

МИНИСТЕРСТВО НА ОБРАЗОВАНИЕТО И НАУКАТА

**ПРОФЕСИОНАЛНА ГИМНАЗИЯ “ГЕН. ВЛАДИМИР ЗАИМОВ” гр. СОПОТ**

4330 гр. Сопот, ул. ”Иван Вазов” №1, тел./факс: /03134/ 83-31, 83-32, e-mail: [pgzaimov@yahoo.com](mailto:pgzaimov@yahoo.com)

**ДИПЛОМЕН**

**ПРОЕКТ**

**Тема: РАЗРАБОТКА НА УЕБ Сайт**

**Магазин за обувки “Comfort”**

*Ученик: Никол Димитрова Кралимаркова*

***Професия:*** *код 481030 „Приложен програмист“*

***Специалност:*** *код 4810301 „Приложно програмиране“*

***Консултант:*** *Тоня Белезирева*

Сопот, 2023 г.

**Съдържание**

**Увод**

**Kратко опсание на тема и областта на дипломнната работа**

**Темата на дипломната работа е разработване на уеб сайт за магазин за обувки, с фокус върху комфорта. Областта на проекта обхваща изграждане на функционалността на уеб платформата, която включва удобно търсене и филтриране на обувките, подробни описания и снимки, удобно пазаруване с възможност за онлайн поръчка и плащане. Важен аспект е и създаването на приятен и интуитивен потребителски интерфейс, който подчертава важността на комфорта при избора на обувки. Дипломната работа ще се фокусира и върху анализа на потребителското поведение, маркетингови стратегии и възможности за усъвършенстване на уеб сайта, за да се увеличи удовлетворението на клиентите и успешно представяне на магазина на пазара.**

**Цели и задачи**

Магазин за обувки „Comfort“ се нуждае от уеб сайт, чрез който да разшири дейността си, давайки възможност за продажба на своите артикули на клиентите, пазаруващи в интернет .

Сайтът трябва да предлага възможност за разглеждане на обувки, чехли и сандали, боти и ботуши и да поддържа подробна информация за всеки артикул. Трябва да се осигури наличието на пазарска кошница, в която регистриран клиент добавя избрани от него артикули в исканото количество, като по този начин формира поръчка.

Сайтът трябва да разполага със следните публично достъпни секции „Начало“, „Мъжки обувки“, „Дамски обувки“, „Детски обувки“, „ПРОМОЦИИ“, „Контакти“.

Трябва да се предостави възможност за администрира с администраторски права за добавяне на нови артикули, промяна и изтриване на съществуващи такива и промяна на някоя от другите страници.

Дизайнът на Сайта трябва да е адаптивен и да изглежда еднакво добре на различни устройства. Изберете аргументирано подходящи технологии.

**ГЛАВА 1.**

**ПРОУЧВАНЕ ПО TEMATA**

* 1. **Предпоставка за създаване на продукта**

Предпоставките за изработване на уеб сайта могат да бъдат следните:

1. Подобряване на маркетинга и продажбите на магазина. Увеличава се популярността на магазина сред обществото и се стимулират продажбите.
2. Предпоставка за изграждане на привлекателен уеб сайт на магазина с акцент върху комфорта.
3. Осигурява са удоволетворяващо потребителски изживяване, чрез лесно навигиране из сайта, ясна информация и удобни опции за пазаруване.
4. Предоставяне на разнообразие от комфортни обувки покриващи различни стилове и потребности.
5. Възможност за въвеждане на функционалност за онлайн поръчки и плащания, улесняващи процеса на пазаруване на клиентите.
   1. **Съществуващи решения и реализации**

Съществуват много решения и реализации за уеб базирани приложения за обувки.

Тези приложения могат да бъдат разработени с различни технологии и функционалности, в зависимост от конкретните нужди на бизнеса и потребителите. Ето някои от възможните решения и функционалности:

* Уеб магазин за обувки за конфорт: Разработването на уеб магазин е едно от най-очевидните решения. Той може да включва каталог с различни модели обувки за конфорт, възможност за търсене и филтриране, добавяне на продукти в кошницата, онлайн плащания и т.н.
* Персонализирани препоръки за продукти: Интелигентни алгоритми могат да се използват, за да предложат персонализирани препоръки за обувки на базата на предпочитанията и предишните покупки на потребителя.
* Отзиви и прегледи на продукти: Даването на възможност на потребителите да споделят своите мнения и отзиви за различните обувки може да бъде полезно за останалите потребители при избора на продукт.
* Интеграция с социални медии: Добавянето на социални елементи към приложението, като например споделяне на продукти в социалните мрежи или интеграция с Instagram за визуално представяне на обувките, може да повиши ангажираността на потребителите.
* Интелигентно управление на запасите: Система за управление на запасите, която автоматизира процесите за управление на запасите, предотвратява прекомерно наличие или липса на определени продукти.

Тези са само някои от възможните решения и функционалности за уеб базирани приложения за обувки за конфорт. Важно е да се проведе подробен анализ на изискванията и потребностите на бизнеса и потребителите, преди да се започне разработката на приложението.

След направено проучване на конкурентните сайтове, стигнах до заключение, че вече има много сайтове за продажба на обувки. Едни от най-посещаваните сайтове за онлай продажба на обувки са:. BuzzSneakers, Footshop, FootCourt и Nike.

На диаграмата на фиг.1 е показано процентното съотношение на посещаемостта на сайтовете. От което става ясно, че най-много се посещава сайта на Nike.

**Фиг.1**

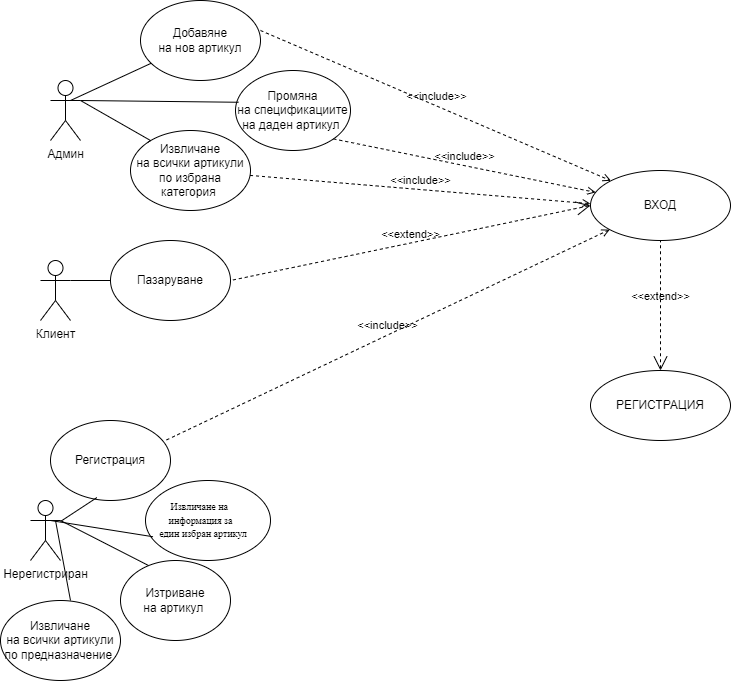
**ГЛАВА 2.**

**ПРОЕКТИРАНЕ НА СТРУКТУРАТА НА WEB БАЗИРАн САЙТ ОРГАНИЗИРАЩ ДЕЙНОСТТА НА МАГАЗИН ЗА ОБУВКИ.**

**2.1. Функционални изисквания към WEB базиран сайт, организиращ дейността на магазин за обувки Comfort /**условието назадачата**/**

Да се реализира уеб приложение за обслужване на **магазин за обувки „Comfort“**. Приложението трябва да поддържа следните функционалности:

* Да се направят роли за работа с приложението:
  + АДМИНИСТРАТОР
  + Клиент
* Добавяне на нов артикул: /с Администраторски функции/:
  + Име / модел
  + Категория /обувки, сандали, боти и ботуши, кецове и маратонки и др./
  + Размер
  + Описание – кратък текст – описание на качеството, цвят, материалите
  + Предназначение – по пол /детско, мъжко, дамско/
  + Снимка
  + Цена на артикул
  + Дата на последна актуализация в системата
* Извличане на информация за един избран артикул
* Изтриване на артикул /с Администраторски функции/
* Промяна на спецификациите на даден артикул /с Администраторски функции/
* Извличане на всички артикули по избрана категория /обувки, сандали, боти и ботуши, кецове и маратонки и др./
* Извличане на всички артикули по предназначение /мъжко, дамско, детско/
* Регистрация на клиент:
  + Име
  + Фамилия
  + Електронна поща
  + Телефон за контакт
* Пазаруване /записване на артикули в пазарска кошница/:
  + Номер на клиент
  + Номер на артикул;
  + Количество стока /брой артикули/
  + Дата / час на направения избор;

**2.2. Изготвяне на Use Cases Diagrams – описание на отделните на случаи на употреба**

**Случай на употреба:** Добавяне на нов артикул

**Цел:** Задейства се, когато възникне нужда от обновяване на налични артикули

**Категория:** основна

**Предусловия:** вход в системата

**Успешни постусловия:** успешно добавяне на нов артикул в системата

**Актьори:** администратор

**Случай на употреба:** Регистрация в системата

**Цел:** Задейства се, когато потребител желае да използва пълната функционалност на приложението

**Категория:** основна

**Предусловия:** отваряне на приложението

**Успешни постусловия:** успешно регистриране в системата

**Актьори:** клиент

**Случай на употреба:** Вход в системата

**Цел:** Задейства се, когато даден потребител направи опит за действие, което е ограничено. Тогава потребителят е подканен да въведе потребителското си име и парола, за да продължи.

