Instant Order eCommerce System Requirements

Langelihle Lucky Mhlongo

2024

Table of Contents

[Overview 3](#_Toc168327447)

[User stories 4](#_Toc168327448)

[Use Case Diagram 5](#_Toc168327449)

[Entity Relationship Diagram 6](#_Toc168327450)

[Class Diagram 7](#_Toc168327451)

[Product Backlog 8](#_Toc168327452)

[Sprint backlog - Week 5 .NET Angular 8](#_Toc168327453)

[Sprint backlog - Week 4 .NET ASP.NET/Blazor 12](#_Toc168327454)

[Sprint backlog - Week 3 .NET Web App 12](#_Toc168327455)

[Sprint backlog - Week 2 .NET REST API 12](#_Toc168327456)

[Sprint backlog – Week 1 .NET Data Access 12](#_Toc168327457)

[Sprint backlog - Week 3. OOD 3 - TDD Solo Project 13](#_Toc168327458)

[Sprint backlog - Week 2. OOD 2 - Code-based Solo Project 13](#_Toc168327459)

[Sprint backlog - Week 1. OOD 1 - Research Project 14](#_Toc168327460)

[Technical Architecture 14](#_Toc168327461)

[What are Microservices? 15](#_Toc168327462)

[What is HTTP? 15](#_Toc168327463)

[User Experience (UX) 15](#_Toc168327464)

[Interaction Design Laws 16](#_Toc168327465)

[Site Map 16](#_Toc168327466)

[Patterns of Visualisation 16](#_Toc168327467)

[UI Flow 17](#_Toc168327468)

[Wireframe 18](#_Toc168327469)

[Sample data for User Registration 26](#_Toc168327470)

[Sample data for User Login 27](#_Toc168327471)

[System logging (log file) and design 27](#_Toc168327472)

[List of REST API endpoints with description 27](#_Toc168327473)

[Source Control 33](#_Toc168327474)

[Directory and naming services 33](#_Toc168327475)

[Assumptions and Dependencies 33](#_Toc168327476)

[Lessons Learned 34](#_Toc168327477)

[Lesson Learned - Week 5 .NET with Angular 34](#_Toc168327478)

[Lesson Learned - Week 4 .NET ASP.NET/Blazor 34](#_Toc168327479)

[Lesson Learned - Week 3 .NET Web App 35](#_Toc168327480)

[Lesson Learned - Week 2 .NET REST API 35](#_Toc168327481)

[Lesson Learned - Week 1 .NET Data Access 35](#_Toc168327482)

[Lesson Learned - Week 3. OOD 3 - TDD Solo Project 35](#_Toc168327483)

[Lesson Learned - Week 2. OOD 2 - Code-based Solo Project 36](#_Toc168327484)

[Lesson Learned - Week 1. OOD 1 - Research Project 36](#_Toc168327485)

# Overview

Instant Order is an e-commerce system that offers customers quality clothes at low cost. Customers can navigate through various categories in the system to view different products catered to by Instant Order. Admins can manage products and customer orders.

The system will have a clear navigation menu for easy browsing through product categories. The system will have a fast and accurate search functionality, allowing customers to find products quickly. Moreover, the system will have filters and sorting options to refine search results based on price, brand, or category criteria.

The system will have well-organised product listings displaying key information such as product name, price, and availability. The system will include product ratings and reviews to help users make informed purchase decisions. Customers will be able to add items to their cart without navigating away from the product page, and they can add products to their Wishlist.

The system will integrate user accounts to store payment and shipping details for seamless checkout. Moreover, it will consist of a streamlined checkout process with minimal steps to complete an order. Customers and Admins will be able to view orders and manage shipping addresses from their account dashboard.

# User stories

This section represents the user stories involved in the project.

As a customer

I want to be able to search for a product using category, brand, and price

So I can find the product I am looking for more quickly

As a customer

I want to be able to view products

So I can add them to my cart

As an admin

I want to be able to manage products

So I can allow customers to purchase them

As a customer

I want to be able to add products to my Wishlist

So I can view them later

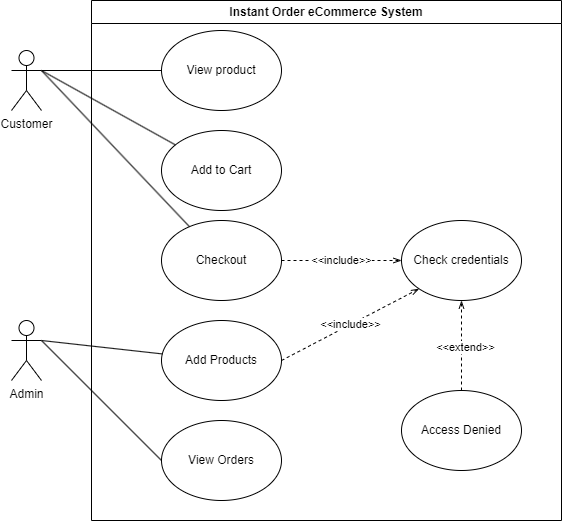
As a customer

I want to be able to checkout

So I can purchase the products in my cart

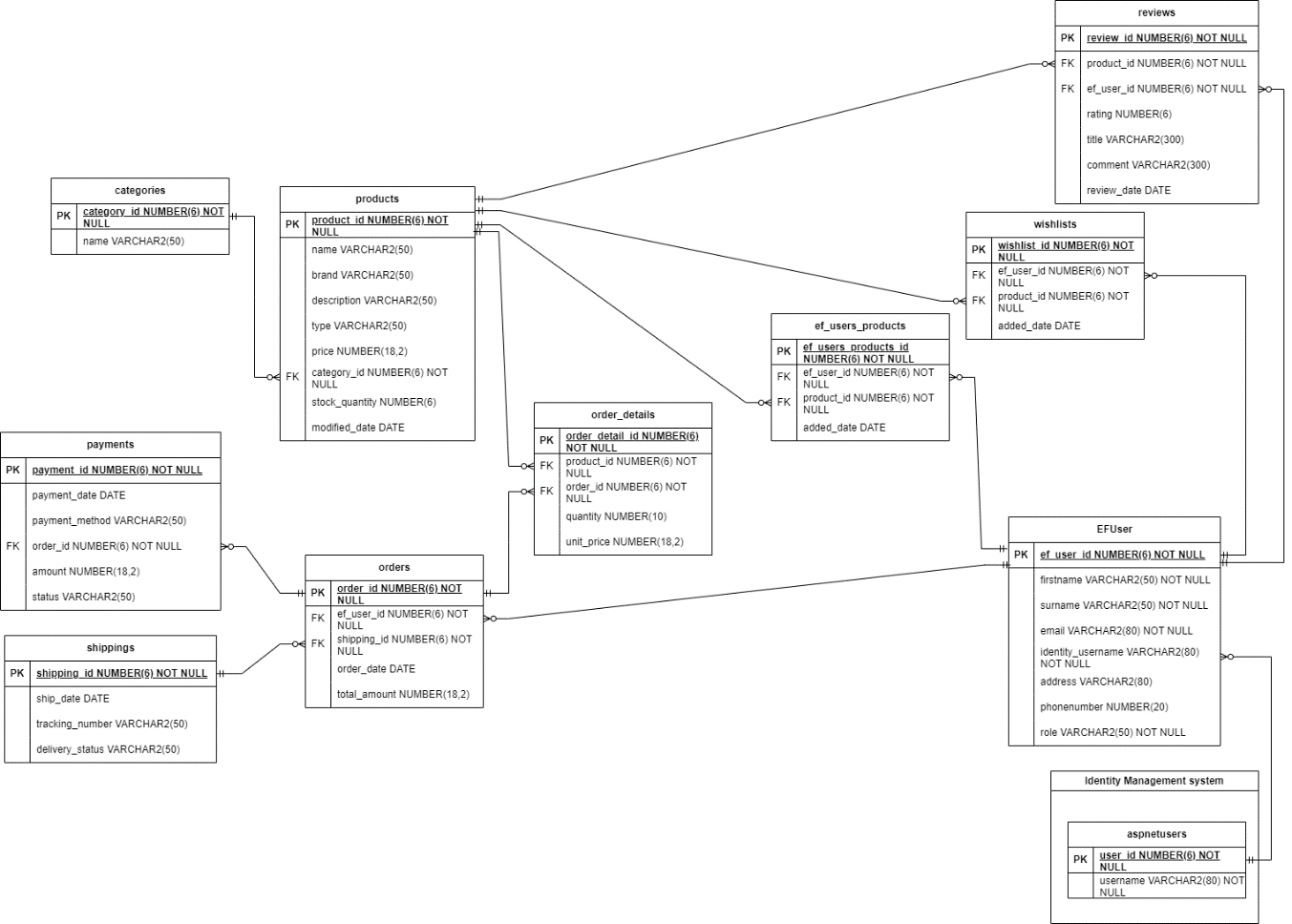
# Use Case Diagram

This section represents the use case diagram that describes the project’s use cases.



