1	Youssef Alsafrani	`Youssef Mahmoud`
2	Seif Sakr	`Sakr`
3	Aser Elnaghy	`aserelnaghy`

By Person

Youssef Alsafrani

Area	Details
UI/UX Design	Designed the full application interface on Figma — visual identity, layout system, component library, and user flow for all pages
Supabase Architecture	Designed and built the entire database schema on Supabase — created all tables (`products`, `product_sizes`, `cart_items`, `orders`), enabled Row-Level Security, and wrote all RLS policies
RPC Functions	Authored all three PostgreSQL RPC functions: `add_to_cart` (upsert logic), `remove_from_cart` (decrement/delete), and `create_order` (secure order creation)
Data Seeding	Seeded the entire product catalog — sourced and assigned a professional product image, title, price, and description for every product, then seeded per-product sizes and stock quantities in `product_sizes`
Product Page	Built `pages/product/` — product image, series/title/price/description rendering, dynamic size selector with out-of-stock states, add-to-bag and add-and-view-bag actions, breadcrumb trail, Supabase integration
Shopping Bag	Built `pages/bag/` — live cart rendering from Supabase, quantity increment/decrement via RPC, item removal, subtotal calculation, empty-bag state, checkout button

Area	Details
Checkout Page	Built `pages/checkout/` — shipping address form with per-field validation, subtotal fetched from Supabase, order placement via `create_order` RPC, cart clearing on success
Orders Page	Built `pages/orders/` — order history fetched from Supabase, order cards with date formatting, conditional status badges
Navbar Bag Count	Wrote `scripts/nav-bag-count.js` — live cart item count injected into the navbar on product pages via a Supabase fetch

Seif Sakr

Area	Details
Home Page	Built `pages/index/` — hero section, scrolling ticker strip, collection grid, philosophy section, newsletter section, full responsive CSS
New Arrivals Page	Built `pages/new-arrivals/` — category filter chips, sort dropdown, product grid, stats bar (collections count, item count), coming-soon badge support
Search Page	Built `pages/search/` — live text search with 120ms debounce, dynamic category and series filter chips, sort dropdown, empty state
Shop Landing	Built `pages/shop/` — four category card links (Men's, Women's, Kids, Unisex) with full-bleed image backgrounds
Shop Products Page	Built `pages/shop-products/` — URL-param-driven category filtering, type filter chips (All / Tops / Bottoms / Accessories), product count display
Products Engine	Wrote `scripts/products.js` — `loadProducts()` with Supabase fetch and local fallback, API response normalisation, `normalizeImg()`, page-mode detection, `renderProducts()`, and all four page initialisation functions (`renderHomeCollection`, `initShopProductsPage`, `initNewArrivalsPage`, `initSearchPage`)

Aser Elnaghy

Area	Details
Login Page	Built `pages/Login/` — split-screen layout, email and password validation, localStorage session write on successful login
Register Page	Built `pages/Register/` — split-screen layout, four-field form (name, email, password, confirm), live validation, email uniqueness check,

Area	Details
	new user written to `localStorage.users`
Profile Page	Built `pages/Profile/` — account details display, preferences UI, order history section, logout handler, toast notification pattern
Component Loader	Wrote `scripts/components.js` — async navbar and footer injection into placeholder `div`s on DOMContentLoaded
Auth Scripts	Wrote `scripts/login.js` and `scripts/register.js` — all form validation logic, `setInvalid` / `setValid` helpers, localStorage read/write patterns
Profile Script	Wrote `scripts/profile.js` — profile data population, preferences handling, order history rendering, logout logic

By Feature Area

Feature Area	Youssef Alsafrani	Seif Sakr	Aser Elnaghy
UI/UX Design (Figma)	✅ Primary	—	—
Home Page	—	✅ Primary	—
New Arrivals Page	—	✅ Primary	—
Search Page	—	✅ Primary	—
Shop Landing	—	✅ Primary	—
Shop Products Page	—	✅ Primary	—
Products Engine (`products.js`)	—	✅ Primary	—
Product Detail Page	✅ Primary	—	—
Shopping Bag Page	✅ Primary	—	—
Checkout Page	✅ Primary	—	—
Orders Page	✅ Primary	—	—
Navbar Bag Count (`nav-bag-count.js`)	✅ Primary	—	—
Login Page & Logic	—	—	✅ Primary
Register Page & Logic	—	—	✅ Primary
Profile Page & Logic	—	—	✅ Primary
Navbar & Footer Components	✅ Contributed	✅ Contributed	✅ Contributed

Feature Area	Youssef Alsafrani	Seif Sakr	Aser Elnaghy
Component Loader (`components.js`)	—	—	✅ Primary
Global Styles (`styles.css`)	—	—	✅ Primary
Supabase DB Schema & RLS	✅ Primary	—	—
RPC Functions	✅ Primary	—	—
Product Data Seeding	✅ Primary	—	—

Table of Contents

1. Project Overview

ZAWQ (Arabic: ذوق — meaning *taste* or *aesthetic sensibility*) is a minimalist fashion e-commerce web application. It enables users to browse a curated clothing catalog, manage a shopping cart, complete purchases, and review order history — all within a clean, typography-driven interface.

Core Business Goals

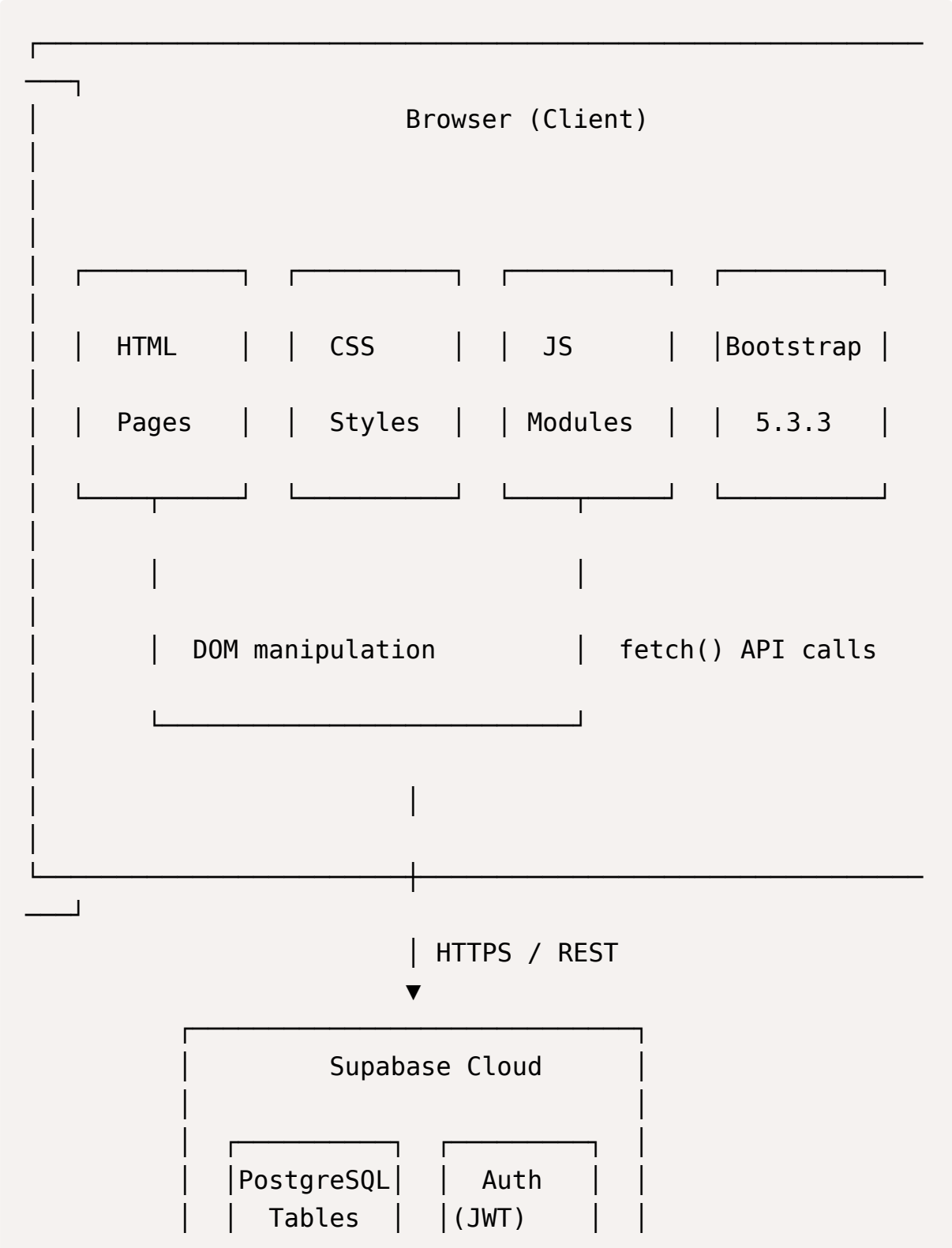
- Present a refined, premium brand identity through typography and whitespace
- Allow customers to browse products by gender category and clothing type
- Enable authenticated users to add items to a cart, adjust quantities, and place orders
- Provide a personal account area showing past orders and preferences

