Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 4.9
DOI: https://doi.org/10.24867/33BE12Vasic

RAZVOJ SAVREMENIH VEB APLIKACIJA U .NET EKOSISTEMU PRIMENOM *WEBASSEMBLY* I *BLAZOR* TEHNOLOGIJA

DEVELOPMENT OF MODERN WEB APPLICATIONS IN THE .NET ECOSYSTEM THROUGH THE USE OF WEBASSEMBLY AND BLAZOR TECHNOLOGIES

Nataša Vasić, *Fakultet tehničkih nauka, Novi Sad*

Oblast – ELEKTROTEHNIČKO I RAČUNARSKO INŽENJERSTVO

Kratak sadržaj – *U ovom radu prikazano je istraživanje WebAssembly i Blazor tehnologija, sa ciljem da se predstave njihovi osnovni koncepti, prednosti i praktične primene. WebAssembly bajtkod može se izvršavati u različitim okruženjima, ali u ovom radu fokus je na .NET okviru zbog njegovih širokih mogućnosti i popularnosti među programerima. Dat je pregled mogućnosti koje ove tehnologije pružaju uz doprinos boljem razumevanju njihove primene u savremenoj industriji.*

Ključne reči: *WebAssembly, Blazor, .NET, veb tehnologije, veb aplikacija*

Abstract – *This paper presents a study of WebAssembly and Blazor technologies, aiming to introduce their fundamental concepts, advantages and practical applications. WebAssembly bytecode can be executed in various environments, however this paper focuses on its use within the .NET ecosystem due to its extensive capabilities and popularity among developers. The paper provides an overview of the opportunities these technologies offer and contributes to a better understanding of their application in modern industry.*

Keywords: *WebAssembly, Blazor, .NET, web technologies, web application*

1. UVOD

Razvoj veb tehnologija omogućio je pojavu kompleksnih aplikacija kao što su igrice, različiti audio i video softveri i slične aplikacije, te zahtevi za efikasnošću i sigurnošću rastu. JavaScript, HTML i CSS i dalje dominiraju u razvoju veb aplikacija, ali zbog svojih ograničenja ne mogu u potpunosti da ispune te zahteve. WebAssembly, otvoreni standard koji omogućava izvršavanje binarnog koda direktno u pregledaču, uspeo je da ispuni zahteve [1, 2]. U okviru .NET Blazor tehnologije WebAssembly se koristi kao ključna tehnologija za pokretanje aplikacija u veb pregledačima. Blazor je UI framework koji omogućava razvoj interaktivnih veb aplikacija koristeći C# i .NET umesto korišćenja JavaScript jezika.

NAPOMENA:
Ovaj rad proistekao je iz master rada čiji mentor je bila dr Dunja Vrbaški, docent

Blazor omogućava programerima da koriste poznate alate i biblioteke iz .NET okruženja. Ovaj pristup eliminiše potrebu za dva programera, jednog za JavaScript, a drugog za backend u klasičnom veb projektu, povećava produktivnost i omogućava kreiranje aplikacija koje mogu da se pokrenu na svim modernim pregledačima [3].

2. WEBASSEMBLY

WebAssembly (Wasm) je tip koda koji se može izvršavati u modernim veb pregledačima. To je jezik niskog nivoa sličan asembleru koji koristi kompaktan binarni format i omogućava izvršavanje sa performansama bliskim nativnim. Pruža mogućnost kompajliranja programskih jezika kao što su C/C++, C#, Rust i drugi u oblik pogodan za izvršavanje unutar veb pregledača. Dizajniran je da funkcioniše uporedo sa JavaScript jezikom, omogućavajući im da rade zajedno [4]. Razvijen je u saradnji četiri glavne kompanije koje se bave razvojem pregledača – Google, Microsoft, Mozilla i Apple [1]. Prema W3C organizaciji, WebAssembly je četvrti jezik za veb koji omogućava izvršavanje koda u pregledaču [5]. JavaScript, HTML i CSS su ostala tri jezika [2]. WebAssembly je dizajniran da bude kompaktan, brz i siguran, a da pritom omogućava validaciju, kompilaciju i bezbedno izvršavanje uz minimalne troškove. Nezavisan je od programskih jezika, hardvera i platforme. Od samog početka dizajniran je sa formalnom semantikom [1].

