**VIŠJA STROKOVNA ŠOLA ACADEMIA**

**MARIBOR**

# KOMPARATIVNA ŠTUDIJA SODOBNIH SPLETNIH OGRODIJ: OGRODJE NEXT.JS IN OGRODJE SVELTEKIT

Kandidatka: Sintija Matjašič

Vrsta študija: študentka izrednega študija

Študijski program: programsko inženirstvo

Mentor predavatelj: mag. Ervin Schaff

Mentor v podjetju: Sergey Lobanov, dip. inž. inf. in teh. kom.

Lektorica: Nina Balažek, mag. angl. in mag. prof. plov.

Maribor, 2026

# IZJAVA O AVTORSTVU DIPLOMSKEGA DELA

Podpisana Sintija Matjašič, sem avtorica diplomskega dela z naslovom Komparativna študija sodobnih spletnih ogrodij: ogrodje Next.js in ogrodje SvelteKit, ki sem ga napisala pod mentorstvom mag. Ervina Schaffa.

S svojim podpisom zagotavljam, da:

- je predloženo delo izključno rezultat mojega dela,
- sem poskrbela, da so dela in mnenja drugih avtorjev, ki jih uporabljam v predloženi nalogi, navedena oz. citirana skladno s pravili Višje strokovne šole Academia Maribor,
- se zavedam, da je plagiatorstvo – predstavljanje tujih del oz. misli, kot moje lastne kaznivo po Zakonu o avtorski in sorodnih pravicah (Uradni list RS, št. 16/07 – uradno prečiščeno besedilo, 68/08, 110/13, 56/15 in 63/16 – ZKUASP); prekršek pa podleže tudi ukrepom Višje strokovne šole Academia Maribor skladno z njenimi pravili,
- skladno z 32.a členom ZASP dovoljujem Višji strokovni šoli Academia Maribor objavo diplomskega dela na spletnem portalu šole.

Maribor, julij 2026 Podpis študenta:

**ZAHVALA**

Iskreno se zahvaljujem svojemu mentorju mag. Schaffu za strokovno pomoč, usmerjanje in koristne nasvete pri izdelavi diplomskega dela.

Zahvaljujem se tudi svoji družini, fantu in bližnjim za podporo, spodbudo in razumevanje v času študija ter nastajanja diplomskega dela.

**POVZETEK**

Diplomsko delo obravnava primerjalno analizo dveh sodobnih spletnih ogrodij, Next.js in SvelteKit. Osredotoča se na arhitekturo, razvijalsko izkušnjo, zmogljivost, varnost in praktično uporabnost. Namen diplomskega dela je ugotoviti ključne razlike med obema ogrodjema in na podlagi izvedene primerjave oblikovati priporočila za njuno uporabo pri različnih vrstah spletnih projektov. Ugotovitve temeljijo na sekundarnih virih, kot so uradna dokumentacija, strokovni članki in obstoječe študije.

Raziskava temelji na deskriptivni, primerjalni in analitični metodi. Z analizo strokovne literature, uradne dokumentacije in rezultatov obstoječih raziskav so primerjani posamezni vidiki obeh ogrodij, pri čemer je poudarek na njihovih arhitekturnih značilnostih, zmogljivosti, varnosti, razvijalski izkušnji in primernosti za različne vrste projektov.

Teoretični del opisuje razvoj spletnih aplikacij, značilnosti obeh ogrodij in sodobne pristope k upodabljanju. Sledi primerjava arhitekture, zmogljivosti, varnosti, razvijalske izkušnje in praktične uporabnosti.

Ugotovitve kažejo, da ima Next.js zaradi povezanosti z Reactom širši ekosistem in večjo podporo skupnosti. SvelteKit pa je preprostejši, bolj dostopen začetnikom in manj obremenjuje sistem. Pri zmogljivosti SvelteKit izstopa pri manj zahtevnih aplikacijah, Next.js pa je bolj stabilen pri večjih obremenitvah. Obe ogrodji imata primerljive varnostne mehanizme, vendar se razlikujeta v tem, koliko odgovornosti prepuščata razvijalcu.

Na podlagi ugotovitev so oblikovana priporočila za izbiro ogrodja. Next.js je bolj primeren za projekte, kjer so pomembne optimizacije za iskalnike, delo v večjih ekipah in dolgoročna vzdržljivost. SvelteKit pa je odlična izbira za manjše projekte, prototipe, MVP-je in ekipe, kjer sta prioriteti hitrost razvoja in preprostost. Rezultati primerjalne analize predstavljajo strnjen pregled ključnih značilnosti obeh ogrodij in služijo kot pomoč razvijalcem, študentom in razvojnim ekipam pri izbiri ustreznega ogrodja glede na zahteve projekta.

**Ključne besede**: spletna ogrodja, Next.js, SvelteKit, primerjalna analiza, spletni razvoj.

**ABSTRACT**

**Comparative Study of Modern Web Frameworks: Next.js and SvelteKit**

This thesis presents a comparative analysis of two modern web frameworks, Next.js and SvelteKit. The study focuses on their architecture, developer experience, performance, security, and practical applicability. The aim of the thesis is to identify the key differences between the two frameworks and, based on the findings, provide recommendations for their use in different types of web development projects. The findings are based on secondary sources, including official documentation, academic publications, and existing research studies.

The research applies descriptive, comparative, and analytical methods. Individual aspects of both frameworks are examined through the analysis of academic literature, official documentation, and the results of existing studies, with particular emphasis on their architectural characteristics, performance, security, developer experience, and suitability for different types of projects.

The theoretical part of the thesis presents the evolution of web application development, the characteristics of both frameworks, and modern rendering approaches. This is followed by a comparison of their architecture, performance, security, developer exeperience, and practical applicability.

The findings indicate that, due to its integration with React, Next.js offers a broader ecosystem and stronger community support. SvelteKit, on the other hand, is simpler, more beginner-friendly, and places less load on system resources. In terms of performance, SvelteKit performs better in less demanding applications, while Next.js demonstrates greater stability under heavier workloads. Both frameworks provide comparable built-in security mechanisms, although they differ in the degree of responsibility assigned to the developer.

Based on the findings, recommendations for selecting an appropriate framework are provided. Next.js is better suited for projects where search engine optimization (SEO), collaboration with larger development teams, and long-term maintainability are of primary importance. SvelteKit is an excellent choice for smaller projects, prototypes, minimum viable products (MVPs), and teams that prioritize rapid development and simplicity. The results of the comparative analysis provide a concise overview of the key characteristics of both frameworks and may serve as a

useful reference for developers, students, and development teams when selecting an appropriate framework according to project requirements.

**Keywords:** web frameworks, Next.js, SvelteKit, comparative analysis, web development.

# KAZALO VSEBINE


**1 UVOD ..... 12**

1.1 Opis področja in opredelitev problema ..... 12

1.2 Namen, cilji in osnovne trditve ..... 12

1.3 Predpostavke in omejitve ..... 13

1.4 Uporabljene raziskovalne metode ..... 14

**2 TEORETIČNA IZHODIŠČA ..... 15**

2.1 Sodobni razvoj spletnih aplikacij ..... 15

2.2 Predstavitev ogrodja Next.js ..... 19

2.3 Predstavitev ogrodja SvelteKit ..... 21

**3 PRIMERJALNA ANALIZA ARHITEKTURE ..... 23**

3.1 Arhitekturne značilnosti Next.js ..... 23

3.2 Arhitekturne značilnosti SvelteKit ..... 28

3.3 Ključne razlike in podobnosti ..... 32

**4 RAZVIJALSKA IZKUŠNJA IN DOSTOPNOST OGRODIJ ..... 34**

4.1 Učenje in dokumentacija ..... 34

4.2 Kompleksnost razvoja ..... 34

4.3 Podpora skupnosti ..... 35

**5 ZMOGLJIVOST IN VARNOST ..... 37**

5.1 Primerjava hitrosti delovanja ..... 37

5.2 Optimizacija in obremenitev sistema ..... 41

5.3 Vgrajeni varnostni mehanizmi ..... 43

**6 PRAKTIČNA UPORABNOST IN PRIMERNOST ZA PROJEKTE ..... 46**

6.1 Značilni primeri uporabe Next.js ..... 46

6.2 Značilni primeri uporabe SvelteKit ..... 47

6.3 Priporočila glede izbire ogrodja ..... 47

**7 DISKUSIJA ..... 49**

7.1 Povzetek ugotovitev ..... 49

7.2 Preverjanje hipotez ..... 50

7.3 Omejitve raziskave ..... 51

7.4 Možnosti nadaljnjih raziskav ..... 52

**8 SKLEP ..... 53**

**9 VIRI, LITERATURA ..... 55**

# KAZALO SLIK

Slika 1 - Časovni pregled razvoja ključnih spletnih tehnologij Vir: Povzeto po virih, navedenih v poglavju 2.1 ..... 15
Slika 2 - Priljubljenost frontend ogrodij Vir: (https://assets.devographics.com/captures/js2024/en-US/front_end_frameworks_ratios.png) ..... 17
Slika 3 - Primer Client Component v Next.js Vir: (https://nextjs.org/docs/app/getting-started/server-and-client-components) 24
Slika 4 - Primer Universal Load funkcije v SvelteKit Vir: (https://svelte.dev/docs/kit/load) ..... 30
Slika 5 - Primer Default Form Actions funkcije v SvelteKit Vir: (https://svelte.dev/docs/kit/form-actions) ..... 31
Slika 6 - Primerjava prisotnosti skupnosti na GitHubu in Stack Overflow Vir: povzeto po (Vercel, brez datuma), (Svelte, brez datuma) in (Stack Overflow, 2026) ..... 36
Slika 7: Priporočila za izbiro med ogrodjema Next.js in SvelteKit glede na značilnosti projekta Vir: lasten prikaz na podlagi rezultatov raziskave ..... 48

# KAZALO TABEL

TABELA 1: PRIMERJAVA USMERJANJA V NEXT.JS IN SVELTEKIT ..... 32
TABELA 2 - PRIMERJAVA OSNOVNIH ZNAČILNOSTI OGRODIJ NEXT.JS IN SVELTEKIT ..... 33
TABELA 3: PRIMERJAVA IZBRANIH KAZALNIKOV ZMOGLJIVOSTI ..... 40

# SEZNAM KRATIC

| | |
|---|---|
| AJAX | Asynchronous JavaScript and XML |
| CDN | Content Delivery Network |
| CLS | Cumulative Layout Shift |
| CSR | Client-Side Rendering |
| CSP | Content Security Policy |
| CSS | Cascading Style Sheets |
| CSRF | Cross-Site Request Forgery |
| DAL | Data Access Layer |
| DOM | Document Object Model |
| FCP | First Contentful Paint |
| HTML | HyperText Markup Language |
| HTTP | Hypertext Transfer Protocol |
| HTTPS | Hypertext Transfer Protocol Secure |
| ISR | Incremental Static Regeneration |
| JAMstack | JavaScript, API, Markup |
| LCP | Largest Contentful Paint |
| MITM | Man-in-the-Middle |
| MV* | Model-View-* |
| PPR | Partial Prerendering |
| SEO | Search Engine Optimization |
| SI | Speed Index |
| SPA | Single Page Application |
| SSR | Server-Side Rendering |
| SSG | Static Site Generation |

| | |
|---|---|
| SWR | Stale-While-Revalidate |
| TBT | Total Blocking Time |
| TTI | Time to Interactive |
| URL | Uniform Resource Locator |
| VPS | Virtual Private Server |
| XSS | Cross-Site Scripting |

# 1 UVOD

Sodobni razvoj spletnih aplikacij se sooča s številnimi izzivi, med katerimi so zmogljivost aplikacij, optimizacija za iskalnike in prijaznost do razvijalcev. Izbira ustreznega ogrodja pomembno vpliva na hitrost razvoja, vzdrževanje aplikacije in končno uporabniško izkušnjo. Med številnimi sodobnimi ogrodji sta Next.js in SvelteKit, ki predstavljata dva nekoliko različna pristopa k razvoju spletnih aplikacij. Razumevanje njunih razlik je zato pomembno za razvijalce, ki se odločajo o tem, katero tehnologijo uporabiti pri svojih projektih.

## *1.1 Opis področja in opredelitev problema*

Področje raziskovanja je sodobni razvoj spletnih aplikacij, pri čemer se osredotočam na dve priljubljeni ogrodji, in sicer – Next.js in SvelteKit. Next.js, ki temelji na knjižnici React, se je uveljavil kot eno vodilnih ogrodij za razvoj spletnih aplikacij s podporo za strežniško upodabljanje in statično generiranje. Po drugi strani je SvelteKit mlajše ogrodje, ki temelji na kompilacijskem pristopu ogrodja Svelte in postaja vedno bolj priljubljeno zaradi svoje preprostosti in učinkovitosti.

Problem, ki je pritegnil mojo pozornost, je pomanjkanje celovitih primerjav med tema dvema ogrodjema, ki bi razvijalcem pomagale pri odločitvi o izbiri tehnologije. Čeprav obstaja veliko člankov in različnih objav, ki obravnavajo posamezne vidike teh ogrodij, je sistematičnih analiz precej manj. Redke raziskave na enem mestu primerjajo arhitekturo, razvijalsko izkušnjo, zmogljivost in varnost. Pogosto se tudi pojavlja vprašanje, katero ogrodje je primernejše za določene vrste projektov, – na primer za manjše prototipe ali za večje poslovne aplikacije. Za obravnavo te teme sem se odločila predvsem, ker me področje spletnega programiranja zanima in ker menim, da bo raziskava koristila študentom in razvijalcem začetnikom, ki se srečujejo z dilemo izbire ogrodja.

