ПРИРОДОМАТЕМАТИЧЕСКА ГИМНАЗИЯ „ХРИСТО СМИРНЕНСКИ“
град Перник

ДИПЛОМЕН ПРОЕКТ

ТЕМА: РАЗРАБОТКА НА УЕБ САЙТ ЗА РИБАРСКИ МАГАЗИН “NASLUKA”

професия код 481030 „Приложен програмист“
специалност код 4810301 „Приложно програмиране“

Изготвил:
Ивайло Петров Ивайлов
Ученик от XII б. клас

Ръководител-консултант:
Радослав Василев

Перник, 2022 г.

Увод

Обект на дипломният проект- уеб приложение направено на MVC модел.

-Предмет на проекта- уеб приложение „Наслука“, което трябва да отговаря на нуждите на хората, да е конкурентноспособно на останалите такива приложения и като цяло да е в крак с обществото ни.

Цел на задачата

Целта на задачата е да се направи: Уеб приложение “Наслука”.

Рибарски магазин, в който да има различни рибарски принадлежности (Въдици, Макари, Корди...), с опция за преглед, достъп до информация за тях (Описание на продуктите, цената им, наличност...), както също и опция за резервиране..

Актуалност на търсенето

В свят, в който интернетът започва все повече да участва в нашето ежедневие, създаването на такива приложения е една неизменна част. Те са в основата на онлайн пазаруването, а както знаем то все повече набира популярност сред младото поколение. Затова това приложение ще е напълно актуално, спрямо търсенето на пазара, както и ще отговори подобаващо на нуждите на потребителите.

Особености на приложението

Приложението трябва да е изградено на три нива, като на най-високото трябва да е администраторът на страницата (човекът имащ право да прави промени по количествата, да наблюдава потребителите в сайта, тяхната дейност и тн.). А на най-обикновеното стъпало да е обикновеният потребител, който има право само да разглежда и да се достъпва до информацията в приложението. За да се постигнат тези резултати и желаното оформление, ще бъде използван софтуер предоставен от “Microsoft”. Ще бъдат изградени база от данни и различни на тип елементи в средата за програмиране, чрез които ще имаме контрол над методите и случващото

се в приложението. Елементите трябва да се съобразят с контекста на задачата и нейната функционалност и цел, което се прави на програмно ниво и невидимо в последствие за обикновения потребител в нашия сайт. Проектирането (планирането) е част от всеки проект . В процеса на проектиране се откриват конкретни задачи свързани със самия проект. Важна част от създаването на дипломния проект е определяне на изискванията и техния анализ. За да създам сайта ще премина през различните етапи на проектиране. Реализацията на сайта ще бъде направено в следващата глава, а тестването ще се извършва по време на разработка. За моя сайт ще използвам NET технологията, в частност .NET Framework и NET Core. Microsoft .NET Framework е платформа, създадена от Microsoft, която предоставя програмен модел, библиотека от класове и среда за изпълнение на написан специално за нея програмен код, докато NET Core е безплатна и с отворен код, управлявана компютърна софтуерна рамка за операционни системи Windows , Linux и macOS. Той е кросплатформен наследник на .NET Framework.

Софтуер за изграждането

MVC шаблона ще оформи визуалното и логическото съдържание на приложението, така че то да се приведе във вид и функционалността му да има правилната логика на работа, за да се изпълняват ежедневни нужди на потребителите му. Използваме този софтуер, заради лесният му контрол, надеждната защита и най-вече наборът от функции и връзки между отделните компоненти, позволяващи тяхната работа и лесна промяна при нужда. Очакваме в най-кратък срок резултати, отговарящи на нуждите на пазара, достатъчно гъвкави, за да откликват на всякакви модификации без намесата на специализиран персонал. Тоест да се направи такава функционалността, че да не налага при смяна на елементи по сайта човек да бъде експерт в областта. Също Microsoft .NET Framework и NET Core,Visual Studio Community 2022, SQL Management Studio.

Разработката на софтуерния продукт минава през три етапа:

• Етап на проучване

• Етап на проектиране

• Етап на реализация

Глава 1

Проучване

1. Предпоставките за създаване на продукта

Има много причини за създаване на софтуерен продукт. Неговата реализация е свързана с необходимостта от *проучването на пазара*, дали има реален потенциал да бъде успешно направен и пуснат в действие. Реализацията на софтуер трябва да удовлетворява както изискванията на собственика на продукта така и изискванията на крайните потребители, като задоволява техните потребности. Бързо променящите се технологии налагат още повече да се направи добро *проучване на пазара* с цел да се избере възможно най-добрият подход за реализация, дългосрочно функциониране, бъдещо подобрение и поддръжка на продукта.

2. Избор на технологии и езици за програмиране

В основата му е MVC архитектурата, който остава един от най-добрите инструменти за този тип задачи. В частност ще използваме ASP.NET Core, MVC, SQ Management Studio - за работа с базата от данни, HTML, CSS, JavaScript, Bootstrap -за презентационния слой, както и трислойния модел. Използвам тези технологии, заради гъвкавостта им, надеждността и разнообразието от функции, които предлагат.

Трислойният модел

Най-разпространената форма на многослойна архитектура е трислойната архитектура. Трислойната архитектура обикновено се състои от:

-Презентационен слой,

-Слой за услуги,

-Слой за данни.

Това е модел, при който:

Фигура 1:Трислоен модел

Приложението е разпределено на слоеве. Всеки слой има строго определена задача. Във Visual Studio можем да създадем такова приложение, създавайки различни проекти в рамките на Solution-а ни.

MVC шаблонът

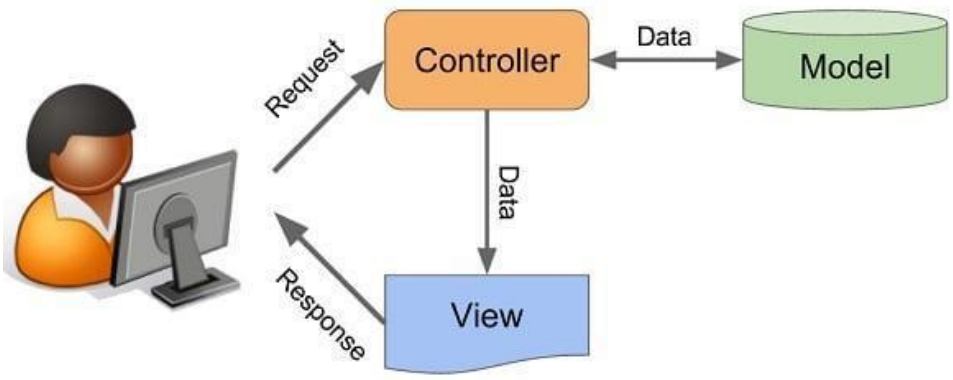
(Model-View-Controller или MVC) е архитектурен шаблон за дизайн в програмирането, основан на разделянето на бизнес логиката от графичния интерфейс и данните в дадено приложение. За пръв път този шаблон за дизайн е използван в програмния език Smalltalk.

Модел – ядрото на приложението, предопределено от областта, за която се разработва; обикновено това са данните от реалния свят, които се моделират и над които се работи – въвеждане, промяна, показване и т.н. Трябва да се прави разлика между реалния обкръжаващ свят и въображаемият абстрактен моделен свят, който е продукт на разума, който се възприема като твърдения, формули, математическа символика, схеми и други помощни средства. Например в банково приложение това са класовете, описващи клиентите, техните сметки, транзакциите, които са осъществили и т.н., както и класовете за извършване на операции над тези обекти (engines) – например клас Transfer с методи като createInterBankTransfer(), createInnerBankTransfer(), getCash() и т.н.

Изглед (англ. View) – тази част от изходния код на приложението, отговорна за показването на данните от модела. Например изгледът може да се състои от PHP шаблонни класове, JSP страници, ASP страници, JFrame наследници в Swing приложение. Зависи от това какъв графичен интерфейс се прави и каква платформа се използва;

Контролер – тази част от сорс кода (клас или библиотека), която взима данните от модела или извиква допълнителни методи върху модела, предварително обработва данните, и чак след това ги дава на изгледа. Например може да бъде създаден един малък обект, в който да бъдат сложени данните за транзакцията – като в контролера бъдат взети данните за транзакцията от модела, бъдат преведени датите от UNIX формат в четим от потребителя формат, бъде преобразувана валутата от долари в евро например, бъде закръглено до втория знак вместо да се виждат данните както са в модела (и в базата) до 10-ия. Също така когато се прави уеб графичен интерфейс това

би довело до много лесна модификация на HTML кода дори от човек, който не е програмист – той ще гледа на шаблона просто като на обикновена HTML страница.

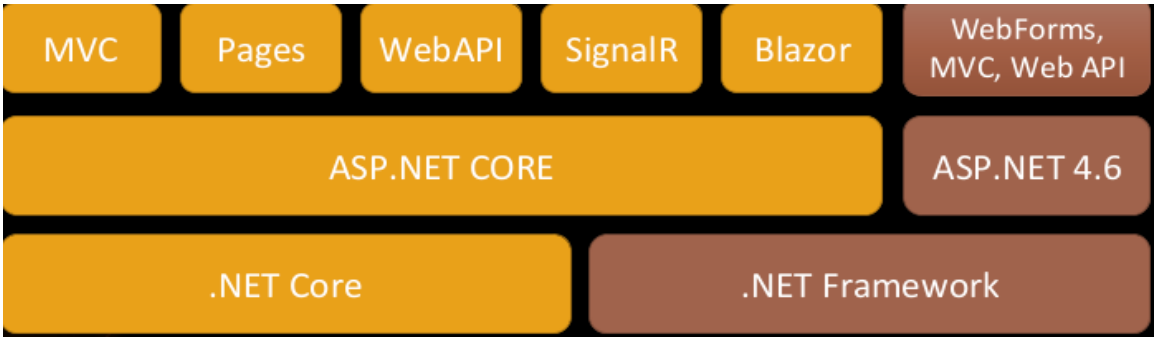


Фигура 2: MVC Шаблон

Входящата заявка се пренасочва към контролер. За уеб: HTTP Заявка Контролерът обработва заявка и създава презентационен Модел Controller също избира подходящ резултат (например: View). Моделът се предава на изгледа Изгледът преобразува Модела в подходящ изходен формат (HTML). Отговорът е предоставен (HTTP отговор) MVC.

ASP.NET Core

Е уеб платформа с отворен код. Можете да изграждате уеб приложения и сървиси, IoT приложения, мобилни приложения и всяко уеб-базирано решение с ASP.NET Core. Главните плюсове са, че Унифицирана рамка за изграждане на уеб потребителски интерфейс и уеб API, лесно за тестване Възможност за разработване и изпълнение под Windows, macOS и Linux Възможност за хостване на IIS, Nginx, Apache, Docker или self-host на собствена машина в собствен процес Вградена dependency injection.



