

SVEUČILIŠTE JURJA DOBRILE U PULI
FAKULTET INFORMATIKE

Tin Pritišanac

Strategije iscrtavanja web aplikacija kroz različite programske okvire

ZAVRŠNI RAD

Pula, rujan, 2025. godine

SVEUČILIŠTE JURJA DOBRILE U PULI
FAKULTET INFORMATIKE

Tin Pritišanac

Strategije iscrtavanja web aplikacija kroz različite programske okvire

ZAVRŠNI RAD

**JMBAG: 0171256219, izvanredni student
Studijski smjer: Informatika**

**Kolegij: Web aplikacije
Znanstveno područje : Društvene znanosti
Znanstveno polje : Informacijske znanosti
Znanstvena grana : Informacijski sustavi i informatologija**

Mentor: doc. dr. sc. Nikola Tanković

Pula, rujan, 2025. godine

Sažetak

Odabir optimalne strategije iscrtavanja web aplikacije ključan je za postizanje dobrih performansi i korisničkog iskustva. Ovaj rad usporedno analizira četiri strategije iscrtavanja (CSR, SSR, SSG, ISR) u tri popularna programska okvira za izradu web aplikacija (Next.js, Nuxt.js i SvelteKit). U svakom od programske okvira implementirana je identična demo aplikacija sa stranicama statičnog i dinamičnog sadržaja, te su na njima vršeni testovi ključnih metrika performansi prikupljeni Lighthouse CLI alatom u kontroliranim mrežnim uvjetima. Analiza rezultata otkriva prednosti i mane svake strategije iscrtavanja ovisno o tipu stranice, te koji programski okviri postižu najbolje performanse u kojim strategijama iscrtavanja.

Ključne riječi : CSR, SSR, SSG, ISR, Next.js, Nuxt.js, SvelteKit, performanse

Abstract

Selecting the optimal rendering strategy for a web application is crucial for achieving good performance and user experience. This paper provides a comparative analysis of four rendering strategies (CSR, SSR, SSG, ISR) across three popular web application frameworks (Next.js, Nuxt.js, and SvelteKit). An identical demo application containing both static and dynamic content pages was implemented in each framework, and key performance metrics were tested using the Lighthouse CLI tool under controlled network conditions. The results highlight the advantages and disadvantages of each rendering strategy depending on the page type, as well as which frameworks achieve the best performance under specific rendering strategies.

Keywords : CSR, SSR, SSG, ISR, Next.js, Nuxt.js, SvelteKit, performance

Sadržaj

1 Strategije iscrtavanja	1
1.1 Iscrtavanje na strani klijenta (CSR)	2
1.2 Iscrtavanje na strani poslužitelja (SSR)	4
1.3 Generiranje statičkih stranica (SSG)	5
1.4 Inkrementalna statička generacija (ISR)	6
2 Metodologija	9
2.1 Opis referentne aplikacije	9
2.2 Implementacija u programskim okvirima	9
2.3 Mjerene metrike Performansi	10
2.4 Iscrtavanje najvećeg sadržaja (LCP)	10
2.5 Ukupno vrijeme blokiranja (TBT)	10
2.6 Kumulativna promjena rasporeda (CLS)	10
2.7 Vrijeme do interaktivnosti (TTI)	11
2.8 Vrijeme do prvog bajta (TTFB)	11
2.9 Dodatne metrike	11
2.10 Alati i testno okruženje	11
2.11 Lighthouse	11
2.12 Chrome DevTools	12
2.13 Lighthouse reporter	12
2.14 Testno okruženje	14
3 Rezultati testiranja	16
3.1 Rezultati testiranja stranice "O nama"	17
3.2 Sažetak rezultata - stranica "O nama"	21
3.3 Rezultati testiranja stranice Blog	22
3.4 Sažetak rezultata - stranica Blog	26
3.5 Rezultati testiranja stranice pojedinog blog posta	27
3.6 Sažetak rezultata - stranica pojedinog blog posta	31
3.7 Dodatne metrike	32
4 Analiza rezultata	37
5 Zaključak	38
Literatura	39
Popis slika	41
Popis tablica	42

1 Strategije iscrtavanja

U današnjoj industriji razvoja web aplikacija, više nego ikada prije, izražena je težnja za postizanjem sve većih performansi, skalabilnosti i boljeg korisničkog iskustva. Ta potreba potaknula je istraživanje i razvoj novih rješenja kao odgovor na pitanje kako najoptimalnije i najbrže dostaviti web sadržaj korisniku. Važni čimbenici u odgovoru na ovo pitanje su upravo odabir odgovarajuće strategije iscrtavanja web aplikacije s obzirom na vrstu sadržaja, te programski okvir na kojem se temelji aplikacija. Web aplikacije uvelike su evoluirale od starog pristupa serviranja statičnih stranica do današnjih vrlo interaktivnih dinamičnih aplikacija koje zahtijevaju sofisticirane pristupe dostavljanja i prikazivanja sadržaja krajnjem korisniku [1]. Tradicionalne metode iscrtavanja poput iscrtavanja na serveru (SSR) i generiranja statičnih stranica (SSG) već duže vrijeme čine temelj i osnovni standard u razvoju web aplikacija. Povećanje kompleksnosti aplikacija i potreba za sve višim performansama rezultirali su i nastankom novih hibridnih metoda iscrtavanja poput inkrementalne statičke generacije (ISR).

Svaka od ovih metoda ima specifične prednosti i nedostatke koji se odražavaju na metrike poput početnog vremena učitavanja sadržaja, razine interaktivnosti, optimizacije za tražilice (SEO) i dr. Odabir strategije iscrtavanja uvelike utječe na percepciju krajnjeg korisnika, troškove infrastrukture te kompleksnost u razvoju i arhitekturi aplikacije.

Popularni programski okviri za razvoj web aplikacija kao što su Next.js, Nuxt i SvelteKit podržavaju većinu novih strategija iscrtavanja čime otvaraju programerima mnoge mogućnosti za optimiziranje procesa dostave sadržaja korisniku.

Pri odabiru programskog okvira uzimaju se u obzir mnogi faktori, a jedan od njih su i performanse, koje direktno utječu na korisničko iskustvo pogotovo u ograničenim mrežnim uvjetima. U ovom radu provedena je usporedna analiza performansi sva tri programska okvira u svakoj od četiri odabrane strategije iscrtavanja (CSR, SSR, SSG i ISR). Postavljeni ciljevi su:

1. Izrada i implementacija funkcionalno i stilski identične web aplikacije s podstranicama dinamičnog i statičnog sadržaja (blog).
2. Sustavno mjerjenje i usporedba odabralih metrika performansi što uključuje: vrijeme do prvog bajta (TTFB), prvo iscrtavanje sadržaja (FCP), iscrtavanje najvećeg sadržaja (LCP), ukupno vrijeme blokiranja (TBT), vrijeme do interaktivnosti (TTI), veličinu paketa (engl. bundle size) i vrijeme izgradnje (engl. build time) za svaku kombinaciju programskog okvira i strategije iscrtavanja.
3. Utvrđivanje prednosti i nedostataka svake strategije iscrtavanja unutar svakog programskog okvira analizom prikupljenih podataka.
4. Na temelju rezultata analize donijeti zaključke o tome koji programski okvir nudi najbolje performanse s obzirom na odabranu strategiju iscrtavanja.

Kako bi rezultati bili reprezentativni i usporedivi sa svakodnevnim korisničkim iskustvom, sva mjerena izmjerena su u stvarnim uvjetima na aplikacijama postavljenima na platformu Vercel, koja u trenutku pisanja ovog rada jedina nudi mogućnost postavljanja sve četiri strategije iscrtavanja u svim navedenim programskim okvirima. Za testiranje je osigurana stabilna i brza internetska veza, no sami testovi provedeni su uz postavljena ograničenja kako bi se jasnije istaknule razlike u performansama. Odabранo je ograničenje veze koje simulira sporu 4G vezu te dvostruko smanjenje brzine procesora.

Ovo istraživanje nastoji pružiti bolji uvid u trenutne mogućnosti iscrtavanja koje su dostupne web developerima, te im time pomoći u donošenju boljih odluka pri odabiru programskog okvira i strategije iscrtavanja za svoju web aplikaciju. Prvi dio rada baviti će se pregledom najpopularnijih strategija iscrtavanja i programske okvira, a drugi dio prezentacijom i analizom dobivenih rezultata, raspravom te zaključnim razmatranjima.

Pojam strategije iscrtavanja odnosi se na proces pretvaranja koda u vizualni sadržaj koji korisnik može vidjeti i s kojim može vršiti interakciju kroz web preglednik [1]. Ovaj proces utječe na korisničko iskustvo određujući koliko brzo će se sadržaj prikazati korisniku i koliko će aplikacija biti prilagodljiva i dinamična. Odabir strategije iscrtavanja također bitno utječe i na SEO tj. vidljivost kod pretraživanja internetskim tražilicama [2].

Moderne SPA aplikacije najčešće podržavaju različite strategije iscrtavanja, a određene platforme poput Vercela i Netlifyja pružaju dodatnu podršku popularnim programskim okvirima, olakšavajući njihovu konfiguraciju i integraciju. Slijedi kratak pregled strategija iscrtavanja obrađenih u ovom radu.

1.1 Iscrtavanje na strani klijenta (CSR)

Ova metoda iscrtavanja nastala je još početkom 21. stoljeća razvojem programskih okvira poput AngularJS-a, Reacta i Vuea kada je nastao veliki prijelaz u industriji sa monolitne web arhitekture na tzv. jednostranične aplikacije (SPA) koje se za ažuriranje sučelja, navigaciju i dohvaćanje podataka oslanjaju na JavaScript kod koji se izvršava u pregledniku.

Kod ove strategije iscrtavanja, na klijent se šalje jedan prazan HTML dokument, zajedno sa svim drugim resursima (CSS i JavaScript paketi). Umjesto da je sav sadržaj već unesen u HTML dokument i odmah spreman za prikazivanje u pregledniku, preglednik najprije mora pričekati da se preuzme sav potreban JavaScript kod, te se njegovim izvršavanjem HTML dokument ispunjava elementima koji će se prikazati. Kod navigacije između stranica, ne dolazi do dohvaćanja novog HTML dokumenta, već JavaScript ažurira postojeći HTML novim sadržajem oslanjajući se na AJAX i XML. Time se izbjegavaju ponovna učitavanja koja usporavaju rad aplikacije i štede mrežne resurse [3].

Prednosti ove strategije su:

- Responzivnost i interaktivnost – promjene na stranici vidljive su odmah, nema potrebe za dohvaćanjem nove stranice prilikom navigacije.
- Ušteda resursa poslužitelja – umjesto dohvaćanja cijelog novog HTML dokumenta za svaku stranicu, dohvaća se samo onaj dio podataka koji je potreban (najčešće u JSON formatu)
- Mogućnost korištenja aplikacije kada mrežna veza nije dostupna (offline) – ovakva funkcionalnost postiže se pametnim predmemoriranjem (engl. caching).
- Uvijek svježi podaci – budući da stranice nisu prethodno statički generirane, osigurava se svježina prikazanih podataka.

Nedostaci ove strategije su:

- Početna brzina učitavanja – iako je aplikacija generalno brža nakon početnog učitavanja JavaScripta, prvo učitavanje kada korisnik posjeti stranicu može biti znatno sporije zbog potrebe da se učita i izvrši sav potreban JavaScript kod prije iscrtavanja sadržaja. Ovo ograničenje posebno dolazi do izražaja prilikom učitavanja na sporijim mrežama.
- Lošiji SEO – iako današnji pretraživači¹ mogu očitati i stranice iscrtane ovom strategijom, primarno su optimizirani su za čitanje statičnog HTML-a. Zbog ovog nedostatka, ova se strategija se sve manje koristi za web aplikacije kod kojih je važan dobar SEO.
- Lošije performanse na slabijim uređajima – zbog činjenice da je potrebno najprije izvršiti JavaScript kod na strani klijenta, kod slabijih i starijih uređaja može doći do usporavanja učitavanja i pada performansi.
- Manjak podrške za korisnike koji nisu omogućili JavaScript u pregledniku – bez JavaScripta sadržaj stranice se ne može iscrtati.

S obzirom na navedene prednosti i nedostatke ova strategija iscrtavanja najbolja je za tipove aplikacija koje imaju slijedeća obilježja:

- potreba za visokom razinom interaktivnosti (dashboards)
- SEO nije bitan čimbenik (admin i korisničke stranice)
- progresivne web aplikacije
- potreba za smanjenjem opterećenja poslužitelja

¹Eng. crawler – automatizirani robot koji tražilica koristi za istraživanje, pronađak i indeksiranje web stranica [4]

1.2 IsCRTavanje na strani poslužitelja (SSR)

Kako bi se uklonili nedostatci koje sa sobom nosi iscrtavanje na strani klijenta, razvila se nova popularna strategija iscrtavanja – iscrtavanje na strani poslužitelja (engl. server). Ova strategija se naizgled vraća korak prema dobro poznatom i prvobitnom načinu iscrtavanja – iscrtavanju na poslužitelju i višestrajničnoj web aplikaciji (MPA) kakve su postojale od začetaka Interneta.

