**ПРОФЕСИОНАЛНА ГИМНАЗИЯ ПО МЕХАНОТЕХНИКА, ЕЛЕКТРОНИКА, ТЕЛЕКОМУНИКАЦИИ И ТРАНСПОРТ „ХРИСТО БОТЕВ“**



**ДИПЛОМЕН ПРОЕКТ**

на Сечкин Салим Рахим ученик от 12А клас

**Тема: Уеб приложение за**

**приемане и предлагане на**

**услуги**

**(Freelancer)**

Професия и специалност: Приложно

програмиране

Ръководител/Консултант: Недялка Йорданова

дата: xx.xx.2023 г.

гр. Шумен

**Съдържание на дипломния проект**

1. Титулна страница
2. Съдържание
3. Увод
4. Основна част
5. Заключение
6. Списък на използваната литература
7. Приложения

**Увод (въведение)**

Целта на уеб приложението е да помага на хора, които имат определени знания и искат да печелят пари, но не знаят от къде и как да започнат. Сайтът позволява качване и приемане на обяви, създаване на специална заявка, комуникация между продавача и купувача, комуникация с администратора. Продавачът може да види напредъка на купувача, но само ако купувача го отбележи в самата обява. Отбелязването става в проценти. Купувачът има право да избере допълнителни услуги от сравнителните таблици, според зависи цената. Ревизията, времето за предаване и другите неща зависят от цената.

**Основна част**

Уеб приложението е лесно за използване с проста навигация и с лесно разбираеми опции. Човек, който може да използва социални медии като Facebook, Instagram и други подобни сайтове, той ще може да управлява и този сайт. Функциите са следните: качване на обява - такава каквато трябва на Вас или на другите. При добавяне, добавящият е задължен да попълва всички полета които имат червена звезда \*. Тези полета са „Какво предлагаш?“, тоест наименованието на обявата; „Крайна дата за предаване“ или „Валидност на самата обява“; „ Цена “; „Описание“, като описанието трябва да е възможно детайлно и подробно без правописни, пунктуационни, лексикални и граматични грешки за да изглежда по-професионално и да извлече повече внимания; „Снимка“ – тя трябва да е оригинална и собствена, а не открадната от интернет; „Сравнителни пакети“ – те дават възможност на купувачите да си изберат подходяща за тях, цена с плюсове и минуси, които също зависят от цената. Тези плюсове и минуси например са броят на ревизиите, по-бързо предаване, допълнителна информация, която включва в себе си изготвяне на лого и подобни неща. След попълването на тези полета, се избира бутонът „Създай обява“ и след секунди обявата се появява в секция „Обяви“. В тази секция са всички споделени оферти и обяви, които съдържат кратко представяне на цялата информация. Достъпването на цялата информация става след кликването на бутона „Детайли“, който се намира в най-долната част на описателната картичка на обявата. Следващата секция е „Услуги“, който включва в себе си категориите и кратка информация под тях за наличните услуги. Секция „За нас“ включва информация за администратора, реклама на социалните му медии и процеса на обработка на самия сайт, използвани езици, технологии и платформи. Следващата публична секция от навигацията е „Контакти“, който включва себе си един формуляр за изпращане на съобщение до администратора със следните полета: Име на изпращащия, И-мейл на изпращащия, заглавие на наличния проблем и поле за въвеждане на съобщение, който няма ограничение на знаци. Секцията „Портфолио“ е падащо меню с две опции, които са „Моите поръчки“ и „Моите заявки“. В първата опция са поръчките, в които се отбелязва напредъка, а във втората опция се намират заявките създадени от мен. Създаването им е доста просто. След натискането на бутона „Създай заявка“, се препраща към друг HTML файл който е стилизиран с CSS, Bootstrap, JavaScript и функционализиран със C#. Той включва падащо меню за избиране от наличните категории, инпут поле за заглавие, сума, описание и срок. След извършването на всичките нужни функции, заявката се появява в Index-a. Създаването на акаунт става от бутона “Регистрация„ в публичната навигация и е напълно безплатно, бързо и изключително лесно, а ако имате вече съществуващ акаунт, можете да влезете в профила си от публичния бутон „Вход“.

**Изработването на сайта**

Уеб приложението е програмиран с езиците HTML, CSS, Bootstrap, JavaScript, MVC .Net Core и C# във Visual Studio 2022.

HTML-ът е използван за скелета (строежа) на самия сайт. С езика CSS е оформен дизайнът, с JavaScript са направени по-напредналите ефекти и някои прости математически пресмятания, като пресмятане на рождена дата. А със C# са направени операциите, като например Качване на нова обява, редактиране на вече съществуваща обява и изтриване на както съществуваща и редактирана, така и съществуваща но не редактирана обява.

**HTML** (***H****yper* ***T****ext****M****arkup****L****anguage*) е основният маркиращ език за описание и дизайн на уеб страници. HTML е стандарт в интернет, а неговите стандарти се определят от международния консорциум W3C. Текущата версия на стандарта е HTML 5.0, а предходната стабилна версия е HTML 4.1.

Описанието на документа става чрез специални елементи, наречени HTML елементи или техните маркери, които се състоят от тагове и съответстващите етикети (HTML tags) и ъглови скоби (като например елемента <html>). HTML елементите са основната градивна единица на кода, който изграждат уеб страниците. Чрез тях се форматира, графично оформя текста и неговите отделните части в рамките на една уеб страница, като например заглавия, цитати, текстови раздели, хипертекстови препратки и т.н. Най-често HTML елементите са групирани по двойки <h1> и </h1>.

В повечето случаи HTML кодът е написан в текстови редактори, с файлов формат .html, .htm, dhtml и се качва и хоства на сървъри, които са онлайн в интернет или са част от www мрежата. Тези .html файлове съдържат програмно на таговете на HTML и текстово съдържание със маркери и коментари – също инструкции за браузъра, за това какъв точно тип е .html страницата, а също за това как да се показва текстът, особено що се отнася до езиковите характеристики. За да се илюстрира как се включва текст в HTML код: <маркер> Някакъв текст. </край на маркера>. уеб браузърите са програмирани от своя страна така, в повечето случаи, макар че някои браузъри могат да имат съответно проблеми на версията, за да могат да прочетат HTML документите и да ги покажат на екрана като уеб страници. Браузърите не показват самите HTML тагове, освен ако не се отиде в менюто за да се направи това, така че те „интерпретират“ (тоест парсват) съдържанието на страницата като код и текст за да могат след работа на процесора да покажат желаното уеб-съдържание.

Основното предимство на HTML е, че уеб-страниците, които са го включват в кода си, могат да се разглеждат чрез показването им от браузъра на екрана на повечето устройства. Уебстраницата може да има дизайн, който дори изглежда с добър дизайн с помощта на CSS или „правилно оформен“ (например с помощта на C#), както върху монитора на персоналния компютър, но също и върху миниатюрния дисплей на пейджър или дисплея на мобилен телефон.

HTML може да прикрепя скриптове писани на езици като JavaScript, който е помощен за HTML, и това променя поведението на дадена уеб страница. Cascading Style Sheets (CSS) се използват, като това се прави за да се определя изгледа и оформлението на текста и други включени в страницата изображения и илюстриращи материали. World Wide Web Consortium (W3C) поддържа както HTML, така и CSS, и насърчава използването на CSS в HTML страниците още от 1997. Това допринася за разделяне съдържанието и структурата на уеб страниците от тяхното визуално представяне.

**CSS** (**Cascading Style Sheets**) е език за описание на стилове (език за стилови файлове, style sheet language) – използва се основно за описание на онлайн представянето на уеббазиран документ, който написан на език за маркиране. Най-често се използва допълнително към чистия HTML, но се прилага и върху XML уебстраници и документи. Спецификацията на CSS официално се поддържа от W3C.

CSS още в началото на развитието на www започва да се добавя към стандартния HTML с цел да бъдат разделени съдържанието и структурата на уеб страниците отделно от тяхното визуално представяне. Преди стандартите за CSS, установени от W3C през 1995 г., съдържанието на сайтовете и стила на техния дизайн са писани в една и съща HTML страницата. В резултат на това HTML кодът се превръща в сложен и нечетлив, а всяка промяна в проекта на даден сайт изисквала корекцията да бъде нанасяна в целия сайт страница по страница. Използвайки CSS, настройките за форматиране могат да бъдат поставени в един-единствен файл и тогава промяната ще бъде отразена едновременно на всички страници, които използват този CSS файл. CSS позволява да се определя как да изглеждат елементите на една HTML страница – шрифтове, размери, цветове, фонове, и др. CSS кодът се състои от последователност от стилови правила, всяко от които представлява селектор, последван от свойства и стойности. Например в следния CSS код.

**Bootstrap** - Основната полезност на използването на bootstrap е да се намали пристрастието в рамките на анализа или, с други думи, да се сближи дисперсията чрез извършване на произволно преизбиране на първоначалната извадка, а не на популацията. По този начин изграждането на статистически модели се улеснява чрез създаване на доверителни интервали и тестове за хипотези.Въпреки че априори може да изглежда много сложна практика, процедурата, на която се основава първоначалното зареждане, е просто създаване на голям брой проби, преместващи данните, приемащи като референция първоначална популационна извадка.Тази техника е особено полезна в ситуации, когато наличните проби са малки или, както беше споменато по-горе, ако разпределението е силно изкривено. В този смисъл те помагат за решаването на множество проблеми с вероятността и приложна статистика. Една от основните характеристики на тази практика е, че тя включва последващо преизбиране, за да се получат затворени изрази и да се реши математическата сложност на тези операции. С развитието на компютрите и технологичните инструменти през последните години стана по-лесно да се разчита на използването на бутристрапинг за сложно преизбиране. Положителен аспект на повторното вземане на проби от bootstrap е, че той е опростил статистическите методи, в смисъл че е заменил изграждането на класически и много сложни математически модели с изчисление с помощта на специфичен софтуер, което е подобрило тяхната приложимост или достъп до други области или изследвания.