**Категория:** основна

**Предусловия:** регистрация в системата

**Успешни постусловия:** успешно влизане в системата

**Неуспешни постусловия:** грешно потребителско име или парола или несъществуващ акаунт

**Актьори:** администратор и клиент

**Случай на употреба:** Изтриване на продукт

**Цел:** Задейства се, когато администратора реши да изтрие, добавен от него, продукт.

**Категория:** основна

**Предусловия:** регистрация в системата

**Успешни постусловия:** успешно изтриване на даден продукт

**Актьори:** администратор

**Случай на употреба:** Промяна на спецификациите на артикул

**Цел:** Задейства се, когато администратор иска да редактира ,добавен от него, продукт.

**Категория:** основна

**Предусловия:** регистрация в системата

**Успешни постусловия:** успешно редактиране на продукт

**Актьори:** администратор

**Случай на употреба:** Пазаруване

**Цел:** Задейства се, когато даден потребител реши да пазарува продукт от системата.

**Категория:** основна

**Предусловия:** регистрация в системата

**Успешни постусловия:** успешно пазаруване на потребителя

**Актьори:** клиент

**Случай на употреба:** Извличане на информация

**Цел:** Задейства се, когато даден потребител иска да открие повече информация за даден продукт

**Категория:** основна

**Предусловия:** регистрация в системата

**Успешни постусловия:** успешно влизане в системата

**Неуспешни постусловия:** грешно потребителско име или парола или несъществуващ акаунт

**Актьори:**  нерегистриран клиент

**2.3. Прототип на потребителският интерфейс /**менюта, екрани, Views**/**

**2.3.1** Начална страница на уебсайта. Съдържа бутони водещи към другите

страници на сайта и бутони за регистрация.

Фиг.

****

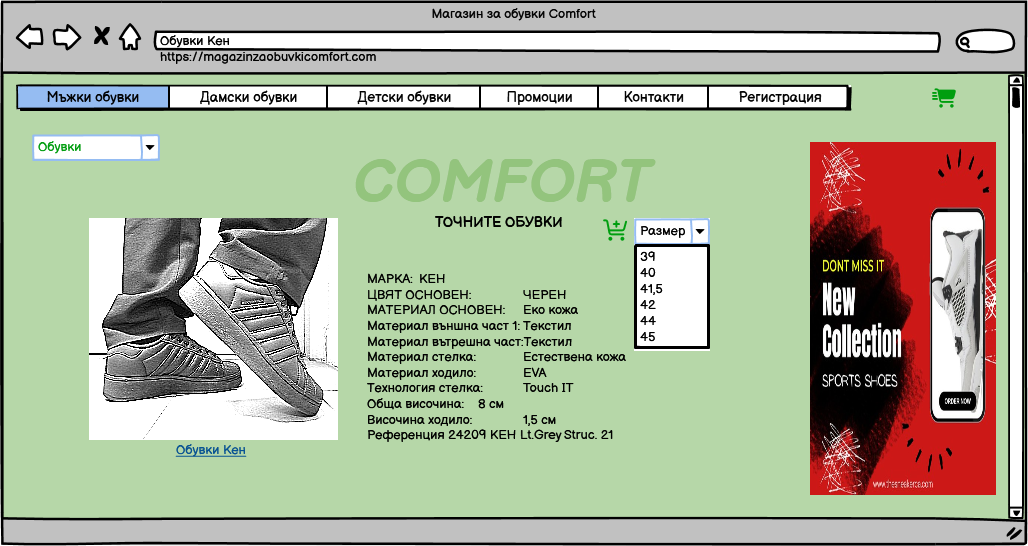
**2.3.2.** Уебстраница, показваща една от категориите за обувки, а именно мъжките. Има филтри, които ни помагат да изберем правилната обувка за нас.

Фиг.

****

**2.3.3.** Тук можем да видим по-обстойно един вид от мъжките обувки в сайта. Има описание за самата обувка и падащо меню с наличните размери. Има и бутон за добавяне в количката.

Фиг.

****

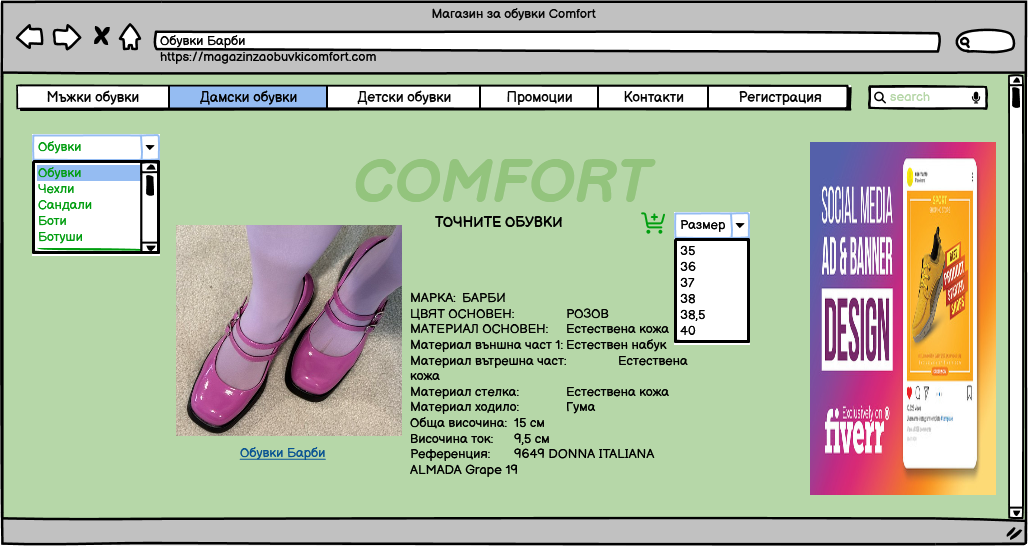
**2.3.4.** Тук виждаме категорията с дамските обувки. В ея има падащо меню за филтриране, за да е по-лесено намирането на желаната обувка.

Фиг.

****

**2.3.5.** На тази страница се намира един от видовете дамски обувки. Както при другите, тук има падащо меню с наличните размери и описание на продукта.

Фиг.

****

**2.3.6.** Това е страницата с промоции. Тук се намират всички намалени артикули за месеца или сезона.

Фиг.

****

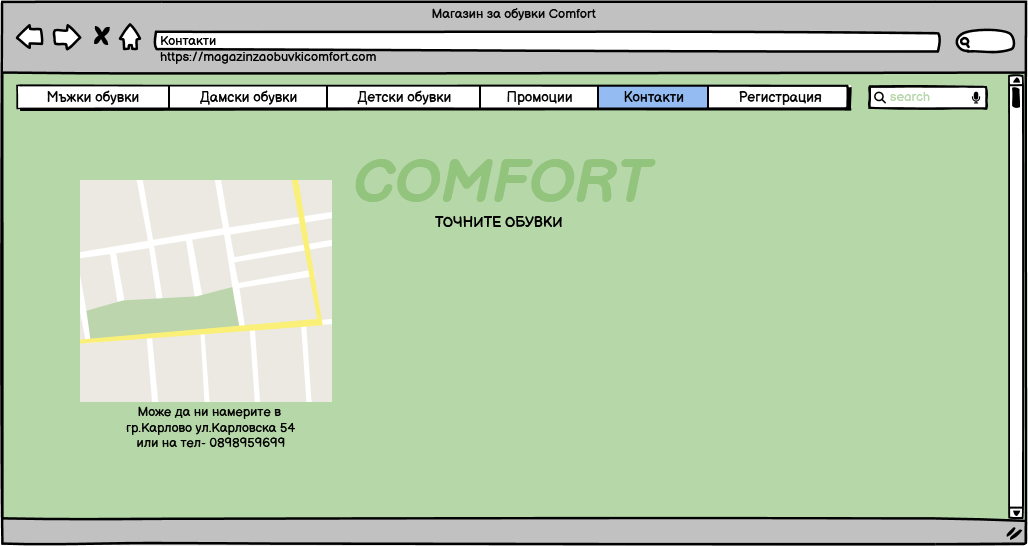
**2.3.7.** Регистрация на нов клиен. Със задължителни полета изикващи Address,и Password (парола).

