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, организиращ дейността на сладкарски цех „Сладко изкушение“**

*Ученик:* ***Мирослав Дечев Николов***

***Професия:*** *код 481020 „Системен програмист“*

***Специалност:*** *код 4810201 „Системно програмиране“*

***Консултант:*** *Стоян Пършев*

Сопот, 2022 г.

**Съдържание**

**Увод..........................................................................................................................................3**

**Кратко описание ...................................................................................................................3**

**Цели и задачи ........................................................................................................................3**

**Глава първа – Проучванe....................................................................................................4**

**1.1 Предпоставка за създаване на продукта ..............................................................4**

**1.2 Съществуващи решения и реализации ................................................................6**

**Глава втора – Проектиране ................................................................................................7**

**2.1 Функционални изисквания ....................................................................................7**

**2.2 Нефункционални изисквания ................................................................................9**

**2.3 Изготвяне на Use Case Diagram ..............................................................................9**

**2.4 Потребителски интерфейс .....................................................................................12**

**2.5 Описание на избраната технология и софтуерните средства……..................14**

**Глава трета - Програмна реализация..............................................................................20**

**3.1 Структурата на базата от данни ................................................................................20**

**3.2 Структура на MVC приложението.......................................................................23**

**3.2.1 Описание и конфигуриране на моделите...........................................................25**

**3.2.2 Описание на CRUD операциите...........................................................................27**

**3.2.3 Работа с View моделите………………………………………..…………………30**

**3.2.4 Razor Pages и Razor Views……………………………...………………………..31**

**3.3 Използване на технологии и алгоритми…………………………………...…...32**

**3.3.1 Routе Engine………………………………………………………………………..32**

**3.3.2 Controller с GET и POST…………………………………………….…………...34**

**3.3.3 Алгоритми за филтриране………………………………………………………36**

**Заключение…………………………….…………………………………………………..37**

**Увод**

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

**Проектът включва реализирането на трислойно MVC приложение отнасящо се за и управляващо уеб магазин на сладкарски цех „Сладко изкушение“. Уеб магазина ще представя продуктите на сладкарския цех, като по този начин ще популяризира неговата работа и ще привлича клиенти заради улеснения си достъп до преглед на продуктите. Освен преглед на продуктите на сладкарски цех „Сладко изкушение“, ще бъде възможно да се направи поръчка към него. Това е благодарение на реализирането на функционалността на „пазарска кошница“. Кошницата за пазаруване е много важна функция, използвана в електронната търговия, за да помогне на хората да правят покупки онлайн, подобно на американско-английския термин „кошница за пазаруване“. Аспектът от бизнес към потребител на електронната търговия е най-видимото бизнес използване на World Wide Web. Основната цел на сайта за електронна търговия е да продава стоки и услуги онлайн. Електронната търговия бързо се налага като приета и използвана бизнес парадигма. Все повече и повече бизнес предприятия внедряват уеб сайт, предоставящ функционалност за техните клиенти.**

**Цели и задачи**

**Целта на този уеб-базиран, мулти-технологичен проект е да реализира трислойна MVC ASP.NET Core Web информационна система, която да послужи като средство за управление, реклама и популяризиране на сладкарски цех „Сладко изкушение“. Като приложението ще улесни средата както за бизнес предприятие, така и за неговите клиенти. Ще бъдат интегрирани роли за достъп и определена функционалност спрямо дадена роля. Ще бъдат включени продуктови списъци, които са интегрирани в бази от данни, както и функционалност за направата на поръчка от страна на клиент на предприятието, което ще управлява приложението. Ще бъде постигнато високо ниво на сигурност спрямо клиентските лични данни. Като се спазват всички Европейски стандарти и конвенции за обработка на лични данни и информация от бисквитките. Приложението ще предоставя директна обратна връзка при интеракция от страна на клиента. Също така ще бъде постигнато моментално регистриране и отчитане на поръчките и потребителските акаунти.**

**ГЛАВА 1.**

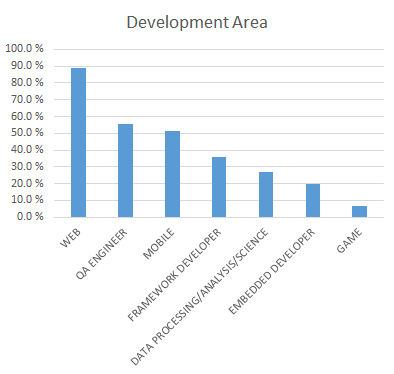
**ПРОУЧВАНЕ**

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

Причините, които налагат нуждата от създаване на този софтуерен продукт, са следните:

* Силния ръст в нуждата от софтуерни решения на бизнес проблеми, в частност на сладкарската бизнес индустрия
* Масово остарялата методология на предприемачите от бизнеса на сладкарите в нашите географски ширини
* Нуждата от популяризация и реклама, свързани със сладкарската дейност, производство и продажбата на сладки изделия

Това е малък пример за предпоставките, довели до създаването на настоящото програмно решение, а и за програмните решения като цяло. Не е тайна, че България е с остаряла или несъществуваща, на места, софтуерна база, но ето, че идва момента, в който водена от бизнес сектора, тя започва изграждането на софтуерни решения за всевъзможни задачи. Това изграждане не е никак лесно, нито пък е сравнително бързо, но с малки и амбициозни стъпки, в правилната посока, се стига наистина далеч. Една такава малка стъпка, както и личен принос към модернизацията на българската софтуерна база, е и този проект за създаването на Уеб Магазин „Сладко изкушение“. В национален мащаб, България има и своите напреднали градове, примери са София, Пловдив, Бургас и др. Техните IT сектори, заедно с бизнеса и индустрията, са изградили своите модерни и високотехнологични методологии и подходи за достигане до крайния потребител или до множество от крайни потребители. За разлика от тях обаче, демографски по-ниско заселените райони и по-малките градове, все още имат доста път да извървят пред себе си. „Сладко изкушение“ е голяма стъпка по пътя им, защото се прави точно с мисъл за едно такова малко предприятие в малкия град. За да може и то да се модернизира и да улесни своята работа, както и досега си с клиентите. Друга предпоставка за създаването на продукта е, че с напредването на технологиите, хората започват да разчитат до голяма част на интернет за почти всичко в своето ежедневие. От извличане на определена информация и обучителни цели, през забавление под формата на интересни, увлекателни и забавни клипчета, та чак до онлайн пазаруване и търсене на продукти. Забелязва се ръст в доверието на хората, живеещи на територията на България, към онлайн пазаруването и онлайн търговията. Това отваря много врати на бизнеса, стига той да намери начини да достигнe до аудиторията от потенциални клиенти. Настоящото уеб приложение -„Сладко изкушение“, е един от тези начини. Една от неговите цели ще бъде именно спомагането на малкия бизнес да достигне по модерен, сигурен и лесен начин да достигне до своята аудитория от потенциални клиенти. Както споменах , обществото има по-високо доверие към интернет услугите. Това е предпоставка настоящия проект да бъде изграден и внедрен. При честото използване на интернет от хората в ежедневието им , „Сладко изкушение“ може да послужи освен за преглед, поръчка и обслужване – за реклама и популяризиране. Това разбира се ще стане, използвайки и други социални мрежи и интернет пространства, но е важно, че ще бъде възможно главно заради съществуването на този уеб проект. Обобщено казано, предпоставките довели до направата на този софтуерен продукт са: нуждата от софтуерни решениа за бизнеса; обновление или създаване на софтуерна база и методологии в българските географски ширини (в частност по-малките населени места); нуждата от средства за популяризиране и реклама на продукти и услуги; Горепосочената информация е в следствие на проучване и търсене на информация за картината в посочения район. В допълнение към информацията прилагам и данни за нуждата и търсенето на използваната технология на настоящия проект – Табл.1. В Табл.1 може да придобием ясна представа колко търсена и необходима на бранша е нашата технология на разработка (Web).



***Табл.1 – Данни от проучване на СофтУни за най-търсените и нужни технологии от бранша***

**1.2. Съществуващи решения и реализации**

**Както вече загатнах по-нагоре, съществуващи решения и реализации на софтуер, подпомагащ бизнеса има. Но главната им внедреност и осъщественост е в по-големите географски ширини на България(големите градове). А по-малките географски райони, в какъвто се намираме ние и за какъвто е предназначен уеб проекта „Сладко изкушение“ имат недостиг, дори липса на софтуерни решения. Включително съществуващите такива са правени преди голям период от време, което прави използваните от тях технологии остарели. Това е в следствие на бързите темпове, с които ИТ сектора се развива и напредва. Технологии от преди 10, а дори и 5 години се смятат за остарели или демоде. Както в текстилната индустрия постоянно се произвеждат дрехи и дрехи от миналогодишни колекции нямат същия, увличащ клиентите, интерес. Да не говорим за времева разлика от да кажем 10 години, там вече има съвсем голяма вероятност дрехите да спадат към различна епоха. Същото е и в ИТ сектора и ИТ индустрията – времето лети, откриват се и се създават нови технологии постоянно, всяка от които с нещо превъзхожда предците си. Дали ще става въпрос за сиигурност , дали за гъвкавост, дали за дизайн и удобство, няма значение, важното е, че предимствата на новите технологии са налице. А „Сладко изкушение“ може да се похвали с най-съвременните, най-удобните, най-сигурните и най-подходящите технологии за изграждане на бизнес инструмент, с помощта на който индустрията ще улесни живота си и ежедневието си, както и ще зарадва своите клиенти. От друга страна пък, настоящия проект освен бърз, сигурен, удобен, автентичен, той изпъква и по друг фактор – евтин и лесен за поддръжка. Той се различава от другите съществуващи софтуерни решения по технологията използвана в него – ASP.NET, за съпоставка, масово се използват PHP, Jscript, HTML, CSS. Нашия проект обединява в себе си модерен подход за разработване на софтуер, горепосочените езици (без PHP), както и технологии за пряка работа с бази от данни. В програмирането това се нарича full-stack разработка или от нулата до завършен и работещ проект с всички необходими функционалности. Но за по-широко обяснение и запознаване с технологиите на проекта ще трябва да погледнете глава 2, точка 5 - (2.5). За целта на този проект се използва средата за разработка на софтуер Visual Studio, версия 2019 community. За разработка на уеб приложение могат да бъдат използвани и други среди за разработка на други езици за програмиране, такива например са Zend Studio и PHP Storm за разработка, използваща езика PHP.**

