****ТЕХНИЧЕСКИ УНИВЕРСИТЕТ – СОФИЯ**

**ФАКУЛТЕТ ПО ПРИЛОЖНА МАТЕМАТИКА И ИНФОРМАТИКА**

**Дипломна работа**

**„Уебсайт за екология и екологичните проблеми“**

Изготвил: **Делян Дянков Димитров**

Фак. №: **181218001**

Специалност: **Приложна математика и информатика**

ОКС: **Бакалавър**

Дипломен ръководител: **гл. ас. д-р Силвия Баева**

**София**

**2021**

|  |  |
| --- | --- |
| **Съдържание:**   * **Глава 1. Увод** * **Актуалност и значимост на темата** * **Популяризиране на темата чрез уеб- пространството** * **Глава 2. Описание на средата за програмиране - Основни понятия и характеристики на език C#** * **Характеристика на езика**   + Структури от данни   + Класове и обекти/ методи и анонимни функции   + Принципи на обектно ориентираното програмиране   + Ламбда изрази и LINQ заявки   + Entity Framework- рамка за обект- релационно картографиране и конвертиране на обекти * **Особености на web разработката** * Майкрософт уеб рамката ASP.NET core * Модел-Изглед - Контролер (Model-View-Controller или MVC) * Уеб сървър - HTTP протокол, асинхронни процеси * Поток на приложението, филтри и междинен софтуер * Динамични уеб страници с Рейзър * Сигурност и самоличност * **Други важни компоненти на езика** * **Глава 3. Софтуерна реализация на заданието**   + **Среда за разработка и архитектура на проекта**     - Модел на базата данни     - Архитектура на проекта   + **Запознаване с потребителския интерфейс**     - Навигация, основни страници, потребителски интерфейс     - Дизайн на сайта   + **Разглеждане на основните функционалности**     - Потребителски форум - коментари и харесвания („Модел“)     - Публикуване на статии - имейли, оценки и принтиране („Контролер“)     - Създаване на събития – създатели и набиране на хора („Страница“)     - Новини – извличане на данни от други сайтове * **Глава 4. Използвани технологии и инструменти** * **Заключение** * **Използвана литература** * **Приложение на кода** | |
| **Глава 1. Увод**  **Глава 2.** **Описание на средата за програмиране - Основни понятия и характеристики на език C#**  **Обща характеристика**  ***C#*** *е съвременен обектно- ориентиран език за програмиране с общо предназначение, създаден и развиван от* ***Microsoft*** *като част от* ***.NET*** *платформата. На езика* ***C#*** *и върху* ***.NET*** *платформата се разработва изключително разнообразен софтуер: офис приложения, уеб приложения и уеб сайтове, настолни приложения, мултимедийни приложения, приложения за мобилни телефони и таблети, игри и много други. Програмите на* ***C#*** *представляват един или няколко файла с разширение* ***.cs****, в които се съдържат дефиниции на класове и други типове. Тези файлове се компилират от компилатора на* ***C#*** *до изпълним код и в резултат се получават асембли – файлове със същото име, но с различно разширение (****.exe*** *и* ***.dll****).*  **2.1 Структури от данни**  Много често, за решаване на дадена задача се нуждаем да работим с последователности от елементи. Например, за да прочетем дадена книга, трябва да прочетем последователно всяка една страница т.е. да обходим последова­телно всеки един от елементите на множеството от нейните страници. В зависимост от кон­крет­ната задача се налага да прилагаме различни операции върху тази съвкупност от данни. Структурите от данни се делят главно на примитивни и непримитивни, непримитивните от своя страна се делят на линейни и нелинейни.  **Примитивни типове от данни в C# са:**  -     Целочислени типове **sbyte**, **byte**, **short**, **ushort**, **int**, **uint**, **long**,**ulong**;  -     Реални типове с плаваща запетая – **float**, **double**;  -     Реални типове с десетична точност –**decimal**;  -     Булев тип – **bool**;  -     Символен тип – **char**;  -     Символен низ (стринг) – **string**;  -     Обектен тип – **object**.   |  |  |  |  | | --- | --- | --- | --- | | **Тип данни** | **Стойност по подразбиране** | **Минимална стойност** | **Максимална стойност** | | **sbyte** | 0 | -128 | 127 | | **byte** | 0 | 0 | 255 | | **short** | 0 | -32768 | 32767 | | **ushort** | 0 | 0 | 65535 | | **int** | 0 | -2147483648 | 2147483647 | | **uint** | 0u | 0 | 4294967295 | | **long** | 0L | -9223372036854775808 | 9223372036854775807 | | **ulong** | 0u | 0 | 18446744073709551615 | | **float** | 0.0f | ±1.5×10-45 | ±3.4×1038 | | **double** | 0.0d | ±5.0×10-324 | ±1.7×10308 | | **decimal** | 0.0m | ±1.0×10-28 | ±7.9×1028 | | **boolean** | false | Възможните стойности са две – **true** или **false** | | | **char** | '\u0000' | '\u0000' | ‘\uffff’ | | **object** | null | - | - | | **string** | null | - | - |   **Непримитивни- линейни типове от данни в C# са:**   * Масивите представляват съвкупности от променливи с дадени стойности. Тези променливи могат да бъдат примитивен или референтен тип. Елементите на масивите в C# са номерирани с числата 0, 1, 2, ... N-1. Тези номера на елементи се наричат индекси, а броят елементи в даден масив N се нарича дължина на масива. Масивите също така, могат да бъдат от различни размерности - едномерни, двумерни, n-мерни. * Стек Това е подреден списък, където съществуващия елемент се изтрива само от горната част на стека (TOS). Тъй като цялото изтриване и вмъкване в стека се извършва от горната част на стека, последният добавен елемент ще бъде първият, който ще бъде премахнат от стека. Тези операции се нарича LIFO (Last in first out). * Опашката добавя нови елементи в задния край. По същия начин, изтриването на съществуващите елементи, става на предния край. Структурата от данни наподобява принципа на опашка пред каса на супермаркет. Тези операции се наричат (FIFO) First in first out.   **Непримитивни- нелинейни типове от данни в C#** са **д**ървета и графи. Те нямат вградени класове в езика, а се имплементират от програмиста в зависимост от ситуацията. Причината затова е, че могат да служат за решаването на различни задачи.   * В програмирането дърветата са изключително често използвана структура от данни, защото те моделират по естествен начин всякакви йерархии от обекти, които постоянно ни заобикалят в реалния свят.   Пример за дърво:   * Граф, накратко, е множество от обекти (наричани "върхове" на графа) и връзки между тях (наричани "ребра" в графа). Както обектите, така и връзките между тях могат да бъдат много различни. В примера с Facebook върховете в графа са хората, докато ребрата са приятелствата. В Google Maps върховете са различните адреси/кръстовища/сгради докато ребрата са улиците.   Пример за граф:  Таблично представяне на структурите от данни    **2.2 Класове и обекти/ методи и анонимни функции**   * **Класове и обекти в C#**   Обектно-ориентираното програмиране е модел на програмиране, който използва обекти и техните характеристики, и взаимодействия за изграждането на компютърни програми. Софтуерните обекти моделират обекти от реалния свят или абстрактни концепции, които също се разглеждат като обекти. Класът дефинира абстрактните характеристики на даден обект. Той е план или шаблон, чрез който се описва даден обект. Класовете са градивните елементи на ООП и са неразделно свързани с обектите. Всеки обект е представител на точно един клас.   * **Методи/ Функции**   Всяко едно парче код, което изпълнява дадена функционалност и което сме отделили логически, може да изземе функционалността на метода. Точно това представляват **методите – парчета код, които са именувани** от нас по определен начин и които могат да бъдат **извикани** толкова пъти, колкото имаме нужда.   * **Анонимни функции в C#**   В обектно-ориентираните езици много често се налага да се дефинират малки класове с цел еднократно използване. Типичен пример за това е класа **Point**, съдържащ само 2 полета – координатите на точка. Създаването на обикновен клас само и единствено за еднократна употреба създава неудобство на програмистите и е свързано със загуба на време.  var myCar = new { XCoordinate = 5.5d, YCoordinate = 6.00d };  **2.3 Принципи на обектно ориентираното програмиране**   * **Абстракцията** е процесът на скриване на ненужни подробности от потребителя. Например, когато натискаме „харесва ми“ бутона във фейсбук. Телефонът ни показва, че съобщението е изпратено, но какво всъщност се случва във фонов режим, когато го натиснем е скрито от нас, тъй като няма никаква необходимост да го знаем . С други думи, абстракцията позволява да опростим максимално сложни концепции и да боравим с по-опростения вариант. * **Капсулирането** скрива информация, която не е необходима за програмиста. Този, който създава класовете, определя какво да е скрито и какво да е публично видимо. Това става чрез изрично дефиниране като private (скрит) на всяко поле или метод, които не искаме да се ползват от друг клас. * **Наследяването** е начин класа родител да даде своите полета и методи на наследниците. Чрез него един клас може да „наследява“ методи и свойства от друг, по-общ клас. Пример, вълкът и лисицата са от семейство „кучеви“. Всички вълци и лисици са четириноги, бозайници, хищници. Тези характеристики могат да се зададат на по-общ клас родител „кучеви“, като след това наследниците могат да получат общите качества от родителя без да се задават наново. * **Полиморфизъм**. Дефинирали сме даден клас – семейство „кучеви“. Само че в това „семейство“ има много и разнообразни хищници като вълци, лисици и кучета. Те имат различно поведение по отношение на издаването на звуци. Полиморфизмът ни позволява да третираме кое да е от тези животни като член на семейство „кучеви“ и да изпълним команда да извършване на звук, без значение какво точно е „животното“, на което задаваме командата. За да извърши това, полиморфизмът използва пренаписване на методи в наследените класове, с цел промяна на първоначалното им поведение, което е прихванато от базовия клас. Това ни дава възможност за множество различни реализации за едно и също нещо. * **СОЛИД (SOLID) принципите:**  |  |  | | --- | --- | | **Инициали** | **Концепция** | | **S** | **Принцип за единствена отговорност**  Даден клас или метод трябва да има една- единствена отговорност. | | **О** | **Принцип отворен/затворен**  Софтуерните единици трябва да са отворени за разширение, но да е затворена за промяна. | | **L** | **Принцип на заместване на Лисков**  Всеки наследник (подтип) трябва лесно да заменя всичките си базови типове. | | **I** | **Принцип за разделяне на интерфейсите**  Много на брой малки интерфейси е по-добре от един голям общ интерфейс. | | **D** | **Принцип на обръщане на зависимостите**  Всички класове трябва да зависят от абстракции и нито един не трябва да зависи от конкретен клас. |   **2.4 Ламбда изрази и LINQ заявки (Разширяващи методи)**   * **Ламбда** изразите представляват анонимни функции, които съдържат изрази или последователност от оператори. Всички ламбда изрази използват ламбда оператора **=>**, който може да се чете като "отива в". Лявата страна на ламбда оператора определя входните параметри на анонимната функция, а дясната страна представлява израз или последователност от оператори, която работи с входните параметри и евентуално връща някакъв резултат. * **LINQ** (**L**anguage-**In**tegrated **Q**uery) представлява редица разширения на .NET Core, които включват интегрирани в езика заявки и операции върху елементи от даден източник на данни (най-често масиви и колекции). LINQ e много мощен инструмент, който доста прилича на повечето SQL (език за структурирани заявки) езици и по синтаксис и по логика на изпълнение. LINQ реално обработва колекциите по подобие на SQL езиците, които обработват редовете в таблици в база данни. Той е част от **C#** синтаксиса и се състои от няколко основни ключови думи. Най- често се използва за филтриране, сортиране, групиране, съединяване на данни, избор на резултати и др. Често се използват в комбинация с анонимни функции.   **2.5 Entity Framework- рамка за обект- релационно картографиране и конвертиране на обекти**  Entity Framework Core е стандарт за ORM (**O**bject **R**elational **M**apping) в C# и .NET Core приложения. EF Core позволява картографиране между релационна база и обектно- ориентиран модел чрез подходите "database first" и "code first" и предоставя мощно обектно-ориентирано API (Приложно-програмният интерфейс) за заявки към базата данни и извършване на CRUD (Създаване, Четене, Ъпдейтване, Изтриване) операции. EF core предоставя както допълнително ниво на абстракция, така и лесен начин за обработка на данните от базата. Значително улеснение за програмистите е използването на LINQ вместо писане на SQL код.  **Особености на web разработката**  **2.6 ASP.NET Core (A**ctive **S**erver **P**ages**)**  ASP.NET Core е безплатна софтуерна рамка за уеб разработка, с отворен код. Неин предшественик е ASP.NET. Тя е разработена съвместно от Майкрософт и общността, която е събрала през годините на своето развитие. ASP.NET Core е модулна 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 уеб пакет, тя има висока степен на съвместимост с ASP.NET MVC. Той поддържа трите модела за разработка, уеб страници, MVC (модел, изглед, контролер) и уеб форми.  **2.7 Модел-Изглед-Контролер (M**odel**-V**iew**-C**ontroller или **MVC)**  MVC представлява архитектурен шаблон при програмния дизайн. Той отговаря за разделянето на бизнес логиката на три взаимосвързани части: “**M**odel”, “**V**iew” и “**C**ontroler”. Техните функции са: **Model** – това представлява „ядрото“ на приложението ви. В него се записват всички данни, върху които искаме да работим. Моделите служат за създаване на обекти най- често репрезентиращи реални обекти от заобикалящия ни свят. Тези класове са с разширения cs. **View** – най-общо казано това е изходна част от софтуера, която визуализира наличните, обработени данни. В тях се съдържа html на уеб страницата и с разширение .html или .cshtml за „Разор страници“ за които ще говорим малко по- долу. **Controller** – това е самия код или още казано бизнес логиката на приложението, която извиква и изпълнява основните методи върху събраните в модела данни и по този начин те могат да бъдат обработвани.  **2.8 Уеб сървър - HTTP протокол, асинхронни процеси**   * **IIS** (**I**nternet **I**nformation **S**erver) e разширяем и модулен уеб сървър, създаден от Microsoft, подходящ за уеб разработка с помощта на технологии на Microsoft като ASP.NET приложенията работят в интегриран режим по подразбиране. Въпреки това, за да се възползваме от предимствата, предоставени от по-тясна интеграция, трябва да направим някои промени в конфигурацията на приложението. Той поддържа всички съвременни протоколи и предлага няколко функции за сигурност и удостоверяване. Според последните статистически данни IIS е вторият най-популярен уеб сървър след HTTP сървъра на Apache. * **IIS Express** е лека, самостоятелна версия на IIS, оптимизирана за разработчици. IIS Express улеснява използването на най-актуалната версия на IIS за разработване и тестване на уебсайтове. Има някои ключови разлики. Важна разлика е начинът, по който се управляват работните процеси. В IIS услугата за активиране на процесите на Windows (WAS) безшумно активира и деактивира уеб приложения и потребителят няма пряк контрол. В IIS Express няма WAS и потребителят има пълен контрол върху активирането и деактивирането на приложението. * **HTTP** протокола (**H**yper**T**ext **T**ransfer **P**rotocol) е стандартният протокол за трансфер на данни през интернет. Протокол е просто стандартизиран формат за трансфер на данни между две устройства. В този случай браузъра е HTTP клиента докато уеб сървъра – този който съхранява търсения уеб сайт се явява HTTP сървър. Браузъра изпраща HTTP заявка, която съдържа информация за търсения ресурс. Тази заявка изисква от сървъра да върне някакъв HTTP отговор, който в повечето случаи се състои от търсените ресурси. HTTP е протокол без състояние. Това се отнася до поведението на заявките – в момента в който сървъра получи една заявка след като бъде обработена то той я “забравя” или бива изтривана от паметта и съответно вече няма достъп до нея. Същото важи и за клиента и неговата заявка. Това прави HTTP ефикасен и прост протокол, а и това е причината поради която е толкова успешен за нуждите за трансфер на данни в уеб пространството. Основните заявки (Requests) са GET, POST, PUT, DELETE. Отговора на сървъра се състои от статус код: Информационни отговори (100–199), Успешен отговор (200–299), пренасочващо съобщение (300–399), Клиентска грешка (400–499), Сървърна грешка (500–599).      * **Многонишково програмиране в C # (Асинхронни процеси)**   **Многонишковите** (multithreaded) програми представляват програми, които могат да изпълняват едновременно няколко редици от програмни инструкции. Всяка такава редица от програмни инструкции наричаме thread (нишка). Изпълнението на многонишкова програма много прилича на изпълнение на няколко програми едновременно. Например в Microsoft Windows е възможно едновременно да слушаме музика, да теглим файлове от Интернет и да въвеждаме текст. Тези три действия се изпълняват от три различни програми (процеси), които работят едновременно.  Ключовите думи **async** и **await** в C# са сърцето на асинхронното програмиране.  Чрез използването на тези ключови думи, могат да се използват ресурсите на .NET  Framework или Windows Runtime, за да се създадат асинхронни методи с  приблизителната сложност на синхронните методи. Асинхронните методи се  дефинират с async, а те се извикват с await.  • Пишем ключовата дума async пред името дефинирания метод.  • Името на метода завършва с Async суфикс по конвенция.  • Възможните връщани резултати са:  o Task (задача) - ако методът връща състояние, където операндът има тип (резултатна задача) TResult.  o Task (задача)- ако методът не връща резултат.  o void (не връща задача) - Ако се създава асинхронен event handler (събитие).  • Методът обикновено включва поне едно await извикване, което маркира точката, през която не може да се продължи докато не се изчака готов резултат. Тогава, методът се прекъсва и изчаква асинхронната операция да приключи.    **Поток на приложението, филтри и междинен софтуер**   * Уеб приложенията обработват заявки (requests) и произвеждат отговори (response). Целият процес е естествено подреден в някакъв вид поток (workflow). В повечето случаи процесът е разширяем и променящи се. Уеб приложенията имат различни среди за внедряване. Средите определят поведението на приложение.      * ASP.NET Core използва метода Configure() в StartUp.cs зa конфигуриране на HTTP заявката, определете поведението за различни среди (Тестова, разработка и продуктова). Това се прави с помощта на IApplicationBuilder и IHostingEnvironment.  1. public void Configure(IApplicationBuilder app, IHostingEnvironment env) 2. { 3. if (env.IsDevelopment()) { app.UseDeveloperExceptionPage(); } 4. else { app.UseExceptionHandler("/Home/Error"); } 6. app.UseHttpsRedirection(); 7. app.UseStaticFiles(); 8. app.UseCookiePolicy(); 10. app.UseMvcWithDefaultRoute(); 11. }  * Опциите за конфигуриране, по конвенция, се задават в ConfigureServices(). Извиква се преди метода Configure() от WebHost. Типичният модел е Add{Service} и след това services.Configure(Service). Добавянето на услуги към контейнера за услуги прави services достъпни в приложението, това се случва с помощта на инжектиране на зависимост (Dependency Injection).  1. public void ConfigureServices(IServiceCollection services) 2. { 3. // Преходните обекти винаги са различни 4. // Нов екземпляр се предоставя на всеки контролер и услуга 5. services.AddTransient<DataService>(); 6. // Обектите с обхват са едни и същи в рамките на заявка 7. // Те са различни при различните заявки 8. services.AddScoped(typeof(DataService)); 9. // Единичните обекти са еднакви за всеки обект и заявка. 10. services.AddSingleton<DataService>(); 11. }  * **Обработване на грешки (Error Handling)**   + **Страница за обработване на грешки от програмиста**      * + **Персонално зададени грешки**   Конфигурирането на персонализирана страница за изключение се извършва с помощта на ExceptionHandlerMiddleware   1. public void Configure(IApplicationBuilder app, IHostingEnvironment env) 2. { 3. app.UseExceptionHandler("/Home/Error"); 4. }    * **Статус код за грешка**      * **Междинен софтуер (Middleware)**   Софтуер, сглобен в конвейер за приложения с междинни компоненти всеки компонент. Обработва заявки и отговори, избира дали да се предаде заявката към следващия компонент в конвейера, може да извършва работа преди или след извикване на следващия компонент в конвейера. В ASP.NET Core делегатите на заявки изграждат конвейера за заявки.  Може да се създаде собствен Middleware или да се използват вградените:   |  |  | | --- | --- | | **Междинен софтуер** | **Начин на използване** | | **Автентикация** | **app. UseAuthentication ()** | | **Бисквитки** | **app. UseCookiePolicy ()** | | **Ядра** | **app. UseCors ()** | | **Диагностика** | **app. UseDevelopmentExceptionPage ()**  **app. UseExceptionHandler (…)**  **app. UseStatusCodePages ()** | | **HTTPS пренасочване** | **app. UseHttpsRedirection ()** | | **HSTS** | **app. UseHsts ()** | | **Статични полета** | **app. UseStaticFiles ()** |  |  |  | | --- | --- | | **Отговор на кеша** | **app. UseResponseCaching ()** | | **Сравнение на отговора** | **App. UseResponseCompression ()** | | **Заявка за локализация** | **app. UseRequestLocalization (…)** | | **Ротиране** | **app. UseRouter (…)** | | **Сесия** | **app. UseSession ()** | | **URL пренаписване** | **app. UseRewriter (…)** | | **WebSockets** | **app. UseWebSockets (…)** | | **Други** | **app. UseWelcomePage ()** |  |  |  | | --- | --- | | **Филтри** | **Описание** | | **Авторизация** | **Определете дали Клиентът е упълномощен за достъп до поисканата функционалност.** | | **Ресурси** | **Стартира веднага след оторизацията. Може да изпълнява код преди и след останалия процеса на работа.** | | **Действия** | **Изпълнява се непосредствено преди и след извикване на отделен метод за действие.** | | **Грешки** | **Използва се за прилагане на глобални политики за необработени грешки, които възникват.** | | **Резултати** | **Изпълнява се непосредствено преди и след изпълнението на отделни резултати от действие.** |  * Филтрите ни позволяват да изпълняваме код преди или след определени етапи в конвейера за обработка на заявки. Филтрите са подобни, но не са същите като Middleware работят на нивото на ASP.NET Core а филтрите работят само на ниво MVC. Могат да бъдат синхронни и асинхронни.     **2.10 Динамични уеб страници с Рейзър**  Страниците в ASP.NET Core MVC използват Razor View Engine за вграждане на .NET код в HTML код. Обикновено те съдържат минимална логика, свързана само с представянето на данни. Данните могат да се предават към изглед с помощта на ViewData, ViewBag или чрез ViewModel (строго типизиран изглед). Позволява използването на if, else, for, foreach и други.    Данните се изпращат от контролера по следния начин:    Рейзър страниците ни позволяват да използваме dependency injection.     * **HTML помощници и помощни маркери** - @HTML има методи, които връщат низ може да се използват за създаване на полета за входни данни (username, password и др.), създаване на връзки/хиперлинкове, създаване формуляри и текстови полета.  |  |  | | --- | --- | | HTML Helpers | | | @Html.ActionLink | @Html.TextBox | | @Html.BeginForm | @Html.TextArea | | @Html.CheckBox | @Html.Password | | @Html.Display | @Html.Hidden | | @Html.Editor | @Html.Label | | @Html.DropDownList | @Html.Action |  * **Частични страници и компоненти**   **Частичните страници** изобразяват само парчета от страница. Те разбиват големите файлове на по-малки страници и по този начин намаляват дублирането на кода.  Обикновено се поставя в споделена папка или в същата директория, където се използват. Те може да се използва с помощта на @HTML (Partial, PartialAsync, RenderPartial) или Tag Helper (<partial name="" model="" view-data="" for="" />).  **Компонентите** от своя страна са подобни на частичните страници, но те преизползват само парче от страницата (а не цялата страница). Те могат да имат параметри или да съдържат бизнес логика. Компонентите се състоят от 2 части-  клас (обикновено част от ViewComponent) и резултат (обикновено самата страница).  **2.11 Сигурност и самоличност**   * **Междусайтови скриптове XSS (Cross Site Scripting).** Като една от най-често срещаните заплахи за киберсигурността, скриптовете между сайтове (XSS). Атаките са възможни в HTML, Flash, ActiveX и CSS вариант. Въпреки това, JavaScript е най-честата цел на киберпрестъпниците. Има няколко правила, които трябва да спазвате, за да бъдем защитени, никога не се поставят ненадеждни данни в HTML- ла. Преди да поставим ненадеждни данни някъде, трябва да се уверим се, че са защитени чрез кодиране, анализиране, валидиране и проверяване за злонамерено съдържание. Razor има вградена защита от този тип атаки, но инсталирането на NuGet пакета и използването на HtmlSanitizer е добра допълнителна мярка срещу такива атаки. * **SQL инжектиране** - обикновено се появява, когато потребител въведи входни данни, като име и парола. Вместо тях, потребителят въвежда злонамерена SQL заявка, която ще изпълни в базата данни. За да защитим от можете да използвате SQL параметри. SQL параметрите са стойности, които се добавят към SQL заявка по време на изпълнение по контролиран начин. Създават се параметризирани заявки които се приемат буквално от езика и не се изпълняват като SQL код. * Фалшифицирането на **междусайтови заявки (CSRF / XSRF)** е атака за уеб сигурност през HTTP протокола. Атаката позволява изпълнение на неоторизирани команди от името на някой потребител чрез използване на неговите бисквитки, съхранени в браузъра. Хакера има валидни разрешения за изпълнение на исканата команда, без знанието на потребителя. Когато използвате тага <form> от ASP.NET Core, той автоматично ще добави специално скрито поле във формуляра с произволна стойност, наречен „токен против фалшифициране“ (**anti-forgery token**). Добавянето на този токен глобално се случва по следния начин:  1. services. AddMvc (options => 2. options. Filters. Add (new AutoValidateAntiforgeryTokenAttribute ()));   **Други известни уеб атаки:**   * URL/HTTP манипулационни атаки (Parameter Tampering) * Атаки с груба сила (също наречени DDoS) * Недостатъчен контрол на достъпа * Твърде много информация в Грешки * Липсващ SSL (HTTPS) / MITM * Phishing/Social Engineering * Семантични URL/HTTP атаки (URL/HTTP манипулация) * Man in the Middle (винаги използваме SSL сертификат) * Недостатъчен контрол на достъпа * Други видове data injection (Винаги проверяваме данните) * DoS and DDoS (CAPTCHA и Firewall) * Проблеми със сигурност в софтуера (Винаги се използват последни версии)   **Удостоверяване и оторизация**  Удостоверяване е процесът на проверка на самоличността на потребител или компютър. Оторизация**/** Упълномощаване е процесът на определяне какво е разрешено на потребителя да прави в компютър или мрежа.  Система за удостоверяване и оторизация в ASP.NET Core, поддържа ASP.NET MVC, страници, уеб API и SignalR. Обработва потребители, потребителски профили, влизане/излизане, роли и др. Обработва съгласието за бисквитки и GDPR (Общ регламент относно защитата на данните). Поддържат се външни доставчици за вход през платформи като Facebook, Google, Twitter и др. Използват се атрибутите [Authorize] и [AllowAnonymous], за да се конфигурира оторизиран/анонимен достъп до конкретен контролер или метод. Мога да се подават и роли когато потребителя бива авторизиран.   1. [Authorize(Roles="Administrator")] 2. public class AdminController: Controller 3. { … }   Видове Удостоверяване и оторизация:   * Базирано на бисквитки (идентификация) * Чрез Windows * Базирано на облак * JSON уеб токен (JWT)   **Други важни функции на езика**   * **Условният оператор** ?: е оператор в езика C#. Известен е още като **тернарен оператор**, тъй като е единственият оператор, който приема 3 операнда.   операнд1 ? операнд2 : операнд3  Първият операнд или условието на условната конструкция може да бъде булева променлива или булев израз и може да приема двете булеви стойности истина и лъжа. Ако след извършването на необходимите преобразувания операнд1 се сведе до истинно твърдение, то тогава след изпълнението си тернарният оператор ще върне стойността на операнд2, в противен случай, върнатата стойност ще бъде стойността на операнд3.   * **Шаблонни типове в C#**   Много често възниква проблемът програмистът да създава класове, които са сходни по функционалност, а се различават само по типа на обектите, с които работят. Например, необходимо е да се изгради списък като елементите на списъка са цели числа. В този случай полетата в класа ще са от целочислен тип. Методите, който биха били включени в подобен клас като добавяне, изтриване и търсене на елемент в списъка, биха работили с променливи от целочислен тип. Ако същата задача бъде поставена за изграждане на списък от дробно-десетични стойности или на низове, структурата и функционалността на следващите два класа ще бъдат идентични с тези на първия клас като единствената разлика ще бъде в типовете на данните. Това налага въвеждането на шаблонните типове (Generics) в езика.  Пример за шаблонен клас и поле:   1. public class Generic 2. { 3. public T Field; 4. }  * **Атрибути**   Атрибути в .NET са специални инструменти, които ни позволяват да вградим допълнителни метаданни. Атрибутите могат да бъдат приложени към целия тип (клас, интерфейс и т.н.) или към отделните му части (метод, свойство и т.н.). Основата на атрибутите е класът, от който се извличат всички други класове атрибути. .NET има много вградени класове атрибути. И също така можем да създадем свои собствени класове атрибути. Примера по- долу показва атрибут за упълномощаването на потребител върху даден метод. Методът е Post заявка.    **Глава 3. Софтуерна реализация на заданието**  **Среда за разработка и архитектура на проекта**  **Visual Studio (VS) е интегрирана среда за разработка (IDE) от Microsoft, използвана за изграждане на приложения и уебсайтове, които поддържат широк спектър от езици за програмиране като C, C++, C# или Visual Basic. Visual Studio е най-сложният наличен инструмент, който отговаря на всички изисквания за бърза, продуктивна и изчистена от към грешки разработка на софтуер. С безброй много функции и поддръжка на хиляди плъгини, рядко се нуждаем от друг инструмент, тъй като можете да разчитате на Visual Studio през целия жизнен цикъл на разработка на приложения. Предлага се в различни издания като Express, Professional или Ultimate.**  **.NET** е безплатна, кросплатформена платформа за разработчици с отворен код за изграждане на много различни видове приложения. С .NET можете да използвате множество езици, редактори и библиотеки, за създаване за уеб сайтове, мобилни приложения, настолно- компютърни приложения, игри, IoT и много други.  **Microsoft SQL Server** е система за управление на релационните бази данни (RDBMS), която поддържа голямо разнообразие от приложения за обработка на транзакции, бизнес разузнаване и анализ в корпоративни ИТ среди. Microsoft SQL Server е една от трите водещи на пазара технологии за бази данни, заедно с Oracle Database и DB2 на IBM. Подобно на друг софтуери за RDBMS, Microsoft SQL Server е изграден върху SQL, стандартизиран език за програмиране, който администраторите на базите данни (DBA) и други ИТ специалисти използват за управление на им.  **Модел на базата данни- UML диаграма и Основни таблици**  За моделиране на обектите в Entity Framework е използван „Код първо“ подходът, обектите са моделирани и конфигуриране чрез C# класове. Използвани са релации едно към много, много към много и връзка към същата таблица. Връзките са създадени чрез главни и второстепенни ключове. Спазени са първа, втора, трета нормална форма за изграждането на базата.  **Основните таблици са:** AspNetUsers, AspNetRoles, AspNetUserTokens, AspNetUserLogins, Settings автоматично създадени от AspNet.  За основните функционалности на сайта са създадени таблиците:  **Proposals** – връзка към AspNetUsers, Images, Votes  **Posts** – връзка към AspNetUsers, Comments, UserLikes  **Evenets** – връзка към AspNetUsers, UserEventSignedIns, UserEventHosts  **News** – връзка към CategoryNews  **\_EFMigrationsHistory** - отделна таблица съдържаща миграциите  **UML диаграма**    **Архитектура на проекта**  Solution- а на проекта е разделен в 4 папки. В тях се съдържат под проектите които отговарят за данните, бизнес логиката, тестовете и уеб частта.    Папката **Data** съдържа 3 библиотеки:  **GreenCap.Data** – се грижи за миграциите, конфигурациите, първоначалното добавяне на данни и общите настойки за проекта.  **GreenCap.Data.Model** – пази всички модели, шаблоните по които се изграждат обектите.  **GreenCap.Data.Common –** съдържа общите полета за изграждане на моделите и техните допустими стойности в repository- то.  Папката **Services** има 4 библиотеки:  **GreenCap.Services** – Съдържа в себе си общите константи на бизнес логиката и скрейпъра за новини.  **GreenCap.Services.Data** – Съдържа бизнес логиката на проект, общи константи, персонализираните грешки и интерфейсите за бизнес логиката.  **GreenCap.Services.Mapping** – Съдържа настройките на AutoМapper- а.  **GreenCap.Services.Messaging** – Отговаря за изпращането на имейли и тяхното конфигуриране чрез SendGrid.  Папката **Tests** има 3 проекта:  **GreenCap.Services.Data.Tests** - Отговарящ за тестовете на бизнес логиката.  **GreenCap.Web.Tests** – Отговарящ за уеб тестовете.  **Sandbox** – за всякакви други тестове.  Папката **Web** има 1 проект и 2 библиотеки:  **GreenCap.Web** – Уеб проекта към които са закачени всички библиотеки, съдържа в себе си контролерите, страниците, статичните файлове, areas (промени в администраторската част и идентичността), настройките на стила на писане и началната точка на цялата разработка.  **GreenCap.Infrastructure** – Тук има възможност да се дефинират собствени атрибути.  **GreeenCap.Web.ViewModels** – В библиотеката се намират всички модели свързани с входно- изходните данни от сайта, пейджирането и модела на грешките.  Отделна от останалите библиотеки:  **GreenCap.Common** – Съдържа всички общи константи за solution- a, както и данните за първия администратор в сайта.    **Запознаване с потребителския интерфейс**  **Навигация, основни страници и потребителския интерфейс**  В снимките по- долу сайта е разгледан и на български и на английски език.  Началната страница за потребители, които не са влезли в сайта, изглежда по следния начин. Тя дава права на потребителя да я прочете, да вижда предложения (proposals), събития (events), новини (news) и формата за регистрация (register) и влизане (login) на потребители.    Формата за регистрация изисква от потребителя да въведе имейл адрес, парола и потвърждаване на парола. Потребителското име на новорегистрирания човек е името от имейла му. За успешна регистрация имейл не трябва да съществува в базата данни, въведените пароли трябва да съвпадат и да съдържат цифра, малка и голяма буква, знак и поне 6 символа дължина.    Формата за влизане в потребителския профил изисква имейл и парола. Има опция за запомняне на данните, за забравена парола и за връзка към регистрационната форма от горната екранна снимка.    Навигацията в сайта се извършва от панела за навигация.    Влезлият в сайта потребител има достъп до всички страници. В долните снимки ще разгледаме от гледна точка на такъв потребител. Той има повече права и достъп до всички страници на сайта. Началната страница визуализира по- подробна информация, която е представяне като статистически данни за активността в ГрийнКап (GreenCap).    Панела за навигация е с много повече полета. Всеки потребител има опция да разпъне чрез падащо меню допълнителните опции за всяка страница. Тези опции са видими само за потребители с регистрация в сайта. Допълнителни полета за създаване на предложение (Proposals) преглеждане на лично създадените предложения (Personal proposals) и опция за преглед на всички (View all). Същите опции са налични за раздел форум (Forum) и раздел събития (Events).    Ще разгледаме по- подробно страницата за **статии**, която се казва **еко- предложения** (Proposals).  Първата опция от падащото меню, за преглед на всички еко- предложения, ни отвежда на страницата за креативни еко идеи създадени от потребителите. На снимката могат да се видят карти които представляват различни споделени постове от потребители.    Втората опция ни отвежда към прозорец за създаване на ново еко- предложение. За тестови цели, създаваме статия за използване на метални сламки.    Трета опция ни отвежда към страницата, където предложенията са създадени от текущо- логнатият потребител. Както виждаме на снимката по- долу има една тема за метални сламки.    От страницата за всички еко- предложения можем да видим и статията публикувана от потребителя. Която е видима за всички посетители на сайта, регистрирани и не регистрирани.    В долната част на карта виждаме, че ние сме създадели на статията за метални сламки. Освен това виждаме и дата на създаване. Ако отворим дадена карта тя ни отвежда към страница с повече информация. За не регистрирани потребители тази страница ни дава опция само за връщане назад. Както и цялата информация отнасяща се за статията.    За регистрирани потребители в долната част на страницата се показват опции за редактиране, принтиране в pdf формат, изтриване на статията, само ако текущо- логнатият потребител, е неин създател, и изпращане на имейл под формата на pdf формат.    Опцията за принтиране ни отвежда на следната страница, където може да зададем желаните опции за принтирането на статията.    Възможно е и оценяването на дадена статия, като пред потребителя се визуализира средната аритметична оценка на базата на гласувалите потребители.    Има възможност и за редактиране на статия само в случай, че ние сме я създали.    Форума на сайта дава възможност на всички регистрирани потребителя да задават въпроси.  Първата опция от падащото меню е да се видят всички теми във форума. Като на тази страница се визуализира част от въпроса, неговото заглавие, броят на харесвания и не харесвания на дадена тема, потребителят задал въпроса, броят коментари и датата на създаване.      За разлика от еко- предложенията тук в падащото меню имаме и категории.    Категории ни отвеждат към друга страница. На тази страница виждаме, че съществуват 6 вида категории основни, здраве, екология, спорт, околна среда и други.    Създаваме примерна тема във форума на тема, добре ли е да се закусва преди тренировка. Избираме категория във форума да е тренировки и използваме редактора на текст за да окрасим нашата тема. Редактора е напълно функционален и включва в себе си най- известните опции като удебеляване, накланяне, уразмеряване, оцветяване, подравняване, подчертаване, добавяне на специални знаци, добавяне на код, добавяне на математически задачи, добавяне на таблици, хиперлинкове и много други.  