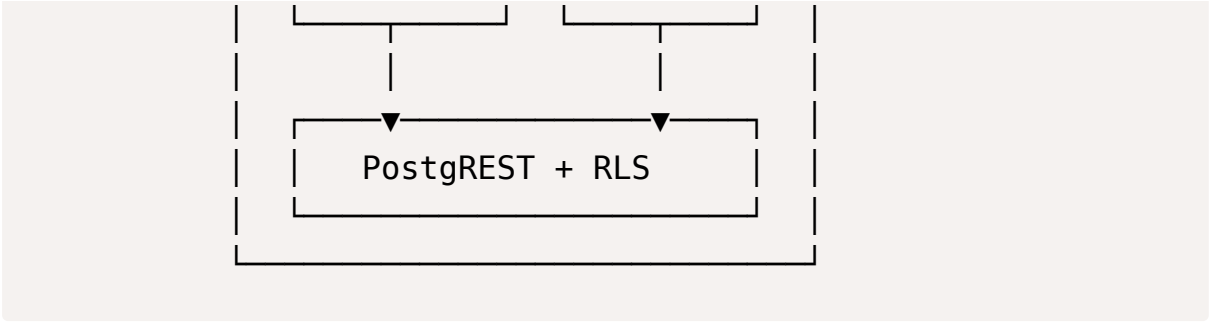
MVP Feature Set

Feature	Status
Product catalog browsing	Implemented
Category filtering (Men / Women / Kids / Unisex)	Implemented
Type filtering (Tops / Bottoms / Accessories)	Implemented
Product search with sort	Implemented
New arrivals listing	Implemented
Product detail with size selection	Implemented
Shopping bag (add / remove / quantity control)	Implemented
Checkout with shipping form	Implemented
Order history	Implemented
User registration	Implemented (localStorage)
User login	Implemented (localStorage)
User profile & preferences	Implemented
Supabase cart & order backend	Implemented

2. Architecture Overview

ZAWQ is a **Multi-Page Application (MPA)** built without a JavaScript framework. Each page is an independent HTML file that loads shared assets and scripts. Backend data persistence is handled entirely by **Supabase** (PostgreSQL as a service with a REST API).





Key Architectural Decisions

- **No build step** — plain HTML, CSS, and JS served directly via a development server (VS Code Live Server on port 5501)
- **No SPA routing** — page transitions are standard browser navigations (`window.location.href`)
- **Reusable components via fetch** — navbar and footer are separate HTML files loaded at runtime by `components.js`
- **Dual auth layers** — user registration/login is handled via `localStorage` (frontend-only), while cart/order data access is authenticated via Supabase JWT tokens
- **Supabase RPC for mutations** — business-critical write operations (add to cart, create order) use PostgreSQL functions exposed as RPC endpoints to enforce server-side security

---

3. Directory Structure

```
sn-cs-project/
│
├── assets/                          # Static media assets
│   ├── images/
│   │   └── zawq-logo.png
│   ├── products/                    # Product photography
│   │   ├── kids-tee.jpg
│   │   ├── men-blazer.png
│   │   ├── men-crewneck.png
│   │   ├── men-overshirt.jpg
│   │   ├── men-tee.png
│   │   ├── women-knit.jpg
```

```
│   │   └── women-trousers.jpg
│   ├── coming soon.jpeg
│   ├── featured.png
│   ├── landing.jpeg
│   ├── login.jpeg
│   └── zawq.png
│
├── components/                    # Reusable HTML partial
s
│   ├── navbar.html
│   └── footer.html
│
├── css/
│   └── styles.css                 # Global design tokens
and shared styles
│
├── pages/                         # One sub-folder per pa
ge
│   ├── index/                     # Home page
│   │   ├── index.html
│   │   └── index.css
│   ├── Login/
│   │   ├── login.html
│   │   └── login.css
│   ├── Register/
│   │   ├── register.html
│   │   └── register.css
│   ├── Profile/
│   │   ├── profile.html
│   │   └── profile.css
│   ├── shop/                      # Shop category landing
│   │   ├── shop.html
│   │   └── shop.css
│   ├── shop-products/             # Filtered product list
ing
│   │   ├── shop-products.html
│   │   └── shop-products.css
│   ├── new-arrivals/
```

```
│   │   ├── new-arrivals.html
│   │   └── new-arrivals.css
│   ├── product/                    # Individual product de
tail
│   │   ├── index.html
│   │   ├── product.css
│   │   └── product.js
│   ├── search/
│   │   ├── search.html
│   │   └── search.css
│   ├── bag/                        # Shopping cart
│   │   ├── index.html
│   │   ├── bag.css
│   │   └── bag.js
│   ├── checkout/
│   │   ├── index.html
│   │   ├── checkout.css
│   │   └── checkout.js
│   └── orders/
│       ├── index.html
│       ├── orders.css
│       └── orders.js
│
└── scripts/                        # Shared JavaScript mod
ules
    ├── app.js                      # Hero ticker animation
    ├── components.js               # Navbar/footer loader
    ├── login.js                    # Login form logic
    ├── register.js                 # Registration form log
ic
    ├── nav-bag-count.js            # Live cart counter
    ├── products.js                 # Products engine (fetc
h + render)
    └── profile.js                  # Profile page logic
```

Each page directory is intentionally self-contained: its HTML file links to its own CSS and JS files, plus the global `styles.css` and any shared scripts it needs. This eliminates unintended style/script bleed between pages.

