



# ДИПЛОМЕН ПРОЕКТ

**Тема: РАЗРАБОТКА НА УЕБ ПРИЛОЖЕНИЕ ЗА  
МУЗИКАЛНИ ИНСТРУМЕНТИ**

**Професия: код 481020 „Системен програмист“  
Специалност: код 4810201 „Системно програмиране“**

**Изготвил:**  
**Мартин Мирославов Петров**  
**Ученик от XII<sup>B</sup> клас**

**Ръководител - консултант:**  
**Румяна Ачанова**

**Перник, 2025 г.**

# Contents

УВОД .....	3
1. ПЪРВА ГЛАВА – ПРОУЧВАНЕ .....	5
1.1. Проучване на пазара .....	5
1.2. Преглед на технологии и езици за програмиране .....	8
1.3. Избор на технологии и езици за програмиране .....	11
2. ВТОРА ГЛАВА – ПРОЕКТИРАНЕ .....	13
2.1. Описание и анализ на изискванията .....	13
2.1.1. Описание и анализ на функционалните изисквания .....	13
2.1.2. Потребителски истории (user stories) .....	14
2.1.3. UML диаграми за случаи на употреба.....	16
2.1.4. Анализ на нефункционалните изисквания .....	16
2.2. Проектиране .....	18
2.2.1. Проектиране на база данни .....	18
3. ТРЕТА ГЛАВА – РЕАЛИЗАЦИЯ .....	23
3.1. Технология на създаване на уеб приложението .....	23
3.2. Структура на базата от данни .....	24
3.3. Реализация на уеб приложението .....	28
3.3.1. Архитектура на приложението .....	28
3.3.2. Контролери .....	29
3.3.3. Сървиси .....	30
3.3.4. Вю модели.....	32
3.3.6. Други технологии и функционалности .....	42
ЗАКЛЮЧЕНИЕ.....	43
ИЗПОЛЗВАНА ЛИТЕРАТУРА .....	44
ПРИЛОЖЕНИЯ .....	45

# УВОД

**Кратко въведение в темата** - В съвременния свят електронната търговия заема все по-важно място, като предлага удобство и бърз достъп до разнообразни продукти. Един от секторите, който се възползва от дигитализацията, е продажбата на музикални инструменти. Традиционните магазини често са ограничени от географското си местоположение и складовите си наличности, докато онлайн платформите позволяват на клиентите да избират измежду богато разнообразие от продукти, да сравняват цени и да получават детайлна информация за характеристиките на инструментите

**Актуалност на проблема** - Проучванията показват, че хората все повече купуват стока онлайн. 90% от тях избират магазините с детайлно описание на продукта и качествена реална снимка.

**Обект на дипломния проект** - уеб приложение за продажба на музикални инструменти.

**Предмет на дипломния проект** - проектиране и разработка на уеб приложение за продажба на музикални инструменти, базирано на ASP.Net Core MVC, на езика C#, което използва база от данни на MS SQL Server.

**Цел на дипломния проект** е да се проектира и разработи уеб приложение за музикални инструменти – Music Shop, разработено на езика C#. Използвана технология ASP.Net Core MVC и бази от данни на MS SQL Server.

**Целевата аудитория** на уеб магазина за музикални инструменти обхваща както любители, така и професионални музиканти, които търсят качествени инструменти. Платформата е насочена както към начинаещи, които се нуждаят от достъпни продукти, така и към опитни музиканти, студия за звукозапис и преподаватели, които търсят надеждно оборудване. Освен това, уеб магазинът може да привлече музикални групи, оркестри и колекционери, за които са важни детайлните описания и автентичността на продуктите. Със своята функционалност и удобен интерфейс, платформата цели да улесни избора и покупката, като предоставя богата информация и сигурни методи на плащане.:

Приложението трябва да поддържа надеждна система за автентикация и авторизация на своите потребители, които трябва да бъдат разпределени минимум в три роли: нерегистрирани потребители (гости), регистрирани потребители (клиенти) и администратор. Администраторското меню трябва да съдържа секция

„Статистически модул“. Нерегистрираните потребители да могат да се регистрират. Клиентите да могат да пускат поръчки. Всички потребители да могат да разглеждат съдържанието на сайта, да търсят продукти в търсачката и да ги филтрират.

Уеб приложението трябва да се базира на съвременни технологии, стандарти и модели, което гарантира лесна бъдеща поддръжка, ефективност и възможност за надграждане.

За постигане на целите на дипломния проект трябва да изпълним следните задачи:

- ✓ Проучване и анализ
  - Проучване на конкурентните сайтове и услугите, които предлагат те.
  - Проучване на известните технологии
  - Избор на подходящи технологии.
- ✓ Проектиране
  - Анализ на изискванията към уеб приложението.
  - Създаване на потребителски истории
  - Описание на случаи на употреба
  - Проектиране на базата данни
  - Създаване на прототипи на потребителския интерфейс на приложението
- ✓ Реализация
  - Създаване на ASP.Net Core MVC проект
  - Създаване на базата от данни
  - Изграждане на навигационна система на сайта
  - Имплементация на функционалностите на сайта

**Софтуер за изграждане на уеб приложението** - за изграждането на уеб приложението ще бъдат използвани: *Visual Studio Community 2022* и *MS SQL Server 2022*. Уеб приложението ще бъде базирано на *ASP.NET Core* технологията на Microsoft, трислойния модел, MVC модела и езика за програмиране *C#*. Ще бъдат използвани и други помощни софтуерни продукти, като *SQL Server Management Studio (SSMS)*, както и други езици и технологии, като *HTML*, *CSS*, *Razor*, *Bootstrap*, *JavaScript* и др.

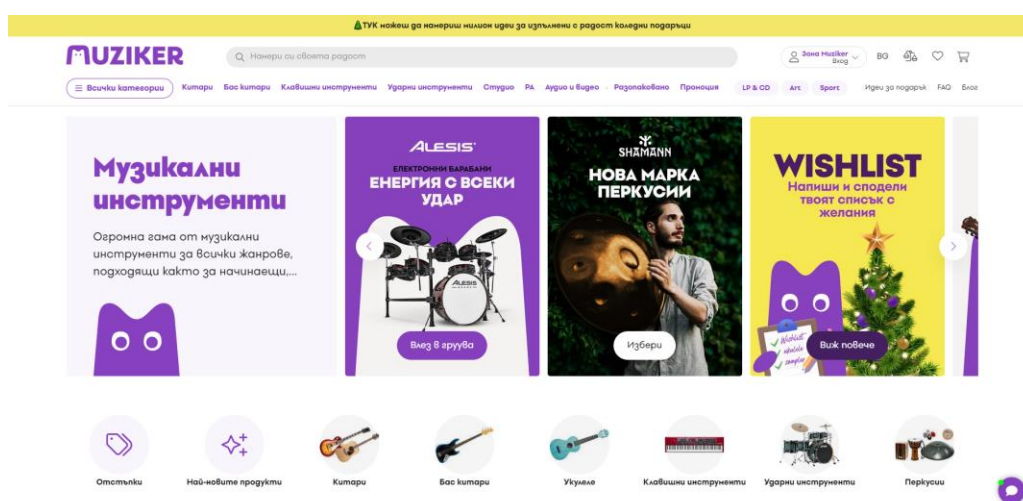
# 1. ПЪРВА ГЛАВА – ПРОУЧВАНЕ

## 1.1. Проучване на пазара

Реализацията на софтуера трябва да задоволява нуждите както на клиента , така и на собственика и разработчика. За да се избере най-добрия подход за реализацията, функционирането и поддръжката на продукта, трябва да се направи проучване на пазара.

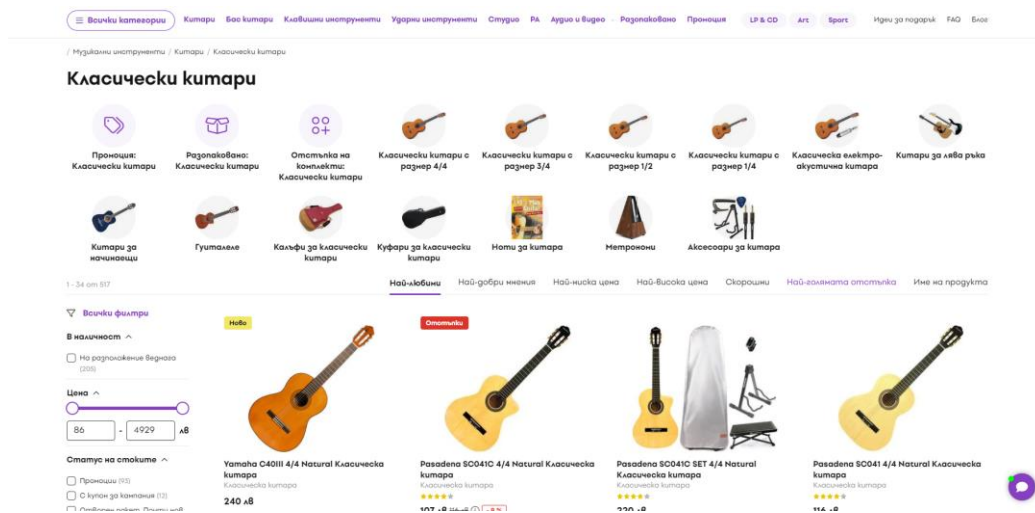
За направата на онлайн магазин за музикални инструменти са разгледани няколко вече съществуващи.

### 1. <https://www.muziker.bg/muzikalni-instrumenti>



Фигура 1.1

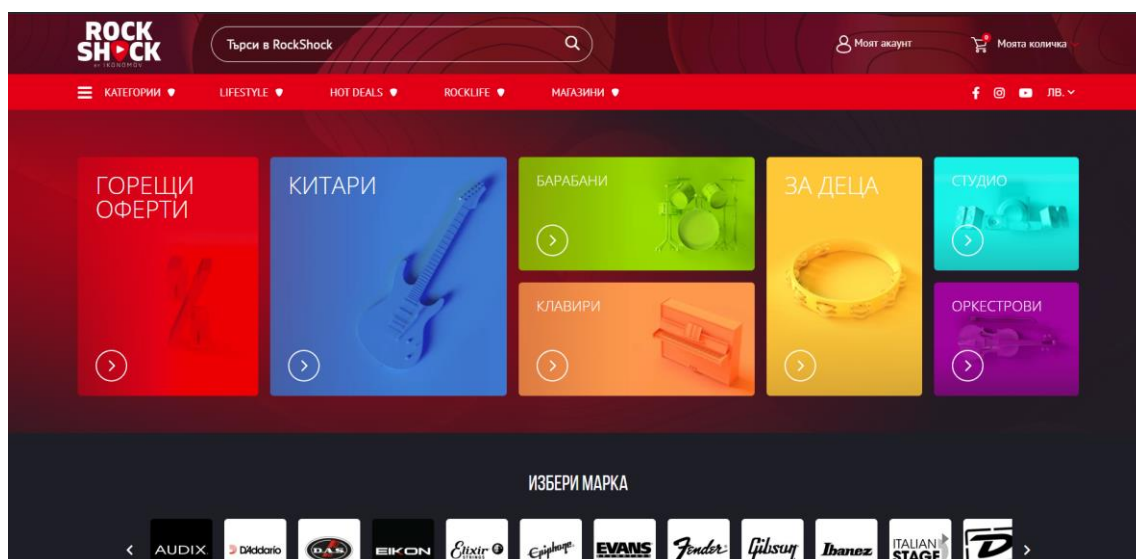
**Силни страни** - има няколко надписа уверяващи клиента, че са са на правилното място.Надолу в страницата има продукти на отстъпка, най-популярни и нови. Има поле за търсене на продукти на видимо място. Както фигура 2 показва може да избираме лесно подкатегиите в случай на нужда. Лесно видим е филтъра.Съществува функционалност за сравнение на продукти.



