

МИНИСТЕРСТВО НА ОБРАЗОВАНИЕТО И НАУКАТА

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

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

**ДИПЛОМЕН**

**ПРОЕКТ**

**Тема: Разработка на Уеб сайт за популяризиране и изяви на любима рокгрупа**

*Ученик:* Самуил Емануилов Костадинов ученик от XII Д клас

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

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

***Консултант:***

Сопот, 2022 г.

**Съдържание**

**Стр.3-Увод**

**Стр.4-Цели и задачи**

**Стр.5- 1.1. Предпоставка за създаване на продукта**

**Увод**

В последното десетилетие се наблюдава нарастване на обвързаността между бизнеса и новите технологии. Организациите се стремят да се разрастват, осигурявайки гъвкав начин на работа, базиран на висока мобилност, и неограничен от времето и мястото достъп до информация.

Всеки потребител, който търси нещо определено в мрежата се стреми да получи достатъчно подробна информация, снимки, видео за продукта или услугата, която му е необходима. Ето защо, една професионално изградена и добре оптимизирана интернет страница, с уникален и привлекателен дизайн дава възможност да представите себе си, дейностите и услугите които извършвате, всички продукти, които произвеждате и предлагате на пазара. По този начин, с течение на времето все повече хора ще разбират за вашия бизнес, което ще Ви направи по-търсени и по-популярни и ще повиши значително рейтинга Ви. Фирмите, разполагащи с уеб сайт се ползват с по-висок кредит на доверие сред своите клиенти. С помощта на сайта, ще имате възможност да осведомявате своевременно клиентите си за всеки нов детайл, дейност или за всеки нов продукт, който смятате да предложите или вече сте предложили на пазара, при това сайтът ще работи за Вас 24 часа в денонощието, без почивен ден. Интернет страницата ще бъде една постоянна реклама за Вас и Вашия бизнес, която наистина не е за изпускане.

**Цели и задачи**

Целта на сайта е да популяризира и менажира изявите на рок група, състав или музикант.

Сайта дава информация са минали или предстоящи концерти, изяви, участия или награди на групата. Тъй като технологиите напредват вече всеки има достъп до интернет, където могат да намерят и нашия сайт.

**ГЛАВА 1.**

**ПРОУЧВАНЕ**

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

Понякога групата се нуждае от сайт, който да ви помогне да разберете какво правят.

В сайта има новини са минали или предстоящи концерти, албуми и случки с членовете на групата. Сайта има за цел да рекламира групата на по-голям мащаб, а не само чрез социалните мрежи, въпреки че социалните мрежи в момента са един от най-добрия начин да се прави реклама на всякакви продукти, групи, организации и т.н.

Много от тези страници в социалните имат оставена таблица с контакти и в тази таблица повечето пъти може да видите оставен линк към техния личен сайт, в който може да намерите информация за групата, стока от сорта на тениски, гривни, суитчъри, плакати рекламиращи групата.

* 1. **Съществуващи решения и реализации**

Мога да дам за пример групата Arctic Monkeys които също имат свой сайт в който е предоставена информация за групата, имат отделна страница в която продават тениски, шапки и също така и винилови плочи със златните им албуми. В сайта им също можем да намерим и техните социални медии като Instagram, Facebook, Tweeter както и канала им в YouTube. Също така са предоставили и най-актуалните новини за групата, като е дадена информация за членовете на групата и най-скорошните им изяви или участия в телевизионни шоута. В сайта им също можем да открием и техните песни и видеа които са подбрани така че да грабнат потребителя искащ да узнае нещо повече за групата и техния стил на музика.

**ГЛАВА 2.**

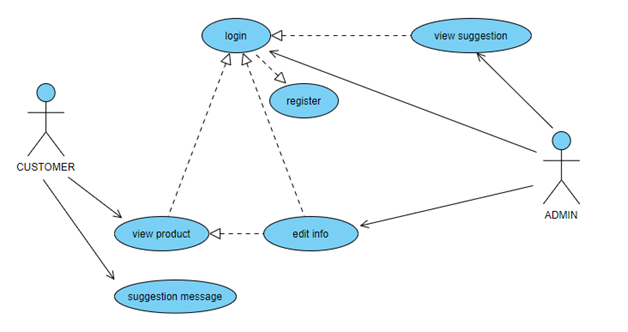
**ПРОЕКТИРАНЕ НА СТРУКТУРАТА НА WEB БАЗИРАН….**

**2.1. Функционални изисквания към WEB базиран......... /**условието назадачата**/**

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

Онлайн системата предоставя среда за извършване на онлайн търговия/онлайн услуга, изразяваща се в създаване на условия за достъп до налични артикули за клиента / потребителя и комуникация с представител на бизнеса. Като такава системата работи, използвайки технологии основани на Интернет свързаност(++++). Сайта е софтуер, достъпът до който се осъществява през интернет при използване на уеб-браузер или мобилно приложение, и чрез който се извършва продажба на стоки/услуги като се предоставя възможност за избор от клиента на стоки/услуги чрез потребителска кошница, както и за предоставяне на информация за контакт с купувача, дата на поръчка. //сравнение на нашия продукт със съществуващи подобни продукти и стандарти//

Бизнес моделът се основава на концепцията клиента да закупува продукти в системата, като се регистрира, влиза, разглежда и избира.



**2.3. Потребителски интерфейс /**менюта, екрани, Views**/**

## Какво всъщност е графичния потребителски интерфейс (GUI)

Този интерфейс е интерфейс, чрез който потребителят взаимодейства с електронни устройства. GUI интерфейс използва икони, менюта и други изображения на визуални индикатори за показване на информация и свързаните с тях потребителски контроли, за разлика от текстово базирани интерфейси, където данните и команди са в текст. GUI представителства се манипулират от насочващо устройство като мишка или пръст върху сензорен екран.