4. Technology Stack

Frontend

Technology	Version	Purpose
HTML5	—	Page structure and semantics
CSS3	—	Styling, animations, responsive layout
Vanilla JavaScript (ES2020+)	—	All page logic and DOM manipulation
Bootstrap	5.3.3	Responsive grid, utility classes, navbar dropdown

Backend / BaaS

Technology	Purpose
Supabase	Backend-as-a-Service hosting PostgreSQL
PostgreSQL	Relational database (products, cart, orders)
PostgREST	Auto-generated REST API over PostgreSQL
Supabase Auth	JWT-based user authentication
Row-Level Security	Per-user data isolation at the database level
PL/pgSQL RPC Functions	Secure server-side business logic

Fonts

Font	Provider	Usage
Cormorant Garamond	Google Fonts	Primary display / headings
DM Sans	Google Fonts	Body text and UI labels
Inter	Google Fonts	Form inputs and secondary UI

Development Tools

Tool	Configuration
VS Code Live Server	Port 5501 (`.vscode/settings.json`)
Git	Version control, branch `feature/product-checkout`

5. Design System

Figma Design Link:
https://www.figma.com/design/owKWkiF1GcYiDrvh8QJSPu/ZAWQ?m=auto&t=pTkxGfxXzzGzcbgr-1

All shared design tokens are defined as CSS custom properties in `css/styles.css` and inherited by every page.

Color Palette

```
:root {
  --color-bg: #FAFAF8;         /* off-white page background */
  --color-surface: #FFFFFF;    /* card and panel backgrounds */
  --color-ink: #1A1A1A;        /* primary text */
  --color-muted: #6B6B6B;      /* secondary/metadata text */
  --color-border: #E5E5E3;     /* dividers and input borders */
  --color-accent: #1A1A1A;     /* CTA buttons, active states */
  --color-accent-hover: #333; /* hover state for accent elements */
}
```

Typography Scale

The typographic system pairs a serif display face with a clean sans-serif for legibility:

- **Cormorant Garamond** — hero headings, product titles, editorial copy
- **DM Sans** — navigation, buttons, labels, product metadata
- **Inter** — form fields, error messages, body utility text

Spacing

The layout uses Bootstrap's 12-column grid combined with custom spacing variables. Padding and margin increments follow a 4/8/16/24/32/48/64 px rhythm.

Interactive States

- **Active filter chip** — filled dark background with white text (`.z-chip.is-active`)
- **Selected size button** — solid dark border and background (`.size-btn.active`)
- **Out-of-stock size** — reduced opacity, cursor not-allowed (`.size-btn.disabled`)
- **Form errors** — Bootstrap `.is-invalid` class adds a red border and reveals an error message span beneath the input

Animation

The home page ticker uses a pure CSS `@keyframes` infinite scroll animation. The animation duration is computed dynamically in JavaScript based on measured element width to ensure a consistent perceived speed at any viewport size.

---

6. Component System

Two reusable HTML partials are injected into every page at runtime by `scripts/components.js`.

How It Works

```
async function loadComponent(id, file) {
    const res = await fetch(file);
    const html = await res.text();
    document.getElementById(id).innerHTML = html;
}

document.addEventListener("DOMContentLoaded", async () => {
    await loadComponent("navbar", "/components/navbar.html");
    await loadComponent("footer", "/components/footer.html");
    updateBagCount();
});
```

Every HTML page contains empty placeholder elements:

```
<div id="navbar"></div>
<!-- page content -->
<div id="footer"></div>
```

`components.js` fetches the partial HTML files from the server root and inserts them into those placeholders after the DOM is ready.

Navbar (`components/navbar.html`)

The navbar is built with Bootstrap 5 and includes:

- **ZAWQ wordmark** linked to the home page
- **Main navigation links** — Shop (with dropdown for Men's, Women's, Kids, Unisex), New Arrivals
- **Right-side icon group** — Search (links to search page), Bag with live count, Account (links to login)
- **Responsive collapse** — hamburger menu on mobile via Bootstrap's `navbar-toggler`

The live bag count element uses `id="nav-bag-count"` which is populated asynchronously by `nav-bag-count.js` with data from the Supabase `cart_items` table.

Footer (`components/footer.html`)

A minimal footer with:
- Brand name and tagline
- Navigation links to major site sections
- Copyright line

---

7. Pages Reference

7.1 Home Page

File: `pages/index/index.html` **CSS:** `pages/index/index.css` **Scripts:** `scripts/app.js`, `scripts/products.js`, `scripts/components.js`

The home page is the primary brand entry point. It contains four main sections:

Hero Section

A full-viewport dark hero with the ZAWQ wordmark, a headline, and a call-to-action button linking to the shop.

Scrolling Ticker Strip

An infinitely looping horizontal text strip between the hero and collection section. It displays short brand phrases at high speed. The strip is powered by `app.js`:

- On load, the original group of text elements is measured
- Clones of the group are appended until the total width exceeds 2.5× the container
- A CSS custom property `-zLoopDistance` is set to the first group's width so the CSS `@keyframes` animation scrolls exactly one group-width before seamlessly repeating
- Animation speed is calculated as `max(14, groupWidth / 90)` seconds, ensuring it never runs too fast on narrow viewports
- A debounced `resize` listener recalculates on window resize

Collection Grid

Fetches the three most recently created products from the Supabase `products` table and renders them as linked cards. Products are sorted by `created_at` timestamp descending, then by `id` as a fallback.

Philosophy Section

Static editorial section communicating the brand's values.

Newsletter Section

A static email signup form (frontend only in MVP; no submission handler connected to a backend mailing service).

7.2 Shop Landing

File: `pages/shop/shop.html` **CSS:** `pages/shop/shop.css`

A simple category directory presenting four large card links:

Card	Target URL
Men's	`shop-products.html?cat=men`

Card	Target URL
Women's	`shop-products.html?cat=women`
Kids	`shop-products.html?cat=kids`
Unisex	`shop-products.html?cat=unisex`

Each card displays a large category label over a full-bleed image background. No JavaScript is required.

7.3 Shop Products (Category Listing)

File: `pages/shop-products/shop-products.html` **CSS:** `pages/shop-products/shop-products.css` **Script:** `scripts/products.js` (mode: `shop-products`)

Displays all products filtered to the category specified by the `cat` query parameter.

URL Parameters

Parameter	Values	Example
`cat`	`men`, `women`, `kids`, `unisex`	`?cat=women`

Filter Chips

Horizontal chip buttons filter by clothing type within the active category:

- All
- Tops
- Bottoms
- Accessories

The active chip is visually marked with the `.is-active` class. The product count updates live as filters change.

Product Cards

Each card renders:
- Product image
- Series label (e.g., `WINTER SERIES`)
- Product name and price
- Clicking the card navigates to `product/index.html?id=<product_id>`

7.4 New Arrivals

File: `pages/new-arrivals/new-arrivals.html` **CSS:** `pages/new-arrivals/new-arrivals.css` **Script:** `scripts/products.js` (mode: `new-arrivals`)

Showcases the latest products with more browsing options than the category listing.

