Usporedba metoda renderiranja web aplikacija

Beran, Ivan

SVEUČILIŠTE U SPLITU
PRIRODOSLOVNO MATEMATIČKI FAKULTET

DIPLOMSKI RAD

Usporedba metoda renderiranja web aplikacija

Ivan Beran

Split, 2023.

TEMELJNA DOKUMENTACIJSKA KARTICA

Diplomski rad

Sveučilište u Splitu
Prirodoslovno-matematički fakultet
Odjel za informatiku
Ruđera Boškovića 33, 21000 Split, Hrvatska

Usporedba metoda renderiranja web aplikacija

Ivan Beran

Sažetak

Cilj ovoga diplomskoga rada je odgovoriti na pitanje koja tehnologija za izradu web aplikacija omogućava stvaranja aplikacija koje su optimizirane za korisnikovu interakciju. Kako bi se dobio odgovor na definirano pitanje u ovome radu biti će prikazana implementacija dviju web aplikacija korištenjem različitih tehnologija za razvoj web aplikacija. Jedna tehnologija obradu i pripremu podataka za prikaz radi na strani korisnika dok druga taj proces radi na strani poslužitelja. Nakon implementacije aplikacija njihova funkcionalnost biti će analizirana korištenjem alata treće strane.

Ključne riječi:: arhitektura mrežnih stranica, JavaScript, API, React, Next.js, Lighthouse

Rad je pohranjen u knjižnici Prirodoslovno-matematičkog fakulteta, Sveučilišta u Splitu

Rad sadrži: 52 stranice, 24 grafička prikaza, 2 tablica i 13 literaturnih navoda. Izvornik je na hrvatskom jeziku.

Mentor: **doc. dr. sc. Goran Zaharija,** *docent Prirodoslovno-matematičkog fakulteta, Sveučilišta u Splitu*

Ocjenjivači: **doc. dr. sc. Goran Zaharija,** *docent Prirodoslovno-matematičkog fakulteta, Sveučilišta u Splitu*

doc. dr. sc. Divna Krpan, *docent Prirodoslovno-matematičkog fakulteta, Sveučilišta u Splitu*

Dino Nejašmić, *predavač Prirodoslovno-matematičkog fakulteta, Sveučilišta u Splitu*

Rad prihvaćen: siječanj, 2023.

BASIC DOCUMENTATION CARD

Thesis

University of Split
Faculty of Science
Department of computer science
Ruđera Boškovića 33, 21000 Split, Croatia

Comparison of web application rendering methods

Ivan Beran

ABSTRACT

The goal of this thesis is to answer the question which technology for creating web applications enables the creation of applications that are optimized for user interaction. To get an answer to the defined question, this paper will present the implementation of two web applications using different technologies for the development of web applications. One technology processes and renders data for display on the user's side, while the other renders data on the server side. After the implementation of the applications, their functionality will be analyzed using third-party tools.

Key words: web architecture, JavaScript, API, React, Next.js, Lighthouse

Thesis deposited in library of Faculty of science, University of Split

Thesis consists of: 52 pages, 24 figures, 2 tables and 13 references.

Original language: Croatian

Mentor: **Goran Zaharija, Ph.D.** *Assistant Professor of Faculty of Science, University of Split*

Reviewers: **Goran Zaharija, Ph.D.** *Assistant Professor of Faculty of Science, University of Split*

Divna Krpan, Ph.D. *Assistant Professor of Faculty of Science, University of Split*

Dino Nejašmić, Ph.D. *Lecturer of Faculty of Science, University of Split*

Thesis accepted: January 2023.

IZJAVA

kojom izjavljujem s punom materijalnom i moralnom odgovornošću da sam diplomski rad s naslovom „Usporedba metoda renderiranja web aplikacija“ izradio samostalno pod voditeljstvom dr.sc. Gorana Zaharije. U radu sam primijenio metodologiju znanstvenoistraživačkog rada i koristio literaturu koja je navedena na kraju diplomskog rada. Tuđe spoznaje, stavove, zaključke, teorije i zakonitosti koje sam izravno ili parafrazirajući naveo u diplomskom radu na uobičajen, standardan način citirao sam i povezao s fusnotama s korištenim bibliografskim jedinicama. Rad je pisan u duhu hrvatskog jezika.

Student
Ivan Beran

Sadržaj

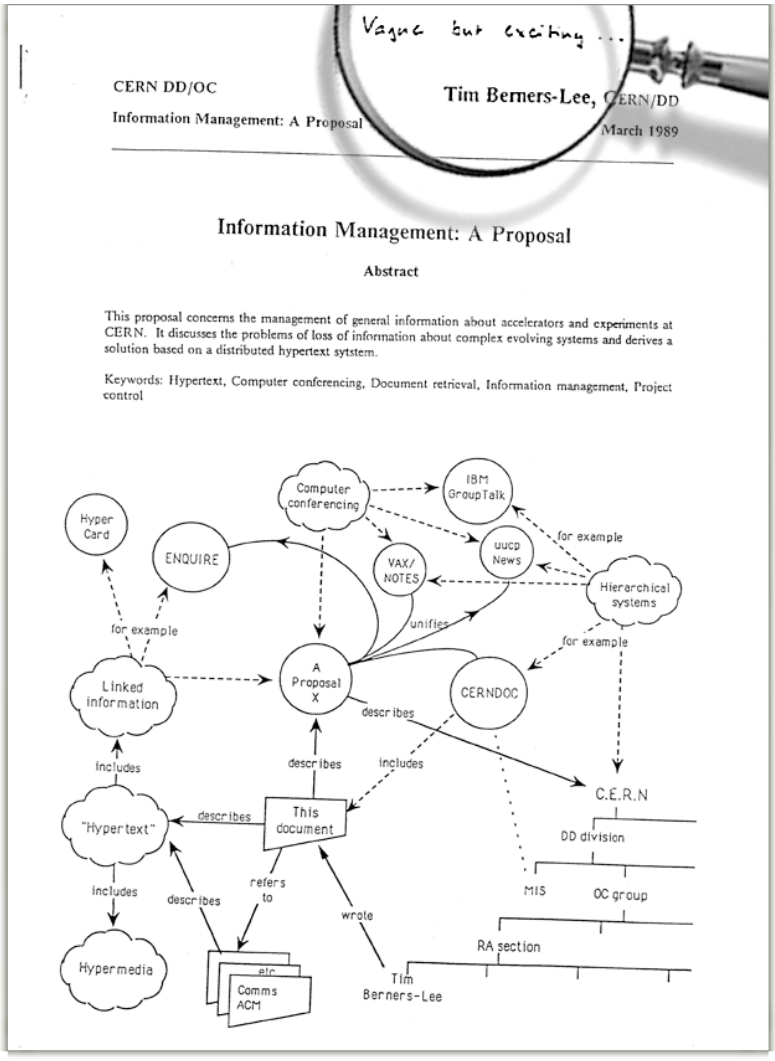
UVOD

Tehnologija koja je proizašla iz potrebe da informacije budu dostupne svakome u svakome trenutku danas je postala neophodan dio svakodnevnog života. Od prve jednostavne mrežne stranice koja je služila kao polazišna točka oko koje su se zaplele mreže računala koje nazivamo Web. Ta prva stranica pokretala se na poslužitelj i isporučivala korisnicima kao gotov proizvod. Razvojem tehnologija od početka Weba do danas razvile su se i tehnologije posluživanja mrežnih stranica. Kroz ovaj rad prikazati će se razvoj tehnologija koje stoje iza web aplikacija te arhitektura kojima se mogu kreirati web aplikacije. Ovaj rad će analizirati dva glavna principa renderiranja podataka na modernim jednostraničnim web aplikacijama te pokušati dati odgovor na pitanje koji od ta dva načina pruža bolju optimizaciju pri korištenju. Kroz prvo poglavlje opisan je nastanak prve mrežne stranice te razvoj Weba kroz generacije. U drugom poglavlju su opisane osnovne tehnologije na kojima se temelji Web. U trećem poglavlju opisana je tehnologija izrade mrežnih stranica i aplikacija. U četvrtom poglavlju objašnjeno je što su to web aplikacije te su objašnjene vrste implementacija web stranica. Nakon toga slijedi praktični dio rada u kojem je opisana izrada aplikacija te analiza dobivenih podataka.

1. POVIJESNI RAZVOJ WEBA

U ožujku 1989. godine, engleski znanstvenik Sir Tim Berners-Lee predstavio je prijedlog za sustav čija je glavna svrha omogućavanje lakšeg upravljanje informacijama. Nemogućnost pristupa informacija u svakom trenutku i limitiranost količine izvora informacija su problemi koje je Tim Berners-Lee želio riješiti svojom idejom. Berners-Lee koji je tada radio u CERN-u kao znanstvenik iznio je ideju svojem šefu Mikeu Sendallu.

Iako njegova u ideja u to vrijeme nije bila popraćena velikim interesom, bila je dovoljno zanimljiva da je dobio dopuštenje da nastavi raditi na tom projektu. [1]



Slika 1. Prvi prijedlog za izradu sustava za upravljanje informacijama koji će kasnije postati WorldWideWeb

U svibnju 1990. godine Berners-Lee piše drugi prijedlog svojeg projekta koji uz pomoć belgijskog sistemskog inženjera Roberta Cailiaua u studenom postaje formalizirani prijedlog pod imenom WorldWideWeb Prijedlog za HTTP projekt (engl. Proposal for a HyperText Project).

U tom prijedlogu opisuju svoj projekt kao hipertekstualni projekt pod nazivom WorldWideWeb u kojem će se mreža (engl. Web) sastojati od hipertekstualnih dokumenata (engl. Hypertext documents) koji će se moći pregledavati koristeći mrežne preglednike (engl. Browsers). Oni nastavljaju raditi na zamišljenom projektu te koristeći računalo NeXT usavršavaju svoj projekt.

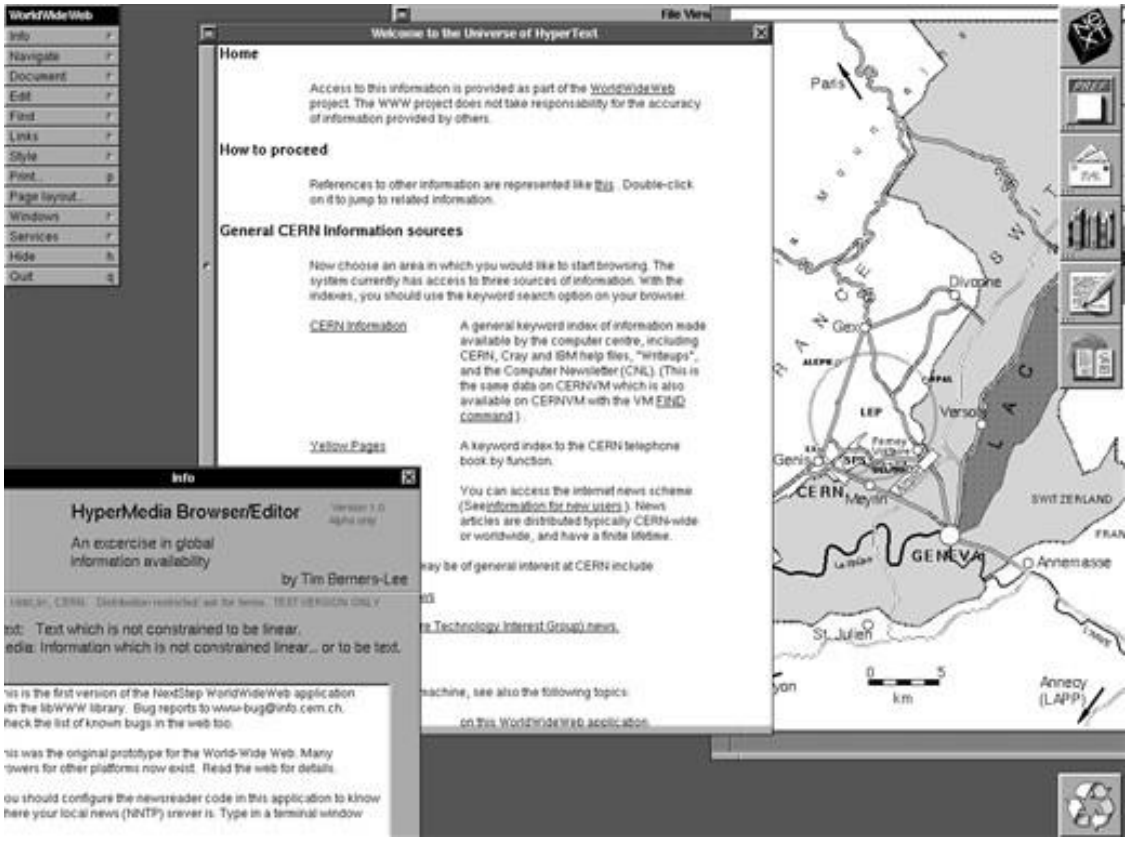
U nadolazećim mjesecima Berners-Lee razvija tri temeljne tehnologije koje će oblikovati konačni proizvod. Te tri tehnologije su:

1. HTML
2. URI
3. HTTP

Paralelno uz razvoj tih tehnologija Berners-Lee je razvio i prvi mrežni preglednik WorldWideWeb.app koji je imao i ulogu uređivača mrežnih stranica te prvi mrežni poslužitelj httpd. Pod pojmom mrežni poslužitelj smatramo programsku podršku, sklopovlje ili kombinaciju programske podrške i sklopovlja.

Sklopovlje internetskog poslužitelja može biti neko računalo koje pohranjuje programski podršku tog mrežnog poslužitelja te samu web stranicu kojoj korisnici mogu pristupiti. Tu podrazumijevamo HTML dokumente, CSS stilove, JavaScript datoteke te ostale podatke poput slika ili videa koje želimo prikazati na mrežnoj stranici. Pri razvoju svog projekta Berners-Lee kao mrežni poslužitelj je koristio računalo NeXT koje je posluživalo mrežnu stranicu.

Mrežna adresa prvog mrežnog mjesta je bila info.cern.ch a prve mrežne stranice http://info.cern.ch/hypertext/WWW/TheProject.html. Na toj mrežnoj stranici su se nalazile informacije o WorldWideWeb projektu uključujući informacije o hipertekstu, tehničkim detaljima izrade mrežnih poslužitelja te poveznice na druge mrežne poslužitelje koji bi bili napravljeni.



Slika 2. Izgled prvog mrežnog poslužitelja i mrežne stranice

Pristup računalnom poslužitelju koji se nalazio na NeXT računalu imala je samo nekolicina ljudi koja se povezivala lokalno u CERN-u. Zbog tog razloga je počeo razvoj na novom jednostavnijem mrežnom pregledniku Line Mode, koji se mogao pokretati na bilo kojem operacijskom sustavu. Nicola Pellow koja je bila studentica koja je radila na CERN-u bila je zaslužna za razvoj tog preglednika.

Godine 1991. Berners-Lee objavljuje svoj WWW program. On je sadržavao Line Mode preglednik i biblioteku za programere. U ožujku iste godine program postaje dostupan svim korisnicima CERN-ovih računala. Kasnije te godine Berners-Lee javno najavljuje računalni program WWW i time dobiva velik interes javnosti.

Prvi web poslužitelj u Ujedinjenim Američkim Državama pušten je u rad u prosincu 1991. godine na Stanford sveučilištu u Kaliforniji.

Godine 1993. CERN proglašava Web javnim vlasništvom i izvorni kôd prvoga mrežnog pretraživača i uređivača objavljuje javnosti.

Ovim započinje doba koje u povijesti razvoja weba nazivamo Web1.0.

1. 1. Web1.0.

Razvojem prvih mrežnih poslužitelja započinje prva faza evolucije World Wide Weba. Web se u to vrijeme sastojao uglavnom od statičkih mrežnih stranica koje nisu bili interaktivne već su imale svrhu pružanja informacija. Korisnici su mogli pasivno pregledavati i čitati podatke koji su se nalazili na mrežnim stranicama bez ikakvog vlastitog unosa podataka. Razdoblje Web 1.0 okvirno se smatra od 1991. do 2004. U razdoblju Weba 1.0 reklame na stranicama koje je korisnik posjećivao bile su zabranjene. Tehnologije na kojima se zasniva Web 1.0 su HTML, HTTP i URI. Glavne karakteristike ovog razdoblja su:

- Statične stranice
- Sadržaj se posluživao iz datotečnog sustava poslužitelja
- Za pozicioniranje sadržaja na stranicama koristili su se okviri i tablice

1. 2. Web2.0

Druga generacija weba započinje 2004. Za ime Web 2.0 zaslužna je konferencija pod imenom „First Web 2.0 Conference" koju su organizirali Tim O'Rillley i Dale Dougherty.

Web 2.0 krase globalne mrežne stranice koje ističu sadržaj generiran od strane korisnika, upotrebljivost i interoperabilnost za krajnje korisnike. Web 2.0 se definira kao *read-write* odnosno omogućava unošenje i čitanje podataka za razliku od *read-only* Weba 1.0 koji je služio samo za prikaz podataka koje bi korisnici mogli čitati. Web postaje društveno mjesto koji korisnicima pruža platforme i sredstva kojima mogu dijeliti svoja ideje, misli i iskustva te se iz tih razloga tada razvijaju prve društvene mreže, blogovi, podcasti, društveni mediji, platforme za dijeljenje sadržaja kao što su slike ili videozapisi i slično. [2]

Web tehnologije koje se koriste u razvoju Web 2.0 uključuju AJAX i JavaScript okvire. Glavne karakteristike ovog razdoblja su:

- Dinamički sadržaj koji je raspoznavan na korisnikov unos
- Korištenje okvira za JavaScript i AJAX
- Razvoj API-ja
- Razvoj CSS-a i HTML5

1. 3. Web3

U godinama nakon što je Edward Snowden objavio povjerljive dokumente koji su pokazali javnosti da postoje mnogobrojni projekti nadziranja privatnih podataka javlja se potreba za sigurnijim i privatnijim načinima razmjene podataka. Godine 2014. računalni znanstvenik koji je ujedno i kreator Ethernuma Gavin Wood na svome blogu piše o novom smjeru u kojem razvoj weba treba ići i daje mu ime web3.

Na svome blogu Wood opisuje mrežne tehnologije SMTP, FTP, HTTP, PHP, HTML i JavaScript kao temelje na kojima su se razvijale i stvarale web aplikacije, poput Facebooka, Google Drivea i Twittera, koje široke mase koriste danas. On smatra da se za budući razvoj tehnologije i društva ovi protokoli i tehnologije moraju revidirati u ovisnosti o načinima na koje to društvo i tehnologije međusobno komuniciraju. On smatra da se Web3 zasniva na četiri temeljne ideje: Korištenje decentraliziranog kriptiranog sustava za objavljivanje informacija, Pseudonimizirani i anonimizirani sustav slanja poruka niske razine koji je zasnovan na autorizaciji, mehanizam konsenzusa kojim se osigurava da će sve buduće interakcije biti automatski i neopozivo rezultat provedbe dogovora i korištenje pretraživača i korisničkih sučelja koja će koristiti sustav razlučivanja adresa temeljem konsenzusa. [3]

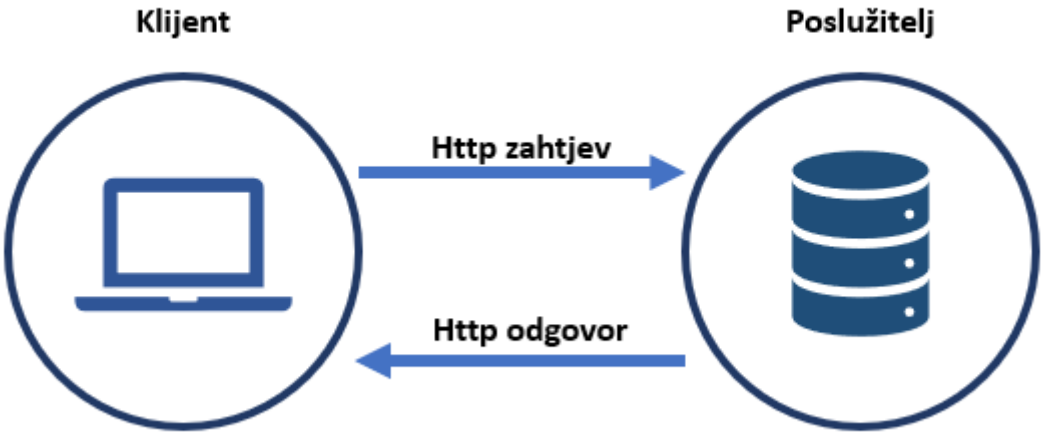
Web3 predstavlja korak u udaljavanju od monopolizacije podataka velikih tvrtki. Tehnologije koje stoje iza Web3 se temelje na peer-to-peer transakcijama, što znači da pojedinci koji koriste web mogu izmjenjivati podatke direktno bez ovisnosti o nekom posredniku.

Jedna od osnovnih tehnologija na kojima se temelji Web3 je struktura podataka koju nazivamo Blockchain. Blockchain je struktura podataka koja se sastoji od blokova povezanih hash pokazivačima u jednosmjerni lanac. Glavna svojstva Blockchaina su transparentnost, nepromjenjivost i distribuiranost.

2. OSNOVNE TEHNOLOGIJE WEBA

2. 1. HTTP

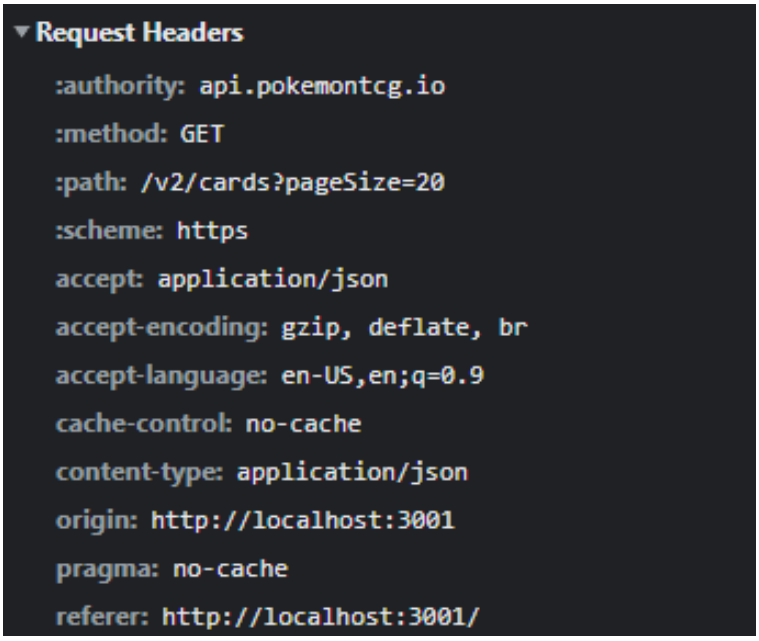
HTTP (engl. Hyper Text Transfer Protocol) je protokol koji služi za dohvaćanje HTML dokumenata. Ovaj protokol je osnova na kojoj se zasniva komunikacija klijent – poslužitelj. HTTP je protokol aplikacijskog sloja koji se šalje putem TCP protokola te je oblikovan za slanje informacija između uređaja na mreži i dio je transportnog sloja. Komunikacija se odvija na način da računalo klijenta šalje HTTP zahtjev na koji mu mrežni poslužitelj šalje HTTP odgovor.



Slika 3. Komunikacija klijent – poslužitelj

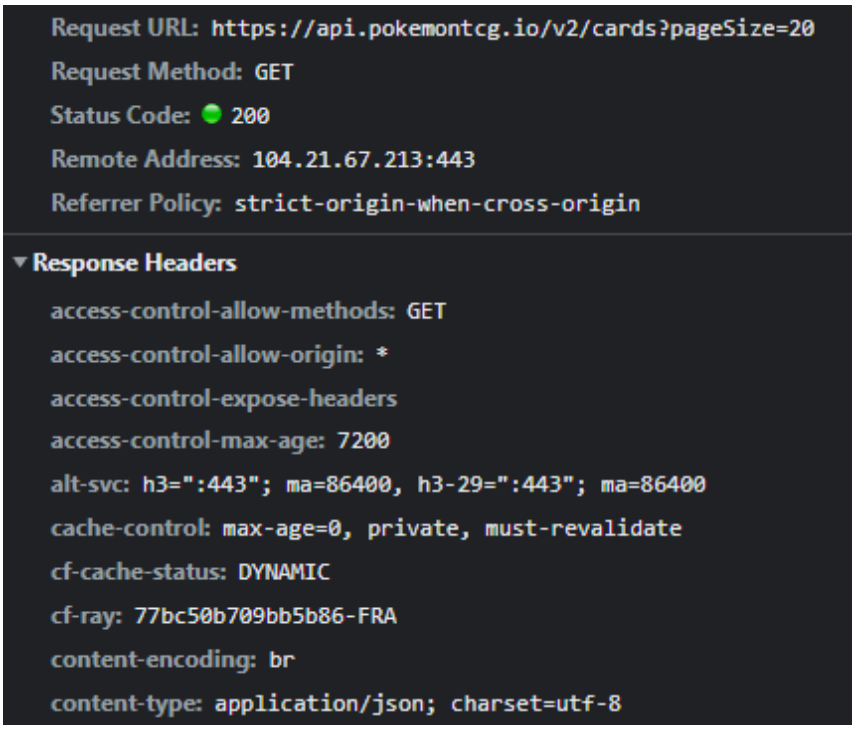
HTTP zahtjev se sastoji od nekoliko elemenata. Prvo što se dodaje u HTTP zahtjev je metoda koja određuje metodu koju klijent želi izvršiti. Uobičajene metode su GET, POST, PATCH, PUT, DELETE, CONNECT, HEAD, OPTIONS, TRACE. Metoda za dohvaćanje podataka je GET a metoda POST je metoda za slanje podataka s neke HTML forme. Iduća stvar od koje se sastoji HTTP zahtjev je URL putanja do podataka koje korisnik želi dohvatiti. Poslije toga se dodaje verzija HTTP protokola. Postoji i dio za zaglavlja u koji se mogu upisivati dodatne informacije koje je potrebno definirati poslužitelju.

Osim ovih nabrojenih elemenata, neke naredbe poput naredbe POST, zahtijevaju tijelo zahtjeva u koje je potrebno dodati informacije koje se žele prenijeti na poslužitelja.



Slika 4. Izgled zaglavlja zahtjeva

Kao što je izgled zahtjeva detaljno definiran, odgovor također ima formu koju mora zadovoljavati. Tako se HTTP odgovor sastoji od: Verzije HTTP protokola, statusnog koda koji daje informacije o uspješnosti zahtjeva, statusne poruke te HTTP zaglavlja. Isto kao i kod zahtjeva postoji tijelo odgovora koje može sadržavati neke dohvaćene podatke.



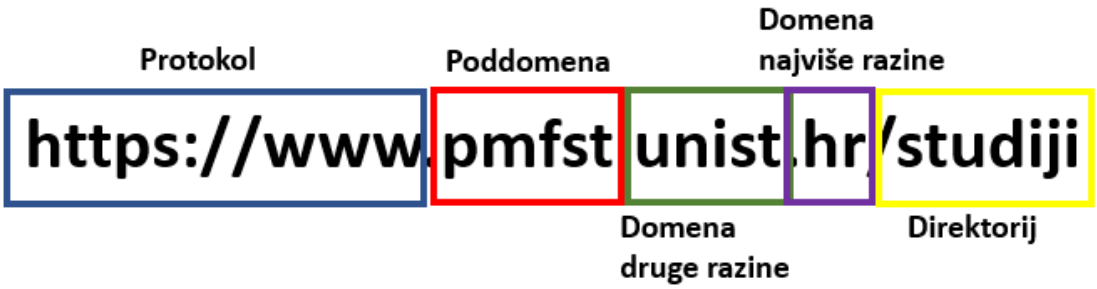
Slika 5. Izgled HTTP odgovora

Statusnih kodova ima veliki broj i stoga su podijeljeni u skupine po brojevima:

- Informativni odgovori – 100 - 199
- Uspješni odgovori – 200 - 299
- Poruke preusmjeravanja – 300 - 399
- Pogreške od strane klijenta – 400 - 499
- Pogreške od strane poslužitelja – 500 - 599

2. 2. URI

Jedinstveni identifikator resursa ili URI (engl. Uniform Resource Identifier) je niz znakova koji jedinstveno označava logički ili fizički izvor informacija na internetu. Ovi identifikatori olakšavaju mrežnim protokolima usmjeravanje podataka na mreži. Najpoznatija vrsta URI je URL odnosno jedinstveni lokator resursa što se obično smatra adresom mrežnog mjesta i služi za jednoznačno označavanje te lokacije. [4]



Slika 6. Prikaz dijelova URL-a

2. 3. HTML

HTML (engl. HyperText Markup Language) je prezentacijski jezik za izradu mrežnih stranica. HTML je osnovni građevni element od kojeg je izrađen Web. Pomoću njega definiramo strukturu i značenje sadržaja koji se prezentira na mrežnoj stranici. HTML nije programski jezik, što znači da HTML ne može izvršavati logičke naredbe, već samo služi za prezentaciju hipertekstualnih dokumenata. Hipertekst se odnosi na poveznice koje povezuju dvije mrežne stranice ili dokumenta unutar jedne web stranice. Poveznice su temeljna značajka Weba. Postavljanjem sadržaja na Internet i povezivanjem na druge stranice korištenjem

poveznica ljudi stvaraju Web. HTML koristi posebne oznake za definiranje teksta, slika ili drugog sadržaja u mrežnom pregledniku. Neke od tih oznaka su `<head>`, `<title>`, `<body>`, `<header>`, `<footer>`, `<article>`, `<section>`, `<p>`, `<div>`, `<span>` i mnoge druge.

Prva verzija HTML-a je bila osmišljena od strane Tim Berners-Leeja 1993. godine. Od onda do sada postoje razne iteracije HTML koje su donosile nove oznake u HTML. Te oznake omogućuju dodavanje raznovrsnijeg sadržaja na mrežne stranice. [5]

Kako HTML sam po sebi ne može izvršavati logičke operacije niti uređivati vizualni izgled i prezentaciju dokumenta u tu svrhu se koriste druge tehnologije.

2. 4. CSS

CSS (engl. Cascading Style Sheets) je digitalni stilski jezik koji se koristi za uređivanje vizualnog izgleda hipertekstualnog dokumenta. Može biti napisan u HTML ili XML tehnologiji. U CSS-u se opisuje na koji način se elementi u dokumentu trebaju prikazivati u sučelju. Dokument za uređivanje stila u CSS-u se sastoji od pravila. Svako pravilo sastoji se od selektora i deklaracijskog bloka. Deklaracijski blok označen je vitičastim zagradama unutar kojih se nalaze deklaracije. Te deklaracije određuju način prikazivanja elementa na ekranu. Svaka deklaracija se sastoji od svojstva i vrijednosti koji su odvojeni dvotočkom. Selektor označava na koji dio u HTML dokumentu će se primijeniti definirani stil. [6]

Taj selektor može biti:

- Osnovna HTML oznaka poput oznake `title`
- Atribut koji je jedinstvena oznaka elementa – id
- Neka klasa - `class`,
- Elementi u odnosu na druge elemente. Postoje četiri selektora za definiranje odnosa elemenata: `div p, Div + p, div > p, div ~ p`
- Pseudoklase, poput `:hover, :first-child, :last-child`, koje omogućuju opisivanje informacija koje nisu dostupne DOM-u (engl. Document Object Modelu).

2. 5. JavaScript

HTML kojim se opisuje struktura mrežne stranice i CSS kojim se definiraju stilovi podataka na toj mrežnoj stranici, nisu pravi programski jezici već opisni jezici koji u kombinaciji s JavaScriptom mrežne stranice čine interaktivnima.

JavaScript je programski jezik pomoću kojeg statičnim komponentama koje ne dopuštaju interakciju s korisnikom na mrežnim stranicama dodajemo funkcionalnost i interaktivnost. Ovaj programski jezik napisao je Brend Eich 1995. godine. Razvijan je za Netscape 2.

JavaScript je skriptni programski jezik. Skriptni jezici su programski jezici koji se koristi za automatiziranje nekih procesa koje bi korisnik trebao inače raditi sam. Osim toga služe za integraciju i komunikaciju s drugim programskim jezicima.

JavaScript kôd možemo dodati na mrežnu stranicu na tri načina:

- Dodavanjem JavaScripta izravno u liniju kôda. Primjer toga je izvršavanje nekog kôda kada se odvije neki događaj.
- Korištenjem `script` oznake. Postavljanjem `script` oznaka i pisanjem JavaScript koda u njih definiramo skriptu koja će se izvršavati.
- Korištenje `script` oznaka ali umjesto pisanja kôda definira se vanjska datoteka u kojoj je napisan JavaScript kôd. To je moguće napraviti pomoću atributa `src` u oznaci.

U današnje vrijeme više od 95% aktivnih mrežnih stranica na internetu koristi JavaScript u nekoj mjeri. [7]

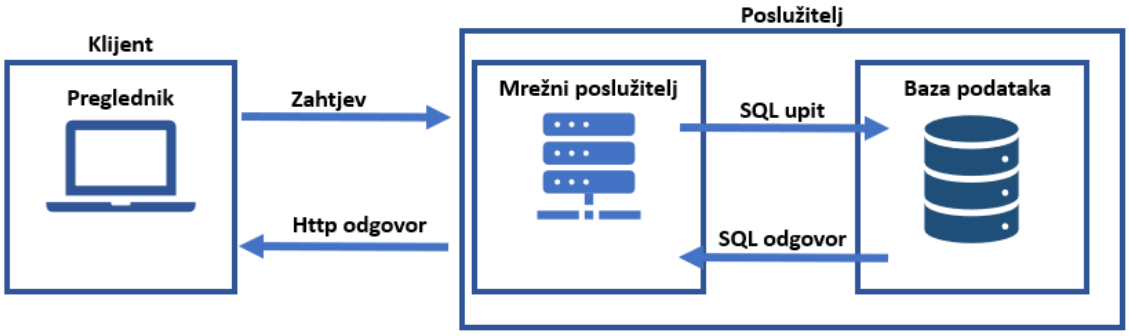
Svoju popularnost JavaScript duguje činjenici da postoji niski prag za početak učenja ovog programskog jezika. Osim toga JavaScript je jako raznovrstan programski jezik koji programeru nudi mogućnosti da pokrije različite aspekte svoje aplikacija. Taj proces još dodatno olakšavaju razni okviri mrežnih aplikacija koji imaju striktnu namjenu za svoje korištenje. Ako je potrebno koristiti JavaScript na *front-end* dijelu aplikacije postoji okvir React.js koji olakšava programiranje ili Node.js ako je potrebno raditi stvari na *back-end* dijelu aplikacije. Još neki od poznatih JavaScript okvira su: Angular. Vue.js, Ember.js i drugi. JavaScript također nudi i mogućnost izrade aplikacija za mobilne uređaje korištenjem React Nativea.

3. TROSLOJNA ARHITEKTURA WEBA

Troslojna mrežna arhitektura je konceptualni način organizacije strukture mrežnih programskih podrški. Troslojna mrežna arhitektura se odnesi na 3 logičko - fizičke razine koje predstavljaju zajednicu i možemo ih podijeliti na:

1. prezentacijski sloj ili korisničko sučelje
2. domenski sloj koji služi za procesiranje podatak
3. podatkovni sloj u kojem se pohranjuju podatci te se upravlja njima

Troslojna mrežna arhitektura definira način komunikacije između klijenta i poslužitelja. Glavna prednost troslojne arhitekture je to što svaki sloj ove cjeline radi na zasebnoj infrastrukturi koja nije ovisna o drugim dijelovima da bi samostalno funkcionirala te se stoga svaki sloj ove arhitekture može istovremeno samostalno i neovisno o drugim dijelovima razvijati, ažurirati ili skalirati po potrebi. [8]



Slika 7. Shema troslojne arhitekture

3. 1. Prezentacijski sloj

Prezentacijski sloj je „najviši“ sloj troslojne arhitekture koji je vidljiv krajnjim korisnicima, pa ga još možemo i nazvati korisničko sučelje. Prezentacijski sloj je sloj koji ima svrhu omogućavanja komunikacije između korisnika i mrežne aplikacije. U prezentacijskom sloju korisniku se prikazuju podaci te se prikupljaju podatci koje korisnik unosi. Prezentacijski sloj se može pokretati na mrežnim preglednicima ili kao aplikacija na radnoj površini.

3. 2. Aplikacijski sloj

Aplikacijski sloj koji je još poznat i kao logički sloj ili 'srednji' sloj arhitekture, smatra se glavnim dijelom aplikacije. Kroz njega prolazi sva komunikacija koja se odvija između prezentacijskog sloja i podatkovnog sloja. Ovaj sloj služi za obradu podataka koje su prikupljene u prezentacijskom sloju od strane korisnika s jedne strane i dohvaćanje te pripremanje podataka za prezentacijski sloj s druge strane. Kako bi omogućio tu funkcionalnost aplikacijski sloj može dodavati, brisati ili mijenjati podatke koje primi iz jednog sloja prije nego ih pošalje u idući sloj. Za izradu aplikacijskog sloja najčešći programski jezici su Python, Java, Perl, PHP ili Ruby. Komunikaciju s podatkovnim slojem ostvaruje pomoću poziva programskog sučelja za aplikacije (engl. Application programming interface – API).

3.2.1. API

Programska sučelja za aplikacije – API (engl. Application programming interface) pojednostavljuju razvoj programske podrške jer omogućuju aplikacijama sigurnu i laganu razmjenu podataka. Izradom API-ja omogućava se pristup pojedinim podacima koji bi bili dostupni samo vlasnicima aplikacija čime se ostvaruje mogućnost korištenja tih podataka u razvijanju aplikacija treće strane. Najveća prednost API-ja leži u tome što korisnik nekog API-ja ne mora znati na koji način API implementira funkcionalnost, odnosno na koji način se odvija razmjena podataka.

Programska sučelja za aplikacije imaju striktno definirana pravila koja oblikuju način komunikacije između korisničkog sučelja i mrežnog poslužitelja. API ima ulogu posredničkog sloja koji obrađuje podatke prilikom prijenosa podataka između slojeva. Funkcionalnost API-ja možemo objasniti kroz nekoliko koraka:

1. Aplikacija klijenta šalje API poziv sa svrhom dohvaćanja podataka. Ta radnja se naziva zahtjev (engl. Request). Ovaj zahtjev se šalje od aplikacije do mrežnog poslužitelja putem jedinstvene adrese resursa - URI (engl. Uniform Resource Identifier). Zahtjev mora sadržavati oznaku o kojoj vrsti upita se radi, zaglavlje i u nekim slučajevima tijelo upita.

2. Nakon validacije zahtjeva kojega primi, API upućuje poziv udaljenom programu ili mrežnom poslužitelju.

3. Mrežni poslužitelj šalje odgovor API-ju s željenim informacijama.

4. API prima podatke i striktno definiranim pravilima prosljeđuje podatke klijentu koji je uputio zahtjev.

Dakle na API možemo gledati kao na vrstu ugovora, čija dokumentacija predstavlja sporazum između dviju strana. Ako jedna strana pošalje zahtjev koji je strukturiran na dogovoreni način, druga strana koja primi taj zahtjev zna na koji način mora poslati odgovor.

API pojednostavljuje način povezivanja vlastite infrastrukture putem razvoja aplikacija u oblaku ali i omogućuj dijeljenje podataka s drugim klijentima i vanjskim korisnicima. Javni API predstavlja poslovnu vrijednost jer olakšava i stvara nove načine povezivanja s partnerima te predstavlja način na koji se podatci u privatnom vlasništvu mogu monetizirati.

Pravila izdavanja API-ja (engl. Release policies) možemo podijeliti u tri kategorije:

- Privatni – služi za internu uporabu unutar neke kompanije.
- Partnerski – API se dijeli s dogovorenim poslovnim partnerima. Ovakav način dijeljenja API-ja predstavlja način na koji se može monetizirati.
- Javni – API koji je dostupan svima na korištenje. Time se omogućava korištenje informacije iz nekog izvora širokim masama ljudi što može dovesti do novih inovacija u tom području.

Ovisno o vrstama poruka koje API razmjenjuje s korisnikom, API-je možemo podijeliti na 4 kategorije:

- SOAP API – Ovi API koristi Simple Object Access Protocol. Klijent i poslužitelj razmjenjuju poruke koristeći XML format. Ovo je manje fleksibilan vrsta API-ja koji je bio popularniji u prošlosti
- RPC API – RPC označava Remote procedure calls . Kod ovih API-ja klijent dovršava funkciju na poslužitelju nakon čaga poslužitelj šalje izlaz funkcije nazad klijentu.
- Websocket API - moderni web API koji koristi JSON (JavaScript Object Notation) objekte za prijenos podataka. Websocket API omogućuje dvosmjernu komunikaciju između klijenta i poslužitelja. Zahvaljujući dvosmjernoj komunikaciji poslužitelj može slati povratne poruke povezanim.
- REST API – Ova vrsta je najpopularnija i najfleksibilnija za korištenje. Kada korisnik napravi zahtjev putem RESTful API-ja, zahtjev se prenosi kao prikaz stanja resursa podnositelju zahtjeva ili krajnjoj točki. Kod REST API-ja odgovori na zahtjeve se šalju putem HTTP-a u jednom od slijedećih formata: JSON, HTML, XLT, Python, PHP ili čisti tekst. JSON je općenito najpopularniji format za slanje jer omogućava čitljivost podataka ljudima i strojevima. S obzirom na to da REST API koristi HTTP za komunikaciju treba imati na umu da su zaglavlja i parametri bitan dio komunikacije. U njima se nalaze identifikatori o vrsti metode koja se šalje,

informacije o meta podacima zahtjeva, autorizaciji, jedinstvenom identifikatoru izvora, pred memoriji, kolačićima i ostalome. Razlikuju se zaglavlja zahtjeva i zaglavlja odgovora, koji imaju striktno definirane strukture o HTTP vezi i statusnim kodovima.

3. 3. Podatkovni sloj

Podatkovni sloj koji se još naziva i sloj baze podataka je dio ove arhitekture u kojem se spremaju podaci. Nakon što korisnik unese podatke u prezentacijskom sloju oni se obrađuju u aplikacijskom sloju i spremaju u podatkovnom. Isto tako ako korisnik zatraži neki podatak koji se potencijalno nalazi u bazi podataka, korisnikov zahtjev se obrađuje u aplikacijskom sloju i kao odgovor mu se šalju podatci koji se nalaze u podatkovnom sloju. Za upravljanje bazama podataka koriste se posebni programi. Ovisno o tome da li je baza podataka relacijska ili nerelacijska ovisit će koji program za upravljanje bazom podatak se koristi. Najpoznatiji programi za upravljanje relacijskim bazama podataka su MySQL, MariaDB i Oracle dok kod nerelacijskih su najpoznatiji MongoDB, Cassandra CouchDB.

4. WEB APLIKACIJE

Web aplikacije su vrste programa koje se spremaju na udaljenim poslužiteljima i putem mrežnih preglenika se isporučuju na računala korisnika. Mnogi mrežni servisi se ubrajaju u web aplikacije. Prema Jaerl Remicku svaka mrežna stranica koja obavlja neku funkcionalnost za korisnika se može smatrati web aplikacijom. [9]

S obzirom na to da se web aplikacije pokreću u mrežnim preglednicima, korisnici nemaju potrebu preuzimati te aplikacije na vlastite uređaje. Većina web aplikacija je pisana korištenjem tehnologija JavaScript, HTML i CSS na *front-end* dijelu aplikacije. Za *back-end* dio aplikacije se koriste tehnologije Java, Ruby, Pyhon i drugi. Prednost ovih aplikacija leži u mogućnostima da istodobno aplikaciji pristupa više korisnika i koristi neometano, u tome što aplikaciju ne treba instalirati kako bi se mogla koristiti te joj se može pristupiti s različitih uređaja.

Mobilne web aplikacija se često uspoređuju s nativnim aplikacijama koje se razvijaju specifično za određeni uređaj ili platformu te se instaliraju na tom uređaju. Pri razvoju web aplikacija često se teži da njihova funkcionalnost bude što sličnija nativnim aplikacijama.

U razvoju web aplikacija postoje dva istaknuta uzorka dizajna koja definiraju način razvoja web aplikacije, konačni proizvod te funkcionalnost tog konačnog proizvoda. Prvi uzorak dizajna koji je stariji, pojavio kada i prve mrežne stranice na internetu, su višestranične mrežne stranice (engl. Multi Page Application). Novi moderniji uzorak dizajna web aplikacija je jednostranična mrežna aplikacija (engl. Single Page Application) koji je postao popularan razvojem AJAX tehnologije.

4. 1. MPA

Višestranične mrežne aplikacije - MPA (engl. Multi Page Appliaction) su tradicionalna vrsta web aplikacija koja se pojavila prva. Ovakve web aplikacije se sastoje od hipertekstualnih dokumenata na kojima se uglavnom prikazuju statički podaci (tekst, slike) i povezuju se poveznicama s drugim stranicama. Ove web aplikacije se sastoje od velikog broja stranica. Prilikom prelaska na drugu stranicu mrežni preglednik mora prvo poslati zahtjev za HTML dokumentima koji se trebaju prikazati na toj stranici, ponovno u potpunosti učitava podatke i ispočetka prikazati sve komponente koje se nalaze na ekranu, pa čak i one poput zaglavlja stranice (engl. Header) i podnožja stranice (engl. Footer) koji su jednaki na svim stranicama.

Kada korisnik želi pristupiti početnoj stranici Web aplikacije on putem URL adrese šalje zahtjev mrežnom poslužitelju. Mrežni poslužitelj obrađuje zahtjev i ovisno o uspješnosti tog zahtjeva šalje nazad korisniku HTML dokument koji se prikazuje u mrežnom pregledniku korisnika. Ukoliko korisnik klikne na neku poveznicu koja se nalazi na toj početnoj stranici, šalje se novi zahtjev mrežnom poslužitelju. Poslužitelj opet obrađuje taj zahtjev i vraća HTML dokument ukoliko je zahtjev uspješan. Taj odgovor uzrokuje ponovno učitavanje stranice s novim podacima.

Slika 8. Životni ciklus MPA

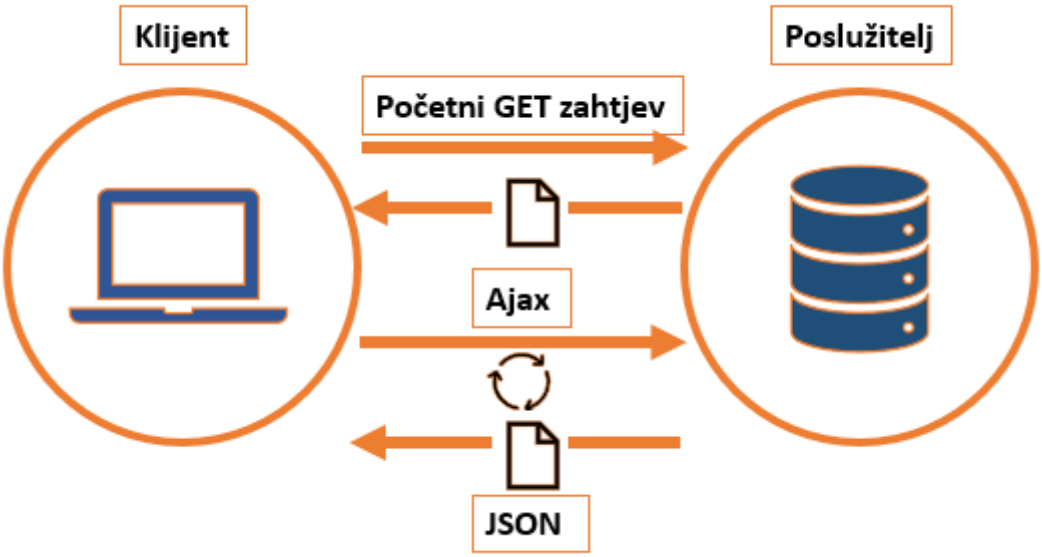
Zbog ovakvog načina funkcioniranja brzina učitavanja mrežnih stranica se može znatno povećati što nije efekt koji se želi postići. Kako bi se postigla bolja responzivnost aplikacija koristi se tehnologija AJAX, koji omogućava dohvaćanje podataka bez ponovnog učitavanja cijele web stranice.

4. 2. SPA

Jednostranične mrežne aplikacije - SPA (engl. Single Page Application) je vrsta implementacije Web aplikacija koja se pokreće na jednom jedinstvenom HTML dokumentu i onda po potrebi dinamički ažurira podatke na toj mrežnoj stranici koristeći AJAX i XML. HTML dokument predstavlja kostur mrežne stranice. Umjesto da se svi elementi mrežne stranice šalju u HTML dokumentu, SPA trebaju JavaScript kôd za stvaranje sadržaja. SPA se pokreću unutar mrežnih preglednika i ne zahtijevaju ponovna učitavanja tijekom korištenja.

SPA mogu dohvatiti sav kôd koji su potreban za prikaz neke Web aplikacije pri početnom učitavanju podataka. Zbog toga to početno učitavanje podataka kod kompleksnih aplikacija može biti značajno i uzrokovati duljim vremenom učitavanja. Nakon toga ove aplikacije mogu učitavati podatke dinamički kako bi ažurirali podatke ovisno o korisničkom unosu ili ovisno o nekim drugim događajima. Podaci se mogu primati u obliku JSON zapisa nakon čega se samo prikazuju na za to predviđena mjesta u aplikaciji, čime se izbjegava ponovno učitavanje cijele stranice.

Kao posljedica toga SPA smanjuju opterećenje mrežnih poslužitelja te cijelo iskustvo korištenja web aplikacije čine ugodnijim jer nakon prvog učitavanja aplikacije ona ne treba dohvaćati dodatne HTML dokumente, CSS stilove ili JavaScript kôd i prelazak s jednog ekrana na drugi čini nesmetanim.



Slika 9. Životni ciklus SPA

Neke od najpoznatijih mrežnih stranica koje koriste ovu vrstu web aplikacija su Gmaila, Netflixa, Facebook i GitHub.

Jedan od glavnih problem SPA leži u optimizacija za internet pretraživače - SEO (engl. Search Engine Optimization). Optimizacija za internet pretraživače određuje mjesto koje zauzima web aplikacija u rezultatima internetskih pretraživača. Ovdje dolazi do problema jer programi za pretraživanje interneta (engl. Web crawler) nemaju uvid u sadržaj web aplikacije koja se generira na ovaj način. Kako bi pretraživač dobio sadržaj web aplikacije on mora izvršit JavaScript naredbe, a tu mogućnost nemaju svi internetski pretraživači.

Postoje dvije arhitekture SPA; aplikacije koje se renderiraju na strani mrežnog poslužitelja - SSR (engl. Server Side Rendering) i one koje se renderiraju u pregledniku

korisnika - CSR (engl. Client Side Rendering). Renderiranje je pojam koji opisuje proces prikazivanja sadržaja mrežne stranice u korisničkom pregledniku na temelju podataka koji su dani u nekom formatu npr. JSON format.

4.2.1. CSR

Renderiranje na strani klijenta se odnosi na proces generiranja HTML-a, CSS-a i JavaScript-a na strani klijenta, umjesto na poslužitelju. To znači da mrežni preglednik korisnika koristi JavaScript kôd kako bi stvorio i prikazao stranicu.

Jedna od glavnih prednosti CSR je stvaranje dojma da se stranice brže učitava jer se ne mora čekati da poslužitelj generira HTML dokument, pošalje ga pregledniku i onda ga prikaže. Umjesto toga, preglednik koristi JavaScript da generira stranicu, što znači da se korisnik može kretati po stranici i vidjeti promjene odmah. Činjenica da mrežni preglednik mora učitati i izvršiti JavaScript kôd može biti dvosjekli mač jer učitavanje JavaScript datoteke koja ima veliku količinu podataka ili komunikacija sa sporim poslužiteljem mogu dovesti do sporog učitavanja mrežne stranice. Osim toga CSR pruža veću interaktivnosti sadržaja mrežne stranice. Kod CSR JavaScript može reagirati na akcije korisnika (na primjer, klik na gumb) i ažurirati stranicu bez potrebe za ponovnim učitavanjem cijele stranice. To znači da se korisnika ne mora preusmjeravati na drugu stranicu kada želi promijeniti nešto na trenutnoj stranici.

Prednosti:

- CSR ne zahtijeva generiranje HTML-a za svaku pojedinačnu stranicu na poslužitelju, što znači da se potreba za resursima poslužitelja smanjuje.
- CSR omogućuje dinamičko ažuriranje stranice bez potrebe za ponovnim učitavanjem cijele stranice, što može poboljšati korisničko iskustvo i interaktivnost stranice.

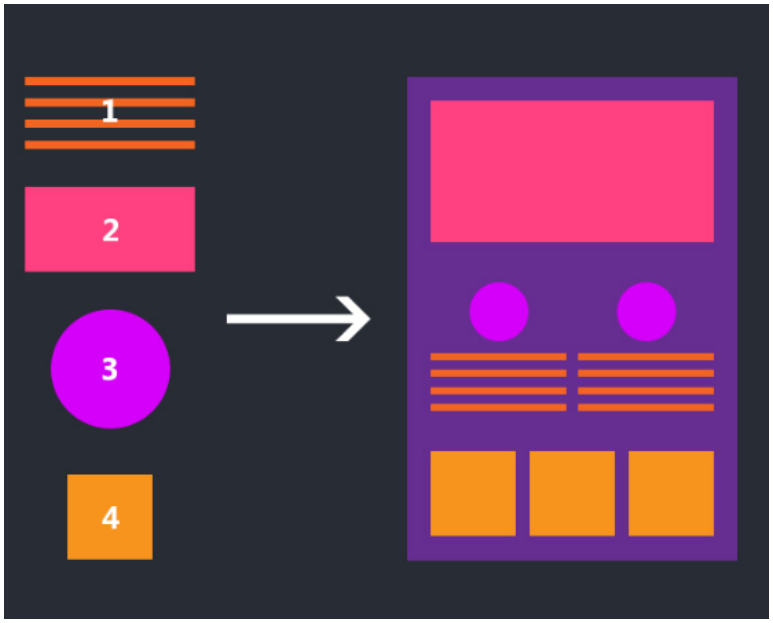
Nedostaci:

- CSR zahtijeva da se JavaScript učita i generira stranica u pregledniku, što može rezultirati sporijim vremenom učitavanja stranice za korisnika.
- Pretraživači bolje razumiju statički HTML nego JavaScript, pa CSR može rezultirati poteškoćama sa SEO-om web stranice.
- Ova vrsta generiranja zahtijeva više resursa na strani klijenta, što može utjecati na performanse kod starijih računala ili mobilnih uređaja uređajima.
- Ako korisnik onemogući JavaScript u pregledniku, stranica neće raditi ispravno jer se HTML i CSS ne mogu generirati bez JavaScripta.

Najpoznatije biblioteke za kreiranje CSR aplikacija su React, Angular, Vue i Ember.

4.2.2. React

React.js je JavaScript biblioteka otvorenog koda. Razvijena i održavana je od strane Facebooka te pojedinih programeri i tvrtki koji razvijaju biblioteke koje se mogu koristiti unutar Reacta. React je primarno stvoren kako bi olakšao razvoj interaktivnih korisničkih sučelja i web aplikacija. U Reactu programer razvija aplikaciju na način da kreira komponente. Jednom kada programer stvori jednu komponentu, ona mu omogućava da je on koristi bilo kada i bilo gdje mu je potrebna. Te komponente možemo smatrati nekakvim kockama kojima gradimo konačni proizvod, odnosno korisničko sučelje.



Slika 10. Korištenje komponenti za izradu sučelja

Glavna uloga Reacta u aplikaciji je da kontrolira prezentacijski sloj aplikacije. Umjesto da upravlja s kompleksnim grafičkim sučeljem kao jednom cjelinom React potiče programere da kompleksna sučelja razvoje na što osnovnije komponente koje će se moći ponovno koristiti. Na takav način React kombinira brzinu i efikasnost JavaScripta što mu omogućava da na efikasniji način manipulira DOM-om. DOM (engl. Document Object Model) je programsko sučelje koje omogućava podatkovni prikaz objekata od kojih se sastoji sadržaj neke web stranice. Na takav način omogućava programu koji ima uvid u DOM da direktno mijenja strukturi stil ili podatke koji se prikazuju na aplikaciji. Kod standardnih JavaScript aplikacija, promjena podataka u aplikaciji zahtjeva ručnu manipulaciju DOM-a kako bi odradio promjene. Ažuriranje podataka u DOM-u uzrokuje da se čitava mrežna stranica ponovo učita. React način kontroliranja podataka omogućava izradu jednostraničnih aplikacija. React se oslanja na virtualni DOM, što predstavlja kopiju pravoga DOM-a kojega sprema u memoriju. Kada dođe

do promjene Reactov virtualni DOM se automatski ponovo učitava s novim podacima koje je primio. Nakon toga on uspoređuje stanje virtualnog DOM-a iz memorije i stvarnog DOM-a koji se prikazuje. Nakon toga React određuje optimalan način koji zahtjeva najmanje memorije i vremena kako bi postavio promjene samo na one dijelove DOM-a koji se razlikuju od virtualnoga.

Osim samog Reacta koji je namijenjen za web aplikacije, postoje i okviri koji omogućavaju da se pomoću React kôda pišu aplikacije za mobilne uređaje ili računala. Pa tako postoji React Native okvir koji je namijenjen za pisanje nativnih aplikacija za mobitele ili pak okvir Electorn koji je namijenjen za izradu aplikacija za računala.

React komponente

U Reactu komponente su osnovni entitet od kojega se izgrađuje aplikacija. Korištenjem biblioteke React DOM omogućava se interakcija s virtualnim DOM-om. Postoje dva načina kreiranja komponenti u Reactu. Prvi način je deklariranje komponente da je funkcionalna komponenta. Komponentu deklariramo kao funkciju koja vraća neki JSX kod. Ova vrsta komponenti nekoć nije imala način da sprema vrijednosti svoga stanja. Podatke su mogle primati samo kroz atribut `props`. Zbog toga još jedan od naziva za ovu vrstu komponenti je bio *statless* komponente. Uvođenjem React *Hookova* ovo ime više nema smisla jer sada postoji način da i one posjeduju stanje i budu *statefull*.

```
function Naslov(props) {
      return <h1> {props.naslov}</h1>;
}
```

Primjer kôda 1. Definiranje funkcionalne komponente

Drugi način je deklariranje komponenti kao klase. Ovako definirane komponente imaju svoje stanje kojem se može pristupiti koristeći ključnu riječ `this`. Ovako deklarirane komponente nemaju mogućnost korištenja React *Hookova*.

```
class Welcome extends React.Component {
render() {
return <h1>Hello, {this.props.name}</h1>;
}
}
```

Primjer kôda 2. Definiranje `class` komponente

JavaScript XML

JavaScript XML ili JSX predstavlja proširenje sintakse JavaScripta. Slično kao i HTML, XML je prezentacijski jezik ali za razliku od HTML koji je dizajniran da bi služio za prikaz podataka, XML je dizajniran kako bi prenosio podatke. Bas kao i HTML, XML također koristi neke oznake kako bi definirao dijelove podataka. U XML-u ne postoje predefinirane oznake već programer mora sam definirati oznake.
JSX omogućava da se u JavaScriptu dinamički definiraju HTML elemente koji će se kasnije postavljati u DOM bez korištenja metode `CreateElement()`. React ne zahtjeva korištenje JSX jer se korištenjem samoga JavaScripta mogu kreirati komponente, ali JSX puno olakšava proces izrade.

React Hooks

U verziji Reacta 16.8 uvodi se nova značajka koja React dodatno uzdiže u usporedbi s ostalim okvirima. React Hooks su posebne vrste funkcija koje omogućavaju funkcionalnim komponentama da se ponašaju kao komponente temeljene na klasama dajući im mogućnosti kao što su korištenje internog stanja. Neki od osnovnih *Hookova* su: `useState, useEffect, useContext, useReducer, useCallback, useMemo` i tako dalje. U izradi ove aplikacije korištena su dva *Hooka*; `useState, useEffect`.

Prvi *Hook* je `useState`. Korištenje tog *Hooka* stvara varijablu stanja u kojoj se sprema neka željena vrijednost te funkciju koja postavlja to stanje.

```
const [state, setState] = useState(pocetnaVrijednost)
```

Primjer kôda 3. Korištenje `useState` *Hooka*

Hooku `useState` se može poslati neka početna vrijednost koju on prilikom renderiranja komponenti postavlja kao vrijednost stanja.

Drugi *Hook* koji je korišten je `useEffect`. Ovaj *Hook* prima neku funkciju. Njegova svrha je osigurati da se funkcija koju on prima izvrši samo nakon prvog stvaranja komponenti.

```
useEffect(getData,[])
```

Primjer kôda 4. Korištenje `useEffect` *Hooka*

Osim toga on može primiti i niz podataka koji se definira nakon funkcije koju prima i odvojen je zarezom. U taj niz postaviti pokazivače na funkcije i varijable čije mijenjanje uzrokuje ponovno izvršavanje `useEffect` *Hooka*. Dakle izvršavanje `useEffect` *Hooka* ovisi o promjenama stanja ostalih elemenata aplikacije. Prilikom korištenja React *Hookova* postoje tri pravila koja se moraju poštovati a to su:

- React *Hookovi* se mogu pozivati samo unutar React funkcionalnih komponenti
- React *Hookovi* se ne mogu pozivati uvjetno, dakle ne mogu se pozivati unutar na primjer `if` bloka
- React *Hookovi* se moraju pozivati na najvišoj razini komponente

4.2.3. SSR

Renderiranje na strani poslužitelja je proces generiranja HTML-a dokumenta mrežne stranice na poslužitelju koji se onda šalje pregledniku. S obzirom na to da se sami HTML generira na mrežnom poslužitelju, kada ga je potrebno prikazati na internetskom pregledniku potrebno je dohvatiti čitavi HTML dokument koji je generiran. Zbog toga postoji vremenska latencija od kada se zatraži mrežna stranica do kad se ona prikaze na mrežnom pregledniku. No jedom kada se učita stranica ona se sprema priručnu memoriju (engl. cache) i može se brzo dohvaćati više puta. S obzirom na to da se stranica generira na strani poslužitelja, on mora osigurati radnu memoriju koja će omogućiti generiranje te stranice.

Prednosti SSR su:

- SSR omogućuje da se HTML dokument stranice generira unaprijed na mrežnom poslužitelju, što znači da preglednik ne mora čekati da se JavaScript učita i izvrši.
- Mrežni pretraživači bolje razumiju statički HTML nego JavaScript, pa SSR može pomoći u poboljšanju SEO-a web stranice.

- Ako se web stranica koristi SSR-om, HTML se generira unaprijed i sprema se u pričuvnoj memoriji preglednika, što omogućuje da se stranica koristi i kada korisnik nema pristup internetu.

Nedostaci:

- SSR zahtijeva više resursa sa poslužitelja nego klijentsko renderiranje, jer se HTML generira za svaku pojedinačnu stranicu. Ovo može rezultirati većim brojem zahtjeva upućenih poslužitelju.
- Ponovna učitavanja korisničkog sučelja te smanjena interaktivnost web aplikacije.
- Razvoj web stranice sa SSR-om može biti složeniji nego sa klijentskim renderiranjem, jer se mora razmišljati o oba okruženja, poslužitelja i klijenta.

SSR se koristi u različitim tehnologijama i okvirima, kao što su React, Angular, Vue.js i drugi.

4.2.4. Next

Next.js je React okvir mrežne aplikacije. Next.js je razvijan od strane kompanije Vercel i služi za kreiranje SSR aplikacija. S obzirom na to da se Next.js zasniva na Reactu on zadržava većinu funkcionalnosti koju pruža React. React *Hookovi,* komponente kao građevni elementi aplikacije, JSX tipovi datoteka su dostupni i u Next.jsu. Glavna razlika ove dvije tehnologije leži u načinu renderiranja same aplikacije. Osnovni način React aplikacija je renderiranje aplikacije na strani korisnika, dok Next.js aplikaciju renderira na strani poslužitelja. [10] Zbog takvog načina renderiranja podataka Next.js dohvaća podatke na drugačiji način. Prije samog dohvaćanja važno je spomenuti jedan od najvažnijih koncepata Next.js *pre-rednering* odnosno postupak unaprijed stvaranja stranica za prikazivanje. Next.js renderira HTML za svaku stranicu unaprijed za razliku od Reacta koji taj posao obavlja u pregledniku na strani korisnika. Unaprijed renderirane stranice mogu rezultirati boljom optimizacijom za mrežne preglednike. Next.js unaprijed stvara izglede HTML stranica a tek nakon toga dodaje JavaScript kôd i čini stranicu interaktivnom. Taj proces dodavanja JavaScripta se naziva hidracija (engl. Hydration).

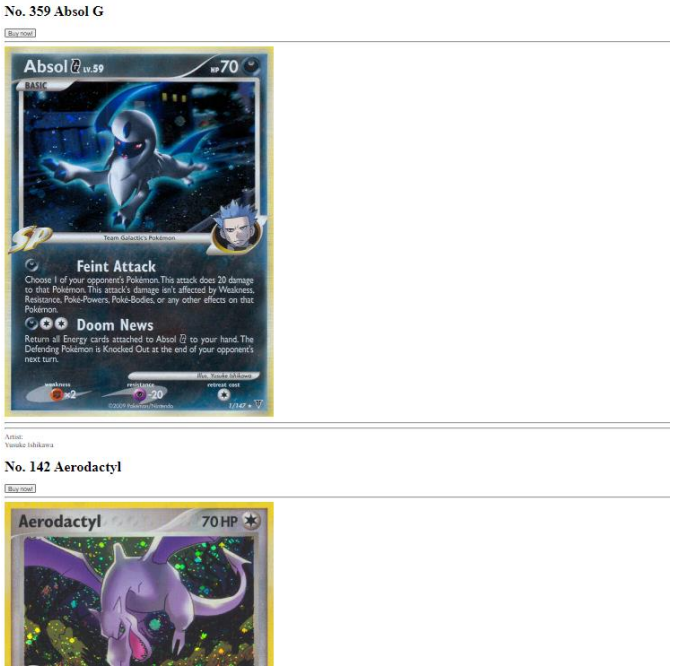
Proces hidracije i razlika u prikazivanju stranice su jasno vidljivi ukoliko se u pregledniku ugasi JavaScript. Ako se ugasi JavaScript i pokuša pokrenuti React aplikacija ona neće rediti i neće prikazati nikakve podatke.

You need to enable JavaScript to run this app.

Slika 11. Greška prilikom pokretanja React aplikacije bez JavaScripta

S obzirom na to da za prikaz komponenti na ekranu je potreban JavaScript, React nije u mogućnosti prikazati niti jedan dio aplikacije.

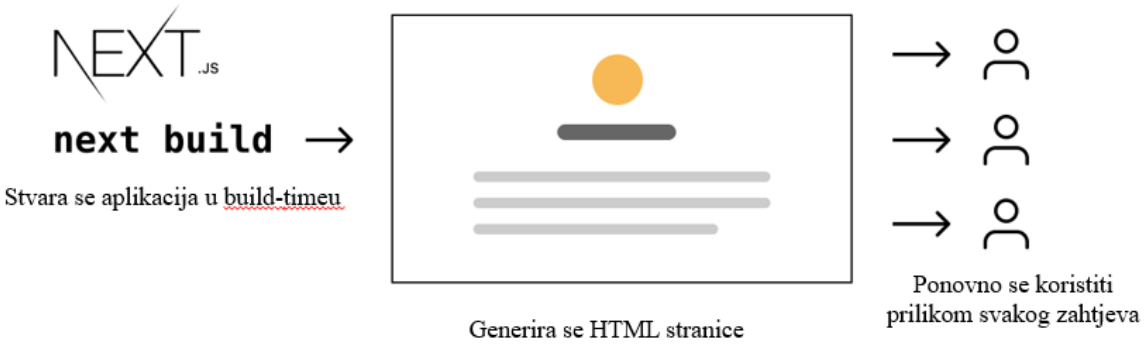
Ako se isto napravi s Next.js aplikacijom dobiti će se ekran vidljiv na slici ispod.



Slika 12. Pokretanje Next.js aplikacije bez uključenog JavaScripta u pregledniku

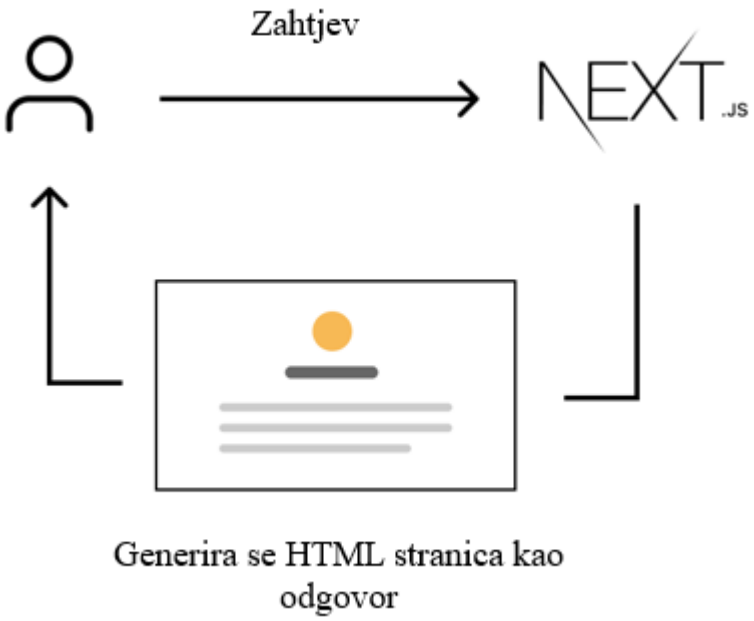
Na slici je vidljivo da Next.js isporučuje HTML stranicu i da prikazuje podatke. Ono što nedostaje su CSS stilovi koji bi se trebali isporučiti putem JavaScripta u procesu hidracije aplikacije.

U Next.jsu postoje dva načina na koji se unaprijed renderira stranica aplikacije: statičko generiranje i renderiranje na strani server (SSR). Statičko generiranje je metoda unaprijed stvaranja stranice koja generira HTML u vremenu izgradnje (engl. build time). Ista HTML stranica se ponovo koristi za svaki zahtjev. Ovaj način je pogodan za izradu stranica koje služe za prikaz statičkih podataka.



Slika 13. Statičko generiranje stranice

Renderiranje na poslužitelju je vrsta unaprijed renderiranja stranice koja generira HTML za svaki zahtjev.



Slika 14. Stvaranje stranice na poslužitelju

Ako se u aplikaciji želi koristiti statičko generiranje stranice za dohvaćanje podataka se mora koristiti funkcija `getStaticProps`. To je ugrađena funkcija kojoj se može definirati funkcionalnost. Ova funkcija se izvršava prilikom *build timea*. Unutar funkcije se definira funkcionalnost dohvaćanja podataka koji se propagiraju svojstvima (engl. props) stranice.

```
export default function Home(props) { ... }
export async function getStaticProps() {
  const data = getData()
  return {
    props: ...
  }
    }
```

Primjer kôda 5. Korištenje getStaticProps funkcije

Ako se u aplikaciji želi koristiti renderiranje na poslužitelju za dohvaćanje podataka će se koristiti funkcija `getServerSideProps`. Ova funkcija se pokreće samo na strani poslužitelja i nikada u pregledniku. Ako stranica koristi `getServerSideProps` onda se dohvaćanje podataka dograda u *request timeu* i stranica se unaprijed generira sa svim dohvaćenim podacima. Kada korisnik pošalje zahtjev, Next.js šalje API zahtjev poslužitelju koji onda pokreće funkciju `getServerSideProps`.

Druga velika razlika od Reacta je u načinu na koji Next.js stvara sustav usmjeravanja na rute. Next.js koristi datotečni sustav kao podlogu za stvaranje ruta stranica. Svaka stranica u aplikaciji je povezana s rutom na temelju naziva direktorija odnosno datoteke. Svaki direktorij može imati jednu datoteku imena indeks koju Next js smatra početnom stranicom.

Dinamičko stvaranja ruta se radi na način da se oko imena datoteke koja će se dinamički generirat stavimo uglate zagrade. Na taj način Next.jsu dajemo do znanja da u tom direktoriju odnosno na toj ruti postoji stranica koja se dinamički stvara u ovisnosti o podacima koje primi.

Povezivanje između stranica se radi korištenjem React funkcionalne komponente `<Link>`. Komponenta `<Link>` ima atribut `href` koji prima putanju na koju će voditi.

5. PRAKTIČNI DIO

U svrhu analize SSR i CSR koja je tema ovog diplomskog rada napravljene su web aplikacije pomoću kojih ćemo vrišti usporedbu. Te aplikacije su napravljene korištenjem dvije različite tehnologije. Te tehnologije su React i Next.js. Biblioteka React je korištena za izradu aplikacije koja se renderira u pregledniku korisnika (CSR). React je *front-end* biblioteka koja je napravljen na takav način da po osnovnim postavkama služi za stvaranje SPA koje se izvršavaju u mrežnom pregledniku korisnika. To znači da React korištenjem JavaScripta dinamički može mijenjati izgled stranice bez dodanih zahtjeva za dohvaćanjem HTML elemenata.

Next.js je React okvir mrežne aplikacije. Okviri mrežnih aplikacija u programiranju su apstrakcije koje funkcioniranju kao nadogradnja na početnu programsku podršku i pomoću gotovih komponenti ili rješenja olakšavaju i poboljšavaju iskustvo korištenja tog programa. Korištenjem Next.js napravljena je aplikacija koja se renderira na strani poslužitelja.

Pri izradi aplikacija u ovome diplomskom radu korišten je javni REST API pod imenom Pokemon TCG API. Za izradu rada odabran je ovaj API jer je bila potrebna velika količina podataka koja će se generirati na stranici te slike koje će popratiti taj sadržaj, kako bi se mogla predočiti razlika u brzini dohvaćanja i prikazivanja podataka na oba dva načina. Ovaj API koristi unaprijed određene URL-ove, prima zahtjeve kodirane s JSON-om te vraća odgovore zapisane u JSON formatu. Pri slanju koristi standardne HTTP kodove odgovora, autentikaciju i standardne vrste zahtjeva. Dohvaćanje podataka se obavlja preko URL-a ovog API-ja kojega šaljemo kao prvi parametar. Drugi parametar je objekt koji se sastoji od nekoliko dijelova. Prvi atribut u tom objektu označava o kojoj vrsti zahtjeva se radi. Obzirom da samo dohvaćamo podatke metoda dohvaćanja će uvijek biti GET. Drugi atribut je zaglavlje. Zaglavlje je isto objekt koji se sastoji od nekih atributa. Prvi atribut je posebni API ključ koji je potreban za autentikaciju zahtjeva prilikom slanja. Taj jedinstveni ključ se može dobiti jedino ukoliko se napravi račun na njihovoj stranici i zatraži ključ. Drugi atribut je `Accept`, kojim se određuje koji tip odgovora se očekuje od mrežnog poslužitelja. Treći atribut je `content-type` koji govori mrežnom poslužitelju koju vrstu zahtjeva treba primiti od korisnika.

```
await fetch("https://api.pokemontcg.io/v2/cards/", {
      method: "GET",
      headers: {
        "X-API-KEY":"3bc13933-0aa4-4bec-89bd-
a9ca5ae35bf2",
        Accept: "application/json",
        "Content-Type": "application/json",
    }
```

Primjer kôda 6. Metoda za dohvaćanje podataka

Ovaj API koristi konvencionalne HTTP kodove odgovora koji se koriste kako bi se označio uspjeh ili neuspjeh nekog poslanog API zahtjeva. Kodovi su idući:

Kodovi u rasponu 200 označavaju uspješan zahtjev

Kodovi u rasponu 4XX označavaju pogrešku koja nije uspjela s obzirom na dane informacije (npr. neki od obaveznih parametara zahtjeva je izostavljen)

Kodovi u rasponu 5xx označavaju pogrešku sa strane poslužitelja API-ja

Kôd statua	Objašnjenje
200 - OK	Sve je u redu
400 - Bad Request	Zahtjev je odbijen, vjerojatno zbog izostanka nekog parametra u upitu
402 - Request Failed	Parametri su dobri ali zahtjev nije uspio
403 - Forbidden	Korisnik nema dopuštenje izvršiti ovaj zahtjev
404 - Not Found	Traženi podaci ne postoje
429 - Too Many Requests	Postignut je dozvoljeni broj upita
500, 502, 503, 504 - Server Errors	Greške na serveru

Tablica 1. Vrste statusa API odgovora

```
{
  "error": {
    "message": "Bad Request. Your request is either
malformed, or is missing one or more required fields.",
    "code": 400
  }
    }
```

Primjer kôda 7. Primjer JSON zapisa neuspješnog odgovora.

5. 1. Implementacija projekta korištenjem React.js

Korištenjem React.js biblioteke napravljena je jednostranična aplikacija. S obzirom na to da jednostranične aplikacije moraju imati jedan početni dokument koji će služiti za prikaz podataka u ovoj aplikaciji to je indeks.js.

```
const root =
ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>
    );
```

Primjer kôda 8. Početni dokument aplikacije

Korištenjem metode `createRoot()` dobiva se objekt `root` kojeg se može koristiti da bi se prikazale komponente u DOM-u.

Za dohvaćanje podataka iz API-ja napravljena je funkcija `getData()`.

```
export const getData = async () => {
  try {
    const res = await
fetch("https://api.pokemontcg.io/v2/cards/", {
      method: "GET",
      headers: {
        "X-API-KEY": "3bc13933-0aa4-4bec-89bd-
a9ca5ae35bf2",
        Accept: "application/json",
        "Content-Type": "application/json",
      },
    });
    if (!res.ok) {
      throw('error')
    }
    const data = await res.json()
    return(data)
  } catch (e) {}
    };
```

Primjer kôda 9. Funkcija `getData` za dohvaćanje podatak

Dohvaćanje podataka je asinkroni postupak pa zbog toga funkciju definiramo kao asinkronu funkciju korištenjem ključne riječi `async`. Dohvaćanje podataka obavlja se pomoću metode `fetch`. Prije same metode mora se postaviti ključna riječ `await`. `Async` i `await` su ključne riječi koje idu u kombinaciji, jer kada se treba označiti da se treba čekati izvršavanje neke radnje koristimo ključnu riječ `await` a samim time to znači da je taj postupak asinkron. Metoda `fetch` prima URL lokacije na kojoj se nalaze podaci koje API dohvaća. Nakon toga je dodana oznaka metode koja je u ovom slučaju GET jer se podaci samo dohvaćaju u aplikaciju. Nakon toga u upitu ide dio s zaglavljima u kojem je definiran API ključ koji omogućava autentikaciju, tip podataka koji API treba primiti te tip podataka odgovora kojeg API treba vratiti. Sve to se nalazi unutar `try – catch` bloka, jer dohvaćanje podataka može biti neuspješno. U tome slučaju ovaj blok će vratiti pogrešku a aplikacije će nastaviti funkcionirati.

```
useEffect(() => {
  const fetchData = async () => {
    const data = await getData();
    setData(data.data);
  };
  fetchData();
    }, []);
```

Primjer kôda 10. Postavljanje funkcije `getData` u `useEffect` *Hook*

Podaci se trebaju dohvatiti samo jednom prilikom prvog prikazivanja ove komponente i zbog toga je niz s ovisnostima prazan. Unutar *Hooka* je napravljena pomoćna funkcija u kojoj se obavlja dohvaćanje podataka i postavljanje tih podataka kao stanje ove komponente. Kako bi mogli koristiti stanje unutar funkcionalne komponente moramo koristi `useState` *Hook*.

```
const [data, setData] = useState({})
```

Primjer kôda 11. Definiranje varijable stanja

Hook `useState` funkcionira na takav način da se funkciji pošalje početna vrijednost, koja je u ovom slučaju prazni objekt. Funkcija vraća niz u kojem prvi element data predstavlja vrijednost stanja a drugi element `setData` je funkcija kojom možemo mijenjati to stanje.

Za izradu komponenti za prikaz podataka korištena je biblioteka StyledComponents. Ona omogućava lakše kreiranje i korištenje CSS stilova unutar Reacta.

```
import styled from "styled-components";
const StyedCard = styled.div`
  position: relative;
  width: 30%;
  margin: auto;
  margin-bottom: 50px;
  background-color: white;
  border-radius: 25px;
  border: 0.4em solid #E8CB5D;
  padding: 50px;
`;
    export default StyedCard
```

Primjer kôda 12. Primjer komponente izrađene korištenjem styledComponents biblioteke

Dodatna je i funkcionalnost prikazivanja detalja svakog pojedinog objekta, odnosno ispisivanje svih atributa objekata. Svaki objekt ima svoju zasebnu rutu koja vodi na stranicu na kojoj se ispisuju podaci samo o toj karti. Dakle dinamički su generirane rute za sve podatke koji se prime. Jedna od glavnih razlika između Reacta i Next.jsa je u načinu na koji se kreiraju rute.

```
 {Object.values(data).map((item) => {
              return (
                <Route
                  path={`/${item.id}`}
                  element={<DetailsPage card={item}
/>}
                />);
                })}
```

Primjer kôda 13. Dinamičko definiranje ruta

U Reactu za definiranje ruta se koristi biblioteka React Router Dom. Postoje i druge biblioteke koje se koriste za taj proces, ali ovo je standardna biblioteka i biblioteka koje je korištena za izradu ovog projekta. Uvođenjem ove biblioteke u projekt, omogućava se

korištenje nekih funkcionalnih komponentama kao što su `BrowserRouter, Route` i `Routes`.

`BrowserRouter` funkcionira na način da pohranjuje trenutnu lokaciju u adresnoj traci preglednika čiste URL-ove i kreće se po stranici koristeći ugrađeni povijesni stog preglednika. Korištenje čistih (engl. clean) URL-ova omogućava urednije adrese što korisnicima olakšava preglednost adrese. Osim toga poboljšava kvalitetu optimizacije za mrežne preglednike stranice.

Standardni URL	Čisti URL
`http://example.com/index.php?page=name`	`http://example.com/name`

Tablica 2. Razlika URL-ova

Sve stranice i komponente koje žele imati pristup usmjeravanju moraju se nalazit unutar `BrowserRouter` oznaka.

```
<BrowserRouter>
  {Ostale komponente}
</BrowserRouter>
```

Primjer kôda 14. Korištenje BrowserRoutera

Funkcionalnu komponentu `Routes` se može koristiti bilo gdje u aplikaciji gdje je potrebna. Ona služi za određivanje komponenti koje će se prikazivati na rutama koje su djeca trenutne rute. Prilikom promjene lokacije `Routes` pregledava sve rute koje su djeca trenutne te nakon što pronađe onu koja najbolje odgovara trenutnoj lokaciji prikazuje komponente.

```
<Routes>
    Route 1.
    Route 2.
    …
</Routes>
```

Primjer kôda 15. Korištenje `Routes` komponente

Kako bi odredili rute koje su djeca trenutne te definirali komponente koje se trebaju prikazivati na toj ruti koristimo komponentu `Route`. Pomoću atributa `path` određuje se lokacija rute a dodavanjem atributa element određujemo koja komponenta će se prikazati na toj lokaciji.

```
<Route path="/"  element={<MainScreen data={data} />} />
```

Primjer kôda 16. Definirnje putanje za rutu

5.1.1. Implementacija projekta korištenjem Next.js

U implementaciji ovog projekta korištenjem Next.js okvira ponovno su iskorištene iste komponente koje su napravljene za implementaciju Reacta projekta. Dohvaćanje podataka je odrađeno na drugačiji način, jer kao što je već navedeno to je jedna od glavnih razlika između Reacta i Next.jsa. Dakle za dohvaćanje podatka korištena je funkcija `getServerSideProps`. To je asinkrona funkcija zbog činjenice da je dohvaćanje podataka asinkroni postupak. Dohvaćanje podataka je napravljeno je s istog API-ja pa je funkcija `fetch` ostala nepromijenjena.

```
export const getServerSideProps = async () => {
  try {
    const res = await
fetch("https://api.pokemontcg.io/v2/cards/", {
      method: "GET",
      headers: {
        "X-API-KEY": "3bc13933-0aa4-4bec-89bd-
a9ca5ae35bf2",
        Accept: "application/json",
        "Content-Type": "application/json",
      },
    });
    if (!res.ok) {
      throw "error";
    }
    const data = await res.json();
    return {
      props: {
        data,
      },
    };
  } catch (e) {
    console.log(e);
  }
```

Primjer kôda 17. Definiranje funkcije `getServerSideProps` za dohvaćanje podataka

Na kraju funkcije `getServerSideProps` više se ne koristi `useState` kako bi se postavili podaci već se podaci vraćaju kao svojstva koja će komponenta primiti u objektu `data`.

```
export default function Home({ data }) {
  …}
```

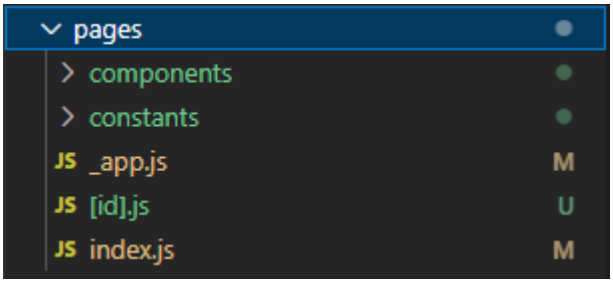
Primjer kôda 18. Primanje svojstava u objekt dana

Za stvaranje ruta nije korištenja komponenta `<Route>` već komponente `<Link>`. Ta komponenta mora imati definiranu komponentu dijete koja će služiti kao poveznica koja se definira u toj komponenti. U ovom slučaju te komponente su
bili slike. Dakle pritiskom na sliku aplikacija preusmjerava na drugu stranicu.

```
<Link href={"/" + id}>
        <StyedImg alt={"Loading"} src={images.large}
/>
</Link>
```

Primjer kôda 19. Definiranje usmjeravanja na rute

Dinamičku stranicu definiramo tako da oko imena dodamo uglate zagrade. To je u ovom slučaju `[id]`. Dakle ovako definiranom datotekom označava se da se na toj putanji šalje dinamička vrijednost `id` koju ta stranica može primiti i koristiti u prikazivanju podataka.



Slika 15. Izgled datotečnog sustava projekta

Ekranu na kojem se prikazuju detalji odabrane karte se ne šalje objekt već samo `id` - identifikator pojedine karte. Nakon toga u *pre-redneru* svaki ekran dohvaća podatke samo one karte koju treba prikazat i generira ekran.

```
  export const getServerSideProps = async (context)
=> {
    const id = context.params.id;
    const res = await
fetch(`https://api.pokemontcg.io/v2/cards/${id}`, {
      method: "GET",
      headers: {
        "X-API-KEY": "3bc13933-0aa4-4bec-89bd-
a9ca5ae35bf2",
        Accept: "application/json",
        "Content-Type": "application/json",
      },
    });
    const data = await res.json();
    return {
      props: {
        card: data,
      },
    };
       };
```

Primjer kôda 20. Dohvaćanje podataka jednog objekta

6. ANALIZA APLIKACIJE KORISTENJEM PROGRAMA LIGHTHOUSE

Za analizu aplikacija korišten je alat Lighthouse. Lighthouse je alat otvorenog koda koji služi za automatizirano vrednovanje i poboljšanje kvalitete web stranica. Razvijan je od strane Googla. S preko 8.5 milijardi pretraživanja na dnevnoj bazi i najkorištenijim mrežnim pretraživačem, Google ima glavnu riječ prilikom vrednovanja stranica koje želi prikazati u svojom pretraživaču. [11] Ovaj program se može koristiti na više načina:

- Chrome DevToolsu
- Google dodatak
- Node module
- Web aplikacija

U analizi kvaliteta Reacta i Next.js aplikacija korišten je Lighthouse iz Chrome DevToolsa. Ovaj program pruža uvid u kategorije vrednovanja mrežne stranice na temelju kojih se može vidjeti koliko kvalitetno su pojedini dijelovi stranice realizirani te pruža opcije kako ih poboljšati. Lighthouse nudi pet kategorija vrednovanja web aplikacije: izvedba aplikacije (engl. Performance), pristupačnost (engl.Accessability), korištenje najboljih praksi u kreiranju stranice (engl. Best Practices), SEO (engl. Search Engine Optimization) i progresivna web aplikacija (engl. Progressive Web App). [12]

6. 1. Izvedba aplikacije

U ovoj kategoriji vrednovanja program Lighthouse računa stavke vezane za brzine učitavanja podataka u aplikaciji . Za analizu tih brzina Lighthouse koristi šest metrika:

- Prvo postavljanje sadržaja (engl. First Contentful Paint - FCP) – ova metrika označava vrijeme koje je potrebno da prvi tekst ili slika postanu vidljivi korisniku stranice.
- Postavljanje najvećeg sadržaja (engl. Largest Contentful Paint - LCP) – ova metrika označava vrijeme kada sadržaj s najvećim količinom podataka na stranici postane vidljiv korisnicima.
- Oznaka brzine (engl. Speed Index - SI) – Oznaka brzine pruža jedinstvenu metriku koja prikazuje brzinu kojom se sadržaj stranice učitava
- Vrijeme potrebno da stranica bude interaktivna (engl. Time to Interactive - TTI) – Ova metrika označava vrijeme potrebno da bi korisnik bio u mogućnosti u potpunosti stupiti u interakciju sa stranicom i sadržajem.

- Ukupni pomak rasporeda (engl. Cumulative Layout Shift - CLS) – Ova metrika računa vizualnu stabilnost stranice prilikom učitavanja komponenti.
- Ukupno vrijeme blokiranja (engl. Total Blocking Time - TBT) – ova metrika mjeri vrijeme između prvog postavljanja sadržaja do vremena kada je stranica postaje interaktivna.

Google daje neke smjernice kojima se može poboljšati ova kategorija. [13] Smjernice su slijedeće:

- Smanjiti resurse koji blokiraju prikaz
- Posluživati slike u next-gen formatima
- Omogućiti kompresiju teksta
- Ukloniti neiskorišteni CSS
- Osigurati da tekst ostane vidljiv prilikom učitavanja webfonta
- Koristiti učinkovite načine spremanja statičkih podataka u pred memoriju
- Izbjegavati korištenje slika većih od ekrana
- Odgoditi učitavanje slika koje se ne prikazuju
- Smanjiti ili komprimirati JS I CSS
- Izbjegavati pretjeranu veličinu DOM-a
- Smanjiti vrijeme pokretanja JavaScripta
- Smanjiti opterećenost glavne dretve

6. 2. Korištenje najboljih praksi

Ova kategorija testira 16 najboljih praksi kojih se treba držati prilikom kreiranja mrežnih stranica. Lighthouse uglavnom testira sigurnosne aspekte web stranica i moderne standarde razvoja web stranica. Provjerava koristi li stranica HTTPS I HTTP/2, provjerava dolaze li resursi iz pouzdanih izvora i procjenjuje ranjivost JS biblioteka, sigurnost veze s bazom podataka korištenje zastarjelih API-ja I slično.

6. 3. Optimizacija za internetske pretraživače

Lighthouse provodi testove kojima utvrđuje koliko dobro mrežne tražilice mogu indeksirati mrežnu stranicu ili aplikaciju i prikazati je u rezultatima pretraživanja. Ovi testovi

su znatno ograničeni i stoga testiraju neke osnovne stvari na stranici poput meta podataka ili zaglavlja. Ova kategorija bi trebala davati maksimalan rezultat kako bi se postigla osnovna razina zadovoljavajuće optimizacije za mrežne preglednike.

6. 4. Pristupačnost

Ova kategorija ispituje koliko dobro mrežnu stranicu ili aplikaciju mogu koristiti osobe s invaliditetom. To uključuje testove važnih elemenata kao što su gumbi ili poveznice, kako bi se vidjelo jesu li dovoljno dobro opisani ili je li slikama dodijeljen alternativni atribut tako da se vizualni sadržaj može opisati i čitaćima zaslona.

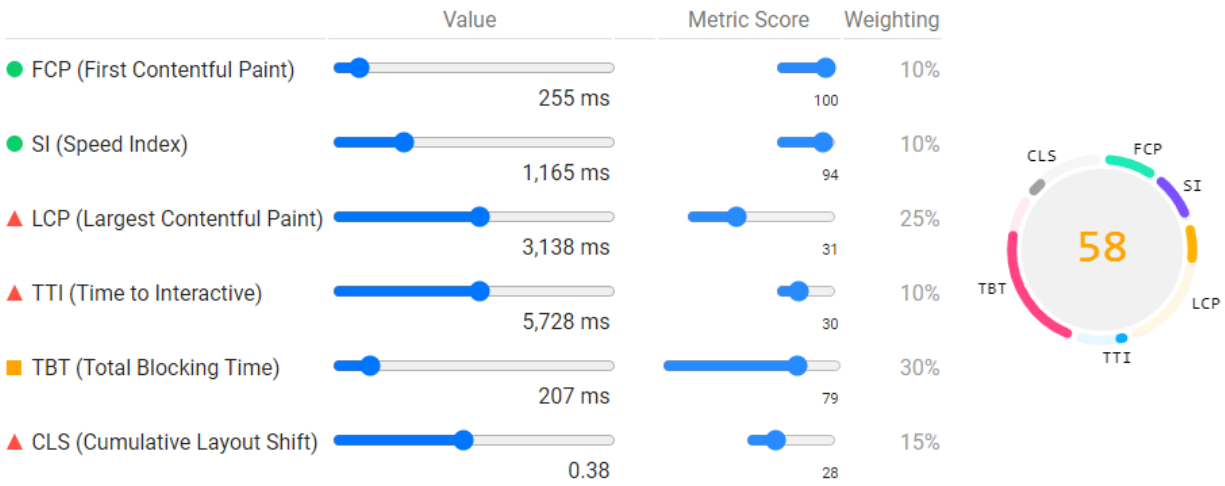
6. 5. Definiranje kriterija

Uspoređivanje prikazivanja će se vršiti na dvije funkcionalno iste aplikacije; jednom napravljenom u Next.jsu i jednom napravljenom u Reactu. React aplikacija renderira stranicu na strani klijenta, odnosno u mrežnom pregledniku, dok Next.js to radi na strani servera i isporučuje već gotovi HTML dokument. Usporedba će se vrišti na 3 različite veličine objekata; 10 objekta, 50 objekata i 100 objekata. Program Lighthouse je korišten za vrednovanje aplikacija. Rezultati testiranja mogu varirati od testa do testa. To može biti posljedica opterećenosti računala i mreže na kojima su testirane aplikacije. Prikazani rezultati su najbolji rezultati dobiveni u pojedinom testu. Veličina pojedinog objekta odabranog uzorka je u rasponu od 487kB do 1.9MB. Fokus analize će biti na prvoj kategoriji – izvedba aplikacije. Svaka od šest metrika u toj kategoriji ima zasebnu težinu koja utječe na konačnu ocjenu analize:

FCP metrika – 10%
SI metrika – 10%
LCP metrika – 25%
TTI metrika – 10%
TBT metrika – 30%
CLS metrika – 15%

6.5.1. Analiza

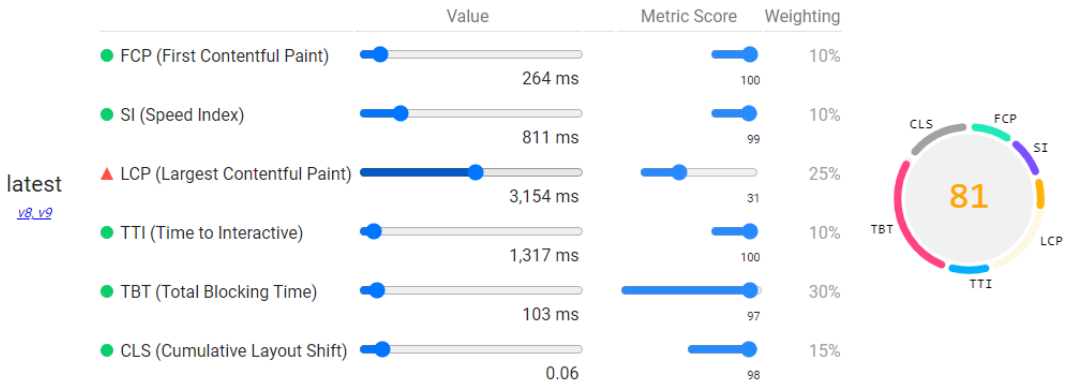
Rezultati analize izvedbe Next.js aplikacije na deset objekata su slijedeći:



Slika 16. Analiza Next.js na 10 objekata

Ukupna količina podataka koja je trebala biti prikazana je 16.5MB. U rezultatima analize vidimo da je dva kritična elementa su LCP i TTI. S obzirom na to da se u Next.jsu aplikacija izvršava na strani poslužitelja i šalje gotovi HTML dokument korisniku, ima smisla da će vrijeme potrebno da stranica postane interaktivna biti duže.

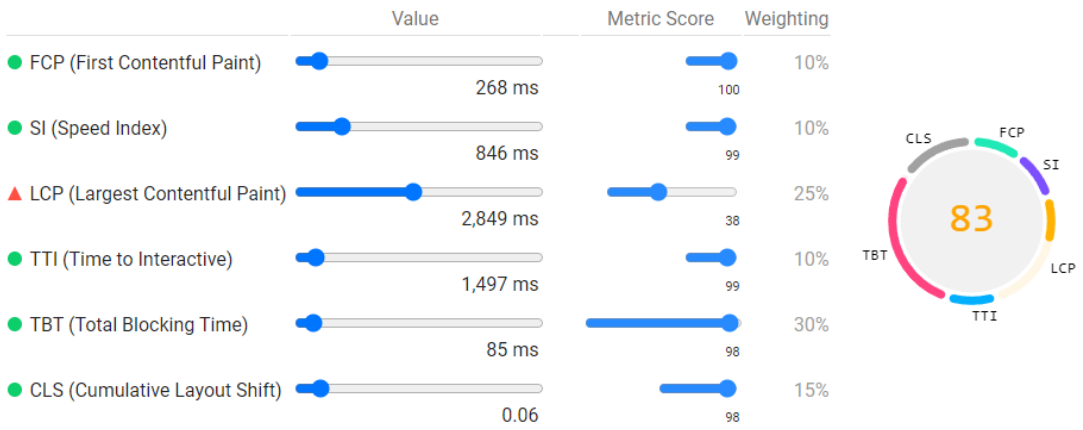
Rezultati analize izvedbe React aplikacije na deset objekata su slijedeći:



Slika 17. Rezultati analize React aplikacije na 10 objekata

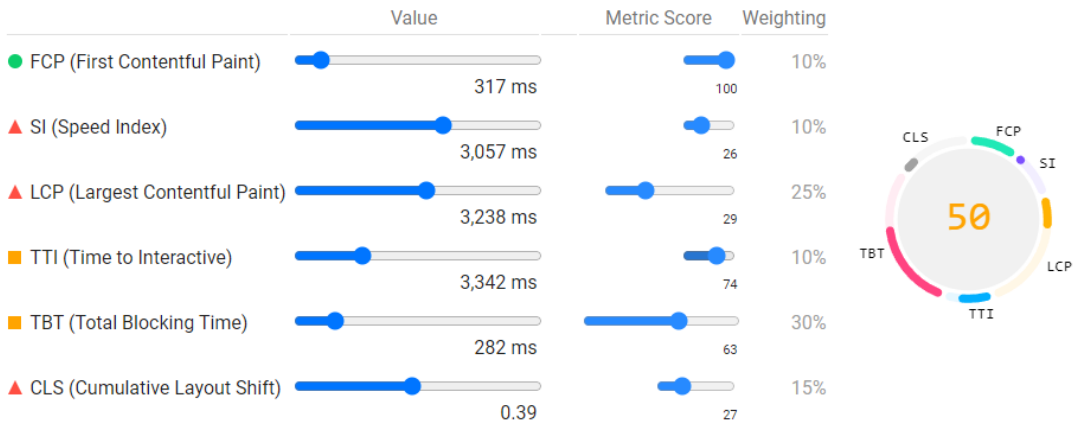
Ukupna količina podataka koja je trebala biti prikazana je 13.3MB. U rezultatima analize vidimo da TTI metrika ima znatno bolji rezultat nego kod Next.js. React dohvaća JavaScript kôd i prikazivanje stranica radi na stani korisnika u pregledniku i iz tog razloga korisnik ima mogućnost znatno prije stupiti u interakciju s stranicom. React posjeduje još jednu opciju kojom pruža dodatnu optimizaciju vremena dohvaćanja i prikazivanja podataka. React omogućava da pojedinim komponenta definiramo u kojem trenutku će se učitavati i to radimo

pomoću atributa `loading`. `Loading` možemo definirati na dva načina `lazy` i `eager`. Postavkom atributa `loading` na `lazy` odgađa se učitavanje podataka do onog trenutka kada se trebaju prikazati, `eager loading` učitava podatke odmah prilikom izvršavanja kôda. Zadana postavka atributa `loading` je `eager`.

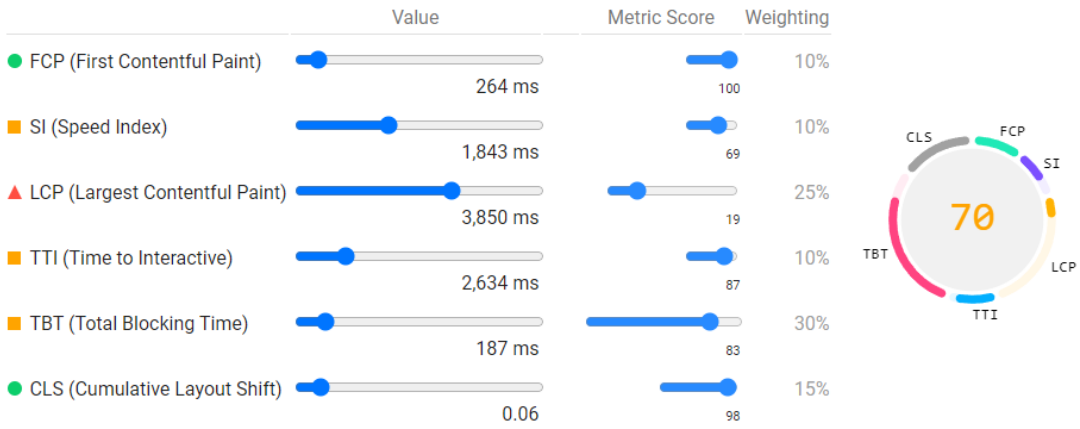


Slika 18. Rezultati analize React aplikacije na 10 objekata korištenjem `lazy loading`

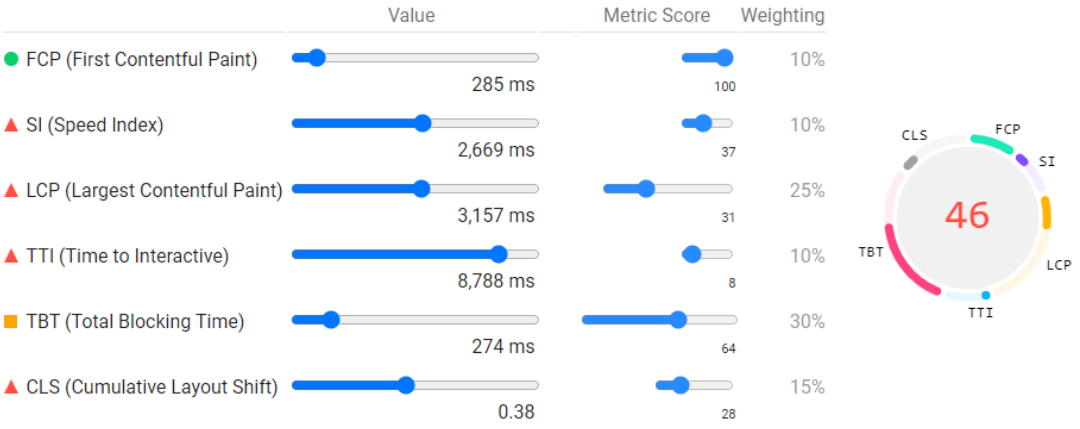
Nakon postavljanja atributa `loading` na `lazy` količina podataka koja se prilikom pokretanja aplikacije trebala prikazati je 9.2MB te se nakon toga prilikom korištenja aplikacije učitava ostatak podataka.



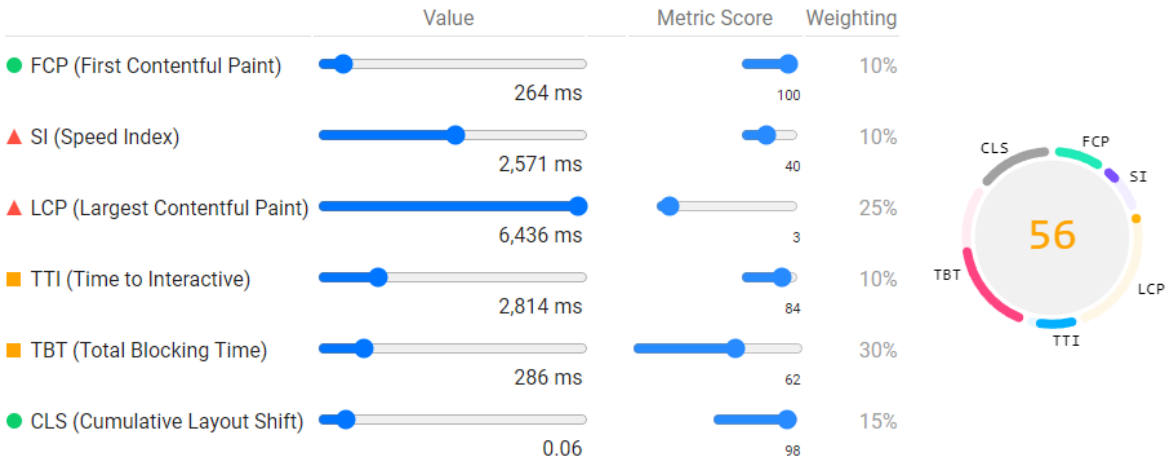
Slika 19. Rezultati analize Next.js aplikacije na 50 objekata



Slika 20. Rezultati analize React aplikacije na 50 objekata



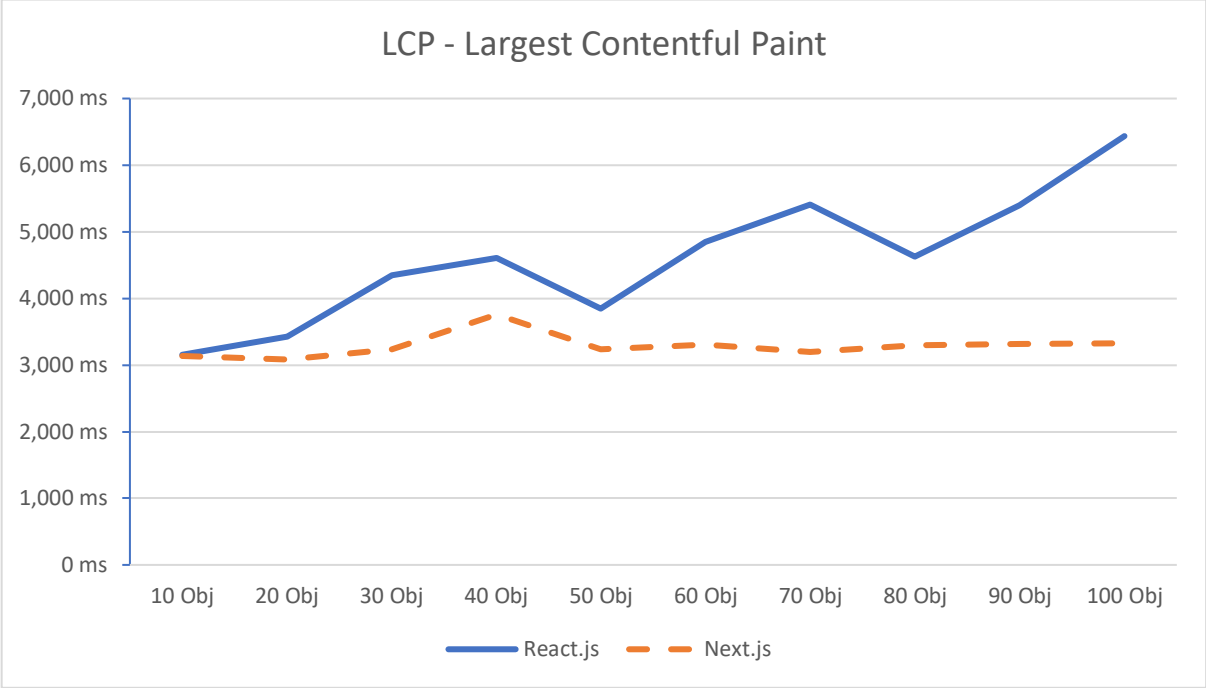
Slika 21. Rezultati analize Next.js aplikacije na 100 objekata



Slika 22. Rezultati analize React aplikacije na 100 objekata

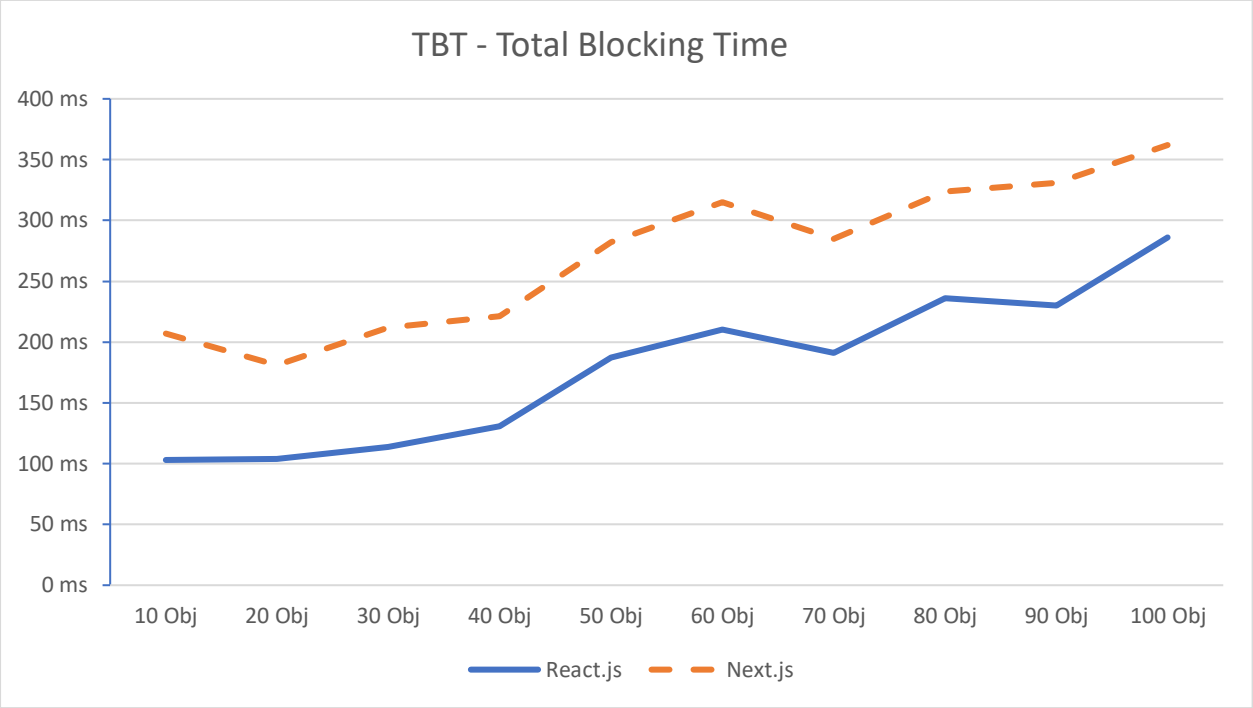
Povećanjem količine podataka koju aplikacija treba renderirati vidljivo je da se vrijeme u rezultatima metrika povećava i za React i Next.js. Nakon izvršavanja ostalih testova vidimo kako React postiže bolje rezultate na testovima za svaku navedenu veličinu podataka koje treba prikazati.

Na grafovima koji se nalaze ispod možemo vidjeti ovisnost veličine podataka koja se treba prikazati s dvije metrike LCP i TBT koje imaju naviše utjecaja prilikom izračuna ocjene. Na prvom grafu se vidi kako je Next.js konzistentan u vremenu potrebnome za prikazivanje glavnog dijela stranice dok React.js s povećanjem količine podatak zahtjeva i više vremena za dohvaćanje tog dijela stranice.



Slika 23. Usporedba LCP metrika

Na drugom grafu je prikazana metrika TBT koja prikazuje vrijeme koje je potrebno da stranica postane interaktivna na korisnikove unose. Vidljivo je da to vrijeme raste s količinom podataka ali React u svakom slučaju zahtjeva manje vremena.



Slika 24. Usporedba TBT metrika

ZAKLJUČAK

Nakon izvršenih testova u programu Lighthouse vidljivo je da biblioteka React postiže bolja vremena u većini metrika. Najbitnije metrike za korisnika su TTI i FCP u kojima React postiže dobar rezultat. React postiže ovakve rezultate zbog svog načina prikazivanja podataka. React podatke obrađuje na strani korisnika u internetskom pregledniku. React posjeduje i dodatnu mogućnost definiranja kada želi da se podaci učitavaju korištenjem atributa `loading`. Samo postavljanjem tog atributa na vrijednost `lazy`, proces koji zahtjeva dodavanje jedne linije koda, Reactu omogućava da brže učita samu stranicu i postigne bolje rezultate na testu.

Iako na prvu izgleda da je React uvjerljivo bolji, Next.js ima neke kvalitete koje nisu mjerene u ovome radu ali ih je bitno za spomenuti. Next.js renderiranje podataka odrađuje na strani poslužitelja i onda korisniku šalje gotov HTML dokument. Zbog toga prvo učitavanje stranice iziskuje duže vrijeme obrade ali ponovna učitavanja iste te stranice nakon prvog puta se događaju znatno brže jer mrežni preglednici spremaju taj HTML dokument u svoju priručnu memoriju.

Još jedna stvar koju Lighthouse provjerava ali samo na površnoj razini je SEO. Naime, Lighthouse je dodijelio 100% ocjenu u kategoriji SEO i Reactu i Next.jsu. Na prvu je to malo iznenađujuće jer cijeli postupak renderiranja podataka kod Next.js je u svrhu dobivanja gotovog HTML dokumenta kojeg će mrežni pretraživači moći pregledati i samim time pružiti kvalitetniju optimizaciju za mrežne preglednike. Ocjena koju Lighthouse pruža za ovu kategoriju je samo prividna ocjena kvalitete SEO-a stranice jer Lighthouse provjerava samo sadrži li aplikacija elemente poput `<head>` i `<meta>` ali ne i ono što piše u njima. Dakle kategorija samo provjerava postoje li osnovni elementi koji su potrebni pri optimizaciji web aplikacije za mrežne preglednike. React je u tom pogledu lošiji jer da bi mrežni pretraživači imali uvid u stanje web aplikacije moraju izvršiti sav JavaScript kôd React aplikacije kako bi dobili HTML dokument.

Dakle ako uzmemo sve to u obzir za izradu dinamičkih aplikacija kojima nije bitna vidljivost na mrežnim preglednicima već optimizacija korištenja bolji odabir bi bila biblioteka React. Ako želimo izraditi aplikaciju koja će prikazivati više statičkog sadržaja i koja zahtjeva da joj rad bude optimiziran za mrežne preglednike, Next.js kao alat za izradu web aplikacije se čini kao bolji odabir.

POPIS SLIKA

POPIS KÔDOVA

POPIS TABLICA

LITERATURA

[1] »A short history of the Web,« 2. 11. 2022.. [Mrežno]. Available: https://home.cern/science/computing/birth-web/short-history-web.

[2] »Comparison Between Web 1.0, Web 2.0 and Web 3.0,« 14. 11. 2022.. [Mrežno]. Available: https://www.geeksforgeeks.org/web-1-0-web-2-0-and-web-3-0-with-their-difference/.

[3] »ĐApps: What Web 3.0 Looks Like,« 18. 12. 2022.. [Mrežno]. Available: https://gavwood.com/dappsweb3.html.

[4] »URI-Uniform-Resource-Identifie,« 28. 11. 2022.. [Mrežno]. Available: https://www.techtarget.com/whatis/definition/URI-Uniform-Resource-Identifier.

[5] »A Brief History of HTML,« 5. 11. 2022.. [Mrežno]. Available: https://www.washington.edu/accesscomputing/webd2/student/unit1/module3/html_history.html.

[6] »CSS Selectors,« 28. 11. 2022.. [Mrežno]. Available: https://www.w3schools.com/css/css_selectors.asp.

[7] »Usage statistics of JavaScript as client-side programming language on websites,« 4. 12. 2022.. [Mrežno]. Available: https://w3techs.com/technologies/details/cp-javascript.

[8] »Website Application Architecture,« 30. 11. 2022.. [Mrežno]. Available: https://mobidev.biz/blog/web-application-architecture-types.

[9] »What is Web application?,« 18. 11. 2022.. [Mrežno]. Available: https://www.techtarget.com/searchsoftwarequality/definition/Web-application-Web-app.

[10] »Basic Features,« 12. 11. 2022.. [Mrežno]. Available: https://nextjs.org/docs/basic-features/pages#server-side-rendering.

[11] »10 Goggle Search Statistics You Need To Know,« 15. 12. 2022.. [Mrežno]. Available: https://www.oberlo.com/blog/google-search-statistics.

[12] »Lighthouse - Chrome Developers,« 13. 12. 2022.. [Mrežno]. Available: https://developer.chrome.com/docs/lighthouse/.

[13] »Google Lighthouse - SEO Glossary,« 16. 12. 2022.. [Mrežno]. Available: https://www.searchmetrics.com/glossary/google-lighthouse/.

SKRAĆENICE

WWW	World Wide Web	
HTTP	Hyper Text Transfer Protocol	
TCP	Transmission Control Protocol	
URI	Uniform Resource Identifier	Jedinstveni identifikator resursa
URL	Uniform Resource Locator	Jedinstveni lokator resursa
HTML	HyperText Markup Language	
CSS	Cascading Style Sheets	
JS	JavaScript	
API	Application programming interface	Programska sučelja za aplikacije
MPA	Multi Page Application	Višestranična Web Aplikacija
SPA	Single Page Application	Jednostranična Web Aplikacija
SSR	Server Side Rendering	Renderiranje na strani poslužitelja
CSR	Server Side Rendering	Renderiranje na strani korisnika
SEO	Search Engine Optimization	Optimizacija za mrežne pretraživače
FCP	First Contentful Paint	Prvo postavljanje sadržaja
LCP	Largest Contentful Paint	Postavljanje najvećeg sadržaja
SI	Speed Index	Oznaka brzine
TTI	Time To Interact	Vrijeme potrebno da stranica bude interaktivna
CLS	Cumulative Layout Shift	Ukupni pomak rasporeda
TBT	Total Blocking Time	Ukupno vrijeme blokiranja