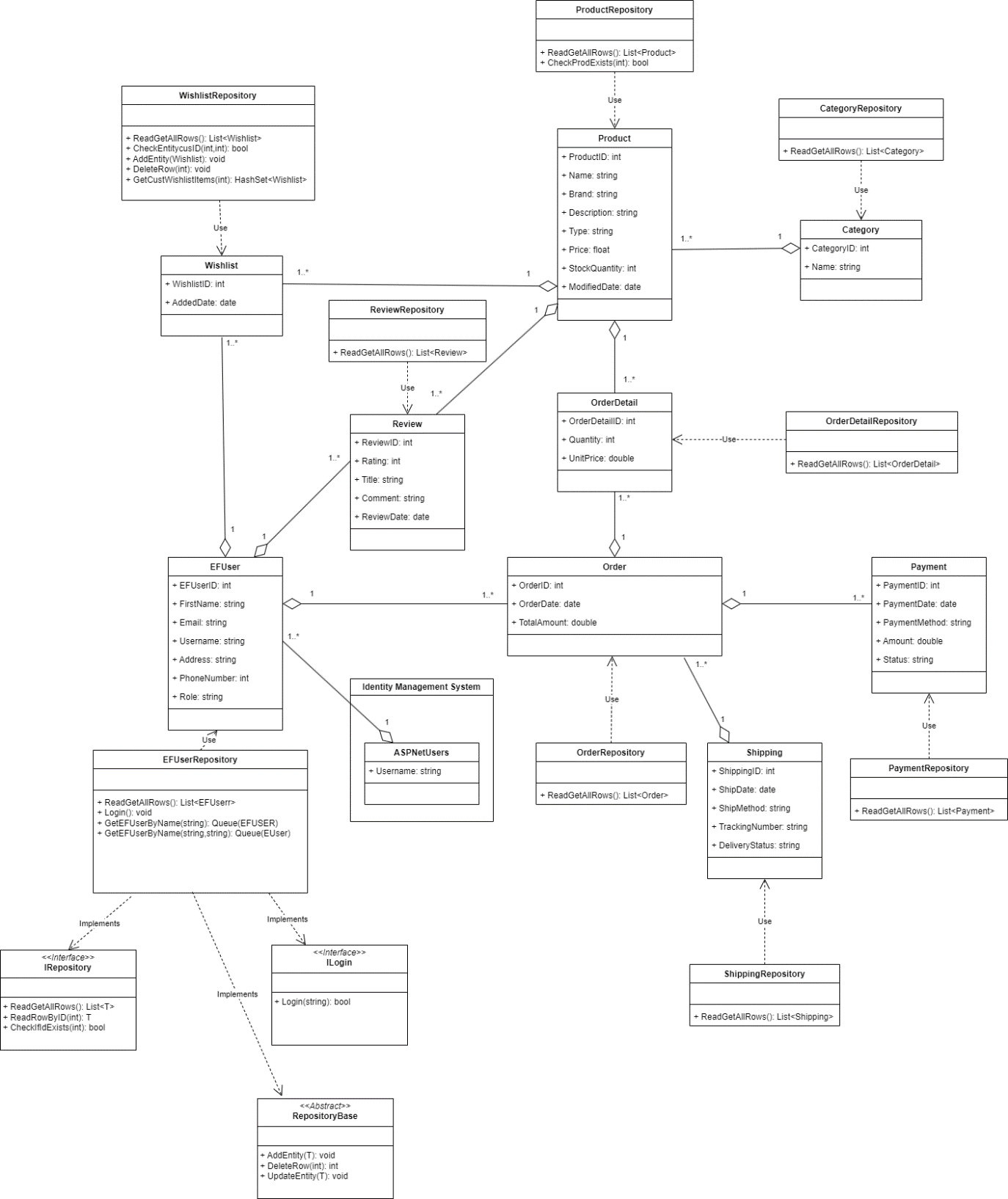
# Entity Relationship Diagram

This section represents the entity relationship diagram comprising all the entities involved in the project.



# Class Diagram

This section represents the class diagram comprising all the classes involved in the project.



# Product Backlog

This section represents the product backlog, which consists of the users, requirements, priority, and completion status.

The **Product Goal/objective** is to create full stack application using C# .NET as shown below:

Add the following Product Backlog to your document, update the status accordingly:



## Sprint backlog - Week 5 .NET Angular

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Product Backlog | | | | | |
| ID | Role | Auth | Component name | Component selector | Description |
| 1 | - | No | DefaultHomeComponent | app-default-home | Landing page |
| 2 | - | No | DefaultFooterComponent | app-default-footer | footer for default pages |
| 3 | Administator and seller | Yes | SpecificUserFooterComponent | app-specific-user-footer | footer for either admin or seller pages |
| 4 | All | Yes & No | MainNavigationComponent | app-main-navigation | Menu header for all user, it changes to a specific user when logged in |
| 5 | All | No | ShopComponent | app-shop | Shop page |
| 6 | All | No | MensComponent | app-mens | Mens products page |
| 7 | All | No | WomensComponent | app-womens | Women products page |
| 8 | All | No | CartComponent | app-cart | Cart page |
| 9 | All | No | ProductDetailsComponent | app-product | Specific product page |
| 10 | Customer | Yes | CheckoutComponent | app-checkout | Admin footer |
| 11 | Customer | Yes | OrderSuccessComponent | app-order-success | Order Success page |
| 12 | Customer | Yes | WishlistComponent | app-wishlist | Wishlist page |
| 13 | All | No | FaqComponent | app-faq | Frequently asked questions page |
| 14 | All | No | AboutUsComponent | app-about-us | About us page |
| 15 |  |  | ContactUsComponent |  |  |
| 16 | All | No | PrivacyInfoComponent | app-privacy-info | Privacy information page |
| 17 | All | Yes | UserLoginComponent | app-user-login | User Login page |
| 18 | All | Yes | UserRegistrationComponent | app-user-registration | User Registration page |
| 19 | All | Yes | UserProfileComponent | app-user-profile | User profile page |
| 20 | Administrator | Yes | AdminsDashboardComponent | app-admins-dashboard | Administrator dashboard page |
| 21 | Administrator | Yes | AdminsCategoriesComponent | app-admins-categories | Administrator categories list page |
| 22 | Administrator | Yes | AdminsCategoriesCreateComponent | app-admins-categories-create | Administrator categories create page |
| 23 | Administrator | Yes | AdminsCategoriesUpdateComponent | app-admins-categories-update | Administrator categories update page |
| 24 | Administrator | Yes | AdminsCategoriesDeleteComponent | app-admins-categories-delete | Administrator categories delete page |
| 25 | Administrator | Yes | AdminsProductsComponent | app-admins-products | Administrator products page |
| 26 | Administrator | Yes | AdminsProductsCreateComponent | app-admins-products-create | Administrator products create page |
| 27 | Administrator | Yes | AdminsProductsUpdateComponent | app-admins-products-update | Administrator products update page |
| 28 | Administrator | Yes | AdminsProductsDeleteComponent | app-admins-products-delete | Administrator products delete page |
| 29 | Administrator | Yes | AdminsOrderHistoryComponent | app-admins-order-history | Administrator order history page |
| 30 | Administrator | Yes | AdminsOrderDetailsComponent | app-admins-order-details | Administrator order details page |
| 31 | Administrator | Yes | AdminsShippingComponent | app-admins-shipping | Administrator shipping page |
| 32 | Administrator | Yes | AdminsShippingPendingComponent | app-admins-shipping-pending | Admin shipping pending page |
| 33 | Administrator | Yes | AdminsShippingUpdateComponent | app-admins-shipping-update | Admin shipping update page |
| 34 | Administrator | Yes | AdminsReviewsComponent | app-admins-reviews | Administrator reviews page |
| 35 | Administrator | Yes | AdminsWishlistsComponent | app-admins-wishlists | Administrator wishlists page |
| 36 | Administrator | Yes | AdminsPaymentsComponent | app-admins-payments | Administrator payments page |
| 37 | Administrator | Yes | AdminsPaymentsPendingComponent | app-admins-payments-pending | Administrator payments pending page |
| 38 | Administrator | Yes | AdminsUsersComponent | app-admins-users | Administrator users page |
| 39 | Administrator | Yes | AdminsUsersCreateComponent | app-admins-users-create | Administrator users create page |
| 40 | Administrator | Yes | AdminsUsersUpdateComponent | app-admins-users-update | Administrator users update page |
| 41 | Administrator | Yes | AdminsUsersDeleteComponent | app-admins-users-delete | Administrator users delete page |
| 42 | Administrator | Yes | AdminsProfileComponent | app-admins-profile | Administrator profile page |
| 43 | Customer | Yes | CustomersDashboardComponent | app-customers-dashboard | Customer dashboard page |
| 44 | Customer | Yes | CustomersOrderHistoryComponent | app-customers-order-history | Customer order history page |
| 45 | Customer | Yes | CustomersOrderDetailsComponent | app-customers-order-details | Customer order details page |
| 46 | Customer | Yes | CustomersReviewsComponent | app-customers-reviews | Customer reviews page |
| 47 | Customer | Yes | CustomersReviewsCreateComponent | app-customers-reviews-create | Customer reviews create page |
| 48 | Customer | Yes | CustomersReviewsUpdateComponent | app-customers-reviews-delete | Customer reviews delete page |
| 49 | Customer | Yes | CustomersReviewsDeleteComponent | app-customers-reviews-update | Customer reviews update page |
| 50 | Customer | Yes | CustomersWishlistsComponent | app-customers-wishlists | Customer wishlists page |
| 51 | Customer | Yes | CustomersProfileComponent | app-customers-profile | Customer profile page |
| 52 | Seller | Yes | SellersDashboardComponent | app-sellers-dashboard | Seller dashboard page |
| 53 | Seller | Yes | SellersProductsComponent | app-sellers-products | Seller products page |
| 54 | Seller | Yes | SellersProductsCreateComponent | app-sellers-products-create | Seller products create page |
| 55 | Seller | Yes | SellersProductsUpdateComponent | app-sellers-products-update | Seller products update page |
| 56 | Seller | Yes | SellersProductsDeleteComponent | app-sellers-products-delete | Seller products delete page |
| 57 | Seller | Yes | SellersOrderHistoryComponent | app-sellers-order-history | Seller order history page |
| 58 | Seller | Yes | SellersOrderDetailsComponent | app-sellers-order-details | Seller order details page |
| 59 | Seller | Yes | SellersProfileComponent | app-sellers-profile | Seller profile page |

## Sprint backlog - Week 4 .NET ASP.NET/Blazor



## Sprint backlog - Week 3 .NET Web App



## Sprint backlog - Week 2 .NET REST API



## Sprint backlog – Week 1 .NET Data Access



## Sprint backlog - Week 3. OOD 3 - TDD Solo Project



## Sprint backlog - Week 2. OOD 2 - Code-based Solo Project



## Sprint backlog - Week 1. OOD 1 - Research Project



# Technical Architecture

The development environment for the system encompasses the tools, frameworks, and methodologies utilised throughout the software development lifecycle. This section outlines the primary components and configurations employed during development.