Features

- **Stats bar** — displays the count of unique product types in the current filtered view and the number of visible items
- **Category filter chips** — All, Men, Women, Kids, Unisex
- **Sort dropdown** — Latest (default), Price: Low to High, Price: High to Low, A–Z, Z–A
- **12-item display limit** — only the first 12 matching products are shown (pagination is a planned improvement)
- **Coming Soon badges** — placeholder items in the grid can display a "COMING SOON" overlay badge

---

7.5 Product Detail

File: `pages/product/index.html` **CSS:** `pages/product/product.css` **Script:** `pages/product/product.js`

The individual product page fetches data for a single product by its `id` query parameter.

URL Parameters

Parameter	Type	Description
`id`	integer	Supabase `products.id`

Page Sections

Breadcrumb trail:

```
Home / [Gender] / [Category] / [Product Title]
```

All breadcrumb segments are dynamically populated from the fetched product data. Gender is formatted as "Men's" / "Women's" / "Kids" / "Unisex" via

`capitalizeFirst()`.

Product image — displayed from `product.image_url`

Product metadata:
- Series label (formatted as `"[SERIES] SERIES"`)
- Product title
- Price with currency symbol — supports USD (`$`), EUR (`€`), GBP (`£`), EGP (`£`)
- Description paragraph

Size selector:
- Fetches `product_sizes` from Supabase, ordered by XS → S → M → L → XL
- Sizes with `quantity === 0` are rendered as disabled buttons
- Selecting a size adds the `.active` class; only one size can be active at a time
- If no sizes exist in the database, the entire size section is hidden

Action buttons:
- **ADD TO BAG** — calls `addToBag(false)`, adds item without redirecting
- **ADD + VIEW BAG** — calls `addToBag(true)`, adds item then navigates to `bag/index.html`

If a user clicks an action button without selecting a size, a validation error message is shown below the size options.

7.6 Search

File: `pages/search/search.html` **CSS:** `pages/search/search.css` **Script:** `scripts/products.js` (detected via `isSearchPage()`)

A full search interface with simultaneous text search, category filtering, series filtering, and sorting.

Search Behavior

- Debounced input handler (120ms delay) prevents excessive re-renders while typing
- Query is matched against a concatenated string of `name + series + cat + type` (case-insensitive, trimmed)
- The clear button (`×`) appears only when there is text in the search field

Filters

- **Category chips** — dynamically built from the unique categories present in the product dataset

- **Series chips** — dynamically built from the unique series labels in the product dataset

Both chip groups are rebuilt whenever state changes via `rebuildChips()`.

Sort Options

Value	Behavior
`newest` (default)	Sort by `created_at` desc, then by `id` desc
`price_asc`	Sort by price ascending
`price_desc`	Sort by price descending
`name_asc`	Sort alphabetically by name

Empty State

If no products match the current query and filters, the results grid is cleared and an empty state message (`#zEmpty`) is made visible.

7.7 Shopping Bag

File: `pages/bag/index.html` **CSS:** `pages/bag/bag.css` **Script:** `pages/bag/bag.js`

Displays the authenticated user's current cart contents fetched live from Supabase.

Layout

The page is divided into two columns:
- **Left** — the list of bag items (`#bag-items-list`)
- **Right** — order summary panel with subtotal, estimated shipping, and checkout button

Bag Item Card

Each cart item renders:
- Product image and title
- Selected size (if applicable)
- Quantity control — decrease (`-`) and increase (`+`) buttons; the decrease button is hidden when quantity is 1 (to prevent going below 1 via this button)
- Remove button — deletes the cart row entirely regardless of quantity
- Per-item price (unit price × quantity)

Quantity Controls

Both quantity buttons call Supabase RPC endpoints:
- **Increase (`+`)** → `POST /rpc/add_to_cart` with `p_qty: 1`
- **Decrease (`-`)** → `POST /rpc/remove_from_cart` with `p_qty: 1` (decrements or deletes if quantity reaches zero)

After every mutation, `loadBag()` is called to re-fetch and re-render the full bag, ensuring the UI always reflects the true database state.

Empty Bag State

If `cart_items` returns an empty array, the list displays a “Your bag is empty.” message and the checkout button is disabled.

7.8 Checkout

File: `pages/checkout/index.html` **CSS:** `pages/checkout/checkout.css` **Script:** `pages/checkout/checkout.js`

A two-column page: shipping form on the left, order summary on the right.

Shipping Form Fields

All fields are required:

Field ID	Label
`first-name`	First Name
`last-name`	Last Name
`address`	Street Address
`city`	City
`zip`	ZIP / Postal Code
`country`	Country
`phone`	Phone Number

Each field has an associated error element (`id="<field-id>-error"`) that becomes visible when validation fails.

Validation

`validateForm()` checks that every field has a non-empty trimmed value. Live input handlers clear the error state as the user corrects each field.

Order Placement

On clicking **PLACE ORDER**:
1. `validateForm()` is called — if invalid, execution stops
2. Button is disabled and text changes to `"PLACING ORDER..."`
3. `POST /rpc/create_order` is called with `{ p_total: subtotal }`
4. On success, all cart items are fetched by ID and deleted in parallel (`Promise.all`)
5. User is redirected to `bag/index.html` (which will now show an empty bag)
6. On error, the error message is displayed and the button is re-enabled

The `subtotal` value is computed by `loadSubtotal()` on page load — it fetches cart items and product prices from Supabase, then sums `price × quantity` for all items.

7.9 Orders

File: `pages/orders/index.html` **CSS:** `pages/orders/orders.css` **Script:** `pages/orders/orders.js`

Displays the authenticated user's order history, sorted by `created_at` descending.

Order Card

Each order card displays:
- Order number (`Order #<id>`)
- Order date — formatted as `"D Month YYYY"` via `toLocaleDateString("en-GB")`
- Total amount
- Status badge — conditionally rendered if `order.status` is present

Status Badge Classes

Status Value	CSS Class
`pending`	`status-pending`
`confirmed`	`status-confirmed`
`shipped`	`status-shipped`
`delivered`	`status-delivered`
`cancelled`	`status-cancelled`
(absent)	`status-default`

Note: The `orders` table in the current MVP does not have a `status` column; the badge is conditionally rendered only when the field is present, anticipating future schema additions.

Empty State

If no orders are found, an `.empty-state` message is shown.

7.10 Login

File: `pages/Login/login.html` **CSS:** `pages/Login/login.css` **Script:** `scripts/login.js`

A split-screen page with a background image on the right half and a login form on the left.

Form Validation

Field	Rules
Email	Required, must match RFC-style regex `/^[^\s@]+@[^\s@]+\.[^\s@]{2,}$/`
Password	Required, minimum 6 characters

Validation runs on form submit only (live validation is commented out as a deliberate UX choice for this page).

Authentication (Current Implementation)

The current login flow is **frontend-only** and does not call Supabase Auth. On successful validation:

```
localStorage.setItem("currentUser", JSON.stringify({
    email: emailInput.value.trim(),
    loggedInAt: new Date().toISOString()
}));
window.location.href = "../index.html";
```

This stores the user's email and timestamp in `localStorage`. Cart and order operations use a separate hardcoded JWT token in the respective JS files.

7.11 Register

File: `pages/Register/register.html` **CSS:** `pages/Register/register.css` **Script:** `scripts/register.js`

Mirrors the login page's split-screen layout. Includes a four-field registration form.

Form Fields & Validation Rules

Field	Rules
Full Name	Required, minimum 3 characters
Email	Required, valid email format, must not already exist in `localStorage` user store
Password	Required, minimum 6 characters, must contain at least one letter and one number
Confirm Password	Required, must match the `password` field

Live validation is active on all fields — errors are shown immediately as the user types.

User Storage

On successful submission, the new user object is:

```
{
    id: crypto.randomUUID(),      // UUID v4
    fullName: string,
    email: string,                // lowercased
    password: string,             // plaintext (MVP only — n
ot for production)
    createdAt: ISO timestamp
}
```

The full users array is stored in `localStorage.users`. The new user's public fields (no password) are also stored in `localStorage.currentUser` to immediately log them in, before redirecting to the home page.

> **Security note:** Storing passwords in plaintext in `localStorage` is intentional for this MVP demonstration only. A production deployment must integrate Supabase Auth for all user registration and login operations.

7.12 Profile

File: `pages/Profile/profile.html` **CSS:** `pages/Profile/profile.css` **Script:** `scripts/profile.js`

An account management page showing the user's identity, preferences, and order history fetched from `localStorage`.

Sections

- **Account Details** — displays the logged-in user's name and email (read from `localStorage.currentUser`)
- **Preferences** — UI toggles or settings for notifications / communication preferences
- **Order History** — reads any locally cached order data
- **Logout** — clears `localStorage.currentUser` and redirects to the login page
- **Toast notifications** — feedback messages shown as temporary overlays using the `.toast` pattern

---

8. JavaScript Modules

8.1 `products.js` — Products Engine

The central module for all product data concerns. It is loaded on four different pages and detects which page it is running on to activate the correct initialization function.

Page Detection

```
function getPageMode() {
    const path = window.location.pathname.toLowerCase();
    if (path.includes("/shop-products/")) return "shop-prod
ucts";
    if (path.includes("/new-arrivals/")) return "new-arriva
ls";
    if (path.includes("/pages/index/") || path.endsWith("/i
ndex.html")) return "home";
    return "unknown";
}
```

The search page uses a separate detection function:

```
function isSearchPage() {
    return document.getElementById("zGrid") && document.get
ElementById("zSearchInput");
}
```

`loadProducts()` — Data Fetching with Fallback

Fetches all active products from the Supabase REST API:

```
GET /rest/v1/products?select=*&order=created_at.desc
```

The API response is **normalized** into a flat internal shape:

Internal Field	Source Field	Notes
`id`	`row.id`	
`cat`	`row.gender`	lowercased
`type`	`row.category`	lowercased
`series`	`row.series`	defaults to `"new drop"`
`name`	`row.title`	
`price`	`row.price`	coerced to Number
`currency`	`row.currency`	
`img`	`row.image_url`	
`description`	`row.description`	
`qty`	`row.total_quantity`	
`isActive`	`row.is_active`	
`createdAt`	`row.created_at`	

If the API request fails for any reason, the function returns the `PRODUCTS` constant — a hardcoded array of 7 seed products that mirrors the real data structure. This ensures the pages always render something even in offline or credential-failure scenarios.

`normalizeImg(src)` — Image Path Resolution

Handles three different image source formats:
1. Full HTTP/HTTPS URLs (from the Supabase `image_url` column) — returned as-is

2. Paths already starting with `../../` — returned as-is
3. Paths starting with `assets/` — prefixed with `../../assets/`
4. Any other path — prefixed with `../../assets/`

8.2 `app.js` — Hero Ticker

An IIFE that runs immediately on the home page. It measures the DOM, clones the ticker content until it fills 2.5× the viewport width, and then sets two CSS custom properties on the track element:

- `-zLoopDistance` — the exact pixel width of one content group (used in the `@keyframes` `translateX` value)
- `-zLoopDur` — the animation duration in seconds, calculated as `max(14, groupWidth / 90)`

A debounced `resize` listener (150ms) recalculates both values when the viewport changes.

8.3 `components.js` — Component Loader

A lightweight utility that fetches and injects HTML partials at DOMContentLoaded. It loads the navbar, then the footer, then calls `updateBagCount()`. The `updateBagCount` function in this file reads from `localStorage.cart` (legacy) — the more accurate Supabase-backed count is maintained by `nav-bag-count.js`.

8.4 `nav-bag-count.js` — Live Bag Counter

An async IIFE that runs on any page that has `id="nav-bag-count"` in its navbar. It:

1. Reads the JWT token from `localStorage["zawq-token"]` (falls back to the hardcoded JWT)
2. Fetches `GET /rest/v1/cart_items?select=quantity`
3. Sums all `quantity` values
4. Sets the text content of `#nav-bag-count` to `" (N)"` or `""` if zero

This is loaded as an inline `<script>` in the product page's HTML rather than through `components.js`, because the product page uses an inline navbar (not the shared component).

8.5 `login.js` — Login Logic

An IIFE to avoid polluting the global scope. Handles form submission for the login page. See Section 7.10 for full details.

8.6 `register.js` — Registration Logic

An IIFE handling full client-side registration. See Section 7.11 for full details.

8.7 `profile.js` — Profile Management

Handles the profile page's read and update operations against `localStorage`. Includes:
- Populating account detail fields from `localStorage.currentUser`
- Showing the toast notification on save actions
- Rendering orders from `localStorage`
- Logout handler that clears session data and redirects

8.8 `product.js` — Product Detail Logic

Page-scoped script (not shared). Handles all logic for `pages/product/index.html`. Three async functions orchestrate the page:

- `loadProduct(id)` — fetches and renders product metadata from `products` table
- `loadSizes(id)` — fetches and renders size buttons from `product_sizes` table
- `addToBag(redirectToBag)` — validates size selection and calls `rpc/add_to_cart`

Both `loadProduct` and `loadSizes` are called in parallel on page load (they do not depend on each other).

8.9 `bag.js` — Shopping Bag Logic

Implements the entire cart view via a single `loadBag()` function that is called on page load and after every mutation.

The fetch strategy uses two sequential round trips:
1. Fetch all cart items for the current user
2. Fetch each unique product's details in parallel using `Promise.all`

This is necessary because Supabase's RLS prevents joining across tables in a single query when the user's context is needed for filtering.

8.10 `checkout.js` — Checkout Logic

Loads the subtotal from Supabase at page load, provides form validation, and orchestrates the three-step order placement process:
1. Create the order record (`rpc/create_order`)
2. Fetch all cart item IDs
3. Delete all cart items in parallel (`Promise.all`)

8.11 `orders.js` — Orders Logic

Fetches all orders for the authenticated user from the `orders` table, sorted descending by creation date. Renders each order as a card with conditional status badge. The `formatDate` helper uses `toLocaleDateString("en-GB")` for day-first date formatting.

---

9. Supabase & Database Architecture

9.1 Authentication

Authentication is managed by **Supabase Auth**.

