

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 i komunikacijske znanosti
Znanstvena grana : Informacijski sustavi i informatologija

Mentor: doc. dr. sc. Nikola Tanković

Pula, rujan, 2025. godine

Sažetak

Odabir optimalne strategije iscrtavanja web aplikacije (CSR, SSR, SSG, ISR) ključan je za postizanje dobrih performansi i korisničkog iskustva. Ovaj rad usporedno analizira navedene strategije u tri popularna programska okvira za izradu web aplikacija Next.js, Nuxt.js i SvelteKit. U svakom od programskih okvira implementirana je identična demo aplikacija sa stranicama statičnog i dinamičnog sadržaja, te su nad njima vršeni testovi ključnih metrika performansi prikupljeni Lighthouse CLI alatom u kontroliranim mrežnim uvjetima. Rezultati prikazuju prednosti i mane svake strategije iscrtavanja ovisno o tipu stranica, 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 (CSR, SSR, SSG, ISR) is crucial for achieving good performance and user experience. This paper provides a comparative analysis of these strategies 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	Uvod	1
1.1	Strategije iscrtavanja	2
1.1.1	Iscrtavanje na strani klijenta (CSR)	2
1.1.2	Iscrtavanje na strani poslužitelja (SSR)	4
1.1.3	Generiranje statičkih stranica (SSG)	5
1.1.4	Inkrementalna statička generacija (ISR)	6
2	Metodologija	9
2.1	Opis referentne aplikacije	9
2.2	Implementacija u Programskim Okvirima	10
2.3	Mjerene Metrike Performansi	10
2.3.1	Iscrtavanje najvećeg sadržaja (LCP)	10
2.3.2	Ukupno vrijeme blokiranja (TBT)	10
2.3.3	Kumulativna promjena rasporeda (CLS)	10
2.3.4	Vrijeme do interaktivnosti (TTI)	11
2.3.5	Vrijeme do prvog bajta (TTFB)	11
2.3.6	Dodatne metrike	11
2.4	Alati i testno okruženje	11
2.4.1	LightHouse	11
2.4.2	Chrome DevTools	11
2.4.3	LightHouse reporter	12
2.4.4	Testno okruženje	13
3	Zaključak	15
	Literatura	16
	Popis slika	18
	Popis tablica	19

1 Uvod

U današnjem okruženju razvoja web aplikacija, više nego ikada prije, vidljiva je težnja i potreba industrije dolazi za postizanjem boljih performansi, skalabilnosti i korisničkog iskustva. Ta potreba potaknula je istraživanje i razvoj novih rješenja kao odgovor na pitanje kako najoptimalnije i najbrže dostaviti web sadržaj korisniku? Kritična stavka u odgovoru na ovo pitanje je upravo odabir odgovarajuće strategije iscertavanja web aplikacija s obzirom na vrstu sadržaja koji se korisniku servira. 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 iscertavanja poput iscertavanja na serveru (SSR) gdje se svježi sadržaj generira na svaki zahtjev korisnika i generiranja statičnih stranica (SSG) kod koje se svaka stranica unaprijed iscertava u statičnu datoteku već duže vrijeme čine temelj i osnovni standard u razvoju web aplikacija. Povećanje kompleksnosti aplikacija i potreba za sve višim performansama izrodili su i nove hibridne metode iscertavanja 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 iscertavanja 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 iscertavanja 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 iscertavanja (CSR, SSR, SSG i ISR). Postavljeni ciljevi su:

1. Izrada i implementacija funkcionalno i stilski identične web aplikacije sa podstranicama dinamičnog i statičnog sadržaja (blog).
2. Sustavno mjerenje i usporedba odabranih metrika performansi što uključuje: vrijeme do prvog bajta (TTFB), prvo iscertavanje sadržaja (FCP), iscertavanje najvećeg sadržaja (LCP), ukupno vrijeme blokiranja (TBT), vrijeme do interaktivnosti (TTI), veličinu paketa (bundle size) i vremena izgradnje (build time) nad svakom kombinacijom programskog okvira i strategije iscertavanja te na sve 3 definirane podstranice.
3. Utvrđivanje prednosti i nedostataka svake strategije iscertavanja 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 iscertavanja.

