**EVARA E-COMMERCE**

FASHION STORE

*Project Technical Documentation*

|  |  |
| --- | --- |
| **Developed By** | Abd El-Rahman Magdy & Fatma El-Zahraa Ahmed |
| **Institution** | ITI - Information Technology Institute |
| **Location** | Knowledge City - Egypt |
| **Year** | 2026 |
| **Document Version** | 1.0 |
| **Supervisor** | Eng. Sandra Raouf |

# 1. Project Overview

Evara is a fully client-side fashion e-commerce web application built with HTML5, CSS3, Bootstrap 5, and JavaScript. The project demonstrates a complete online shopping flow, from browsing products on the homepage to completing a checkout process. It integrates with the DummyJSON public REST API for dynamic product data, and uses Supabase as the backend authentication service for user registration and login.

## 1.1 Project Purpose

The application was built as an academic front-end development project at ITI (Information Technology Institute). Its goal is to demonstrate mastery of modern web technologies, third-party API integration, user authentication via a Backend-as-a-Service (BaaS) platform, and client-side state management using browser localStorage.

## 1.2 Key Highlights

* Fully responsive design using Bootstrap 5 grid and utilities
* Dynamic product data fetched from DummyJSON REST API
* User authentication (Registration & Login) powered by Supabase
* Persistent shopping cart using browser localStorage
* Paginated product listing with real-time search functionality
* Category browsing via interactive Swiper.js carousel
* Complete checkout flow with order summary and shipping method selection

## 1.3 Project Technology Stack

|  |  |  |
| --- | --- | --- |
| **Category** | **Technology / Library** | **Version / Notes** |
| Markup | HTML5 | Semantic, valid HTML structure |
| Styling | CSS3 | Custom styles via external .css files |
| UI Framework | Bootstrap | v5.3.2 (CDN) |
| Icons | Font Awesome | v6.5.1 (CDN) |
| Slider / Carousel | Swiper.js | v12 (CDN) |
| Authentication Backend | Supabase | v2 JS client (CDN) |
| Product Data API | DummyJSON | https://dummyjson.com |
| State Management | localStorage | Native browser API |

**2. Project File Stru**

**cture**

The project is organized as a flat collection of HTML pages, JavaScript files, and CSS stylesheets. All pages share a common header (navigation bar with search, cart icon, and wishlist icon) and a consistent footer.

|  |  |  |
| --- | --- | --- |
| **File Name** | **Type** | **Role / Description** |
| index.html | HTML Page | Homepage - hero banner, category carousel, and featured products tabs |
| Products.html | HTML Page | Product listing page with pagination and category filter |
| ProductDetails.html | HTML Page | Single product detail view with add-to-cart, size selection, and rating display |
| Cart.html | HTML Page | Shopping cart - displays cart items, quantities, subtotal, shipping, and total |
| CheckOut.html | HTML Page | Checkout page - order summary and shipping/personal info form |
| RealLogin.html | HTML Page | Login form integrated with Supabase authentication |
| Register.html | HTML Page | Registration form with client-side password validation and Supabase sign-up |
| indexScript.js | JavaScript | Homepage logic: category carousel population (DummyJSON), Swiper init, tab nav |
| ProductsDetailsScript.js | JavaScript | Product detail logic: fetch single product, render details, add-to-cart function |

*Note: Additional pages referenced in the navigation (FAQ.html, ContactUs.html, TermsAndCond.html) and their corresponding CSS stylesheets (indexStyle.css, CartStyle.css, ProduuctDetailsStyle.css) are part of the project but were not included in this submission set.*

# 3. Page-by-Page Technical Documentation

## 3.1 index.html — Homepage

The homepage is the primary landing page of the Evara store. It is made up of three main content areas: a hero/banner section, a category carousel, and a tabbed featured-products section.

### 3.1.1 Sections & HTML Structure

* Top Header Bar: Displays the contact phone number, a promotional message, and a dynamic welcome span (#nameOfLoggedInSpan) that shows the logged-in user's first name fetched from localStorage key loggedInUser.
* Navigation Bar: Bootstrap responsive navbar with links to Home, Shop (Products.html), FAQ, Cart, and Contact Us. Includes a live search input (#InputBox) and a cart count badge (#CountCartAddedItems). The wishlist icon is present but purely decorative (no functionality implemented).
* Hero Section (.Home.section-lg): Full-width banner with promotional text (Hot promotions / Fashion Trending / Great Collection) and a call-to-action Shop Now button linking to Products.html.
* Category Carousel (.swiper.categorySwiper): A horizontal sliding carousel (#category-list) dynamically populated with product category cards via indexScript.js.
* Featured Products Section: Contains three tab buttons (Top Rated, Discount Hunter, New Arrivals) with IDs #TopRated, #Discount, and #NewArrivals. Product cards are injected dynamically into #CardsContainer.

### 3.1.2 JavaScript Logic (indexScript.js)

The FillTheSlider() async function is the core logic of the homepage:

|  |
| --- |
| async function FillTheSlider() {  const response = await fetch('https://dummyjson.com/products?limit=200');  const data = await response.json();  const allProducts = data.products;  // Filter only fashion-related categories  const fashionCategories = [  'mens-shirts', 'womens-tops', 'mens-shoes', 'mens-watches',  'womens-dresses', 'womens-shoes', 'womens-watches', 'womens-bags',  'womens-jewellery', 'sunglasses', 'fragrances', 'skin-care'  ];  // De-duplicate: keep one product per category  allProducts.forEach(product => {  if (!seen.has(product.category) && fashionCategories.includes(product.category)) {  seen.add(product.category);  uniqueCategories.push(product);  }  });  uniqueCategories.sort(() => Math.random() - 0.5); // Randomize order  // Inject each category into the Swiper wrapper  uniqueCategories.forEach(item => { wrapper.innerHTML += `...swiper-slide...`; });  // Initialize Swiper carousel  new Swiper('.categorySwiper', {  slidesPerView: 2, spaceBetween: 20,  breakpoints: { 640: { slidesPerView: 3 }, 1024: { slidesPerView: 6 } }  });  } |

Additionally, the file sets a default product ID in localStorage upon load:

|  |
| --- |
| localStorage.setItem('ChosenProduuctID', 83); |

*Note: The typo ChosenProduuctID (double 'u') is intentional as it is used consistently across indexScript.js and ProductsDetailsScript.js.*

The tab buttons (Top Rated, Discount Hunter, New Arrivals) use a simple active class toggle. The actual filtering logic for these tabs resides in a script block inside Products.html.

## 3.2 Products.html — Product Listing Page

This page displays all available products fetched from the DummyJSON API in a paginated grid. Users can browse by category and navigate between pages.

### 3.2.1 HTML Structure

* Navigation: Same shared navbar as all other pages, with live search and cart count badge.
* Products Section (#ProductsPage): A full-width section with the title 'TIMELESS PIECES FOR YOU'. Product cards are dynamically rendered into #containerProductsPage.
* Pagination Controls: Two buttons — #Btn-Previous (calls prevPage()) and #Btn-Next (calls nextPage()) — handle page navigation. The Previous button is initially disabled.
* Logged-in User Display: An anchor tag (#nameOfLoggedIn) in the top header links to Register.html and is dynamically updated to show the logged-in user's name.

### 3.2.2 Key JavaScript Features (inline script in Products.html)

* Fetches all products from https://dummyjson.com/products?limit=200
* Filters products to only show the 12 predefined fashion categories
* Implements client-side pagination: 8 products per page
* Each product card renders an image, title, truncated description, price, rating stars, and an 'Add to Cart' button
* Clicking a product card stores its ID to localStorage key ChosenProduuctID and navigates to ProductDetails.html
* truncateWords() helper function limits the description to 110 characters to keep card UI uniform
* addToCart(prdct, qunt) function (shared with ProductsDetailsScript.js pattern) saves items to localStorage key ShoppingCart

## 3.3 ProductDetails.html — Product Detail Page

This page shows detailed information about a single product. The product to display is determined by reading the ChosenProduuctID value from localStorage, which is set when the user clicks on any product card.