**JavaScript -** JavaScript е един от най-използваните скриптови езици за програмиране в Интернет. Разработен е през 1995 година от Брендан Айк и осигурява моментното и динамично взаимодействие между потребител и браузър. Вярно е, че извървява дълъг път, но днес смело може да заявим, че почти няма нещо, което JavaScript не може да направи. Благодарение на него уеб страниците, които използваме в момента, изпълняват много повече функции от това просто да зареждат данни.

Последните проучвания за 2017 показват, че 94,9% от всички уебсайтове в мрежата използват този език за програмиране. Най-популярните сред които са Google, Yahoo, Amazon, Facebook, Wikipedia и т.н.  Защо? Защото JavaScript позволява създаването на интерактивни и адаптивни интерфейси с много добър UI дизайн и динамични функционалности, изцяло с насоченост към крайния потребител. Функциите на JavaScript са се разширили значително през годините. Първоначално езикът просто е регулирал начинa, по който потребителят си взаимодейства с уебсайта: онлайн въпросници и форми за контакт, подаване на обратна връзка под формата на известия и важни съобщения и др. Днес обаче езикът е еволюирал до такава степен, че с него може дори да управлявате дрон. За какво още се използва JavaScript:

* Създаване на мобилни приложения
* Разработване на десктоп и уеб апликации
* Зареждане на информация чрез AJAX
* Изработване на операционни системи за смарт часовници
* Разпознаване на използвания браузър и променяне на уеб страниците спрямо това
* Анимиране на елементи от уеб страниците
* Създаване на поп-ъп прозорци
* Изготвяне на презентации под формата на уебсайт
* Периодично обновяване на променливи данни без намесата на потребителя
* Функционира дори и офлайн
* и др.

Предимствата на JavaScript са много. Именно те го превръщат в един от любимите езици за програмиране и на нашите девелъпъри.. Освен че разширява функционалността на уеб страниците, той е изключително лесен език за научаване и практикуване. Насочен е изцяло към осигуряване на добро потребителско преживяване, при това бързо и качествено.

**Asp.NET Core -** ASP.NET Core е безплатна софтуерна рамка за уеб разработка, с отворен код. Също така тя се явява и следващата стъпка в еволюцията ASP.NET. Тя е разработена съвместно от Microsoft и общността, която е събрала през годините на своето развитие. ASP.NET Core е модуларна софтуерна рамка, която може да върви както на пълната .NET рамка, така и на крос-платформената .NET Core. Въпреки, че е нова софтуерна рамка, изградена върху нов web stack, тя има висока степен на съвместимост с ASP.NET MVC. Спрямо своите предшественици ASP.NET Core поддържа нова функция – т.нар. „side by side versioning”. При нея различни приложения, които използват една и съща машина, могат да таргетират различни версии на ASP.NET Core, в зависимост от версиите (и нуждите) си. Това не е възможно с по-стари издания на ASP.NET.

**MVC -** MVC е съкратено от “Model-view-controller”. Представлява архитектурен шаблон при програмния дизайн. Той отговаря за разделянето на бизнес логиката на три взаимосвързани части: “Model”, “View” и “Controler”. Какво представляват те и какви са техните функции?

* „**Model“ (Модел)** – това представлява „ядрото“ на приложението ви. В него се записват всички данни, върху които искаме да работим; 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.
* **„View“(Изглед)** – най-общо казано това е изходна част от софтуера, която визуализира наличните, обработени данни; 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. Зависи от това какъв графичен инерфейс се прави и каква платформа се използва;
* **„Controller“(Контролер)** – това е тази част от библиотеката (library) или самия код, която извиква допълнителни методи върху събраните в модела данни и по този начин те могат да бъдат обработвани. 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.

**C# -** Той е създаден от Microsoft през 2001 г., а последното обновление за него излезе в края на 2021, т.е. това е един **актуален език**. Може да се каже, че C# e Microsoft-ският отговор на програмния език Java, продукт на компанията Sun Microsystems (понастоящем собственост на Oracle Corporation).C# е **обектно-ориентиран език**, т.е. той изгражда системи посредством обекти, които взаимодействат помежду си. Обектно-ориентираното програмиране се отличава от традиционното програмиране, при което програмата е просто списък от инструкции, които компютърът изпълнява.C# е програмен **език от високо ниво**. Това го прави изключително достъпен за начинаещи програмисти. Като език от високо ниво, той е по-лесно четим и разбираем, тъй като е по-близо до хората, отколкото до машините (които, както вероятно знаеш, говорят само с нули и единици). Това дава възможност на програмиста да се фокусира върху разработката, правейки процеса по-лесен, бърз и разбираем.

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

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

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

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

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

***Подход CodeFirst***

При изработването на уеб приложението е използван подходът CodeFirst, при който ние пишем кода, а таблицата (базата данни) се генерира автоматично.