**Преглед на GUI**

GUI включва GUI обекти като икони, курсори и бутони. Тези графични елементи понякога се подобряват със звуци или визуални ефекти като прозрачност и падащи сенки. Използвайки тези обекти, потребителят може да използва компютъра, без да трябва да знае команди.

По-долу е снимка на работния плот на Windows 7 и пример за GUI операционна система. В този пример можете да използвате мишката, за да преместите показалец и да щракнете върху икона на програма, за да стартирате програма.



**Какви са елементите на GUI?**

За да направите GUI възможно най-удобен за потребителя, има различни елементи и обекти, които потребителят използва за взаимодействие със софтуера. По-долу е даден списък на всеки от тях с кратко описание.

Бутон – Графично представяне на бутон, който извършва действие в програма при натискане

Диалогов прозорец – Тип прозорец, който показва допълнителна информация и пита потребител за въвеждане.

Икона – Малко графично представяне на програма, функция или файл.

Меню - Списък с команди или възможности за избор, предлагани на потребителя чрез лентата с менюта.

Лента с менюта - Тънка хоризонтална лента, съдържаща етикетите на менютата.

Лента - Замяна на менюто с файлове и лентата с инструменти, която групира програмите дейности заедно.

Раздел – област, върху която може да се кликва, в горната част на прозорец, който показва друга страница или област.

Лента с инструменти – Ред от бутони, често в горната част на прозореца на приложението, който контролира софтуерните функции.

Прозорец - Правоъгълна част от дисплея на компютъра, която показва програмата, която се използва в момента.

**Как работи GUI?**

GUI използва прозорци, икони и менюта за изпълнение на команди, като отваряне, изтриване и преместване на файлове. Въпреки че операционната система с GUI се навигира основно с помощта на мишка, клавиатурата може да се използва и чрез клавишни комбинации или клавишите със стрелки.

Като пример, ако искате да отворите програма в GUI система, трябва да преместите показалеца на мишката върху иконата на програмата и да щракнете двукратно върху нея. С интерфейс на командния ред ще трябва да знаете командите, за да отидете до директорията, съдържаща програмата, да изброите файловете и след това да стартирате файла.

Какви са предимствата на GUI?

GUI се счита за по-удобен за потребителя от текстов интерфейс на командния ред, като MS-DOS, или обвивката на Unix-подобни операционни системи.

За разлика от операционната система от команден ред или CUI, като Unix или MS-DOS, GUI операционните системи са по-лесни за научаване и използване, тъй като командите не трябва да се запомнят. Освен това потребителите не трябва да знаят езици за програмиране. Поради тяхната лекота на използване и по-модерен външен вид, GUI операционните системи доминират на днешния пазар.

Какви са примерите за GUI операционна система?

* Microsoft Windows
* Apple System 7 и macOS
* Chrome OS
* Варианти на Linux като Ubuntu, използващи GUI интерфейс.
* Какви са примерите за GUI интерфейс?
* Apple macOS
* Microsoft Windows
* GNOME
* KDE
* Всяка програма на Microsoft, включително Word, Excel и Outlook.
* Интернет браузъри, като Internet Explorer, Chrome и Firefox.

**Как потребителят взаимодейства с GUI?**

Посочващо устройство, като мишката, се използва за взаимодействие с почти всички аспекти на GUI. По-модерните (и мобилни) устройства също използват сензорен екран.

**GUI изисква ли мишка?**

Не. Почти всички GUI интерфейси, включително Microsoft Windows, имат опции за навигация в интерфейса с клавиатура, ако знаете клавишните комбинации.

**2.4. Описание на избраната технология и софтуерните средства за разработка на приложението**

**ASP.NET** е рамка за уеб приложения от страна на сървъра с отворен код, предназначена за уеб разработка за създаване и реализиране на динамични уеб страници. Тя е разработенa от Microsoft, за да позволи на програмистите да създават динамични уеб сайтове, приложения и услуги.

Какво е ASP.NET?

ASP.NET е уеб рамка с отворен код за изграждане на уеб приложения върху .NET (dotNET) рамка. Той е създаден от Microsoft и версия 1.0 е пусната през 2002 г., за да позволи на разработчиците да създават динамични уеб приложения, услуги и сайтове. Рамката е изградена да работи със стандартния HTTP протокол, който е стандартният протокол, използван във всички уеб приложения.

ASP.NET е наследник на технологията ASP (Active Server Pages) и беше значително подобрение по отношение на гъвкавостта и мощността. Това е разширение на платформата .NET с допълнителни инструменти и библиотеки, специално за изграждане на неща в мрежата, включително уеб приложения и уебсайтове.

Най-новата версия на ASP.NET е кросплатформената версия, наречена ASP.NET Core, която беше пусната през 2016 г. ASP.NET все още се поддържа и актуализира, но напредъкът фокусът за Microsoft е да разработи новата кросплатформена версия .

**Как работи ASP.NET архитектурата?**

Архитектурата на ASP .NET рамката се състои от набор от ключови компоненти, които се използват при разработката.

**1. Език**

Езикът за програмиране, използван за рамката на ASP.NET, е VB.NET (Visual Basic) или C#. Umbraco CMS е написан на C#.

**2. Библиотека**

ASP.NET включва базовите библиотеки от .NET рамката, но също така включва библиотеки за общи уеб модели. Една такава библиотека е Model View Controller (MVC), който ви позволява да използвате модела за проектиране на MVC за проектиране на уеб приложения и сайтове. MVC моделът ви позволява да изградите уеб приложение като композиция от три роли - бизнес слой, дисплейен слой и контрол на въвеждане.