**ГЛАВА 2.**

**ПРОЕКТИРАНЕ НА СТРУКТУРАТА НА WEB БАЗИРАНОТО ПРИЛОЖЕНИЕ – “Сладко Изкушение”**

**2.1. Функционални изисквания към WEB базиранoто приложение**

Да се реализира уеб приложение за онлайн пазаруване от сладкарски цех „Сладко изкушение“. Приложението трябва да поддържа следните функционалности:

* Добавяне/актуализиране/изтриване на торта: /с Администраторски функции/
  + Каталожен номер на торта
  + Име на торта
  + Размер торта /грамаж/
  + Описание – съставки, количество, хранителна стойност
  + Снимка
  + Цена
  + Дата на вписване в системата
* Добавяне/актуализиране/изтриване на сладкарско изделие: /с Администраторски функции/
  + Каталожен номер на изделие
  + Име на изделие
  + Описание – състав, количество, хранителна стойност, алергени
  + Грамаж
  + Снимки
  + Цена
  + Дата на вписване в системата
* Добавяне/актуализиране/изтриване на мъфини: /с Администраторски функции/
  + Каталожен номер на мъфин
  + Име на мъфин
  + Описание – състав, количество, хранителна стойност, алергени
  + Грамаж
  + Снимки
  + Цена
  + Дата на вписване в системата
* Извличане на информация за един избран артикул
* Извличане на всички артикули по избрана категория
* Извличане на всички артикули по въведено име на артикул
* Да се направят роли за работа с приложението:
  + АДМИНИСТРАТОР
  + Клиент
* Регистрация на клиент:
  + Име
  + Фамилия
  + Електронна поща
  + Телефон за контакт
* Пазаруване /записване на артикули в пазарска кошница/:
  + Номер на клиент
  + Вид артикул – избор от списък /торта, сладкиш, мъфин/
  + Номер на артикул;
  + Количество стока /брой артикули/
  + Единична цена на артикул
  + Дата / час на направения избор;

Пазарската кошница трябва да може да се визуализира на отделен екран, като избраните от клиента стоки са подредени в добре форматиран списъчен вид заедно с техните количества. Да се пресмята и общата стойност на поръчката, която може да се финализира с бутон „Поръчай“. Дизайнът е напълно свободен и авторски. След финализиране на поръчка:

* Пазарската кошница се изпразва;
* Поръчката се записва в отделна таблица в БД на приложението /№ поръчка, списък артикули с техните количества и цени, крайна сума за плащане, номер на клиента, който е поръчал/
* Потребителя да има достъп до преглед на продуктите
* Потребителя да има достъп до поръчване след ауторизация
* Потребителя да има достъп до общодостъпни страници от менюто

Да се изгради трислойно ASP.NET MVC приложение с БД, което поддържа гореописаната функционалност на различни екрани /страници/. Страниците да имат структурата на HTML5 и да са форматирани със CSS3, Java Script. Да се използва HTTP комуникация.

**2.2. Нефункционални изисквания**

* Всички данни за системата (потребители, профили и т.н.) трябва да се съхраняват в базата данни в подходящи таблици. Не се разрешава съхранение на данни от системата във файловата система.
* Автентикацията в ASP.NET Web приложението трябва да става по метода ”ASP.NET Core MVC Authentication” или „Identity System“
* Приложението трябва да обработва клиентските данни по сигурен начин, например по метода „ASP.NET GDPR Compliant Service“
* Както и преноса на данните в address bar-а да става със съответните методи, които не позволяват изтичане или прочитане на личните данни на потребител.
* Работата с и употребата на „бисквитки“ да става в съответствие на правилата и конвенциите за работа с „бисквитки“

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

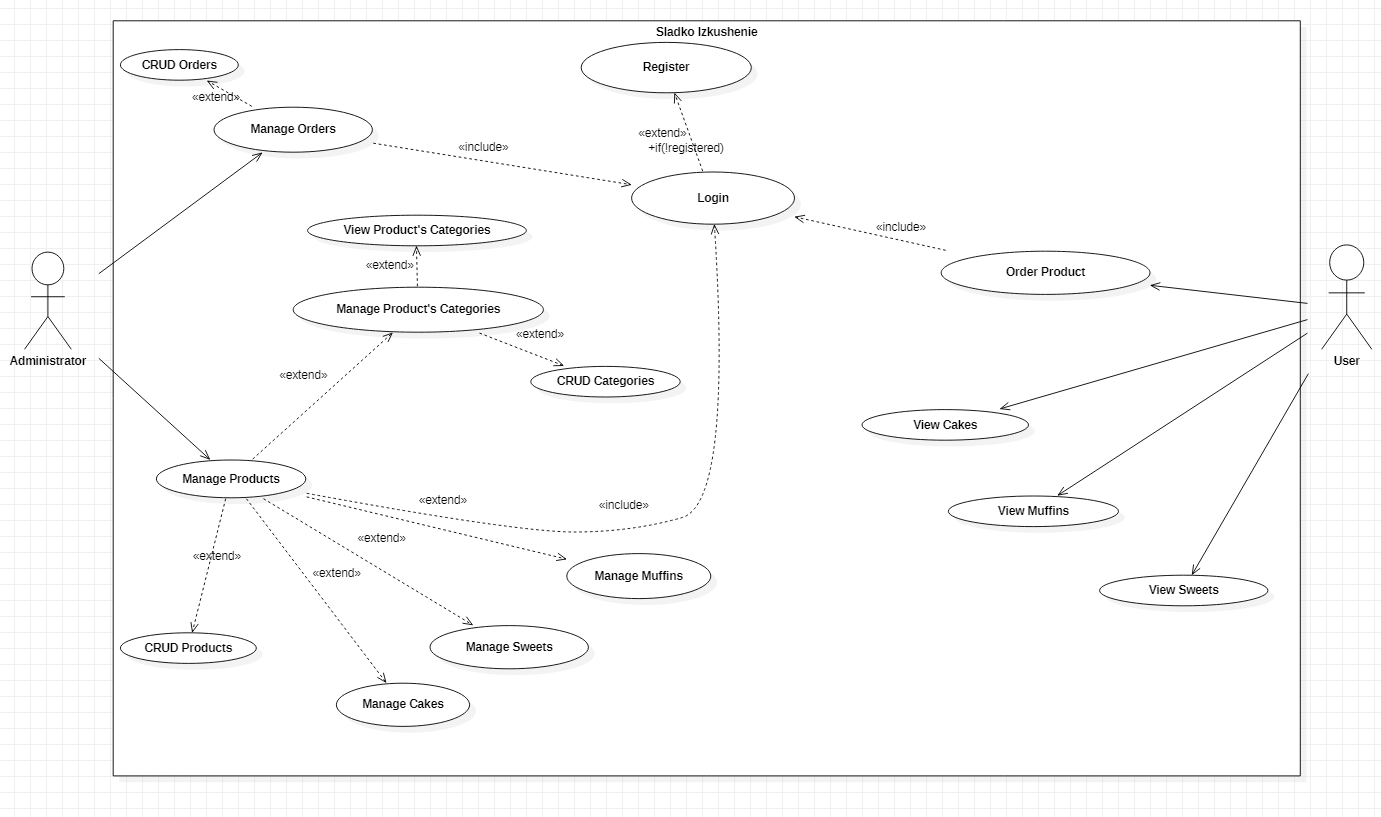
Use Case Diagram или Диаграма на потребителските случаи:

В процеса на проектиране на софтуерния продукт, диаграмата на потребителските случаи е първата диаграма, която се създава от проектантите, когато се започне проект. Тази диаграма позволява да се опишат на най-високо ниво целите на потребителя, които системата трябва да изпълнява. Тези цели не е необходимо да са задачи или действия, а може да са по общи изисквания към функционалността на системата. С други думи това е техника за определяне на функционалните изисквания на една система. Те описват типичните взаимодействия между потребителите и системата, предоставят описание на начина, по който тя се използва. Настоящия проект използва диаграма на потребителските случаи, същата може да бъде видяна по-долу на Фиг.1

* **Структура на системата**

Системата се състои от следните функционални звена:

* Регистрационен модул – регистрация на потребител от тип клиент;
* Потребителски профил – попълване на лични данни на клиента;
* Администраторски профил – идентифицира потребители със специфични функции
* Секция „преглед на продукт“ – позволява на потребителите да разгледат предлаганите продукти
* Секция „добавяне на продукти“ – позволява на администраторите да добавят нов продукт
* Секция „отчет“ – позволява на администраторите да извлекат информация за направените поръчки/поръчани продукти
* **Случаи на употреба**



***Фиг.1. UseCase диаграма на приложението***

Диаграмата на потребителските случаи за уеб проекта „Сладко изкушение“ използва приложението за създаване на UseCase диаграми – StarUML, като се използва безплатната версия на лиценза за употреба. В диаграмата на потребителските случаи имаме две роли – администраторска и клентска.