## *1.2 Namen, cilji in osnovne trditve*

Namen diplomskega dela je izvesti komparativno analizo dveh sodobnih spletnih ogrodij, Next.js in SvelteKit, in na podlagi raziskave ugotoviti njune glavne prednosti, slabosti in možnosti uporabe. Pri analizi je poseben poudarek namenjen arhitekturi ogrodij, razvijalski izkušnji, zmogljivosti in varnosti.

Cilji diplomskega dela so:

- analizirati arhitekturne značilnosti obeh ogrodij in ugotoviti ključne razlike v njunem pristopu;
- ovrednotiti dostopnost obeh ogrodij za razvijalce začetnike z vidika dokumentacije, učenja in kompleksnosti razvoja;
- primerjati zmogljivost aplikacij, razvitih v Next.js in SvelteKit, na podlagi obstoječih raziskav;
- analizirati vgrajene varnostne mehanizme in njihov vpliv na zaščito aplikacij;
- na podlagi ugotovitev oblikovati priporočila za izbiro ogrodja glede na vrsto projekta in zahteve razvojne ekipe.

Na podlagi pregleda literature in obstoječih raziskav so v delu zastavljene naslednje hipoteze:

- H1: Next.js ponuja širši ekosistem in bolj razvito podporo skupnosti v primerjavi s SvelteKit.
- H2: SvelteKit je bolj dostopen razvijalcem z manj izkušnjami.
- H3: Spletne aplikacije, razvite v SvelteKit, pogosto dosegajo boljše rezultate pri hitrosti delovanja in manjši obremenitvi sistema kot aplikacije v Next.js.
- H4: Ogrodje Next.js zagotavlja več vgrajenih varnostnih mehanizmov, ki pripomorejo k večji zaščiti spletnih aplikacij kot ogrodje SvelteKit.

## *1.3 Predpostavke in omejitve*

Pri izdelavi diplomskega dela so upoštevane naslednje predpostavke in omejitve:

- Analiza temelji na sekundarnih virih, kot so uradna dokumentacija, strokovni članki in že obstoječe raziskave. Ker se področje spletnih tehnologij zelo hitro razvija, lahko novejše različice ogrodij prinesejo spremembe, ki bi lahko vplivale na ugotovitve v delu.
- Pri analizi zmogljivosti so uporabljeni rezultati več različnih virov (Zaidan & Suyatno, 2025) (DevMorph Team, 2026) (Sanchez, 2024), ki uporabljajo različne metodologije, zato rezultato ni mogoče neposredno primerjati in ne predstavljajo popolnoma splošne slike vseh možnih scenarijev.

- Analiza varnosti se osredotoča na vgrajene varnostne mehanizme, ne pa tudi na širše varnostne vidike, kot so varnost odvisnosti ali rednost posodabljanja.
- Delo ne vključuje lastne implementacije testnih aplikacij, temveč temelji na že obstoječih študijah, analizah in dokumentaciji.
- Zaradi širine in kompleksnosti področja so v analizo vključeni predvsem ključni vidiki, ki so pomembni pri odločanju o izbiri ogrodja.

## *1.4 Uporabljene raziskovalne metode*

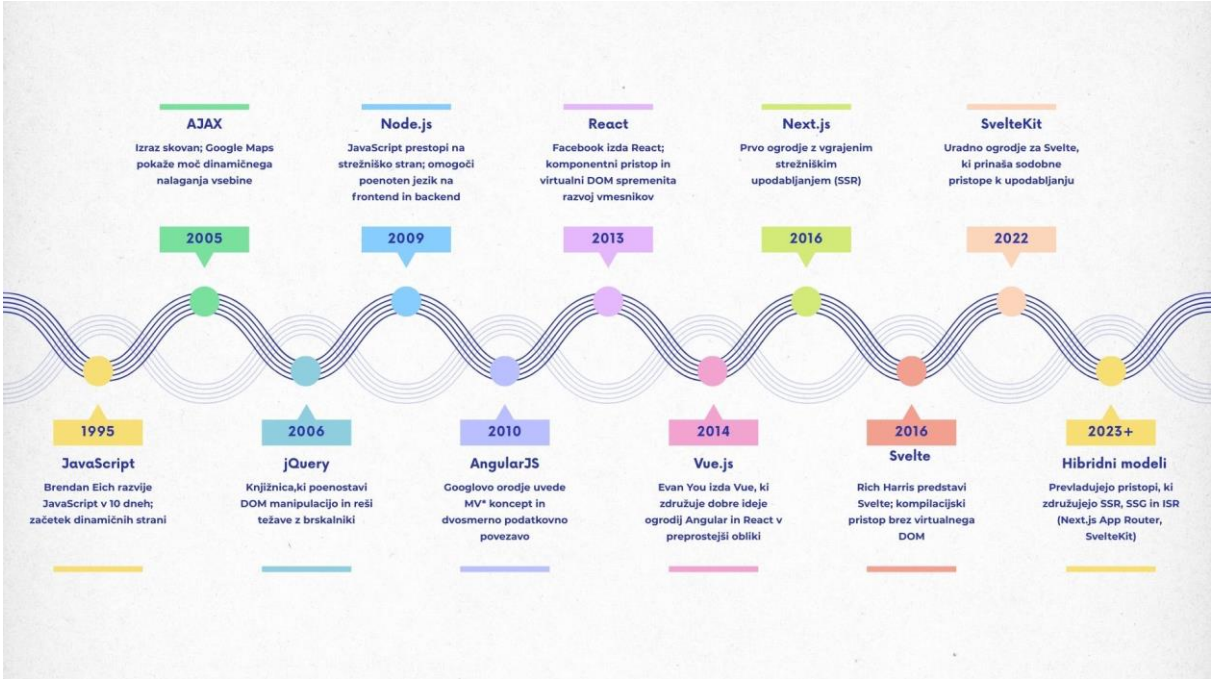
Pri izdelavi diplomskega dela so uporabljene naslednje raziskovalne metode:

- Desk research: – Sistematično zbiranje in analiza sekundarnih virov, vključno z uradno dokumentacijo ogrodij, strokovnimi članki, knjigami in obstoječimi raziskavami.
- Komparativna metoda: – Primerjava obeh ogrodij po vnaprej določenih kriterijih (arhitektura, razvijalska izkušnja, zmogljivost, varnost, praktična uporabnost).
- Metoda analize in sinteze: – Analiza posameznih vidikov ogrodij in sinteza ugotovitev v celovito primerjavo.
- Metoda klasifikacije: – Razvrščanje ugotovitev v smiselne kategorije za lažjo primerjavo in razumevanje.

# 2 TEORETIČNA IZHODIŠČA

## *2.1 Sodobni razvoj spletnih aplikacij*

Slika 1 prikazuje ključne mejnike v razvoju spletnih tehnologij, in sicer od začetkov JavaScripta do sodobnih hibridnih pristopov.




Slika 1 – Časovni pregled razvoja ključnih spletnih tehnologij
Vir: Povzeto po virih, navedenih v poglavju 2.1

### 2.1.1 Zgodnji splet: od statičnih strani do dinamičnih izkušenj

Začetki razvoja spleta so bili zaznamovani s statičnimi spletnimi stranmi. Te so bile sestavljene iz HTML datotek, ki so bile občasno dopolnjene z osnovnimi oblikovnimi predlogami CSS. Vsaka stran je predstavljala svoj dokument, shranjen na strežniku, uporabniki pa so prejeli enako, nespremenjeno vsebino. Ker na strani ni bilo prave interaktivnosti ali obdelave podatkov v brskalniku, je bilo treba vsako posodobitev vsebine ročno urejati v posameznih datotekah.

Pomembna prelomnica je bila uvedba programskega jezika JavaScript leta 1995, saj je omogočila izvajanje skript na strani odjemalca in s tem bolj dinamično delovanje spletnih strani (Wirfs-Brock & Eich, 2020). Spletne strani so se tako začele spreminjati iz preprostih prikazovalnikov informacij v bolj interaktivne sisteme, ki so se odzivali na uporabnikova

dejanja. Zgodnja uporaba JavaScripta je vključevala predvsem preverjanje obrazcev, enostavne animacije in spreminjanje modela dokumentnega objekta (ang. *Document Object Model*, kratica DOM) brez ponovnega nalaganja strani. Čeprav so bila ta orodja na začetku precej omejena in so se med različnimi brskalniki razlikovala, so postavila pomembne temelje za razvoj sodobnega spleta.

### 2.1.2 JavaScript ogrodja in vzpon enostranskih aplikacij (SPA)

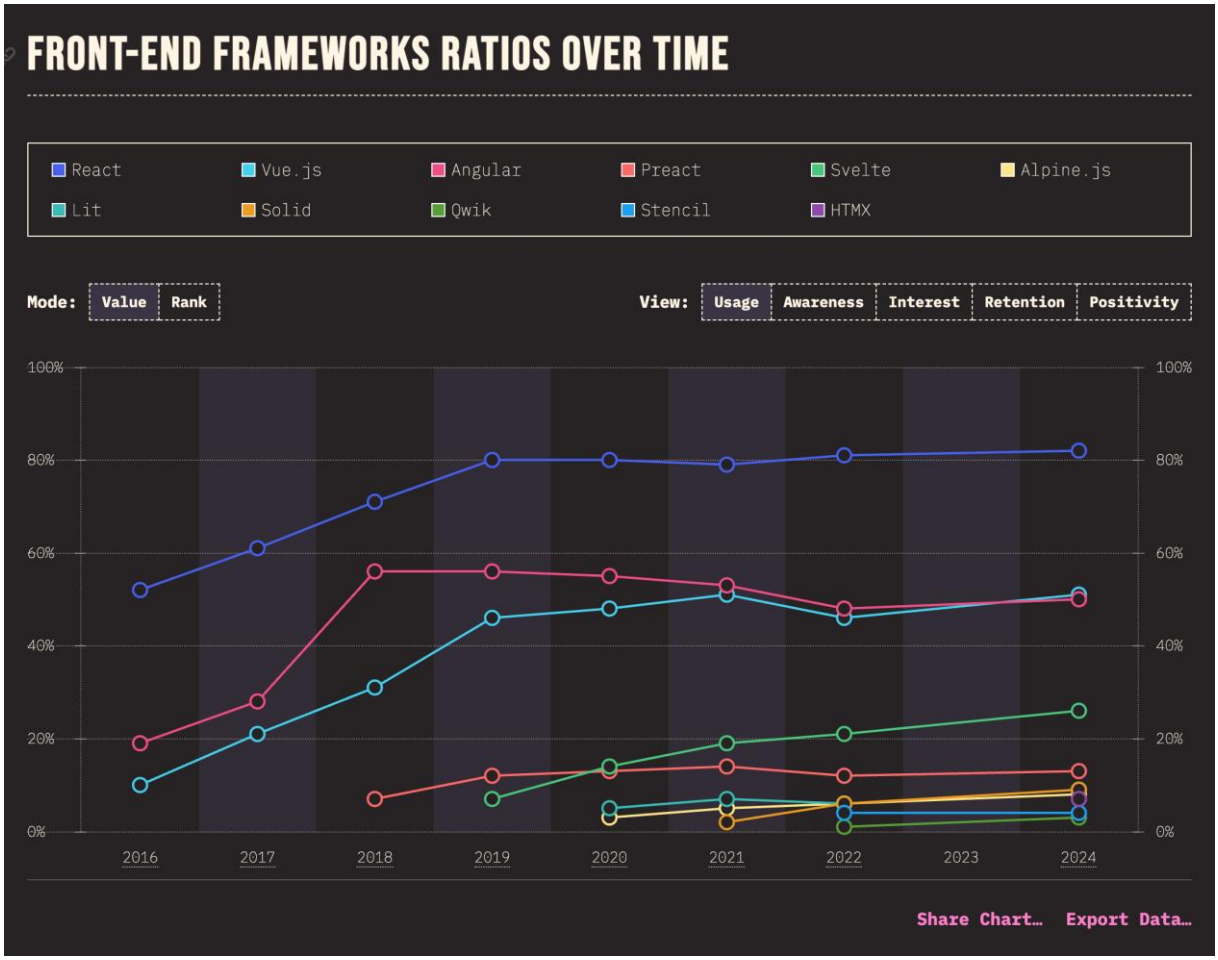
Naraščajoča kompleksnost spletnih aplikacij je spodbudila razvoj učinkovitejših pristopov za upravljanje kode in manipulacijo z objektnim modelom dokumenta (DOM). Leta 2006 je knjižnica jQuery precej poenostavila vsakodnevne naloge v JavaScriptu (Bernstein, 2010) in hkrati rešila veliko težav, povezanih z razlikami med brskalniki. Kljub temu so ostale težave pri razširljivosti in strukturi večjih aplikacij. Že leto prej, leta 2005, je Jesse James Garrett uvedel izraz AJAX (Garrett, 2005), s katerim je opisal pristop k dinamičnemu nalaganju vsebine brez ponovnega nalaganja celotne strani, ki so ga popularizirali Google Maps in druge inovacije v tistem času.

Pomemben mejnik se je zgodil leta 2009, ko je bil predstavljen Node.js. Ta je omogočil izvajanje JavaScripta na strežniški strani, kar je poenotilo uporabo istega jezika na odjemalskem in strežniškem delu aplikacije ter odprlo pot za celovite JavaScript aplikacije (Ornbo, 2012).