**3. Време за изпълнение на общия език**

Common Language Runtime (CLR) е платформата, където се изпълняват вашите .NET програми. Използва се за извършване на ключови дейности като активиране на обекти, извършване на проверки за сигурност, поставянето им в паметта, изпълнение и обработка на събирането на боклука.

Освен трите ключови компонента в рамката, той също така разширява .NET с други инструменти, за да улесни живота на уеб разработчика. Един от тях е синтаксисът на шаблоните, използван за уебсайтове, наречени Razor. Този синтаксис на шаблони се използва силно при изграждането на динамични уеб страници в C# и позволява работа с главни страници и шаблони за създаване на уеб страници с динамични стойности и съдържание.

При работа с .NET е възможно да се използва Microsoft Visual Studio като интегрирана среда за разработка (IDE). Visual Studio е IDE от Microsoft, използвана за разработване на уеб приложения и се състои от редактор на изходен код, дебъгер, компилатор и дизайнер, които са достъпни чрез един-единствен интерфейс.

**C#** е език за професионално програмиране. C# е език за програмиране, предназначен за изграждане на широк спектър от корпоративни приложения, които работят на. NET

Framework. Целта на C# е да предостави прост, безопасен, модерен, обектно-ориентиран, високопроизводителен, здрав и издръжлив език за програмиране.

C# е модерен, обектно-ориентиран и безопасен за типа език за програмиране. C# позволява на разработчиците да създават много видове сигурни и стабилни приложения, които работят в .NET. C# има своите корени в семейството на езиците C и веднага ще бъде познат на програмистите на C, C++, Java и JavaScript. Тази обиколка предоставя преглед на основните компоненти на езика в C# 8 и по-ранни версии. Ако искате да изследвате езика чрез интерактивни примери, опитайте въведението в C# уроци.

C# е обектно-ориентиран, компонентно-ориентиран език за програмиране. C# предоставя езикови конструкции за пряка поддръжка на тези концепции, което прави C# естествен език, на който се създават и използват софтуерни компоненти. От своя произход C# е добавил функции за поддръжка на нови работни натоварвания и нововъзникващи практики за проектиране на софтуер. В основата си C# е обектно-ориентиран език. Вие определяте типове и тяхното поведение.

Няколко функции на C# помагат за създаването на здрави и издръжливи приложения. Събирането на боклука автоматично възстановява паметта, заета от недостижими неизползвани обекти. Типовете с нулеви стойности предпазват от променливи, които не се отнасят към разпределени обекти. Обработката на изключения осигурява структуриран и разширяем подход за откриване и възстановяване на грешки. Ламбда изразите поддържат техники за функционално програмиране. Синтаксисът на езиково интегрирана заявка (LINQ) създава общ модел за работа с данни от всеки източник. Езиковата поддръжка за асинхронни операции осигурява синтаксис за изграждане на разпределени системи. C# има унифицирана система от типове. Всички типове C#, включително примитивни типове, като int и double, наследяват от един тип основен обект. Всички типове споделят набор от общи операции. Стойности от всякакъв вид могат да се съхраняват, транспортират и оперират по последователен начин. Освен това C# поддържа както потребителски дефинирани референтни типове, така и типове стойности. C# позволява динамично разпределение на обекти и in-line съхранение на олекотени структури. C# поддържа общи методи и типове, които осигуряват повишена безопасност и производителност на типа. C# предоставя итератори, които позволяват на имплементаторите на колекционни класове да дефинират персонализирано поведение за клиентски код.

C# набляга на версиите, за да гарантира, че програмите и библиотеките могат да се развиват с течение на времето по съвместим начин. Аспектите на дизайна на C#, които бяха пряко повлияни от съображенията за управление на версиите, включват отделните виртуални и отменящи модификатори, правилата за разрешаване на претоварване на метода и поддръжка за изрични декларации на членове на интерфейса.

**HTML** (HyperText Markup Language) е най-основният градивен елемент на уеб мрежата. Той определя значението и структурата на уеб съдържанието. Други технологии освен HTML които обикновено се използват за описване на външния вид/презентацията са (CSS) или функционалността/поведението на уеб страницата (JavaScript).

HTML, или HyperText Markup Language, позволява на уеб потребителите да създават и структурират секции, абзаци и връзки, използвайки елементи, тагове и атрибути. Въпреки това, заслужава да се отбележи, че HTML не се счита за език за програмиране, тъй като не може да създава динамична функционалност.

HTML има много случаи на употреба, а именно:

* Уеб разработка. Разработчиците използват HTML код, за да проектират как браузърът показва елементи на уеб страница, като текст, хипервръзки и медийни файлове.
* Интернет навигация. Потребителите могат лесно да навигират и да вмъкват връзки между свързани страници и уебсайтове, тъй като HTML се използва силно за вграждане на хипервръзки.
* Уеб документация. HTML дава възможност за организиране и форматиране на документи, подобно на Microsoft Word.

Също така си струва да се отбележи, че HTML вече се счита за официален уеб стандарт. World Wide Web Consortium (W3C) поддържа и разработва HTML спецификации, заедно с предоставяне на редовни актуализации.

Средно един уебсайт включва няколко различни HTML страници. Например, начална страница, страница за информация и страница за контакт ще имат отделни HTML файлове.

HTML документите са файлове, които завършват с разширение .html или .htm. Уеб браузърът чете HTML файла и изобразява съдържанието му, така че интернет потребителите да могат да го видят.

Всички HTML страници имат поредица от HTML елементи, състоящи се от набор от тагове и атрибути. HTML елементите са градивните елементи на уеб страница. Тагът казва на уеб браузъра къде започва и завършва даден елемент, докато атрибутът описва характеристиките на елемент.

Трите основни части на елемента са:

* Отварящ маркер – използва се, за да посочи къде даден елемент започва да влиза в сила. Етикетът е обвит с отварящи и затварящи ъглови скоби. Например, използвайте началния маркер <p>, за да създадете абзац.
* Съдържание – това е изходът, който виждат другите потребители.
* Затварящ маркер – същият като началния таг, но с наклонена черта пред името на елемента. Например </p> за край на абзац.

Друг атрибут, HTML класът, е най-важен за разработката и програмирането. Атрибутът клас добавя информация за стил, която може да работи върху различни елементи със същата стойност на класа.

Затварящ етикет – същият като отварящия таг, но с наклонена черта пред името на елемента. Например </p> за край на абзац.

**Плюсове и минуси на HTML**

Както всеки друг компютърен език, HTML има своите силни страни и ограничения. Ето плюсовете и минусите на HTML:

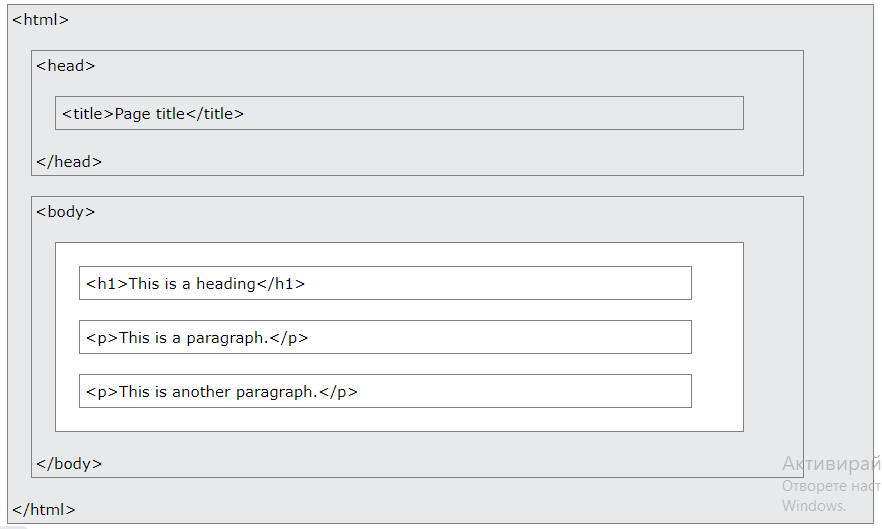
**Плюсове:**

* Подходящо за начинаещи. HTML има чисто и последователно маркиране, както и плитка крива на обучение.
* Поддържка. Езикът е широко използван, с много ресурси и голяма общност зад него.
* Достъпен. Той е с отворен код и напълно безплатен. HTML работи естествено във всички уеб браузъри.
* Гъвкав. HTML е лесно интегрируем с бекенд езици като PHP и Node.js.

**Минуси:**

* Статичен. Езикът се използва предимно за статични уеб страници. За динамична функционалност може да се наложи да използвате JavaScript или бек-енд език като PHP.
* Отделна HTML страница. Потребителите трябва да създават отделни уеб страници за HTML, дори ако елементите са едни и същи.
* Съвместимост с браузъри. Някои браузъри приемат новите функции бавно. Понякога по-старите браузъри не винаги изобразяват по-нови тагове.

**Структура на HTML страница**

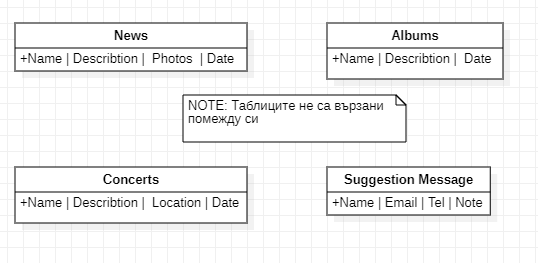


**ГЛАВА 3.**

**ПРОГРАМНА РЕАЛИЗАЦИЯ НА WEB БАЗИРАН ……………….**

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

Тъй като страниците на изготвения от мен сайт не са вързани една за друга, затова и в структурата на БД не са вързани помежду си.



* 1. **Структура на MVC приложението – Controllers Class Diagram**
* Описание и конфигуриране на моделите за работа и обмен на данни между контролери и изгледи

MVC(Model-View-Controller) е софтуерен модел, първоначално разработен през 70-те години на миналия век, когато графичните потребителски интерфейси са били в много ранните фази на разработка. MVC се стреми да раздели приложението на три области:

Моделът

Тук се съхраняват данни и бизнес логика. В типично уеб приложение тази

част от приложението би представлявала таблица на база данни

или всеки друг обект, с който приложението трябва да работи.

Гледката

Гледката е действителното визуално представяне на модела изработен от вас. В типично уеб приложение това би била страницата, която показва вашия модел на

потребителя, независимо дали е форма за въвеждане на данните или изход на

данните или комбинация от двете. Очевидно изгледът не трябва да показва

всички части на модела, а моделът може да има повече от един изглед.

**Контролерът**

Контролерът трябва да обработва комуникацията между вашия модел и

вашия изглед. В уеб приложенията методите на контролера се

извикват от потребителя при отваряне на страница или щракване върху

бутон. След това контролерът актуализира модела, ако е необходимо, и след

това връща нов изглед, ако е

необходимо.

**Какво е контролер?**

Контролерът действа като посредник - той ще комбинира вашия модел с изглед и ще предостави резултата на крайния потребител. Въпреки това, не се изисква нито модел,

нито изглед - контролерът може да действа самостоятелно за най-основните операции, напр. доставяне на просто текстово съобщение или пренасочване на потребителя към някъде другаде.