***Подход DataBase First***

Този подход е обратният на подхода CodeFirst. Базата се създава от нас, а кодът се генерира автоматично.

**Какво е база данни?**

Представлява колекция от логически свързани данни в конкретна предметна област, които са структурирани по определен начин. В първоначалния смисъл на понятието, използван в компютърната индустрия, базата от данни се състои от записи, подредени систематично, така че компютърна програма да може да извлича информация по зададени критерии. Например БД може да се използват в моделирането на хотелските системи, за да се проверява дали има налични свободни стаи в даден хотел.

Поддръжката на база от данни се осъществява от т.нар. система за управление на бази от данни (СУБД).

Система за управление на бази данни е компютърно приложение (софтуер) създадено за комуникация между потребителя, други приложения, както и други БД, с цел да се сравнят и анализират данни. Общото специфично предназначение на СУБД е да позволи определянето, създаването, заявки, актуализацията и администрирането на бази данни. Добре известни СУБД включват MySQL, PostgreSQL, Microsoft SQL Server, Oracle, Sybase, SAP HANA, и IBM DB2. Бази данни не са съвместими с различните СУБД, за това различните СУБД работят със стандартни като SQL и ODBC или JDBC, за да позволи на всяко приложение да работи с различни СУБД, а така и с различни БД. Управлението на БД често се избира от модела им, които те подкрепят. Най-използвани системи от бази данни от 1980 г. насам са всички поддържани релационния модели на езика SQL. Често срещано е СУБД да се нарича само „база данни“.

При изработването на сайта „Предлагане и приемане на услуги“ е използвано приложението Microsoft SQL Server Management Studio 18 (SSMS). То улеснява работата с базата данни. Лесно се достъпват създадените таблици и съществуващите в тях, полета. Притежава свойства, като автоматично създаване на диаграма с техните връзки и много други многофункционални функции, улесняващи работа.Съществуват три типа организация на данните и връзките между тях.

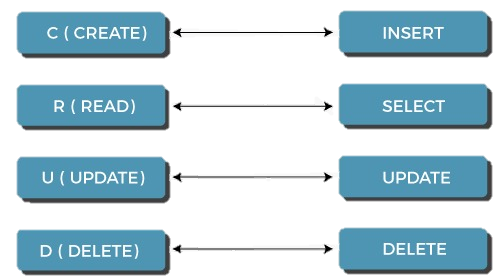
**CRUD операции**

В проекта CRUD операциите са използвани при обявите за качване, прочитане, редактиране и изтриване на записи. CRUD парадигмата е обичайно явление при създаването на уеб приложения, защото осигурява начин, който да напомни на разработчиците как да конструират пълноценни и полезни използваеми модели. В този контекст, CRUD абревиатурата означава **Create, Read, Update, Delete**. Това са четирите основни функции, свързани с т.нар. **persistent storage**, т.е. съхранение на данни, при което данните се запазват и след като захранването към устройството за съхранение бъде прекъснато. Повечето приложения разполагат с някаква форма на CRUD функционалности и на практика всеки програмист работи с такива в даден момент. Такива приложения имат **три основни съставки** – API, база данни, която съхранява информацията и позволява на потребителите да я достъпят, и UI, чрез който да го направят. А в следващите редове ще разгледаме и самите операции една по една.

## **Как работят отделните CRUD операции със SQL?**

Всяка една от буквите на акронима може да бъде свързана с SQL, затова двете вървят ръка за ръка, когато работиш с релационни бази данни. А действието на операциите е сравнително очевидно:

* **CREATE операции** – операцията създава нов запис, например нов акаунт или задача. В SQL база данни, за да създаваш, ще използваш INSERT.
* **READ операции** – чрез тях, използвайки потребителския интерфейс, можеш да достъпиш и видиш създадените записи. В релационни бази данни ще използваш SELECT, за да избереш запис.
* **UPDATE операции** – чрез тях можеш да променяш и обновяваш съществуващите записи, отново посредством наличния UI. В релационна база данни ще използваш UPDATE.
* **DELETE операции** – чрез тях можеш да премахваш записи от базата данни. В SQL база данни на тази операция съответства



**Таблици в проекта**

1. AppUser – полета:

* Id
* FirstName
* LastName

1. Category – полета:

- Id

- Name

1. ContactUs – полета:

* Id
* Name
* E-mail
* Subject
* Message

1. Image – полета

* Id
* Extension

1. Job – полета:

* Id
* Name
* Description
* Price
* DeadLine
* Progress
* CategoryId
* AppUserId
* Progress
* Status
* Полета на сравнителните таблици

1. Request – полета:

* Id
* Title
* Sum
* Description
* DeadLine

1. Skill – полета:

* Id
* Name

1. UserSkill – полета:

**Релационни връзки между таблиците**

--------------------------------------(***ЗА ДОПИСВАНЕ***)--------------------------------

**Релационни връзки**

Връзките (релациите) обединяват отделните таблици, което повишава ефективността от използването им. Таблиците могат да се използват всяка сама за себе си, но възможностите нарастват значително, когато се използват заедно. Връзките са ключ към поддържане коректността на информацията и целостта на базата от данни. Връзка между две таблици се установява като във всяка от тях се дефинират полета, които съдържат информация, обща за двете таблици. Обикновено първичният ключ в едната от таблиците се свързва към съответното поле в другата таблица.