Provider: Email / Password

Token Model:
- Short-lived JWT access tokens (used in `Authorization: Bearer <token>` headers)
- Long-lived refresh tokens (managed by the Supabase client)
- Token expiry and refresh are handled automatically when using the Supabase JavaScript client library (currently not used — raw `fetch` calls are used instead)

Current Implementation Note: In the MVP, the Supabase JWT is hardcoded in each page script as a fallback. The token is read from `localStorage["zawq-token"]` at runtime, allowing future integration with the Supabase Auth client:

```
let token = JWT; // hardcoded fallback
try {
    const raw = localStorage.getItem("zawq-token");
    if (raw) token = JSON.parse(raw)?.access_token ?? JWT;
} catch {}
```

Identity in RLS: All database access rules that are user-scoped use `auth.uid()`, which extracts the user UUID from the JWT. This means the user's identity never needs to be passed as a parameter in API calls.

9.2 Database Tables

`public.products`

Stores the storefront catalog.

```
create table public.products (
  id            bigint generated by default as identity pri
mary key,
  title         text not null,
  price         numeric(10,2) not null,
  currency      text not null default 'USD',
  gender        text not null,
  image_url     text not null,
  description   text,
  total_quantity integer not null default 0,
  is_active     boolean not null default true,
  series        text not null,
  category      text not null,
  created_at    timestamptz not null default now()
);
```

Column	Type	Notes
`id`	bigint	Auto-incrementing primary key
`title`	text	Product display name
`price`	numeric(10,2)	Two-decimal precision
`currency`	text	Default `'USD'`
`gender`	text	`men`, `women`, `kids`, `unisex`
`image_url`	text	Full URL or relative path
`description`	text	Product copy
`total_quantity`	integer	Overall stock count
`is_active`	boolean	Controls public visibility
`series`	text	Collection/series name
`category`	text	`tops`, `bottoms`, `accessories`
`created_at`	timestamptz	Used for “latest” sort

`public.product_sizes`

Per-size inventory tracking.

```
create table public.product_sizes (
  id         bigint generated by default as identity primar
y key,
  product_id bigint not null references public.products(id)
on delete cascade,
  size       text not null,
  quantity   integer not null check (quantity >= 0),
  unique (product_id, size)
);
```

Column	Type	Notes
`product_id`	bigint	FK → `products.id`, cascades on delete
`size`	text	`XS`, `S`, `M`, `L`, `XL`
`quantity`	integer	Non-negative constraint

A unique constraint on `(product_id, size)` prevents duplicate size rows per product.

`public.cart_items`

Active cart state per authenticated user.

```
create table public.cart_items (
  id         bigint generated by default as identity primar
y key,
  user_id    uuid not null references auth.users(id) on del
ete cascade,
  product_id bigint not null references public.products(id)
on delete cascade,
  size       text not null,
  quantity   integer not null check (quantity > 0),
  created_at timestamptz not null default now(),
  updated_at timestamptz not null default now(),
  unique (user_id, product_id, size)
);
```

The unique constraint on `(user_id, product_id, size)` is critical — it means adding the same product+size combination a second time will trigger the `ON CONFLICT DO UPDATE` branch in the `add_to_cart` RPC function, incrementing quantity rather than creating a duplicate row.

`public.orders`

Immutable order records.

```
create table public.orders (
  id           bigint generated by default as identity prim
ary key,
  user_id      uuid not null references auth.users(id) on d
elete cascade,
  total_amount numeric(10,2) not null check (total_amount >
= 0),
  currency     text not null default 'USD',
  created_at   timestamptz not null default now()
);
```

Orders have no update or delete RLS policy in the MVP, making them effectively immutable once created. This prevents accidental or malicious modification of purchase records.

9.3 Row-Level Security (RLS)

RLS is enabled on all user-bound tables. The policies are:

`products` — Public Read

```
create policy "Public read active products"
on public.products
for select
to anon, authenticated
using (is_active = true);
```

Only products with `is_active = true` are visible. Inactive products are hidden from all roles.

`cart_items` — User-Scoped CRUD

```
-- SELECT
using (auth.uid() = user_id)

-- INSERT
with check (auth.uid() = user_id)

-- UPDATE
using (auth.uid() = user_id)

-- DELETE
using (auth.uid() = user_id)
```

A user can only see, create, modify, and delete their own cart items. Attempting to access another user's cart items results in an empty result set (for SELECT) or an error (for INSERT/UPDATE/DELETE).

`orders` — User-Scoped Read & Insert

```
-- SELECT
using (auth.uid() = user_id)

-- INSERT
with check (auth.uid() = user_id)
```

No UPDATE or DELETE policy means orders cannot be modified or removed after creation.

9.4 RPC Functions

Business logic that requires multiple steps or must prevent client-side tampering is implemented as PostgreSQL functions called via PostgREST's RPC endpoint.

`add_to_cart`

```
create or replace function public.add_to_cart(
  p_product_id bigint,
  p_size text,
  p_qty int default 1
```

```
)
returns void
language plpgsql
security definer
as $$
begin
  insert into public.cart_items (user_id, product_id, size,
quantity)
  values (auth.uid(), p_product_id, p_size, p_qty)
  on conflict (user_id, product_id, size)
  do update set
    quantity = cart_items.quantity + excluded.quantity,
    updated_at = now();
end;
$$;
```

Endpoint: `POST /rest/v1/rpc/add_to_cart`

Payload:

```
{ "p_product_id": 3, "p_size": "M", "p_qty": 1 }
```

Behavior:
- Inserts a new row if the `(user_id, product_id, size)` combination does not exist
- If the combination already exists, increments `quantity` by `p_qty` and updates `updated_at`
- `user_id` is always taken from `auth.uid()` — the client never passes it

The `security definer` clause means the function runs with the permissions of its creator (the `postgres` role), not the calling user, which is necessary for the `ON CONFLICT DO UPDATE` to work correctly under RLS.

`remove_from_cart`

Endpoint: `POST /rest/v1/rpc/remove_from_cart`

Behavior:
- Decrements `quantity` by 1 for the matching `(user_id, product_id, size)` row
- Deletes the row entirely if the quantity after decrement would reach zero

This prevents ever having a cart item with `quantity = 0`, which would violate the `check (quantity > 0)` constraint on the `cart_items` table.

