# **AYAAM – Project Documentation**

## **1. Introduction :**

## AYAAM is a Single Page Application (SPA) E-commerce platform designed to serve three main roles: Customer, Seller, and Admin. It provides a smooth browsing and shopping experience using JavaScript modules, hash-based routing, and lazy-loaded pages. The project is built with plain JavaScript, Bootstrap, CSS, and FontAwesome without external frameworks or APIs. All data is handled using LocalStorage and SessionStorage for persistence.

*GitHub Repository* : [Githup](https://github.com/YasserFat7alah/e-commerce-iti)

*Live Demo* : [AYAAM.com](https://e-commerce-iti-delta.vercel.app/#/home)

## **2. Features :**

### **2.1 Customer**

Sign up and login

Browse products by category, brand, filters

Add products to cart

Checkout flow with order confirmation

View and manage profile information

### **2.2 Seller**

Add, edit, and delete products

Manage product stock, discounts, and offers

View sales dashboard with performance insights

### **2.3 Admin**

Manage users (customers & sellers)

Manage all products

Monitor system reports

Role-based access control

## **3. Technical Architecture**

### **3.1 SPA and Routing**

* Hash-based routing (#/home, #/catalog, #/product)
* Router module handles navigation and lazy loads the required page modules.
* Role-based protection:
  + Customer-only routes (checkout, profile, cart)
  + Seller-only routes (seller dashboard)
  + Admin-only routes (admin dashboard)
* Fallback to /404 page for unknown routes.

### **3.2 Data Management**

* **LocalStorage**: Products, Users, Orders (seeded on first load)
* **SessionStorage**: User sessions, redirects, and temporary states
* No backend / API integration (mock data only)

### **3.3 Shared Components**

* components/ → reusable UI building blocks
  + Cart
  + Landing
  + Dashboard elements
  + Layout
  + UI
* scripts/utils/ → helper functions
  + Loader utility
  + Navigation
  + Storage abstraction
  + Data seeding

**3.4 Security & Role Management :**  
 role-based access for Admin, Seller, Customer .

**3.5 Technology Stack**  
 Plain JS, Bootstrap, CSS, Font Awesome, LocalStorage/SessionStorage .

## **4. Project Structure :**

**E-COMMERCE-ITI**

**│── assets/ # CSS, images, JS assets**

**│── components/ # Reusable components (auth, cart, profile, etc.)**

**│── data/ # Local JSON/data & seeders**

**│── pages/ # SPA pages (Home, Catalog, Product, Cart, Admin, Seller, etc.)**

**│── scripts/ # Core scripts (router, utils, cartScripts)**

**│── theme-files/ # Extra theme/config files**

**│── index.html # Entry point**

**│── app.js # App initialization & routing**

**│── README.md**

### **5. Data Model :** The project uses object-oriented modules (User, Product, Order) to structure and enforce data consistency. These classes wrap the raw JSON objects and add validation, formatting, and serialization (toJSON()) for safe storage in LocalStorage / SessionStorage.

**4.1 User Model (User Class)** :  
 Represents system users with role-based access control



*Notes*

* Enforces roles (master, admin, seller, user).
* Defaults to "user" role.
* toJSON() makes the object serializable for LocalStorage.
* Plain-text passwords only for prototype/demo purposes.

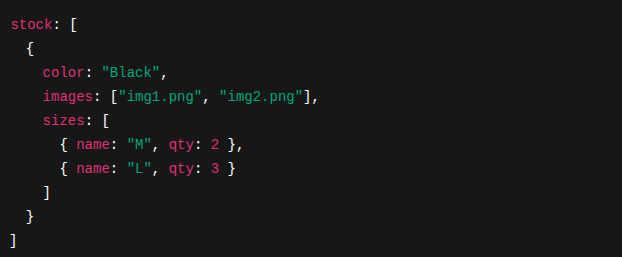
### **4.2 Product Model (Product Class) :**

Represents catalog products, including categories, variants, and inventory.