**Видове връзки**

Съществуват три вида връзки:

1. **1:1** (едно-към-едно/one-to-one) – на един запис от едната таблица отговаря точно един запис от друга таблица.
2. **1:М** (едно-към-много/one-to-many) – на един запис от първичната таблица отговарят няколко записа от вторичната таблица.
3. **М:М** (много-към-много/many-to-many) – на няколко записа от една таблица отговарят няколко записа от друга таблица. При такива връзки няма еднозначност. Те могат да се преобразуват посредством междинна таблица в две връзки 1:много.

Таблицата, която съдържа първичния ключ, се намира от страна едно на връзка. Прието е да се нарича първична таблица. Таблицата от страна много на връзката се нарича свързана, а съответстващото поле външен ключ.

**Автентикация и авторизация**

**Автентикация**

Процес, при който се проверява кой е потребителят, който ще ползва дадена система.

Потребителите могат да се идентифицират по различни начини – чрез парола, пръстов отпечатък,

смарт карта и др. Валидиране може да се осъществи и чрез изпращане на SMS на телефон, писмо

на е-майл и др. Възможно е също през външен провайдър – google акаунт, facebook и др.

**Авторизация**

Означава какви права има даден потребител, до какви функции и ресурси има достъп. Определя

страниците, които може да вижда даден потребител от уеб приложението.

**ASP.NET Core Идентичност**

1. Системата ASP.NET Core Identity

* Система за удостоверяване и упълномощаване за ASP.NET Core
* Поддържа ASP.NET MVC, Pages, Web API (JWT), SignalR
* Работи с потребители, потребителски профили, влизане / излизане,

роли и т.н.

* Работи със съгласието за бисквитки и GDPR
* Поддържа външни доставчици за вход

Facebook, Google, Twitter и т.н.

* Поддържа база данни, Azure, Active Directory, потребители на

Windows и т.н.

* Обикновено данните за идентичност на ASP.NET Core се

съхраняват в релационна база данни

* Данните се запазват с помощта на Entity Framework Core
* Имате известен контрол върху вътрешната схема на базата
* Данни
* Настройка на идентичността на ASP.NET
* Използване на шаблоните на ASP.NET за проекти от Visual Studio
* И след това го персонализирайте
* На ръка
* Инсталирайте NuGet пакети, ръчна конфигурация, създавайте EF карти (модели), преглеждайте модели, контролери, изгледи и т.н.
* Необходим пакет NuGet
* Microsoft.AspNetCore.Identity.EntityFrameworkCore

**Интернет**

* Определение - Устройства, които а свързани помежду си в една обща мрежа.
* Как работи? – работи с мрежови протоколи – HTTP, FTP, IP и др.
* Пакети – Съдържат IP адресите на потребителит на тези, които изпращат и получават.
* Протоколи – IPv4, IPv6

**Протоколи**

* **HTTP**

**Протокол за пренос на хипертекст** : ***H****yper****t****ext****T****ransfer****P****rotocol, HTTP*) е Мрежов протокол, от приложния слой на OSI модел за пренос на информация в компютърната мрежа. Създаден като средство за публикуване на HTML страници, протоколът довежда до формирането на Свтовната мрежа.

* **Методи на заявките**

HTTP определя 8 различни клиентски метода за заявки:

* **HEAD** – иска изпращане на заглавията отговарящи на посочения с URL ресурс. Отговорът на сървъра е идентичен с този на GET, но е с липсващо тяло. По този начин може да се вземе само метаинформацията, свързана с ресурса, спестявайки трансфера на целия файл.
* **GET** – с него клиентът прави заявка за ресурс, зададен чрез URL. Могат да се изпращат и ограничено количество данни, закодирани директно в самия URL (отделени чрез въпросителен знак).
* **POST** – позволява клиентът да изпрати данни на сървъра. Тази заявка обикновено се генерира при изпращането на уеб формуляр, а данните могат да бъдат: текст, написан от потребителя във формуляра; файл на клиентския компютър и др.
* **PUT** – качва файл, който в бъдеще ще отговаря на посочения URL
* **DELETE** – изтрива посочения ресурс.
* **TRACE** – сървърът връща получената заявка със статус OK. Позволява да се провери в какъв вид пристига заявката при сървъра и дали (и как) е била модифицирана по трасето от междинни прокси сървъри.
* **OPTIONS** – сървърът трябва да отговори с поддържаните от него клиентски методи, съответстващи на зададения URL, или с поддържаните от сървъра методи като цяло, ако е зададено \* вместо URL.
* **CONNECT** – използва се при комуникация през прокси.