No moderni programski okviri poput Nuxta i SvelteKita spajaju SSR i CSR na način da se prilikom prvobitnog posjeta stranici ona iscrtava na poslužitelju, te se korisniku šalje već popunjena HTML dokument koji preglednik može odmah prikazati, a u pozadini se događa proces zvan hidracija. Ovo je proces u kojem se učitava sav potreban JavaScript kod koji se izvršava i povezuje sa HTML elemenima stranice te čini stranicu interaktivnom, slično kao kod CSR strategije. Svaki slijedeći zahtjev za navigaciju između stranica ili ažuriranje podataka događa se na strani klijenta i funkcioniра kao CSR aplikacija.

Next.js programski okvir u novijim verzijama koristi tzv. iscrtavanje na strani poslužitelja u stvarnom vremenu (Streaming SSR). U ovom načinu rada, na serveru se postupno iscrtavaju zasebni dijelovi HTML dokumenta po redoslijedu kako podaci postaju dostupni i takav dokument se odmah šalje pregledniku na prikaz. Ovime se nastoji izbjegći nedostatak klasičnog SSR-a a to je čekanje na poslužitelj da generira cijeli HTML dokument prije nego ga pošalje klijentu [5].

Kako bi SSR strategija mogla funkcioniрати, na poslužitelju je potrebno odgovarajuće okruženje koje podržava izvođenje JavaScript koda i generiranje HTML stranica. To je najčešće Node.js [6].

Prednosti ove strategije su:

- Nema čekanja na učitavanje JavaScripta – budući da se na zahtjev klijenta na poslužitelju iscrtava HTML datoteka koja mu se odmah dostavlja spremna za prikaz, nema potrebe za čekanjem na preuzimanje i izvršavanje JavaScript koda.
- Bolji SEO – statičke HTML datoteke popunjene sadržajem mnogo su bolje za mrežne pretraživače, koji iz njih brže i lakše dolaze do relevantnih podataka o stranici.
- HTML datoteke mogu se spremiti u pričuvnu memoriju preglednika, što omogućava pregled stranice i kada nema pristupa internetu.
- Svježina podataka – na svaki zahtjev klijenta generira se novi HTML dokument sa ažuriranim podacima.

Nedostaci ove strategije su:

- Čekanje na poslužitelju – iako nema čekanja na izvršavanje JavaScript koda (prije prikaza sadržaja) na strani klijenta, prilikom prve posjete stranici potrebno je pričekati na poslužitelja da iscrta HTML dokument koji nije pretvodno generiran. Prethodno spomenuti streaming SSR pokušava ublažiti i ovaj nedostatak.
- Čekanje do interaktivnosti – unatoč brzom prikazu početnog sadržaja stranice, ipak je potrebno pričekati na proces hidracije kako bi stranica postala interaktivna. To uključuje preuzimanje JavaScript paketa i izvršavanje koda što može blokirati glavnu procesorsku nit i time povećati vrijeme do interaktivnosti (TTI).
- Povećano korištenje resursa poslužitelja – generiranje HTML dokumenata na svaki zahtjev korisnika rezultira i većom potrošnjom računalnih resursa poput memorije i procesorske snage, što je kod CSR-a prebačeno na uređaj klijenta.
- Složeniji proces razvoja – zbog činjenice da se određeni dijelovi koda mogu izvršavati samo na klijentu ili samo na poslužitelju, raste i kompleksnost u razvoju aplikacije [3].

S obzirom na navedene prednosti i nedostatke ova strategija iscrtavanja najbolja je za tipove aplikacija koje imaju slijedeća obilježja [1]:

- stalna potreba za svježim i ažuriranim podacima
- visoka razina personalizacije (custom dashboards)
- vizualizacija podataka u realnom vremenu
- dobar SEO

1.3 Generiranje statičkih stranica (SSG)

Kod ove strategije iscrtavanja stranice se unaprijed iscrtavaju na poslužitelju prilikom izgradnje aplikacije, te se zatim poslužuju klijentu na zahtjev, bez potrebe za ponovnim iscrtavanjem pri svakom zahtjevu kao kod SSR-a. Ovakve stranice se mogu i spremiti u priručnu memoriju (engl. cache) koristeći CDN radi brže distribucije korisnicima [7], [8].

Glavne prednosti ove strategije su [1]:

- Najbrže moguće učitavanje stranice – budući da su stranice statični HTML, nije potrebno čekati na njihovo iscrtavanje na poslužitelju ili na preuzimanje i izvršavanje JavaScript koda.
- Odlične su za SEO – budući da se brzo učitavanju i sadrže sav potreban sadržaj idealne su za pregled od strane pretraživača.

- Nisko opterećenje poslužitelja – nema potrebe za obradom ili iscrtavanjem HTML-a, već je spreman za slanje klijentu.
- Najniži troškovi infrastrukture - manjak potrebe za velikim računalnim resursima.

Nedostatci ove strategije su:

- Duže vrijeme izgradnje ako postoji veliki broj stranica.
- Za ažuriranje sadržaja potrebno je ponovno inicirati izgradnju i postavljanje (engl. build and deploy).
- Nije pogodno za stranice sa dinamičnim i često promjenjivim sadržajem.

S obzirom na navedene prednosti i nedostatke ova strategija iscrtavanja najbolja je za stranice kod kojih se sadržaj gotovo nikada ili vrlo rijetko mijenja poput:

- marketinških stranica
- blog postova
- e-commerce proizvoda
- dokumentacije i pomoći

Generalno pravilo je postaviti pitanje: može li se stranica iscrtati unaprijed, prije korisničkog zahtjeva? Ako je odgovor potvrđan, SSG se nameće kao logičan izbor [7].

Postoji mogućnost kombiniranja SSG-a i CSR-a gdje se stranica servira sa prethodno iscrtanim statičnim dijelom koji se ne mijenja, a po učitavanju JavaScripta na klijentu se ispunjavaju dinamični dijelovi stranice svježim podacima. Ovo je alternativa korištenju SSR strategije, koja također ima svoje prednosti i nedostatke [7].

1.4 Inkrementalna statička generacija (ISR)

Ovu strategiju iscrtavanja neki nazivaju i hibridnim web razvojem jer kombinira generiranje statičnih stranica (SSG) sa iscrtavanjem na strani poslužitelja (SSR).

Funkcionira na način da se najprije iscrtava statična stranica kojoj se odredi vrijeme revalidacije, tj. vrijeme nakon kojega će se ona smatrati zastarjelom i biti potrebno ponovno iscrtati. No okidač za novo iscrtavanje biti će zahtjev prvog posjetitelja nakon vremena isteka revalidacije. Tom prvom posjetitelju će se odmah isporučiti stara verzija stranice bez obzira koliko je dugo prekoračeno njeno vrijeme validacije, te će se nakon uspješnog ponovnog iscrtavanja u pozadini, nova verzija stranice poslužiti prvom slijedećem korisniku. Na ovaj način novo generirana stranica se dodaje u web aplikaciju.

Uobičajeni način postavljanja na poslužitelj (engl. deployment) jest atomsko postavljanje, tj. kada se cjelokupni kod, resursi i konfiguracija ažuriraju u isto vrijeme. Ovaj način postavljanja čuva integritet stranice i omogućava jednostavan opoziv i povratak na prijašnju verziju u slučaju potrebe (engl. rollback). Svako pojedinačno postavljanje je jedna cjelevita verzija web aplikacije. ISR strategija narušava ovaj integritet budući da stalno nadodaje novo generirane stranice u web aplikaciju, odvojeno od početno izgrađenog koda. Zbog ovog spremanja u pričuvnu memoriju (engl. caching) je vrlo teško učiniti povratak na prethodnu verziju, a pri tome osigurati da svi korisnici dobiju istu verziju stranice [9].

Potrebno je naglasiti da se ova strategija oslanja na mrežu za isporuku sadržaja (CDN) koju pruža platforma na koju je postavljena web aplikacija. Platforma poput Vercela također olakšava cijeli proces konfiguracije i postavljanja aplikacije. Ovu metodu moguće je implementirati i neovisno o platformi, ali to podiže kompleksnost.

Implementacija ove strategije uvelike se razlikuje među programskim okvirima. Npr. Next.js generira statične stranice za dinamične rute prilikom vremena izgradnje aplikacije (build time) te ih obnavlja na prvu posjetu korisnika nakon isteka vremena revalidacije. Nuxt pak generira traženu stranicu tek na prvi zahtjev korisnika a ne prilikom izgradnje aplikacije [10].

Prednosti ove strategije su:

- visoke performanse slične SSG strategiji – za većinu korisnika koji dobiju prethodno spremljenu stranicu putem CDN-a
- mogućnost periodičnog ažuriranja stranica novim sadržajem bez potrebe za ponovnom izgradnjom čitave aplikacije
- niži trošak od SSR-a
- efektivno skaliranje do velikog broja stranica
- odličan SEO

Nedostatci ove strategije su [9]:

- korisnici koji prvi posjeti stranicu nakon isteka perioda revalidacije dobivaju zastarjelu verziju stranice
- potrebna veća razina konfiguriranja – postavljanje optimalnog vremena revalidacije na svaki tip stranice posebno
- složenije debugiranje – teže razumjevanje problema koji nastanu zbog spremanja u priručnu memoriju



Slika 1: Pregled strategija iscrtavanja u Next.js programskom okviru [11]

2 Metodologija

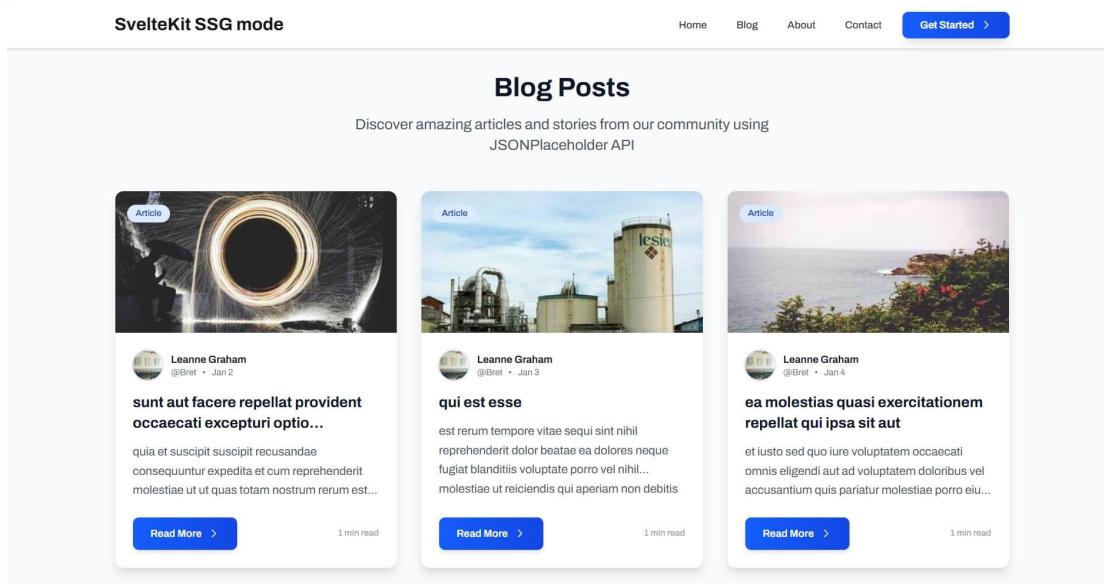
2.1 Opis referentne aplikacije

Za potrebe testiranja performansi različitih programskih okvira kroz različite strategije iscrtavanja bilo je potrebno napraviti demo aplikaciju koja bi bila reprezentativna, ali ne presložena, te bi uključivala stranice sa statičnim i dinamičnim sadržajem.

Predstavnik statične stranice je stranica „O nama“. Ovakve stranice najčešće koriste statičan sadržaj koji se vrlo rijetko mijenja, te često čine dio web aplikacija raznih tvrtki. Dinamični dio predstavljaju stranice bloga, također popularnog sadržaja na webu.

Kako bi se održala konzistentnost implementacije u svim programskim okvirima navedene stranice imaju istu strukturu (HTML) i za stiliziranje koriste Tailwind CSS okvir koji je vrlo popularan i raširen zbog svoje jednostavnosti korištenja i integracije sa modernim programskim okvirima.

Za font je korišten Google font obitelji Archivo. Sve slike osim open graph slike poslužuju se kroz servis Lorem Ipsum prema id-ju ili seedu kako bi se izbjeglo učitavanje nasumičnih slika pri svakoj posjeti stranice. Za dohvaćanje sadržaja bloga korišten je popularni servis za posluživanje testnih podataka JSONPlaceholder.



Slika 2: Prikaz blog podstranice

2.2 Implementacija u programskim okvirima

Work in progress

2.3 Mjerene metrike Performansi

Za mjerjenje performansi aplikacije odabrane su standardne metrike poznate pod nazivom Web vitals nastale na Googleovu inicijativu kako bi se standardiziralo mjerjenje performansi i dobio bolji uvid u korisničko iskustvo. Osim glavnih metrika u istraživanje su uključene i dodatne metrike poput vremena izgradnje, vremena skriptiranja i veličine JavaScript paketa koji se dostavlja klijentu. Slijedi kratak pregled svih korištenih metrika.

2.4 Isrtavanje najvećeg sadržaja (LCP)

LCP, odnosno „Largest Contentful Paint“, mjeri koliko je vremena potrebno da se na stranici prikaže njen najveći vidljivi element. Proteklo vrijeme do 2,5 sekunde smatra se dobrom pokazateljem performansi. U obzir se uzima samo onaj dio elementa koji korisnik može vidjeti – dijelovi koji su odrezani ili nisu vidljivi se zanemaruju. Tijekom učitavanja stranice, koje se često odvija u nekoliko koraka, taj najveći element može se promijeniti. Prvo se zabilježi vrijeme učitavanja tada najvećeg elementa, a ako se kasnije pojavi još veći, mjeri se vrijeme učitavanja novog elementa. Kada korisnik počne koristiti stranicu, preglednik prestaje pratiti nove elemente jer interakcije mogu promijeniti ono što je vidljivo. LCP se zato koristi kao pokazatelj trenutka kada je glavni sadržaj stranice postao dostupan korisniku [12].