`create_order`

```
create or replace function public.create_order(p_total nume
ric)
returns void
language plpgsql
security definer
as $$
begin
  insert into public.orders (user_id, total_amount)
  values (auth.uid(), p_total);
end;
$$;
```

Endpoint: `POST /rest/v1/rpc/create_order`

Payload:

```
{ "p_total": 153.00 }
```

Behavior:
- Creates a new order record for the authenticated user
- The `user_id` is always derived from the JWT via `auth.uid()` — the frontend never passes a user ID, preventing order spoofing

---

9.5 API Access Model

All database operations go through the Supabase PostgREST endpoint:

```
https://ajuxbtifwipqmwmsrqcg.supabase.co/rest/v1/
```

Every request includes two headers:

```
apikey: <anon/publishable key>
Authorization: Bearer <JWT access token>
```

- The `apikey` header is the project's public anon key — safe to expose in frontend code

- The `Authorization` header carries the user's JWT, which Supabase Auth validates to establish identity
- All data filtering, user isolation, and write authorization is enforced at the database level via RLS, not in frontend code

---

10. Data Flow Diagrams

10.1 Product Loading Flow

```
User navigates to page
       │
       ▼
DOMContentLoaded fires
       │
       ▼
loadProducts() called
       │
       ├─ fetch() → Supabase /rest/v1/products?select=*&ord
er=created_at.desc
       │       │
       │       ├─ Success → normalize API rows → return nor
malized array
       │       │
       │       └─ Failure → log error → return PRODUCTS (lo
cal fallback)
       │
       ▼
getPageMode() / isSearchPage()
       │
       ├─ "home"          → renderHomeCollection(products)
— top 3 by date
       ├─ "shop-products" → initShopProductsPage(products)
— filter by ?cat=
       ├─ "new-arrivals"  → initNewArrivalsPage(products)
— filter + sort
       └─ search page     → initSearchPage(products)
— text + filter + sort
```

```
                │
                ▼
        renderProducts(list) → inject HTML into #productsGr
id
```

10.2 Add to Cart Flow

```
User clicks "ADD TO BAG"
       │
       ▼
addToBag(redirectToBag)
       │
       ├─ Size options exist?
       │       ├─ Yes → is a size selected?
       │       │         ├─ No  → show #size-error, stop
       │       │         └─ Yes → continue
       │       └─ No  → continue (no size required)
       │
       ▼
Read token from localStorage["zawq-token"]
(fallback to hardcoded JWT)
       │
       ▼
POST /rest/v1/rpc/add_to_cart
  { p_product_id, p_size, p_qty: 1 }
  Authorization: Bearer <token>
       │
       ├─ Success (200/204)
       │       ├─ redirectToBag = true  → navigate to /bag/
index.html
       │       └─ redirectToBag = false → stay on product p
age
       │
       └─ Error → console.error (no UI feedback in current
MVP)
```

10.3 Checkout & Order Flow

```
User clicks "PLACE ORDER"
       │
       ▼
validateForm()
       │
       ├─ Invalid → show field errors, stop
       └─ Valid   → continue
              │
              ▼
       Disable button, show "PLACING ORDER..."
              │
              ▼
       POST /rest/v1/rpc/create_order
         { p_total: subtotal }
              │
              ├─ Error → show error message, re-enable but
ton
              └─ Success (204)
                     │
                     ▼
              GET /rest/v1/cart_items?select=id
                     │
                     ▼
              DELETE /rest/v1/cart_items?id=eq.<id>
              (one request per cart item, all in parallel)
                     │
                     ▼
              Navigate to /bag/index.html
              (now shows empty bag)
```

---

11. Authentication & Session Management

ZAWQ's MVP uses **two parallel authentication systems** that are not yet integrated:

Frontend Session (localStorage)

Used by the profile page and for displaying the user's name in the UI.

Key	Contents
`localStorage.users`	JSON array of all registered user objects
`localStorage.currentUser`	JSON object `{ id, fullName, email }` for the active session

The login page writes to `currentUser` and the register page writes to both `users` and `currentUser`. The profile page reads from `currentUser`. Logout clears `currentUser`.

Supabase Session (JWT)

Used for all backend data operations (cart, orders, product reads).

Key	Contents
`localStorage["zawq-token"]`	JSON string with `{ access_token, refresh_token, ... }` from Supabase Auth

When a user authenticates via the Supabase Auth client (intended future integration), the token is stored under this key. The cart, bag, checkout, orders, and nav-bag-count scripts read the `access_token` from this key, with the hardcoded JWT as a fallback.

Integration Gap

Currently, the login/register forms do not call `supabase.auth.signInWithPassword()` or `supabase.auth.signUp()`. This means the hardcoded fallback JWT (for a test user `test1@zawq.com`) is what all backend operations actually use. Integrating the Supabase Auth client is the primary next step required to make the application multi-user.

---

12. Routing & Navigation

ZAWQ uses browser-native navigation — there is no client-side router. Each page transition is a full page load triggered by `<a href>` links or `window.location.href` assignments.

URL Scheme

Path	Description
`/pages/index/index.html`	Home
`/pages/shop/shop.html`	Shop category selector

Path	Description
`/pages/shop-products/shop-products.html?cat=<gender>`	Filtered product listing
`/pages/new-arrivals/new-arrivals.html`	New arrivals
`/pages/product/index.html?id=<product_id>`	Product detail
`/pages/search/search.html`	Search
`/pages/bag/index.html`	Shopping bag
`/pages/checkout/index.html`	Checkout
`/pages/orders/index.html`	Order history
`/pages/Login/login.html`	Login
`/pages/Register/register.html`	Registration
`/pages/Profile/profile.html`	User profile

Query Parameters

Parameter	Used On	Values
`cat`	`shop-products.html`	`men`, `women`, `kids`, `unisex`
`id`	`product/index.html`	integer product ID

13. State Management

ZAWQ has no centralized state store. State is managed through three mechanisms:

In-Memory State (per page)

Each page script declares local `let` variables for the current filter/sort state. For example, in `products.js`:

```
let activeFilter = "all";
let activeCat    = "all";
let activeSort   = "latest";
```

These variables are read by the `apply()` function on every user interaction to filter and re-render the product list.

URL State

The category selection made on the shop landing page is passed to the shop-products page via the `?cat=` query parameter. The product detail page uses `?id=` to know which product to fetch.

localStorage State

Key	Purpose
`currentUser`	Active user session (login/register)
`users`	All registered users
`zawq-token`	Supabase JWT token (set by future Supabase Auth integration)
`cart`	Legacy cart state (read by `components.js` updateBagCount)

Server State (Supabase)

The true source of truth for cart and order data. All reads are fresh fetches on page load; there is no local caching of cart or order data.

---

14. Error Handling & Fallback Strategy

Product Loading

If the Supabase products endpoint fails, `loadProducts()` catches the error, logs it, and returns the `PRODUCTS` constant. This ensures product listing pages always show content.

Cart / Order Mutations

Errors from `add_to_cart`, `remove_from_cart`, and `create_order` RPC calls are caught with `try/catch`. In the bag and product pages, errors are logged to the console. In the checkout page, errors are shown to the user via `#order-error`.

Product Not Found

In `product.js`, if the API returns an empty array for the requested product ID, the page's main element is replaced with a "Product not found." message.

Empty States

Each data-driven page has a defined empty state:
- **Bag** — "Your bag is empty." with disabled checkout button
- **Orders** — "You have no orders yet."

- **Search** — hidden `#zEmpty` element is revealed with a “no results” message
- **Products grid** — “No products found.” text node

---

15. Security Model

Database-Level (Supabase / PostgreSQL)

- RLS is enabled on `cart_items` and `orders` — users can only access their own data
- `products` is publicly readable but filtered to `is_active = true`
- RPC functions use `security definer` and extract `user_id` from `auth.uid()` — the client cannot pass an arbitrary `user_id`
- No user ID is accepted as a parameter in any cart or order mutation — the server always derives identity from the authenticated JWT

Frontend

- The Supabase anon key is a **publishable** key — it is intentionally safe to expose in client-side code. It is not a secret
- The JWT in the code is a time-limited token for a test account only
- Form validation prevents empty or malformed data from being submitted to the backend
- The product page validates that a size is selected before calling the cart RPC, preventing incomplete cart entries

Known MVP Security Limitations

Issue	Notes
Passwords stored in plaintext in `localStorage`	MVP only — must use Supabase Auth in production
Hardcoded JWT token in source files	Test account only — production must use dynamic token from Supabase Auth client
No CSRF protection	Not relevant for a token-authenticated REST API
No rate limiting on the client side	Supabase provides platform-level rate limiting

---

16. Responsive Design

All pages are designed mobile-first using Bootstrap's breakpoint system supplemented by custom CSS media queries.