Фигура 3: ASP.NET Core

C#

Е обектно ориентиран език за програмиране, разработен от Microsoft като част от софтуерната платформа .NET. Стремежът още при създаването на C# езика е бил да се създаде прост, модерен, обектно ориентиран език с общо предназначение.

C++

Е език за програмиране от високо ниво, който има възможности за програмиране на Операционни системи, големи програмни системи, с високо ниво на функционалност, представяне и ефикасност.

C# vs C++

-C++ е известен като език на средно ниво, който добавя обектно-ориентирани функции към своя базов C, докато C# е език от високо ниво.

-C++ компилира програми в машинни кодове, а C# компилира програми в Common Language Runtime или CLR.

-C++ не предупреждава потребителите, ако има някакви грешки преди компилацията, когато се следва синтаксисът. C# предупреждава потребителите за грешките на компилатора, което прави работата по-малко досадна.

-В C++ програмистите могат да използват указатели навсякъде по всяко време. В C# програмистите могат да използват указатели само в небезопасен режим.

Управлението на паметта в C++ се извършва ръчно от програмиста. C# работи на виртуална машина и по този начин управлението на паметта става автоматично.

.NET

NET е софтуерна крос-платформена технология (software framework) с отворен код, която се поддържа от Microsoft и .NET общността в GitHub. Всички аспекти на .NET са с отворен код, включително библиотеки с класове, старт и изпълнение, компилатори, езици, уеб рамка на ASP.NET Core, настолни рамки на Windows, библиотека за достъп до данни Entity Framework Core и др.

.NET Foundation

.NET Foundation е независима организация с нестопанска цел, създадена да поддържа иновативна, удобна за търговия екосистема с отворен код около платформата .NET.

.NET Framework

.NET Framework е първоначалната имплементация на .NET и поддържа разработка на уеб сайтове, системни услуги, настолни приложения и др.. .NET технологията макар замислена като крос-платформа, нейната реализация .NET Framework се използва предимно за Windows, защото съдържа специфични библиотеки за тази операционна система. За използването като крос-платформа е създаден проектът Mono. Спонсориран от Microsoft, Mono е крос-платформена реализация с отворен код на .NET Framework, базиран на стандартите ECMA за C# и Common Language Runtime. Чрез проектът Mono могат да се стартират приложения на Microsoft .NET междуплатформено на Android, Linux, Solaris, Mac OS, PlayStation и др.

.NET Core

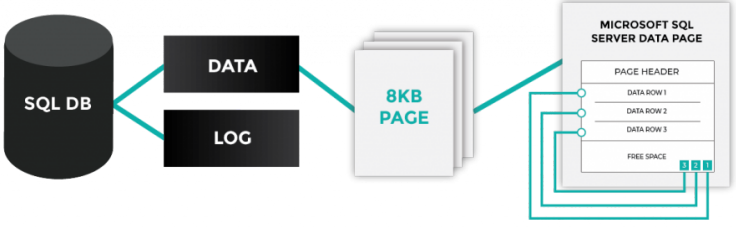
.NET Core е нова версия на .NET Framework, която е безплатна платформа за разработка с отворен код с общо предназначение, поддържана от Microsoft. Това е междуплатформена рамка, която работи на операционни системи Windows, macOS и Linux.

ASP.NET vs ASP.NET Core:

Една от основните причини да избера ASP.Net Core пред ASP.NET, е че ASP.NET Core е сравнително нова рамка с отворен код, също че е междуплатформена рамка, Подходящ за изграждане на модерни, облачно базирани уеб приложения на Windows, macOS или Linux.

Microsoft SQL Server

Е сървърна система за управление на база от данни. Избрах нея, заради надеждността й, както и заради широкият спектър от функции. Предназначена е за управление на големи сървърно базирани БД, за разлика от MS Access, която е desktop базирана и не е предназначена за управление на големи корпоративни БД.



Фигура 4:SQL Server

Microsoft SQ Management Studio

Това е софтуерно приложение, стартирано за първи път с Microsoft SQL Server 2005, което се използва за конфигуриране, управление и администриране на всички 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 потребителя да търси, избира и да работи с всеки от обектите на сървъра. Приложението има и „експресна“ версия, която може да бъде изтеглена безплатно.

HTML

(съкращение от термина на английски: HyperText Markup Language, 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 е стандарт в интернет, а правилата се определят от международния консорциум W3C. Текущата версия на стандарта е HTML 5.0 (от 28 октомври 2014 г.).

CSS

Е създаден 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.

JavaScript

Е интерпретируем език за програмиране, разпространяван с повечето уеб браузъри. Поддържа обектно ориентиран и функционален стил на програмиране. Създаден е в Netscape през 1995 г. Най-често се прилага към HTML-а на интернет страница с цел добавяне на функционалност и зареждане на данни. Може да се ползва също за писане на сървърни скриптове JSON, както и за много други приложения. JavaScript не трябва да се бърка с Java, съвпадението на имената е резултат от маркетингово решение на Netscape. Javascript е стандартизиран под името EcmaScript.

Глава 2

ПРОЕКТИРАНЕ

1. Представяне

Проектирането (планирането) е част от всеки проект . В процеса на проектиране се откриват конкретни задачи свързани със самия проект. Важна част от създаването на дипломния проект е определяне на изискванията и техния анализ. За да се създаде сайта ще се премине през различни етапи на проектиране. Реализацията на сайта ще бъде направено в следващата глава, а тестването ще се извършва по време на разработка.

2. Изисквания

2.1. Общо описание на изискванията

Изискванията определят услугите,които клиентът изисква от системата (уеб приложението) и на ограниченията, при които тя работи и се разработва. Изискванията са описания на системните услуги и на ограниченията на системата. Те могат да варират от много абстрактни описания на дадена системна функционалност до много точни математически функционални спецификации. Проблеми при изискванията могат да възникнат:

• Поради неточно описание на изискванията.

• Двусмисленост - изисквания могат да бъдат интерпретирани по различен начин от разработчиците и потребителите. Например, изискването „подходящата среда“ за потребителя може да означава, среда подходяща за отваряне на документ. За разработчиците „подходящата среда“ може да означава среда в която да разработят успешно своя проект.

Типове изисквания:

• Системни изисквания – представляват структурирана документация с описание на функциите, услугите и работните ограничения на системата. Определя какво трябва да се реализира и може да бъде част от договора между клиента и

разработчика

• Потребителски изисквания – Указания на естествен език и диаграми на услугите, които системата ще предоставя, както и съответните оперативни ограничения.

2.2. Функционални изисквания

Функционалните изисквания определят, какво трябва да прави системата. В нашият случай системата представлява уеб приложение. Функционалността на системата определя изхода спрямо нейния вход. Функционални изисквания представляват описание на услугите, които системата трябва да предостави, начинът по който трябва да реагира на конкретни входни данни и нейното поведение в конкретни ситуации.

Функционалните изисквания включват:

• Описание на функции или услуги на системата.

• Зависят от типа на софтуера, потенциалните потребители и типа на системата, в която ще бъде използван софтуера.

• Функционалните потребителски изисквания могат да бъдат описания на високо ниво на това какво трябва да прави системата, но функционални системните изисквания трябва да описват системната функционалност в детайли.

Бележка: Функционалните изисквания ще бъдат подробно разгледани в раздел „Анализ на функционалните изисквания“.

2.3. Нефункционални изисквания

Ограничения върху услугите или функционалността на системата като времеви ограничения, ограничения върху процеса на разработване, използваните стандарти и др. Нефункционалните изисквания могат да бъдат по-критични от функционалните и от тях може да зависи пригодността на системата.

Нефункционалните изисквания определят:

• Системни характеристики и ограничения като надеждност, време за отговор и др.

• Изисквания към хардуера, технологията за разработка, програмния език.

Бележка: Нефункционалните изисквания ще бъдат подробно разгледани в раздел „Анализ на нефункционалните изисквания“.

3. Анализ на функционалните изисквания

3.1. Преглед

Фаза анализ се занимава с това какво трябва да върши приложението. В раздел проектиране и глава реализация ще бъде разгледан въпросът как точно приложението ще постигне набелязаните цели от фазата на анализа. Целта на анализа е да предостави ясна спецификация за потребителя. За разлика от фазата на проектиране и реализация, фазата на анализа не акцентира на техническите въпроси в проекта. Резултатът от анализа представлява документиране на функционалните изисквания за уеб приложението.

3.2. Изисквания към правата на потребителите

Функционалността на сайта се определя в голяма степен и от това какви са правата на определена група потребители. В уеб приложението ще има три или четири групи потребители, както е показано на фигура 1.

Фигура 1. Потребителски права в уеб приложението

Администратор – той има най-големи права при използването на сайта. Той може да въвежда данни за продукти, данни за потребители, да актуализира и изтрива данни от базата с данни. Това е потребителят с най-много правомощия в приложението. Той има по-голям набор от функции спрямо останалите потребители (промяна по информацията за продуктите добавяне, премахване, преглед на потребителите и тн.)

Регистриран потребител – има достъп до определен набор от функции на приложението и може да използва услугите на сайта: разглеждане на продукти, пазаруване, информация за негови поръчки, информация за адрес за получаване, различни начини на разплащане и др. Неговите права са по-големи от тези на гостите.

Гост – има ограничени права и може да разглежда сайта, като в някой реализации той може дови да ползва услугата за онлайн пазаруване. Неговите права са по-малки от тези на регистрираните потребители.

3.3. Изисквания към функционалния дизайн

Уеб приложението ще разполага със следните публично достъпни секции: категории, опции за поръчка, за разглеждане на артикули, опция за преглед на наличността, снимки към всеки продукт.

Начална страница - с влизането в сайта потребителят ще види началната страница, която представя следната информация:

- преход към други страници, чрез избор на бутони от менюто

- друга актуална информация и материали отнасяща се към сайта - например, файл с каталог на продукти, цените им, описание към тях и всичко, от което потребителят може да се нуждае при избор на продукт.

Функционалността на началната страница се изразява в това какъв избор може да направи потребителя на сайта, например, кликва на реклама, сваля файл, избира бутон от меню и др. Потребителят ще има възможност да избере бутони и други

функционални елементи за управление на екраните и достъп до съдържанието в сайта.

Страница регистрация – ще предостави възможност за регистрация на потребителски акаунти. Потребителят ще трябва да въведе име, имейл, парола, адрес и никнейм, за да се регистрира и да получи права като регистриран потребител.

Страница продукти - ще разполага с информация в реално време на всичко налично и всичко изчерпано което не е налично от продуктите. Обновяването е в реално време, за да има постоянно актуална информация за продуктите предлагани в сайта. Функционалността на страницата с продукти се изразява в това, че потребителят ще има възможност да кликне върху избран продукт и да види подробности за него. Когато потребителят се намира в тази страница менюто за преход към други страници остава достъпно.