Фиг.

****

**2.3.8.** На последнат уебстраница намираме точната локация на физическият магазин и телефонен номер за връзка с лицата намиращи се в магазина.

Фиг.

****

**2.4. Описание на избраната технология и софтуерните средства за разработка на приложението и основните алгоритми/**технологии: *ASP.Net, ASP.Net Core, SQL Server, средства: C#, HTML, CSS, JavaScript*, *Bootstrap***/**

ASP.NET представлява технология за разработка на мощни web приложения базирани на .NET платформата на Майкрософт, предлагайки някои съществени предимства в сравнение с предишните модели за разработка на web приложения. Създаваните с ASP.NET приложения представляват код, който се извършва на един уеб сървър. За разлика от така познатите ни съвременни, по-интелигентни технологии, ASP.NET е в състояние да се възползва от предимствата на ранното свързване, така нареченото just-in-time компилиране, оптимизация на кеширане (на различни нива) и др. По този начин получавате висока производителност преди да сте написали и ред програмен код. ASP.NET приложения могат да се разработват дори на Notepad, но истинско повишаване на производителността се наблюдава при използването на мощния инструментариум и средства за дизайн на Visual Studio. Потребителския интерфейс се осъществява в WYSIWYG (What You See Is What You Get) среда, като добавянето на контроли става посредством drag-and-drop. Налице е опция за автоматично разгръщане (deployment) и още много интересни характеристики.ASP.NET приложението има много характеристики които му помагат да е с всяка изминала години стабилна платформа. Нека започнем с мощността и гъвкавостта. Тази функция позволява на разработчеците да изберът и да работят с един език за програмиране по техен избор, който трябва да бъде подходящ за да може да се разработват проекти или приложения които са на .NET. Следващата функционалност на ASP .NET приложението е лесното конфигуриране на данни, тази функция облекчава изпълнението на всевъзможни задачи от прости форми до пълната конфигурация на самият сайт.  Благдарение на което ASP.NET приложението позволява създаването на едни приложения, при които ясно може да се види разграничаването на приложенчестата логика от презентационният код, като обработката на събитията и създаването на един потребителси интерфейс. Нека да продължим със следващата функция на ASP.NET приложението, която е много по специална от другите изброени. Дадената архитектура на приложението позволява на разработчеците да разширяват възможностите, които се предоставени от средата. Девелапърът даже може дори да замени някои от субкомпонентнитте на даденото приложение със разработените вече от него модули. ASP.NET е проектирано да бъде мащабируемо, тоест с по-голям обем, като предоставя възможностите за подобряването и обновяването на произжодителността на клъстерната или многопроцесорната система. Тези процеси с внимателно следени и управлявани по време на тяхното изпълнение, което позволява да бъде създаден нов процес ако даденият се държи неадекватно, което води до положителната позиция към това вашето приложение винаги да бъде на ваше разположение.

***Предимства на ASP.NET:***

-ASP.NET кодът е лесен за преглед и проверка. То позволява не само да се пише много по-лесно,  но също така улеснява девелапърите в процес на качествен контрол.

-Подобрява мащаба, сигурността и надежността на приложенията, също така и предлага по-качествената поддръжка на уеб браузърите

**2.4.2 ASP.NET Core MVC**

ASP.NET Core MVC предоставя следните функции за изграждане на уеб API и уеб         приложения:

- Използва модела на дизайна на View-Controller (MVC)

- Лек, с отворен код, тестируем, добър инструментариум

- RESTful услуги с ASP.NET Core Web API

- Вградена поддръжка за множество формати на данни

- Постигнат висококачествен архитектурен дизайн, оптимизирайки работата на девелапърите

- Конвенция за конфигуриране

- Обвързването на модела автоматично картографира данни от HTTP заявки

- Проверка на модела с валидиране от страна на клиента и от страна на сървъра

 заедно с тези ASP.NET Core MVC предоставя функции като:

-Routing

-Инжектиране на зависимостта – DI

-Силно типизирани изгледи с двигателя Razor

-Помощниците на маркери активират код от страна на сървъра в HTML елементи

-Частични изгледи и преглед на компоненти

-Филтри, области, Middlewares

-Вградени функции за защита

-Идентичност с потребители, роли и външни доставчици

**2.4.3 SQL Server**

Microsoft SQL Server е система за управление на релационни бази данни, разработена от Microsoft. В тази си роля (на система за управление) SQL Server има основната функция да съхранява и извлича данни, по заявки на други софтуерни приложения, които могат да вървят на същия или друг компютър в дадена мрежа (*в т.ч. и Интернет*). MS SQL Server има над дузина различни издания, които са подходящи за различни аудитории и различни натоварвания: от приложения за отделни персонални компютри до масивни приложения работещи постояно в онлайн режим и обхващащи огромно количество устройства. MS SQL Server се заражда преди малко повече от 30 години, на 12-ти юни 1988 г. Microsoft сливат Ashton-Tate и Sybase, за да създадат вариант на Sybase SQL Server за операционната система OS/2, която по това време се разработва съвместно от IBM и Microsoft. Първото издание на MS SQL Server излиза на следващата година и се превръща в първия продукт на Microsoft, който е предназначен за бизнес пазарът на бази данни, съревновавайки се с конкурентни продукти на компании като Oracle, IBM и по-късно Sybase. В следващите няколко версии програмата върви в комплект с операционната система OS/2, като едва версия SQL Server 6.0 е първата, която е изцяло интегрирана за работа единствено с Windows NT, както и че системата се дистанцира по-осезаемо от единия от първоизточниците си: Sybase. До 1994 г. MS SQL Server съдържа в лицензионното споразумение три препратки за авторски права към Sybase, които показват първоначалния произход на софтуера.Дистанцирането се дължи на различия във вижданията на Sybase и Microsoft по отношение на качествата на крайния продукт – различни визии, цялостен дизайн и пазарни схеми. Microsoft успяват да спечелят пълни и изключителни права за всички версии на SQL Server, които са написани за операционната система Windows. Това принуждава през 1996 г. Sybase да сменят името на своя продукт на Adaptive Server Enterprise, за да избегнат объркване с продукта на Microsoft – MS SQL Server. Базите данни са стандартният начин за използване на динамична информация в съвременните приложения, правейки ги неизменна част от почти всяко едно приложение.

**2.4.4 C#**

C# се произнася като „Си Шарп“. Той е създаден от Microsoft през 2001 г., а последното обновление за него излезе в края на 2021, т.е. това е един актуален език. Може да се каже, че C# e Microsoft-ският отговор на програмния език Java, продукт на компанията Sun Microsystems (понастоящем собственост на Oracle Corporation).C# е обектно-ориентиран език, т.е. той изгражда системи посредством обекти, които взаимодействат помежду си. Обектно-ориентираното програмиране се отличава от традиционното програмиране, при което програмата е просто списък от инструкции, които компютърът изпълнява.C# е програмен език от високо ниво. Това го прави изключително достъпен за начинаещи програмисти. Като език от високо ниво, той е по-лесно четим и разбираем, тъй като е по-близо до хората, отколкото до машините (които, както вероятно знаеш, говорят само с нули и единици). Това дава възможност на програмиста да се фокусира върху разработката, правейки процеса по-лесен, бърз и разбираем.Освен това, C# разчита и на т.нар. garbage collector, който се грижи автоматично за правилното управление на паметта като ресурс, който използваш по време на разработване. Това дава изключително много свобода на програмистите, освобождава ти време и премахва излишни затруднения в процеса на работа.

Днес С# е един от най-използваните езици за програмиране. Той е език за back-end разработка, т.е. се грижи за сървърната логика, a през технологията Blazer вече опитва да се конкурира с JavaScript за целите на front-end разработката. Ето какво ще можеш да създаваш, ако овладееш C#:

1. **Десктоп приложения**- От Microsoft използват C# във всички свои продукти в това число всичките си десктоп приложения за Windows 8 и 10, т.е. езикът става все по-актуален и широкоизползван, което дава повече шансове за реализация.

2. **Уеб приложения**- ASP.NET е технологична рамка с отворен код, която също е част от .NET. Бидейки част от едно и също семейство с програмния език C#, това улеснява тяхното взаимодействие и дори го предполага. Това означава, че от Microsoft се стремят да предразположат разработчиците да използват колкото се може повече MS продукти в разработването на своите приложения и платформи. Доброто взаимодействие между C# и уеб фреймуърка ASP.NET правят програмния език подходящ и за уеб сайтове, платформи и уеб приложения.

