Usporedba izomorfnih JavaScript radnih okvira

Lipovac, Bruno

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

Stručni studij

USPOREDBA IZOMORFNIH JAVASCRIPT RADNIH OKVIRA

Završni rad

Bruno Lipovac

Osijek, 2022.

SADRŽAJ

1. UVOD

U posljednjih je 30 godina vjerojatno najviše pozornosti posvećeno razvoju tehnologija, alata i metodologija za razvoj internetskih stranica, odnosno mrežnih aplikacija. Od jednostavnih, neuglednih i nezanimljivih stranica koje su učitavane po nekoliko minuta, danas internetske stranice uz pomoć modernih alata donose estetski primamljiv sadržaj uz vrhunsko korisničko iskustvo milijunima korisnika uz rekordne brzine učitavanja sadržaja. Zahvaljujući programskom jeziku *Javascript* internetskim stranicama dodane su brojne nove mogućnosti kao što su interaktivnost, ažuriranje sadržaja, kontrola teksta, mogućnost reagiranja na određene događaje, spremanje podataka i još mnogo toga.

Porastom popularnosti internetskih stranica *Javascript je* nametnut kao nezaobilazni programski jezik. Zbog svoje fleksibilnosti i jednostavnosti mnoge tvrtke, programerske zajednice, ali i pojedinci počeli su razvijati svoje radne okvire za *Javascript* kako bi donijeli što veći set mogućnosti programerima, skraćeno vrijeme razvoja aplikacija, povećanu sigurnost, stabilnost i skalabilnost aplikacija.

Dosadašnji *Javascript* radni okviri uvelike su korišteni za razvoj SP aplikacija (engl. *Single Page*). Kod klasične aplikacije svaka je stranica predstavljena zasebnim *HTML* (engl. *Hypertext Markup Language*) dokumentom koji bi morao biti dohvaćen s poslužitelja pri svakom pristupu stranici, dok kod SP aplikacija postoji samo jedan veliki dokument koji sadrži sav potreban *HTML*, *CSS* (engl. *Cascading Style Sheets*) i *Javascript* potreban za prikaz bilo koje stranice aplikacije. Mana ovakvog pristupa povećano je vrijeme za inicijalno učitavanje aplikacije, ali je učitavanje svake druge stranice nakon prve brže nego kao kod klasičnog pristupa. S druge strane razvijane su aplikacije kod kojih je dinamički stvaran *HTML* na poslužiteljskom računalu te je konačni *HTML* sadržaj prenesen internetskom pregledniku. Generalno, poslužiteljski renderirane aplikacije (engl. *Server Side Render*) nisu imale probleme koje su imale SP aplikacije, međutim skuplje su za održavanje zbog puno većeg broja zahtjeva prema poslužiteljskom računalu, ovise o brzini interneta krajnjeg korisnika te o broju trenutno aktivnih korisnika jer je za svaku izmjenu na stranici potrebno izvršiti mrežni zahtjev prema poslužiteljskom računalu.

Cilj izomorfnih *Javascript* aplikacija spajanje je najboljeg od oba spomenuta pristupa. Veći dio stranice renderira se na poslužiteljskom računalu te se šalje internetskom pregledniku, međutim ta stranica sadrži sve elemente nužne za ažuriranje sadržaja bez potrebe za naknadnim mrežnim zahtjevima.

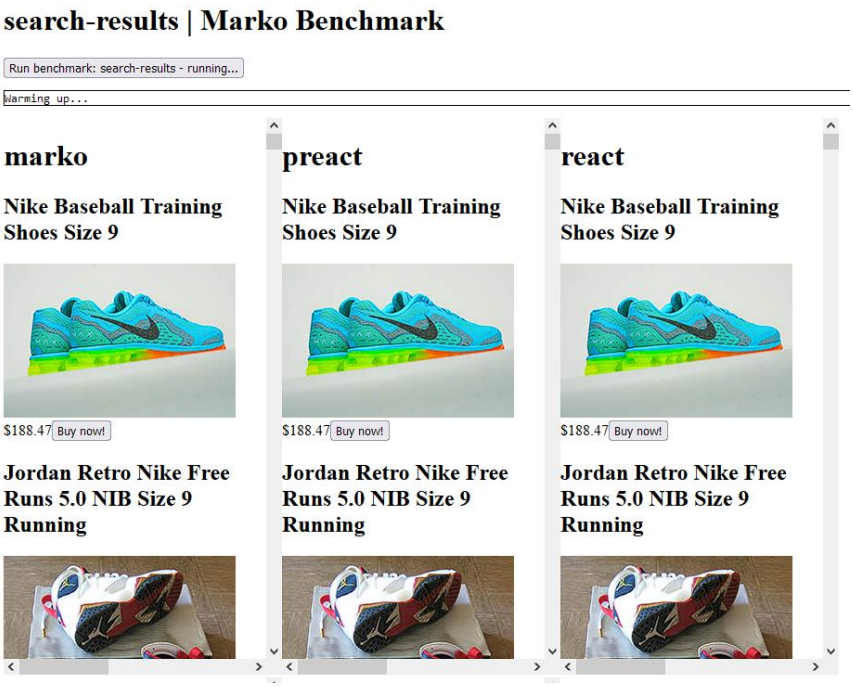
U ovom završnom radu analizirani su trenutno najpopularnijih izomorfnih *Javascript* radnih okvira: *Meteor, Nuxt, Next.* Njihove performanse uspoređene su kroz niz testova kojima je

prikazana brzina izvođenja raznih zadataka, zauzeće memorije nakon njihova izvršenja te vrijeme potrebno da stranica postane konzistentno interaktivna. Osim što su rezultati izomorfnih okvira uspoređeni međusobno, uspoređeni su i s rezultatima testova radnih okvira za SP aplikacije od kojih navedeni izomorfni okviri proizlaze (*React* i *Vue)*.

2. DOSADAŠNJI RADOVI U PREDMETNOM PODRUČJU

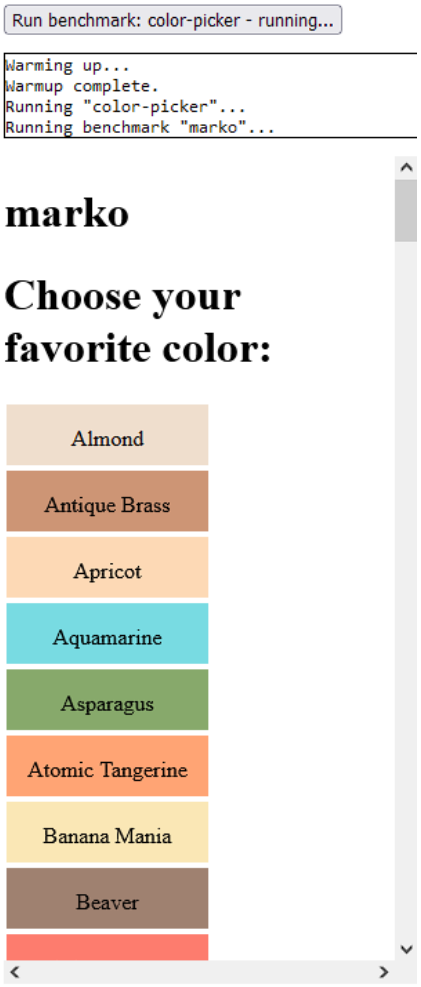
Mnogo je radova na temu usporedbi performansi radnih okvira za stvaranje SP aplikacija. Spomenuti radovi uvelike se oslanjaju na uporabu istog softvera koji će biti korišten za potrebe ovog rada.

S druge strane radovi na temu usporedbe performansi izomorfnih radnih okvira slabo su zastupljeni. Prva među rezultatima istaknuta je usporedba performansi *Marko.js* [1] radnog okvira (izomorfni radni okvir koji je razvio eBay za potrebe ekspresnog prikazivanja sadržaja na stranicama) s raznim drugim radnim okvirima za kreiranje korisničkog sučelja. Ispitivanja performansi izvršena su za sve radne okvire na poslužitelju kao i u internetskom pregledniku. Svi radni okviri ispitani su pomoću dva testa. Prvim testom ispitano je vrijeme potrebno za prikaz stotinu elemenata na stranici. Svakom iteracijom testa stotinu novih elemenata bilo bi prikazano, što zahtijeva ažuriranje znatnog broja DOM čvorova [2]. Prikaz testa moguće je vidjeti na slici 2.1.



Sl 2.1 *Testiranje brzine prikazivanja elemenata*

Drugim ispitivanjem provjerena je brzina izmjene označenog elementa. U testu je stvorena lista elemenata, kao što je vidljivo na slici 2.2, te je mjereno vrijeme potrebno za promjenu indeksa označenog elementa, micanje obruba prethodno označenog elementa te iscrtavanje obruba novo označenog elementa.



Sl 2.2 *Ispitivanje brzine izmjene označenog elementa*

Sljedeće istraživanje provedeno je u svrhu razvoja renderiranja stranica na poslužitelju za potrebe radnog okvira *Solid* [3]. Ispitivanje je provedeno od strane tvorca radnog okvira te je nekoliko metoda renderiranja na serveru ispitano u svrhu odabira najboljeg pristupa za implementaciju u radni okvir. Performanse su ispitivane koristeći alat *Timeline* dostupan u pregledniku *Google Chrome*. Na isti način softver upotrijebljen u ovom radu mjeri performanse, ali na automatiziran način.

3. OPIS KORIŠTENIH TEHNOLOGIJA

U ovom poglavlju dan je pregled korištenih tehnologija i vezanih pojmova koji će se pojavljivati u daljnjem tekstu ovoga rada.

3.1. Javascript

Javascript je programski jezik nastao 1995. godine te je internetskim stranicama omogućio dotada neviđenu i priželjkivanu interaktivnost. Učitavanje novog sadržaja bez osvježavanja stranice, pomicanja ili promjene veličine vizualnih elemenata stranice, kontrola reprodukcije multimedijskog sadržaja te validacija korisničkog unosa i slično dotad su bile nepostojeće mogućnosti. S obzirom na to da je taj programski jezik bio nevjerojatno koristan i lagan za naučiti te je praćen od strane otvorene i brzorastuće zajednice, nametnuo se kao dominantan jezik u svim sferama razvoja računalnih programa [4].

Javascript podržava nekoliko programskih paradigmi:

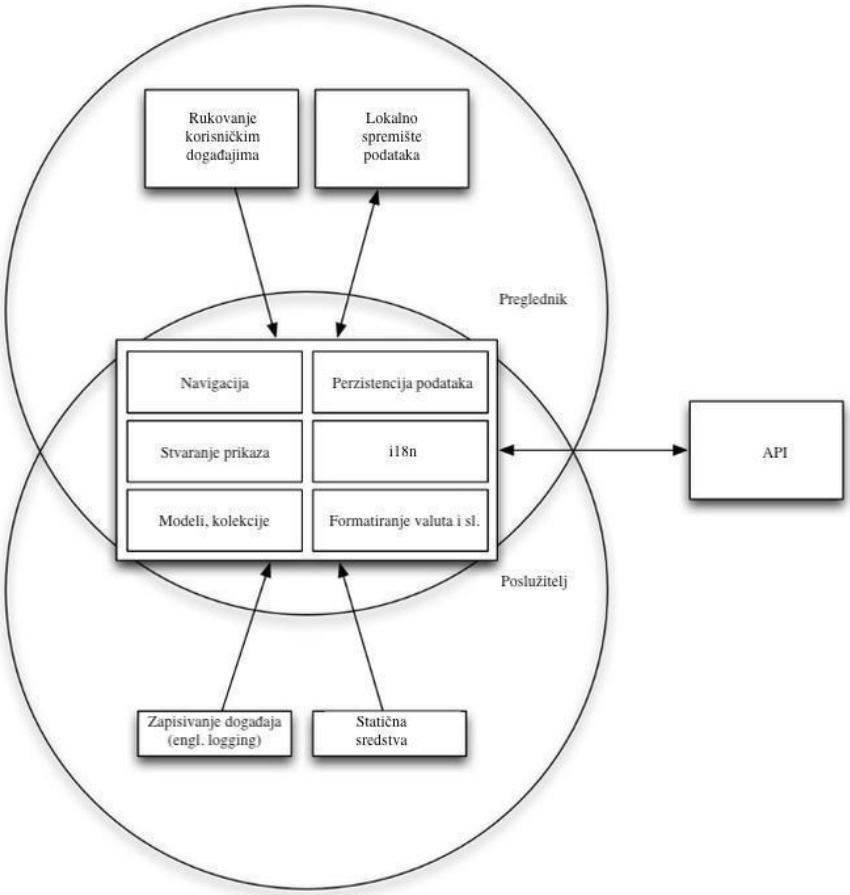
- paradigma pogonjena događajima – tijek programa određuju događaji poput klika mišem, pritiska tipke na tipkovnici, dodira ekrana i sl.
- funkcionalna paradigma – funkcije daju rezultate bez izmjena stanja programa
- imperativna paradigma – klasična paradigma kojom naredbe u nizu mijenjaju stanje programa, odnosno definiraju ponašanje programa

3.2. Node.js

Node.js programsko je okruženje kojim je omogućeno izvršavanje *Javascript* koda na računalima. Nastao je kao odgovor na *Apache HTTP Server* koji je smatran neadekvatnim nositi se s velikim brojem istovremenih konekcija na poslužiteljsko računalo (preko 10 000). *Node.js* koristi tzv. petlju događaja koja omogućava izvršavanje operacija bez blokiranja glavne programske niti, čime je omogućeno posluživanje brojnih istovremenih konekcija na poslužiteljsko računalo. Petlja događaja omogućuje da svaka operacija, čijim bi izvršenjem došlo do blokiranja programske niti (I/O operacije), bude prepuštena jezgri operacijskog sustava na izvršenje koji nakon što je operacija izvršena javlja petlji događaja da je rezultat spreman te da ga može ponovno preuzeti u svoj slijed izvršavanja [5].

3.3. Izomorfni Javascript

Izomorfnost nije pojam vezan isključivo za *Javascript*, već predstavlja način gradnje mrežne aplikacije (Slika 3.1), u kojem se internetska stranica prvotno renderira na poslužiteljskom računalu, koristeći jezike poput *PHP*-a (engl. *Hypertext Preprocessor*), *Jave, Rubyja* ili *Javascripta,* a potom se izvršavanje aplikacije nastavlja u pregledniku kao SP aplikacija koristeći *Javascript*.



Sl 3.1 *Primjer strukture izomorfne Javascript aplikacije*

Izomorfnost također opisuje onaj *Javascript* kod čije je izvršavanje omogućeno u internetskom pregledniku ili na poslužiteljskom računalu. Izomorfnim *Javascriptom* smatraju se elementi standardne *Javascript* biblioteke te biblioteke koje su napisane tako da apstrahiraju svoju funkcionalnost u jedinstveno programsko sučelje za preglednik i poslužiteljsko računalo. Primjerice, biblioteka *Axios* [6], ako je pokrenuta u pregledniku, za izvršavanje mrežnih zahtjeva koristit će klasu *XmlHttpRequest* specifičnu za DOM (engl. *Document Object Model*) programsko

sučelje, a ako je pokrenuta u poslužiteljskom okruženju Node.js, bit će iskorištena klasa *http*, što je vidljivo u ispisu koda ispod.

```
function getDefaultAdapter() {
   var adapter;
   if (typeof XMLHttpRequest !== 'undefined') {
      // For browsers use XHR adapter
      adapter = __webpack_require__(/*! ./adapters/xhr */ "./lib/adapters/xhr.js");
      }
   else if (typeof process !== 'undefined' &&
               Object.prototype.toString.call(process) === '[object process]') {
      // For node use HTTP adapter
      adapter = __webpack_require__(/*! ./adapters/http */ "./lib/adapters/xhr.js");
      }
   return adapter;
}
```

Ispis koda 3.1 Ispis koda biblioteke *Axios* za primjenu odgovarajućeg *http* adaptera

3.4. Radni okviri

Radni okvir predstavlja temelj razvoja aplikacija. Gotove funkcionalnosti dobivaju se uporabom radnih okvira, zbog čega se uvelike smanjuje vrijeme potrebno za razvoj aplikacijskih sustava. U kontekstu mrežne aplikacije radni okvir imat će spremna rješenja za navigaciju unutar aplikacije, autorizaciju korisnika, perzistenciju podataka i slično. Primjenom takvih rješenja smanjuje se vrijeme razvoja aplikacije kao i mogućnost grešaka tijekom tog procesa.

Lakša izgradnja aplikacije, u smislu korištenja objekata klasa iz naše aplikacije ili iz vanjskih biblioteka, bit će olakšana na način da će radni okvir često imati implementirane oblikovne obrasce poput ubrizgavanja ovisnosti (engl. *dependency injection*) ili kontejnere za inverziju kontrole (engl. *inversion of control container*) kako bi zahtjevnije aplikacije bilo što lakše složiti, testirati i održavati.

Funkcionalnosti u radnim okvirima implementiraju se kao oblikovni obrasci. Primjerice, razni konektori (za bazu podataka, vanjski servis itd.) bit će izvedeni kao *singleton* (klasa koje je izvedena kao *singleton* dozvoljava instanciranje samo jednog objekta te klase), autorizacija će biti izvedena kao strategija (oblikovni obrazac koji omogućuje promjene ponašanja klase za vrijeme izvođenja programa), postojat će specifične strategije za uobičajenu autorizaciju (korisničko ime i lozinka) ili za autorizaciju putem društvenih mreža itd. Životni ciklus upita i odgovora prema poslužiteljskom računalu bit će izveden kao lanac odgovornosti (oblikovni obrazac u kojem svaki komad programa odlučuje hoće li obraditi podatak ili ga proslijediti sljedećoj kariki u nizu) itd. Ukratko, radni okviri postoje za olakšani, ubrzani razvoj stabilnih aplikacija.

3.5. Izomorfni Javascript radni okviri

Krajnjem korisniku najočitija prednost uporabe izomorfne aplikacije vrijeme je potrebno da stranica postane interaktivna. Problem SP aplikacija taj je što se čitava aplikacija učita mrežnim putem prilikom prvog pristupa nekoj od stranica aplikacije. Kod izomorfnih aplikacija samo je *Javascript* kod potreban za funkcioniranje stranice kojoj je pristupljeno poslan, stoga puno brže dolaze u stanje potpune interaktivnosti.

Sljedećim primjerom opisano je zašto izomorfne aplikacije bolje kotiraju prilikom indeksiranja za internetske tražilice. Internetski pretraživač prilikom indeksiranja stranice fiktivne internetske trgovine otvora stranicu na kojoj bi trebao biti sadržan popis svih dostupnih brdskih bicikala, npr. https://trgovina-bicikala.hr/bicikli/brdski. Izomorfna aplikacija prvo prikuplja sve brdske bicikle dostupne u trgovini bilo iz podataka, bilo iz nekog *REST* (engl. *Representational State Transfer*) *API*-ja (engl. *Application Programming Interface*) ili slično na poslužiteljskom dijelu aplikacije. Tek nakon što su sve informacije prikupljene, odgovor se šalje pretraživaču sa statusnim kodom 200 te s *HTML* datotekom ispunjenom svim informacijama o dostupnim biciklima koje pretraživač prilikom indeksiranja vrlo lako "skenira". SP aplikacija prilikom posjeta spomenutoj stranici (https://trgovina-bicikala.hr/bicikli/brdski) odmah šalje odgovor sa statusnim kodom 200 te se sadržaj koji bi trebao prikazan na toj stranici naknadno stvara. Budući da zbog navedenog sadržaj nije dostupan na početnom učitavanju stranice, aplikacije dobiva loš SEO (engl. *Search Engine Optimization*) rezultat.

Slanje odgovarajućih HTTP kodova prilikom posjeta stranici još je jedna od prednosti izomorfnog pristupa izgradnji web aplikacija. Primjerice, u adresnu je traku utipkan nepostojeći URL na domeni na kojoj je poslužena izomorfna aplikacija, vrlo dobro poznati statusni kod 404 će biti vraćen te će se jednoznačno dati do znanja da je tražena stranica nepostojeća. U slučaju SP aplikacije statusni kod 200 bit će vraćen i on nam govori da je upit prema poslužiteljskom računalu prošao bez greške iako nam je vjerojatno prikazana stranica koja nam govori da sadržaj nije pronađen. Opisani slučaj vjerojatno ne znači puno uobičajenom krajnjem korisniku, međutim ima loš utjecaj na rezultat za internetske tražilice.

Pregledavanjem dostupnog materijala na internetu utvrđeno je kako je *Airbnb* bio jedan od pionira razvoja tehnologije izomorfnih *Javascript* radnih okvira. S njihove strane razvijen je radni okvir *Rendr* [7], a u to vrijeme još su bili popularni *Derby.js* [8], *Catberry* [9] it*d.* Trenutno se najviše razvijaju *Meteor* [10], *Nuxt* [11] i *Next* [12], izomorfni radni okviri o kojima će se nešto više reći u nastavku teksta. Osim što su trenutno izuzetno popularni, utemeljeni su na radnim

okvirima za izradu SP aplikacija (*React* i *Vue*). Ta će činjenica poslužiti za utvrđivanje odgovora na pitanje dolazi li do gubljenja performansi stvarajući aplikacije na izomorfni način.

2.5.1. Next.js

Next.js (u daljnjem tekstu *Next*) izomorfni je radni okvir nastao na osnovi *React*.js-a (*Javascript* biblioteka za gradnju komponenata korisničkog sučelja čija je glavna namjena kontrola stanja aplikacije i prikazivanje komponenata). Nekoliko je mogućnosti posluživanja stranica ponuđeno u *Next*-u: stvaranje stranica na poslužiteljskom računalu prilikom svakog zahtjeva internetskog preglednika za nekom stranicom, statično posluživanje stranica (stranice se stvaraju prilikom pokretanja aplikacije na poslužitelju, a potom se navigacija vrši kao kod SP aplikacije) te mogućnost potpunog izvršavanja u pregledniku (SP aplikacija).
Neke su od mogućnosti za dohvaćanje podataka s poslužiteljskog dijela aplikacije koje *Next* donosi:

- ***getStaticProps*** **-** služi za dohvaćanje podataka prilikom podizanja poslužiteljskog dijela aplikacije, obično namijenjen blog unosima, stranicama nekakvih proizvoda, odnosno stranicama koje se uglavnom ne mijenjaju
- ***getStaticPaths*** **-** služi kao podrška za *getStaticProps,* stvara listu putanja, odnosno URL-ova (engl. *Uniform Resource Locator*) za statički stvorene stranice
- ***getServerSideProps*** **-** služi za popunjavanje stranice podacima prilikom svakog zahtjeva internetskog preglednika na stranicu

2.5.2. NuxtJS

NuxtJS (u daljnjem tekstu Nuxt) radni je okvir stvoren po uzoru na *Next,* ali je zasnovan na radnom okviru *Vue.js*. Riječ je o izuzetno laganom radnom okviru s fokusom na stvaranju elemenata za prikaz aplikacije, dok naprednije značajke poput navigacije unutar aplikacije ili kontrole stanja aplikacije nude se kao dodatne biblioteke. Identične metode prikazivanja stranica kao u *Next*-u koriste se i u *Nuxt*-u, odnosno mogućnost stvaranja sadržaja na poslužitelju, u pregledniku te statično posluživanje odnosno stvaranje stranica prilikom pokretanja poslužiteljskog dijela aplikacije [13].
Metode za dohvaćanje podataka koje *Nuxt* donosi ne razlikuju se puno od onih predstavljenih u radnom okviru *Next*:

- ***nuxtServerInit*** - prvi korak u kojem se postavljaju početni podaci (za statične stranice i sl.)

- ***asnycData/fetch*** **-** poziva se prije prikazivanja stranice, dohvaća podatke kojima se popunjavaju komponente
- ***fetch*** **-** može služiti kao metoda za dohvaćanje podataka nakon učitavanja stranice te je inače metoda kojom *Vue* dohvaća podatke

2.5.3. Meteor

Za razliku od prethodna dva promatrana radna okvira svrha radnog okvira *Meteor* nešto je drugačija. Dok prethodna dva radna okvira za cilj imaju maksimizirati potencijal za izgradnju korisničkog sučelja, Meteor služi za stvaranje *full-stack* aplikacija koje koriste bazu podataka *MongoDB* za pohranjivanje informacija, postojeće radne okvire *Javascript* za izgradnju korisničkog sučelja (*Vue, React, Blaze*) te interne mehanizme za komunikaciju između preglednika i poslužiteljskog računala.

HTTP komunikacija s poslužiteljskim računalom ne koristi se kod radnog okvira *Meteor* za prijenos podataka. Umjesto toga koristi se protokol DDP (engl. *Distributed Data Protocol*), koji je zasnovan na modelu izdavača i pretplatnika. Ovim protokolom omogućeno je udaljeno pozivanje procedura (eng. *RPC, remote procedure call*) s klijenta (preglednik) na poslužitelj te je klijentu omogućeno "pretplaćivanje" na dokumente iz baze podataka *MongoDB* u svrhu ažuriranja sadržaja stranice u stvarnom vremenu. Samim time *Meteor* ne šalje kompletne *HTML*-ove putem mreže (osim *HTML*-a potrebnog za stvaranje korisničkog sučelja), nego šalje podatke u *JSON* (eng. *Javascript Object Notation*) koje pritom prikazuje klijentska aplikacija [14].

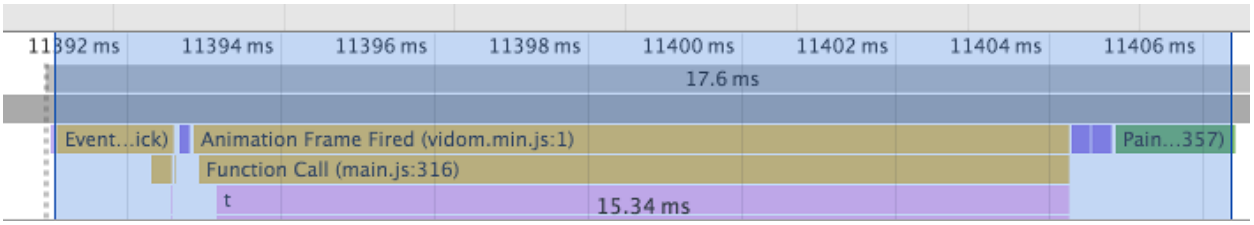
4. METODE MJERENJA PERFORMANSI RADNIH OKVIRA

Za analizu performansi koristit će se aplikacija *js-framework-benchmark*. U daljnjem tekstu koristi se naziv Mjeritelj, čija je namjena mjerenje performansi radnih okvira *Javascript*. Da bi radni okvir bio testiran, potrebno je implementirati funkcionalnosti koje će biti testirane od strane Mjeritelja. Neki od testova izvode se u više iteracija te se srednje vrijeme izvođenja tih iteracija uzima kao referentno kako bi što konzistentniji rezultati bili dobiveni. Radni okviri bit će podvrgnuti sljedećim testovima [16]:

- Vrijeme potrebno za stvaranje tablice s 1000 redaka nakon što je stranica u potpunosti učitana
- Vrijeme potrebno za zamjenu svih 1000 redaka u tablici, pet iteracija
- Vrijeme potrebno za izmjenu svakog desetog reda tablice s 10,000 redaka, pet iteracija
- Vrijeme potrebno za označavanje reda drugom bojom nakon što je kliknut, pet iteracija
- Vrijeme potrebno za zamjenu dva reda (drugog i pretposljednjeg) u tablici s 1000 redova, pet iteracija
- Vrijeme potrebno za stvaranje tablice s 10,000 redaka
- Vrijeme potrebno za brisanje 10,000 redaka iz tablice
- Zauzeće memorije nakon učitavanja stranice
- Zauzeće memorije nakon dodavanja 1,000 redaka
- Zauzeće memorije nakon što se tablica s 1,000 redaka ažurira (iznova stvori), pet iteracija
- Zauzeće memorije nakon kreiranja tablice s 1,000 redaka pet puta
- Zauzeće memorije nakon što stvorimo i obrišemo tablicu s 10,000 redaka pet puta
- Vrijeme potrebno za učitavanje koda *Javascript* i kreiranja stranice aplikacije
- Vrijeme do konzistentne interaktivnosti (nakon što mreža i procesor nisu obavljali nikakve zadatke preko 50 ms)
- Ukupnu količinu podataka prenesenih preko mreže svih resursa učitanih u stranicu

Funkcionalnosti će se testirati tako što će Mjeritelj pokrenuti testni poslužitelj, s kojeg će aplikacija izrađena u pojedinom radnom okviru biti poslužena. Internetski preglednik pokrenut će se koristeći *Selenium* kako bi se testovi za ranije spomenute funkcionalnosti izvršili u kontroliranim uvjetima, a rezultati svih testova zabilježili u interaktivnoj tablici.

Selenium je alat za automatizaciju internetskih preglednika, koji se uvelike koristi za testiranje funkcionalnosti mrežnih aplikacija. *Selenium* se koristi zbog sposobnosti bilježenja događaja iz alata *Timeline*, koji možemo vidjeti na slici ispod, internetskog preglednika *Chromium*. Dakle, performanse su dobivene mjerenjem vremena od izvršenja prve *Javascript* skripte do završetka iscrtavanja korisničkog sučelja u internetskom pregledniku.



Sl 4.1 *Timeline Google Chrome žuto - vrijeme izvršenja skripte, ljubičasto - stvaranje HTML-a, zeleno - iscrtavanje HTML-a* [15]

Prva ispitivanja performansi izvedena su na radnim okvirima *Vue* i *React* jer su dobiveni rezultati primijenjeni kao osnova za usporedbu performansi izomorfnih i klasičnih SP aplikacija. Nakon što su ispitane performanse okvira *Vue* i *React*, ispitane su performanse njihovih izomorfnih varijanti, a dobiveni su rezultati uspoređeni.

Preduvjet za korištenje Mjeritelja instalirano je *Java* izvršno okruženje koje je potrebno za pokretanje automatiziranih *Selenium* testova u internetskom pregledniku te izvršno okruženje *Node.js* čija je namjena pokretanje poslužitelja pomoću kojeg će internetske aplikacije izrađene u pojedinim radnim okvirima *Javascript* biti poslužene internetskom pregledniku. Osim za pokretanje poslužitelja *Node.js* koristi se za zapisivanje rezultata ispitivanja u datoteke te za oblikovanje zapisanih rezultata u interaktivnu tablicu. Ako su spomenuti sistemski preduvjeti ispunjeni, aplikaciju kloniramo s repozitorija na *GitHub*-u (projekt je otvorenog tipa te programerska zajednica pridonosi implementaciji svojih omiljenih radnih okvira te, uz glavnog autora, pridonosi razvoju kompletnog sustava za ispitivanje radnih okvira *Javascript*).

Upotreba mjeritelja započinje instaliranjem biblioteka potrebnih za rad poslužiteljske aplikacije čime je omogućen pristup ispitivanim stranicama.

```
npm ci          // instalira sve biblioteke potrebne za pokretanja Mjeritelja
npm start       // pokreće poslužitelja koji će nam posluživati aplikacija za testiranje
```

Ispis koda 4.1 Pokretanje poslužitelja ugrađenog u Mjeritelja

Npm (engl. *Node Package Manager*) naredbeni je alat čija je namjena preuzimanje te instaliranje biblioteka *Node.js*. *Npm* je i naziv repozitorija odakle se instalirane biblioteke *Node.js* preuzimaju prilikom instaliranja.

Nakon pokretanja poslužitelja stvoreni su preduvjeti za pokretanje jedne od aplikacija. Prije pokretanja aplikacije potrebno je instalirati sve biblioteke nužne za ispitivanje aplikacije. Potom se aplikacija pokreće izvršenjem naredbe specifične za radni okvir. Navedeni koraci vidljivi su u ispisu koda ispod.

```
cd frameworks/keyed/vue      // uđemo u mapu gdje se nalazi aplikacija koju ćemo testirati
npm ci                       // instaliramo biblioteke potrebne za rad aplikacije
npm run build-prod           // pokrećemo aplikaciju u produkcijskom načinu
```

Ispis koda 4.2 Pokretanje aplikacije testiranje od strane mjeritelja

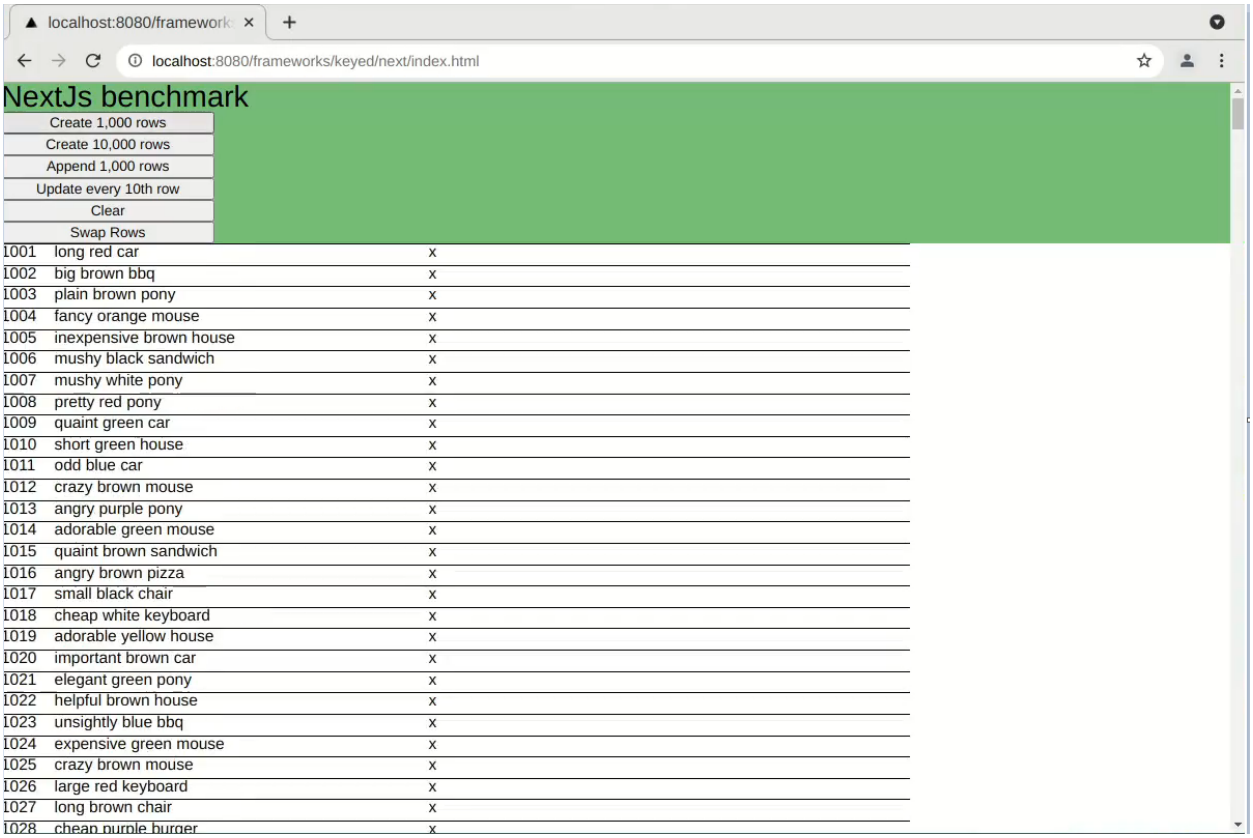
Nakon pokretanja aplikacije moguće joj je pristupiti putem internetskog preglednika ako je u adresnu traku upisano npr. *http://localhost:8080/frameworks/keyed/vue/*. Otvaranjem stranice i prikazom sadržaja svi prijašnji koraci uspješno su izvedeni.

Zadnji je korak izgradnja pogonitelja testova. Pogonitelj se gradi pozicioniranjem u mapu u kojoj se nalazi izvorni kod pogonitelja, pokretanjem naredbe za instaliranje potrebnih biblioteka za rad te pokretanjem naredbe za prevođenje aplikacije.

```
cd ../../..                  // vraćamo se u izvornu mapu sustava
cd webdriver-ts              // ulazimo u mapu pogonitelja testova
npm run compile              // pokrećemo prevođenje pogonitelja
npm run bench keyed/vue      // pokrećemo testiranje performansi aplikacije
```

Ispis koda 4.3 Pokretanje testova nad aplikacijom

Aplikacija testira stranicu otvaranjem internetskog preglednika *Chromium* te pristupa adresi na kojoj je poslužena testirana aplikacija (u ovom slučaju *http://localhost:8080/frameworks/keyed/vue/*) te automatizirano klika po gumbima. Ti gumbi na događaj klika pozivaju funkcije (implementacijski kod nalazi se u P.3.1), koje odrađuju posao definiran u scenariju testiranja, te interno snima rezultate. Nakon završetka svih ispitivanja dobiveni su rezultati snimljeni u datoteku *JSON*, u koju su pohranjeni rezultati svih izvršenih ispitivanja. Rad Mjeritelja moguće je vidjeti na slici 4.2 te u prilogu P.3.2.



Sl 4.2 *Snimka ekrana za vrijeme izvođenja testova radnog okvira*

Nakon prvih izvršenih ispitivanja nad radnim okvirima *React* i *Vue*, ubrzo su se pojavile poteškoće zato što je aplikacija za mjerenje performansi radnih okvira namijenjena aplikacijama koje se u cijelosti izvode u internetskom pregledniku.

Mjeritelj stvara vlastiti poslužitelj pomoću kojeg poslužuje stranice, čime zauzima port 8080 na računalu, odnosno onemogućuje pokretanje poslužiteljske komponente izomorfnih aplikacija. S obzirom na to da je poslužitelj zaseban proces Mjeritelja, nepokretanjem tog procesa zaobilazi se ovaj problem te je omogućeno pokretanje poslužiteljskih procesa, koji su sastavni dio izomorfnih aplikacija, na portu 8080.

Svaka od izomorfnih aplikacija svoju poslužiteljsku komponentu pokreće na unaprijed definiranom portu, obično portu 3000. Izmjena porta na kojem poslužitelj očekuje mrežne zahtjeve na aplikacijama *Nuxt* i *Next* izvršena je dodavanjem argumenta naredbi za pokretanje aplikacije u produkcijskom modu kao što se može vidjeti u ispisu koda 4.4.

```
"scripts": {
  "dev": "next dev",
  "build": "next build",
  "start": "next start --port 8080"
  …
```

```
}
```

Ispis koda 4.4 Pokretanje izomorfne aplikacije na željenom portu izmjenom datoteke *package.json*

Kod *Meteora* situacija je nešto drugačija jer ta aplikacija ima potpuno drugačiji način pokretanja u produkcijskom modu te se oslanja na dodavanje vrijednosti varijablama programskog okruženja.

```
meteor build --directory ../meteor-react-benchmark-prod // izgradimo aplikaciju
cd ..
cd meteor-react-benchmark-prod/                    // pozicioniramo se u mapu
export ROOT_URL="http://localhost"                 // dodamo varijable okruženja
export PORT="8080"
node main.js                                       // pokrećemo aplikaciju Meteor
```

Ispis koda 4.5 Pokretanje aplikacije *Meteor* na željenom portu dodavanjem sistemskih varijabli

Osim posluživanja aplikacije na portu 8080 aplikaciju je potrebno poslužiti na točno definiranoj adresi, primjerice *http://localhost:8080/frameworks/keyed/nuxt/index.html*. Navedeno je ostvareno postavljanjem datoteke *.vue* ili *.jsx* unutar strukture datoteka koja je jednaka adresi stranice, npr. */pages/frameworks/keyed/nuxt/index.html/index.vue*.

Radni okviri obično ne koriste datoteke *HTML*, nego pripadajuće predložene sintakse napisane u datotekama *.vue* ili *.jsx* koje su kasnije prevedene u *HTML*. Kada je takvim stranicama pristupljeno internetskim preglednikom, one nemaju *.html* ekstenziju u adresnoj traci te ih Mjeritelj „ne vidi". Ovaj je problem lako riješen tako što je direktoriju u kojem je sadržan naš kod dodano „*html"* u naziv, npr. *vue-aplikacija/index.html/index.vue (index.html* je ime direktorija).

Za vrijeme pisanja rada verzija preglednika *Chromium* automatski je ažurirana s verzije 93 na verziju 95. Podržana verzija od strane Mjeritelja bila je verzija 93 te je pokretanje testovabilo nemoguće, no promjenom verzije biblioteke *chromedriver*, korištene za pokretanje *Chromiuma* unutar Mjeritelja, na odgovarajuću verziju problem je riješen.

Uspješnim savladavanjem problema stvoreni su uvjeti za početak testiranja radnih okvira. Prije pokretanja bilo koje od aplikacija osigurano je da na računalu na kojem se vrše ispitivanja nisu pokrenuti nikakvi korisnički procesi osim terminala za pokretanje aplikacije i terminala za pokretanje testova. Preglednik *Chromium* pokretan je automatski prilikom pokretanja svakog individualnog testa. Ispitivanja su izvršena na računalu s procesorom AMD Ryzen 3 1200, 16GB DDR4 RAM-a 3200 Mhz, diskom Gigabyte M2.2280 NVMe, grafičkom karticom AMD RX 570 (hardversko ubrzanje na pregledniku bilo je omogućeno). Kada su svi rezultati ispitivanja

prikupljeni, složeni su u zasebne tablice prema testovima za brzinu izvođenja, zauzeća memorije te vremena potrebnog do potpune interaktivnosti. Najbolji rezultati svakog pojedinog testa, neovisno o tome koji ih je radni okvir ostvario, složeni su u pomoćni stupac koji nam je služio za usporedbu svih rezultata ostvarenih ispitivanjima. Primjer dobivenih rezultata vidljiv je u tablici priloženoj niže.

Testovi brzine izvođenja Test/Naziv Radnog Okvira	***Vue***	***React***	***Meteor-Vue***	***Meteor-React***	***Nuxt***	***Next***	**Idealni Rezultat**
Stvaranje 1000 redaka (ms)	105,30	126,90	196,70	170,70	189,10	194,10	105,30
Zamjena svih redaka (1000 redaka, pet uvodnih iteracija) (ms)	98,60	113,00	167,90	149,90	159,00	169,20	98,60
Djelomično ažuriranje (ažuriranje svakog desetog retka od 1000 redaka, tri uvodne iteracije) (ms)	202,30	224,90	365,90	294,60	365,60	342,10	202,30
Odabir retka (isticanje retka drugom bojom, 16x usporenje CPU-a) (ms)	29,40	112,30	357,50	133,30	411,30	171,40	29,40
Zamjena redaka (zamjena dva retka u tablici s 1000 redaka, pet uvodnih iteracija, 4x usporenje CPU-a) (ms)	51,00	354,90	82,20	529,70		535,20	51,00
Brisanje retka (pet uvodnih iteracija) (ms)	22,30	23,50	35,00	26,60	32,30	28,60	22,30
Stvaranje 10,000 redaka (ms)	950,10	1.360,50	1.745,70	1.845,60	1.624,60	2.004,00	950,10
Dodavanje 1000 redaka tablici od 10,000 redaka (2x usporenje CPU-a) (ms)	200,10	263,10	402,80	369,20	392,40	401,30	200,10
Brisanje 1000 redaka (8x usporenje CPU-a) (ms)	120,10	134,30	143,70	136,00	146,90	146,20	120,10

Tablica 4.1 Rezultati ispitivanja uz pomoćni stupac s idealnim rezultatima

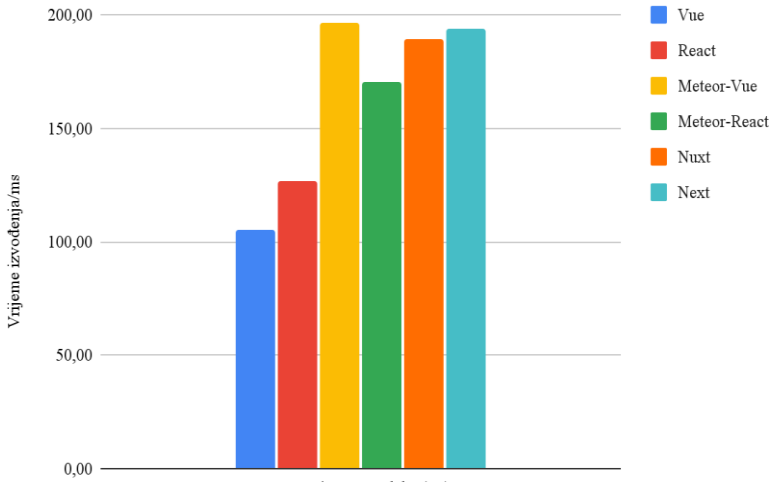
Izračunata je geometrijska sredina tog idealnog rezultata, a u retku naziva geometrijska sredina prikazano je koliko rezultati svakog pojedinog radnog okvira, odnosno stupca, odudaraju od idealnog rezultata. Što je rezultat bliži vrijednosti 1.00, ispitani radni okvir ostvaruje bolje rezultate.

5. REZULTATI MJERENJA

5.1. Rezultati mjerenja brzine izvođenja

5.1.1. Ispitivanje brzine kreiranja tablice od 1000 redaka

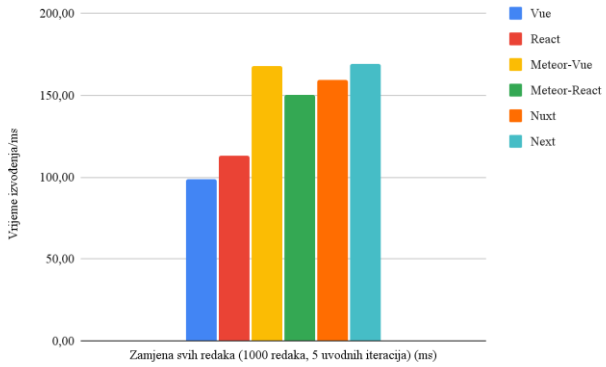
Test brzine kreiranja 1000 redaka pokazao je malu razliku između radnih okvira *Vue* i *React*, a gotovo dvostruko sporiji rezultati ostvareni su kod izomorfnih radnih okvira. Međutim, razlika između najbržeg i najsporijeg radnog okvira iznosi svega 100 ms (Sl. 5.1).



Sl 5.1 *Rezultati ispitivanja vremena izvođenja kreiranja 1000 redaka*

5.1.2. Ispitivanje brzine zamjene svih redaka u tablici

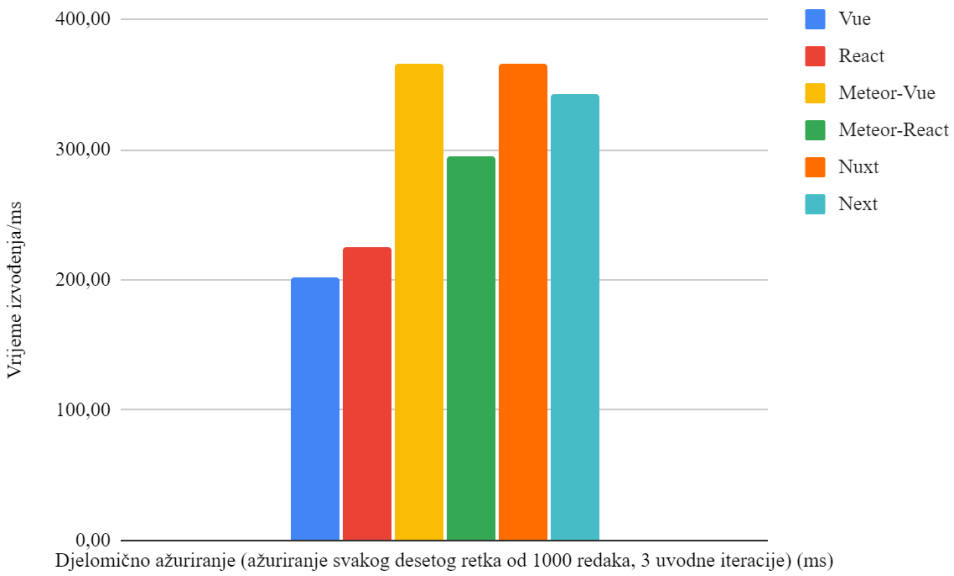
Zamjena svih 1000, odnosno test uklanjanja svih 1000 redaka te stvaranja novih 1000 redaka, obavljena je otprilike jednako brzo kao i stvaranje 1000 redaka kod radnih okvira *React* i *Vue*. Primijećeno je nešto niže vrijeme izvođenja kod izomorfnih radnih okvira naspram rezultata ispitivanja kreiranja redaka gdje je bilo potrebno oko 200 ms za izvršenje (Sl.5.1) .



Sl 5.2 *Rezultati brzine izvođenja zamjene svih redaka tablice*

5.1.3. Ispitivanje brzine ažuriranje svakog desetog retka tablice

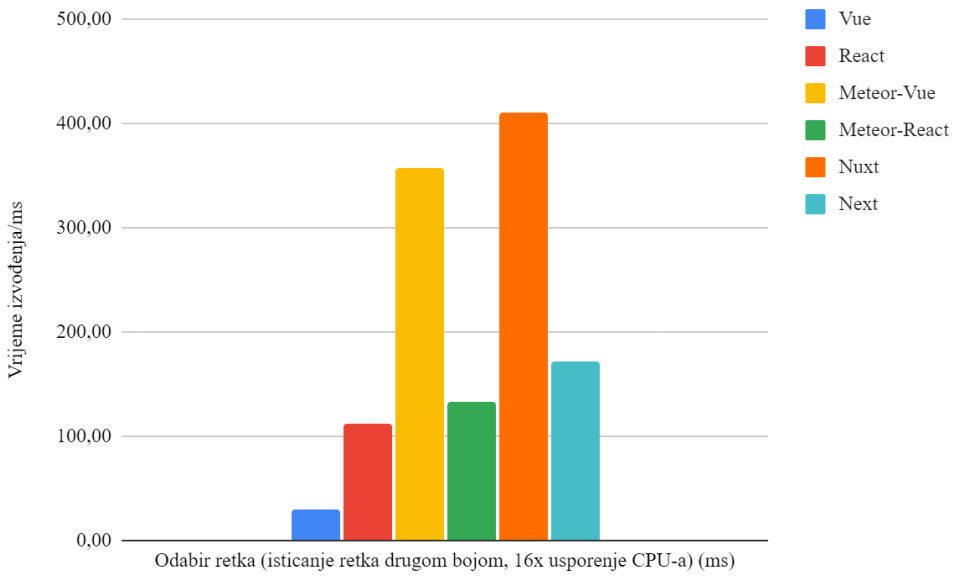
Kod ažuriranja svakog desetog retka ponovno je zabilježena mala razlika rezultata ispitivanja radnih okvira *Vue* i *React* (Sl.5.3). Kod izomorfnih radnih okvira zadatak je izvršen u vremenu oko 350 ms, odnosno 300 ms kod radnog okvira *Meteor* u kombinaciji s *Reactom*.



Sl 5.3 *Rezultati brzine izvođenja izmjene svakog desetog retka u tablici s 1000 redaka*

5.1.4. Ispitivanje brzine označavanja jednog retka u tablici

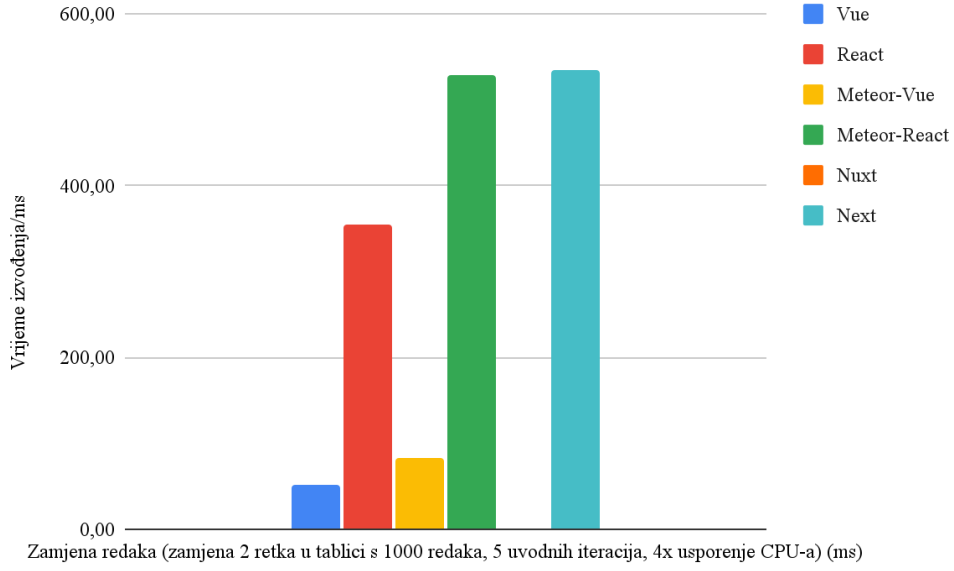
Prilikom ispitivanja brzine označavanja jednog retka drugom bojom kod radnog okvira *Vue* zabilježeni su daleko najbrži rezultati (Sl. 5.4). *Meteor* u kombinaciji s *Reactom* gotovo je jednako brz kao obični *React*, što je podudarno s činjenicom da *Meteor* samo iskorištava postojeće radne okvire za stvaranje korisničkog sučelja, a funkcionalnosti koje donosi više se odnose na komunikaciju tog sučelja i poslužitelja. Međutim, izrazito spori rezultati zabilježeni su kod radnih okvira *Meteor-Vue* i *Nuxt* usprkos generalnom očekivanju da će radni okviri utemeljeni na *Vue* radnom okviru inherentno biti brži od onih utemeljenih na *Reactu*.



Sl 5.4 *Brzina izvođenja isticanja retka drugom bojom u tablici s 1000 redata*

5.1.5. Ispitivanje brzine zamjene dvaju redaka u tablici

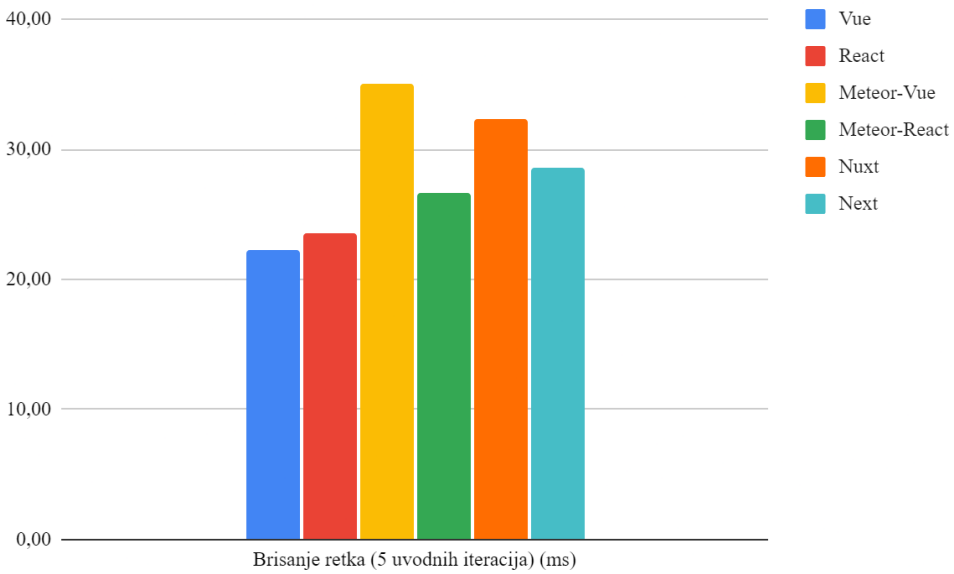
Ispitivanje zamjene dvaju redaka pokazalo je da postoji jako velika razlika u brzini izvođenja zadataka između radnih okvira utemeljenih na radnom okviru *Vue*, odnosno *React* (Sl.5.5). Pri ispitivanju zamjene dvaju redaka test na radnom okviru *Nuxt* bilo je nemoguće uspješno izvršiti te grešku nije bilo moguće pronaći i ukloniti.



Sl 5.5 *Rezultati ispitivanja brzine izvođenja zamjena dvaju redaka u tablici s 1000 redata*

5.1.6. Ispitivanje brzine izvođenja brisanja retka u tablici

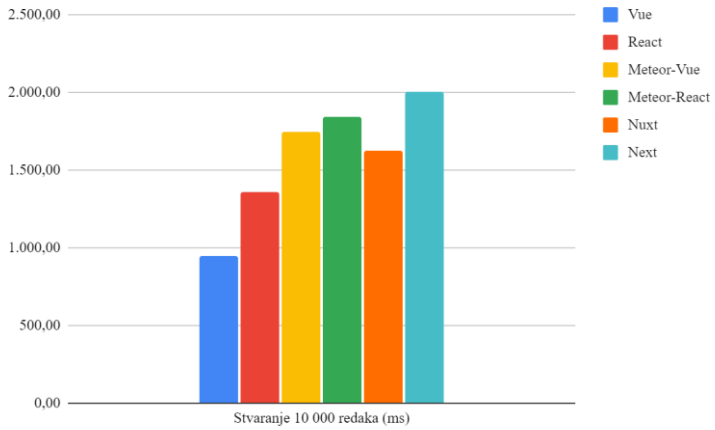
Ispitivanje brzine brisanja jednog retka u tablici pokazalo je neznatne razlike u brzini izvođenja te je zadatak obavljen u vremenskom okviru od 20 do 35 ms (Sl.5.6).



Sl 5.6 *Rezultati brzine izvođenja brisanja jednog retka*

5.1.7. Ispitivanje brzine kreiranja tablice od 10 000 redaka

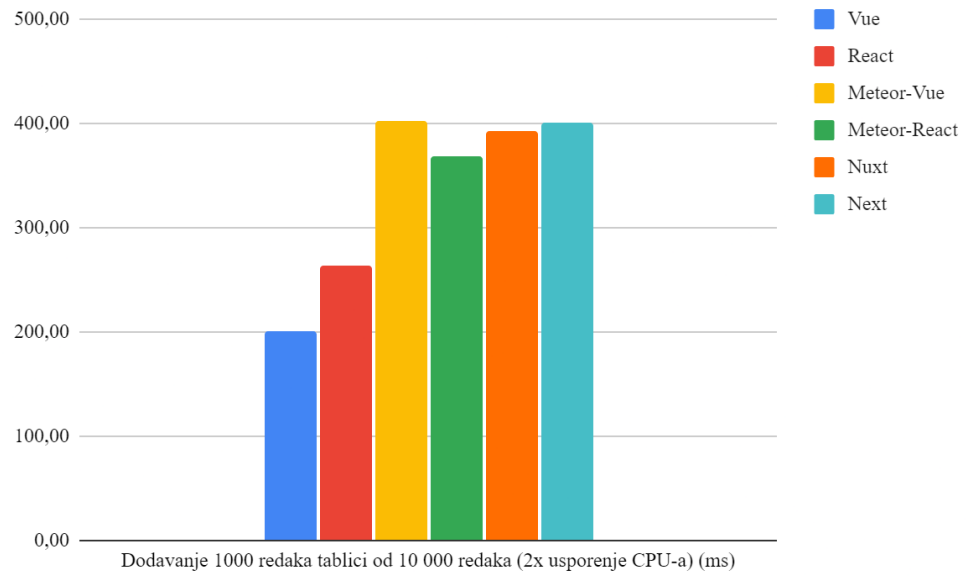
Vrijeme izvođenja kreiranja 10 000 redaka izvršeno je u trajanju od otprilike 10 puta više nego kreiranje 1000 redaka, što je očekivano (Sl.5.7). Međutim kod nekih radnih okvira bolji su rezultati ostvareni kod kreiranja 10 000 redaka nego 1000 redaka, odnosno bolji je rezultat ostvaren kod radnih okvira *Nuxt* i *Meteor-Vue* nego kod radnog okvira *Meteor-React*, što nije bio slučaj prilikom ispitivanja kreiranja 1000 redaka.



Sl 5.7 *Rezultati brzine izvođenja kreiranja 10 000 redaka*

5.1.8. Ispitivanje brzine dodavanja 1000 redaka na tablicu od 10 000 redaka

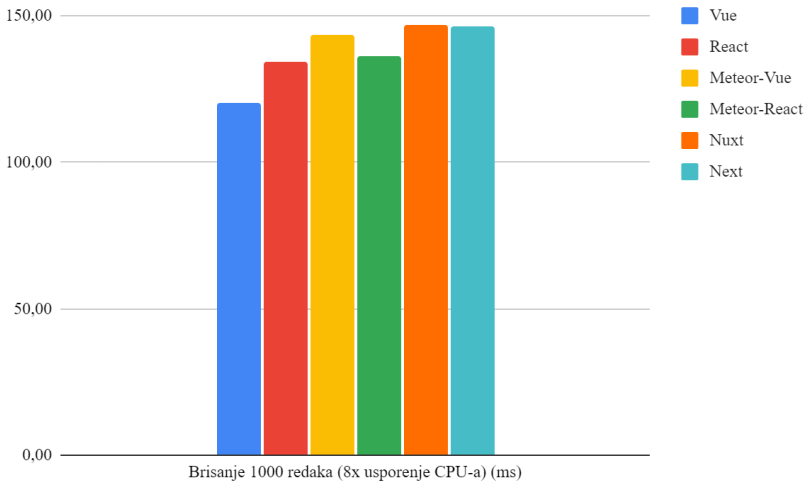
Ispitivanjem dodavanja dodatnih 1000 redaka na tablicu od 10 000 redaka pokazalo se da radni okviri imaju donekle jednaku raspodjelu rezultata kao i kod kreiranja 1000 redaka (Sl. 5.8), međutim svi dobiveni rezultati veći su za stotinjak ms, što je izravna posljedica postojanja 10,000 redaka u tablici.



Sl 5.8 *Rezultati brzine izvođenja dodavanja 1000 redaka u tablicu s 10000 redaka*

5.1.9. Ispitivanje brzine brisanja 1000 redaka iz tablice

Najmanje razlike rezultata ostvarene su pri ispitivanju brisanja 1000 redaka iz tablice te je izvršen u vremenu manjem od 50 ms od strane svih radnih okvira (Sl. 5.9).



Sl 5.9 *Rezultati ispitivanja brzine brisanja 1000 redaka*

5.1.10. Pregled rezultata ispitivanja brzine izvođenja

Rezultatima mjerenja brzine izvođenja (Tablica 5.1) prikazani su donekle očekivanu raspodjelu rezultata. Ispitni zadaci brže su izvršeni kod radnih okvira za izradu SP aplikacija. Općenito gledano, bolje performanse ostvarene su kod radnih okvira temeljenih na *Vueu* nego kod radnih okvira temeljenih na *Reactu*, dok su implementacije na *Meteoru* brže nego na *Nuxtu* i *Nextu*.

Testovi brzine izvođenja Test/Naziv Radnog Okvira	**Vue**	**React**	**Meteor-Vue**	**Meteor-React**	**Nuxt**	**Next**
Stvaranje 1000 redaka (ms)	105,30	126,90	196,70	170,70	189,10	194,10
Zamjena svih redaka (1000 redaka, pet iteracija) (ms)	98,60	113,00	167,90	149,90	159,00	169,20
Djelomično ažuriranje (ažuriranje svakog desetog retka od 1000 redaka) (ms)	202,30	224,90	365,90	294,60	365,60	342,10
Odabir retka (isticanje retka drugom bojom) (ms)	29,40	112,30	357,50	133,30	411,30	171,40
Zamjena redaka (zamjena dva retka u tablici s 1000 redaka, pet iteracija) (ms)	51,00	354,90	82,20	529,70	N/A	535,20
Brisanje retka (pet iteracija) (ms)	22,30	23,50	35,00	26,60	32,30	28,60
Stvaranje 10,000 redaka (ms)	950,10	1.360,50	1.745,70	1.845,60	1.624,60	2.004,00
Dodavanje 1000 redaka tablici od 10,000 redaka (ms)	200,10	263,10	402,80	369,20	392,40	401,30
Brisanje 1000 redaka (ms)	120,10	134,30	143,70	136,00	146,90	146,20
Geometrijska sredina	1,00	1,65	2,10	2,11	2,34	2,35

Tablica 5.1 *Rezultati ispitivanja vremena izvođenja zadataka*

5.2. Rezultati ispitivanja reaktivnosti

Ispitivanje reaktivnosti (Tablica 5.2) pokazalo je da aplikacije izrađene koristeći izomorfne radne okvire dođu u stanje potpune reaktivnosti u kraćem roku nego što je potrebno aplikacijama

izrađenim koristeći *Vue* i *React*. Stranica se smatra u potpunosti reaktivnom kada se prikazuje iskoristiv sadržaj, kada su rukovatelji događajima (engl. *event handlers*) registrirani na većini vidljivih elemenata na stranici te kada stranica reagira na korisničku interakciju unutar 50 ms.

Također, iz rezultata je vidljivo da je znatno manja količina podataka prenesena preko mreže, što je još jedna od prednosti specifičnih za izomorfne radne okvire.

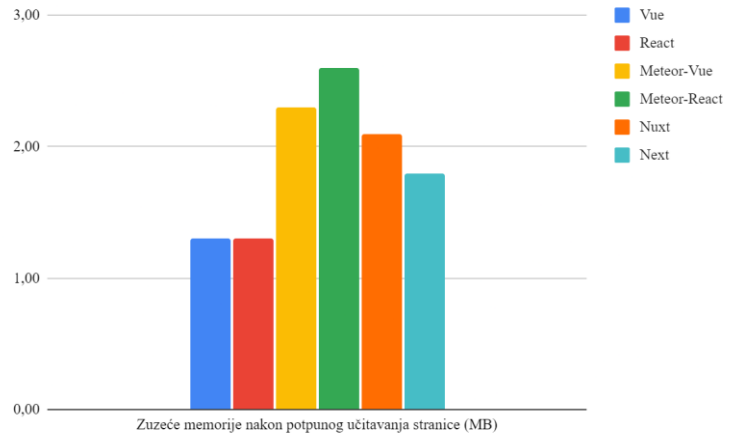
Reaktivnost Test/Naziv Radnog Okvira	**Vue**	**React**	**Meteor-Vue**	**Meteor-React**	**Nuxt**	**Next**
Konzistentna interaktivnost (pesimistično vrijeme potrebno do potpune interaktivnosti kada su CPU i mreža u stanju mirovanja) (ms)	2.116,20	2.580,50	1.758,90	2.031,20	1.568,20	1.816,20
Ukupna veličina stranice (količina učitanih podataka u preglednik putem mreže koji su potrebni za prikazivanje stranice) (kB)	194,40	274,20	101,60	134,80	88,40	73,00
Geometrijska sredina	1,90	2,49	1,25	1,55	1,10	1,08

Tablica 5.2 *Rezultati ispitivanja reaktivnosti*

5.3. Ispitivanje korištenja memorije

5.3.1. Ispitivanje zauzeća memorije nakon potpunog učitavanja stranice

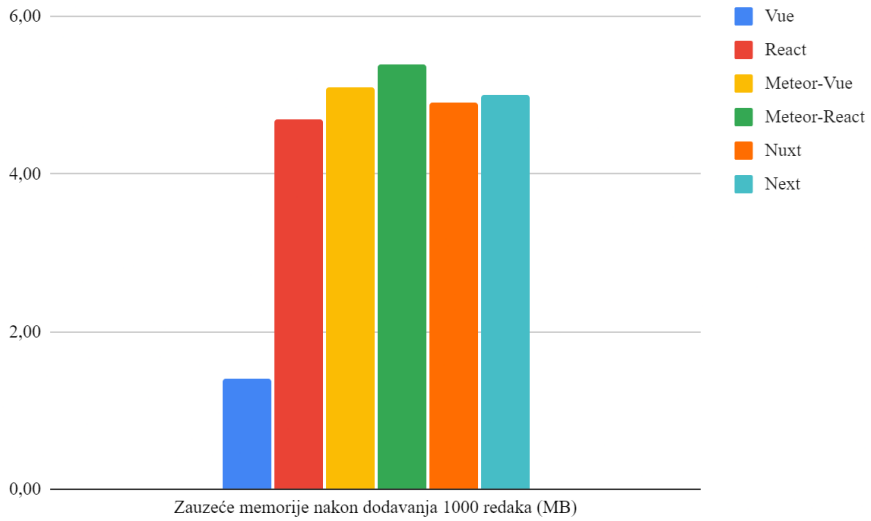
Otprilike jednaka količine memorije zauzeta je aplikacijama *Vue* i *React*, nešto više od 1 MB (Sl. 5.10). Zanimljivo je da je veća količina memorije zauzeta izomorfnom aplikacijom iako je manja količina podataka potrebnih za učitavanje aplikacije u preglednik prenesena preko mreže.



Sl 5.10 *Rezultati ispitivanja zauzeća memorije nakon učitavanja stranice*

5.3.2. Ispitivanje zauzeća memorije nakon dodavanja 1000 redaka u tablicu

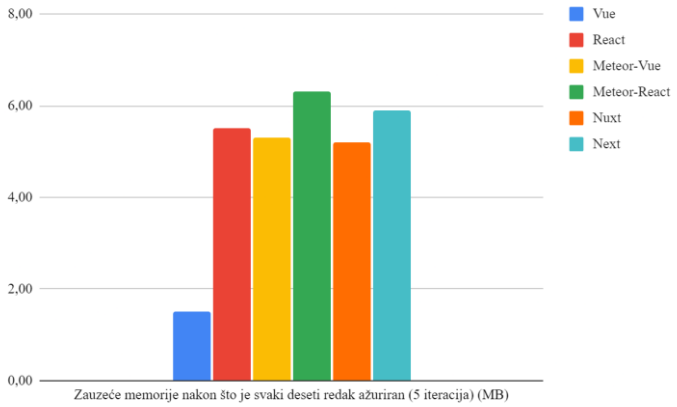
Gotovo jednaka količina memorije zauzeta je nakon dodavanja 1000 redaka te iznosi više od 4 MB za sve aplikacije osim one izrađene u radnom okviru *Vue*, gdje je zauzeće memorije ispod 2 MB (Sl. 5.11).



Sl 5.11 *Rezultati ispitivanja zauzeća memorije nakon dodavanja 1000 redaka u tablicu*

5.3.3. Ispitivanje zauzeća memorije nakon ažuriranja svakog desetog retka

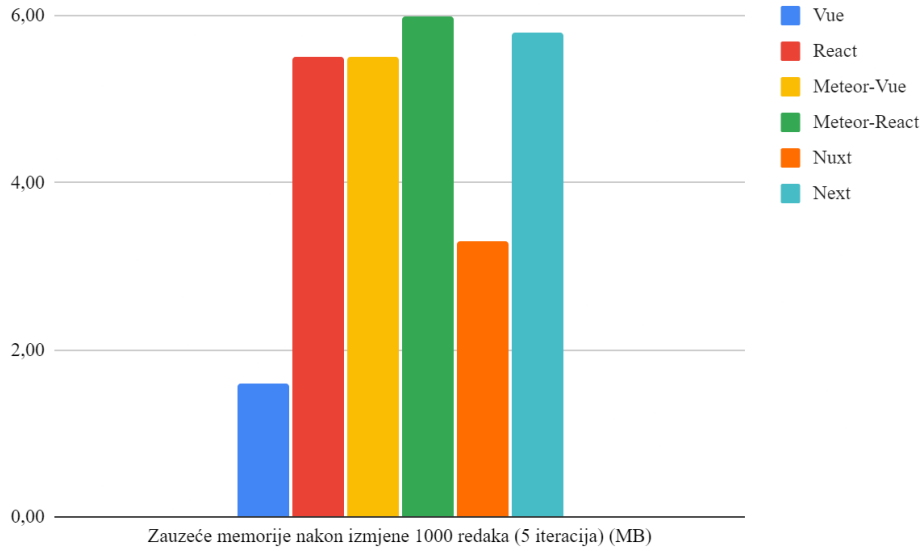
Nakon ažuriranja svakog desetog retka stanje zauzeća memorije ostaje nepromijenjeno naspram zauzeća memorije poslije kreiranja 1000 redaka (Sl 5.12).



Sl 5.12 *Zauzeće memorije nakon ažuriranja svakog desetog retka*

5.3.4. Ispitivanja zauzeća memorije nakon zamjene 1000 redaka u tablici

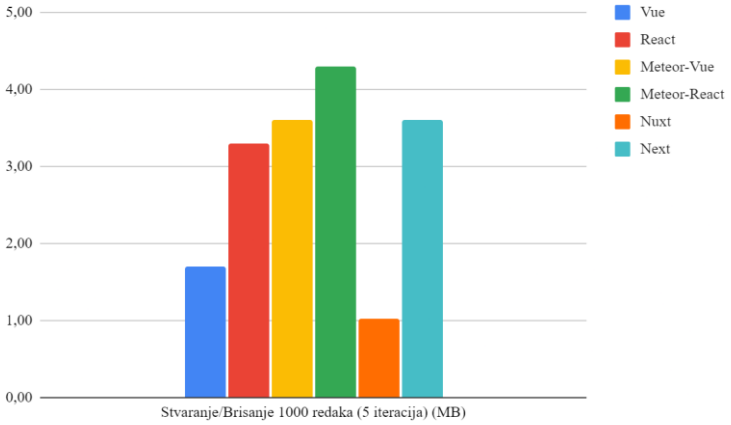
Nakon ispitivanja zamjene 1000 redaka zauzeće memorije gotovo je nepromijenjeno. Jedina je iznimka zauzeće memorije aplikacije izrađene koristeći radni okvir *Nuxt*, čije je zauzeće memorije smanjeno sa 6 MB na ispod 3 MB (Sl 5.12).



Sl 5.12 *Rezultati ispitivanja zauzeća memorije nakon izmjene 1000 redaka*

5.3.5. Ispitivanje zauzeća memorije nakon brisanja i kreiranja 1000 redaka u tablici

Poslije brisanja i kreiranja novih 1000 redaka aplikacija izrađena u radnom okviru *Nuxt* zauzima najmanje memorije, čak manje i od aplikacije *Vue* kod koje su dosad ostvarivani najbolji rezultati u gotovo svim ispitivanjima (Sl 5.13).



Sl 5.13 *Rezultati ispitivanja zauzeća memorije nakon brisanja i kreiranja 1000 redaka u tablici*

5.3.6. Pregled rezultata ispitivanja korištenja memorije

Ispitivanjem zauzeća memorije (tablica 5.3) ostvareni su neočekivani rezultati. Uzevši u obzir rezultate ispitivanja reaktivnosti, izomorfne aplikacije prenesu manju količinu podataka putem mreže kako bi učitale stranicu u internetski preglednik. Međutim izomorfne aplikacije zauzimaju veću količinu memorije za vrijeme rada.

Memorijska alokacija Test/Naziv Radnog Okvira	**Vue**	**React**	**Meteor-Vue**	**Meteor-React**	**Nuxt**	**Next**
Zauzeće memorije nakon potpunog učitavanja stranice (MB)	1,30	1,30	2,30	2,60	2,10	1,80
Zauzeće memorije nakon dodavanja 1000 redaka (MB)	1,40	4,70	5,10	5,40	4,90	5,00
Zauzeće memorije nakon što je svaki deseti redak ažuriran (pet iteracija) (MB)	1,50	5,50	5,30	6,30	5,20	5,90
Zauzeće memorije nakon izmjene 1000 redaka (pet iteracija) (MB)	1,60	5,50	5,50	6,00	3,30	5,80
Stvaranje/brisanje 1000 redaka (pet iteracija) (MB)	1,70	3,30	3,60	4,30	1,03	3,60
Geometrijska Sredina	1,11	2,67	3,07	3,48	2,10	3,01

Tablica 5.3 *Rezultati ispitivanja zauzeća memorije*

6. ZAKLJUČAK

U ovom radu objašnjen je princip rada izomorfnih aplikacija te je objašnjeno što podrazumijeva pojam izomorfnosti u kontekstu čitavog aplikacijskog sustava te programskog jezika *Javascript*. Pojašnjene su tehnologije kojima je omogućen razvoj izomorfnih aplikacija te je pobliže predstavljen koncept radnih okvira. Donesen je pregled radnih okvira *Javascript* koji omogućavaju razvoj izomorfnih aplikacija te su detaljnije opisani radni okviri koji će biti ispitivani u ovom radu uz poseban naglasak na metodama dohvaćanja podataka. Opisan je rad sustava koji će biti korišten za provođenje ispitivanja te su pojašnjene metodologije koje sustav primjenjuje kako bi performanse radnih okvira *Javascript* bile ispitane. Izrađene su jednostavne testne aplikacije koristeći radne okvire *React* i *Next*, odnosno izomorfnu inačicu *Reacta*, te *Meteor* koristeći *React* kao radni okvir za izradu korisničkog sučelja. Testne aplikacije izrađene su koristeći radni okvir *Vue* te njegovu izomorfnu inačicu *Nuxt*, kao i *Meteor* uz korištenje radnog okvira *Vue* za izradu korisničkog sučelja. Izvršena su ispitivanja nad svim navedenim radnim okvirima te su rezultati prikupljeni, prikazani te su donesena opažanja. Ispitivanjima je dokazano da aplikacije izrađene pomoću izomorfnih radnih okvira imaju nešto niže performanse nego aplikacije izrađene koristeći SP radne okvire u gotovo svim ispitivanim slučajevima.

LITERATURA

[1] M. Rawlings, et al. “Marko.js Readme”. Github.com. https://github.com/marko-js/marko (pristupljeno: 31. ožujak 2022.)

[2] M. Rawlings, et al. “Isomorphic ui benchmarks”. Github.com. https://github.com/marko-js/isomorphic-ui-benchmarks (pristupljeno: 31. ožujak 2022.)

[3] R. Carniato. „Solid JS“. https://www.solidjs.com (pristupljeno 02. travanj 2022.).

[4] T.J. DeGroat. “The History of JavaScript: Everything You Need to Know“. Springboard.com https://www.springboard.com/blog/data-science/history-of-javascript/ (pristupljeno 22. studeni 2021.)

[5] Open Js Foundation. „The Node.js Event Loop, Timers, and process.nextTick()“. Nodejs.org. https://nodejs.org/en/docs/guides/event-loop-timers-and-nexttick/ (pristupljeno 22. studeni 2021.).

[6] M. Zabriskie. ”Axios source code”. Github.com. https://github.com/axios/axios/blob/6b4fd93e6886c281ef1a51fca556616ce17f8fba/dist/axios.js (Pristupljeno: 16. studeni 2021)

[7] Rendr. “Rendr repozitorij”. Github.com. https://github.com/rendrjs/rendr (Pristupljeno: 06. travanj 2022)

[8] DerbyJS. “Derby repozitorij” Github.com. https://github.com/derbyjs/derby. (Pristupljeno: 06. travanj 2022)

[9] Catberry.js. “Carberry repozitorij” Github.com. https://github.com/catberry/catberry (06. travanj 2022)

[10] Meteor. “Meteor.js repozitorij”. Github.com. https://github.com/meteor/meteor. (Pristupljeno: 06. travanj 2022.)

[11] Nuxt. “NuxtJS repozitorij”. Github.com. https://github.com/nuxt/nuxt.js. (Pristupljeno: 06. travanj 2022.)

[12] Vercel. “Next.js repozitorij” Github.com. https://github.com/vercel/next.js

[13] Nuxt. „Rendering“. Nuxtjs.org. https://nuxtjs.org/docs/features/rendering-modes (pristupljeno 30. studeni 2021.).

[14] Meteor „Publications and Data Loading | Meteor Guide“. Meteor.com http://guide.meteor.com (pristupljeno 01. prosinac 2021.).

[15] S. Krause. „Benchmarking JS-Frontend Frameworks“. Stefankrause.net https://www.stefankrause.net/wp/?p=218 (pristupljeno 05. prosinac 2021.).

[16] S. Krause. „A comparison of the performance of a few popular javascript frameworks“. Github.com. https://github.com/krausest/js-framework-benchmark (pristupljeno 05. prosinac 2021.).

SAŽETAK

U završnom radu objašnjeno je načelo rada izomorfnih aplikacija te su pojašnjene tehnologije kojima je omogućen njihov razvoj. Donesen je pregled radnih okvira *Javascript* koju su ispitani u ovom radu. Opisane su metodologije kao i rad sustava korištenog za ispitivanje performansi radnih okvira. Izvršena su ispitivanja radnih okvira *Nuxt, Next, Vue, React* i *Meteor*. Rezultati ispitivanja su zabilježeni i prikazani te su donesena zapažanja. Ispitivanjima je dokazano da izomorfne aplikacije imaju nešto niže performanse nego SP aplikacije.

Ključne riječi: Javascript, radni okvir, *Vue*, *React*, *Next*, *Nuxt*, *Meteor*, testiranje, performanse, izomorfni, univerzalni

COMPARISON OF ISOMORPHIC JAVASCRIPT FRAMEWORK

ABSTRACT

In this paper the working principles of isomorphic applications are described. Technologies which enable development of isomorphic applications are outlined. A brief observation of *Javascript* frameworks tested in this paper is made. Methodologies and tools employed to test the performance of frameworks are described. Tests are made on *Nuxt, Next, Vue, React* and *Meteor* frameworks. The test results were recorded, displayed and observations have been made. The tests have shown that there was a performance penalty observed in isomorphic applications.

Keywords: Javascript, framework, *Vue*, *React*, *Next*, *Nuxt*, *Meteor*, benchmarks, performance, isomorphic, universal

ŽIVOTOPIS

Bruno Lipovac rođen je 10. 11. 1990. u Slavonskom Brodu. Pohađao je Osnovnu školu Ivan Goran Kovačić, gdje je sudjelovao u izvannastavnim aktivnostima iz informatike i robotike. Nakon osnovne škole upisao je Tehničku školu Slavonski Brod, smjer tehničar mehatronike. Za vrijeme srednjoškolskog obrazovanja sudjelovao je na natjecanjima iz engleskog jezika. Godine 2009. upisao je Filozofski fakultet u Osijeku, studijski smjer Engleski jezik i književnost i Hrvatski jezik i književnost te se ispisao s fakulteta 2012. godine. Od 2013. do 2016. radio je uglavnom u službama za korisničku podršku, a 2016. je godine upisao Fakultet elektrotehnike, računarstva i informacijskih tehnologija, stručni studij informatike. Na trećoj godini studija, putem student-servisa, započeo je s radom u tvrtki Gauss Development. Nakon završetka akademske godine 2019. prekinuo je studij kako bi se posvetio poslu. U tvrtki Gauss Development radio je do 2021. godine, kada je prešao u tvrtku Barrage, a iste je godine upisao završetak započetog studija.