3.4. Потребителски истории (user stories)

Потребителските истории се използват при разработването на софтуер, като начин да се помогне на разработчиците да разберат желанията и нуждите на своите потребители. Потребителските истории са кратки и следват прост шаблон за изразяване. Потребителските истории могат да се вграждат в епоси, които са по-широки твърдения описващи „голямата картина“ на потребителския опит.

Предимства:

• кратки

• разбираеми за потребителите и разработчиците

• Не е нужна поддръжка

• Не налагат особени усилия

Недостатъци:

• Може да са непълни

• Различна интерпретация

Потребителските истории имат за цел да представят гледната точка на потребителя, а не гледната точка на създателя (разработчика на програмата). Потребителите не са експерти относно реализацията на програмата и нямат гледната точка на създателя. Потребителските истории могат да са от голяма полза при

определяне на случаи на употреба.

• „Като администратор искам да напиша ясна информация, за продуктите, за да спечеля доверието на клиентите.“

• „Като потребител искам да видя добра информация, за продуктите, за да зная какво купувам.“

„Като потребител искам да мога да изтегля каталог, за продуктите, за да разгледам продуктите в по удобен вид.“

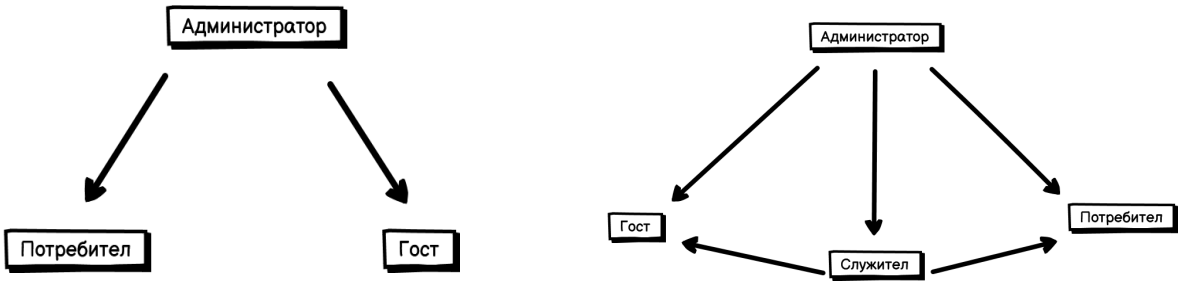
• „Като администратор искам да създам отчет, за да виждам информация за клиенти изтеглили каталога за продуктите.“

• „Като администратор искам да създам реклама за продукти, която се показва на клиенти изтеглили каталога, за да спечеля тяхното внимание.“

• „Като администратор мога да осъществя достъп до контролния панел, за да управлявам сайта.“

Няма доказателства, че използването на потребителски истории повишава успеха на софтуера или производителността на разработчиците. Въпреки това, потребителските истории улесняват проектирането на софтуерния продукт.

3.5. Случаи на употреба (use cases)



Случаите на употреба представя множество от възможности при използване на уеб приложението. Тези възможности са свързани с начина по който сайта се използва от различни потребители. Анализирането на случаите на употреба дефинират отговори на различни въпроси, като „Какво трябва да направим в даден случай?“, „Какви трябва да бъдат правата на определен тип потребителите?“ и др. Това е рамка за документиране, която съответства на случаи на избор и решения, подобно на

оператори if...then... else, които помагат на програмистите да обмислят проблемите и да предоставят решението в стандартното програмиране. Определянето на случаите на употреба, за уеб приложението, ще се направи според информацията представена в „Изисквания към функционалния дизайн“ и „Потребителски истории“

Стъпки за оформяне на случаи на употреба

• Определя се кои са типовете потребители.

• Определя се целта, която всеки потребител трябва да постигне.

• Попълват се всички критерии посочени в картата.

• Прави се оценка на случаите на употреба, преди да се премине към реализация в уеб приложението.

Общ вид на карта за определяне на случай на употреба

Основно действащо лице	Регистриран потребител
Предварителни условия	Потребителят е регистриран и след избор отива на менюто за поръчки, за да поръча продукт.
Основен успешен сценарии	Поръчката е успешна и потребителят вижда това.
Алтернативен сценарии	Недостатъчно количество или грешка в системата. Поръчката е неуспешна.
Специални изисквания	Изисквания които не влизат в Основен успешен сценарии и Алтернативен сценарии.
Неразрешими проблеми	Въпроси които трябва да получат отговор, преди случаят на употреба да бъде завършен.

Създаване на карти за случаи на употреба за уеб приложението

Карта 1

Основно действащо лице	Потребител
Предварителни условия	Клиентът вижда формата за поръчване на продукт и бутон за изпращане на данните.

Основен успешен сценарии	Клиентът попълва формата, изпраща данните с натискане на бутона. Системата приема успешно попълнените данни във формата и извежда съобщение за приети данни.
Алтернативен сценарии	Системата открива грешни данни в попълнената форма и показва отново формата със съобщение за грешка.
Специални изисквания	Формата и бутоните да бъдат удобни с размери, които позволяват на хора със слабо зрение да попълнят формата
Неразрешими проблеми	Ще се използват ли форми, които позволяват на потребителя да избира от списък с опции (например град, област и т.н.) или той ще трябва на ръка да попълва всички полета.

Карта 2

Основно действащо лице	Администратор
Предварителни условия	Администът попълва данните за продукта, който иска да създаде.
Основен успешен сценарии	Администът успешно попълва данните и с необходимият бутон добавя продукта към страницата.
Алтернативен сценарии	Системата не разпознава данните и извежда грешка.
Специални изисквания	Бутоните да бъдат ясно различими, дори за хора с увреждания.
Неразрешими проблеми	Грешка в данните и администраторът ще трябва ръчно да попълва данните на продукта.

Карта 3

Основно действащо лице	Гост
Предварителни условия	Гостът влиза в приложението и отива на регистрационното меню.
Основен успешен сценарии	Успешно попълва данните си и се вписва в базата данни, което му дава права на потребител.
Алтернативен сценарии	Системата извежда грешка поради ясни или неясни причини.
Специални изисквания	Полетата са добре описани за повече разбираемост.

Неразрешими проблеми	Отказан достъп на гостът, поради грешка в системата.

Карта 4

Основно действащо лице	**Администратор**
Предварителни условия	Администраторът отива на страницата за правата на потребителите, с цел да даде повече права на даден потребител.
Основен успешен сценарии	Успешно се добавят правата на потребителят, като системата ги отчита за напред.
Алтернативен сценарии	Системата не приема промените, като не добавя права на потребителя.
Специални изисквания	Да е ясно за администът на кого дава права, с ясни и точни полета.
Неразрешими проблеми	Системата отказва да разпознае администратора и изобщо да му даде опция да дава права.

Карта 5

Основно действащо лице	Потребител
Предварителни условия	Потребителят след излизане от системата да се върне обратно в нея.
Основен успешен сценарии	Успешно се логва, с въвеждане на необходимите данни.
Алтернативен сценарии	Не са разпознати данните му и му е отказан достъпа до приложението.
Специални изисквания	Да е ясно, какви полета се изискват за попълване
Неразрешими проблеми	Системата е изтрила потребителя и той не съществува вече в базата от данни.

Карта 6

Основно действащо лице	Администратор
Предварителни условия	Администраторът иска да види повече информация за регистрираните потребители в базата данни.
Основен успешен сценарии	Успешно вижда информацията, която го интересува за потребителя.
Алтернативен сценарии	Достъпът му е отказан или изобщо не извежда таблица на базата данни.
Специални изисквания	Да е ясно, коя таблица с информация е отворена.
Неразрешими проблеми	Базата от данни да е изтрита като това е необратим процес.

Карта 7

Основно действащо лице	Потребител
Предварителни условия	Потребителят иска да види всички налични продукти като отиде на съответната страница.
Основен успешен сценарии	Успешно излиза списък на продуктите, с вписаната за тях информация и наличност.
Алтернативен сценарии	Списъкът е празен или изобщо не се показва, като извежда грешка на екрана.
Специални изисквания	Бутонът за списъка да е на достъпно място и лесен за използване от потребителя.
неразрешими проблеми	Данните за продуктите са изтрити от базата данни, което автоматично ги прави несъществуващи.

3.6 UML диаграми за случаи на употреба

Функционалният модел позволява изготвянето на UML диаграми за различни случаи на употреба. Тези диаграми не винаги са достатъчно детайлни, за да бъдат използвани отделно от текстовите описания. Въпреки това те са добро допълнение, което осигурява визуализация на случаите на употреба и действащите лица.

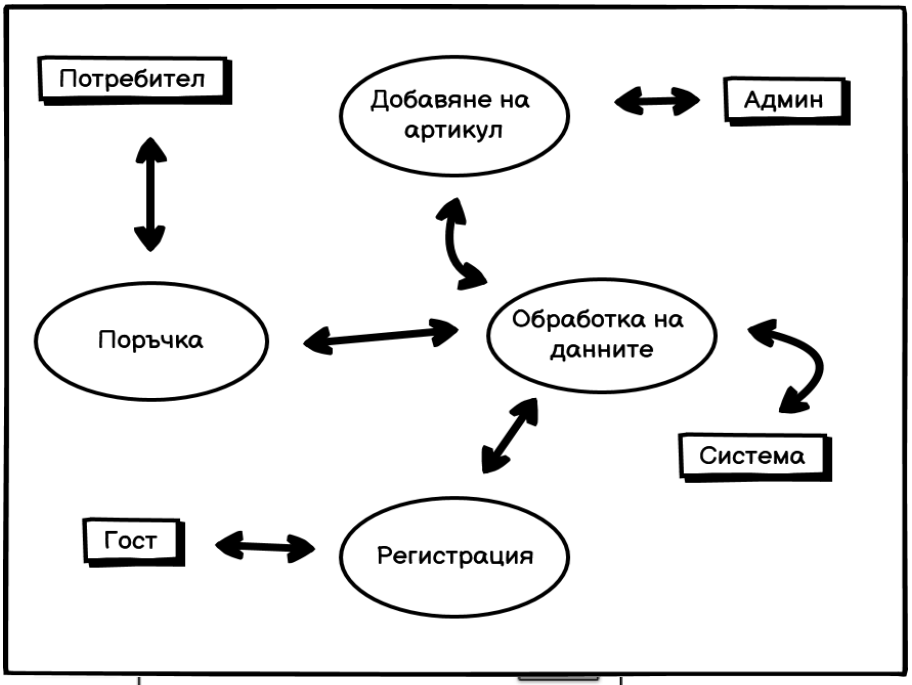
UML диаграми по създадените карти:

• На фигура 2 е показана диаграма на функционален модел за случаи на употреба по

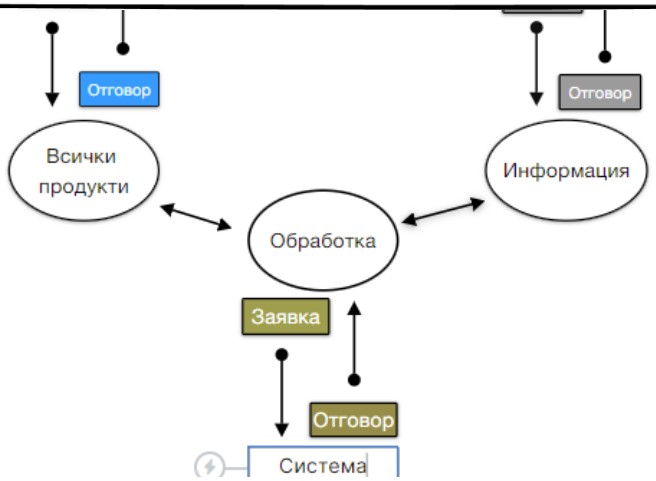
Карта 1,2,3

• На фигура 3 е показана диаграма на функционален модел за случаи на употреба по Карта 4,5.