* Клиент

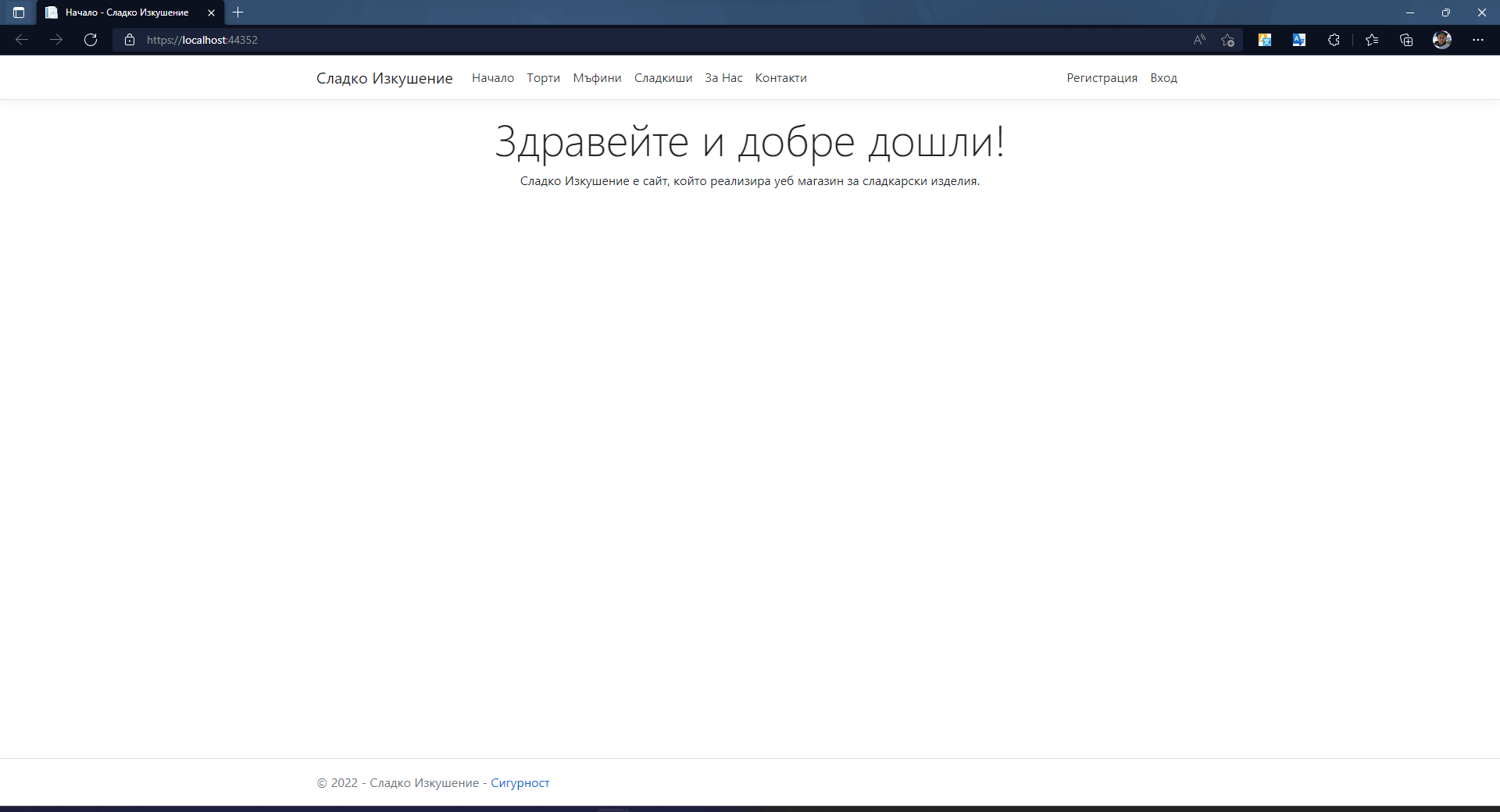
- Клиентът може да преглежда торти, както и техните детайли(име, грамаж, цена и т.н). Може да преглежда мъфини – име, грамаж цена и т.н и накрая, може да преглежда сладкиши с техните детайли. За тези функционалности клиента не се нуждае от вход в системата, но все пак може да влезе при желание. Клиентът може да осъществява и поръчка на продукт, тази функция обаче изисква клиентът да влезе в системата с потребителски акаунт. Ако клиентът няма такъв, то той трябва да премине към регистрационната страница и да си създаде акаунт. Също така клиентът има достъп до общодостъпни страници на уеб приложението като за тях не е нужно, да е влезнал в системата с потребителски акаунт, такива страници са „Начало“, „За Нас“ и „Контакти“

* Администратор

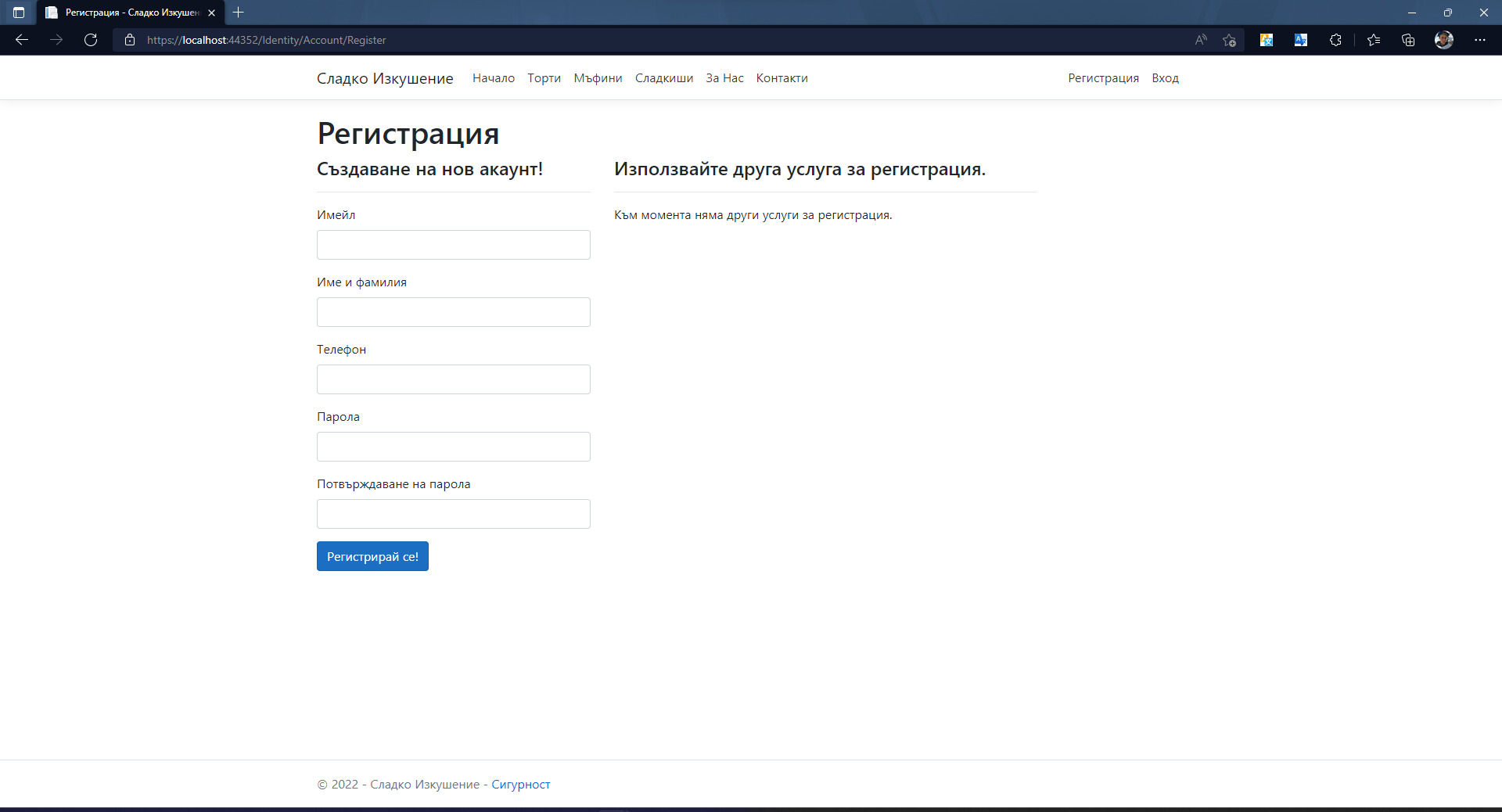
-Администраторът може да менажира продукти, в частност – да менажира торти, да менажира мъфини, да менажира сладкиши. Разширени опции на менажирането са преглед на детайли на съществуващ продукт, създаване на нов продукт, редактиране на съществуващ продукт, както и изтриване на съществуващ продукт. За целта администратора трябва да е влезнал в системата със съответния администраторски акаунт и при повикване на екран с менажиране на продукти той ще бъде ауторизиран да извършва изброените по-горе действия. За разлика от потребителския акаунт, администраторския акаунт се създава ръчно от разработчика. Администраторът може да менажира продуктови категории. И тук разширени опции са прочит на детайли, редактиране, триене и създаване. Последната функционалност на администратора е – да менажира поръчки, тук разширените опции са прочит на детайли, триене и редактиране, възможно е да се създаде поръчка от администратора, но в общия случай няма смисъл, изключение прави аварийно създаване(когато клиент твърди, че е поръчал, а поръчка няма.

**2.4. Потребителски интерфейс**

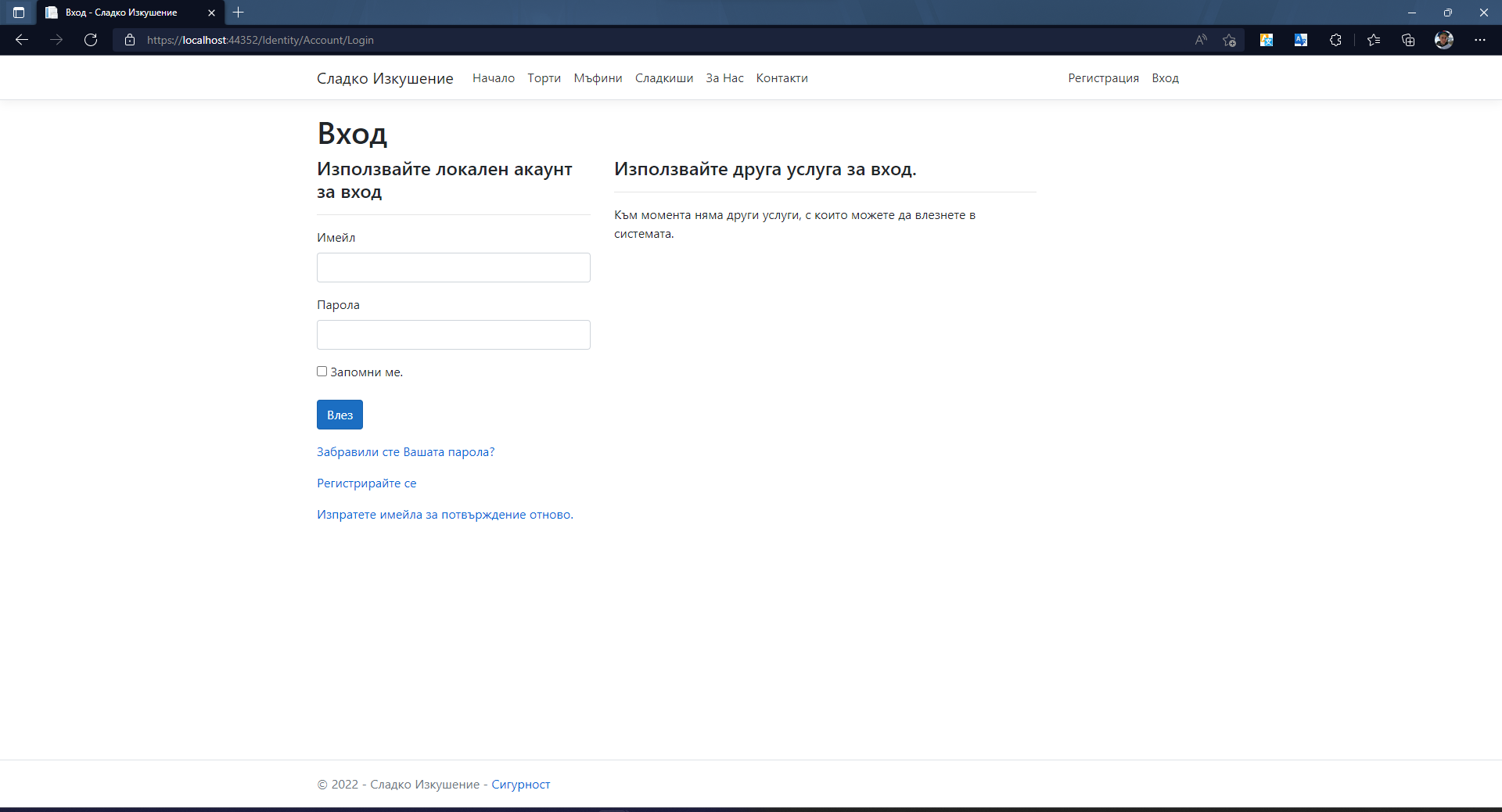
В тази точка ще покажа няколко интерфейса от проекта си, които са в подкрепа на Use Case диаграмата и показно на част от проекта (като нека не забравяме, че проектите са в процес на разработка към момента на писане на теоретичната част и е възможно да има леки забележки. Пълни резултати и последни версии на проекта ще бъдат представени на практическа част). Снимки на интерфейси: 2.4.1, 2.4.2, 2.4.3, 2.4.4, 2.4.5.