3. **Мобилни и cross-platform мобилни разработки**

C# е подходящ също и за създаване на мобилни и междуплатформени приложения. Visual Studio – IDE-то на Microsoft e съвместимо с Xamarin – платформа за разработка на native Android, iOS и Windows приложения.

4. **Игри**

Разработката на игри със C# също е една от възможностите за реализация с езика. Един от най-популярните гейм енджини, подходящ както за PC, така и за конзолни и мобилни приложения, е Unity, който е микс от няколко езика от семейството на „С“ – С, С++ и разбира се С#. С това далеч не се изчерпват игровите енджини, писани частично или изцяло на C# и това също е една добра насока за развитие.

**2.4.5 HTML**

**HTML** (съкращение от термина на английски: ***H****yper* ***T****ext****M****arkup****L****anguage*, произнасяно най-често като *„ейч-ти-ем-ел“*, в превод „*език за маркиране на хипертекст*“) е основният маркиращ език за описание и дизайн на уеб страници. HTML кодът казва на браузърите как да показват различни елементи на уеб страница, като изображения, текстови полета, таблици и др. Има много тагове, използвани за създаване на HTML страници; обаче има някои често срещани, които ще виждате често. Най-основният етикет е . Този маркер съдържа мета-информация за вашия уебсайт, включително неговото заглавие и друга важна информация, като типа съдържание, което използва (текст/HTML). HTML е една от основните технологии на подхода за създаване на съдържание в World Wide Web. Тя ви позволява да проектирате и публикувате свои собствени уеб страници, за да ги виждат другите, да ги разпространявате по интернет и да имат достъп до тях от всеки уеб браузър. HTML документите са написани на специфичен HTML език, който включва HTML елементи като параграфи, заглавия, списъци, таблици и т.н. Използването на този език за маркиране става много лесно да създавате спретнати изглеждащи документи с добавена семантична стойност към тях. По долу може също така да прочетете малко по подробно неща свързани с този описателен език:

-HTML се използва за описване на структурата на даден документ използвайки маркери – така наречените тагове

-HTML документите са съвкупност от маркиращи тагове

-Тези тагове описват съдържанието на документа и са изграждащите го      блокове

-Тези блокове се наричат HTML елементи

-HTML таговете се явяват един вид етикет на типа съдържание – напр. heading, paragraph, table и т.н

-HTML документите съдържат само HTML тагове и чист текст

-HTML документите, също се наричат и уеб страници

-Браузърите не визуализират HTML таговете, а ги използват за показване на съдържанието

HTML работи по структура, която ни помага да видим резултатите в една HTTP страничка които сме написали като код на този описателен език. Също така той е полезен за редактация на заглавия, параграфи, секции, таблици и много други основни елементи на една уебаджийска страница. Когато някой кликне с курсора на мишката на адресното поле и въведе един обикновен URL адрес, браузърът изпраща GET заявка с която въведения адрес се взима и се изпраща на сървъра където е качена страницата посредством HTTP(Hypertext Transport Protocol). Следователно след това сървърът прочита заявката, която е испратена и изпраща POST заявка с която се испраща и визуализира отговорът на сървърът към потребителя.

***Тагове в HTML***

Маркиращите HTML тагове, биват наричани по един прост начин, просто тагове. Общо прието е да се наричат по този прост начин, поради простата причина, че в тази сфера това са единствените тагове, които се изплозвта за маркиране.

*HTML таговете са ключови думи (имена) оградени с ъглови скоби – <html>*

*HTML таговете повечето пъти се пишат по двойки – <p>*

Във всяка двойка, първия таг е отварящия, а вторият затварящ. Втория таг се пише точно като първият с разликата, че се добавя наклонена черта преди името му.

**2.4.6 CSS – Cascading Style Sheet**

CSS (Cascading Style Sheets) е език за описание на стилове (език за стилови файлове, style sheet language) – използва се основно за описание на онлайн представянето на уеббазиран документ, който написан на език за маркиране. Най-често се използва допълнително към чистия HTML, но се прилага и върху XML уебстраници и документи. Спецификацията на CSS официално се поддържа от W3C.CSS още в началото на развитието на www започва да се добавя към стандартния HTML с цел да бъдат разделени съдържанието и структурата на уеб страниците отделно от тяхното визуално представяне. Преди стандартите за CSS, установени от W3C през 1995 г., съдържанието на сайтовете и стила на техния дизайн са писани в една и съща HTML страницата. В резултат на това HTML кодът се превръща в сложен и нечетлив, а всяка промяна в проекта на даден сайт изисквала корекцията да бъде нанасяна в целия сайт страница по страница. Използвайки CSS, настройките за форматиране могат да бъдат поставени в един-единствен файл и тогава промяната ще бъде отразена едновременно на всички страници, които използват този CSS файл. CSS позволява да се определя как да изглеждат елементите на една HTML страница – шрифтове, размери, цветове, фонове, и др. CSS кодът се състои от последователност от стилови правила, всяко от които представлява селектор, последван от свойства и стойности.

***Съществуват няколко начина за прилагане на CSS стила:***

1.Външен стил - Записът, който направихме в горния пример, се нарича външен стил (External Style Sheet) и се използва когато трябва да се контролират множество HTML документи, като нужните параметри се задават във външен файл (file.css).

2.Вътрешен стил - Съществува и вътрешен за HTML документа стил (Internal Style Sheet), който се използва за да се зададе вида на един отделен HTML документ, като нужните свойства се задават със специалния таг <style> в секцията <head> на HTML страницата.

3.Вграден стил - Другият начин за налагане на стил е Inline Styles – вътрешни за HTML таговете стилове. CSS стилът се разполага като атрибут директно в HTML тага например <p>.

Когато в един документ се използват и трите начина за налагане на CSS стил – с най-висок приоритет е стилът на Inline Style (вътрешните за HTML таговете стилове), след тях на по приоритет са Internal Style Sheet (стиловете от секцията head на HTML документа) и последни по приоритет са External Style Sheet, т.е. стиловете, декларирани във външен CSS файл.

**2.4.7 JS(Java Script)**

JavaScript е популярен език за програмиране, който се използва за създаване на интерактивни и динамични уеб страници. Той е един от основните езици за програмиране на уеб, заедно с HTML и CSS. JavaScript е изпълнен от страна на браузъра на потребителя и се използва за създаване на ефектни елементи като анимации, динамични форми, динамично генерирани списъци, изображения и много други. Той също така се използва за извличане на данни от уеб страници и комуникация с уеб сървъри. JavaScript е мултиплатформен език за програмиране и може да се използва както за уеб, така и за мобилни приложения. Той има широк кръг от библиотеки и фреймуърки, които помагат за ускоряване на процеса на разработка на уеб приложения. JavaScript е един от най-широко използваните езици за програмиране в света и се използва не само за уеб, но и за различни други приложения като игри, мобилни приложения, десктоп приложения и други. JavaScript е език с динамичен тип данни, който означава, че променливите могат да съдържат различни типове данни, като числа, символни низове, булеви стойности, обекти и други.

Един от най-големите предимства на JavaScript е, че може да се използва за манипулиране на HTML документи и стилове, което позволява на програмистите да създават динамични и интерактивни уеб страници, които да реагират на действията на потребителя. JavaScript има много библиотеки и фреймуърки, като React, Angular и Vue, които помагат на програмистите да създават по-бързо и ефективно уеб приложения. Освен това, той се поддържа от всички основни браузъри като Chrome, Firefox, Safari и Edge, което го прави много удобен за уеб разработка. В заключение, JavaScript е мощен език за програмиране, който е важен инструмент за създаване на уеб приложения и интерактивни уеб страници. JavaScript предоставя много възможности за създаване на динамично генерирани уеб страници и уеб приложения. Той се използва за създаване на ефектни елементи като слайдшоутове, анимации и различни видове визуални ефекти. JavaScript също така има много различни библиотеки и фреймуърки, като jQuery, Bootstrap, React, Angular и други, които помагат на програмистите да разработват уеб приложения по-лесно и бързо. JavaScript може да се използва и за обработка на данни и комуникация с уеб сървъри чрез AJAX заявки. Това позволява на уеб приложенията да взаимодействат с бази данни и други услуги на уеб сървъри, като например извличане на информация за потребителите и изпращане на формуляри.

JavaScript е език, който се променя бързо и постоянно се актуализира с нови функционалности и подобрения. Това означава, че програмистите трябва да следят развитието на езика и да бъдат в крак с последните тенденции, за да могат да създават най-ефективните и съвременни уеб приложения.