Тези методи се пишат в тага <form method=” POST/GET…”>…</form>.

**Миграции**

Позволява да се правят промени върху таблиците.

Миграциите са удобен начин за промяна на таблицата на базата данни с течение на времето по последователен начин. Те използват Ruby DSL, така че да не се налага да пишете SQL на ръка, което позволява таблицата и промените да бъдат независими от базата данни.

Всяка миграция е като нова "версия" на базата данни. Една схема започва с нищо в нея и всяка миграция я променя, за да добави или премахне таблици, колони или записи. Active Record знае как да актуализира схема по тази времева линия, привеждайки я от каквато и точка да е в история до най-новата версия. Active Record също ще актуализира файла, за да съответства на актуалната структура на базата данни.

**Създаване на миграция**

Създаването на миграция става по няклко начина. Пирвият начин е чрез Power Shell-а, а вторият е чрез Visual Studio с Console Manager-а. При първия начин се инсталират нужните пакети и след инсталацията се пише следния код:

dotnet ef migrations add (*името на миграцията*)

и миграцията се създава, но разбира се ако няма грешки. Грешките могат да са: липса на първичен ключ в съответна таблица, нужда от първичен ключ, грешка в самия код, ако приложението е стартирано и др.

Вторият начин е по-лесен. Във Visual Studio се отваря Console Manager-а и там се пише следния код:

Add-migration (*името на миграцията*)

След това е нужно да се ъпдейтне базата данни. Той става със следния код:

Update-database. Този код се пише, след като се създаде миграцията, пак в същия конзол мениджър и базата се ъпдейтва автоматично, разбира се ако няма грешки.

**Съдържание на папките**

1. Папки
   1. wwwroot
      * всички снимки
      * Css файлове
      * JavaScript файлове
      * Bootstrap файлове
   2. Areas
      * Логин и регистър файловете се появяват в тази папка след скафолдването.
   3. Controller
      * Входната точка на уеб приложението, която поема всяка заявка от браузъра. Всеки контролер представлява клас, който наследява класа Controller. Името на контролера задължително трябва да съдържа две неща: име + Controller.
      * Вески такъв клас притежава методи, които се наричат екшъни. Имената на екшъните задължително трябва да съвпадат с имената на HTML файловете, иначе конкретният файл няма да се отвори и ще даде грешка.
      * HomeController – контролер, който се появява автоматично след създаването на проекта.
      * JobController – съдържа CRUD операциите...
      * RequestController
      * CategoryController – съдържа само CRUD операциите на файла за добавяне на категория.
   4. Model
      * .cs файлове, които не са към базата данни
   5. View
      * .cshtml файлове – html код заедно със C# код
      * Изгледа на сайта
   6. Data

* Model
  + - Съдържанието на таблиците с техните полета
    - .cs файлове, които са към базата данни
* ApplicationDbContext
  + - Описанието на таблиците с DbSet<>-ове
    - Кода за създаване на миграция
  1. Layout
     + Общите части на сайта – фютър и навигация
  2. Program.cs (Startup)
     + Може да конфигурира удостоверяване въз основа на бисквитки (или JWT);
     + Може да активира външно влизане (напр. Вход във Facebook);
     + Може да промени настройките за идентификация по подразбиране;
     + Може да активира RoleManager с .AddRoles <IdentityRole> ();

**Razor View Engine**

Позволява комбинация между C# и HTML. Позволява предаване на данни между изгледа и контролера. Когато се пише код на C# в HTML-а, пред кода се слага кльомба (**@**).

Идеята зад Razor е да предостави оптимизиран синтаксис за генериране на HTML, използвайки фокусиран върху кода шаблонен подход, с минимален преход между HTML и код. Дизайнът намалява броя на знаците и натисканията на клавиши и позволява по-плавен работен процес на кодиране, като не изисква изрично обозначени сървърни блокове в HTML кода. Други предимства:

* Поддържа IntelliSense – поддръжка за завършване на изрази
* Поддържа „оформления“ – алтернатива на концепцията „главна страница“ в класическите уеб формуляри (.aspx)

**Предаване на параметри**

Данните, които потребителят въвежда в Уеб приложението се предават към сървъра с помощта на параметри. Параметрите могат да бъдат променливи или свойства на даден клас.

1. *Model Binding*

Механизъмът за предаване на данните се нарича Model Binding. Представлява съвкупност от класове, чрез които данните от браузъра се транспортират до уеб сървъра. Всички данни на потребителя се добавят като параметри на конкретен екшън.

Public IActionResult ById (string name1)

ById е мотодът, който още се нарича екшън, а name1 приема параметри. На мястото на name1 може да стои обект на конкретен клас, ако данните са повече от един брой. Параметърът nam1 ще приеме данните, които потребителят въвежда в браузъра. След това тези данни се подават към съответното View.

* Query string
* С форми

Формите по подразбиране изпращат GET заявка и добавя това което е написано в текстовото поле от потребителя, горе в адреса.