2.1. Primene

Google, eBay i Norton implementirali su WebAssembly u svoje projekte kako bi unapredili performanse. Primeri primene uključuju čitače bar kodova [7], mašinsko učenje sa TensorFlow.js bibliotekom [8], prepoznavanje obrazaca, grafičke alate, kompresiju podataka, kriptografske biblioteke, igrice, obradu slika, numeričke proračune i druge specijalizovane zadatke [6]. Jednostavnost i univerzalnost podstakla je njegovu primenu u različitim domenima, uključujući serversku stranu u kombinaciji sa Node.js, serverless računarstvo u oblaku, Internet stvari (IoT) i integrisane uređaje, pa čak i kao samostalno okruženje za izvršavanje [9]. Unutar pregledača koristi se i za uređivanje slika i video sadržaja, podržava različite vrste igrica, razvoj aplikacija, prepoznavanje slika, VPN i drugo. Izvan pregledača koristi se za distribuciju računarskih igara, izvršavanje nepoverljivog koda na serveru, server-side aplikacije i slično [10].

2.2. Razlozi za razvoj novih veb tehnologija

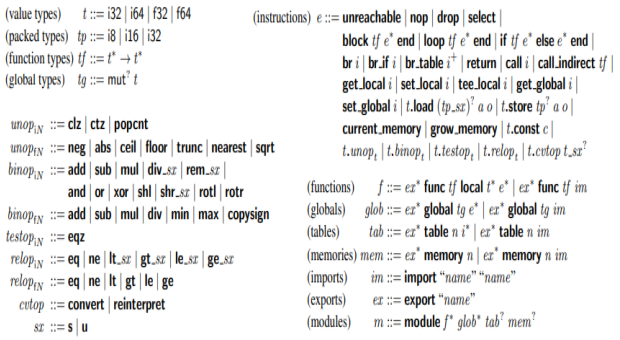
JavaScript je dugo bio jedina opcija za kreiranje interaktivnih aplikacija u pregledaču. Međutim, razvoj aplikacija koje zahtevaju visoke performanse procesora, kao što su igre, bio je ograničen zbog slabih performansi. Da bi se rešili problemi napravljeni su brojni pokušaji da se prednosti nativnog koda prenesu na veb. Adobe je promovisao Flash platformu, Microsoft je predložio ActiveX, Google uvodi Native Client tehnologiju [11].

2.3. Prethodna rešenja

ActiveX kojeg je uveo Microsoft, Flash platforma koju je promovisao Adobe, Native Client kojeg je predložio Google i asm.js, prvi su pokušali da reše izazove sigurnog, brzog i prenosivog koda niskog nivoa. Međutim, to nisu u potpunosti uspeli jer je svaki od njih zahtevao dodatne plugin-ove i bio vezan za specifičan pregledač. Ove tehnologije često su imale problema sa sigurnošću, kompatibilnošću i performansama što je ograničavalo njihovu upotrebu [1, 11].

2.4. Struktura i funkcionalnost binarnih fajlova

Binarni format koda definisan je tako da može da se posmatra kao jezik sa sintaksom i strukturom. Ovaj format olakšava razumevanje, a da pritom ne ugrožava kompaktnu formu ili jednostavnost dekodiranja. Struktura u smislu apstraktne sintakse može se videti na Slici 1 [1].



Slika 1. *WebAssembly pravila sintakse [1]*

Neki od osnovnih koncepata koje WebAssembly koristi su moduli, funkcije, instrukcije, lokalne varijable, globalne varijable i tako dalje [1].

Moduli su binarni fajlovi koji sadrže funkcije, globalne promenljive, tabele i memoriju. Sastoje se od različitih tipova sekcija kojih ima ukupno 11, od kojih su četiri najvažnije: sekcija koda, sekcija podataka i sekcije importa i eksporta. Dok moduli predstavljaju statičku strukturu, instanca modula omogućava dinamičko izvršavanje sa memorijom i stekom. Instanciranje modula obezbeđuje okruženje kao što je JavaScript VM ili operativni sistem. Sekcija koda predstavlja najveću sekciju obuhvatajući sve funkcije modula. Slika 2 prikazuje primer funkcije iz ove sekcije, koja uključuje osnovne operacije poput kontrolnih naredbi, oduzimanja i množenja [1, 11].

C program code	Binary	Text representation
	20 00	get_local 0
	42 00	i64.const 0
	51	i64.eq
int factorial(int n) {	04 7e	if i64
if (n == 0)	42 01	i64.const 1
return 1	05	else
else	20 00	get_local 1
return n * factorial(n-1)	20 00	get_local 1
}	42 01	i64.const 1
	7d	i64.sub
	10 00	call 0
	7e	i64.mul
	0b	end

Slika 1. *Jednostavna C funkcija sa leve strane i odgovarajući WebAssembly bajtkod zajedno sa tekstualnom reprezentacijom poznatom kao Wat format sa desne strane [11]*

Kod u modulu organizovan je u funkcije koje primaju vrednosti kao parametre i vraćaju vrednosti kao rezultate, prema definisanom tipu funkcije. Funkcije mogu međusobno pozivati jedna drugu, uključujući rekurziju, ali ne mogu biti ugnježdene [1].

Izvršavanje operacija u WebAssembly tehnologiji bazira se na stek mašini. Funkcije se sastoje od instrukcija koje manipulišu vrednostima na steku. Sistem sa tipom omogućava statičko određivanje rasporeda steka, što omogućava direktnu kompilaciju tokova podataka bez stvarnog materijalizovanja steka. Ova organizacija steka omogućava kompaktno predstavljanje programa [1].

Neke instrukcije mogu izazvati izuzetke koji odmah prekidaju izvršavanje. WebAssembly kod ne može obraditi izuzetke direktno, ali ih JavaScript okruženje može obraditi. WebAssembly izuzetak će generisati (eng. throw) JavaScript izuzetak koji sadrži trag steka (eng. stacktrace) sa JavaScript i WebAssembly stekom. Trag steka se može uhvatiti i pregledati uz pomoć JavaScript koda [1].

U funkcijama, lokalne promenljive se inicijalizuju na nulu i njima se upravlja pomoću instrukcija get_local i set_local. Instrukcija tee_local omogućava upis u lokalnu promenljivu dok ulazna vrednost ostaje na steku [1].

Moduli mogu deklarisati globalne promenljive koje se čitaju i pišu pomoću instrukcija get_global i set_global. Globalne promenljive mogu biti promenljive ili nepromenljive i moraju imati početnu vrednost koja je konstantan izraz [1].

WebAssembly koristi linearnu memoriju koja predstavlja veliki niz bajtova. Svaki modul može definisati samo jednu memoriju koja se može deliti između različitih instanci putem uvoza/izvoza. Memorija se kreira sa određenom veličinom, ali se može dinamički proširivati pomoću instrukcije za povećanje memorije. Usled nedostatka memorije, proširenje neće uspeti i biće vraćena vrednost -1 kao signal neuspeha. Trenutna veličina memorije može se proveriti korišćenjem instrukcije za ispitivanje memorijskog stanja. WebAssembly memorija definisana je da koristi little-endian redosled bajtova, što znači da platforme sa big-endian redosledom zahtevaju eksplicitne konverzije endian redosleda. Ove konverzije mogu biti optimizovane od strane WebAssembly kompajlera [1].

2.5. Arhitektura

WebAssembly je dizajniran tako da ne definiše način na koji se programi učitavaju u izvršno okruženje, niti kako se obavljaju I/O operacije. Ovakva arhitektura omogućava fleksibilnu integraciju WebAssembly tehnologije u različita izvršna okruženja. Sistem koji implementira WebAssembly preuzima odgovornost za učitavanje modula, povezivanje uvoznih i izvoznih funkcija, obezbeđivanje pristupa I/O operacijama i tajmerima, kao i rukovanje izuzecima [1].

2.6. Performanse

Kompaktni binarni format omogućava brzo učitavanje i dekodiranje. Analiza je dovela do sledećih zaključaka:

• WebAssembly kompajleri uglavnom su zasnovani na LLVM-u, gde optimizacije nisu specifično prilagođene za WebAssembly.

• JIT optimizacije značajno utiču na performanse JavaScript jezika, dok za WebAssembly nema značajne razlike u performansama između verzija sa i bez JIT-a.

• Performanse JavaScript-a i WebAssembly-ja variraju u zavisnosti od pregledača i platforme. Na desktop računarima Firefox pruža bolje rezultate u izvršavanju WebAssembly koda u odnosu na Chrome, dok Edge postiže najlošije rezultate. Nasuprot tome, na mobilnim uređajima, Firefox je sporiji od Chrome-a, dok Edge nadmašuje oba pregledača. Performanse JavaScript jezika takođe variraju u zavisnosti od platforme. Tako je na desktop računarima Firefox sporiji u odnosu na Chrome, dok je na mobilnim uređajima brži.

• WebAssembly zahteva više memorije u poređenju sa JavaScript jezikom. Ovo se može pripisati činjenici da WebAssembly koristi linearni model memorije koji ne oslobađa memoriju automatski, dok JavaScript koristi sakupljanje smeća za automatsko oslobađanje memorije [6].

Rezultati pokazuju da iako se očekuje da WebAssembly bude brži od JavaScript jezika, to nije uvek slučaj i performanse mogu značajno varirati. Kada WebAssembly želi da pristupi ili manipuliše DOM-om, mora da zatraži od JavaScript-a da izvrši te operacije. Ova upotreba uvodi dodatno opterećenje, što može smanjiti performanse WebAssembly koda. WebAssembly se iz tog razloga koristi za zadatke koji zahtevaju intenzivno računanje [12]. JavaScript često postiže bolje rezultate u odnosu na WebAssembly, naročito ako je ulaz u program velik. Prilikom instanciranja WebAssembly modula, veliki deo linearne memorije se inicijalizuje kako bi se simulirale memorijske lokacije. Kada se linearna memorija potpuno popuni, umesto oslobađanja memorije koja više nije u upotrebi, ona se proširuje na veću veličinu. Nasuprot tome, JavaScript koristi automatsko upravljanje memorijom, koje dinamički prati alokaciju memorije i oslobađa onu koja više nije potrebna. Ova dinamika čini JavaScript efikasnijim u korišćenju memorije u poređenju sa WebAssembly tehnologijom [6].

Još jedno od ključnih poboljšanja koje WebAssembly nudi odnosi se na energetsku efikasnost, koja je u proseku poboljšana za 30%. Iako JavaScript u nekim situacijama daje bolje rezultate, opšte je prihvaćeno da je WebAssembly ne samo brži od njega nego i koristi manje energije, što ga čini boljim izborom za razvoj aplikacija [2].

3. BLAZOR

Blazor je .NET frontend veb framework koji omogućava kreiranje interaktivnih korisničkih interfejsa korišćenjem programskog jezika C#, uz mogućnost deljenja aplikativne logike između serverske i klijentske strane. Renderovanje korisničkog interfejsa ostvaruje primenom HTML i CSS koda, čime se omogućava široka podrška za različite pregledače, uključujući i one na mobilnim uređajima. Blazor pruža značajne prednosti kao što su pisanje koda u C#, korišćenje postojećeg .NET ekosistema i integraciju sa savremenim alatima kao što su Docker, Visual Studio i Visual Studio Code, što doprinosi produktivnosti i sigurnosti u procesu razvoja veb aplikacija [13].

Blazor aplikacije zasnivaju se na komponentama, koje predstavljaju osnovne jedinice korisničkog interfejsa, poput stranica, dijaloga ili formi za unos podataka. Komponente su klase implementirane u C# jeziku koje omogućavaju fleksibilno definisanje logike za prikaz korisničkog interfejsa, upravljanje događajima, ugnježdavanje i ponovno korišćenje. Ove komponente mogu se deliti i distribuirati putem Razor biblioteka ili NuGet paketa. Komponente se pišu u formi Razor stranica sa ekstenzijom .razor i koriste Razor sintaksu koja kombinuje HTML i C# kod. Ovaj pristup doprinosi većoj produktivnosti i omogućava efikasnije programiranje unutar integrisanih razvojnih okruženja kao što je Visual Studio. U formalnom kontekstu ove komponente se nazivaju Razor komponente, dok se neformalno često nazivaju Blazor komponente [13].

3.1. Tipovi

Blazor pruža različite tipove implementacije, među kojima su server-side, client-side i hosted. Svaki od ovih tipova dolazi sa specifičnim šablonima koji su dostupni unutar Visual Studio okruženja. Nije moguće odrediti koji tip je najbolji, budući da izbor zavisi od zahteva i potreba projekta [3].

Server-side Blazor omogućava izvršavanje celokupne aplikacione logike na serverskoj strani, koristeći WebSocket tehnologiju za uspostavljanje komunikacije između klijenta i servera. Prednost ovog pristupa leži u mogućnosti pisanja frontend logike u programskom jeziku C#, čime se pojednostavljuje razvoj aplikacije. Međutim, ključni nedostatak ovog tipa je eliminacija potrebe za API pozivima, jer se sve potrebne biblioteke direktno integrišu u frontend, što može dovesti do smanjene efikasnosti ovog rešenja [3].

Client-side Blazor funkcioniše isključivo na klijentskoj strani unutar pregledača. Iako su stranice host-ovane na serveru, sva logika se izvršava na klijentu. Ova opcija je posebno pogodna za prezentacione veb sajtove ili jednostavne veb aplikacije, ali može postati neefikasna u situacijama kada je potrebna interakcija sa bazama podataka ili kada aplikacija već koristi postojeće API servise [3].

Hosted Blazor predstavlja najefikasnije rešenje, gde se logika aplikacije izvršava na klijentskoj strani, čime se optimizuje korišćenje resursa servera. Ovaj pristup

integriše client-side Blazor sa posebnim API projektom, omogućavajući im da funkcionišu zajedno kao jedna celina. Ova kombinacija pruža optimalno rešenje za aplikacije koje zahtevaju intenzivnu interakciju sa serverom [3].

3.2. Blazor WebAssembly Hosted projekat – struktura i sastavni delovi

Blazor WebAssembly šablon automatski kreira inicijalne fajlove i strukuru direktorijuma prilikom generisanja početnog projekta, uključujući demonstracioni kod koji služi kao primer implementacije osnovnih funkcionalnosti. Struktura projekta sastoji se od tri glavna segmenta: klijentski deo (Client), serverskog dela (Server) i deljenih resursa (Shared). Ova struktura omogućava jasno razdvajanje poslovne logike aplikacije od korisničkog interfejsa, pri čemu se zajednički kod održava u okviru posebnog modula kako bi se olakšala njegova ponovna upotreba i konzistentnost između klijentskog i serverskog dela aplikacije.

3.3. Opšti pregled

Blazor pruža širok spektar funkcionalnosti, ali i dalje postoje situacije u kojima je neophodno koristiti JavaScript. Blazor omogućava jednostavnu i efikasnu integraciju sa JavaScript kodom, olakšavajući pristupanje skladištu podataka, rad sa fajlovima i korišćenje postojećih JavaScript biblioteka. Interakcija sa JavaScript kodom u Blazor aplikacijama ostvaruje se putem IJSRuntime interfejsa koji se injektuje u odgovarajuću stranicu. HTML elementi podržavaju širok spektar događaja, od kojih su neki od njih generički, a drugi specifični za određene elemente. Ovi događaji se mogu koristiti direktno u Blazor-u bez potrebe za interakcijama sa JavaScript kodom [3].