### 3.3.1 HTML Structure

* Breadcrumb navigation: Home > Shop (using Bootstrap breadcrumb component)
* Product Wrapper (#Productwrapper): An empty div inside a Bootstrap .row — all product content is injected dynamically via ProductsDetailsScript.js

### 3.3.2 JavaScript Logic (ProductsDetailsScript.js)

The ShowProductDetails() async function handles all rendering:

|  |
| --- |
| async function ShowProductDetails() {  const productID = localStorage.getItem('ChosenProduuctID');  const APIresponse = await fetch(`https://dummyjson.com/products/${productID}`);  const product = await APIresponse.json();  // Build star rating HTML  const rate = Math.ceil(product.rating);  for (let i = 1; i <= 5; i++) {  starinHtml += i <= rate ? '<i class="fa-solid fa-star"></i>'  : '<i class="fa-regular fa-star"></i>';  }  // Check availability: disable Add to Cart if not 'In Stock'  const isAvailable = product.availabilityStatus === 'In Stock';  wrapper.innerHTML = `... full product HTML template ...`;  // Attach event listener to Add to Cart button  document.getElementById('AddCartbtn').addEventListener('click', () => {  addToCart(product, parseInt(Productquantity.value));  });  } |

### 3.3.3 Rendered Product Detail Elements

* Product thumbnail image (large, contained, on a light background)
* Product title (h2)
* Star rating display (1–5 filled/empty Font Awesome stars based on Math.ceil of API rating)
* Brand name (highlighted in teal color #00797a)
* Current price, original price (calculated from price and discountPercentage), and discount badge
* Description paragraph
* Policy info list: Return Policy, Cash on Delivery note, Shipping Information
* Size selector buttons: M, L, XL, XXL (styling only, no functional size tracking)
* Quantity input (#ProductQuant) with minimum value of 1
* Add to Cart button — disabled if product.availabilityStatus is not 'In Stock'
* Wishlist heart button (decorative, no functionality)

### 3.3.4 addToCart() Function

|  |
| --- |
| function addToCart(prdct, qunt) {  let ShoppingCart = JSON.parse(localStorage.getItem('ShoppingCart')) || [];  const ProductExistance = ShoppingCart.findIndex(p => p.id === prdct.id);  if (ProductExistance === -1) {  // New product: push to cart  ShoppingCart.push({  id: prdct.id,  title: prdct.title,  img: prdct.thumbnail,  descrip: prdct.description,  price: prdct.price,  quantity: qunt  });  } else {  // Existing product: increment quantity  ShoppingCart[ProductExistance].quantity += qunt;  }  localStorage.setItem('ShoppingCart', JSON.stringify(ShoppingCart));  alert(`${prdct.title} added to cart successfully!`);  } |

The cart object stored in localStorage uses the following data shape per item:

|  |  |  |
| --- | --- | --- |
| **Property** | **Type** | **Source / Description** |
| id | Number | product.id from DummyJSON API |
| title | String | product.title from DummyJSON API |
| img | String (URL) | product.thumbnail URL from DummyJSON API |
| descrip | String | product.description from DummyJSON API |
| price | Number | product.price from DummyJSON API |
| quantity | Number | User-selected quantity from #ProductQuant input |

**3.4 Cart.html — Shopping Cart Page**

The cart page reads the ShoppingCart array from localStorage and renders all items in an HTML table. It also calculates subtotal, applies a fixed shipping fee, and displays the grand total.

### 3.4.1 HTML Structure

* Cart Items Table: A responsive Bootstrap table with columns: Image, Name, Price, Quantity (editable input), Subtotal, and Remove button.
* Table body (#TableBody) is populated dynamically via the ShowCartContent() JavaScript function.
* Cart Total Card: Displays three rows — Subtotal (#SubtotalSpan), Shipping (#ShippingSpan), and Total (#FinalTotal). A 'Proceed to Checkout' button links to CheckOut.html.

### 3.4.2 JavaScript Logic (inline in Cart.html)

|  |
| --- |
| function ShowCartContent() {  let TotalValue = 0;  let ShoppingCart = JSON.parse(localStorage.getItem('ShoppingCart'));  if (ShoppingCart.length === 0) {  TableBody.innerHTML = '<tr><td colspan="6">Your cart is empty!</td></tr>';  return;  }  ShoppingCart.forEach(product => {  TotalValue += (product.quantity \* product.price);  ItemsNumber++;  TableBody.innerHTML += `<tr>...product row...</tr>`;  });  localStorage.setItem('ItemsInCart', ItemsNumber);  document.getElementById('SubtotalSpan').textContent = '$' + TotalValue.toFixed(2);  document.getElementById('ShippingSpan').textContent = '$50'; // Fixed shipping fee  document.getElementById('FinalTotal').textContent = '$' + (TotalValue + 50).toFixed(2);  } |

* ShowCartContent() is called automatically on page load.
* RemoveFromCart(productId) filters out the item with the matching id from the ShoppingCart array, updates localStorage, and calls ShowCartContent() again to refresh the UI.
* Important: The quantity input in the table is rendered with the current value but changes to that input are NOT persisted back to localStorage in the current implementation — quantity updates require the user to remove and re-add items.
* A fixed shipping fee of $50 is always applied regardless of cart contents or delivery address.

## 3.5 CheckOut.html — Checkout Page

The checkout page presents the user's pending order alongside a shipping details form. It is the final step before order confirmation.

### 3.5.1 HTML Structure

* Order Summary Card: Lists all items from localStorage ShoppingCart with title, quantity, price, and description. Displays subtotal, delivery fee ($50 fixed), and total.
* Shipping Method Form (#CheckoutForm): Lets the user choose between Door Delivery or Store Pickup via radio buttons.
* Personal Info Section: Input fields for Full Name (#chkOutFullName) and Phone Number (#chkOutPhoneNumber).
* Address Details Section: Input fields for Street Address (#chkOutStreetAddress), City (#chkOutCity), and Zip Code (#chkOutZipCode).
* Additional Notes: A textarea (#notes) for delivery instructions.
* Proceed to Payment button (#id\_proceedToPayment) validates all fields and displays a confirmation alert.

### 3.5.2 JavaScript Logic (inline in CheckOut.html)

|  |
| --- |
| function addToOrders() {  let UserOrders = JSON.parse(localStorage.getItem('ShoppingCart')) || [];  UserOrders.forEach((order) => {  subtotal += parseFloat(order.price) \* order.quantity;  container.innerHTML += `...order item li...`;  });  const deliveryFee = 50;  document.getElementById('Total').textContent = `${(subtotal + deliveryFee).toFixed(2)} $`;  }  // Proceed to Payment validation  function testProceedBtn() {  if (doorDelivery.checked && fullName && phone && city && zip && street) {  alert('We will Ship your Order To your Address');  } else if (storePickup.checked && fullName && phone && city && zip && street) {  alert('We are waiting you in our store');  }  } |

*Note: The checkout form does not send data to any server or database in the current implementation. It is a front-end-only demonstration. Upon valid submission, a browser alert() confirms the order.*

## 3.6 RealLogin.html — Login Page

The login page provides a clean, centered form for registered users to authenticate. It uses the Supabase JavaScript client library for authentication.

### 3.6.1 HTML Structure

* A single-column centered card layout (using Bootstrap responsive col-\* classes).
* Email input (#login-email): type=email, required
* Password input (#login-password): type=password, required
* 'Keep me logged in' checkbox (#remember\_me): UI element only — no session persistence logic implemented.
* Log In submit button styled in the project's teal brand color (#00797a).
* Link to Register.html for users without an account.

### 3.6.2 JavaScript Logic & Supabase Integration

|  |
| --- |
| const supabaseUrl = 'https://qsmvtalwscuvqktjlleq.supabase.co';  const supabaseKey = 'eyJhbGciOiJIUzI1NiIsIn...'; // Anon public key  supabase = window.supabase.createClient(supabaseUrl, supabaseKey);  document.getElementById('login-form').addEventListener('submit', async (e) => {  e.preventDefault();  const email = document.getElementById('login-email').value;  const password = document.getElementById('login-password').value;  const { data, error } = await supabase.auth.signInWithPassword({ email, password });  if (!error) {  const { data: { user } } = await supabase.auth.getUser();  // Store user's full name from metadata  localStorage.setItem('loggedInUser', user.user\_metadata.fName + ' ' + user.user\_metadata.lName);  window.location.href = './index.html'; // Redirect to homepage  }  }); |

* The Supabase anon (public) key is embedded directly in the HTML — this is a standard practice for client-side Supabase projects since the anon key is safe to expose publicly.
* On successful login, the user's first and last name (stored as custom metadata during registration) are saved to localStorage under the key loggedInUser.
* The user is then redirected to index.html where the welcome message displays their name.
* Error handling is present in the code but the error message display is commented out (messageEl.textContent lines are commented). No visible feedback is shown to the user on failed login attempts.

## 3.7 Register.html — Registration Page

The registration page allows new users to create an account. It includes client-side password validation and uses Supabase's signUp() API method.

### 3.7.1 HTML Structure

* First Name input (#reg-firstName): type=text, required
* Last Name input (#reg-lastName): type=text, required
* Email input (#reg-email): type=email, required
* Password input (#reg-password): type=password, required. Helper text (#passwordCheck) displays validation feedback.
* Terms agreement checkbox (#AgrementCheck): required — links to TermsAndCond.html.
* Sign Up button (full-width, teal color).
* Link to RealLogin.html for existing users.

### 3.7.2 Password Validation Function

|  |
| --- |
| function PasswordConstraint(\_password) {  if (\_password.length < 8) {  passwordCheck.textContent = 'At least 8 Letters';  passwordCheck.style.color = 'red';  return false;  }  const specialCharPattern = /[^a-zA-Z0-9]/;  if (!specialCharPattern.test(\_password)) {  passwordCheck.textContent = 'Need Special character';  return false;  }  const uppercasePattern = /[A-Z]/;  if (!uppercasePattern.test(\_password)) {  passwordCheck.textContent = 'At least Capital letter';  return false;  }  return true; // All constraints passed  } |

* Minimum 8 characters
* At least one special character (non-alphanumeric)
* At least one uppercase letter

### 3.7.3 Supabase Registration Logic

|  |
| --- |
| const { data, error } = await supabase.auth.signUp({  email: email,  password: password,  options: {  data: { fName, lName } // Stored as user\_metadata in Supabase  }  });  if (!error) {  localStorage.setItem('loggedInUser', fName + ' ' + lName);  window.location.href = './index.html';  } |

The first and last name are stored as custom user\_metadata fields in Supabase. This allows them to be retrieved on login via user.user\_metadata.fName and user.user\_metadata.lName.

# 4. Data Flow & State Management

## 4.1 localStorage Keys Reference

The application uses the browser's localStorage as its primary state management mechanism. No session data is stored on a server (except Supabase authentication tokens managed by the Supabase SDK).

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Key Name** | **Data Type** | **Set By** | **Read By** | **Description** |
| ShoppingCart | JSON Array | ProductDetails.html, Products.html | Cart.html, CheckOut.html | Array of cart item objects (id, title, img, descrip, price, quantity) |
| ItemsInCart | Number (string) | Cart.html (ShowCartContent) | Navbar badge | Count of total items in cart for badge display |
| ChosenProduuctID | Number (string) | indexScript.js (default: 83), Products.html | ProductsDetailsScript.js | ID of the product selected for the detail view |
| loggedInUser | String | RealLogin.html, Register.html | index.html (#nameOfLoggedInSpan) | Full name (fName + lName) of the authenticated user |

## 4.2 DummyJSON API Integration

The project uses two DummyJSON API endpoints:

|  |  |  |
| --- | --- | --- |
| **Endpoint** | **Used In** | **Purpose** |
| GET /products?limit=200 | indexScript.js, Products.html | Fetch all products (up to 200). Then filter by fashion category, de-duplicate, and display. |
| GET /products/{id} | ProductsDetailsScript.js | Fetch a single product by ID (read from localStorage ChosenProduuctID). |

## 4.3 Supabase Authentication Flow

Both the registration and login pages share the same Supabase project URL and anon key. The authentication flow works as follows:

* Registration: User fills in the form → PasswordConstraint() validates → supabase.auth.signUp() is called with email, password, and custom metadata (fName, lName) → On success, name is stored in localStorage and user is redirected to index.html
* Login: User fills in email/password → supabase.auth.signInWithPassword() is called → On success, supabase.auth.getUser() retrieves the user object with metadata → Name stored to localStorage → Redirect to index.html
* Session: The Supabase SDK manages auth tokens automatically in localStorage. The project does not implement a logout feature or session expiry handling in the provided files.

**5. Shared Components**

## 5.1 Navigation Bar (Shared Across All Pages)

Every page shares an identical navigation bar structure. Key elements:

* Brand logo (images/logo.svg) links back to the top of the page
* Nav links: Home (index.html), Shop (Products.html), FAQ (FAQ.html), Cart (Cart.html), Contact Us (ContactUs.html)
* Live search input (#InputBox) with a results dropdown div (.ResultBox) — search functionality is implemented inline per-page
* Wishlist icon with a static count badge (.CountAddedItems) — value is always 0 as the wishlist feature is not fully implemented
* Cart icon linking to Cart.html with a count badge (#CountCartAddedItems) — value is populated from localStorage

## 5.2 Footer (Shared Across All Pages)

All pages share an identical footer with these sections:

* Newsletter Sign-up Bar: Email input with Subscribe button. Background color #abcac5 (soft teal).
* Contact Column: Displays logo, address (ITI - Knowledge City - Egypt), and phone number. Social media links: Facebook, Twitter, Instagram, YouTube.
* Address Links Column: Terms & Conditions, Contact Us
* My Account Column: Sign In (RealLogin.html), View Cart (Cart.html)
* Payment Gateways: A single image (images/payment-method.png) showing accepted payment logos
* Copyright Bar: '© 2026 ITI. All rights reserved' and developer credit

## 5.3 Color Scheme & Branding

|  |  |  |
| --- | --- | --- |
| **Color Name** | **Hex Value** | **Used For** |
| Primary Teal | #00797a | Buttons, links, active states, headings, badges |
| Soft Teal | #abcac5 | Footer newsletter bar background |
| Light Teal | #E0F2F1 | Info box backgrounds, light highlights |
| Dark Gray | #2D3748 | Body text |
| Medium Gray | #718096 | Subtitles, secondary text, muted labels |
| White | #FFFFFF | Card backgrounds, nav background |
| Light Gray | #F7FAFC | Alternating table rows, section backgrounds |

# 6. External Libraries & CDN Dependencies

|  |  |  |  |
| --- | --- | --- | --- |
| **Library** | **Version** | **CDN URL** | **Purpose** |
| Bootstrap CSS | 5.3.2 | cdn.jsdelivr.net/npm/bootstrap@5.3.2 | Layout, grid, components, utilities |
| Bootstrap JS Bundle | 5.3.2 | cdn.jsdelivr.net/npm/bootstrap@5.3.2 | Interactive components (navbar collapse, etc.) |
| Font Awesome CSS | 6.5.1 | cdnjs.cloudflare.com/ajax/libs/font-awesome | Icons (stars, trash, cart, social) |
| Font Awesome Kit | Custom | kit.fontawesome.com/eadda94817.js | Extended icon pack via personal kit |
| Swiper.js CSS | 12 | cdn.jsdelivr.net/npm/swiper@12 | Carousel styling |
| Swiper.js JS | 12 | cdn.jsdelivr.net/npm/swiper@12 | Category card carousel functionality |
| Supabase JS | 2 | cdn.jsdelivr.net/npm/@supabase/supabase-js@2 | Auth: signUp, signInWithPassword, getUser |

# 7. Setup & Running the Project

## 7.1 Requirements

* A modern web browser (Chrome, Firefox, Edge, Safari)
* A local web server (e.g., VS Code Live Server extension, XAMPP, or Python http.server) — required because the project fetches data from external APIs and uses ES modules
* Internet connection — required to load CDN libraries and fetch from DummyJSON API

## 7.2 Steps to Run Locally

1. Clone or download the project files to a local folder.
2. Ensure all HTML files, JS files, CSS files, and the images/ folder are in the same directory.
3. Open the project folder in VS Code and install the Live Server extension.
4. Right-click on index.html and select 'Open with Live Server'.
5. The browser will open at http://127.0.0.1:5500/index.html (or similar port).
6. To test authentication, register a new account via Register.html, then log in via RealLogin.html.

## 7.3 No Build Process Required

This project is a pure front-end application with no build tools, package managers, or compilation steps required. All dependencies are loaded from CDN links in the HTML <head> sections. Simply serve the static files and the application runs as-is.

# 8. Known Limitations & Future Improvements

## 8.1 Current Limitations

* Quantity Update Not Persisted: On the Cart page, the quantity input field is editable by the user, but changes are not saved back to localStorage. The user must remove and re-add items to change quantity.
* No Logout Functionality: The project does not include a logout button or session termination flow. Users remain 'logged in' (via localStorage name) until they clear browser storage.
* No Order Database: Checkout form submission does not send data to any server. Orders are not stored or processed — the confirmation is a browser alert() only.
* Wishlist Not Functional: The heart/wishlist icon exists in the UI on all pages but has no backing logic. The count badge always shows 0.
* 'Keep me logged in' Checkbox: Present in the login form UI but not wired to any session persistence logic.
* Size Selector Not Tracked: Size buttons (M, L, XL, XXL) on the product detail page have no event handlers. Selected size is not captured or stored in the cart.
* Error Display Commented Out: In both RealLogin.html and Register.html, the error message display logic is commented out. Users receive no feedback on authentication failures.

## 8.2 Suggested Future Improvements

* Implement quantity update persistence in the cart (add an Update Cart button that saves the new quantities to localStorage)
* Add a logout button that calls supabase.auth.signOut() and clears the loggedInUser localStorage key
* Connect the checkout form to a Supabase database table to store order history
* Implement wishlist functionality using a separate localStorage key (WishList array)
* Add category-based filtering on the Products page using URL query parameters
* Improve responsive design for the product detail size selector row
* Display proper error/success messages for all authentication operations

# 9. Additional Pages Documentation

The following three pages complete the full site structure. They are informational/utility pages that support the core shopping flow.

## 9.1 ContactUs.html — Contact Us Page

The Contact Us page provides store location information, direct contact details, and a message submission form. It also embeds a Google Maps iframe showing the store location.