****

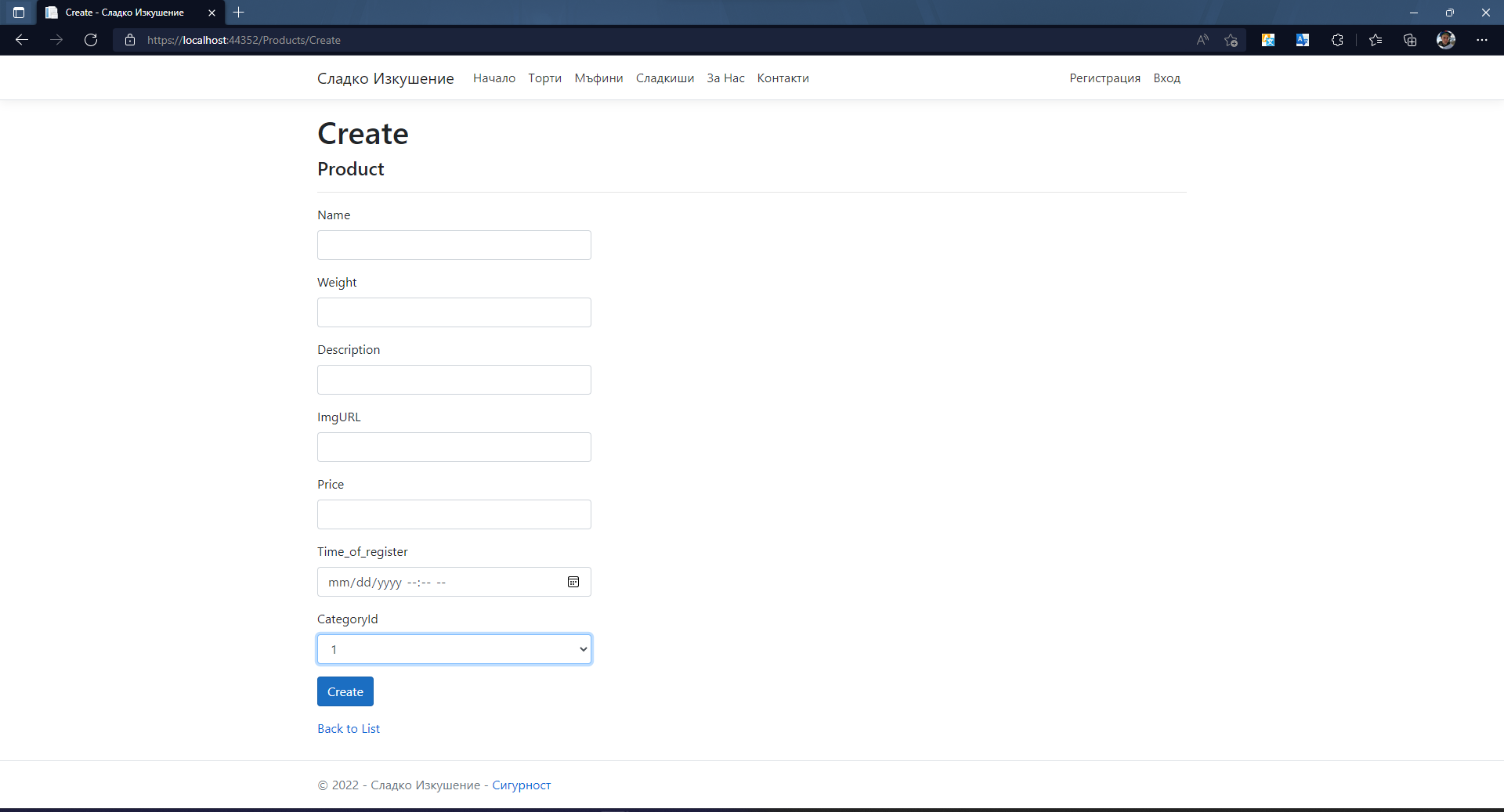
***2.4.1 – Начална страница (Изглед при стартиране)***

****

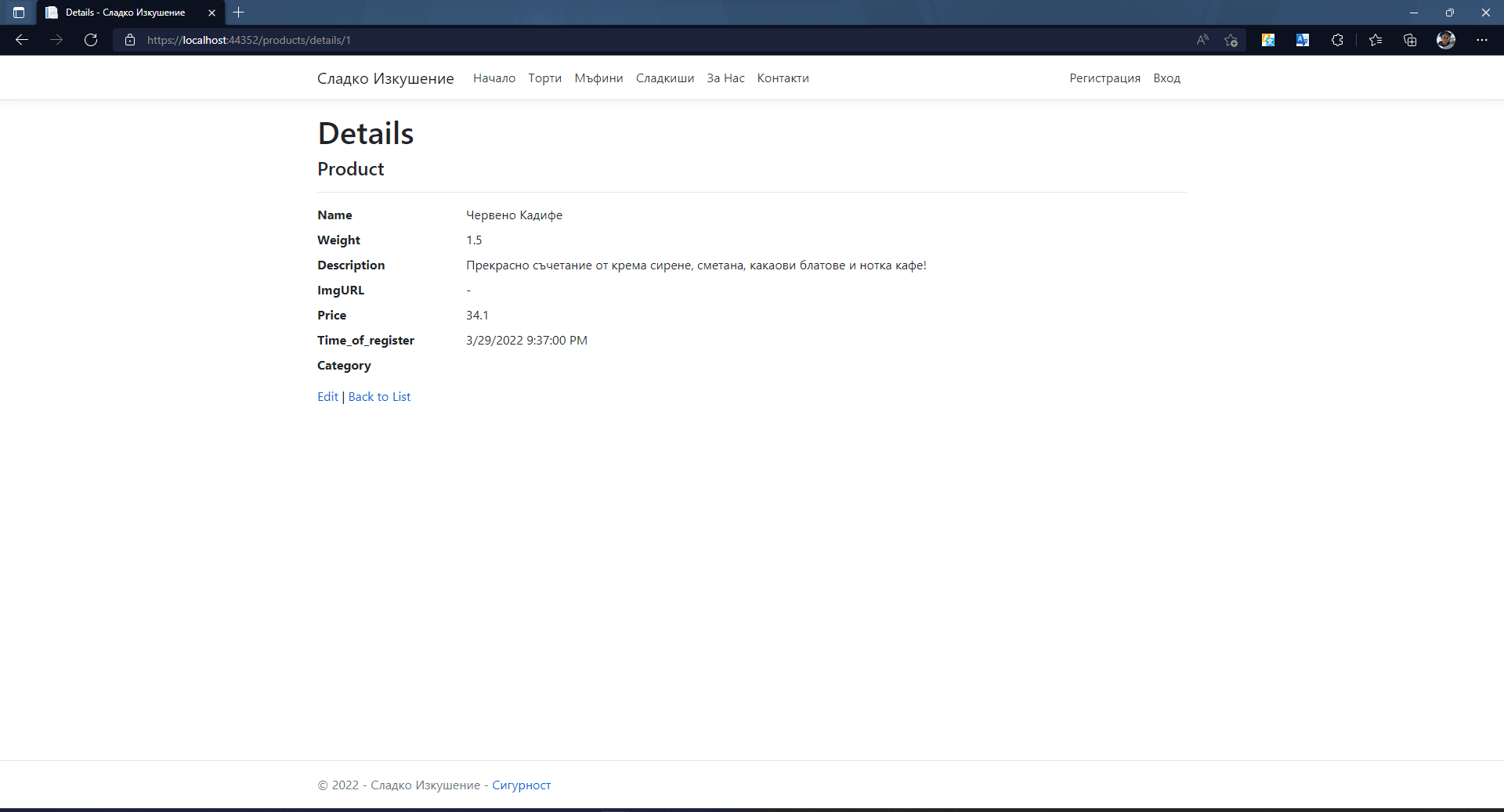
***2.4.2 – Регистрационна форма за създаване на потребителски акаунт***

****

***2.4.3 – Форма за вход в системата, използвайки потребителски или администраторски акаунт***

****

***2.4.4 – Меню за създаване на продукт / с администраторски функции /***

****

***2.4.5 – Представяне на детайли, за даден продукт, на потребителя***

**2.5. Описание на избраната технология и софтуерните средства за разработка на приложението /**технологии: ASP.NET Core MVC, SQL Server Management Studio, Entity Framework; средства: C#, HTML, CSS, JavaScrip, Bootstrap**/**

В съвременния свят има много различни начини и технологии да бъде създадено уеб приложение или да бъде създаден уеб сайт. Повечето от начините са като отделни фрагменти при изграждане на уеб приложение. Например за да направим структура(показване и подреждане на информация) на сайт ни е нужен описателния език HTML, за да включим приятни цветове и допълнителни визуални екстри ни е нужен CSS, за да раздвижим дадена страничката и да я направим динамична ни е нужен Jscript. С тези 3 езика можем да направим фасадната част на даден сайт, но както при къщите така и при сайтовете – фасадата е най-горния и най-явния слой за случаен наблюдател. В случая с къщата освен фасадата имаме стабилни основи, съобразна архитектура и най-накрая идва тя(фасадата). В едно уеб приложение, гореспоменатите езици помагат за изграждането на „фасадата“ му, докато неговите основи и архитектура са базите от данни, както и бизнес логиката, която ги кара да заработят правилно.

Досега говорихме за отделни технологии, сега идва момента на ASP.NET. Това е съвременен начин, които обединява гореспоменатите езици, бизнес логика, език за програмиране и др. За съпоставка – ASP.NET е като строителна фирма,която управлявана по правилния начин от нас програмистите, може да построи къща от основи до фасада. В настоящата точка ще обясня по-обширно технологиите, които използвам, езиците за програмиране и софтуерните инструменти, които са ми били необходими при създаването и изграждането на моя ASP.NET Core MVC проект.

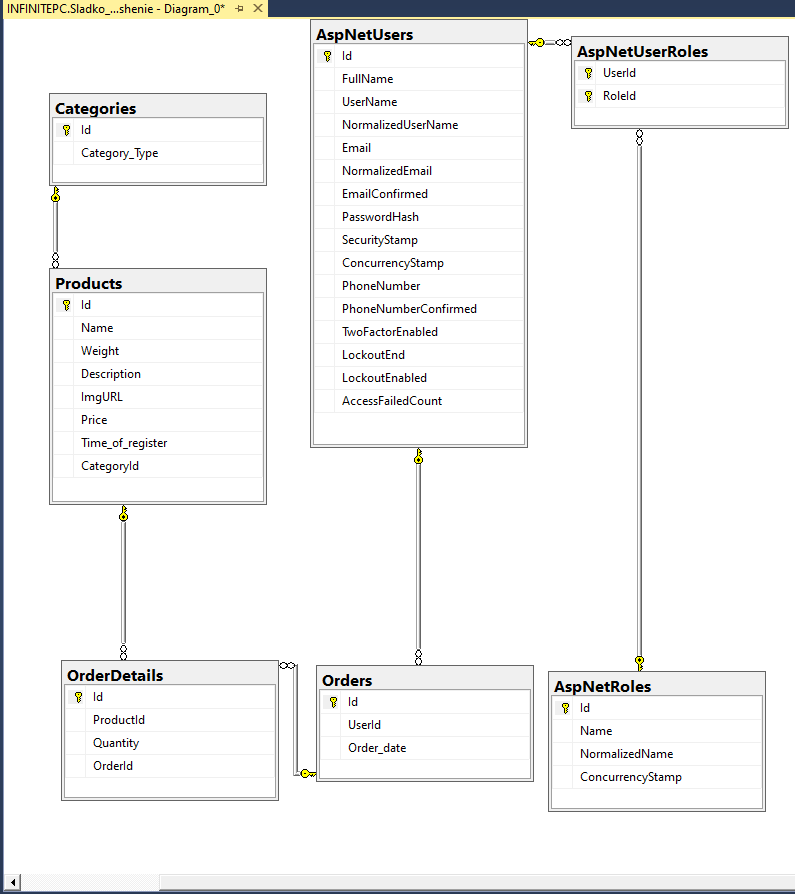
* **C# -** Произнася се Си Шарп. Си Шарп е обектно ориентиран език за програмиране, разработен от Microsoft (Майкрософт), като част от тяхната софтуерна платформа .NET (Дот Нет). Си Шарп е прост, модерен, обектно ориентиран език с общо предназначение. Основа за Си Шарп са С++ (Си Плюс Плюс), Java (Джава) и др. Той е проектиран да балансира мощност (C++) с възможност за бързо разработване (Visual Basic и Java). В същността си, Си Шарп е съвкупност от дефиниции на класове, които съдържат в себе си методи, а в методите е разположена програмната логика – инструкциите, които компютърът изпълнява. Програмите на Си Шарп представляват един или няколко файла с разширение „ .cs “, в които се съдържат дефиниции на класове и други типове. Тези файлове се компилират от компилатора на Си Шарп до изпълним код и в ррезултат се получават асемблита – файлове със същото име, но с различно разширение ( .ехе или .dll)
* **HTML –** е съкращение от Hyper Text Markup Language, което на български език означава „език за маркиране на хупертекст“. HTML е основният маркиращ език за описание и дизайн на уеб страници. Още повече той е стандарт в интернет, а неговите стандарти се определят от международния консорциум W3C(Консорциум на световната мрежа). Текущата стабилна версия на стандарта и версията, която се използва в настоящия проект е HTML 5.0. Описанието на документа става чрез специални елементи, наречени HTML елементи или HTML тагове. HTML елементите са основната градивна единица на кода, който изгражда уеб страниците. Чрез тях се форматира текста и неговите отделни части в рамките на една уеб страница, като например заглавия, цитати, текстови раздели, хипертекстови препратки и други.
* **CSS –** е съкращение от Cascading Style Sheets (Каскадни Стилови Слоеве). Си Ес Ес е език за програмиране и описание на уеб дизайн. Използва се основно за описание на онлайн представянето на уеббазиран документ, писан на езика за маркиране – HTML. Най-често се използва допълнително към чистия HTML, но се прилага и върху XML уебстраници (XML e друг език, които няма да засягаме, поради не използването му в настоящия проект). Спецификацията на Си Ес Ес официално се поддържа от W3C, както при HTML. Преди стандартите за CSS, установени от W3C през 1995г., съдържанието на сайтовете и стила на техния дизайн са писани в една и съща HTML страница. В резултат на това HTML кодът се превръща в сложен и нечетлив, а всяка промяна в част от проекта на даден сайт изисквала корекцията да бъде нанасяна в целия сайт – страница по страница. Използвайки Си Ес Ес промените могат да бъдат нанесени едновременно на всички страници.
* **Jscript –** съкратено от JavaScript, на български – ДжаваСкрипт. Това е интерпретируем език за програмиране, разпространяван с повечето уеб браузъри. Поддържа обектно ориентиран и функционален стил на програмиране. Създаден е в Netscape(браузър, чиято поддръжка е приключила към днешно време) през 1995 г. Най-често се прилага към HTML-a на интернет страница с цел добавяне на функционалност и зареждане на данни. Може да се ползва също за писане на сървърни скриптове, както и за много други приложения. ДжаваСкрипт не трябва да се бърка с Джава(Друг език за програмиране, наподобяващ Си Шарп), съвпадението на имената е резултат от маркетингово решение на Netscape. Джава Сктипт е стандартизиран под името EcmaScript (Екма Скрипт).
* **Bootstrap –** на български – буутстрап. Това е client-side среда с отворен код, която съдържа набор от инструменти за създаване на уеб приложения и уебсайтове. Интересен факт за буутстрап-а е, че той е пуснат през 2011 г. от Twitter, след като е стартирал като затворена библиотека, създадена за вътрешна употреба на компанията. Буутстрап се състои от три части:
* **CSS** – Буутстрап има огромна колекция от класове CSS
* **Компоненти на потребителския интерфейс** ( допълнителни компоненти за HTML, писани на CSS и Jscript
* **JavaScript** – скриптове, плъгини и разширена функционалност
* **SQL Server Management Studio** – това е един от допълнителните инструменти, които спомагат работата с базата от данни на настоящия проект. Студиото за управление на SQL Server е софтуерно приложение, което се използва за конфигуриране, управление и администрация на компоненти в SQL Server. То съчетава широка група от графични инструменти с голям брой текстови редактори, осигуряващи на разработчиците и администраторите всички нива на достъп до сървъра. Водещ елемент в SSMS е Object Explorer, който позволява на потребителите да търсят, избират и да работят с всеки от обектите на сървъра. Версият, която използвам за целите на този проект е Dev Express, която се разпространява с безплатен лиценз за образователни цели и обучение на разработчици.
* **ASP.NET Core –** ASP.NET е рамка за създаване на уеб сайтове и уеб приложения. Тя обединява гореописаните езици HTML, CSS, Jscript и C#, като позволява писането им в едно IDE (Среда за разработка) и то е Visual Studio, версия 2019 community (безплатна за обучение на разработчици). ASP.NET също така позволява използването и създаването на уеб API, както и достъпа до технологии в реално време като уеб сокети. ASP.NET Core (Ей Ес Пи Дот Нет Кор) е новата версия на уеб рамката ASP.NET, насочена главно към .NET Core платформата. От своя страна пък .NET Core платформата е по-новата версия на .NET Framework, една от най-съществените разлики е, че .NET Core е мулти-платформена, което и позволява да върви идеално гладко, както на Уиндоус, така и на МакОС и Линукс, също и на мобилни операционни системи. Казано на кратко: има междуплатформена интеграция. ASP.NET Core е безплатно разпространявана софтуерна рамка, с отворен код, която позволява изграждането на облачно-базирани приложения, като уеб приложения, IoT(Интернет на нещата) приложения и мобилни бекендове(сървърна част и бази от данни). ASP.NET Core, подобно на фреймуърка си - .NET Core, е проектиран модулно с минимални режимни разходи и след това могат да се добавят други по-разширени функции като например NuGet пакети според нуждите на проекта, върху който работим. Това води до висока производителност, изисква по-малко ресурси от компютъра(РАМ, дисково пространство, процесорно време и др.) и лесна поддръжка на готовия продукт.
* **MVC -** MVC е съкращението на Model-View-Controller, на български Модел – Изглед – Контролер. Това софтуерен начин на писане, който датира от 70-те години на миналия век, когато GUI (Graphical User Interface, на български – Графичен Потребителски Интерфейс) е в доста ранен стадий на развитие. MVC се стреми да раздели приложението на три области:
* **Модел** – това е мястото на бизнес логиката в приложението. В уеб приложение тази част представлява таблицата на базата данни или други обекти, с които приложението работи.
* **Изглед** – това е визуаното представяне на нашия модел. В уеб приложение това е страницата, която показва нашия модел на потребителя, без значение дали е формуляр за въвеждане на данни, или показване на вече съществуващи данни от базата, или пък е комбинация от двете. Важно е, обаче, изгледа да не показва всички данни от модела(някой от тях не са предназначени за потребителя), както и че моделът може да има повече от един изглед.
* **Контролер –** контролерът обработва цялата комуникация между модела и изгледа. В уеб приложение методите на контролера се зареждат при стартиране на приложението или при натискане на някой линк/бутон. След успешно преминаване на даден метод, контролера актуализира данните в модела, ако това е необходимо, и след това изкарва нов изглед (или действие, но повече за това по-късно) , отново, ако е необходимо.
* **ASP.NET MVC –** досега говорехме за MVC като цяло, а сега ще се фокусираме върху употребата му в ASP.NET MVC уеб приложения. Майкрософт представи ASP.NET MVC през 2007г., като първата стабилна версия е налична през 2009г. Важно е да отбележим, че реализацията на MVC е open source (с отвроен код), като Майкрософт я разпространява под Apache License 2.0, което позволява преглед и модификация на рамката и дори преразпределяне на промените. ASP.NET компонентите първоначално са пуснати като рамка със затворен код през 2002г. В последствие Майкрософт решава, че желае да направи рамката общодостъпна за всички разработчици и прави рамката с отворен код. Това се случва около 2016г., след което има множество подобрения, нови версии и все по-голяма бързина с всяка нова стабилна версия. ASP.NET MVC включва нови тендиции в разработката на уеб приложения, притежава много добър контрол върху HTML и дава възможност за създаване на всякакви приложения. Платформата може да бъде много лесно тествана и допълвана, защото е изградена от оделни модули, изцяло независими един от друг. Чрез платформата се създават цялостни приложения, които се стартитат, а не единични скриптове (като при PHP например).
* **Entity Framework –** на български – „ентити фреймуърк“. Ентити фреймуърк е сет от технологии в ASP.NET, които поддържат работата със софтуери за управление на бази данни. Ентити фреймуърк позволява мапване(напасване) между релационна база от данни и обектно-ориентиран модел чрез подходите „database first“ и „code first“ (съответно – първо база и първо код) и предоставя мощен обектно-ориентиран подход за заявки към базата данни и извършване на CRUD операции (Менажиращи операции). EF предоставя както допълнително ниво на абстракция, така и лесен начин за обработка на данните от базата.
* **Entity Framework Core –** Ентити Фреймуърк Кор е лека, разширена платформа с отворен код и кросплатформена версия на популярната тахнология за достъп до данни на Entity Framework. EF Core може да служи като обектно-релационнен картограф (ORM), който:
* Позволява на .NET разработчиците да работят с бази от данни, използвайки .NET обекти.
* Елиминира необходимостта от по-голямата част от кода за достъп до данни, който обикновено трябва да бъде написан.