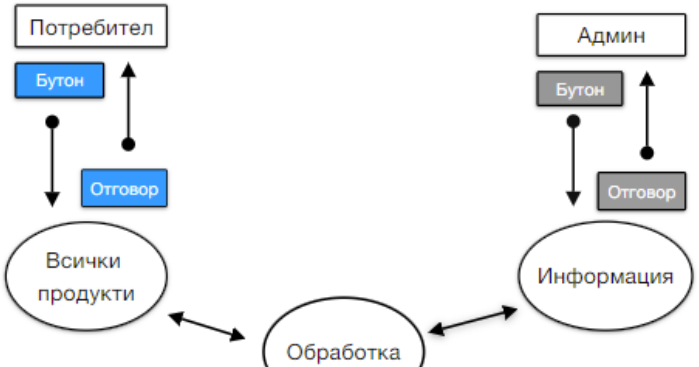
• На фигура 4 е показана диаграма на функционален модел за случаи на употреба по Карта 6,7



Фигура 2: UML диаграма



Фигура 3: Диаграма на логването и правата



Фигура 4: Диаграма на всичките продукти и информацията

4. Анализ на нефункционалните изисквания

4.1. Изисквания към разработката на уеб приложението

Някой от нефункционалните изисквания посочени в тази точка се базират на анализа направен в глава първа, поради което тук няма да се разглеждат в детайли.

Изисквания към дизайна

Сайтът ще бъде със зелена лента при бутоните, текстът им ще бъде черен-обикновен шрифт. Бутоните за регистрация,edit, delete,create,details ще бъдат сини, със стил “Times New Roman”, бутоните за поръчка и изтриване ще бъдат червени.

Изисквания към софтуера за разработка

• Visual Studio Community 2019

• Библиотеки необходими за проекта: ASP.NET Core Identity;

• MS SQL Server, .Net Framework, ASP.NET Core, Proxy Lazy Loading, Razor.

Изисквания към технологиите и езиците за разработка

• Базов проект за .NET включващ MVC (Controller-View-Model)

• C#, SQL, HTML, CSS, Java script и др.

Изисквания към хардуера за разработка

- 1,8 GHz или по-бърз процесор. Препоръчително е четириядрен или по-добър.
- 2 GB RAM; Препоръчва се 8 GB RAM (минимум 2,5 GB, ако работи на виртуална машина).
- Пространство на твърдия диск: Минимум 800MB до 210 GB налично пространство, в зависимост от инсталираните функции; типичните инсталации изискват 20-50 GB свободно пространство.
- Скорост на твърдия диск: за да подобрите производителността, инсталирайте Windows и Visual Studio на твърд диск (SSD).
- Видеокарта, която поддържа минимална резолюция на дисплея от 720p (1280 на 720); Visual Studio ще работи най-добре при резолюция WXGA (1366 на 768) или по-висока.

4.2. Изисквания към хостинга на уеб приложението

Уеб хостът, който ще се използва за работа на приложението в интернет, трябва да поддържа използваните от уеб приложението технологии: .NET 6.0, MS SQL Server, MVC, LINQ, Mail Server и др. Доставчикът на хостинг услуги трябва да:

• използва бързи дискове и кеширане на данни, за минимално време на отговор към клиента;

• поддържа подходящ трансфер на данни за уеб приложението;

• да осигури работа на уеб приложението, 24 часа и 7 дни вседмицата;

• осигури достатъчно дисковото пространство за уеб приложението и базата данни, както и за тяхното разрастване в бъдеще;

• позволява безпроблемно преминаване към друг хостинг план;

• поддържа добър и удобен панел за управление на услугите.

Бележка: С цел да не се прави реклама тук, няма да се посочват подходящи доставчици на хостинг услуги. Търсенето в Интернет на такива доставчици може да стане по ключови думи: „хостинг .net“.

5. Фаза проектиране

5.1. Представяне

Във фаза проектиране ще се използват функционалните изисквания от фаза анализ, за да се определят различните подсистеми и класове. Във фаза проектиране ще се извлекат отделни елементи, които ще бъдат описани с класове, ще се определят задачите за тези класове както и отношенията помежду им. Целта на тази фаза е да се изготви скица (техническа документация) на уеб-приложението, което трябва да се създаде. Скицата включва всички подсистеми в това число и базата данни, класове и техните отношения, като предоставя ясно определяне на зависимостите и взаимодействията между отделните системи и класове. Създадената скица ще предостави помощ и разбиране на процесите в етапа на разработка. Във фазата на проектиране няма да се определят техниките и инструментите, които ще се използват във фазата на реализация. Тъй като изграждането на базата данни е по модела отгоре-надолу първо ще разгледаме необходимите класове, които трябва да създадем, а след това ще представим таблиците, които ще бъдат създадени в базата данни и връзката между тях.

5.2. Класове и взаимовръзка

В проекта има няколко класа:

-Клас за продуктите.

Класът за продуктите, отговаря за самите продукти. Това какви полета ще имат при създаването на нов продукт, какво ще се вижда от потребителя и всичко свързано с продуктите и тяхната информация.

-Клас за поръчките.

Класът за поръчките е клас, от който зависят поръчките на продуктите. В него се вписват всички условия по тях и цялата им функционалност.

-Клас за категориите.

Класът за категориите отговаря за категориите на продуктите. Понеже в сайта артикулите ще имат категории е необходим този клас. В него се съдържа цялата информация за категориите, това колко са, какви са и тн.

За реализацията на класове и връзка между тях ще се използват резултатите от анализа направен за функционалните изисквания. Нефункционалните изисквания определят използването за специфичен избор на технологии с които уеб приложението да бъде реализирано и тази част ще бъде разгледана в следващата глава. В раздела на функционалните изисквания разгледахме функционалността на уеб приложението за отделните страници, направихме множество случаи на употреба и потребителски истории. От направения анализ се вижда, че се нуждаем от класове в следните страници:

Начална страница – тази страница е съвкупност от няколко класа, вюта и тн. На нея ще могат да се видят продуктите, за което се грижи класът на продуктите

Страница регистрация - регистрацията зависи от скафолнат айтем, който системата създава автоматично

Страница продукти - за тази страница отговаря отново класът за продуктите, но този път участва и класът за категориите, вюта, модели и др.

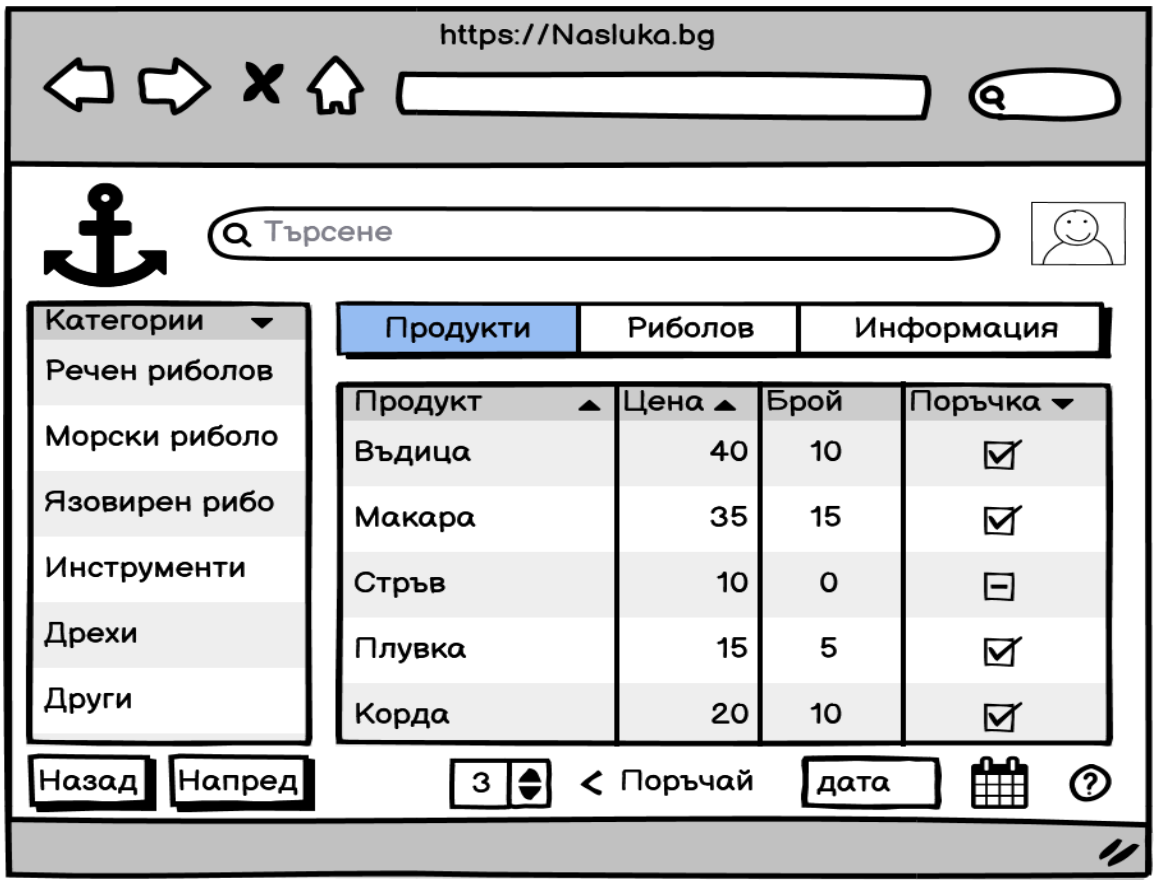
5.3. Реализация на база данни

Нефункционалните изисквания определят използването на MS SQL Server за управление на базата данни. За функционалните изисквания определянето на таблиците и връзката между тях (релациите) в базата данни са следствие от класове, които реализират случаите на употреба. Таблиците в базата данни, които автоматично се генерират в проекта от Visual Studio 2019 ще бъдат посочени в следващата глава, когато реално бъдат създадени.