**Integrated Development Environment (IDE):**

Visual Studio 2022 - The primary IDE for developing, debugging, and deploying the system. Visual Studio provides a rich set of features, including code editing, version control integration, and project management tools.

**Programming Language and Framework:**

C# - The system is primarily developed using the C# programming language, leveraging its robust features, strong typing, and extensive .NET libraries.

.NET Framework 6.0 - The latest version of the .NET framework is utilised for building and running the system, offering enhanced performance, security, and language features.

**Development Methodology**

The development methodology adopted for the project is Agile, with a focus on iterative development, frequent releases, and continuous feedback loops. Specifically, the Scrum framework is utilised, employing sprint-based development cycles characterised by regular sprint planning, daily stand-ups, and sprint reviews.

In terms of development environment setup, the developer is provided with a high-performance workstation running Windows 11 or later. The workstation is equipped with ample RAM, CPU, and storage resources to support development tasks efficiently.

Visual Studio 2022 Community edition is the primary IDE utilised for development. It is configured with necessary components including the .NET Framework, ASP.NET, and Git integration. Additionally, the developer may integrate additional development tools and plugins into Visual Studio as per project requirements. These tools and plugins may include code analysers, debugging extensions, and productivity tools, enhancing the development process and productivity.

# What are Microservices?

Microservices refers to a software architecture pattern where an application is built as a collection of small, independent, and loosely coupled services that work together to provide a complete functionality. Each microservice is a self-contained unit with its own dedicated business functionality, database, and communication mechanisms. This independence enables teams to work on different microservices simultaneously without interfering with each other.

# What is HTTP?

Hypertext Transfer Protocol (HTTP) is a protocol that allows people to exchange files using the World Wide Web. Hypertext refers to text that includes hyperlinks, unidirectional links that connect documents with other documents or with another place within the same document.

In computing, a protocol is a system of rules that allow for data to be exchanged between different devices. HTTP is an application-level protocol, meaning it defines the format by which application processes on different devices exchange data. Hence, Hypertext Transfer Protocol is the system of rules which allows data to be transferred using the World Wide Web, where Uniform Resource Locators or URLs (for example, https://www.google.com) can be linked together by hypertext.

Essentially, the purpose of HTTP is to allow for the encoding and transference of information between a client and web server. The basic HTTP prefix is http://, but many websites use https://, which means that the website is using a secure or encrypted form of communication for requests and responses.

# User Experience (UX)

When designing the navigation menus and site structure for the HTML web application, the developer prioritises creating an experience that helps users find the information or features they seek. To achieve this, the developer focuses on several key principles:

The developer ensures all navigation labels are clear, concise, and descriptive. Users should immediately understand the purpose of each menu item without ambiguity. Furthermore, the developer organises products logically and hierarchically within each category, grouping similar items together. For example, under the “Men’s Clothing” category, users may find subcategories such as “Jeans,” “Shirts,” and “Shoes” each containing relevant product listings.

Consistency is also maintained across navigation menus throughout the eCommerce application. Whether users are browsing categories, searching for specific products, or accessing their shopping cart, they encounter familiar navigation patterns and layouts. This consistency fosters a sense of familiarity and predictability, leading to a smoother user experience.

# Interaction Design Laws

Miller's Law suggests that the average person can only keep 7 (plus or minus 2) items in their working memory. The developer intends to apply this Law to design forms, menus, and other interfaces with limited options or chunks of information. By breaking down complex tasks or content into smaller, manageable chunks, the aim is to reduce cognitive overload and improve user comprehension and retention.

# Site Map

The developer plans to use a hierarchical site map structure to organise the content and navigation of the website. A hierarchical site map organises pages into a structured hierarchy, with parent and child pages, reflecting the logical relationships between different website sections.

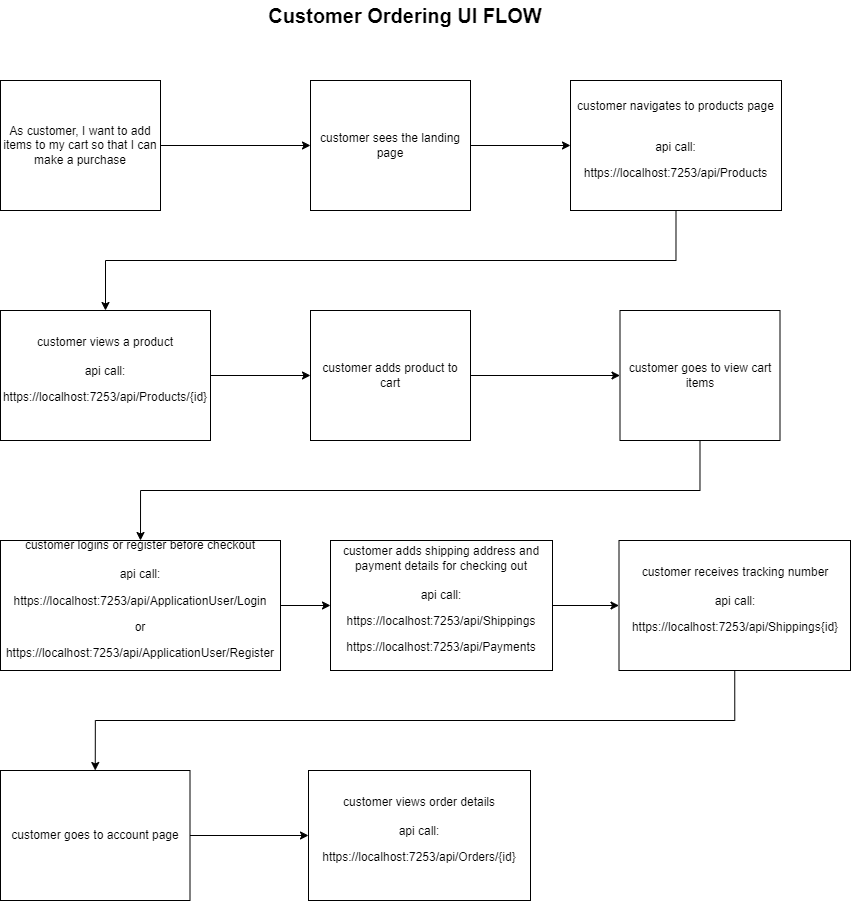
Why Hierarchical Site Map?

A hierarchical site map provides clear organisation and structure to the website’s content. Users can easily navigate through different sections and understand the relationships between pages. Hierarchical site maps facilitate user-friendly navigation by providing a logical hierarchy of pages. Users can navigate from broader categories to more specific content, reducing cognitive load and making it easier to find relevant information. Hierarchical site maps promote consistency in navigation and user experience. Users can expect similar navigation patterns and page structures throughout the website, enhancing usability and familiarity. Search engines favour websites with well-organised and hierarchical site structures. A hierarchical site map improves the website’s indexability, potentially boosting its visibility and search engine ranking.

# Patterns of Visualisation

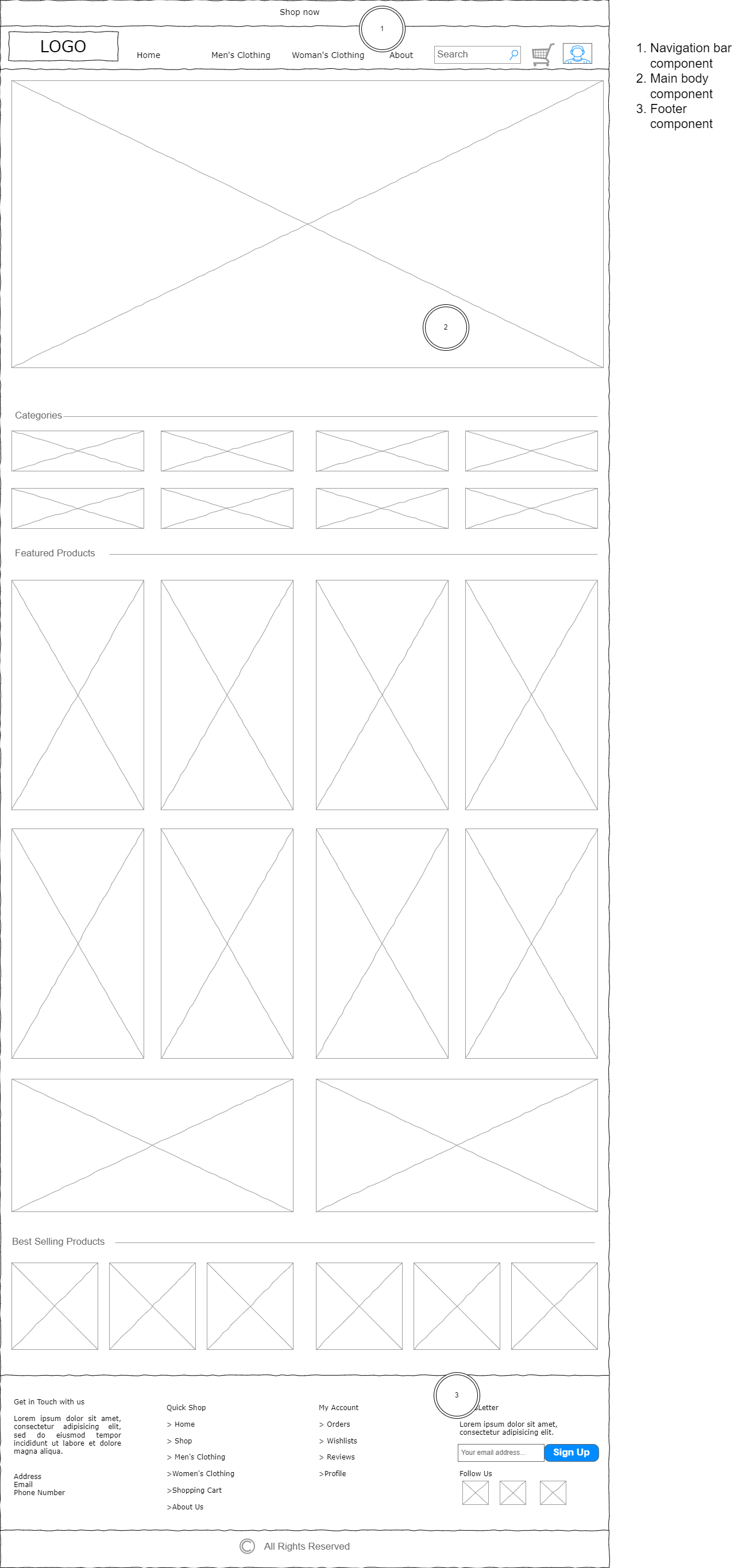
The developer plans to use the Z-Pattern Layout visualisation pattern for the web application. The Z-pattern layout mimics the natural eye movement pattern of users when scanning content on a page. Users typically start at the top left corner, move horizontally to the right, then diagonally down to the bottom left, and finally horizontally to the right again. This layout aligns with users’ natural reading habits, making it intuitive and easy to follow. Organising content along the Z-pattern allows the developer to create a clear visual hierarchy that guides users’ attention from the most important elements to secondary and tertiary content. Users can quickly identify and focus on the main message or action, followed by supporting information or additional details as they scan the page. Furthermore, the Z-pattern layout is well-suited for responsive design, as it works effectively across different screen sizes and resolutions. Whether users view the application on a desktop, tablet, or smartphone, the Z-pattern layout ensures a consistent and optimised user experience.