**ГЛАВА 3.**

**ПРОГРАМНА РЕАЛИЗАЦИЯ НА WEB БАЗИРАНОТО ПРИЛОЖЕНИЕ – „Сладко Изкушение“**

* 1. **Структурата на базата от данни**

***3.1.1 – Извадка на E/R Диаграмата отнасяща се към структурата на базата от данни***

****

Е/R Диаграмата, или Entity/Relational Диаграма, е диаграма описваща структурата, връзките и отношенията на таблиците от дадена база от данни. Е/Р диаграмата на уеб проекта „Сладко изкушение“ е изготвена, използвайки SQL Management Studio, това позволява да се онагледи базата на приложението и да се провери дали няма някакви проблеми на етап – „създаване на база данни“. Понеже базата на нашия проект е изготвена с технологията на Ентити Фреймуърк – “code first”. Което означава, че кода на C# е транслиран от описанието на класове в АСП към таблици на базата данни.

***3.1.2 – Описание на структурата на базата данни***

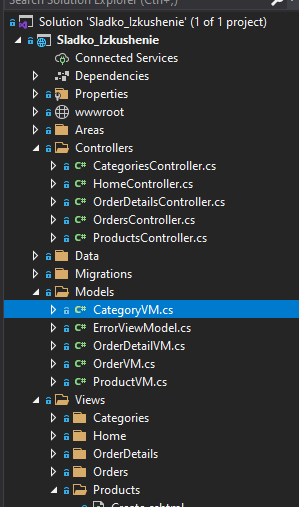
Базата данни се състои от 7 отделни таблици, като между различните таблици имаме различни връзки (връзките могат да бъдат 1 към много, много към 1, 1 към 1):

* 1 към много – това е връзка, при която 1 запис от настооящата таблица може да се среща много пъти в друга таблица под формата на неговото ИД.
* Много към 1 – това е връзка, при която много записи под формата на ИД се окачествяват на 1 запис от отделна таблица.
* 1 към 1 – това е връзка, при която един единствен запис от една таблица отговаря на друг един единствен запис от друга таблица. Тази връзка е уникална връзка, но такъв тип връзка рядко се използва.

Описание на таблиците на базата данни на уеб проект „Сладко изкушение“:

* **Categories** таблицата е проста таблица, понеже тя се състои само от два записа, един за ИД (сервизно необходимо поле, което пази данни от целочислен тип в автоматично нарастващ ред) и един за тип категория (поле, което пази данни от текстов формат). Таблицата има взаимодействие от типа 1 към много с таблицата на продуктите.
* **Products** таблицата е главна таблица, която държи данните за продуктите в себе си. Тя има ИД поле ( отново целочислено и авто-нарастващо), поле за име на продукта (то пази текстова информация), поле грамаж (число с плаваща запетая), поле за описание (пази текстова информация, с по-широк диапазон), поле за път към снимка на продукт(пази текстова информация), поле за цена(пази информацията под формата на число с плаваща запетая), поле за дата на регистриране на продукта (пази информацията във формат за дата и време) и накрая поле-ключ за връзка с таблицата на категориите. Таблицата има взаимодействие от тима много към 1 с таблицата на категориите и взаимодействие от тима 1 към много с таблицата за детайли на поръчка.
* **OrderDetails** таблицата е свързваща таблица на други две – Products и Orders. Тя има ИД поле, поле-ключ за връзката ѝ с таблицата на продуктите, поле за количество (пази данни от целочислен тип), поле-ключ за връзката ѝ с таблицата на поръчките. Таблицата има взаимодействие от типа много към 1 с таблицата на продуктите. Същото е и взаимодействието с таблицата на поръчките.
* **Orders** таблицата е таблицата, в която се пазят данни за направените поръчки. Тя има ИД поле, ключ-поле към таблицата с потребители и поле за дата на поръчка (пази се информация във формат за дата и час). Таблицата има взаймодействие от типа 1 към много с таблицата за детайли на поръчка и взаимодействие много към 1 с тази на потребителите.
* **AspNetUsers** е таблица, която АСП.НЕТ предоставя за внедряването на системата за потребителски акаунти. Макар и таблицата да е създадена от АСП автоматично, за целите на уеб проекта „Сладко изкушение“, тя е променена като са добавени ръчно някои полета. Както и самата таблица е конфигурирана за да работи и взаймодейства с останалите таблици, които съм създал за нуждите на проекта. Използвал съм технологията за наследяване в C#. АспНетЮзърс има ИД(отново сервизно и задължително поле, но този път то е от дълъг автоматично генериран стринг), поле за пълното име на потребителя(пази текст), поле за потребителско име(пази текст), поле за имейл(пази текст), поле за парола, което е във хеширан вид ( тоест криптиране на паролата, така , че дори и в базата тя не може да бъде видяна или прочетена), поле за телефонен номер(пази информацията в целочислен вид). Таблицата има и още доста сервизни полета като например NormalizedEmail и NormalizedUserName, те оеднаквяват въведените от потребителя потребителско име и имейл откъм голяма/малка буква, като по този начин улесняват работата на приложението с тези две полета. Но за останалите сервизни полета няма нужда от подробности, защото те касаят единствено разработчика в процеса на разработка. Таблицата има взаимодействие от типа 1 към много с таблицата на поръчките и взаимодействие от типа много към 1 със свързващата таблица AspNetUserRoles.
* **AspNetUserRoles** е таблица, която е свързваща за AspNetUsers и AspNetRoles.
* **AspNetRoles** е таблица, която съдържа информация за ролите на ползвателите на приложението. Тя има ИД, поле за име на ролята(пази текст), както и 2 сервизни полета. Таблицата има взаимодействие от типа 1 към много с свързващата таблица AspNetUserRoles.
  1. **Структура на MVC приложението**

Както вече споменах, МVC е начин за разделяне на програмния код в отделни направления – М като модел, където стоят класовете, от които се ражда базата данни, V като изглед, където стои логиката за показване на данни на крайния потребител, с възможност за филтриране на показваното, С като контролер, където се извършва бизнес логиката и връзката между базата и изгледа.



***3.2.1 – Визуализация на разделението на уеб проект „Сладко изкушение“***

В снимка 3.2.1 може да бъде видяно раделението на уеб проекта „Сладко изкушение“ по MVC модела като в контролерс папката има 5 контролера нужни за работата на приложението. В папката модълс стоят 5 ViewModel-a, които спомагат филтрирането на данни към крайния потребител. В папката на изгледите имаме 5 подпапки, съдържащи изгледите на всеки контролер. Обикновено по 5 изгледа за всеки контролер - Index, Create, Edit, Details, Delete, като според нуждата те могат да са повече или по-малко. В папката Data стоят класовете, от които е създадена базата данни. Ще дам примерен код за реализирането на таблицата с продуктите:

Така изглежда всяка таблица под формата на код, от тук с помощта на Entity Framework се създава базата.

namespace Sladko\_Izkushenie.Data

{

public class Product

{

public int Id { get; set; }

public string Name { get; set; }

public float Weight { get; set; }

public string Description { get; set; }

public string ImgURL { get; set; }

public float Price { get; set; }

public DateTime Time\_of\_register { get; set; }

public int CategoryId { get; set; }

public Category Category { get; set; }

public ICollection<OrderDetail> OrderDetails { get; set; }

}

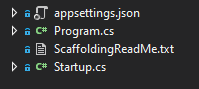
}

В едно MVC приложение, както можем да видим на снимката по-горе имаме и папка Areas. Areas (области) са отличителен белег на Асп.Нет, използвани за организиране на свързана функционалност в група като по отделно:

* Имена за рутиране (routing)
* Папкова структура за изгледите и Razor станиците

Използвайки областите се създава йерархия за причината на рутирането като се добавя друг раут параметър – area. Областите доставят начи за разделяне на АСП.Нет приложението на по-малки функциониращи групи със свой собствен сет от Razor Pages, контролери, изгледи и модели. Областта е ефективна структура в приложението. В АСП.НЕТ проект, логическите компоненти като страници, модел, контролер и изглед са пазени в различни папки. АСП.Нет кор използва конвенции за именоване за да създаде връзка между тези компоненти. За голямо приложение като „Сладко изкушение“ е преимуществено да използва разделение на приложението в отделни области на функционалността. В папката области също се намират и файловете за контрол над визуализацията на логин/регистър формите.

Друга папка, част от структурата на MVC, е папката wwwroot. Там се намират всички статични файлове на проекта като снимки, файлове и др. Също от там се контролира дизайна на приложението, посредством на CSS/Html файловете, както и Bootstrap класовете.