Ако заявката е POST данните, които потребителят въвежда, се добавят към Body частта на заявката. Във формата стойността на атрибута name е важен, Защото това е името, което Model Binding търси в заявката за да предаде като параметър на екшъна

* Route values

Стойностите, които се задават, третата наклонена черта се наричат Route стойности

/Article/ById/123

**Какво е Responsive design?**

В най-обща рамка това е очакване уеб сайтът да се скалира, пренарежда според устройството, през което се отваря.

Фирмите виждат, че клиентите им използват лаптопи, PC-та, смартфони и таблети, съответно искат сайтовете им да изглеждат добре. Проблемни в случая са смартфоните и таблетите, заради тъчскрийн екраните си, както и по-малките им дименсии.

Неоспорим факт е, че хората ползват сайтове през най-различни устройства, които имат интернет свързаност. Това налага промяна в изграждането на фирмените сайтове, за да могат същите да отговорят на нуждите на потребителите.

Изработката на адаптивен дизайн за уеб сайт включва множество особености. Нещата не опират само до правилното подреждане на т.нар. лейаут на сайта. Допълнителните точки, които можем да засегнем са следните:

* Оптимизиране на изображенията с цел бързо зареждане на различните устройства.
* Промяна на навигационното меню съобразно ориентацията, в която работи устройството.
* Преработка на бутоните и връзките в сайта, за да са по-удобни за работа с тъчскрийн.
* Осигуряване на адекватно визуализиращи се видео клипове и таблици.
* Оразмеряване на шрифтовете според различните устройства и резолюции.

Изграждането на адаптивен сайт, който се явява добро хоризонтално решение, покривайки повечето устройства, не е лека задача. Не мислете за него като дребна екстра. Той налага кардинално различен подход при всяка изработка на сайт.

Създаването на добър и наистина оптимизиран responsive design отнема време и съответно финансови ресурси. Обещанията за изработка на адаптивен сайт трябва да са подплатени с техническо задание, което да съдържа всичко, което би следвало да представлява. По този начин ще бъдете сигурни какво получавате срещу Вашите пари и гласувано доверие.

Както моят сайт така и повечето сайтове са съвместими с други устройства.

# **Какво трябва да се избягва в уеб дизайна?**

* **Бъркотия:**

Неорганизираните и хаотични сайтове шашкат хората и ги карат да се чувстват глупаво. Когато един потенциален клиент пристигне на интернет страницата Ви, той търси ценна информация. Потенциалният клиент не търси пъзел, който да разреши. Точно поради тази причина се нуждаете от добре изглеждащ, добре работещ и балансиран уеб дизайн. Това се постига чрез правилното използване на пространствата, показване на най-важната информация - тази, която знаете, че клиентите търсят. Предоставянето на тези материали е от огромно значение, защото на база тях клиентът ще вземе своето решение. Дезорганизираните уеб сайтове изглеждат непрофесионално и могат само да навредят върху репутацията на Вашия бизнес. Добрият сайт е като огледало на целите, които са поставени пред Вашата компания.

* **Купища графични елементи:**

Отличният уеб сайт е този, който е създаден от гледна точка на интерактивност и ползваемост. Уеб сайт, който има труден за схващане дизайн или прекалено сложна навигация, няма да успее да извлече дивиденти. Уеб сайтовете с прекалено много графични елементи или тромави анимации могат значително да забавят времето за зареждане. Друг особен момент е използването на шрифтове, които може да изглеждат много интересни, уникални, но в същото време да са трудни за четене, а също така и да не се отварят през някои браузери.

* **Прекалената креативност и шеги:**

Задачата на един бизнес уеб сайт е да представя компанията Ви в интернет и да продава. Сайтът предоставя, по много удобен за потребителите начин, информация за Вашите услуги или стоки. Понякога е лесно да се отнесеш във вихъра на креативността, в опитите да покажеш чувство за хумор. Но помнете, че това не е универсална формула за всеки бизнес. С подобен креативно-комичен подход може да накарате потенциалните клиенти да се замислят относно Вашата сериозност, отговорност и намерения.

Най-важната част от сайта е началната страница. Началната страница на уеб сайта е МЯСТОТО. Точно на него се събират Вашите потенциални клиенти. Това е отправната точка, от където започва всичко и в зависимост от интелигентната му структура - завършва с провал или продажба. Началната страница трябва на мига да предава информация, която да прихване вниманието на потребителя. Трябва да кажете на сърфиращия какво точно предлагате. Това може да стане чрез подходящи изображения, видеа, типография.