# UI Flow

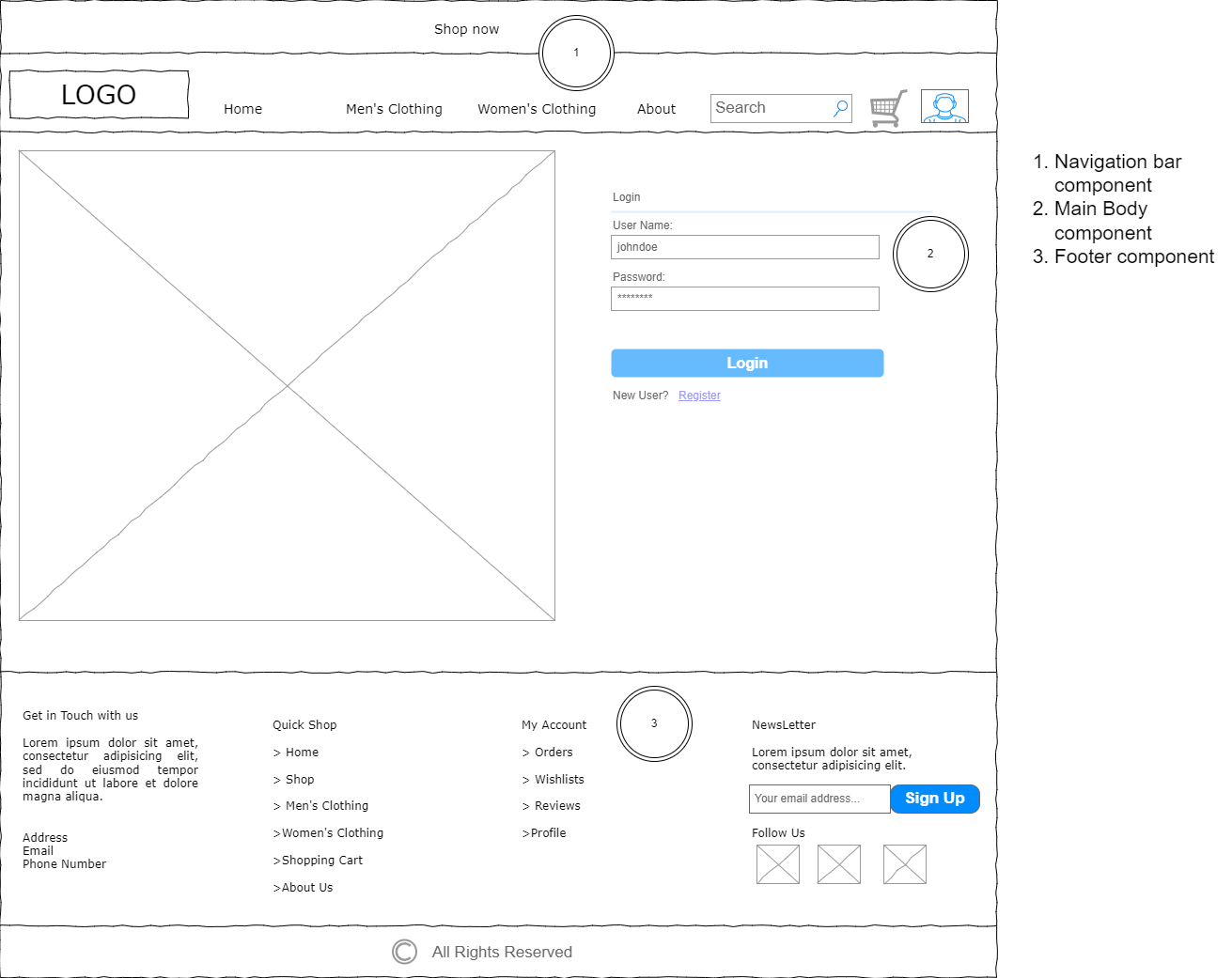


# Wireframe

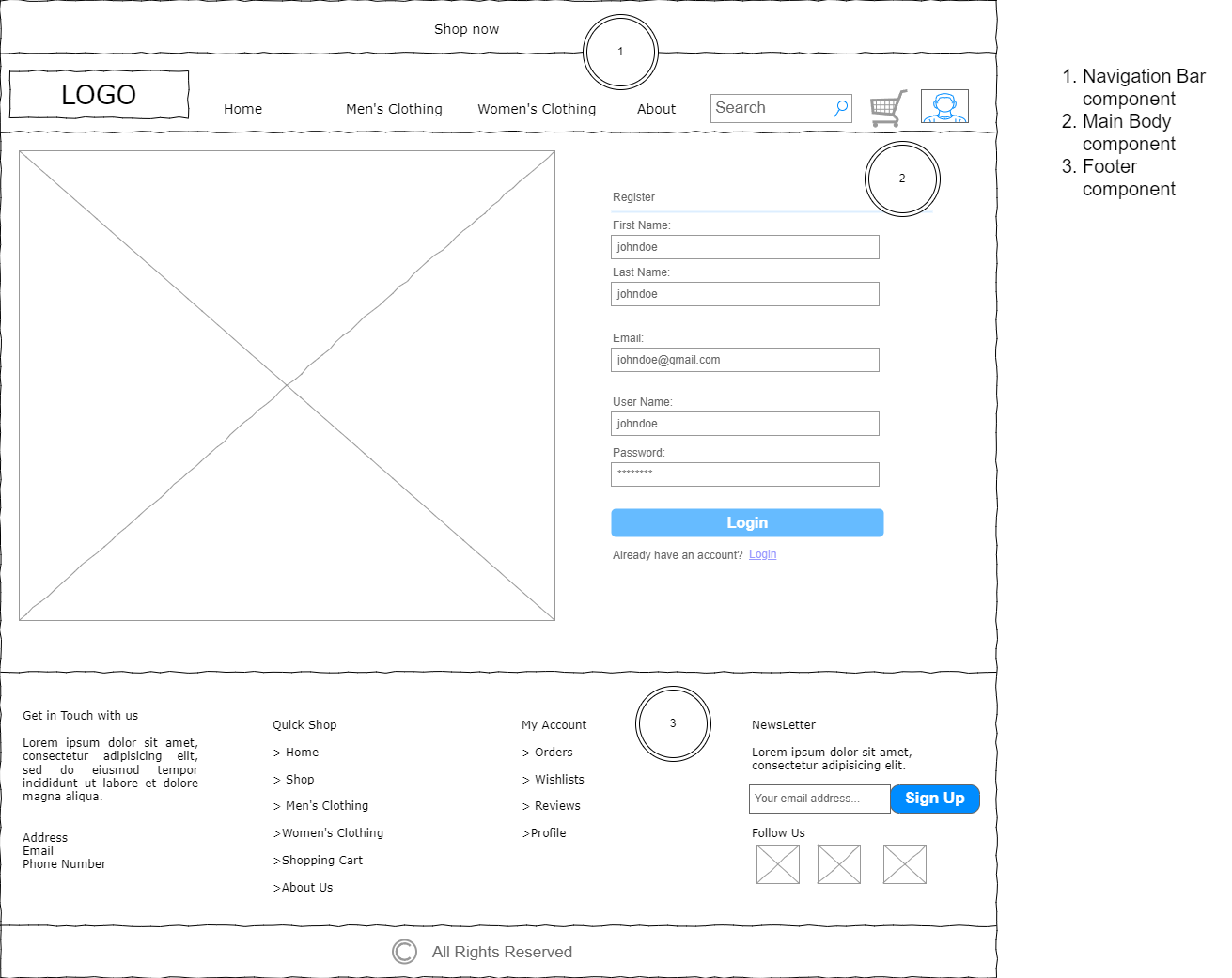
Desktop - Home



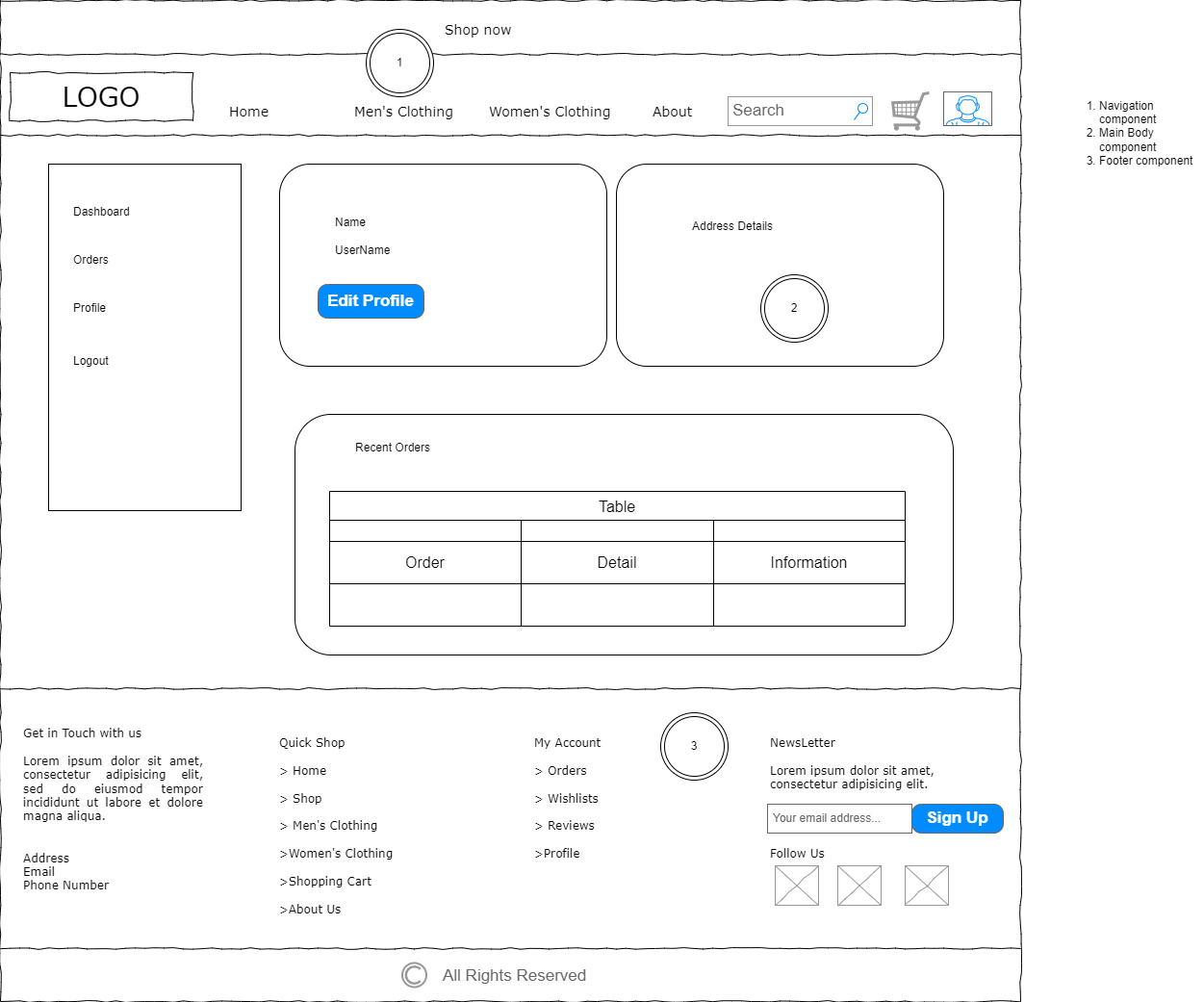
Login Page



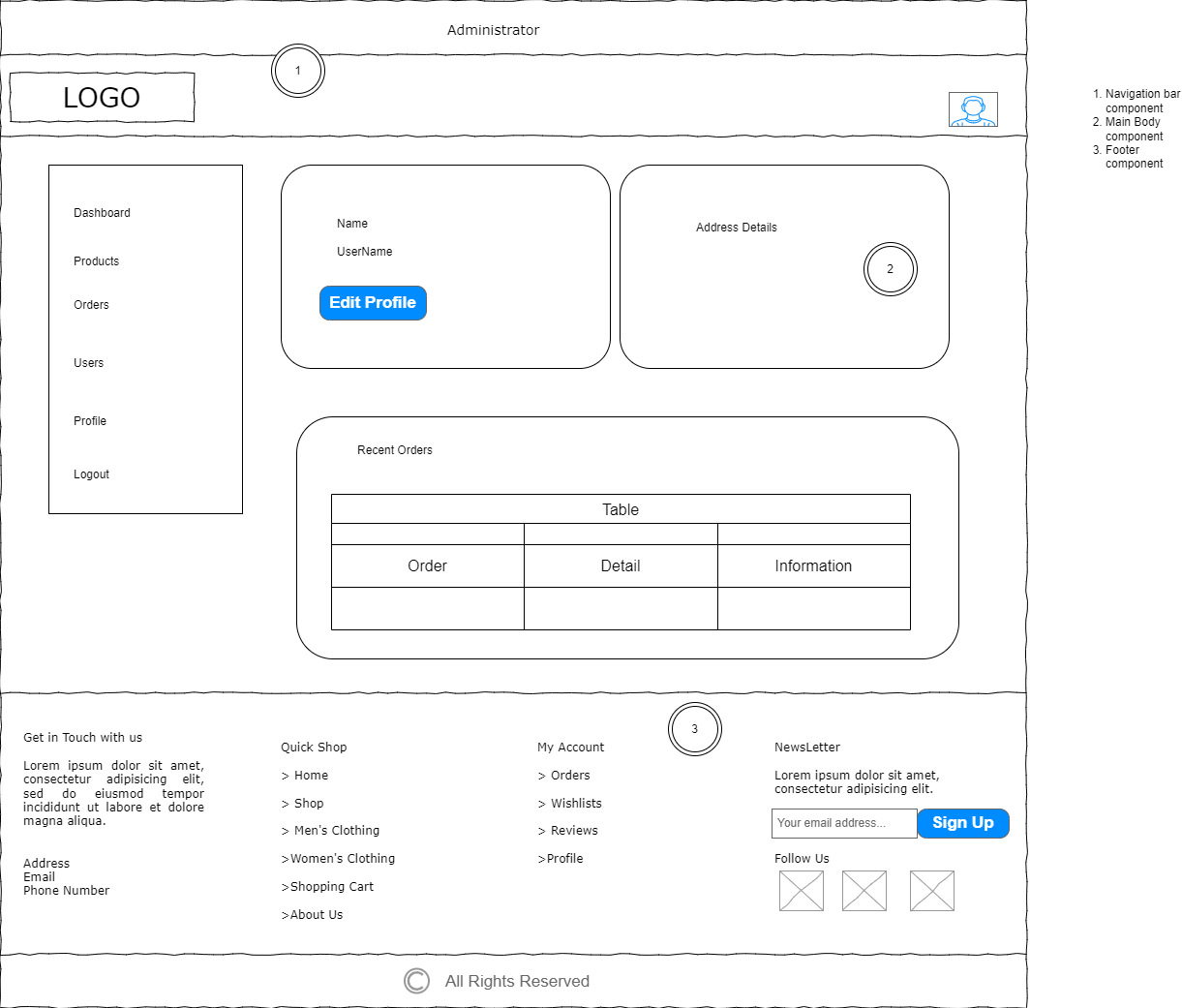
Sign Up

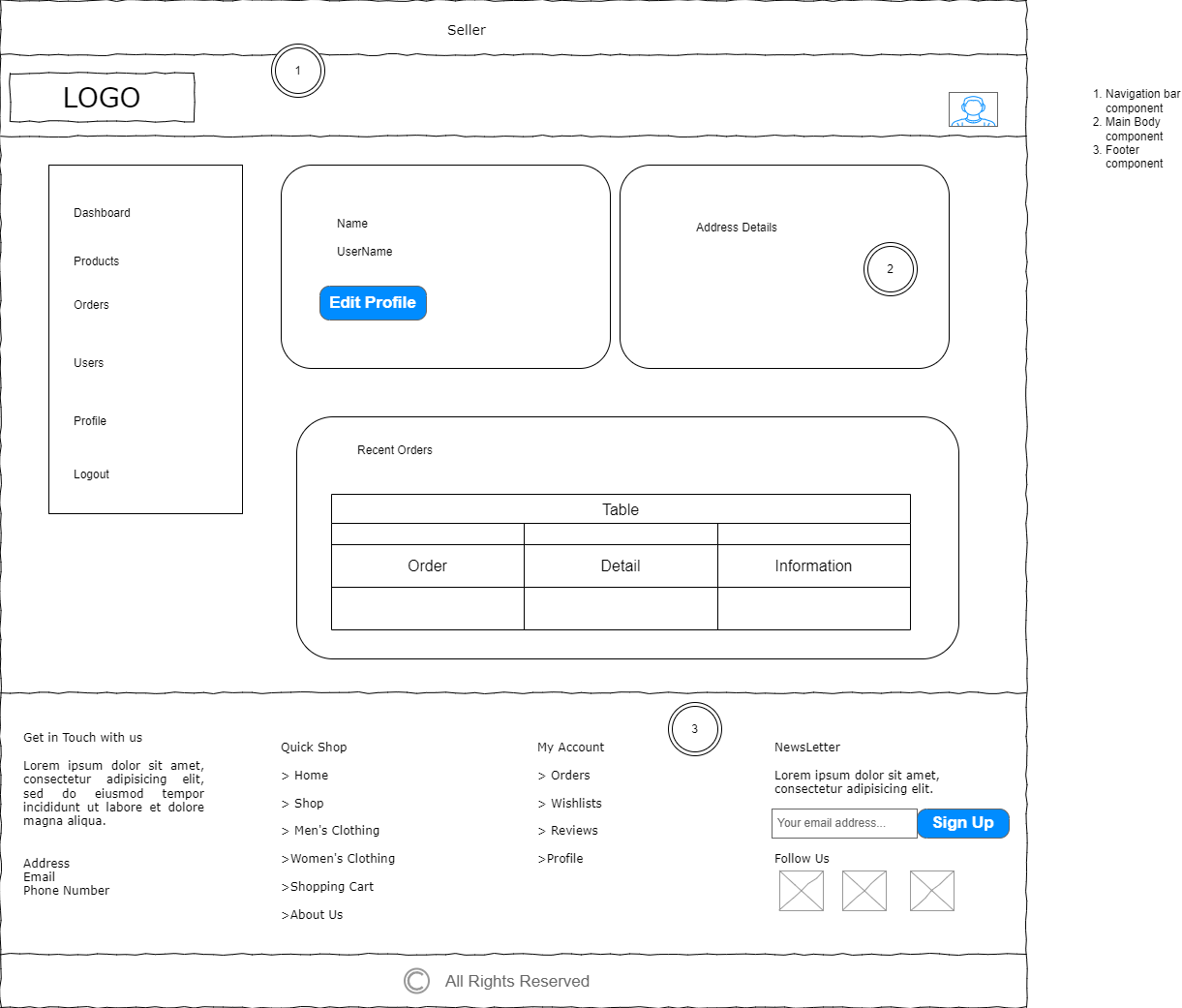


Customer Dashboard

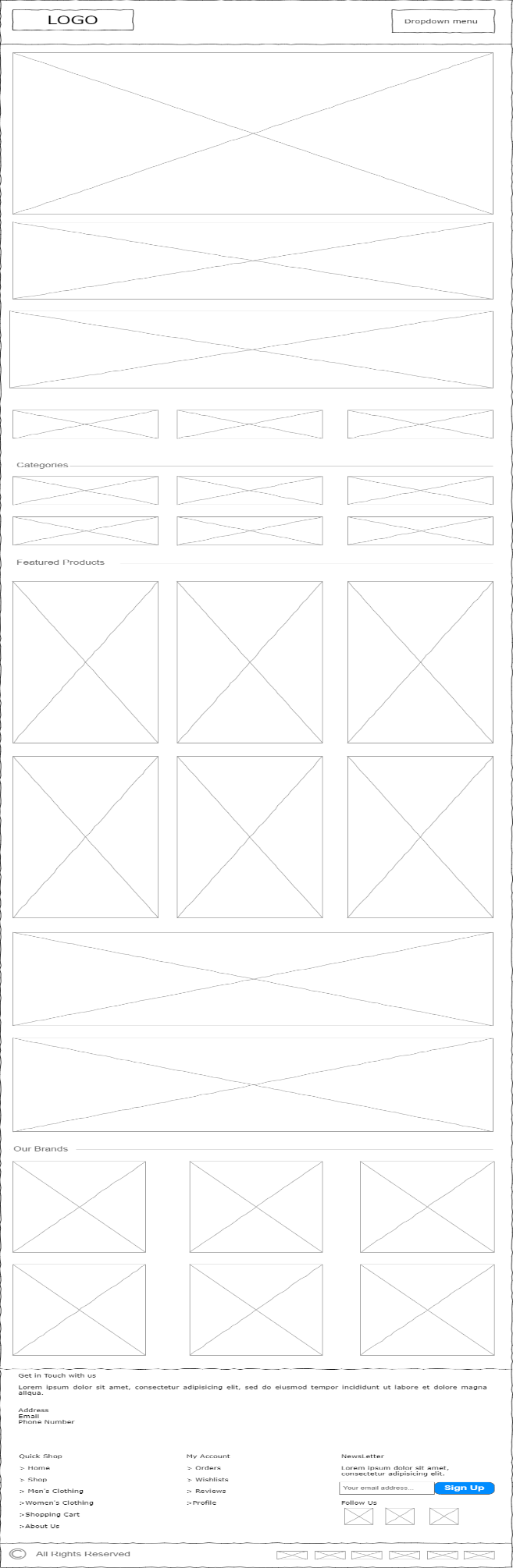


Admin Dashboard

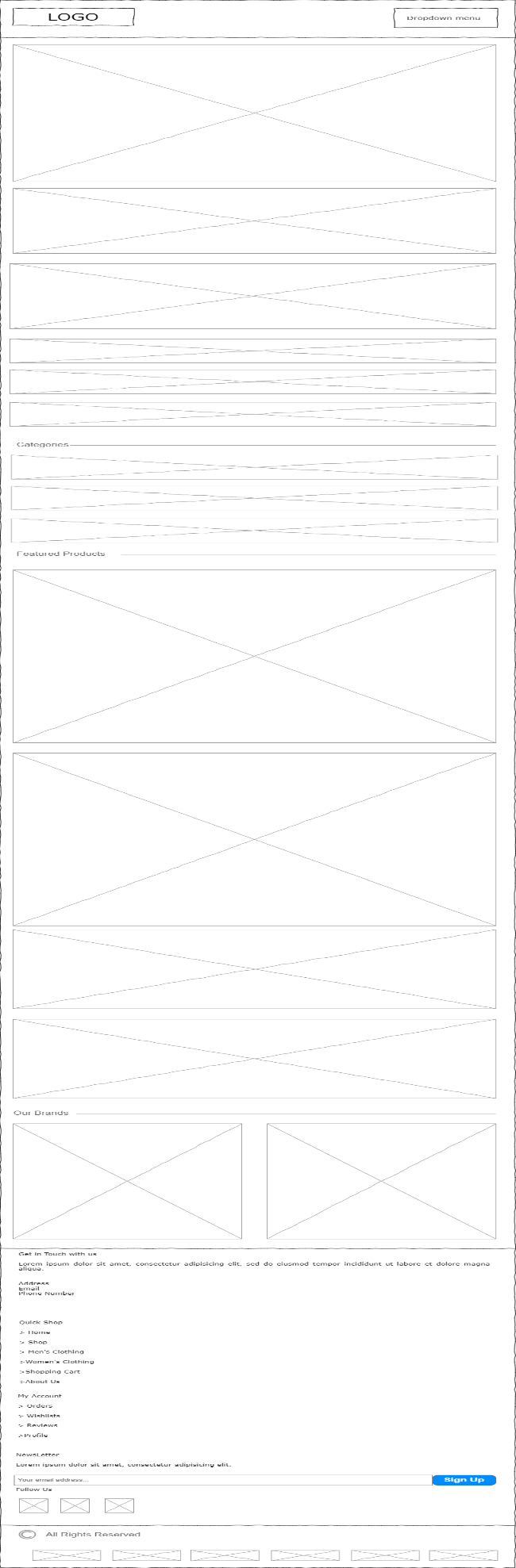


Seller Dashboard

Tablet- Home



Mobile Phone - Home



# Sample data for User Registration

{

"userName": "HaalandS2Admin",

"email": "HaalandS@gmail.com",

"address": "43 Big Street, 4302",

"phoneNumber": "0743929000",

"password": "Haaland@1234",

"firstName": "Sergio",

"lastName": "Haaland",

"role": "Administrator"

}

{

