MOBISTORE

E-Commerce App

Project Overview

MobiStore

E-commerce Web Application

Project Description

MobiStore is an e-commerce application designed to streamline the online shopping experience. The platform allows users to browse, search, and purchase products from various categories. It supports product listings, user account management, shopping cart functionality, and a secure checkout process. The application efficiently handles orders, ensuring a smooth shopping journey from selection to payment.

Technologies Used

- **Back-End**: .NET MVC
- **Front-End**: jQuery, HTML, CSS
- **Database**: SQL Server

System Architecture

MVC Structure

Model

The Model represents the data and business logic in the MobiStore application. It manages product data, customer information, and order details, all stored in the SQL Server database. The key data models include Product, ApplicationUser, and StoreSearchModel, each responsible for managing related data fields and business rules.

View

The View is responsible for rendering the user interface and presenting data to the user. It includes product listings, the shopping cart interface, and the checkout page. These views use HTML, CSS, and jQuery to create a responsive and user-friendly experience. Users interact with these views to browse products, manage their cart, and complete purchases.

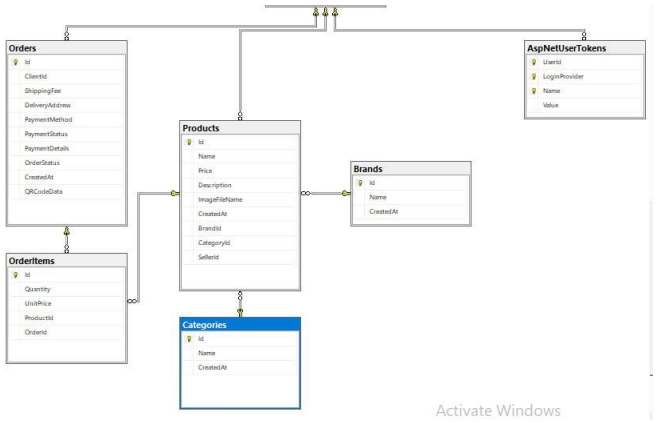
Model

The Controller acts as the intermediary between the Model and the View. It handles incoming HTTP requests, retrieves the required data from the Model, and returns the appropriate View to the user. For example, the ProductsController manages requests related to product browsing and selection, while the StoreController handles actions like listing and searching stores.

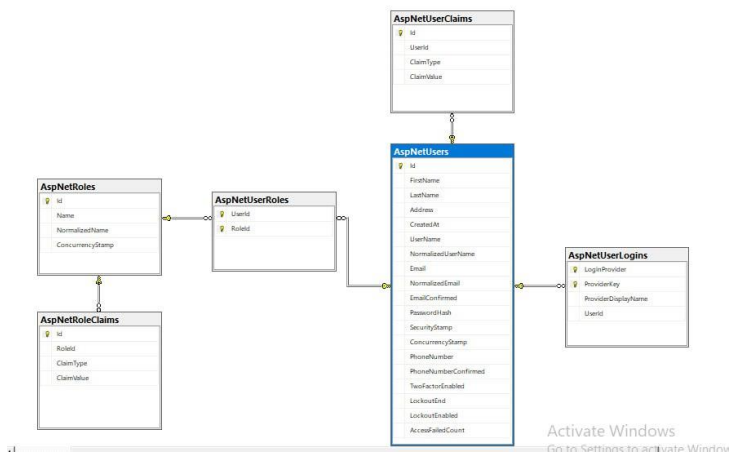
Database Design

The MobiStore database is built using SQL Server, with a well-structured entity-relationship model. Below are two visual representations of the database structure:
Main E-Commerce Entities: This diagram covers the core e-commerce-related tables :

Products: Stores product information such as name, price, description, category, and brand.
Orders: Contains order data like shipping fee, delivery address, and payment status.
OrderItems: Represents individual items in an order with their quantity and price.
Categories: Defines the product categories.
Brands: Stores brand data for products.



User Authentication and Roles: This diagram covers tables related to user authentication and role management:
AspNetUsers: Stores user details such as email, password hash, and security stamp.
AspNetRoles: Defines different roles for user permissions.
AspNetUserRoles: Links users with their respective roles.



Installation and Setup

Prerequisites

Ensure you have the following tools installed before setting up the MobiStore project:

- .NET SDK: Install the latest version of the .NET SDK to run the MVC application.
- SQL Server: The application requires a SQL Server instance to connect to the database.
- jQuery: The project utilizes jQuery for some front-end interactions.
- HTML & CSS: Basic knowledge of HTML and CSS for UI customization.
- Visual Studio: Recommended for project development

Step-by-Step Installation Guide

1. Clone the repository:
   Open your terminal and clone the MobiStore repository:
   ```
   git clone https://github.com/your-username/mobistore.git
   ```

2. Open the project:
   Navigate to the project folder and open it in **Visual Studio**:
   ```
   cd mobistore
   ```

3. Install dependencies:
   Restore NuGet packages for .NET by running:
   ```
   dotnet restore
   ```

4. Set up the database:
   • Ensure your SQL Server is running.
   • Update the connection string in the appsettings.json file to match your SQL Server

configuration.
• Apply migrations to set up the database schema:

```
dotnet ef database update
```

5. Run the project:
   Restore NuGet packages for .NET by running:

```
dotnet run
```

Step-by-Step Installation Guide

The application defines three distinct user roles with specific permissions:

1. **Admin:**
   • Access to the Admin Dashboard.
   • Can manage products, users, orders, categories, and brands.
   • Full control over the platform, including updating orders/payments status

2. **Customer:**
   • Can browse products, view details, and add items to the cart.
   • Can complete orders and track orders history
   • Has access to order history and can update personal information.