5.4. Прототипи на потребителския интерфейс

Прототипите дават нагледа представа за визията на софтуерния продукт. Те позволяват на дизайнерите да покажат продукта си, което го прави по-лесен за разбиране. Прототипи могат да се създават по време на всеки етап от процеса по дизайн, за да помогнат да се демонстрират идеи, които трудно биха се изразили.

5.5 Очакван интерфейс



Фигура 5: Очакван интерфейс

Това за момента е очакваният интерфейс. На него се виждат полетата, които ще има, категориите, опцията за поръчка, броят на артикулите, датата на поръчка, бутон за напред и назад, човече за профила, бутон за помощ, както и лента за търсене, информация за продуктите, тяхната цена, брой.

Глава 3

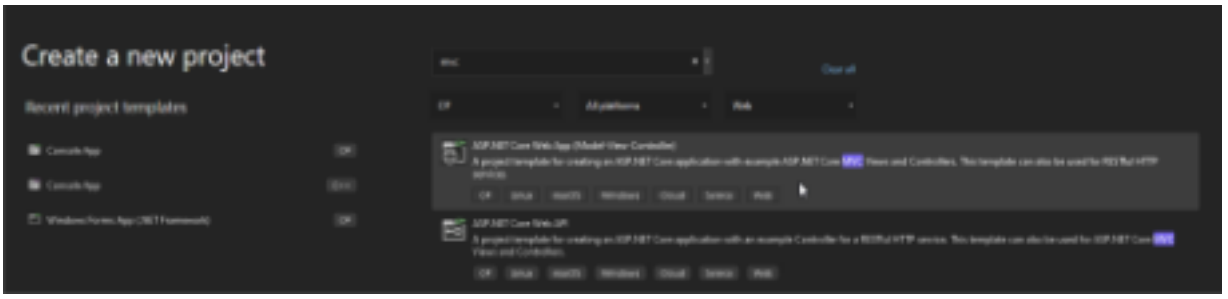
РЕАЛИЗАЦИЯ

1. Създаване на базов проект

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

Стъпки за създаване на базов проект

Създаване на нов проект



Тук избираме, че ще работим с MVC.

2.Какво е Authentication и Authorization.

Authentication - удостоверяването е процес на проверка на идентификационните данни на потребител или устройство, което се опитва да получи достъп до системата. Такива данни са например: потребителско име и парола.

Authorization - упълномощаването е процес на проверка дали на потребителя или устройството е разрешено да изпълнява определени задачи в дадена система. Упълномощаването определя, какво е достъпно в системата за конкретен потребител.

2.1Типове Authentications

Автентикацията на електронен документ има за цел той да бъде защитен от възможни злоумишлени действия, като например:

• активно прихващане – нарушителят се включва в компютърната мрежа и прихваща документа (файла);

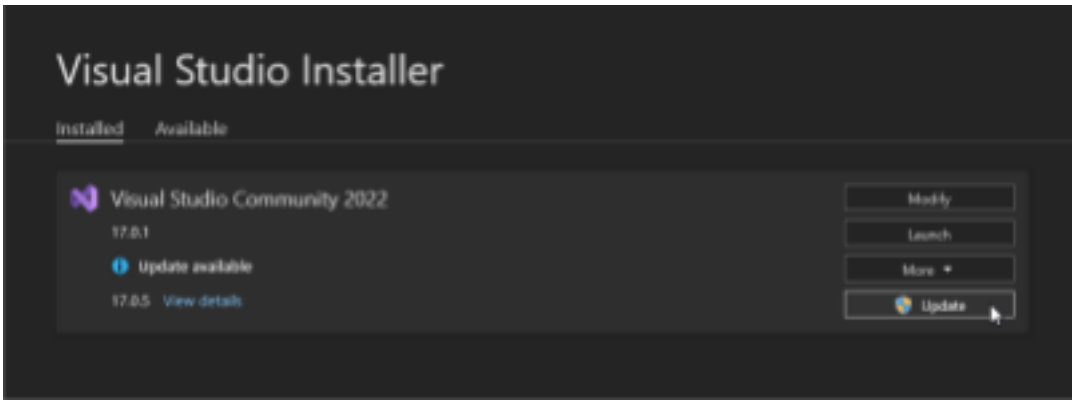
• маскарад – абонатът X изпраща документ на абоната Б от името на абоната А; • ренегатство – абонатът А заявява, че не е изпращал съобщения на абоната Б, макар всъщност да е изпратил;

• подмяна – абонатът Б изменя или формира нов документ и заявява, че го е получил от абоната А;

• повторение – абонатът X повтаря документ, който абонатът А е изпратил до абоната Б.

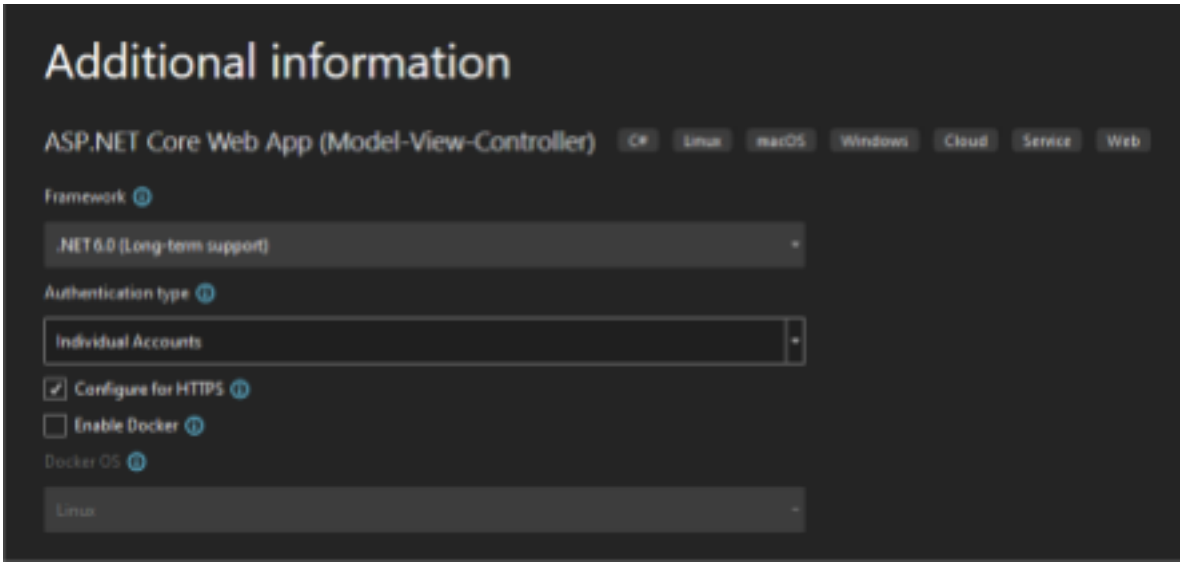
Docker platform

Docker е набор от PaaS продукти, използващи виртуализация на ниво операционна система (контейнеризация) и предоставящи софтуерни пакети, наречени контейнери. Контейнерите са изолирани един от друг и съдържат определен софтуер, библиотеки и конфигурационни файлове. Те могат да комуникират помежду си по строго дефинирани канали. Тъй като всички контейнери споделят сервизите на едно единствено OS ядро, те използват по-малко ресурси в сравнение с виртуалните машини.



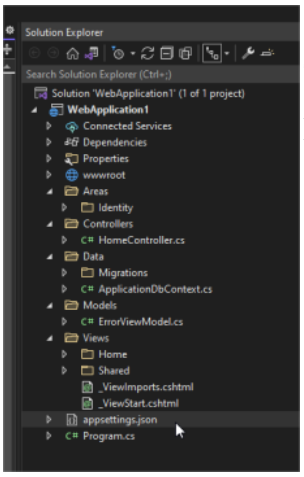
Тук ъпдейтваме Visual Studio Community 2022.

Изберете следните опции за проект и генерирайте проекта:



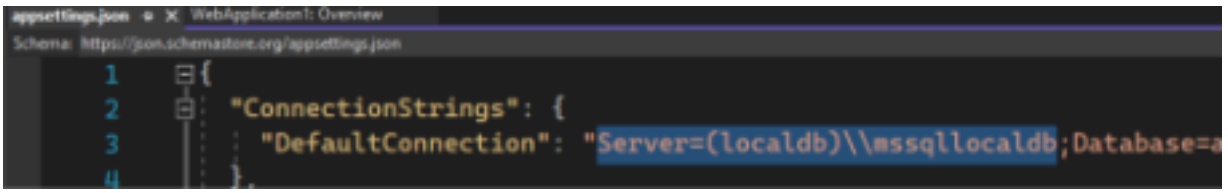
Тук избираме версия на Framework, Authentication, както и дали да имаме конфигурация за HTTPS и дали ще включим Docker.

Отворете .json файла:



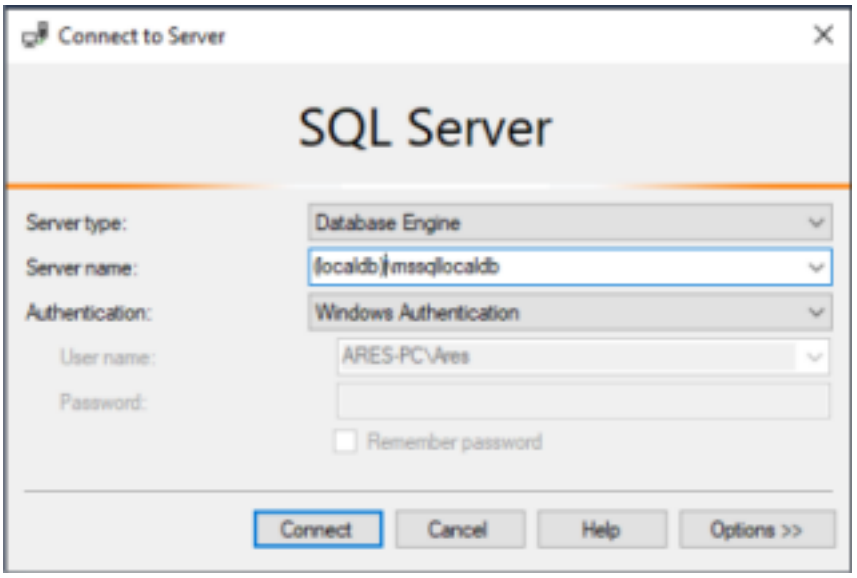
Тук можем да видим всички папки и какво има в тях. Виждаме Вютата, Контролерите, Моделите и други.

Вижте тази информация

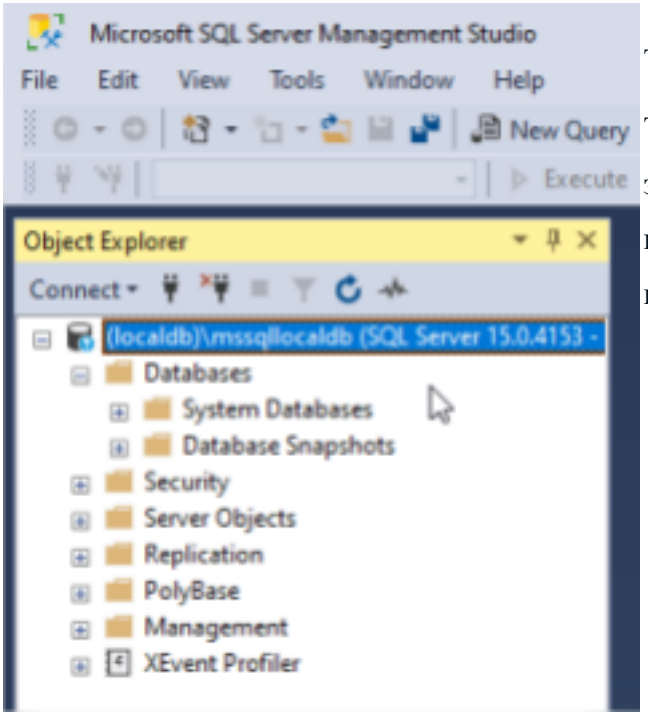


Това е сървърът, към който ще се свърже базата ни от данни.

Свържете се със сървъра като напишете информацията, която сте взели от приложението.

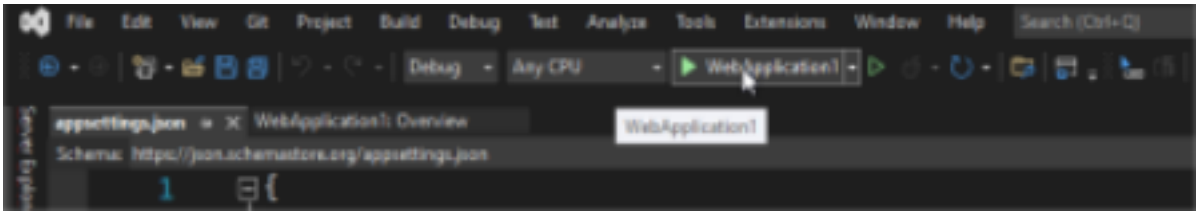


Ще видите такъв прозорец:



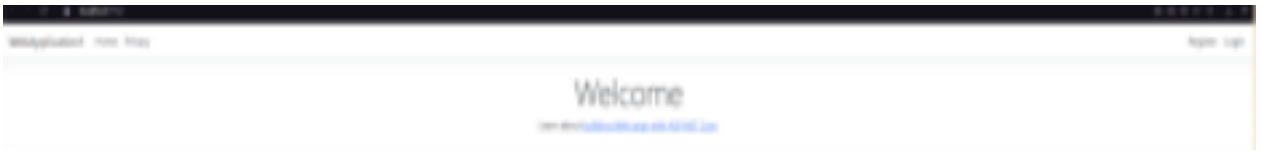
Това са папките на базата от данни.

Тук можем да видим информация за съдържанието им, наличието им и всякаква информация, която ни интересува свързана с тях.



Стартирайте уеб приложението

Ще видите следния екран в браузъра:

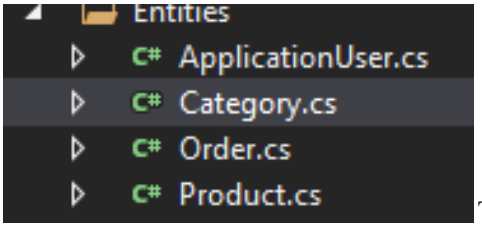


Това са стъпките по създаването на проект, неговата функционалност и съдържание зависят от нас. Като ние добавяме всичко, което ни е нужно.

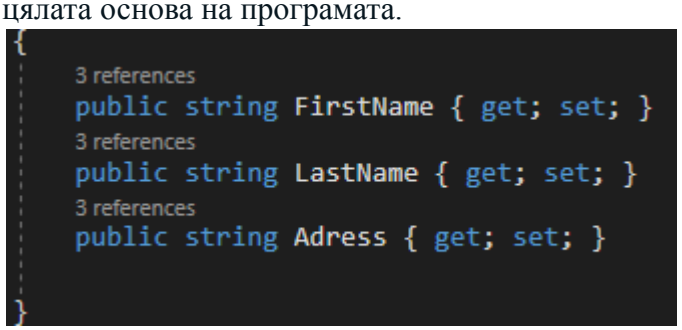
2. Изграждане на уеб приложението.

2.1 Базата от данни

Това са класовете, които образуват базата данни. Те са в основата на програмата и в тях са основните полета, по които се води цялата функционалност на програмата.



Това са основните класове в програмата, от тях зависи цялата основа на програмата.



Клас ApplicationUser, с неговите полета. Които са направени спрямо условията на дипломния проект.

Клас Category-отговаря за категориите на продуктите.

```
public class Order
{
        2 references
        public Order()
        {
            this.Id = Guid.NewGuid().ToString();
        }
    [Key]
    [Required]
    1 reference
    public string Id { get; set; }
    [Required]
    2 references
    public DateTime CreatedOn { get; set; }
    [Required]
    [Range(1,1
    4 references
    public int CountProducts { get; set; }
    [Required]
    1 reference
    public string UserId { get; set; }
    0 references
    public virtual ApplicationUser User { get; set; }
    [Required]
    3 references
    public int ProductId { get; set; }
    [Required]
    1 reference
    public virtual Product Product { get; set; }
    0 references
    public decimal TotalPrice
    {
        get
        {
            return CountProducts * Product.Price;
        }
    }
}
```

```
readonly struct System.DateTime
Represents an instant in time, typically expressed as a date and tim
```

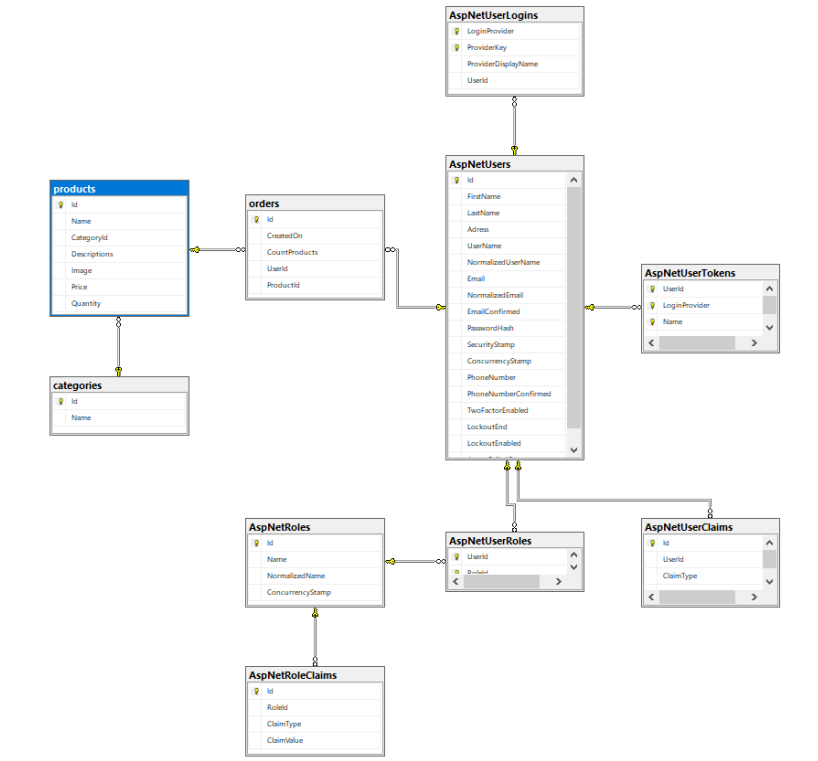
Това е класът за поръчки, той отговаря за поръчките и цялата им функционалност. Има полета за Идто на продуктите, кога са направени, наличността им и изобщо, всичко свързано с тях. Накрая имаме код за пресмятане на цялата цена.

```
18 references
public class Product
{
    [Key]
    [Required]
    6 references
    public int Id { get; set; }
    [Required]
    7 references
    public string Name { get; set; }
    [Required]
    5 references
    public int CategoryId { get; set; }
    0 references
    public virtual Category Category { get; set; }
    [Required]
    [MaxLength(150)]
    [MinLength(5)]
    6 references
    public string Description { get; set; }
    [Required]
    7 references
    public string Image { get; set; }
    [Required]
    [Range(0.01, 10000)]
    9 references
    public decimal Price { get; set; }
    [Required]
    [Range(1, 1000)]
    9 references
    public int Quantity { get; set; }
    0 references
    public virtual IEnumerable<Order> Orders { get; set; } = new List<Order>();

}
```

А това е класът отговарящ за продуктите и всичките им полета. Това включва идто на категориите, снимките, цената им и всичко, което трябва на продуктите, за да се появят в системата.

Полетата в класовете определят, какви данни ще се взимат или въвеждат от човека. Те са добавени собственоръчно и могат да се модифицират по всяко време от разработчика.

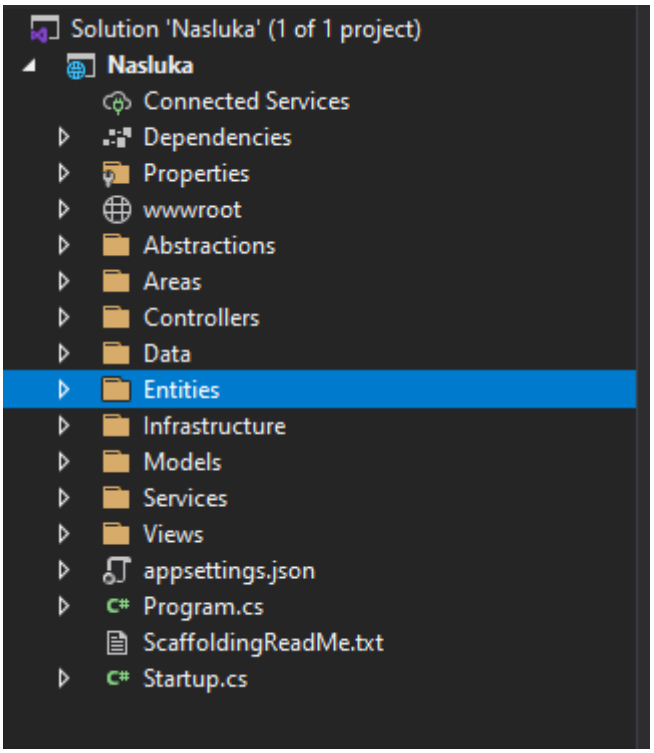
2.2 Диаграма на таблиците в програмата



Фигура 6: *Диаграма на таблиците в програмата и техните връзки.*

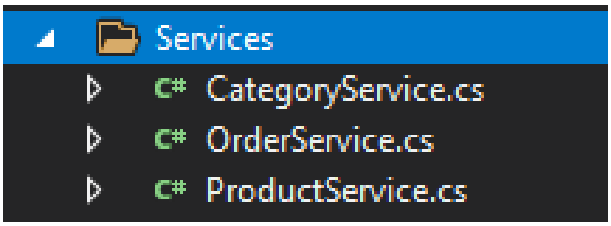
На тази диаграма може да се видят всичките таблици на програмата, както и връзката между тях. Те се свързват с наследяване. Освен това може да се погледне, и какви полета имат различните класове и да се разбере, какви данни се взимат или връщат между тях.

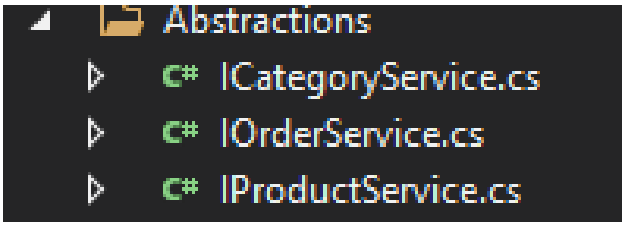
2.3 Съдържанието на приложението



☐ Съдържанието е съвкупност от: Контролери, вюта, вю-модели, сървиси, лейаути, ай-сървисис скафолднати айтеми и др. Всеки в съответна папка. Те са свързани помежду си и заедно образуват цялата функционалност на програмата, така че тя да работи правилно.

2.4 Сървиси и ай-сървисис





-Слоят на услугата е допълнителен слой в ASP.NET MVC приложение, който посредничи в комуникацията между контролера и слоя на хранилището. Слоят на услугата съдържа бизнес логика. По-специално, той съдържа логика за валидиране.

-Това са сървисите и ай-сървисите на програмата. Намират се в папките Services и Abstractions, което не им пречи да са зависими едни от други.

-Сървисите в една програма са инструментите, с които контролерите извличат информация от базата данни и я връщат. Те трябва да са съобразени с останалия код, за да работят правилно.

-Ай-сървисите регистрират полетата на сървисите, така че да се разпознават от програмата и останалия код.

-Те са съобразени с базата от данни, защото когато контролерът за поръчки приеме полетата на класът, сървисът ще му даде функционалността да прави промените, които искаме.

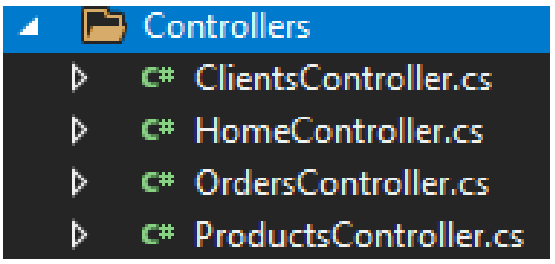
```
4 references
public interface ICategoryService
{
    3 references
    List<Category> GetCategories();
    1 reference
    Category GetCategory(int categoryId);
    1 reference
    List<Category> GetCategorieId(int categoryId);
}
```

☐ Това е ай-сървиса за категориите, направен е на интерфейс, за да може сървиса за категориите да го наследява.

```
2 references
public class CategoryService : ICategoryService
{

    private readonly ApplicationDbContext _context;

    0 references
    public CategoryService(ApplicationDbContext context)
    {
        _context = context;
    }

    0 references
    public Category GetCategoryById(int categoryId)
    {
        return _context.Categories.Find(categoryId);
    }

    3 references
    public List<Category> GetCategories()
    {
        List<Category> categories = _context.Categories.ToList();
        return categories;
    }

    0 references
    public List<Category> GetProductsByCategory(int categoryId)
    {
        return _context.Categories
            .Where(x => x.Id ==
            categoryId)
            .ToList();
    }

    1 reference
    public Category GetCategory(int categoryId)
    {
```

☐ Това е сървиса за категориите, които отговаря за категориите при създаване на продукт или неговата поръчка. Той работи с класът за категориите, както и с контролерът.

2.5 Контролери



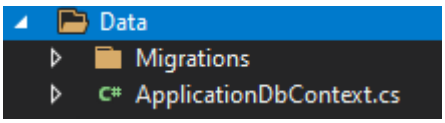
☐ Това са контролерите в програмата, те отговарят на останалия код, по такъв начин, че да работи правилно приложението и да не дава грешки. Те са в различна папка от сървисите, за да е подреден кодът и да работи правилно

```
1 reference
public class ProductsController : Controller
{
    public readonly ApplicationDbContext context;
    public readonly ICategoryService _categoryService;
    public readonly IProductService _productService;

    0 references
    public ProductsController(ApplicationDbContext context, ICategoryService categoryService, IProductService productService)
    {
        this.context = context;
        _categoryService = categoryService;
        _productService = productService;
    }

    0 references
    public ActionResult Create()
    {
        var product = new ProductCreateViewModel();
        product.Categories= _categoryService.GetCategories()
            .Select(c => new CategoryChooseViewModel()
            {
                Id = c.Id,
                Name = c.Name
            })
            .ToList();
        return View(product);
    }
    [HttpPost]
    0 references
    public IActionResult Create(ProductCreateViewModel bindingModel)
    {
        if (ModelState.IsValid)
```

☐ Това е част от контролерът за продуктите. В него се съдържа функционалността за създаването на продуктите, тяхното обновяване, изтриване и тн. Има различни сървиси в кодът му, защото контролерите работят с тях. За другите контролери важат почти същите полета, заради условията на приложението и неговата функционалност.

Контролерът е отговорен за контролирането на начина, по който потребителят взаимодейства с MVC приложение. Контролерът съдържа логиката за управление на потока за ASP.NET MVC приложение. Контролерът определя какъв отговор да изпрати обратно на потребител, когато потребителят направи заявка за браузър.

2.6 ApplicationDbContext

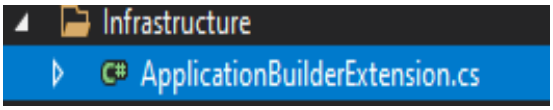


☐ ApplicationDbContext свързва сървъра на база данни с класовете на модела на данни в приложението Asp.Net.

```
22 references
public class ApplicationDbContext : IdentityDbContext<ApplicationUser>
{
    5 references
    public DbSet<Category> Categories { get; set; }
    4 references
    public DbSet<Order> Orders { get; set; }
    7 references
    public DbSet<Product> Products { get; set; }
    0 references
    public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options)
        : base(options)
    {
        this.Database.EnsureCreated();
    }
    0 references
    public DbSet<Nasluka.Models.OrderCreateBIndingModel> OrderCreateBIndingModel { get; set; }
    //public DbSet<Nasluka.Models.ProductAllViewModel> ProductAllViewModel { get; set; }
    //public DbSet<Nasluka.Models.ProductCreateViewModel> ProductCreateViewModel { get; set; }
    //public DbSet<Nasluka.Models.ProductDetailsViewModel> ProductDetailsViewModel { get; set; }
    //public DbSet<Nasluka.Models.OrderCreateBIndingModel> OrderCreateBIndingModel { get; set; }

    // public DbSet<Nasluka.Models.ProductAllViewModel> ProductBindingAllViewModel { get; set; }

    // public DbSet<Nasluka.Models.ProductCreateViewModel> ProductCreateViewModel { get; set; }
}
```

☐ В нашия случай ApplicationDbContext има полета за поръчка, продуктите и категориите им, което ще ги свърже с базата.

2.7 Application Builder Extension

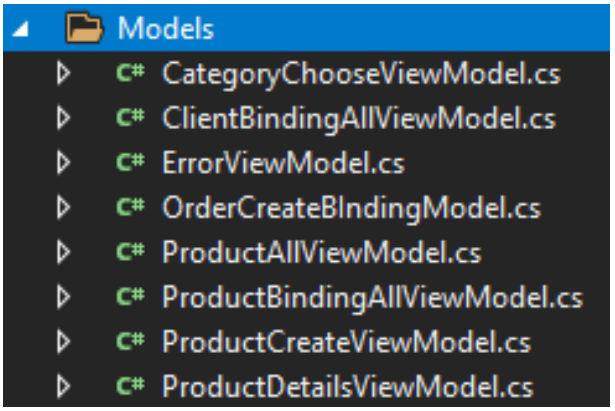


☐ В Application Builder Extension се съдържа информация за ролите в приложението, за администраторския профил, както и за категориите. Този клас играе роля на валидатор на информацията и я пази. Когато се наложи да бъде достъпена и използвана се взима от там.

```
0 references
public static class ApplicationBuilderExtension
{
    1 reference
    public static async Task<ApplicationBuilder> PrepareDatabase(this IApplicationBuilder
    {
        using var serviceScope = app.ApplicationServices.CreateScope();

        var service = serviceScope.ServiceProvider;
        var data = serviceScope.ServiceProvider.GetRequiredService<ApplicationDbContext>()


        await RoleSeeder(service);
        await SeedAdministrator(service);
        SeedCategories(data);
        return (ApplicationBuilder)app;
    }
    1 reference
    private static async Task RoleSeeder(IServiceProvider serviceProvider)
    {
        var roleManager =
            serviceProvider.GetRequiredService<RoleManager<IdentityRole>>();

        string[] roleNames = { "Administrator", "Client" };
        IdentityResult roleResult;

        foreach (var role in roleNames)
        {
            var roleExist = await roleManager.RoleExistsAsync(role);

            if (!roleExist)
            {
                roleResult = await roleManager.CreateAsync(new IdentityRole(role));
            }
        }
    }
```

☐ Това е част от кода на класа. В него се виждат ролите, както и част от сървиси, които са използвани, за да работи правилно кодът и да се вземат данните правилно данните в него.

2.8 Вю-Модели



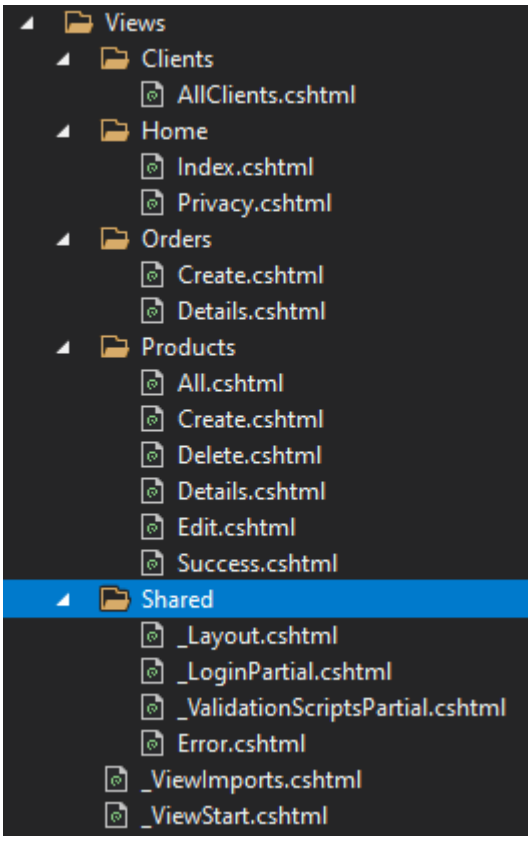
☐ В ASP.NET MVC ViewModel е клас, който съдържа полетата, които са представени в строго типизирания изглед. Използва се за предаване на данни от контролера към строго въведен изглед. В програмата има полета, които са направени за нея, но могат да се направят всякакви, в зависимост от програмата.

```
3 references
public ProductCreateViewModel()
{
    Categories = new List<CategoryChooseViewModel>();
}
[Key]
4 references
public int Id { get; set; }

[Required]
[MaxLength(30)]
[Display(Name = "Name")]
12 references
public string Name { get; set; }
[Required]
[MaxLength(30)]
[Display(Name = "Description")]
12 references
public string Description { get; set; }
[Required]

[Display(Name = "Category")]
10 references
public int  CategoryId { get; set; }
[Required]
[Display(Name = "Price")]
[Range(0.01, 10000)]
12 references
public decimal Price { get; set; }
[Required]
[Display(Name = "Image")]
12 references
public string Image { get; set; }
[Required]
[Range(1, 1000)]
[Display(Name = "Quantity")]
```

В този случай полетата са съобразени с нуждите ни. Добавени са неща свързани със създаването на продукт, защото това е вю-моделът за създаване на продукти, ако беше вю-моделът за детайлите, полетата щяха да са други. В кодът може да се видят [Required], [MaxLength] и различни други условия за даденото поле. Има всякакви условия, но в този случай значи, че полето е задължително и до определена дължина.

2.9 Вюта

В модела Model-View-Controller (MVC) изгледът обработва представянето на данните на приложението и взаимодействието с потребителя. Изгледът е HTML шаблон с вградено маркиране на Razor. Razor маркирането е код, който взаимодейства с HTML маркирането, за да създаде уеб страница, която се изпраща на клиента. В програмата са създадени вюта, които да отговарят на функционалността ѝ. Те се намират в папка Views, като всяко вю е в отделна папка, спрямо функциите, които върши и кодът, към който принадлежи.

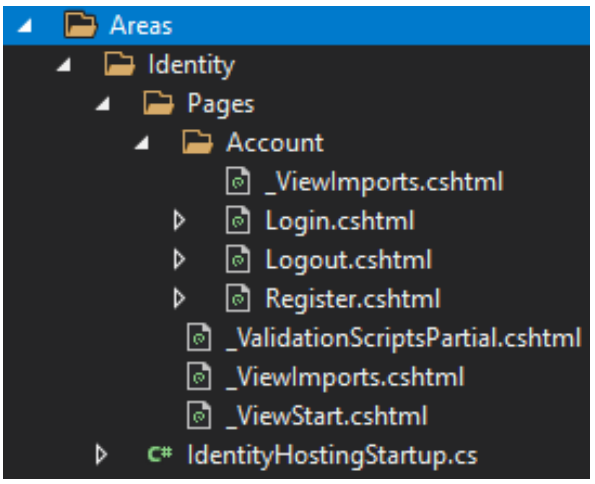



```
@model Nasluka.Models.ProductCreateViewModel

@{
    ViewData["Title"] = "Create";
}

<h1>Create</h1>

<h4></h4>
<hr />
<div class="row">
    <div class="col-md-4">
        <form asp-action="Create">
            <div asp-validation-summary="ModelOnly" class="text-danger"></div>
            <div class="form-group">
                <label asp-for="Name" class="control-label"></label>
                <input asp-for="Name" class="form-control" />
                <span asp-validation-for="Name" class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="Description" class="control-label"></label>
                <input asp-for="Description" class="form-control" />
                <span asp-validation-for="Description" class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="CategoryId" class="control-label"></label>
                <select asp-for="CategoryId" class="form-control">
                    @foreach (var category in Model.Categories)
                    {
                        <option value="@category.Id">@category.Name</option>
                    }
                </select>
                <span asp-validation-for="CategoryId" class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="Price" class="control-label"></label>
                <input asp-for="Price" class="form-control" />
                <span asp-validation-for="Price" class="text-danger"></span>
```

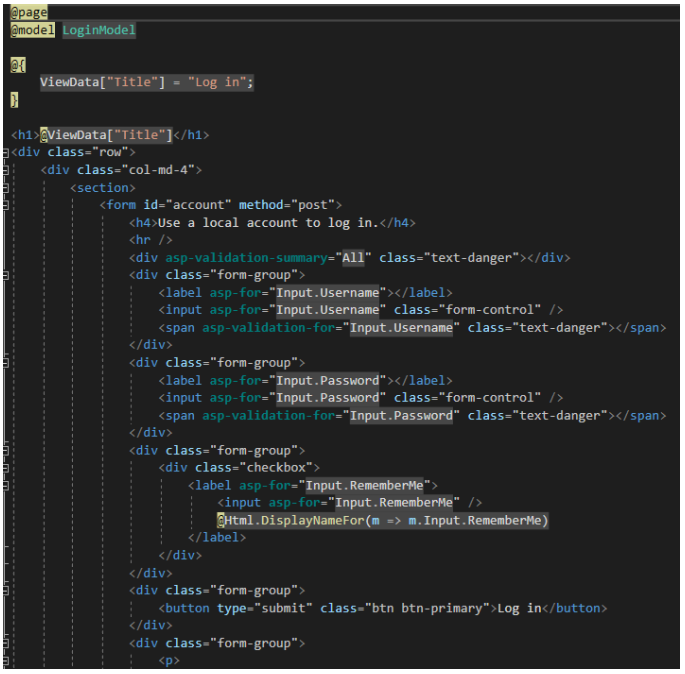
```
@model Nasluka.Models.ProductCreateViewModel

@{
    ViewData["Title"] = "Delete";
}

<h1>Delete</h1>

<h3>Are you sure you want to delete this?</h3>
<div>

    <hr />
    <dl class="row">
        <dt class = "col-sm-2">
            @Html.DisplayNameFor(model => model.Name)
        </dt>
        <dd class = "col-sm-10">
            @Html.DisplayFor(model => model.Name)
        </dd>
        <dt class = "col-sm-2">
            @Html.DisplayNameFor(model => model.Description)
        </dt>
        <dd class = "col-sm-10">
            @Html.DisplayFor(model => model.Description)
        </dd>
        <dt class = "col-sm-2">
            @Html.DisplayNameFor(model => model.CategoryId)
        </dt>
        <dd class = "col-sm-10">
            @Html.DisplayFor(model => model.CategoryId)
        </dd>
        <dt class = "col-sm-2">
            @Html.DisplayNameFor(model => model.Price)
        </dt>
        <dd class = "col-sm-10">
            @Html.DisplayFor(model => model.Price)
        </dd>
        <dt class = "col-sm-2">
```

Това са две вюта, които отговарят за Create и за Delete. Кодът вътре в тях показва, какво ще изведат на екрана, както и какви заглавия имат, кой вю-модел използват и като цяло цялата им функционалност, която им трябва, за да работят правилно.

3.0 Скафолднат айтем



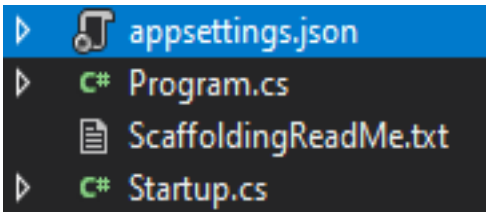
☐ Това е айтем генериран автоматично от компютъра, като ние му задаваме само нужните cshtml, които искаме. Отговаря за логването и излизането от сайта, както и за регистрацията в него.



```
@page
@model LoginModel

@{
    ViewData["Title"] = "Log in";
}

<h1>@ViewData["Title"]</h1>
<div class="row">
    <div class="col-md-4">
        <section>
            <form id="account" method="post">
                <h4>Use a local account to log in.</h4>
                <hr />
                <div asp-validation-summary="All" class="text-danger"></div>
                <div class="form-group">
                    <label asp-for="Input.Username"></label>
                    <input asp-for="Input.Username" class="form-control" />
                    <span asp-validation-for="Input.Username" class="text-danger"></span>
                </div>
                <div class="form-group">
                    <label asp-for="Input.Password"></label>
                    <input asp-for="Input.Password" class="form-control" />
                    <span asp-validation-for="Input.Password" class="text-danger"></span>
                </div>
                <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>
                <div class="form-group">
                    <button type="submit" class="btn btn-primary">Log in</button>
                </div>
                <div class="form-group">
                    <p>
```

☐ Прилича на вю, но върши повече функции. В него е събрана логиката на логването, както и самото вю. Това е csthml отговарящ за логването на потребителите, като полетата са генерирани автоматично, а ние можем да променяме съдържанието му.

3.1 Startup.cs, Program.cs, appsettings.json



☐ Program.cs е мястото, където стартира приложението. Програма. cs клас файл е входна точка на нашето приложение и създава екземпляр на IWebHost, който хоства уеб приложение. Startup.cs файл е входна точка и ще бъде извикан след Program. cs файл се изпълнява на ниво приложение. Той обработва тръбопровода за заявки. Класът за стартиране задейства в секундата, когато приложението стартира. Appsetting json е мястото, в което си кръщаваме базата данни и точката и за достъп.

Заключение

1.Уеб приложението

- ☐ Програмата има своето бъдеще, заради постоянно развиващият се свят и компютърни технологии. Тя е с такъв код, който може да се променя и усъвършенства, от програмисти и разработчици на такива програми.
- ☐ Предлага гъвкавост и разнообразие във функциите си, отговарящи на пазара и неговото текущо ниво, като доставя опция за развитие по всяко едно време.

2. Функционалността

- ☐ Функционалността е направена, така че да работи и е свързана по такъв начин, че да не дава грешки ако се използва правилно.
- ☐ Кодът е написан съгласно условията поставени в дипломния проект и е в постоянна готовност за развитие.
- ☐ Всички необходими компоненти са направени.

3. Приложение

- ☐ Приложението може да се използва свободно, като сайт. Трябва да бъде качено на сървър и да се поддържа единствено.
- ☐ При промяна на нуждите, кодът единствено трябва да се допише и свърже по такъв начин, че да работи с текущия такъв.

Източници:

https://bg.wikipedia.org

https://docs.microsoft.com

https://classroom.google.com

https://www.google.com/?hl=bg