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

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

1.1 Strategije iscertavanja

Pojam strategije iscertavanja 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 učitati korisniku i koliko će aplikacija biti prilagodljiva i dinamična. Odabir strategije iscertavanja također bitno utječe i na SEO tj. vidljivost kod pretraživanja internetskim tražilicama. [2]

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

1.1.1 Iscertavanje na strani klijenta (CSR)

Ova metoda iscertavanja 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 iscertavanja, 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 iscertavanje u pregledniku, preglednik najprije mora pričekati da se preuzme sav potreban JavaScript kod, te se njegovim izvršavanjem HTML dokument ispunjava elementima koji će se iscertati. 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 svježiji 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 iscertavanja sadržaja. Ovo ograničenje posebno dolazi do izražaja prilikom učitavanja na sporijim mrežama.
- Lošiji SEO – iako današnji pretraživači¹ mogu očitati i stranice iscertane 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 iscertati.

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

- Potreba za visokom razinom interaktivnosti (dashboards)
- Gdje 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, pronalazak i indeksiranje web stranica [4]

1.1.2 Iscrtavanje na strani poslužitelja (SSR)

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

No moderni programski okviri poput Nuxta i SvelteKite spajaju SSR i CSR na način da se prilikom prvobitnog posjeta stranici ona iscrtava na poslužitelju, te se korisniku šalje već popunjen HTML dokument koji preglednik može odmah krenuti prikazivati, 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 elementima 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 funkcionira 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 izbjeći 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 funkcionirati, 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 generira 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 datoteka mogu se spremati 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 izgradi HTML dokument koji nije prethodno 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 klijentski uređaj.
- 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:

- Stalna potreba za svježim i ažuriranim podacima
- Visoka razina personalizacije (custom dashboards)
- Vizualizacija podataka u realnom vremenu
- Dobar SEO [1]

1.1.3 Generiranje statičkih stranica (SSG)

Kod ove strategije iscrtavanja HTML stranice se unaprijed generiraju na serveru prilikom izgradnje aplikacije (engl. build time), te se zatim poslužuju klijentima na zahtjev, bez potrebe za ponovnim generiranjem na 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 generiranje na poslužitelju ili na preuzimanje i izvršavanje JavaScript koda
- Odlične su za SEO – budući da se brzo učitavanju i sadrže sav potrebni sadržaj idealne su za pregled od strane pretraživača
- Nisko opterećenje poslužitelja – nema potrebe za obradom ili generiranjem HTML-a, već je spreman za slanje klijentu

- Najniži troškovi infrastrukture

Nedostatci ove strategije su:

- Duže vrijeme izgradnje ako postoji veliki broj stranica
- Za ažuriranje sadržaja potrebno je ponovno inicirati izgradnju i postavljanje (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:

- Marketinške stranice
- Blog postovi
- E-commerce proizvodi
- Dokumentacija i pomoć

Generalno pravilo je postaviti pitanje, može li se stranica generirati 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 generiranim 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]

1.1.4 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 generira statična stranica kojoj se odredi vrijeme revalidacije, tj. vrijeme nakon kojega će se ona smatrati zastarjelom i biti će ju potrebno ponovno generirati. No okidač za ovu generaciju biti će zahtjev prvog posjetitelja nakon vremena isteka revalidacije. Tom posjetitelju će se odmah isporučiti ova stara verzija stranice bez obzira koliko je dugo prekoračeno njeno vrijeme validacije, te će se nakon uspješne ponovne generacije u pozadini, nova verzija stranice poslužiti prvom slijedećem klijentu. Na ovaj način novo generirana stranica se dodaje u web aplikaciju.

Uobičajeni način objavljivanja na poslužitelj jest atomsko objavljivanje, tj. kada se cjelokupni kod, resursi i konfiguracija ažuriraju u isto vrijeme. Ovaj način

objavljivanja čuva integritet stranice i omogućava jednostavan opoziv objave i povratak na prijašnju verziju u slučaju potrebe (engl. rollback). Svaka pojedinačna objava je jedna cjelovita verzija web aplikacije. ISR strategija razbija ovaj integritet budući da stalno nadodaje novo generirane stranice u web aplikaciju, odvojeno od početno izgrađenog koda. Zbog ovog spremanja u pričuvenu 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 drastično olakšavajući cijeli proces konfiguracije i postavljanja. 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 poput SSG strategije – 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:

- Korisnici koji prvi posjete 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 (engl. cache). [9]