### 9.1.1 HTML Structure

* Top Header Bar: Contact phone (+971 (58) 135-9871) and promotional message. Unlike other pages, this header does NOT include the logged-in user name display or a Register/Login link — it is the only page that omits this element from the top header.
* Breadcrumb Navigation: Home > Contact Us (using Bootstrap breadcrumb with '>' divider).
* Google Maps Embed (.map-section > .map-wrapper): A full-width iframe embedding Google Maps centered on Dubai Mall (coordinates: 25.197, 55.274). Key iframe attributes: width=600, height=450, allowfullscreen, loading=lazy, referrerpolicy=no-referrer-when-downgrade.
* Two-Column Contact Layout (.container.py-5 > .row):
* Left Column (col-md-6): Contact information with three info-box elements — Address (60-49 Road 11378 New York), Phone (+65 11.188.888), Email (hellocolorlib@gmail.com). Each info-box uses a Font Awesome icon (fa-map-pin, fa-phone, fa-envelope) paired with a label and value.
* Right Column (col-md-6): 'Leave A Comment' form with Name input (type=text), Email input (type=email), Message textarea (5 rows), and a Send Message submit button styled in teal (#00797a). Note: The form has no action attribute and no JavaScript handler — submissions are not processed.

*Note: There is a discrepancy between the footer contact details (ITI - Knowledge City - Egypt, +971 (58) 135-9871) and the Contact Us page body details (New York address, +65 phone). The footer uses the real project address while the body uses placeholder lorem ipsum-style content.*

### 9.1.2 External Dependencies

* Google Maps Embed API (no API key required for basic embed iframes)
* External stylesheet: ContactUsStyle.css (custom styling for .map-section, .map-wrapper, .info-box, .info-icon, .send-btn)
* Scripts: indexScript.js (for the search functionality), Swiper.js, Bootstrap JS Bundle

**Suggested Screenshot Location — ContactUs.html:**

Take a full-page screenshot of ContactUs.html in your browser showing the map embed, contact info section, and the comment form side by side.



## 9.2 FAQ.html — Frequently Asked Questions Page

The FAQ page presents common customer questions using a Bootstrap 5 accordion component. It uses a two-column layout with an illustrative image on the left and the accordion on the right.

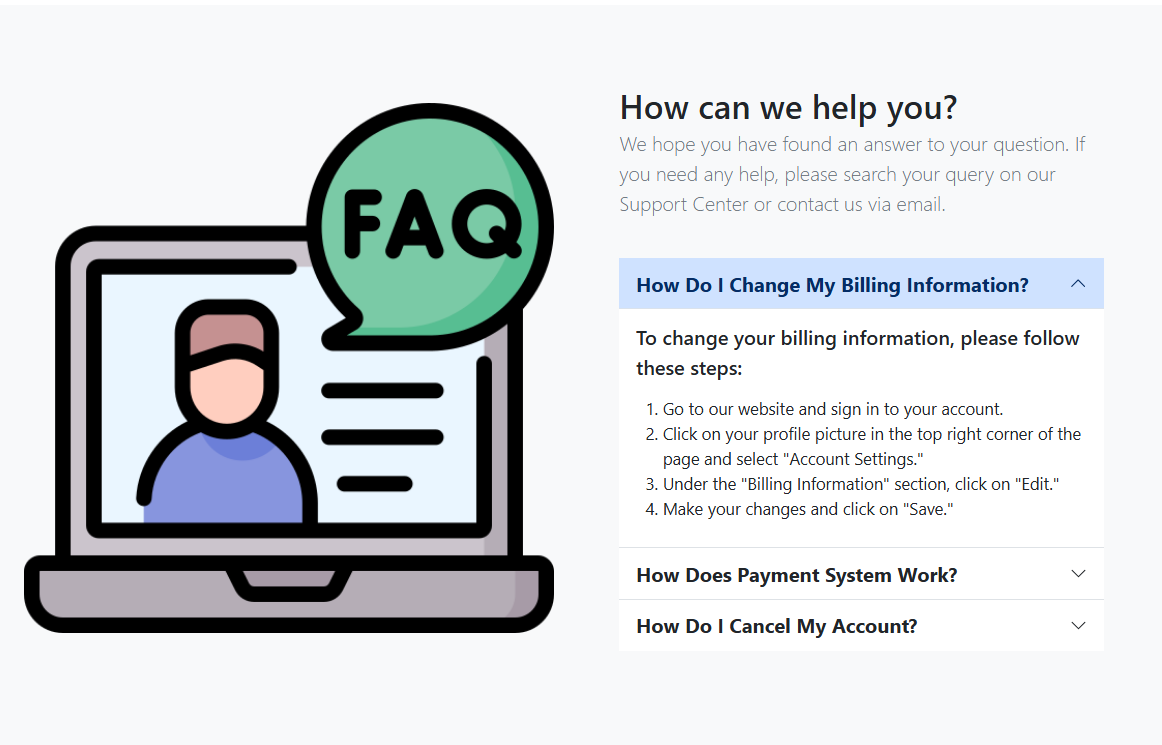
### 9.2.1 HTML Structure

* Top Header: Includes the #nameOfLoggedIn anchor element (links to Register.html, updated dynamically with the logged-in user's name).
* Main Section (.bg-light.py-3.py-md-5.min-vh-100): Uses min-vh-100 to ensure the section fills at least the full viewport height.
* Two-Column Layout (Bootstrap .row): Left column (col-12 col-lg-6) holds the FAQ image (./images/faq.png). Right column (col-12 col-lg-6) holds the text and accordion.
* Heading Text: 'How can we help you?' with a lead paragraph directing users to the Support Center or contact email.

### 9.2.2 Bootstrap Accordion Component

The FAQ accordion (#accordionExample) uses data-bs-parent='#accordionExample' to enforce that only one item can be open at a time. It contains three questions:

showing the page with the first accordion item expanded (default state), and one showing a different item expanded to demonstrate the toggle behavior.



### 9.2.3 Accordion Implementation Detail

|  |
| --- |
| <!-- Accordion structure — data-bs-parent enforces single-open behavior -->  <div class='accordion accordion-flush' id='accordionExample'>  <div class='accordion-item'>  <h2 class='accordion-header fw-bold' id='headingOne'>  <button class='accordion-button fw-bold fs-4'  data-bs-toggle='collapse'  data-bs-target='#collapseOne'  aria-expanded='true'>  How Do I Change My Billing Information?  </button>  </h2>  <div id='collapseOne' class='accordion-collapse collapse show'  data-bs-parent='#accordionExample'>  <div class='accordion-body'>  <ol class='fs-5'>  <li>Go to our website and sign in to your account.</li>  <li>Click on your profile picture ... select 'Account Settings.'</li>  <li>Under 'Billing Information', click 'Edit.'</li>  <li>Make your changes and click 'Save.'</li>  </ol>  </div>  </div>  </div>  <!-- Additional accordion items follow same pattern -->  </div> |

### 9.2.4 External Dependencies

* Bootstrap 5.3.3 CSS from unpkg.com CDN (different CDN source than other pages which use jsdelivr.net)
* Additional CSS: https://unpkg.com/bs-brain@2.0.4/components/faqs/faq-2/assets/css/faq-2.css — a third-party FAQ component style from the bs-brain library
* indexStyle.css for shared nav/header/footer styles
* Font Awesome 6.5.1 for icons
* Bootstrap JS Bundle from unpkg.com for accordion interactivity

## 9.3 TermsAndCond.html — Terms and Conditions Page

The Terms and Conditions page presents the website's legal usage policies in a clean two-column layout. It is a static informational page with no JavaScript logic beyond Bootstrap and shared nav scripts.

### 9.3.1 HTML Structure

* Top Header: Includes the #nameOfLoggedIn anchor element with href to Register.html.
* Main Section (.bg-light.py-5.py-md-5): Two-column Bootstrap layout with a terms-related image on the left and policy text on the right.
* Left Column (col-12 col-lg-6): Displays the image ./images/terms2.png as a full-width fluid image.
* Right Column (col-12 col-lg-6): Contains the full terms text divided into 5 numbered sections.

### 9.3.2 Terms Content Sections

|  |  |  |
| --- | --- | --- |
| **Section #** | **Title** | **Summary** |
| 1 | Acceptance of Terms | By accessing the site, the user confirms they accept and will comply with these Terms and Conditions. |
| 2 | User Responsibilities | Users must not misuse services, attempt unauthorized access, or engage in harmful activities. |
| 3 | Privacy Policy | The site respects user privacy and refers to the Privacy Policy for data collection details. |
| 4 | Changes to Terms | Terms may be updated at any time. Continued use of the site constitutes acceptance of revised terms. |
| 5 | Contact Us | Users with questions can contact the team via email or the project address at ITI - Knowledge City - Egypt. |

### 9.3.3 Contact Information in Terms Page

The Terms page uniquely exposes the developer email addresses directly in the HTML:

|  |
| --- |
| <!-- Developer contact emails listed in Section 5 of Terms page -->  <p>Abdelrahman.magdy.iti@gmail.com</p>  <p>Fatmaelzahraaahmed98@gmail.com</p>  <p>Address: ITI - Knowledge City - Egypt</p> |

### 9.3.4 Navigation & Linking

* This page is linked from the footer of every other page in the project under the 'Address' column as 'Terms & Conditions'.
* It is also linked from the Register.html page — the registration form's agreement checkbox reads: 'I agree to the terms and conditions' with a hyperlink to TermsAndCond.html.
* There is no JavaScript logic in this page — it is fully static HTML with Bootstrap for styling.