V začetku leta 2010 je Google predstavil AngularJS, ki je uvedel MV* koncept in dvosmerno podatkovno vezavo (freeCodeCamp, 2017). Temu je leta 2013 sledil React podjetja Facebook, ki je prinesel komponentni pristop in uporabo virtualnega DOM-a (Maurya, Keshari, Kumar, & Katiyar, 2023). Leta 2014 pa se je pojavil še Vue.js (Vue.js, 2014), ki je v poenostavljeni obliki združil nekatere prednosti ogrodij Angular in React. Ta ogrodja so omogočila razvoj enostranskih aplikacij (ang. *Single Page Applications*, kratica SPA), kjer se večina vsebine izriše v uporabnikovem brskalniku namesto na strežniku. Takšen pristop je imenovan odjemalsko upodabljanje (ang. *Client-Side Rendering*, kratica CSR) in je izboljšal odzivnost ter omogočil bolj tekočo uporabniško izkušnjo, saj se ob posodobitvah osveži le del uporabniškega vmesnika.

Kljub tem prednostim pa so se pojavile tudi pomanjkljivosti. Med najpogosteje omenjenimi sta počasno začetno nalaganje zaradi velikih JavaScript paketov in slabša optimizacija za spletne iskalnike, saj se vsebina naloži šele po prejemu strani v brskalniku (Pati & Zaki, 2025). Po

podatkih raziskave *State of JS Survey 2024* sta React in Vue med najbolj razširjenimi frontend tehnologijami (tehnologijami za razvoj uporabniškega vmesnika) v profesionalni rabi (State of JS Survey, 2024). Hkrati pa razvijalci kot pogoste slabosti frontend ogrodij navajajo pretirano kompleksnost in težave z zmogljivostjo (State of JS Survey, 2024).




Slika 2 – Priljubljenost frontend ogrodij

Vir: (https://assets.devographics.com/captures/js2024/en-US/front_end_frameworks_ratios.png)

Z naraščajočo kompleksnostjo odjemalskih pristopov se zato vse bolj pojavlja zanimanje za pristope, ki združujejo strežniško (ang. *Server-Side Rendering*, kratica SSR) in odjemalsko upodabljanje. Pri tem ima pomembno vlogo hidracija (ang. *hydration*), to je proces, pri katerem se vnaprej izrisana HTML struktura v brskalniku poveže z JavaScript logiko in tako interaktivna. Takšna kombinacija pristopov danes predstavlja osnovo hibridnega modela razvoja spletnih aplikacij (Osmani & Miller, 2026).

### 2.1.3 Sodobni trendi: SSR, SSG, CSR, hidracija in arhitektura JAMstack

Zgodovina spletnega razvoja je ponazorila ciklično vračanje k določenim konceptom. V zgodnjih fazah so vse spletne strani temeljile na strežniškem upodabljanju (SSR), kjer je strežnik za vsak zahtevek ustavil celoten HTML in ga poslal brskalniku. Ta pristop je bil ustrezen za takrat pretežno statične strani, vendar je vsaka interakcija zahtevala ponovno nalaganje celotne strani, kar je slabšalo uporabniško izkušnjo (UIC, brez datuma).

Z razvojem bolj zmogljivih brskalnikov in JavaScripta je v ospredje stopilo odjemalsko upodabljanje (CSR). Omogočilo je izdelavo hitrih in dinamičnih enostranskih aplikacij (SPA). Pri CSR brskalnik prejme le HTML, nato pa s pomočjo prenesene JavaScript kode dinamično ustvari vsebino in upravlja z nadaljnjo interakcijo brez novih zahtevkov na strežnik (UIC, brez datuma). Kljub izboljšani uporabniški izkušnji po začetnem nalaganju pa je CSR prinesel tudi določene pomanjkljivosti. Med njimi so počasnejši prvi prikaz vsebine, težave z optimizacijo za iskalnike (ang. *Search Engine Optimization*, kratica SEO) in povečanje obsega JavaScript kode. To je še posebej prizadelo uporabnike s počasnejšimi napravami in povezavami (UIC, brez datuma).

Kot odgovor na te izzive se je spletna skupnost ponovno začela bolj osredotočati na SSR, ampak v prenovljeni obliki. Glavni motivi za vrnitev k SSR so:

- Izboljšan začetni prikaz vsebine: Ker strežnik vrne že pripravljen HTML, lahko brskalnik takoj prikaže vsebino, kar skrajša čas do prvega vsebinskega prikaza.
- Boljša optimizacija za iskalnike (SEO): Iskalniki lahko neposredno indeksirajo vsebino iz prejeta HTML, ne da bi morali izvajati JavaScript.
- Boljše delovanje na šibkejših napravah: Prenos zahtevnejšega procesa upodabljanja na strežnik zmanjša obremenitev uporabnikove naprave.
- Hitrejši čas do interaktivnosti: Kombinacija SSR s poopkom hidracije omogoča, da uporabnik hitro vidi vsebino, nato pa v ozadju statična stran postane polno interaktivna aplikacija

(UIC, brez datuma).

Ta sodobni hibridni model pogosto združuje SSR s CSR in postopkom hidracije. Pri tem pristopu strežnik poleg začetnega HTML pogosto pošlje začetno stanje aplikacije, ki ga nato JavaScript ob zagonu »rehidrira« v polno delujočo aplikacijo (UIC, brez datuma).

Podoben pristop je statično generiranje strani (ang. *Static Site Generation*, kratica SSG), kjer se strani vnaprej ustvarijo že ob gradnji in se nato strežejo kot statične datoteke. Tak način je zelo učinkovit za vsebine, ki se redkeje spreminjajo. Še naprednejši mehanizem je inkrementalna statična generacija (ang. *Incremental Static Regeneration*, kratica ISR). Ta omogoča posodabljanje statičnih strani na zahtevo, brez ponovne gradnje aplikacije (UIC, brez datuma).

Pomemben sodoben pristop je arhitektura JAMstack (JavaScript, API, Markup). Ta ločuje predstavitveno plast od strežniške logike. To lahko izboljša varnost in razširljivost ter poenostavi objavo spletnih aplikacij (Jamstack, brez datuma). Nadaljnji razvoj infrastrukture, kot so omrežja za dostavo vsebine (ang. *Content Delivery Networks*, kratica CDN), dodatno izboljšujejo zmogljivost spletnih aplikacij, saj omogočajo dostavo vsebine s strežnikov, ki so bližje končnemu uporabniku (Mozilla, brez datuma). Po podatkih W3Techs JavaScript ostaja prevladujoč jezik spleta in je prisoten na ~98,9 % spletnih mest na odjemalski strani (W3Techs, 2025). Kombinacija različnih strategij upodabljanja, statične generacije in uporabe API-jev kaže, kako se je splet razvil iz bolj monolitnih rešitev v modularne in visoko zmogljive sisteme, ki so vse bolj osredotočeni na uporabniško izkušnjo.

## *2.2 Predstavitev ogrodja Next.js*

### 2.2.1 Razvoj in nastanek ogrodja Next.js

Ogrodje Next.js je bilo prvič predstavljeno leta 2016 kot odprtokodna rešitev za razvoj spletnih aplikacij, ki temeljijo na knjižnici React (Mika & Vasylenko, 2025). Njegov nastanek je vodilo podjetje Vercel (takrat Zeit), ki je prepoznalo potrebo po bolj standardiziranem in zmogljivejšem razvojnem ogrodju za React (Ekpobimi, 2024). Glavni cilj je bil poenostaviti uporabo naprednih arhitekturnih pristopov, kot je strežniško upodabljanje (SSR) in razvijalcem ponuditi jasno strukturo projektov s privzetimi optimizacijami, kar bi omogočilo boljšo zmogljivost in lažjo vzpostavitev produkcijskega okolja brez zapletene ročne konfiguracije (Ekpobimi, 2024).

Next.js je nastal kot odziv na omejitve takrat prevladujočih enostranskih aplikacij (SPA), kjer je bilo upodabljanje večinoma prepuščeno brskalniku (Mika & Vasylenko, 2025). Ta pristop je pogosto povzročal počasnejše začetno nalaganje strani in težave pri optimizaciji za iskalnike

(Mika & Vasylenko, 2025). Z združitvijo moči Reacta z vgrajeno podporo za strežniško logiko je Next.js razvojnim ekipam omogočil, da so te izzive premagale, kar je hitro pritegnilo pozornost širše razvijalske skupnosti (Kazanivska & Muzyka, 2024).

Zaradi tesne povezanosti z ekosistemom React, ki je že takrat veljal za eno najbolj razširjenih frontend knjižnic, se je Next.js hitro uveljavil v manjših projektih in v večjih produkcijskih aplikacijah (Kazanivska & Muzyka, 2024). Njegov razvoj je sovpadal s širšim industrijskim trendom prehoda od izključno odjemalskega upodabljanja k hibridnim pristopom, ki združujejo prednosti strežniškega in odjemalskega upodabljanja. S svojo osredotočenostjo na reševanje praktičnih izzivov sodobnega spletnega razvoja si je Next.js utrdil položaj enega vodilnih ogrodij v React okolju.

### 2.2.2 Osnovne značilnosti ogrodja Next.js

Ogrodje Next.js je celovita rešitev za razvoj zmogljivih in razširljivih spletnih aplikacij, ki temeljijo na knjižnici React (Next.js, brez datuma). Zasnovan je tako, da razvijalcem omogoča uporabo različnih pristopov k upodabljanju vsebine in ponuja jasno strukturiran razvojni okvir, primeren za vse, od preprostih vsebinskih strani pa do kompleksnih, dinamičnih aplikacij (Kazanivska & Muzyka, 2024).

Ena pomembnih prednosti Next.js je njegova tesna povezanost z ekosistemom React. Aplikacije temeljijo na komponentnem pristopu, kar omogoča ponovno uporabo kode in večjo preglednost uporabniškega vmesnika (Mika & Vasylenko, 2025). Prav tako izkorišča bogat React ekosistem, kar pomeni, da lahko razvijalci brez težav vključijo na tisoče obstoječih knjižnic za najrazličnejše funkcionalnosti (Lucky, 2023).

Next.js podpira več različnih načinov upodabljanja vsebine. Poleg strežniškega (SSR) in odjemalskega upodabljanja (CSR) omogoča statično generiranje strani (SSG) in naprednejši mehanizem inkrementalne statične regeneracije (ISR) (Mika & Vasylenko, 2025). Izbira načina upodabljanja je prilagodljiva in jo lahko določimo na ravni posamezne strani.

Za usmerjanje uporablja datotečno osnovano usmerjanje (ang. *file-based routing*), kjer struktura map neposredno določa poti znotraj aplikacije (Next.js, brez datuma). To poenostavi upravljanje navigacije in strukture strani.

Next.js vključuje tudi številne vgrajene optimizacije za izboljšanje zmogljivosti. Med najpomembnejšimi so samodejna delitev kode (ang. *automatic code splitting*), optimizacija slik preko komponente *next/image* in podpora za TypeScript (Lucky, 2023) (Kazanivska & Muzyka, 2024).

Omogoča tudi implementacijo strežniških funkcionalnosti znotraj istega projekta, na primer z uporabo API poti (Lucky, 2023). Takšna zasnova omogoča razvoj celovitih spletnih aplikacij v enotnem razvojnem okolju, in sicer brez potrebe po ločenem backend (zalednem) projektu.

## *2.3 Predstavitev ogrodja SvelteKit*

### 2.3.1 Razvoj in nastanek ogrodja SvelteKit

Ogrodje SvelteKit izhaja iz razvoja knjižnice Svelte, ki jo je leta 2016 ustvaril Rich Harris (developyn, 2024). Svelte je nastal kot alternativa uveljavljenim frontend knjižnicam, kot sta React in Vue, predvsem z idejo poenostaviti razvoj uporabniških vmesnikov ter zmanjšati količino in kompleksnost kode, ki se izvaja v brskalniku. Namesto da ogrodje deluje med izvajanjem (ang. *runtime*), je Harris zasnoval Svelte kot kompilator, ki kodo pretvori v čisti (ang. *vanilla*) JavaScript že med gradnjo. S tem se izogne potrebi po virtualnem DOM-u in zmanjša obremenitev brskalnika (developyn, 2024).

Z izdajo Svelte 3 leta 2019 je knjižnica doživela pomemben preboj. Nova različica je poenostavila sintakso in uvedla drugačen način pisanja reaktivne kode, kar je razvijalcem omogočilo bolj pregledno in manj obsežno kodo v primerjavi s starejšimi različicami in nekaterimi drugimi ogrodji (developyn, 2024). To je pripomoglo k večji prepoznavnosti in širši uporabi Svelta v razvijalski skupnosti.

Z večjo priljubljenostjo Svelta se je pojavila potreba po celovitem ogrodju, ki bi poleg uporabniškega vmesnika ponujalo tudi standardizirane rešitve za usmerjanje, upravljanje podatkov, strežniško upodabljanje in pripravo na produkcijo, zato je bil razvit SvelteKit, uradno izdan decembra 2022, in sicer kot naslednik projekta Sapper (developyn, 2024) (Svelte, 2022).

Leta 2022 so se Rich Harris in nekateri ključni sodelavci pridružili podjetju Vercel, kar je omogočilo polno podporo in pospešilo razvoj (developyn, 2024).

### 2.3.2 Osnovne značilnosti ogrodja SvelteKit

Kot je bil predstavljeno, SvelteKit temelji na kompilacijskem pristopu, kar pomeni da se med gradnjo aplikacije pretvori v optimiziran, čisti JavaScript, namesto da bi v brskalnik pošiljal celotno ogrodje (Lucky, 2023). Posledica tega so manjši paketi in hitrejše delovanje, kar je še posebej pomembno na mobilnih napravah (Ulili, 2025).

SvelteKit je zasnovan kot celostno grodje, ki pokrije vse vidike sodobnega spletnega razvoja (Lucky, 2023). Tako kot Next.js uporablja datotečno usmerjanje, in sicer – mapa *src/routes* in njene podmape neposredno določajo poti v aplikaciji (SvelteKit, 2026). Uporabne so tudi postavitve (ang. *layouts*), s katerimi lahko skupne dele uporabniškega vmesnika (npr. glavo/nogo) definiramo enkrat in jih uporabimo na več straneh (Bratslavsky, 2025).

SvelteKit podpira več načinov upodabljanja, in sicer SSR, SSG in CSR. Način lahko izberemo na ravni posamezne strani (Ulili, 2025). Ko pride čas za objavo, so na voljo adapterji za različna okolja (Node, Vercel, Netlify ...) (SvelteKit, 2026).

SvelteKit ima vgrajeno podporo za TypeScript. Ko ustvarjamo projekt, lahko izberemo, ali ga želimo uporabljati, SvelteKit pa poskrbi za vso potrebno konfiguracijo (Bratslavsky, 2025).

Podatke pridobivamo preko load funkcij, ki so povezane s posamezno stranjo. Te funkcije tečejo na strežniku ob prvem obisku in na odjemalcu ob navigaciji, koda pa je povsod enaka (Bratslavsky, 2025).

Za upravljanje stanja ima SvelteKit vgrajene t. i. *stores*. Te omogočajo preprosto in reaktivno deljenje podatkov med komponentami, ne da bi morali uporabljati zunanje knjižnice (Ulili, 2025).

# 3 PRIMERJALNA ANALIZA ARHITEKTURE

## *3.1 Arhitekturne značilnosti Next.js*

Arhitektura spletnega ogrodja opredeljuje temeljne mehanizme, ki določajo, kako se aplikacija gradi, kako se koda izvaja in kako podatki potujejo skozi sistem. V različici 13 je Next.js predstavil App Router, ki danes predstavlja priporočen način gradnje aplikacij. App Router uvaja nov konceptualni model, ki temelji na React Server Components, gnezdenih layoutih in hibridnem upodabljanju (Next.js, brez datuma).

### 3.1.1 Usmerjanje

Next.js uporablja datotečno usmerjanje. V App Routerju mapa »*app*« predstavlja korensko mapo. Struktura map neposredno določa URL poti. Na primer mapa *app/about/page.tsx* ustreza URL-ju */about* (Next.js, brez datuma). Vsaka mapa lahko vsebuje posebne datoteke z rezerviranimi imeni:

- *layout.tsx* – za definiranje skupnih postavitev, ki se uporabljajo na več straneh;
- *page.tsx* – za posamezne strani aplikacije;
- *loading.tsx* – za prikaz vmesnega stanja med nalaganjem vsebine, uporablja React Suspense;
- *error.tsx* – za obravnavanje napak znotraj segmenta;
- *route.ts* – definicija API končnih točk;

(Next.js, brez datuma).

Dinamične poti se označujejo z oglatimi oklepaji ([slug]), kar omogoča zajemanje spremenljivih delov URL-ja. Route groups z oklepaji (group) omogočajo logično organizacijo poti brez vpliva na URL (Next.js, brez datuma).

Ta intuitivni sistem usmerjanja je ena ključnih prednosti Next.js, saj odpravlja potrebo po kompleksni konfiguraciji usmerjanja in naredi strukturo aplikacije takoj razumljivo (Mika & Vasylenko, 2025).

### 3.1.2 Komponentni model

V App Routerju so vse layouti in strani privzeto Server Components (Next.js, brez datuma). To pomeni, da se izvajajo na strežniku, ne pošiljajo kode v brskalnik in imajo neposreden dostop do podatkovnih virov. Uporabljajo se za pridobivanje podatkov in zmanjševanje količine JavaScripta, ki je poslan odjemalcu (Next.js, brez datuma).

Client Components se označujejo z direktivo »use client« na začetku datoteke (Next.js, brez datuma). Uporabljajo se, ko komponenta potrebuje stanje (useState), dogodke (onClick), življenjske cikle (useEffect) ali brskalniške API-je (Next.js, brez datuma). Ko je datoteka označena z »use client«, so vsi njeni importi in podkomponente del odjemalskega paketa (Next.js, brez datuma).

```
'use client'

import { useState } from 'react'

export default function Counter() {
  const [count, setCount] = useState(0)

  return (
    <div>
      <p>{count} likes</p>
      <button onClick={() => setCount(count + 1)}>Click me</button>
    </div>
  )
}
```

Slika 3 – Primer Client Component v Next.js

Vir: (https://nextjs.org/docs/app/getting-started/server-and-client-components)

Server Components omogočajo dostop do backend virov (baz, datotečnega sistema) brez vmesnih API plasti, kar poenostavlja arhitekturo (Next.js, brez datuma). Server Components privzeto uporabljajo statično upodabljanja, razen če uporabljajo dinamične API-je (Next.js, brez datuma).

### 3.1.3 Upodabljanje

Next.js podpira več načinov upodabljanja, ki jih razvijalec kombinira znotraj iste aplikacije.

Statično upodabljanje pomeni, da se stran generira ob izgradnji. Rezultat se predpomni in deli med uporabnike. Kot že omenjeno, je statično upodabljanje privzeto za Server Components, razen če komponenta uporablja dinamične API-je, kot so cookies(), headers() ali searchParams. Statično upodabljanje dosega odlične rezultate pri meritvah Core Web Vitals, še posebej pri LCP (Umeh, 2025).

Dinamično upodabljanje pomeni, da se HTML generira ob vsaki zahtevi. Dinamično upodabljanje je primerno za prikaz podatkov v realnem času in za personalizirano vsebino (Umeh, 2025).

Inkrementalno statično generiranje omogoča, da se statične strani posodabljajo tudi po tem, ko je bila spletna stran že zgrajena. Namesto da bi morali ob vsaki spremembi vsebine znova graditi celotno stran, ISR poskrbi da se stran ob naslednjem obisku uporabnika v ozadju tiho osveži. Uporabnik njaprej vidi staro različico, vendar ta takoj ne zastara, saj se v ozadju pripravlja nova. Ko je pripravljena, jo naslednji obiskovalec že dobi svežo. Poleg tega obstaja tudi možnost on-demand ISR, kjer lahko posodobitev sprožimo ročno prek posebnega API-ja. Na primer takoj, ko objavimo nov članek ali spremenimo ceno izdelka (Ayebola, 2025).

Streaming je mehanizem, pri katerem strežnik začne pošiljati HTML odjemalcu že preden so vsi podatki pripravljeni. Stran se tako prikaže postopoma. Najprej se prikažejo hitrejši deli, počasnejši pa se dodajajo sproti, ko so na voljo. To se izvaja z React Suspense (Ayebola, 2025). Poleg tega Next.js eksperimentalno podpira tudi delno predhodno upodabljanje (ang. *Partial Prerendering*). S tem pristopom je lahko na isti strani del vsebine statičen, del pa dinamičen. Statični del se prikaže takoj, dinamični pa se naloži malo kasneje (Ayebola, 2025).

### 3.1.4 Podatkovni tokovi

V spletni aplikaciji podatki potujejo med strežnikom, bazo podatkov in uporabnikovim brskalnikom. Next.js ponuja več načinov, kako te podatke pridobiti in jih prikazati.

V starejšem Pages Routerju so za pridobivanje podatkov obstajale ločene funkcije, in sicer: *getStaticProps* za statično generiranje, *getServerSideProps* za strežniško upodabljanje in

*getStaticPaths* za dinamične poti (GeeksForGeeks, 2025). V App Routerju so ti koncepti še vedno veljavni, vendar se implementirajo na bolj neposreden način znotraj komponent.

V sodobnem App Routerju poteka pridobivanje podatkov neposredno znotraj Server Components z uporabo *fetch* funkcije. Razvijalec se lahko odloči, ali želi podatke predpomniti (cache: 'force-cache') ali pa jih vedno znova sveže pridobiti (cache: 'no-store') (Next.js, brez datuma).

Za pridobivanje podatkov na odjemalcu se pogosto uporablja knjižnica SWR, ki jo razvija Vercel. SWR samodejno predpomni podatke, jih v ozadju osvežuje in omogoča optimistične posodobitve. Uporabnik pa vidi spremembo takoj, čeprav se podatki še pošiljajo na strežnik (Saboori, 2024). V App Routerju SWR deluje znotraj Client Components (označenih z 'use client').

Za spreminjanje podatkov (mutacije) Next.js uporablja Server Functions (označene z 'use server'). Te funkcije se izvajajo na strežniku in jih je mogoče klicati iz odjemalca, na primer ob oddaji obrazca ali kliku na gumb. Po uspešni spremembi lahko razvijalec s funkcijama *revalidatePath* ali *revalidateTag* posodobi predpomnjene podatke, da se na strani prikažejo sveže vsebine (Next.js, brez datuma).

Predpomnjenje (caching) v App Routerju poteka na več ravneh. Z direktivo 'use cache' lahko razvijalec eksplicitno označi, katere funkcije ali komponente naj se predpomnijo. Podatki, ki se ne spreminjajo pogosto, se lahko predpomnijo in vključijo v statično lupino strani, medtem ko se sveži podatki pridobivajo sproti s pomočjo Suspense (Next.js, brez datuma).

### 3.1.5 Vgrajene optimizacije

Next.js vključuje kar nekaj optimizacij, ki jih ni treba posebej nastavljati ali konfigurirati (Mika & Vasylenko, 2025).

Med najbolj uporabnimi je optimizacija slik s komponento *next/image*. Ta samodejno prilagodi velikost slike glede na napravo, jo pretvori v sodobnejše formate (WebP, AVIF) in jo naloži šele takrat, ko postane vidna v oknu brskalnika. S tem se skrajša čas začetnega nalaganja strani, saj se ne nalagajo vse slike hkrati (Mika & Vasylenko, 2025).

Podobno velja za pisave. Komponenta *next/font* poskrbi, da se pisave naložijo tako, da ne povzročajo neprijetnega premikanja vsebine (CLS). Uporabnik med nalaganjem strani torej ne opazi, da bi mu besedilo spreminjalo položaj (Next.js, brez datuma).

Next.js samodejno izvaja *code splitting*. Vsaka stran dobi svoj paket JavaScripta, zato se ob obisku naloži le tista koda, ki je za to stran res potrebna. To pospeši začetno nalaganje in zmanjša količino prenesenih podatkov, kar je še posebej pomembno na počasnejših omrežjih (Mika & Vasylenko, 2025).

### 3.1.6 Celovitost ogrodja

Next.js ni zgolj orodje za izdelavo uporabniških vmesnikov, ampak omogoča pisanje strežniške kode znotraj istega projekta. Takšen pristop imenujemo celovita arhitektura (ang. *full-stack*), saj razvijalcu ni treba vzdrževati ločenega backenda, ampak lahko vse skupaj gradi v enem okolju (Mika & Vasylenko, 2025).

Strežniška logika, na primer dostop do podatkovne baze, je lahko implementirana neposredno v komponentah Server Components, kar poenostavi razvoj aplikacije. Kadar želi bolj strukturirane API točke (npr. za komunikacijo z zunanjimi storitvami), lahko uporabi datoteke *route.ts*, ki so del iste strukture kot druge strani (Next.js, brez datuma).

Ko je aplikacija pripravljena, je njen deployment (objava aplikacije) mogoč na različnih platformah. Next.js deluje na Vercelu, Azure, AWS, Virtual Private Server (VPS) in drugod. Vercel ponuja najbolj optimizirano izkušnjo, to je samodejno statično optimizacijo, *edge caching* in podporo za serverless funkcije (Kazanivska & Muzyka, 2024).

### 3.1.7 Zaključek

Arhitektura Next.js sloni na treh ključnih mehanizmih, in sicer datotečnem usmerjanju, ki poenostavlja organizacijo kode, Server Components, ki omogočajo izvajanje logike na strežniku in s tem zmanjšujejo količino poslanega JavaScripta in hibridnem upodabljanju, ki razvijalcu omogoča, da za vsako stran posebej izbere najprimernejšo strategijo. Poleg tega Next.js ni namenjen zgolj izdelavi uporabniških vmesnikov, ampak s podporo za pisanje strežniške kode, vgrajenimi optimizacijami in prilagodljivimi možnosti postavitve predstavlja

celovito (ang. *full-stack*) rešitev, primerno za najrazličnejše vrste spletnih aplikacij (Mika & Vasylenko, 2025) (Kazanivska & Muzyka, 2024).

## *3.2 Arhitekturne značilnosti SvelteKit*

Arhitektura spletnega ogrodja opredeljuje temeljne mehanizme, ki določajo, kako se aplikacija gradi, kako se koda izvaja in kako podatki potujejo skozi sistem. Za razliko od ogrodij, ki se izvajajo v brskalniku, Svelte deluje kot prevajalnik, to pomeni, da se komponente ob izgradnji pretvorijo v navaden JavaScript, ki neposredno spreminja stran. SvelteKit zato ne potrebuje navideznega DOM-a (ang. *virtual DOM*), kar pomeni manjšo porabo pomnilnika in hitrejše delovanje (Bratslavsky, 2025) (Ulili, 2025).

### 3.2.1 Usmerjanje

SvelteKit uporablja datotečno usmerjanje. Mapa src/routes predstavlja korensko mapo, struktura map pa neposredno določa URL poti. Mapa src/routes/bout/+page.svelte na primer ustreza URL-ju /about (SvelteKit, 2026). Vsaka mapa lahko vsebuje posebne datoteke z rezerviranimi imeni:

- +layout.svelte – za definiranje skupnih layoutov, ki se uporabljajo na več straneh;
- +page.svelte – za posamezne strani aplikacije;
- +error.svelte – za prikaz ob napaki;
- +page.js ali +page.server.js – za podatke strani (.server.js se izvaja samo na strežniku);
- +layout.js ali +layout.server.js – za podatke layouta

(SvelteKit, 2026) (Bratslavsky, 2025).