2.5 Ukupno vrijeme blokiranja (TBT)

TBT (engl. Total Blocking Time) predstavlja ukupan vremenski period koji protekne od otvaranja stranice do trenutka kada korisnik može normalno stupiti u interakciju s njom. Pod interakcijom se podrazumijevaju radnje poput klika mišem, dodira zaslona ili unosa putem tipkovnice. Vrijeme se računa tako da se zbroje svi dijelovi dugih zadataka (onih koji traju više od 50 milisekundi), a koji sprječavaju preglednik da odmah odgovori na korisničke radnje. Ovi se zadaci bilježe u razdoblju između početnog prikaza sadržaja (FCP) i trenutka kada stranica postane potpuno funkcionalna. Sve što prelazi granicu od 50 ms kod pojedinog zadatka ulazi u ukupnu vrijednost TBT-a [12].

2.6 Kumulativna promjena rasporeda (CLS)

Pomicanje sadržaja na web stranici tijekom njenog učitavanja naziva se pomak izgleda (engl. layout shift). Do toga najčešće dolazi zbog asinkronog učitavanja sadržaja ili dodavanja novih DOM elemenata preko postojećih komponenti. CLS prati najveći niz takvih pomaka koji se dogode unutar jednog vremenskog okvira od maksimalno 5 sekundi, pri čemu između pojedinih pomaka ne smije proći više od jedne sekunde. Ova metrike je važna jer pokazuje stabilnost stranice, a poželjno je da rezultat bude što niži – vrijednost 0.1 smatra se dobrom rezultatom [12].

2.7 Vrijeme do interaktivnosti (TTI)

Ova metrika označava koliko je vremena potrebno da web stranica postane potpuno interaktivna. To znači da se prvo prikaže koristan sadržaj (što mjeri FCP), zatim da stranica reagira na korisničke radnje u roku kraćem od 50 milisekundi te da je većina upravljača događaja (engl. event handler) učitana i spremna. Vrijeme kraće od 3,8 sekundi smatra se dobrim rezultatom. Spor TTI često je uzrokovani loše optimiziranim JavaScript kodom. [12]

2.8 Vrijeme do prvog bajta (TTFB)

Vrijeme do prvog bajta označava koliko vremena prođe od trenutka kada preglednik pošalje zahtjev prema serveru do trenutka kada primi prvi bajt povratnog odgovora. Ova metrika prikazuje koliko brzo se uspostavlja veza i koliko brzo server reagira, uključujući sve faze poput preusmjeravanja, DNS rezolucije, TLS dogovora i slanja zahtjeva. Niža vrijednost TTFB-a znači da se stranica učitava brže, što pozitivno utječe i na ostale važne metrike poput FCP-a. Idealno bi bilo da TTFB ostane ispod 0,8 sekundi radi optimalnog korisničkog iskustva [13].

2.9 Dodatne metrike

- Veličina paketa (engl. bundle size) predstavlja ukupnu veličinu svih JavaScript datoteka koje se preuzimaju prilikom početnog učitavanja stranice – uključujući sve JS dijelove potrebne za prikaz početne stranice.
- Vrijeme izgradnje (engl. build time) odnosi se na trajanje procesa generiranja statičkih stranica prilikom pretvaranja izvornog koda u izvršni kod. Vrijeme izgradnje može se očitati iz terminala prilikom lokalne izgradnje ili pak iz Vercelovih zapisa izgradnje (eng. build logs).
- Ukupno trajanje izvršavanja skripti (engl. total scripting time) predstavlja zbroj vremena izvršavanja svih JavaScript skripti na stranici. Ova metrika pomaže pri identifikaciji potencijalnih uskih grla u performansama aplikacije.

2.10 Alati i testno okruženje

2.11 Lighthouse

Glavni i de facto standard za testiranje performansi web aplikacija je Googleov alat otvorenog koda imena Lighthouse. Ovaj alat integriran je u Chrome preglednik, ali i dostupan kao zasebna CLI aplikacija. Pruža sve što je potrebno za provođenje testova performansi i SEO-a i generiranje izvještaja u različitim formatima, što ga čini vrlo funkcionalnim i korisnim u unapređivanju tehničke strane web aplikacija i korisničkog iskustva [14].

2.12 Chrome DevTools

Chrome DevTools je set alata unutar Chrome preglednika koji između ostalog omogućuju analizu performansi web stranice. U kontekstu ovog rada najvažniji su paneli Performance i Network koji prikazuju brzinu učitavanja stranice i resursa. Ključna funkcionalnost je Performance Insights, koja snima korisničku sesiju i bilježi metrike poput FCP i LCP. Ovaj alat korišten je u analizi veličine JavaScript paketa koje klijent preuzima i izvršava [12].

2.13 Lighthouse reporter

Za potrebe ovog rada bilo je potrebno izvesti 360 Lighthouse testova² te prikupljene podatke obraditi, grupirati i sažeti (izraditi master tablice sa prosječnim vrijednostima). Zbog opsežnosti testiranja jedini praktični način bilo je napraviti Node.js skriptu koja će na temelju definiranih parametara koristeći Lighthouse CLI alat izvršiti sve testove i generirati datoteke s rezultatima u CSV formatu za daljnju analizu podataka. Aplikacija za testiranje sastoji se od 3 ključna dijela:

1. konfiguracijske JSON datoteke u kojoj su definirani svi parametri testiranja
2. datoteke LightHouseTestRunner.js koja izvozi istoimenu klasu sa svim metodama važnim za različite načine testiranja
3. datoteke cli.js koja upravlja testovima i pokreće ih kroz terminal sa odgovarajućim argumentima

Aplikacija za testiranje stavlja korisniku na raspolaganje raznovrstan set testova (prema programskom okviru, prema strategiji iscrtavanja itd.), no za potrebe ovoga rada korištena je naredba koja pokreće testiranje svih aplikacija po određenoj podstranici.

Primjer naredbe u bash terminalu: `node cli.js all slow4g blog`

Ova naredba pokreće cli.js skriptu koja testira blog podstranicu u svim aplikacijama u konfiguraciji slow4g.

Skripta će ovom naredbom kreirati sljedeće datoteke za podstranicu blog:

- 12 datoteka sa podacima svih 10 mjerenja po programskom okviru i strategiji iscrtavanja
- 4 datoteke prosječnih vrijednosti za svaku strategiju iscrtavanja
- 1 datoteku sa ukupnim rezultatima za odabranu podstranicu (konačne prosječne vrijednosti svih 12 kombinacija)

²3 aplikacije * 4 strategije iscrtavanja * 10 testova po kombinaciji * 3 podstranice (about, blog, blog detalji)

Ova naredba ponovljena je za svaku podstranicu čime je generirana 51 datoteka³. Za daljnju analizu korištene su 3 datoteke glavnih sažetaka za svaku stranicu. Svaka od tih datoteka sadrži prosječne vrijednosti testova za svaku od 12 kombinacija strategija i programskih okvira.

Zbog velikog broja provedenih testova, bilo je nužno osigurati njihovu potpunu uspješnost i cjelovitost podataka. Za tu svrhu u skriptu je ugrađen i mehanizam provjere uspješnosti testiranja, koji će u slučaju neuspješnog testa pokušati ponoviti test još dva puta, te ako ni tada ne uspije, naznačiti će gdje je došlo do greške i pokušati nastaviti s testiranjem. Po završetku izvršavanja test skripte moguće je ponovno inicirati izvođenje neuspješnih testova.

Prilikom testiranja skripte ovim mehanizmom otkrivene su greške u dvije web-aplikacije, koje su uzrokovale nedostatak određenih metrika. Zbog neispravne konfiguracije, pojedine podstranice su u produkcijskoj okolini vraćale grešku 404, što je rezultiralo nepotpunim rezultatima testiranja. Nakon ispravka konfiguracije, svi testovi su uspješno provedeni već pri prvom pokretanju.

Izvorni kod skripte, zajedno sa dobivenim rezultatima dostupan je na autorovom Git repozitoriju.⁴

³3 podstranice * 3 programska okvira * 4 strategije iscrtavanja + 12 sažetaka + 3 glavna sažetka

⁴<https://github.com/AlphaActual/lighthouse-reporter>

```

Lloyds@TIN-LLOYDS MINGW64 ~/Documents/lighthouse-reporter (main)
$ node cli.js all slow4g blogPost

♦ Starting comprehensive Lighthouse testing
♦ Profile: SLOW4G
♦ Page: Blog Post (/blog/1)
♦ Total applications: 12

♦ Testing Client-Side Rendering Applications
♦ Profile: SLOW4G
♦ Page: Blog Post (/blog/1)

♦ Starting Lighthouse test for Next.js CSR
♦ Profile: SLOW4G
♦ Page: Blog Post (/blog/1)
♦ URL: https://render-strategy-demo-csr-next-app-c.vercel.app/blog/1
♦ Runs: 10
  ♦ Running audit 1/10 (attempt 1/3)...
    LH:status Connecting to browser +0ms
    LH:status Navigating to about:blank +6ms
    LH:status Benchmarking machine +6ms
    LH:status Preparing target for navigation mode +1s
    LH:status Cleaning origin data +17ms
    LH:status Cleaning browser cache +19ms
    LH:status Preparing network conditions +21ms
    LH:status Navigating to https://render-strategy-demo-csr-next-app-c.vercel.app/blog/1 +55ms
    LH:status Getting artifact: DevtoolsLog +4s
    LH:status Getting artifact: Trace +1ms
    LH:status Getting artifact: ConsoleMessages +1ms
    LH:status Getting artifact: CSSUsage +0ms
    LH:status Getting artifact: DOMStats +16ms
    LH:status Getting artifact: ImageElements +9ms
    LH:status Getting artifact: JsUsage +97ms
    LH:status Getting artifact: MetaElements +1ms

```

Slika 3: Testiranje stranice pojedinog bloga kroz terminal

2.14 Testno okruženje

Kako bi se osigurali konzistentni uvjeti testiranja koji odgovaraju stvarnom korisničkom iskustvu, te omogućila podrška za specifične strategije iscrtavanja (kao što je ISR), odabran je sljedeći pristup.

Sve aplikacije postavljene su na platformu Vercel koja u trenutku pisanja ovog rada jedina podržava sve odabrane strategije iscrtavanja za svaki od prethodno navedenih programskih okvira. Ovo osigurava maksimalnu kompatibilnost i integraciju bez komplikiranih konfiguracija [15].

Vercel platforma također omogućava globalnu distribuciju putem CDN-a, što znači da se sadržaji serviraju iz čvorova najbližih korisniku. Zbog ovoga i činjenice da su sve aplikacije i njihov izvorni kod javno dostupni na Internetu, moguće je provođenje dodatnih istraživanja i proširivanja ovog rada, te osiguravanje relevantnosti podataka, bez obzira na geografsku lokaciju računala na kojem bi se potencijalno radila nova testiranja. Testiranja provedena u ovom radu izvršena su na slijedećoj računalnoj konfiguraciji:

- Model: Lenovo IdeaPad Slim 5 16ABR8
- Procesor: AMD Ryzen 5 7530U sa Radeon grafikom - 2.00 GHz

- Radna memorija: 16.0 GB (13.9 GB iskoristivo)
- Tip sustava: 64-bitni operacijski sustav, procesor baziran na arhitekturi x64
- Verzija operacijskog sustava: Windows 11 Home (24H2)
- Instalirana Node.js verzija: v22.14.0

Prije izvođenja testiranja zabilježeni su slijedeći mrežni uvjeti korištenjem Speedtest web alata:

- Server: Tehnoline Telekom 45.142.9.180
- Ping: 15ms
- Jitter: 2ms
- Download: 291.6 Mbps
- Upload: 102.0 Mbps

3 Rezultati testiranja

Rezultati testiranja podjeljeni su prema stranicama koje predstavljaju određeni tip web sadržaja:

1. **Stranica O nama** - statični sadržaj
2. **Stranica Blog** - dinamički sadržaj s listom blog postova
3. **Stranica pojedinog blog posta** - dinamički sadržaj s detaljima pojedinog blog posta

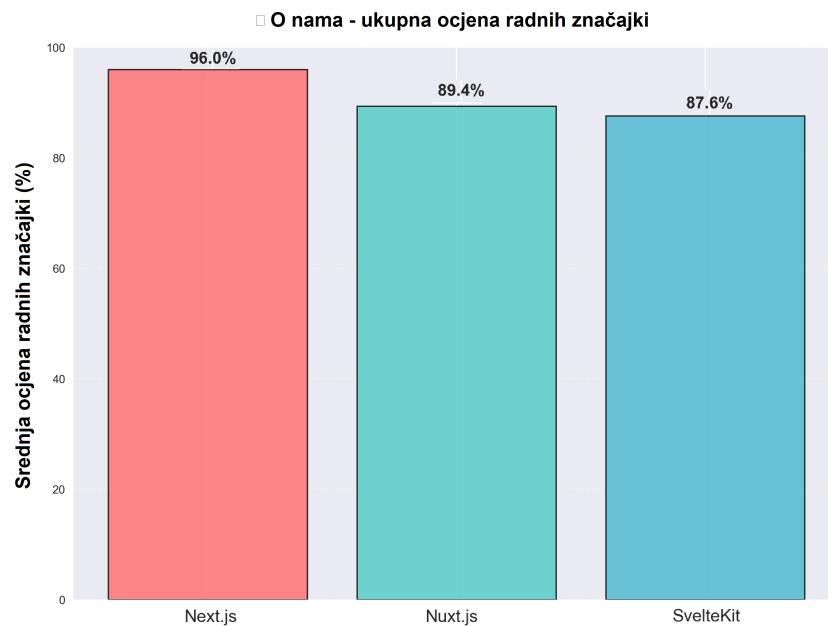
Za prikaz rezultata pojedine stranice odabrani su slijedeći grafovi i tablice:

- **Ukupna ocjena radnih značajki programskog okvira**
- **Ukupna ocjena radnih značajki programskog okvira po metriци**
- **Ocjena kombinacije programskog okvira i strategije iscrtavanja po metriци (postotak)**
- **Ocjena kombinacije programskog okvira i strategije iscrtavanja po metriци (vrijednosti)**
- **Ocjena strategije iscrtavanja po programskom okviru**
- **Sažetak rezultata po kategorijama** - tablica

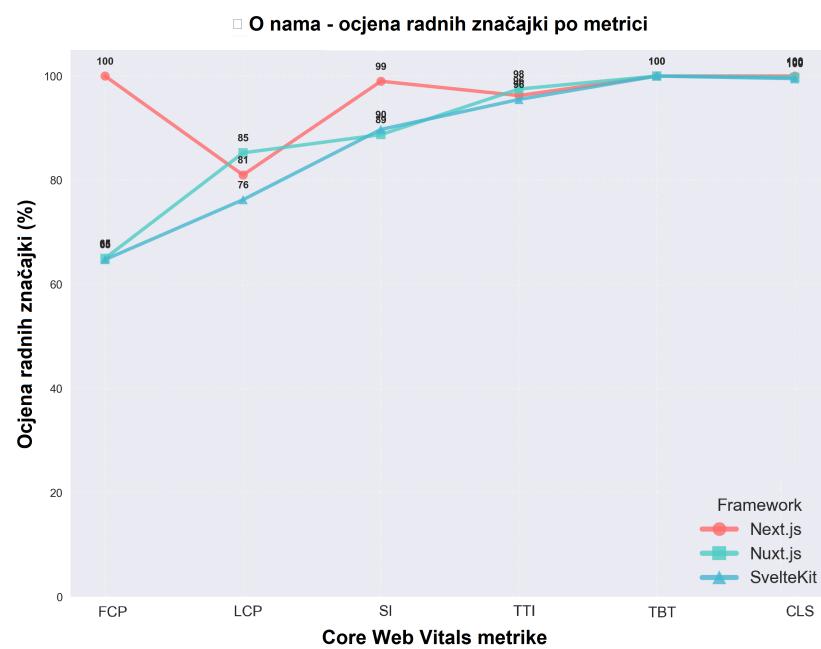
Za prikaz dodatnih metrika odabrani su slijedeći grafovi i tablice:

- **Srednje vrijednosti vremena izgradnje po strategiji iscrtavanja**
- **Srednja veličina JS paketa po strategiji iscrtavanja**
- **Srednja vremena skriptiranja po strategiji iscrtavanja**
- **Mapa topline vremena izgradnje po strategiji iscrtavanja**
- **Konzistentnost vremena izgradnje po strategiji iscrtavanja**
- **Raspodjela veličine JS paketa po programskom okviru**
- **Odnos između veličine JS paketa i vremena izgradnje**
- **Mapa topline učinkovitosti programskog okvira po strategiji iscrtavanja**
- **Srednja vrijednost vremena izgradnje po programskom okviru**
- **Kompromis između vremena skriptiranja i veličine JS paketa**

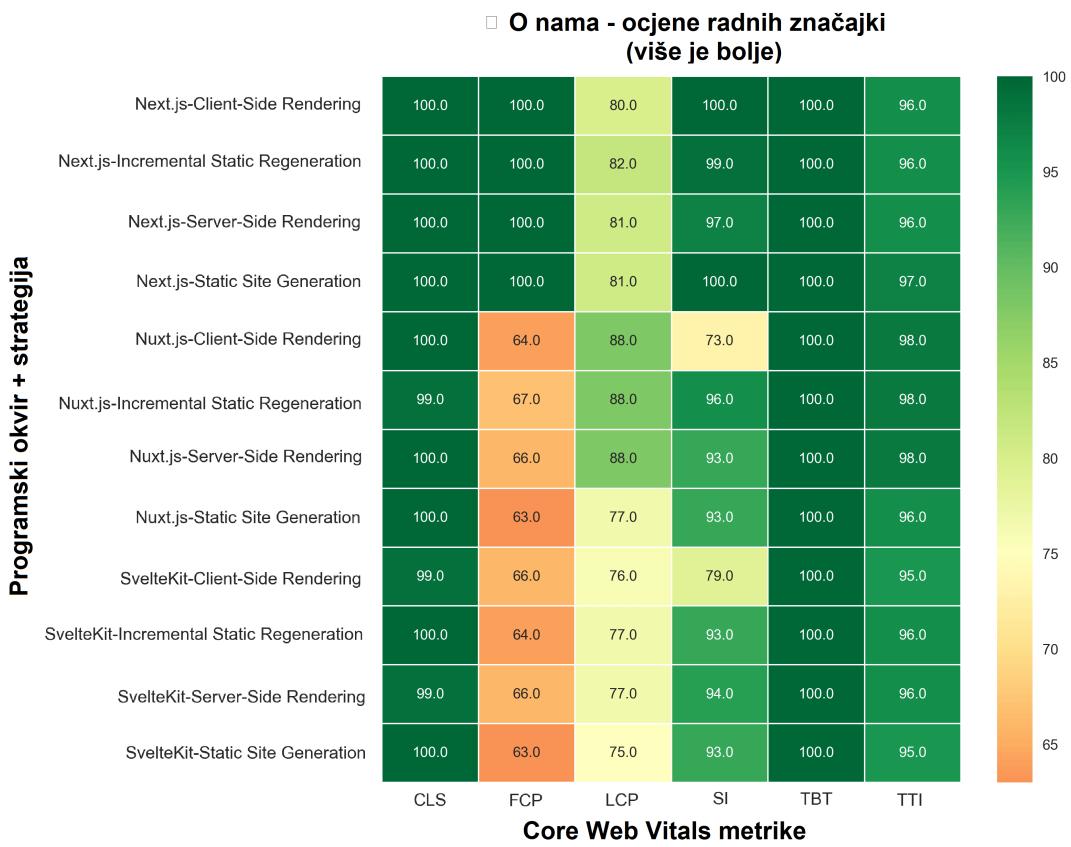
3.1 Rezultati testiranja stranice "O nama"



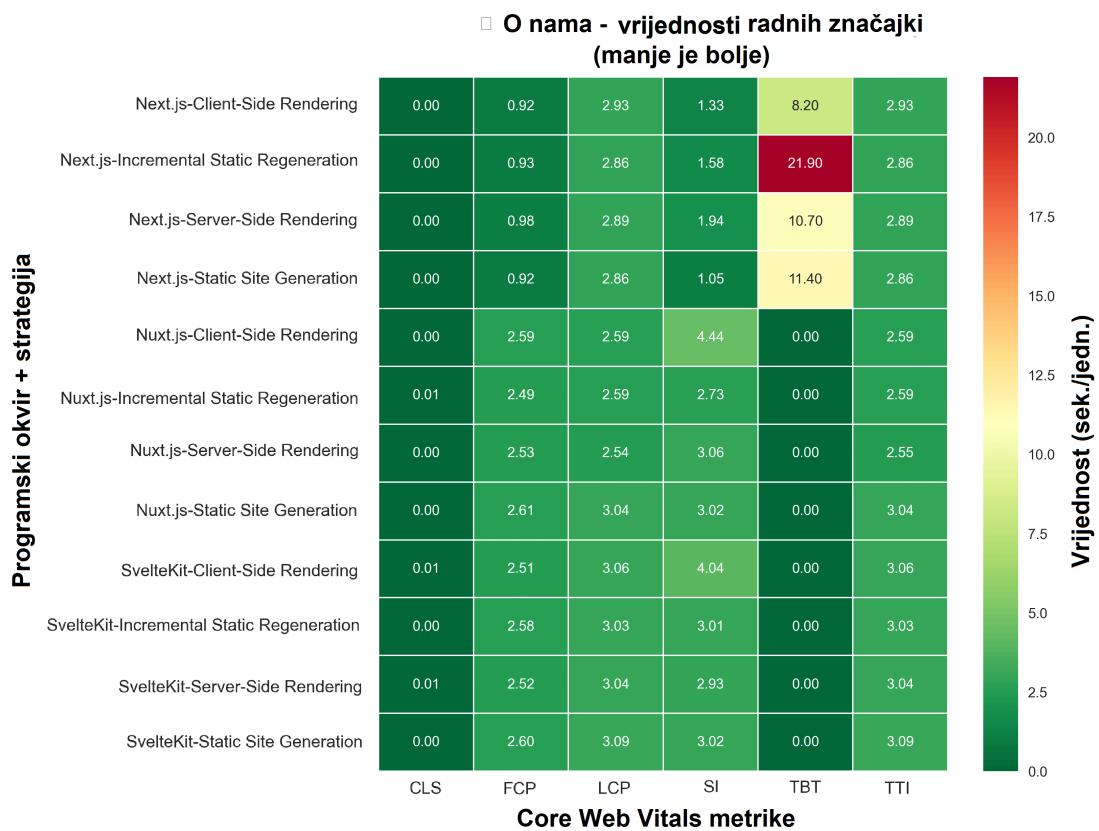
Slika 4: Ukupna ocjena radnih značajki programskih okvira (stranica O nama)



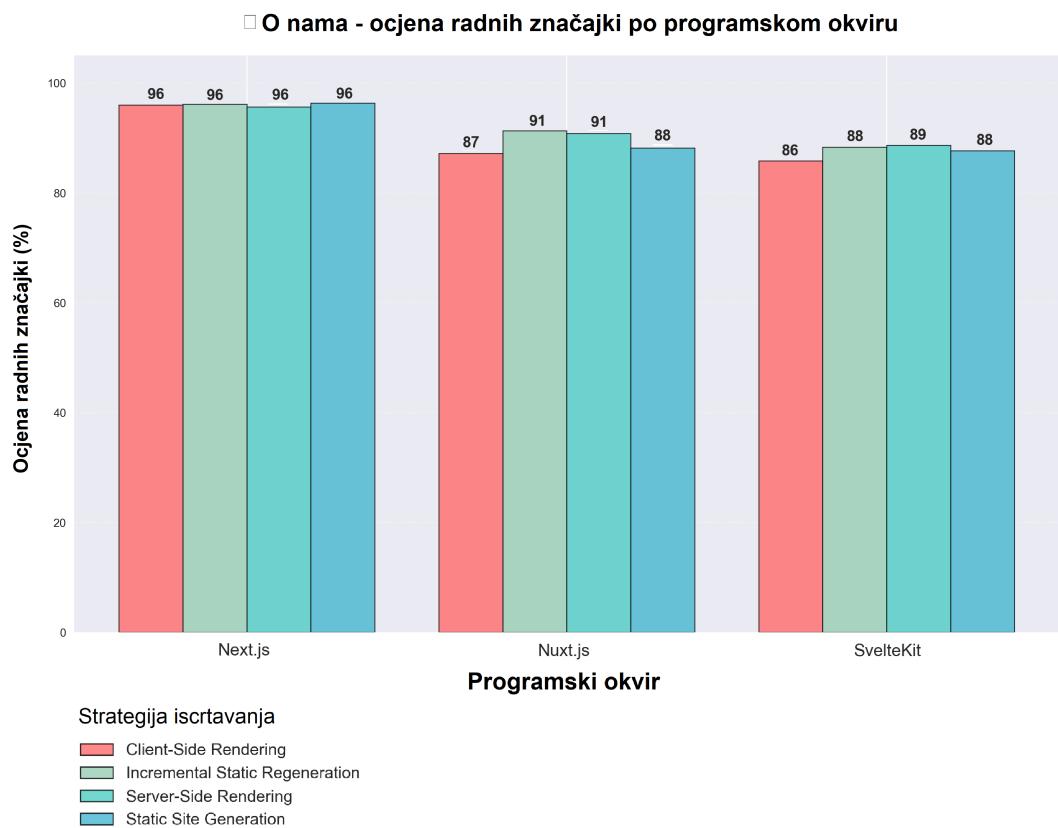
Slika 5: Ukupna ocjena radnih značajki programskih okvira po metrici (stranica O nama)



Slika 6: Ocjene radnih značajki - postotak (stranica O nama)



Slika 7: Ocjene radnih značajki - vrijednosti (stranica O nama)



