PANEVROPSKI UNIVERZITET APEIRON, BANJA LUKA
FAKULTET INFORMACIONIH TEHNOLOGIJA

Redovne studije
Smjer : "Inženjering Informacionih Tehnologija"

Predmet:
Viši programski jezici i RAD alati

Pregled i praktična implementacija Svelte framework-a u modernim web projektima

(Seminarski rad)

Predmetni nastavnik
Prof. dr Branimir Trenkić

Student: Vukoman Boris
Br. Indeksa : 101-20/RITP-S

Banja Luka, Avgust 2025

SADRŽAJ

UVOD

U posljednjih nekoliko godina, razvoj web aplikacija postao je složen i zahtjevan proces koji traži efikasne, inovativne i skalabilne pristupe za izgradnju brzih i responzivnih rješenja. Pored tradicionalnih tehnologija, pojavili su se moderni frontend frameworkovi koji olakšavaju kreiranje interaktivnih korisničkih interfejsa i optimizuju performanse. Među njima, Svelte se ističe kao revolucionaran alat jer uvodi jedinstven pristup – umjesto da radi u runtime-u poput Reacta, Vue-a ili Angulara, Svelte kompajlira kod u efikasan JavaScript koji direktno manipuliše DOM-om, čime se smanjuju veličina fajlova i vrijeme učitavanja stranica.

Ovaj rad detaljno istražuje Svelte framework, uključujući njegovu istoriju, razvojne verzije i ključne saradnike, kao i strukturu Svelte komponenti koje integrišu HTML, CSS i JavaScript u jednom fajlu. Poseban fokus stavljen je na načine integracije sa backend tehnologijama, uključujući API komunikaciju, autentifikaciju, autorizaciju i real-time WebSocket interakcije.

Kroz analizu prednosti i mana Svelte-a, rad pruža uvid u njegove konkurentske karakteristike u poređenju sa drugim popularnim frameworkovima i ističe situacije u kojima Svelte može značajno unaprijediti razvoj modernih web aplikacija. Na taj način, cilj je pružiti kompletnu sliku potencijala i ograničenja ovog frameworka, omogućavajući programerima i timovima da donesu informisane odluke pri izboru tehnologije za frontend razvoj.

1. Svelte Framework

„Svelte predstavlja besplatan i open-source okvir zasnovan na komponentama, namijenjen za razvoj modernih web aplikacija i dinamičnih korisničkih interfejsa. Autor ovog alata je Rich Harris, a kontinuirano ga unapređuje i održava zvanični Svelte core tim, uz doprinos široke zajednice programera. Za razliku od većine popularnih JavaScript biblioteka i frameworka, poput Reacta ili Vue-a, koji zahtijevaju da se glavni dio biblioteke učita u preglednik prilikom pokretanja aplikacije, Svelte koristi pristup kompajliranja. To znači da se aplikacija prevodi unaprijed, pri čemu se HTML šabloni transformišu u optimizovan JavaScript kod koji direktno manipuliše DOM-om, bez dodatnog sloja interpretacije.“ [1].

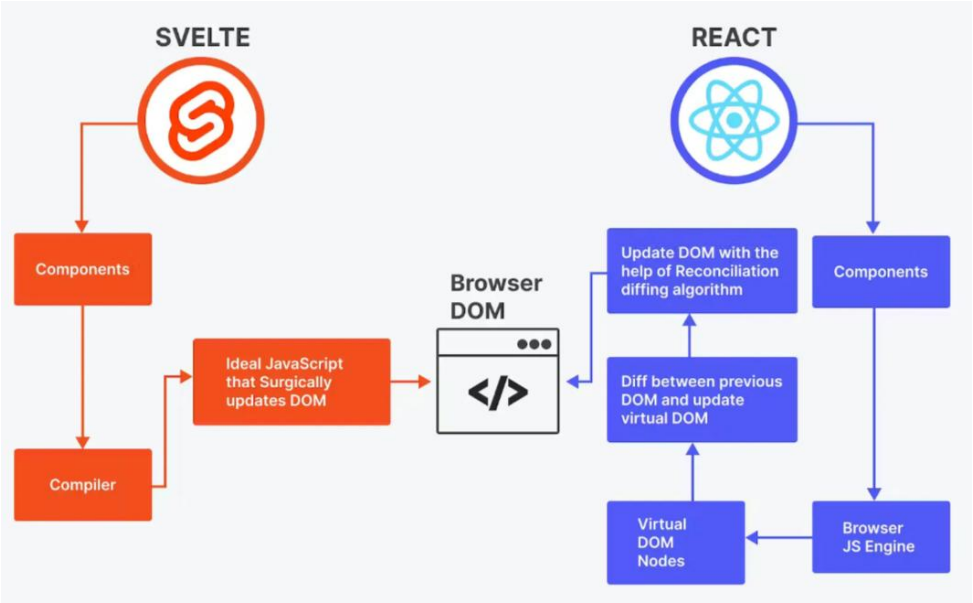
„Ovakav način rada ima više prednosti. Prvo, veličina fajlova koji se šalju korisniku je manja, što rezultira bržim učitavanjem stranice i smanjenim opterećenjem mreže. Drugo, ovakav kod omogućava bolje performanse, jer izbjegava nepotrebne apstrakcije i slojeve koje druge biblioteke uvode. Treće, Svelte kompajler automatski implementira reaktivnost kada se podaci promijene, odgovarajući elementi korisničkog interfejsa se trenutno ažuriraju bez potrebe za ručnim pozivima funkcija za ponovno renderovanje.

Još jedna značajna prednost Sveltea je jednostavnost pisanja koda. Programeri mogu kombinovati HTML, CSS i JavaScript unutar jedne .svelte datoteke, što olakšava organizaciju projekta i smanjuje potrebu za razdvajanjem logike i prezentacije. Osim toga, Svelte podržava napredne funkcionalnosti poput CSS scoping-a, tranzicija, animacija i bindovanja podataka sa minimalnom sintaksom.

Zbog svoje filozofije da većinu posla obavi u fazi kompajliranja, Svelte je stekao reputaciju alata koji omogućava izradu izuzetno brzih aplikacija sa malim zahtjevima prema resursima, što ga čini pogodnim ne samo za klasične web stranice, već i za progresivne web aplikacije (PWA), dashboard rješenja i čak integracije u mobilne hibride.“ [1].

Slika 1 - Svelte logo i logotip

„Ovakav pristup u potpunosti eliminiše potrebu za međureprezentacijama tokom izvršavanja aplikacije, poput virtuelnog DOM-a, koji koriste tradicionalni okviri kao što su React i Vue. Za razliku od njih, gdje se većina logike obrađuje u samom pregledniku u realnom vremenu, Svelte sav taj posao obavlja u fazi kompajliranja. Time se smanjuje potrošnja memorije, ubrzava renderovanje elemenata i izbjegavaju dodatni slojevi apstrakcije koji mogu usporiti aplikaciju. “ [1].



Slika 2 - Razlika manipulacije DOM-a korištenjem Svelte Framework-a i korištenjem React Framework-a

1.1. Istorija Svelte Framework-a

„Prije nego što je nastao Svelte, njegov tvorac Rich Harris 2013. godine razvio je biblioteku Ractive.js, koja je poslužila kao svojevrsni temelj za kasniji razvoj ovog okvira. Prva zvanična verzija Sveltea objavljena je 29. novembra 2016. godine i bila je napisana u JavaScriptu. U suštini, radilo se o Ractive-u obogaćenom kompajlerom, a ime “Svelte” odabrali su Harris i njegovi kolege iz redakcije lista The Guardian.

Druga verzija Sveltea ugledala je svjetlo dana 19. aprila 2018. godine, donoseći niz poboljšanja i ispravki grešaka iz prve verzije, uključujući promjenu sintakse dvostruke vitičaste zagrade zamijenjene su jednostrukim, što je pojednostavilo kod i poboljšalo čitljivost.

Slika 3 - Rich Harris

Treća verzija, objavljena 21. aprila 2019. godine, donijela je značajnu promjenu, okvir je prepisan u TypeScriptu. Ova verzija je redefinisala koncept reaktivnosti, uvodeći sistem u kojem kompajler automatski upravlja promjenama vrijednosti varijabli, omogućavajući programerima da postignu reaktivno ponašanje bez dodatnog koda.

U oktobru 2020. najavljen je SvelteKit, prateći okvir namijenjen izradi potpunih web aplikacija. Već u martu 2021. SvelteKit je ušao u beta fazu, a ubrzo je postao standardni izbor za razvoj složenijih projekata baziranih na Sveltu.

Verzija 4 Sveltea lansirana je 22. juna 2023. godine, sa ciljem optimizacije i smanjenja veličine koda u odnosu na prethodnu verziju. Iako je i dalje korišten TypeScript, interni kod je konvertovan nazad u JavaScript uz upotrebu JSDoc anotacija, što je izazvalo određenu zabunu u zajednici. Na ovu kontroverzu lično je odgovorio Rich Harris, objašnjavajući razloge za promjenu.

Kroz godine, Svelte je imao niz važnih saradnika. Među njima su Conduitry, koji se pridružio još u vrijeme prve verzije, Tan Li Hau (2019), te Ben McCann (2020). Godine 2022. Rich Harris i Simon Holthausen pridružili su se kompaniji Vercel kako bi punim radnim vremenom radili na Sveltu, a 2023. timu se priključio i Dominic Gannaway, bivši član React core tima, dodatno osnaživši razvoj ovog okvira.“ [1].

1.2. Upotreba Svelte Framework-a

„Svelte se može primijeniti i za razvoj pojedinačnih komponenti korisničkog interfejsa i za izgradnju čitavih aplikacija. Programeri mogu započeti projekat od nule, dopuštajući Sveltu da preuzme potpunu kontrolu nad UI-jem, ili ga postepeno dodavati u postojeće projekte bez potrebe za radikalnom promjenom arhitekture.

Posebno dolazi do izražaja u sljedećim scenarijima:

- Web aplikacije namijenjene uređajima sa ograničenim resursima – Zahvaljujući maloj veličini generisanog koda, Svelte aplikacije se brže učitavaju i troše manje procesorske snage, što ih čini pogodnim za uređaje sa sporim internet vezama ili slabijim hardverom. Manje linija koda znači manju količinu podataka za preuzimanje, parsiranje i izvršavanje, a samim tim i nižu potrošnju memorije.

- Aplikacije sa visokim stepenom interaktivnosti i složenim vizualizacijama – Kada je potrebno istovremeno prikazati veliki broj DOM elemenata ili brzo reagovati na korisničke akcije, izostanak virtuelnog DOM-a i minimiziranje runtime opterećenja omogućavaju fluidno i responzivno iskustvo.

- Brza obuka novih programera – S obzirom na to da Svelte koristi sintaksu vrlo sličnu standardnom HTML-u, CSS-u i JavaScriptu, osobe sa osnovnim znanjem web tehnologija mogu ga savladati u kratkom vremenskom periodu i odmah početi razvijati aplikacije.

- Progresivne web aplikacije (PWA) – Kombinacija male veličine paketa, brzog učitavanja i mogućnosti rada u offline režimu čini Svelte odličnim izborom za PWA rješenja, koja na mobilnim uređajima nude gotovo nativno korisničko iskustvo.

- Server-side rendering (SSR) – Uz pomoć SvelteKita, programeri mogu renderovati sadržaj na strani servera, što skraćuje vrijeme prvog prikaza stranice i poboljšava SEO rezultate, što je posebno važno za dinamične sajtove poput blogova ili e-commerce platformi.

- IoT kontrolne ploče – Male aplikacije optimizovane za brz prikaz podataka savršeno odgovaraju zahtjevima IoT sistema, gdje su niska potrošnja resursa i brzo osvježavanje ključni faktori.
- Animacije i tranzicije – Svelte ima ugrađene mehanizme za kreiranje složenih vizuelnih efekata bez potrebe za dodatnim bibliotekama, što je idealno za multimedijalne projekte i interaktivne prezentacije.

Kako bi se upotpunila njegova primjena u profesionalnom razvoju aplikacija, tim koji stoji iza Sveltea je kreirao SvelteKit, napredni okvir koji donosi funkcionalnosti modernih web platformi, uključujući rutiranje zasnovano na strukturi fajlova, server-side rendering, različite načine renderovanja pojedinačnih stranica, offline podršku i druge optimizacije koje olakšavaju rad na većim projektima.“ [2].

1.3. Svelte vs. React, Vue, Angular Framework-ova

„U poređenju s najpoznatijim okvirima za web razvoj, poput Reacta, Angulara i Vuea, Svelte se izdvaja specifičnim pristupom i arhitekturom. Dok svaki od ovih alata ima svoje jasne prednosti i oblasti primjene, ključna razlika je u tome što Svelte nije klasični framework koji se oslanja na biblioteku učitanu u preglednik, već kompajler koji generiše optimizovani JavaScript kod još tokom procesa izgradnje aplikacije.

Slika 4 - Ranging Framework-a tokom proteklih par godina i njihove upotrebe

React se često bira zbog ogromne i aktivne zajednice, te velikog ekosistema dodatnih biblioteka i gotovih rješenja. Angular uživa reputaciju stabilnog, sveobuhvatnog okvira koji dolazi s moćnim alatima za kompleksne projekte, dok Vue pruža balans između jednostavnosti i fleksibilnosti, što ga čini privlačnim za brzi razvoj. Međutim, svi oni obrađuju značajan dio logike u runtime-u, za razliku od Sveltea koji taj posao obavlja unaprijed.

Ovaj pristup rezultira značajnim performansnim prednostima, aplikacije razvijene u Sveltu obično se učitavaju brže i reagiraju odzivnije, jer ne nose "balast" cijele biblioteke tokom rada u pregledniku. Umjesto slojeva poput virtuelnog DOM-a, Svelte generiše kod koji direktno manipuliše DOM-om, čime se smanjuje potrošnja memorije i vrijeme renderovanja.

Dodatno, Svelte se ističe jednostavnošću koda. Zahvaljujući svojoj sintaksi i ugrađenoj reaktivnosti, programerima je često potrebno znatno manje linija koda za postizanje iste funkcionalnosti nego u Reactu, Angularu ili Vueu. Manje koda ne znači manje mogućnosti upravo suprotno, rezultira preglednijim i lakše održivim projektima, s manjim prostorom za greške i bržim razvojnim ciklusima. " [3]

2. Svelte komponente i sintaksa

2.1. Struktura Svelte komponenti

"U Sveltu, osnovni građevni blok svake aplikacije je komponenta. Komponente se mogu posmatrati kao samostalne jedinice koje u sebi objedinjeno sadrže strukturu (HTML), stilove (CSS) i logiku (JavaScript) sve u jednom fajlu sa ekstenzijom .svelte. Ovakav pristup omogućava lakšu organizaciju koda i veću preglednost projekta, jer su svi dijelovi vezani za jednu funkcionalnost smješteni na jednom mjestu.

HTML – Definiše vizuelnu strukturu korisničkog interfejsa. Ovdje se kreiraju elementi koje korisnik vidi i s kojima može interagovati. Na primjer:

```
<h1>Hello, {name}!</h1>
```

CSS – Sadrži stilove koji su izolovani isključivo na nivo te komponente, što znači da neće nenamjerno mijenjati izgled drugih dijelova aplikacije. Ovakva lokalna primjena stilova eliminiše potrebu za komplikovanim konvencijama imenovanja klasa. Primjer:

```
<style>
  h1 {
    color: blue;
  }
</style>
```

JavaScript – U ovom dijelu se definiše logika komponente: varijable, funkcije, importi drugih modula, te reaktivne vrijednosti koje automatski ažuriraju prikaz u interfejsu kada im se promijeni sadržaj. Primjer:

```
<script>
  export let name = 'World';
</script>
```

Ovakva organizacija olakšava rad i početnicima i iskusnim programerima, jer omogućava intuitivan tok razvoja od definisanja strukture, preko dodavanja stila, do implementacije funkcionalnosti, sve unutar jednog preglednog fajla.” [4]

2.2. Bindovanje podataka, događaji i stilizacija

„U Sveltu, bindovanje (engl. binding) omogućava uspostavljanje dvosmjerne veze između vrijednosti u JavaScript kodu i elemenata korisničkog interfejsa. To znači da, kada promijenite vrijednost varijable u kodu, prikaz u interfejsu se automatski osvježi, a kada korisnik izmijeni vrijednost u polju (npr. u tekstualnom inputu), ta promjena se odmah prenese u varijablu.

Primjer jednostavnog ***two-way bindinga***:

```
<input bind:value={name}>
<p>Your name is: {name}</p>
```

Ova funkcionalnost uklanja potrebu za ručnim pisanjem dodatnog koda za praćenje i ažuriranje vrijednosti, što razvoj čini efikasnijim i čitljivijim.

Za upravljanje događajima, Svelte koristi direktivu on:event, koja omogućava lako povezivanje korisničkih akcija (klik, promjena u formi, pritisak tipke itd.) s funkcijama koje obrađuju te događaje.

Primjer ***rukovanja klikom na dugme***:

```
<button on:click={handleClick}>Click me</button>

<script>
  function handleClick() {
    alert('Button clicked!');
  }
</script>
```

Ovaj pristup je jednostavan i intuitivan, događaji su jasno vezani uz elemente u samom HTML-u, bez potrebe za dodatnim slojevima koda.

Stilizacija u Sveltu je lokalizovana po komponentama. Stilovi napisani unutar <style> taga važe samo za elemente u toj komponenti, što eliminiše konflikte sa stilovima iz drugih dijelova aplikacije. Osim statičnih stilova, moguće je primjenjivati i dinamičke klase i stilove, gdje se CSS pravila aktiviraju ili deaktiviraju na osnovu vrijednosti varijabli.

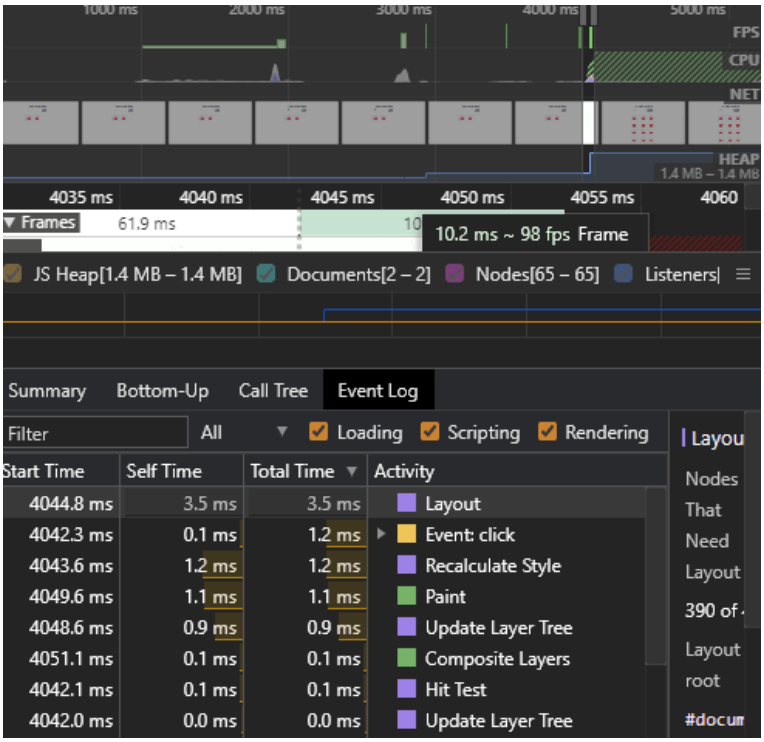
```
<style>
  .highlight {
    background-color: yellow;
  }
</style>

<p class:highlight={isHighlighted}>This is a highlighted paragraph.</p>

<script>
  let isHighlighted = true;
</script>
```

Ovakav mehanizam omogućava da se vizuelne promjene u interfejsu odvijaju potpuno automatski, bez ručnog dodavanja ili uklanjanja klasa pomoću JavaScript-a.

3. Prednosti i mane Svelte Framework-a

3.1. Prednosti Svelte-a

Visoke performanse – Svelte se izdvaja po svom kompajlerskom pristupu, gdje se komponente prevode u optimizovani JavaScript kod koji direktno upravlja DOM-om. Ovaj metod eliminiše potrebu za virtuelnim DOM-om i smanjuje overhead tokom izvršavanja, što omogućava brže i efikasnije ažuriranje korisničkog interfejsa.

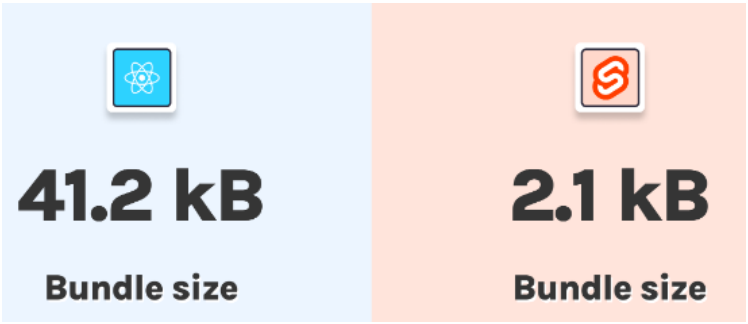


Slika 5 - Test Svelte performansi u Devtools Chrome pretraživaču

Svelte takođe generiše manji izlazni kod u poređenju sa mnogim drugim framework-ovima. Samo funkcionalnosti koje su zaista potrebne u aplikaciji bivaju uključene u konačni JavaScript fajl, što rezultira manjim paketima, bržim učitavanjem stranica i boljim performansama na uređajima sa ograničenim resursima.

Jednostavna i intuitivna sintaksa – Svelte objedinjuje HTML, CSS i JavaScript u okviru jedne komponente, što čini kod preglednijim i lakšim za razumijevanje. Ova struktura olakšava

održavanje i smanjuje složenost, jer programeri ne moraju razdvajati logiku, stilove i strukturu na različite fajlove.



Slika 6 - Razlika u veličini datoteka između Svelte i React framework-a

Niska krivulja učenja – S obzirom na jednostavnu sintaksu i minimalan broj naprednih koncepata, Svelte je pristupačan programerima sa osnovnim znanjem web tehnologija. To ga čini idealnim za brzi razvoj prototipova ili uvođenje novih članova tima u projekt, bez potrebe za dugotrajnim učenjem kompleksnih paradigma.

Efikasnost razvoja – Kombinacija reaktivnosti, dvosmjernog bindovanja i lokalizovanih stilova omogućava da se većina interaktivnosti implementira sa vrlo malo koda, čime se smanjuje rizik od grešaka i ubrzava razvojni proces.

```
<script>
  let items = [
    { id: 1, name: "Milk", done: false },
    { id: 2, name: "Bread", done: true },
    { id: 3, name: "Eggs", done: false }
  ];

  const remove = item => {
    items = items.filter(i => i !== item);
  };
</script>

<div>
  <h1>Things to Buy</h1>

  <ul>
    {#each items as item}
      <li>
        <span>{item.name}</span>
        <button on:click={() => remove(item)}>❌</button>
      </li>
    {/each}
  </ul>
</div>
```

Slika 7 - Primjer sintakse Svelte-a

3.2. Mane Svelte-a

Jedna od glavnih slabosti Sveltea je njegova relativna novost u poređenju s dugogodišnjim framework-ovima poput Reacta i Vuea. Zbog toga ekosistem biblioteka, dodataka i alata još nije tako razvijen, što može otežati pronalaženje specifičnih paketa ili rješenja koja su odmah dostupna u drugim okvirima.

Kompajlerski pristup, iako donosi značajne performanse, može uvesti dodatni nivo kompleksnosti u razvojni proces. Programeri moraju razumjeti kako Svelte kompajlira kod i pravilno konfigurirati build alate, što može biti izazovno za one koji preferiraju jednostavnije, runtime-orijentisane frameworkove.

Manja rasprostranjenost Sveltea znači i ograničeniju dostupnost resursa, tutorijala i podrške zajednice. Za razliku od popularnijih frameworkova, rješavanje problema ili pronalaženje gotovih rješenja može zahtijevati više vremena i samostalnog istraživanja.

Integracija Sveltea u postojeće projekte može biti izazovna. Migracija sa drugog frameworka često zahtijeva značajne izmjene u kodnoj bazi i prilagođavanje arhitekture, što može povećati troškove i vrijeme razvoja.

Konačno, budućnost Sveltea je i dalje manje predvidiva u odnosu na dugoročno etablirane alate. Iako se trenutno koristi u produkciji i ima aktivnu podršku, postoji neizvjesnost oko dugoročnog usvajanja i potencijalnih promjena u njegovom razvoju.

4. Integracija Svelte Framework-a sa Backend-om

Svelte omogućava jednostavnu komunikaciju sa backend serverima koristeći standardne JavaScript metode, kao što su fetch ili biblioteka Axios. Ovi API pozivi služe za dobijanje podataka, slanje informacija na server ili upravljanje stanjem aplikacije.

Jedna od prednosti Sveltea je integracija sa životnim ciklusom komponenti. Na primjer, funkcija onMount omogućava da se API pozivi izvrše odmah nakon što se komponenta prikaže na stranici, što olakšava inicijalno učitavanje podataka.

Primjer dohvaćanja liste stavki sa servera:

```
<script>
  import { onMount } from 'svelte';

  let items = [];

  onMount(async () => {
    const response = await fetch('https://api.example.com/items');
    items = await response.json();
  });
</script>

<ul>
  {#each items as item}
    <li>{item.name}</li>
  {/each}
</ul>
```

U ovom primjeru, kada komponenta bude montirana, izvršava se asinhroni poziv API-ju i podaci se spremaju u varijablu items. Zatim, #each direktiva automatski generiše HTML elemente za svaki dobijeni objekat, što omogućava da se UI ažurira bez dodatnog koda za manipulaciju DOM-om.

Ovaj pristup pojednostavljuje integraciju sa serverima i olakšava razvoj dinamičnih aplikacija koje zavise od vanjskih podataka, dok istovremeno zadržava performanse i čitljivost koda.

4.1. Autentifikacija i autorizacija

U modernim web aplikacijama, upravljanje korisnicima je ključni aspekt sigurnosti i funkcionalnosti. U Sveltu se za ovo često koriste standardizirane metode poput JSON Web Tokens

(JWT) ili OAuth, koje omogućavaju sigurno provjeravanje identiteta korisnika i kontrolu pristupa različitim dijelovima aplikacije.

JWT (JSON Web Token) je popularna metoda za autentifikaciju. Token se generiše na serveru nakon što se korisnik uspješno prijavi i zatim se šalje klijentu. Token sadrži šifrirane podatke o korisniku, uključujući uloge i prava pristupa. Klijent ga može koristiti pri svakom sljedećem zahtjevu prema serveru, što eliminiše potrebu za stalnim provjerama identiteta na serveru.

Na strani Svelte aplikacije, token se obično čuva u localStorage, sessionStorage ili cookies, ovisno o sigurnosnim zahtjevima. Kada korisnik pokuša pristupiti zaštićenoj ruti ili funkciji, aplikacija provjerava validnost tokena i na osnovu toga dopušta ili odbija pristup.

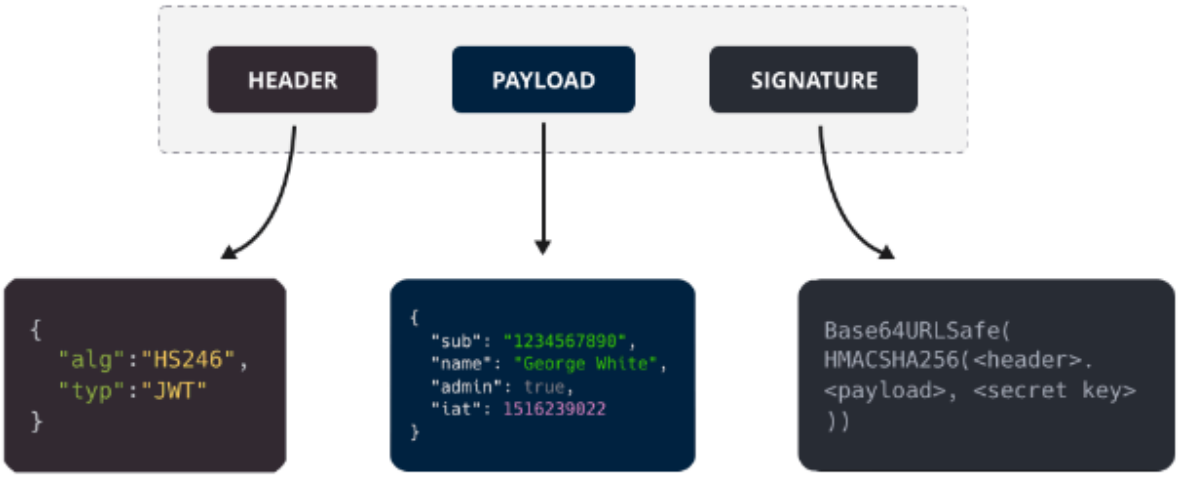
Primjer provjere tokena u Svelte komponenti:

```
<script>
  import { onMount } from 'svelte';

  let user = null;

  onMount(() => {
    const token = localStorage.getItem('jwt');
    if (token) {
      // Obično se ovdje dešifrira token ili šalje API poziv za verifikaciju
      user = JSON.parse(atob(token.split('.')[1]));
    }
  });
</script>

{#if user}
  <p>Welcome, {user.name}!</p>
{:else}
  <p>Please log in to access this content.</p>
{/if}
```

OAuth je alternativa JWT-u koja omogućava korisnicima da se prijave koristeći postojeće naloge, poput Google, Facebook ili GitHub računa. Svelte aplikacija u tom slučaju djeluje kao klijent koji prima autorizacijski token od provajdera i koristi ga za pristup podacima ili resursima na serveru. OAuth je posebno koristan u projektima gdje je potreban brz i siguran način integracije sa eksternim servisima bez potrebe za kreiranjem vlastitog sistema autentifikacije.

Osim samog prijavljivanja, autorizacija je važan dio. To znači da ne samo da provjeravamo ko je korisnik, nego i koje akcije može izvršavati. U Svelte aplikacijama, ovo se može implementirati kombinovanjem JWT podataka ili uloga korisnika sa uvjetnim prikazom elemenata i ruta. Na primjer, administrator može vidjeti posebnu stranicu ili dugmad za upravljanje korisnicima, dok obični korisnik te opcije neće imati.

Svelte također može koristiti dodatne biblioteke za upravljanje sesijama i pristupom, kao što su svelte-kit-auth ili svelte-simple-auth, koje pojednostavljuju implementaciju prijave, odjave i zaštite ruta. Ove biblioteke obično dolaze sa ugrađenim metodama za rad sa tokenima, zaštitu ruta i automatizovano preusmjeravanje korisnika na login stranicu kada nisu autentifikovani.



Slika 8 - Struktura JSON Web Token-a

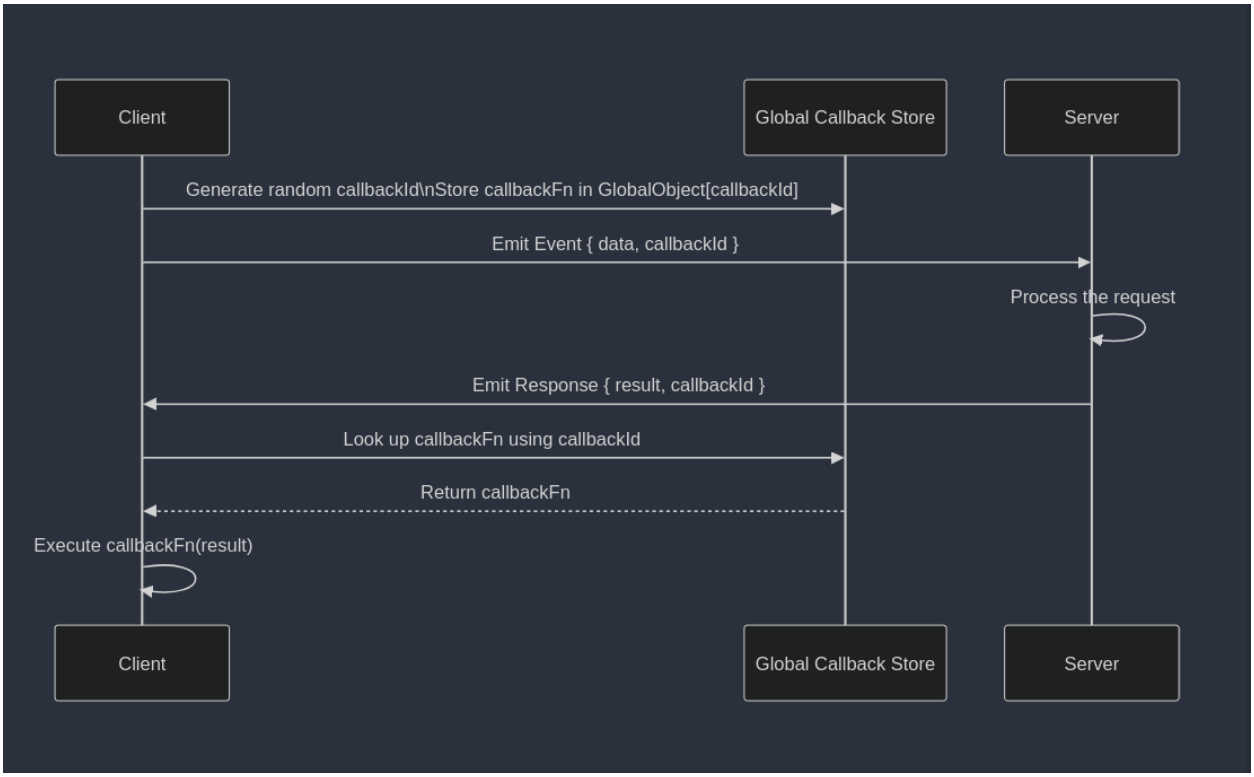
Sigurnosne preporuke:

- Tokeni se trebaju čuvati na siguran način (preporučuju se HttpOnly cookies za smanjenje rizika od XSS napada).
- Provjera validnosti tokena se mora vršiti na serveru, ne samo na klijentu.
- Koristiti HTTPS protokol kako bi se spriječilo presretanje podataka.
- Ograničiti vrijeme trajanja tokena i omogućiti automatsko osvježavanje (refresh) kako bi se povećala sigurnost.

Kombinacija Svelte-a sa ovim tehnikama omogućava razvoj modernih, sigurnih i skalabilnih web aplikacija, gdje korisnici imaju kontrolisan pristup, a backend ostaje zaštićen od neautorizovanih zahtjeva.

4.2. WebSocket komunikacija u Svelte-u

Za aplikacije koje zahtijevaju real-time razmjenu podataka, poput chat aplikacija, online igara, live feedova ili finansijskih aplikacija, WebSocket predstavlja idealno rješenje. WebSocket protokol omogućava stalnu, dvosmjernu TCP vezu između klijenta i servera, što omogućava serveru da šalje podatke klijentu u stvarnom vremenu, bez potrebe da klijent stalno šalje zahtjeve (polling). Svelte, kao moderni framework koji kompajlira HTML, CSS i JavaScript direktno u optimizovan kod, odlično se uklapa u real-time scenarije. Njegov reaktivni sistem automatski ažurira korisnički interfejs kada se promijene podaci, što olakšava prikaz dolaznih poruka ili novih informacija preko WebSocket-a.