Фигура 1.2

**Слаби страни-**в началната страница елементите за продукти и категории са твърде големи, побирайки по-малко.

## 2. <https://rockshock.eu>

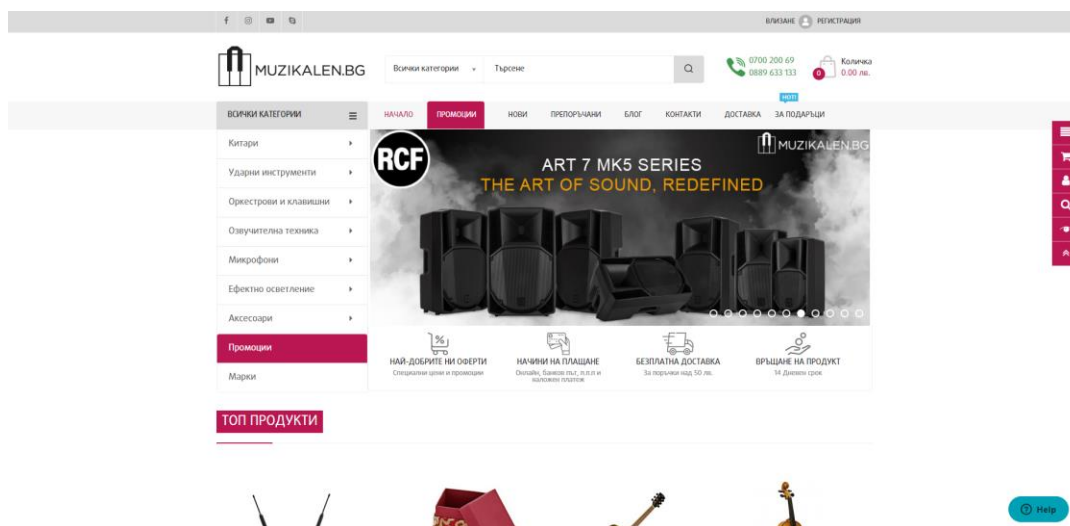


Фигура 1. 3

**Силни страни-**темата на страницата е разноцветна, давайки по-добра ориентация.Зареждането на страниците в сайта става плавно, без да се презарежда горната част с навигационната лента и търсачката.

**Слаби страни-** броят на филтрите е малък. Съществува бутон за смяна на паричната единица, но той не работи. На места се ползва ненужно втори език.

### 3. <https://www.muzikalen.bg/>



Фигура 1.4

**Силни страни-**в началната страница е побрано почти всичко(контакти, категории, информация за доставка и начин на плащане и др.) без да е объркващо.

**Слаби страни-**Целият сайт е бял, липсват цветовете.

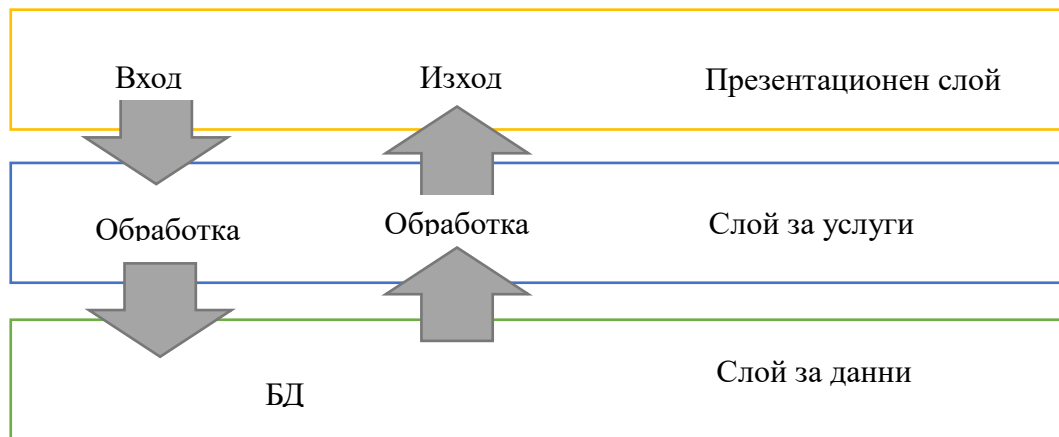
Проучвайки конкурентните сайтове, установих, че повечето от тях имат твърде много категории на началната страница и клиентът трудно се ориентира. На други облика е твърде прост и не привлича вниманието с нищо. Проблем също е, че на някои продукти липсва описание или то е доста малко.

За да може да решим какви функционалности трябва да предлага продуктът ни, трябва да станем клиент и да се запитаем: “Какво искам от този сайт и какво ще го направи по-добър?”. Затова влязох в ролята на клиент в тези сайтове и анализирах дизайна, подредеността и облика им. По мои наблюдения няма един сайт в който всички тези неща да са изпълнени без забележки. Виждайки проблемите реших да осигуря тези функционалности на клиентите за по-добро пазаруване.

## 1.2. Преглед на технологии и езици за програмиране

При изграждането на съвременните приложения могат да се използват различни модели: трислоен, MVC, MVVM

**Трислоен модел** – състои се от презентационен слой (потребителски интерфейс), слой за бизнес логика, служи за обработка на данните и слой за данни, който се състои от сървър база данни.[3]



Фигура 1.5 Трислоен модел

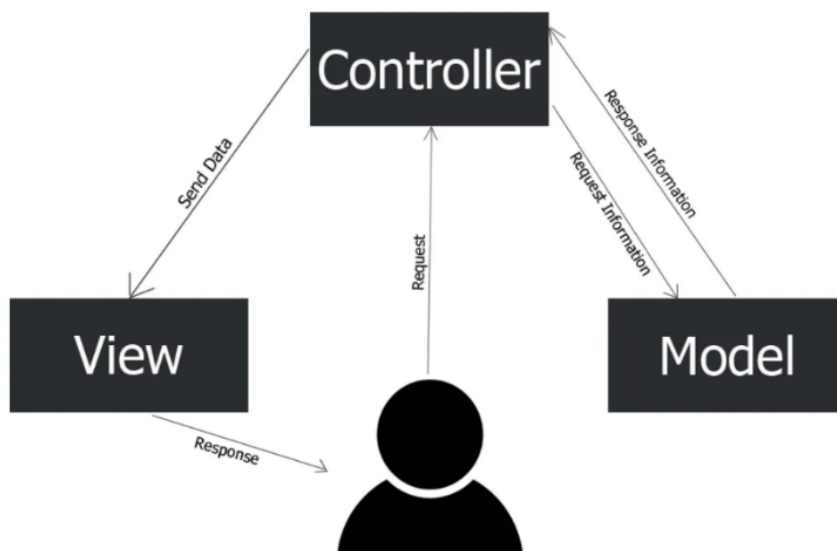
**MVC шаблон** – Model-View-Controller е архитектурен шаблон, който подрежда папките и компонентите на приложението.[3]

*Моделът* - съдържа множество от класове, с които ще работим. Може да съдържа валидация и капсулация на данните. Работи с данните независимо от потребителския интерфейс. Прилича на Data Access Layer от трислойния модел.

*Изгледът* – определя как ще се показва потребителския интерфейс на приложението. Изгледите са главни и частични.

*Контролерът* – главният MVC компонент, който е логиката на приложението.





Фигура 1.6 MVC

**MVVM (Model-View-ViewModel)** е архитектурен модел. Моделът и изглежда са както в MVC модела, но има нов компонент ViewModel, който работи с модела и помага да се запази изгледът отделен от модела, в същото време действа като контролер за улеснена комуникация между View и Model.[3]

В разработката на уеб приложения могат да се използват множество езици. Те се разделят на две главни категории Back-end и Front-end. Едни от най-популярните за Front-end са JavaScript, HTML/CSS, а за Back-end са Python, PHP, Java, C#.[5]

За разработката на сървърната част на приложението могат да се използват езиците:



- **Python** – популярността си дължи на своята универсалност и лесен за изучаване синтаксис. Той е широко използван в науката за данни, изкуствения интелект и уеб разработката;



- **Java** – продължава да бъде основен за разработка на мобилни приложения за Android. Това е езикът с една от най-големите общности от разработчици;



- **C#** – изключително популярен в разработката на игри и десктоп приложения, особено за платформата .NET. C# се отличава със своята мощност и гъвкавост.[6]



- **PHP** е бърз и гъвкав програмен език, на който се създават динамични уебсайтове – от блогове до онлайн магазини. Най-известната и използвана CMS платформа WordPress е написана на PHP. За разлика от останалите

програмни езици PHP е специализиран език за уеб разработка и създаването на уебсайтове. [7]

**За стилистичната част(Front-end) се използват:**



- **JavaScript** –адаптивен скриптов език, който се ползва предимно за разработка на уеб интерфейси. Осигурява динамика и интерактивност на страницата.[8]



- **HTML/CSS** HyperText Markup Language е основния език за описание на уеб страници, а CSS (Cascading Style Sheets) е езикът за описание на стилове – определя как ще изглеждат елементите в страницата, работи с HTML.



- **Bootstrap** – най-популярната платформа с отворен код, която комбинира HTML, CSS и JavaScript код. Bootstrap се състои от вече готово написани класове. Има страхотна документация, лесно се персонализира и улеснява работата на разработчиците.
- **Razor** – поддържа C#. Това е много полезно в случай, че искаме да комбинираме C# и HTML код. Razor рендерира C# кода и го изкарва в брауъра като чист HTML

**Други технологии разледани в уеб приложението са:**

**ASP.NET Core-framework**, разработен от Microsoft за разработка на уеб приложения. Някои от предимствата е че поддържа различни платформи, и е с отворен код.

**Laravel**-е уеб фреймуърк на PHP за разработка на уеб приложения, който следва шаблона model-view-controller. PHP е сложен за поддръжане и трябва постоянно да се следи какви нови технологии излизат.[10]

Производителността и сигурността не са толкова развити при Laravel, докато ASP.Net Core поддържа силна автентикация и авторизация, които го защитават от атаки като например Cross-site scripting (XSS).

Съвременните приложения използват бази данни за съхранение на информацията.Базите данни биват релационни и нерелационни.



**SQL** – Structured Query Language е език за работа с релационни бази от данни. С него правим заявки към бази данни, тъй като той е структурирен език за заявки



**MS SQL Server** – система за управление на релационна база от данни, разработена от Microsoft

### 1.3. Избор на технологии и езици за програмиране

За разработката на моето уеб приложение избрах технологията ASP.Net Core с езика C# и бази данни MS SQL Server, както и други езици и шаблони, които ще се използват за разработването на проекта, заради описаните по-долу предимства.

Предимствата на MVC са че компонентите са отделени и могат да се използват многократно и в други приложения.

ASP има доста предимства. NET платформата се поддържа от Microsoft, което е голям плюс, защото това е огромна, доказала се със своите продукти и услуги, компания.

NET Core е само част от NET платформата, която осигурява най-добрата среда за разработка и полезен дебъгер. Има страхотна документация и голяма общност, където хората споделят опита си и пробват нови проекти. Тази технология е с отворен код. NuGet – има над 166,000 библиотеки.

Чрез NET Core се създават десктоп приложения, уеб приложения, мобилни приложения, облачни услуги, игри, приложения за интернет на нещата. Голям плюс е че уеб приложенията разработени на тази платформа работят както на Windows, така и на macOS, и на Linux.[2]

Технологията позволява и работа с AZURE, облачна технология, за хостване на уеб приложението. ASP.Net Core се използва и за сложни проекти, което е още едно предимство, тъй като може да се направи сериозно приложение

C# - обектно-ориентиран език, който е много лесен за учене, има красив синтаксис, грешките са лесни за разбиране. C# е един от най популярните езици, като от 2019 година той става още по-използван и популярен, което го прави надежден и може да се разчита на този език дълго време. Има добре развити garbage collector, threading, LINQ и други. Можем да пишем C# код в браузъра – Blazor. C#, освен като ООП, може да се ползва като функционално програмиране или да се пише и Machine learning.[1]

Причината да избира HTML, CSS, JavaScript и Bootstrap е че те са най-популярните и универсални езици.

## **2. ВТОРА ГЛАВА – ПРОЕКТИРАНЕ**

Проектирането е важен етап от цикъла на софтуерната разработка. Необходимо е да се анализират изискванията към програмния продукт, да се проектира БД, да се създадат прототипи на потребителския интерфейс.

### **2.1. Описание и анализ на изискванията**

#### **2.1.1.Описание и анализ на функционалните изисквания**

Функционалните изисквания определят какво трябва да прави системата. Те са свързани с видовете потребители и техните права.

Приложението ще поддържа надеждна система за автентикация и авторизация на своите потребители, които ще бъдат разпределени в три роли: нерегистрирани потребители (гости), регистрирани потребители (клиенти) и администратор.

Всички потребители трябва да виждат меню със секции „Начало“, „Продукти“ и „Контакти“. Страницата „Продукти“ ще позволява разглеждане на продуктите в сайта. При разглеждането да има възможност за филтриране на информацията по цена и производител. Нерегистрираните потребители (гостите) трябва да могат да се регистрират в системата. Регистрираните потребители (клиенти) трябва да могат да се логват в системата чрез потребителско име и парола. Приложението ще позволява логнатите потребители да правят поръчка на продукти. Те ще могат да разглеждат своите собствени поръчки на продукти. При поръчка на продукти системата ще следи за изчерпване на наличните количества. Преди финализиране на поръчката потребителят ще вижда общата дължима сума.