**Другите най-важни фактори в един сайт са:**

* Бързото зареждане;
* Сайт печелещ доверието;
* Проста навигация;
* Респонсив дизайн

**Дебъгване**

Дебъгване е процесът на проследяване на изпълнението на дадена компютърна програма с цел намиране и отстраняване на грешки („бъгове“) в нея. Извършва се с помощта на специализирани програмни инструменти наречени дебъгери. Дебъгването е с обхват на сложност, от определяне на прости грешки до извършването на продължителни и отегчителни задачи за събиране на данни, анализи и насрочване на обновления. Уменията на програмиста за дебъгване могат да бъдат основен фактор в способността да се реши един проблем, но сложността на софтуерното дебъгване варира значително със сложността на системата, а също така зависи до известна степен и от използвания езика за програмиране и наличните инструменти като дебъгери. Дебъгерите са софтуерни инструменти които дават възможност на програмиста да следи изпълнението на дадена програма, да го спре, да го рестартира, да заложи точки на прекъсване на програмата (*breakpoints*), както и да променя стойностите в паметта. Терминът дебъгер може да се използва също и за лицето, което отстранява грешките.

В определени случаи основните инструменти които са специфични за езика на които е написан софтуерът могат да бъдат много полезни. Те са представени като статични инструменти за анализ на код. Тези инструменти търсят в рамките на изходния код както много специфични и общи, така и много редки проблеми. Всички засечени проблеми ще бъдат поети от компилатора и интерпретатора които не проверяват синтактично а семантично кода. За някои инструменти се твърди че могат да откриват над 300 уникални по рода си проблема. Съществуват както безплатни така и платени инструменти. Тези инструменти могат да бъдат изключително полезни когато се проверяват огромни по размер проекти. Типичен пример за откриване на проблем би бил код сочещ към променлива която се извиква преди на променливата да е зададена стойност. Друг пример би било да се извършва строга проверка на типа, когато езика не го изисква. По този начин тези инструменти са по-добри в намирането на евентуалните грешки, в превес на действителните грешки.

**Таг хелпъри**

Tag Helpers позволяват на кода от страна на сървъра да участва в създаването и изобразяването на HTML елементи в Razor файлове. Например, вграденият ImageTagHelperможе да добави номер на версия към името на изображението. Всеки път, когато изображението се промени, сървърът генерира нова уникална версия за изображението, така че клиентите са гарантирани да получат текущото изображение (вместо остаряло кеширано изображение). Има много вградени Tag Helpers за обичайни задачи - като създаване на формуляри, връзки, зареждане на активи и други - и още повече налични в публични хранилища на GitHub и като пакети NuGet. Помощниците за етикети са създадени в C# и са насочени към HTML елементи въз основа на име на елемент, име на атрибут или родителски етикет. Например, вграденото LabelTagHelperможе да се насочи към HTML <label>елемента, когато LabelTagHelperатрибутите се прилагат. Ако сте запознати с помощниците за HTML , помощниците за етикети намаляват изричните преходи между HTML и C# в изгледите на Razor. В много случаи HTML Helpers осигуряват алтернативен подход към конкретен Tag Helper, но е важно да се признае, че Tag Helper не заместват HTML Helpers и няма Tag Helper за всеки HTML Helper. Tag Helpers в сравнение с HTML Helpers обяснява разликите по-подробно. Това е в рязък контраст с HTML Helpers, предишния подход за създаване на маркиране от страна на сървъра в изгледите на Razor. Tag Helpers в сравнение с HTML Helpers обяснява разликите по-подробно. Поддръжката на IntelliSense за Tag Helpers обяснява средата IntelliSense. Дори разработчиците с опит със синтаксиса на Razor C# са по-продуктивни, използвайки Tag Helpers, отколкото писането на C# Razor маркиране.

Таг хелърите са:

* Asp-action
* Asp-area
* Asp-controller
* Asp-fragment
* Asp-host
* Asp-page-handler
* Asp-protocol
* Asp-route
* Asp-all-route-data

**Dependency injection**

Позволява в контролерите да се инжектира различни методи. Може да се използва дори във View-то.

Service-метод, чрез който искаме да получим конкретен резултат.

Стъпки:

1. Създаване на папка services в проекта
2. Създаване на интерфейс
3. Създаване на имплементация на този интерфейс
4. Регистриране на Service
5. Използване на Service
   1. Добавяне на поле ;
   2. Създаване на конструктор и задаване на параметър;
   3. Ново свойство в модела;
   4. Задаване на стойност на свойството;
   5. Във View-то се извиква свойството за резултат.

**Най–важни изплозвани елементи в проекта:**

* <form> - Формите са използвани за записване на данните в базата. Данните се изпращат там.
* <input> - Полетата за въвеждане на данни.
* <button> - за записване на данните и за изпращане в базата.
* DbSet<> - за описване на таблиците.
* db.SaveChanges() – за записване на промените в базата.

**Приноси на дипломния проект**

* Намиране на работа – добавяне на нови обяви и създаване на заявка
* Предобиване на умения чрез опити
* Запознаване с други хора – комуникация с продавача

**Задачи за извършване в проекта:**

* Публикуване на нови обяви, предоствяне на услуги, търсене на доставчик
* Регистриране и предлагане на инфомация, цена, срокове за изпълнение, начин на доставка.
* Спиране, стартиране и изтриване на офертата
* Комуникация
* Статус – преговаряне, стартиране на услугата, приключване, feedback, спор
* Рейтинг