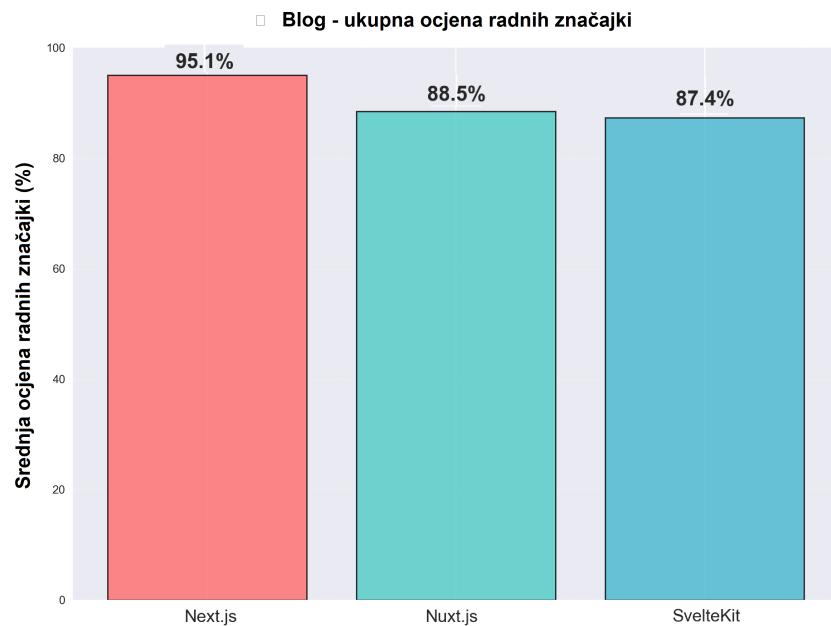
Slika 8: Ocjene radnih značajki - usporedba strategija (stranica O nama)

3.2 Sažetak rezultata - stranica "O nama"

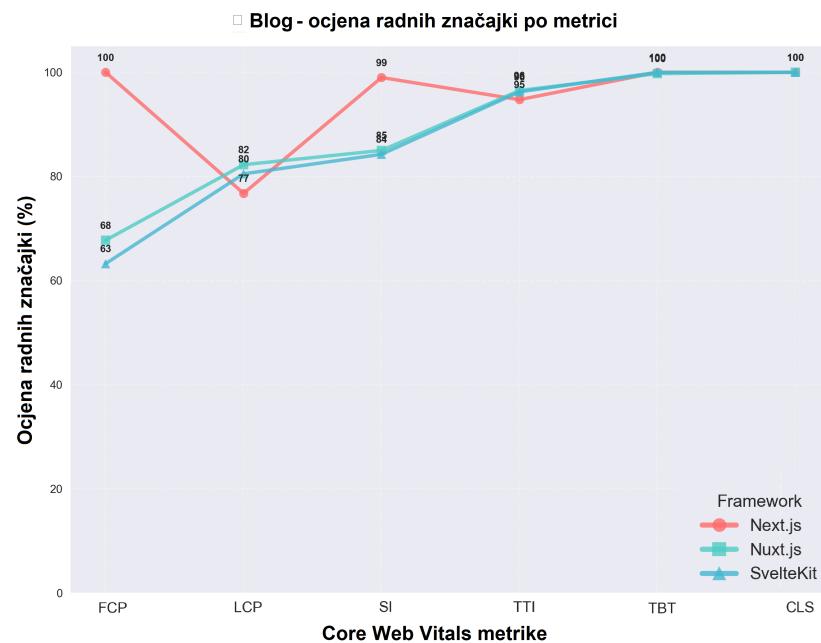
Kategorija	Element	Ocjena
Programski okviri	1. Next.js	96.0% ($\pm 7.0\%$)
	2. Nuxt.js	89.4% ($\pm 13.2\%$)
	3. SvelteKit	87.6% ($\pm 13.5\%$)
Strategije iscrtavanja	1. Incremental Static Regeneration	91.9% ($\pm 11.7\%$)
	2. Server-Side Rendering	91.7% ($\pm 11.5\%$)
	3. Static Site Generation	90.7% ($\pm 13.0\%$)
	4. Client-Side Rendering	89.7% ($\pm 13.0\%$)
Najbolje kombinacije	1. Next.js + Static Site Generation	96.3% ($\pm 7.6\%$)
	2. Next.js + Incremental Static Regeneration	96.2% ($\pm 7.1\%$)
	3. Next.js + Client-Side Rendering	96.0% ($\pm 8.0\%$)
	4. Next.js + Server-Side Rendering	95.7% ($\pm 7.4\%$)
	5. Nuxt.js + Incremental Static Regeneration	91.3% ($\pm 12.7\%$)
Vodeći po metriци	FCP: Next.js + Client-Side Rendering	100.0% (0.920s)
	LCP: Nuxt.js + Client-Side Rendering	88.0% (2.590s)
	SI: Next.js + Client-Side Rendering	100.0% (1.330s)
	TTI: Nuxt.js + Client-Side Rendering	98.0% (2.590s)
	TBT: Next.js + Client-Side Rendering	100.0% (8.200ms)
	CLS: Next.js + Client-Side Rendering	100.0% (0.000)

Tablica 1: Sažetak rezultata testiranja stranice O nama

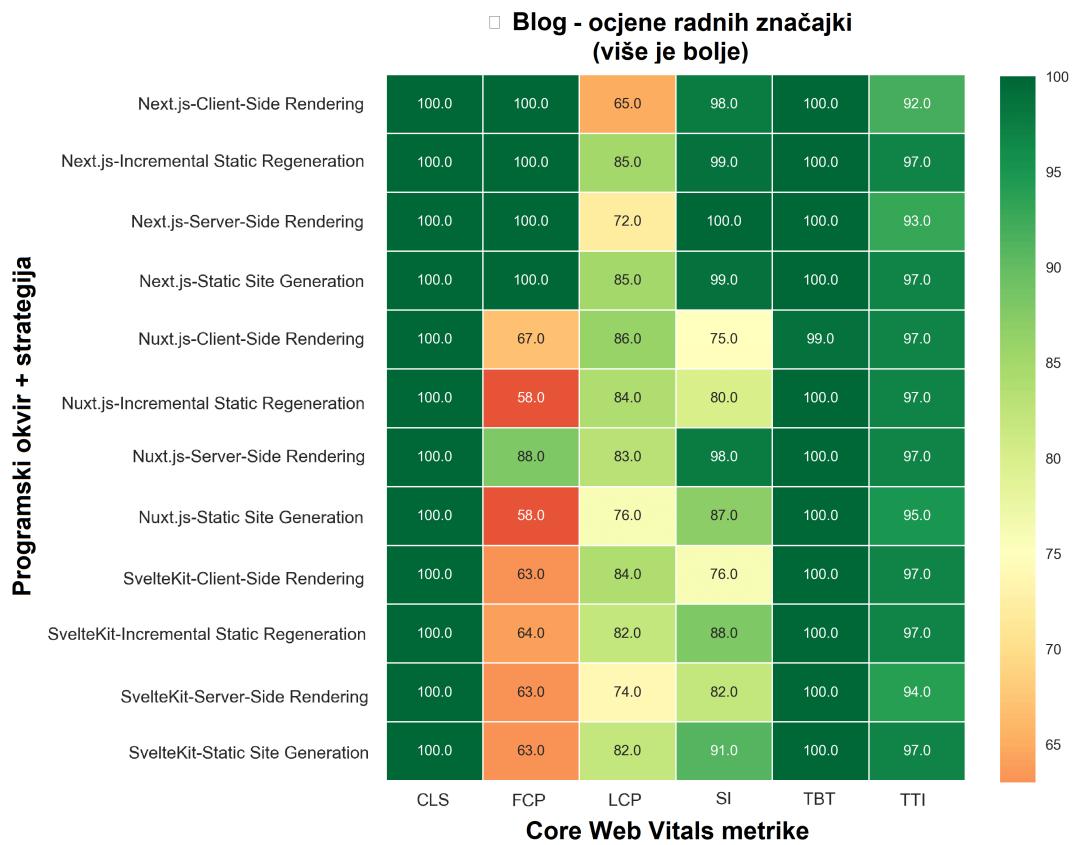
3.3 Rezultati testiranja stranice Blog



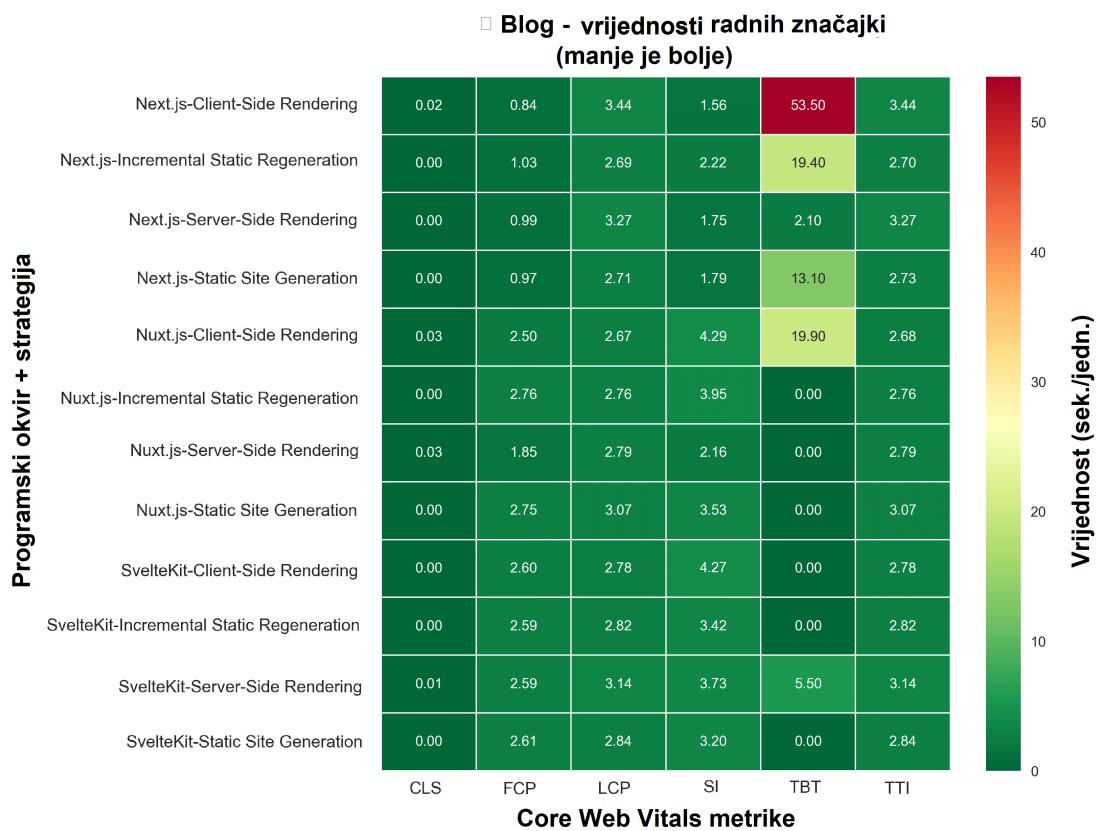
Slika 9: Ukupne ocjene radnih značajki (stranica Blog)



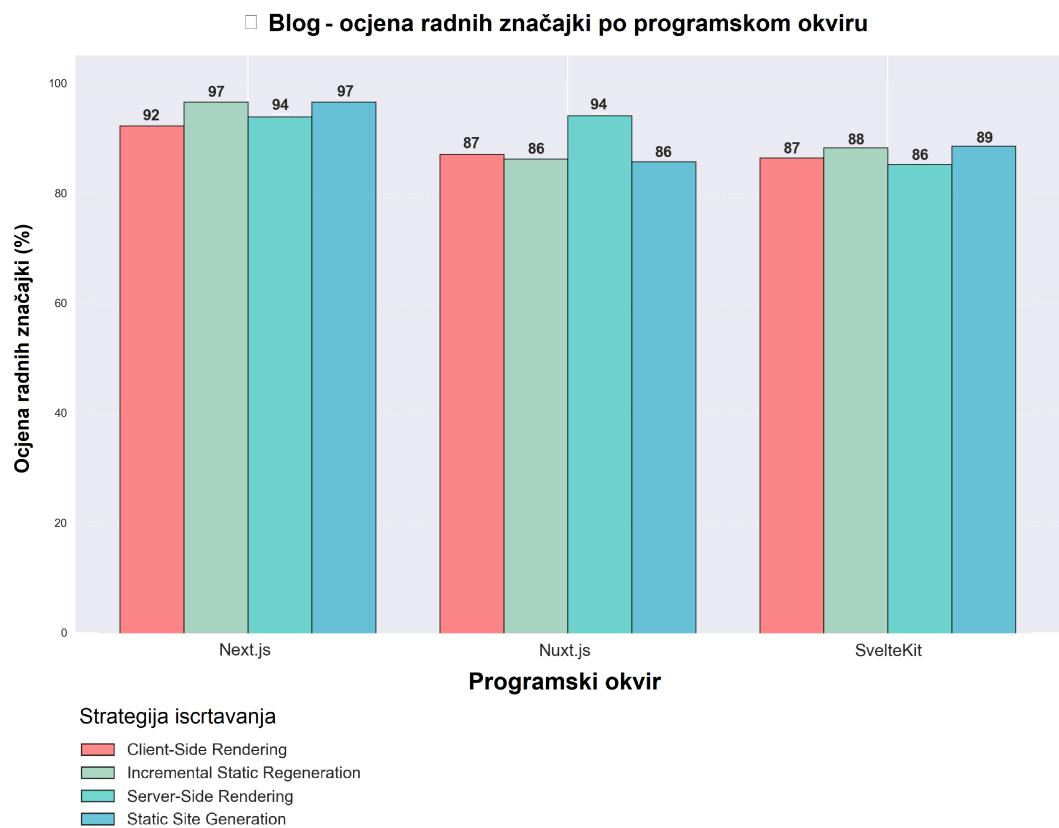
Slika 10: Ukupne ocjene radnih značajki po metrići (stranica Blog)