Приложението ще има един администратор с вградени в системата данни. Администраторът ще има възможност да добавя, редактира и изтрива данни за продуктите, предлагани в сайта. Той ще може да вижда направените поръчки на всички клиенти. Администраторът ще има възможност да вижда данните и да изтрива регистрирани потребители, които не са направили поръчки. Администраторското меню ще съдържа секция „Статистически модул“ със статистическа информация за брой регистрирани потребители, брой продукти.

### 2.1.2. Потребителски истории (user stories)

Потребителските истории се използват при разработването на софтуер, като начин да се помогне на разработчиците да разберат желанията и нуждите на своите потребители. Потребителските истории са кратки и следват прост шаблон.

#### **Предимства:**

- Кратки
- Разбираеми за потребителите и разработчиците
- Не е нужна поддръжка
- Не налагат особени усилия

#### **Недостатъци**

- Може да са непълни
- Различна интерпретация

Потребителските истории имат за цел да представят гледната точка на потребителя, а не гледната точка на създателя (разработчика на програмата). Те могат да са от голяма полза при определяне на случаите на употреба.

#### **Шаблон за потребителски истории (кой-какво-защо):**

***„Като (потребител) искам да (действие/възможност), за да (цел).”***

***„Като (потребител) мога да (действие/възможност), за да (цел).”***

Този шаблон изобразява представата на потребителя, като идентифицира какво иска/може/ и защо го иска.

#### **Примери:**

##### **Гост**

- „Като гост искам да мога да разглеждам продуктите в сайта, за да мога да избира предпочитан продукт.“
- „Като гост искам да имам възможност за филтриране на продукти по даден критерий, с цел по-бързото им намиране.“
- „Като гост искам да мога да се регистрирам в системата, за да стана регистриран потребител.“

##### **Регистриран потребител – клиент**

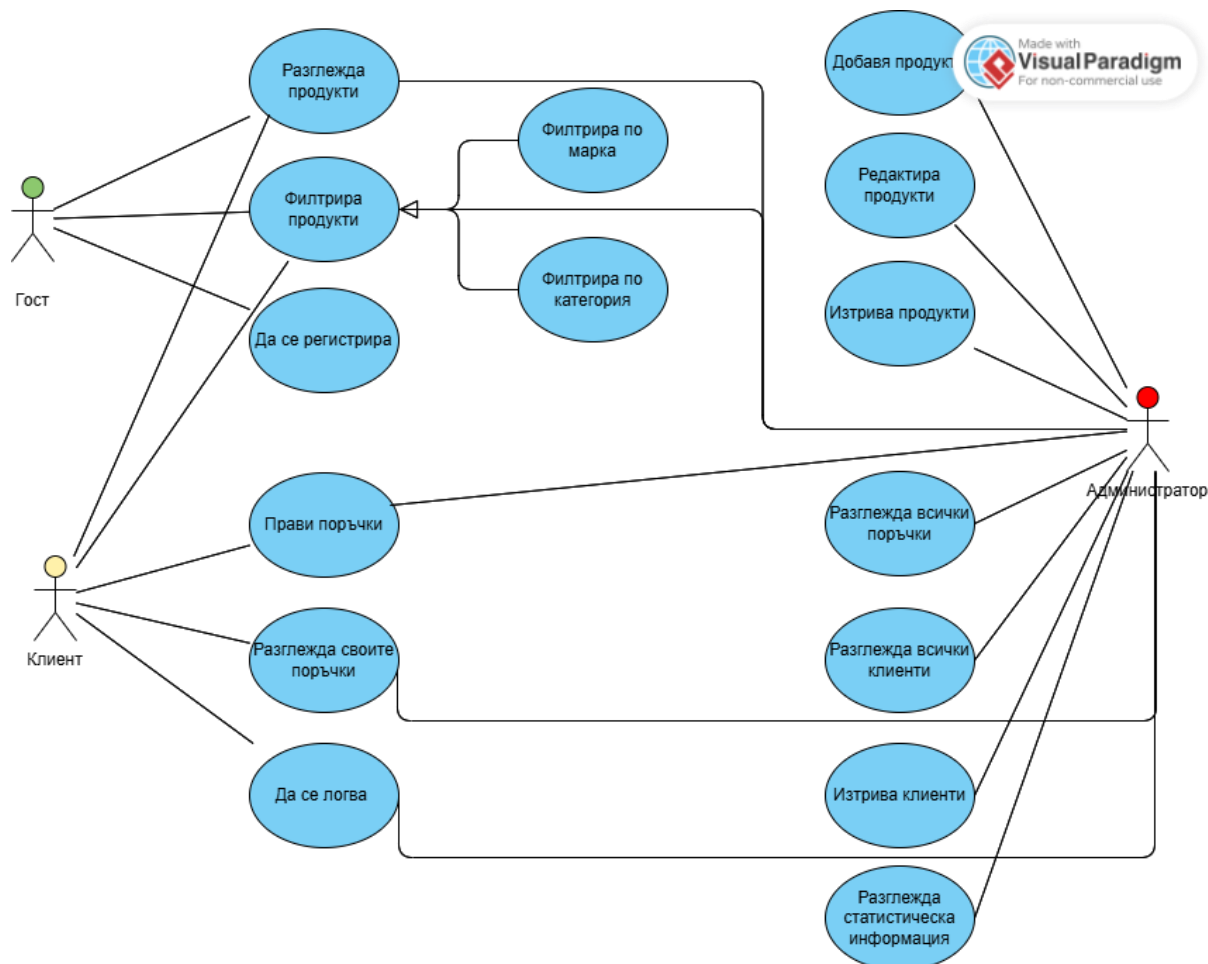
- „Като клиент искам да мога да се логвам в системата с потребителския си профил, за да мога да поръчвам продукти.“
- „Като клиент искам да имам възможност да разглеждам продуктите, за да знам какво да си поръчам.“
- „Като клиент искам да имам възможност за филтриране на продукти по даден критерий, с цел по-бързото им намиране.“
- „Като клиент искам да мога да поръчвам продукти, с цел тяхното закупуване.“
- „Като клиент искам при поръчка на продукт да виждам дължимата сума, с цел да осигуря средства за плащането.“
- „Като клиент искам да имам възможност да виждам историята на всички мои поръчки, направени досега.“
- „Като клиент искам да имам количка и да мога да я достъпвам, за да мога да поръчвам няколко поръчки наведнъж.“

#### **Администратор**

- „Като администратор искам да мога да се логвам в системата с администраторски профил, за да имам достъп до всички модули.“
- Като администратор искам да мога да добавям, редактирам и изтривам продукти, с цел закупуването им от клиентите.“
- „Като администратор искам да имам достъп до всички направени поръчки от клиенти, за да мога да виждам подробности за тях.“
- „Като администратор искам да виждам всички регистрирани потребители и тяхната информация.“
- „Като администратор искам да мога да изтривам регистрирани потребители.“
- „Като администратор искам да имам достъп до статистическия модул в системата, за да мога да виждам различна статистическа информация:брой поръчки, брой продукти, брой клиенти и обща сума на всички поръчки.“

### 2.1.3. UML диаграми за случаи на употреба

Случаите на употреба показват какво могат да правят потребителите в различни роли. Един ясен начин на представянето им е чрез UML диаграми. Те осигуряват визуализация на случаите на употреба и действащите лица в системата.



Фигура 2.1-UML диаграма

### 2.1.4. Анализ на нефункционалните изисквания

Потребителският интерфейс трябва да е лесно четим и организиран. Да бъде използван един вид шрифт, най-много два. Цветовете да са подбрани коректно, в зависимост от темата на уеб приложението. Да има анимации на изображенията в началната страница. Уеб приложението трябва да бъде адаптивно за всички видове устройства.



Нефункционални изисквания към нашето приложение се отнасят до:

- Сигурност
- Потребителски интерфейс
- Качество на софтуера
- Архитектура на приложението
- Технологии и софтуер за разработка

#### **Сигурност на използването на приложението**

Приложението трябва да поддържа надеждна система за автентикация и авторизация на своите потребители. Това означава, че в зависимост от ролята си потребителят има възможност да достъпва само определени страници, за другите ще му бъде отказан достъп.

#### **Изисквания към потребителския интерфейс**

Потребителският интерфейс трябва да бъде лек и интуитивен, с подходящо подбрани цветове и размер на компонентите. Това ще позволи лесно навигиране и ориентиране в сайта. При въвеждане и редактиране на данни, трябва да има контрол на валидността им, за да не се допуска трафик на невалидни данни до сървър. Дизайнът на страниците трябва да е адаптивен и да изглежда добре на различни устройства.

#### **Качество на софтуера**

Приложението трябва да бъде създадено чрез спазване на добрите практики за софтуерна разработка и писане на код. Това означава спазване на SOLID принципите при създаване на кода: класовете и методите да имат единствена отговорност, кодът да бъде „отворен“ за разширение и „затворен“ за модификация, интерфейсите да включват само най-необходимите методи, да има „инжектиране“ на външните за класа зависимости. Кодът да бъде добре форматиран и при необходимост - коментиран.

#### **Архитектура на приложението**

Уеб приложението ще бъде трислойно. Слой за достъп до данните, бизнес слой и презентационният слой ще бъдат обособени като отделни проекти.

#### **Изисквания към софтуера за разработка**

Приложението ще бъде създадено чрез средата за програмиране *Visual Studio Community 2022*, БД ще се управлява от *MS SQL Server 2022*, а за работа с нея ще се използва *SQL Server Management Studio (SSMS) 19.1*

#### **Изисквания към технологиите и езиците за разработка**

- Базов проект за ASP.NET Core включващ MVC - .Net 6.0
- Библиотеки, необходими за проекта:
  - Microsoft.EntityFrameworkCore.SqlServer,
  - Microsoft.AspNetCore.Identity.EntityFrameworkCore
  - Microsoft.EntityFrameworkCore.Tools
  - Microsoft.EntityFrameworkCore.Proxies
  - Microsoft.AspNetCore.Diagnostics.EntityFrameworkCore
  - Microsoft.VisualStudio.Web.CodeGeneration.

C#, SQL, HTML, CSS, Bootstrap, JavaScript и др.

## 2.2. Проектиране

### 2.2.1. Проектиране на база данни

Приложението за продажба на музикални инструменти ще използва БД. Преди да започне разработката, е необходимо да изясним какви таблици ще има в БД и какви ще бъдат връзките между тях.

Базата от данни на приложението ще включва таблиците от системата за идентичност на ASP.NET Core и още 6 таблици.

Таблицата с потребителите, идваща от системата за идентичност на ASP.Net Core, ще бъде разширена със следните колони /полета/:

- FirstName – тип string
- LastName – тип string
- Address – тип string

За всеки продукт ще се съхранява информация в таблица **Products**:

• Id – задължително поле от тип int, идентификационен номер. Ограничения: първичен ключ, уникални стойности

- Name - задължително поле от тип string за име на продукта
- Description – поле от тип string за описание на продукта
- Picture – поле от тип string за URL на снимката на продукта
- CategoryId - задължително поле от тип int. Ограничения: външен ключ към таблица Categories

• BrandId - задължително поле от тип int. Ограничения: външен ключ към таблица Brands

- Price - задължително поле от тип decimal за цена на продукта
- Quantity - задължително поле от тип int за наличност на продукта

За всяка категория ще се съхранява информация в таблица **Categories**:

- Id – задължително поле от тип int, Ограничения: първичен ключ, уникални стойности

- Name - име на категория - задължително поле от тип string

За вид марка ще се използва таблицата **Brand**:

- Id- първичен ключ
- BrandName- задължително поле от тип string в обхвата 2-15 символа
- Products – списък от тип Products

За поръчки ще се използва таблицата **Order**:

- Id- първичен ключ
- OrderDate- задължително поле от тип DateTime за съхранение на датата
- Product – външен ключ към Product
- User – външен ключ към ApplicationUser
- Quantity – тип int
- Price – decimal
- Discount – decimal
- TotalPrice – decimal, като в get метода ще се смята общата цена

За количка ще ползвам таблицата **ShoppingCartItem**:

- Id- ключ
- Product – външен ключ към Product
- Quantity – задължително поле от тип int
- User – външен ключ към ApplicationUser
- Total - тип decimal, където се смята общата цена