Останалата част от структурата на MVC уеб проекта се състои от няколко файла, необходими за правилна конфигурация и правилно интегриране на всички ресурси, необходими за работата му и те са:

* **Appsettings.json** - тук се пише connection string (текста за връзка) към базата данни, настройва се към кой сървър да бъде вързан, посочва се Trusted\_connection за уведомление на сървъра да позволи връзка с приложението ми.
* **Program.cs** – в този файл се стартира и „построява“приложението. Извикват се готови автоматизирани методи, които събират отделните ресурси и ги обединяват в едно, като по този начин приложението се „построява“.
* **Startup.cs** - в този файл се конфигурират различни услуги и „services“– сървиси. Тук се добавят и отделни функционалности на кода.

**Описание и конфигуриране на моделите за работа и обмен на данни между контролери и изгледи**

Моделът както казах, се използва за бизнес логиката. В моето уеб приложение тази част от приложението представлява например таблица на базата данни или всеки друг обект, с който приложението работи. Следва пример на View Model – „изглед модел“ от проекта:

namespace Sladko\_Izkushenie.Models

{

public class ProductVM

{

public int Id { get; set; }

[Required(ErrorMessage = "Въведете името на продукта!")]

public string Name { get; set; }

[Required(ErrorMessage = "Въведете грамаж!")]

public float Weight { get; set; }

[Required(ErrorMessage ="Въведете кратко описание!")]

public string Description { get; set; }

public string ImgURL { get; set; }

[Column(TypeName= "decimal(10,2)")]

public float Price { get; set; }

public DateTime Time\_of\_register { get; set; }

public int CategoryId { get; set; }

public Category Category { get; set; }

public string UserId { get; set; }

public int Quantity { get; set; }

public ICollection<OrderDetail> OrderDetails { get; set; }

}

}

Тъй като технологията се нарича MVC, използването на (View) модел определено е най-чистият и най-често срещания подход за предаване на данни в View. Вю моделите спомагат преноса на данни от чистия модел , по който е създадена базата, от папката Data, през контролерите и накрая достигат до Изгледите. Във фрагмента код, който се намира по-горе е показано как се описва един вю модел. Подобно на чистия модел се описват отделните полета, като се посочва тяхното ниво на достъп - public, private, protected, internal, след което се посочва типа на данните, които ще съхранява полето на класа. След това се изписва името на полето, като се спазват конвенциите за писане на пропърти, затова всяко име започва с главна буква. И накрая чрез get и set се показва, че това е пропърти на полето. Това позволява данните от базата да бъдат четени и поправяни.

ProductVM productVM = new ProductVM()

{

Id = (int)id,//product.Id,

Name = product.Name,

Weight = product.Weight,

Description = product.Description,

ImgURL = product.ImgURL,

Price = product.Price,

Time\_of\_register = product.Time\_of\_register,

CategoryId = product.CategoryId,

Quantity = 1

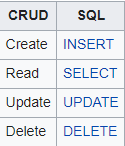
};

return View(productVM);

От фрагмента код по-горе става ясно, как се конфигурира преноса на данни от контролера към вю модела и изгледа. Създава се нов обект от тип класа ProductVM. Чрез него, и по-точно чрез свойствата на полетата му, се прехвърлят данните, от които има нужда класа. Накрая се връща интерфейс, под формата на изглед, на който подаваме информацията от обекта на вю модела.

* **Описание на CRUD операциите- особености на оделните ACTION**

В програмирането CRUD означава Create, Read, Update, Delete (Създаване, Четене, Актуализиране, Триене). Това са четирите основни операции, които могат да се упражняват към даден източник, в случая към базата данни. Също така CRUD понякога се използва за да опише конвенциите на потребителския интерфейс, които улесняват гледане, търсене и промяна на информацията, използвайки компютърно-базирани форми и доклади. Акронима CRUD е препратка към големи операции, които са имплементирани в базата данни. Всяка буква в акронима може да бъде свързана със стандартни SQL заявки.



***Снимката показва съответствието на CRUD операция със SQL заявка.***

* **C –** Create, създаване. Това е начин за създаване на нов обект. В уеб проекта „Сладко изкушение“ има два Action метода, за създаване:

public IActionResult Create()

{

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

return View();

}

Първият метод извежда празна форма, за попълване, в изгледа, при интеракция на потребителя. Този метод използва GET HTTP заявка за комуникация със сървъра.

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Create([Bind("Id,Name,Weight,Description,ImgURL,Price,Time\_of\_register,CategoryId")] Product product)

{

if (ModelState.IsValid)

{

\_context.Add(product);

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

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

return View(product);

}

Вторият метод се активира, след успешно попълване на формата от потребителя. Той обира информацията от полетата и с POST HTTP заявка изпраща информацията към сървъра.

* **R** – Read, четене. Това в моя проект съответства на Details (Детайли за продукта). Тук се осъществява връзка с базата, извличане на данните и показването им в изглед. Action метода, използван в моят проект за четене на детайли е само един.

Action метода отново използва HTTP GET заявка за извличане на данните. В фрагмента код по-долу има показно на Action метода за показване на детайли.

* **U –** Update, актуализиране. В моя проект тази операция съответства на Edit. Подобно на операцията за създаване, операцията за поправяне също ползва два Action метода, един за зареждане на форма, която показва моментното състояние на обекта ( данните, които съществуват в базата данни) и един за подаване на нови данни, които да заместят старите. Отново имаме HTTP GET за зареждането на съществуващите данни и HTTP POST за подаване на данните към базата данни.

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();

}

//тук стои прехвърлянето на данните през вю модел,но то вече е показано малко по-горе

return View(productVM);

}

* **D** – Delete, триене. В проекта тази операция се асоциира с триене. Подобно на операцията за показване на детайли, операцията за триене има само един Action метод, който по ИД на обект, ходи и трие обекта от базата. Action метода използва HTTP GET за прихващане на обекта и неговото изтриване.

Понеже за обяснението на CRUD операциите използвах на няколко места Action метод, сега ще обясня какво наричаме Action метод.

Action метод, както и името му говори, е метод, който предизвиква действие. Всеки контролер има своите Action методи, те най-често изкарват на екрана на потребителя, изглед с дадена информация. Тези методи имат стандартизиран вид, но при работа с бази данни използваме определен вариант на метода. Използваме го в следния вид:

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

{

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

return View(product);

}

Този вариант на метода използва асинхронна технология. За разлика от обикновен ActionMethod, този използва Task<IActionResult> структура заедно с async маркера и await маркера пред определено действие. Асинхронните методи позволяват по-бърз достъп и работа с базата данни при едновременно свързване на повече хора към приложението.

* **Работа с View моделите. Разлика с използването на ViewBag и ViewData**

Вю моделът представя данните които искам да изведа на изгледа, без значение дали става въпрос за статичен текст или за полета за въвеждане на данни, които могат да бъдат добавени в базата данни (или поправяни). Вю моделът е различен от основния модел, който се ползва за създаване на базата данни. Той е модел, който е предназначен да бъде ползван главно от изгледа. Вю моделите се различават от основните модели и по това, че вю моделите съдържат информацията само от пропъртитата на основните модели. Тоест те нямат пряк достъп до базата, което от своя стана означава доста по-високо ниво на сигурност при пренос на данни.

* ViewBag и ViewData – това са контейнери за пренос на данни. Те са алтернатива на строго въведения подход за предаване на данни към изгледа. В тях може да се добавят неща от контролера и след това автоматично да имам досъп до съхранените в тях данни от моите изгледи. Основната разлика между ViewBag/ViewData и преноса на данни през ViewModels ( каквито използвам и аз) е, че няма проверка по време на компилиране на тези свойства и няма помощ от IntelliSense( вграден инструмент за проверка на код във Visual Studio) при записването им. С други фуми, мога лесно да напиша грешно име на свойство и няма да забележа, докато не опитам да използвам изгледа. Следователно трябва да използвам ViewData/ViewBag само за много малки количества данни. Трябва обаче да помним, че не е препоръчително да използваме ViewData/ViewBag вместо модели за данни /ModelVM/. Свойствата ViewData и ViewBag се отнасят до едно и също нещо, което е речник на ключове и стойности (обекти). Въпреки това ViewBag е представяне на динамични обекти на речника, което позволява да получа достъп до данните, сякаш са свойства на обекта ViewBag, вместо записи в речник. Аз имам свободата да използвам и двете свойства взаимозаменяемо, тъй като те винаги се отнасят до едни и същи данни, така че за сравнение те са почти еднакви.
* **Описание на Razor Pages и Razor Views (TagHelpers и HtmlHelpers)**
* **Razor Views**

Докато контролерът обработва връзката между back-end и front-end, изгледът е визуалния резултат от действие на контролера. Така че, когато искам да представя на потребителя нещо визуално в уеб приложението ми, то се поставя във View. Изгледът съдържа маркиране (HTML) и Razor код (C#) и e визуално представяне на моят модел. С други думи, контролерът генерира обект на модела и след това го предава на изгледа, който след това използва модела, за да представи визуално съдържанието на модела на потребителя. Изгледът е просто HTML файл с добавена поддръжка за Razor код. Той използва разширението .cshtml, за да посочи тези допълнителни възможности. Във един такъв файл (Рейзър Вю) за да оточня за компилатора, че пиша C#(Razor) код използвам „@“ пред всеки ред C# код. Razor View Engine (Рейзър Вю „Двигател“) е средство за маркиращ синтаксис, което помага да пишем HTML и код от страна на сървъра, използвайки C#.

* **Razor Pages**

Рейзър страниците са ново средство за работа в ASP.NET Core 2.0 и нагоре. Те ни снабдяват с по-прост начин на организация на кода в ASP.NET Core приложенията като моето, позволявайки имплементирането на логиката и вю моделите по-близко до имплементацията на основните модели. Рейзър страниците, освен да помагат с организацията на проекта, са и мястото, където се взимат готови елементи на приложението от Identity System. Веднъж взети , те остават там, позволявайки да им се нанасят промени и рефакториране на кода.