3. **Seller:**
   • Can manage their products.
   • Can view orders related to their products.
   • Does not have access to managing other sellers' or customers' products or orders.

Core Features

Authentication

- The platform uses **role-based authentication**, supporting **three primary user roles**: Admin, Customer, and Seller.
- **Admin** users manage all aspects of the platform, including products, orders, and users.
- **Seller** accounts can add, edit, and manage their own products.
- **Customers** can browse products, add items to their cart, and complete purchases via the checkout system.

CRUD Operations

- **Categories and Brands**: Admins can create, update, and delete **categories** and **brands** that are assigned to products.
- **Products**: Admins and Sellers can add, edit, and delete products. Each product is associated with a **category**, **brand**, and **seller**.
- **Orders**: Customers can place orders, and Admins can manage order status, and payment methods from the dashboard.

Cart and Checkout

- Customers can add products to their **cart**, proceed to **checkout**, and place orders.
- **Orders** are saved in the **order history**, and customers can download a **receipt** after completing a purchase.

Order History and Receipts

- Admins have access to a comprehensive **order history** section, where they can view order details, shipping status, and payment information.
- **Downloadable order receipts** are available for customers to keep track of their purchases.

Search and Filtering

- The platform provides **search** and **filtering** functionalities to help customers find products by category, brand, or product name.

Challenges and Solutions

Challenges

- **Handling complex product filtering**: Implementing efficient search and filter functionality for products with multiple attributes (category, brand, price) was a challenge due to potential performance bottlenecks.
- **Optimizing database queries**: Some queries involving product and order history were slowing down page loads, particularly for the Admin's dashboard.
- **Order receipt generation**: Ensuring that downloadable receipts are generated dynamically with accurate formatting and data.

Solutions

- **Optimized filtering**: We leveraged **SQL indexing** and applied efficient **LINQ** queries to handle complex filtering and search efficiently.
- **Performance improvements**: By optimizing **SQL queries** and utilizing **lazy loading** in Entity Framework, the database operations became significantly faster.
- **Receipt generation**: Implemented a dynamic **PDF generation** tool to create professional receipts that users can download post-purchase, ensuring all order details are correctly formatted and clear.

Future Enhancements

- **Enhanced Product Recommendations**: Implement machine learning algorithms to recommend products to users based on their browsing and purchase history.
- **Live Inventory Management**: Improve inventory tracking by adding real-time inventory updates that sync across all user sessions.
- **Mobile Application**: Develop a native mobile app for iOS and Android to provide a more accessible and responsive shopping experience.
- **Advanced Analytics for Sellers**: Provide sellers with advanced sales reports and customer insights to help them optimize their listings.
- **Multilingual Support**: Expand the platform to include multiple languages, making it accessible to a global audience.
- **Third-party Payment Gateway Integrations**: Add support for multiple payment gateways like PayPal, Stripe, or Apple Pay.
- **Subscription Services**: Introduce a subscription model where customers can subscribe to products and receive them at regular intervals.

Conclusion

Project Summary

The **MobiStore** e-commerce platform was developed using the **.NET MVC** framework, with a focus on providing a seamless user experience for both customers and sellers. The project implemented core e-commerce functionalities such as product listings, a shopping cart, user authentication, and order management.

During the development process, the integration of **Sendinblue's API** for email communication, efficient product management, and smooth user authentication stood out as key elements in ensuring a robust platform.

Key Takeaways

- **Technology Stack Mastery**: Improved expertise in **.NET MVC**, **SQL Server**, and **jQuery**, reinforcing the fundamentals of web application development.
- **API Integration**: Gained experience integrating third-party services like **Sendinblue** for authentication-related emails.
- **E-Commerce System Design**: Learned the intricacies of designing a user-friendly and scalable e-commerce system, from inventory management to checkout and order processing.

Appendix

References

- **.NET MVC Documentation**: https://docs.microsoft.com/en-us/aspnet/mvc/overview
- **Sendinblue API**: https://developers.sendinblue.com/docs/getting-started
- **SQL Server**: https://docs.microsoft.com/en-us/sql/sql-server/?view=sql-server-ver15

Glossary

- **CRUD**: Refers to **Create, Read, Update, Delete** operations on data.
- **MVC**: **Model-View-Controller**; an architectural pattern used in software development.
- **Dependency Injection**: A design pattern used to implement **IoC (Inversion of Control)**, where the control of creating dependencies is transferred to a framework or service.
- **ERD**: **Entity Relationship Diagram**; a data modeling technique to visually describe the relationships between entities in a database.