Темата е налична в категория упражнения.    Отваряме темата за да видим пълния текст. Отново имаме опциите за промяна на темата и изтриване, само ако сме създатели на дадента тема. Освен това имаме и опция за коментиране.    Коментарите също имат възможност да се възползват от текстовия редактор. Освен това имаме опцията и да коментираме друг коментар. Ако сме създател на коментар имаме възможност да го изтрием или редактираме. Коментарите могат да се скриват или показват.    Последната ни опция е за даване на положителен или отрицателна оценка.    Страницата за създаване на събитие ни дава ни възможност да зададем име, URL път към снимка, начална и крайна дата и описание на събитието. Имаме опция и за задаване на брой хора нужни за даденото събитие, като различните потербители на по- късен етап могат да се запишат. Друга опция е и добавянето на организатори. Организатор/и за дадено събитие може/могат да бъде/ат един или много потребители. Организатора има право да изтрива или променя събитието, както и да се откаже от ролята си.  Събитието се визулизира като карта със заглавие и описание. Може да видим продължителността на събитието, имената на създателите и всички записани потребители. Записаните потребители и максималния брой нужни хора се визуализират като лента на прогреса. Всеки записал се потребител може да се откаже преди изтичането на събитието.    Страницата новини (News) зарежда последните новини от екологичния свят. Тези новини не са споделени от потребители на GreenCap, а се извличат от друг световно известен новинарски сайт. Този сайт може да бъде също посетен чрез линка „натисни тук“ и съдържанието на новините заедно със заглавията и снимките са същите.    Когато отворим дадена статия в нашия сайт тя се зарежда и може да бъде прочетена. Основната информация за създател на статията, дата на създаване и категория се взимат от оригиналният новинарски сайт. Новините могат да се прочетат и от потербители които не са регистрирани в сайта. Наличен е и линк през който може да се посети оригиналното съдържание на статиите. Всички правила и политики за извличане на данни от сайта [Ecology News - Biology News](https://phys.org/biology-news/ecology/sort/date/all/) са спазени.    Регистрираните потебители в GreenCap имат възможноста чрез страницата профил (profile) да коригират своите данни. Налична е възможност за смяна на парола, смяна на имейл адрес, добавяне на телефонен номер, изтриване на профила заедно с всичките му данни и сваляне на всички личните данни на потребителя, който са запазени в сайта.    Администраторите в сайта имат разширен панел с падащо меню. От там те могат да достъпят всички публикации в сайта, да ги редактират, изтриват или цензурират. Администраторите получават и буквата (A) до името си.    Избираме през администраторския панел да влезем в страницата новини. Виждаме сбита информация за новините които разгледахме малко по- нагоре. Възможни са опциите за изтриване, редактиране или отваряне на новината за преглед. Страницата е аналогична за еко- предложенията, форума и събитията. Администратора може да търси по ключова дума или да селектира броя новините които се показват на страницата.    Когато статиите в дадена страница се натрупат, се активира навигационно меню. То дава възможност бързо да навигираме до следващите три, първата или последната страница, чрез кликане на стрелките или цифрите.    GreenCap има политика за поверителност и съхранение на данните. Тя може да се достъпи от долната част на страницата през линка (privacy). Страницата политика е разделена на 20 точки: Въведение, дефиниция, информация за събиране на данни, типове събирани данни, използване на данни, следене на кукитата, други данни, използване на данни, задържане на данни, трансфер на информация, разкриване на данни, бизнес транзакции, сигурност, права и основна политика на GDPR, съдържание на личната информация, статистика, линкове към други сайтове, контакти в сайта и други.    **Дизайн на сайта**  Използвам е **Bootstrap** платформата (framework) за изработката, дизайна и потребителски интерфейс. Софтуерът Bootstrap е client-side среда с отворен код, която съдържа набор от инструменти за създаване на уеб приложения и уебсайтове. Платформата използва комбинация от HTML, CSS и JavaScript код, позволяващи модерен и лесно преоразмерим интерфейс, подходящ за различни браузъри и платформи.  Използваните навигационни полета, бутони, форми, индикатори, контейнери и карти са представени на снимките по- долу.      **Разглеждане на основните функционалности**  Всички модели в проекта GreenCap    **Потребителски форум - коментари и харесвания („Модел“)**  Моделите в потребителския форум се управлява от класа **PostService** и имплементират интерфейса IPostService. Асинхронните методи на класа завършват с наставката Async. Някой от методите приемат Generic типове. Основните методи на класа са:   1. IEnumerable<T> GetAllPersonal<T> (int page, int itemsPerPage, string id); 3. Task CreateAsync (PostInputViewModel model, string id); 5. Task DeleteByIdAsync (int id, string userId); 7. Task UpdateAsync (int id, PostEditViewModel input, string userId); 9. int GetCount (); 11. int GetCountPersonal(string id); 13. int GetCountByCategory (string categoryName);   Методът **CreateAsync** проверява потребителят дали съществува в базата данни, ако не хвърля грешка. Ако потребителя е регистриран, методът създава обект от типа Post и добавя всички нужни данни. Накрая запазва поста асинхронно в базата данни.   1. public async Task CreateAsync(PostInputViewModel model, string id) 2. { 3. var creator = this.userDb.All().FirstOrDefault(x => x.Id == id); 5. if (creator == null) 6. { 7. throw new NullReferenceException(ExceptionMessages.UserDoesNotExist); 8. } 10. var modelToCreate = new Post 11. { 12. ProblemTitle = model.ProblemTitle, 13. Category = model.Category, 14. CreatedById = id, 15. Description = model.Description, 16. }; 18. await this.forumDb.AddAsync(modelToCreate); 19. await this.forumDb.SaveChangesAsync(); 20. }   Методът **GetAll**, първо проверява дали страницата съществува. След това чрез LINQ заявка от базата данни взима всички записи които не са soft- deleted подрежда ги по дата на създаване. Взима само постовете от текущо заредената страница и ги добавя в лист за по- нататъшното им използване.   1. public IEnumerable<T> GetAll<T>(int page, int itemsPerPage) 2. { 3. CheckIfPageAndItemsPerPageIsCorrect(page, itemsPerPage); 5. return this.forumDb 6. .AllAsNoTracking() 7. .OrderByDescending(x => x.CreatedOn) 8. .Skip((page - 1) \* itemsPerPage) 9. .Take(itemsPerPage) 10. .To<T>() 11. .ToList(); 12. }   Методът **GetByIdAsync** e помощен метод. Получава като параметър id- то на поста и проверява дали съществува в базата данни. Ако съществува взима първия пост с такова id и зарежда всичките данни за дадения пост.   1. public async Task<T> GetByIdAsync<T>(int id) 2. { 3. CheckIfIdIsCorrect(id); 5. return await this.forumDb 6. .AllAsNoTracking() 7. .Where(x => x.Id == id) 8. .To<T>() 9. .FirstOrDefaultAsync(); 10. }   Методът **UpdateAsync** проверява дали поста съществува в базата и дали логнатият потребител и негов създател. Ако условията са изпълнени промените се запазват в базата данни.   1. public async Task UpdateAsync(int id, PostEditViewModel input, string userId) 2. { 3. var post = await this.forumDb.All().FirstOrDefaultAsync(x => x.Id == id); 5. if (post == null) 6. { 7. throw new NullReferenceException(ExceptionMessages.PostNotFound); 8. } 10. if (post.CreatedById != userId) 11. { 12. throw new NullReferenceException(string.Format(ExceptionMessages.YouHaveToBeCreatorException, post.ProblemTitle)); 13. } 15. post.ProblemTitle = input.ProblemTitle; 16. post.Description = input.Description; 17. post.Category = input.Category; 19. await this.forumDb.SaveChangesAsync(); 20. }   Методът **DeleteByIdAsync** проверява дали записа съществува и дали логнатият потребител е негов създател. Ако условията са изпълнени се извършва soft- delete на записа.   