4. ZAKLJUČAK

Rezultati ovog rada potvrdili su važnost integracije WebAssembly i Blazor tehnologija u modernom razvoju veb aplikacija. Implementacija WebAssembly tehnologije omogućava visok nivo performansi i fleksibilnosti u aplikacijama, omogućavajući izvršavanje koda na strani klijenta mnogo brže nego tradicionalni pristup zasnovan na JavaScript jeziku. Blazor koristeći C# jezik pojednostavljuje razvoj, pružajući programerima mogućnost da koriste poznat ekosistem i alate i smanjuje potrebu za korišćenjem više programskih jezika. Korišćenje ovih tehnologija dovelo je do ubrzanja razvoja aplikacija, poboljšanja održavanja koda i smanjenja upotrebe JavaScript koda.

5. LITERATURA

[1] A. Haas, A. Rossberg, D. L. Schuff, B. L. Titzer, M. Holman, D. Gohman, L. Wagner, A. Zakai, and J. F. Bastien, „Bringing the Web up to Speed with WebAssembly“, In Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation, pp. 185-200, 2017.

[2] J. De Macedo, R. Abreu, R. Pereira, and J. Saraiva, „WebAssembly versus JavaScript: Energy and Runtime Performance“, In 2022 International Conference on ICT for Sustainability (ICT4S), pp. 24-34, 2022.

[3] T. Litvinavicius, Exploring Blazor: Creating Hosted, Server-side, and Client-side Applications with C#, 1st ed., 2019.

[4] Mozilla Developer Network, „WebAssembly“, https://developer.mozilla.org/en-US/docs/WebAssembly, (pristupljeno u septembru 2024.)

[5] World Wide Web Consortium, „World Wide Web Consortium (W3C) brings a new language to the Web as WebAssembly becomes a W3C Recommendation“, https://www.w3.org/press-releases/2019/wasm/, (pristupljeno u septembru 2024.)

[6] Y. Yan, T. Tu, L. Zhao, Y. Zhou, and W. Wang, „Understanding the Performance of WebAssembly Applications“, In Proceedings of the 21st ACM Internet Measurement Conference, pp. 533-549, 2021.

[7] S. Padmanabhan, and P. Jha, „WebAssembly at eBay: A Real-World Use Case“, https://innovation.ebayinc.com/tech/engineering/webassembly-at-ebay-a-real-world-use-case/, (pristupljeno u oktobru 2024.)

[8] D. Smilkov, N. Thorat, and A. Yuan, „Introducing the WebAssembly backend for TensorFlow.js“, https://blog.tensorflow.org/2020/03/introducing-webassembly-backend-for-tensorflow-js.html, (pristupljeno u oktobru 2024.)

[9] D.Lehmann, J. Kinder, and M. Pradel, „Everything Old is New Again: Binary Security of WebAssembly“, In 29th USENIX Security Symposium (USENIX Security 20), pp. 217-234, 2020.

[10] WebAssembly, „WebAssembly“, https://webassembly.org/, (pristupljeno u septembru 2024.)

[11] M. Musch, C. Wressnegger, M. Johns, and K. Rieck, „New Kid on the Web: A Study on the Prevalence of WebAssembly in the Wild“, In Detection of Intrusions and Malware, and Vulnerability Assessment: 16th International Conference, DIMVA 2019, Gothenburg, Sweden, Springer International Publishing, pp. 23-42, 2019.

[12] D. Kievits, „What effect does applying WebAssembly have on a compute intensive client-side application versus JavaScript?“, 2021.

[13] Microsoft, „ASP.NET Core Blazor“, https://learn.microsoft.com/en-us/aspnet/core/blazor/?view=aspnetcore-8.0, (pristupljeno u avgutu 2024.)

Kratka biografija:

Nataša Vasić rođena je u Novom Sadu 1999. god. Master rad na Fakultetu tehničkih nauka iz oblasti Elektrotehnike i računarstva odbranila je 2025.god.
kontakt: natasavas00@gmail.com