Slika 11: Ocjene radnih značajki - postotak (stranica Blog)



Slika 12: Ocjene radnih značajki - vrijednosti (stranica Blog)



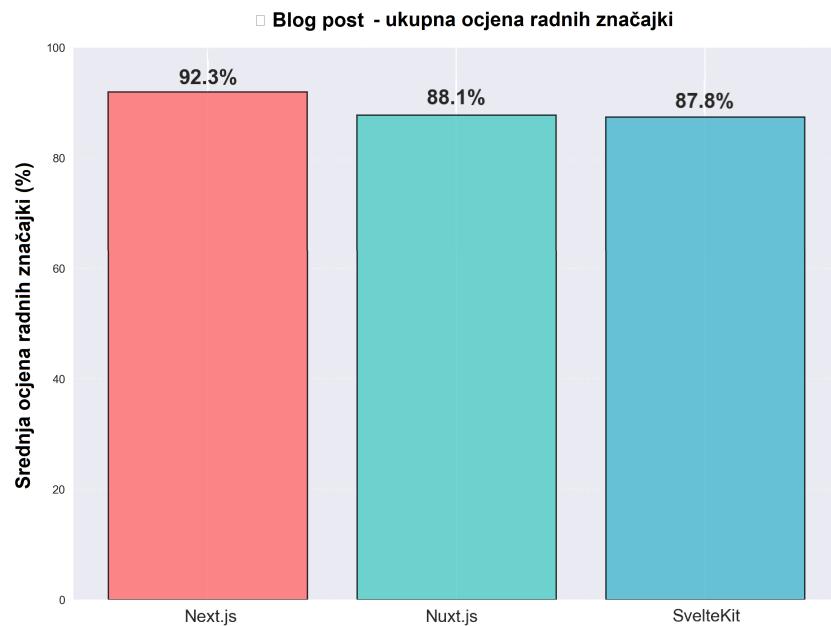
Slika 13: Ocjene radnih značajki - usporedba strategija (stranica Blog)

3.4 Sažetak rezultata - stranica Blog

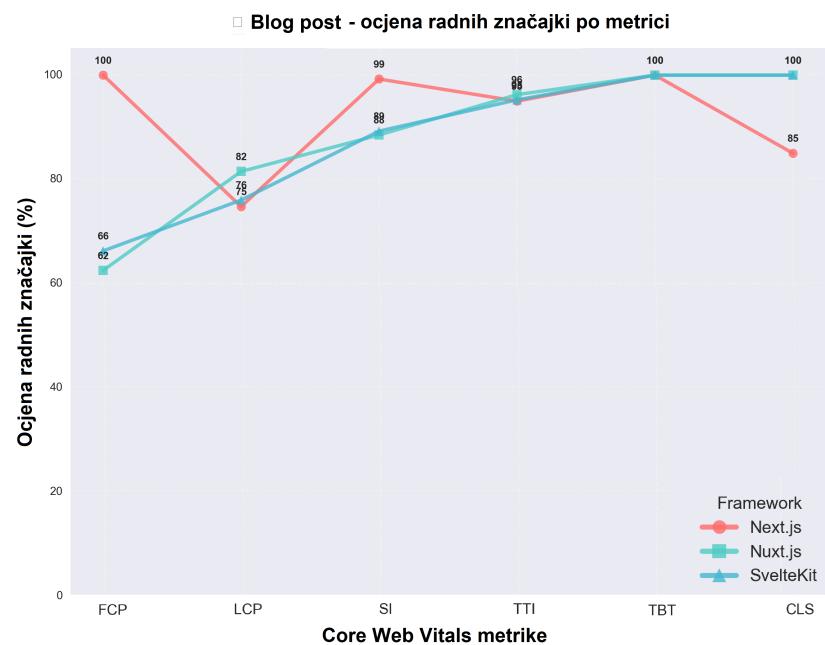
Tablica 2: Sažetak rezultata testiranja stranice Blog

Kategorija	Element	Ocjena
Programski okviri	1. Next.js	95.1% ($\pm 9.4\%$)
	2. Nuxt.js	88.5% ($\pm 13.5\%$)
	3. SvelteKit	87.4% ($\pm 13.7\%$)
Strategije iscrtavanja	1. Server-Side Rendering	91.3% ($\pm 11.7\%$)
	2. Incremental Static Regeneration	90.6% ($\pm 12.9\%$)
	3. Static Site Generation	90.6% ($\pm 13.2\%$)
	4. Client-Side Rendering	88.8% ($\pm 13.7\%$)
Najbolje kombinacije	1. Next.js + Incremental Static Regeneration	96.8% ($\pm 5.9\%$)
	2. Next.js + Static Site Generation	96.8% ($\pm 5.9\%$)
	3. Nuxt.js + Server-Side Rendering	94.3% ($\pm 7.1\%$)
	4. Next.js + Server-Side Rendering	94.2% ($\pm 11.2\%$)
	5. Next.js + Client-Side Rendering	92.5% ($\pm 13.8\%$)
Vodeći po metrici	FCP: Next.js + Client-Side Rendering	100.0% (0.840s)
	LCP: Nuxt.js + Client-Side Rendering	86.0% (2.670s)
	SI: Next.js + Server-Side Rendering	100.0% (1.750s)
	TTI: Nuxt.js + Client-Side Rendering	97.0% (2.680s)
	TBT: Next.js + Client-Side Rendering	100.0% (53.500ms)
	CLS: Next.js + Client-Side Rendering	100.0% (0.020)

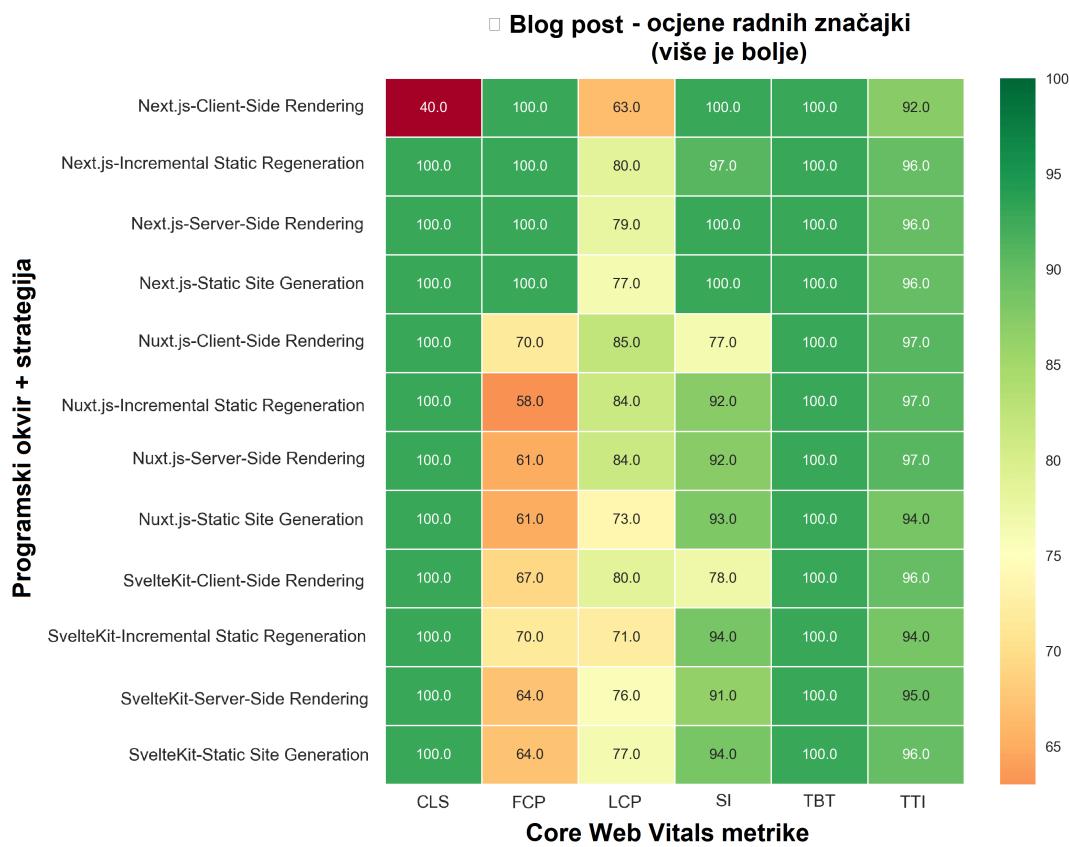
3.5 Rezultati testiranja stranice pojedinog blog posta



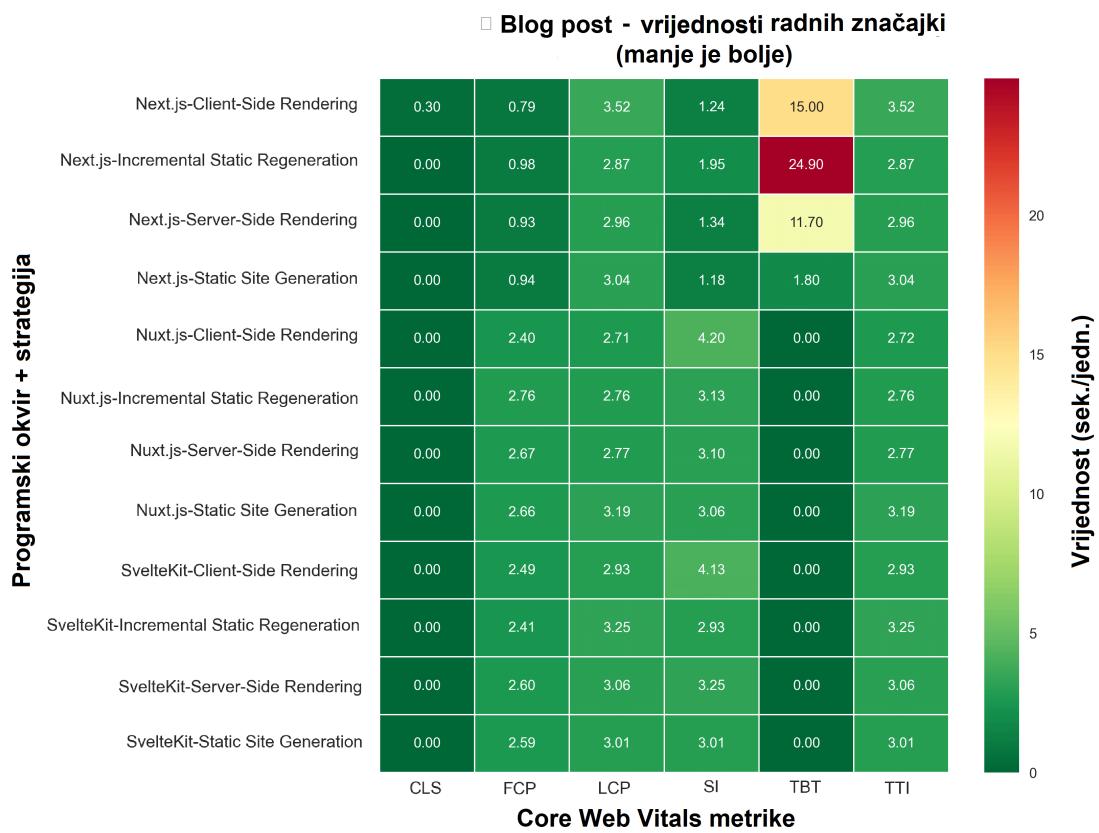
Slika 14: Ukupne ocjene radnih značajki (stranica pojedinog bloga)



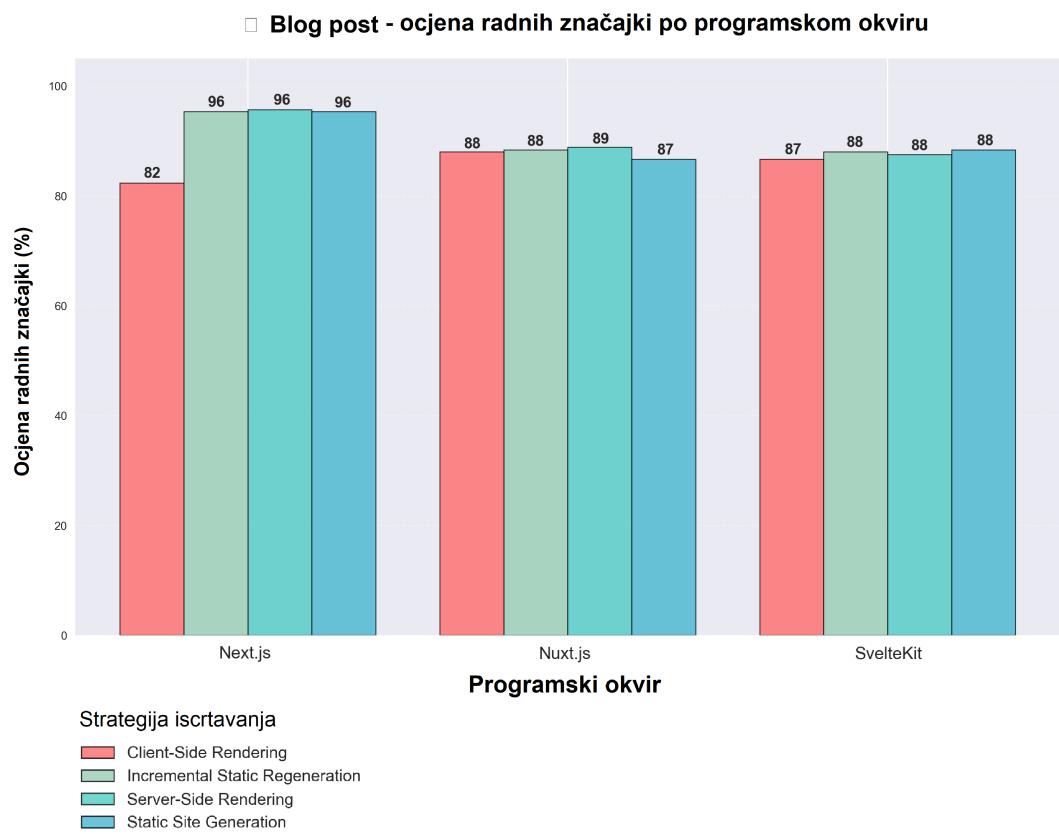
Slika 15: Ukupne ocjene radnih značajki po metrići (stranica pojedinog bloga)



