********ТЕХНИЧЕСКИ УНИВЕРСИТЕТ – СОФИЯ**

**ФАКУЛТЕТ ПО ПРИЛОЖНА МАТЕМАТИКА И ИНФОРМАТИКА**

**Дипломна работа**

**Тема:**

**„Уебсайт за екология и формиране на екологична култура“**

Изготвил: **Делян Дянков Димитров**

Фак. №: **181218001**

Специалност: **Приложна математика и информатика**

ОКС: **Бакалавър**

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

**София**

**2022**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Съдържание**   |  |  |  |  | | --- | --- | --- | --- | | **Увод** | | | 3 | | **Глава 1. Описание на средата за програмиране - основни понятия и характеристики на език C#** | | |  | |  | 1.1. | Структури от данни | 5 | |  | 1.2. | Класове и обекти, методи и анонимни функции | 10 | |  | 1.3. | Принципи на обектно ориентираното програмиране | 10 | |  | 1.4. | Ламбда изрази и LINQ заявки | 12 | |  | 1.5. | Entity Framework- рамка за обект-релационно картографиране и конвертиране на обекти | **12** | | **Глава 2.** **Особености на web разработката** | | |  | |  | 2.1. | Майкрософт уеб рамката ASP.NET core | 13 | |  | 2.2. | Модел-Изглед-Контролер (Model-View-Controller) | 13 | |  | 2.3. | Уеб сървър- HTTP протокол, асинхронни процеси | 13 | |  | 2.4. | Поток на приложението, филтри и междинен софтуер | 16 | |  | 2.5. | Динамични уеб страници с Рейзър | 20 | |  | 2.6. | Сигурност и самоличност | 22 | |  | 2.7. | Други важни компоненти на езика | 24 | | **Глава 3. Софтуерна реализация на заданието** | | |  | |  | 3.1. | Среда за разработка и архитектура на проекта   * Модел на базата данни | 26 | |  | 3.2. | Запознаване с потребителския интерфейс   * Навигация, основни страници, дизайн | 30 | |  | 3.3. | Разглеждане на основните функционалности   * Потребителски форум (Модел) * Създаване на събития (Изглед) * Публикуване на статии (Контролер) | 46 | | **Заключение** | | | 63 | | **Използвани технологии и инструменти** | | | 64 | | **Използвана литература и източници** | | | 65 |   **Увод**  **Актуалност и значимост на темата**  Екологията е наука, раздел на биологията, която изучава взаимодействията между организмите, както и средата, в която живеят. Нейното наименование е от гръцки произход – от oikos, което означава „дом”, „къща” и logos- „наука“. Главна цел на науката е да изследва взаимоотношенията между животинските и растителните организми в дадените условия на околната среда, както и влиянието на средата за измененията, които настъпват в организмите.  В днешно време, природните катастрофи и катаклизми, предизвикват опасните за човека изменения в околната среда, като по този начин понятието екология придобива по-широк смисъл. Вече става дума не само за измененията в организмите, които настъпват при естествено променящи се природни условия, но и такива с външна намеса от действията на хората. Отрицателното въздействие на нарушената хармония и последиците от това са причинители на заболявания и световни катаклизми. Насищат се с отровни вещества въздухът, почвите, водите, които довежда до нарушения във функциите на организмите.  Многото специалности в екологията, като морска, растителна и статистическа, предоставят информация, за да се разбере по-добре света около нас. Тази информация също може да помогне да се подобри околната среда, да се управляват природните ресурси и да се защити нашето здраве. Нарастването на ролята на екологията засяга отношението ѝ към явления от социалния и културния живот. Опазването на природата се налага като важна тема. Това води до нейното лансиране и в уеб пространството.  **Популяризиране чрез уеб пространството**  Световна мрежа или само уеб, е информационна система от взаимно свързани хипертекстови документи, налични през интернет. Всеки човек има бърз и лесен достъп до информация. Само с една конкретна дума Гугъл търсачката ще предостави необходимите отговори. Общоизвестно е, че интернет може да отговори на всеки един въпрос.  Различните видове онлайн обучения представляват възможност за самоусъвършенстване. Могат да бъдат намерени безплатни съвети, които стъпка по стъпка обясняват какво трябва се да направи за постигането на целите. Интернет форум, също като уебфорум е онлайн платформа за дискусии, където анонимни или регистрирани потребители могат да участват. За разлика от чата, имат статичен вид и се заражда като уебсайт с различни теми за дискусии. От технологична гледна точка форумите са уеб приложения, в които потребителите сами генерират текстовото съдържание.  През последните години събитията в онлайн сайтовете се превърнаха от модерна и новаторска форма за организиране на ивенти в основен начин за осъществяване на такива. Предвид разпространението на COVID-19 и свързаните с него социална изолация и ограничения, значимостта на събитията и тяхното разпространение нарасна неимоверно, промотирането им през уеб- пространството стана по-лесно за изпълнение, а достигането до масовата аудитория - в пъти по-бързо.  **Целта** на настоящата дипломна работа е да се създаде уебсайт за екология и формиране на екологична култура, като за тази цел трябва да се подберат подходящи програмни среди.  **Дипломната работа се състои** от 3 глави, заключение с изводи, използвани технологии и инструменти и използвана литература и източници.  **Глава 1** Описва средата за програмиране – основни понятия и характеристики на език С#.  **Глава 2** Дава особеностите на web разработката.  **Глава 3** Представя софтуерната реализация на заданието.  **Глава 1.** **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**).   * 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 (таблица 1.1):**  -  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 | - | - |   **Таблица 1.1.** Примитивни типове от данни в C#  **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 могат да бъдат от различни размерности - едномерни, двумерни, n-мерни.  1. // Примери: 2. int [] arr = new int [10]; 3. char [] str = new char[80]; 4. int[] myArray = { 1, 2, 3, 4, 5, 6 }; 5. string[] daysOfWeek = { "Monday", "Tuesday", "Wednesday","Thursday", "Friday", "Saturday", "Sunday" };  * 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).  1. Stack myStack = new Stack(); 2. myStack.Push("Example"); 3. myStack.Push("1000"); 4. myStack.Pop(“Example”); 5. myStack.Peek(); 6. myStack.Count();  * 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.  1. Queue queue = new Queue(); 2. queue.Enqueue(1); 3. queue.Dequeue(1); 4. queue.Enqueue(3); 5. queue.Peek(); 6. queue.Count();   **Непримитивни - нелинейни типове от данни в C#** садървета и графи (фигура 1.1). Те нямат вградени класове в езика, а се имплементират от програмиста в зависимост от ситуацията. Причината 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.   **Фигура 1.1.** Пример за дърво   * Граф (фигура 1.2), 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.   **Фигура 1.2.** Пример за граф   * Речниците са известни още като **асоциативни масиви (associative arrays)** или **карти (maps)**. Всяко едно от различните имена подчертава една и съща характеристика на тази структура от данни, а именно, че в тях всеки елемент представлява съответствие между ключ и стойност – наредена двойка. Аналогията идва от факта, че в един речник, например тълковния речник, за всяка дума (**ключ**) имаме обяснение (**стойност**). Има два вида речници SortedDcitionary<key, value> който използват двуично дърво за сортиране на елементите и обикновеният речник Dictiornary<key, value> който използва хеш таблици за запазване на елементите.  1. IDictionary<string, int> countryNumOfPeople = new Dictionary<string, int>(); 2. countryNumOfPeople["Bulgaria"] = 7; 3. Console.WriteLine("Bulgaria has " + countryNumOfPeople["Bulgaria"] + "million residents.");  * Хеш- таблици (Hashtable) - Реализацията с хеш-таблица има важното предимство, че времето за достъп до стойност от речника, при правилно използване, теоретично не зависи от броя на елементите в него. За сравнение да вземем списък с елементи, които са подредени в случаен ред. Искаме да проверим дали даден елемент се намира в него. В най-лошия случай, трябва да проверим всеки един елемент от него, за да дадем категоричен отговор на въпроса "съдържа ли списъкът елемента или не". Очевидно е, че броят на тези сравнения зависи (линейно) от броят на елементите в списъка. При хеш-таблиците, ако разполагаме с ключ, броят сравнения, които трябва да извършим, за да установим има ли стойност с такъв ключ, е константен и не зависи от броя на елементите в нея.  1. Hashtable numberNames = new Hashtable(); 2. numberNames.Add(1,"One"); 3. numberNames.Add(2,"Two"); 4. numberNames.Add(3,"Three");     **Фигура 1.3.** Схематично представяне на структурите от данни   * 1. **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, 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 };   * 1. **Принципи на обектно ориентираното програмиране** * **Абстракцията** е процесът на скриване на ненужни подробности от потребителя. Например, когато натиснем бутона „харесва ми“ във фейсбук. Телефонът ни показва, че съобщението е изпратено, но какво всъщност се случва във фонов режим, е скрито от нас, тъй като няма никаква необходимост да го знаем. 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“. Но 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 за 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 са представени в таблица 1.2.**  |  |  | | --- | --- | | **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. |   **Таблица 1.2.** СОЛИД (SOLID) принципи   * 1. **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.   **1.5 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.  **Глава 2. Особености на web разработката**  **2.1 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. Разработена е съвместно от Майкрософт и общността, 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.  **2.2 Модел-Изглед-Контролер (M**odel**-V**iew**-C**ontroller или **MVC)**  MVC представлява архитектурен шаблон при програмния дизайн. Който 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, репрезентиращи реални обекти от заобикалящия ни свят. Тези класове са с разширения cs. **View** – това е изходна част от софтуера, която визуализира наличните, обработени данни. В тях се съдържа html на уеб страницата с разширение .html или .cshtml за „Разор страници“. **Controller** – това е самият код или бизнес логиката на приложението, която извиква и изпълнява основните методи върху събраните в модела данни и по този начин те могат да бъдат обработвани.  **2.3 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. За да се възползват от предимствата, предоставени от по-тясна интеграция, трябва да се направят някои промени в конфигурацията на приложението. Сървърът 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. Протоколът 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. 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. Отговорът 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).   **Фигура 2.1.** Схема на уеб сървър   * **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 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.    **Фигура 2.2.** Схема на асинхронни процеси  **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 променящ се. Уеб приложенията имат различни среди за внедряване. Средите определят поведението на приложение.     **Фигура 2.3.** Поток на приложението   * ASP.NET Core използва метода Configure() в StartUp.cs зa конфигуриране на HTTP заявката, определя поведението за различните 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**     **Фигура 2.4.** Примерна страница с грешка   * + **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**     **Фигура 2.5.** Код за грешка „Страницата не е намерена“   * **Междинен софтуер (Middleware)**   Софтуер, сглобен в конвейер за приложения с междинни компоненти, 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 представени в таблица 2.1.   |  |  | | --- | --- | | **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 ()** |   **Таблица 2.1.** Вграден междинен софтуер в езика   * 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.  |  |  | | --- | --- | | **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 оставащият процес 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.** |   **Таблица 2.2.** Филтри  **2.4 Динамични уеб страници с Рейзър**  Страниците в ASP.NET Core MVC използват Razor View Engine за вграждане на .NET код в HTML код. Обикновено 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 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 |   **Таблица 2.3.** Помощни HTML маркери   * **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 две части:  клас (обикновено част от ViewComponent) и резултат (обикновено самата страница).  **2.5 Сигурност и самоличност**   * **Междусайтови скриптове XSS (Cross Site Scripting).** Една от най-често срещаните заплахи за киберсигурността са 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 спазват, за да бъде защитен сайта, никога не се поставят ненадеждни данни в HTML- ла. Преди да се поставят ненадеждни данни някъде, трябва да се провери, 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 въведе входни данни, като име и парола. Вместо тях, потребителят въвежда злонамерена SQL заявка, която ще се изпълни в базата данни. За да се защитим може да се използват 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. Хакерът има валидни разрешения за изпълнение на исканата команда без знанието на потребителя. Когато се използва тага <form> от ASP.NET Core, 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)   **2.4 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 три 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 проблема 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, които са включени в подобен клас като добавяне, изтриване и търсене на елемент в списъка, са 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 позволяват да се вградят 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“. Могат да бъдат създадени собствени „класове атрибути“. Примера показва атрибут за упълномощаването на потребител върху даден метод. Методът е Post заявка.    **Глава 3. Софтуерна реализация на заданието**  **3.1 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 откъм грешки разработка на софтуер. С безброй много функции и поддръжка на хиляди плъгини, разработчиците рядко се нуждаят от друг инструмент, тъй като може да се разчита на Visual Studio през целия жизнен цикъл на разработка на приложениeто. 