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

2 Metodologija

2.1 Opis referentne aplikacije

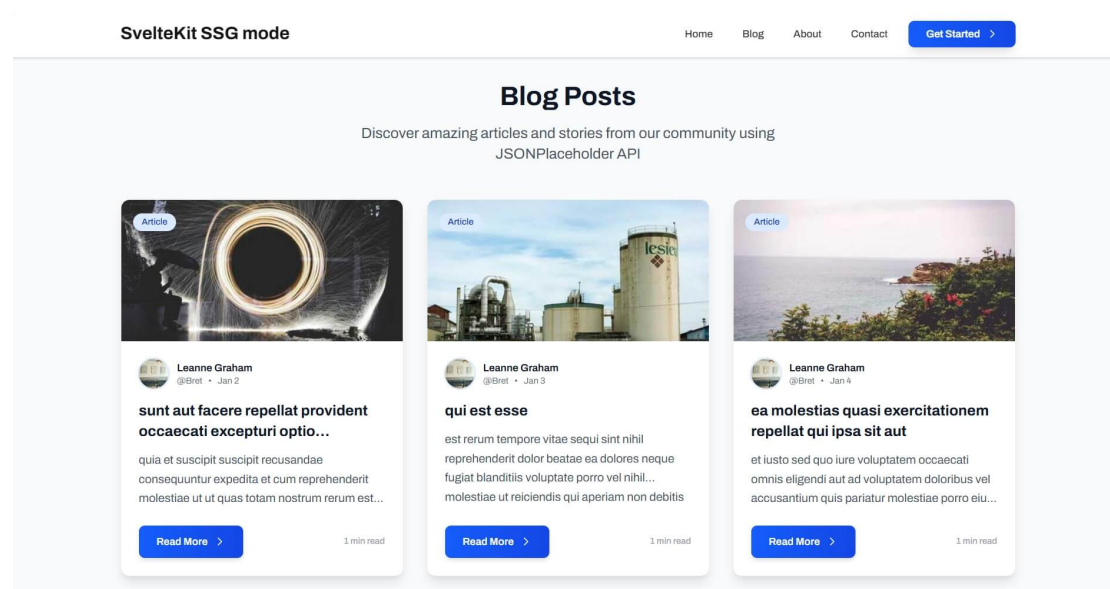
Za potrebe testiranja performansi različitih programskih okvira kroz različite strategije iscrtavanja bilo je potrebno napraviti demo aplikaciju koja bi bila dovoljno reprezentativna, ali ne presložena, te bi uključivala stranice sa statičnim i dinamičnim sadržajem.

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.

Za dinamični dio stranice kreirane su stranica 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 sike poslužuju se kroz servis Lorem Picsum 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 [autorski rad]

2.2 Implementacija u Programskim Okvirima

2.3 Mjerene Metrike Performansi

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

2.3.1 Iscrtavanje 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 sekundi smatra se dobrim 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]

2.3.2 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]

2.3.3 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 metrika je važna jer pokazuje stabilnost stranice, a poželjno je da rezultat bude što niži – vrijednost 0.1 smatra se dobrim rezultatom. [12]

2.3.4 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 su većina upravljača događaja (engl. event handler) učitana i spremna. Vrijeme kraće od 3,8 sekundi smatra se dobrim rezultatom. Spor TTI često je uzrokovan loše optimiziranim JavaScript kodom. [12]

2.3.5 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 kako 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]

2.3.6 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.

2.4 Alati i testno okruženje

2.4.1 LightHouse

Glavni i de facto standard za testiranje performansi web aplikacija je Googleov alat otvorenog koda naziva 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 lakim za korištenje i vrlo korisnim u unapređivanju tehničke strane web aplikacija i korisničkog iskustva. [14]

2.4.2 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]

2.4.3 LightHouse reporter

Za potrebe ovog rada bilo je potrebno izvesti 360 Lighthouse testova² te prikupljene podatke obraditi, grupirati i sumirati (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 iscertavanja 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 slow3g 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 iscertavanja
- 4 datoteke prosječnih vrijednosti za svaku strategiju iscertavanja
- 1 datoteku sa ukupnim rezultatima za odabranu podstranicu (konačne prosječne vrijednosti svih 12 kombinacija)

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.

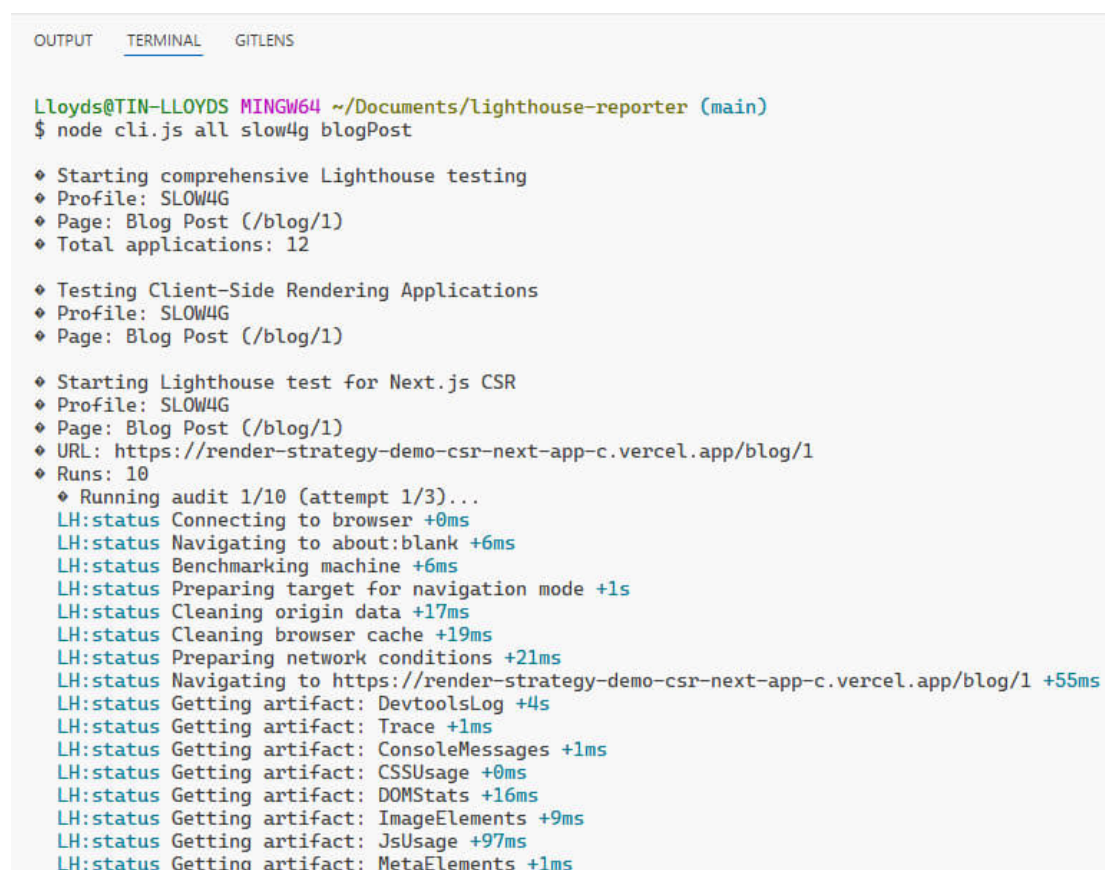
²3 aplikacije * 4 strategije iscertavanja * 10 testova po kombinaciji * 3 podstranice (about, blog, blog detalji)

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

Zbog velikog broja provedenih testiranja, bilo je nužno osigurati njihovu potpunu uspješnost i cjelovitost podataka. U 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



```
OUTPUT  TERMINAL  GITLENS

Llloyd@sTIN-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 [autorski rad]

2.4.4 Testno okruženje

Kako bi se osigurala konzistentnost pri testiranju i simulacija stvarnih uvjeta u kojima se korisnici nalaze kada koriste web aplikacije, te podrška određenih strategija iscrtavanja (poput ISR-a) odabran je slijedeći pristup.

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

Sve aplikacije postavljene su na platformu Vercel koja u trenutku pisanja ovog rada jedina podržava sve odabrane strategije iscrtavanja za svaki od odabranih programskih okvira. Ovo osigurava maksimalnu kompatibilnost i integraciju bez kompliciranih 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 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 [autorski rad]	9
3	Testiranje stranice pojedinog bloga kroz terminal [autorski rad]	13

Popis tablica