### 9.3.5 External Dependencies

* Bootstrap 5.3.3 CSS from unpkg.com
* Bootstrap 5.3.2 CSS from jsdelivr.net (both are loaded — redundant inclusion)
* indexStyle.css for shared header/footer styles
* Font Awesome 6.5.1 for icons
* Swiper.js CSS (loaded but not used on this page — included as part of the shared head template)

**Suggested Screenshot Location — TermsAndCond.html:**

Capture the full Terms page showing both columns — the image on the left and all 5 policy sections visible on the right.

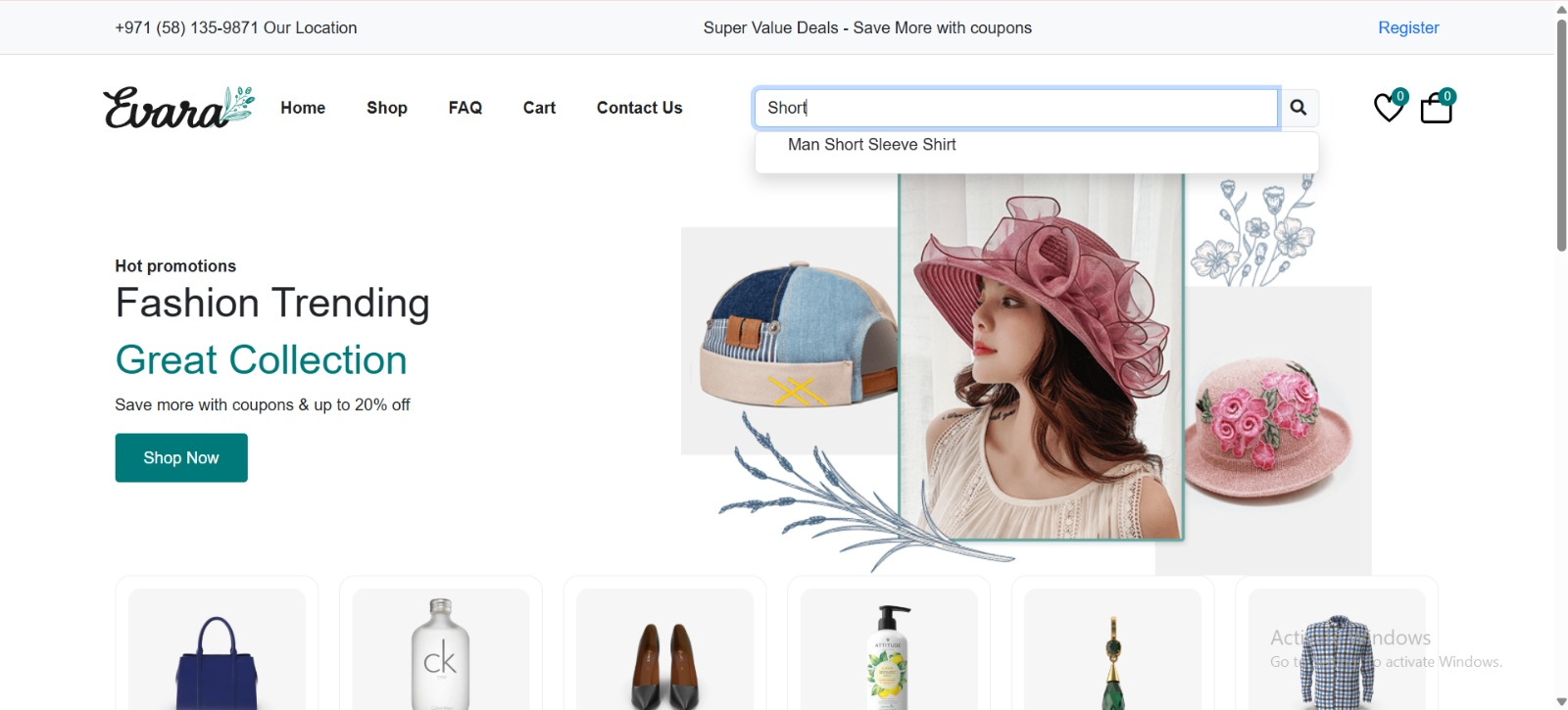


# 10. Website Screenshot Gallery

This section is reserved for visual documentation of the Evara website. Add screenshots of each page in the designated areas below. Screenshots should be taken in a desktop browser at a standard resolution (1280px or 1440px width) for consistency.

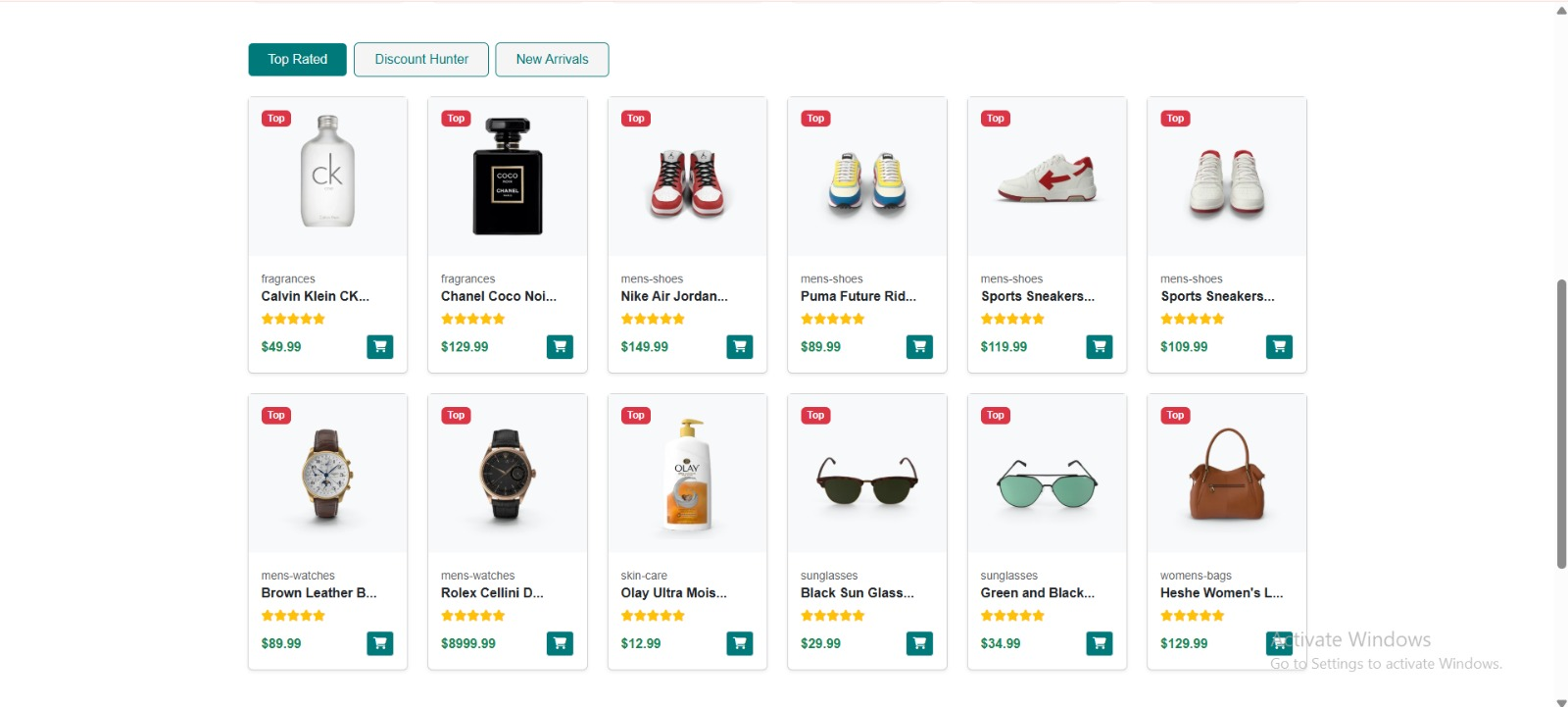
Recommended tool: Use the Snipping Tool (Windows) or Cmd+Shift+4 (Mac) for partial captures, or browser DevTools (F12 > Toggle Device Toolbar) for responsive screenshots.