Slika 16: Ocjene radnih značajki - postotak (stranica pojedinog bloga)



Slika 17: Ocjene radnih značajki - vrijednosti (stranica pojedinog bloga)



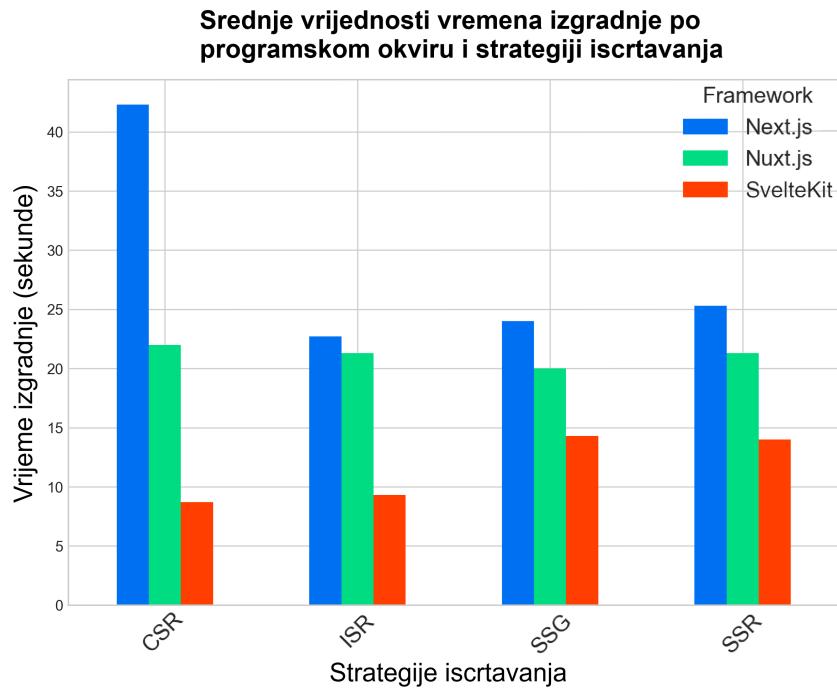
Slika 18: Ocjene radnih značajki - usporedba strategija (stranica pojedinog bloga)

3.6 Sažetak rezultata - stranica pojedinog blog posta

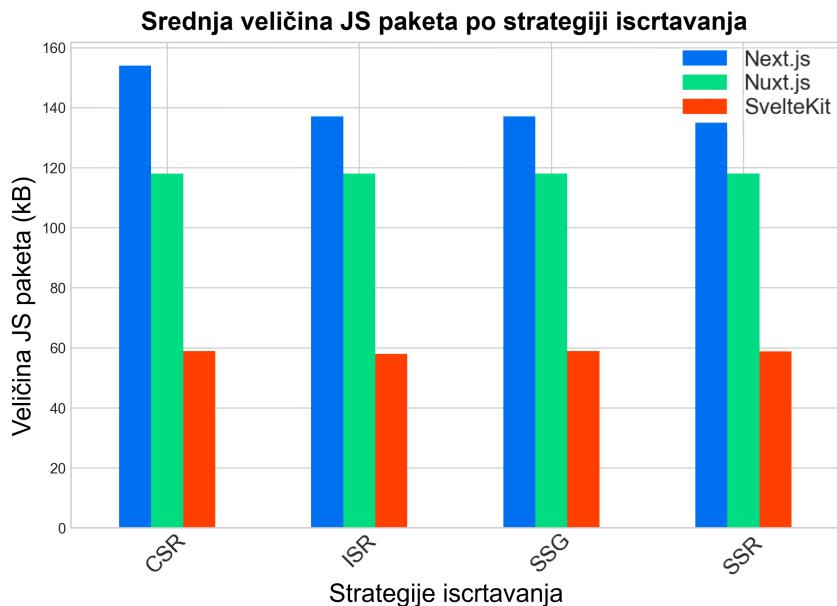
Tablica 3: Sažetak rezultata testiranja stranice pojedinog bloga

Kategorija	Element	Ocjena
Programski okviri	1. Next.js	92.3% ($\pm 14.8\%$)
	2. Nuxt.js	88.1% ($\pm 14.1\%$)
	3. SvelteKit	87.8% ($\pm 13.3\%$)
Strategije iscrtavanja	1. Server-Side Rendering	90.8% ($\pm 12.8\%$)
	2. Incremental Static Regeneration	90.7% ($\pm 12.8\%$)
	3. Static Site Generation	90.3% ($\pm 13.4\%$)
	4. Client-Side Rendering	85.8% ($\pm 17.3\%$)
Najbolje kombinacije	1. Next.js + Server-Side Rendering	95.8% ($\pm 8.4\%$)
	2. Next.js + Incremental Static Regeneration	95.5% ($\pm 7.8\%$)
	3. Next.js + Static Site Generation	95.5% ($\pm 9.2\%$)
	4. Nuxt.js + Server-Side Rendering	89.0% ($\pm 15.0\%$)
	5. SvelteKit + Static Site Generation	88.5% ($\pm 14.7\%$)
Vodeći po metrici	FCP: Next.js + Client-Side Rendering	100.0% (0.790s)
	LCP: Nuxt.js + Client-Side Rendering	85.0% (2.710s)
	SI: Next.js + Client-Side Rendering	100.0% (1.240s)
	TTI: Nuxt.js + Client-Side Rendering	97.0% (2.720s)
	TBT: Next.js + Client-Side Rendering	100.0% (15.000ms)
	CLS: Nuxt.js + Client-Side Rendering	100.0% (0.000)

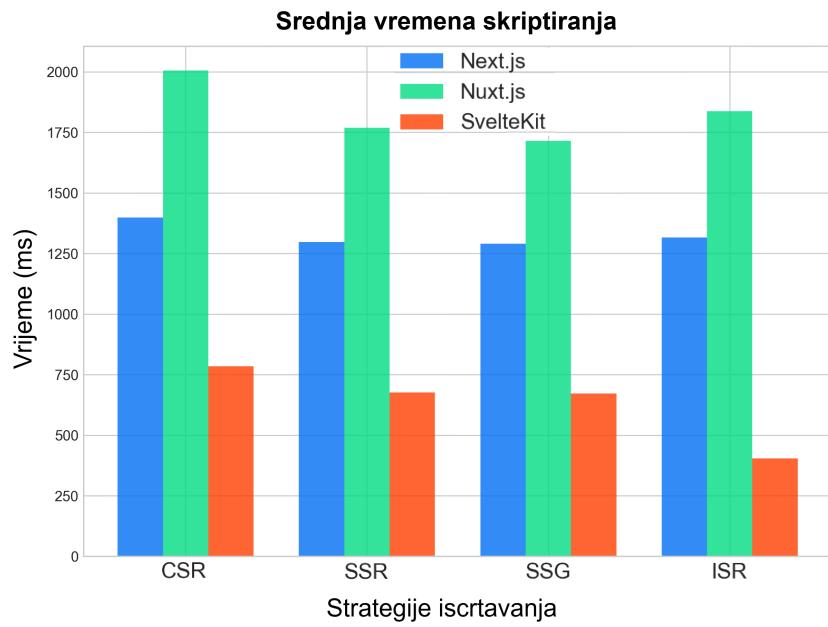
3.7 Dodatne metrike



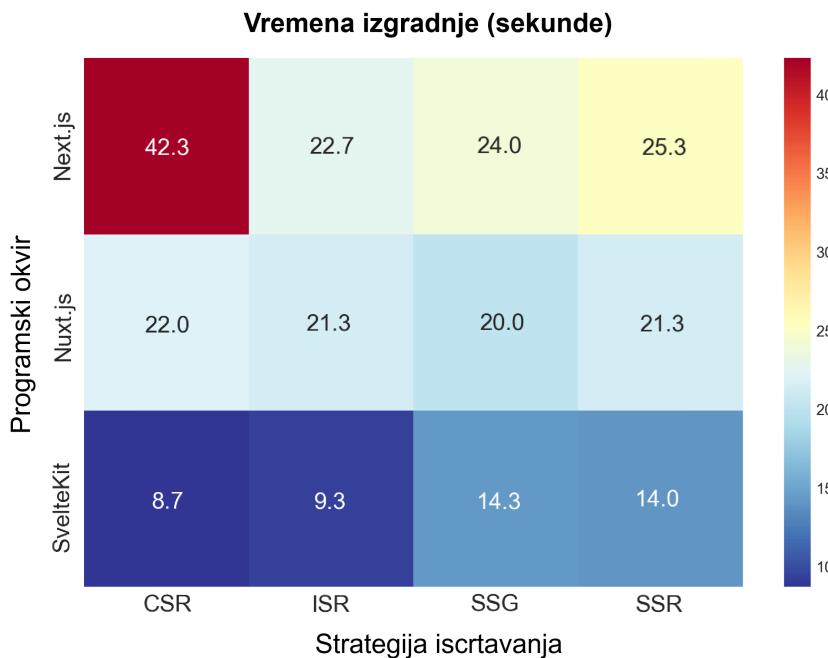
Slika 19: Prosječna vremena izgradnje po okviru i strategiji



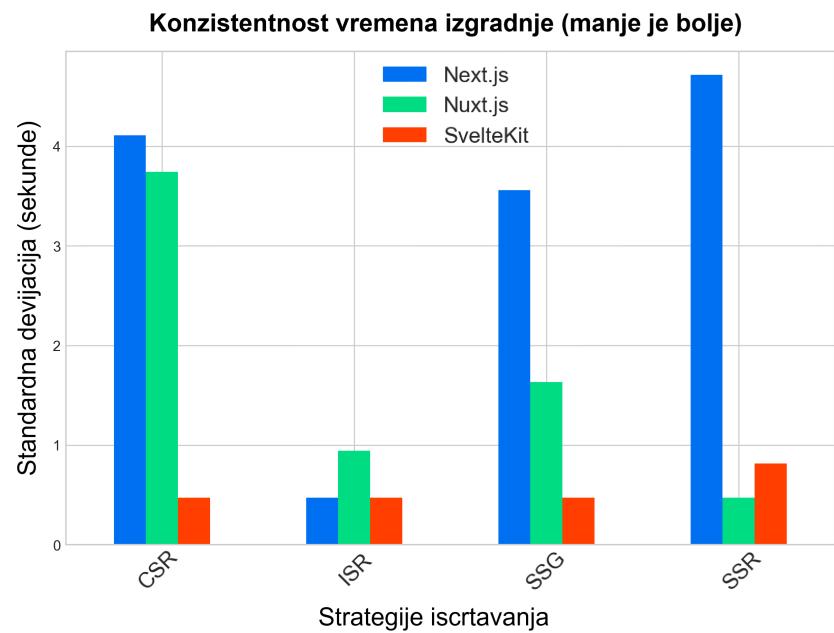
Slika 20: Prosječna veličina paketa po strategiji



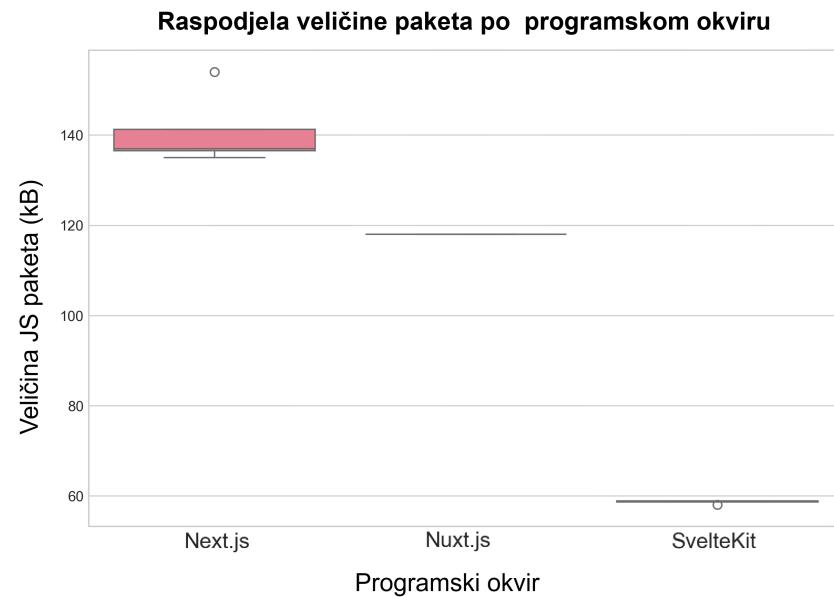
Slika 21: Prosječna vremena skriptiranja