В заключение, JavaScript е един от най-широко използваните езици за програмиране в света и предоставя много възможности за създаване на уеб приложения и интерактивни уеб страници. Той е мощен език, който продължава да се развива и да се подобрява с нови функционалности и инструменти.

**2.4.8 Bootstrap**

Bootstrap е един от най-популярните фронтенд фреймуърки, който предоставя готови компоненти и стилове за създаване на модерни и респонсивни уеб страници. Фреймуъркът е създаден от Twitter и е първоначално наречен "Twitter Blueprint", но по-късно е преименуван на Bootstrap. Bootstrap е написан на HTML, CSS и JavaScript и включва готови компоненти като бутони, формуляри, навигационни менюта, карусели, модални прозорци, и други. Тези компоненти са лесни за използване и могат да бъдат настроени за да отговарят на нуждите на всеки проект. В допълнение към компонентите, Bootstrap предоставя и готови CSS класове, които могат да бъдат използвани за настройка на стиловете на елементите на уеб страницата. Тези класове включват грид система, която позволява да се създават респонсивни макети за различни размери на екраните. Bootstrap също така включва и набор от JavaScript плъгини, които добавят допълнителна функционалност към компонентите, като например валидация на формуляри, модални прозорци, карусели и други. Bootstrap е лесен за използване, има голяма заедница и предоставя безплатен и отворен код. Това го прави идеален избор за начинаещи и професионални уеб разработчици.

Bootstrap е създаден с идеята да улесни процеса на създаване на уеб страници, като предоставя готови компоненти и стилове, които могат да бъдат настроени за да отговарят на нуждите на всеки проект. Той също така е изключително респонсивен, което означава, че дизайнът му се променя автоматично в зависимост от размера на екрана на устройството, на което се използва.

***Bootstrap има много предимства, някои от които са:***

1. Готови компоненти: Bootstrap предоставя готови компоненти, като формуляри, бутони, навигационни менюта, карусели, модални прозорци, и други. Тези компоненти могат да бъдат лесно използвани и настроени за да отговарят на нуждите на всеки проект.

2. Грид система: Bootstrap включва грид система, която позволява да се създават респонсивни макети за различни размери на екраните. Това прави процеса на създаване на респонсивен дизайн много по-лесен.

3. Готови стилове: Bootstrap предоставя готови CSS класове, които могат да бъдат използвани за настройка на стиловете на елементите на уеб страницата. Тези класове също така могат да бъдат настроени, за да отговарят на нуждите на всеки проект.

4. Поддръжка на JavaScript плъгини: Bootstrap включва набор от JavaScript плъгини, които добавят допълнителна функционалност към компонентите, като например валидация на формуляри, модални прозорци, карусели и други.

5. Безплатен и отворен код: Bootstrap е безплатен и отворен код, което го прави идеален за начинаещи и професионални уеб разработчици.

 Bootstrap има голяма заедница и множество ресурси за учене, като например официалната документация, уроци и курсове онлайн. Това прави Bootstrap лесен за използване и подходящ за проекти от всякакъв вид

**ГЛАВА 3.**

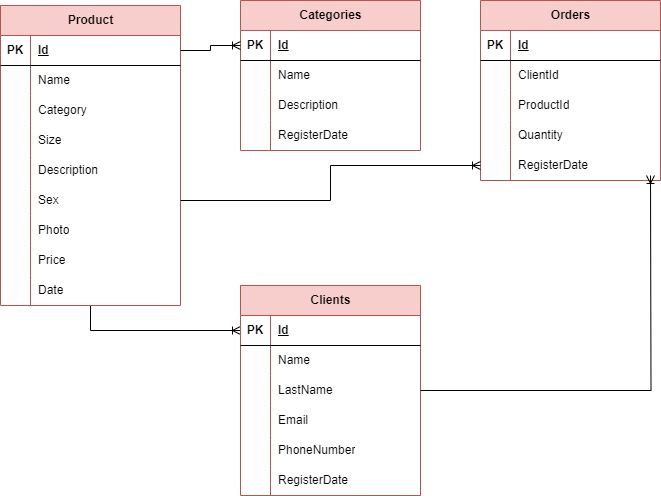
**ПРОГРАМНА РЕАЛИЗАЦИЯ НА WEB БАЗИРАН**

**сайт за обувки “comfort”**

* 1. **Структурата на базата от данни – Data Models**

**3.1.1 Проектиране на Е/R диаграма на проекта преди да бъдат създадени и реализиране класовете в ASP.NET приложението, което е показано на фигура**

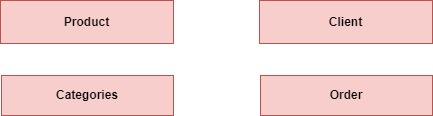
*Фиг.*

****

**3.1.2. Описание на обектите – реализира се с правоъгълник с описание на предметната област**

На фигура е показано, че E/R диаграмата се състои от 4 обекта, описващи предметната област на проекта. Това са:

Фиг.

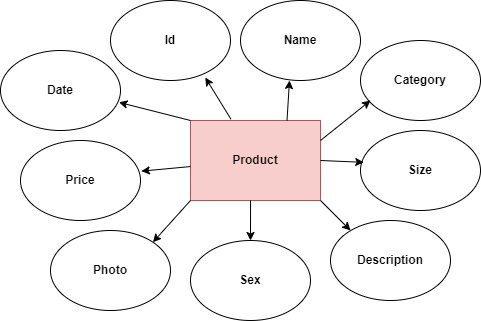


**3.1.3 Атрибути, реализиращи свойствата на обектите в E/R диаграмата**

**3.1.3.1 Обектът Product**

На фигура е показан обектът Product(Продукт). Това е основният обект, който реализира добавянето на нов продукт в базата от данни на магазина за обувки. Той съдържа следните публично достъпни свойства:

Фиг.



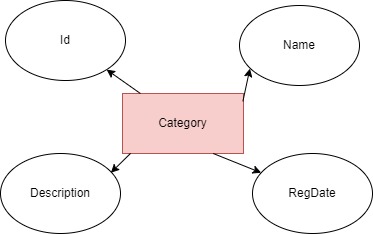
**Свойства на обекта Product:**

* **Id – Това свойство присъства в почти всички обекти в базата от данни на нашето уеб приложение. Това е първичният ключ, по който се идентифицира всеки нов запис в базата и трябва да е от тип Int.**
* **Name – Това свойство представлява наименованието на нов продукт в базата и трябва да е от тип string.**
* **Category – Това свойство е от тип string и представлява типът категория от която е продуктът.**
* **Sex – Това свойство ни дава възможност да изберем от какъв пол е потребителят.**
* **Price – Представлява цената на обувката, която ще се добавя. Трябва да бъде от тип Decimal, за да бъде десетично число.**
* **Description – Описание на обувките, трябва да бъде String.**
* **Date– Това е дата на актуализация или добавяне на обувки в системата. Трябва да бъде от тип DateTime.**
* **Photo – Това свойство показва точният външен вид на обувките.**

**3.1.3.2 Обектът Category**

На фигура е показан обектът Category(Продукт). В него се съдържат следните елементи: Id, Name, Description, RegDate.

Фиг.

****

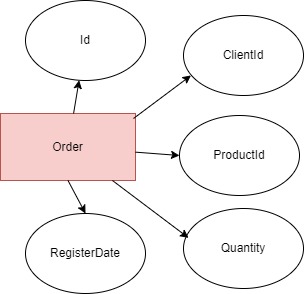
**Свойства на обекта Category:**

* **Id – Това свойство присъства в почти всички обекти в базата от данни на нашето уеб приложение. Това е първичният ключ, по който се идентифицира всеки нов запис в базата и трябва да е от тип Int.**
* **Name – Това свойство представлява наименованието на нов продукт в базата и трябва да е от тип string.**
* **Description – Описание на обувките, трябва да бъде String.**
* **RegDate– Това е дата на актуализация или добавяне на обувки в системата. Трябва да бъде от тип DateTime.**

**3.1.3.3 Обектът Order**

На фигура е показан обектът Order(Поръчка). Това е един от най-важните обекти в базата от данни на приложението, защото чрез него потребителят може да направи поръчка на избран от него артикул. Той съдържа свойтвата: Id, ClientId, ProductId, Quantity, RegisterDate.

Фиг.



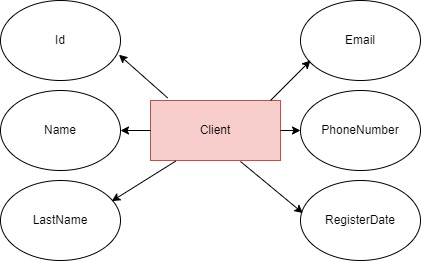
**Свойства на обекта Order:**

* **Id – Първичен ключ за таблицата за поръчка на обувки.**
* **Client Id– Външен ключ, който свързва таблицата Orders(поръчка) с таблицата Clients(Клиенти). По този начин се постига връзка много поръчки към един клиент, който ще прави самата поръчка. Свойството ClientId трябва да бъде string, тъй като Id-то на клиента по подразбиране също е string.**
* **ProductId – Външен ключ към таблицата Products. Осъществява се връзка 1 обувки към много поръчки. Трябва да бъде тип Int.**
* **Quantity – Количество на поръчаните обувки. Трябва да е тип Int.**
* **RegisterDate – Дата на създаване на поръчка.**

**3.1.3.4. Обектът Client**

**На фигура 18 е показан обектът Client(Клиент). Този клас наследява данните, на регистриран потребител в системата от Identity модула, когато той се регистрира. Добавени са допълнителни свойства за името и фамилията на клиента, също и датата на актуализация в системата.**

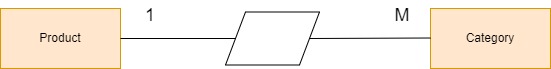
Фиг.

****

**Свойства на обекта Client:**

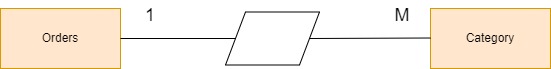
* **Name – Името на клиента, което е тип string.**
* **LastName – Фамилия на клиента, която е тип string.**
* **RegisterDate – Дата на регистриране на клиент.** **Трябва да бъде от тип DateTime.**
* **Email – Имейла на клиента, който е тип string.**
* **Phone Number – Телефонния номер на клиента, който е тип Int.**
  + - 1. **Описание на връзките между обектите в базата от данни**

На (фиг.) е представена връзката на 1 обект от таблицата Products(Продукти) към много обекти от таблицата Categories(Категории).



Фиг.

На (фиг.) е представена връзката на 1 обект от таблицата Orders(Поръчки) към много обекти от таблицата Products(Продукти).



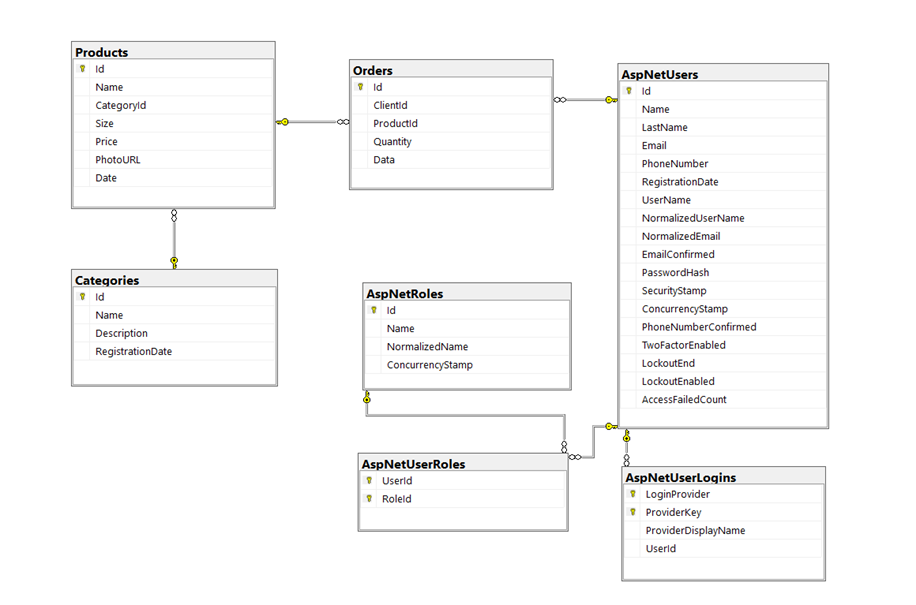
Фиг.

На (фиг.) е представена връзката на 1 обект от таблицата Categories(Категории) към много обекти от таблицата Products(Продукти).



Фиг.

**3.1.5 Структура на базата от данни**

****

**3.1.5.1 Описание на данните в таблиците:**

* **Таблицата Products**

**Id-**Това е поле от тип int и трябва да присъства във всяка една таблица.

**Name-** Това е поле от типа string, с което ще въвеждаме името на обувките.

**CategoryId-** Свързочно поле от тип int, с което се осъществява връзка M:1 с таблицата Categories.

**Size-** Това е поле от тип decimal. С него ще въвеждаме наличният размер на обувки.

**Price-** Това е поле от тип decimal, което ще показва цената на всяка обувка.

**PhotoURL-** Поле от тип string, с което ще прочитаме връзките на изображенията и те ще се визуализират като снимки в таблицата.

**Date-** Поле от тип DateTime, което ще показва текущата дата.

Фиг.

public class Product

{

public int Id { get; set; }

public string Name { get; set; }

public int CategoryId { get; set; }

public Category Categories { get; set; }

public decimal Size { get; set; }

public decimal Price { get; set; }

public string PhotoURL { get; set; }

public DateTime Date { get; set; }= DateTime.Now;

public ICollection<Order> Orders { get; set; }

}

* **Таблицата Orders**

**Id-**Това е поле от тип int и трябва да присъства във всяка една таблица.

**Name-** Това е поле от типа string, с което ще въвеждаме името на категориите обувки.

**Description-** Поле от типа string, с което ще въвеждаме описанието за всяка обувка.

**RegistrationDate-** Поле от тип DateTime, което ще показва текущата дата.

public class Order

{

public int Id { get; set; }

public string ClientId { get; set; }

public Client Clients { get; set; }

public int ProductId { get; set; }

public Product Products { get; set; }

public int Quantity { get; set; }

public DateTime Data { get; set; }= DateTime.Now;

}

* **Таблицата Category**

**Id –** Това свойство присъства в почти всички обекти в базата от данни на нашето уеб приложение. Това е първичният ключ, по който се идентифицира всеки нов запис в базата и трябва да е от тип Int.

**Name –** Това свойство представлява наименованието на нов продукт в базата и трябва да е от тип string.

**Description –** Описание на обувките, трябва да бъде String.

**RegDate–** Това е дата на актуализация или добавяне на обувки в системата. Трябва да бъде от тип DateTime

public class Category

{

public int Id { get; set; }

public string Name { get; set; }

public string Description { get; set; }

public DateTime RegistrationDate { get; set; }= DateTime.Now;

public ICollection<Product> Products { get; set; }

}

**3.2. Структура на MVC приложението – *Controllers*, *Views*, *ViewModels*,**

**3.2.1 Какво представлява MVC?’**

MVC (Model-View-Controller) е шаблон за дизайн на софтуер, който се използва за построяване на приложения, които имат ясно разделение между данните (модел), потребителския интерфейс (изглед) и бизнес логиката (контролер).

**1. Модел (Model)**

* Моделът представлява данните и бизнес логиката на приложението. Той съдържа обработка на данни, валидация и всякаква друга логика, която се отнася за манипулиране на информацията, с която работи приложението.
* В MVC моделът е отделен от потребителския интерфейс и контролера, като това позволява преизползване на кода и лесно променяне на данните без да се нарушават други части от приложението.

**2. Изглед (View)**

* Изгледът е това, което потребителят вижда и с което взаимодейства. Той представлява представянето на данните, които се подават от модела, но без да включва логика за тяхното обработване.
* В MVC изгледът е отделен от модела и контролера, което позволява различни изгледи да се използват за едни и същи данни, като се поддържа разделението между логиката за представяне и логиката за обработка.

**3. Контролер(Controller)**

* Контролерът управлява входната точка на приложението и управлява потребителските заявки. Той приема входни данни от потребителя, ги обработва и изпраща необходимата информация към модела или изгледа.
* В MVC контролерът свързва изгледите и моделите, като координира дейностите им. Той се грижи за комуникацията между изгледите и моделите, като осигурява, че потребителските заявки се обработват правилно и ефективно.

**3.2.2 Как работи шаблона на Controller с GET и POST заявки**

В ASP.NET MVC контролерът е също като всеки друг клас в .NET приложението и има разширение .cs файл. Той обаче има някои характеристики, които го правят разпознаваем от .NET рамката като MVC контролер.

* Поставя се в папка "Controllers" в проекта.
* Наследява Microsoft.AspNetCore.Mvc.Controller.
* Името на класа завършва на "Controller", например "HomeController". Ако тази конвенция не се спазва, .NET рамката няма да разпознае класа като контролер.

Когато класът наследи Microsoft.AspNetCore.Mvc.Controller, получава допълнителна функционалност за уеб приложенията, включително възможността за връщане на изгледи/частични изгледи и достъп до информация, свързана с HTTP заявките, благодарение на свойството HttpContext.  
 Методите на контролера се наричат "действия" (Actions) и представляват връзката между браузъра (и следователно потребителя) и приложението. Те обикновено съответстват на конкретни действия в приложението, които след това връщат резултат на потребителя. За да се извика действие на контролера, браузърът изпраща HTTP заявка към съответния URL адрес, като този адрес се маршрутизира към съответния контролер и действие (метод) чрез концепцията за маршрутизация. Важно е да се отбележи, че всички публични методи в клас Controller се считат за действия. Това означава, че ако не искате дадено действие да бъде достъпно чрез URL, трябва да го маркирате като private, или алтернативно може да се използва атрибута [NonAction].

Относно HTTP заявките:

* GET методът се използва за извличане на информация от сървъра и е подходящ, когато се изисква предаване на ограничен обем данни през URL адреса.
* POST методът се използва за изпращане на данни към сървъра и е подходящ, когато се изисква предаване на голям обем информация или когато се извършва създаване, обновяване или изтриване на ресурси.

**3.2.3 Описание на CRUD операциитe**

Съкратено от Create(Създаване), Read(Четене), Update(Актуализация) и Delete (Изтриване), тези операции са основните функции за управление на данни в системите за съхранение и манипулиране на информация

**Create** - операция за създаване/добавяне на нов запис в базата от данни.  
**Read** - операцията за четене се използва за извличане на информация от системата. Това може да включва извличане на цялата база данни или част от нея, четене на конкретен ред или обект.

**Update** - Операцията позволява промяна на вече съществуващи записи или обекти в системата. Това може да включва промяна на стойности на полета, добавяне или изтриване на информация.  
**Delete** - Операцията за изтриване на съществуващи записи или обекти в базата от данни.

**CRUD операции в уеб приложението на магазин за обувки „Comfort“**

* **Операцията CREATE**

В моето приложение за магазин за обувки „Comfort“ разполагаме с два Action метода, които изпълняват операцията CREATE. Това е метод, който извиква HTTP GET заявка за визуализация на изглед за добавяне на нова котка и метод, който извиква HTTP POST заявка, чрез която котката ще се добави в базата от данни.

// GET: Products/Create

public IActionResult Create()

{

ProductsVM model = new ProductsVM();

model.Category = \_context.Categories.Select(x => new SelectListItem

{

Text = x.Name,

Value = x.Id.ToString(),

Selected = (x.Id == model.CategoryId)

}

).ToList();

return View(model);

}

* **Операцията DELETE**

public async Task<IActionResult> Delete(int? id)

{

if (id == null)

{

return NotFound();

}

var product = await \_context.Products

.Include(p => p.Category)

.FirstOrDefaultAsync(m => m.Id == id);

if (product == null)

{

return NotFound();

}

return View(product);

}

Това Delete е операцията на Products Controller. Тази операция ни позволява да изтриваме вече съществуващи полета с данни.

* **Операцията UPDATE**

public async Task<IActionResult> Edit(int? id)

{

if (id == null)

{

return NotFound();

}

var product = await \_context.Products.FindAsync(id);

if (product == null)

{

return NotFound();

}

ViewData["CategoryId"] = new SelectList(\_context.Categories, "Id", "Id", product.CategoryId);

return View(product);

ProductsVM model = new ProductsVM()

{

Id = product.Id,

Name = product.Name,

CategoryId = product.CategoryId,

Description = product.Description,

Price = product.Price,

Size = product.Size

};

return View(model);

}

Тази операция позволява свойствата на даден обект от базата данни да бъдат променени и съответно да бъдат презаписани с вече направените промени в базата.

Това е Details операцията на Products Controller

// GET: Products/Details/5

public async Task<IActionResult> Details(int? id)

{

if (id == null)

{

return NotFound();

}

var product = await \_context.Products

.Include(p => p.Category)

.FirstOrDefaultAsync(m => m.Id == id);

if (product == null)

{

return NotFound();

}

return View(product);

}

**3.2.4 RazorPages и RazorViews**

Razor Pages и Razor Views са два компонента от ASP.NET Core framework, които се използват за създаване на уеб приложения. Те се използват за генериране на HTML съдържание за уеб страниците, като се основават на концепцията за модел-изглед-контролер (Model-View-Controller или MVC) архитектура.  
 **Razor Pages** е елемент от ASP.NET Core framework, представляващ алтернативен начин за създаване на уеб приложения в сравнение с по-класическия модел-изглед-контролер (MVC) подход. В Razor Pages, всяка уеб страница се представя като самостоятелен компонент, който включва както изгледа(View) (HTML шаблона), така и кода за обработка на заявките (C# кода). Този модел прави създаването на прости и по-лесни за поддръжка уеб приложения по-достъпен начин. Когато се създава Razor Page, се създават два основни файла:

* Razor View (cshtml файл): Това е HTML файл, който съдържа статичния HTML код на уеб страницата. С помощта на специалния синтаксис на Razor, този файл също така позволява вмъкване на C# код за динамично генериране на съдържание, като например извличане на данни от модела.
* Page Model (cs файл): Този файл съдържа C# код, който се изпълнява при заявка към страницата. Тук може да се реализира логика за обработка на заявката, валидация на данни, извличане на данни от базата данни и други операции. Той се свързва със съответния Razor View, предоставяйки данните, които трябва да бъдат показани на страницата.

Разликата между Razor Pages и по-конвенционалния MVC модел е, че в Razor Pages не е необходимо да се създават отделни контролери за всяка страница. Вместо това, всичко, което се отнася до определена страница, е групирано в един Razor Page компонент, който е по-лесен за поддръжка и разбиране, особено за по-малки приложения.

**Razor Views**, от друга страна, са част от ASP.NET MVC framework и се използват за дефиниране на HTML шаблони, които се генерират динамично от уеб приложението. Те обикновено се използват за презентиране на данни, които са предоставени от контролерите. Razor Views предоставят по-голяма гъвкавост от Razor Pages, като позволяват на разработчиците да създават множество изгледи, които могат да бъдат използвани от различни части на приложението.

В Razor Views, HTML шаблоните могат да бъдат комбинирани с C# код, който се изпълнява по време на генерирането на страницата. Това позволява на разработчиците да вмъкват данни от моделите или да извършват логика за визуализация директно в шаблоните.  
 Така че, докато Razor Pages се фокусират върху организирането на логиката и представянето за една конкретна страница в един файл, Razor Views са по-подходящи за ситуации, където има нужда от по-голяма гъвкавост и множество изгледи, които да се използват от различни части на приложението.

**3.2.5 TagHelpers и HTMLHelpers**

**TagHelpers** са нова технология в ASP.NET Core, която позволява на разработчиците да дефинират собствени HTML тагове, които се интерпретират и обработват от уеб приложението. Те позволяват на разработчиците да създават HTML код по-лесно и елегантно, като вместо да се използват специални атрибути или помощни методи, таговете сами по себе си носят логика за обработка на страна на сървъра. Таговете могат да бъдат декларирани с помощта на C# класове, които наследяват TagHelper класа. Посредством презаписване на различни методи в този клас, разработчиците могат да определят поведението на таговете при генериране на HTML. TagHelpers са особено полезни за улесняване на работата с форми, валидация на данни, свързване на модели и други общи задачи при разработката на уеб приложения.  
 **HtmlHelpers** в ASP.NET MVC са набор от методи, които се използват за генериране на HTML код динамично в Razor Views. Те предоставят удобен начин за създаване на HTML елементи, форми, връзки, изображения и други елементи на уеб страницата директно от кода на приложението.

• Генериране на HTML елементи: HtmlHelpers позволяват генерирането на различни HTML елементи като текстови полета, бутони, селектори, чекбокси, радио бутони и др. Това е изключително полезно при създаването на форми и интерактивни елементи на уеб страницата.  
 • Създаване на форми: С HtmlHelpers, можем лесно да генерираме HTML форми и техните елементи, като текстови полета, селектори, чекбокси и други, които са свързани с модела на данните. Това улеснява връзката между уеб страницата и контролера, тъй като данните от формата могат автоматично да бъдат прехвърлени към методите на контролера за обработка.  
 • Създаване на връзки: HtmlHelpers също така позволяват създаването на връзки (links) към други уеб страници или методи на контролерите. Те осигуряват възможност за генериране на URL адреси, които са съответни на маршрутите в приложението, и ги вграждат във връзките. Изображения и други медийни елементи: HtmlHelpers предлагат методи за генериране на HTML тагове за включване на изображения, аудио и видео файлове, и други медийни елементи в уеб страницата.

**3.2.6 Routes Engine**

Маршрутният механизъм (Route Engine) в ASP.NET Core е отговорен за маркирането и съпоставянето на URL адреси към конкретни контролери и техните методи във вашия ASP.NET Core приложение. Той е съставна част от ASP.NET Core middleware pipeline и играе ключова роля в определянето на това как заявките от клиентите се обработват от приложението.