* **Tag Helpers**

Таг хелпърите позволяват на сървърен код да участва в създаването и render-ването на HTML елементи в Рейзър файлове. Например, вградения ImageTagHelper може да залепи номер за версия към имеро на снимка. Когато снимката се промени, сървъра генерира нова уникална версия за снимката, така че на клиентите им е гарантирано, че ще получат настоящата снимка ( вместо останала стара версия на снимката от кеш паметта). Има много вградени таг хелпъри за основни задачи – като създаване на форми, линкове, зареждане активи и още. Има дори още таг хелпъри достъпни в Гит Хъб репота и като NuGet пакети. Таг хелпърите са написани на C# и целят HTML елементи базирани на име на елемента, неговите атрибути или неговия таг „родител“. Таг хелпърите намалят ненужните преходи между HTML и C# в рейзър изгледите. В много случаи HTML хелпърите предлагат алтернативен подход към специфичен таг хелпър, но е важно да се отбележи, че таг хелпърите не заменят изцяло HTML хелпърите и няма еквивалентен таг хелпър за всеки HTML хелпър.

* **HTML Helpers**

HTML хелпъри използвам във изгледа за да рендирам HTML съдържание. HTML хелпър в повечето случаи е просто метод, който връща стойност. АСП.Нет приложение може да бъде построено изцяло без помощта на HTML хелпъри, но те улесняват работата ми. Във следващият фрагмент от код ще опитам да онагледя и разгранича таг хелпърите от HTML хелпърите:

<div class="form-group">

<div class="checkbox">

<label **asp-for**="Input.RememberMe">

<input **asp-for**="Input.RememberMe" />

@Html.DisplayNameFor(m => m.Input.RememberMe)

</label>

</div>

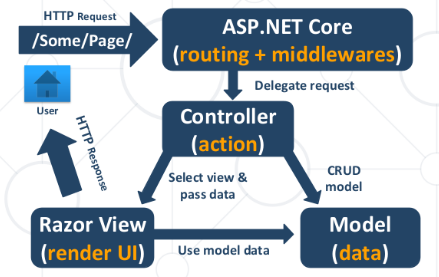
</div>

В фрагмента има примери и за двата вида хелпъри, „@Html.DisplayNameFor()“ е HTML хелпър, докато “<label **asp-for**="Input.RememberMe">” е пример за таг хелпър.

* 1. **Използване на технологии и алгоритми**
* **Как работи схемата Routе Engine**

В ASP.NET Core MVC се използват така наречените middleware за навигиране/рутиране на клиентските заявки. Middleware е софтуер, който се намира посредата между клиента (клиентска заявка от клиент устройство) и сървъра. Като този софтуер може да бъде разделен на отделни слоеве, които посрещат клиентската заявка. Route (Пътя) описва как искания URL път трябва да достигне до Контролера и/или Action-а. Има два вида рутиране – конвенционално и чрез атрибути. Конвенционалното рутиране се описва в Startup.cs > app.UseEndpoints() и неговия вид в адресната лента е „/Controller/Action/?id“. Рутирането чрез атрибути използва сет от атрибути за да ни насочи action директно към шаблонен път.Това означава, че за да използваме рутиране чрез атрибути поставяме атрибутт като HttpGet(“Name”) пред Action Method и когато се обърнем към /Name ще бъде повикан съответния Action Method.

***3.3.1 Схема на Route Engine***

****

При получаване на Http заявка приложението чете заявката, като първо разшифрова за кой контролер е предназначена, съответно за кой негов Action и накрая, ако има ИД прочита и него. След това middleware пренасочва към разшифрования контролер/метод/?ид там action-a извършва действие ( най-често CRUD) с модела, променя или извлича информация, след което праща тази информация към изгледа, където тя се „опакова“ и придобива визуални оформления от изгледа. Накрая приложението връща Http отговор към клиента. ASP.NET Routing използва таблица с маршрути за обработка на входящи заявки. Тази таблица с маршрути се създава, когато уеб приложение „Сладко Изкушение“ стартира за първи път. Когато ASP.NET приложение стартира за първи път, се извиква Program.cs, който извиква Startup.cs, където се създава таблицата с маршрути по подразбиране. Всеки Route има име и шаблон, по който търси съответствие с URI от входящата HTTP заявка.

app.UseEndpoints(endpoints =>

{

endpoints.MapControllerRoute(

name: "default",

pattern: "{controller=Home}/{action=Index}/{id?}");

endpoints.MapRazorPages();

});

В шаблона има 3 секции, отделени с наклонена черта „/“. В първа секция се търси съответствие с контролер, във втора секция е извиквания метод от контролера /Action/, а в трета секция се посочва параметър, като този метод няма такъв описан в дефиницията си. Никоя от трите секции не е задължителна, затова има посочени стойности по подрабиране.

* **Как работи шаблона на Controller с GET и POST заявки към Actions. *Извикване* на конкретни View, използвайки подходящ Model.**

В ASP.NET MVC контролерът е точно като всеки друг клас, така че има разширение на .cs файл и изглежда като всеки друг .NET клас. Има обаче няколко неща, които го правят разпознаваем от .NET рамката като MVCконтролер:

* Обикновено се поставя в папка, наречена „Controllers“ в корена на проекта
* Той наследява от Microsoft.AspNetCore.Mvc.Controller (или от един от собствените ми класове, които след това наследява класа Microsoft.AspNetCore.Mvc.Controller)
* Името на класа обикновено завършва с думата Controller, например „HomeController“ или „ProductsController“

При наследяване на класа Microsoft.AspNetCore.Mvc.Controller, получавам допълнителна функционалност, която може да се използва за целите на MVC, например възможността за връщане на Views/Partial Views. Също така класът Controller има достъп до информация, свързана с HTTP, като низа на заявката, благодарение на свойството HttpContext на класа Controller. С други думи се превръща обикновен С# клас в клас за уеб, което ми позволява да правя неща в мрежовата комуникация.

**Actions**

Методите на клас Controller се наричат действия- метод обикновено съответства на действие във приложението ми, което след това връща нещо на браузъра/потребителя. Това всъщност са обикновени .NET атрибути, които казват на .NET рамката как може да се осъществи достъп до действие. Без тези атрибути може да се осъществи достъп до действие чрез всички възможни HTTP методи (най-често срещаните са GET и POST):

Действието Edit може да бъде достъпвано само с GET заявка. Това е допълнително предимство, защото позволява да имаме няколко метода с едно и също име, стига да не приемат един и същ метод на заявка. Например може да има два метода Edit: първият ще бъде наличен за GET заявки и ще генерира формуляр за редактиране на елемент, докато вторият е достъпен само за POST заявки и се използва за актуализиране на елемента, когато формуляра е изпратен обратно към сървъра. Може да има и следната визия:

[HttpGet]

public IActionResult Edit()

{

return View();

}

[HttpPost]

public IActionResult Edit(Product product)

{

product.Save();

return Content( "Product updated");

}

[HttpGet]

public IActionResult Edit()

{

return Content(Edit);

}

Сега, когато правя заявка към метода/действието Edit(), действителният метод, отговарящ на заявката, се основава на това дали е GET или POST заявка. В някои ситуации се налага да се посочат множество глаголи за действие ( методи), например за да посоча, че едно действие може да бъде достъпно както от POST, така и от GET заявки, но не и от други типове. Това става лесно, по следния начин:

[HttpGet]

[HttpPost]

public IActionResult Edit()

{

...

Думичките в прави скоби – „[“ и “]”, се наричат анотации. Обикновено зад тях стои множество написан код. Те са атрибути, които могат да бъдат прилагани към класове или техните обекти за да уточнят връзката между класовете, да опишат как данните се показват в интерфейсната част и да приложат правила за валидация на данните.

* **Алгоритми за филтриране и сортиране**

public async Task<IActionResult> GenreFilter()

{

var products = \_context.Products

.Include(product => product.Category)

.Where(product => product.CategoryId == 1);

return View(await products.ToListAsync());

}

Филтрите работят и се прилагат предимно към основния (Index) метода/методите на проекта „Сладко изкушение“. Филтър означава да подредим и покажем дадена таблица във изгледа на приложението по определено условие. Например - без някое поле или според данните идващи от някое поле, ако са равни на очаквания от нас резултат, то… да се извърши нещо. Филтъра зарежда списък с всички записи от таблицата, като се прилага към него някакво ограничение /а именно свойството, по което ще бъде извършена филтрацията/. В горния пример съответно зареждам всички продукти от таблицата Products, като към нея свързвам и таблицата с категории. Връзката се осъществява посредством вторичен ключ. Чрез употребата на филтъра от горния пример, аз извеждам към изгледа информация само за продуктите, чието CategoryId = 1. А продукти от категория едно са тортите. Следователно горния пример ще изведе списък с продуктите, които са торти.

**ЗАКЛЮЧЕНИЕ**

**Общение на реализирания проект, предимства и недостатъци, теденции за бъдещи разработка и развитие**

Макар и сложно и доста трудно, проекта беше реализиран, преминах през всички изисквания и спецификации. Целите:

* Да се изгради средство за управление, реклама и популяризиране на сладкарски цех „Сладко изкушение“
* Да бъдат интегрирани роли за достъп
* Да бъдат включени продуктови списъци, които да бъдат интегрирани в база от данни
* Да бъде постигнато високо ниво на сигурност
* Да бъдат спазени всички европейски конвенции и стандарти за обработка на клиентски лични данни
* Приложението да предоставя директна обратна връзка
* Да бъде постигнато моментално и прецизно регистриране и оттчитане на поръчките
* Да се изгради структурата и логиката на базата данни
* Да се изгради MVC структура в ASP.NET проект

Всички цели бяха благополучно постигнати. Това означава, че до клиента ще достигне едно готово и работещо ASP.NET Core MVC приложение, което да подпомогне бизнеса и предприятията в малкия град, както по предназначение. В бъдещ план освен поддръжка на приложението, каквато е нужна и необходима на всеки софтуерен продукт. Ще бъде интегрирана нова функционалност според нуждата на клиента, както и ще бъде добавена доста по-широка дизайн част. Това ще направи интерфейса по-приятен за потребителите и е напълно възможно да привлече нови клиенти.

**ИЗПОЛЗВАНА ЛИТЕРАТУРА**

Примери: /*трябва да се следва точно формата/*

1. C# Users’ Guide, Microsoft Docs, [MSDN C# Guide](https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/)
2. Наков С., Тенденции в софтуерната индустрия в БГ, [www.nakov.com](https://nakov.com), April 2017
3. Свободна енциклопедия – Уикипедия [www.wikipedia.com/bg](https://bg.wikipedia.org/wiki/ASP.NET)
4. [Свободно разпространяема документация](https://docs.microsoft.com/en-us/aspnet/core/mvc/views/tag-helpers/intro?view=aspnetcore-6.0#tag-helpers-compared-to-html-helpers), Microsoft April 2022
5. [CRUD WIki](https://en.wikipedia.org/wiki/Create,_read,_update_and_delete)**, Wikipedia, April 2022**