В ASP.NET MVC контролерът е точно като всеки друг клас, така че има

разширение на .cs файл и изглежда като всеки друг .NET клас. Има обаче

няколко неща, които го правят разпознаваем от .NET рамката като MVC

контролер:

* Обикновено се поставя в папка, наречена “Controllers” в корена на

вашия проект

* Той наследява от Microsoft.AspNetCore.Mvc.Controller
* Името на класа обикновено завършва с думата Controller,

например “HomeController&quot; или “ProductsController”

Ако не спазвате тези конвенции, .NET рамката няма да може да разпознае

вашия клас като контролер, така че има смисъл да ги следвате. Въпреки това**,**

ако настоявате например да наименувате класовете си Controller по различен

начин, можете да го украсите с атрибута [Controller] , поставен точно преди

декларацията на класа.

Като наследите класа Microsoft.AspNetCore.Mvc.Controller , вие получавате

допълнителна функционалност, която можете да използвате за целите на

MVC, например възможността за връщане на Views/Partial Views. Той също

така позволява на вашия клас Controller да има достъп до информация,

свързана с HTTP, като низа на заявката, благодарение на

свойството HttpContext на класа Controller. С други думи, той превръща

обикновения .NET клас в клас за уеб, което ви позволява да правите неща,

които искате да правите в мрежова комуникация.

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

Kонтролерът е просто обикновен .NET клас, той може да има полета,

свойства и методи. Особено интересни са методите на клас Controller, тъй като те

са връзката между браузъра и вашето приложение. Поради тази причина методите на клас Controller се наричат:

**Действия** - метод обикновено съответства на действие във вашето приложение,

което след това връща нещо на браузъра/потребителя.

Тъй като браузърът работи чрез извикване на URL, имате нужда от нещо, което

превежда URL адресите към съответен контролер и действие (метод). Например

браузърът може да поиска URL като /products/1/ и след това искате

вашият ProductsController да обработва тази заявка с метод/действие,

наречено Details . Това се прави с концепцията за маршрутизирането.

/маршрутизирането е това, което свързва URL адресите с действията на

вашите контролери/.

Когато създавате своите контролери, имайте предвид, че всички публични

методи в клас Controller се считат за действие . Това означава, че ако сте

дефинирали всеобхватни правила за маршрутизиране за вашия контролер (а това

е често срещано нещо), всички методи във вашия клас Controller на теория могат

да бъдат достигнати с помощта на URL. Така че, ако имате методи на вашия

контролер, които не искате крайният потребител да може да извиква, не

забравяйте да го маркирате като private . Като алтернатива, ако наистина имате

нужда от метод да бъде публичен, но да не е достъпен чрез URL, можете да

маркирате метода с атрибута [NonAction] .

**Глаголи за действие като имена**

За да получите повече контрол върху това как се наричат вашите действия,

можете да ги украсите с така наречените глаголи за действие. Това всъщност са

обикновени .NET атрибути, които ще кажат на .NET рамката как може да се

осъществи достъп до действие. Без тези атрибути може да се осъществи достъп

до действие чрез всички възможни HTTP методи (най-често срещаните са GET и

POST), но можете да промените това доста лесно:

[HttpGet]

public IActionResult Edit()

{

return Content("Edit");

}

Сега действието Edit може да бъде достъпно само с GET заявка. Това има

допълнителното предимство, че ви позволява да имате няколко метода с едно и

също име, стига да не приемат един и същ метод на заявка. Така например

можете да имате два метода, наречени Edit: първият ще бъде наличен за GET

заявки и ще генерира формуляр за редактиране на елемент, докато вторият ще

бъде достъпен само за POST заявки и ще се използва за актуализиране на

елемента, когато формуляр беше изпратен обратно на сървъра. Може да

изглежда така:

[HttpGet]

public IActionResult Edit()

{

return View();

}

[HttpPost]

public IActionResult Edit(Product product)

{

product.Save();

return Content("Product updated!");

}

Сега, когато се направи заявка към метода/действието Edit(), действителният

метод, отговарящ на заявката, ще се основава на това дали е GET или POST

заявка.

В някои ситуации може да искате да посочите множество глаголи за действие,

например за да посочите, че едно действие може да бъде достъпно както от

POST, така и от GET заявки, но не и от други типове. Това е също толкова лесно:

[HttpGet]

[HttpPost]

public IActionResult Edit()

{

.....

Интересно е как два различни action-метода се обръщат към един и същ

изглед /View/ в зависимост от това каква HTTP заявка ги стартира.

1. От RouteMap =>URI: Products/Create => извикване на следния метод

// GET: Products/Create

public IActionResult Create()

{

return View();

}

2. return View(); извиква съответния изглед Create.cshtml /със същото име

като името на action-метода/.

<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="Price" class="control-label"></label>

<input asp-for="Price" class="form-control"/>

<span asp-validation-for="Price" 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">

<input type="submit" value="Create" class="btn btn-primary"/>

</div>

</form>

3. При натискане на бутона от реда

<input type=”submit“ value=”Create” class”btn btn-primary” /> се предизвиква

действието, описано в отварящия таг на формуляра

<form asp-action=”Create”> който събира данните със съответните имена

като параметри и генерира нова HTTP заявка с указания метод /get или

post/. Ако HTTP метод не е указан, се счита за POST.

4. Отново сме в същия контролер и се е стартирал втория метод

Create( <params> ), който приема всички параметри от заявката и има

атрибут /анотация/ [HttpPost]

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult>Create([Bind(”Id,Name,Price,Description”)]

Product product)

{

if (ModelState.IsValid)

{

\_context.Add(product);

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

return View(product);

}

Този метод прави реалното записване в БД и ме препраща в изгледа със

списъка на продукти, от таблица Products.