## 10.1 Homepage (index.html)

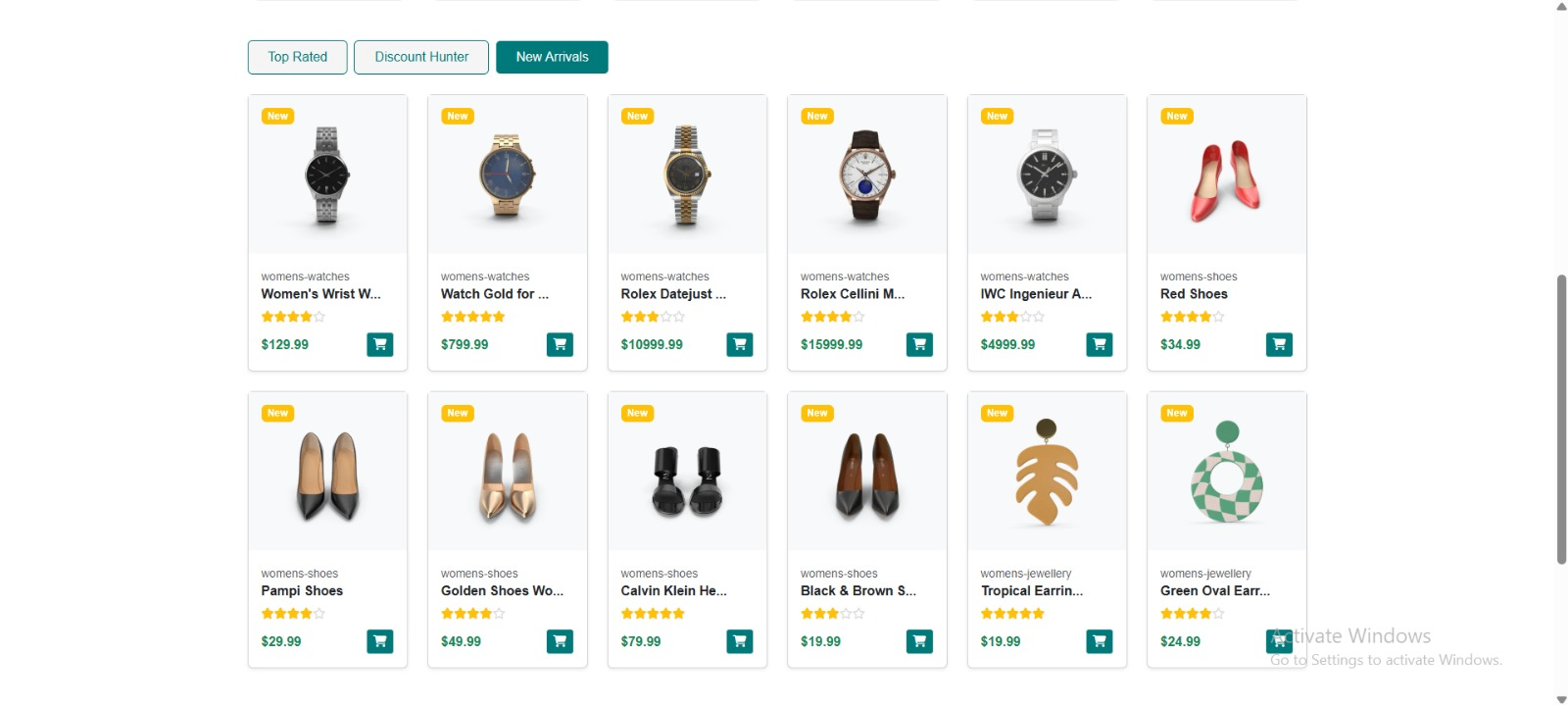


*Capture:Hero Section*

*Capture: Featured Products section with tab buttons (Top Rated / Discount Hunter / New Arrivals)*



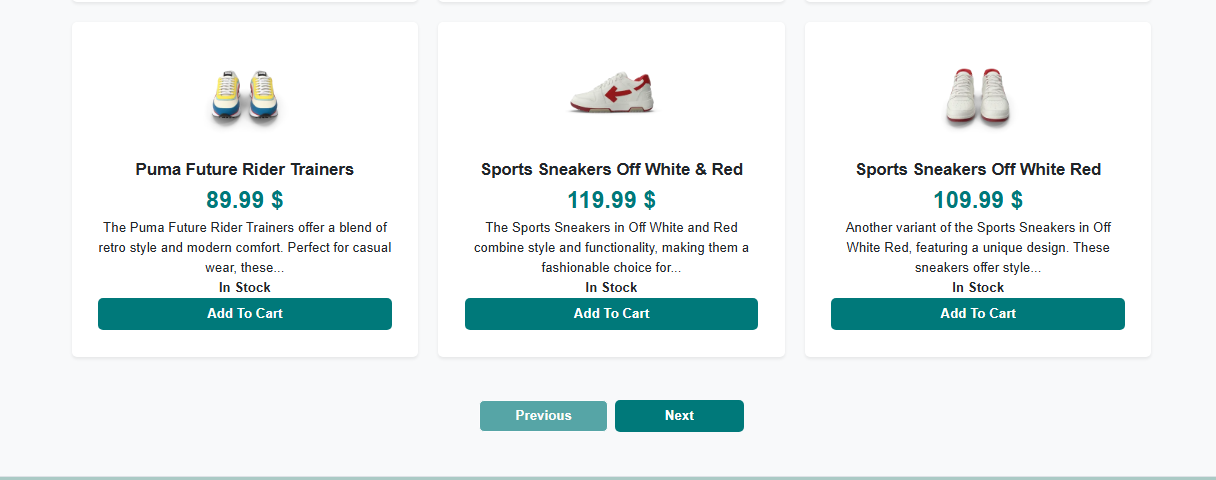




## 10.2 Products Page (Products.html)

*Capture: Product grid with cards showing images, titles, prices, ratings, and Add to Cart buttons*





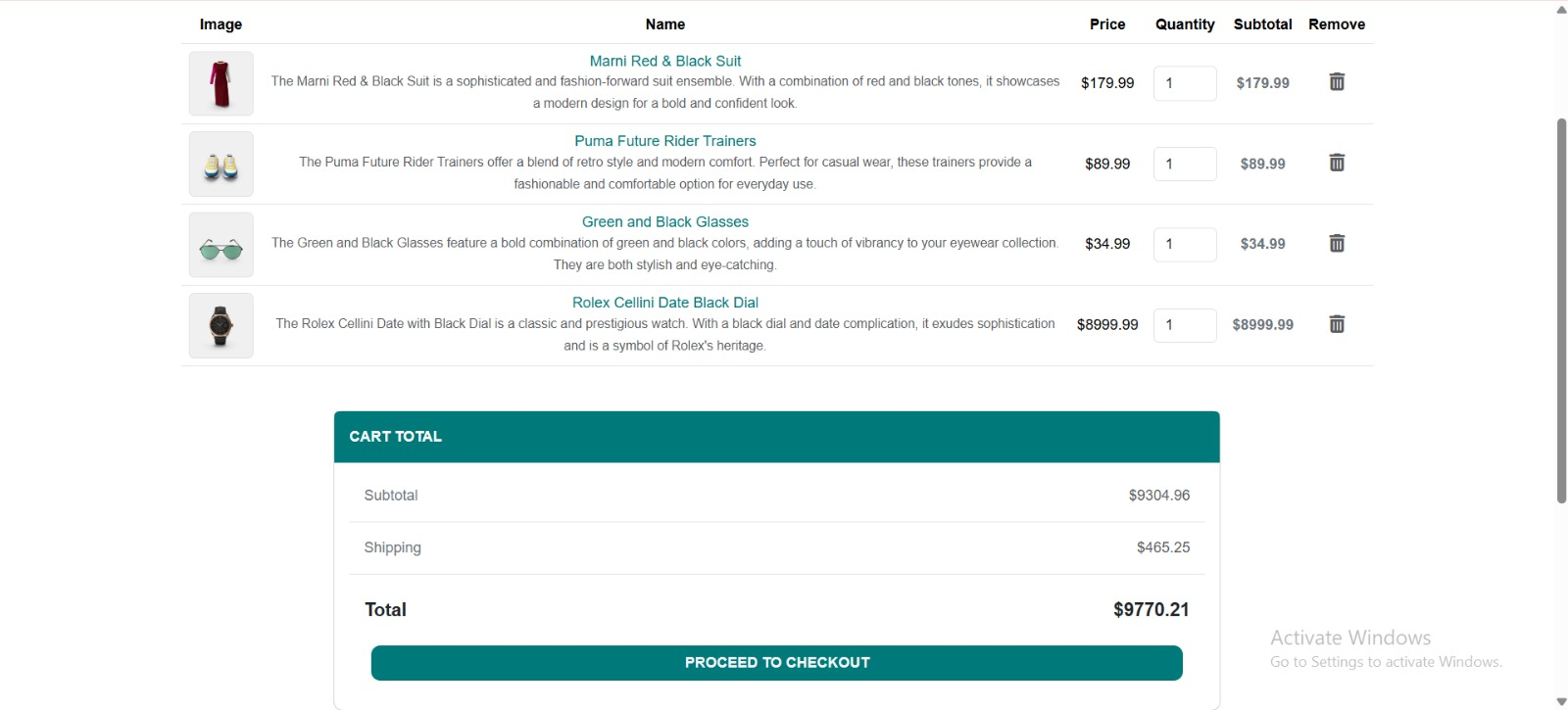
## 10.3 Product Details Page (ProductDetails.html)

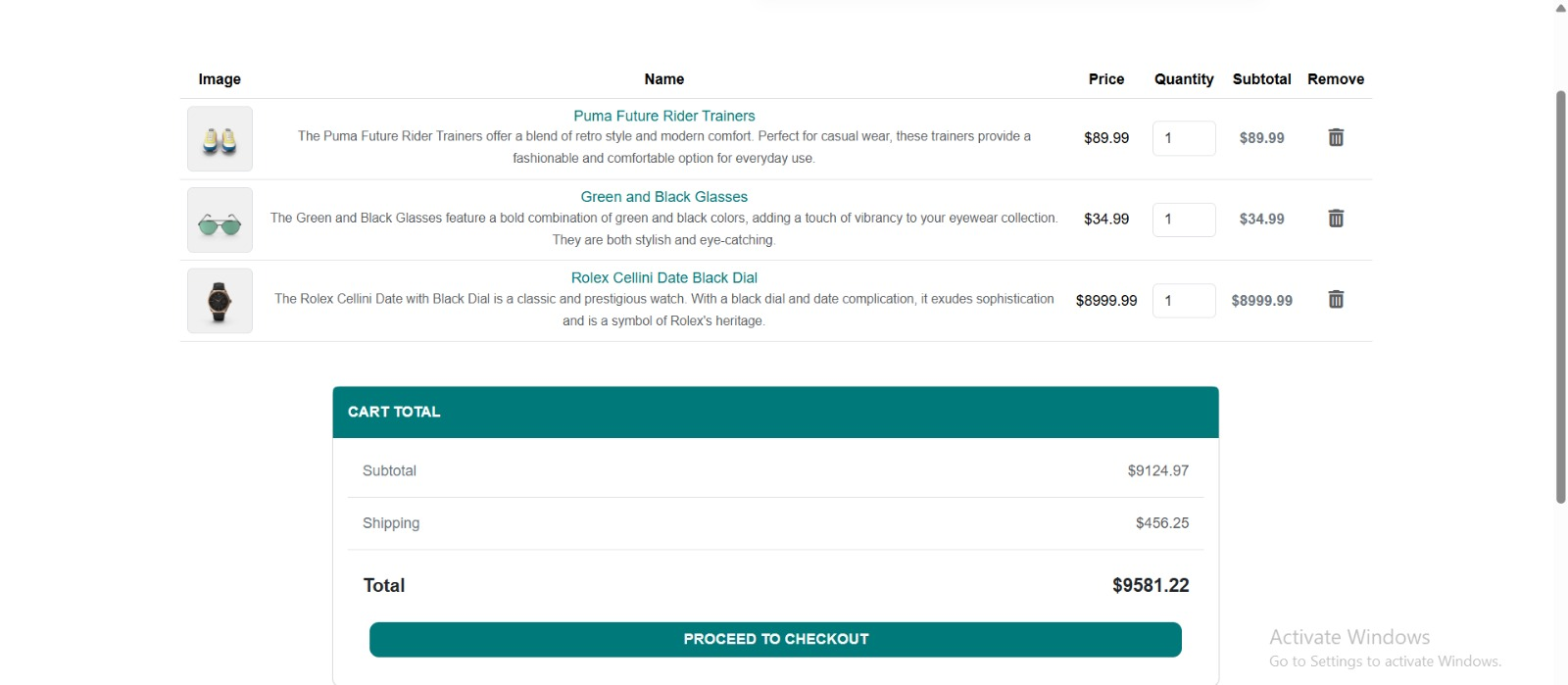
## 

*Capture: Full product detail view — image, title, price, discount badge, description, size buttons, quantity input, and Add to Cart button*

## 10.4 Cart Page (Cart.html)

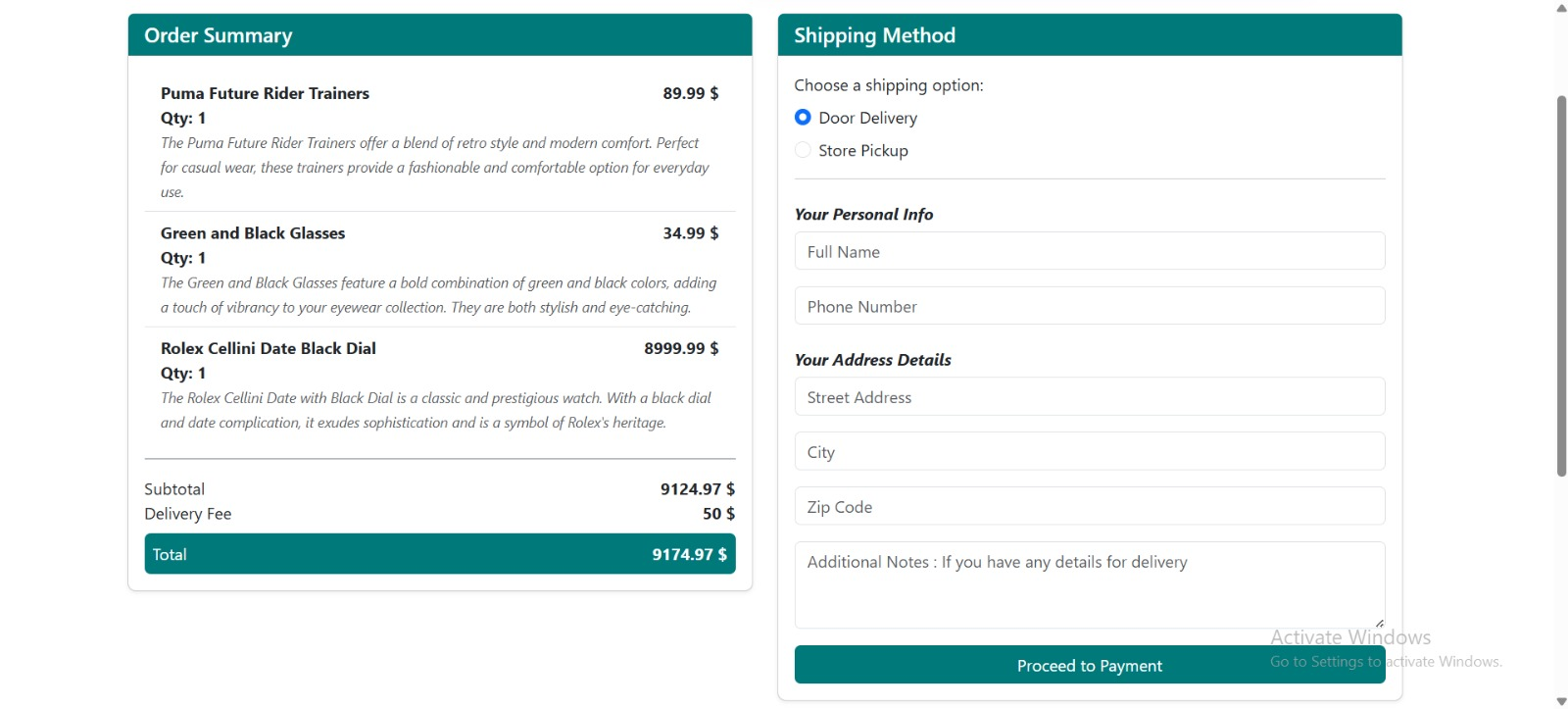
*Capture: Cart table with at least 2 items, showing subtotal, $50 shipping, and grand total. Include the Proceed to Checkout button with button to remove items or increase quantity.*





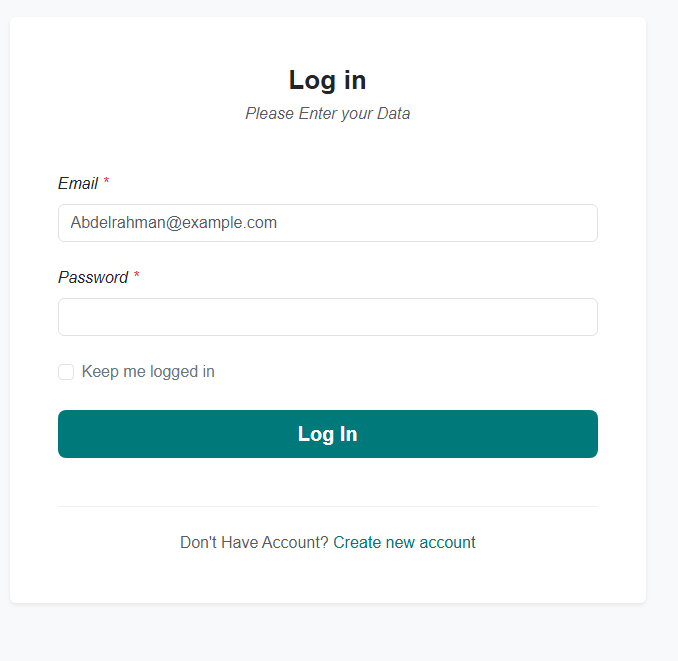
## 10.5 Checkout Page (CheckOut.html)