"userName": "JackG",

"email": "Jackg@gmail.com",

"address": "45 Manchester Road, 4367",

"phoneNumber": "0749323223",

"password": "JackG@123456",

"firstName": "Jack",

"lastName": "Grealish",

"role": "Seller"

}

{

"userName": "Havertz02",

"email": "Harvertzj@gmail.com",

"address": "43 Arsenal Road, 3453",

"phoneNumber": "0743232223",

"password": "Harvetz@1234",

"firstName": "Julian",

"lastName": "Harvetz",

"role": "Customer"

}

# Sample data for User Login

|  |  |  |  |
| --- | --- | --- | --- |
| **Security - User Login** |  |  |  |
|  | Username | Password | Role |
| Customer | Efronz | Efron@123456 | Customer |
| Seller | cfarquarson0 | corlissF@123 | Seller |
| Administrator | ZzimelaAdmin | zimelaZ@1234 | Administrator |

# System logging (log file) and design

|  |  |
| --- | --- |
| **logging framework** |  |
| Name of Framework | log4net |
| Name (and path) of log file | LLM\_eCommerce\_RestAPIRolling, located in the bin folder of the project |

|  |  |
| --- | --- |
| **Design Patterns** |  |
| Name of Design Patterns | Composite Pattern |
| Where/how it's used | In the MVC application, the Composite Pattern was used to aggregate multiple objects into one view model, so that the data can be returned to the view and used in the user dashboards. E.g. AdminOrderHistory is a view model with multiple objects which returns as one view model that shows the customers orders and orderdetails for products added by admins. |

List of REST API endpoints with description

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **ID** | **Role** | **Authentication required?** | **HTTP Method** | **Endpoint: ApplicationUser** | **Description** |
| 1 | All | No | POST | <https://localhost:7253/api/ApplicationUser/Register> | Register user |
| 2 | All | No | POST | <https://localhost:7253/api/ApplicationUser/Login> | Login user |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **ID** | **Role** | **Authentication required?** | **HTTP Method** | **Endpoint: UserProfile** | **Description** |
| 1 | Administrator, Seller, Customer | Yes | GET | <https://localhost:7253/api/UserProfile> | Get current logged in user's profile |
| 2 | Administrator, Seller, Customer | Yes | PUT | <https://localhost:7253/api/UserProfile> | Put or updates user profile |
| 3 | Administrator | Yes | PUT | <https://localhost:7253/api/UserProfile/Admin> | Put or updates any user profile |
| 4 | Administrator | Yes | GET | <https://localhost:7253/api/UserProfile/specificUser/Zack> | Get a specific user profile |
| 5 | Administrator | Yes | DELETE | <https://localhost:7253/api/UserProfile/DeleteUser/Zack> | Deletes a User Profile by the username |
| 6 | Administrator | Yes | GET | <https://localhost:7253/api/UserProfile/AllUsers> | Get all user profiles |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **ID** | **Role** | **Authentication required?** | **HTTP Method** | **Endpoint: Categories** | **Description** |
| 1 | All | No | GET | <https://localhost:7253/api/Categories> | Get all Categories |
| 2 | Administrator | Yes | POST | <https://localhost:7253/api/Categories> | Post Category |
| 3 | All | No | GET | <https://localhost:7253/api/Categories/1> | Get Category by ID. Returns Sneaker |
| 4 | Administrator | Yes | POST | <https://localhost:7253/api/Categories/1> | PUT Category by ID. Returns Category Updated - CategoryId:1 |
| 5 | Administrator | Yes | DELETE | <https://localhost:7253/api/Categories/11> | PUT Category by ID. Returns Category Deleted |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **ID** | **Role** | **Authentication required?** | **HTTP Method** | **Endpoint: OrderDetails** | **Description** |
| 1 | All | No | GET | <https://localhost:7253/api/OrderDetails> | Get all OrderDetails |
| 2 | Administrator | Yes | POST | <https://localhost:7253/api/OrderDetails> | Post OrderDetails |
| 3 | All | No | GET | <https://localhost:7253/api/OrderDetails/1> | Get OrderDetails by ID. Returns OrderDetails 1 |
| 4 | Administrator | Yes | POST | <https://localhost:7253/api/OrderDetails/21> | PUT Customer by ID. Returns OrderDetails Updated - OrderDetailId:1 |
| 5 | Administrator | Yes | DELETE | <https://localhost:7253/api/OrderDetails/1> | PUT OrderDetails by ID. Returns OrderDetail Deleted |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **ID** | **Role** | **Authentication required?** | **HTTP Method** | **Endpoint: Orders** | **Description** |
| 1 | All | No | GET | <https://localhost:7253/api/Orders> | Get all Orders |
| 2 | Administrator | Yes | POST | <https://localhost:7253/api/Orders> | Post Orders |
| 3 | All | No | GET | <https://localhost:7253/api/Orders/1> | Get Orders by ID. Returns Order 1 |
| 4 | Administrator | Yes | POST | <https://localhost:7253/api/Orders/1> | PUT Orders by ID. Returns Orders Updated - OrderId:1 |
| 5 | Administrator | Yes | DELETE | <https://localhost:7253/api/Orders/1> | PUT Orders by ID. Returns Order Deleted |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **ID** | **Role** | **Authentication required?** | **HTTP Method** | **Endpoint: Payments** | **Description** |
| 1 | All | No | GET | <https://localhost:7253/api/Payments> | Get all Payments |
| 2 | Administrator | Yes | POST | <https://localhost:7253/api/Payments> | Post Payments |
| 3 | All | No | GET | <https://localhost:7253/api/Payments/1> | Get Payments by ID. Returns Payment 1 |
| 4 | Administrator | Yes | POST | <https://localhost:7253/api/Payments/1> | PUT Payments by ID. Returns Payments Updated - PaymentId:1 |
| 5 | Administrator | Yes | DELETE | <https://localhost:7253/api/Payments/1> | PUT Payments by ID. Returns Payment Deleted |
| 6 | All | No | GET | <https://localhost:7253/api/Payments/SpecificMethod/Card> | Get Payments by Method. Returns Payment 1 |
| 7 | All | No | GET | <https://localhost:7253/api/Payments/SpecificDateASyyyy-mm-dd/2017-04-22> | Get Payments by Date. Returns Payment in that Date |
| 8 | All | No | GET | <https://localhost:7253/api/Payments/SpecificDateASyyyy-mm-dd/2017-04-22/2018-05-05> | Get Payments between Dates. Returns Payment between those Dates |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **ID** | **Role** | **Authentication required?** | **HTTP Method** | **Endpoint: Products** | **Description** |
| 1 | All | No | GET | <https://localhost:7253/api/Products> | Get all Products |
| 2 | Administrator | Yes | POST | <https://localhost:7253/api/Products> | Post Products |
| 3 | All | No | GET | <https://localhost:7253/api/Products/1> | Get Products by ID. Returns Product 1 |
| 4 | Administrator | Yes | POST | <https://localhost:7253/api/Products/1> | PUT Products by ID. Returns Products Updated - ProductId:1 |
| 5 | Administrator | Yes | DELETE | <https://localhost:7253/api/Products/1> | PUT Products by ID. Returns Products Deleted |
| 6 | All | No | GET | <https://localhost:7253/api/Products/SpecificProduct/Nike%20Air%20Force%201> | Get Products by Name. Returns Products 1 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **ID** | **Role** | **Authentication required?** | **HTTP Method** | **Endpoint: Reviews** | **Description** |
| 1 | All | No | GET | <https://localhost:7253/api/Reviews> | Get all Reviews |
| 2 | Administrator | Yes | POST | <https://localhost:7253/api/Reviews> | Post Reviews |
| 3 | All | No | GET | <https://localhost:7253/api/Reviews/1> | Get Reviews by ID. Returns Review 1 |
| 4 | Administrator | Yes | POST | <https://localhost:7253/api/Reviews/1> | PUT Reviews by ID. Returns Reviews Updated - ReviewId:1 |
| 5 | Administrator | Yes | DELETE | <https://localhost:7253/api/Reviews/1> | PUT Reviews by ID. Returns Review Deleted |
| 6 | All | No | GET | <https://localhost:7253/api/Reviews/SpecificRating/5> | Get Reviews by Rating. Returns Review 1 |
| 7 | All | No | GET | <https://localhost:7253/api/Reviews/SpecificDateASyyyy-mm-dd/2017-04-22> | Get Reviews by Date. Returns Review in that Date |
| 8 | All | No | GET | <https://localhost:7253/api/Reviews/SpecificDateASyyyy-mm-dd/2017-04-22/2018-05-05> | Get Reviews between Dates. Returns Review between those Dates |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **ID** | **Role** | **Authentication required?** | **HTTP Method** | **Endpoint: Shippings** | **Description** |
| 1 | All | No | GET | <https://localhost:7253/api/Shippings> | Get all Shippings |
| 2 | Administrator | Yes | POST | <https://localhost:7253/api/Shippings> | Post Shippings |
| 3 | All | No | GET | <https://localhost:7253/api/Shippings/1> | Get Shippings by ID. Returns Shipping 1 |
| 4 | Administrator | Yes | POST | <https://localhost:7253/api/Shippings/1> | PUT Shippings by ID. Returns Shippings Updated - ShippingId:1 |
| 5 | Administrator | Yes | DELETE | <https://localhost:7253/api/Shippings/1> | PUT Shippings by ID. Returns Shipping Deleted |
| 6 | All | No | GET | <https://localhost:7253/api/Shippings/SpecificMethod/1> | Get Shippings by Method. Returns Shipping 1 |
| 7 | All | No | GET | <https://localhost:7253/api/Shippings/SpecificAddress/43%20Havana%20Road> | Get Shippings by Address. Returns Shipping 1 |
| 8 | All | No | GET | <https://localhost:7253/api/Shippings/SpecificDeliveryStatus/Delivered> | Get Shippings by Delivery Status. Returns Shipping 1 |
| 9 | All | No | GET | <https://localhost:7253/api/Shippings/SpecificTrackingNumber/234dds3> | Get Shippings by Tracking Number. Returns Shipping 1 |
| 10 | All | No | GET | <https://localhost:7253/api/Shippings/SpecificDateASyyyy-mm-dd/2017-04-22> | Get Shippings by Date. Returns Shipping in that Date |
| 11 | All | No | GET | <https://localhost:7253/api/Shippings/SpecificDateASyyyy-mm-dd/2017-04-22/2018-05-05> | Get Shippings between Dates. Returns Shippings between those Dates |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **ID** | **Role** | **Authentication required?** | **HTTP Method** | **Endpoint: Wishlists** | **Description** |
| 1 | All | No | GET | <https://localhost:7253/api/Wishlists> | Get all Wishlists |
| 2 | Administrator | Yes | POST | <https://localhost:7253/api/Wishlists> | Post Wishlists |
| 3 | All | No | GET | <https://localhost:7253/api/Wishlists/1> | Get Wishlists by ID. Returns Wishlist 1 |
| 4 | Administrator | Yes | POST | <https://localhost:7253/api/Wishlists/1> | PUT Wishlists by ID. Returns Wishlists Updated - WishlistId:1 |
| 5 | Administrator | Yes | DELETE | <https://localhost:7253/api/Wishlists/1> | PUT Wishlists by ID. Returns Wishlist Deleted |
| 6 | All | No | GET | <https://localhost:7253/api/Wishlists/SpecificDateASyyyy-mm-dd/2017-04-22> | Get Wishlists by Date. Returns Wishlist in that Date |
| 7 | All | No | GET | <https://localhost:7253/api/Wishlists/SpecificDateASyyyy-mm-dd/2017-04-22/2018-05-05> | Get Wishlists between Dates. Returns Wishlist between those Dates |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **ID** | **Role** | **Authentication required?** | **HTTP Method** | **Endpoint: CategoriesProducts** | **Description** |
| 1 | All | No | GET | <https://localhost:7253/api/CategoriesProducts> | Get all Categories Products |
| 2 | All | No | GET | <https://localhost:7253/api/CategoriesProducts/1> | Get Categories Products by ID. Returns Categories Products 1 |
| 3 | All | No | GET | <https://localhost:7253/api/CategoriesProducts/All> | Get all Category Products |
| 4 | All | No | GET | <https://localhost:7253/api/CategoriesProducts/Prod/1> | GET Category Products by ID. Returns Category Products Updated - productId:1 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **ID** | **Role** | **Authentication required?** | **HTTP Method** | **Endpoint: CustomersOrdersDetails** | **Description** |
| 1 | Customer | Yes | GET | <https://localhost:7253/cusordd/MyOrders> | Get current logged in user's order details |
| 2 | Administrator | Yes | GET | <https://localhost:7253/cusordd> | Get All CustomersOrderDetails information |
| 3 | Administrator, Seller | Yes | GET | <https://localhost:7253/curordd/userId/1> | Get a CustomersOrderDetails with User id |
| 4 | Administrator, Seller, Customer | Yes | GET | <https://localhost:7253/cursordd/OrderDetailsInfo/1> | Get OrderDetails Info passing order details id |
| 5 | Administrator, Seller, Customer | Yes | GET | <https://localhost:7253/cusordd/customersByOrderId/1> | Get OrderDetails with Order id |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **ID** | **Role** | **Authentication required?** | **HTTP Method** | **Endpoint: CustomersOrders** | **Description** |
| 1 | Customer | Yes | GET | <https://localhost:7253/cusord/MyOrders> | Get current logged in user's orders |
| 2 | Administrator | Yes | GET | <https://localhost:7253/cusord> | Get All CustomersOrders information |
| 3 | Administrator, Seller | Yes | GET | <https://localhost:7253/curord/userId/1> | Get a CustomersOrders with User id |
| 4 | Administrator, Seller, Customer | Yes | GET | <https://localhost:7253/cursord/OrdersInfo/1> | Get Orders Info passing order details id |
| 5 | Administrator, Seller, Customer | Yes | GET | <https://localhost:7253/cusordd/customersShippingId/1> | Get Orders with shipping id |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **ID** | **Role** | **Authentication required?** | **HTTP Method** | **Endpoint: CustomersOrdersPayments** | **Description** |
| 1 | Customer | Yes | GET | <https://localhost:7253/cusordPay/MyOrdersPayments> | Get current logged in user's order payments |
| 2 | Administrator | Yes | GET | <https://localhost:7253/cusordPay> | Get All CustomersOrdersPayments information |
| 3 | Administrator, Seller | Yes | GET | <https://localhost:7253/cusordPay/userId/1> | Get a CustomersOrdersPayments with User id |
| 4 | Administrator, Seller, Customer | Yes | GET | <https://localhost:7253/cusordPay/OrdersPaymentInfo/1> | Get OrdersPaymentDetails with OrdersPayment id |
| 5 | Administrator, Seller, Customer | Yes | GET | <https://localhost:7253/cusordPay/OrderId/1> | Get OrdersPaymentDetails with Order id |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **ID** | **Role** | **Authentication required?** | **HTTP Method** | **Endpoint: CustomersReviews** | **Description** |
| 1 | Customer | Yes | GET | <https://localhost:7253/cusrev/MyReviews> | Get current logged in user's reviews |
| 2 | Administrator | Yes | GET | <https://localhost:7253/cusrev> | Get All CustomersReviews information |
| 3 | Administrator, Seller | Yes | GET | <https://localhost:7253/cusrev/userId/1> | Get a CustomersReviews with User id |
| 4 | Administrator, Seller, Customer | Yes | GET | <https://localhost:7253/cusrev/ReviewInfo/1> | Get ReviewDetails with Review id |
| 5 | Administrator, Seller, Customer | Yes | GET | <https://localhost:7253/cusrev/ProductId/1> | Get ReviewDetails with Product id |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **ID** | **Role** | **Authentication required?** | **HTTP Method** | **Endpoint: CustomersReviews** | **Description** |
| 1 | Customer | Yes | GET | <https://localhost:7253/cuswish/MyWishlists> | Get current logged in user's wishlists |
| 2 | Administrator | Yes | GET | <https://localhost:7253/cuswish> | Get All CustomersWishlists information |
| 3 | Administrator, Seller | Yes | GET | <https://localhost:7253/cuswish/userId/1> | Get a CustomersWishlists with User id |
| 4 | Administrator, Seller, Customer | Yes | GET | <https://localhost:7253/cuswish/WishlistInfo/1> | Get WishlistDetails with Wishlist id |
| 5 | Administrator, Seller, Customer | Yes | GET | <https://localhost:7253/cuswish/wishlistsByProductId/1> | Get WishlistDetails with product id |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **ID** | **Role** | **Authentication required?** | **HTTP Method** | **Endpoint: UsersProducts** | **Description** |
| 1 | Administrator, Seller | Yes | GET | <https://localhost:7253/usrsprds/MyProducts> | Get current logged in user's products |
| 2 | Administrator | Yes | GET | <https://localhost:7253/usrsprds> | Get All UsersProducts information |
| 3 | Administrator | Yes | GET | <https://localhost:7253/usrsprds/userId/1> | Get a UsersProducts with User id |
| 4 | Administrator | Yes | GET | <https://localhost:7253/usrsprds/ProductInfo/1> | Get a ProductDetails with Product id |
| 5 | Administrator | Yes | GET | <https://localhost:7253/usrsprds/usersProductsId/1> | Get a ProductDetails with User Product id |

# Source Control

Git Repo: **Angular-Full-Stack-System**

# Directory and naming services

LLM\_eCommerce\_EFCODE1ST

LLM\_eCommerce\_RESTAPI

LLM\_eCommerce\_Ang

# Assumptions and Dependencies

The assumptions and dependencies of this project are outlined in this section.

**Assumption:** Developers have access to high-performance workstations running Windows 11 or later. This assumption is based on the expectation that developers will have sufficient computing power, memory, and storage capacity to compile, test, and debug code efficiently. This assumption also implies that developers are familiar with the Windows operating system and can leverage its features and functionalities during development.

**Dependency:** Dependency on Visual Studio 2022 Community edition. The project relies on Visual Studio 2022 Community edition as the primary IDE for software development activities. This dependency is critical as Visual Studio provides a comprehensive suite of tools, libraries, and features necessary for building, debugging, and deploying applications using various programming languages, including C#. Furthermore, the dependency on Visual Studio 2022 Community edition offers a compelling combination of features, compatibility, accessibility, and community support, making it an ideal choice for facilitating efficient and cost-effective software development within the project.

The Angular Web App project depends on the Entity Framework (EF) Code 1st and Rest API projects to run effectively. Hence, the EF Code 1st project should be run first, and then add migrations and then update the database. This will create the database for the entities involved in the system. After the initial process the Rest API Project needs to be opened and run the migration commands in package manager console making sure to state the context to add migration to and making sure to database is updated with the migrations so that the identity database tables are created in order to enable registration and login functionality and to be able to access critical data. The Angular also needs to update its packages before running properly, make sure to run npm install on the terminal, so all the node modules are added to the project. Make sure to keep the Rest API running and open the Angular Web app now to use it.

# Lessons Learned

This section details the lessons learned throughout the project lifecycle.

## Lesson Learned - Week 5 .NET with Angular

Lesson learned: Component-Based Architecture. Angular promotes a component-based architecture that helps create modular and reusable code. Each component encapsulates its logic and template, making it easier to manage and test.

Lesson learned: Two-Way Data Binding. Angular's two-way data binding allows for automatic synchronization of data between the model and the view, which simplifies the process of building dynamic and interactive applications.

Lesson learned: Dependency Injection. Angular's dependency injection framework facilitates the creation of services that can be easily injected into components and other services. This promotes a modular design and makes it easier to manage dependencies and conduct unit testing.

Lesson Learned: Routing. Angular's Router module allows for defining and navigating between different views or pages within a single-page application. It supports lazy loading of modules, which can improve performance by loading only the necessary parts of the application.

## Lesson Learned - Week 4 .NET ASP.NET/Blazor

Lesson: Understanding ASP.NET Core. This lesson was taught through a combination of lectures and hands-on exercises. The developer set up various ASP.NET Core projects using Visual Studio, explored the middleware pipeline, and implemented a simple web application utilizing MVC and Razor Pages.

Lesson: Setting Up ASP.NET Core Projects. The developer created new ASP.NET Core projects using different templates in Visual Studio. The coach demonstrated how to configure dependency injection and set up the project structure. The developer then practiced by creating his own projects and adding necessary services such as http client, and client settings.

Lesson: Razor Pages. The concept of Razor Pages was introduced through lecture notes and examples. The developer built a sample web application using Razor Pages, focusing on the syntax and structure. Coding exercises reinforced the developers understanding of embedding C# code in HTML.

## Lesson Learned - Week 3 .NET Web App

Lesson Learned: Keeping HTML for structure, CSS for styling, and JavaScript for functionality ensures maintainability and clarity by structuring the project with distinct roles for each technology, making updates easier and more organized.

Lesson Learned: Robust error handling is crucial for a smooth user experience and debugging by adding try...catch blocks around API requests to manage network failures and unexpected responses.

Lesson Learned: Dynamically updating the DOM based on API data provides interactive and real-time updates, by using JavaScript to create, append, and update HTML elements based on the data received from API calls.

Lesson Learned: Ensuring the application works well on different devices enhances user accessibility and experience by applying responsive design principles and testing the application on various screen sizes and devices.

## Lesson Learned - Week 2 .NET REST API

Lesson Learned: The importance of specifying CORS policies to control access to the API from different origins. Clear CORS policies help mitigate security risks and enable controlled sharing of resources across domains.

Lesson Learned: Experience showed that specifying monitoring and logging requirements, including logging levels, log formats, and monitoring metrics, enhances the observability and manageability of the API. Clear guidelines aid in diagnosing issues and optimising performance.

## Lesson Learned - Week 1 .NET Data Access

Lesson Learned: Each ADO.NET approach (Connected, Disconnected, Disconnected with procedures) has its advantages and disadvantages, and the choice depends on the specific requirements of the application. ADO.NET disconnected approach is preferable for applications requiring offline data manipulation and improved performance.

Lesson Learned: Stored procedures offer better security, performance, and maintainability but require additional development effort upfront.

Lesson Learned: Careful consideration should be given to the trade-offs between performance, scalability, and development complexity when selecting an approach for ADO.NET applications.

## Lesson Learned - Week 3. OOD 3 - TDD Solo Project

Lesson Learned: XML comments help to make code clear and understandable by providing structured documentation that describes the purpose, behaviour, and usage of code elements. By documenting requirements using XML comments, developers create a formal reference for the behaviour and functionality of the software, aiding in communication and understanding among team members.

Lesson Learned: Test-Driven Development ensures that code meets requirements and functions correctly by writing tests before implementation. This approach validates that the code behaves as expected and helps catch bugs early in the development process.

## Lesson Learned - Week 2. OOD 2 - Code-based Solo Project

Lesson Learned: Break down requirements into smaller, manageable units to facilitate understanding and implementation. Instead of a single requirement for “customer cart”, the requirement should be split into separate requirements for “Add to cart”, “view cart items,” and “remove cart items.” This helps with prioritising tasks to complete and manage your time effectively.

Lesson Learned: Every class should focus on a single responsibility to ensure the code is easy to understand and manageable. For example, a customer class should concentrate on activities specific to a customer, such as getting customer details, updating details, etc. This also helps other developers on the team to understand your code easily.

Lesson Learned: Always test your code using debugging tools to see what is happening with the variables; this helps identify flaws in the code. Also, it helps with being familiar with code so that when you encounter an error, you know what to do to fix it.

## Lesson Learned - Week 1. OOD 1 - Research Project

Lesson Learned: Treat the requirements specification as a living document that evolves over time. Regularly review and update requirements based on feedback, lessons learned from previous projects, and changes in business or technical environments. Continuous improvement ensures that the requirements remain relevant and up-to-date throughout the project lifecycle.

Lesson Learned: Implement revision control and versioning mechanisms to manage changes to requirements effectively. Versioning ensures that the evolution of requirements is documented, and changes are tracked over time. It also facilitates collaboration among team members and allows for the rollback of changes if needed.