****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   + **Запознаване с потребителския интерфейс**     - Навигация и дизайн     - Основни страници на сайта   + **Разглеждане на основните функционалности**     - Регистрация и вход на потребители     - Потребителски форум, публикуване на статии, създаване на събития, коментари и оценки, статистически данни     - Извличане и визуализиране на новини * **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 др.    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**  **Други важни характеристики на езика**  **<Т>, тернарен оператор, филтри**  **Глава 3. Среда за разработка и архитектура на проекта**  **V**isual **S**tudio (VS) is an integrated development environment (IDE) from Microsoft used to build applications and websites that support a wide range of programming languages such as C, C++, C#, or Visual Basic. Currently, in version 2014, Visual Studio is the **most complex tool**available that meets all the requirements for fast, productive, and error-, prone software development. With countless numbers of features and plugin support, you rarely need any other tool as you can rely on Visual Studio in the entire application development life cycle. It comes in various editions such as Express, Professional or Ultimate that might slightly differ across the versions.  .NET  .NET is a free, cross-platform, open-source developer platform for building many different types of applications.  With .NET, you can use multiple languages, editors, and libraries to build for web, mobile, desktop, games, IoT, and more. | | |  |
|  |  |

**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**

[C# / Java / JavaScript / Python / C++ Programming Books (introprogramming.info)](https://introprogramming.info/)

[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/)

**Приложение**

Код на програмата от софтуерната реализация: