SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 21

USPOREDBA JAVASCRIPTOVIH KNJIŽNICA REACT, VUE I ANGULAR ZA RAZVOJ NA STRANI KLIJENTA

Fran Kristijan Jelenčić

Zagreb, lipanj 2023.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 21

USPOREDBA JAVASCRIPTOVIH KNJIŽNICA REACT, VUE I ANGULAR ZA RAZVOJ NA STRANI KLIJENTA

Fran Kristijan Jelenčić

Zagreb, lipanj 2023.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Zagreb, 10. ožujka 2023.

DIPLOMSKI ZADATAK br. 21

Pristupnik:	**Fran Kristijan Jelenčić (0036518901)**
Studij:	Računarstvo
Profil:	Programsko inženjerstvo i informacijski sustavi
Mentor:	izv. prof. dr. sc. Alan Jović

Zadatak: **Usporedba JavaScriptovih knjižnica React, Vue i Angular za razvoj na strani klijenta**

Opis zadatka:

JavaScript je programski jezik koji je jedna od temeljnih tehnologija World Wide Weba zajedno s HTML-om i CSS-om. JavaScript se smatra jednim od najvažnijih programskih jezika za razvoj na strani klijenta. Međutim, rad na velikim projektima može biti izazovan, a izgradnja aplikacije od nule može biti dugotrajna i zahtjevna. Stoga mnogi programeri koriste popularne JavaScriptove knjižnice i radne okvire kako bi ubrzali proces razvoja i poboljšali kvalitetu koda. Tri najpopularnije JavaScriptove knjižnice su React, Vue i Angular te sve tri knjižnice omogućavaju izgradnju brzih, dinamičnih i modernih web stranica, međutim svaka od njih ima svoje prednosti i mane. U ovom diplomskom radu potrebno je provesti komparativnu kvalitativnu i kvantitativnu analizu JavaScriptovih knjižnica React, Vue i Angular. Za kvantitativnu usporedbu potrebno je razviti programski sustav koji ispituje zajedničke značajke web aplikacije s istim parametrima. Dobivene rezultate potrebno je komentirati i objasniti. Programski sustav treba ispitivati brzinu i učinkovitost manipulacije knjižnica s DOM-om (engl. Document Object Model) ispitne web stranice, potrošnju radne memorije prilikom rada te potrošnju procesorskog vremena. Uz razvoj programskog sustava za ispitivanje potrebno je istražiti i objasniti pouzdanost i sigurnost knjižnica te kako jedinstvene značajke svake knjižnice utječu na njene performanse i lakoću korištenja. Dobivene rezultate potrebno je usporediti s rezultatima srodnih istraživanja provedenih na ovim knjižnicama.

Rok za predaju rada: 23. lipnja 2023.

Sadržaj

Uvod

U svijetu web razvoja JavaScriptove knjižnice su postale nezamjenjivi dio alata koji programerima omogućuju razvoj kompleksnih aplikacija s mnogo značajki. Među najpopularnijim i najutjecajnijim su knjižnice React, Angular i Vue od kojih svaka ima svoje jedinstvene značajke i mogućnosti. Kako potražnja za brzim i efikasnim web stranicama raste postaje vrlo važno poznavanje performansi i odabira alata s kojima se one razvijaju. Cilj ovog rada je prezentirati rezultate kvalitativne i kvantitativne analize spomenutih knjižnica te razvoj alata koji bi omogućio evaluaciju istih na objektivan način uzimajući u obzir njihove performanse, skalabilnost, lakoću održavanja i podršku zajednice. Kvalitativna analiza detaljno će opisati glavne značajke, dokumentaciju, lakoću učenja, korištenja i podršku zajednice. Kvantitativna analiza bavit će se rezultatima testova performansi koji će obuhvatiti različite scenarije i opterećenja. Glavne metrike bit će korištenje memorije, procesorskog vremena te sveukupna efikasnost. Rezultati kvantitativne analize će omogućiti objektivnu usporedbu knjižnica na temelju empirijskih podataka. U sljedećem poglavlju opisat ću rezultate kvalitativne analize te prezentirati svaku od knjižnica u više detalja.

1. Kvalitativna analiza

Ovo poglavlje fokusira se na kvalitativnu analizu knjižnica React, Angular i Vue. Kvalitativna analiza metoda je istraživanja koja se koristi za razumijevanje i interpretaciju složenih pojava koristeći nenumeričke podatke s naglaskom na subjektivno razumijevanje i istraživanje društvenih i kulturnih konteksta. Podaci prikupljeni za analizu knjižnica temelje se na iskustvu rada s knjižnicama, raznim člancima u kojima autori opisuju svoja iskustva rada s knjižnicama te službenoj dokumentaciji.

1.1. Knjižnica React

React.js, popularno zvan React je JavaScriptova knjižnica otvorenog koda koja se koristi za razvoj korisničkih sučelja web aplikacija. Razvijena je od strane Meta Platforms, Inc. koji je prije bio poznat kao Facebook, Inc te zajednice individualnih razvijatelja i tvrtki. Neke od ključnih značajki Reacta su: arhitektura temeljena na komponentama, virtualni DOM, sintaksna nadogradnja JSX, jednosmjerni protok podataka te takozvani React Native koji omogućuje razvoj nativnih aplikacija za platforme iOS i Android.

1.1.1. Dokumentacija i podrška zajednice

React je jedna od najpopularnijih knjižnica koje se koriste za razvoj web stranica. O njegovoj popularnosti svjedoči činjenica da je gotovo 5% pitanja na web sjedištu Stack Overflow vezano baš za razvoj koristeći React [2]. Opširna službena dokumentacija koja uključuje primjere i upute korisne početnicima i iskusnijim programerima razlog je zašto je React prva knjižnica koja se spominje prilikom planiranja tehnologija za web projekt. Dokumentacija je dobro strukturirana, lako se njome navigira i redovito je ažurirana kako bi reflektirala najnovije značajke Reacta. Zbog činjenice da React ima veliki broj korisnika, podrška zajednice je vrlo velika, postoje konferencije na kojima razvijatelji u Reactu imaju priliku dijeljenja iskustva i informacija o najnovijim trendovima i najboljim praksama u ekosustavu Reacta.

1.1.2. Arhitektura temeljena na komponentama

React koristi arhitekturu temeljenu na komponentama što znači da se elementi korisničkog sučelja dijele na manje, ponovno uporabljive cjeline tj. komponente. Te se komponente zatim mogu kombinirati kako bi se dobila složenije korisnička sučelja. U komponentama razlikujemo stanje (engl. *state*) i svojstva (engl. *props*). Stanje predstavlja interne podatke komponente, modelira se kao JavaScriptov objekt koji sadrži podatke koji se mogu mijenjati tijekom vremena, svaka komponenta može imati svoje stanje te svaka komponenta brine samo o svojem stanju. Glavna svrha stanja je spremanje i praćenje podataka koji uzrokuju promjene u ponašanju i izgledu komponente, kada se dogodi promjena stanja React automatski osvježi prikaz komponente kako bi se konstantno prikazivalo trenutno stanje komponente. Svojstva se koriste kako bi omogućila komponenti roditelju prijenos podataka u komponentu djeteta. Svojstva se ne mogu mijenjati unutar komponente djeteta, samo se mogu čitati.

Preporučeni proces kreiranja web aplikacije u Reactu je [1]:

- Stvoriti hijerarhiju komponenata korisničkog sučelja
- Izgraditi statičnu verziju (bez stanja)
- Identificirati minimalnu, ali potpunu reprezentaciju stanja korisničkog sučelja
- Identificirati gdje se stanje treba nalaziti
    - Za svaki dio stanja u aplikaciji:
        - Identificirati svaku komponentu koja prikazuje podatke temeljene na tom stanju
        - Pronaći zajedničkog roditelja (komponenta u hijerarhiji iznad svih drugih komponenata koje koriste to stanje)
        - Taj zajednički roditelj ili neka komponenta iznad zajedničkog roditelja u hijerarhiji treba biti vlasnik tog stanja
        - Ako se ne može pronaći komponenta za koju ima smisla da bude roditelj stanja potrebno je kreirati novu komponentu koja će biti vlasnik tog stanja i dodati ju u hijerarhiju iznad zajedničkog roditelja
- Dodati inverzni tok podataka

- Prilikom prikazivanja komponenti inicijalno se stanje i svojstva šalju prema dnu hijerarhije
- Inverzni tok podataka odnosi se na tok podataka iz komponenata koje se nalaze unutar hijerarhije i na njihovo ponašanje prilikom interakcije s korisnikom (na pritisnuti gumb, unos teksta, itd.)

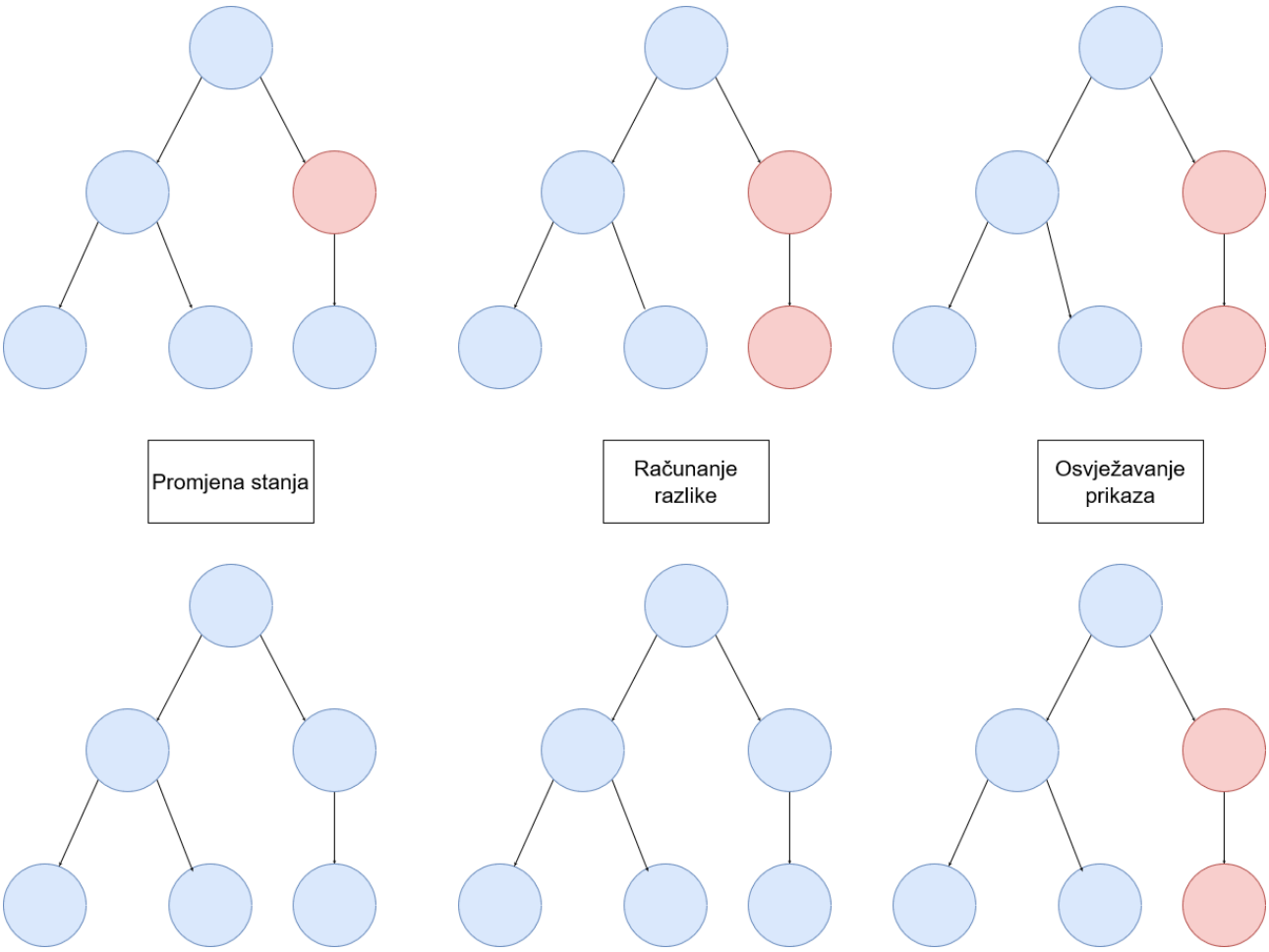
1.1.3. Virtualni DOM

DOM ili Document Object Model predstavlja strukturu web stranice u obliku stablaste strukture, gdje je svaki element, atribut i čvor objekt sa svojstvima i metodama. DOM pruža programerima mogućnost dinamičke manipulacije sadržajem, strukturom i stilovima web stranice. Kada se web stranica otvori u web pregledniku, DOM parsira HTML te kreira DOM-stablo kojim se zatim može upravljati kako bi se postigla interaktivna korisnička sučelja, dohvaćanje i ažuriranje podataka i drugo. DOM predstavlja standardizirani model te se njime može upravljati koristeći različite programske jezike poput Jave, Pythona ili JavaScripta.

React koristi virtualni DOM kako bi optimirao performanse i omogućio što brže i efikasnije ažuriranje pravog DOM-a. Virtualni DOM je koncept u kojem se kreira replika stvarnog DOM-a u obliku objekata te se sprema u preglednikovoj memoriji, ali izravno se ne mijenja ono što se prikazuje korisniku. Postoji zabluda da je virtualni DOM brži ili da je identičan stvarnom DOM, međutim to nije istina. Funkcija virtualnog DOM-a je da nadogradi funkcije i mogućnosti stvarnog DOM-a. U suštini virtualni DOM pruža mehanizam koji omogućuje računanje minimalnog broj operacija nad DOM-om potrebnih za osvježavanje prikaza korisničkog sučelja. Kod korištenja stvarnog DOM-a kada dođe do promjene elementa, DOM će ponovo osvježiti prikaz tog elementa te sve njegove djece, a u slučaju kada se radi o kompleksnijim web stranicama u kojima postoji mnogo interaktivnih elemenata ovakav pristup je vrlo neefikasan i spor. U Reactovom pristupu s virtualnim DOM-om, kada je potrebno napraviti promjenu korisničkog sučelja, prvo se promijeni virtualni DOM nakon čega React usporedi promijenjeni virtualni DOM sa snimkom DOM-a napravljenom prije promjene te odredi koji element je potrebno osvježiti i nakon toga samo taj element u stvarnom DOM-u osvježi. Zahvaljujući tome programeri

se ne trebaju brinuti o promjenama stanja, kada se dogodi promjena stanja React osigurava da DOM reflektira tu promjenu.

Srž strategije virtualnog DOM-a su dvije faze: faza iscrtavanja (eng. *rendering phase*) i faza usklađivanja (eng. *reconciliation phase*). Faza iscrtavanja se događa prilikom prikazivanja korisničkog sučelja gdje React stvara stablo virtualnog DOM-a i sprema ga u memoriju. Faza usklađivanja započinje kada se dogodi promjena stanja. Nakon promjene stanja React automatski kreira novo stablo virtualnog DOM-a za usporedbu. Usporedba se vrši koristeći algoritam za razlikovanje (engl. *diffing*) „reconciliation“ iliti usklađivanje. Na slici 1.1 su vidljiva oba DOM-a prije i nakon promjene, gornji red reprezentira stanje virtualnog DOM-a nakon promjene, a donji red prije.



Slika 1.1 Prikaz strategije za osvježavanje DOM-a stranice [prilagođeno od https://programmingwithmosh.com/react/react-virtual-dom-explained/]

1.1.4. Sintaksna nadogradnja JSX

JSX (engl. *JavaScript Syntax Extension*) je sintaksna nadogradnja na HTML (engl. *HyperText Markup Language*) koja omogućuje definiciju i izvršavanje JavaScript koda unutar elemenata HTML-a prilikom pretvorbe u čisti HTML. Kod Reacta logika iscrtavanja je povezana s cijelom logikom korisničkog sučelja (engl. *user interface*, UI): kako se obrađuju događaji, kako se stanje mijenja tijekom vremena te kako se podaci obrađuju prije prikazivanja korisniku. React, umjesto razdvajanja tehnologija na logiku i *markup*, razdvaja zahtjeve u slabo povezane jedinice - komponente, opisane ranije. JSX omogućuje programerima kreiranje koda sličnom HTML-u u JavaScriptu, on omogućuje deklarativnu specifikaciju strukture i sadržaja Reactovih komponenti. Na slici 1.2 vidljiv je primjer vrlo jednostavnog isječka koda koji koristi JSX. U ovom primjeru vidljivo je miješanje HTML elemenata s JavaScriptom. Prepoznajemo deklaraciju konstantne varijable *name* te konstantne varijable *element* u kojoj se referencira varijabla ime unutar HTML elementa h1 koji predstavlja naslov (engl. *heading*). Slika 1.3 predstavlja rezultat koda slike 1.2.

```
1  function App() {
2
3      const name = 'Fran Kristijan Jelenčić';
4      const element = <h1>Hello, {name}</h1>;
5
6    return (element);
7  }
8
9
10 export default App;
```

Slika 1.2 Kod – Primjer koda jednostavne aplikacije

Slika 1.3 Rezultat prethodnog koda

1.1.5. Protok podataka

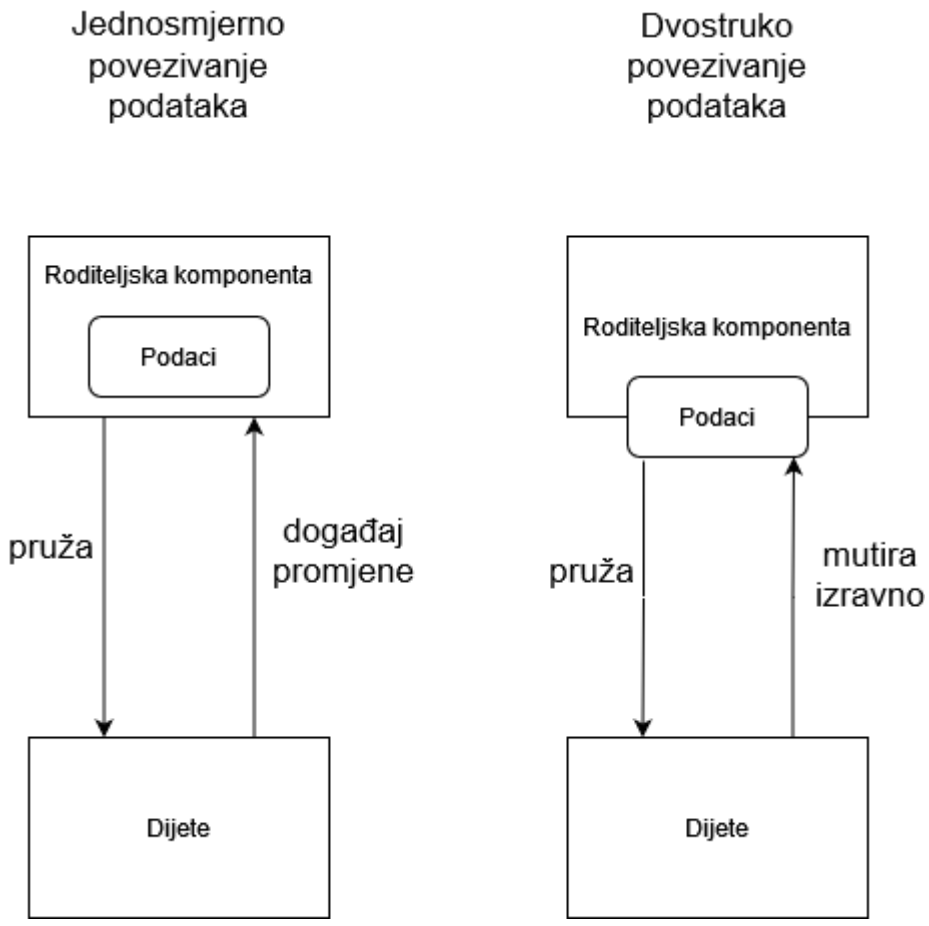
Jednosmjerni protok podataka je koncept u Reactu koji definira kako se podaci predaju i modificiraju unutar arhitekture temeljene na komponentama. Reactove komponente koriste dva moguća tipa pohranjivanja podataka: svojstva i stanje. Objekt svojstva se prosljeđuje od roditeljske komponente prema komponenti djetetu te je nepromjenjiv. Podaci koji se mijenjaju se postavljaju unutar objekta stanja koji je za svaku instancu komponente jedinstven. React komponente dizajnirane su za jednosmjerni protok podataka, što znači da kada se u roditeljskim komponentama dogodi promjena ta promjena putuje prema djeci, međutim suprotno nije moguće. Ovaj način povezivanja komponenti sprječava mogućnost komponente djeteta da promijeni stanje roditeljske komponente. Iako su Reactove komponente dizajnirane za jednosmjerni protok podataka, to ne znači da slanje u oba smjera nije moguće, jer React podržava *callback* funkcije koje omogućuju komponentama djeci emitiranje događaja promjene i mutiranje stanja roditeljskih komponenti.

Korake jednosmjernog povezivanja podataka možemo objasniti s ovih 6 točaka:

- Podaci roditeljske komponente
    - Tim podacima upravlja roditeljska komponenta, najčešće su to podaci definirani unutar njenog stanja ili dobiveni iz vanjskih izvora poput korisničkog unosa ili API poziva
- Svojstva (engl. *props*)
    - Roditeljska komponenta prosljeđuje podatke svojoj djeci putem svojstva. Oni se ne modificiraju u djetetu i omogućuju djeci korištenje podataka roditeljskih komponenti
- Iscrtavanje komponente djeteta
    - Komponenta djeteta zadužena je za grafički prikaz i implementaciju potrebne logike na temelju primljenih svojstava
- Upravljanje događajima
    - Kada komponenta djeteta treba promijeniti podatke ona to ne može izravno napraviti već poziva *callback* funkciju koju je dobila od roditeljske komponente i predaje podatke koje je potrebno ažurirati

- Ažuriranje podataka roditeljske komponente
    - Roditeljska komponenta prima ažurirane podatke od upravitelja događaja (engl. *event handler*) te na temelju tih podataka ažurira svoje stanje
- Ponovno iscrtavanje
    - Kada dođe do promjene stanja roditeljske komponente, React pokrene ponovno iscrtavanje komponenti koje su zahvaćene tom promjenom. Promjene se propagiraju hijerarhijom komponenti te se komponente djece ažuriraju s novim svojstvima

Slika 1.4 prikazuje koncept jednosmjernog i dvosmjernog povezivanja.



Slika 1.2 Grafički prikaz razlike između dvosmjernog i jednosmjernog povezivanja

[Prilagođeno od https://sandroroth.com/blog/react-two-way-data-binding]

1.2. Knjižnica Vue

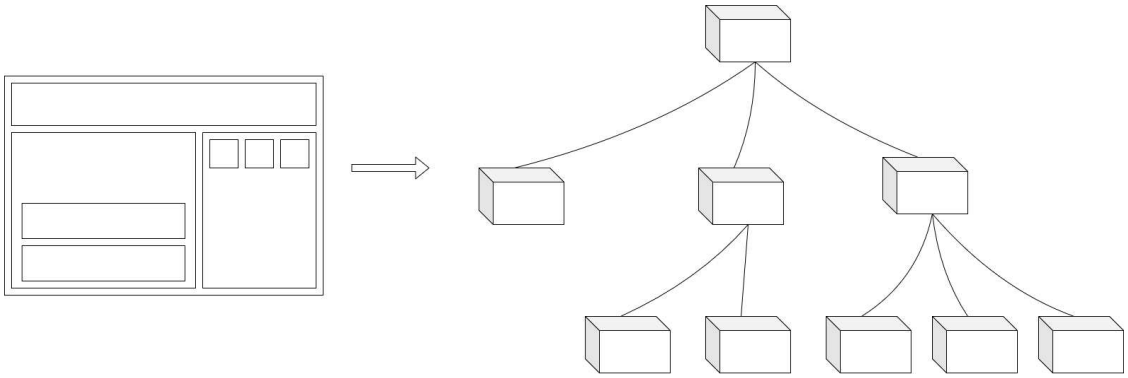
Vue.js je progresivni radni okvir koji služi za izradu korisničkih sučelja te jednostraničnih aplikacija. Kreirao ga je Evan You s ciljem izgradnje rješenja koje bi uzelo sve najbolje dijelove AngularJS-a i popravilo neke njegove nedostatke. Vue kombinira najbolje dijelove jednog od najpopularnijih JavaScriptovih radnih okvira AngularJS-a te dijelove JavaScriptove knjižnice React. Vue je zbog tih značajki poznat po svojoj jednostavnosti, fleksibilnosti i lakoći integracije. Glavni koncepti Vuea su: arhitektura bazirana na komponentama, jednodatotečne komponente, Vueova reaktivnost (virtualni DOM) te jednosmjeran protok podataka i direktive [4].

1.2.1. Dokumentacija i podrška zajednice

Vue je poznat kao radni okvir kojeg je lagano krenuti učiti i početi koristiti. Činjenica da je dokumentacija vrlo detaljna i pristupačna početnicima privlači velik broj korisnika. Na Stack Overflowu pitanja vezana za Vue čine 1.5% svih pitanja te taj broj samo raste posljednjih godina. Zajednica možda nije velika kao Reactova ili Angularova, ali velikom brzinom raste. Službena dokumentacija je vrlo detaljna, sadržaj je objašnjen na jasan i koncizan način te je dostupna u nekoliko jezika. Dokumentacija sadrži sve, od osnovnih koncepata do naprednih tema s mnoštvo primjera koda.

1.2.2. Arhitektura temeljena na komponentama

Arhitektura temeljena na komponentama znači da se aplikacija koja se razvija gradi od ponovno iskoristivih komponenata koji idealno ne zavise jedni o drugima. U Vueu svaki dio aplikacije je komponenta, na slici 1.5 vidljiva je ta podjela na manje komponente. Glavna komponenta sadrži sve ostale i njihovu djecu, gdje svaka komponenta može imati komponente djecu i to se može ponavljati rekurzivno.



Slika 1.5 Prikaz strukture komponenata jedne stranice [Prilagođeno od https://buddy.works/tutorials/introduction-to-vue-key-concepts]

Ključne točke Vueove arhitekture bazirane na komponentama su:

- Ponovna iskoristivost
    - Komponente implementiraju specifičan dio funkcionalnosti i mogu se iskoristiti na više mjesta u aplikaciji što smanjuje duplikaciju koda i poboljšava održivost koda
- Enkapsulacija
    - Svaka komponenta sadrži svoj HTML kod, JavaScriptov kod i stilove. Ovakva enkapsulacija omogućava izgradnju komponenata s jasno definiranim granicama i odgovornostima
- Hijerarhija komponenata
    - Komponente se organiziraju u hijerarhijsku strukturu gdje komponente roditelji sadrže komponente djecu koje mogu sadržavati svoju djecu i djeca djece svoju djecu i tako dalje. To omogućuje podjelu kompleksnog korisničkog sučelja na manje lakše održive dijelove te stvaranje strukture nalik stablu.
- Reaktivnost (virtualni DOM)
    - Vue koristi svoj sustav reaktivnosti kako bi efektivno osvježavao DOM na temelju promjene podataka. Kada se podaci u komponenti promijene, Vue automatski osvježi samo dio korisničkog sučelja koji je zahvaćen promjenama. Reaktivnost se postiže korištenjem posebnog reaktivnog podatkovnog objekta (engl. *data object*) i korištenjem direktive v-bind.

1.2.3. Jednodatotečne komponente

Jednodatotečne komponente Vuea koriste format .vue koji je poseban format datoteka koji omogućuje enkapsulaciju HTML koda, logike i stilova Vueove komponente u jednoj datoteci. Na slici 1.6 vidimo primjer sadržaja jedne datoteke .vue. Unutar oznaka *script* nalazi se JavaScriptov kod koji opisuje logiku komponente, njeno stanje i svojstva, oznaka *template* sadrži HTML kod koji opisuje HTML elemente koji sačinjavaju komponentu, a oznaka *style* opisuje CSS (engl. *Cascading Style Sheets*) stilove kojima se komponente koristi.

```
<script>
export default {
  name: "Primjer",
  data() {
    return {
      greeting: 'Pozdrav svijetu!'
    }
  }
}
</script>

<template>
  <p class="greeting">{{ greeting }}</p>
</template>

<style>
.greeting {
  color: red;
  font-weight: bold;
}
</style>
```

Slika 1.6 Primjer koda jedne komponente u datoteci .vue

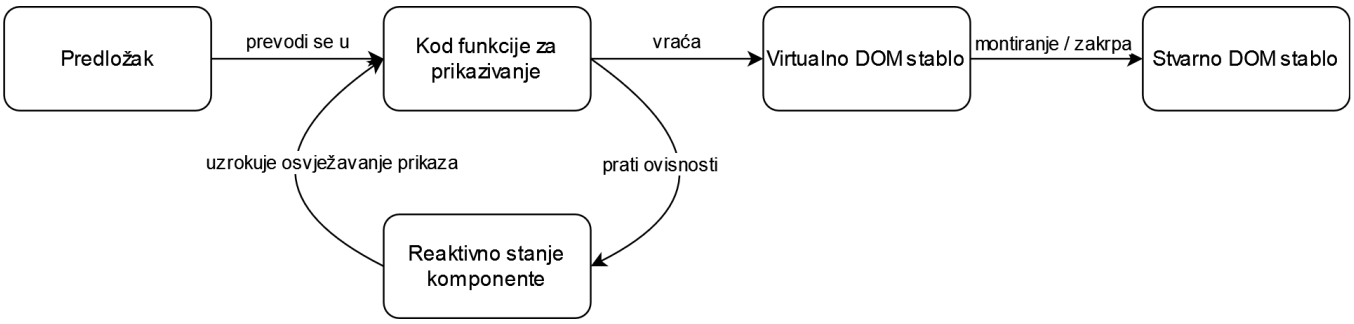
1.2.4. Reaktivnost Vuea

Reaktivnost Vuea odnosi se na osvježavanje prikaza korisničkog sučelja nakon promjene stanja unutar aplikacije. Vrlo slično Reactu, koristi se virtualni DOM, idealna, to jest, „virtualna“ reprezentacija korisničkog sučelja se čuva u memoriji te se sinkronizira s

„pravim“ DOM-om. Ne postoji samo jedna implementacija virtualnog DOM-a, već više njih, i Vue koristi sljedeći način kako bi ostvario virtualni DOM. Postoje tri faze koje se prilikom prikazivanja komponente odvijaju i one su:

- Prevođenje
    - Sadržaj predloška (engl. *template*) komponente prevodi se u kod funkcije za prikazivanje (funkcija za prikazivanje je funkcija koja vraća virtualno stablo DOM-a).
- Montiranje
    - Izvršni prikazivač (engl. *runtime renderer*) poziva funkciju za prikaz, prolazi kroz virtualno stablo DOM-a i kreira stvarne DOM-ove elemente na temelju tog prolaza. Ovaj korak izvodi se kao reaktivni efekt što znači da prati komponente i njihove ovisnosti koje su iskorištene prilikom prikazivanja.
- Zakrpa
    - Ako dođe do promjene neke od komponenata o kojoj druge ovise tijekom montiranja ponovno se osvježava prikaz, ali ovaj put se stvara novo, ažurirano virtualno stablo DOM-a. Izvršni prikazivač prolazi kroz novo stablo, uspoređuje ga sa starim i primjenjuje potrebne promjene u stvarnom DOM-u.

Na slici 1.7 vidimo proces s nacrtanim koracima te redoslijedom događanja.



Slika 1.3 Prikaz procesa reaktivnosti Vuea

1.2.5. Protok podataka

Vrlo slično Reactu, Vue za predaju podataka između komponenata koristi jednosmjerni protok podataka. Isto kao u Reactu postoje svojstva (*props*) i stanje (*state*). Svi objekti svojstva formiraju jednosmjerno povezivanje podataka prema djeci, kada se roditeljsko svojstvo promijeni ta promjena će se proslijediti djetetu, ali ne i obrnuto. Tako se sprječava mutacija roditeljskog stanja od strane komponenata djece. Prilikom svakog ažuriranja, podatak roditeljske komponente koji sadržava sve objekte svojstava djece je ažuriran s najnovijom vrijednošću. Važno je napomenuti da zbog ograničenja jezika JavaScript postoji mogućnost da se mutira roditeljsko stanje ako je dio objekta svojstva niz (engl. *array*), jer se objekti u JavaScriptu predaju preko referenci. Ovaj slučaj Vue ne sprječava, jer je provjera takvih mutacija računalno zahtjevna. Vue podržava jednosmjerno i dvosmjerno povezivanje podataka pomoću direktiva *v-bind* i *v-model*. Više detalja o direktivama i primjeri bit će prikazani u sljedećem potpoglavlju.

1.2.6. Direktive

Direktive u Vueu su specijalni atributi u HTML-ovim oznakama koji se koriste za manipulaciju DOM-om i za ostvarivanje reaktivnog ponašanja u komponentama. Svaka direktiva ima prefiks „v-„ i nalaze se u *template* dijelu komponente. Najčešće korištene direktive su:

- *v-bind*
    - Omogućuje jednosmjerno povezivanje atributa ili svojstva nekog elementa sa specificiranim izrazom koji se dinamički po potrebi ažurira. Primjer korištenja direktive vidljiv je na slici 1.8 gdje se koristi unutarelementa paragrafa (<p>) za prikaz poruke i postavljanje atributa slike *src*.

```
<template>
  <div>
    <p>{{ message }}</p>
    <img v-bind:src="imageSrc">
  </div>
</template>

<script>
export default {
  data() {
    return {
      message: 'Pozdrav svijetu!',
      imageSrc: 'https://primjer.hr/slika.jpg'
    }
  }
}
```

Slika 1.4 Kod – primjer korištenja direktive *v-bind*

- *v-model*
    - Omogućuje dvosmjerno povezivanje podataka između elementa za unos i podataka neke komponente, glavna funkcija je automatska sinkronizacija vrijednosti unosa s podacima unutar komponente. Na slici 1.9 vidljiv je primjer korištenja dvosmjernog povezivanja podataka, prilikom unosa u polje za unos teksta ažurira se tekst „Vaše korisničko ime je:“ s korisničkim unosom.

```
<template>
  <div>
    <p>Vaše korisničko ime je: {{username}}</p>
    <input v-model="username" placeholder="Unesite korisničko ime">
  </div>
</template>

<script>
export default {
  data() {
    return {
      username: ''
    }
  }
}
</script>
```

Slika 1.5 Kod – Primjer korištenja direktive *v-model*

- *v-if* / *v-else* / *v-else-if*
    - Direktive koje se koriste za uvjetni prikaz: direktiva *v-if* određuje hoće li se element prikazati na temelju istinitosti izraza definiranog u direktivi, *v-else* specificira koji element će se prikazati ako je izraz u direktivi *v-if* neistina te *v-else-if* omogućuje stvaranje lančanih uvjeta za prikaz. Na slici 1.10 vidljiv je primjer korištenja direktive za generiranje sadržaja temeljnog na istinitosti tvrdnje *value*.

```
<template>
  <div>
    <p v-if="value === 1">Vrijednost je 1.</p>
    <p v-else-if="value === 2">Vrijednost je 2.</p>
    <p v-else>Vrijednost je različita od 1 i 2.</p>
  </div>
</template>

<script>
export default {
  data() {
    return {
      value: 1,
    }
  }
}
</script>
```

Slika 1.10 Kod – Primjer korištenja direktiva *v-if*, *v-else, v-else-if*

- *v-for*
    - Koristi se prikaz lista ili za iteraciju kroz listu. Omogućuje dinamičko generiranje elemenata na temelju sadržaja liste. Na slici 1.11 vidljivo je korištenje direktive za generiranje tri elementa liste (<*li*>) s vrijednostima „Auto“, „Autobus“ i „Vlak“.

```
<template>
  <div>
    <ul>
      <li v-for="(item, index) in items" :key="index">{{ item }}</li>
    </ul>
  </div>
</template>

<script>
export default {
  data() {
    return {
      items: ['Auto', 'Autobus', 'Vlak'],
    }
  }
}
</script>
```

Slika 1.11 Kod – Primjer korištenja direktive *v-for*

- *v-on*
    - Direktiva koja se koristi za upravljanje događajima. Omogućuje dodavanje promatrača na elemente i izvršavanje funkcija na taj događaj. Na slici 1.12 nalazi se kod koji, kada korisnik pritisne na gumb „Inkrementiraj“ poveća vrijednost brojača za 1.

```
<template>
  <div>
    <button v-on:click="increment">Inkrementiraj</button>
    <p>Brojač: {{ count }}</p>
  </div>
</template>

<script>
export default {
  data() {
    return {
      count: 0,
    }
  },
  methods: {
    increment() {
      this.count += 1;
    }
  }
}
</script>
```

Slika 1.12 Kod – Primjer korištenja direktive *v-on*

1.3. Knjižnica Angular

Angular je radni okvir otvorenog koda baziran na TypeScriptu, razvijan od strane Googlea i zajednice. Angular je zapravo termin za sve verzije radnog okvira od vremena inicijalnog objavljivanja, verzija koja je istražena za ovaj rad je Angular verzija 16. Verzija 16 se temelji na svojem prethodniku koji je poznat kao AngularJS čija je podrška završila u siječnju 2022. godine. Glavne značajke radnog okvira Angular su: arhitektura bazirana na komponentama, arhitekturni obrazac model-pogled-nadzornik (engl. *Model-View-Controller*, dalje: MVC), inkrementalni DOM, jednosmjerni protok podataka i direktive [3].

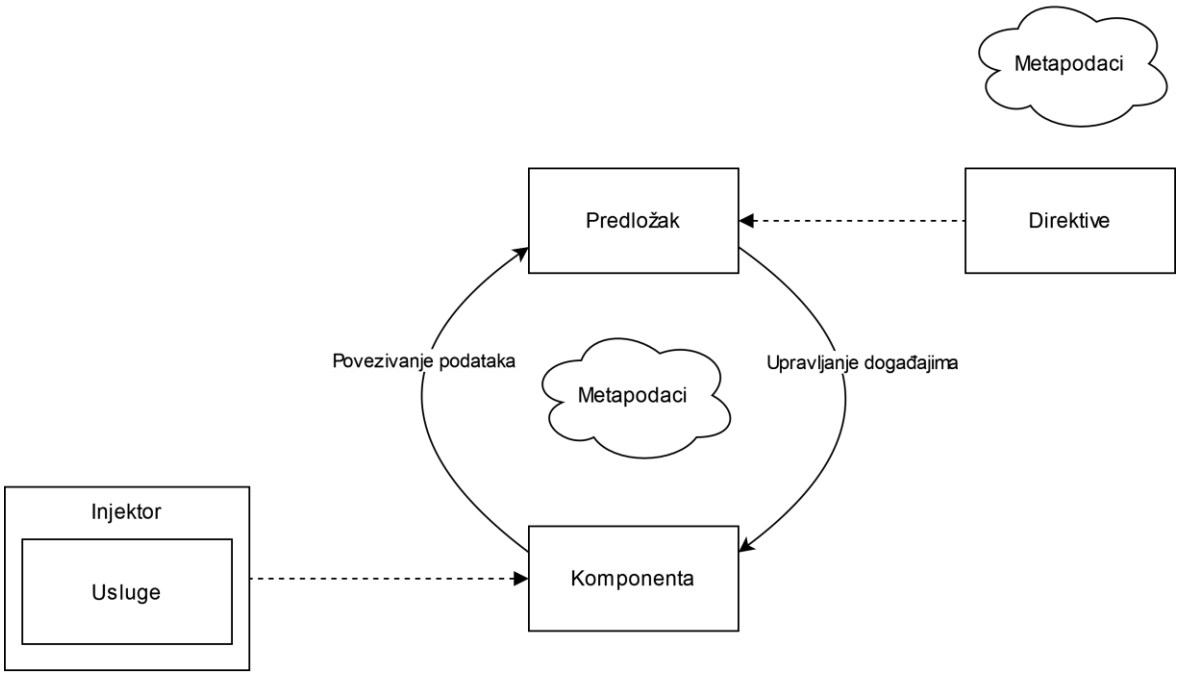
1.3.1. Dokumentacija i podrška zajednice

Angular razvija Google i samim time garantirana je velika kvaliteta i količina dokumentacije. Službena dokumentacija posebno sadrži skupinu uputa „Tour of Heroes“ namijenjenu početnicima, koja uz primjere pruža uvod u korištenje Angulara. Ekosustav Angulara je iznimno velik, resursi za učenje Angulara mogu se pronaći na popularnim web stranicama za učenje poput Udemya, Pluralsighta, Coursere i freeCodeCampa. Međutim, naučiti Angular je relativno teško zbog njegove kompleksnosti. Budući da je Angular Googleov projekt, ažuriranja i sigurnosne zakrpe su vrlo česte te je vrlo popularan u velikim poslovnim okruženjima zbog skalabilnosti i strogih arhitekturnih smjernica.

1.3.2. Arhitektura temeljena na komponentama

Angular, kao i React i Vue, koristi arhitekturu temeljenu na komponentama, gdje je svaka komponenta kohezivan paket koji sadrži HTML, CSS i JavaScriptov kod koji je potreban za njen rad. Srž Angularove aplikacije su komponente koje su organizirane u NgModule (engl. *NgModules*) koji sadrže kod u funkcionalnim skupovima. Komponente definiraju poglede (engl. *view*) koji predstavljaju skup elemenata korisničkog sučelja kojima Angular upravlja na temelju programske logike i podataka. Komponente mogu koristiti usluge (engl. *services*) koje ostvaruju funkcionalnosti koje nisu direktno povezane s pogledima. Usluge se u komponentama mogu definirati kao ovisnosti (engl. *dependency*) čime kod postaje modularan, ponovo upotrebljiv i efikasan. Struktura komponente je najčešće definirana u tri datoteke: predlošku koji sadrži HTML kod, datoteci s poslovnom logikom koja sadrži TypeScriptov kod i CSS stilovima koje predložak koristi. Na slici 1.13 prikazan

je dijagram koji predstavlja kako se odnose navedeni dijelova. Predložak i poslovna logika predstavljaju Angularov pogled, dekorator na razredu komponente dodaje metapodatke, uključujući pokazivač na povezan predložak, a direktive i povezivanje podataka unutar predloška mijenjaju pogled na temelju podataka i poslovne logike.



Slika 1.13 Dijagram povezanosti arhitekture temeljene na komponentama

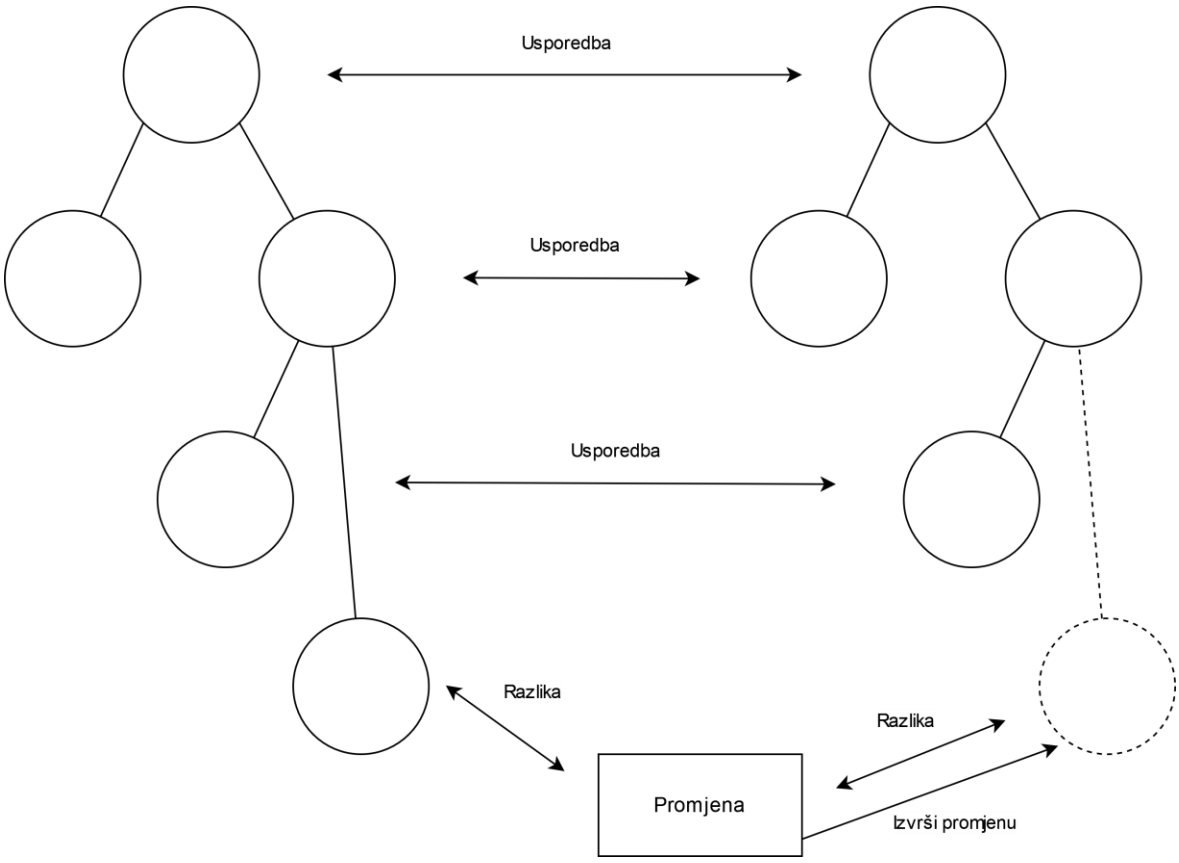
1.3.3. MVC arhitekturni obrazac

Arhitekturni obrazac MVC temelji se na podjeli arhitekture na tri dijela: model (obrada), nadzornik (ulaz), pogled (izlaz). U Angularu funkciju pogleda izvršavaju predlošci s HTML kodom i CSS stilovima, oni podatke dobivaju iz datoteka s poslovnom logikom, s time da datoteke s poslovnom logikom služe za interakciju s uslugama tako da poslovna logika zajedno s uslugama predstavlja nadzornički dio arhitekture. Između pogleda i nadzornika podaci su reprezentirani u obliku modela koji predstavljaju podatke u aplikaciji i tipično su to TypeScriptovi razredi.

1.3.4. Inkrementalni DOM

Za razliku od Reacta i Vuea koji koriste virtualni DOM, Angular koristi inkrementalni DOM. Inkrementalni DOM je termin koji se odnosi na način upravljanja DOM-om koji je memorijski efikasniji i može dovesti do većih performansi u određenim slučajevima u odnosu na virtualni DOM. Kod strategije inkrementalnog DOM-a umjesto kreiranja

poptune virtualne reprezentacije DOM-a i izvršavanja algoritma usklađivanja sa stvarnim DOM-om, prate se samo promijenjeni elementi i njihove promjene se primjenjuju na stvarni DOM. Angular, međutim, ne koristi čistu implementaciju inkrementalnog DOM-a već koristi sličan pristup koji se temelji na detekciji promjena. Angular tijekom prevođenja stvori pogled (engl. *View*) za svaku komponentu, koji predstavlja direktno preslikavanje predloška komponente zajedno s njezinim poveznicama s podacima i djecom. U trenutku kada se dogodi promjena Angular mehanizam detekcije promjena se aktivira. On provjerava za svaki podatak povezan preko direktiva prethodnu i trenutnu vrijednost te ako postoji razlika, ažurira se DOM. Ovaj pristup može se smatrati mješavinom virtualnog DOM-a i inkrementalnog DOM-a. Kao u virtualnom DOM-u Angular stvara virtualnu reprezentaciju DOM-a (pogled), ali kao inkrementalni DOM u trenutku događaja promjene podataka komponente Angular izravno mijenja DOM s time da mijenja samo dijelove DOM-a koje je zahvatila promjena. Angularov mehanizam detekcije promjena je optimiran tako da korisničko sučelje bude responzivno i kod aplikacija koje imaju kompleksna korisnička sučelja. Na slici 1.14 vidljiv je primjer inkrementalnog DOM-a i njegove strategije pronalaska promijenjenih elemenata te njihovog ažuriranja.



Slika 1.14 Prikaz strategije inkrementalnog DOM-a

1.3.5. Protok podataka

Angular, poput Vuea i Reacta, implementira strategiju jednosmjernog protoka podataka za upravljanje komunikacije između komponenata. U Angularu jednosmjeran protok podataka ostvaruje se kombinacijom komponenata, načina povezivanja podataka, upravljanja događajima i uslugama. Komponente koriste @Input() i @Output() dekoratore kako bi omogućile jednosmjeran protok podataka od roditelja prema djetetu i obrnuto. Kada dođe do promjene u dekoratoru *input*, Angularov sustav za detekciju promjena je aktiviran i on kreće s ažuriranjem pogleda s izmijenjenim podacima. Kod protoka podataka potrebno je spomenuti način povezivanja podataka i poput Vuea, Angular podržava jednosmjerno i dvosmjerno povezivanje podataka. Postoje tri načina povezivanja podataka:

- Od izvora podataka prema pogledu
    - Ostvaruje se korištenjem izraza unutar []

```
<div [innerText]="message"></div>
```

Slika 1.15 Kod – Primjer jednosmjernog povezivanja – svojstvo

- Od pogleda prema izvoru podataka
    - Ostvaruje se korištenjem izraza unutar ()

```
<button (click)="onClick()">Pritisni me</button>
```

Slika 1.16 Kod – Primjer jednosmjernog povezivanja – događaj

- Dvosmjerno povezivanje pogleda i izvora podataka
    - Ostvaruje se korištenjem izraza unutar [()]

```
<input [(ngModel)]="name"> <p>Pozdrav, {{name}}!</p>
```

Slika 1.17 Kod – primjer dvosmjernog povezivanja

1.3.6. Direktive

Direktive u Angularu predstavljaju načine dodavanja dodatnih mogućnosti elementima predložaka ili modifikacija postojećih mogućnosti. U Angularu postoje tri vrste direktiva i one su:

- Komponente – direktive s predloškom
- Strukturalne direktive – mijenjaju DOM dodajući ili mičući elemente
    - direktiva *ngIf* – prikazuje element ovisno o istinitosti tvrdnje, primjer vidljiv na slici 1.18

```
<div *ngIf="showDiv">Vidljivo ako je 'showDiv' istina.</div>
```

Slika 1.6 Kod – primjer direktive *ngIf*

    - direktiva ngFor – omogućuje dinamički prikaz elemenata liste, primjer vidljiv na slici 1.19

```
<div *ngFor="let item of itemsList">{{ item }}</div>
```

Slika 1.7 Kod – primjer direktive *ngFor*

    - direktiva ngSwitch – uvjetni prikaz elemenata ovisno o izrazu, primjer vidljiv na slici 1.20

```
<div [ngSwitch]="transportationType">
    <div *ngSwitchCase="'Auto'">Vozimo se autom.</div>
    <div *ngSwitchCase="'Autobus'">Vozimo se autobusom.</div>
    <div *ngSwitchCase="'Vlak'">Vozimo se vlakom.</div>
    <div *ngSwitchDefault>Vozimo se nepoznatim vozilom.</div>
</div>
```

Slika 1.20 Kod - primjer direktive *ngSwitch*

- Atributne direktive – mijenjaju izgled ili ponašanje elementa, komponente ili druge direktive
    - *ngStyle* – koristi se za dinamičku modifikaciju CSS stilova, primjer vidljiv na slici 1.21

```
<div [ngStyle]="{color: 'red'}">Ja sam crvene boje.</div>
```

Slika 1.21 Kod – primjer direktive *ngStyle*

- *ngClass* – omogućuje dinamičko dodavanje CSS razreda, primjer vidljiv na slici 1.22

```
<div [ngClass]="{'active': isActive}">
    Ja sam div s razredom "active" ako je isActive istina.
</div>
```

Slika 1.22 Kod – primjer direktive *ngClass*

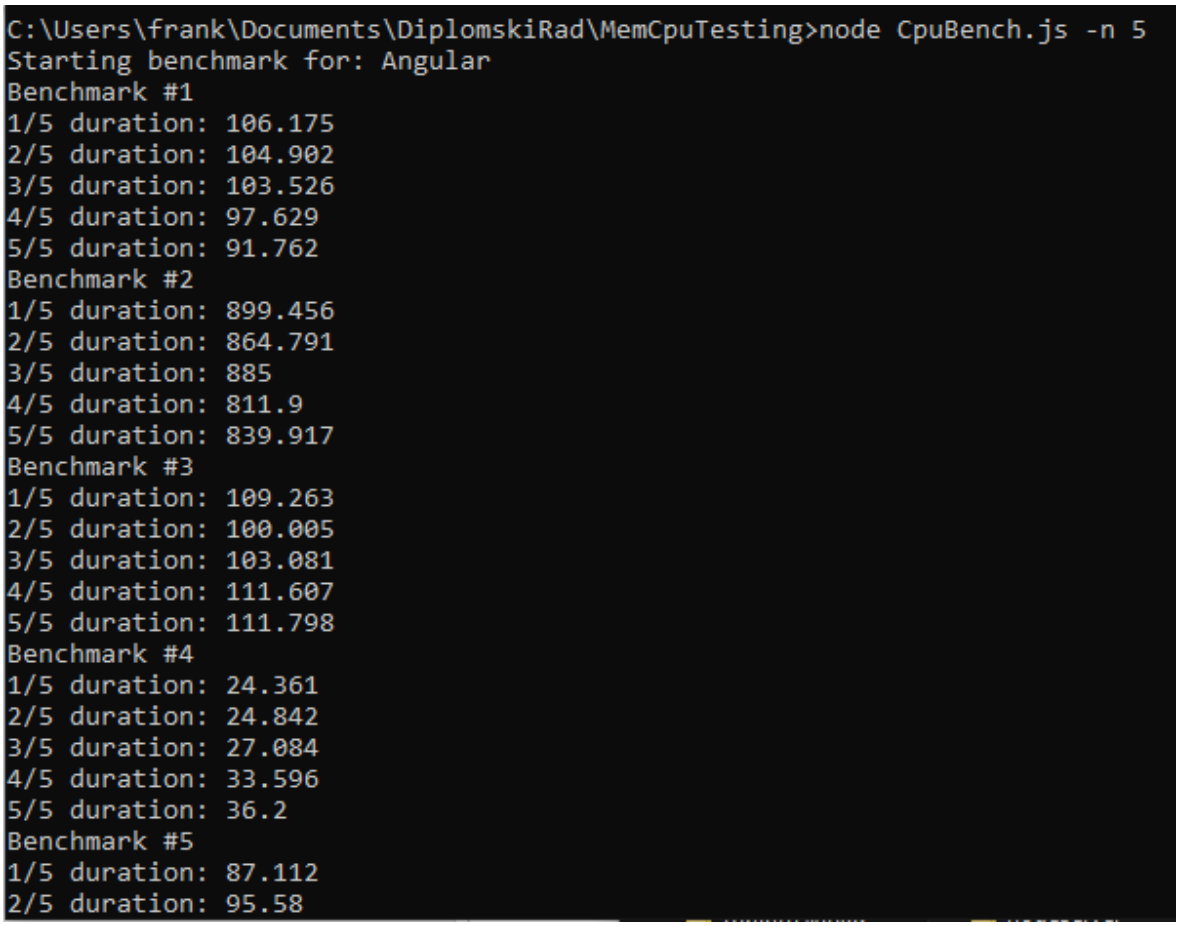
2. Kvantitativna analiza

Ovo poglavlje fokusira se na kvantitativnu analizu knjižnica React, Angular i Vue. Kvantitativna analiza metoda je istraživanja u kojoj se prikupljaju i analiziraju numerički podaci. Koristi se za pronalaženje uzoraka, prosjeka i generalizacija rezultata na šire populacije. U ovom poglavlju predstavit će se razvijeni alat za testiranje performansi JavaScriptovih knjižnica i rezultate koji su dobiveni pomoću tog alata.

2.1. Alat za ispitivanje performansi

Za svrhe ovog diplomskog rada razvijen je alat čija je svrha korištenje memorije, procesorskog vremena te sveukupna efikasnost knjižnica. Alat je razvijen koristeći programski jezik JavaScript i tehnologiju Node.js uz pomoć dodatne knjižnice Pupeteer, koji omogućuje automatizaciju web preglednika i alata Lighthouse, koji omogućuje jednostavnu analizu performansi, pristupačnosti i korištenje najboljih praksi. Navedeni alati za rad koriste web preglednike bazirane na bazi koda (eng. *codebase*) Chromium te je za ispitivanje korišten web preglednik Chrome verzije 114.0.5735.110. Ispitivanje je izvršeno na prijenosnom računalu s procesorom Intel Core i5-10210U i 8 GB radne memorije. Alat se sastoji od tri modula koji obavljaju različite vrste ispitivanja; modul za analizu korištenja procesorskog vremena, modul za analizu korištenja radne memorije te modul za analizu raznih metrika uključujući veličinu datoteka potrebnih za rad stranice i vremena potrebnih za učitavanje svih JavaScriptovih skripti i CSS stilova na stranici. Svaki od modula definiran je u zasebnoj datoteci. Srž alata čini poslužitelj Node.js s kojeg web preglednik Chrome pod kontrolom knjižnice Pupeteer ili alata Lighthouse dohvaća knjižnice. Poslužitelj se pokreće na vratima (eng. *port*) 3000 i ovisno o putanji poslužuje Angular, React ili Vue inačice web stranice. Moduli se pokreću iz naredbenog retka (engl. *command line*) koristeći Node.js te uz pomoć sučelja koja pružaju Pupeteer i Lighthouse dohvaćaju relevantne podatke iz web preglednika. Ispitni slučajevi su definirani u posebnoj datoteci tako da se dodavanje novih inačica ispitnih slučajeva sastoji od dodavanja koda tih ispitnih slučajeva u tu posebnu datoteku te dodavanja ispitnog slučaja u listu ispitnih slučajeva za izvršavanje. Na slici 2.1 vidimo primjer pokretanja i izvršavanja ispitivanja procesorskog vremena s parametrom *n* koji određuje broj iteracija, dok ostali moduli imaju

identičan ispis, samo pokreću različite ispitne slučajeve. Rezultati se agregiraju i spremaju u dvije Excel datoteke, jedna datoteka za tablice koje će biti prikazane u sljedećim potpoglavljima i jedna u kojima se nalaze rezultati svakog ispitivanja.



Slika 2.1 Primjer pokretanja modula za analizu korištenja procesorskog vremena

2.1.1. Knjižnica Pupeteer

Pupeteer je Node.js-ova knjižnica koju je razvio Chromeov tim, a koja pruža API visoke razine za upravljanje bezglavim (eng. *headless*) instancama web preglednika Chrome koristeći protokol DevTools. Bezglavi web preglednici su preglednici bez grafičkog sučelja. Knjižnica Pupeteer primarno se koristi za automatizaciju poslova u web pregledniku, poput generiranja slika ekrana, automatizirano ispitivanje, mjerenje performansi i indeksiranje web stranica u svrhe optimizacije web tražilica. Ona omogućuje programiranje interakcija s web stranicama, to jest emulaciju korisnika, omogućuje ispunjavanje formi, simulaciju korisničke navigacije i korisničke interakcije s web stranicom. Knjižnica Pupeteer koristi se u modulu za analizu procesorskog vremena i modulu za analizu korištenja radne memorije.

2.1.2. Alat Lighthouse

Lighthouse je alat otvoren koda koji je razvio Google radi poboljšanja kvalitete web stranica. Alat omogućuje analizu performansi, pristupačnosti, progresivnih web aplikacija i optimizaciju web tražilica. Može se pokrenuti iz Chromeovog alata za razvijatelje ili kao modul Node.js-a. Na željenoj stranici Lighthouse provede niz ispitivanja i generira izvješće s rezultatima. Koristeći ta izvješća programeri mogu optimirati svoj kod kako bi njihove web stranice imale bolje performanse.

2.1.3. Oblikovanje ispitivanja

U provedenim eksperimentima cilj je bio ispitati performanse knjižnica i osigurati neovisnost rezultata o broju izvršavanja eksperimenta. Kako bi se osigurala ponovljivost, način prikupljanja podataka o vremenu izvršavanja, potrošnji memorije i ostale metrike su potpuno automatizirane, korisnik treba samo pokrenuti ispitivanja. U svakoj od knjižnica generirana je web stranica koja koristi identičnu DOM-ovu strukturu i funkcionalnost kako bi se osigurala konzistentnost i ravnopravnost rezultata. Razvijena web stranica sastoji se od tablice i šest gumba, od kojih svaki upravlja jednom mogućnosti manipulacije DOM-om. Funkcije gumbi su:

1. Kreiraj 1000 redaka
2. Kreiraj 10000 redaka
3. Dodaj 1000 redaka na trenutačni broj redaka
4. Ažuriraj sadržaj svakog desetog retka
5. Pobriši sve retke
6. Zamijeni drugi i predzadnji redak

Svaki redak u tablici predstavlja jedan element DOM-a. Tablica se nalazi ispod gumbi i inicijalno je prazna. Ispitivanja se sastoje od pozivanja različitih kombinacija navedenih funkcija kako bi se pratile performanse pod različitim naporima i simulacijama različitih slučajeva korištenja. Testovi su napravljeni modularno što znači da je moguće dodavati dodatne JavaScriptove knjižnice nad kojima se testovi mogu izvršavati i prikupljati podaci o njihovim performansama. Svaki od ispitivanja procesorskog vremena pokrenut je na dva načina: sa zagrijavanjem i bez, kako bi se demonstrirale razlike prilikom prvog učitavanja stranice i produljene interakcija. Zagrijavanje se odnosi na izvršavanje promjena DOM-a

prije pokretanja pravog ispitnog slučaja u kojem se računa korištenje procesorsko vrijeme. Svi ispitni slučajevi sa zagrijavanjem koriste 5 iteracija zagrijavanja, a zagrijavanja su različita ovisno o tome koje ispitivanje se izvodi.

2.2. Analiza korištenja procesorskog vremena

Računanje korištenja procesorskog vremena temelji se na računanju proteklog vremena nakon pritiska na gumb i vremena kada se pojave svi elementi na web stranici. Kako bi se dobili najtočniji rezultati prikupljaju se svi događaji unutar web preglednika tijekom izvršavanja ispitnih slučajeva. Ti događaji uključuju: lijevi klik miša, događaj iscrtavanja ekrana, događaj računanja rasporeda elemenata, događaj pokretanja animacije, događaj ažuriranja DOM-a i ostali. Navedeni događaji su oni koji se posebno prate prilikom izvršavanja ispitivanja i potrebno je napomenuti da su to događaji unutar web preglednika, a ne knjižnica. Kod računanja korištenja procesorskog vremena korišteni su događaji lijevog klika, računanja rasporeda elemenata i iscrtavanja ekrana. Za početak vremena koristi se vrijeme lijevog klika te se zatim traži događaj i računa raspored elemenata i nakon njega iscrtavanje ekrana. Trajanje jednog ažuriranja DOM-a je od klika do prvog iscrtavanja s ažuriranim elementima i izraženo je u milisekundama.

2.3. Ispitivanje kreiranja elemenata DOM-a

2.3.1. Prvo ispitivanje – Dodavanje 1000 redaka

Prvo ispitivanje sastoji se od dodavanja 1000 redaka na web stranicu. U tablici 2.1 prikazani su rezultati ispitivanja sa zagrijavanjem u kojemu je za svaku knjižnicu provedeno 100 iteracija istog ispitivanja. Rezultati pokazuju kako najbolju prosječnu vrijednost ima Angular, zajedno s najmanjom standardnom devijacijom, gotovo dva puta manjom od ostalih knjižnica, što upućuje da ujedno i pruža najkonzistentnije performanse. Nakon Angulara je Vue, s prosjekom od samo 4 milisekunde većim od Angulara, međutim većom devijacijom i najvećim trajanjem za jedno ažuriranje DOM-a za čak 130 milisekundi. React je na posljednjem mjestu, s prosječnim vremenom gotovo deset milisekundi većim od Angulara i s najvećom devijacijom. U tablici 2.2 ponovno najbolje performanse pokazuje Angular s rezultatima gotovo deset milisekundi boljim od ostalih knjižnica i s uvjerljivo najmanjom varijancom. Nakon Angulara dolazi React, čija

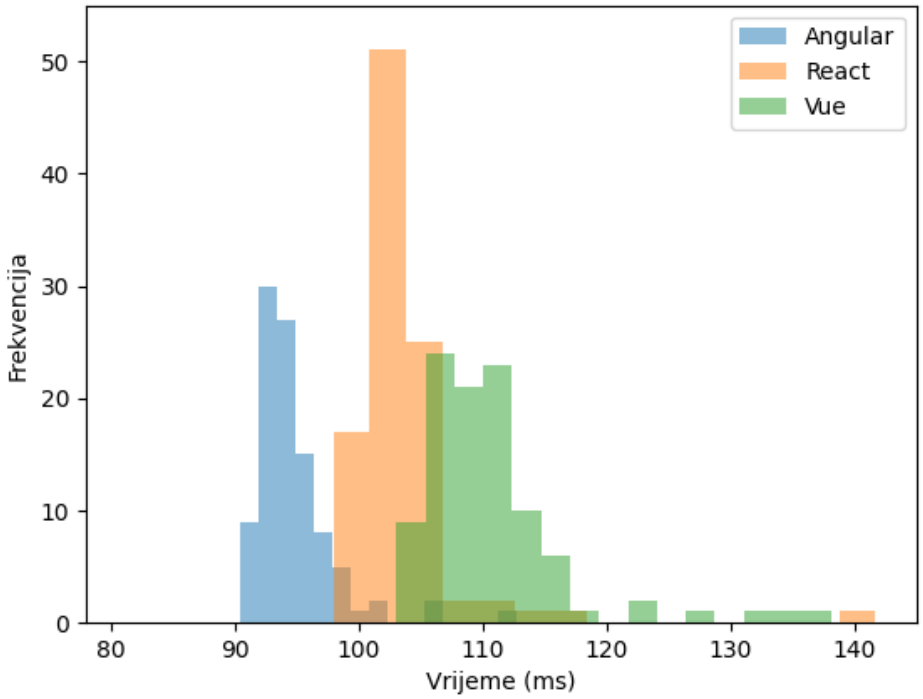
prosječna vrijednost utrošenog vremena slično ispitivanju sa zagrijavanjem iznosi deset milisekundi više nego Angular, ali u ispitivanju bez zagrijavanja pokazuje mnogo manju varijancu, gotovo 2 puta manju. Najgore rezultate pokazuje Vue koji je 20 milisekundi sporiji u ispitivanju bez zagrijavanja. Na slici 2.2 i slici 2.3 vidljivi su histogrami vrijednosti vremena i na njima je vidljivo kako Angular stvarno ima najmanje stršećih podataka, a React i Vue imaju veći raspon vrijednosti.

Tablica 2.1 Rezultati sa zagrijavanjem (u milisekundama)

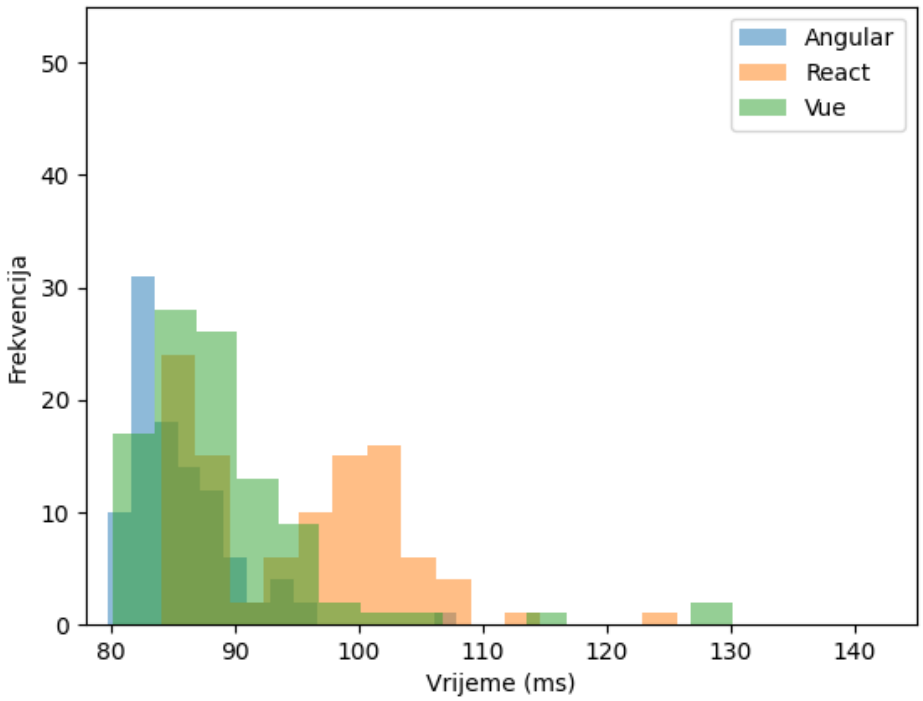
Ime	Prosjek	Standardna devijacija	Min	Max
Angular	85,487	4,302	79,738	107,913
React	95,080	8,046	84,013	125,672
Vue	89,242	7,856	80,190	130,147

Tablica 2.2 Rezultati bez zagrijavanja (u milisekundama)

Ime	Prosjek	Standardna devijacija	Min	Max
Angular	94,702	3,256	90,467	112,655
React	103,394	4,777	97,971	141,687
Vue	110,644	5,952	103,021	138,044



Slika 2.2 Ispitivanje 1 – Histogram vrijednosti sa zagrijavanjem



Slika 2.3 Ispitivanje 1 – Histogram vrijednosti bez zagrijavanja

2.3.2. Drugo ispitivanje – Dodavanje 10000 redaka

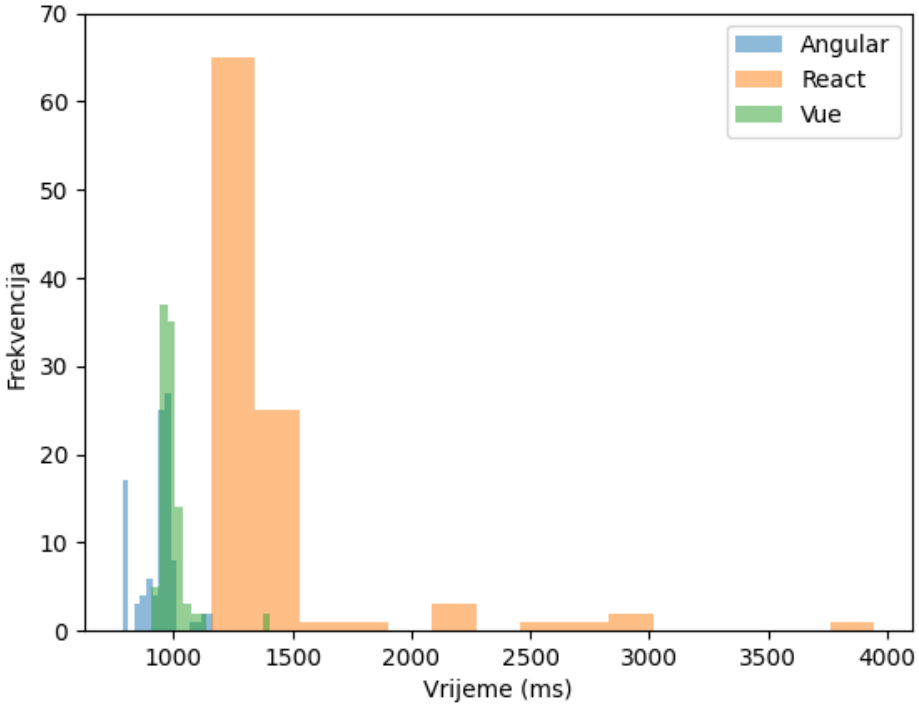
Drugo ispitivanje je gotovo identično prvom, jedina razlika je što se sada kreira 10000 redaka. Ideja je bolje razumijevanje skalabilnosti aplikacija i određivanja kolika je degradacija performansi prilikom velikog povećanja broja elemenata na stranici. Tako se simulira teže opterećenje koje se može dogoditi u stvarnom svijetu i ispituje se učinkovitost algoritama za upravljanje DOM-om. Ponovno postoje dvije vrste ispitivanja, sa i bez zagrijavanja. U tablici 2.3 prikazani su rezultati testa bez zagrijavanja i slično rezultatima prvog testa najmanju prosječnu vrijednost korištenja procesorskog vremena ima Angular, iza njega slijedi Vue s prosječnim kašnjenjem od 60 milisekundi i React, čije performanse pokazuju vrlo velike varijacije što je vidljivo iz standardnih devijacija gdje je Reactova devijacija 5 puta veća od ostalih knjižnica. Razlog toliko velike devijacije kod Reacta može biti velik broj faktora, među kojima je najvjerojatniji Reactovo grupiranje elemenata u veće cjeline te provođenje skupnog ažuriranja (eng. *batch update*). Međutim razlog također može biti i algoritamska složenost strategije računanja ažuriranog DOM-a. Zanimljivost ovdje pokazuje Angular koji, kao što je vidljivo na histogramu na slici 2.4, ima vrlo konzistentne rezultate, pogotovo u intervalu od 780 ms do 1000 ms. Nadalje, raspon rezultata Reacta vidljivo odstupa i, unatoč činjenici da je frekvencija rezultata najviše koncentrirana na 1200 milisekundi, njegov prosjek je daleko veći zbog nekonzistentnosti vremena. U tablici 2.4 rezultati sa zagrijavanjem pokazuju manje prosjeke i mnogo manje standardne devijacije. React u odnosu na test bez zagrijavanja pokazuje mnogo manji raspon vrijednosti i vrijednosti su mnogo manje raspršene čemu svjedoči devijacija od 20,5 milisekundi, daleko najmanja od svih knjižnica. Na slici 2.5 vidljive su tri grupe vrijednosti gotovo bez preklapanja, s jasnim pobjednikom po pitanju brzine Angularom.

Tablica 2.3 Rezultati bez zagrijavanja (u milisekundama)

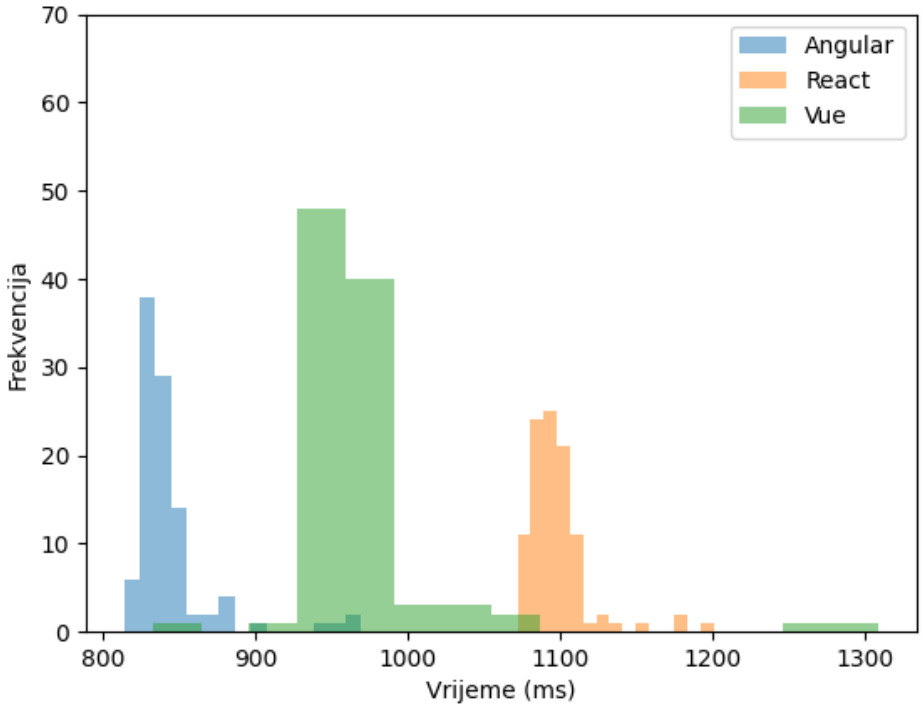
Ime	Prosjek	Standardna devijacija	Min	Max
Angular	936,897	83,387	784,830	1168,021
React	1429,018	417,335	1159,656	3947,250
Vue	993,182	68,017	908,433	1407,038

Tablica 2.4 Rezultati sa zagrijavanjem (u milisekundama)

Ime	Prosjek	Standardna devijacija	Min	Max
Angular	844,285	27,427	814,003	969,560
React	1097,646	20,583	1072,043	1200,489
Vue	969,632	52,899	832,737	1308,816



Slika 2.4 Ispitivanje 2 – Histogram vrijednosti bez zagrijavanja



Slika 2.5 Ispitivanje 2 – Histogram vrijednosti sa zagrijavanjem

2.3.3. Treće ispitivanje – Dodavanje 1000 redaka na trenutačne

Treće ispitivanje temelji se na iterativnom povećanju broja redaka, dodaje se 1000 redaka prije pokretanja mjerenja vremena te se pokreće mjerenje vremena i dodaje se 1000 redaka na postojećih 1000 tako da ukupno postoji 2000 redaka na web stranici. Dodavanje redaka je česta operacija na web stranicama, na primjer prilikom beskonačnog listanja (engl. *infinite scrolling*), dodavanje redaka u tablici ili čak prilikom dodavanja poruka u aplikaciji za razmjenu trenutnih poruka. Rezultati u tablici 2.5 predstavljaju rezultate bez zagrijavanja, a rezultati u tablici 2.6 sa zagrijavanjem. Angular je ponovno po prosječnoj vrijednosti na prvom mjestu u oba ispitivanja, Vue je na drugom mjestu i React na trećem, s time da je Reactova standardna devijacija ponovno mnogo veća od ostalih. React u ispitivanju sa zagrijavanjem ponovno pokazuje velik raspon trajanja, njegova maksimalna vrijednost je tri puta veća od maksimalnih vrijednosti ostalih knjižnica. Na histogramu na slici 2.6 vidljivo je da React ima najviše stršećih podataka dok su ostale knjižnice konzistentne po performansama. Histogram na slici 2.7 prikazuje kako svaka knjižnica ima po dva vrha te postepen pad frekvencija nakon vrha. U ispitivanjima bez zagrijavanja Vue

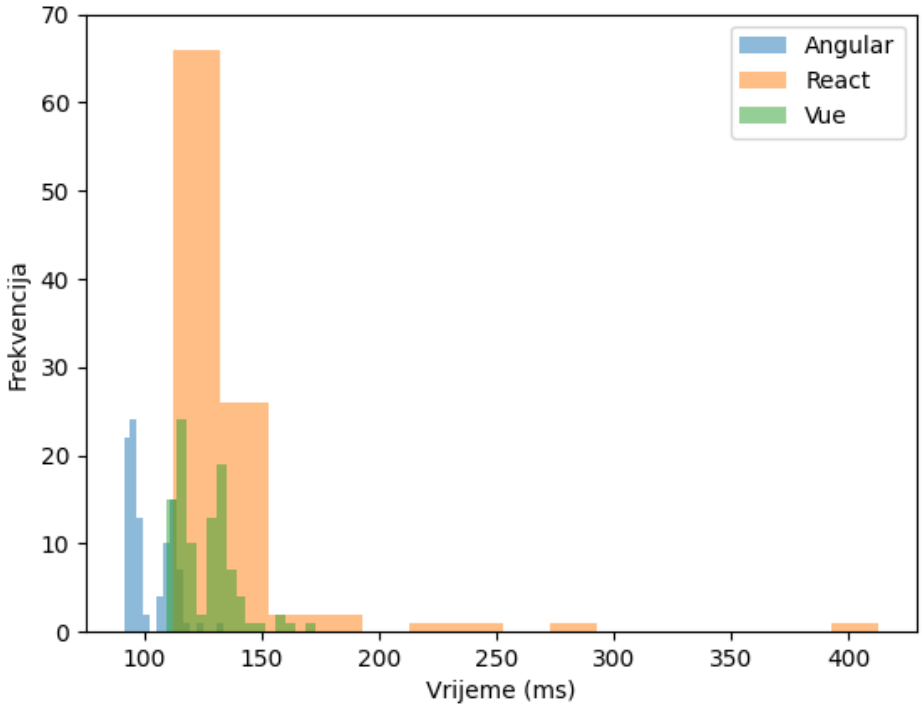
pak pokazuje najveću varijancu i veći broj stršećih podataka slično Reactu u ispitivanju sa zagrijavanjem.

Tablica 2.5 Rezultati bez zagrijavanja (u milisekundama)

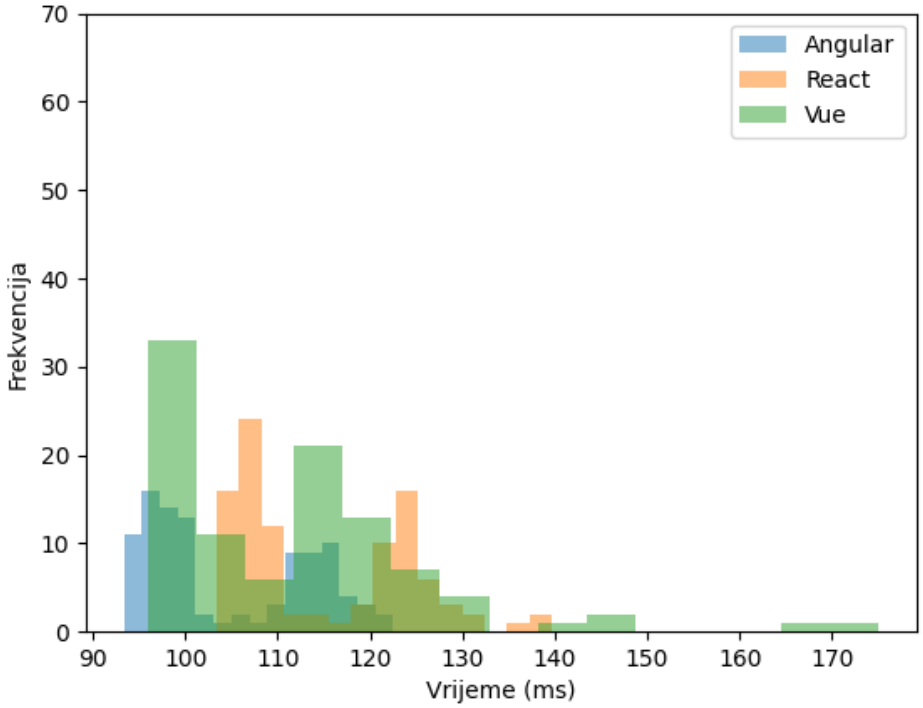
Ime	Prosjek	Standardna devijacija	Min	Max
Angular	102,133	9,065	91,466	133,621
React	133,263	37,887	112,731	413,035
Vue	125,632	12,109	109,685	172,997

Tablica 2.6 Rezultati sa zagrijavanjem (u milisekundama)

Ime	Prosjek	Standardna devijacija	Min	Max
Angular	104,685	8,715	93,336	122,505
React	114,974	9,427	103,390	139,644
Vue	112,000	14,200	95,917	175,087



Slika 2.6 Ispitivanje 3 – Histogram vrijednosti bez zagrijavanja



Slika 2.7 Ispitivanje 3 – Histogram vrijednosti sa zagrijavanjem

2.3.4. Četvrto ispitivanje – Ažuriranje postojećih redaka

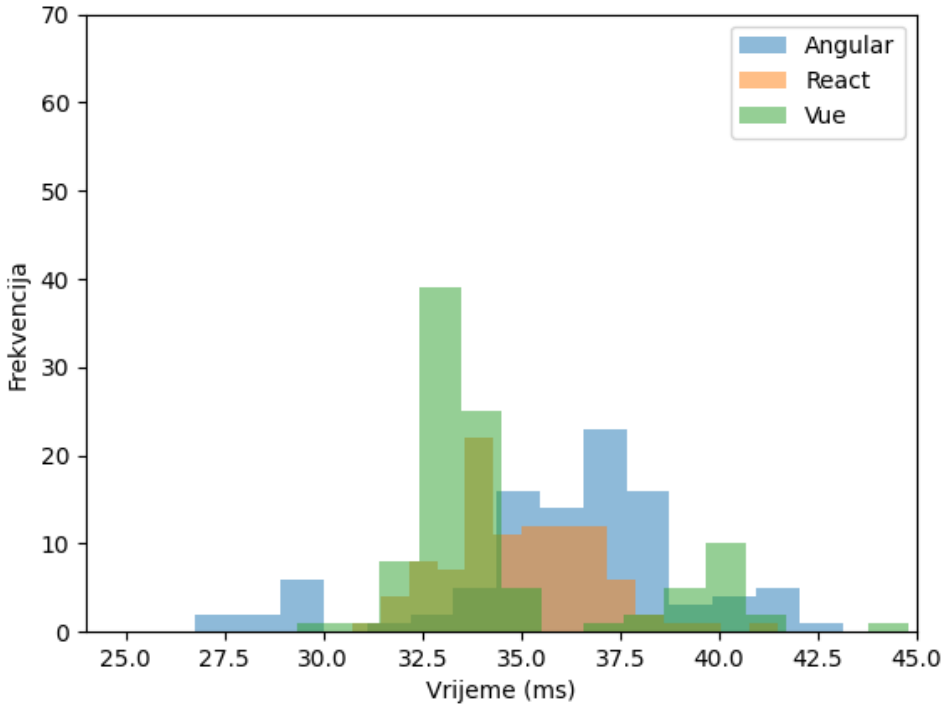
Četvrto ispitivanje sastoji se od kreiranja 1000 redaka te zatim ažuriranjem svakog desetog retka. Vrijeme izvođenja mjeri se od trenutka pritiska na gumb za ažuriranje do prvog događaja iscrtavanja sa svim ažuriranim elementima. U tablici 2.7 nalaze se rezultati ispitivanja bez zagrijavanja. Prosječne performanse knjižnica React i Vue su vrlo slične, React pokazuje konzistentnije rezultate što vidimo iz standardne devijacije, koja je gotovo za milisekundu manja od ostalih knjižnica. Razlika između performansi sve tri knjižnice je minimalna. U tablici 2.8 su rezultati sa zagrijavanjem te je Angular ponovno na prvom mjestu. Vrlo slično rezultatima bez zagrijavanja, razlika je od nekoliko milisekundi između knjižnica. Angularu se najviše prosječno trajanje ažuriranja povećava u testovima bez zagrijavanja. Na histogramu na slici 2.8 vidljiv je razlog povećane standardne devijacije Angulara. Histogram na slici 2.9 prikazuje mnogo konzistentnija vremena uz Vue koji pokazuje dva vrha vremena.

Tablica 2.7 Rezultati bez zagrijavanja (u milisekundama)

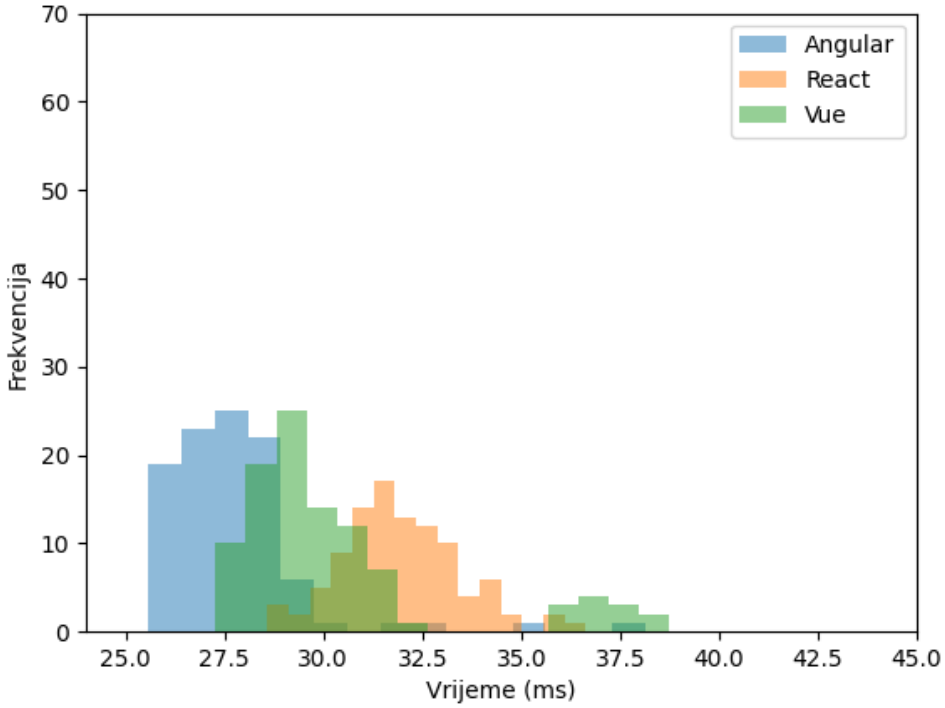
Ime	Prosjek	Standardna devijacija	Min	Max
Angular	36,094	3,202	26,736	43,116
React	34,997	1,857	30,723	41,469
Vue	34,606	2,905	29,344	44,797

Tablica 2.8 Rezultati sa zagrijavanjem (u milisekundama)

Ime	Prosjek	Standardna devijacija	Min	Max
Angular	27,759	1,773	25,583	38,126
React	31,943	1,508	28,587	36,626
Vue	30,357	2,751	27,278	38,754



Slika 2.8 Ispitivanje 4 – Histogram vrijednosti bez zagrijavanja



Slika 2.9 Ispitivanje 4 – Histogram vrijednosti sa zagrijavanjem

2.3.5. Peto ispitivanje – Zamjena redaka

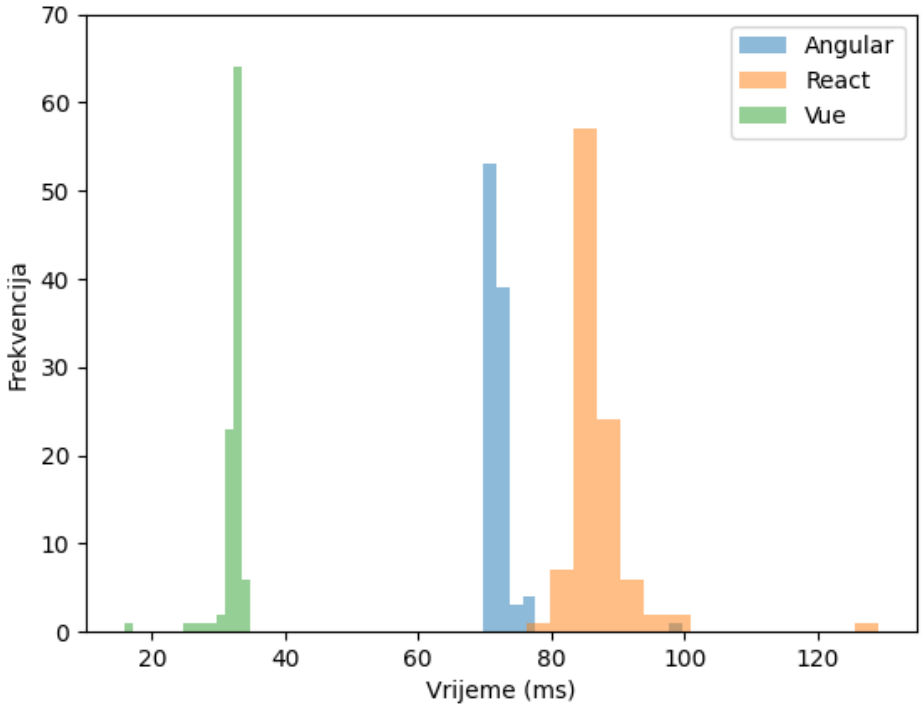
U petom ispitivanju zamjenjuju se drugi i predzadnji redak tablice. U prošlom ispitivanju, samo se sadržaj ažurirao, u ovom ispitivanju se cijeli element zamjenjuje. U tablici 2.9 i tablici 2.10 daleko najbolje rezultate ima Vue čiji se prosjek kreće u intervalu od 20 milisekundi do 32 milisekunde. Vueov prosjek je tri puta manji od ostalih knjižnica zajedno s vrlo malom devijacijom u oba ispitivanja, što pokazuje optimiranost operacije zamjene elementa. Angular i React imaju vrlo slične rezultate s time da React u ispitivanjima sa zagrijavanjem postane brži, a Angular uspori. Na histogramu na slici 2.10 vidljivo je kako svaka knjižnica ima vrlo konzistentna vremena s malim varijacijama. Na slici 2.11 slično kao na slici 2.10 postoje vrhovi oko prosječnog trajanja, međutim manje su izraženi i postoji više devijacije.

Tablica 2.9 Rezultati bez zagrijavanja (u milisekundama)

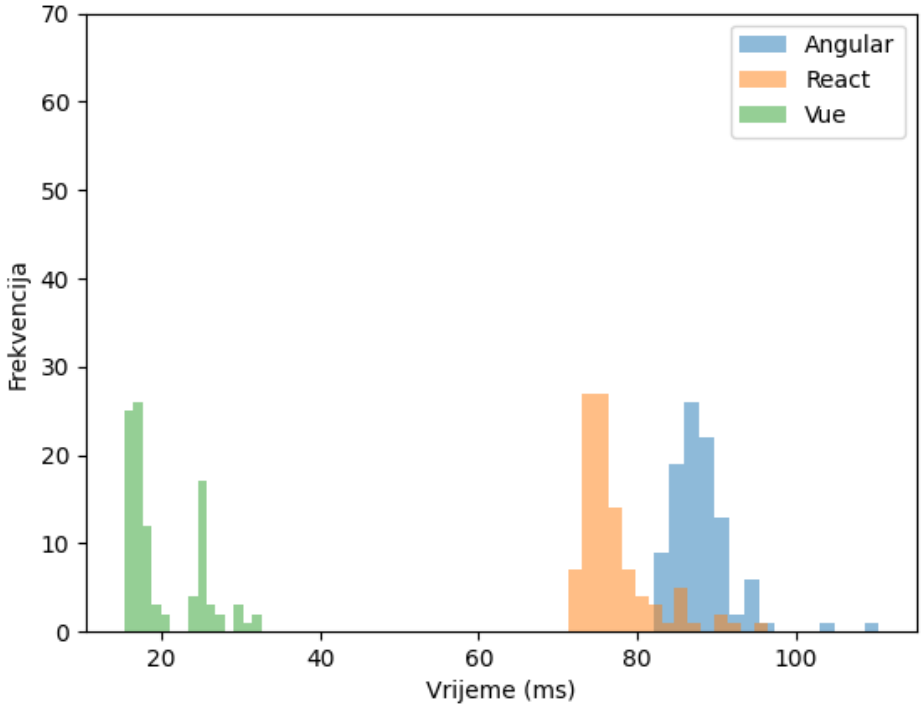
Ime	Prosjek	Standardna devijacija	Min	Max
Angular	72,054	3,110	69,651	99,757
React	87,103	5,340	76,323	129,286
Vue	32,217	2,106	15,752	34,821

Tablica 2.10 Rezultati sa zagrijavanjem (u milisekundama)

Ime	Prosjek	Standardna devijacija	Min	Max
Angular	88,039	4,016	82,133	110,481
React	77,200	4,587	71,326	96,442
Vue	19,948	4,615	15,260	32,667



Slika 2.10 Ispitivanje 5 – Histogram vrijednosti bez zagrijavanja



Slika 2.11 Ispitivanje 5 – Histogram vrijednosti sa zagrijavanjem

2.3.6. Šesto ispitivanje – Brisanje redaka

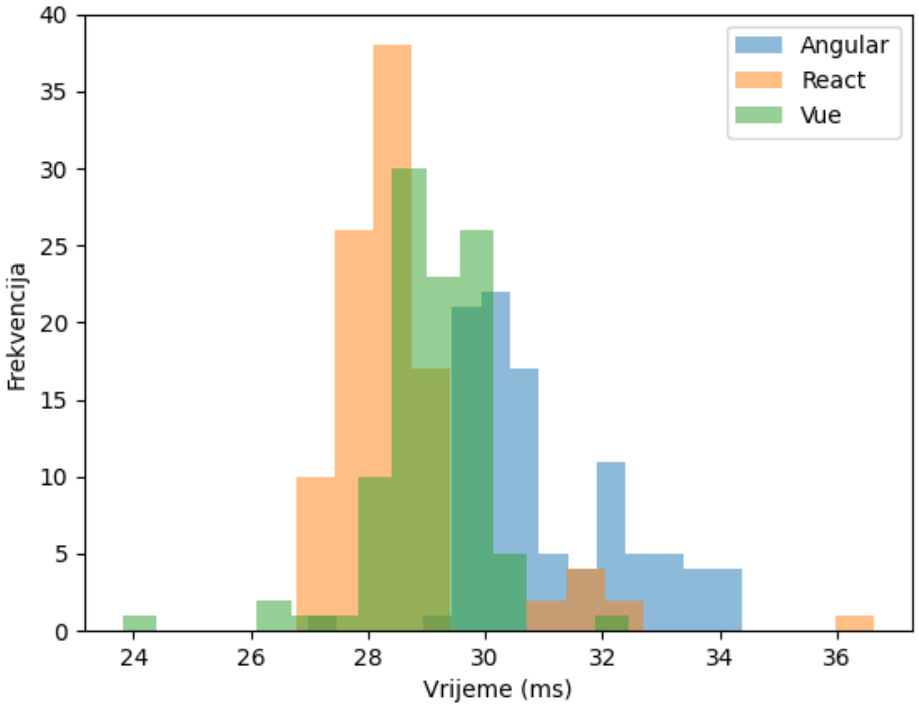
U šestom ispitivanju odabire se redak koji se briše i mjeri se vrijeme potrebno za njegovo brisanje. U tablici 2.1 prikazani su rezultati brisanja redaka bez zagrijavanja gdje React ima najbolja vremena, ali razlika brzina brisanja redaka između knjižnica je zanemariva s obzirom na to da su unutar par milisekundi razlike. Devijacija je vrlo mala kod sve tri knjižnice, s Vueom koji ima najmanju devijaciju. U tablici 2.12 prikazani su rezultati ispitivanja sa zagrijavanjem gdje Angular ima najbolje rezultate s 3 milisekunde manje u prosjeku, ali s najvećom devijacijom. Vue i React imaju vrlo slične rezultate, s time da React ima manje minimalno vrijeme, ali i veću devijaciju. Na histogramu na slici 2.12 vidljivo je kako React ima najviši vrh što upućuje na najveći broj brisanja sa sličnim vremenima, nakon njega je Vue, a posljednji je Angular. Na histogramu na slici 2.13 vidljivo je kako Angular ima najveći vrh, ali i najveću raširenost vremena.

Tablica 2.11 Rezultati bez zagrijavanja (u milisekundama)

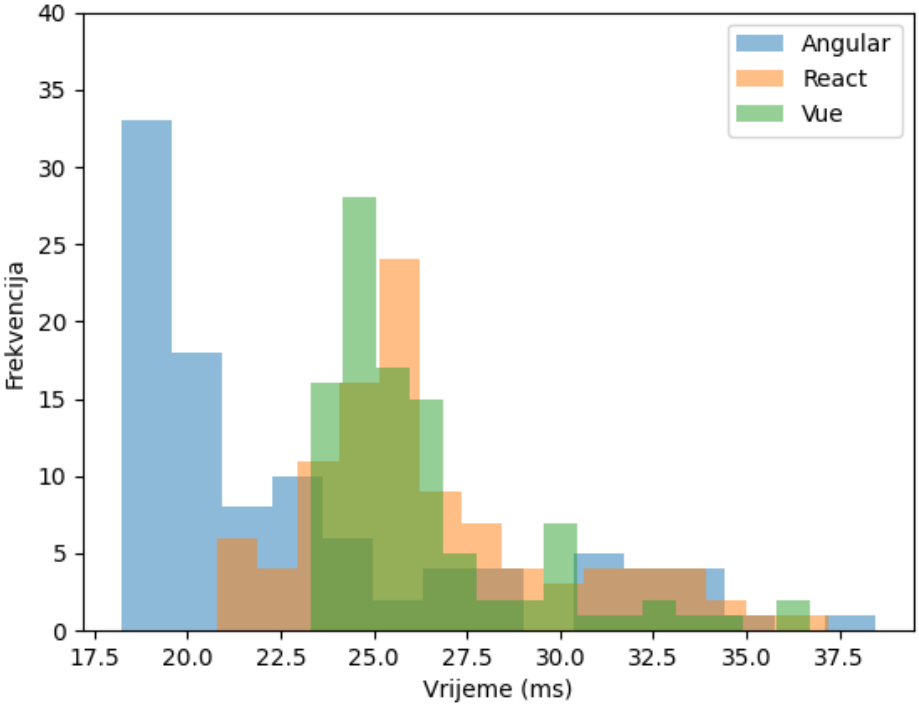
Ime	Prosjek	Standardna devijacija	Min	Max
Angular	31,019	1,393	26,971	34,371
React	28,584	1,339	26,771	36,646
Vue	29,094	0,984	23,806	32,443

Tablica 2.12 Rezultati sa zagrijavanjem (u milisekundama)

Ime	Prosjek	Standardna devijacija	Min	Max
Angular	23,054	4,899	18,214	38,477
React	26,551	3,471	20,777	37,208
Vue	26,290	2,762	23,295	36,687



Slika 2.12 Ispitivanje 6 – Histogram vrijednosti bez zagrijavanja



Slika 2.13 Ispitivanje 6 – Histogram vrijednosti sa zagrijavanjem

2.3.7. Sedmo ispitivanje – brisanje svih redaka

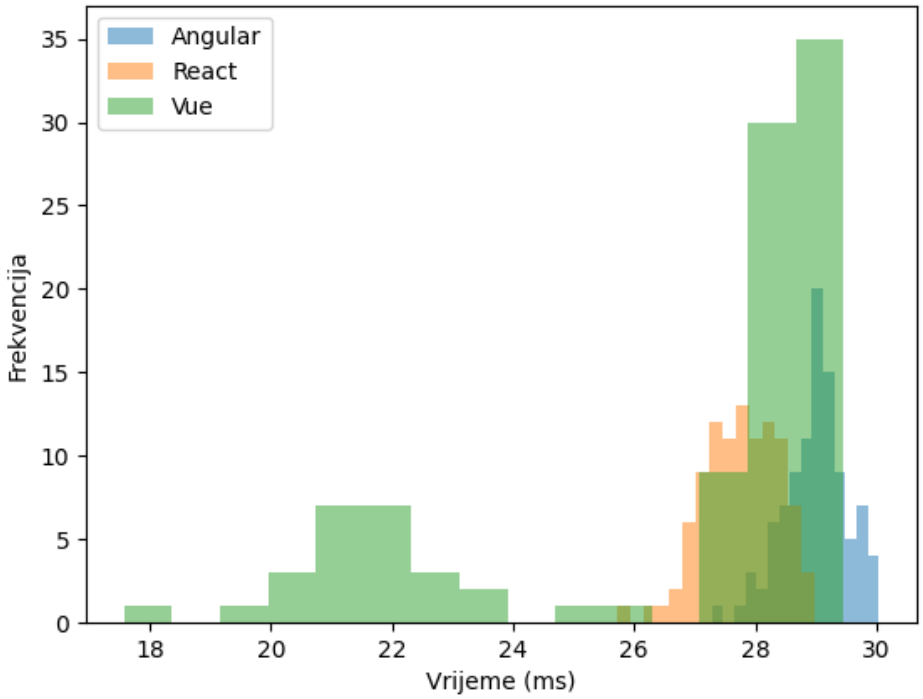
Sedmo ispitivanje sastoji se od brisanja svih trenutačnih redaka na stranici se kreira 1000 redaka, a zatim se mjeri potrebno vrijeme nakon pritiska na gumb za brisanje svih redaka. U tablici 2.13 prikazani su rezultati bez zagrijavanja, gdje sve tri knjižnice imaju vrlo slična vremena, oko dvije milisekunde razlike između najbrže i najsporije knjižnice. Angular i React imaju vrlo malu devijaciju dok Vue ima gotovo šest puta veću, što je vidljivo i na histogramu na slici 2.14. U tablici 2.14 su rezultati ispitivanja sa zagrijavanjem, gdje Vue ima najbolje rezultate, ali i najveću devijaciju. Angular i React imaju slične devijacije što je također vidljivo na histogramu na slici 2.15.

Tablica 2.13 Rezultati bez zagrijavanja (u milisekundama)

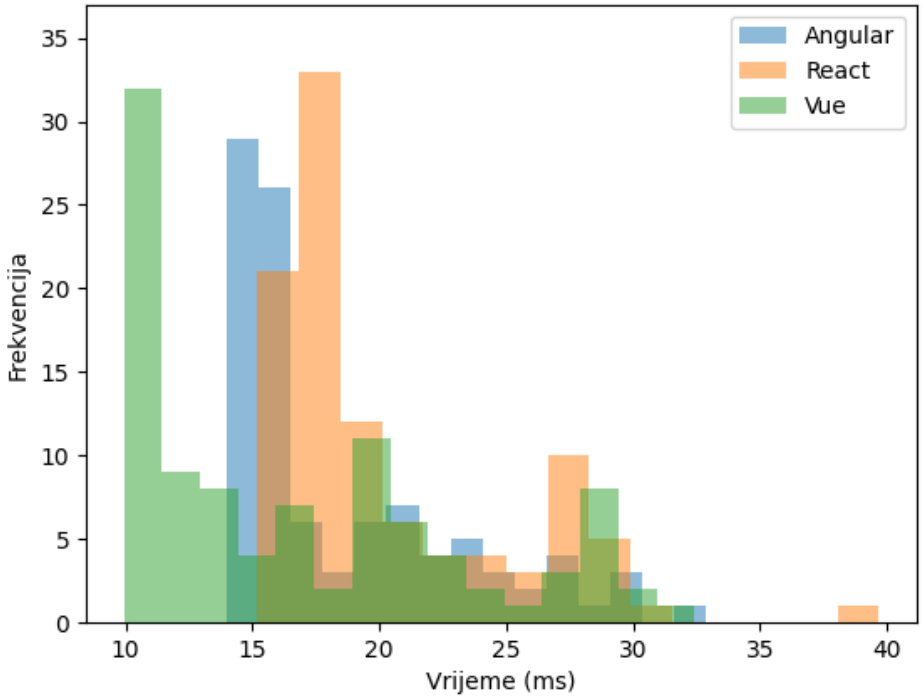
Ime	Prosjek	Standardna devijacija	Min	Max
Angular	28,987	0,524	27,279	30,042
React	27,774	0,615	25,716	28,978
Vue	26,795	3,145	17,571	29,459

Tablica 2.14 Rezultati sa zagrijavanjem (u milisekundama)

Ime	Prosjek	Standardna devijacija	Min	Max
Angular	18,495	4,501	13,984	32,892
React	20,215	4,729	15,195	39,697
Vue	16,868	6,396	9,927	32,426



Slika 2.14 Ispitivanje 7 – Histogram vrijednosti bez zagrijavanja



Slika 2.15 Ispitivanje 7 – Histogram vrijednosti sa zagrijavanjem

2.4. Analiza korištenja memorije

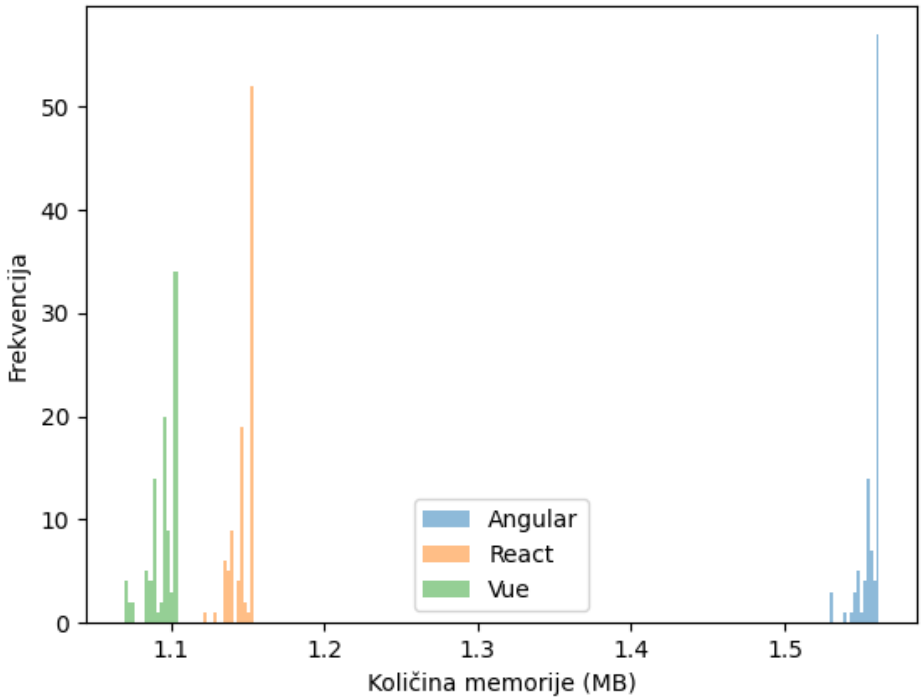
Računanje korištenja memorije radi se pomoću JavaScriptove funkcije ***measureUserAgentSpecificMemory*()** objekta ***performance*** koji je dostupan u svim novijim web preglednicima. Metoda se koristi za izračun koliko memorije koristi web aplikacija zajedno sa svojim *iframeovima* i radnim nitima (eng. *workers*). Rezultat izvršavanja metode je broj koji predstavlja količinu memorije koju web aplikacija koristi, a izražava se u bajtovima. Metoda se koristi kako bi se pronašlo potencijalno curenje memorije i za općenito testiranje potrošnje memorije. Vrijednosti koje funkcija vraća su ovisne o vrsti preglednika, pa čak i verziji istog preglednika, jer su različite implementacije u različitim verzijama.

2.4.1. Prvo ispitivanje – dodavanje 1000 redaka

Prvo ispitivanje odnosi se na potrošnju memorije nakon kreiranja 1000 redaka stranici. Provedeno je 100 iteracija ispitivanja. U tablici 2.15 vidljivi su rezultati potrošnje memorije nakon dodavanja 1000 redaka. Vue ima najmanju potrošnju memorije, samo 1,1 megabajt potrošnje, zatim React s 50 kilobajta više, a Angular je posljednji s 1,56 megabajta. Devijacija sve tri knjižnice je minimalna i gotovo identična što možemo vidjeti i na histogramu na slici 2.16 gdje za svaku knjižnicu postoji jedan vrh s najvišim iznosom te ostatak vrijednosti lijevo od tog vrha.

Tablica 2.15 Rezultati korištenja memorije (u megabajtima)

Ime	Prosjek	Standardna devijacija	Min	Max
Angular	1,557	0,007	1,530	1,562
React	1,148	0,007	1,121	1,154
Vue	1,095	0,009	1,069	1,104



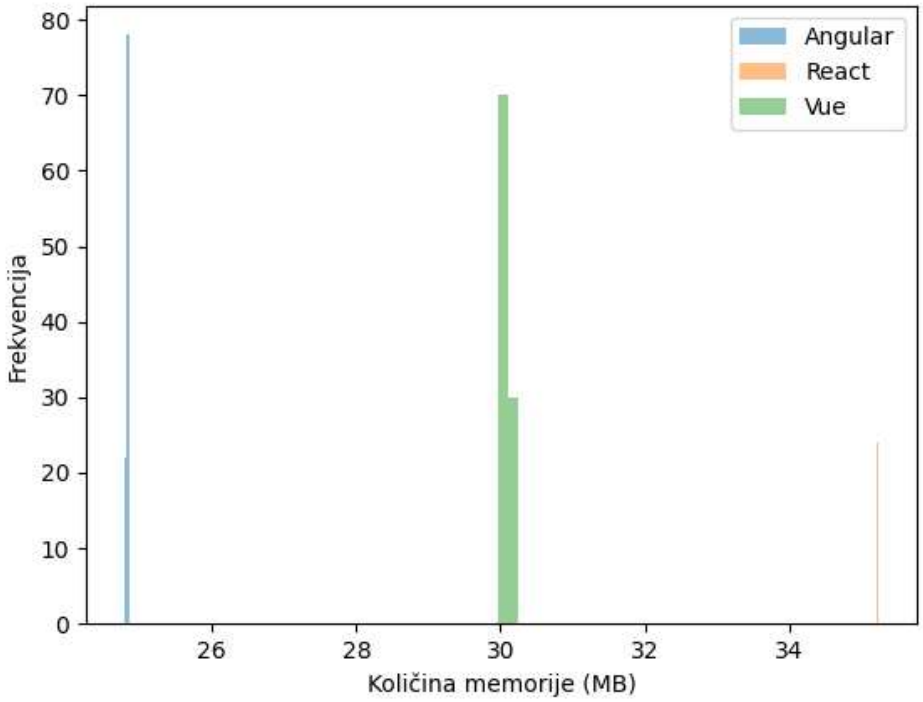
Slika 2.16 Ispitivanje 1 – Histogram vrijednosti korištenja memorije

2.4.2. Drugo ispitivanje – Dodavanje 10000 redaka

U drugom ispitivanju povećao se broj kreiranih redaka pa je očekivan i porast potrošnje memorije za barem deset puta. U tablici 2.16 vidljivi su rezultati drugog ispitivanja. Na prvom mjestu je Angular s 24,8 megabajta potrošnje, zatim je Vue pa React. Razlika između knjižnica je oko 5 megabajta. Najmanje povećanje memorije ima Angular – 16 puta, zatim Vue s 27 puta i konačno React s čak 31 puta većom potrošnjom memorije. Očekivano je linearno povećanje memorije otprilike 10 puta, međutim, rezultati ukazuju na nelinearno povećanje. Devijacija je ostala vrlo mala i to je vidljivo na histogramu na slici 2.17 gdje sve tri knjižnice imaju svaka jedan vrh.

Tablica 2.16 Rezultati korištenja memorije (u megabajtima)

Ime	Prosjek	Standardna devijacija	Min	Max
Angular	24,824	0,014	24,777	24,852
React	35,233	0,010	35,205	35,246
Vue	30,062	0,098	29,965	30,251



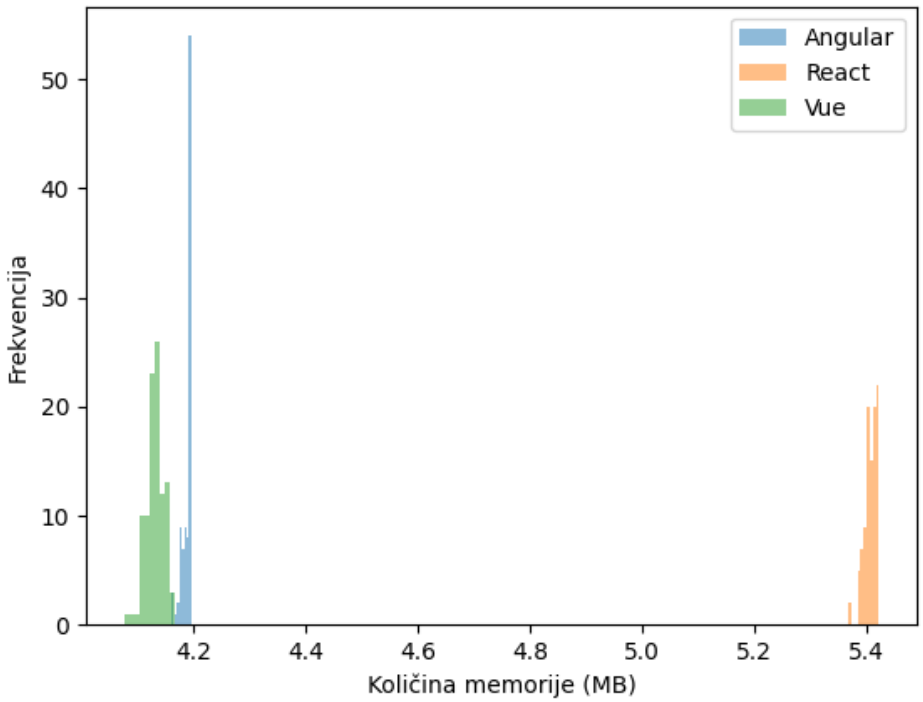
Slika 2.17 Ispitivanje 2 – Histogram vrijednosti korištenja memorije

2.4.3. Treće ispitivanje – Ažuriranje postojećih redaka

U trećem ispitivanju kreirano je 1000 redaka te se zatim pet puta izvršavalo ažuriranje svakog desetog retka s novim sadržajem. U tablici 2.17 vidljivi su rezultati ispitivanja. Vue ima najmanju potrošnju memorije, Angular je vrlo blizu s 50 kilobajta više, a React ima daleko veću potrošnju memorije, s 5,4 megabajta. Na histogramu na slici 2.18 vidljivo je kako su Vue i Angular vrlo blizu po vrijednostima, ali i kako Vue ima konzistentno manju potrošnju memorije.

Tablica 2.17 Rezultati korištenja memorije (u megabajtima)

Ime	Prosjek	Standardna devijacija	Min	Max
Angular	4,189	0,008	4,161	4,196
React	5,407	0,011	5,368	5,423
Vue	4,132	0,016	4,077	4,168



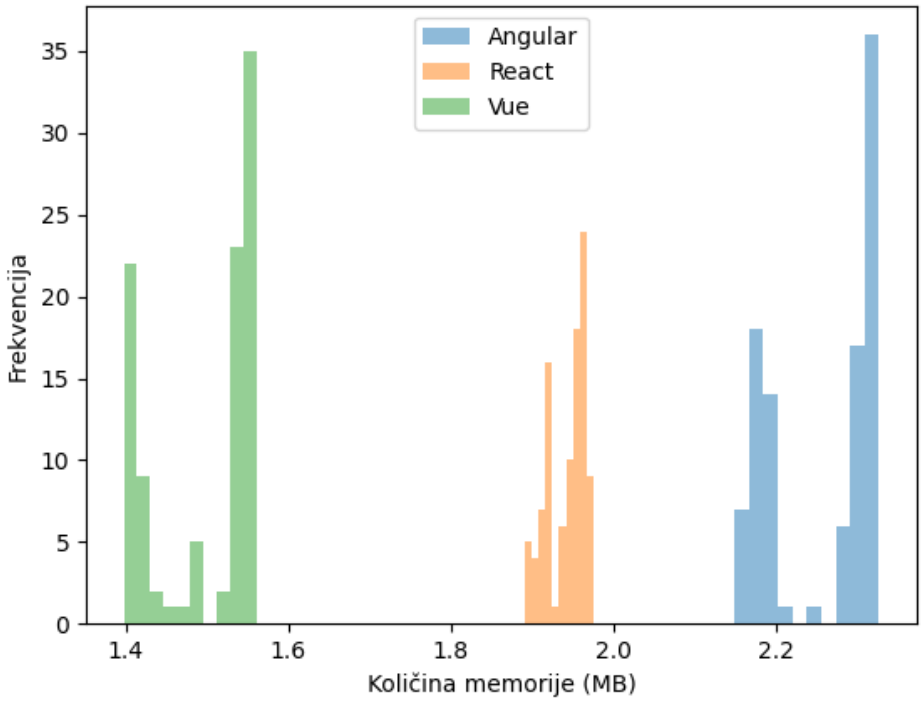
Slika 2.18 Ispitivanje 3 – Histogram vrijednosti korištenja memorije

2.4.4. Četvrto ispitivanje – brisanje svih redaka

Četvrto ispitivanje sastoji se od kreiranja 1000 redaka na stranici te zatim izvršavanja brisanja svih novokreiranih redaka i ponavljanja tog istog postupka pet puta. Tablica 2.18 sadrži rezultate četvrtog ispitivanja iz kojih je vidljivo kako Vue ponovno ima najmanju potrošnju memorije, nakon njega slijedi React, a na zadnjem mjestu je Angular. Devijacija je vrlo mala, što je ponovno vidljivo na histogramu na slici 2.19.

Tablica 2.18 Rezultati korištenja memorije (u megabajtima)

Ime	Prosjek	Standardna devijacija	Min	Max
Angular	2,257	0,064	2,149	2,326
React	1,942	0,024	1,890	1,976
Vue	1,497	0,062	1,397	1,560



Slika 2.19 Ispitivanje 4 – Histogram vrijednosti korištenja memorije

2.5. Analiza rezultata knjižnice Lighthouse

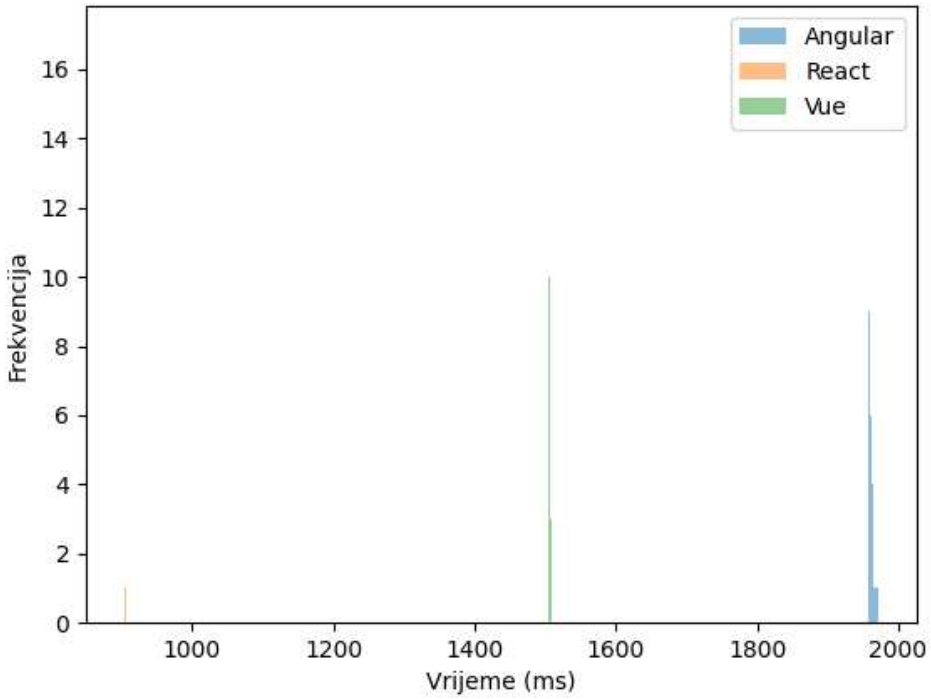
Ispitivanja koja su izvršena pomoću knjižnice Lighthouse su: mjerenje vremena potrebnog da se sve JavaScriptove skripte i CSS učitaju na web stranicu i da stranica postane interaktivna i ispitivanje koje mjeri ukupnu količinu podataka potrebnu za učitavanje i rad stranice koja koristi određenu knjižnicu.

2.5.1. Prvo ispitivanje – konzistentna interaktivnost

Prvo ispitivanje je vrijeme potrebno za konzistentnu interaktivnost web stranice. Provedeno je 100 iteracija ispitivanja te su rezultati prikazani u tablici 2.19 i na histogramu na slici 2.20. Daleko najbrže vrijeme do interaktivnosti ima React, koji u prosjeku nakon 900 milisekundi postane interaktivan i pokazuje najmanju devijaciju. Nakon njega je Vue s 1500 milisekundi, a posljednji je Angular s 1950 milisekundi. Zbog velikog raspona vrijednosti na histogramu su neki stupci gotovo nevidljivi jer im je širina manja od jednog piksela.

Tablica 2.19 Rezultati vremena potrebnog za konzistentnu interaktivnost

Ime	Prosjek	Standardna devijacija	Min	Max
Angular	1961,313	2,815	1957,304	1972,771
React	904,342	0,756	902,948	906,206
Vue	1505,861	1,144	1504,098	1511,222



Slika 2.20 Ispitivanje 1 – Histogram vrijednosti vremena interaktivnosti

2.5.2. Drugo ispitivanje – veličina

Drugo ispitivanje odnosi se na količinu podataka koje je potrebno prenijeti od poslužitelja do klijenta kako bi web stranica bila funkcionalna. Mjerenje se izvršavalo uz pomoć alata Lighthouse, a podaci su dobiveni iz metrike *total-byte-weight*. U tablici 2.20 vidljive su veličine konačnih datoteka svake od knjižnica. Vue ima gotovo dva puta manju veličinu od Reacta i Angulara čije veličine su vrlo bliske.

Tablica 2.20 Veličina knjižnica u kilobajtovima

Ime	Veličina
Angular	142,671
React	151,733
Vue	83,6865

3. Rasprava i sažetak rezultata

U prvom poglavlju provedena je kvalitativna analiza i predstavljene su sve tri knjižnice s njihovim najistaknutijim značajkama. U drugom poglavlju opisan je alat razvijen za mjerenje performansi i prezentirani su rezultati ispitivanja. U tablici 3.1 nalaze se sažeti rezultati kvalitativne i kvantitativne analize, a tablica 3.2 sadrži knjižnice rangirane po rezultatima ispitivanja sa zadnjim retkom koji predstavlja ukupni rezultat za svaku knjižnicu (manje je bolje).

Tablica 3.1 Sažeti rezultati obje analize

Značajke	Angular	React	Vue
Tip	Radni okvir	Knjižnica	Radni okvir
Teškoća učenja	Teško	Srednje	Lagano
Model DOM	Inkrementalni DOM	Virtualni DOM	Virtualni DOM
Povezivanje podataka	Dvosmjerno, jednosmjerno (opcionalno)	Jednosmjerno	Dvosmjerno, jednosmjerno (opcionalno)
Temeljena na komponentama	Da	Da	Da
Jezik	TypeScript	JavaScript (JSX), TypeScript	JavaScript, TypeScript
Popularnost	Stabilna	Velika	Rastuća
Podrška zajednice	Velik	Velika	Velika
Performanse	Brz	Brz	Najbrži
Dokumentacija	Dobra	Dobra	Odlična

Tablica 3.2 Rangirane knjižnice po rezultatima ispitivanja

Ispitivanje br.	Vrsta ispitivanja	Zagrijavanje	Angular	React	Vue
1	Procesorsko vrijeme	Da	1.	3.	2.
1	Procesorsko vrijeme	Ne	1.	2.	3.
2	Procesorsko vrijeme	Ne	1.	3.	2.
2	Procesorsko vrijeme	Da	1.	3.	2.
3	Procesorsko vrijeme	Ne	1.	3.	2.
3	Procesorsko vrijeme	Da	1.	3.	2.
4	Procesorsko vrijeme	Ne	3.	2.	1.
4	Procesorsko vrijeme	Da	1.	3.	2.
5	Procesorsko vrijeme	Ne	2.	3.	1.
5	Procesorsko vrijeme	Da	3.	2.	1.
6	Procesorsko vrijeme	Ne	3.	1.	2.
6	Procesorsko vrijeme	Da	1.	3.	2.
7	Procesorsko vrijeme	Ne	3.	2.	1.
7	Procesorsko vrijeme	Da	2.	3.	1.
8	Potrošnja memorije	Ne	3.	2.	1.
9	Potrošnja memorije	Ne	1.	3.	2.
10	Potrošnja memorije	Ne	2.	3.	1.
11	Potrošnja memorije	Ne	3.	2.	1.
12	Interaktivnost	Ne	3.	1.	2.
13	Veličina	Ne	1.	2.	3.
	Ukupno		37	49	34

Sveukupno gledajući, Vue se čini najbržim radnim okvirom u većini ispitivanja, posebno kada se radi o zamjeni redaka i brisanju redaka, a koristi i najmanje resursa. Angular ima tendenciju biti najsporiji, ali iznenađujuće ima najbolje performanse prilikom dodavanja redaka. React se uglavnom nalazi negdje između, osim u vremenu potrebnom za interaktivnost, ali često ima najveće vrijeme izvršavanja, posebno prilikom dodavanja velikog broja redaka. Međutim, valja napomenuti da se performanse mogu razlikovati ovisno o specifičnim uvjetima i zahtjevima projekta. Sva ova ispitivanja su barem 10 – 100 puta veće opterećenje od očekivanog za knjižnice i gotovo nikada se neće dogoditi takvo opterećenje u stvarnom svijetu. Radovi poput [5] i [6] provodili su vrlo slična ispitivanja s kreiranjem, brisanjem i ažuriranjem elemenata te rezultati koji su dobiveni su u skladu s rezultatima prikazanim u ovom radu. React ima daleko najviše vrijeme kreiranja elemenata, a Vue je najkonzistentniji u svim testovima. React, međutim, u dobivenim rezultatima pokazuje lošije performanse u odnosu na rezultate dobivene u spomenutim radovima. Jedan od mogućih razloga tome je način na koji je implementirana aplikacija za ispitivanje, koji moguće da je suboptimalan, ali je bio potreban radi konzistentnosti funkcionalnosti. Rad [5] je također proveo analizu veličine generiranih datoteka, međutim njihovi rezultati su različiti od onih dobivenih u ovom radu najvjerojatnije zbog različite implementacije i moguće korištenje različite strategije prevođenja. Sveukupno rezultati ispitivanja slažu se s rezultatima spomenutih radova te zaključkom da je Vue generalno najbrža knjižnica.

4. Zaključak

Odabir knjižnice za projekt ovisi o nizu čimbenika, uključujući specifične potrebe i zahtjeve projekta, preferencije tima i znanje, kao i dugoročne planove za održavanje i skaliranje projekta. Svaka od analiziranih knjižnica ima svoje prednosti i mane, a izbor najviše ovisi o tome što je najprikladnije za određeni kontekst. Korištenje knjižnice Vue preporuča se kada je potrebna velika količina ažuriranja elemenata DOM-a, knjižnice React kada je interaktivnost bitna i Angulara kada je potrebna velika količina kreiranja i brisanja elemenata DOM-a te za razvoj aplikacija na poslovnoj razini.

Literatura

[1] React developers, Thinking in React, https://react.dev/learn/thinking-in-react , pristupljeno 4.6.2023.

[2] Stack Overflow, Stack Overflow Trends, https://insights.stackoverflow.com/trends?tags=reactjs%2Cvue.js%2Cangular%2Cangularjs%2Cvuejs3, pristupljeno 10.6.2023.

[3] Angular developers, Introduction to Angular concepts, https://angular.io/guide/architecture, pristupljeno 1.6.2023.

[4] Vue developers, Vue develeopers guide, https://vuejs.org/guide/introduction.html, pristupljeno 28.5.2023.

[5] R. N. V. Diniz-Junior *et al*., "Evaluating the performance of web rendering technologies based on JavaScript: Angular, React, and Vue," *2022 XVLIII Latin American Computer Conference (CLEI)*, Armenia, Colombia, 2022, pp. 1-9, doi: 10.1109/CLEI56649.2022.9959901.

[6] Hutagikar, V., & Hegde, V. (2020). Analysis of Front-end Frameworks for Web Applications. *Int. Research J. of Engineering and Tech*, *7*(4), 3317-3320.

Sažetak

Naslov: Usporedba JavaScriptovih knjižnica React, Vue i Angular za razvoj na strani klijenta

Sažetak: U ovom radu provedene su kvalitativna i kvantitativna analiza JavaScriptovih knjižnica Angular, React i Vue. Za svaku knjižnicu predstavljene su ključne značajke te analizirana je njihova popularnost, dokumentacija i lakoća učenja. Za kvantitativnu analizu, uspješno je razvijen alat za ispitivanje performansi, korištenja memorije i mrežnog prometa, a predstavljeni su i analizirani rezultati za svaku knjižnicu. U rezultatima gledajući performanse prvi je Vue, zatim Angular i zadnji je React. Vue pokazuje vrlo visok stupanj konzistentnosti, dok Angular i React pokazuju visoke performanse u specifičnim područjima. Alat za testiranje performansi razvijen je tako da je moguće analizirati performanse bilo koje postojeće tehnologije za razvoj na strani klijenta.

Ključne riječi: React, Angular, Vue, razvoj na strani klijenta, performanse, analiza

Summary

Title: Comparison of React, Vue and Angular JavaScript libraries for frontend development

Summary: In this work, a qualitative and quantitative analysis of the JavaScript libraries Angular, React, and Vue is conducted. Key features for each library are presented, and their popularity, documentation, and ease of learning are analyzed. For the quantitative analysis, a tool was successfully developed for testing performance, memory usage, and network traffic, and the results for each library are presented and analyzed. The results show that Vue is ranked first by performance, Angular is ranked second, and the last one is React. Vue has shown highly consistent high performance results, while Angular and React show high performance in specific fields. The tool was developed in such a way that it is possible to analyze the performance of any existing client-side development technology.

Keywords: React, Angular, Vue, frontend development, performance, analysis