Dinamične poti se označujejo z oglatimi oklepaji ([slug]), kar omogoča zajemanje spremenljivih delov URL-ja. Skupine poti z oklepaji (ang. *group*) omogočajo logično organizacijo poti brez vpliva na URL (Lucky, 2023).

### 3.2.2 Komponentni model

Svelte komponente so zapisane v eni datoteki (.svelte), ki združuje HTML, CSS in JavaScript. Reaktivno deluje samodejno, – ko razvijalec spremeni vrednost spremenljivke, se posodobi tisti del zaslona, ki je od nje odvisen (Brown, 2025).

V nasproti z Reactom Svelte nima ločenih »strežniških« in »odjemalskih« komponent na ravni datotek. Razvijalec se odloči ali bo podatke pridobil na strežniku (+page.server.js) ali pa bo ista koda tekla na strežniku in v brskalniku (+page.js). Komponente so povsod enake (Bratslavsky, 2025) (SvelteKit, 2026).

### 3.2.3 Upodabljanje

SvelteKit podpira več načinov upodabljanja, ki jih razvijalec kombinira znotraj iste aplikacije.

Statično generiranje pomeni, da se stran pripravi vnaprej, ob izgradnji. Primerno je za vsebine, ki se redko spreminjajo, kot so blogi ali dokumentacija. Razvijalec ga omogoči z izvozom *export const prerender = true* v datoteki poti (SvelteKit, 2026).

Strežniško upodabljanje pomeni, da se stran sestavi ob vsaki zahtevi. Primerno je za podatke, ki se pogosto spreminjajo, ali za personalizirano vsebino. Privzeto SvelteKit uporablja SSR, razen če razvijalec eksplicitno določi drugače (Brown, 2025).

Odjemalsko upodabljanje pomeni, da se stran izriše v brskalniku. Uporabno je za dele aplikacije, ki jih iskalniki ne rabijo indeksirati (Brown, 2025). Razvijalec ga omogoči z izvozom *export const ssr = false* v datoteki poti (SvelteKit, 2026).

### 3.2.4 Podatkovni tokovi

SvelteKit ponuja več načinov pridobivanja in prikazovanja podatkov.

Za pridobivanje podatkov SvelteKit uporablja *load* funkcije, ki jih lahko definiramo v dveh vrstah datotek:

- *+page.js* ali *+layout.js* – univerzalne *load* funkcije, ki se izvajajo na strežniku (ob prvem obisku) in v brskalniku (ob nadaljnjih navigacijah).

- *+page.server.js* ali *+layout.server.js* – strežniške *load* funkcije, ki se izvajajo izključno na strežniku. Primerne so za dostop do baze podatkov, uporabo zasebnih okoljskih spremenljivk ali nastavljanje piškotkov

(SvelteKit, 2026).

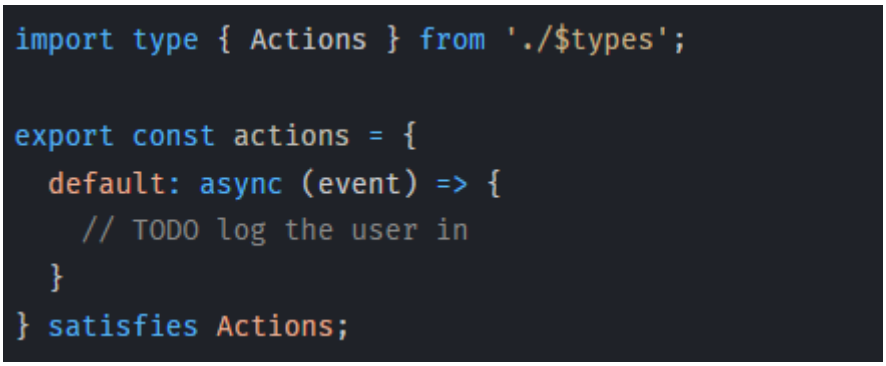
Podatki, ki jih *load* funkcija vrne, so v komponenti na voljo preko *data* objekta. Če imamo na isti poti obe vrsti load funkcij, se najprej izvede strežniška, nato pa univerzalna (SvelteKit, 2026). Tak enoten način pridobivanja podatkov poenostavlja razvoj, saj ni treba uporabljati različnih pristopov za različna okolja (Bratslavsky, 2025).

```
import type { PageLoad } from './$types';

export const load: PageLoad = ({ params }) => {
  return {
    post: {
      title: `Title for ${params.slug} goes here`,
      content: `Content for ${params.slug} goes here`
    }
  };
};
```

Slika 4 – Primer Universal Load funkcije v SvelteKit
Vir: (https://svelte.dev/docs/kit/load)

Za spreminjanje podatkov (na primer ob oddaji obrazca) SvelteKit uporablja form actions. Te definiramo v isti *+page.server.js* datoteki. Vsaka akcija je asinhrona funkcija, ki prejme dogodek z zahtevo (SvelteKit, 2026). Lahko imamo privzeto akcijo z imenom *default* ali več poimenovanih akcij (npr. login, register) v isti datoteki. V obrazcu akcijo pokličemo tako, da v *action* atributu navedemo pot z *?/ime-akcije* (SvelteKit, 2026).




Slika 5 – Primer Default Form Actions funkcije v SvelteKit
Vir: (https://svelte.dev/docs/kit/form-actions)

Če akcija uspe lahko vrne podatke, ki so v komponenti na voljo preko *form* objekta. Ob napaki lahko uporabimo pomožno funkcijo *fail*, ki statusno kodo in podatke o napaki pošlje nazaj na obrazec (SvelteKit, 2026). Po uspešni izvedbi akcije se stran samodejno osveži s posodobljenimi podatki. Če želimo uporabnika preusmeriti na drugo stran, lahko znotraj akcije uporabimo *redirect*.

### 3.2.5 Vgrajene optimizacije

SvelteKit vključuje kar nekaj optimizacij, ki jih ni treba posebej nastavljati.

Majhna velikost paketov: Ker Svelte prevaja komponente v običajen JavaScript, v brskalnik ni treba pošiljati celega ogrodja, zato so začetni paketi opazno manjši kot pri ogrodjih, ki uporabljajo virtualni DOM (Bratslavsky, 2025).

Avtomatsko cepljenje kode (ang. *code splitting*): Vsaka stran dobi svoj paket JavaScripta, zato se ob obisku naloži le tista koda, ki je za to stran res potrebna. To pospeši začetno nalaganje in zmanjša količino prenesenih podatkov (SvelteKit, 2026).

Prednalaganje sredstev (ang. *asset prelaoding*): SvelteKit samodejno prednaloži datoteke, potrebne za prikaz strani, s čimer prepreči zaporedno nalaganja, ki bi upočasnilo prikaz (SvelteKit, 2026).

Istočasno nalaganje podatkov (ang. *parallel loading*): Več *load* funkcij se izvaja hkrati, kar preprečuje zaporedno čakanje in pospeši prikaz strani (SvelteKit, 2026).

## *3.3 Ključne razlike in podobnosti*

Kot je prikazano v poglavjih 3.1.1 in 3.2.1, Next.js in SvelteKit uporabljata datotečno usmerjanje, vendar se razlikujeta v začetni mapi in poimenovanju datotek. Next.js v App Routerju uporablja mapo *app*, medtem ko SvelteKit uporablja mapo *src/routes*. Razlike so prikazane v tabeli 1.

Tabela 1: Primerjava usmerjanja v Next.js in SvelteKit

| **Ogrodje** | **Začetna mapa** | **Route/datoteka** | **Končni URL** |
|---|---|---|---|
| **Next.js** | app | app/about/page.tsx | /about |
| **SvelteKit** | src/routes | src/routes/about/+page.svelte | /about |
| **Next.js** | app | app/blog/[slug]/page.tsx | /blog/:slug |
| **SvelteKit** | src/routes | src/routes/blog/[slug]/+page.svelte | /blog/:slug |
| **Next.js** | app | app/api/users/route.ts | /api/users |
| **SvelteKit** | src/routes | src/routes/api/users/+server.js | /api/users |

*Vir: povzeto po* (Next.js, brez datuma) *in* (SvelteKit, 2026)

Čeprav sta si sistema podobna, saj oba uporabljata strukturo map za določanje poti, je glavna razlika v tem, da Next.js ločuje strani z datoteko page.tsx. SvelteKit pa uporablja datoteke z znakom +, na primer +page.svelte, +layout.svelte, +server.js. Next.js je tesno povezan z Reactovim komponentnim modelom in App Routerjem, SvelteKit pa uporablja enoten pristop, kjer so strani, layouti in strežniška logika jasno označeni s posebnimi datotekami.

Tabela 2 - Primerjava osnovnih značilnosti ogrodij Next.js in SvelteKit

| **Ogrodje** | **Leto nastanka** | **Zadnja verzija na dan 22. 6. 2026** | **Cena** |
|---|---|---|---|
| **Next.js** | 2016 | 16.2.9 | brezplačno, odprtokodno, MIT licenca |
| **SvelteKit** | 2022 | 2.65.0 | brezplačno, odprtokodno, MIT licenca |

*Vir: povzeto po* (Vercel, 2016), (Svelte, 2022), (Vercel, brez datuma) *in* (Svelte, brez datuma)

Next.js je starejše in bolj uveljavljeno ogrodje, saj je bilo prvič izdano leta 2016, medtem ko je SvelteKit različico 1.0 dosegel decembra 2022. Next.js ima zato večji ekosistem in širšo podporo skupnosti, SvelteKit pa je mlajši, enostavnejši in bolj usmerjen v manjšo količino kode ter hitrejše začetno delovanje. Obe ogrodji sta brezplačni in odprtokodni, zato cena ni odločilen dejavnik izbire. Pomembnejši so arhitekturni pristop, izkušnje razvojne ekipe, zahteve projekta in način objave aplikacije.

# 4 RAZVIJALSKA IZKUŠNJA IN DOSTOPNOST OGRODIJ

Razvijalska izkušnja je pri izbiri spletnega ogrodja skoraj tako pomembna kot tehnične zmogljivosti, saj vpliva na to, kako hitro lahko ekipa začne z razvojem, kako enostavno je vzdrževati projekt in koliko časa porabimo za iskanje rešitev, ko se nekaj zatakne. V tem poglavju primerjam Next.js in SvelteKit z vidika učenja, kompleksnosti razvoja in podpore skupnosti. To so vidiki, ki jih razvijalci najbolj občutijo pri vsakdanjem delu.

## *4.1 Učenje in dokumentacija*

Dostopnost učenja in kakovost dokumentacije sta pri izbiri spletnega ogrodja zelo pomembni, predvsem za razvijalce začetnike. Dobra dokumentacija pomeni manj časa za iskanje odgovorov in manj začetne frustracije.

Next.js ima zelo obsežno in podrobno dokumentacijo, ki pokriva vse od osnov do naprednih tem (Next.js, brez datuma). Ker Next.js temelji na Reactu, ima dostop do ogromnega ekosistema, in sicer ima na voljo ogromno paketov, knjižnic in orodij, ki jih je razvila skupnost (Lucky, 2023). To pomeni, da lahko razvijalci hitro najdejo rešitve za pogoste probleme. Ulili (Ulili, 2025) pa vseeno ugotavlja, da Next.js zahteva solidno predznanje Reacta. Razvijalec mora najprej razumeti hook-ov, življenjski cikel komponent in React vzorce, šele nato se lahko loti konceptov, ki so značilni za Next.js, kot so strežniške komponente, odjemalske komponente in App Router. Začetnike lahko to hitro preobremeni.

SvelteKit dokumentacija je krajša, bolj osredotočena na bistvo in razvijalca postopoma vodi skozi koncepte (Ulili, 2025). Ker Svelte uporablja običajen HTML in CSS, se razvijalec ne uči povsem nove sintakse. Reaktivnost deluje z enostavnim prirejanjem vrednosti, napake pa prevajalnik pokaže točno tam, kjer so nastale. Učna krivulja je minimalna, saj uporablja standardne spletne tehnologije, medtem ko Next.js zahteva poznavanje Reacta, JSX in hook-ov (Lucky, 2023).

## *4.2 Kompleksnost razvoja*

Pomemben faktor je tudi, kako zahtevno je dejansko razvijati aplikacijo. Pri tem gre za količino potrebne konfiguracije, število odločitev, ki jih mora sprejeti razvijalec, ter obseg nastavitev, ki jih mora opraviti ročno.

Lucky (Lucky, 2023) v svoji primerjavi ugotavlja, da je razvijalska izkušnja pri SvelteKitu bolj poenostavljena, saj je manj standardne kode, sintaksa je intuitivna, reaktivnost deluje preprosto, razvojni strežnik pa z Vite-om omogoča takojšnje osveževanje ob spremembah. Next.js po drugi strani ponuja zrela orodja in bogat ekosistem, ampak zahteva bolj eksplicitino upravljanje stanja preko hook-ov.

(Mika & Vasylenko, 2025) poudarjata, da Next.js temelji na filozofiji »nič konfiguracije«. Ko razvijalec ustvari nov projekt, takoj dobi avtomatsko cepljenje kode, vgrajeno podporo za CSS in Sass, integracijo TypeScripta, optimizirane slike in hitro osveževanje. To pomeni, da se razvijalec lahko hitro loti pisanja kode, ne da bi izgubljal čas z osnovno konfiguracijo. Za tiste, ki so novi React okolju, to močno zmanjša kognitivno obremenjenost.

(Ulili, 2025) dodaja, da ima SvelteKit vgrajeno upravljanje stanja (ang. *stores*), ki pokrije večino potreb, brez dodatnih zunanjih knjižnic. Next.js pa se bolj zanaša na zunanje rešitve, kot je npr. Redux. To razvijalcu da več svobode, ampak zahteva tudi več odločanja. SvelteKit ima manjšo količino standardne kode in manj konfliktnih adapterjev, kar pomeni manjšo vzdrževalno obremenitev, ko je aplikacija enkrat v produkciji (Lucky, 2023).

Na splošno lahko ugotovimo, da SvelteKit zaradi vgrajenih rešitev in manjšega števila odločitev omogoča hitrejši začetek razvoja, Next.js pa zaradi svoje fleksibilnosti zahteva več načrtovanja.

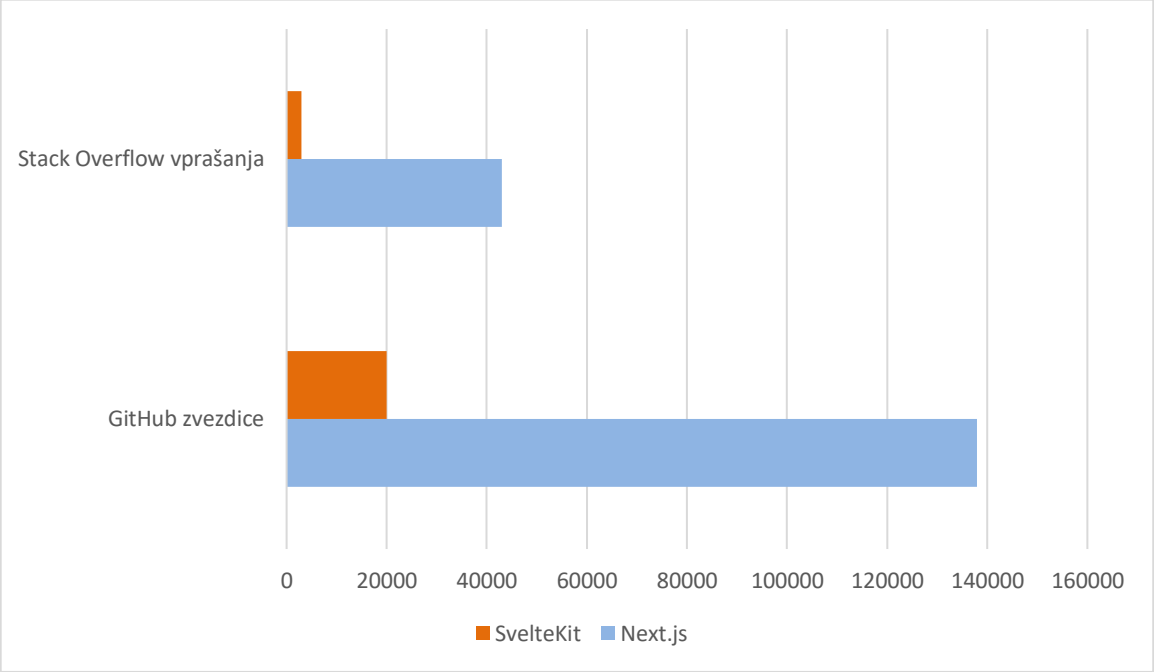
## *4.3 Podpora skupnosti*

Ko med razvojem naletimo na težavo je pomembno, kako hitro lahko najdemo rešitev. Tu pride do izraza velikost in aktivnost skupnost okoli ogrodja.

Aktivna skupnost omogoča izmenjavo znanja, razvoj dodatnih orodij in hitro odzivanje na težave, s katerimi se razvijalci srečujejo pri delu z ogrodjem. Pri primerjavi Next.js in SvelteKit se razlike v podpori skupnosti kažejo predvsem v velikosti skupnosti, zrelosti ekosistema in količini razpoložljivih virov.

Next.js je tesno povezan z ekosistemom React, ki je eden najbolj razširjenih in uveljavljenih v sodobnem spletnem razvoju. To se odraža v številu zvezdic na GitHubu, kjer ima Next.js približno 138.000 zvezdic (Vercel, brez datuma), medtem ko jih ima SvelteKit okoli 20.000 (Svelte, brez datuma). Next.js ima zato dostop do širše baze razvijalcev in več odprtokodnih knjižnic.

Velikost skupnosti je mogoče oceniti tudi s pomočjo platforme Stack Overflow, kjer razvijalci postavljajo vprašanja in delijo rešitve oziroma odgovore. Kot prikazuje Tabela 3, ima oznaka Next.js več kot 43.000 vprašanj, medtem ko jih ima SvelteKit približno 3000. Razlika je opazna tudi pri osnovnih tehnologijah, saj ima React bistveno večjo prisotnost na platformi Stack Overflow kot Svelte. Večje število vprašanj in odgovorov razvijalcem omogoča hitrejše iskanje rešitev in dostop do večjega nabora praktičnih primerov.




Slika 6 – Primerjava prisotnosti skupnosti na GitHubu in Stack Overflow

Vir: povzeto po (Vercel, brez datuma), (Svelte, brez datuma) in (Stack Overflow, 2026)

Podatki iz raziskave (State of JS, 2025) kažejo, da so tehnologije iz React ekosistema zelo razširjene, kar prispeva k obsegu in stabilnosti skupnostne podpore ogrodja. Prav tako pa Next.js in SvelteKit uvršča med najbolj uporabljeni meta ogrodji, kar kaže, da imata obe ogrodji aktivno skupnost, čeprav se po velikosti razlikujeta. V uradni dokumentaciji Next.js je navedeno, da ima Next.js več kot 5 milijonov tedenskih prenosov in veliko aktivno skupnost po vsem svetu (Next.js, brez datuma).

Kljub manjši skupnost pa SvelteKit v zadnjih letih pridobiva na priljubljenosti in velja za enega najbolj obetavnih meta ogrodij med razvijalci (State of JS, 2025).

# 5 ZMOGLJIVOST IN VARNOST

Zmogljivost in varnost sta pri razvoju sodobnih spletnih aplikacij zelo pomembni, saj imata neposreden vpliv na uporabniško izkušnjo in delovanje sistema. Uporabniki pričakujejo hitro nalaganje strani, nemoteno delovanje in zaščito osebnih podatkov. Po drugi strani pa razvijalci in organizacije stremijo k rešitvam, ki so dovolj stabilne in zanesljive tudi na dolgi rok.

V tem poglavju je zato narejena primerjalna analiza Next.js in SvelteKit z vidika zmogljivosti in varnosti. Ker se v praksi zmogljivost in varnost pogosto obravnavata skupaj, je smiselno ta vidika obravnavati v istem sklopu, pri čemer se najprej osredotočam na zmogljivost, nato pa še na varnostne mehanizme.

## *5.1 Primerjava hitrosti delovanja*

Hitrost delovanja spletne aplikacije je eden ključnih dejavnikov uporabniške izkušnje, saj vpliva nato kako odziven se sistem zdi uporabniku in, na to, ali uporabnik na strani sploh ostane. Na zmogljivost aplikacije vpliva več stvari, predvsem način upodabljanja vsebine, količina JavaScript kode v brskalniku in učinkovitost nalaganja podatkov. Next.js in SvelteKit omogočata sodobne pristope k upodabljanju, ampak se razlikujeta v implementaciji, kar se lahko pozna pri hitrosti delovanja.

Next.js uporablja hibridni pristop, ki združuje strežniško upodabljanje, statično generiranje in odjemalsko upodabljanje. Ker temelji na Reactu, v brskalnik pošilja runtime ogrodja, kar povečuje začetno velikost paketov. Po podatkih raziskave (DevMorph Team, 2026) znaša začetni JavaScript paket pri Next.js približno 120KB, medtem ko je pri SvelteKitu 42KB. (Ulili, 2025) prav tako navaja, da so paketi pri SvelteKitu 60-80 % manjši kot pri Next.js. To se kaže tudi v času do interaktivnosti (ang. *Time to Interactive*), ki je pri Next.js znašal 2,4 sekunde, pri SvelteKitu pa 0,8 sekunde (DevMorph Team, 2026).

SvelteKit do visoke zmogljivosti pride na drugačen način. Temelji na kompilacijskem pristopu, pri katerem se komponente prevedejo v optimiziran JavaScript že v času gradnje aplikacije. Ker v brskalniku ni dodatnega runtime okolja, je količina kode, ki se mora izvesti na strani odjemalca manjša (Ulili, 2025) (Lucky, 2023). Posledično so začetni paketi manjši in nalaganje strani hitrejše (Ulili, 2025).

Za podporo teoretični analizi je bila upoštevana znanstvena študija, ki primerja več sodobnih SSR ogrodij, med njimi Next.js in SvelteKit, z uporabo orodij Google Lighthouse, PageSpeed Insights in JMeter (Zaidan & Suyatno, 2025). Poleg tega so bili uporabljeni tudi rezultati neodvisnih benchmarkov (Sanchez, 2024) (DevMorph Team, 2026). Pri tem je treba upoštevati, da gre za različne metodologije in okolja, zato rezultatov ni smiselno posploševati na vse primere. Kljub temu dajejo dober vpogled v splošne trende.

Pri primerjavi zmogljivosti se pogosto uporabljajo standardizirane metrike, ki omogočajo bolj objektivno oceno uporabniške izkušnje. V analizi so bile uporabljene naslednje:

- Čas do prve vsebine (ang. *First Contentful Paint*, kratica FCP): Predstavlja čas, ki preteče od začetka nalaganja strani do trenutka, ko se v brskalniku prvič prikaže vsebina, kot je besedilo ali slika. Nižja vrednost FCP pomeni hitrejši prvi vizualni odziv strani.
- Indeks hitrosti (ang. *Speed Index*, kratica SI): Meri hitrost, s katero se vsebina strani postopoma izriše v vidnem delu zaslona. Nižja vrednost pomeni, da se večina vsebine uporabniku prikaže hitreje.
- Skupni čas blokade (ang. *Total Blocking Time*, kratica TBT): Označuje skupni čas, v katerem je glavna nit brskalnika blokirana zaradi izvajanja JavaScript kode in zato ne more odgovarjati na uporabniške interakcije. Višje vrednosti TBT negativno vplivajo na odzivnost aplikacije.
- Največji prikazani element (ang. *Largest Contentful Paint*, kratica LCP): Meri čas nalaganja največjega vidnega elementa na strani, kot je večja slika ali naslov. Ta metrika je pomembna za zaznano hitrost nalaganja strani.
- Kumulativni premik postavitve (ang. *Cumulative Layout Shift*, kratica CLS): Meri stabilnost postavitve strani med nalaganjem. Nižje vrednosti pomenijo manj nepričakovanih premikov vsebine, kar izboljša uporabniško izkušnjo

(conductor, 2025).

Rezultati testiranja z orodjema Google Lighthouse in PageSpeed Insight v študiji (Zaidan & Suyatno, 2025) kažejo, da se zmogljivosti razlikujejo glede na posamezne metrike. Pri testiranju z Google Lighthouse je Next.js dosegel najboljše rezultate pri CLS, kjer je bil prvi v 15 od 21 testnih scenarijev. Pri drugih metrikah (FCP, TBT, SI, LCP) se je Next.js večinoma uvrščal na drugo oziroma tretje mesto, pri čemer je bil najbolj uspešen Astro JS.

SvelteKit je pri testiranju z Google Lighthouse najboljši rezultat prav tako dosegel pri metriki CLS (drugo mesto), medtem ko je pri drugih metrikah bil nekoliko slabši.

Pri testiranju s PageSpeed Insights so bili rezultati nekoliko drugačni. Next.js je bil najboljši pri TBT, kjer je bil skupaj z Astro JS zmagovalec v 9 od 21 scenarijev, in pri metriki LCP, kjer je bil zmagovalec v 12 scenarijih. Pri metriki CLS je bil drugi, pri FCP tretji, pri SI pa četrti. SvelteKit je pri tem testiranju izstopal predvsem pri metriki CLS, kjer je bil najboljši v 18 od 21 scenarijev. Pri drugih metrikah se je uvrščal na tretje ali četrto mesto, pri FCP in LCP pa ni bil zmagovalec v nobenem scenariju (Zaidan & Suyatno, 2025).

Študija z orodjem JMeter je proučevala tudi obnašanje ogrodij pod obremenitvijo in sicer pri 100, 300 in 500 sočasnih uporabnikih. Pri vseh treh obremenitvah se je Next.js izkazal kot najbolj konsistenten in stabilen, z dobrimi rezultati pri odzivnem času, številu napak in prepustnosti. SvelteKit se je pri vseh treh scenarijih uvrstil na drugo mesto, takoj za Next.js (Zaidan & Suyatno, 2025).

Neodvisni benchmark Pau Sancheza je pokazal, da SvelteKit pri preprostem »hello world« primeru doseže 4.063 zahtevkov na sekundo, Next.js pa 2.570. Zahtevki na sekundo pomenijo, koliko uporabnikov lahko strežnik postreže v eni sekundi. Višje število pomeni boljšo zmogljivost in mamnjšo potrebo po dodatnih strežnikih. Tudi pri testu s komponento so rezultati pokazali podobno razmerje med ogrodjema. SvelteKit je dosegel 4.265 zahtevkov na sekundo, Next.js pa 2.794. Pri testu s klicem na lokalni API pa je SvelteKit dosegel 2.286, Next.js pa 388 zahtevkov na sekundo. V vseh testih je SvelteKit precej prehitel Next.js, pri zadnjem testu pa je bil padec pri Next.js še posebej izrazit (Sanchez, 2024). (DevMorph Team, 2026) poroča, da je SvelteKit pri obremenitvenih testih dosegel 1.200 zahtevkov na sekundo, Next.js pa 850, kar pomeni 41 % večjo zmogljivost na enaki strojni opremi.

Tabela 3: Primerjava izbranih kazalnikov zmogljivosti

| **Kazalnik** | **Next.js** | **SvelteKit** | **Vir** |
|---|---|---|---|
| **Začetni JavaScript paket** | 120 KB | 42 KB | (DevMorph Team, 2026) |
| **Time to Interactive** | 2,4 s | 0,8 s | (DevMorph Team, 2026) |
| **Hello world zahtevki/s** | 2570 | 4063 | (Sanchez, 2024) |
| **Komponenta zahtevki/s** | 2794 | 4265 | (Sanchez, 2024) |
| **Lokalni API zahtevki/s** | 388 | 2286 | (Sanchez, 2024) |

Vir: povzeto po (DevMorph Team, 2026), (Sanchez, 2024)

Next.js torej dosega zelo dobre rezultate pri metrikah, kot so TBT, LCP in CLS, poleg tega pa se dobro obnese tudi pri večjih obremenitvah. SvelteKit izstopa predvsem pri stabilnosti prikaza (CLS) in pri hitrem nalaganju v manj kompleksnih scenarijih. Prav tako ima manjšo velikost paketov in boljšo zmogljivost pri visokih obremenitvah po večini benchmarkov. Razlike sicer niso zanemarljive, vendar jih je treba vedno razumeti v kontekstu konkretne aplikacije in načina uporabe (Zaidan & Suyatno, 2025) (DevMorph Team, 2026) (Sanchez, 2024).

## *5.2 Optimizacija in obremenitev sistema*

Poleg hitrosti nalaganja je pomembno tudi, kako aplikacija obremenjuje sistemske vire, kot so procesor, pomnilnik in omrežje. Optimizacija rabe teh virov lahko vpliva na delovanje aplikacije, zlasti na napravah z omejenimi zmogljivostmi.

### 5.2.1 Merjenje obremenitve in diagnostična orodja

Za analizo obremenitve sistema imamo na voljo različna orodja. Brskalnikova orodja za razvijalce omogočajo spremljanje omrežnih zahtevkov, velikosti datotek in porabe pomnilnika (Dutton, brez datuma). SvelteKit v dokumentaciji priporoča uporabo orodij, kot sta PageSpeed Insights in WebPageTest za analizo zmogljivosti. Prav tako pa opozarja, da testiranje v razvojnem načinu ni zanesljivo in ga je treba opraviti na že zgrajeni aplikaciji (SvelteKit, 2026). Orodja kot je Lighthouse, poleg hitrosti nalaganja analizirajo tudi metrike, kot sta količina neuporabljene kode in velikost DOM strukture (Dutton, brez datuma).

### 5.2.2 Obremenitev pomnilnika in procesorja

Na to koliko sta obremenjena pomnilnik in procesor, vpliva predvsem količina JavaScript kode in način kako ogrodje posodablja uporabniški vmesnik.

SvelteKit med gradnjo aplikacije kodo pretvori v čisti JavaScript, ki neposredno spreminja DOM. Kot smo že ugotovili v poglavju 5.1, v brskalniku ni obsežnega ogrodja, zato so paketi manjši (20 - 40 KB) in koda neposredno manipulira z DOM, kar zmanjša porabo pomnilnika in obremenitev procesorja (Lucky, 2023). To se še posebej opazi pri zahtevnejših operacijah, kot je prikaz velikih tabel, in na mobilnih napravah, ki imajo omejene zmogljivosti (Lucky, 2023).

Next.js, ki temelji na Reactu, v brskalnik pošilja runtime (približno 70 - 90 KB) in uporablja virtualni DOM. Reactov mehanizem za usklajevanje (ang. *reconciliation*) porabi več procesorskega časa med drobnimi posodobitvami, prav tako pa ima virtualni DOM višji pomnilniški odtis (ang. *memory overhead*) (Lucky, 2023). Po drugi strani pa omogoča napredne vzorce, kot je sočasno upodabljanje (ang. *concurrent rendering*) (Lucky, 2023).

V Chromu lahko preko upravitelja opravila spremljamo porabo pomnilnika. Za podrobnejšo analizo pa lahko uporabimo orodja za profiliranje, kot sta časovnica (ang. *timeline*) in profiliranje pomnilnika (Dutton, brez datuma).

### 5.2.3 Optimizacija prenosa podatkov

Na količino podatkov, ki se prenesejo med strežnikom in odjemalcem, vplivajo predvsem vgrajene optimizacije ogrodij. Next.js in SvelteKit ponujata različne mehanizme za zmanjšanje prenosa in hitrejše nalaganje vsebin.

Next.js v uradni dokumentaciji opisuje več vgrajenih optimizacij. Komponenta *next/image* samodejno optimizira slike z lenim nalaganjem (ang. *lazy loading*), prilagajanjem velikosti glede na napravo in streženjem, v formatih kot je WebP. Komponenta *next/link* omogoča prednalaganje povezanih strani v ozadju. Next.js prav tako samodejno deli kodo, tako da uporabnik ob prvem obisku prenese le kodo, potrebno za trenutni pogled (Next.js, brez datuma).

SvelteKit opisuje več tehnik za optimizacijo prenosa podatkov. Omenja združevanje zahtevkov (ang. *request coalescing*), vzporedno nalaganje podatkov iz *load* funkcij in preprečevanje t.i. *waterfalls* – zaporedni zahtevki, ki povečujejo zakasnitve. Pri optimizaciji vsebin SvelteKit priporoča stiskanje in pretvorbo videoposnetkov v spletne formate, *subsetting* za pisave in prednalaganje preko *handle* funkcije. Za slike svetuje uporabo pristopov, ki omogočajo leno nalaganje in prilaganje velikosti glede na napravo (SvelteKit, 2026).

### 5.2.4 Vpliv načinov upodabljanja na obremenitev strežnika

Način upodabljanja vpliva na to, kako močno je obremenjen strežnik ob vsakem obisku. Hygraph (Chidi, 2025) v preglednici primerjave navaja, da obe ogrodji podpirata SSR in SSG. Pri SSG se strani pripravijo vnaprej med gradnjo, kar pomeni, da strežnik ob obisku ne opravlja dodatnega dela. Next.js ima vgrajeno podporo za ISR, ki omogoča posodabljanje statičnih strani po gradnji. SvelteKit te funkcije nima vgrajene, ampak jo je mogoče doseči z drugimi pristopi (Chidi, 2025). Leobit (Kazanivska & Muzyka, 2024) opisuje različne mehanizme predpomnjenja v Next.js, kot so *Data Cache*, *Full Route Cache* in *Router Cache*. Ti zmanjšajo obremenitev strežnika, saj ponovno uporabijo že pripravljene vsebine, namesto da bi jih vsakič znova generirali.

## *5.3 Vgrajeni varnostni mehanizmi*

Pri razvoju spletnih aplikacij je varnost eden temeljnih vidikov, ki ga je treba upoštevati že v zgodnjih fazah načrtovanja. Splet je namreč pol napadalcev, ki želijo dostopati do podatkov, jih spreminjati ali onemogočiti delovanje strani (Mozilla, brez datuma). Sodobna ogrodja ponujajo različne vgrajene mehanizme in priporočene vzorce, ki razvijalcem pomagajo zaščititi aplikacije pred pogostimi ranljivostmi. Ključno vodilo pri tem je, da spletna aplikacija ne sme zaupati nobenim podatkom, ki prihajajo iz brskalnika. Vedno jih je treba preveriti in očistiti (Mozilla, brez datuma).

### 5.3.1 Varnostni mehanizmi v Next.js

Next.js ponuja več vgrajenih funkcij in smernic za izboljšanje varnosti. Ena ključnih novosti je ločevanje med strežniškimi in odjemalskimi komponentami. Strežniške komponente tečejo izključno na strežniku in lahko varno dostopajo do okoljskih spremenljivk, skrivnosti in baz podatkov. Odjemalske komponente pa tečejo tudi v brskalniku, zato zanje veljajo strožji varnostni pogoji, in sicer ne smejo dostopati do občutljivih podatkov (Next.js, brez datuma).

Next.js priporoča vzpostavitev namenske plasti za dostop do podatkov (ang. *Data Access Layer*, kratica DAL), ki poskrbi za preverjanje avtorizacije in vrača le varna in minimalna podatkovna ogrodja. S tem se zmanjša tveganje za napake pri avtorizaciji in poenostavi pregled nad tem, kateri podatki so dostopni (Next.js, brez datuma). Prav tako svetuje uporabo oznake server-only, ki prepreči, da bi se koda, namenjena izključno strežniku, izvajala na odjemalcu.

Strežniške akcije (ang. *Server Actions*) so dodatno zaščitene pred CSRF napadi. Pri teh napadih napadalec žrtev ali njen brskalnik prisili, da pošlje zahtevo na ciljno stran z žrtvinimi poverilnicami in tako izvede nezaželeno akcijo (Mozilla, brez datuma). Server Actions uporabljajo izključno HTTP POST metodo in preverjajo, ali zahteva prihaja iz istega strežnika. Če je treba dovoliti zahteve z drugih domen, lahko to izrecno nastavimo v konfiguraciji (Next.js, brez datuma).

### 5.3.2 Varnostni mehanizmi v SvelteKit

Tudi SvelteKit vključuje več vgrajenih varnostnih mehanizmov, ki jih razvijalci konfigurirajo preko nastavitvene datoteke svelte.config.js.

Eden ključnih je zaščita pred CSRF napadi. SvelteKit za zahtevke POST, PUT, PATCH in DELETE privzeto preverja glavo origin (ta pove, s katere strani zahteva prihaja) in jo primerja z izvornim strežnikom. Če je treba dovoliti zahteve z drugih domen, lahko to izrecno dovolimo.

SvelteKit omogoča tudi natančno konfiguracijo vsebinske varnostne politike (ang. *Content Security Policy*, kratica CSP). Z njo lahko določimo, s katerih domen se smejo nalagati skripte, slike in druge vsebine, s čimer se lahko zaščitimo pred XSS napadi. To so napadi, kjer napadalci v spletno stran vstavijo škodljivo kodo, ki se nato izvaja v brskalnikih drugih uporabnikov (Mozilla, brez datuma) (Mozilla, brez datuma). Politiko lahko prilagodimo glede na to ali gre za dinamično generirane ali vnaprej pripravljene strani (SvelteKit, 2026).

Poleg tega SvelteKit omogoča varno upravljanje z okoljskimi spremenljivkami. Spremenljivke z določeno predpono (privzeto »PUBLIC_«) so varne za izpostavitev na odjemalcu, medtem ko so vse druge na voljo le na strežniku. To preprečuje razkritje občutljivih podatkov (SvelteKit, 2026).

### 5.3.3 Splošni varnostni koncepti in najboljše prakse

Poleg mehanizmov, ki jih ponujata Next.js in SvelteKit, morajo razvijalci poznati in uporabljati širše varnostne koncepte, ki veljajo za vse spletne aplikacije.

Zelo pomembno je upravljanje s skrivnostmi (gesla, API ključi, žetoni). Shranjevanje skrivnosti je na frontendu lahko izjemno tvegano, saj so te lahko dostopne napadalcem preko orodij za razvijalce ali ranljivosti, kot je XSS (Pentest Wizard, brez datuma). Priporočljivo je centralizirano upravljanje skrivnosti z namenskimi orodji (npr. AWS Secrets Manager). Ta omogočajo nadzor dostopa, avtomatsko rotacijo (zamenjava starih skrivnosti z novimi) in beleženje dostopov. Skrivnosti nikoli ne smejo biti zapisane v izvorni kodi (OWASP, brez datuma).

Prav tako je treba redno pregledovati in posodabljati vse komponente, knjižnice in ogrodja, saj neznane in zastarele različice pogosto vsebujejo znane ranljivosti, ki jih napadalci lahko izkoristijo (OWASP, 2021).

Poleg vsega naštetega so HTTP glave ena tistih stvari, ki jih pogosto spregledamo, lahko pa naredijo ogromno razliko. Če jih pravilno nastavimo, lahko preprečimo kup napadov, še preden se sploh začnejo. OWASP priporoča nekaj osnovnih glav, ki bi jih morala imeti vsaka resna spletna aplikacija:

- Content-Security-Policy: Skrbi za to, od kod lahko brskalnik nalaga skripe, slike in druge vire. S tem učinkovito preprečuje XSS in Clickjacking napade. Pri zadnjih gre za to, da napadalec žrtvino stran vstavi v skriti okvir (<iframe>) na svoji strani in uporabnika zvabi, da nevede klikne nekaj, kar ne bi smel (Mozilla, brez datuma) (OWASP, brez datuma).
- X-Content-Type-Options – nosniff: Ta glava brskalniku prepreči, da bi sam ugibal MIME vrsto datoteke. To je pomembno, ker lahko napadalec sicer doseže, da se datoteka (npr. slika) obnaša kot izvršljiva koda (Mozilla, brez datuma).
- Strict-Transport-Security: Brskalniku pove, naj s strežnikom komunicira izključno preko HTTPS. S tem preprečimo napade vrste »manipulator v sredini« (ang. *Manipulator in the Middle* ali *Man in the Middle*, kratica MITM), pri katerih napadalec prestreza ali spreminja komunikacijo med uporabnikom in strežnikom (Mozilla, brez datuma).
- Permissions-Policy: Omogoča nadzor nad tem, katere brskalnikove zmožnosti (npr. kamera, mikrofon) lahko stran uporablja. Če jih ne potrebujemo, jih raje onemogočimo.
- Odstranitev informativnih glav, kot sta Server in X-Powered-By: Te namreč razkrivajo, katere tehnologije tečejo na strežniku. Napadalcem to olajša iskanje ranljivosti, značilnih za določene strežnike ali ogrodja (OWASP, brez datuma).