1. public async Task DeleteByIdAsync(int id, string userId) 2. { 3. var modelToDelete = await this.forumDb.All().FirstOrDefaultAsync(x => x.Id == id); 5. if (modelToDelete == null) 6. { 7. throw new NullReferenceException(ExceptionMessages.PostNotFound); 8. } 10. if (modelToDelete.CreatedById != userId) 11. { 12. throw new NullReferenceException( 13. string.Format(ExceptionMessages.YouHaveToBeCreatorException, modelToDelete.ProblemTitle)); 14. } 16. this.forumDb.Delete(modelToDelete); 18. await this.forumDb.SaveChangesAsync(); 19. }   **Харесванията** има 3 основни метода. Няма нищо специално в тях и съдържанието на методите наподобява на тези от предишните примери.   1. Task SetLikeAsync(int postId, string userId, bool isPositive); 3. int GetLikes(int postId); 5. int GetDisslikes(int postId); 6. public int GetDisslikes(int postId) 7. { 8. return this.likesDb 9. .All() 10. .Where(x => x.PostId == postId) 11. .Where(x => !x.IsPositive) 12. .Count(); 13. } 14. public int GetLikes(int postId) 15. { 16. return this.likesDb 17. .All() 18. .Where(x => x.PostId == postId) 19. .Where(x => x.IsPositive) 20. .Count(); 21. } 22. public async Task SetLikeAsync(int postId, string userId, bool isPositive) 23. { 24. var like = this.likesDb.All() 25. .FirstOrDefault(x => x.PostId == postId && x.UserId == userId); 27. if (like == null) 28. { 29. like = new UserLike 30. { 31. PostId = postId, 32. UserId = userId, 33. }; 35. await this.likesDb.AddAsync(like); 36. } 38. like.IsPositive = isPositive; 39. await this.likesDb.SaveChangesAsync(); 40. }   **Коментари** имат 3 основни метода, подобни на методите от горните примери.   1. Task CreateAsync(int postId, string userId, string content, int? parentId = null); 3. bool IsInPostId(int commentId, int postId); 5. Task<int> DeleteByIdAsync(int id, string userId);   **Публикуване на статии - имейли, оценки и принтиране (“Контролер”)**   1. IEnumerable<T> GetAllPersonal<T>(int page, int itemsPerPage, string id); 3. Тask CreateAsync(ProposalViewModel model, string userId, string imagepath); 5. Тask DeleteByIdAsync(int id, string userId); 7. Task UpdateAsync(int id, ProposalEditViewModel input, string userId); 9. int GetCount(); 11. int GetCountPersonal(string id);   Методите на „Модела“, в публикуваните статии, са подобни като структура на разгледаните по- рано методи, в постове от раздел форум. Сега ще разгледаме класовете „Контролер“ които отново имат идентично поведение.  Get методът **All** връща като резултат страница с модела, зареден по- рано от service- a ProposalService. ViewModel който помага за визуализирането на резултатите в дадена страница приема 4 полета: брой еко- предложения на страница, номер на страница, брой на статиите и всички обекти от тип предложение (създадени от service- a).   1. public IActionResult All(int id = 1) 2. { 3. if (id <= 0) 4. { 5. return this.NotFound(); 6. } 8. const int ItemsPerPage = 9; 10. var viewModel = new ProposalsListOutputViewModel 11. { 12. ItemsPerPage = ItemsPerPage, 13. PageNumber = id, 14. EntitiesCount = this.proposalService.GetCount(), 15. Proposals = this.proposalService.GetAll<ProposalOutputViewModel>(id, ItemsPerPage), 16. AspAction = nameof(this.All), 17. }; 19. return this.View(viewModel); 20. }   Get методът **Edit** взима модела от service класа и връща задача със страницата и модeла за редакция.   1. public async Task<IActionResult> Edit(int id) 2. { 3. var inputModel = await this.proposalService.GetByIdAsync<ProposalEditViewModel>(id); 5. return this.View(inputModel) 6. }   Post методът **Edit** Проверява дали моделът е валиден, намира текущо логнатият потребител и обновява данните асинхронно чрез service- a ProposalService.   1. public async Task<IActionResult> Edit(int id, ProposalEditViewModel proposal) 2. { 3. if (!this.ModelState.IsValid) 4. { 5. return this.View(); 6. } 8. var userId = this.User.FindFirstValue(ClaimTypes.NameIdentifier); 9. await this.proposalService.UpdateAsync(id, proposal, userId); 11. return this.RedirectToAction(nameof(this.Details), new { id }); 12. }   Get методът **Create.**   1. [Authorize] 2. public IActionResult Create() 3. { 4. return this.View(); 5. }   Post методът **Create,** валидира модела, намира логнатият потребител, създава статия и пренасочва потребителя към страницата с всички постове.   1. [Authorize] 2. [HttpPost] 3. public async Task<IActionResult> Create(ProposalViewModel proposal) 4. { 5. if (!this.ModelState.IsValid) 6. { 7. return this.View(); 8. } 10. // get id from cookie 11. var userId = this.User.FindFirstValue(ClaimTypes.NameIdentifier); 12. var pathImages = $"{this.environment.WebRootPath}/Images"; 14. try 15. { 16. await this.proposalService.CreateAsync(proposal, userId, pathImages); 17. } 18. catch (Exception ex) 19. { 20. this.ModelState.AddModelError(string.Empty, ex.Message); 21. return this.View(proposal); 22. } 24. return this.Redirect(nameof(this.All)); 25. }   Get методът **Details,** взима модела на еко- предложение с всичките му данни и препраща към страница за визуализацията на предложението.   1. public async Task<IActionResult> Details(int id) 2. { 3. var model = await this.proposalService.GetByIdAsync<ProposalDetailsOutputViewModel>(id); 4. return this.View(model); 5. }   Post методът **Delete,** намира id- то на текущо- логнатият потребител и изтрива публикувана от негова статия по id- то на статията. След това пренасочва към всички публикации.   1. [Authorize] 2. [HttpPost] 3. public async Task<IActionResult> Delete(int id) 4. { 5. var userId = this.User.FindFirstValue(ClaimTypes.NameIdentifier); 7. await this.proposalService.DeleteByIdAsync(id, userId); 9. turn this.RedirectToAction(nameof(this.All)); 10. }   Post методът **SendToEmail,** намира имейла на потребителя, създава съдържанието като HTML и го изпраща чрез service- a emailSender. Накрая пренасочва към същата статия.   1. [HttpPost] 2. public async Task<IActionResult> SendToEmail(int id) 3. { 4. var targetEmail = this.User.FindFirstValue(ClaimTypes.Email); 6. var proposal = await this.proposalService.GetByIdAsync<ProposalDetailsOutputViewModel>(id); 7. var html = new StringBuilder(); 9. html.AppendLine($"<h1>{proposal.Title}</h1>"); 10. html.AppendLine($"<h3>{proposal.ShortDescription}</h3>"); 11. html.AppendLine($"<img src=\"{proposal.Images}\" />"); 12. html.AppendLine($"<h1>{proposal.Description}</h1>"); 14. await this.emailSender.SendEmailAsync( 15. GlobalConstants.AdministratorEmail, 16. GlobalConstants.SystemName, 17. targetEmail, 18. proposal.Title, 19. html.ToString()); 21. return this.RedirectToAction(nameof(this.Details), new { id }); 22. }   **Оценяване на статия/еко- предложение**  Контролера за оценки и лайкове са малко по- различен от останалите. За разлика от обикновените контролери ApiControllers са специализирани в връщането на данни те се грижат за прозрачно сериализиране на данните във формат, поискан от клиента. Освен това те следват различна схема за маршрутизиране по подразбиране, предоставяйки REST- full API по конвенция.   1. [ApiController] 2. [Route("api/[controller]")] 3. public class VotesController : BaseController 4. { 5. private readonly IVotesService votesService; 7. public VotesController(IVotesService votesService) 8. { 9. this.votesService = votesService; 10. }   **Създаване на събития – създатели и набиране на хора “Страници”**  **Новини – извличане на данни от други сайтове**  **Глава 4. Използвани технологии и инструменти** |  |
|  |  |