Стъпки на работния процес на Routes Engine:  
 • Регистрация на маршрути: При стартиране на приложението си, ASP.NET Core приложението регистрира своите маршрути. Това се прави чрез методите за конфигуриране на маршрутите, обикновено в метода Configure на класа Startup. Маршрутите се дефинират чрез шаблони, които определят структурата на URL адресите и как те се съпоставят с контролерите и техните методи.  
 • Обработка на HTTP заявките: Когато ASP.NET Core приложението получи HTTP заявка от клиент, маршрутният механизъм е активиран, за да определи къде трябва да бъде изпратена заявката за обработка. Той използва информацията от URL адреса на заявката, за да я съпостави с най-подходящия маршрут, който е предварително регистриран.  
 • Съпоставяне на маршрут: Когато маршрутният механизъм намери подходящ маршрут, той извлича информацията от URL адреса, като параметри и данни, и определя кой контролер и метод трябва да бъдат извикани за обработка на заявката. Този процес на съпоставяне на маршрут може да включва различни стратегии за определяне на най-подходящия маршрут, като например шаблониране на URL адресите, определени параметри и т.н.  
 • Изпълнение на контролер и метод: След като маршрутът е определен успешно, маршрутният механизъм извиква съответния контролер и метод, който е асоцииран с маршрута. Контролерът обработва заявката, извършва необходимите операции, като например обработва получените данни от клиента и връща резултат, който трябва да бъде върнат на клиента.  
 • Връщане на HTTP отговор: След като контролерът завърши обработката на заявката, маршрутният механизъм генерира HTTP отговор, който се връща на клиента. Този отговор може да включва HTML код, JSON данни, изображения или други ресурси, в зависимост от характера на заявката и логиката на приложението.

**3.2.7 Работа с View моделите – ViewBag и ViewData**

ViewBag и ViewData са два механизма, които ASP.NET предоставят за пренасяне на данни от контролера към изгледа (View) при използване на модел-изглед (Model-View-Controller).

Въпреки че ViewBag и ViewData са възможности за пренасяне на данни към изгледа в ASP.NET MVC (или ASP.NET Core MVC), използването на модел (Model) е предпочитан и по-често срещан подход. Моделът е прост клас, който представлява данни от един или повече класове, които ще бъдат показани или презентирани на потребителя. ViewModel, съкратено от "модел на изглед", е просто модел, предаван на изгледа. Моделът може да бъде клас или специално дефиниран клас, представящ данните, до които трябва да се достъпи от изгледа. Използването на модел осигурява по-структуриран и чист начин за пренасяне на данни към изгледа, като същевременно осигурява статична проверка на типовете по време на компилацията.

* **ViewBag** – динамичен обект, който позволява добавянето на данни към него и тяхното използване в изгледа без явно дефиниране на свойства.
* **ViewData** – представя като речник (Dictionary) от ключ-стойностни двойки, които могат да се използват за пренасяне на данни от контролера към изгледа.

И двете ViewBag и ViewData предоставят механизми за пренасяне на данни от контролера към изгледа, като всеки от тях има свои собствени характеристики и употреба.

@{

ViewData["Title"] = "Index";

}

Фиг.

* 1. **Авторизация и оторизация на потребителите на приложението –**

**3.3.1 Какво е авторизация и оторизация?**

Авторизирането е механизъм за защита, използван за определяне на привилегии на потребителя или нива на достъп, свързани със системните ресурси, включително компютърни програми, файлове, услуги, данни и функции на приложението. Оторизацията обикновено се предхожда от удостоверяване за проверка на идентичността на потребителя.

Администраторите обикновено получават нива на разрешения, обхващащи всички системни и потребителски ресурси. По време на оторизацията система проверява правилата за достъп на удостоверени потребители и предоставя или отказва достъп до ресурси. Оторизацията е процес на свързване на потребител в дадени права. За оторизиран се счита потребител, който има право да работи с поискания ресурс или да извърши конкретната операция.

Във веригата на сигурността това е следващият процес след автентикацията – след като разберем кой е потребителят, ние трябва да знаем какви са неговите права. В ASP.NET за оторизация се използва моделът, който всеки потребител може да се извършва не само на ниво потребител, но и на ниво роля. [Authorize] - използва се когато искаме да забраним влизането в определено View без потребителя да е влезнал акаунта си.

[Authorize]

public class AdoptionsController : Controller

**Фиг.**

На (фиг. ) е показано как Authorize може да се сложе над цял Controller.

**3.3.2 Конфигуриране на ролите в приложението**

За да конфигурираме ролите, трябва да добавим папка, която съдържа специален код за добавяне на роли. С този код можем да създаваме каквито си роли поискаме, като най-важната от тях е администраторската роля.

* **Добавяне на роли**

(фиг. )

public static async Task SeedRolesAsync(RoleManager<IdentityRole> roleManager)

{

await roleManager.CreateAsync(new IdentityRole("Admin"));

await roleManager.CreateAsync(new IdentityRole("User"));

}

На (фиг. ) със следния код създаваме двете роли в приложението – “Admin”(Администратор) и “User”(Потребител). Всяка роля си има предварително зададени права на достъп. Това е честа практика в приложенията, които използват система за управление на потребители като ASP.NET Identity, за да се уверят, че в системата винаги има минимален набор от необходими роли.

* **Как става добавянето на администраторската роля**

public static async Task SeedSuperAdminAsync(UserManager<Client> userManager)

{

var defaultUser = new Client

{

UserName = "Admin",

Email = "admin@gmail.com",

FirstName = "Nikol",

LastName = "Kralimarkova",

PhoneNumber = "0893959336",

EmailConfirmed = true,

PhoneNumberConfirmed = true

};

var user = await userManager.FindByEmailAsync(defaultUser.Email);

if (user == null)

{

var result = await userManager.CreateAsync(defaultUser, "Gm123@");

if (result.Succeeded)

{

await userManager.AddToRoleAsync(defaultUser, "Admin");

}

}

}

На (фиг. ) е показано как се създава нов потребител със зададено потребителско име, електронна поща, и други лични данни, като име и телефонен номер. Създаваме потребителя със зададената информация и паролата, която се въвежда ръчно. След успешното създаване на потребителя, му се присвоява ролята "Admin", което означава, че той ще има административни права в системата и ще може да управлява различни аспекти на приложението

**ЗАКЛЮЧЕНИЕ**

**Обобщение на реализирания проект, предимства и недостатъци, тенденции за бъдещи разработка и развитие**

Целите, които бяха поставени за създаването на проекта:

- Уеб приложението да поддържа CRUD операции за продуктите, което е изпълнено.

- Да бъде реализирана пазарска кошница за завършване на поръчка след регистрация или вход в сайта, което е изпълнено.

- Уеб приложението да има администраторска роля с разширени функционални възможности и клиентска роля с ограничени функционални възможности, което е изпълнено.

- Уеб приложението да поддържа следните функционалности: **добавяне на нов продукт с неговата категория; промяна и изтриване на вече съществуващи артикули от базата данни; , което е изпълнено.**

След моментното приключване на разработка на проекта, винаги има място за надграждане и подобрения, или както се казва ъпдейти и ъпгрейди. Единият от аспектите, в който сайтът може да се подобри е дизайнът и потребителски интерфейс. В този конкретен момент бих отчел това като слаба точка на проекта, която обаче има поле за развитие и градеж. Дизайнът винаги може да става по-модерен и по-интуитивен от настоящия.

Предимствата на реализирания досега проект е, че е лесен за използване както от потребител, така и от администратор или гост. Достатъчната функционалност без нищо излишно и натоварващо е друго предимство на моя краен продукт.

**ИЗПОЛЗВАНА ЛИТЕРАТУРА**

Примери: /трябва да се следва точно следния формат/

1. Наков С., Колев В. и колектив, Въведение в със C#, София, 2015, ISBN 978-954-400-527-6

2. Колисниченко Д., HTML 5 & CSS 3 - практическо програмиране за начинаещи. Трето издание, Асеневци, София, 2017, ISBN 9786197356113

3. Pro ASP.NET Core 3: Develop Cloud-Ready Web Applications Using MVC, Blazor, and Razor Pages 8th ed. Edition, Apress, New York, 2020, ISBN 1484254392

4. Learn ASP.NET Core 3: Develop modern web applications with ASP.NET Core 3, Visual Studio 2019, and Azure, 2nd Edition Paperback, Packt, Birmingham, UK, 2019, ISBN 1789610133

Електронни ресурси

5. https://www.w3schools.com/,

6. https://docs.microsoft.com/en-us/aspnet/core/?view=aspnetcore-3.1; http://www.introprogramming.info/intro-csharp-book/

7. https://to6esko.github.io/ - Български превод на Eloquent JavaScript