Slika 22: Mapa topline vremena izgradnje

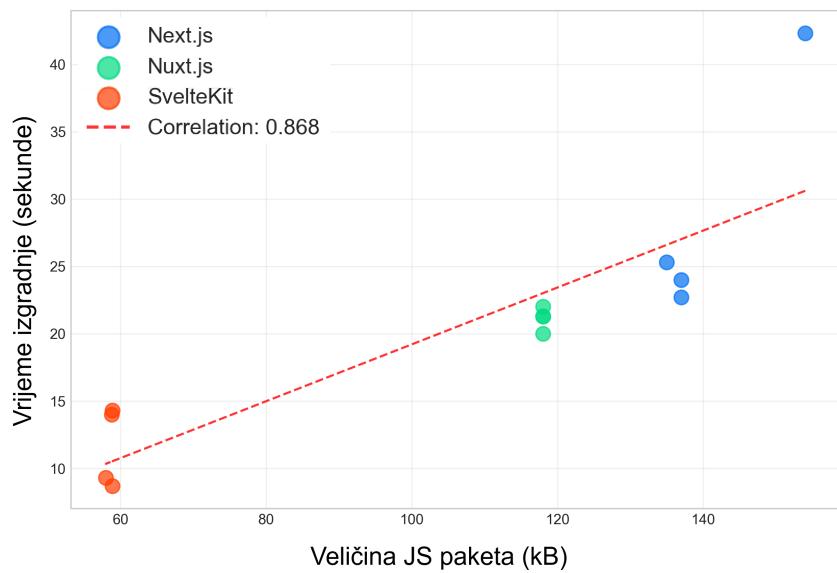


Slika 23: Konzistentnost vremena izgradnje

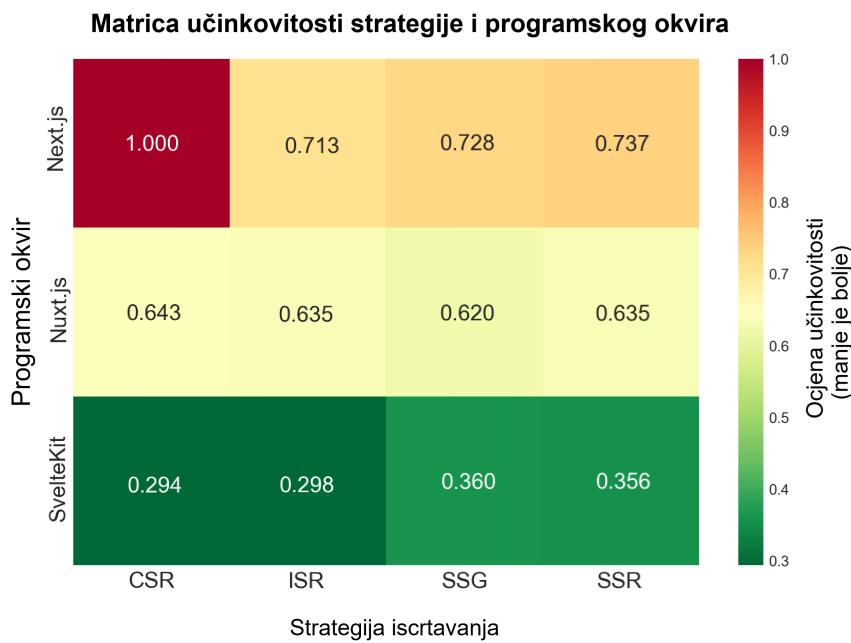


Slika 24: Distribucija veličine paketa po okviru

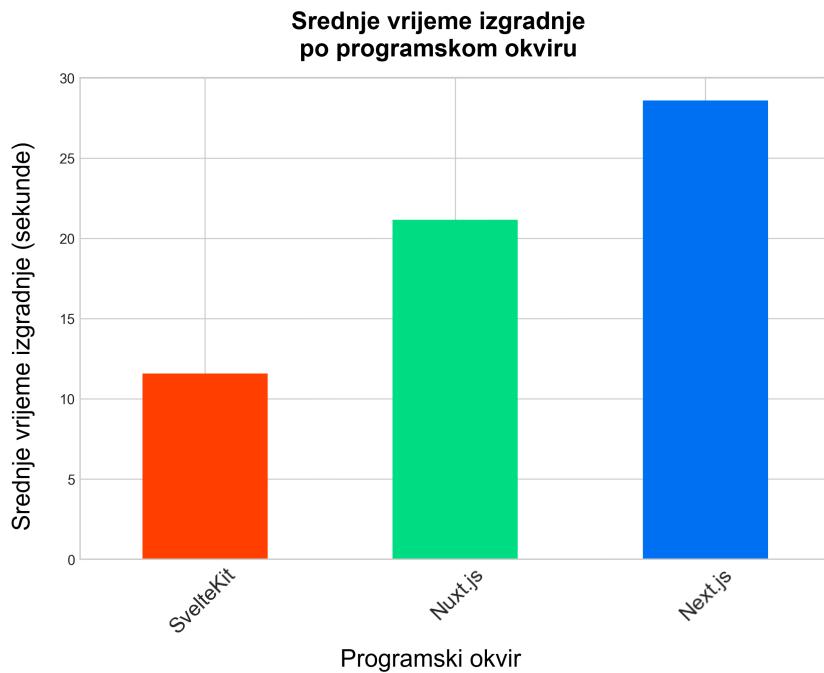
Odnos između veličine JS paketa i vremena izgradnje



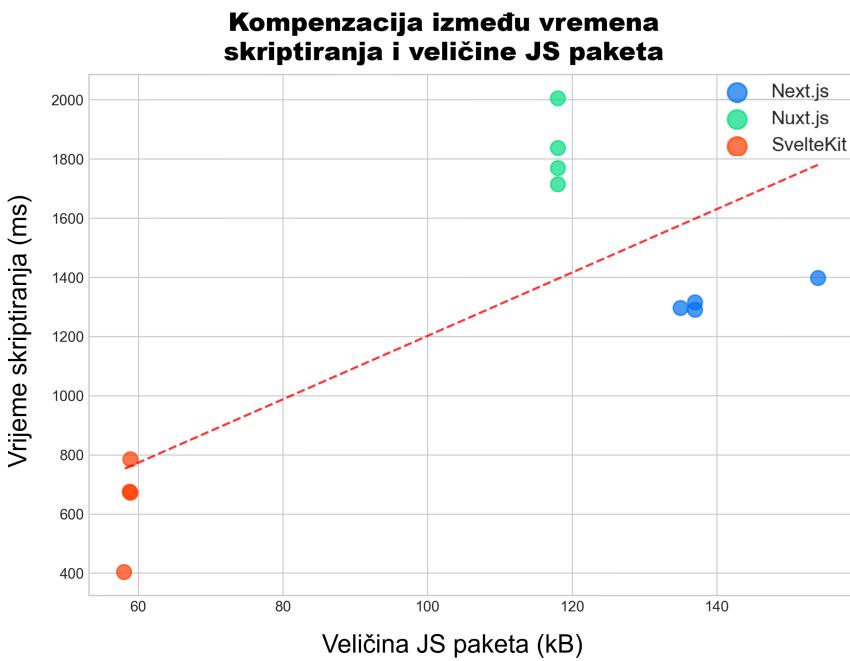
Slika 25: Korelacija između veličine paketa i vremena izgradnje



Slika 26: Matrica efikasnosti okvira i strategije



Slika 27: Ukupne ocjene radnih značajki izgradnje okvira



Slika 28: Kompromis između performansi i veličine paketa

4 Analiza rezultata

5 Zaključak

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

Literatura

- [1] A. A. Moore. "How to choose the best rendering strategy for your app". Vercel, srpanj 2024. [Na internetu]. Dostupno: <https://vercel.com/blog/how-to-choose-the-best-rendering-strategy-for-your-app> [pristupano 12. lipnja 2025.].
- [2] P. Bratslavsky. "What Is Website Rendering: CSR, SSR, and SSG Explained". strapi, svibanj 2025. [Na internetu]. Dostupno: <https://strapi.io/blog/what-is-website-rendering> [pristupano 13. lipnja 2025.].
- [3] I. Beran. "Usporedba metoda renderiranja web aplikacija". Prirodoslovno-matematički fakultet u Splitu, Sveučilište u Splitu, siječanj 2023. [Na internetu]. Dostupno: <https://repozitorij.pmfst.unist.hr/islandora/object/pmfst:1621> [pristupano 13. lipnja 2025.].
- [4] Google. "In-depth guide to how Google Search works". [Na internetu]. Dostupno: <https://developers.google.com/search/docs/fundamentals/how-search-works?hl=en> [pristupano 14. lipnja 2025.].
- [5] Next.js. "Loading UI and Streaming". [Na internetu]. Dostupno: <https://nextjs.org/docs/14/app/building-your-application/routing/loading-ui-and-streaming> [pristupano 14. lipnja 2025.].
- [6] Vue.js. "Server-Side Rendering (SSR)". [Na internetu]. Dostupno: <https://vuejs.org/guide/scaling-up/ssr.html> [pristupano 14. lipnja 2025.].
- [7] Next.js. "Static Site Generation (SSG)". [Na internetu]. Dostupno: <https://nextjs.org/docs/pages/building-your-application/rendering/static-site-generation> [pristupano 14. lipnja 2025.].
- [8] Sanity. "Static Site Generation (SSG) definition". [Na internetu]. Dostupno: <https://www.sanity.io/glossary/static-site-generation> [pristupano 14. lipnja 2025.].
- [9] Netlify. "Incremental Static Regeneration: Its Benefits and Its Flaws", ožujak 2021. [Na internetu]. Dostupno: <https://www.netlify.com/blog/2021/03/08/incremental-static-regeneration-its-benefits-and-its-flaws> [pristupano 15. lipnja 2025.].
- [10] O. Troyan. "Comparing Nuxt 3 Rendering Modes: SWR, ISR, SSR, SSG, SPA". RisingStack, svibanj 2024. [Na internetu]. Dostupno: <https://blog.risingstack.com/nuxt-3-rendering-modes/#isr> [pristupano 15. lipnja 2025.].
- [11] G. Dumais. "Next.js: The Ultimate Cheat Sheet To Page Rendering". Jams-tack, srpanj 2021. [Na internetu]. Dostupno: <https://guydumais.digital/blog/next-js-the-ultimate-cheat-sheet-to-page-rendering/> [pristupano 14. lipnja 2025.].

- [12] Carl Nordström and A. Danielsson. "Comparisons of Server-side Rendering and Client-side Rendering for Web Pages", rujan 2023. [Na internetu]. Dostupno: <https://uu.diva-portal.org/smash/get/diva2:1797261/FULLTEXT02.pdf> [pristupano 13. lipnja 2025.].
- [13] W. J. Pollard Barry. "Time to First Byte (TTFB)". web.dev. [Na internetu]. Dostupno: <https://web.dev/articles/ttfb> [pristupano 19. lipnja 2025.].
- [14] Google. "Introduction to Lighthouse". [Na internetu]. Dostupno: <https://developer.chrome.com/docs/lighthouse/overview> [pristupano 19. lipnja 2025.].
- [15] Vercel. "Frameworks on Vercel". [Na internetu]. Dostupno: <https://vercel.com/docs/frameworks> [pristupano 19. lipnja 2025.].

Popis slika

1	Pregled strategija iscrtavanja u Next.js programskom okviru [11]	8
2	Prikaz blog podstranice	9
3	Testiranje stranice pojedinog bloga kroz terminal	14
4	Ukupna ocjena radnih značajki programskih okvira (stranica O nama)	17
5	Ukupna ocjena radnih značajki programskih okvira po metrici (stranica O nama)	17
6	Ocjene radnih značajki - postotak (stranica O nama)	18
7	Ocjene radnih značajki - vrijednosti (stranica O nama)	19
8	Ocjene radnih značajki - usporedba strategija (stranica O nama)	20
9	Ukupne ocjene radnih značajki (stranica Blog)	22
10	Ukupne ocjene radnih značajki po metrici (stranica Blog)	22
11	Ocjene radnih značajki - postotak (stranica Blog)	23
12	Ocjene radnih značajki - vrijednosti (stranica Blog)	24
13	Ocjene radnih značajki - usporedba strategija (stranica Blog)	25
14	Ukupne ocjene radnih značajki (stranica pojedinog bloga)	27
15	Ukupne ocjene radnih značajki po metrici (stranica pojedinog bloga)	27
16	Ocjene radnih značajki - postotak (stranica pojedinog bloga)	28
17	Ocjene radnih značajki - vrijednosti (stranica pojedinog bloga)	29
18	Ocjene radnih značajki - usporedba strategija (stranica pojedinog bloga)	30
19	Prosječna vremena izgradnje po okviru i strategiji	32
20	Prosječna veličina paketa po strategiji	32
21	Prosječna vremena skriptiranja	33
22	Mapa topline vremena izgradnje	33
23	Konzistentnost vremena izgradnje	34
24	Distribucija veličine paketa po okviru	34
25	Korelacija između veličine paketa i vremena izgradnje	35
26	Matrica efikasnosti okvira i strategije	35
27	Ukupne ocjene radnih značajki izgradnje okvira	36
28	Kompromis između performansi i veličine paketa	36

Popis tablica

1	Sažetak rezultata testiranja stranice O nama	21
2	Sažetak rezultata testiranja stranice Blog	26
3	Sažetak rezultata testiranja stranice pojedinog bloga	31