SocioBrains

ISSN 2367-5721 (online), JOURNAL HOMEPAGE: WWW.SOCIOBRAINS.COM
Publisher: SMART IDEAS – WISE DECISIONS, Ltd., Sofia, Bulgaria

ISSUE 92, APRIL 2022

WEB DEVELOPMENT WITH MICROSOFT .NET BLAZOR

Abstract: The Microsoft .NET platform is one of the most popular for developing web applications. Recently it introduced a new technology, called Blazor, which allows developers to create modern and rich web applications using entirely the C# programming language. The article describes Blazor and compares and analyzes its projects supported by the .NET platform.

Author information:

Vladimir Sulov
Professor, PhD
University of Economics – Varna
✉ vsulov@ue-varna.bg
Bulgaria

Keywords:
web development, .NET platform, Blazor

Въведение

Платформата .NET на Microsoft включва фреймуърк[1] за изпълнение на код, специализирани сървъри, средства за разработка и езици за програмиране [7, 15].

.NET и фреймуъркът .NET Framework са обявени през 2000 г., а първата версия е предложена на пазара през 2002 г. [6]. В следващите години платформата включва възможности за разработка на приложения за десктоп, уеб, мобилни устройства, игри, услуги, за Interenet of Things (IoT) – устройства, инструменти за работа и хостване в облак, за изкуствен интелект и др. [3, 4, 10].

По отношение на разработването на уеб приложения, през 2014 г. Microsoft обявяват, а през 2016 г. реализират нов фреймуърк – .NET Core, с цел по-добра оптимизация и мултиплатформеност [5]. През 2020 г. .NET Framework и .NET Core се обединяват и използват общото кратко наименование .NET, като се поддържат както класическите, така и новите възможности, а понастоящем платформата е една от най-използваните при разработката на уеб приложения [13, 14].

Паралелно, през 2018 г. Microsoft въвеждат нова технология при разработката на уеб приложения, наречена Blazor, чрез която да създадат възможността разработчиците да не учат и използват JavaScript и JavaScript-базирани фреймуърци, а да разчитат изцяло на езика за програмиране C#. Технологията предлага два основни вида, различаващи се в немалка степен, проекта: Blazor Server App и Blazor WebAssembly App [8, 12]. Индивидуалните разработчици и компаниите следва да познават техните особеностите, за да са в състояние да правят подходящ избор при изграждане на нови приложения или трансформиране на съществуващи.

На тази основа, цел на настоящата статия е да се представи технологията Blazor и да се анализират и сравнят основните видове Blazor проекти за уеб приложения, поддържани от платформата .NET.

[1] Понятието framework се среща на български език в специализираната литература като „рамка“, „среда“ или „платформа“, но в настоящата статия е предпочетено използването му като заемка – „фреймуърк“, за разграничаване.

Технология Blazor

Основен език за програмиране на бизнес логиката в уеб приложнията в платформата .NET, още от нейното създаване, е C#. В по-редки случаи за някои от проектите се ползва още Visual BASIC (макар че поддръжката му е по-скоро за съвместимост с по-стари приложения) или F# (който пък е по-подходящ за специфичен код, нуждаещ се от език за функционално програмиране). По тази причина, разработчиците на .NET познават и ползват C#.

Първоначално съществуващите проекти (понастоящем наречени ASP.NET Web Application) могат да се създават и без познаването и използването на JavaScript или други JavaScript-базирани фреймуърци. Взаимодействието между интерфейсните елементи (макар и в известна степен ограничено) също може да се реализира на C#, но този тип проект има редица недостатъци.

С въвеждането на .NET Core, на разработчиците се налага да използват JavaScript за взаимодействие между интерфейсните елементи, а за някои от видовете проекти – дори и доста сложните фреймуърци Angular или React [2]. Това поставя твърде разнородни изисквания към служителите и евентуална необходимост от усвояване на нов и нетипичен за платформата .NET инструментариум.

По тази причина Microsoft разработват Blazor, за да направят възможно взаимодействието и между интерфейсните елементи на езика за програмиране C# при използване на по-съвременния тип проекти на .NET Core и съответно тяхното разработване без познаването на JavaScript или други JavaScript-базирани фреймуърци.

Кодът на типична уеб страница, създадена с технологията Blazor, е представен на фиг. 1.

```
1   @page "/counter"
2   @page "/counter/{startingValue:int}"
3
4   <PageTitle>Counter</PageTitle>
5
6   <h1>Counter</h1>
7
8   <p role="status">Current count: @currentCount</p>
9
10  <button class="btn btn-primary" @onclick="IncrementCount">Click me</button>
11
12  @code {
13      private int currentCount = 0 ;
14
15      [Parameter]
16      public int IncreaseBy { get; set; } = 1;
17      [Parameter]
18      public int StartingValue { get; set; } = 0;
19
20      protected override void OnParametersSet()
21      {
22          currentCount = StartingValue;
23          base.OnParametersSet();
24      }
25
26      private void IncrementCount()
27      {
28          currentCount+=IncreaseBy;
29      }
30  }
```

Фиг. 1. Код на уеб страница, създадена с Blazor

Източник: собствена разработка

Като цяло страницата съдържа HTML код (тагове и съдържание), но интегрирани с други .NET тагове и директиви и програмен код на C#. Изграждането на изгледа по този начин е въз основа на технологията Razor на Microsoft.

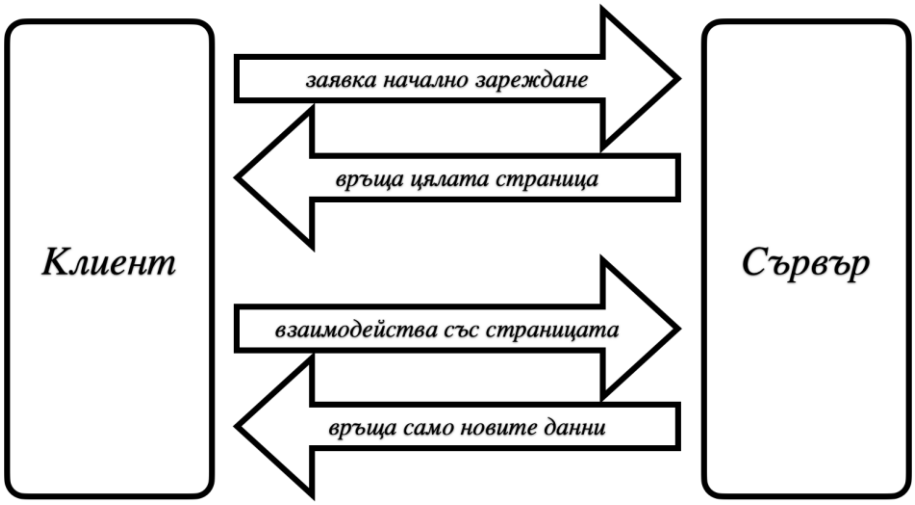
По отношение на взаимодействието с интерфейсните елементи могат да бъдат отбелязани няколко момента:

- в страниците може да съществува блок с код на C#, вкл. с променливи и методи (редове 12-30);
- на страницата може да бъде предаван параметър (при преход от друга страница, редове 2, 17-18);
- може да се създава връзка между интерфейсни елементи и променливи на C# (редове 8, 13);
- може да се създават методи на C#, които да се изпълняват при взаимодействие на потребителя с интерфейса (редове 10, 26-29).

Особеноси на Blazor проектите

Както беше отбелязано, технологията предлага два основни вида проекта: Blazor Server App и Blazor WebAssembly App. Разликите между тях са основно по отношение на взаимодействието между клиента (браузъра на потребителя) и сървъра [1].

На фиг. 2 е показано взаимодействието между клиента и сървъра при Blazor Server App.

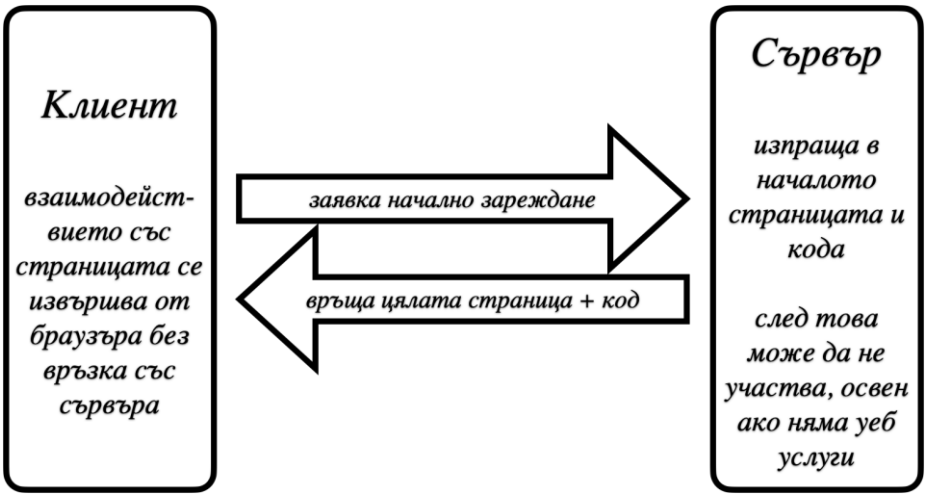


Фиг. 2. Взаимодействие между клиент и сървър при Blazor Server App приложенията
Източник: собствена разработка

Началната заявка от клиента води до изпращане на цялата страница от страна на сървъра. Когато потребителят взаимодейства със страницата, при което възниква нужда от промяна на съдържанието или свойствата на интерфейсните елементи, това предизвиква нова комуникация със сървъра, който изпраща актуализирани данни или част от страницата, които съответно се използват за нейното обновяване. Сходна е ситуацията и ако приложението се нуждае от изпращане и/или получаване на друга информация, например за запис или извличане от сървърна база от данни.

В резултат на този подход може да се отбележи, че сървърът и мрежовата връзка са относително по-силно натоварени, поради нуждата от комуникация и обработка при почти всяко действие на потребителя. От друга страна, браузърът е разтоварен, защото в него не се извършва почти никаква обработка.

На фиг. 3 е показано взаимодействието между клиента и сървъра при Blazor WebAssembly App, което се извършва по коренно различен начин.



Фиг. 3. Взаимодействие между клиент и сървър при Blazor WebAssembly App приложенията

Източник: собствена разработка

Началната заявка от клиента води до изпращане на цялата страница от страна на сървъра, но също така и на целия необходим код за работа на страницата. Това е възможно благодарение на технологията WebAssembly, при която кодът, написан от разработчика, се транслира и после компилира до вариант, който се изпълнява изцяло от браузъра на клиента без необходимост от връзка със сървъра след първоначалното зареждане [9].

Подходът натоварва браузъра и съответно устройството на клиента повече, но разтоварва в много голяма степен сървъра и мрежата. След първоначалното зареждане на страницата, реално обмен по мрежата и съответно действия от страна на сървъра не се извършват до евентуално зареждане на друга страница.

В същото време, редица дейности не биха били възможни, ако последваща връзка със сървъра липсва – напр. използване на база от данни, която се съхранява на сървъра. В подобни ситуации е възможно да се създадат уеб услуги и интерфейс към тях, работещи на сървъра, към които клиентското приложение да се обръща при необходимост и да изпраща и получава данни, да заявява допълнителна сървърна обработка и др. [11].

Сравнението на двата вида Blazor проекта по отношение на натоварването на сървъра, мрежовата връзка и клиента е представено на табл. 1.

ТАБЛИЦА 1
СРАВНЕНИЕ МЕЖДУ BLAZOR ПРОЕКТИТЕ

Тип проект	**натоварване на сървъра**	**натоварване на мрежовата връзка**	**натоварване на клиента**
Blazor Server App	високо	високо	ниско
Blazor WebAssembly App	ниско	ниско	високо

Заключение

Разгледаните особености на съществуващите видове проекти за разработката на уеб приложения в рамките на технологията Blazor на платформата .NET показват, че те притежават сходни изисквания по отношение на познанията на разработчиците, но различни характеристики по отношение на взаимодействието и натоварването на сървъра, мрежовата връзка и клиента.

Софтуерните компании и индивидуалните разработчици следва да познават особеностите на Blazor и въз основа на анализ на нуждите при изграждане на уеб приложения, да избират подходящ тип проект за всяка конкретна ситуация.

References:

1. Aponte, M. 2020. Blazor Server vs. Blazor WebAssembly. In: Building Single Page Applications in .NET Core 3. Apress.
2. Damir, A. 2022. Developing Web Applications in .NET (Different Approaches and Current State). Dot Net Curry. https://www.dotnetcurry.com/aspnet-core/1501/web-development-in-dotnet. Accessed: 10.03.2022.
3. Esposito, D. 2008. Programming ASP.NET 3.5. Microsoft Press.
4. Garcia, D. 2019. The History of ASP.NET – Part II (Covers ASP.NET MVC). Dot Net Curry. https://www.dotnetcurry.com/aspnet/1493/aspnet-history-part-2-mvc. Accessed: 9.03.2022.
5. Garcia, D. 2019. The History of ASP.NET – Part III (Covers ASP.NET Core). Dot Net Curry. https://www.dotnetcurry.com/aspnet/1494/aspnet-history-part-3-core. Accessed: 9.03.2022.
6. Linnik, I. 2019. A Brief History of .NET Framework. SoftTeco. https://softteco.com/blog/a-brief-history-of-net-framework. Accessed: 10.03.2022.
7. Prosise, J. 2002. Programming Microsoft .NET. Microsoft Press.
8. Roth, D et al. 2022. Blazor for ASP.NET Web Forms Developers. Microsoft Press.
9. Šipek, M et al. 2021. Next-generation Web Applications with WebAssembly and TruffleWasm. 44th International Convention on Information, Communication and Electronic Technology (MIPRO), pp. 1695-1700.
10. Sulov, V. 2014. Implementation of Programming Languages on the Platform .NET at Developing Software Applications. Izvestia. Journal of the University of Economics – Varna. Issue 1. pp. 13-22.
11. Whitesell, S., Richardson, R. and Groves, M. 2022. Pro Microservices in .NET 6. Apress.
12. .NET Documentation. 2022. Microsoft. https://docs.microsoft.com/bg-bg/dotnet/?WT.mc_id=dotnet-35129-website. Accessed: 4.03.2022.
13. Most used web frameworks among developers worldwide, as of 2021. 2022. Statista. https://www.statista.com/statistics/1124699/worldwide-developer-survey-most-used-frameworks-web/. Accessed: 10.03.2022.
14. Web Application Frameworks. 2022. Web Tech Survey. https://webtechsurvey.com/technology-type/web-application-frameworks. Accessed: 10.03.2022.
15. What is .NET? 2022. Microsoft. https://dotnet.microsoft.com/en-us/learn/dotnet/what-is-dotnet. Accessed: 11.03.2022.