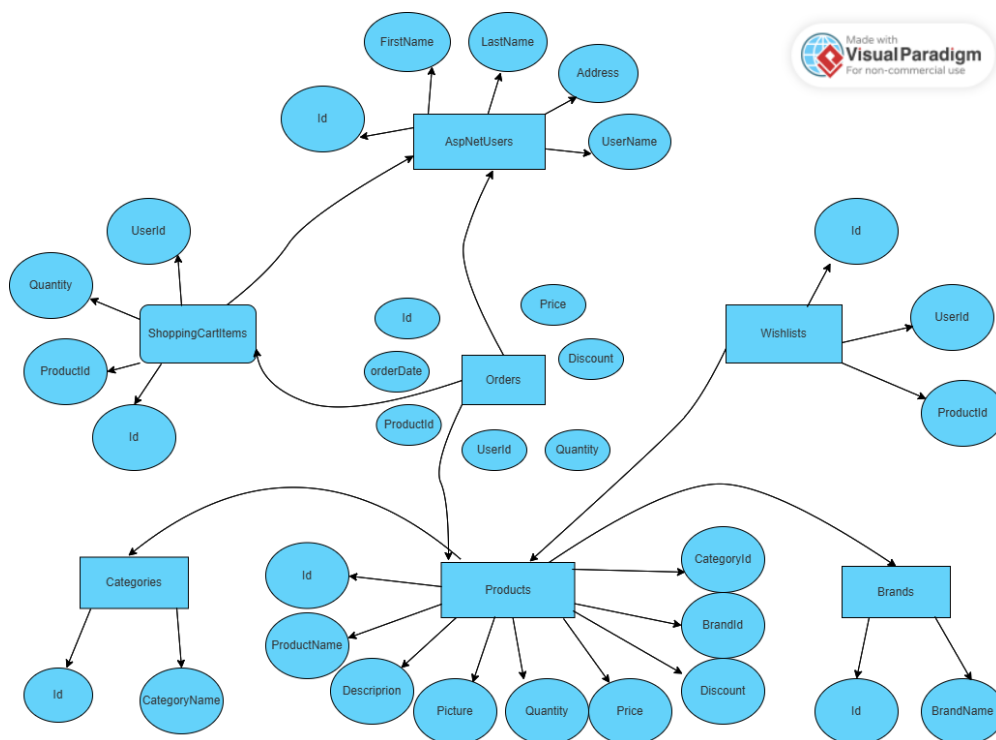
За любими **Wishlist**:

- Id – първичен ключ
- User – външен ключ към ApplicationUser
- Product – външен ключ към Product

Връзките между таблиците са следните:

- Categories - Products – Едно към много, реализирана чрез полето Id на таблица Categories и полето CategoryId на таблицата Products.
- Brands – Products – Едно към много, реализирано чрез полето Id на таблицата Brands и полето BrandId на таблицата Products.

- Orders – Products – Много към едно, реализирана чрез външния ключ ProductId в таблицата Orders.
- ShoppingCartItems – Products – Много към едно, реализирана чрез външния ключ ProductId в таблицата ShoppingCartItems.
- Wishlists – Products – Много към едно, реализирана чрез външния ключ ProductId в таблицата Wishlists.
- ShoppingCartItems – AspNetUsers – Много към едно, реализирана чрез външния ключ UserId в таблицата ShoppingCartItems.
- Wishlists – AspNetUsers – много към едно, реализирана чрез външния ключ UserId в таблицата Wishlists.
- Orders – AspNetUsers – Много към едно, реализирана чрез външния ключ ProductId в таблицата Orders.

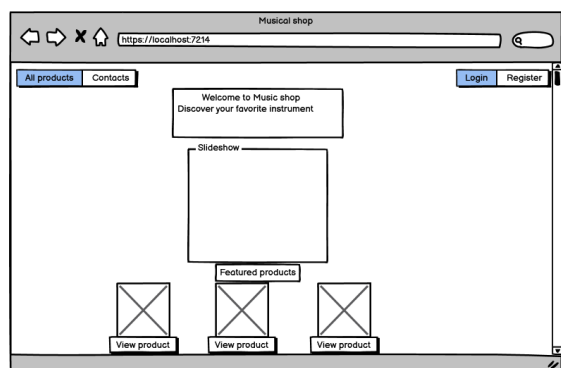


*Прототип на базата данни*

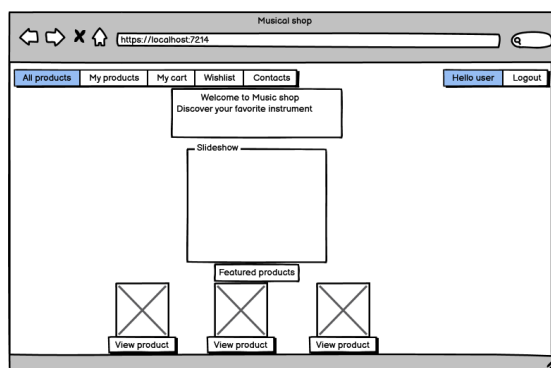
### 2.2.2. Прототипи на потребителския интерфейс

Прототипите дават нагледна представа за визията на софтуерния продукт. Те позволяват на дизайнерите да покажат продукта си, което го прави по-лесен за разбиране. Прототипи могат да се създават по време на всеки етап от процеса по

дизайн, за да помогнат да се демонстрират идеи, които трудно биха се изразили само с думи.

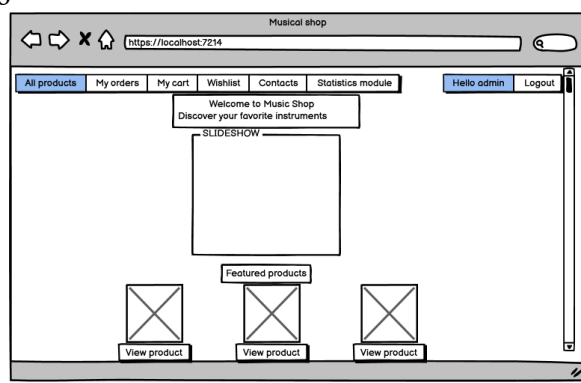


Фиг.2.2 Ноте като гост

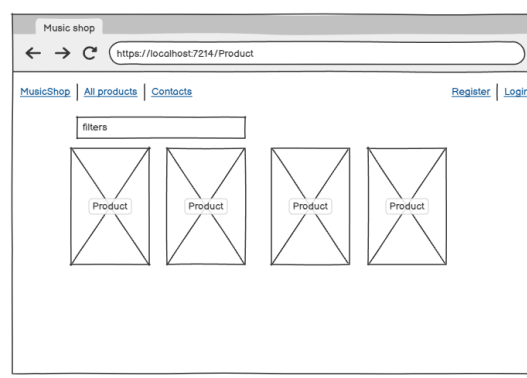


Фиг.2.3 Ноте като потребител

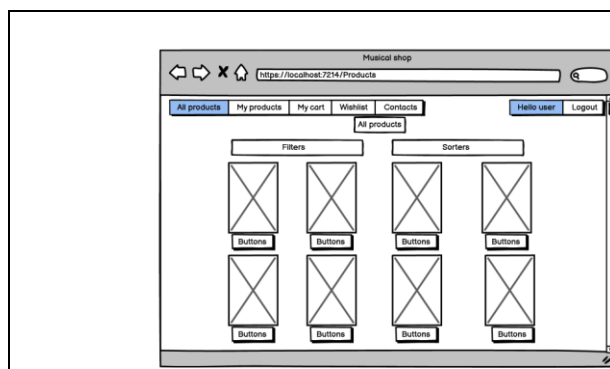
О



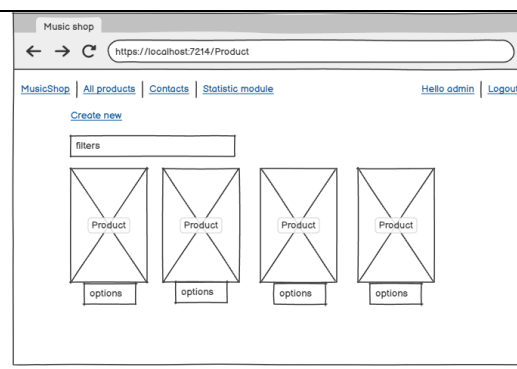
Фиг. 2.3 Ноте като администратор



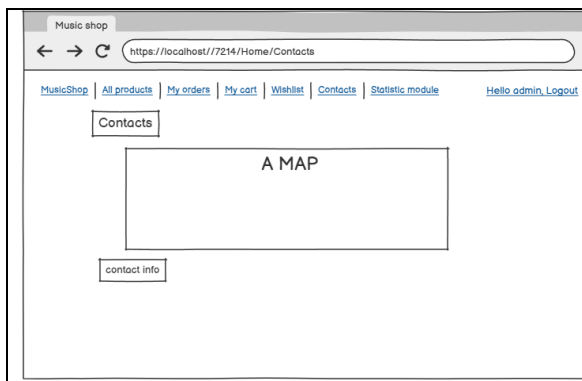
Фиг. 2.4 Всички продукти като гост



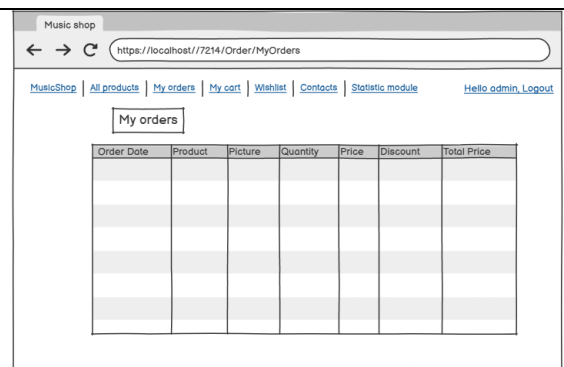
Фиг. 2.5 Всички продукти като потребител



Фиг. 2.6 Всички продукти като администратор



Фиг. 2.6 Контакти като администратор



Фиг. 2.7 My orders като администратор

## 3. ТРЕТА ГЛАВА – РЕАЛИЗАЦИЯ

Реализацията на приложението изисква да се направи избор между двата подхода на изграждане на уеб приложение с използване на БД: Code First и Database First.

### 3.1. Технология на създаване на уеб приложението

За реализация на сайта ще се използва базов проект за ASP.NET Core на езика C#. Базовият проект има вграден MVC модел, който ще бъде разширен за да се изпълнят целите на уеб приложението.

#### Стъпки за изграждане на уеб приложението

##### По-важните стъпки при изграждане на уеб приложението са:

- ✓ Създаване на базов проект от тип ASP.NET Core Това ще бъде презентационният слой на приложението.
- ✓ Добавяне на два нови проекта от тип Class Library. Проектът MusicShop.Infrastructure представлява слой за достъп до БД, а проектът MusicShop.Core – бизнес слой на приложението.
- ✓ За работа с потребители и роли се използва системата за идентичност на ASP.NET, като се прави scaffold /автоматично генериране на код/ и корекции на Register, Login, Logout. Автоматично е създаден администратор с вградени в приложението данни.
- ✓ Тъй като за реализация на приложението е избран подходът Code First, следващата стъпка е създаването на класовете, описващи данните, в папка Data=>Entities на проекта MusicShop.Infrastructure. При първоначалното създаване на БД, автоматично се записват данните за категориите и производителите в съответните таблици.
- ✓ Създаване на система за навигация в сайта.
- ✓ Имплементиране на CRUD операциите за продукт/услуга.
- ✓ Имплементиране на създаване на поръчка и преглед на поръчки.
- ✓ Създаване на количка и възможност да се преглежда
- ✓ Създаване на функция за добавяне в любими и преглед на любими продукти.

### 3.2. Структура на базата от данни

За успешното създаване на базата от данни на приложението са създадени множество класове за описание на данните. Приложението използва следните класове: ApplicationUser, Product, Brand ,Category, Order, Product. ShoppingCartItem, Wishlist.

Класа ApplicationUser наследява IdentityUser, за да ни позволи работата с Identity System на ASP.NET Core. В него имаме полетата:

- FirstName-задължително поле от тип string. Ограничения: максимална дължина 30 символа.
- LastName-задължително поле от тип string. Ограничения: максимална дължина 30 символа.
- Address- задължително поле от тип string. Ограничения: максимална дължина 50 символа.

```
public class ApplicationUser : IdentityUser
{
    [Required]
    [MaxLength(30)]
    5 references
    public string FirstName { get; set; } = null!;

    [Required]
    [MaxLength(30)]
    5 references
    public string LastName { get; set; } = null!;

    [Required]
    [MaxLength(50)]
    5 references
    public string Address { get; set; } = null!;
}
```

Класа Product има външни ключове към Category и Brand. В него имаме полетата:

- Id - първичен ключ.
- ProductName - задължително поле от тип string. Ограничения: максимална дължина 25 символа.
- Description - задължително поле от тип string.
- Picture - задължително поле от тип string. В него се пази снимката на продукта.
- Quantity - задължително поле от тип int. Ако не съответства на обхвата 0-400, ще изписва съобщение за грешка.
- Price - задължително поле от тип decimal.



- Discount – свойство от тип decimal за отстъпка.
- CategoryId – външен ключ към класа Category.
- Category – виртуално навигационно свойство.
- BrandId – външен ключ към класа Brand. Brand – виртуално навигационно свойство.
- Orders – виртуално колекция от класа Order.

```
public class Product
{
    [Key]
    11 references
    public int Id { get; set; }
    [Required]
    [MaxLength(25)]
    13 references
    public string ProductName { get; set; } = null!;
    [Required]
    5 references
    public string Description { get; set; } = null!;
    [Required]
    12 references
    public string Picture { get; set; } = null!;
    [Required]
    [Range(0, 400, ErrorMessage = "Quantity must be between 0 and 400")]
    14 references
    public int Quantity { get; set; }
    [Required]
    [Range(0.01, double.MaxValue, ErrorMessage = "Price must be 0.01 or higher")]
    17 references
    public decimal Price { get; set; }
    12 references
    public decimal Discount { get; set; }

    0 references
    public virtual IEnumerable<Order> Orders { get; set; } = new List<Order>();

    [Required]
    [ForeignKey(nameof(Brand))]
    6 references
    public int BrandId { get; set; }
    9 references
    public virtual Brand Brand { get; set; } = null!;

    [Required]
    [ForeignKey(nameof(Category))]
    6 references
    public int CategoryId { get; set; }
    9 references
    public virtual Category Category { get; set; } = null!;
}
```

Класа Brand има пропъртита за Id – първичен ключ и BrandName – задължително поле с максимална дължина 15 и минимална - 2. Направена е и виртуална колекция с продукти - Products.

```
public class Brand
{
    [Key]
    3 references
    public int Id { get; set; }
    [Required]
    [MinLength(2)]
    [MaxLength(15)]
    13 references
    public string BrandName { get; set; } = null!;
    0 references
    public virtual IEnumerable<Product> Products { get; set; } = new List<Product>();
}
```

Класа Category има виртуална колекция с продукти с продукти - Products. В него са създадени и Id – първичен ключ и CategoryName – задължително поле от тип string с максимална дължина 15 символа и минимална – 2.

```
public class Category
{
    [Key]
    3 references
    public int Id { get; set; }
    [Required]
    [MinLength(2)]
    [MaxLength(15)]
    14 references
    public string CategoryName { get; set; } = null!;
    0 references
    public virtual IEnumerable<Product> Products { get; set; } = new List<Product>();
}
```

В класа Order има:

- Id - първичен ключ.
- OrderDate – задължително поле от тип DateTime.
- ProductId – външен ключ, задължително поле, виртуално навигационно свойство към класа Product.
- UserId – външен ключ, задължително поле, виртуално навигационно свойство към класа ApplicationUser.
- Quantity – поле от тип int за количество.
- Price – от тип decimal.
- Discount – от тип decimal.

- TotalPrice – от тип decimal за смятане на крайната цена с приложена отстъпка.

В класа ShoppingCartItem има:

- Id – от тип int – първичен ключ
- ProductId – външен ключ, задължително поле, виртуално навигационно свойство към класа Product.
- Quantity – задължително поле от тип int.
- UserId – външен ключ, задължително поле, виртуално навигационно свойство към класа ApplicationUser.
- Total – от тип decimal за пресмятане на крайната цена спрямо количеството продукти, цена и отстъпка.

В класа Wishlist има:

- Id – първичен ключ
- UserId – външен ключ, задължително поле, виртуално навигационно свойство към класа ApplicationUser.
- ProductId – външен ключ, задължително поле, виртуално навигационно свойство към класа Product.

**Връзките** на таблиците са:

Orders – Products – много към едно

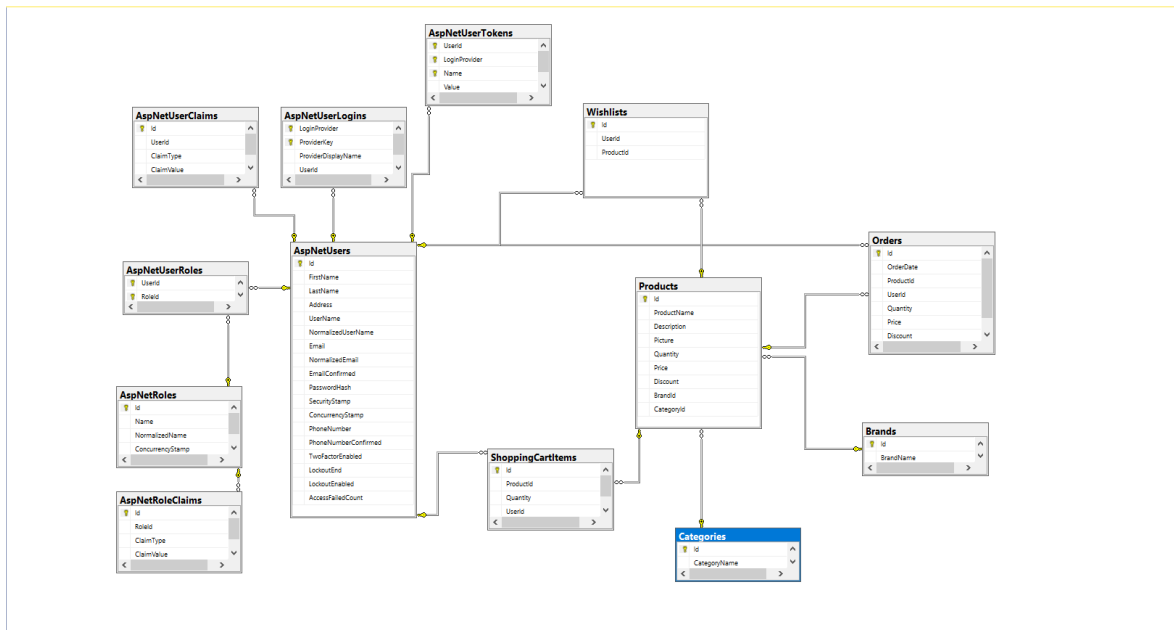
ShoppingCartItems – Products – много към едно

Wishlist – Products – много към едно

ShoppingCartItems –AspNetUsers –много към едно

Wishlists – AspNetUsers – много към едно

Orders – AspNetUsers –много към едно

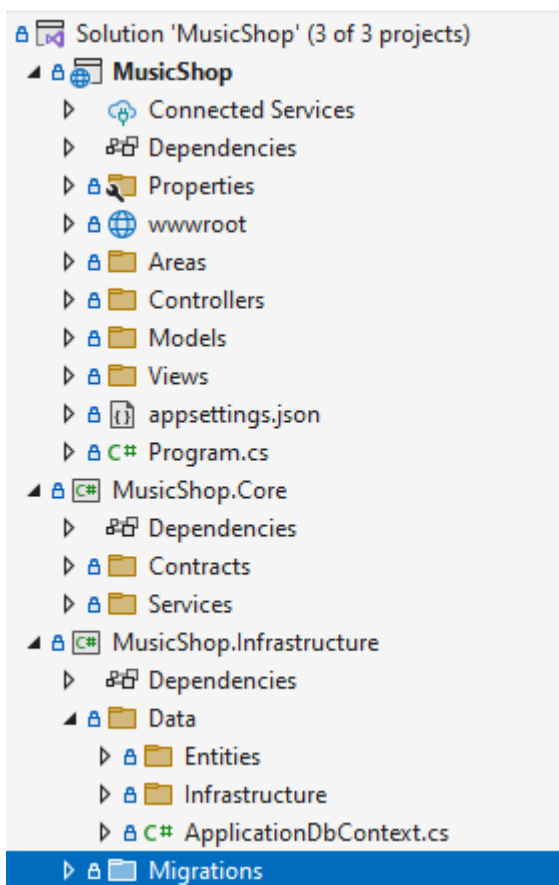


E/R диаграма на базата данни

### 3.3. Реализация на уеб приложението

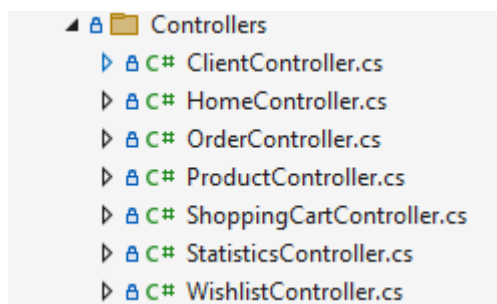
#### 3.3.1. Архитектура на приложението

Архитектура на приложението:



Освен основната архитектура на приложението, допълнително създаваме папка Contracts в която имаме 7 интерфейса: IBrandService, ICategoryService, IOrderService, IProductService, IShoppingCartService, IStatisticsService, IWishlistService, които ги имплементираме в папка Service. В папка Areas се намират скафолднатите елементи. В папка Controllers имаме 7 контролера: ClientController, HomeController, OrderController, ProductController, ShoppingCartController, StatisticsController, WishlistController. В папка Data е класа ApplicationDbContext, който се свързва с базата данни и се създават таблиците. В Entities, както е показано в предната точка, са класовете за базата данни. В Infrastructure се сидват ролите на потребителите, категориите и дизайнерите. В папка Models са всички модели за всеки от класовете. Във Views са всички изгледи за различните страници.

### 3.3.2. Контролери



В контролера **ProductController.cs** имаме action (Index), за създаване (Create), редактиране (Edit), детайл (Details) и изтриване на продукт (Delete). Екшъните Create, Edit, Delete работат както на Get, така и на Post.

За контролера сме сложили атрибут за авторизация, за да може само администратора да има достъп до тях. `[Authorize(Roles = "Administrator")]`

Върху action Index, Details имаме атрибут, който позволява достъпа до тях за всички роли `[AllowAnonymous]`

В контролерите инжектираме сървисите, които ни трябва, и правим конструктор. Всички контролери наследяват базовия клас Controller.

```
public class ProductController : Controller
{
    private readonly IProductService _productService;
    private readonly ICategoryService _categoryService;
    private readonly IBrandService _brandService;
    0 references | nzaksum-marto, 61 days ago | 1 author, 1 change
    public ProductController(IProductService productService, ICategoryService categoryService, IBrandService brandService)
    {
        this._productService = productService;
        this._categoryService = categoryService;
        this._brandService = brandService;
    }
}
```

Достъп до контролера **OrderController** имат само логнатите потребители чрез атрибута `[Authorize]`. Администраторът може да вижда екшъна Index, който зарежда всички поръчки направени от всички потребители, чрез атрибута

`[Authorize(Roles = "Administrator")]`. MyOrders служи за виждане на историята на

всички направени поръчки от клиента, а Create е свързан с OrderCreateVM, който е за потвърждаване на поръчката.

В **ClientController** имаме само екшъни за Index, get и post за Delete и Success, който просто връща view. Инжектиран е IOrderService. Достъп до него има само администраторът чрез `[Authorize(Roles = "Administrator")]`.

В контролера **StatisticsController.cs** инжектираме сървис IStatisticsService. Достъп до контролера има само администратора чрез атрибута: `[Authorize(Roles = "Administrator")]`. В този кантролер имаме само един екшън Index, който ни показва статистическа информация – брой клиенти, брой продукти, брой поръчки и общата сума на всички поръчки.

В **HomeController.cs** само правим един допълнителен екшън – Contacts, който ни връща view за страницата Контакти.

**WishlistController** отговаря за добавянето на продукт към списъка за любими. Инжектирани са IWishlistService и IProductService. Има екшъни: Index-връща списъка с продукти, Add, отговарящ за добавянето и Remove, който премахва продукти (виж прил.1). Достъп до него трябва да имат само логнатите потребители.

**ShoppingCartController** е отговорен за количката. В него са инжектирани сървисите: IShoppingCartService, IOrderService, IProductService. Има екшъни: AddToCart за добавяне на продукт в количката, PlaceOrder - създава поръчка от текущото съдържание на количката и я изчиства, RemoveFromCart – премахва продукт от количката, OrderSuccess, UpdateCart - актуализира количеството на продукт в количката и Index, който показва текущите продукти за потребителя (приложение 2). Достъп до този контролер имат само логнатите потребители благодарение на атрибута `[Authorize]`.

### 3.3.3. Сървиси

За всеки сървис първо е направен интерфейс с описание на методите, които сървисът трябва да имплементира. Интерфейсите се създават в папка Contracts, а сървисите – в папка Services. Във всеки сървис се създава поле \_context от тип AppliactionDbContext и чрез конструктора се инициализира.

- **BrandService**-наследява IBrandService. Използва се в ProductController.
  - GetBrandById(int brandId)-връща марка на база id.
  - GetBrands - връща списък с всички марки от базата данни.
  - GetProductsByBrand(int brandId) - връща всички продукти, свързани с дадена марка.
- **CategoryService** – наследява ICategoryService. Използва се в ProductController.
  - GetCategories- връща списък с всички категории от базата данни

- GetCategoryById(int categoryId) - връща конкретна категория по нейното ID
- GetProductsByCategory(int categoryId) - връща всички продукти, принадлежащи към дадена категория.
- **OrderService** – наследява IOrderService. Използва се в ClientController, OrderController и ShoppingCartController
  - Create - Създава единична поръчка за конкретен продукт и намалява количеството му в склада.
  - GetOrders() – Връща списък с всички поръчки, подредени по дата.
  - GetOrdersByUser(userId) – Връща поръчките на конкретен потребител.
  - CreateOrderFromCart(cartItems) – Създава поръчки от всички продукти в количката и актуализира наличностите им.
- **ProductService** – наследява IProductService. Използва се в HomeController, OrderController, ProductController, ShoppingCartController и ShoppingCartController.
  - Create(...) – създава нов продукт.
  - GetProductById(productId) – връща продукт по дадено ID
  - GetProducts() – връща списък с всички продукти
  - GetProducts(searchCategory, searchBrand) – връща продукти, филтрирани по категория и/или марка
  - Update(...) – актуализира данни за съществуващ продукт
  - RemoveById(productId) – изтрива продукт по дадено ID
- **ShoppingCartService** наследява IShoppingCartService. Използва се в ShoppingCartController.
  - AddToCart – Добавя продукт към количката или увеличава количеството, ако вече съществува
  - ClearCart – Изтрива всички продукти от количката на даден потребител
  - GetShoppingCartByUser – Връща списък с всички артикули в количката на конкретен потребител
  - RemoveFromCart – Премахва конкретен артикул от количката по ID
  - UpdateCart – Актуализира количеството на даден артикул, ако то е валидно и налично в склада
- **StatisticsService** наследява IStatisticsService. Използва се в StatisticsController.

- CountClients() – Връща броя на клиентите (изключвайки администратора)
- CountOrders() – Връща общия брой направени поръчки
- CountProducts() – Връща броя на наличните продукти в магазина
- SumOrders() – Изчислява общата стойност на всички поръчки с включена отстъпка
- **WishlistService** – наследява IWishlistService. Използва се в WishlistController.
  - AddToWishList(userId, productId) – Добавя продукт в списъка с желания на потребителя, ако вече не съществува
  - GetWishListByUserId(userId) – Връща всички продукти в списъка с желания на конкретен потребител
  - RemoveFromWishList(userId, productId) – Премахва конкретен продукт от списъка с желания на потребителя

### 3.3.4. Вю модели

В моя проект съм създал следните ViewModels:

- **BrandPairVM**, който има полетата int Id и string Name.

```
6 references | nzkaksum-marto, 62 days ago | 1 author, 1 change
public class BrandPairVM
{
    4 references | nzkaksum-marto, 62 days ago | 1 author, 1 change
    public int Id { get; set; }
    [Display(Name = "Brand")]
    4 references | nzkaksum-marto, 62 days ago | 1 author, 1 change
    public string Name { get; set; } = null!;
}
```

- **CategoryPairVM**, който има полетата Id и Name.

```
public class CategoryPairVM
{
    public int Id { get; set; }
    [Display(Name = "Category")]
    public string Name { get; set; } = null!;
}
```

- За работа с клиентите съм създал два вю модела- **ClientDeleteVM** и **ClientIndexVM**. И двата имат полетата Id, UserName, FirstName, LastName, Address и Email. Единствената разлика е в ClientIndexVM, където има допълнително булево поле IsAdmin.



```

public class ClientDeleteVM
{
    public string Id { get; set; } = null!;
    [Display(Name = "Username")]

    public string UserName { get; set; } = null!;

    [Display(Name = "First name")]

    public string FirstName { get; set; } = null!;

    [Display(Name = "Last name")]

    public string LastName { get; set; } = null!;

    public string Address { get; set; } = null!;
    3 references | nzkaksum-marto, 47 days ago | 1 author, 1 change
    public string Email { get; set; } = null!;
}

```

- **OrderIndexVM** има следните полета Id, OrderDate, UserId, User, ProductId, Product, Picture, Quantity, Price, Discout, TotalPrice.
- **OrderCreateVM** има полетата: Id, OrderDate,ProductId,ProductName, QuantityInStock, Picture, Quantity, Price, Discount, TotalPrice.
- **ProductIndexVM** с полетата: Id, ProductName, BrandId, BrandName, CategoryId, CategoryName, Picture, Quantity, Price, Discount.

- **ProductEditVM** има полетата:

```
public class ProductEditVM
{
    [Key]
    2 references | nzkaksum-marto, 55 days ago | 1 author, 1 change
    public int Id { get; set; }

    [Required]
    [MaxLength(30)]
    [Display(Name = "Product Name")]
    5 references | nzkaksum-marto, 55 days ago | 1 author, 1 change
    public string ProductName { get; set; } = null!;

    [Required]
    [Display(Name = "Description")]
    5 references | nzkaksum-marto, 55 days ago | 1 author, 1 change
    public string Description { get; set; } = null!;

    [Required]
    [Display(Name = "Brand")]
    5 references | nzkaksum-marto, 55 days ago | 1 author, 1 change
    public int BrandId { get; set; }
    2 references | nzkaksum-marto, 55 days ago | 1 author, 1 change
    public virtual List<BrandPairVM> Brands { get; set; } = new List<BrandPairVM>();

    [Required]
    [Display(Name = "Category")]
    5 references | nzkaksum-marto, 55 days ago | 1 author, 1 change
    public int CategoryId { get; set; }
    2 references | nzkaksum-marto, 55 days ago | 1 author, 1 change
    public virtual List<CategoryPairVM> Categories { get; set; } = new List<CategoryPairVM>();

    [Display(Name = "Picture")]
    5 references | nzkaksum-marto, 55 days ago | 1 author, 1 change
    public string Picture { get; set; } = null!;
    [Range(0, 500)]
    [Display(Name = "Quantity")]
    5 references | nzkaksum-marto, 12 days ago | 1 author, 2 changes
    public int Quantity { get; set; }
    [Range(0, 50000)]
    [Display(Name = "Price")]
    5 references | nzkaksum-marto, 12 days ago | 1 author, 2 changes
    public decimal Price { get; set; }
    [Range(0, 100)]
    [Display(Name = "Discount")]
    5 references | nzkaksum-marto, 12 days ago | 1 author, 2 changes
    public decimal Discount { get; set; }
}
```

- **ProductDetailsVM** има полетата:

```
public class ProductDetailsVM
{
    [Key]
    2 references | nzkaksum-marto, 55 days ago | 1 author, 1 change
    public int Id { get; set; }

    [Display(Name = "Product Name")]
    3 references | nzkaksum-marto, 55 days ago | 1 author, 1 change
    public string ProductName { get; set; } = null!;

    [Display(Name = "Description")]
    3 references | nzkaksum-marto, 55 days ago | 1 author, 1 change
    public string Description { get; set; } = null!;

    1 reference | nzkaksum-marto, 55 days ago | 1 author, 1 change
    public int BrandId { get; set; }
    [Display(Name = "Brand")]
    3 references | nzkaksum-marto, 55 days ago | 1 author, 1 change
    public string BrandName { get; set; } = null!;
    1 reference | nzkaksum-marto, 55 days ago | 1 author, 1 change
    public int CategoryId { get; set; }
    [Display(Name = "Category")]
    3 references | nzkaksum-marto, 55 days ago | 1 author, 1 change
    public string CategoryName { get; set; } = null!;
    [Display(Name = "Picture")]
    3 references | nzkaksum-marto, 55 days ago | 1 author, 1 change
    public string Picture { get; set; } = null!;

    [Display(Name = "Quantity")]
    3 references | nzkaksum-marto, 55 days ago | 1 author, 1 change
    public int Quantity { get; set; }

    [Display(Name = "Price")]
    3 references | nzkaksum-marto, 55 days ago | 1 author, 1 change
    public decimal Price { get; set; }

    [Display(Name = "Discount")]
    3 references | nzkaksum-marto, 55 days ago | 1 author, 1 change
    public decimal Discount { get; set; }
}
```

- **ProductDeleteVM** има полетата:

```
public class ProductDeleteVM
{
    [Key]
    2 references | nzkaksum-marto, 55 days ago | 1 author, 1 change
    public int Id { get; set; }

    [Display(Name = "Product Name")]
    3 references | nzkaksum-marto, 55 days ago | 1 author, 1 change
    public string ProductName { get; set; } = null!;

    [Display(Name = "Description")]
    1 reference | nzkaksum-marto, 55 days ago | 1 author, 1 change
    public string Description { get; set; } = null!;

    1 reference | nzkaksum-marto, 55 days ago | 1 author, 1 change
    public int BrandId { get; set; }
    [Display(Name = "Brand")]
    3 references | nzkaksum-marto, 55 days ago | 1 author, 1 change
    public string BrandName { get; set; } = null!;
    1 reference | nzkaksum-marto, 55 days ago | 1 author, 1 change
    public int CategoryId { get; set; }
    [Display(Name = "Category")]
    3 references | nzkaksum-marto, 55 days ago | 1 author, 1 change
    public string CategoryName { get; set; } = null!;
    [Display(Name = "Picture")]
    3 references | nzkaksum-marto, 55 days ago | 1 author, 1 change
    public string Picture { get; set; } = null!;

    [Display(Name = "Quantity")]
    3 references | nzkaksum-marto, 55 days ago | 1 author, 1 change
    public int Quantity { get; set; }

    [Display(Name = "Price")]
    3 references | nzkaksum-marto, 55 days ago | 1 author, 1 change
    public decimal Price { get; set; }

    [Display(Name = "Discount")]
    3 references | nzkaksum-marto, 55 days ago | 1 author, 1 change
    public decimal Discount { get; set; }
}
```

- **ProductCreateVM** има следните полета:

```

5 references | nzaksum-marto, 12 days ago | 1 author, 4 changes
public class ProductCreateVM
{
    [Key]
    0 references | nzaksum-marto, 62 days ago | 1 author, 1 change
    public int Id { get; set; }
    [Required]
    [MaxLength(30)]
    [Display(Name = "Product Name")]
    4 references | nzaksum-marto, 62 days ago | 1 author, 1 change
    public string ProductName { get; set; } = null!;

    [Required]
    [Display(Name = "Description")]
    4 references | nzaksum-marto, 55 days ago | 1 author, 1 change
    public string Description { get; set; } = null!;

    [Required]
    [Display(Name = "Brand")]
    4 references | nzaksum-marto, 62 days ago | 1 author, 1 change
    public int BrandId { get; set; }
    2 references | nzaksum-marto, 62 days ago | 1 author, 1 change
    public virtual List<BrandPairVM> Brands { get; set; } = new List<BrandPairVM>();

    [Required]
    [Display(Name = "Category")]
    4 references | nzaksum-marto, 62 days ago | 1 author, 1 change
    public int CategoryId { get; set; }
    2 references | nzaksum-marto, 62 days ago | 1 author, 1 change
    public virtual List<CategoryPairVM> Categories { get; set; } = new List<CategoryPairVM>();
    [Display(Name = "Picture")]
    4 references | nzaksum-marto, 62 days ago | 1 author, 1 change
    public string Picture { get; set; } = null!;
    [Range(0, 500)]
    [Display(Name = "Quantity")]
    4 references | nzaksum-marto, 12 days ago | 1 author, 3 changes
    public int Quantity { get; set; }
    [Range(0, 50000)]
    [Display(Name = "Price")]
    4 references | nzaksum-marto, 12 days ago | 1 author, 3 changes
    public decimal Price { get; set; }
    [Range(0, 100)]
    [Display(Name = "Discount")]
    4 references | nzaksum-marto, 12 days ago | 1 author, 3 changes
    public decimal Discount { get; set; }
}

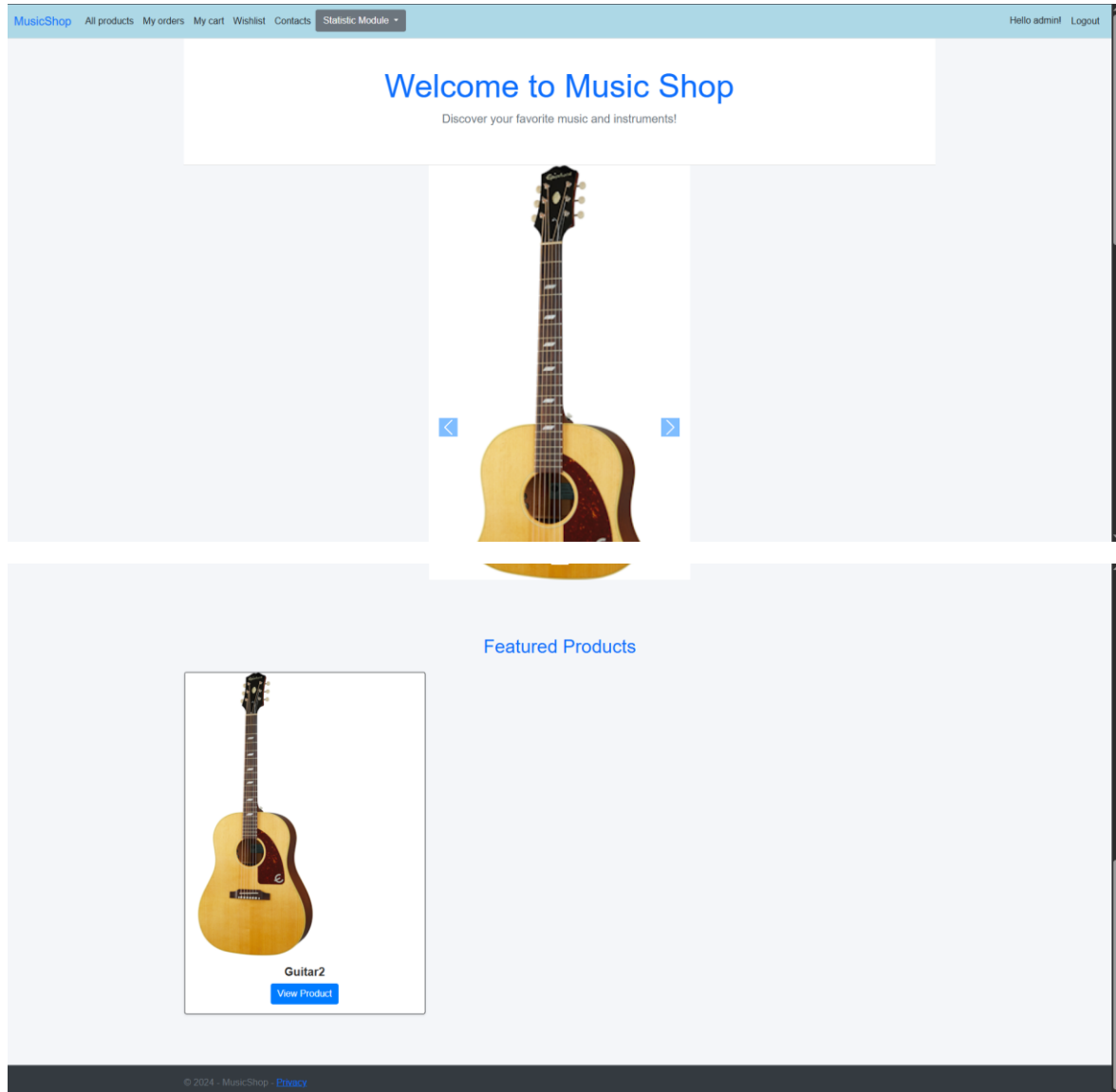
```

- **ShoppingCartIndexVM** има полетата Id, ProductId, ProductName, QuantityInStock, Quantity, Price, Discount UserId и Total, като Total смята общата цена
- **StatisticsVM** има полетата CountClients, CountProducts, CountOrders, SumOrders.
- **WishlistDTO**-вю модела има полетата Id, ProductId ProductName, Picture, Price.

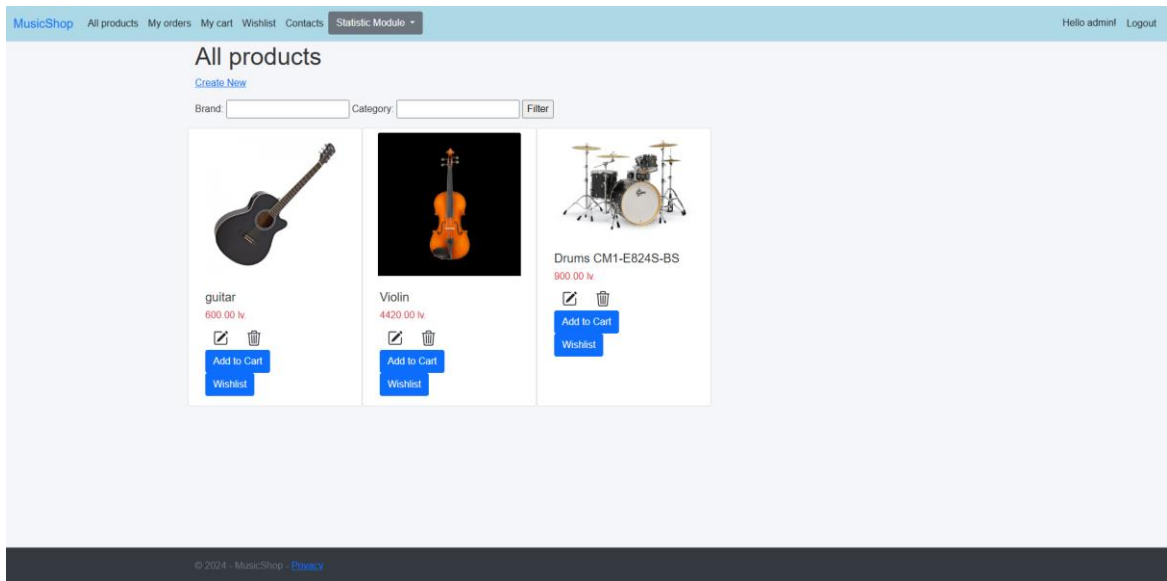
### 3.3.5. Изгледи

В началната страница, намираща се във Views/Home/Index.cshtml има слайдшоу, направено с Bootstrap, а отдолу са всички продукти. Гостите, регистрираните потребители и администратора виждат едно и също с

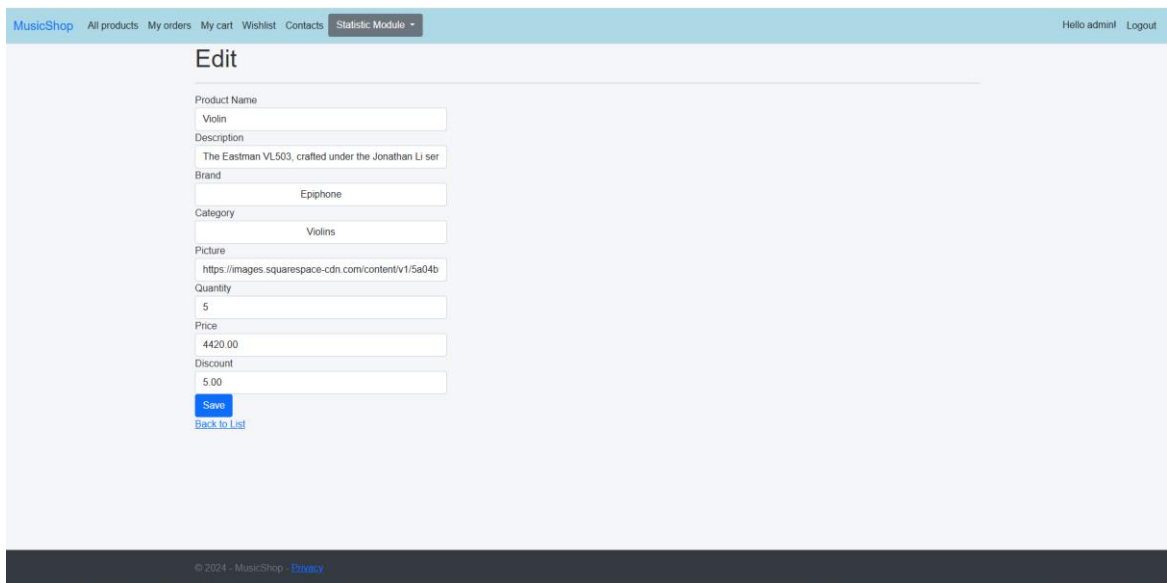
изключение на навигационната лента. Администраторът има достъп до статистическия модул. Гостите могат да виждат само всички продукти и контакти, а регистрираните потребители имат достъп до количка и списък с любими продукти. В началната страница е добавено slideshow, което показва какви продукти се предлагат в сайта.



Изгледът за всички продукти има опция за филтриране по категория и марка. Администраторът вижда бутона за създаване на нов продукт, На отделните продукти той има и възможността да ги изтрива или редактира. Потребителите и гостите могат да виждат всички продукти, но само регистрираните ще могат да го добавят към любими или към количка



Изгледът за редактиране който може само администраторът да го достъпи. Могат да се редактират всички полета. Има бутон за запазване и връщане обратно към всички продукти.



Добавяне на продукт, вю достъпно само за администратора.

MusicShop All products My orders My cart Wishlist Contacts **Statistic Module** Hello admin! Logout

## Create

Product Name

Description

Brand

Category

Picture

Quantity

Price

Discount


[Create](#)  
[Back to List](#)

© 2024 - MusicShop - [Privacy](#)

Вю My orders след направена 1 поръчка. Достъпно е за администраторът и регистрираният потребител.

MusicShop All products My orders My cart Wishlist Contacts **Statistic Module** Hello admin! Logout

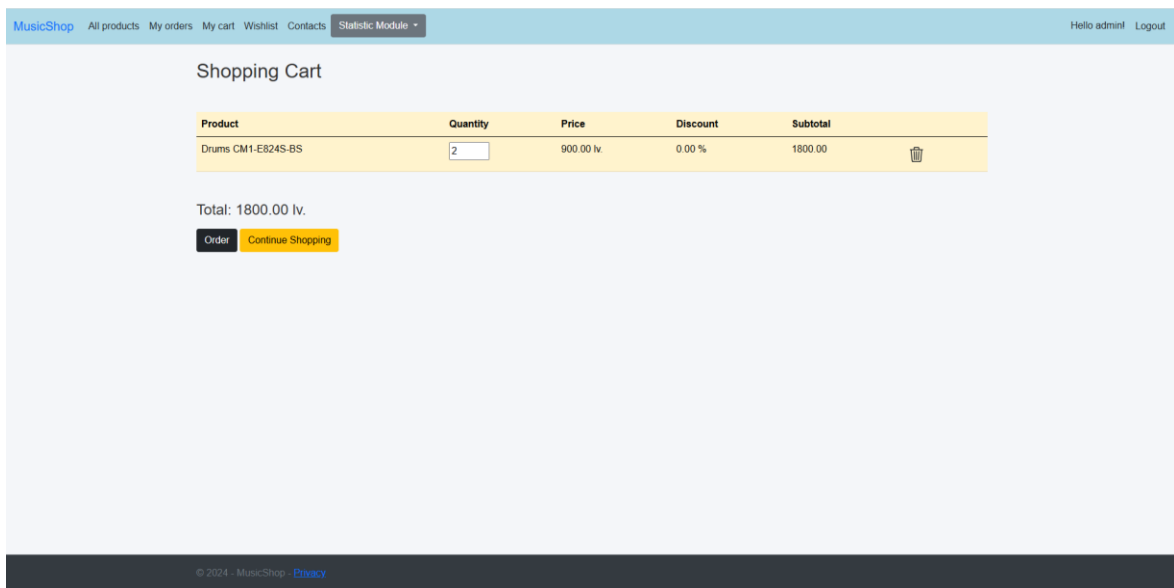
## My orders

OrderDate	Product	Picture	Quantity	Price	Discount	TotalPrice
23-Apr-2025 09:40	Violin		1	4420.00	5.00	4199.00

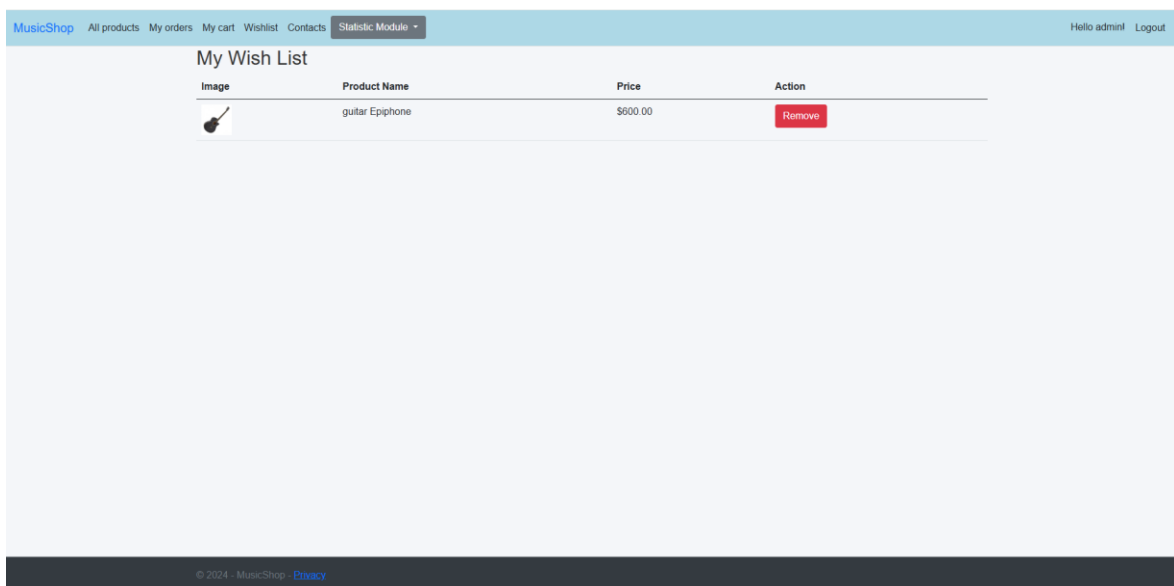
© 2024 - MusicShop - [Privacy](#)

Вю за количка. Достъпно е за регистрираният потребител и администраторът. Количеството може да се задава от тук, като веднага се изчислява общата цена и за 1 вид продукт. Има бутон Continue Shopping, което отвежда към всички продукти. Бутонът Order завършва покупката и изчиства количката. Има и символ кошче за премахване на продукта от количката.

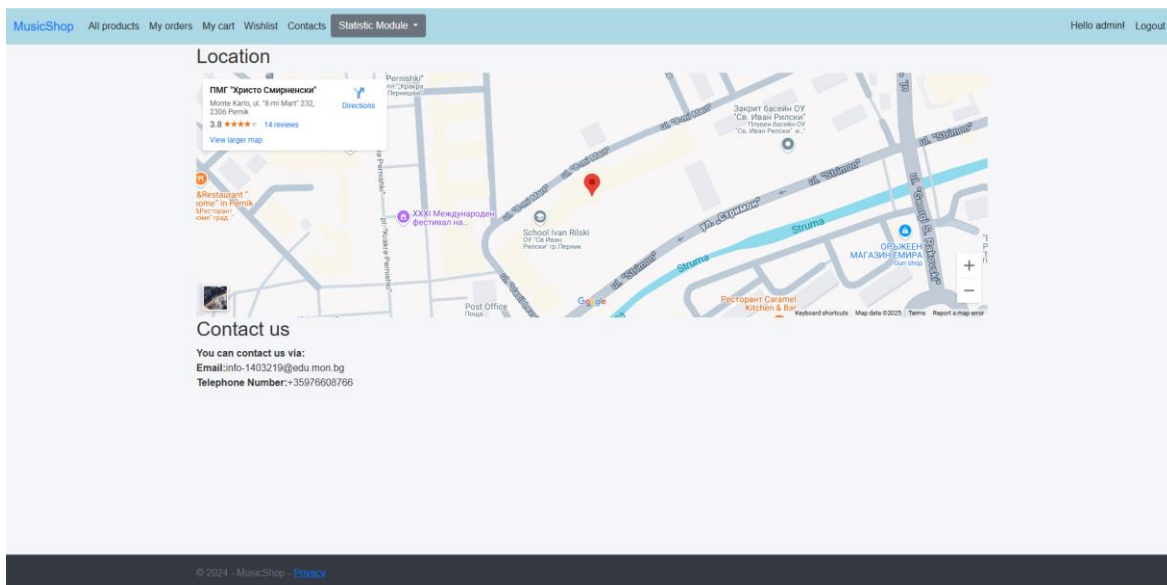




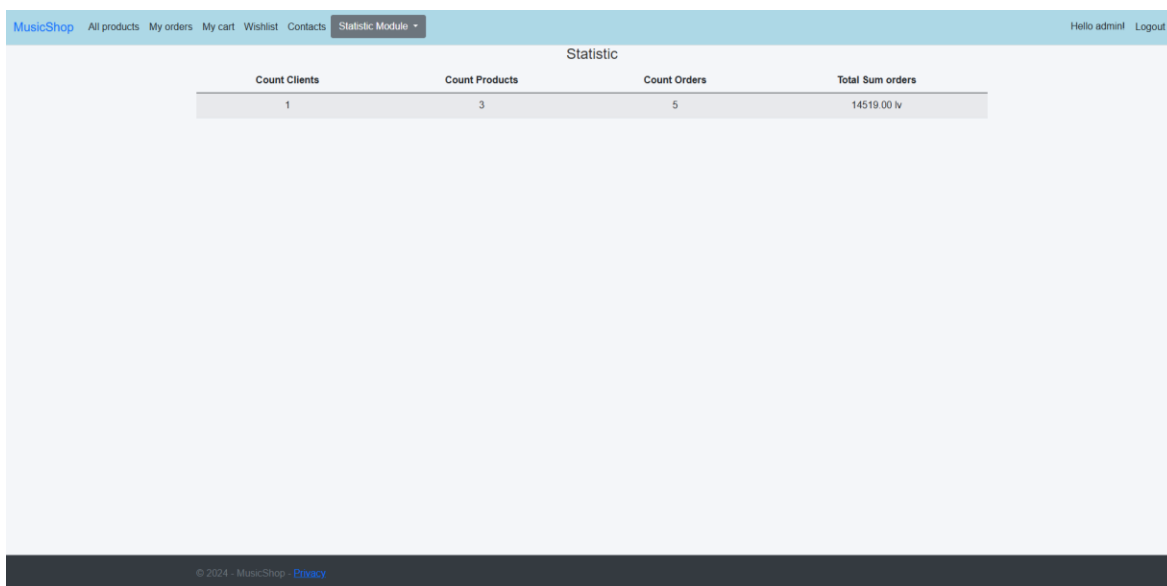
Wishlist view позволява за съхранение на любими продукти. Има бутон за премахване.



Страницата за контакти, достъпна за всеки. Вградена е карта.



Статистическият модул е достъпен само за администратора. Там той има отдални вюта за клиенти и възможността да ги изтрива, списък със всички поръчки, и други статистики



### 3.3.6. Други технологии и функционалности

Както по-горе споменах в началната страница съм направил слайдшоу, на което се представят продуктите в сайта.

Добавена е функционалност за количка, където клиентите могат да добавят и премахват продукти и накрая да ги поръчат.

Допълнително е реализиран и списък с любими, където клиентът може да добавя и премахва продукти за свои любими, които се запазват.

## ЗАКЛЮЧЕНИЕ

Основната задача за тази дипломна работа е да се разработи уеб приложение на ASP.NET Core MVC модел на езика C#. За реализирането на настоящия сайт са използвани езиците HTML, CSS, JavaScript, Razor, както и Bootstrap.

Поставената цел на дипломната работа за разработка на уеб приложението е изпълнена след като преминах през трите основни етапи на изпълнение – проучване, проектиране и реализация.

В началото на дипломната работа бяха поставени изисквания, на които трябва да отговаря приложението, които смятам, че постигнах с крайната версия на този продукт. В сравнение с останалите подобни сайтове в Интернет пространството, може да се каже, че изготвеният сайт се нарежда на едно от първите места по атрактивност, функционалност и лесно навигиране. Уеб приложението има възможност за по-голямо развитие и усъвършенстване за в бъдеще. Една от допълнителните функционалности, които могат да се включи са: вход и регистрация с Google акаунт.

## ИЗПОЛЗВАНА ЛИТЕРАТУРА

1. //Колсиченко, Д. Адаптивен уеб дизайн с Bootstrap, Асеновци, 2017
2. //Наков, С. и колектив. Принципи на програмирането със C#. Фабер, 2018
3. //Наков, С. и колектив. Програмиране за .NET Framework. БАРС, София, 2006
4. //Свободно учебно съдържание по програмиране и ИТ за българските училища на Работна група „Образование по програмиране и ИТ“ – <https://github.com/BG-IT-Edu>
5. <https://survey.stackoverflow.co/2024/technology>
6. <https://note.bg/kakvi-vidove-ezici-za-programirane-sa-populyarni-prez-2023-g>
7. <https://blog.superhosting.bg/5-most-popular-programming-languages-in-web-development.html>
8. <https://www.techbulgaria.com/?p=1701>
9. <https://developer.mozilla.org/en-US/docs/Web/HTML>
10. <https://bg.wikipedia.org/wiki/Laravel>
11. // <https://dotnet.microsoft.com/en-us/learn/aspnet/what-is-aspnet>
12. // <https://www.w3schools.com/js/default.asp>

# ПРИЛОЖЕНИЯ

## Приложение 1:

```
public class WishListController : Controller
{
    private readonly IWishlistService _wishlistService;
    private readonly IProductService _productService;

    0 references
    public WishListController(IWishlistService wishlistService, IProductService productService)
    {
        _wishlistService = wishlistService;
        _productService = productService;
    }

    1 reference
    public IActionResult Index()
    {
        var currentUserId = this.User.FindFirstValue(ClaimTypes.NameIdentifier);
        var wishlist = _wishlistService.GetWishlistByUserId(currentUserId);

        var model = wishlist.Select(w => new WishlistDTO
        {
            Id = w.Id,
            ProductId = w.Product.Id,
            ProductName = w.Product.ProductName,
            Picture = w.Product.Picture,
            Price = w.Product.Price
        }).ToList();

        return View(model);
    }

    [HttpPost]
    0 references
    public IActionResult Add(int productId)
    {
        var userId = User.FindFirstValue(ClaimTypes.NameIdentifier);
        if (userId == null)
        {
            return RedirectToAction("Login", "Account");
        }

        _wishlistService.AddToWishlist(userId, productId);
        return RedirectToAction("Index");
    }

    0 references
    public IActionResult Remove(int productId)
    {
        var currentUserId = this.User.FindFirstValue(ClaimTypes.NameIdentifier);
        _wishlistService.RemoveFromWishlist(currentUserId, productId);
        if (currentUserId == null) return NotFound();
        return RedirectToAction(nameof(Index));
    }
}
```

## Приложение 2:

```
public class ShoppingCartController : Controller
{
    private readonly IShoppingCartService _cartService;
    private readonly IOrderService _orderService;
    private readonly IProductService _productService;

    0 references
    public ShoppingCartController(IShoppingCartService cartService, IOrderService orderService, IProductService productService)
    {
        _cartService = cartService;
        _orderService = orderService;
        _productService = productService;
    }

    [HttpPost]
    0 references
    public ActionResult AddToCart(int productId, int quantity)
    {
        string currentUserId = this.User.FindFirstValue(ClaimTypes.NameIdentifier);
        _cartService.AddToCart(productId, currentUserId, quantity);
        return RedirectToAction("Index");
    }

    0 references
    public ActionResult PlaceOrder()
    {
        string currentUserId = this.User.FindFirstValue(ClaimTypes.NameIdentifier);
        var cartItems = _cartService.GetShoppingCartByUser(currentUserId);

        // Create order from cart items
        _orderService.CreateOrderFromCart(cartItems);
        // Clear the shopping cart
        _cartService.ClearCart(currentUserId);
        // Redirect to order success page or home page
        return RedirectToAction("OrderSuccess");
    }
}
```

---

```

[HttpPost]
0 references
public ActionResult RemoveFromCart(int cartItemId)
{
    _cartService.RemoveFromCart(cartItemId);

    return RedirectToAction("Index");
}
0 references
public ActionResult OrderSuccess()
{
    return View();
}
[HttpPost]
0 references
public ActionResult UpdateCart(int cartItemId, int quantity)
{
    try
    {
        _cartService.UpdateCart(cartItemId, quantity);
        return RedirectToAction("Index");
    }
    catch (ArgumentException ex)
    {
        ModelState.AddModelError(string.Empty, ex.Message);
    }
    catch (InvalidOperationException ex)
    {
        ModelState.AddModelError(string.Empty, ex.Message);
    }
    catch (Exception ex)
    {
        ModelState.AddModelError(string.Empty, "Error updating cart.");
    }
    string currentUserId = this.User.FindFirstValue(ClaimTypes.NameIdentifier);
    var cartItems = _cartService.GetShoppingCartByUser(currentUserId);
    return View("Index", cartItems);
}

```

```
public ActionResult Index()
{
    string currentUserId = this.User.FindFirstValue(ClaimTypes.NameIdentifier);

    List<ShoppingCartItem> cartItems = _cartService.GetShoppingCartByUser(currentUserId)
        .Select(item => new ShoppingCartItem
        {
            Id = item.Id,
            ProductName = item.Product.ProductName,
            Quantity = item.Quantity,
            QuantityInStock = item.Product.Quantity,
            Price = item.Product.Price,
            Discount = item.Product.Discount,
            UserId = currentUserId,
        })
        .ToList();
    return View(cartItems);
}
```