*Capture: Both columns visible — Order Summary on the left, Shipping form (with Door Delivery selected) on the right*



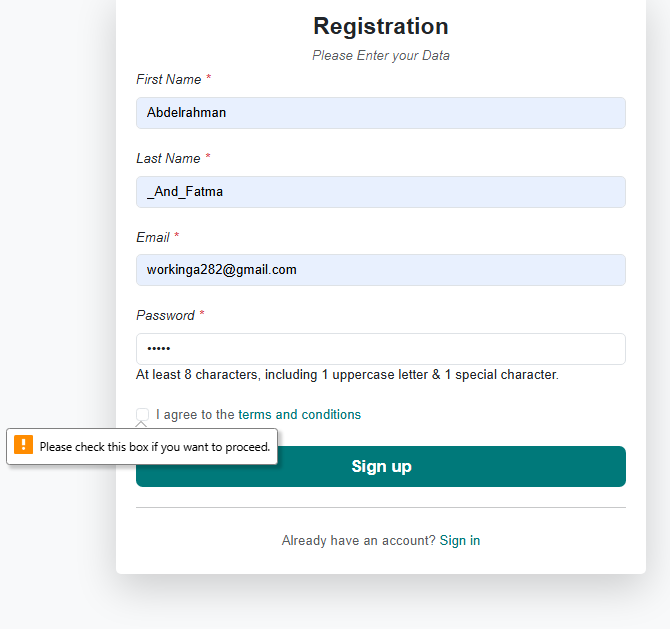
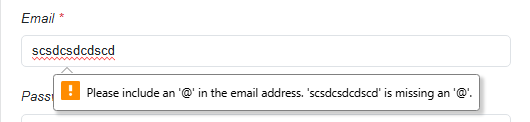
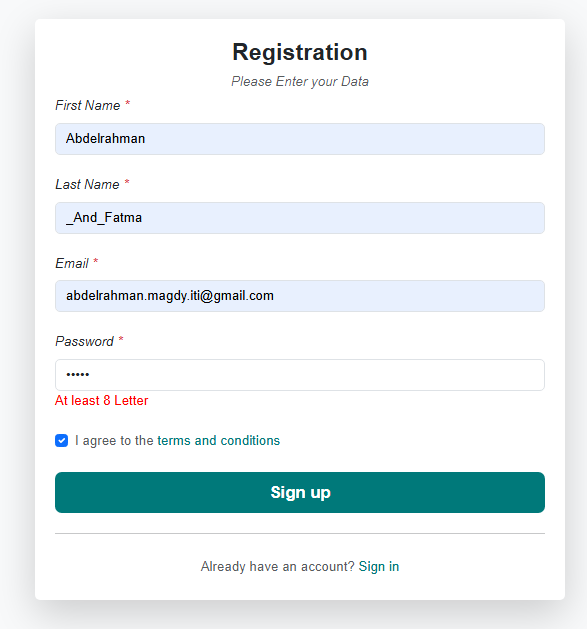
## 10.6 Login Page (RealLogin.html)

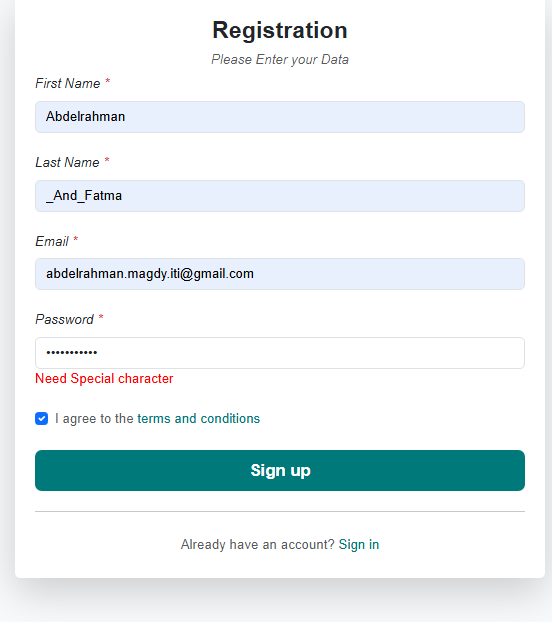
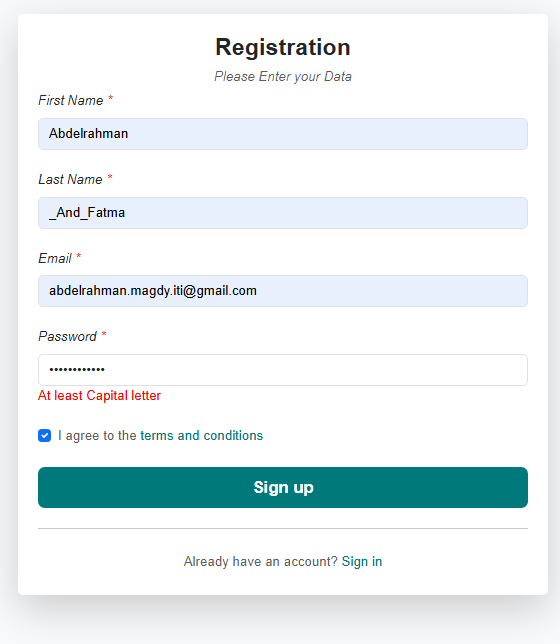
*Capture: Login card with email and password fields, Log In button, and 'Don't Have Account?' link*



## 10.7 Registration Page (Register.html)

*Capture: Registration card showing all fields (First Name, Last Name, Email, Password with hint, Terms checkbox) and the Sign Up button with validation to all fields Password, Emal and restrict signing up without filling all inputs*



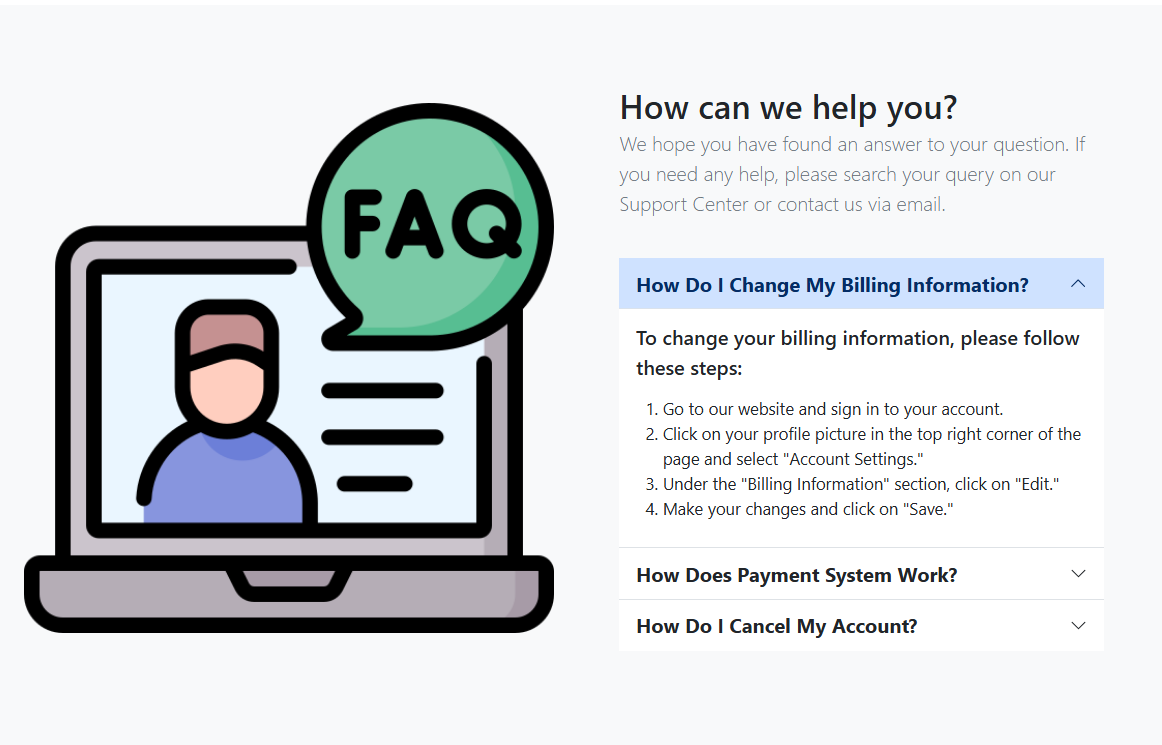
## 10.8 Contact Us Page (ContactUs.html)

*Capture: Map embed + two-column contact section (info boxes on left, message form on right)*



## 10.9 FAQ Page (FAQ.html)

*Capture: FAQ page showing image on left and accordion with first question expanded on right*



## 10.10 Terms & Conditions Page (TermsAndCond.html)

*Capture: Full Terms page showing the terms image on the left and all 5 policy sections on the right*



## 10.11 Mobile Responsive Views

Take at least 2 mobile screenshots to demonstrate responsive design. Use Chrome DevTools (F12) and switch to a mobile viewport (e.g., iPhone 12 Pro at 390px width).