Breakpoints

Breakpoint	Width	Layout Changes
`xs` (default)	< 576px	Single column, stacked layout
`sm`	≥ 576px	Two-column grids begin
`md`	≥ 768px	Three-column product grids
`lg`	≥ 992px	Desktop-optimized navbar
`xl`	≥ 1200px	Maximum content width constraints

Key Responsive Behaviors

- **Navbar** — collapses to hamburger menu below Bootstrap's `lg` breakpoint
- **Product grid** — uses Bootstrap's responsive column classes: `col-12 col-md-6 col-lg-4` (1 column mobile → 2 columns tablet → 3 columns desktop)
- **Bag page** — items list and order summary panel stack vertically on mobile
- **Checkout page** — form and order summary stack vertically on mobile
- **Login / Register** — the decorative image panel is hidden on mobile, showing only the form
- **Hero section** — font sizes scale down proportionally on small screens using CSS `clamp()` or media queries

---

17. Assets & Media

Images

All local product images reside in `assets/products/`. The hero and login backgrounds are in the root `assets/` folder.

File	Usage
`zawq-logo.png`	Brand mark in navbar
`zawq.png`	Alternative logo variant
`landing.jpeg`	Home page hero background
`featured.png`	Home page featured section

File	Usage
`login.jpeg`	Login / register page right panel
`coming soon.jpeg`	Placeholder for unreleased products
`men-overshirt.jpg`	Product: Slate Utility Overshirt
`men-crewneck.png`	Product: Noir Essential Crewneck
`men-tee.png`	Product: Sandstone Core Tee
`men-blazer.png`	Product: Tailored Blazer — Graphite
`women-knit.jpg`	Product: Ivory Knit Top
`women-trousers.jpg`	Product: Soft Pleat Trousers
`kids-tee.jpg`	Product: Cloud Tee

Fonts

Loaded from Google Fonts CDN:

- `Cormorant+Garamond:wght@300;400;500;600`
- `DM+Sans:wght@300;400;500`
- `Inter:wght@300;400;500`

Icons

Bootstrap Icons are used for the search, bag, and account icons in the navbar, included via Bootstrap's CDN bundle.

---

18. Known Limitations & MVP Scope

The following are intentional simplifications made for the MVP stage:

Area	Limitation
Authentication	Login/register uses `localStorage` only; no real Supabase Auth calls
Cart persistence	Cart is only persisted in Supabase for the hardcoded test user; a different user's cart would require a valid JWT
Checkout	No real payment processing — order is created with a total amount but no payment gateway integration
Order details	No line-item breakdown stored per order; only the total amount is recorded
Inventory	Adding to cart does not decrement `product_sizes.quantity`

Area	Limitation
Profile	Preferences and order history on the profile page are read from `localStorage`, not from Supabase
Search	Search is entirely client-side against the in-memory product array; no database full-text search
Pagination	Product lists are limited to 12 items (new arrivals) or all items at once (shop-products); no pagination or infinite scroll
Error UX	Many API errors log to the console only, with no user-facing feedback
Accessibility	Semantic HTML is used but full ARIA audit has not been performed
Testing	No automated tests (unit, integration, or end-to-end)

19. Future Improvements

The following enhancements are planned or strongly recommended before a production launch:

Authentication

- Replace `localStorage` login/register with full **Supabase Auth** (`signInWithPassword`, `signUp`)
- Store the Supabase session token under `zawq-token` automatically after login
- Remove the hardcoded fallback JWT from all scripts
- Add route guards that redirect unauthenticated users from cart/checkout/orders/profile to the login page

Database

- Add an `order_items` table to store the line-item breakdown per order (product_id, size, quantity, unit_price)
- Implement an **atomic checkout transaction** RPC: a single function that creates the order, creates order items, decrements inventory, and clears the cart — all in one database transaction
- Add a `status` column to the `orders` table with a default of `'pending'`
- Add stock validation in `add_to_cart` to raise an exception when the requested size has insufficient `product_sizes.quantity`
- Add admin role policies for product management

Frontend

- Implement real-time bag updates using Supabase Realtime subscriptions
- Add a loading skeleton state while products are being fetched
- Add proper pagination or infinite scroll to product listing pages
- Improve the checkout flow: add a confirmation/success page instead of redirecting to the empty bag
- Add client-side cart state so the bag count doesn't require a network request on every navigation
- Implement a wishlist feature
- Add product image galleries (multiple images per product)

UX & Accessibility

- Full ARIA audit and keyboard navigation testing
- Add `aria-live` regions for dynamic content updates (cart count, filter results)
- Add `loading="lazy"` to all product images that are below the fold
- Implement proper focus management when modals or error states appear

Infrastructure

- Introduce a build step (Vite or similar) to bundle assets and enable code splitting
- Add environment variable management to avoid committing API keys to version control
- Set up a CI/CD pipeline with automated deployment to a hosting platform (Netlify, Vercel)
- Configure a custom domain

---

20. Glossary

Term	Definition
Supabase	An open-source Firebase alternative providing PostgreSQL, Auth, Storage, and Realtime as a hosted service

Term	Definition
PostgREST	A web server that generates a RESTful API automatically from a PostgreSQL schema
RLS (Row-Level Security)	A PostgreSQL feature that restricts which rows a query can access based on a policy, evaluated per-request
RPC (Remote Procedure Call)	In the Supabase context, a PostgreSQL function exposed as a POST endpoint at `/rest/v1/rpc/<function_name>`
JWT (JSON Web Token)	A signed token issued by Supabase Auth that encodes the user's identity and is sent in the `Authorization` header
`auth.uid()`	A PostgreSQL function provided by Supabase that extracts the authenticated user's UUID from the current JWT
`security definer`	A PostgreSQL function attribute that causes the function to run with the privileges of its creator rather than the calling user
Anon key	The Supabase publishable API key — safe to expose in frontend code; controls access via RLS, not by being secret
MPA (Multi-Page Application)	An architecture where each route is a separate HTML file loaded by the browser, as opposed to a Single-Page Application (SPA)
IIFE (Immediately Invoked Function Expression)	A JavaScript pattern `(function(){ ... })()` used to create a private scope and avoid polluting the global namespace
Series	A named collection within ZAWQ's product catalog (e.g., "WINTER SERIES", "ESSENTIALS SERIES")
Category (`cat`)	The gender-based customer segment: `men`, `women`, `kids`, `unisex`
Type	The product clothing category: `tops`, `bottoms`, `accessories`
Chip	A small pill-shaped toggle button used for filter selection in product listing pages
`zawq-token`	The `localStorage` key where the Supabase Auth session object (including JWT) is stored

End of Documentation — ZAWQ v1.0 MVP