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 може да се използват 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 IT 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 други софтуери за RDBMS, Microsoft SQL Server е изграден върху SQL, стандартизиран език за програмиране, който администраторите на базите данни (DBA) и други IT специалисти използват за управление им.  **jQuery** е бърза, малка и богата на функции JavaScript библиотека. Работи като обхожда и манипулира HTML документа, обработва събития, анимации и **Ajax** заявки. С нейна помощ се улеснява работата с JavaScript- a. Библиотеката е подържана в множество браузъри.  **AJAX** (Asynchronous JavaScript And XML), не е език за програмиране. AJAX използва комбинация от вградени в браузъра обекти XMLHttpRequest (за заявка на данни от уеб сървър) и JavaScript и HTML DOM (за показване или използване на данните). AJAX позволява уеб страниците да се актуализират асинхронно чрез обмен на данни с уеб сървър зад кулисите. Това означава, че е възможно да се актуализират части от уеб страница, без да се презарежда цялата страница.  **Модел на базата данни- UML диаграма и Основни таблици**  За моделиране на обектите в Entity Framework е използван „Код първо“ подхода. Обектите са моделирани и конфигурирани 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 и трета нормална форма за изграждането на базата.    **Фигура 3.1** UML диаграма на базата данни  **Основните таблици:** AspNetUsers, AspNetRoles, AspNetUserTokens, AspNetUserLogins, Settings автоматично създадени от AspNet.  За 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  **Архитектура на проекта**  Solution- а на проекта е разделен в четири папки. В тях се съдържат подпроектите, които отговарят за данните, бизнес логиката, тестовете и уеб частта.    Папката **Data** съдържа три 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 четири 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 три проекта:  CONFIDENTIAL  **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 един проект и две библиотеки:  **GreenCap.Web** – Уеб проектът към който 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.    **3.2 Запознаване с потребителския интерфейс**  **Навигация, основни страници и потребителски интерфейс**  Началната страница (фигура 3.2) за потребители, които не са влезли в сайта, изглежда по следния начин. Дадени са права на потребителя да я прочете, да вижда предложения (proposals), събития (events), новини (news), формата за регистрация (register) и влизане (login) на потребители.    **Фигура 3.2.** Начална страница  Формата за регистрация (фигура 3.3) изисква от потребителя да въведе имейл адрес, парола и потвърждаване на паролата. Потребителското име на новорегистрирания човек е името от имейла му. За успешна регистрация имейла 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 (фигура 3.3) 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.    **Фигура 3.3.** Форма за регистрация и форма за влизане  Навигацията в сайта се извършва от панела за навигация (фигура 3.4).    **Фигура 3.4.** Панел за навигация  Влезлият в сайта потребител има достъп до всички страници. Следващите фигури ще бъдат разгледани 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).    **Фигура 3.5.** Начална страница за регистриран потребител  Панелът 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).    **Фигура 3.6.** Навигационно меню  Ще бъде разгледана по- подробно страницата за **статии**, която се казва **еко- предложения** (Proposals).  Първата опция от падащото меню, за преглед на всички еко-предложения, отвежда към страницата за креативни еко-идеи създадени от потребителите. На снимката са изобразени карти, които представляват различни споделени постове от потребители.    **Фигура 3.7.** Началнастраница с всички статии  Втората опция отвежда към прозорец за създаване на ново еко-предложение. За тестови цели създаваме статия за използване на метални сламки.    **Фигура 3.8.** Създаване на нова статия  Трета опция отвежда към страницата, където предложенията са създадени от текущо логнатия потребител. Както се вижда на снимката има една тема за метални сламки.    **Фигура 3.9.** Създадени от потребителя статии  От страницата за всички еко-предложения може да се види и статията публикувана от потребителя, която е видима за всички посетители на сайта, регистрирани и не регистрирани.    **Фигура 3.10.** Началната страница на статиите с еко-предложения  В долната част на картата виждаме, че сме създали на статията за метални сламки. Освен това, има и дата на създаване. Ако отворим дадена карта тя ни отвежда към страница с повече информация. За не регистрирани потребители тази страница дава опция само за връщане назад, както и цялата информация отнасяща се за статията.    **Фигура 3.11.** Индивидуална страница на статия  За регистрирани потребители в долната част на страницата се показват опции за редактиране, принтиране от pdf формат, изтриване на статията, само ако текущо- логнатият потребител е неин създател и изпращане на имейл под формата на pdf формат.    **Фигура 3.12.** Бутони за редактиране на статия  Опцията за принтиране отвежда на следната страница, където може да се зададат желаните опции за принтирането на статията.    **Фигура 3.13.** Принтиране в pdf формат на статия  Възможно е и оценяването на дадена статия, като пред потребителя се визуализира средната аритметична оценка на базата на гласувалите потребители.    **Фигура 3.14.** Оценяване на статия  Има възможност и за редактиране на статия само в случай, че ние сме я създали.    **Фигура 3.15.** Редактиране на статия  Форума на сайта дава възможност на всички регистрирани потребителя да задават въпроси.  Първата опция от падащото меню е да бъдат видени 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.    **Фигура 3.16.** Начална страница с всички постове във форума    **Фигура 3.17.** Единична тема във форума с част от съдържанието  За разлика от еко-предложенията тук в падащото меню имаме и категории.    **Фигура 3.18.** Падащоменю на раздел форум  Категории отвеждат към друга страница. На тази страница се вижда, че съществуват шест вида категории: основни, здраве, екология, спорт, околна среда и други.    **Фигура 3.19.** Страница на категориите във форума  Създаваме примерна тема във форума „Добре ли е да се закусва преди тренировка“. Избираме категория във форума да е „тренировки“ и използваме редактора на текст, за да украсим нашата тема. Редакторът е напълно функционален и включва в себе си най- известните опции като удебеляване, накланяне, оразмеряване, оцветяване, подравняване, подчертаване, добавяне на специални знаци, добавяне на код, добавяне на математически задачи, добавяне на таблици, хиперлинкове и много други.    **Фигура 3.20.** Създаване на нова статия във форума  Темата е налична в категория „упражнения“.    **Фигура 3.21.** Създадената тема в раздел упражнения  Отваряме темата, за да видим пълния текст. Имаме опциите за промяна на темата и изтриване, само ако сме създатели на дадента тема. Има и опция за коментиране.    **Фигура 3.22.** Темата във форума в отделен прозорец  Коментарите също имат възможност да се възползват от текстовия редактор. 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.    **Фигура 3.23.** Потребителски коментари  Последната ни опция е за даване на положителена или отрицателна оценка.    **Фигура 3.24.** Позитивни и отрицателни оценки  Страницата за създаване на събитие дава възможност да зададем име, URL път към снимка, начална и крайна дата и описание на събитието. Имаме опция за задаване на брой хора необходими за даденото събитие, като останалите потербители на по- късен етап могат да се запишат. Друга опция е добавянето на организатори. Организатор/и за дадено събитие може/могат да бъде/ат един или много потребители. Организаторът има право да изтрива или променя събитието, както и да се откаже от ролята си.    **Фигура 3.25.** Създаване на събитие  Събитието се визуализира 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.    **Фигура 3.26.** Изглед на събитие  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 посетен чрез линка „натисни тук“ и съдържанието на новините ще бъдат същите.    **Фигура 3.27.** Страница за новини  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 оригиналния новинарски сайт. Новините могат да се прочетат и от потребители, 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.    **Фигура 3.28.** Статия от новините  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.    **Фигура 3.29.** Промяна на потребителски данни  Администраторите в сайта имат разширен панел с падащо меню. От там могат да достъпят всички публикации в сайта, да ги редактират, изтриват или цензурират. Администраторите получават и буквата (A) до името си.    **Фигура 3.30.** Падащо меню наадминистраторския панел  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. Администраторът може да търси по ключова дума или да селектира броя на новините, които се показват на страницата.    **Фигура 3.31.** Администраторски изглед на новините  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.    **Фигура 3.32.** Навигация встраниците на сайта  GreenCap има политика за поверителност и съхранение на данните, която може да се достъпи от долната част на страницата през линка (privacy). Страницата „Политика“ е разделена на двадесет 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.    **Фигура 3.33.** Политика за сигурност  **Дизайн на сайта**  Използвана 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.    **Фигура 3.34.** Използвани форми    **Фигура 3.35.** Използвани форми  **Разглеждане на основните функционалности**  Всички модели в проекта GreenCap    **Потребителски форум - коментари и харесвания („Модел“)**  Моделите в потребителския форум се управляват от класа **PostService** и се имплементират интерфейса IPostService. Асинхронните методи на класа завършват с наставката Async. Някои 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- съществува 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 и 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 е 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 три 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); 2. int GetLikes(int postId); 3. int GetDisslikes(int postId); 4. public int GetDisslikes(int postId) 5. { 6. return this.likesDb 7. .All() 8. .Where(x => x.PostId == postId) 9. .Where(x => !x.IsPositive) 10. .Count(); 11. } 12. public int GetLikes(int postId) 13. { 14. return this.likesDb 15. .All() 16. .Where(x => x.PostId == postId) 17. .Where(x => x.IsPositive) 18. .Count(); 19. } 20. public async Task SetLikeAsync(int postId, string userId, bool isPositive) 21. { 22. var like = this.likesDb.All() 23. .FirstOrDefault(x => x.PostId == postId && x.UserId == userId); 25. if (like == null) 26. { 27. like = new UserLike 28. { 29. PostId = postId, 30. UserId = userId, 31. }; 33. await this.likesDb.AddAsync(like); 34. } 36. like.IsPositive = isPositive; 37. await this.likesDb.SaveChangesAsync(); 38. }   **Коментарите** имат три 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 като приема четири 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 логнатия 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 логнатия 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**  Контролерът за оценки и харесвания е малко по- различен от останалите. За разлика от обикновените контролери ApiControllers са специализирани във връщането на данни и се грижат за прозрачно сериализиране на данните във формат, поискан от клиента. Освен това, 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 на събития и набиране на хора “Страници”**  „Модела“ и „Контролера“ отговарящи за събитията имат аналогична структура и начин на работа. За раздел Събития ще разгледаме само кода за визуализирането на потребителския интерфейс. „Страниците“ съдържат HTML&CSS, JavaScript, JQuery и използват технологията Razor. Разширението на файловете е .cshtml, защото комбинира CSS и HTML.    Събитията са разделени в четири файла, всеки от които е страница в сайта.  Страницата **All** импортира модела, потребителските данни, модела за визуализиране, dependency injection за логнатият потребител и модела за визуализирането на данните.   1. @using GreenCap.Data.Models 2. @using Microsoft.AspNetCore.Identity 3. @using GreenCap.Web.ViewModels.OutputViewModel 4. @inject SignInManager<ApplicationUser> SignInManager 5. @model EventsListOutputViewModel   Чрез синтаксиса задаваме заглавие на страницата.   1. @{ this.ViewData["Title"] = "All events " + Model.PageNumber; }   След това се възползваме от възможностите на Razor страниците, като проверяваме дали съществува поне една страница.   1. @if (Model.PageNumber == 1) 2. { 3. <div class="jumbotron bg-dark"> 4. <h1 class="display-3 text-primary text-center">Take a part in events happening around you!</h1> 5. </div>}   Използва се foreach цикъл за визуализиране на всички събития и данните подадени от контролера за визуализиране на всяко едно събитие. В тялото на цикъла има няколко if проверки: за броя хора нужни за събитието и дали текущо логнатият потребител е създател на това събитие. В различните html тагове се използват boostrap ключови думи за оформление. Пример: “bg-gradient-dark” придава тъмно син цвят на контейнера, „text-center“ центрира текста, „text- primary“ прави текста с по- голям шрифт. Към всяка една от специалните думи в класовете се прилага и съответния css. Могат да се използват и функции от C# като Math.abs().   1. @foreach (var eventModel in this.Model.Events) 2. { 3. <div class="card mb-3 text-center bg-gradient-dark"> 4. <h3 class="card-header display-3 text-primary">@eventModel.Name</h3> 5. <div class="card-body"> 6. <h6 class="card-subtitle text-muted">Event duration: @eventModel.StartDate - @eventModel.EndDate</h6> 7. </div> 8. <div class="container"> 9. <img class="card-img-top" src=@eventModel.ImagePath alt="No available image." /> 10. </div> 11. <div class="card-body"> 12. <p class="card-text text-light">@eventModel.Description</p> 13. </div> 14. <ul class="list-group list-group-flush text-left text-info"> 15. <li class="list-group-item text-center">Total needed people: @eventModel.TotalPeople</li> 16. <div class="container bg-gradient-info"> 17. <p class="float-right text-dark">100%</p> 18. <p class="text-dark">0%</p> 19. <div class="progress"> 20. <div class="progress-bar progress-bar-striped progress-bar-animated bg-primary" role="progressbar" style="width: @((eventModel.TotalPeople-eventModel.NeededPeople)\*(100/eventModel.TotalPeople))%" aria-valuenow="@((eventModel.TotalPeople-eventModel.NeededPeople)\*(100/eventModel.TotalPeople))" aria-valuemin="0" aria-valuemax="100"></div> 21. </div> 22. @if (eventModel.NeededPeople >= 0) 23. { 24. <p class="text-center text-dark">@eventModel.NeededPeople left</p> } 25. else 26. { 27. <p class="text-center text-dark">@Math.Abs(eventModel.NeededPeople) extra</p> 28. } 29. </div> 30. <li class="list-group-item">Hosted by: <p class="text-muted">@eventModel.HostedByNames</p></li> 31. <li class="list-group-item">Sign in users: <p class="text-muted">@eventModel.SignedInByNames </p> </li> 32. </ul> 33. @if (this.SignInManager.IsSignedIn(this.User)) 34. { 35. <div class="card-body" id="@eventModel.Id"> 36. @if (!eventModel.SignedInByNames.Contains(this.User.Identity.Name.Split('@', StringSplitOptions.RemoveEmptyEntries)[0])) 37. { 38. <a class="btn btn-success btn-lg float-left m-2" href="/Events/Join?id=@eventModel.Id" role="button">Join event</a> 39. } 40. else 41. { 42. <a class="btn btn-warning btn-lg float-left m-2" href="/Events/Cancel?id=@eventModel.Id" role="button">Cancel attending</a> 43. } 44. @if (eventModel.HostedByNames.Contains(this.User.Identity.Name.Split('@', StringSplitOptions.RemoveEmptyEntries)[0])) 45. { 46. <form method="post" class=" m-1" asp-controller="Events" asp-action="Delete" id="deleteForm" asp-route-id="@eventModel.Id"> 47. <button class="btn btn-danger btn-lg m-1 float-right" data-toggle="modal" data-target="#deleteModal">Delete</button> 48. </form> 49. <a class="btn btn-secondary btn-lg m-1 float-right" href="/Events/Edit?id=@eventModel.Id" role="button">Edit</a> 50. } 51. </div> 52. } 53. <div class="card-footer text-muted text-left"> 54. @eventModel.CretedDaysAgo 55. </div> 56. </div>}   Накрая на html зарежда и частта от страницата за пейджиране, чрез partial елемент.   1. <partial name="\_PagingPartial" model="@Model" />   Втората страница е за създаване на събитие. В нея има Post форма за създаване на събитие. Полетата изискват от потребителя въвеждане на име на събитие, път към снимка от интернет, начална и крайна дата, брой нужни хора, създатели на събитието и описание на събитието. Използвани са ASP.Net вградените тагове за валидиране и свързване към ViewModel- лите. Примерни тагове са „asp-for="Name““ отговарящо на полето „name“ в класа EventInputViewModel. ASP.Net позволява чрез атрибути в този клас да се валидира полето автоматично, още при попълване от потребителя и да изкара грешка отговаряща на некоректно попълнените данни.   1. <form method="post"> 2. <fieldset> 3. <div class="form-group"> 4. <label asp-for="Name"></label> 5. <input asp-for="Name" type="text" class="form-control col-4" placeholder="Event name"> 6. <span asp-validation-for="Name" class="text-danger"></span> 7. </div> 9. <div class="form-group"> 10. <label asp-for="ImagePath"></label> 11. <input asp-for="ImagePath" type="text" class="form-control col-md-4" placeholder="Image URL"> 12. <span asp-validation-for="ImagePath" class="text-danger"></span> 13. </div> 15. <div class="form-group"> 16. <label asp-for="StartDate"></label> 17. <input asp-for="StartDate" class="form-control col-md-4"> 18. <span asp-validation-for="StartDate" class="text-danger"></span> 19. </div> 21. <div class="form-group"> 22. <label asp-for="EndDate"></label> 23. <input asp-for="EndDate" class="form-control col-md-4"> 24. <span asp-validation-for="EndDate" class="text-danger"></span> 25. </div> 27. <div class="form-group"> 28. <label asp-for="TotalPeople">Number</label> 29. <input asp-for="TotalPeople" class="form-control col-md-4" type="number" max="100" min="0" value="1"> 30. </div> 32. <div class="form-group"> 33. <label asp-for="CreatorsNames"></label> 34. <textarea asp-for="CreatorsNames" class="form-control" rows="2" placeholder="Separate with space, comma or slash other hosts names "></textarea> 35. <span asp-validation-for="CreatorsNames" class="text-danger"></span> 36. </div> 38. <div class="form-group"> 39. <label asp-for="Description"></label> 40. <textarea asp-for="Description" class="form-control" rows="3" placeholder="Share in details about the event."></textarea> 41. <span asp-validation-for="Description" class="text-danger"></span> 42. </div> 44. <button type="submit" class="btn btn-info">Submit</button> 45. </fieldset> 46. </form>   Последният клас, за Edit страницата е подобен на другите два. Всички страници се възползват от силата на ASP.Net, Razor pages и boostrap, за да визуализират страниците.  Особени са оценките в еко- предложенията и харесванията на постове във форума. Те се изпълняват чрез следния код. Когато бутона е натиснат id- то на поста е изпратено към JavaScript файл, който слуша за натискането на този бутон.   1. <button type="button" class="btn btn-danger fas fa-thumbs-down float-right p-xl-2 m-1" data-vote="0"> <span id="disslikesValue\_@(post.Id)">@post.DissLikes</span></button> 2. <button type="button" class="btn btn-success fas fa-thumbs-up float-right p-xl-2 m-1" data-vote="1"> <span id="likesValue\_@(post.Id)">@post.Likes</span></button>   JavaScript се зарежда по следният начин.   1. @section Scripts 2. { 3. <script src="~/js/LikesPost.js"></script> 4. }   Може да намери файла в статичните файлове на проекта, намиращи се в папката wwwroot/js/LikePosts.js. Там се намира следният код, който изпраща post заявка към специалния ApiController, който е разгледан по- рано. Това се осъществява чрез jQuery и AJAX, които позволяват уеб страниците да се актуализират асинхронно чрез обмен на данни с уеб сървъра. Като резултат се актуализира частта от уеб страница отговорна за промяна на броя харесвания, без да се презарежда цялата страница. По подобен начин са направени и оценките в страницата за еко- предложения. Тази пост заявка е възможно да бъде подменена злонамерено, използва се antiForgeryToken.   1. $("button[data-vote]").each(function (el) { 2. $(this).click(function () { 4. var IsPositive = Boolean(parseInt($(this).attr("data-vote"))); 5. var postId = parseInt(this.parentElement.getAttribute("id")); 7. var antiForgeryToken = $('#antiForgeryForm input[name=\_\_RequestVerificationToken]').val(); 8. var data = { postId: postId, IsPositive: IsPositive }; 10. $.ajax({ 11. type: "POST", 12. url: "/api/Likes", 13. data: JSON.stringify(data), 14. headers: { 15. 'X-CSRF-TOKEN': antiForgeryToken 16. }, 17. success: function (data) { 18. $('#disslikesValue\_' + postId).html(data.dissLikes); 19. $('#likesValue\_' + postId).html(data.likes); 20. }, 21. contentType: 'application/json', 22. }); 23. }) 24. });   **Новини – извличане на данни от други сайтове**  Извличането на данни за страницата новини се осъществява от класа PhysNewsScraperService, намиращ се в библиотеката GreenCap.Services.    Класа има четири метода, като първите два са подобни на методите от другите services. Те служат за импортиране на новините от базата данни и създава нова категория, ако не съществува такава.   1. public async Task ImportNewsAsync(int countNews) 2. private async Task<int> GetOrCreateCategoryAsync(string categoryName) 3. private ConcurrentBag<NewsDto> ScraperNews(int countNews) 4. private NewsDto GetNews(int countNews)   Методът **ScraperNews** асинхронно взима новините от другия сайт. След това асинхронно ги добавя в колекция и ги връща за визуализиране от страницата за новини.   1. private ConcurrentBag<NewsDto> ScraperNews(int countNews) 2. { 3. var concurrentBag = new ConcurrentBag<NewsDto>(); 5. // count news 6. Parallel.For(0, countNews, i => 7. { 8. try 9. { 10. var news = this.GetNews(i); 12. concurrentBag.Add(news); 13. } 14. catch 15. { 16. // ignored 17. } 18. }); 20. return concurrentBag; 21. }   Методът GetNews отваря сайта асинхронно и взима всички резултати.   1. var url = string.Format(BaseUrl, currentPage); 3. var document = this.context 4. .OpenAsync(url) 5. .GetAwaiter() 6. .GetResult();   След това нужната информация се изтегля от DOM дървото на другия сайт, обработва се, за да е подходяща за нашите модели и се запазва в обекти на нашия клас.   1. Var shortNews = new NewsShortIntroDTO(); 2. var mainNews = new NewsDto(); 4. var x = document.GetElementsByClassName("sorted-article")[newsNumber]; 6. shortNews.MainPageUrl = x.GetElementsByClassName("sorted-article-figure")[0] 7. .GetElementsByTagName("a")[0] 8. .GetAttribute("href"); 10. shortNews.SmallPhotoUrl = x.GetElementsByClassName("sorted-article-figure")[0] 11. .GetElementsByTagName("img")[0] 12. .GetAttribute("data-src"); 14. shortNews.Title = x.GetElementsByClassName("sorted-article-content")[0] 15. .GetElementsByTagName("a")[0] 16. .TextContent; 18. shortNews.Summary = x.GetElementsByClassName("sorted-article-content")[0] 19. .GetElementsByTagName("p")[0] 20. .TextContent 21. .Trim(); 23. shortNews.CategoryName = x.GetElementsByClassName("article\_\_info")[0] 24. .GetElementsByTagName("p")[0] 25. .TextContent 26. .Trim(); 28. var date = x.GetElementsByClassName("article\_\_info")[0] 29. .GetElementsByTagName("p")[1] 30. .TextContent 31. .Trim();   **Иползвани технологии и инструменти**   * Visual Studio 2022 * ASP.NET 6 MVC * ASP.NET areas * MS SQL Server * MS SQL Server Management Studio 2018 * ORM Entity Framework *6* * SendGrid API * HighChart * HTML 5 & CSS * Bootstrap 4.0 * HtmlSanitizer * StyleCop Analyzers -Theme by bootswatch * Sonar lint- static code analysis * Custom database seeder * Automapper * Web Api controllers + AJAX real-time Requests * jQuery and any kind of jQuery plugins * AJAX   **Използвана литература на български език**   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 език**   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/)  [freeCodeCamp.org - YouTube](https://www.youtube.com/c/Freecodecamp)  [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/) |  |
|  | |  |