Slika 9 - Integracija WebSocket-a u Svelte

```
<script>
  import { onDestroy } from 'svelte';

  let messages = [];
  let newMessage = '';
```

```
  // Uspostavljanje WebSocket veze
  const socket = new WebSocket('ws://example.com/socket');

  // Primanje poruka od servera
  socket.onmessage = (event) => {
    messages = [...messages, event.data];
  };

  // Slanje poruka serveru
  function sendMessage(message) {
    if(message.trim()) {
      socket.send(message);
      newMessage = '';
    }
  }

  // Čišćenje i zatvaranje veze prilikom uništavanja komponente
  onDestroy(() => {
    socket.close();
  });
</script>

<input bind:value={newMessage} placeholder="Type a message">
<button on:click={() => sendMessage(newMessage)}>Send</button>

<ul>
  {#each messages as message}
    <li>{message}</li>
  {/each}
</ul>
```

U ovom primjeru, bind:value omogućava dvosmjerno bindovanje između input polja i varijable newMessage. Svaka dolazna poruka automatski se dodaje u listu messages, a Svelte-ova reaktivnost osigurava da se UI odmah ažurira bez dodatnog koda za manipulaciju DOM-om.

Dodatne napomene i najbolje prakse:

- Upravljanje vezom – Važno je pratiti životni ciklus komponente i pravilno zatvarati WebSocket vezu koristeći onDestroy, kako bi se izbjegli memory leak-ovi i nepotrebna opterećenja servera.

- Error handling – Preporučuje se dodati event listener-e za onerror i onclose, kako bi aplikacija mogla reagovati na prekide veze ili greške u komunikaciji:

```
socket.onerror = (error) => {
```

```
  console.error('WebSocket error:', error);
};

socket.onclose = () => {
  console.log('WebSocket connection closed');
};
```

- Ponovno povezivanje – Za pouzdane aplikacije, može se implementirati logika automatskog ponovnog povezivanja ukoliko veza bude prekinuta.

- Sigurnost – Kada se koristi u produkciji, preporučuje se koristiti wss:// protokol (WebSocket preko TLS-a) kako bi se zaštitili podaci i spriječio presretanje komunikacije.

- Skalabilnost – Za veće aplikacije sa velikim brojem korisnika, server mora podržavati skalabilno rukovanje WebSocket konekcijama, često uz pomoć tehnologija poput Redis-a ili specijalizovanih WebSocket servera.

Kombinovanjem WebSocket-a sa Svelte-ovom reaktivnošću, programeri mogu brzo kreirati dinamične, interaktivne aplikacije koje u stvarnom vremenu prikazuju nove podatke, dok se kod održava jednostavnim i preglednim.

ZAKLJUČAK

Svelte predstavlja značajan korak naprijed u razvoju frontend tehnologija, nudeći jedinstven pristup koji se razlikuje od tradicionalnih frameworkova kao što su React, Vue i Angular. Sa svojom sposobnošću da generiše visokokvalitetan, optimizovan JavaScript kod, Svelte omogućava brže učitavanje stranica i bolje performanse, dok njegova jednostavna sintaksa i manjak boilerplate koda olakšavaju razvoj i održavanje aplikacija.

Jedna od ključnih prednosti Sveltea je njegova reaktivnost i kompajlerski pristup, koji smanjuje opterećenje na runtime-u i omogućava efikasno upravljanje podacima i događajima. Ova karakteristika je posebno korisna u razvoju interaktivnih aplikacija, progresivnih web aplikacija (PWA) i kontrolnih ploča za IoT uređaje, gdje su brzina i odziv kritični faktori.

Upotreba Svelte frameworka može donijeti značajne prednosti, posebno u projektima gde su performanse, veličina fajlova i brzina razvoja ključni faktori. Njegova intuitivna struktura komponenti i lokalizovani CSS olakšavaju timovima brzi razvoj i smanjuju vrijeme potrebno za onboarding novih programera.

Ipak, postoje i izazovi. Svelte ima manji ekosistem biblioteka i alata u poređenju sa popularnijim frameworkovima, što može otežati pronalaženje gotovih rješenja za specifične funkcionalnosti. Integracija sa postojećim projektima ili prelazak sa drugih frameworkova može zahtijevati dodatne prilagodbe. Također, iako zajednica raste, podrška i resursi još uvijek nisu tako rasprostranjeni kao kod Reacta ili Vuea.

U poređenju sa Reactom, Vueom i Angularom, Svelte nudi konkurentske prednosti u pogledu performansi, čitljivosti koda i jednostavnosti razvoja, ali sa sobom nosi i određene ograničene aspekte koji se moraju uzeti u obzir pri odabiru tehnologije. Za timove i projekte koji cijene brzinu, efikasnost i minimalizam u kodu, Svelte predstavlja modernu i perspektivnu opciju.

U budućnosti, kako Svelte i SvelteKit nastavljaju da evoluiraju, očekuje se rast ekosistema, dodatak novih alata i poboljšana integracija sa drugim tehnologijama, što će dodatno učvrstiti njegovu poziciju među vodećim front-end frameworkovima. Za programere koji žele inovativan pristup razvoju web aplikacija, Svelte pruža balans između performansi, fleksibilnosti i jednostavnosti korišćenja.

POPIS SLIKA

CITATNI IZVORI

[1] »WikiPedia,« [Mrežno]. Available: https://en.wikipedia.org/wiki/Svelte.

[2] »MDN web docs,« [Mrežno]. Available: https://developer.mozilla.org/en-US/docs/Learn/Tools_and_testing/Client-side_JavaScript_frameworks/Svelte_getting_started.

[3] »Svelte.dev,« [Mrežno]. Available: https://svelte.dev/docs/basic-markup.

[4] »DEV,« [Mrežno]. Available: https://dev.to/ideoagency/svelte-the-new-frontend-framework-4ghf.