Notes

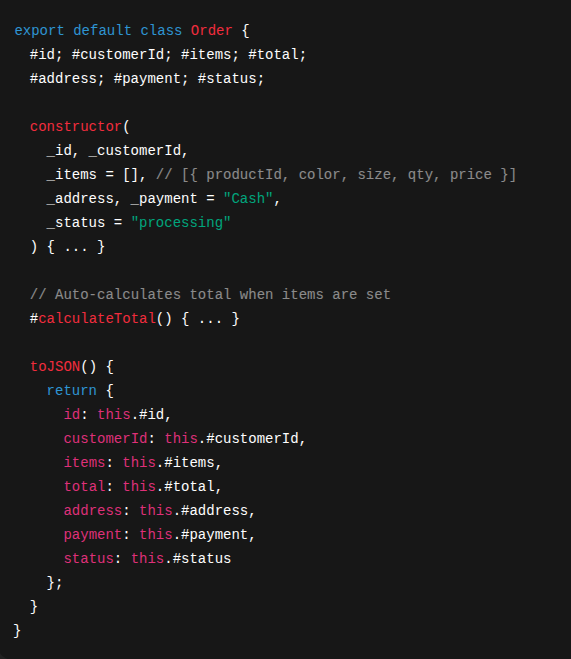
* Enforces strict validation on setters (e.g., Sale between 0 and 1).
* Normalizes capitalization for Name, Category, Subcategory, and Brand.
* Stock structure:



* isApproved() is used to filter products in the catalog.

### **4.3 Order Model (Order Class)**

Encapsulates customer orders with private fields and auto-calculated totals.



Notes

* Total is always consistent, recalculated automatically.
* Payment defaults to "Cash".
* Status workflow: "processing" → "shipped" → "completed" | "canceled".

### **4.4 Relationships**

* **User → Product**
  + Sellers (role: seller) own multiple products via Product.SellerId.
* **User → Order**
  + Customers (role: user) create orders via Order.CustomerId.
* **Product → Order**
  + Orders reference products by productId inside Order.Items[].

**6. Pages & Views (SPA Routes) :**

AYAAM is built as a Single Page Application (SPA) using a custom Router that listens to hashchange and DOMContentLoaded events.

Each route corresponds to a page module dynamically imported (lazy-loaded) only when needed. This optimizes performance and reduces initial load size.

### **5.1 Public Routes**

|  |  |  |  |
| --- | --- | --- | --- |
| **Route** | **View Module** | **Access** | **Description** |
| /home | HomePage.js | Public | Landing page with featured products and categories. |
| /about | AboutPage.js | Public | Information about the project/brand. |
| /catalog | CatalogPage.js | Public | Displays product list with filters (brand, category, price, etc.). |
| /product | Product.js | Public | Single product details page with images, sizes, colors, and add-to-cart. |
| /info | SupportPage.js | Public | Customer support, FAQs, or static info page. |
| /404 | NotFound.js | Public | Fallback page for invalid routes. |

### **5.2 Authentication Routes**

|  |  |  |  |
| --- | --- | --- | --- |
| **Route** | **View Module** | **Access** | **Description** |
| /login | LoginPage.js | Public | User login form, authenticates and stores session in SessionStorage. |
| /signup | SignupPage.js | Public | Registration form for new customers/sellers. |

### **5.3 Customer Routes**

|  |  |  |  |
| --- | --- | --- | --- |
| **Route** | **View Module** | **Access** | **Description** |
| /cart | CartPage.js | Customer only | Shopping cart, manages items before checkout. |
| /checkout | CheckOutPage.js | Customer only | Checkout flow with address, payment, and confirmation. |
| /profile | Profile.js | Customer only | User profile: personal info, past orders, settings. |

### **5.4 Seller Routes**

|  |  |  |  |
| --- | --- | --- | --- |
| **Route** | **View Module** | **Access** | **Description** |
| /seller | SellerDashboard.js | Seller only | Dashboard for sellers: add/edit/delete products, view sales. |

### **5.5 Admin Routes**

|  |  |  |  |
| --- | --- | --- | --- |
| **Route** | **View Module** | **Access** | **Description** |
| /admin | AdminDashboard.js | Admin/Master | Manage users, approve/reject products,  Handle order status, oversee reports. |

### **5.6 Routing Logic**

* Routes are matched via **longest prefix match** in Router.resolve().
* Restricted routes check the current user role before rendering.
  + If unauthorized → redirect to /404.
  + If unauthenticated → redirect to /login.
* Dynamic imports ensure **lazy-loading**:



## **7. User Flows**

* **Customer flow**: Signup → Browse → Add to cart → Checkout → Profile
* **Seller flow**: Login → Add products → Manage stock/offers → View dashboard
* **Admin flow**: Login → Manage users → Manage products → Reports

## **8. Conclusion**

AYAAM demonstrates modular **SPA architecture** for an e-commerce platform using only **Vanilla JS** with Bootstrap styling.  
 It provides **role-based features** for customers, sellers, and admins, showcasing real-world flows in a simplified educational project.