* **Работа с View моделите (където е необходимо). Разлика с използванет на ViewBag и ViewData ( описание на двата обекта)**

View(Изглед)

### Докато контролерът обработва връзката между bаck-end и front-end, изгледът **е** визуалният резултат от действие на контролера. Така че, когато искате да представите на потребителя нещо визуално, а това обикновено е, което искате, когато разработвате уеб приложение, то се поставя в **View** . Изгледът съдържа маркиране (HTML) и Razor код и често ще бъде визуално представяне на вашия модел. С други думи, контролерът генерира обект на модела и след това го предава на изгледа, който след това използва модела, за да представи визуално съдържанието на модела на потребителя. Изгледът е основно просто HTML файл с добавена поддръжка за Razor код. Той използва разширението .cshtml , за да посочи тези допълнителни възможности.

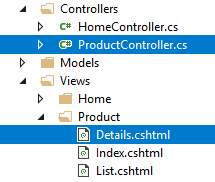
**Къде се поставят Views?**

Прегледните файлове обикновено се поставят в папка, наречена **Views** , в корена на вашия MVC проект. За да улесните .NET рамката и вашите контролери да намерят правилния изглед, обикновено създавате подпапка във вашата папка Views за всеки от вашите контролери, носеща името на контролера. Така че, ако имате HomeController и ProductController, вашата папка Views може да има подпапки с имената "Home" и "Product". Всяка от тези папки ще има един или няколко изгледа, свързани с действията на вашите контролери. Може да изглежда нещо подобно в Solution Explorer:

Когато следвате тези конвенции, вие позволявате на .NET рамката да намира вашите изгледи автоматично, чрез процес, наречен View Discovery. Ще обсъдим този процес в една от следващите статии.

контролерът може да върне изглед, просто като извика метода View(). Това дори работи без параметри, така че в този момент може да се чудите как в света контролер знае точно кои от вашите изгледи да върне за конкретно действие. Отговорът е нещо, наречено **View Discovery** – процес, при който ASP.NET MVC ще се опита да отгатне кой изглед да използва, без да ви принуждава да го посочите.

View Discovery работи, когато следвате конвенция, когато създавате структурата на проектa. Винаги поставяме контролери в папка, наречена "Controllers", модели в папка, наречена "Models" и така нататък. С Views трябва да ги поставите в подпапка, наречена на контролера, а името на файла на View трябва да съвпада с името на действието. Така например, ако имате контролер, наречен "ProductController", с действие, наречено "Подробности", структурата на вашата папка трябва да изглежда така:



С други думи, местоположението и именуването на вашите изгледи трябва да следват тази конвенция:

/Views/[NameController]/[Action\_Name].cshtml

Ако е така, можете просто да извикате метода View() от действията на вашия контролер и да накарате .NET рамката автоматично да намери правилния изглед за вас:

public class ProductController : Controller  
{  
 public IActionResult Index()  
 {  
     return View();  
 }  
  
 public IActionResult Details()  
 {  
 return View();  
 }  
}

Ако рамката не може да намери съвпадащ изглед с помощта на споменатата по-горе конвенция, тя ще търси на още едно място: /Views/Shared/[Action Name].cshtml

Shared папка, която можете да добавите във вашата папка Views, обикновено се използва за файлове с Layout и Partial изгледи, споделени в множество контролери. Ще обсъдим това по-късно, но засега най-лесното нещо е да следвате конвенцията за структурата на папките, описана по-горе.

Посочване на изглед

Ако не можете (или не искате) да следвате конвенцията, можете да помогнете на рамката да намери съответстващия изглед, като предоставите параметър на метода View(). Има два начина: Можете или да посочите само името, което все още изисква от вас да следвате конвенцията за структура на папка, но да предоставите изглед с друго име (мислете за него като псевдоним) по следния начин:

public IActionResult Test()  
{  
 return View("Details");  
}

Или можете да предоставите целия път, което ви дава пълна гъвкавост при създаването на структурата на вашия проект:

public IActionResult Test()  
{  
 return View("/ViewFolderName/SomeFolderName/ViewName.cshtml");  
}

**Изгледът** може да съществува без **модел** или други данни от бекенда, но в повечето случаи трябва по някакъв начин да получите достъп до данните, предоставени от контролер във вашия изглед. Тъй като вашият изглед не трябва да знае за вашия контролер (съгласно правилото за разделяне на притесненията), вашият контролер е отговорен за предоставянето на данните на изгледа. Има два основни начина да направите това: Или дефинирате строго въведен модел и след това го предадете на изгледа, или можете да използвате контейнерите ViewData/ViewBag, за да направите данните достъпни за изгледа. Нека сега да обсъдим тези подходи.

**Използване на View модел**

Тъй като тази технология се нарича MVC, съкратено от Model-View-Controller, използването на (View) модел определено е най-чистият и най-често срещаният подход за предаване на данни в View. За модела знайте, че това е просто клас, обикновено представляващ данни от един или повече класове, които ще бъдат показани/презентирани при потребителя. Ще се сблъскате и с концепцията за „ViewModel“ – отново ще се задълбочим в това по-късно, но за да го опишем накратко, моделът на изглед е просто модел, предаван на изглед. Така че може да бъде нещо, което вече съществува във вашия проект, като обект на  UserилиProoduct , или клас, който специално дефинирате, за да представлява данните, до които искате да получите достъп от вашия изглед.

Моделът обаче не трябва да е сложен клас - на теория може да бъде нещо толкова просто като String, Integer, Enum или DateTime. С други думи, всичко, което се намира в .NET рамката, може да се използва като модел, но на практика вашият модел обикновено ще бъде клас, състоящ се от поне няколко свойства. Така например, модел, използван за представяне на продукт, може да изглежда така:

public class Product  
{  
 public string Title { get; set; }  
  
 public double Price { get; set; }  
}

Връзката между модела и изгледа се осъществява от контролера - той може да вземе вход от потребителя и да го превърне в подходящи данни за изгледа, например така:

public IActionResult Details(int id)  
{  
 Product product = new Product()  
 {  
 Title = "Toilet Paper",  
 Price = 1.99  
 };  
 return View(product);  
}

Забележете как е показан екземпляра на Productкъм View, когато извикамe метода View(). Вътре в изгледа на продукта/подробности сега мога да дефинирам класа на продукта като модела, който този изглед може (и трябва) да очаква, като използвам директивата **@model** , разположена в горната част на вашия изглед файл:

@model HelloMVCWorld.Models.Product

С това вече можете да започнете да използвате своя модел (View) и неговите свойства във вашия изглед:

@model HelloMVCWorld.Models.Product  
  
<h1>@Model.Title</h1>  
Price: @Model.Price

Благодарение на факта, че посочвате силен тип като модел на изгледа, вие ще получите помощ за достъп до членове на обекта чрез IntelliSense и вашият код ще бъде валидиран по време на процеса на изграждане, за да сте сигурни, че използвате само намерени свойства/методи върху обекта. За целта използваме обекта Model след @ в таговете на View. Model се счита за динамичен обект, където можете да получите достъп до свойства и методи на модела, обявен с клаузата @model в началото на всяко View.

## **Използване на контейнерите ViewData/ViewBag**

Като алтернатива на строго въведения подход за предаване на данни към View, можете да използвате така наречените контейнери ViewData и ViewBag. Можете да добавяте неща към тях от контролера и след това автоматично да имате достъп до съхранените данни във вашите изгледи. Всъщност е доста лесно и можете да постигнете почти същите неща, както ако сте използвали силно въведен модел. Ето един пример:

public IActionResult DetailsViewData(int id)  
{  
 ViewBag.ProductTitle = "Toilet Paper";  
 ViewBag.ProductPrice = 1.99;  
 return View();  
}

Вече имате достъп до тези данни във вашия изглед също толкова лесно, благодарение на винаги наличното свойство, наречено ViewBag:

<h1>@ViewBag.ProducTtitle</h1>  
Price: @ViewBag.ProductPrice

Основната разлика тук е, че няма проверка по време на компилиране на тези свойства и няма помощ от IntelliSense при записването им.

можете лесно да напишете грешно име на свойство и няма да забележите, докато не опитате да използвате View. Следователно трябва да използвате ViewData/ViewBag само за много малки количества данни. Трябва обаче да запомните, че не е **препоръчително да използвате ViewData/ViewBag вместо модели за данни.**

**Каква е разликата между ViewData и ViewBag?**

Ще сте забелязали, че използвах два термина за работа с данни за динамичен изглед: свойствата ViewData и ViewBag. И двете всъщност се отнасят до едно и също нещо, което е речник на ключове и стойности (обектите). Въпреки това ViewBag е представяне на DynamicObject на речника, което ви позволява да получите достъп до данните, сякаш са свойства на обекта ViewBag, вместо записи в речник. Вие сте свободни да използвате и двете свойства взаимозаменяемо, тъй като те винаги се отнасят до едни и същи данни, така че за сравнение, ето пример за това как би изглеждало, ако използвате и двата подхода:

**Controller:**

public IActionResult DetailsViewData(int id)  
{  
 ViewData["ProductTitle"] = "Toilet Paper";  
 ViewBag.ProductPrice = 1.99;  
 return View();  
}

**View:**

<h1>@ViewData["ProducTtitle"]</h1>  
Price: @ViewBag.ProductPrice

Забележете как се използва ViewData и следователно подход, базиран на речник за четене и писане на ProductTitle, докато използвам обектно-базирана точка-нотация за достъп до ProductPrice. Това ви оставя с няколко по-малки разлики, например възможността да проверите наличието на даден ключ/стойност в колекцията ViewData.

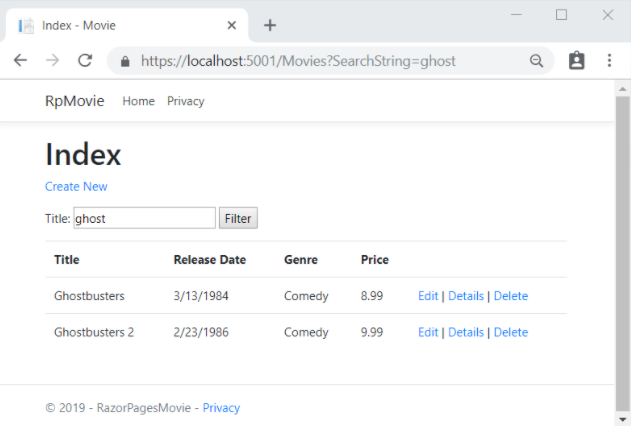
## Резюме

Можете да предавате данни от контролер към изглед, като използвате няколко различни подхода, но препоръчителният почти винаги е строго въведеният изглед, където определяте типа модел, който вашият изглед може да очаква. Това ви дава поддръжка на IntelliSense и проверка по време на компилиране на свойствата и методите, които се опитвате да използвате. Отразява се и на сигурността при работа с данните на вашия сайт.

* **Описание на Razor Pages и Razor Views (TagHelpers и HtmlHelpers)**

Razor Pages е по-нов, опростен модел за програмиране на уеб приложения. Той премахва голяма част от церемонията на ASP.NET MVC, като приема подход за маршрутизиране, базиран на файлове. Всеки файл Razor Pages, намерен в директорията Pages, се равнява на крайна точка.

Ето така изглежда една Razor Page страница:



Razor Page страницата се осъществява в Program.cs със подобен код:

var builder = WebApplication.CreateBuilder(args);

builder.Services.AddRazorPages();

var app = builder.Build();

if (!app.Environment.IsDevelopment())

{

app.UseExceptionHandler("/Error");

app.UseHsts();

}

app.UseHttpsRedirection();

app.UseStaticFiles();

app.UseRouting();

app.UseAuthorization();

app.MapRazorPages();

app.Run();

В показания код AddRazorPages добавя услуги за Razor Pages към приложението. A

MapRazorPages добавя крайни точки за Razor Pages към IEndpointRouteBuilder.

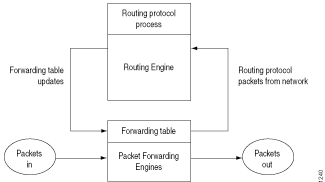
* 1. **Използване на технологии и алгоритми**
* **Как работи схемата Routs Engine**

Routing Engine обработва всички процеси на протокола за маршрутизиране, както и софтуерните процеси, които контролират интерфейсите на рутера, компонентите на шасито, управлението на системата и потребителския достъп до рутера. Тези маршрутни и софтуерни процеси се изпълняват върху ядрото, което взаимодейства с механизма за пренасочване на пакети.

Routing Engine включва следните функции и характеристики:

* Обработка на пакети с протокол за маршрутизиране — Routing Engine обработва всички пакети, които се отнасят до протоколи за маршрутизиране, освобождавайки Packet Forwarding Engine да обработва само пакети, които представляват интернет трафик.
* Софтуерна модулност – Тъй като всеки софтуерен процес е посветен на различна функция и използва отделно пространство на процеса, отказът на един процес има малък или никакъв ефект върху другите.
* Задълбочена интернет функционалност — Всеки протокол за маршрутизиране е реализиран с пълен набор от интернет функции и осигурява пълна гъвкавост за рекламиране, филтриране и модифициране на маршрути. Политиките за маршрутизиране се задават според параметрите на маршрута (например префикс, дължина на префикса и атрибути на Border Gateway Protocol [BGP]).
* Мащабируемост — Маршрутните таблици на Junos са проектирани да задържат всички маршрути в текущите мрежи с достатъчен капацитет за разширяване. Освен това, ОС Junos може ефективно да поддържа голям брой интерфейси и виртуални схеми.
* Интерфейс за управление—Предоставени са различни нива на инструменти за управление на системата, включително интерфейса на командния ред на ОС Junos (CLI), протокола за управление на Junos XML, интерфейса за изработка и SNMP.
* Съхранение и управление на промените — Конфигурационните файлове, системните изображения и микрокодовете могат да се съхраняват и поддържат в първични и вторични системи за съхранение, което позволява локални или отдалечени надстройки.
* Мониторинг на ефективност и гъвкавост — Рутерът поддържа функции като обработка на аларми и броене на пакети на всеки порт, без да влошава производителността на препращане на пакети.

Routing Engine изгражда и поддържа една или повече таблици за маршрутизиране. От таблиците за маршрутизиране, Routing Engine извлича таблица с активни маршрути, наречена таблица за пренасочване, която след това се копира в машината за пренасочване на пакети. Дизайнът на ASIC позволява да се актуализира таблицата за препращане в механизма за пренасочване на пакети, без да се прекъсва работата на препращане.



* **Как работи шаблона на Controller с GET и POST заявки към Actions. Извикване на конкретни View, изпозвайки подходящ Model.**
* GET и POST са два различни типа HTTP заявки. GET се използва за преглед на нещо, без да го променя, докато POST се използва за промяна на нещо. Например, страница за търсене трябва да използва GET за получаване на данни, докато формуляр, който променя паролата ви, трябва да използва POST. По същество GET се използва за извличане на отдалечени данни, а POST се използва за вмъкване/актуализиране на отдалечени данни.
* GET извлича представяне на посочения ресурс и включва всички необходими данни в URL адреса. Трябва да можете да изисквате един и същ URL отново и отново безобидно. Имайте предвид, че GET не трябва да се използва за операции, които причиняват странични ефекти, като например използването му за извършване на действия в уеб приложения. Една от причините за това е, че GET може да се използва произволно от роботи или роботи, които не трябва да вземат предвид страничните ефекти, които една заявка трябва да причини.
* POST е за писане на данни, изпраща данни за обработка (например от HTML формуляр) на идентифицирания ресурс. Това може да доведе до създаване на нов ресурс или актуализации на съществуващи ресурси или и двете. Може да има странични ефекти, използвайки една и съща заявка няколко пъти, защото вероятно ще доведе до множество записвания. Браузърите обикновено ви предупреждават за това. POST не е защитен, данните са включени в тялото на заявката вместо в URL адреса, но е тривиално лесен за преглед/редактиране.
* **Алгоритми за филтриране и сортиране**

**ЗАКЛЮЧЕНИЕ**

Разработен е прототип на уеб-приложението за любимата ми рок група. За реализиране на уеб приложението са използвани средства като HTML,ASP.NET,C#. Разработен е лесен потребителски интерфейс, който подпомага потребителя при работа с приложението и

улеснява неговата функционалност. Предоставя възможност на

потребителя види най-актуалните новини за групата, да види техните албуми и предстоящи концерти и също така да даде предложение на групата за изява.

Изгледът на албумите и концертите е подреден,

така че да е интуитивен за потребителя и да дава ясна информация без потребителя да бъде объркан.

**ИЗПОЛЗВАНА ЛИТЕРАТУРА**

1. <https://bg.theastrologypage.com/graphical-user-interface>
2. <https://www.w3schools.com/html/html_intro.asp>
3. <https://www.hostinger.com/tutorials/what-is-html>
4. <https://umbraco.com/knowledge-base/aspnet/>
5. <https://dotnet.microsoft.com/en-us/learn/aspnet/what-is-aspnet>
6. <https://www.computerhope.com/jargon/g/gui.htm>
7. <https://lazaroibanez.com/difference-between-the-http-requests-post-and-get-3b4ed40164c1>
8. <https://www.juniper.net/documentation/en_US/release-independent/junos/topics/concept/routing-engine-m7i-architecture.html>