Optimizacija performansi Web aplikacija kroz analizu i razvoj programa

Kalinić, Robert

Undergraduate thesis / Završni rad

2024

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Zagreb, Faculty of Organization and Informatics / Sveučilište u Zagrebu, Fakultet organizacije i informatike**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:211:119928

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Robert Kalinić

OPTIMIZACIJA PERFORMANSI WEB APLIKACIJA KROZ ANALIZU I RAZVOJ PROGRAMA

ZAVRŠNI RAD

Varaždin, 2024.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Robert Kalinić

Matični broj: 0016156568

OPTIMIZACIJA PERFORMANSI WEB APLIKACIJA KROZ ANALIZU I RAZVOJ PROGRAMA

ZAVRŠNI RAD

Mentor:

Doc. dr. sc. Neven Vrček

Varaždin, rujan 2024.

Robert Kalinić

Izjava o izvornosti

Izjavljujem da je moj završni rad izvorni rezultat mojeg rada te da se u izradi istoga nisam koristio drugim izvorima osim onima koji su u njemu navedeni. Za izradu rada su korištene etički prikladne i prihvatljive metode i tehnike rada.

Autor potvrdio prihvaćanjem odredbi u sustavu FOI-radovi

1. Sažetak

Rad istražuje različite aspekte analize i razvoja programa u kontekstu web aplikacija s ciljem poboljšanja njihovih performansi. Fokusiramo se na identifikaciju ključnih faktora koji utječu na brzinu i efikasnost web aplikacija te primjenjujemo pristupe i tehnike programiranja kako bismo optimizirali njihovo izvođenje. Također, istražujemo alate i metode za mjerenje performansi te analiziramo stvarne primjere implementacije optimizacija.

Kroz ovu analizu i razvoj programa, cilj je stvoriti smjernice i pristupe za razvijanje bržih i efikasnijih web aplikacija.

Ključne riječi: metrike, performanse, optimizacija, tehnike, web aplikacija, server, keširanje, kod

2. Sadržaj

1. Uvod

Razvoj web aplikacija danas je jedan od ključnih aspekata modernog softverskog inženjerstva, pri čemu performanse igraju presudnu ulogu u uspjehu aplikacija na tržištu. Korisnici očekuju da se web stranice učitavaju brzo i nesmetano funkcioniraju, dok se poslovni subjekti oslanjaju na brzinu i pouzdanost svojih aplikacija kako bi osigurali zadovoljstvo korisnika, povećali konverzije i održali konkurentnost. Međutim, sa sve složenijim zahtjevima i rastućom kompleksnošću web aplikacija, optimizacija performansi postala je izazov koji zahtijeva dubinsku analizu i pažljivo planiranje.

“Without quantifiable metrics, website optimization (WSO) is a guessing game. But with hundreds of billions of e-commerce dollars at stake, most companies cannot afford to guess.” [1, str. 297]. Zato se tvrtke sve više oslanjaju na precizne analitičke alate i metode za praćenje performansi svojih web stranica kako bi maksimizirale povrat na ulaganje (ROI) i smanjile rizik od donošenja pogrešnih odluka.

Kontekst ovog istraživanja smješten je u okvire važnosti performansi u današnjim web aplikacijama. Loše performanse ne samo da dovode do frustracije korisnika, već mogu uzrokovati i ozbiljne poslovne gubitke zbog smanjene angažiranosti korisnika, povećanja troškova održavanja te negativnog utjecaja na pozicioniranje u pretraživačima (SEO).

Cilj ovog rada je istražiti i primijeniti različite metode za optimizaciju performansi web aplikacija. Istraživanje će obuhvatiti analizu performansi javno dostupnim web stranicama. Ovim pristupom nastojat će se identificirati ključni faktori koji utječu na performanse, te implementirati konkretne tehnike za njihovu optimizaciju

2. Metode i tehnike rada

Za izradu ovog rada korišten je širok spektar metoda i tehnika. Analiza performansi web aplikacija obuhvatila je pregled ključnih metričkih indikatora. Temeljito je istražena literatura, uključujući stručne članke i pouzdane izvore dostupne na internetu. Također, korištene su različite tehnike optimizacije programskog koda i evaluacija postojećih alata za mjerenje performansi, kako bi se osigurala cjelovita i detaljna analiza. Od alata korišteni su *Google Lighthouse* i *PageSpeed Insights*.

3. Analiza performansi web aplikacija

Performanse web aplikacija predstavljaju ključni faktor za uspjeh bilo koje digitalne platforme. U današnjem digitalnom dobu, korisnici su navikli na trenutni pristup informacijama, a strpljenje za sporo učitavanje stranica gotovo da više ne postoji. Usponom platformi kao što su Instagram i TikTok, koje korisnicima nude kratak, dinamičan sadržaj u obliku Reels-a i kratkih videa raspon vremena pažnje kod ljudi znatno se smanjio.

Kako se očekivanja korisnika povećavaju, brzina i responzivnost aplikacija postaju temeljne odrednice kvalitete korisničkog iskustva. Studije pokazuju da kašnjenje od samo jedne sekunde može smanjiti konverzije za čak 7%, dok brže aplikacije imaju bolje SEO rezultate i niže stope napuštanja stranica [2]. Također, istraživanje koje je proveo Google otkriva da 53% korisnika napušta mobilne web stranice koje se učitavaju duže od tri sekunde [3]. Stoga, analiza performansi postaje neizostavan korak u procesu razvoja i održavanja web aplikacija.

Osim SEO rezultata, kada se analiziraju performanse web aplikacija, najčešće se uzima u obzir nekoliko ključnih metrika koje pomažu u ocjeni korisničkog iskustva i tehničke učinkovitosti. Metrike se uglavnom dijele u dvije skupine; *Core Web Vitals* i *Other Web Vitals* [4]. Prema toj raspodijeli određene metrike su raspoređene na slijedeći način:

3. **Core Web Vitals**
    - Largest Contentful Paint (LCP)
    - Interaction to Next Paint (INP)
    - Comulative Layout Shift (CLS)
4. **Other Web Vitals**
    - First Contentful Paint (FCP)
    - Time to First Byte (TTFB)
    - Total Blocking Time (TBT)
    - Time to Interactive (TTI)
    - First Input Delay (FID)
    - Speed Index

3.1 Largest Contentful Paint (LCP)

Largest Contentful Paint (LCP) mjeri vrijeme potrebno da se najveći vidljivi element na stranici, poput slike, videa ili velikog bloka teksta, potpuno učita i prikaže u korisnikovom pregledniku. LCP smatra se ključnim pokazateljem brzine učitavanja jer direktno utječe na percepciju korisnika o tome koliko je stranica brza.

Google koristi LCP kao jednu od metrika za rangiranje stranica u svojim rezultatima pretraživanja. Web stranice s brzim LCP-om imat će prednost u rangiranju, dok sporije stranice mogu biti penalizirane.

Vrijeme za ovu metriku smatra se dobrim ako je ispod 2.5 sekundi, iako prihvatljivo je i do 4 sekunde. Sve duže od toga smatra se sporim te je potrebno poduzeti optimizacijske mjere kako bi se poboljšala brzina učitavanja ključnih elemenata na stranici, kao što su optimizacija slika, smanjenje veličine datoteka ili korištenje bržih servera i mreža za dostavu sadržaja (CDN-ova). Također, neefikasan ili blokirajući JavaScript i CSS mogu usporiti učitavanje stranice i tako produžiti LCP [5].

3.2 Interaction to Next Paint (INP)

Interaction to Next Paint (INP) je nova metrika u analizi performansi web stranica koja mjeri sveukupnu responzivnost aplikacije, odnosno koliko brzo se stranica odaziva na interakcije korisnika. INP prati koliko vremena protekne od trenutka kada korisnik izvrši interakciju (npr. klik, dodir, pritiskanje tipke) do trenutka kada stranica prikaže sljedeći „frame“ koji odražava rezultat te interakcije.

INP je ključan jer direktno utječe na to kako korisnik doživljava interakciju s aplikacijom. Ako je INP visok, korisnik može steći dojam da je aplikacija troma ili neodgovorna. Ova metrika ne mjeri samo prvi ulaz kao što to čini FID (kasnije u tekstu), već uzima u obzir sve interakcije na stranici, pružajući sveobuhvatniju sliku o responzivnosti aplikacije.

Idealno je da INP bude ispod 200 milisekundi kako bi korisničko iskustvo bilo glatko i zadovoljavajuće. Vrijednosti između 200 i 500 milisekundi smatraju se prihvatljivima, ali mogu izazvati osjećaj da je aplikacija spora. Vrijednosti iznad 500 milisekundi često rezultiraju lošim korisničkim iskustvom, jer korisnik može primijetiti kašnjenje između svoje akcije i reakcije stranice [6].

3.3 Comulative Layout Shift (CLS)

Cumulative Layout Shift (CLS) je metrika koja mjeri vizualnu stabilnost stranice tijekom učitavanja. Konkretno, CLS procjenjuje koliko se elementi na stranici pomiču dok se stranica učitava i dok korisnik već pokušava s njom interagirati. Ova metrika je ključna za osiguranje pozitivnog korisničkog iskustva, jer neočekivani pomaci elemenata mogu biti frustrirajući za korisnike, posebno kada pokušavaju kliknuti na nešto, a element iznenada promijeni poziciju.

Elementi koji se dinamički dodaju na stranicu, poput oglasa ili slika, u kombinaciji s fontovima i slikovnim elementima bez unaprijed definiranih dimenzija, mogu uzrokovati nepredvidivo pomicanje sadržaja, što negativno utječe na korisničko iskustvo. Osim toga, loše izvedene animacije koje pomiču elemente na stranici mogu dodatno pridonijeti povećanju CLS-a, što naglašava potrebu za pažljivim planiranjem i implementacijom svih vizualnih elemenata kako bi se osigurala maksimalna stabilnost stranice tijekom učitavanja.

CLS se izražava kao broj koji predstavlja zbroj svih pojedinačnih "layout shift" vrijednosti tijekom životnog vijeka stranice. Manje vrijednosti ukazuju na stabilniju stranicu, dok veće vrijednosti upućuju na potencijalne probleme s vizualnom stabilnošću.

Idealno je da CLS bude ispod 0.1 kako bi korisnici imali glatko iskustvo bez iritantnih pomicanja elemenata. Vrijednosti između 0.1 i 0.25 se smatraju prihvatljivima, ali bi mogle izazvati određenu nelagodu korisnicima. Sve iznad 0.25 obično rezultira lošim korisničkim iskustvom i zahtijeva hitnu optimizaciju [7].

3.4 First Contentful Paint (FCP)

First Contentful Paint (FCP) označava trenutak kada se prvi vidljivi element pojavi na korisnikovom ekranu tijekom učitavanja stranice. Ovaj element može biti tekst, slika, SVG ili pozadinska slika, ali ne uključuje bijeli prostor ili nevidljive dijelove stranice. FCP je kritična metrika jer daje korisnicima vizualni signal da se stranica aktivno učitava, što može značajno poboljšati percepciju brzine i smanjiti rizik od napuštanja stranice.

Preporučeno vrijeme za FCP je ispod 1.8 sekundi. Vrijednosti između 1.8 i 3 sekunde su prihvatljive, dok sve što traje dulje od 3 sekunde može izazvati frustraciju korisnika i povećati stopu napuštanja stranice [8].

3.5 Time to First Byte (TTFB)

Time to First Byte (TTFB) je metrika koja mjeri vrijeme koje protekne od trenutka kada korisnik pošalje zahtjev za stranicu do trenutka kada njegov preglednik primi prvi bajt podataka s web servera. TTFB je važan pokazatelj performansi servera i mreže jer označava koliko brzo server počinje isporučivati sadržaj nakon što primi zahtjev.

TTFB je ključan jer predstavlja prvu priliku da server odgovori na korisnikov zahtjev, čime direktno utječe na percepciju brzine i učitavanja stranice. Brz TTFB sugerira da server i mreža učinkovito obrađuju zahtjev, dok spor TTFB može uzrokovati kašnjenje u cijelom procesu učitavanja, negativno utječući na korisničko iskustvo.

Na TTFB utječu faktori poput performansi servera, brzine DNS razrješavanja, mrežne latencije, kao i vrijeme koje je potrebno serveru da obradi zahtjev i generira odgovor.

Kako bi se osigurala brza isporuka sadržaja, idealno je da TTFB ne bude veće od 200 milisekundi. Vrijednosti između 200 i 500 milisekundi su prihvatljive, ali sve iznad toga može rezultirati sporim učitavanjem stranice [9].

3.6 Total Blocking Time (TBT)

Total Blocking Time (TBT) mjeri ukupno vrijeme tijekom kojeg je glavna nit preglednika bila blokirana, sprječavajući korisnika da interagira sa stranicom. TBT se izračunava kao zbroj svih "dugih zadataka" koji traju duže od 50 milisekundi, a odvijaju se između prikazivanja First Contentful Paint (FCP) i Time to Interactive (TTI). Ovi zadaci mogu uključivati izvršavanje JavaScript-a, učitavanje resursa i druge procese koji blokiraju glavni tok preglednika, odgađajući time odgovor stranice na korisničke unose.

Na TBT najviše utječu teški JavaScript zadaci, neoptimizirani kodovi, te veliki resursi koji blokiraju glavnu nit. Kako bi se smanjio TBT, važno je optimizirati i minimizirati JavaScript, koristiti asinkrono učitavanje resursa, te osigurati da se teški zadaci podijele na manje dijelove ili se izvršavaju izvan glavne niti.

Za TBT također bi bilo idealno da bude ispod 200 milisekundi. Vrijednosti između 200 i 600 milisekundi su prihvatljive, ali mogu uzrokovati primjetno kašnjenje u interakcijama. Sve iznad 600 milisekundi može značajno utjecati na korisničko iskustvo, pa bi bilo potrebno optimizirati stranicu kako bi se smanjio TBT [10].

3.7 Time to Interactive (TTI)

Time to Interactive (TTI) je metrika koja mjeri vrijeme potrebno da stranica postane potpuno interaktivna, odnosno kada je korisnik može nesmetano koristiti bez ikakvih kašnjenja u odgovorima. TTI se računa od trenutka kada se stranica počne učitavati do trenutka kada je glavni sadržaj stranice prikazan i spreman za interakciju, a svi dugotrajni zadaci su završeni. To znači da su svi resursi učitani, JavaScript izvršen, i stranica reagira na korisničke unose u stvarnom vremenu.

TTI je izuzetno važan za korisničko iskustvo jer označava trenutak kada korisnici mogu početi nesmetano koristiti aplikaciju ili web stranicu. Ako je TTI visok, korisnici mogu biti frustrirani zbog kašnjenja između učitavanja stranice i trenutka kada postane potpuno funkcionalna, što može dovesti do napuštanja stranice i negativnog utiska o njezinoj performansi.

Vrijeme do interaktivnosti trebalo bi biti ispod 5 sekundi kako bi korisnici mogli brzo započeti s interakcijom. Vrijednosti između 5 i 10 sekundi su još uvijek prihvatljive, ali mogu negativno utjecati na korisničko iskustvo. Vrijeme do interaktivnosti duže od 10 sekundi smatra se lošim te je potrebno primijeniti tehnike optimizacije kako bi se isto smanjilo [11].

3.8 First Input Delay (FID)

First Input Delay (FID) je metrika koja mjeri vrijeme između prve interakcije korisnika sa stranicom (npr. klik na gumb, link ili drugo interaktivno područje) i trenutka kada preglednik počne procesuirati taj događaj. FID je ključan jer pokazuje koliko je stranica responzivna na korisničke unose, a time i koliko je korisničko iskustvo fluidno i ugodno.

FID je posebno važan na stranicama koje zahtijevaju brzu interakciju, poput formulara, trgovina ili aplikacija. Ako je FID visok, korisnici će osjetiti kašnjenje prilikom pokušaja interakcije, što može rezultirati frustracijom i smanjenjem angažmana.

FID bi trebao biti ispod 100 milisekundi kako bi korisničko iskustvo bilo optimalno. Vrijednosti između 100 i 300 milisekundi su prihvatljive, ali mogu uzrokovati primjetna kašnjenja u interakcijama. Vrijednosti iznad 300 milisekundi često rezultiraju lošijim korisničkim iskustvom [12].

3.9 Speed Index

Speed Index je metrika koja mjeri koliko brzo se vizualni sadržaj stranice prikazuje korisniku tijekom učitavanja. Konkretno, Speed Index izračunava prosječno vrijeme potrebno da se svi vidljivi dijelovi stranice učitaju u pregledniku, uzimajući u obzir brzinu i progresivnost prikazivanja sadržaja. Niža vrijednost Speed Index-a ukazuje na brže prikazivanje stranice, što obično znači bolje korisničko iskustvo.

Speed Index je važan jer pruža sveobuhvatan pregled koliko brzo korisnici vide sadržaj stranice. Brži Speed Index ukazuje na to da je sadržaj stranice vidljiv u kratkom vremenu, što smanjuje percepciju sporog učitavanja i poboljšava zadovoljstvo korisnika. Ako je Speed Index visok, korisnici mogu imati osjećaj da se stranica učitava sporo, čak i ako su pojedine komponente stranice već učitane.

Speed Index bi trebao biti što manji, s ciljem da bude ispod 3 sekunde, kako bi se osigurala brza i neprekidna vizualna prezentacija stranice. Vrijednosti između 3 i 5 sekundi su još uvijek prihvatljive, no mogu utjecati na korisničku percepciju brzine učitavanja stranice. Ako Speed Index prelazi 5 sekundi, to često ukazuje na potrebu za optimizacijom, jer dulje vrijeme može značajno smanjiti zadovoljstvo korisnika i povećati stopu napuštanja stranice [13].

4. Faktori koji utječu na performanse

Performanse web aplikacija mogu biti pogođene brojnim tehničkim i infrastrukturnim aspektima. Od brzine servera i mrežne latencije, preko optimizacije resursa kao što su slike, JavaScript i CSS, do tehnika keširanja i učinkovitosti koda – svi ovi faktori igraju značajnu ulogu u ukupnoj brzini i efikasnosti aplikacije.

U okviru ovog poglavlja, razmotrit ćemo ključne čimbenike koji utječu na performanse web aplikacija, uključujući performanse servera, mrežnu brzinu, optimizaciju resursa, učitavanje i renderiranje stranica, tehnike keširanja, i optimizaciju koda [14] [15]. Analizirat ćemo kako svaki od tih faktora doprinosi cjelokupnim performansama i pružiti smjernice za njihovu optimizaciju kako bi se osiguralo brzo i učinkovito korisničko iskustvo.

4.1 Performanse servera

Performanse servera predstavljaju jedan od ključnih čimbenika koji direktno utječu na brzinu i efikasnost web aplikacija. Server je srce svake web aplikacije, jer obrađuje zahtjeve korisnika, generira odgovore i šalje podatke natrag korisnicima. Ako server ne funkcionira optimalno, cijela aplikacija može trpjeti, što rezultira sporim učitavanjem stranica, povećanim vremenom odgovora i nezadovoljstvom korisnika. Ključni aspekti performansi servera uključuju brzinu odziva, kapacitet obrade i optimizaciju servera.

4.1.1 Brzina odziva

Brzina odziva servera odnosi se na vrijeme potrebno da server odgovori na zahtjev korisnika. Ovaj aspekt performansi servera mjeri se kroz metrike poput Time to First Byte (TTFB), koja označava vrijeme potrebno da prvi bajt podataka stigne do korisnika nakon što je zahtjev poslan. Brzina odziva ovisi o nekoliko faktora, uključujući:

- **Hardverske specifikacije:** Brzina procesora, količina RAM-a i brzina diska.
- **Konfiguracije softvera**: Optimizacija web servera (npr. Nginx, Apache), baze podataka (MySQL, PostgreSQL) i aplikacijskog koda.
- **Opterećenje servera**: Broj istovremenih korisnika i zahtjeva koji server mora obraditi.

4.1.2 Kapacitet obrade

Kapacitet obrade servera odnosi se na sposobnost servera da učinkovito upravlja velikim brojem zahtjeva i istovremenih korisnika. Ovo uključuje:

- **Skalabilnost servera:** Mogućnost servera da poveća svoje resurse u slučaju povećanog opterećenja. To se može postići vertikalnom skalabilnošću (dodavanjem više resursa pojedinačnom serveru) ili horizontalnom skalabilnošću (dodavanjem više servera u klaster)
- **Balansiranje opterećenja:** Korištenje tehnika poput balansiranja opterećenja (load balancing) omogućava raspodjelu zahtjeva na više servera, smanjujući opterećenje pojedinog servera i poboljšavajući performanse.
- **Optimizacija baze podataka:** Učinkovito upravljanje bazama podataka, uključujući korištenje indeksa, optimizaciju upita i keširanje rezultata, može značajno povećati kapacitet obrade.

4.1.3 Optimizacija servera

Optimizacija servera uključuje niz tehnika i alata koji mogu poboljšati performanse servera, smanjujući vrijeme odziva i povećavajući kapacitet obrade. To uključuje:

- **Keširanje na strani servera:** Korištenje keširanja omogućava serveru da pohranjuje rezultate čestih upita i isporučuje ih brže, bez potrebe za ponovnim izvršavanjem zahtjeva. Popularni alati za keširanje uključuju Varnish i Memcached.
- **Kompresija sadržaja:** Kompresija statičkog sadržaja, poput HTML, CSS i JavaScript datoteka, pomoću alata kao što su Gzip ili Brotli, može smanjiti veličinu podataka koje server šalje korisnicima, čime se ubrzava vrijeme učitavanja stranice.
- **Optimizacija aplikacijskog koda:** Pojednostavljenje i optimizacija koda aplikacije, smanjenje složenosti i eliminacija nepotrebnih procesa može značajno smanjiti opterećenje servera.

4.2 Brzina mreže

Brzina mreže igra ključnu ulogu u performansama web aplikacija, jer utječe na vrijeme potrebno za prijenos podataka između servera i korisnika. Iako brzina servera i optimizacija aplikacije mogu biti besprijekorni, mrežna infrastruktura kroz koju podaci prolaze može značajno utjecati na krajnje vrijeme učitavanja i korisničko iskustvo. Ključni faktori koji određuju brzinu mreže uključuju latenciju, propusnost i udaljenost između korisnika i servera.

4.2.1 Latencija

Latencija se odnosi na vrijeme koje je potrebno da podaci putuju od korisnika do servera i natrag. Ovo kašnjenje može biti uzrokovano različitim faktorima, uključujući:

- **Fizička udaljenost:** Što je veća udaljenost između korisnika i servera, to će latencija biti veća. Ova udaljenost uzrokuje kašnjenje u prijenosu podataka, što može usporiti vrijeme odziva aplikacije.
- **Kvaliteta mreže:** Kvaliteta internetske infrastrukture, uključujući brzinu internetske veze koju koristi korisnik, može značajno utjecati na latenciju. Slabije veze, poput onih s velikim brojem međupostaja ili nestabilnih veza, mogu povećati latenciju.
- **Rješenje za smanjenje latencije:** Korištenje mreža za dostavu sadržaja (CDN) može smanjiti latenciju tako što se sadržaj distribuira kroz mrežu servera bližih krajnjim korisnicima. Time se smanjuje udaljenost koju podaci moraju prijeći, ubrzavajući isporuku sadržaja.

4.2.2 Propusnost

Propusnost mreže odnosi se na količinu podataka koja se može prenijeti kroz mrežu u određenom vremenskom razdoblju, obično mjerenu u megabitima po sekundi (Mbps). Propusnost utječe na:

- **Brzinu prijenosa velikih datoteka:** Veća propusnost omogućuje brži prijenos velikih datoteka, kao što su slike visoke rezolucije, videozapisi i drugi multimedijski sadržaji.
- **Kapacitet mreže:** Mreže s višom propusnošću mogu podnijeti veći broj istovremenih korisnika bez smanjenja brzine prijenosa podataka, što je ključno za skalabilnost web aplikacija.

- **Uslužni paketi:** Brzina internetske veze korisnika često ovisi o paketu koji je odabrao kod svog davatelja internetskih usluga (ISP). Korisnici s većim paketima propusnosti imat će brži prijenos podataka, dok će korisnici s nižim paketima doživjeti sporije učitavanje sadržaja.

4.2.3 Udaljenost između korisnika i servera

Udaljenost između korisnika i servera značajno utječe na vrijeme potrebno za prijenos podataka. Ova udaljenost može uzrokovati povećanje latencije i smanjenje brzine učitavanja, jer podaci moraju prijeći veći put.

4.3 Optimizacija resursa

Optimizacija resursa odnosi se na proces smanjenja veličine i broja elemenata koje web aplikacija treba učitati kako bi se poboljšale performanse i ubrzalo vrijeme učitavanja stranice. Ovo je ključna komponenta u optimizaciji web aplikacija, jer značajno smanjuje opterećenje na mrežnu infrastrukturu, server i korisnički uređaj, što rezultira bržim i fluidnijim korisničkim iskustvom. Glavni resursi koji zahtijevaju optimizaciju su slike, CSS i JavaScript datoteke, kao i fontovi i multimedijalni sadržaji.

4.3.1 Optimizacija slika

Slike su često najveći pojedinačni resursi koje web stranice učitavaju, pa njihova optimizacija može donijeti značajne prednosti u smislu performansi:

- **Komprimiranje slika:** Smanjenje veličine datoteka slika bez gubitka kvalitete pomoću alata za kompresiju poput TinyPNG, ImageOptim, ili korištenje modernih formata slika kao što su WebP i AVIF može značajno smanjiti vrijeme učitavanja.
- **Lazy loading:** Korištenje tehnike "lazy loading" omogućava učitavanje slika samo kada postanu vidljive na korisnikovom ekranu, što smanjuje početno vrijeme učitavanja stranice.
- **Prilagođavanje dimenzija:** Korištenje slika optimiziranih za različite veličine ekrana i rezolucije, kako bi se izbjeglo učitavanje većih slika nego što je potrebno.

4.3.2 Optimizacija CSS i JavaScript datoteka

CSS i JavaScript datoteke igraju ključnu ulogu u izgledu i funkcionalnosti web stranice, ali mogu značajno povećati vrijeme učitavanja ako nisu pravilno optimizirane:

- **Minifikacija:** Proces uklanjanja nepotrebnih znakova iz CSS i JavaScript datoteka (poput razmaka, komentara i novih redova) kako bi se smanjila njihova veličina. Alati poput UglifyJS za JavaScript i CSSNano za CSS pomažu u ovom procesu.
- **Kombiniranje datoteka:** Spajanje više CSS ili JavaScript datoteka u jednu može smanjiti broj HTTP zahtjeva, što ubrzava učitavanje stranice.
- **Asinkrono učitavanje:** Učitavanje JavaScript datoteka asinkrono omogućuje da se sadržaj stranice učita bez čekanja na izvršavanje skripti.

4.3.3 Optimizacija fontova

Fontovi mogu značajno utjecati na performanse web stranice, osobito ako se učitavaju s vanjskih izvora:

- **Preload fontova:** Korištenje `preload` atributa omogućuje preglednicima da unaprijed učitaju ključne fontove, što smanjuje vrijeme potrebno za prikazivanje teksta na stranici.
- **Korištenje lokalnih fontova:** Umjesto učitavanja fontova s vanjskih izvora, razmatranje upotrebe lokalnih fontova može smanjiti vrijeme učitavanja.
- **Optimizacija WOFF/WOFF2:** Korištenje modernih formata fontova poput WOFF2, koji su komprimirani i optimizirani za web, može smanjiti vrijeme učitavanja.

4.4 Učitavanje i renderiranje stranice

Učitavanje i renderiranje stranice odnosi se na proces kojim se web stranica prenosi iz servera na korisnikov uređaj i prikazuje na ekranu. Ovaj proces uključuje preuzimanje resursa, interpretaciju i izvršavanje koda te konačno prikazivanje sadržaja na korisničkom sučelju. Brzina i učinkovitost ovog procesa ključni su za pružanje dobrog korisničkog iskustva, jer sporo učitavanje ili nepravilno renderiranje može rezultirati gubitkom korisnika i smanjenjem konverzija.

4.4.1 Faze učitavanja stranice

Učitavanje stranice može se podijeliti u nekoliko ključnih faza:

- **DNS pretraživanje:** Kada korisnik zatraži web stranicu, preglednik prvo mora prevesti URL u IP adresu koristeći DNS (Domain Name System). Ovaj proces može uzeti nekoliko milisekundi do nekoliko sekundi, ovisno o mrežnoj infrastrukturi i udaljenosti do DNS servera.
- **Uspostavljanje veze:** Nakon što je IP adresa poznata, preglednik uspostavlja vezu sa serverom putem TCP protokola. Ako se koristi HTTPS, dodatna faza SSL/TLS pregovora dodaje vrijeme uspostavi veze, ali osigurava sigurnu komunikaciju.
- **Slanje HTTP zahtjeva:** Preglednik šalje HTTP zahtjev serveru za preuzimanje HTML dokumenta. Server obrađuje zahtjev i vraća odgovor, koji uključuje početni HTML sadržaj stranice.
- **Učitavanje resursa:** Kada preglednik primi početni HTML dokument, on analizira i počinje učitavati dodatne resurse poput CSS-a, JavaScript-a, slika i fontova. Preglednik paralelno učitava ove resurse kako bi ubrzao proces učitavanja stranice.

4.4.2 Proces renderiranja

Nakon što su resursi učitani, preglednik započinje proces renderiranja, koji uključuje:

- **Parsiranje HTML-a i CSS-a:** Preglednik analizira (parsira) HTML i CSS datoteke kako bi izgradio DOM (Document Object Model) i CSSOM (CSS Object Model) stablo. Ova stabla predstavljaju strukturu i stilove stranice.
- **Izgradnja render stabla:** DOM i CSSOM stabla kombiniraju se kako bi se izgradilo render stablo, koje sadrži samo one elemente koji će biti prikazani na stranici. Ovo stablo ne uključuje nevidljive elemente ili elemente sa display: none.
- **Layout:** Preglednik određuje položaj i dimenzije svakog elementa na stranici na temelju render stabla. Ova faza također uzima u obzir različite stilove i veličine ekrana kako bi osigurao ispravno prikazivanje sadržaja.
- **Painting:** Nakon što je layout završen, preglednik započinje "bojanje" stranice, što znači crtanje piksela na ekranu prema informacijama iz render stabla.
- **Compositing:** Posljednja faza je compositing, gdje preglednik sastavlja različite slojeve sadržaja i prikazuje ih na ekranu. Ova faza može uključivati rješavanje složenih animacija, transformacija i prelaza.

4.4.3 Faktori koji utječu na učitavanje i renderiranje stranice

Nekoliko faktora koji utječu na brzinu i učinkovitost učitavanja i renderiranja stranice:

- **Veličina i broj resursa:** Veći broj resursa ili velike datoteke mogu značajno usporiti učitavanje stranice. Optimizacija resursa, kao što su kompresija slika i minifikacija koda, može pomoći u ubrzavanju ovog procesa.
- **Blokirajući JavaScript:** JavaScript datoteke koje nisu učitane asinkrono mogu blokirati parsiranje HTML-a i odgoditi renderiranje stranice. Upotreba async ili defer atributa na skriptama može spriječiti takva kašnjenja.
- **Kaskadno učitavanje CSS-a:** CSS datoteke također mogu blokirati renderiranje stranice dok se ne učitaju i analiziraju. Korištenje kritičnog CSS-a (Critical CSS) koji se odmah učitava, dok se ostatak CSS-a učitava kasnije, može ubrzati prikazivanje početnog sadržaja.
- **Renderiranje na strani klijenta:** Kod aplikacija koje koriste JavaScript frameworke kao što su React ili Angular, renderiranje se često vrši na strani klijenta. Iako ovo omogućava bogato korisničko iskustvo, može dodati kašnjenja u početnom učitavanju stranice.

4.5 Tehnike keširanja

Keširanje je jedna od najefikasnijih tehnika za poboljšanje performansi web aplikacija. Keširanje omogućava spremanje često korištenih podataka na klijentskoj strani, serveru ili između njih, kako bi se smanjila potreba za ponovnim dohvaćanjem istih podataka svaki put kada se učitava stranica.

4.5.1 Klijentsko keširanje (Browser caching)

Klijentsko keširanje se odnosi na pohranu podataka u pregledniku korisnika, omogućavajući da se resursi (poput slika, CSS, i JavaScript datoteka) učitavaju iz lokalne memorije umjesto ponovnog preuzimanja s servera. Ovo je posebno korisno kod ponovnih posjeta stranici, kada većina sadržaja može biti preuzeta iz keša.

- **Cache-Control header:** Postavljanjem Cache-Control zaglavlja u HTTP odgovoru, server može specificirati koliko dugo preglednik treba čuvati resurs u kešu. Vrijednosti poput max-age, public, i private definiraju trajanje i vidljivost keša.

- **Expires header:** Koristi se za postavljanje datuma nakon kojeg se resurs smatra zastarjelim, potičući preglednik da preuzme novu verziju s servera.
- **ETag header:** ETag (Entity Tag) je jedinstveni identifikator koji omogućava serveru da prepozna je li se određeni resurs promijenio od posljednjeg učitavanja. Ako se resurs nije promijenio, server može odgovoriti da koristi verziju iz keša.

4.5.2 Server-side keširanje

Keširanje na strani servera uključuje pohranu dinamički generiranih podataka kako bi se smanjio broj zahtjeva prema bazi podataka i ubrzalo generiranje odgovora. Ova tehnika je posebno korisna za dinamičke web aplikacije koje često dohvaćaju podatke iz baza podataka ili izvršavaju složene procese.

- **Keširanje upita baze podataka:** Ova tehnika pohranjuje rezultate upita u keš, smanjujući potrebu za ponovnim izvršavanjem istih upita svaki put kada se učitava stranica. Alati poput Redis ili Memcached često se koriste za ovu svrhu.
- **Keširanje cijelih stranica:** U ovoj tehnici, cijela HTML stranica ili njezini dijelovi se pohranjuju u keš, smanjujući potrebu za ponovnim generiranjem stranice svaki put kada korisnik učita stranicu.
- **Keširanje API odgovora:** API odgovori se također mogu keširati kako bi se smanjilo opterećenje na serveru i ubrzao proces dohvaćanja podataka u aplikacijama koje koriste RESTful ili GraphQL API-je.

4.5.3 CDN keširanje

Content Delivery Network (CDN) keširanje koristi distribuiranu mrežu servera za pohranu i isporuku resursa korisnicima na osnovu njihove geografske lokacije. Ovo značajno smanjuje vrijeme potrebno za isporuku sadržaja i poboljšava performanse web stranica globalno.

- **Distribuirani keš:** CDN mreže pohranjuju kopije statičkih resursa (poput slika, videozapisa, CSS i JavaScript datoteka) na različitim lokacijama širom svijeta. Kada korisnik zatraži sadržaj, resursi se isporučuju s najbližeg CDN servera, smanjujući latenciju i vrijeme učitavanja.
- **Edge caching:** Ova vrsta keširanja omogućuje pohranu sadržaja na rubnim (edge) serverima CDN mreže, koji su geografski najbliži korisnicima. Ovo smanjuje potrebu za povratkom na izvorni server za svaki zahtjev.

5. Tehnike optimizacije programskog koda

Nakon što smo identificirali ključne faktore koji utječu na performanse web aplikacija i razumjeli kako ti faktori mogu negativno utjecati na brzinu, responzivnost i ukupno korisničko iskustvo, sljedeći korak je istražiti i primijeniti učinkovite tehnike optimizacije programskog koda. Optimizacija koda igra ključnu ulogu u poboljšanju performansi web aplikacija jer omogućuje smanjenje vremena učitavanja, potrošnje resursa te poboljšanje efikasnosti i skalabilnosti aplikacije.

Ove tehnike uključuju niz strategija koje ciljaju različite aspekte razvoja i implementacije aplikacije, od minimizacije i kombiniranja datoteka, preko asinkronog učitavanja i optimizacije JavaScript-a i CSS-a, pa sve do sofisticiranih metoda keširanja i optimizacije baza podataka [16]. Razumijevanje i primjena ovih tehnika omogućava programerima da prevladaju prepreke vezane uz performanse te stvore aplikacije koje pružaju brzo, glatko i zadovoljavajuće korisničko iskustvo.

Optimizacija programskog koda nije samo tehnička nužnost, već i strateški pristup koji može značajno poboljšati konkurentsku poziciju web aplikacije na tržištu. Brze i efikasne aplikacije ne samo da poboljšavaju zadovoljstvo korisnika, već također mogu dovesti do boljih SEO rezultata, povećanja stope konverzije i smanjenja stope napuštanja stranica. Stoga, duboko razumijevanje i vješta primjena ovih tehnika postaje neizostavan dio procesa razvoja modernih web aplikacija.

Nekoliko ključnih tehnika optimizacije programskog koda:

- Minifikacija i kombinacija datoteka
- Lazy loading
- Optimizacija JavaScript-a i CSS-a
- Upravljanje predmemorijom
- Optimizacija baze podataka
- Refaktoriranje i modularizacija koda

5.1 Minifikacija i kombinacija datoteka

Minifikacija, ili minimizacija, je postupak kojim se iz JavaScript koda uklanjaju svi nepotrebni znakovi, poput razmaka, komentara i točaka sa zarezom, bez promjene

funkcionalnosti samog koda. Također, ovaj proces uključuje korištenje kraćih naziva za varijable i funkcije kako bi se smanjila ukupna veličina datoteke [17] [18. str. 141].

```css
/* Stilizacija osnovnih elemenata stranice */
body {
    background-color: #f5f5f5;
    font-family: Arial, sans-serif;
    margin: 0;
    padding: 0;
}

/* Ovdje stiliziramo h1 atribut */
h1 {
    color: #333;
    text-align: center;
    margin-top: 20px;
}

.button {
    display: inline-block;
    padding: 10px 20px;
    color: #fff;
    background-color: #007bff;
}
```

Slika 1. CSS kod prije minifikacije [autorski rad]

```
body{background-color:#f5f5f5;font-family:Arial,sans-serif;margin:0;padding:0;}h1{color: #333;text-align
```

Slika 2. CSS kod nakon minifikacije [autorski rad]

Kao što možemo vidjeti, prije minifikacije kod se sastojao od 21 linije koda, dok se nakon minifikacije sve svelo u jednu liniju. Komentari i razmaci su uklonjeni te je sve spojeno. Minifikacijom smo primjetno smanjili čitljivost koda, ali poboljšali performanse što čini posjetitelje stranice, a i tražilice, sretnijima.

Spajanje više CSS i JavaScript datoteka u jednu je tehnika optimizacije web stranica koja ima za cilj smanjenje broja HTTP zahtjeva koje preglednik mora napraviti prilikom učitavanja stranice. Manji broj zahtjeva znači manje vremena provedenog u komunikaciji između preglednika i servera, što rezultira bržim učitavanjem stranice. Kada stranica koristi više različitih CSS datoteka, poput reset.css, main.css i responsive.css, sve ove datoteke mogu se spojiti u jednu, npr. styles.css.

Slično se može primijeniti i na JavaScript datoteke. Ako se, primjerice, koriste datoteke jquery.js, plugins.js i main.js, one se mogu kombinirati u jednu datoteku poput scripts.js. Ovo smanjenje broja zahtjeva ne samo da ubrzava učitavanje stranice, već i smanjuje latenciju.

Umjesto pristupa gdje se koriste zasebne datoteke za svaki stil ili skriptu:

```
<link rel="stylesheet" href="reset.css">
<link rel="stylesheet" href="main.css">
<link rel="stylesheet" href="responsive.css">
```

Preporučljivo je kombinirati sve stilove u jednu datoteku, primjerice:

```
<link rel="stylesheet" href="styles.css">
```

Isto se odnosi i na JavaScript datoteke. Umjesto učitavanja više pojedinačnih datoteka:

```
<script src="jquery.js"></script>
<script src="plugins.js"></script>
<script src="main.js"></script>
```

Poželjno je sve skripte kombinirati u jednu datoteku:

```
<script src="scripts.js"></script>
```

Zaključno, spajanje CSS datoteka i JavaScript datoteka donosi značajne prednosti u smislu poboljšanja performansi web stranica. Smanjenjem broja HTTP zahtjeva i smanjenjem latencije, postiže se brže učitavanje stranice, što izravno utječe na kvalitetu korisničkog iskustva. Iako spajanje datoteka nudi mnoge prednosti, važno je pažljivo pratiti verzioniranje i izbjeći moguće konflikte unutar kombiniranih datoteka kako bi se osigurala stabilnost i funkcionalnost aplikacije.

5.2 Lazy loading

Lazy loading tehnika je optimizacije web stranica koja odgađa učitavanje resursa poput slika, videa ili čak cijelih dijelova stranice, sve dok oni ne postanu potrebni korisniku. Umjesto da se svi elementi stranice učitavaju odmah prilikom posjete, lazy loading omogućuje da se resursi učitaju tek kada su zaista potrebni, primjerice kada se korisnik pomakne prema dolje do dijela stranice gdje se nalaze ti resursi.

Kod lazy loadinga, slike i videozapisi učitavaju se tek kada korisnik dođe u njihov vidokrug. Na taj način, resursi koji se nalaze niže na stranici ne opterećuju mrežu odmah prilikom otvaranja stranice. Cijeli dijelovi stranice ili određeni elementi, poput widgeta ili reklamnih bannera, mogu se učitati kasnije, kada postanu relevantni za korisnika.

Recimo da na stranici ima niz slika. Umjesto standardnog načina učitavanja svih slika:

```
<img src="large-image.jpg" alt="Example Image">
```

Dodaje se atribut `loading="lazy"` kako bi se omogućio lazy loading:

```
<img src="large-image.jpg" alt="Example Image" loading="lazy">
```

Budući da se samo resursi koji su odmah vidljivi učitavaju na početku, stranica se brže učitava, što poboljšava performanse i korisničko iskustvo. Korisnici neće preuzeti nepotrebne resurse, što smanjuje ukupnu potrošnju podataka, posebno važno za korisnike s ograničenim ili sporim internetom. Dodatno, brže učitavanje stranice može pozitivno utjecati na SEO, jer brzina stranice igra ulogu u rangiranju na pretraživačima.

5.3 Optimizacija JavaScripta i CSS-a

Optimizacija JavaScripta i CSS-a igra ključnu ulogu u poboljšanju performansi web stranica, jer direktno utječe na brzinu učitavanja i responzivnost. JavaScript i CSS kod, ako je neoptimiziran, može značajno usporiti učitavanje stranice, povećati latenciju i smanjiti kvalitetu korisničkog iskustva.

U optimizaciji JavaScripta, osim već spomenutih tehnika poput minifikacije i spajanja datoteka, važno je razmotriti i asinkrono učitavanje skripti. Korištenjem atributa `async` ili `defer` prilikom učitavanja JavaScript datoteka, možemo osigurati da se skripte učitavaju bez blokiranja prikaza stranice. `Async` omogućava paralelno učitavanje i izvršavanje skripti, dok `defer` odgađa izvršavanje skripti dok se cijela stranica ne učita, čime se poboljšava brzina i responzivnost.

Još jedna ključna tehnika je uklanjanje nepotrebnog JavaScript koda. Korištenjem alata za "tree shaking" možemo eliminirati neiskorištene dijelove koda iz JavaScript datoteka. To smanjuje veličinu datoteka i poboljšava performanse, jer stranica preuzima i izvršava samo ono što je zaista potrebno.

Slično tome, kod optimizacije CSS-a, pored minifikacije i spajanja datoteka, važno je implementirati kritični CSS. Kritični CSS odnosi se na stilove koji su potrebni za inicijalni prikaz sadržaja stranice, dok se ostatak stilova učitava kasnije. Ovo značajno ubrzava prikaz stranice, jer se korisnicima odmah prikazuje osnovni sadržaj, dok se ostatak učitava u pozadini.

Također, treba izbjegavati blokirajući CSS koji može odgoditi prikaz stranice dok se svi stilovi ne učitaju. Strateško smještanje i strukturiranje CSS-a može spriječiti ovakvo blokiranje i omogućiti brže učitavanje stranice.

Optimizacija JavaScripta i CSS-a nije samo tehnički zadatak, već ključni korak prema pružanju boljeg korisničkog iskustva. Smanjenjem veličine datoteka i korištenjem tehnika koje smanjuju vrijeme učitavanja, možemo značajno poboljšati performanse naših web stranica.

```
function dohvatiPodatke() {
    const xhr = new XMLHttpRequest();
    xhr.open('GET', 'https://api.primjer.com/data', false);
    xhr.send();
    if (xhr.status === 200) {
        console.log('Dohvaćeni podaci:', xhr.responseText);
    } else {
        console.log('Greška pri dohvaćanju podataka');
    }
}

dohvatiPodatke();
```

Slika 3. Primjer neasinkrone funkcije za dohvaćanje podataka [autorski rad]

```
async function dohvatiPodatke() {
    try {
        const odgovor = await fetch('https://api.primjer.com/data');
        if (odgovor.ok) {
            const podaci = await odgovor.text();
            console.log('Dohvaćeni podaci:', podaci);
        } else {
            console.log('Error fetching data');
        }
    } catch (error) {
        console.log('Greška:', error);
    }
}

dohvatiPodatke();
```

Slika 4. Primjer iste funkcije, ali asinkrone [autorski rad]

5.4 Upravljanje predmemorijom

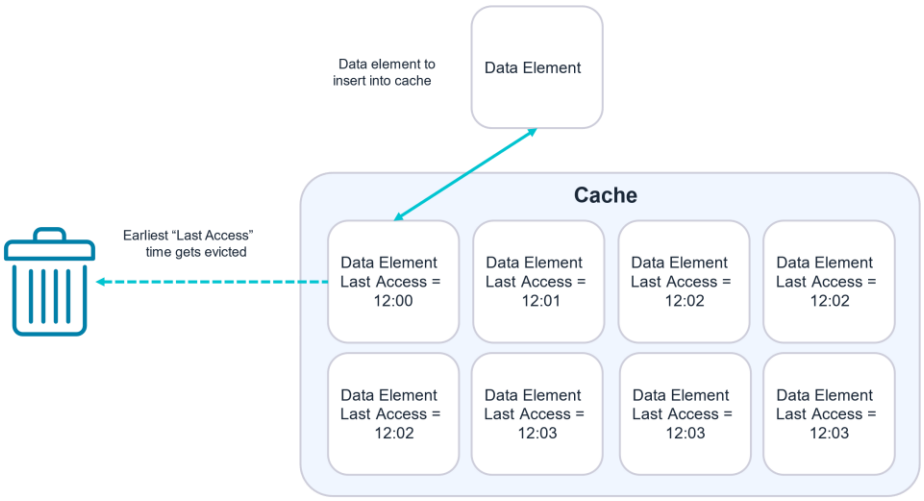
Upravljanje predmemorijom (keširanje) predstavlja ključnu komponentu optimizacije performansi web aplikacija i računalnih sustava. Predmemorija omogućuje brzo i učinkovito pristupanje često korištenim podacima, smanjujući potrebu za ponovnim dohvaćanjem ili regeneriranjem tih podataka. No, s obzirom na ograničeni kapacitet predmemorije, izuzetno je važno primijeniti odgovarajuće strategije upravljanja koje određuju koje podatke treba zadržati, a koje ukloniti iz predmemorije kada ona postane puna.

Postoji nekoliko strategija upravljanja predmemorijom, od kojih svaka ima svoje prednosti i slabosti te je prikladna za različite situacije. Među najpoznatijim su strategije poput Least Recently Used (LRU), Least Frequently Used (LFU), i First In, First Out (FIFO) [19].

5.4.1 Least Recently Used (LRU)

Ova strategija uklanja resurse koji su najmanje korišteni u nedavnom periodu kako bi napravila mjesto za nove resurse. Ideja je da se pretpostavlja da resursi koji nisu korišteni neko vrijeme vjerovatno neće biti korišteni ni u skorijoj budućnosti.

Zamislimo scenarij u kojem predmemorija može pohraniti četiri elementa. U slučaju da se dodaje peti element, algoritam uklanja onaj kojem se najdulje nije pristupalo kako bi napravio mjesta za novi. Ova strategija je osobito korisna u situacijama kada se očekuje da će se nedavno korišteni podaci ponovno koristiti u bliskoj budućnosti.

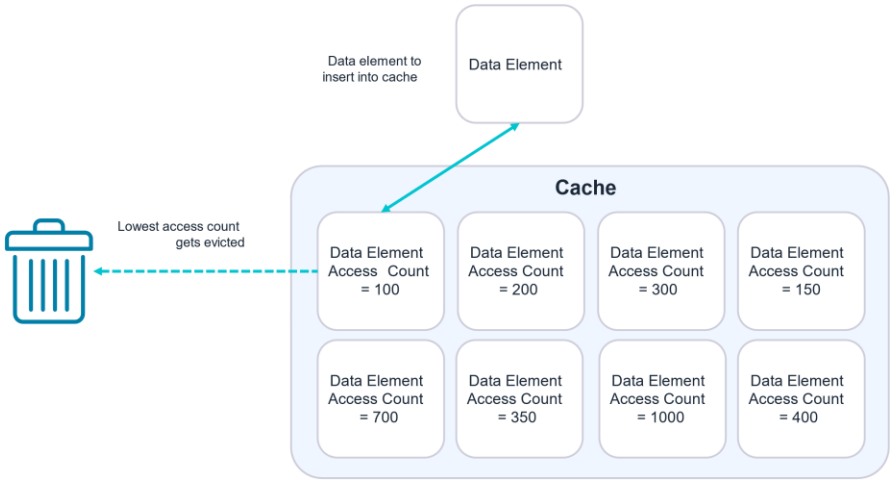


Slika 5. Slikovni prikaz LRU tehnike keširanja [19]

5.4.2 Least Frequently Used (LFU)

Least Frequently Used (LFU) strategija temelji se na broju pristupa određenim elementima u predmemoriji. Ideja je ukloniti one elemente kojima se najmanje puta pristupalo, jer se pretpostavlja da će se njima i u budućnosti rijetko pristupati. LFU prati koliko je puta svaki element korišten i kada predmemorija postane puna, uklanja element s najmanjim brojem pristupa.

Na primjer, u slučaju predmemorije koja može pohraniti četiri elementa, LFU algoritam će pri dodavanju petog elementa ukloniti onaj kojem je najmanje puta pristupano. Ova strategija pokazuje svoju učinkovitost u situacijama gdje je ključno zadržati često korištene podatke u predmemoriji što je dulje moguće.

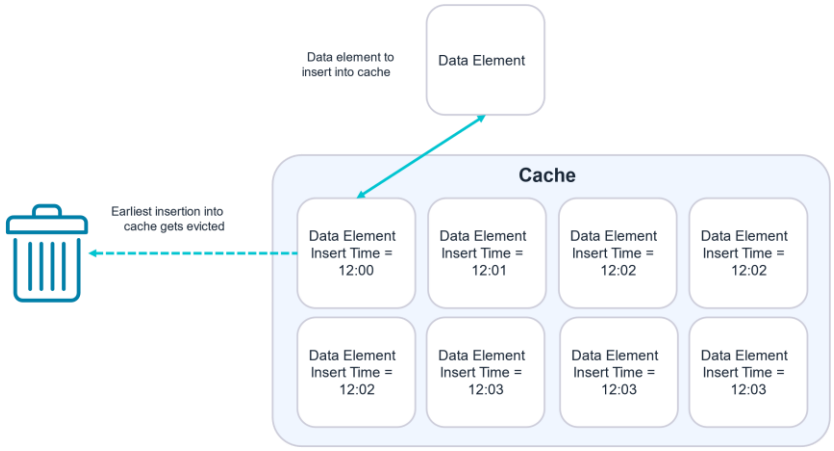


Slika 6. Slikovni prikaz LFU tehnike keširanja [19]

5.4.3 First In, First Out (FIFO)

First In, First Out (FIFO) je jedna od najjednostavnijih strategija upravljanja predmemorijom. Kao što naziv sugerira, Ova strategija uklanja najnoviji resurs iz keša kada treba osloboditi prostor. Posljednji pohranjeni resurs bit će prvi uklonjen..

Ako imamo predmemoriju koja može pohraniti četiri elementa, i ako dodamo peti element, FIFO algoritam će ukloniti onaj element koji je prvi dodan u predmemoriju. Ova strategija je prikladna za jednostavne scenarije gdje su podaci jednako važni i nema potrebe za praćenjem učestalosti ili redoslijeda pristupa. Međutim, FIFO može dovesti do suboptimalnih rezultata jer može ukloniti često korištene podatke samo zato što su najdulje u predmemoriji.



Slika 7. Slikovni prikaz FIFO tehnike keširanja [19]

5.5 Optimizacija baze podataka

Optimizacija baze podataka ključno je područje u poboljšanju performansi web aplikacija, jer baze podataka često predstavljaju usko grlo u sustavu. Optimizacija se odnosi na različite tehnike koje pomažu smanjiti vrijeme izvršenja upita, poboljšati odziv baze i smanjiti opterećenje sustava. Glavni pristupi optimizaciji baze podataka bili bi:

- **Indeksiranje**: Indeksi pomažu ubrzati pretragu i dohvaćanje podataka. Kreiranjem indeksa na stupcima koji se često koriste u upitima, smanjuje se vrijeme pretraživanja i izvršenja upita. Međutim, treba biti oprezan jer preveliki broj indeksa može usporiti upis i ažuriranje podataka.
- **Normalizacija podataka**: Normalizacija osigurava da su podaci pohranjeni na logičan i efikasan način, eliminirajući duplicirane informacije. To smanjuje potrebu za pohranom velikih količina redundantnih podataka i ubrzava upite.
- **Keširanje upita**: Keširanje rezultata često korištenih upita pomaže u smanjenju broja upita prema bazi podataka. Kada aplikacija ponovno zatraži iste podatke, keširani rezultati mogu biti vraćeni brže nego ponovno izvršavanje upita.
- **Denormalizacija**: U nekim slučajevima, posebno kada su performanse čitanja važnije od prostora pohrane, uvođenje određenog stupnja redundantnosti može poboljšati brzinu izvršavanja upita. To omogućuje dohvaćanje podataka iz manje tablica, smanjujući broj potrebnih pridruživanja (JOIN) u upitima.

Zamislimo li bazu podataka koja sadrži tablicu s milijun zapisa, a nad njom se redovno izvršavaju upiti poput „SELECT * from korisnici WHERE grad = 'Zagreb'“, kreiranje indeksa na stupcu grad može značajno ubrzati pretragu baze podataka jer se više neće pregledavati svaki redak kako bi se pronašli korisnici iz Zagreba. Umjesto da baza podataka pregledava svaki redak tablice kako bi provjerila grad za svakog korisnika, koristi indeks da direktno pronađe sve zapise koji se odnose na grad 'Zagreb'.

Ova metoda omogućava brže lociranje zapisa jer indeks funkcionira kao mapa koja direktno vodi do redaka koji sadrže tražene vrijednosti. Umjesto pregledavanja cijele tablice, sustav koristi indeks kako bi brzo pronašao lokaciju traženih podataka. Ova metoda značajno ubrzava pretrage, osobito kod velikih tablica.

5.6 Refaktoriranje i modularizacija koda

Također još jedna vrlo značajna tehnika u održavanju i optimizaciji softverskog razvoja je refaktoriranje i modularizacija koda. Ne samo da poboljšava performanse, već i oslakšava razumijevanje, održavanje i proširivost koda.

5.6.1 Refaktoriranje koda

Refaktoriranje je proces restrukturiranja postojećeg koda bez promjene njegove vanjske funkcionalnosti. Cilj je poboljšati internu strukturu koda, učiniti ga čitljivijim, lakšim za održavanje i učinkovitijim. [18, str. 216] Pod prednosti refaktoriranja spadaju:

- Poboljšana čitljivost
- Smanjena složenost
- Laka proširivost
- Povećane performanse

Ako imamo funkciju koja radi nekoliko stvari odjednom – validira korisničke podatke, pohranjuje ih u bazu podataka i šalje obavijest. Umjesto takve jedne velike funkcije, refaktoriranjem bismo ove zadatke podijelili u manje, specifične funkcije koje se pozivaju unutar glavne funkcije, čime se poboljšava čitljivost i održivost.

5.6.2 Modularizacija koda

Modularizacija se odnosi na razdvajanje koda u manje, neovisne module ili komponente koje obavljaju specifične zadatke. Svaki modul ima određenu funkcionalnost i može se ponovno koristiti ili testirati neovisno od ostatka sustava.

Jedna od ključnih prednosti je mogućnost ponovne upotrebe modula na različitim dijelovima aplikacije, čime se smanjuje potreba za dupliciranjem koda. Osim toga, svaki modul može se testirati izolirano, što pojednostavljuje otkrivanje i ispravljanje pogrešaka te poboljšava pouzdanost sustava. Kôd organiziran u module često rezultira i boljim performansama, jer omogućuje izolaciju ključnih dijelova koda koji zahtijevaju optimizaciju.

Na primjer, ako aplikacija sadrži veliki dio koda koji se bavi korisničkom registracijom, modularizacijom ga možemo podijeliti na odvojene module. Možemo imati modul za validaciju korisničkih podataka, drugi modul za pohranu tih podataka u bazu podataka, te zaseban modul za slanje obavijesti. Na ovaj način, kôd postaje preglednij, organiziraniji i lakše se održava, a svaki modul može se neovisno koristiti i mijenjati bez utjecaja na ostatak aplikacije.

Takav primjer modularizacije u JavaScript jeziku može se prikazati sljedećim kodom, gdje su funkcionalnosti podijeljene u odvojene module za validaciju, pohranu podataka i slanje obavijesti.

```
// validacija.js
export function validirajPodatkeKorisnika(podaci) {
    if (!podaci.korime || podaci.korime.length < 3) {
        return { valid: false, message: Korisničko ime mora biti duže od 3 znaka.' };
    }
    if (!podaci.email.includes('@')) {
        return { valid: false, message: 'Neispravna email adresa.' };
    }
    if (!podaci.lozinka || podaci.lozinka.length < 6) {
        return { valid: false, message:Lozinka mora biti duža od 6 znakova.' };
    }
    return { valid: true };
}
```

Modul „validacija.js“ provjerava valjanost korisničkih podataka.

```
// pohrana.js
export function spremiPodatkeKorisnika(podaci) {
    console.log('Spremanje korisničkih podataka u bazu:', podaci);
}
```

Modul „pohrana.js“ pohranjuje korisničke podatke u bazu podataka.

```
// obavijesti.js
export function posaljiObavijest(email) {
    console.log(`Slanje obavijesto na email adresu: ${email}`);
}
```

Modul „obavijesti.js“ šalje obavijest putem emaila.

```
// registracija.js
import { validirajPodatkeKorisnika } from './ validacija.js';
import { spremiPodatkeKorisnika } from './ pohrana.js';
import { posaljiObavijest } from './ obavijesti.js';

function registrirajKorisnika(podaci) {
    const validacija = validirajPodatkeKorisnika (podaci);
    if (!validacija.valid) {
        console.log('Neuspješna registracija:', validacija.message);
        return;
    }
    spremiPodatkeKorisnika(podaci);
    posaljiObavijest(podaci.email);
    console.log('Korisnik uspješno registriran.');
}
const noviKorisnik = {
    korime: 'rkalinic',
    email: robert@email.com',
    lozinka: 'lozinka123'
};
registrirajKorisnika (noviKorisnik);
```

Modul „registracija.js“ koristi sve tri funkcije kako bi implementirao cjelokupni proces registracije korisnika.

Zaključno, refaktoriranje je proces koji se više odnosi na poboljšanje kvalitete koda unutar postojeće strukture, dok je modularizacija metoda organiziranja koda u neovisne jedinice, što olakšava održavanje i proširenje aplikacije.

6. Alati za mjerenje performansi

U ovom dijelu rada pružit će se kratak pregled najvažnijih alata za mjerenje performansi web aplikacija. Ovi alati omogućuju detaljno praćenje ključnih metrika koje utječu na korisničko iskustvo, poput brzine učitavanja, interaktivnosti i stabilnosti prikaza stranice. Iako će ovdje biti samo ukratko spomenuti, njihova praktična primjena bit će obrađena u poglavlju 6, gdje će se koristiti za analizu postojećih web aplikacija.

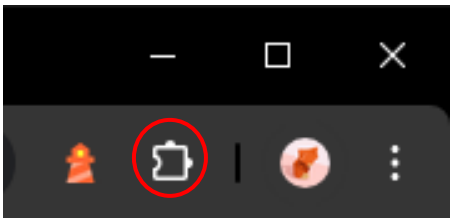
Prvi mnogih moćnih alata za procjenu performansi web aplikacija kojim ćemo se koristiti u ovom radu je „Google Lighthouse“ [20], ugrađen u Chrome preglednik. Nudi detaljne izvještaje o metrikama kao što su Largest Contentful Paint (LCP), First Input Delay (FID) i Cumulative Layout Shift (CLS). Alat ne samo da pruža konkretne vrijednosti za svaku metriku, već i preporuke za poboljšanje performansi. Lighthouse također evaluira i druge aspekte web stranice poput SEO optimizacije, pristupačnosti i progresivnosti aplikacije. Rezultati testiranja mogu se iskoristiti za detaljnu analizu i implementaciju konkretnih rješenja za optimizaciju.

Također, još jedan od alata koje ćemo koristiti u istraživanju je online dostupan alat PageSpeed Insights [21]. To je alat koji koristi podatke iz Google-ove baze stvarnih korisnika kako bi pružio dublji uvid u stvarno korisničko iskustvo pri korištenju web stranice. Pored analiza za desktop i mobilne uređaje, PageSpeed Insights pruža detaljne upute o tome kako smanjiti vrijeme učitavanja, optimizirati resurse, ukloniti blokirajući JavaScript, i poboljšati ukupne performanse web aplikacije.

7. Analiza postojećih web aplikacija

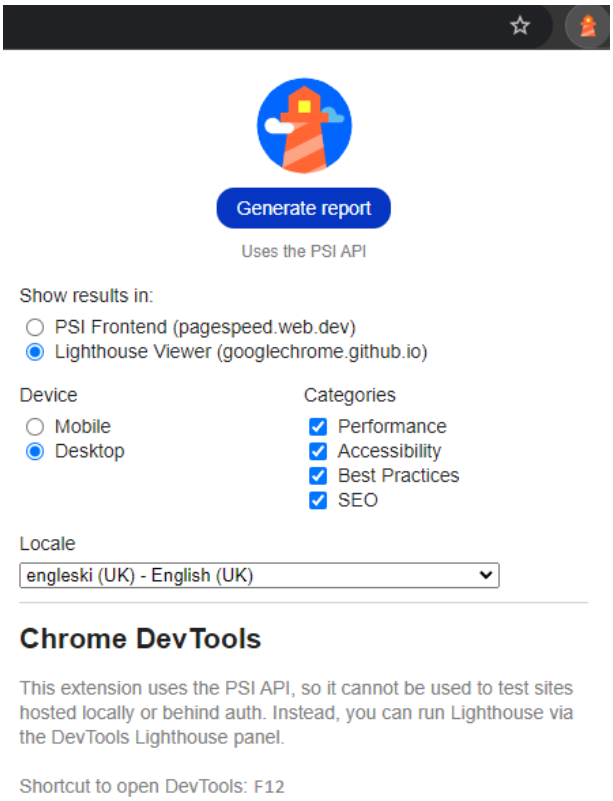
Kao što je navedeno u poglavlju 5 (Alati za mjerenje performansi), u ovom dijelu rada koristit ćemo se ranije navedenim alatima. Budući da su alati relativno slični, kao i njihovi rezultati, nećemo analizirati metrike iz oba alata već ćemo se fokusirati na one pružene od strane Google Lighthouse.

Konkretno za primjer analizirat ćemo postojeću web aplikaciju Upwork [22]. Kako bimo se uopće mogli koristiti alatom za analizu performansi za stranicu koju smo odabrali, potrebno je dodati Google Lighthouse u proširenja u preglednik. Nakon što je dodan u proširenja, možemo ga pronaći u gornjem desnom kutu u pregledniku pod ikonicom „Proširenja“ ili eng. „Extensions“.



Slika 8. Ikonica za proširenja u Google pregledniku

Nakon što je alat dodan, potrebno je otvoriti web mjesto željene stranice te zatim kliknuti na ikonicu Google Lighthouse. Prikazat će nam se razne opcije koje alat nudi. Rezultati se mogu prikazati u ranije navedenom alatu PageSpeed Insights ili konkretno u Lighthouse Viewer-u. Odabrat ćemo Lighthouse Viewer. Također, moguće je web aplikaciju analizirati u desktop verziji i mobilnoj verziji budući da rezultati mogu biti različiti. Odabrati Desktop. Od kategorija koje se mogu prikazati u analizi odabrat ćemo sve navedene. Za jezik odabrat ćemo engleski. Konačno, za generiranje izvješća kliknut ćemo na plavi gumb „Generate report“.

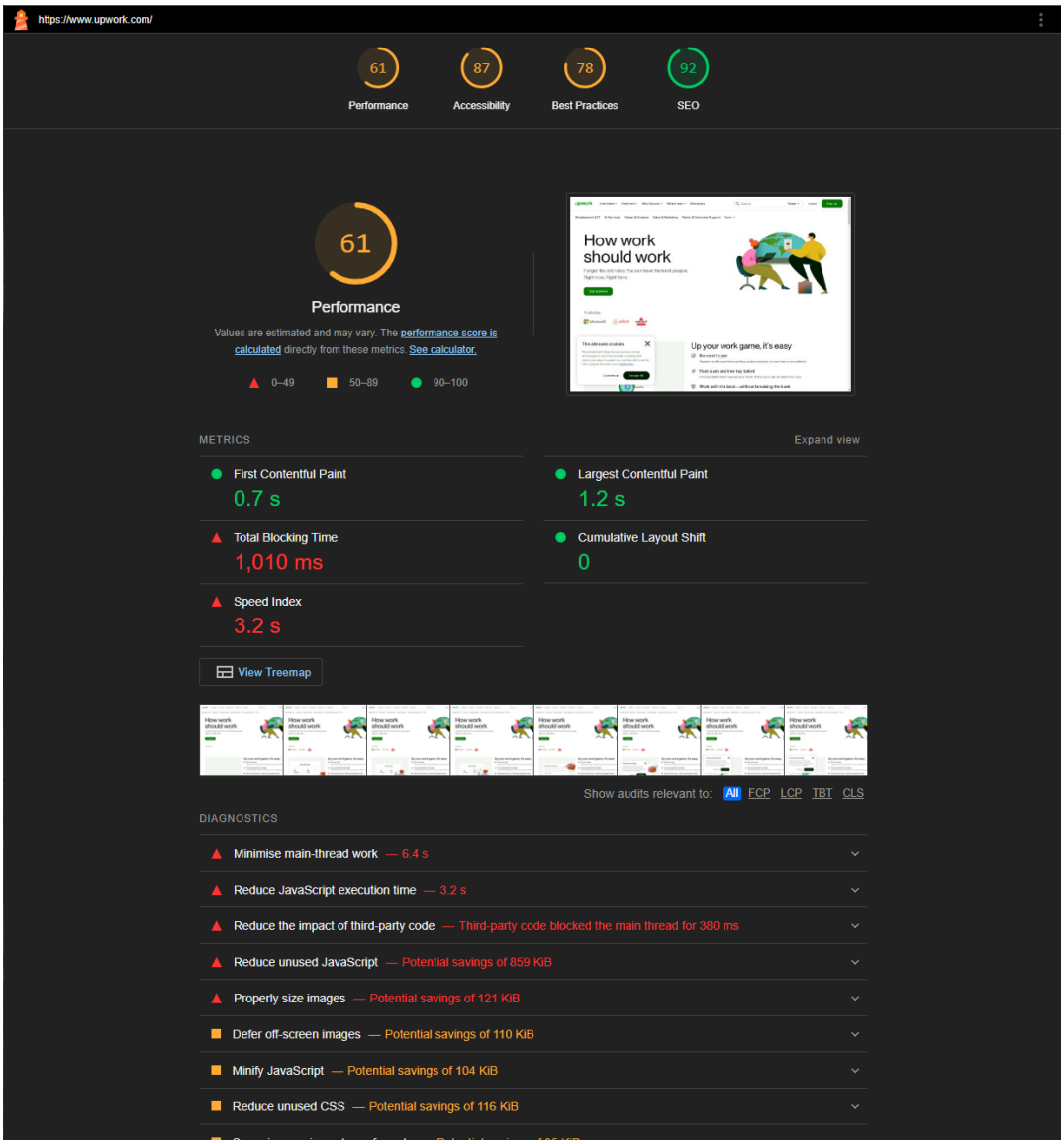


Slika 9. Izgled postavki za generiranje izvješća (Google Lighthouse)

7.1 Rezultati analize i dijagnostika

Nakon vremena od otprilike minutu, na slici 10 možemo vidjeti generirano izvješće. Na vrhu ekrana, u smislu postotka kvalitete stranice, prikazani su glavni rezultati: performanse (61), pristupačnost (87), najbolje prakse (78) i SEO (92). Na temelju dobivenih rezultata može se zaključiti kako je aplikacija dobro optimizirana, ali i da postoji prostora za napredak. Idealni postotci za svaki od rezultata bili bi između 90 i 100.

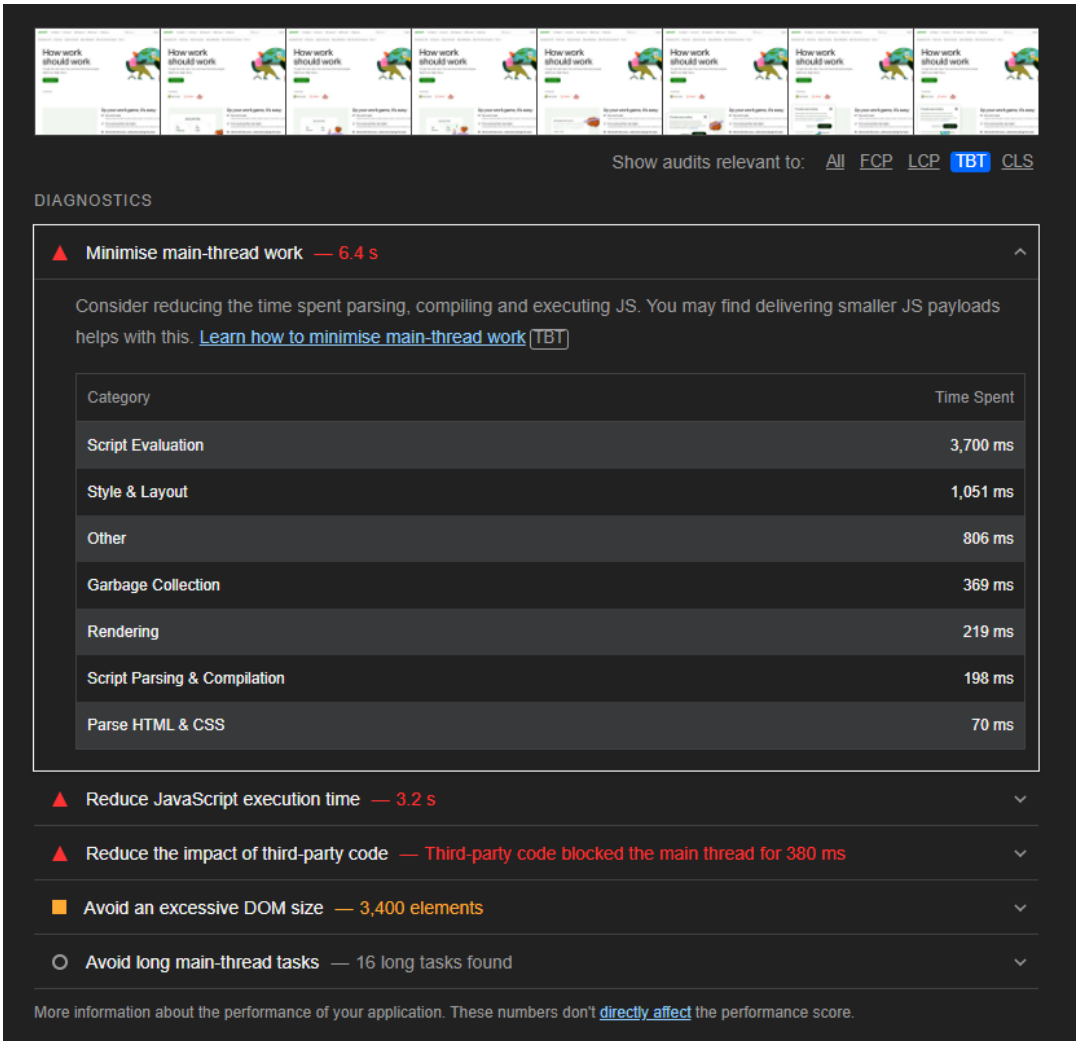
Na sredini ekrana vidljivi su rezultati metrika za performanse. Izmjereni su neki od *Core Web Vitals* i *Other Web Vitals* (poglavlje 2. „Analiza performansi web aplikacija“). Prema bojama (zelena, narančasta i crvena) možemo zaključiti koje metrike su dobro optimizirane, koje bi mogle biti optimiziranije i koje zahtijevaju optimizaciju. Crvena boja znači loše, odnosno sporo, pa je potrebna primjena nekih od tehnika optimizacije performansi web aplikacije.



Slika 10. Prikaz rezultata (Google Lighthouse)

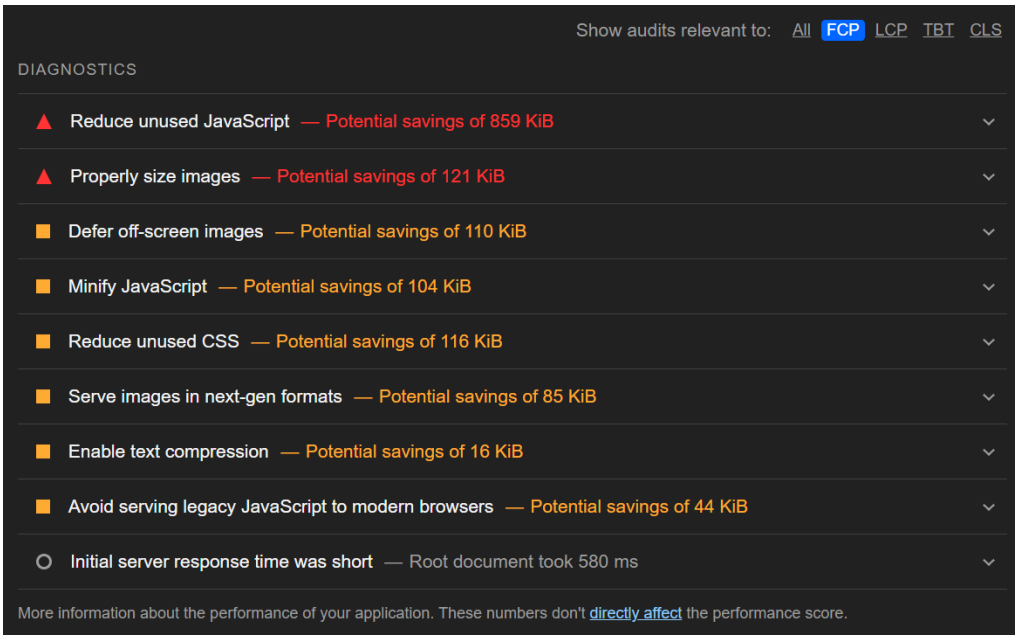
Konkretno, u ovom primjeru vidimo da Total Blocking Time (TBT) ima period od 1,010 ms. Ako odaberemo da nam se prikažu dijagnostički podaci samo o TBT, možemo vidjeti u čemu je točno problem, slika 12.

Preopterećenje glavnog niza zadataka (6.4 sekunde) i izvršavanje JavaScript koda (3.2 sekunde) uzrokuje značajan problem u performansama aplikacije. Posebno su problematične dugotrajne operacije poput evaluacije skripti i stiliziranja. Također, prekomjerna veličina doma s 3400 elementa doprinosi lošijim performansama.



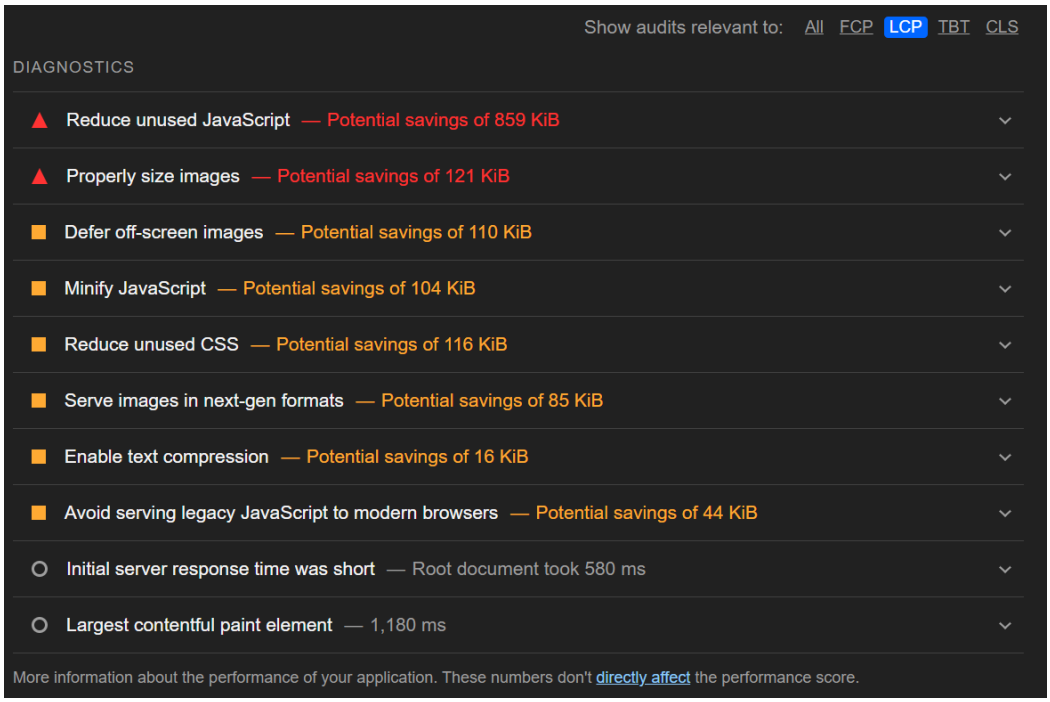
Slika 11. TBT dijagnostika (Google Lighthouse)

Problemi za JavaScript dio mogu se riješiti na nekoliko načina. Jedan od njih bio bi korištenje lazy loading-a i tehniku minifikacije. Budući da preveliki DOM može izazvati dugotrajne stilizacijske operacije, bilo bi poželjno smanjiti broj elemenata i pojednostaviti ih kako bi se smanjilo vrijeme potrebno za renderiranje. Još jedna od tehnika koja bi se ovdje mogla primijeniti je refaktoriranje i optimizacija dugotrajnih zadataka. Primjerice, korištenjem Web Workers-a za teške procese mogao bi se rasteretiti glavni thread.



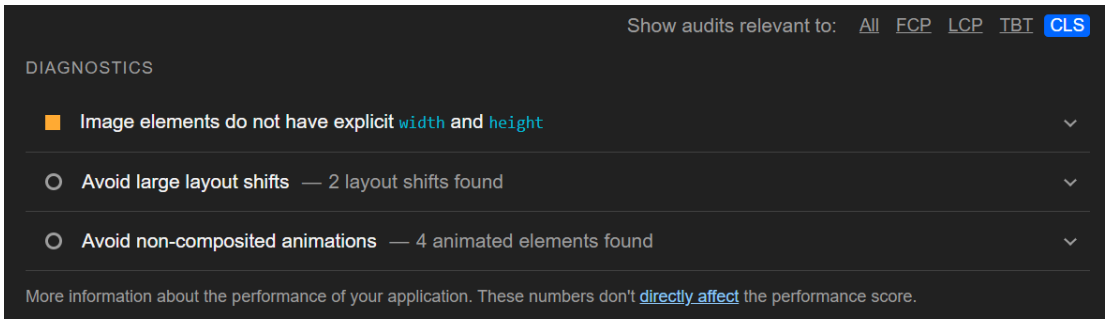
Slika 12. FCP dijagnostika (Google Lighthouse)

Dijagnostika za First Contentful Paint (FCP) aplikacije Upwork ukazuje na nekoliko ključnih područja gdje se mogu poboljšati performanse. Jedno od glavnih poboljšanja odnosi se na smanjivanje neiskorištenog JavaScript-a, što može značajno ubrzati vrijeme učitavanja stranice smanjujući količinu nepotrebnog koda koji se izvršava. Osim toga, ispravno prilagođavanje veličine slika može optimizirati vrijeme učitavanja, jer slike koje nisu optimizirane za dimenzije prikaza mogu usporiti proces učitavanja. Također, korištenje tehnike odgađanja učitavanja slika koje se nalaze izvan vidnog polja (lazy loading) može dodatno smanjiti vrijeme učitavanja, jer će slike koje nisu odmah vidljive biti učitane kasnije. Minifikacija JavaScript-a i CSS-a dodatno doprinosi ubrzanju stranice, uklanjanjem nepotrebnih znakova iz koda, dok upotreba modernih formata slika, kao što je WebP, omogućava smanjenje veličine slika i brže učitavanje. Implementacijom ovih optimizacija, vrijeme do prvog prikaza sadržaja na stranici može biti značajno poboljšano, što vodi do boljeg korisničkog iskustva.



Slika 13. LCP dijagnostika (Google Lighthouse)

Ako odaberemo dijagnostiku za Largest Contentful Paint (LCP), dobit ćemo identične preporuke za optimizaciju jer na FCP i LCP utječu skoro pa isti faktori. Jedina dodana dijagnostika je ona na dnu, „Largest contentful paint element“, međutim ovaj jedan element dovoljno se brzo učitava pa nam ne stvara razliku i nema potrebe za njegovom optimizacijom.



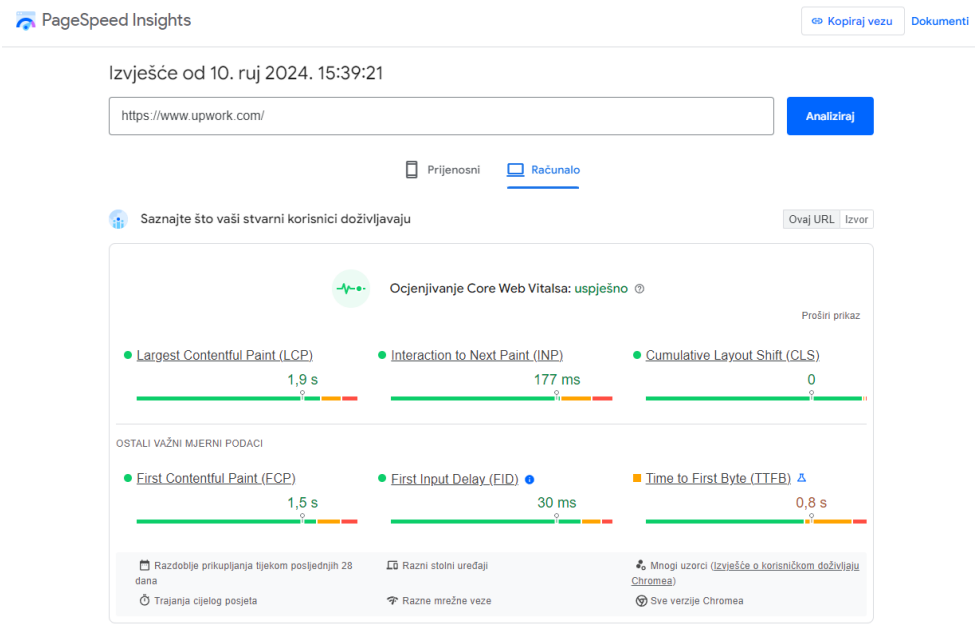
Slika 14. CLS dijagnostika (Google Lighthouse)

Ova dijagnostika mjeri metriku Cumulative Layout Shift (CLS), koja utječe na vizualnu stabilnost stranice. Primarno se preporučuje da svi elementi slika imaju definirane atribute visine

i širine kako bi se spriječili neočekivani pomaci rasporeda tijekom učitavanja stranice. Također, preporučuje se izbjegavanje velikih pomaka u izgledu (layout shifts) i korištenje "composited" animacija koje će se izvoditi efikasnije, smanjujući vizualne smetnje korisnicima prilikom pregledavanja.

7.2 PageSpeed Insights dijagnostika

PageSpeed Insights nudi detaljne informacije o performansama web stranice, ali u usporedbi s Google Lighthouseom, on se fokusira na stvarne korisničke podatke. Dok Google Lighthouse pruža simulirane testove izvedbe, PageSpeed Insights koristi stvarne podatke iz Chrome User Experience Reporta (CrUX) kako bi pokazao kako stvarni korisnici doživljavaju web stranicu. Ovi podaci dolaze iz korisničkih sesija i pokrivaju ključne metrike poput Largest Contentful Paint (LCP), Interaction to Next Paint (INP), Cumulative Layout Shift (CLS) i drugih, omogućujući analizu stvarnog vremena učitavanja, kašnjenja prilikom prve interakcije te ukupnu stabilnost stranice. Ova dodatna razina podataka čini PageSpeed Insights izuzetno korisnim alatom za razumijevanje performansi stranice iz perspektive stvarnih korisnika.



Slika 15. Dodatni rezultati analize (PageSpeed Insights)

8. Zaključak

Zaključak ovog rada usmjeren je na važnost analize i optimizacije performansi web aplikacija kako bi se postigla bolja korisnička iskustva i učinkovitije korištenje resursa. U modernom razvoju web aplikacija, ključno je kontinuirano praćenje performansi koristeći alate kao što su Google Lighthouse i PageSpeed Insights. Ovi alati pružaju vrijedne metrike poput Largest Contentful Paint (LCP), Cumulative Layout Shift (CLS) i Total Blocking Time (TBT), koje omogućuju programerima da prepoznaju gdje ima prostora za optimizacijom.

PageSpeed Insights nudi dodatnu prednost u vidu analize stvarnih korisničkih podataka, što daje precizniji uvid u ponašanje aplikacije u različitim uvjetima, dok Lighthouse pruža detaljne preporuke na temelju simuliranih testova. Optimizacija uključuje niz strategija poput smanjenja korištenja neiskorištenih JavaScript i CSS resursa, minifikacije koda, implementacije keširanja te korištenja modernih formata slika. Modularizacija i refaktoriranje koda dodatno olakšavaju održavanje i poboljšavaju performanse, dok strategije upravljanja predmemorijom, poput LRU i LFU, pomažu u učinkovitijem korištenju memorije.

U konačnici, optimizacija performansi nije jednokratan proces, već stalni ciklus analize, testiranja i prilagodbe. Pravilnim pristupom, uz korištenje spomenutih alata i metoda, moguće je značajno poboljšati korisničko iskustvo, smanjiti vrijeme učitavanja stranica i osigurati stabilnost web aplikacija u dugom roku.

9. Popis literature

[1] A. B. King, *Website Optimization*. O'Reilly Media, Inc., 2008.

[2] D. Goswami, „Top 8 Advanced Optimization Techniques For Improved Web App Performance“, EvinceDev Blog. Pristupljeno: 30. kolovoz 2024. [Na internetu]. Dostupno na: https://evincedev.com/blog/optimization-techniques-for-improved-web-app-performance/

[3] „How mobile latency impacts publisher revenue“, Think with Google. Pristupljeno: 30. kolovoz 2024. [Na internetu]. Dostupno na: https://www.thinkwithgoogle.com/intl/en-emea/marketing-strategies/app-and-mobile/need-mobile-speed-how-mobile-latency-impacts-publisher-revenue/

[4] „Core Web Vitals“. Pristupljeno: 30. kolovoz 2024. [Na internetu]. Dostupno na: https://crystallize.com/learn/best-practices/frontend-performance/core-web-vitals

[5] „Largest Contentful Paint (LCP)“. Pristupljeno: 31. kolovoz 2024. [Na internetu]. Dostupno na: https://crystallize.com/learn/best-practices/frontend-performance/core-web-vitals/largest-contentful-paint-lcp

[6] „Interaction to Next Paint (INP)“. Pristupljeno: 31. kolovoz 2024. [Na internetu]. Dostupno na: https://crystallize.com/learn/best-practices/frontend-performance/core-web-vitals/interaction-to-next-paint-inp

[7] „Cumulative Layout Shift (CLS)“. Pristupljeno: 31. kolovoz 2024. [Na internetu]. Dostupno na: https://crystallize.com/learn/best-practices/frontend-performance/core-web-vitals/cumulative-layout-shift-cls

[8] „First Contentful Paint (FCP)“. Pristupljeno: 31. kolovoz 2024. [Na internetu]. Dostupno na: https://crystallize.com/learn/best-practices/frontend-performance/other-web-vitals/first-contentful-paint-fcp

[9] „Time to First Byte (TTFB)“. Pristupljeno: 31. kolovoz 2024. [Na internetu]. Dostupno na: https://crystallize.com/learn/best-practices/frontend-performance/other-web-vitals/time-to-first-byte-ttfb

[10] „Total Blocking Time (TBT)“. Pristupljeno: 31. kolovoz 2024. [Na internetu]. Dostupno na: https://crystallize.com/learn/best-practices/frontend-performance/other-web-vitals/total-blocking-time-tbt

[11] „Time to Interactive (TTI)“. Pristupljeno: 31. kolovoz 2024. [Na internetu]. Dostupno na: https://crystallize.com/learn/best-practices/frontend-performance/other-web-vitals/time-to-interactive-tti

[12] „First Input Delay (FID)“. Pristupljeno: 31. kolovoz 2024. [Na internetu]. Dostupno na: https://crystallize.com/learn/best-practices/frontend-performance/other-web-vitals/first-input-delay-fid

[13] „Speed Index“. Pristupljeno: 31. kolovoz 2024. [Na internetu]. Dostupno na: https://crystallize.com/learn/best-practices/frontend-performance/other-web-vitals/speed-index

[14] „Reduce initial server response time“. Pristupljeno: 05. rujan 2024. [Na internetu]. Dostupno na: https://gtmetrix.com/reduce-initial-server-response-time.html

[15] H. Falcao, *Value Negotiation: How to Finally Get the Win-Win Right*. FT Press, 2012.

[16] „Ways to Reduce Initial Server Response Time“, WP Engine. Pristupljeno: 05. rujan 2024. [Na internetu]. Dostupno na: https://wpengine.com/resources/reduce-server-response-time/

[17] „Why minify JavaScript code?“ Pristupljeno: 05. rujan 2024. [Na internetu]. Dostupno na: https://www.cloudflare.com/learning/performance/why-minify-javascript-code/

[18] A. B. King, *Speed Up Your Site: Web Site Optimization*. New Riders, 2003.

[19] „Caching Strategies“, Hazelcast. Pristupljeno: 05. rujan 2024. [Na internetu]. Dostupno na: https://hazelcast.com/glossary/caching-strategies/

[20] „Overview | Lighthouse“, Chrome for Developers. Pristupljeno: 10. rujan 2024. [Na internetu]. Dostupno na: https://developer.chrome.com/docs/lighthouse/overview

[21] „PageSpeed Insights“. Pristupljeno: 06. rujan 2024. [Na internetu]. Dostupno na: https://pagespeed.web.dev/

[22] „Upwork | The World’s Work Marketplace“, Upwork. Pristupljeno: 10. rujan 2024. [Na internetu]. Dostupno na: https://www.upwork.com/

10. Popis slika