**Използвана литература на български**

1. В. Колев, С. Наков, „Основи на програмирането със C#“, София, 2011.
2. Въведение в програмирането със C#, Наков, Колев и колектив, 2011.
3. Програмиране за .NET Framework, (том 1 и том 2), Наков и колектив, 2005-2006.
4. Д. Албахари, Б. Албахари, „C# джобен справочник - Бърз помощник за C# програмисти“, изд. Асеневци, 2020.
5. Научна конференция „Иновационни ИКТ в бизнеса и обучението: тенденции, приложения и разработване“, Пампорово, 24-25 Ноември 2016

**Използвана литература на английски**

1. C# 10 in a Nutshell Книга от Джозеф Албахари
2. Clean Code от Робърт Сесил Мартин
3. Cracking the Coding Interview от Гейл Лаакман Макдауъл
4. CSharp Principles от Светлин Наков 2018
5. C# in Depth от Джон Скит

**Използвани сайтове и източници**

[Stack Overflow - Where Developers Learn, Share, & Build Careers](https://stackoverflow.com/)

[Wikipedia](https://www.wikipedia.org/)

[Обучение по програмиране - Софтуерен университет (softuni.bg)](https://softuni.bg/?utm_source=google&utm_medium=cpc&utm_campaign=S-SU-OurBrand-ProgrammingBasics&utm_term=softuni&utm_content=SoftUni&gclid=Cj0KCQjwwJuVBhCAARIsAOPwGASRqV8KPaSxqLsOmlhsbE-Txnft4JSQNw8gbMPtlmBLq5FIChndxA8aAqksEALw_wcB)

[Technology skills for business | Pluralsight](https://www.pluralsight.com/product/skills?utm_term=&pslp=product-skills&aid=7010a000002LZ5aAAG&promo=&utm_source=branded&utm_medium=digital_paid_search_google&utm_campaign=XYZ_EMEA_Brand_E&utm_content=&gclid=Cj0KCQjwwJuVBhCAARIsAOPwGARkiuY8Y-J855Eq1Yvyv5bA68OId0G4EnXqi8OrxqSyTOr2PMTOCV0aAk9zEALw_wcB)

[Nikolay Kostov's Blog](https://nikolay.it/)

[Bootstrap · The most popular HTML, CSS, and JS library in the world.](https://getbootstrap.com/)

[Bootswatch: Solar](https://bootswatch.com/solar/)

[Font Awesome](https://fontawesome.com/)

[C# docs - get started, tutorials, reference. | Microsoft Docs](https://docs.microsoft.com/en-us/dotnet/csharp/)

[JavaScript | MDN (mozilla.org)](https://developer.mozilla.org/en-US/docs/Web/JavaScript)

[SQL Server technical documentation - SQL Server | Microsoft Docs](https://docs.microsoft.com/en-us/sql/sql-server/?view=sql-server-ver16)

[W3Schools Online Web Tutorials](https://www.w3schools.com/)

[NikolayIT/ASP.NET-Core-Template: A ready-to-use template for ASP.NET Core with repositories, services, models mapping, DI, and StyleCop warnings fixed. (github.com)](https://github.com/NikolayIT/ASP.NET-Core-Template)

[RegExr: Learn, Build, & Test RegEx](https://regexr.com/)

[referati.org](http://www.referati.org/)

[(1478) freeCodeCamp.org - YouTube](https://www.youtube.com/c/Freecodecamp)

[(1478) IAmTimCorey - YouTube](https://www.youtube.com/user/IAmTimCorey)

[C# Corner - Community of Software and Data Developers (c-sharpcorner.com)](https://www.c-sharpcorner.com/)

[Ecology News - Biology News( ... sorted: date/all) (phys.org)](https://phys.org/biology-news/ecology/sort/date/all/)

**Приложение**

Код на програмата от софтуерната реализация: