# USPOREDBA REACT, ANGULAR I SVELTE RADNIH OKVIRA ZA RAZVOJ WEB APLIKACIJA

**Dedić, Marin**

SVEUČILIŠTE U RIJECI
**TEHNIČKI FAKULTET**
Prijediplomski studij računarstva

Završni rad

# Usporedba React, Angular i Svelte radnih okvira za razvoj web aplikacija

Rijeka, rujan 2023.

Marin Dedić
0069090550

SVEUČILIŠTE U RIJECI
**TEHNIČKI FAKULTET**
Prijediplomski studij računarstva

Završni rad

# Usporedba React, Angular i Svelte radnih okvira za razvoj web aplikacija

Mentor: doc. dr. sc. Marko Gulić

Rijeka, rujan 2023.

Marin Dedić
0069090550

**SVEUČILIŠTE U RIJECI**
**TEHNIČKI FAKULTET**
POVJERENSTVO ZA ZAVRŠNE ISPITE

Rijeka, 8. ožujka 2023.

Zavod: **Zavod za računarstvo**
Predmet: **Razvoj web aplikacija**
Grana: **2.09.06 programsko inženjerstvo**

# ZADATAK ZA ZAVRŠNI RAD

Pristupnik: **Marin Dedić (0069090550)**
Studij: Sveučilišni prijediplomski studij računarstva

Zadatak: **Usporedba React, Angular i Svelte radnih okvira za razvoj web aplikacija / The comparison of React, Angular and Svelte frameworks for web application development**

Opis zadatka:

Treba napraviti detaljnu usporedbu React, Angular i Svelte radnih okvira klijentske strane koji se koriste za razvoj web aplikacija. Izvršit će se usporedba ta 3 radna okvira pomoću tri identične jednostranične aplikacije (Single Page Application, SPA) koje će biti izrađene s ta tri zadana okvira. Aplikacije trebaju sadržavati funkcionalnost rješavanja određenih optimizacijskih funkcija pomoću odabranog prirodom inspiriranog algoritma kako bi se maksimalno opteretilo resurse dostupne za rad SPA aplikacije na klijentskoj strani. Treba analizirati arhitekturu, načine razvoja web aplikacija, kao i performanse te potrošnju resursa ova tri radna okvira.

Rad mora biti napisan prema Uputama za pisanje diplomskih / završnih radova koje su objavljene na mrežnim stranicama studija.

Zadatak uručen pristupniku: 26. ožujka 2023.

Mentor:

Doc. dr. sc. Marko Gulić

Predsjednik povjerenstva za završni ispit:

Prof. dr. sc. Miroslav Joler

## Izjava o samostalnoj izradi rada

Izjavljujem da sam samostalno izradio ovaj rad.

Rijeka, rujan 2023.

Marin Dedić

# Zahvala

Zahvaljujem mentoru na podršci tijekom pisanja ovoga rada i korisnim raspravama i savjetima.

# Sadržaj

# Poglavlje 1

# Uvod

U suvremenom digitalnom dobu, optimizacija i učinkovitost softverskih rješenja postali su ključni kako i za performanse i korisničko iskustvo,tako i za održivost i ekološku odgovornost. Kako se sve više pažnje pridaje ekološkom otisku tehnologije, važno je promotriti kako različite tehnologije i tehnološke metode utječu na potrošnju energije.

Ovaj rad fokusira se na analizu i usporedbu tri vodeća frontend okvira (frameworka) - React (iniciran pomoću CRA i Vitea), Angular i Svelte, s posebnim naglaskom na njihovu potrošnju električne energije prilikom izvršavanja kompleksnog algoritma i operacija koje opterećuju korisničko sučelje. Odabrani algoritam je genetski algoritam koji je poznat po svojoj složenosti i računalnoj zahtjevnosti te samim time predstavlja idealnu testnu okolinu za ovu analizu.

Cilj ovog rada je pružiti uvid u performanse i energetsku učinkovitost svakog od ovih radnih okvira, te istražiti kako specifične karakteristike i načini implementacije svakog okvira utječu na ukupnu potrošnju energije. Kroz analizu i mjerne testove, ovaj rad će pokušati dati odgovore na pitanje koji okvir je najviše eko-prijateljski.

# Poglavlje 2

# Korištene tehnologije

## 2.1 React

React.js [1] je knjižnica otvorenog koda za izgradnju korisničkih sučelja na komponentski orijentiran način. Omogućuje programerima razvijanje ponovno upotrebljivih UI (user interface) komponenti, upravljanje stanjem tih komponenti te dinamičko renderiranje ovisno o stanjima. Iako danas postoji mnogo bržih knjižnica, React nosi titulu najpopularnije knjižnice za izgradnju korisničkih sučelja te je zaslužan za revoluciju načina izgradnje web aplikacija. React.js objavljen je 2013. godine od strane Mete (bivši Facebook). Razvijen je s namjerom rješavanja izazova s kojima se tvrtka susretala pri održavanju visokih performansi korisničkog sučelja. Glavna ideja bila je uvođenje virtualnog DOM-a (Document Object Model), koji učinkovito ažurira samo potrebne dijelove korisničkog sučelja prilikom promjena, smanjujući opterećenje renderiranja i poboljšavajući performanse. Danas se Virtualni DOM smatra „overheadom“ (potrošnja resursa na zadatke koji ne doprinose glavnom cilju/zadatku) te ga većina novih knjižnica izbjegava koristiti.

Virtualni DOM radi na način da sprema lagani objekt stvarnog DOM-a (u obliku

stabla) te na temelju njega računa najmanji skup čvorova djece unutar stabla koje je potrebno ažurirati. Ta tehnika naziva se *diffing*. Ova tehnika čini React iznimno učinkovitim zbog toga što smanjuje ukupan broj skupih izmjena DOM-a. React.js slijedi koncept arhitekture zasnovane na komponentama, gdje se korisničko sučelje dijeli na samostalne komponente. Svaka komponenta sadrži svoju logiku i zasebno se renderira, omogućujući laku ponovnu upotrebljivost i održavanje. Ove komponente mogu se zajedno sastavljati kako bi se stvorila složena korisnička sučelja, čime se pruža skalabilan i fleksibilan pristup razvoju.

Pravilan rad Reacta bazira se na korištenju kuka (eng. hooks) [2]. Kuke su funkcije koje programeri koriste za pristup i izmjenu React stanja u aplikaciji. Dvije najpoznatije kuke, bez kojih se gotovo nijedna React aplikacija ne može razviti, su *useEffect* i *useState*.

*UseState* je kuka koja omogućuje pohranu i ažuriranje lokalnog stanja u React komponentama. Vraća polje koje sadrži 2 elementa. U kodnom isječku 2.1 može se vidjeti primjer kuke *useState*. Prvi element je varijabla, a drugi funkcija koja mijenja sadržaj te varijable. Kada se pozove funkcija *setBaza* s novom vrijednošću, React otkriva promjenu i ažurira samo relevantne dijelove korisničkog sučelja koji ovise o tom stanju. U ovom primjeru ažurira se paragraf koji sadrži bazu te njezin kvadrat. To rezultira učinkovitijim ažuriranjem i boljim performansama.

*UseEffect* omogućuje izvršavanje efekata unutar komponenata. Efekti su radnje koje se obavljaju izvan samog renderiranja komponente, poput dohvata podataka s API-ja, pretplate na događaje ili manipulacije s DOM-om. *useEffect* na prvom argumentu prima funkciju koja će se izvršiti kada se bilo koje stanje iz polja u drugom argumentu promijeni. U slučaju kada je spomenuto polje prazno, funkcija se izvršava samo pri montiranju komponente. Korištenjem *useEffect* kuke održava se pravilan redoslijed izvršavanja i izbjegavaju problemi poput curenja memorije ili nepredvidivih rezultata. U kodnom isječku 2.1 *useEffect* je zaslužan za nadgledanje

vrijednosti varijable *baza*. Kada vrijednost baze dostigne vrijednost 10, pojavi se skočni prozor sa porukom "Kliknuo si 10 puta!!!".

```
import React, { useState, useEffect } from 'react';

const Primjer = () => {
    const [baza, setBaza] = useState(0);
    const kvadrat = baza ** 2;

    useEffect(() => {
        if (baza === 10) {
            alert('Kliknuo si 10 puta!!!');
        }
    }, [baza]);

    function handleClick() {
        setBaza(baza + 1);
    }

    return (
        <div>
            <button onClick={handleClick}>
                Klikni za povecavanje baze
            </button>
            <p>
                Baza je {baza}.
                Kvadrat baze je {kvadrat}
            </p>
        </div>
    );
}

export default Primjer;
```

Kodni isječak 2.1 React primjer kuka *useState* i *useEffect* - "baza i kvadrat"

Jedna od ključnih prednosti Reacta je mogućnost stvaranja ponovno upotrebljivih komponenti, omogućavajući programerima brzu i učinkovitu izgradnju složenih korisničkih sučelja. Upravo ta ponovna uporabljivost omogućuje praćenje prakse odvajanja odgovornosti (eng. separation of concerns) i olakšava održavanje.

Arhitektura Reacta olakšava testiranje i otklanjanje pogrešaka. Komponente se

mogu testirati zasebno, što olakšava pisanje testova jedinica (eng. unit tests) i osigurava ispravnost pojedinih dijelova aplikacije. Reactova popularnost otvara širok spektar knjižnica i paketa koji su često prilagođeni da radu uz React, olakšavaju rad te znatno smanjuju broj linija koda od strane programera. Također, uz veliku popularnost dolazi i prednost velike zajednice.

Unatoč brojnim prednostima, React ima neka ograničenja. Jedan od nedostataka je krivulja učenja. Promjena načina razvoja na način zasnovan na komponentama i razumijevanje koncepta poput virtualnog DOM-a i upravljanja stanjem na prvu može biti zbunjujuć i zahtjevan. Nadalje, React se fokusira na efektivnu izgradnju i prikaz korisničkog sučelja, što znači da se programeri moraju osloniti na dodatne knjižnice kako bi izgradili potpune web aplikacije (knjižnice poput Reduxa, React Routera...). Iako se React može koristiti s drugim bibliotekama, poput React Routera za usmjeravanje i Axiosa za dohvat podataka, to zahtijeva korištenje softvera koji nisu nativni React.

Osim toga, optimizacije performansi React aplikacija mogu dovesti do složenijeg koda i detalja implementacije. Iako *diffing* algoritam virtualnog DOM-a značajno poboljšava učinkovitost renderiranja, može uvesti dodatne troškove u određenim scenarijima. Na primjer, kod rada s velikim skupovima podataka ili renderiranja visoko dinamičkih korisničkih sučelja. Programeri moraju biti svjesni performansi i primjenjivati optimizacijske tehnike kad je to potrebno. Unatoč navedenim ograničenjima, React nastavlja biti široko korišten u modernom razvoju web aplikacia. Njegova fleksibilnost i skalabilnost, obimna dokumentacija i podrška velike zajednice čine ga popularnim izborom za izgradnju korisničkih sučelja u različitim domenama i platformama.

## 2.2 Angular

Angular [3] je otvorena JavaScript knjižnica za izgradnju web aplikacija. Angular je prva verzija Angulara, a počevši od Angulara 2, Google je započeo s potpuno novim pristupom i arhitekturom. Razvijena od strane Googlea i objavljena 2010. godine, kroz godine većinom je bila druga najpopularnija knjižnica za stvaranje korisničkih sučelja (React prva). Angular je značajan zbog svoje sposobnosti da olakša razvoj aplikacija kroz primjenu modela pogona MVW (Model-View-Whatever) i deklarativnog pristupa programiranju. *W* u MVW označava *Whatever* ili *Whatever You Want*. To znači da Angular ne nameće određeni stil ili pristup pri implementaciji *W* dijela arhitekture.

Jedna od glavnih karakteristika Angulara je dvosmjerno vezanje podataka koje se postiže upotrebom [(*ngModel*)] direktive [4]. Ova direktiva omogućuje vezanje podataka između svojstva komponente i korisničkog sučelja, omogućujući ažuriranje podataka u korisničkom sučelju i komponenti istovremeno. Kada se podaci promijene u korisničkom sučelju, automatski se ažurira povezano svojstvo komponente. Ova funkcionalnost smanjuje količinu koda koji je potreban za održavanje stanja između modela (eng. Model) i prikaza (eng. View).

U isječku koda 2.2, kada korisnik unese nešto u *<input>* polje, vrijednost se automatski ažurira u *inputValue* svojstvu komponente te se istovremeno ažurira i paragraf element. I obrnuto, ako se ažurira inputValue iz komponente, vrijednost u polju se automatski ažurira. S druge strane, u Reactu se koristi jednosmjerno vezanje podataka (eng. one-way data binding). U Reactu, podaci teku u jednom smjeru - od roditeljske komponente prema djetetu. Ako se podaci promijene u djetetu, ne utječu izravno na roditeljsku komponentu.

```
1 import { Component } from '@angular/core';
2
3 @Component({
```

```
  selector: 'zavrsni-primjer',
  template: `
    <input [(ngModel)]="inputValue" placeholder="Unesi ne to" />
    <p>Uneseno: {{ inputValue }}</p>
  `,
})
export class PrimjerKomponenta {
  inputValue = '';
}
```

Kodni isječak 2.2 Angular - primejer dvostrukog vezanja

Angular koristi koncept direktiva kako bi omogućio programerima da prošire HTML oznake sa dodatnim funkcionalnostima. Direktive se koriste za manipuliranje DOM-om, povezivanje podataka s korisničkim sučeljem, obavljanje logike događaja i još mnogo toga.

Druga važna značajka Angulara je podrška za razvoj jednostraničnih aplikacija (eng. Single Page Applications, skraćeno SPA). SPA pristup omogućava brže učitavanje stranica i poboljšava korisničko iskustvo. Angular pruža rute i usmjeravanje za upravljanje navigacijom između različitih dijelova aplikacije bez ponovnog učitavanja cijele stranice. Kako bi se SPA podrška omogućila u Reactu, mora se uvesti dodatna knjižnica (kao što je React Router) u sam projekt.

Kao i React, Angular također ima svoj skup kuki (eng. hooks) koji se nazivaju servisi. Servisi su *singleton* objekti koji se koriste za dijeljenje podataka i funkcionalnosti između komponenti. Primjer servisa u Angularu može se vidjeti u kodnom isječku 2.3. Cilj *singleton* objekata je osigurati da postoji samo jedna instanca objekta koja može biti dijeljena i korištena iz svih dijelova aplikacije. Ova praksa promiče ponovnu upotrebljivost i olakšava upravljanje stanjem aplikacije [5].

```
import { Injectable } from '@angular/core';

@Injectable({
  providedIn: 'root'
})
```

```
export class ServicePrimjer {
  getData() {
    // Kod za dohvacanje podataka
  }

  sendData(data: any) {
    // Kod za slanje podataka
  }
}
```

Kodni isječak 2.3 Angular - primjer servisa

*ServicePrimjer* je klasa koja predstavlja uslugu. Koristimo anotaciju *@Injectable* kako bi označili da je ova klasa usluga. *providedIn: 'root'* označava da će usluga biti dostupna na razini cijele aplikacije.

U kodnom isječku 2.4, *PrimjerKomponenta* ubrizgava *ServicePrimjer* preko konstruktora. Na ovaj način komponenta može koristiti metode i funkcionalnosti koje pruža usluga.

```
export class PrimjerKomponenta {
  constructor(private servicePrimjer: ServicePrimjer) {
    // Ubrizgavanje usluge
  }

  // Koristenje usluge u komponenti
  someMethod() {
    const data = this.servicePrimjer.getData();
    this.servicePrimjer.sendData(data);
  }
}
```

Kodni isječak 2.4 Primjer ubrizgavanja u Angularu

Za usporedbu Reactove i Angularove sintakse, koristiti će se isti primjer baze i kvadrata. Isječak 2.5 prikazuje funkcionalnu perspektivu angularove komponente. *baza* i *kvadrat* su svojstva komponente. U konstruktoru se poziva funkcija *updateKvadrat* te se na taj način varijabla *kvadrat* inicijalizira na nulu (jer je *baza* jednaka nuli). Zbog dvosmjernog vezanja podataka, kada se promijeni vrijednost varijable

*baza*, automatski se ažurira i vrijednost varijable *kvadrat*. Iz tog razloga nema potrebe za kukom kao *useEffect* uReactu.

```
import { Component } from '@angular/core';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html'
})
export class PrimjerKomponenta {
  baza: number = 0;
  kvadrat: number;

  constructor() {
    this.updateKvadrat();
  }

  handleClick(): void {
    this.baza += 1;
    this.updateKvadrat();

    if (this.baza === 10) {
      alert('Kliknuo si 10 puta!!!');
    }
  }

  private updateKvadrat(): void {
    this.kvadrat = this.baza ** 2;
  }
}
```

Kodni isječak 2.5 Angular primjer "baza i kvadrat"

Također, u Angularu se HTML sprema u zasebnu datoteku prikazanu u kodnom isječku 2.6

```
<button (click)="handleClick()">
  Klikni za povecanje baze
</button>
<p>
  Baza je {{ baza }}.
  Kvadrat baze je {{ kvadrat }}
</p>
```

Kodni isječak 2.6 Angular primjer "baza i kvadrat" - HTML

Jedna od prednosti Angulara je njegova dokumentacija i podrška od strane zajednice. Također, posjeduje širok izbor službenih i zajedničkih modula koji olakšavaju integraciju s drugim bibliotekama i alatima.

Uz sve te prednosti, Angular dolazi uz par nedostataka. Jedan od njih je veća složenost u usporedbi s drugim bibliotekama poput Reacta. Angular uvodi mnoge koncepte i pojmove koji mogu biti zahtjevni za nove programere ili one koji nisu upoznati s konceptima MVC (Model-View-Controller) arhitekture.

Još jedno ograničenje je veličina biblioteke. Angular uključuje mnoge značajke, što ga čini pogodnim za veće aplikacije. Međutim, to znači da je Angular prevelik za manje projekte koji zahtijevaju samo osnovnu funkcionalnost.

Unatoč ovim ograničenjima, Angular i dalje ostaje popularan izbor za izgradnju web aplikacija, posebno u okruženjima gdje je potrebna poznata arhitektura i podrška za velike timove. Sa svojim moćnim alatima i konceptima, Angular pruža strukturu i organizaciju za razvoj aplikacija.

## 2.3 Svelte

Svelte [6] je moderni JavaScript okvir za izgradnju korisničkih sučelja koji se ističe svojim inovativnom pristupom. Za razliku od Reacta koji većinu posla obavlja u pregledniku, Svelte taj posao obavlja pri kompajliranju što se događa prilikom izgradnje aplikacije.

Umjesto virtualnog DOM *diffinga* koji uspoređuje cjelokupno stablo DOM-a radi otkrivanja promjena, Svelte generira specifičan kod koji direktno manipulira odgovarajućim dijelovima DOM-a koji zahtijevaju ažuriranje. Kada se stanje aplikacije promijeni u Svelteu, generira se optimizirani kod koji zna točno koje dijelove DOM-a treba ažurirati. Umjesto da se ažurira cijelo stablo DOM-a, samo se potrebni dijelovi

mijenjaju na temelju izračunatih razlika.

Jedna od ključnih značajki Sveltea je njegov koncept reaktivnosti. Svelte omogućuje vezanje podataka i automatsko ažuriranje korisničkog sučelja kada se podaci promijene. Umjesto korištenja posebnih sintaksa ili kuka kao što to rade React i Angular, Svelte koristi direktnu sintaksu u HTML-u za vezivanje podataka što čini kod čitljivijim i jednostavnijim za razumijevanje.

U kodnom isječku 2.7 prikazan je primjer načina rada reaktivnosti Sveltea. *Baza* je varijabla čija se vrijednost mijenja pritiskom na gumb. Bilo koji blok koda na najvišoj razini (izvan bloka ili funkcije) može se učiniti reaktivnim dodavanjem prefiksa „$:“. To znači da će se označeni blok izvršiti svaki put kada se vrijednosti u tom bloku promijene (npr. *Kvadrat* se mijenja svaki put kada i *baza*). Kako bi se ista funkcionalnost replicirala u Reactu, potrebno je koristiti *useState* za kvadrat i bazu, te *useEffect* za ažuriranje kvadrata i provjeru *if* uvjeta.

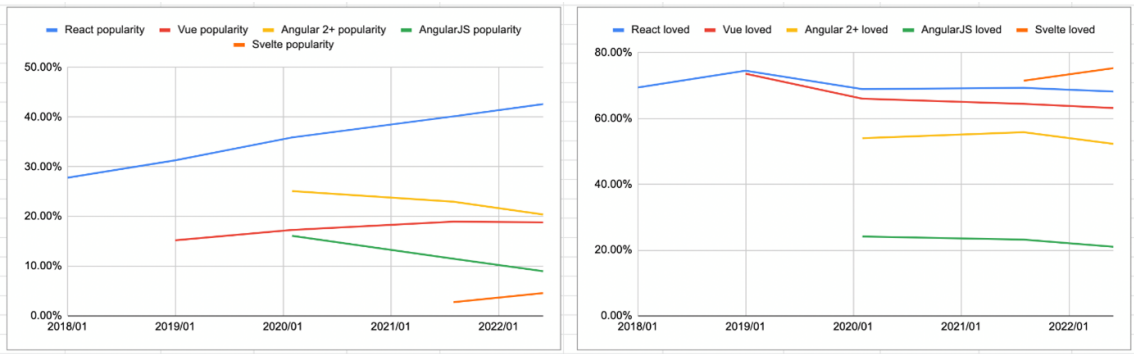
```
<script>
  let baza = 0;
  $: kvadrat = baza**2

  $: if (baza == 10) {
    alert(‘Kliknuo si 10 puta!!!‘);
  }

  function handleClick() {
    baza += 1;
  }
</script>

<button on:click={handleClick}>
  Klikni za povecavanje baze
</button>
<p>
  Baza je {baza}.
  Kvadrat baze je {kvadrat}
</p>
```

Kodni isječak 2.7 Svelte funkcionalnost

Glavna prednost Sveltea je njegova veličina. Budući da generira optimizirani kod tijekom kompilacije, Svelte aplikacije su iznimno male. U usporedbi s drugim okvirima poput Reacta ili Angulara, Svelte aplikacije mogu biti i do deset puta manje. To znači brže učitavanje i bolje performanse na strani klijenta.

U usporedbi s Angularom, Svelte se također ističe svojom jednostavnošću. Angular je vrlo moćan okvir s mnogo naprednih značajki, ali dolazi s većim opterećenjem. Svelte, s druge strane, pruža slične mogućnosti kao i Angular, ali s manje složenosti i manjom veličinom aplikacije.

Prema nedavnim istraživanjima, koje je obuhvatilo brojne programere, Svelte je proglašen najomiljenijom frontend knjižnicom, a njegova popularnost i zadovoljstvo korisnika jasno se vide na slici 2.1 [7]




Slika 2.1 Graf popularnosti i voljenosti knjižnica [8]

U konačnici, odabir između Sveltea, Reacta i Angulara ovisi o specifičnim potrebama projekta. Ako je potrebna mala veličina i brze performanse, Svelte je odličan izbor. Ako je fleksibilnost i velika zajednica važna, React može biti pravi izbor. Ako je potrebna potpuna funkcionalnost i skalabilnost, Angular je dobar izbor. Sve ove tehnologije imaju svoje prednosti i odabir ovisi o kontekstu i potrebama projekta.

## 2.4 Create-React-App

Create-React-App (CRA) [9] je službeno sučelje naredbenog retka (CLI) razvijeno od strane tima koji je razvio React. Svrha CRA alata je olakšati izazove s kojima se programeri susreću prilikom konfiguracije projekta tako što omogućava inicijalizaciju novih React projekata bez potrebe za konfiguracijom. Prije predstavljanja CRA-a, inicijalizacija novog React projekta često je uključivala složene konfiguracije Webpacka, Babela i drugih alata što je dovodilo do takozvanog "konfiguracijskog pakla".

Osnovna značajka CRA-a jest pružanje nulte konfiguracije. To rezultira brzim pokretanjem React projekta bez ikakve potrebe za ručnom konfiguracijom. CRA uključuje niz alata kao što su Webpack za *bundling*, Babel za transpilaciju ES6+ koda, ESLint za analizu koda (linting) i Jest za testiranje. Prilikom razvoja, programeri koriste razvojni poslužitelj u stvarnom vremenu (eng. Live Development Server) koji omogućuje brzo učitavanje koda čime se izbjegava osvježavanje stranice svaki put kada se napravi promjena. Također, projekti inicijalizirani koristeći CRA su optimizirani za produkciju te dolaze sa podrškom za progresivne web aplikacije (eng. Progressive Web Apps).

Za pokretanje novog React projekta, potrebno je unijeti jednu komandu prikazanu u kodnom isječku 2.8. U složenijim projektima, CRA može biti restriktivan te da bi se dobio pristup skrivenoj konfiguraciji koristi se *ejecting*. *Ejecting* je postupak koji omogućuje pristup i kontrolu nad skrivenom konfiguracijom Webpacka i Babela u CRA projektu te je često nepreporučljiv jer izlaže programera kompleksnim konfiguracijama kojima je često vrlo teško upravljati.

```
1 npx create-react-app my-app
```

Kodni isječak 2.8 CRA inicijalizacija

## 2.5 Vite

Vite [10] je alat za izgradnju koji pruža bržu ponovnu izgradnju aplikacije i vrijeme učitavanja. Osnovni zadatci Vitea su pružanje razvojnog servera tijekom razvoja te slaganje (eng. bundling) JavaScripta, CSS-a i ostalih elemenata u jedan paket za preglednik (eng. browser). Razvojni server koji pruža je na bazi ES modula te poslužuje kod pregledniku u takvom obliku što eliminira potrebu za bundlingom tijekom razvoja.

Jedna od glavnih značajki Vitea je *vruća zamjena modula* (eng. hot module replacement) što omogućuje pregled promjena u pregledniku tijekom razvoja bez potrebe za osvježavanjem stranice. Tradicionalni slagači (eng. bundleri), poput Webpacka, rade na način da "zapakiraju" cijeli projekt u nekoliko datoteka, čak i tijekom razvoja. Ovaj proces postaje sporiji kako projekt raste. S druge strane, Vite poslužuje datoteke pojedinačno koristeći HTTP 2/3, što omogućuje znatno brže vrijeme učitavanja, posebno za velike projekte.

Vite nudi podršku za izradu aplikacija u Reactu, Preactu, Svelteu, LitElementu i mnogim drugima. U kodnom isječku 2.9 prikazan je način inicijalizacije React projekta pomoću Vite alata.

```
1 create-vite my-react-project --template react
```

Kodni isječak 2.9 Inicijalizacija React projekta pomoću Vite alata

## 2.6 Bootstrap

Bootstrap [11] je zbirka alata otvorenog koda koji olakšava razvoj responsivnih web stranica. Jedan je od ključnih alata koji olakšava kompleksnost web dizajna i omogućuje programerima brzu izradu responsivnih web stranica neovisno o platformi na kojoj se aplikacija pokreće. Bootstrap pruža niz prethodno dizajniranih kompo-

nenti, poput navigacijskih traka, kartica, obrazaca i drugih, koje programeri mogu lako uključiti u svoje projekte. Također, uz prethodno dizajnirane komponente, dolazi i sa prethodno definiranim klasama. Ukoliko programeri žele promijeniti izgled ili funkcionalnost nekih komponenti ili klasa, imaju mogućnost detaljne konfiguracije pomoću SCSS varijabli.

Bootstrap pruža određene predloške koji su responzivne naravi, odnosno prilagođavaju se raznim veličinama ekrana. Jedna od ključnih komponenti je njegov mrežni sustav (eng. grid) koji se bazira na konceptu sa 12 stupaca po redu. To znači da ovisno o veličini ekrana, može se definirati koliko stupaca određeni element zauzima. Veličine ekrana su preddefinirane:

- xs: Ekstra mali uređaji (telefoni) - manje od 576px
- sm: Mali uređaji (tableti) - više od 576px
- md: Srednji uređaji (desktop) - više od 768px
- lg: Veliki uređaji (veći desktop) - više od 992px
- xl: Ekstra veliki uređaji (HD ekrani) - 1200px i nadalje

U primjeru prikazanom u kodnom isječku 2.10, željeni rezultat jest da div element ID-a *red1* zauzima tri četvrtine širine na tabletu (sm veličina), a div ID-a *red2* zauzima jednu četvrtinu širine. Ukoliko bi uređaj bio veći, npr. desktop, oba div-a zauzimala bi cijelu širinu ekrana.

```
1 <div class="container">
2     <div class="row">
3         <div id="red1" class="col-sm-9 col-md-12">Stupac
    zauzima 9/12 sirine reda na tabletu i 12/12 na desktopu.</
    div>
4         <div id="red2" class="col-sm-3 col-md-12">Stupac
    zauzima 3/12 sirine reda na tabletu i 12/12 na desktopu.</
    div>
5     </div>
```

```
6 </div>
```

Kodni isječak 2.10 Grid primjer u Bootstrapu

Jedan od nedostataka Bootstrapa je veličina same knjižnice, pogotovo kada se koristi velik broj značajki. Također, ukoliko programeri ne konfiguriraju izgled i klase Bootstrapa, stranica postaje veoma slična ostalim stranicama izrađenih pomoću Bootstrapa.

# Poglavlje 3

# Izrada aplikacija

## 3.1 Opis aplikacije genetskog algoritma

Cilj aplikacije je izvršavanje skupog algoritma, u ovom slučaju genetskog algoritma, u svrhu opterećenja računala kako bi se mogla izmjeriti i usporediti potrošnja energije za sljedeće okvire:

- React (kreiran pomoću alata CRA)
- Angular
- React (kreiran pomoću alata Vite)
- Svelte

Razlog testiranja Reacta sa CRA i Vite alatom je zbog toga što je CRA kreiran 2016. i ne koristi najnovije značajke kao što ih koristi Vite za optimizaciju brzine i veličina paketa.

Aplikacija se sastoji od 4 input polja koja služe za mijenjanje parametara za izvršavanje genetskog algoritma te gumba za pokretanje algoritma. Podesivi parametri:

- Veličina populacije
- Duljina kromosoma
- Broj generacija
- Vjerojatnost mutacije

## 3.2 Genetski algoritam

Genetski algoritam (GA) [12] je bioinspirirana metaheuristička metoda optimizacije koja simulira proces prirodne selekcije. Pripada grupi evolucijskih algoritama koji koriste metodu prirodne selekcije za aproksimaciju rješenja za određeni problem. GA metoda je za rješavanje ograničenih i neograničenih optimizacijskih problema koja se temelji na prirodnoj selekciji. Prije objašnjenja GA-a, bilo bi korisno proći primjer problema kojeg GA rješava.

Problem: Čovjek se sprema za put i sa sobom u ruksak može ponijeti maksimalno 3 kilograma prtljage. Na raspolaganju mu stoje predmeti poput laptopa, mobitela, šalice, kape, itd. Svakom navedenom predmetu pridružuje određenu vrijednost (bodove) koja je povezana s time koliku mu vrijednost predstavlja određeni predmet. Npr. laptop mu je korisniji od šalice te će radi toga laptopu biti pridruženo 500 bodova, dok šalici 60. Čovjekov zadatak je maksimizirati broj bodova, ali da pri tome ne prijeđe granicu kilaže od 3 kilograma. Ovaj problem naziva se problem naprtnjače (eng. Knapsack problem).

U slučaju malog broja predmeta, čovjeku to ne predstavlja problem. Čak bi i računalo nasumičnim odabirom došlo do rješenja u par milisekundi ili čak mikrosekundi. Međutim, povećamo li broj artikala na broj veći od 80, računalu bi nasumičnim odabirom trebalo par milijardi godina (eksponencijalni rast).

Iz tog razloga, potrebno je koristiti neki metaheuristički algoritam, npr. genetski

algoritam. Genetski algoritam radi na način da ima početnu populaciju koja je skup nasumičnih mogućih rješenja, a svaki kromosom predstavlja jedno moguće rješenje. Svaka pozicija (gen) u kromosomu označava određeni predmet. Specifično za navedeni problem, jedinica označava da je predmet uključen u ruksak, dok nula označava da predmet nije uključen. Na slici 3.1 prikazan je primjer jednog kromosoma, odnosno mogućeg rješenja koje je dio populacije. Sva trenutna rješenja u algoritmu nazivaju se generacijom.


Slika 3.1 Primjer s ruksakom i predmetima [13]

Početna (nulta) generacija je kombinacija nasumičnih rješenja. U ovom trenutku započinje proces prirodne selekcije. *Fitness* funkcija je funkcija koja računa koliko je pojedino rješenje optimalno. U slučaju problema s ruksakom, funkcija zbraja vrijednosti pojedinih predmeta, te ako je ukupna kilaža manja od granice (3kg) onda se zadržava izračunata suma, u suprotnom *fitness* vrijednost iznosi 0. Nakon toga, odabiru se roditelji (eng. parents) za sljedeću generaciju gdje pri odabiru vrijedi da roditelji sa većom *fitness* vrijednosti imaju veće šanse postanka roditeljom (kao u pravom životu). Nakon odabira roditelja, uzimaju se njihove vrijednosti kromosoma

te se na nasumičnom indeksu isprepletu - *crossover* funkcija. Npr., K1 ima kromosom 111111, K2 ima kromosom 000000, nasumični indeks iznosi 3 (indeksirano od nule) što znači da će njihovi potomci imati vrijednosti 111100 i 000011. Ovaj proces se odvija dok se ne dostigne ista populacija kao u prijašnjoj generaciji.

Na ovaj način bi, u pravilu, svaka generacija imala novo najbolje rječenje. Međutim, zbog nasumičnosti selekcije i *crossover* funkcije, najbolje rješenje se ne može garantirati te se može dogoditi uništenje najboljeg rješenja u bilo kojem trenutku (u bilo kojoj generaciji) tijekom izvođenja algoritma. Kako bi se smanjila šansa uništenja najboljeg rješenja, uvodi se proces zvan elitizam koji omogućava odabir N najboljih rješenja iz prijašnje generacije te njihovo prenošenje u novu generaciju.

Uvodi se i pojam mutacije koja pomaže u pronalaženju rješenja do kojih se nebi moglo doći prethodno opisanim procesom. Mutacija je proces mijenjanja nasumičnog gena u kromosomu s nekom vjerojatnošću.

Opisani algoritam se izvršava dok god se ne zadovolji neki uvjet ili dok se ne dostigne zadana vrijednost generacija.

## 3.3 Pseudokod korištene varijacije genetskog algoritma

U isječku 3.1 nalazi se pseudokod korištene varijacije genetskog algoritma. U odnosu na prethodno opisani genetski algoritam, kod korištene varijacije ne postoji gornja granica određene vrijednosti već se gleda isključivo ukupni zbroj - što veći zbroj, bolje rješenje. U korištenoj varijaciji prikazanoj u kodnom isječku 3.1, nema eliticizma te je šansa postanka roditelja direktno proporcionalna zbroju, odnosno kvaliteti rješenja (nema dodatnih uvjeta).

```
1 POKRENI Genetski Algoritam:
```

```
    postavi najbolje rjesenje kao praznu listu
    postavi pocetnu populaciju sa slucajnim kromosomima
    postavi fitnes vrijednosti za pocetnu populaciju

    ZA svaku generaciju DO broj generacija:
        postavi trenutnu generaciju
        // Selekcija
        odaberi roditelje iz trenutne populacije na temelju fitnesa
        // Krizanje
        ZA svaki kromosom U populaciji:
            odaberi dva roditelja sa nekom vjerojatnoscu
            stvori potomka krizanjem odabranih roditelja
            dodaj potomka u listu potomaka

        // Mutacija
        ZA svakog potomka:
            mutiraj potomka s odredenom vjerojatnoscu

        zamijeni staru populaciju s novom (potomcima)
        izracunaj nove fitnes vrijednosti

        pronadi najbolje trenutno rjesenje i usporedi ga s globalno najboljim rjesenjem
        ako je trenutno rjesenje bolje, postavi ga kao globalno najbolje

        // Stvaranje nove generacije - pola od prethodne generacije, pola novo generiranih kromosoma
        sortiraj kromosome na temelju fitnesa i uzmi prvih 50%
        dodaj ostatak populacije generiranjem novih kromosoma
        postavi novu populaciju i izracunaj nove fitnes vrijednosti

    ispisi globalno najbolje rjesenje i potrebno vrijeme izvodenja
KRAJ
```

Kodni isječak 3.1 Pseudokod GA

## 3.4 Implementacija GA u Reactu

U kontekstu implementacije, genetski algoritam u Reactu ne razlikuje se bitno od onoga u Angularu ili Svelteu. Logičke operacije i struktura algoritma ostaju iste bez obzira na korišteni okvir. Razlika u implementaciji postaje vidljiva prilikom upravljanja stanjima aplikacije. Dok React koristi sustav kuka (eng. hooks) za upravljanje i praćenje promjena stanja, Angular i Svelte se oslanjaju na dvosmjerno vezivanje (eng. two-way data binding) i reaktivnost za promjene vrijednosti varijabli.

Isječak koda 3.2 prikazuje početnu fazu genetskog algoritma u Reactu. U toj fazi, polje koje sadrži najbolje rješenje inicijalizira se na prazno polje.

```
const t1 = Date.now();
let bestSolution = [];

setSolution(null);
let initialPopulation = [];

for (let i = 0; i < populationSize; i++) {
  let chromosome = generateRandomChromosome(chromosomeLength);
  initialPopulation.push(chromosome);
}
let newPopulation = initialPopulation;
let fitnessValues = initialPopulation.map((chromosome) =>
  fitnessFunction(chromosome)
);

let newFitnessValues = fitnessValues;
```

Kodni isječak 3.2 Inicijalizacija vrijednosti za GA

Zatim, za stvaranje početne populacije koristi se funkcija koja vraća nasumično generirane kromosome (3.3).

```
function generateRandomChromosome(length) {
  let chromosome = [];
  for (let i = 0; i < length; i++) {
    chromosome.push(Math.floor(Math.random() * 2));
  }
  return chromosome;
```

```
}
```

Kodni isječak 3.3 Stvaranje nasumičnog kromosoma

Nakon formiranja inicijalne populacije, za svaki kromosom izračunava se *fitness* vrijednost korištenjem funkcije prikazane u isječku 3.4. *FitnessFunction* zbraja svaki gen unutar kromosoma te vraća dobiveni zbroj.

```
const fitnessFunction = (chromosome) => {
  let sum = chromosome.reduce((acc, val) => acc + val, 0);
  return sum;
};
```

Kodni isječak 3.4 Fitness funkcija

U sučelju aplikacije korisnik određuje željeni broj generacija za izvršavanje genetskog algoritma. Slijedeći isječci koda izvršavaju se za svaku od tih generacija.

Isječak 3.5 prikazuje dio koda zaslužan za stvaranje potomaka koristeći roditelje trenutne populacije (prijašnje generacije).

```
setGeneration(i);

// Selekcija
let parents = selectParents([...newPopulation],
   newFitnessValues);
// Crossover
let offspring = [];
for (let j = 0; j < populationSize; j++) {
    let parent1 = parents[Math.floor(Math.random() * parents.
   length)];
    let parent2 = parents[Math.floor(Math.random() * parents.
   length)];
    let child = crossover(parent1, parent2);
    offspring.push(child);
}

// Mutacija
for (let j = 0; j < offspring.length; j++) {
    offspring[j] = mutate(offspring[j], mutationProbability);
}

// Zamijeni staru populaciju s potomcima
```

```
20 newPopulation = offspring;
21
22 newFitnessValues=newPopulation.map((chromosome)=>
      fitnessFunction(chromosome));
```

Kodni isječak 3.5 Generacija potomaka

Za početak, vrši se ažuriranje stanja trenutne generacije postavljanjem indeksa generacije na odgovarajući indeks iteracije. Nadalje, koristi se funkcija *selectParents* za odabir kvalificiranijih roditelja iz trenutne populacije. Funkcija *selectParents*, prikazana zajedno sa *rouletteWheelSelection* funckijom u isječku 3.6, odabire dvoje roditelja za svakog člana populacije korištenjem selekcije pomoću kotača ruleta (eng. roulette wheel selection).

Funkcija *rouletteWheelSelection* koristi *fitness* vrijednost svakog kromosoma kako bi odredila vjerojatnost njegove selekcije. Npr., *fitnessValues* iznosi [2,3,5]. *totalFitness* je zbroj *fitness* vrijednosti te u ovom slučaju iznosi 10. Nakon toga, odabire se nasumična vrijednost između 0 i ukupne *fitness* vrijednosti. Prolazi se kroz polje *fitness* vrijednosti, zbrajajući njihove vrijednosti dok suma ne prekorači prethodno odabranu nasumičnu vrijednost. Indeks na kojem se to dogodi je indeks odabranog kromosoma. Na taj način će vrijednosti odabira roditelja biti proporcionalne kvaliteti rješenja, u ovom slučaju [0.2, 0.3, 0.5].

```
1  function selectParents(population, fitnessValues) {
2    let parents = [];
3    for (let i = 0; i < population.length; i++) {
4      let parent1 = population[rouletteWheelSelection(
     fitnessValues)];
5      let parent2 = population[rouletteWheelSelection(
     fitnessValues)];
6      parents.push(parent1);
7      parents.push(parent2);
8    }
9    return parents;
10 }
11
12 function rouletteWheelSelection(fitnessValues) {
```

```
  let totalFitness = fitnessValues.reduce((acc, val) => acc +
   val, 0);
  let randomFitness = Math.random() * totalFitness;
  let sum = 0;
  for (let i = 0; i < fitnessValues.length; i++) {
    sum += fitnessValues[i];
    if (sum > randomFitness) return i;
  }
  return fitnessValues.length - 1;
}
```

Kodni isječak 3.6 Pomoćne funkcije za odabir roditelja

Slijedeći korak uključuje križanje odabranih roditelja kako bi se generirala nova generacija potomaka. Potomci su, zbog proporcionalne vjerojatnosti selekcije sa kvalitetom rješenja, često rezultat kombinacije gena najboljih rješenja prethodne generacije.

Križanje između dva roditelja provodi se odabirom nasumičnog indeksa N unutar kromosoma roditelja. Zatim, potomak nasljeđuje prvih N gena od prvog roditelja, dok preostale gene preuzima od drugog roditelja. *crossover* funkcija, prikazana u isječku 3.7, vrši križanje roditelja.

```
function crossover(parent1, parent2) {
  if (!parent1 || !parent2) {
    console.error("Parent arrays are undefined");
    return [];
  }

  let child = [];
  let crossoverPoint = Math.floor(Math.random() * parent1.
   length);
  for (let i = 0; i < parent1.length; i++) {
    if (i < crossoverPoint) child.push(parent1[i]);
    else child.push(parent2[i]);
  }
  return child;
}
```

Kodni isječak 3.7 Pomoćna funkcija križanja roditelja

Radi potencijalnog otkrića novog optimalnog rješenja koja se možda ne bi pojavila samo kroz križanje, svaki potomak prolazi kroz proces mutacije. U sučelju aplikacije definira se vjerojatnost mutacije gena koja onda zapravo postaje šansa da gen kromosoma promijeni vrijednost iz 1 u 0 i obrnuto. Funkcija *mutate*, prikazana u isječku 3.8, vrši mutaciju kromosoma.

```
function mutate(chromosome, mutationProbability) {
  let mutatedChromosome = chromosome;
  for (let i = 0; i < chromosome.length; i++) {
    if (Math.random() < mutationProbability)
      mutatedChromosome[i] = 1 - mutatedChromosome[i];
  }
  return mutatedChromosome;
}
```

Kodni isječak 3.8 Pomoćna funkcija za mutaciju kromosoma

Nova generacija potomaka postaje trenutna populacija, nakon čega se za svaki kromosom izračunava *fitness* vrijednost.

U isječku 3.10 nalazi se kod za pronalazak potencijalno novog najboljeg rješenja. Za početak, iz nove generirane populacije izdvaja se kromosom s najvećom *fitness* vrijednošću. Ako ta vrijednost nadmašuje *fitness* vrijednost trenutno najboljeg rješenja, odgovarajući kromosom postavlja se kao novo najbolje rješenje. Nevezano za funkcionalnost algoritma, svakih 10 generacija ispisuje se trenutno najbolje rješenje.

```
let bestIndexCurrent = 0;
let bestSolutionCurrent = [];
bestIndexCurrent = newFitnessValues.indexOf(Math.max(...
   newFitnessValues));
bestSolutionCurrent = newPopulation[bestIndexCurrent];

if ((i + 1) % 10 === 0) {
    console.log([...bestSolutionCurrent])
}

if (fitnessFunction(bestSolutionCurrent) > fitnessFunction(
   bestSolution)) {
    bestSolution = [...bestSolutionCurrent];
```

```
}
```

Kodni isječak 3.9 Pronalazak najboljeg rješenja

Zadnji korak unutar iterativnog pronalaska najboljeg rješenja je zamjena stare generacije novom. Nadolazeća generacija kombinacija je polovice najboljih rješenja prijašnje te nasumično generiranih kromosoma kao drugom polovicom.

Prva polovica dobiva se sortiranjem trenutne populacije koristeći *fitness* vrijednost kao ključ sortiranja. Nakon sortiranja, populacija je sortirana po *fitness* vrijednosti u padajućem redoslijedu gdje se uzima prva polovica tih rješenja.

Druga polovica dobiva se korištenjem prethodno opisane funkcije *generateRandomChromosome* (3.3).

Postavljanje nove populacije te izračun novih *fitness* vrijednosti za tu populaciju zadnji je korak unutar petlje.

```
//Nadolaze a generacija - bolja polovica prija nje i polovica nove
let populationNewGeneration = [];
const newFitnessValuesObject = newFitnessValues.map((value, index) => ({
    fitness: value,
    indexSolution: index
}));
newFitnessValuesObject.sort((a, b) => b.fitness - a.fitness);
const firstHalf = newFitnessValuesObject.slice(0, Math.ceil(newFitnessValuesObject.length / 2));


// Prva polovica nove generacije sastoji se od najboljih rje enja iz prija nje generacije
firstHalf.forEach(obj => {
    populationNewGeneration.push(newPopulation[obj.indexSolution]);
});

// Druga polovica generacije je nasumi no generirana
for (let i = 0; i < populationSize - Math.ceil(newFitnessValuesObject.length / 2); i++) {
```

```
18      let chromosome = generateRandomChromosome(chromosomeLength)
    ;
19      populationNewGeneration.push(chromosome);
20  }
21
22  //Nova populacija - nova generacija
23  newPopulation = populationNewGeneration;
24  newFitnessValues = newPopulation.map((chromosome) =>
    fitnessFunction(chromosome));
```

Kodni isječak 3.10 Pronalazak najboljeg rješenja

Nakon iteracije kroz zadani broj generacija, najbolje rješenje sprema se u stanje *solution*. U slučaju Reacta, to se radi pomoću kuke *setSolution*.

## 3.5 Opis aplikacije opterećenja korisničkog sučelja

Cilj aplikacije je opteretiti korisničko sučelje (eng. User Interface, skraćeno UI) kako bi se otkrilo koji se okvir bolje nosi sa takvim opterećenjima i koliko energije potroši pri tome. Za razliku od prošle aplikacije koju opterećuje JavaScript više nego okvir (framework), ovdje dolazi do izražaja optimizacija okvira za visokofrekventne i velike promjene sučelja. Aplikacija se testira na ista 4 okvira:

- React (kreiran pomoću alata CRA)
- Angular
- React (kreiran pomoću alata Vite)
- Svelte

Aplikacija se sastoji od jednog gumba, jednog input polja te 20 000 div elemenata koji sadrže tekst koji se nalazi u input polju. Pritiskom na gumb, pokreće se funkcija koja svakih 100 milisekundi dodaje 1 znak iz niza znakova "ABCDEFGHIJKLMNOPQRSTUVWXYZ123456789" u input polje. Svakom promjenom vrijednosti

input polja, mijenja se i sadržaj u svih 20 000 div elemenata.

Zbog potrebe ažuriranja velike količine elemenata na sučelju u kratkom vremenskom okviru, aplikacija stavlja iznimno visoko opterećenje na korisničko sučelje te pruža dobru podlogu za mjerenje potrošnje energije.

## 3.6 Pseudokod aplikacije opterećenja korisničkog sučelja

U isječku 3.11 nalazi se pseudokod skripte korištene za opterećenje korisničkog sučelja.

```
Kada se vrijednost inputa promijeni:
    Azuriraj listu koja sadrzi div elemente s novom vrijednoscu
    inputa za svaki element u listi

Kada se klikne na gumb "Start":
    Definiraj text koji sadrzi "
   ABCDEFGHIJKLMNOPQRSTUVWXYZ123456789"
    Inicijaliziraj brojac i na -1

    Pokreni interval koji ce svakih 100ms:
        Inkrementiraj brojac
        Ako je brojac jednak duzini teksta:
            Prekini interval
        Inace, dodaj sljedeci znak iz teksta u input

//Dio korisnickog su elja
Prika i korisnicko sucelje koje se sastoji od:
    Gumb "Start" koji pokrece unos teksta
    Input polje koje prikazuje trenutni unos
    Listu od 20 000 div elemenata koji prikazuju trenutni unos
```

Kodni isječak 3.11 Pseudokod aplikacije opteređenja UI-a

## 3.7 Implementacija aplikacije opterećenja UI-a u Reactu

Za početak, definiraju se inicijalne vrijednosti za varijable *input* i *list*, koje služu čuvanju vrijednosti input polja te liste iz koje će svih 20 000 div elemenata vući sadržaj. Također, definira se funkcija koja je zaslužna za rukovanje stanjem *input* varijable prilikom promjene vrijednosti input polja.

```
const [input, setInput] = useState("");
const [list, setList] = useState([])

const handleChange = (e) => {
    setInput(e.target.value);
}
```

Kodni isječak 3.12 Inicijalizacija vrijednosti

```
return (
<div>
  <button onClick={startTyping}>
    Start
  </button>
  <input type='text' value={input} onChange={handleChange} />
  {list.map((i, index) => <div key={index}>{i}</div>)}
</div>
);
```

Kodni isječak 3.13 Elementi korisničkog sučelja

Pritiskom na gumb, pokreće se funkcija *startTyping*. *startTyping* inicijalizira indeks varijablu na vrijednost -1, *i* te *text* varijablu na vrijednosti "ABCDEFGHIJKLMNOPQRSTUVWXYZ123456789" koja će eventualno biti ispisana u input polju i u 20 000 div elemenata.

Nakon toga, pokreće se interval koji svakih 100 milisekundi:

1. Inkrementira indeks polje (metodom pokušaja-greške otkriveno je da inicijalizacija indeksa na -1, te njegovo inkrementiranje na početku intervala proizvodi

najispravnije rezultate)

2. Provjerava je li indeks dosegnuo vrijednost duljine teksta koji se treba ispisati, odnosno provjerva je li ispisan cijeli tekst
3. Nadodaje se sljedeći znak u nizu vrijednosti *input* varijable

Paralelno sa intervalom, *useEffect* funkcija prati vrijednosti *input* stanja te ažurira vrijednosti polja na aktualno stanje. *startTyping* i *useEffect* funkcije prikazane su na isječku .

```
1  useEffect(() => {
2    const newList = Array(20000).fill(input);
3    setList(newList)
4  }, [input]);
5
6  const startTyping = () => {
7    const text = "ABCDEFGHIJKLMNOPQRSTUVWXYZ123456789";
8    let i = -1;
9    const interval = setInterval(() => {
10     i++;
11     if (i === text.length) {
12       clearInterval(interval);
13       return;
14     }
15     setInput((inp) => inp + text[i]);
16   }, 100);
17 }
```

Kodni isječak 3.14 *startTyping* i *useEffect* funkcije

# Poglavlje 4

# Usporedba okvira

## 4.1 Usporedba pri izradi aplikacije

Veliki faktor pri odabiru okvira je njegova zrelost, ekosustav i zajednica. React ima ogromnu zajednicu, mnogo resursa za učenje i bogat ekosustav. Postoji mnogo knjižnica i alata koji su razvijeni specifično za React. Angular ima sličnu poziciju kao React, ali primjetno slabiju. Svelte, pokrenut 2016.,je relativno noviji okvir, ali je dobio popularnost zbog svoje inovativne arhitekture. Zajednica je manja, ali brzo raste. Iz tog razloga, novi programeri često odabiru React zbog podrške zajednice iako je Svelte objektivno bolji u većini faktora. [14]

U kontekstu jednostavnosti učenja, Svelte se ističe kao najbolji izbor. Smatra se inuitivnim i jednostavnim za učenje zbog svoje sintakse i jednostavnosti početnog projekta. React nije pretjerano složen, no koncept kuka može biti kompleksan novim programerima. Angular je potpuni okvir s modulima, komponentama, servisima, direktivama itd. Zbog svog opsežnog okvira, ima strmiju krivulju učenja. Optimiziran je za velike aplikacije te radi stroge arhitekture pogodniji je za timske projekte. Međutim, sve je namješteno pri izradi projekta, što može ubrzati razvoj za iskusne

programere.

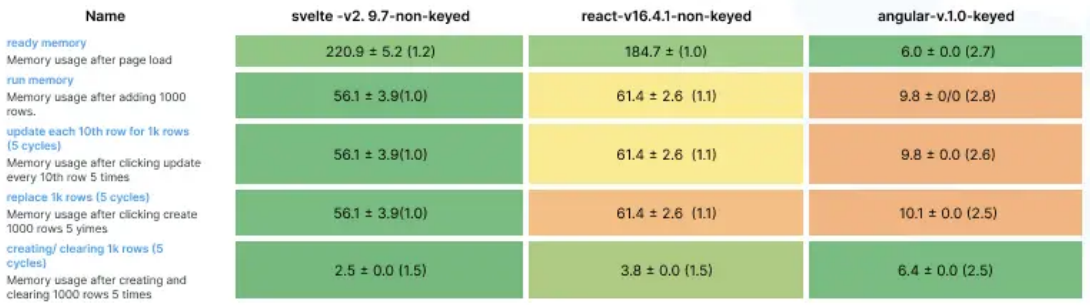
Kada se radi o performansama, Svelte opet izlazi kao pobjednik. Na slici 4.1 može se vidjeti da Svelte pokazuje najveću brzinu izvođenja od svih okvira. Međutim, razlika nije drastična.

**Duration in milliseconds ± standard deviation (Slowdown = Duration/Fastest)**

| Name | svelte -v2. 9.7-non-non-keyed | react-v16.4.1-non-keyed | angular-v.1.0-keyed |
|---|---|---|---|
| create rows<br>Duration for creating 1000 rows after the page loaded. | 220.9 ± 5.2 (1.2) | 184.7±(1.0) | 185.2 ± 10.2(1.0) |
| replace all rows<br>Duration for updating all 1000 rows of the table (with 5 warmup iterations) | 56.1 ± 3.9(1.0) | 61.4±2.6 (1.1) | 161.2 ± 2.7(2.9) |
| partial update<br>Time to update the text of every 10th row (with 5 warmup iterations) for a table with 10k rows. | 65.2 ± 2.5(1.0) | 81.3±(1.2) | 68.8 ± 3.7(1.1) |
| select row<br>Duration to highlight a row in response to a click on the row (with 5 warmup iterations). | 8.8 ± 3.4(1.0) | 10.4 ± 2.3(1.0) | 7.9 ± 4.3(1.0) |
| not aligned<br>Time to swap 2 rows on a 1K table. (with 5 warmup iterations). | 15.1 ± 4.2(1.0) | 14.8 ± 4.5(1.0) | 105.8 ± 1.8(6.6) |
| remove row<br>Duration to remove a row. (with 5 warmup iterations). | 29.8 ± 0.7(1.0) | 37.8 ± 1.5 (1.3) | 47.1 ± 3.0 (1.6) |
| creat many rows<br>Duration to remove a row. ( with 5 warmup iteration) | 2,313.9 ± 49.0(1.4) | 1.945.2 ± 24.4(1.1) | 1,693.9 ± 70.1 (1.0) |
| append rows to large table<br>Duration for adding 1000 rows on a table of 10.000 rows | 296.5 ± 4.6(1.2) | 267.8 ± 5.5(1.1) | 243.3 ± 6.3 |
| clear rows<br>Duration to cleare the table filled with 10.000 rows. | 174.6 ± 2.5(1.0) | 175.3 ± 1.6(1.0) | 263.9 ± 3.0 (1.5) |
| slowdown geomertic mean | 1.08 | 1.0 | 1.54 |

Slika 4.1 Usporedba brzine izvršavanja operacija u okvirima [14]

Slika 4.2 prikazuje rezultate testa pokretanja. Test pokretanja pokazuje koliko vremena je potrebno da se aplikacija u određenom okviru pokrene.



| Name | svelte -v2. 9.7-non-keyed | react-v16.4.1-non-keyed | angular-v.1.0-keyed |
|---|---|---|---|
| ready memory<br>Memory usage after page load | 220.9 ± 5.2 (1.2) | 184.7 ± (1.0) | 6.0 ± 0.0 (2.7) |
| run memory<br>Memory usage after adding 1000 rows. | 56.1 ± 3.9(1.0) | 61.4 ± 2.6 (1.1) | 9.8 ± 0/0 (2.8) |
| update each 10th row for 1k rows (5 cycles)<br>Memory usage after clicking update every 10th row 5 times | 56.1 ± 3.9(1.0) | 61.4 ± 2.6 (1.1) | 9.8 ± 0.0 (2.6) |
| replace 1k rows (5 cycles)<br>Memory usage after clicking create 1000 rows 5 yimes | 56.1 ± 3.9(1.0) | 61.4 ± 2.6 (1.1) | 10.1 ± 0.0 (2.5) |
| creating/ clearing 1k rows (5 cycles)<br>Memory usage after creating and clearing 1000 rows 5 times | 2.5 ± 0.0 (1.5) | 3.8 ± 0.0 (1.5) | 6.4 ± 0.0 (2.5) |

Slika 4.2 Usporedba brzine pokretanja aplikacija u okvirima [14]

U većini slušajeva Svelte se pokazao najbržim. Razlog tomu je nedostatak virtualnog DOM-a koji uzrokuje *overhead*, te zato što radi kao kompajler. Dok React koristi virtualni DOM i Angular koristi stvarni DOM za ažuriranje preglednika, Svelte nema okvir za vrijeme izvođenja. Umjesto toga, on kompajlira komponente izravno u kod koji ažurira DOM.

Što se tiče potrebnog vremena za izradu aplikacije, Svelte je opet pobjednik. Generirani kod je čitljiviji i jednostavniji, kraći od Reacta ili Angulara. React i Angular su slični u pogledu vremena razvoja. Ipak, postoje male razlike. React je lakše naučiti od Angulara i pri izradi projekta ne mora pratiti strogu arhitekturu kao u Angularu.

## 4.2 Prethodna istraživanja potrošnje

Performanse okvira (brzina, veličina paketa i responzivnost) se često ocjenjuju i uspoređuju između okvira. Međutim, potrošnja energije se veoma rijetko uspoređuje

te zapravo nema validnih članaka o tome koliko svaki okvir potroši energije za jedan zadatak.

Poznavajući arhitekturu i način rada pojednog okvira, može se doći do nekih zaključka vezanih za potrošnju energije.

Okviri koji koriste virtualni DOM (React) mogli bi trošiti više energije tijekom procesa usporedbe i prilikom ažuriranja stvarnog DOM-a. S druge strane, Svelte kompajlira komponente pri izgradnji u obični JavaScript čime se miče potreba za virtualnim DOM-om dovodi do uštede energije.

U kontekstu veličine paketa, veći JavaScript paketi zahtijevaju više vremena i energije za preuzimanje, procesiranje i izvođenje. Okviri koji imaju manje pakete imaju prednost u potrošnji energije. U ovoj kategoriji Svelte pobjeđuje.

Ostali troškovi više se odnose na aplikaciju koja se gradi. Ako se koristi lijeno učitavanje komponenti, lagana knjižnica za upravljanje stanjem, optimizacije u runtimeu i slično, trošiti će se manje energije.

Također, veoma je bitan način na koji je aplikacija napisana. Neučinkovito napisana Svelte aplikacija može trošiti više energije od dobro optimizirane Angular ili React aplikacije i obrnuto.

## 4.3 Specifikacija mjernog okruženja i alata

Za mjerenje potrošene električne energije, koristio se instrument EMOS P5822 Digital Power Meter. P5822 elektronički je uređaj koji služi za mjerenje potrošnje energije, a također može izračunati i grafički prikazati troškove rada mjerenog uređaja. Sučelje P5822 sadrži grafikon koji se može postaviti da prikazuje zadnjih 7 dana/tjedana/mjeseci. Uređaj je prikazan na slici 4.3


Slika 4.3 P5822 mjreni uređaj [15]

Uređaj na kojem se vršilo mjerenje je laptop HP OMEN 15-dc1059nm. U nastavku su navedene relevantne specifikacije navedenog uređaja:

- Procesor: Intel® Core™ i5-9300H (2.4 GHz base frequency, up to 4.1 GHz base with Intel® Turbo Boost Technology, 8 MB cache, 4 cores)
- Čipset: Intel® HM370

- RAM: 8 GB DDR4-2666 SDRAM (1 x 8 GB)
- Tvrdi disk: 256 GB PCIe® NVMe™ M.2 SSD
- Vrsta napajanja: 150 W AC power adapter
- Vrsta baterije: 3-cell, 52 Wh Li-ion polymer

Operacijski sustav je Windows 10 Education verzija 22H2.

Svako mjerenje provedeno je dvaput, s izuzetkom mjerenja za React s CRA, gdje je postupak mjerenja izveden tri puta. Dodatna mjerenja za React rađena su zbog visokih i neočekivanih skokova u potrošnji energije tijekom testiranja.

## 4.4 Problem blokiranja petlje događaja (event loop) u JavaScriptu

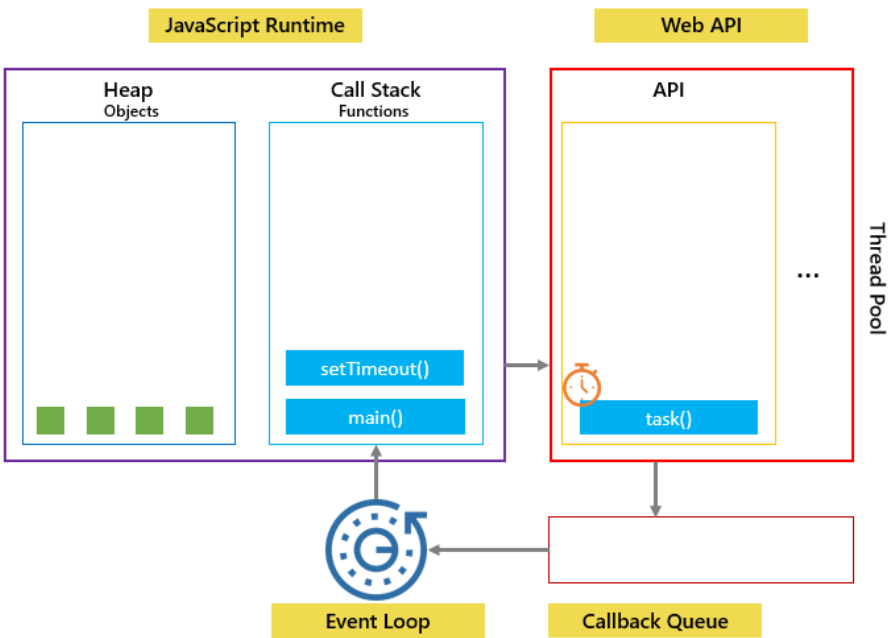
JavaScript je programski jezik koji je često na lošem glasu zbog svoje arhitekture s jednom dretvom (single-threaded). Jednodretveni jezici mogu obavljati samo jednu operaciju istovremeno. Kako bi se izbjegnuo problem asinkronosti (npr. blokiranja aplikacije za vrijeme čekanja odgovora od servera na zahtjev), koristi se koncept pod nazivom petlja događaja (event loop). Petlja događaja [16], prikazana na slici 4.4, često je na glasu neshvatljivog i zbunjujućeg koncepta, no njen princip je veoma jednostavan:

1. Ako je stog poziva (call stack) prazan, provjerava se ako nešto čeka u redu za izvršenje (callback queue).
2. Ako postoji nešto u redu za izvršavanje, sljedeća funkcija u redu prelazi na stog poziva i izvršava se.
3. Ako nema ničega u redu, JavaScript će nastaviti provjeravati dok nešto ne

postane dostupno.

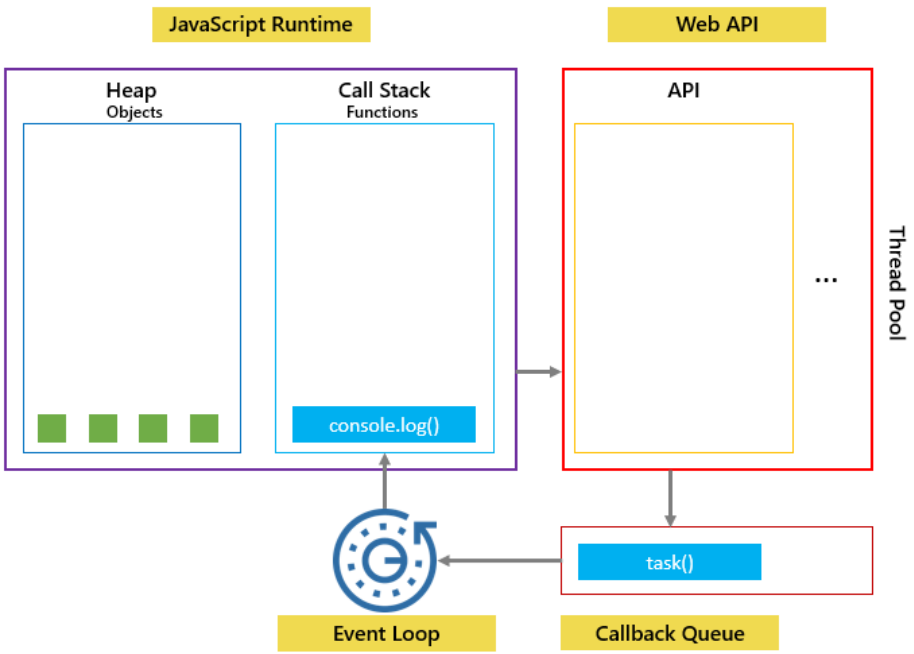
Petlja događaja neprekidno nadzire i red za izvršavanje i stog poziva.

U primjeru prikazanom na slici 4.4, prilikom pozivanja funkcije setTimeout(), JavaScript ju stavlja na stog poziva, a Web API stvara tajmer koji ističe za 1 sekundu (*task()*).




Slika 4.4 Vizualizacija petlje događaja 1 [16]

Nakon toga, JavaScript smješta funkciju *task* u red za izvršenje. Ako stog poziva nije prazan, petlja događaja čeka dok ne postane prazan te stavlja sljedeću funkciju iz reda za izvršavanje na stog poziva gdje će se i izvršiti kada dođe na red (slika 4.5). Ako je red za izvršavanje prazan, ništa se neće dogoditi.




Slika 4.5 Vizualizacija petlje događaja 2 [16]

Ako postoje operacije koje traju dugo i zauzimaju stog poziva (poput teških petlji ili intenzivnih izračuna), one blokiraju petlju događaja. To znači da se druge operacije (npr. ažuriranje UI-a) neće izvršiti dok se petlja ne odblokira. U slučaju web preglednika, kada je event loop blokiran, stranica postaje zamrznuta.

Opisano svojstvo blokiranja znatno je utjecao na mjerenja potrošnje električne energije. Izvorna ideja bila je opteretiti okvir kroz brojna ažuriranja korisničkog sučelja, s ciljem mjerenja performansi svakog okvira pri upravljanju visokofrekventnim izmjenama UI-a. Točnije, svakih 10 iteracija ažuriralo bi se trenutno najbolje rješenje na ekranu.

Međutim, zbog blokiranja petlje događaja, trenutno najbolje rješenje bi se ažuriralo tek na samom kraju izvršavanja genetskog algoritma. Iz tog razloga, potrošena energija je većinskim djelom vezana za izračune genetskog algoritma.

## 4.5 Mjerenja pri izvršavanju GA

U ovoj sekciji opisati će se i prikazati dobiveni rezultati mjerenja potrošnje pri izvršavanju genetskog algoritma 3.1. Vjerojatnost mutacije je uvijek 10%.

### 4.5.1 Populacija: 100, Duljina kromosoma: 100, Broj generacija: 10 000

Za početak, provodit će se mjerenje energetske potrošnje uz uzorak populacije od 100 jedinki, duljine kromosoma postavljene na 100, te s ukupno 10 000 generacija.

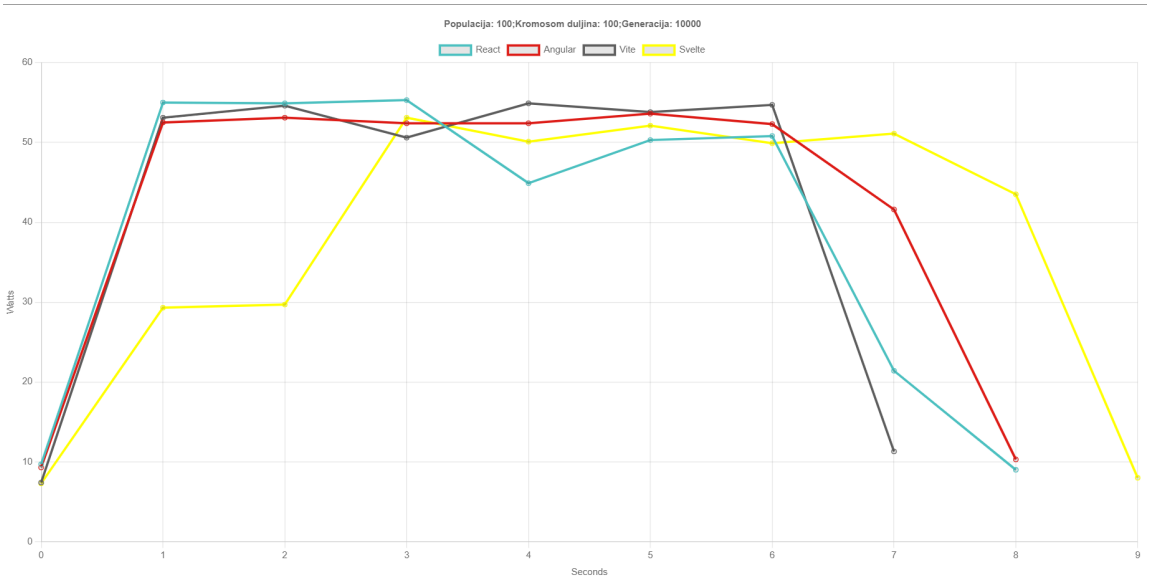
Graf 4.6 prikazuje energetsku potrošnju tijekom implementacije genetskog algoritma. Iz grafa se vidi da je Svelte imao najblaži početni intenzitet, ali isto tako i najnižu maksimalnu točku energetske potrošnje.

U sljedećoj listi, prvi argument prikazuje ukupnu potrošnju u kWh za vrijeme izvođenja algoritma, drugi argument vrijeme izrvšavanja algoritma u milisekundama, a treći je maksimalna potrošnja za vrijeme izvršvanja u wattima. Zbog toga što je vrijeme izvršavanja iznimno kratko, na konačni rezultat imaju jaki utjecaj vrijednosti potrošnje u trenutku kada program taman prestaje izvoditi algoritam. Iz tog razloga će se u ukupnu potrošnju uzimati u obzir samo one vrijednosti veće od 20W.

Rezultati mjerenja:

- **React (CRA)** Potršnja: 0.00009239 kWh; Trajanje: 6 150 ms; Maksimalna snaga: 55.3 W
- **Angular** Potrošnja: 0.00009942 kWh; Trajanje: 6 260 ms: Maksimalna snaga: 53.6 W
- **React (Vite)** Potrošnja: 0.00008936 kWh; Trajanje:5 965 ms; Maksimalna snaga: 54.9 W

- **Svelte** Potrošnja: 0.00008606 kWh; Trajanje: 6 139 ms; Maksimalna snaga: 53.1 W




Slika 4.6 Mjerenje 100;100;10000

Iz mjerenja se može izvući zaključak o tome da Svelte da troši najmanje energije što bi učvrstilo istraživanja opisana u sekciji 4.2.

Međutim, prosječno vrijeme izvršavanja predstavljenih mjerenja je 6 141 mlisekundi zbog čega ona ne predstavljaju kvalificiranog kandidata za bilo kakvo donošenje zaključaka.

### 4.5.2 Populacija: 500, Duljina kromosoma: 100, Broj generacija: 10 000

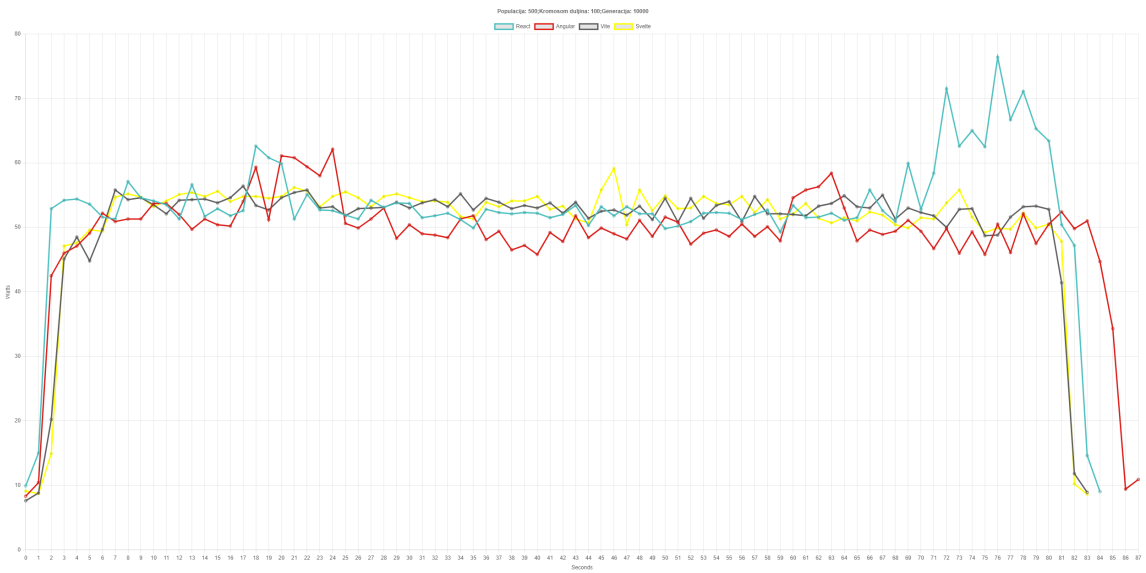
Na sveukupno potrebno vrijeme izvršavanja najveći utjecaj ima populacija. Sljedeća mjerenja koriste uzorak populacije od 500 jedinki, duljine kromosoma postavljene na 100, te s ukupno 10 000 generacija. Prosječno vrijeme izvršavanja sa navedenim

parametrima iznosilo je 80 979 miilsekundi.

Rezultati mjerenja:

- **React (CRA)** Potrošnja: 0.001246 kWh; Trajanje: 81 523 ms; Maksimalna snaga: 76.9 W
- **Angular** Potrošnja: 0.001178 kWh; Trajanje: 82 208 ms: Maksimalna snaga: 62.1 W
- **React (Vite)** Potrošnja: 0.001161 kWh; Trajanje: 79 783 ms; Maksimalna snaga: 56.4 W
- **Svelte** Potrošnja: 0.001177 kWh; Trajanje; 80 403 ms; Maksimalna snaga: 59.1 W

Nakon sagledavanja grafa 4.7 te rezultata mjerenja, može se doći do zaključka da je React aplikacija kreirana putem CRA-a najveći potrošač energije. Nasuprot tome, React aplikacija izrađena s Viteom pokazala se kao najefikasnija i najbrža. No, prema kraju mjerenja za React s CRA-om, uočavaju se značajna odstupanja od prosječne potrošnje. Radi sigurnijeg zaključivanja i izbjegavanja slučajnosti u okolini (npr., moguće da se neko pozadinsko ažuriranje pokrene tijekom mjerenja) , u nastavku se vrši dodatna analiza i pregled drugih mjerenja.




Slika 4.7 Mjerenje 500;100;10000

### 4.5.3 Populacija: 1000, Duljina kromosoma: 100, Broj generacija: 10000

Sljedeća mjerenja koriste uzorak populacije od 1000 jedinki, duljine kromosoma postavljene na 100, te s ukupno 10 000 generacija.

Prosječno vrijeme izvršavanja je 274 042 ms, što se može zaokružizi na približno četiri i pol minute. Iz tog razloga, ova mjerenja služe kao solidna podloga za izvođenje zaključaka vezanih uz potrošnju energije.

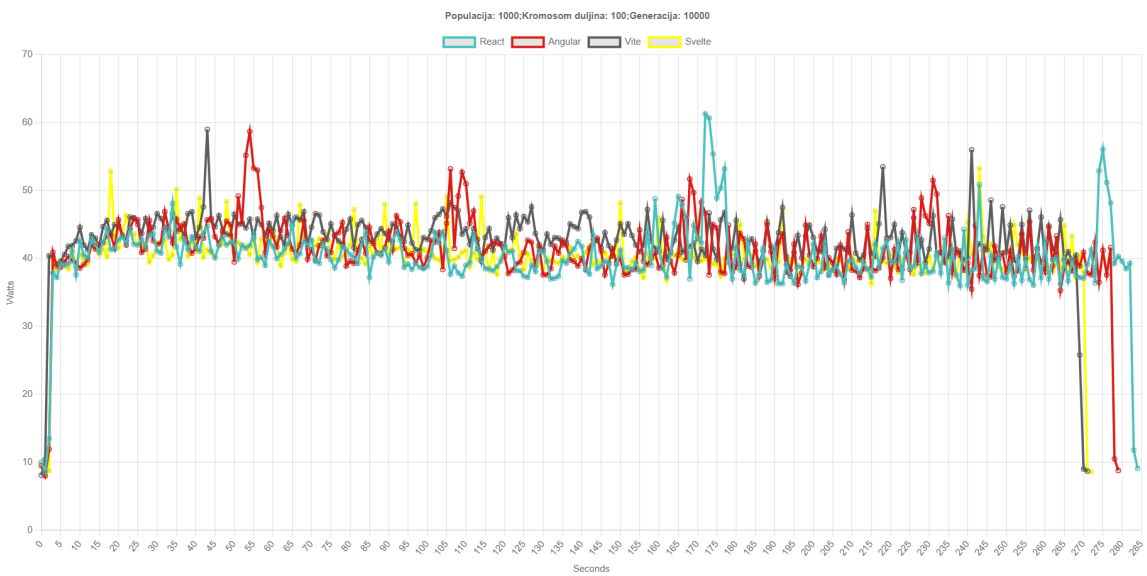
Rezultati mjerenja:

- **React (CRA)** Potrošnja: 0.003176 kWh; Trajanje: 280 555 ms; Maksimalna snaga: 61.3 W
- **Angular** Potrošnja: 0.003185 kWh; Trajanje: 280 063 ms: Maksimalna snaga: 58.7 W

- **React (Vite)** Potrošnja: 0.003221 kWh; Trajanje: 267 665 ms; Maksimalna snaga: 59 W
- **Svelte** Potrošnja: 0.304 kWh Trajanje:; 267 885 ms; Maksimalna snaga: 53.3 W

Na grafu 4.8 vidi se da je React (CRA) opet imao najviše primjetnih skokova, također je imao i najveću vrijednost maksimalne potrošnje u jednoj sekundi. Svelte je ponovno imao najnižu maksimalnu potrošnju u jednoj sekundi te je također potrošio najmanje energije kroz cijelo vrijeme izvršavanja algoritma.

Prosječno vrijeme izvršavanja je 274 042 ms, što se može zaokružiti na približno četiri i pol minute. Iz tog razloga, ova mjerenja služe kao solidna podloga za izvlačenje zaključaka vezanih uz potrošnju energije.




Slika 4.8 Mjerenje 1000;100;10000

### 4.5.4 Populacija: 100, Duljina kromosoma: 1000, Broj generacija: 10000

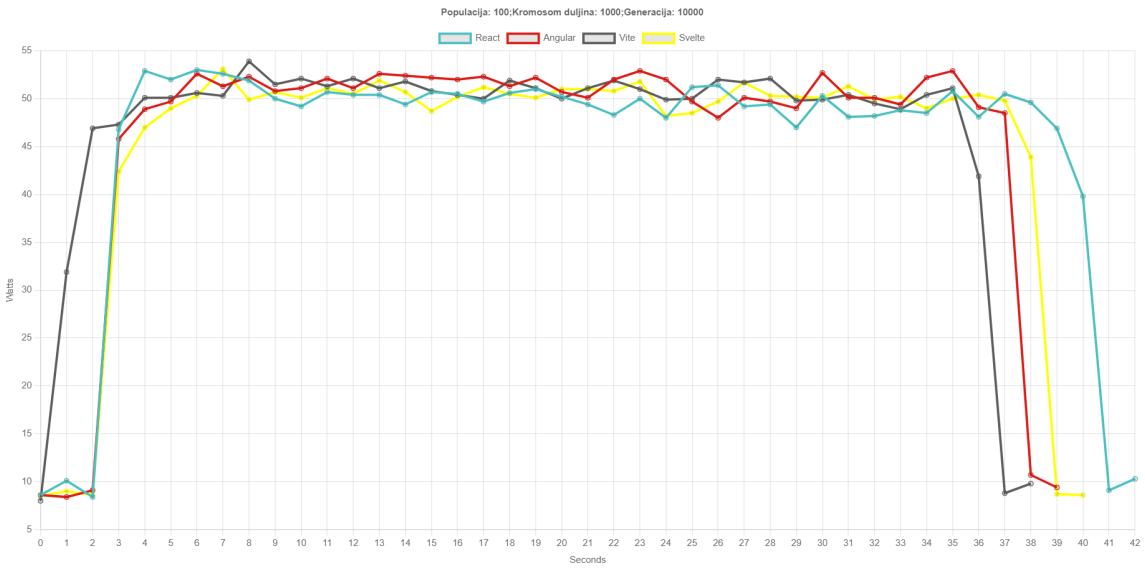
Sljedeća mjerenja koriste uzorak populacije od 100 jedinki, duljine kromosoma postavljene na 1000, te s ukupno 10,000 generacija.

Prosječno vrijeme izvršavanja je 35960 ms, odnosno približno 36 sekundi.

Rezultati mjerenja:

- **React (CRA)** Potrošnja: 0.0005428 kWh; Trajanje: 38 716 ms; Maksimalna snaga: 53 W
- **Angular** Potrošnja: 0.0004944 kWh; Trajanje: 35 062 ms: Maksimalna snaga: 52.9 W
- **React (Vite)** Potrošnja: 0.0004991 kWh; Trajanje: 34 892 ms; Maksimalna snaga: 53.9 W
- **Svelte** Potrošnja: 0.0004987 kWh; Trajanje: 35 173 ms; Maksimalna snaga: 53.1 W

Iz grafa 4.9 ne može se izvući relevantan zaključak, jer su gotovo sve instance istog ponašanja bez nekih anomalija. Angular je ovoga puta, začuđujuće, najmanje energije potrošio te je imao i najmanju potrošnju u jednoj sekundi.




Slika 4.9 Mjerenje 100;1000;10000

### 4.5.5 Populacija: 100, Duljina kromosoma: 100, Broj generacija: 100000

Sljedeća mjerenja koriste uzorak populacije od 100 jedinki, duljine kromosoma postavljene na 100, te s ukupno 100,000 generacija.

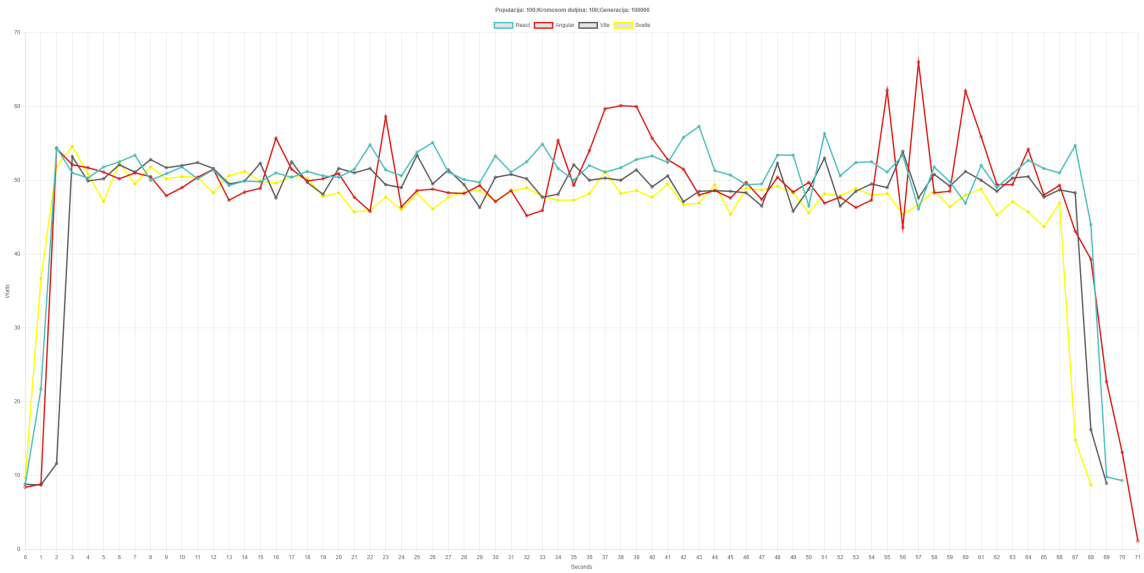
Prosječno vrijeme izvršavanja je 65254 ms, odnosno približno 65 sekundi.

Rezultati mjerenja:

- **React (CRA)** Potrošnja: 0.0009641 kWh; Trajanje: 64 983 ms; Maksimalna snaga: 57.3 W
- **Angular** Potrošnja: 0.0009471 kWh; Trajanje: 65 471 ms: Maksimalna snaga: 66 W
- **React (Vite)** Potrošnja: 0.0009021 kWh; Trajanje: 65 216 ms; Maksimalna snaga: 53.9 W

- **Svelte** Potrošnja: 0.0008834 kWh; Trajanje: 65 349 ms; Maksimalna snaga: 54.6 W

Iako je React (CRA) u prijašnjim mjerenjima imao najviše skokova, u ovom mjerenju je Angular imao čak šest primjetnih skokova te je imao daleko veću maksimalnu potrošnju energije u jednoj sekundi od ostalih okvira. React (CRA) je, čak i uz svoju relativno stablinu putanju potrošnje u ovom mjerenju, potrošio najviše energije kroz cijelo izvršavanje.




Slika 4.10 Mjerenje 100;100;100000

Na temelju analize energetske potrošnje tijekom izvođenja genetskog algoritma, utvrđeno je da izabrani okvir (framework) nema značajan utjecaj na energetsku potrošnju kada se obavljaju operacije koje ne opterećuju korisničko sučelje intenzivno. Razlike u potrošnji vidljive na grafovima i iščitane iz rezultata mjerenja najvećim djelom posljedica su različitih veličina paketa, dodatnih biblioteka ili vanjskih čimbenika što je minimizirano višestrukim mjerenjem.

Veličine projekta:

- React (CRA): 277 MB
- Angular: 325 MB
- React (Vite): 61,6 MB
- Svelte: 19 MB

Iako je razlika u veličinama paketa drastična, iz rezultata je vidljivo da je utjecaj veličine primjetljiv, ali minimalan. Okviri sa većim paketima, Angular i React (CRA), imali su primjetne skokove u potrošnji te su u većini slušajeva potrošili najviše energije tijekom izvođenja algoritma.

Svelte se općenito pokazao kao najučinkovitiji okvir s obzirom na potrošnju energije. Međutim, razlika u energetskoj učinkovitosti između Sveltea i ostalih okvira postaje manje primjetna s većim uzorcima, što ukazuje da za duže zadatke, izbor okvira nema značajno velik utjecaj na energetsku potrošnju.

Iako se React (CRA) pokazao kao najveći energetski potrošač, imao je najviše neočekivanih skokova što ukazuje na to da postoje nekakve pozadinske operacije unutar React okvira koje mogu dovesti do varijacija u potrošnji, ili su uzrok vanjskih čimbenika.

Kada je algoritram u stanju izvršavanja, uglavnom se radi o izvođenju sirovog JavaScripta. Razlika u potrošnji energije pri izvođenju JavaScripta biti će otprilike ista u svim okvirima.

## 4.6 Mjerenja pri opterećenju korisničkog sučelja

U ovoj sekciji opisati će se i prikazati dobiveni rezultati mjerenja tijekom izvršavanja aplikacije koja opterećuje korisničko sučelje 3.5.

Sva mjerenja rađena su 2 puta. Prikazani rezultati na grafovima i listama rezultati

su srednjih vrijednosti ta 2 mjerenja.

Rezultati mjerenja:

- **React (CRA)** Potrošnja: 0.0002293 kWh; Trajanje: 19247 ms; Maksimalna snaga: 45 W
- **Angular** Potrošnja: 0.0006732 kWh; Trajanje: 67 792 ms: Maksimalna snaga: 44.2 W
- **React (Vite)** Potrošnja: 0.0002331 kWh; Trajanje: 19 260 ms; Maksimalna snaga: 48.6 W
- **Svelte** Potrošnja: 0.0001222 kWh; Trajanje: 10 356 ms; Maksimalna snaga: 40.4 W

Analizom grafa 4.11 može se primijetiti kako su obje instance Reacta (i CRA i Vite) imali veoma sličnu putanju potrošnje, gotovo isto vrijeme izvršavanja i veoma bliske vrijednosti ukupno potrošene energije. Također, instance Reacta istaknule su se najvećom potrošnjom energije u jednoj sekundi.

Iako se varijacije u potrošnji po sekundi nisu pretjerano razlikovale među različitim okvirima, ukupno vrijeme izvršavanja i time ukupna potrošnja energije drastično se razlikovala.

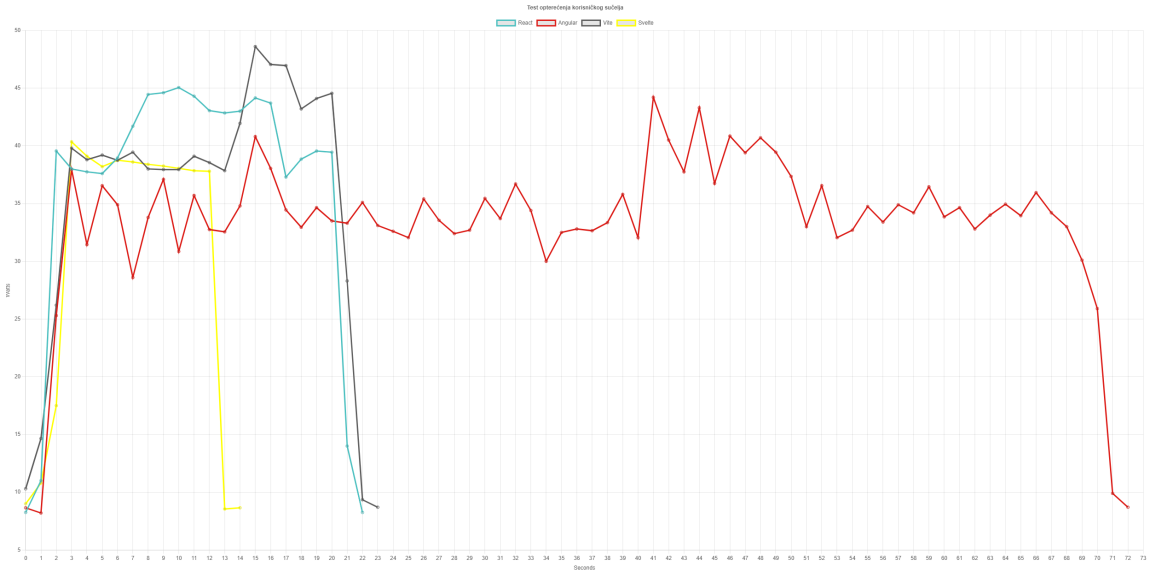
Od Sveltea se, zbog nedostatka virtualnog DOM-a i kompajlerske arhitekture, očekivalo da će s najmanje problema rukovati silnim promjenama na korisničkom sučelju. Rezultati mjerenja idu u korist ovim očekivanjima - Svelte je izvršio algoritam više od šest puta brže od Angulara i gotovo dva puta brže od Reacta (i CRA i Vite).

Nadalje, Svelte je potrošio oko 82% manje energije u usporedbi s Angularom i oko 47% manje u odnosu na oba React okvira.

S druge strane, Angular je pokazao najgore performanse s najvećom ukupnom potrošnjom energije i najdužim vremenom izvršavanja. Angularovo sporo vrijeme

izvršavanja velikim je dijelom posljedica njegovog mehanizma detekcije promjena.

Angular koristi mehanizam za detekciju promjena koji automatski provjerava cijelo stablo komponenata kada se dogodi neka promjena. Ako postoji velik broj komponenata, kao u ovom slučaju, ovakva provjera može biti skupa po pitanju performansi. Dodatno, Angular često dolazi s više standardnog (eng. boilerplate) koda u usporedbi s Reactom i Svelteom što isto loše utječe na performanse.




Slika 4.11 Mjerenje potrošnje pri opterećenju UI-a

# Poglavlje 5

# Zaključak

Cilj ovog istraživanja bio je pronaći energetski najučinkovitiji okvir između Reacta, Angulara i Sveltea za razvoj web aplikacija. Usporedba se vršila mjerenjem potrošnje energije tijekom izvršavanja genetskog algoritma te mjerenjem potrošnje energije tijekom operacije koja opterećuje korisničko sučelje.

Genetski algoritam je optimizacijska metoda inspirirana procesom prirodne selekcije koji korištenjem metoda poput selekcija, križanja i mutacije generira nove populacija kandidata, odnsosno rješenja. Cilj je pronaći najbolje rješenje za određeni problem kroz više generacija.

Izvršavanjem genetskog algoritma, prilikom čega dolazi do značajnog opterećenja aplikacije, stvara se dobra podloga za mjerenje potrošnje električne energije. Također, vrijeme i intenzitet izvršavanja može se diktirati promjenom parametara populacije, duljine kromosoma te broja generacija.

Iako je svaki okvir različit u načinu rada i arhitekturi, kada se radi o energetskoj efikasnosti prilikom resursno intenzivnih operacija koje ne opterećuju korisničko sučelje, sva tri okvira pokazala su slične rezultate s time da je Svelte pokazao neznatno bolje performanse.

Prilikom istraživanja učinkovitosti različitih okvira u kontekstu operacija s visokim opterećenjem korisničkog sučelja, razlike u performansama postaju iznimno značajne. Svelte se izdvaja kao najefikasniji u ovoj kategoriji, dok Angular zauzima posljednje mjesto.

U današnjem vremenu gdje su resursi veoma obilni, teme poput memorije i energetske potrošnje ne spominju se često. U okruženjima gdje je maksimalna efikasnost kljjčna, ovo istraživanje ukazuje na prednost Sveltea nad Reactom i Angularom. Međutim, prilikom odabira odgovarajućeg alata potrebno je uzeti u obzir i ostale parametre poput prethodnog iskustva tima, specifičnih zahtjeva projekta i drugih faktora. Iako svaki od ovih okvira ima svoje prednosti i izazove, svaki od njih je sposoban razviti pouzdano i efikasno rješenje.

# Popis slika

# Popis kodnih isječaka

*Popis kodnih isječaka*

# Bibliografija


[1] React. , s Interneta, https://react.dev/ posjećeno srpanj 2023.

[2] React kuke. , s Interneta, https://legacy.reactjs.org/docs/hooks-overview.html posjećeno srpanj 2023.

[3] Angular. , s Interneta, https://angular.io/ posjećeno srpanj 2023.

[4] Angular direktive. , s Interneta, https://angular.io/guide/built-in-directives posjećeno srpanj 2023.

[5] Angular arhitektura. , s Interneta, https://www.interviewbit.com/blog/angular-architecture/ posjećeno srpanj 2023.

[6] Svelte. , s Interneta, https://svelte.dev/ posjećeno kolovoz 2023.

[7] Popularnost web okvira. , s Interneta, https://stackdiary.com/front-end-frameworks/ posjećeno kolovoz 2023.

[8] Popularnost i voljenost okvira. , s Interneta, https://gist.github.com/tkrotoff/b1caa4c3a185629299ec234d2314e190 posjećeno kolovoz 2023.

[9] Create React App. , s Interneta, https://create-react-app.dev/ posjećeno kolovoz 2023.

[10] Vite. , s Interneta, https://vitejs.dev/ posjećeno kolovoz 2023.

[11] Bootstrap. , s Interneta, https://getbootstrap.com/docs/5.3/getting-started/introduction/ posjećeno kolovoz 2023.

[12] Genetski algoritam. , s Interneta, https://towardsdatascience.com/introduction-to-genetic-algorithms-including-example-code-e396e98d8bf3 posjećeno kolovoz 2023.

[13] Genetski Algoritam YouTube video. , s Interneta, https://www.youtube.com/watch?v=uQj5UNhCPuo&t=354s posjećeno kolovoz 2023.

[14] Usporedba okvira. , s Interneta, https://www.softermii.com/blog/why-sveltejs-is-the-most-in-demand-framework-for-web-development posjećeno kolovoz 2023.

[15] P5822. , s Interneta, https://www.mall.hr/ampermetri/emos-mjerac-potrosnje-elektricne-energije-schuko-p5822 posjećeno kolovoz 2023.

[16] Petlja događaja. , s Interneta, https://www.javascripttutorial.net/javascript-event-loop/ posjećeno kolovoz 2023.

# Sažetak

Ovaj rad proučava i uspoređuje tri vrlo popularna okvira za razvoj web aplikacija (React, Svelte, Angular). Usporedba se temelji na više kriterija, uključujući performanse, brzinu izvođenja, jednostavnost učenja i poseban naglasak stavljen je na efikasnost prilikom izvršavanja operacija. Kako bi se ocijenila efikasnost, korišten je uređaj EMOS P5822 koji služi za mjerenje potrošnje energije. Mjerenja su rađena za vrijeme izvođenja genetskog algoritma koji intenzivno koristi resurse računala, što ga čini odgovarajućim za ovu svrhu.

***Ključne riječi*** **— okvir, React, Angular, Svelte, potrošnja, genetski algoritam**

# Abstract

This paper examines and compares three highly popular frameworks for web application development. The comparison is based on multiple criteria, including performance, execution speed, and ease of learning, emphasizing efficiency during execution operations. In order to assess efficiency, the EMOS P5822 device was utilized for measuring energy consumption. Measurements were conducted during the execution of a genetic algorithm that heavily utilizes computer resources, rendering it suitable for this purpose.

***Keywords*** **— framework, React, Angular, Svelte, consumption, genetic algorithm**