# 6 PRAKTIČNA UPORABNOST IN PRIMERNOST ZA PROJEKTE

Arhitekturne značilnosti, zmogljivost, varnost in razvijalska izkušnja so pomembni dejavniki pri izbiri ogrodja, vendar je z akončno odločitev pomembno tudi, kako se ogrodje obnese v praksi. Torej za katere vrste projektov je najbolj primerno in kdaj se splača izbrati enega ali drugega. V tem poglavju so zato predstavljeni značilni primeri uporabe obeh ogrodij, na podlagi česar so podana priporočila za izbiro glede na značilnosti projekta.

## *6.1 Značilni primeri uporabe Next.js*

Next.js se je zaradi svoje zrelosti, obsežnega ekosistema in prilagodljivosti uveljavil kot ogrodje, ki je primerno za širok nabor projektov. Posebej pogosto se uporablja v okoljih, kjer so pomembna stablnost, razširljivost in integracija z obstoječimi React rešitvami.

**E-trgovine in vsebinsko bogata spletna mesta**

Next.js omogoča kombinacijo statičnega generiranja za hitro nalaganje in strežniškega upodabljanja za dinamične vsebine. Primeri uspešnih implementacij je *The Washington Post* in *Walmart* (Mika & Vasylenko, 2025).

**Kompleksne poslovne aplikacije in nadzorne plošče**

Next.js je primeren za aplikacije, ki zahtevajo avtentikacijo, nadzor dostopa in prikaz podatkov v realnem času. Vercel lastno nadzorno ploščo gradi z Next.js (Mika & Vasylenko, 2025).

**Spletna mesta z visokimi zahtevami glede SEO**

Zaradi podpore strežniškemu upodabljanju in statičnemu generiranju Next.js omogoča dobro optimizacijo za iskalnike. Vgrajene optimizacije, ko sta *next/image* in *next/font* prispevajo k boljšim rezultatom pri *Core Web Vitals* (Umeh, 2025).

**Projekti, kjer je pomembna povezanost z React ekosistemom**

Next.js je primeren za ekipe, ki že obvladajo React, saj omogoča dostop do tisočih React knjižnic in orodij (Lucky, 2023) (Ulili, 2025).

## 6.2 *Značilni primeri uporabe SvelteKit*

SvelteKit je zaradi svoje preprostosti, majhne velikosti paketov in kompilacijskega pristopa primeren za projekte, kjer so hitrost in nizka poraba virov ključni.

**Manjši in srednje veliki projekti s poudarkom na hitrosti**

Zaradi majhne velikosti paketov in odsotnosti runtime ogrodja je SvelteKit primeren za vsebinske strani, bloge, portfeljske strani in spletne trgovine z manjšim številom izdelkov (Bratslavsky, 2025) (Ulili, 2025). Meritve kažejo, da SvelteKit pri manj kompleksnih scenarijih dosega boljše rezultate od Next.js (Sanchez, 2024).

**Aplikacije z omejenimi viri na odjemalski strani**

Ker SvelteKit v brskalnik pošilja manj kode, je primeren za aplikacije, ki jih uporabljajo mobilne naprave ali naprave z omejenimi zmogljivostmi (Ulili, 2025) (Lucky, 2023).

**Hitri razvoj in prototipiranje**

SvelteKit zaradi enostavne sintakse in minimalne količine standardne kode omogoča hiter razvoj. Razvijalci, ki že poznajo HTML, CSS in JavaScript se lahko hitro znajdejo (Ulili, 2025) (Brown, 2025).

**Projekti, kjer je pomembna prilagodljivost deploymenta**

SvelteKit s sistemom adapterjev omogoča *deployment* na različne platforme (Node, Vercel, Netlify, Cloudflare, statični strežniki) brez spreminjanja kode (Lucky, 2023) (Vercel, brez datuma).

## 6.3 *Priporočila glede izbire ogrodja*

Na podlagi celotne analize arhitekturnih značilnosti, zmogljivosti, varnosti in razvijalske izkušnje lahko oblikujemo priporočila za izbiro ogrodja glede na značilnosti projekta.

**Priporočilo 1: Projekti z izrazitimi SEO zahtevami in veliko vsebine**

Pri vsebinskih portalih, e-trgovinah in spletnih mestih z velikim številom podstrani je pogosto primernejši Next.js, saj omogoča širok nabor upodabljanja in inkrementalno statično generiranje (ISR), kar je praktično pri rednem osveževanju vsebin. Takšna rešitev je smiselna, ko je pomemben kompromis med zmogljivostjo statične dostave in ažurnostjo podatkov.

**Priporočilo 2: Projekti z manjšimi ekipami ali kratkimi roki**

Pri manjših ekipah in projektih z omejenimi roki je pogosto ključna hitrost razvoja in zmanjšanje kompleksnosti konfiguracije. V takih primerih se SvelteKit pogosto izkaže kot primernejši zaradi poenotenega razvojnega modela in preglednejše strukture projekte.

**Priporočilo 3: Dolgoročni produkti in zrelost ekosistema**

Pri projektih z daljšo življenjsko dobo (npr. SaaS sistemi) je pomembna razpoložljivost razvijalcev, obstoječih rešitev in širša podpora skupnosti. V takih primerih je pogosto smiselno upoštevati razširjenost tehnologije v ekosistemu. Podatki o uporabi in razširjenosti meta ogrodij so prikazani v raziskavi State of JS Survey 2025, kar omogoča bolj informirano oceno tveganja izbire tehnologije (State of JS, 2025).

**Priporočilo 4: Infrastruktura in način deploymenta**

Če projekt zahteva fleksibilen deployment ali prilagajanje različnim produkcijskim okoljem, lahko SvelteKit ponudi prednost zaradi adapterjev, ki standardizirajo postopek priprave aplikacije za izbrano okolje. Pri Next.js je proces deploymenta prav tako dobro podprt, vendar je v okviru te primerjave poudarjena specifična značilnost SvelteKit, ki je neposredno vezana na mehanizem adapterjev.




Slika 7: Priporočila za izbiro med ogrodjema Next.js in SvelteKit glede na značilnosti projekta

Vir: lasten prikaz na podlagi rezultatov raziskave

# 7 DISKUSIJA

## *7.1 Povzetek ugotovitev*

V diplomskem delu sem primerjala dve sodobni spletni ogrodji, Next.js in SvelteKit. Analiza je zajela arhitekturo, zmogljivost, varnost, razvijalsko izkušnjo in praktično uporabnost. Ugotovitve so pokazale, da se ogrodji bistveno razlikujeta v temeljnih arhitekturnih pristopih.

Pri analizi arhitekture sem ugotovila, da Next.js temelji na Reactu in uvaja ločitev na strežniške in odjemalske komponente, kar omogoča prilagodljivo upodabljanje in dostop do obsežnega ekosistema. SvelteKit pa temelji na kompilacijskem pristopu, kjer se komponente pretvorijo v čisti JavaScript že ob gradnji, kar odpravlja potrebo po virtualnem DOM-u in zmanjšuje obremenitev odjemalca.

Pri zmogljivosti je SvelteKit dosegal boljše rezultate pri velikosti paketov, času do interaktivnosti in porabi pomnilnika, medtem ko se je Next.js izkazal kot bolj konsistenten pri obremenitvenih testih z večjim številom sočasnih uporabnikov. Obe ogrodji ponujata primerljive vgrajene varnostne mehanizme, ampak se razlikujeta v pristopu, in sicer ima Next.js več vgrajenih funkcionalnosti na ravni ogrodja, SvelteKit pa pušča več konfiguracije razvijalcu.

Razvijalska izkušnja se najbolj razlikuje v učni krivulji in kompleksnosti razvoja. SvelteKit je zaradi uporabe standardnih spletnih tehnologij in minimalne količine standardne kode bolj dostopen začetnikom, medtem ko Next.js zahteva predznanje Reacta, vendar zaradi širše skupnosti in obsežnega ekosistema omogoča lažje iskanje rešitev.

Glede na analizo sem oblikovala priporočila za izbiro ogrodja. Next.js je primernejši za večje, kompleksne projekte z visokimi SEO zahtevami kjer je pomembna prilagodljivost upodabljanja in dostop do obsežnega ekosistema. SvelteKit je primernejši za manjše in srednje velike projekte, kjer so ključni hitrost nalaganja, enostavnost razvoja in majhna poraba virov.

## *7.2 Preverjanje hipotez*

Na podlagi ugotovitev iz prejšnjih poglavij je mogoče ovrednotiti zastavljene hipoteze.

**H1: Next.js ponuja širši ekosistem in bolj razvito podporo skupnosti v primerjavi s SvelteKit.**

Hipotezo lahko potrdim. Analiza v poglavju 4.3 je pokazala, da ima Next.js občutno večjo skupnost (približno 138.000 zvezdic na GitHubu v primerjavi z 20.000 pri SvelteKitu). V poglavju 4.1 smo ugotovili, da ima Next.js zaradi povezanosti z Reactom dostop do ogromnega ekosistema z na tisoče paketi, knjižnicami in orodji, kar potrjuje širši ekosistem. SvelteKit ima v primerjavi z Next.js manjšo skupnost in omejen nabor paketov, čeprav postopoma raste.

**H2: SvelteKit je bolj dostopen razvijalcem z manj izkušnjami.**

Hipotezo lahko potrdim. V poglavju 4.1 sem ugotovila, da SvelteKit pri dokumentaciji zavzema bolj poenostavljen pristop, ki razvijalca postopoma vodi skozi proces razvoja. Učna krivulja je minimalna, saj uporablja standardne spletne tehnologije, medtem ko Next.js zahteva poznavanje Reacta, JSX in *hook*-ov. Dokumentacija SvelteKita je krajša in bolj osredotočena na bistvo. V poglavju 4.2 je zapisano, da je razvijalska izkušnja pri SvelteKitu bolj poenostavljena z manj standardne kode in intuitivno sintakso. Nasprotno pa Next.js zahteva solidno predznanje Reacta, kar predstavlja oviro za začetnike, če se z njim še niso spoznali.

**H3: Spletne aplikacije, razvite v SvelteKit, pogosto dosegajo boljše rezultate pri hitrosti delovanja in manjši obremenitvi sistema kot aplikacije v Next.js.**

Hipoteza je potrjena z omejitvami. Meritve v poglavju 5.1 kažejo, da ima SvelteKit manjšo velikost paketov in krajši čas do interaktivnosti v primerjavi z Next.js. V poglavju 5.2.2 je zapisano, da SvelteKit zaradi odsotnosti virtualnega DOM-a zmanjša porabo pomnilnika in obremenitev procesorja, kar je še posebej opazno pri zahtevnejših operacijah in na mobilnih napravah. Benchmarki potrjujejo višje število zahtevkov na sekundo pri SvelteKitu v primerjavi z Next.js, vendar študija (Zaidan & Suyatno, 2025) kaže, da je Next.js pri obremenitvenih testih z JMeter bolj konsistenten in stabilen pri večjem številu sočasnih uporabnikov, medtem ko SvelteKit izstopa pri manj kompleksnih scenarijih. Hipoteza torej drži za manjše in srednje kompleksne aplikacije in za obremenitev odjemalca, ne pa nujno za vse vrste aplikacij. Ker različni viri uporabljajo različne metodologije in metrike, rezultatov ni mogoče neposredno primerjati, zato hipoteze ne morem potrditi kot dokončne, lahko pa jo označim kot delno potrjeno oziroma potrjeno z omejitvami.

**H4: Ogrodje Next.js zagotavlja več vgrajenih varnostnih mehanizmov, ki pripomorejo k večji zaščiti spletnih aplikacij kot ogrodje SvelteKit.**

Hipotezo lahko delno potrdim. V poglavju 5.3.1 sem ugotovila, da Next.js ponuja ločevanje med strežniškimi in odjemalskimi komponentami, Data Access Layer, oznako server-only in CSRF zaščito za Server Actions. V poglavju 5.3.2 je zapisano, da SvelteKit vključuje CSRF zaščito, konfiguracijo CSP in varno upravljanje z okoljskimi spremenljivkami. Obe ogrodji ponujata primerljiv nabor vgrajenih varnostnih mehanizmov, vendar se razlikujeta v pristopu. Next.js ima več vgrajenih funkcionalnosti na ravni ogrodja, medtem ko SvelteKit pušča več konfiguracije razvijalcu. Hipoteza, da Next.js zagotavlja več vgrajenih mehanizmov je delno potrjena, vendar obe ogrodji ob pravilni uporabi omogočata visoko stopnjo varnosti.

## *7.3 Omejitve raziskave*

Pri interpretaciji ugotovitev je treba upoštevati določene omejitve raziskave:

- Primerjava temelji na sekundarnih virih, in sicer člankih, dokumentaciji in obstoječih študijah, ne pa na lastnih meritvah in eksperimentalnem razvoju. To pomeni, da so ugotovitve odvisne od kakovosti in objektivnosti uporabljenih virov.
- Meritve zmogljivosti iz različnih virov uporabljajo različne metodologije, testna okolja in metrike, zato rezultatov ni mogoče neposredno primerjati. Študije so pogoste izvedene na preprostih primerih, ki ne odražajo nujno kompleksnosti realnih produkcijskih sistemov.
- Ogrodji se hitro razvijata, zato so lahko nekateri podatki ob objavi te diplome že zastareli. Prav tako se lahko nove različice ogrodij pomembno razlikujejo od tistih, na katere se sklicujem tukaj.
- Analiza varnostnih mehanizmov temelji na uradni dokumentaciji in priporočilih, ne pa na praktičnem preverjanju ranljivosti. Varnost je odvisna tudi od pravilne implementacije s strani razvijalca, ne le od vgrajenih mehanizmov ogrodja.

## *7.4 Možnosti nadaljnjih raziskav*

Raziskavo bi bilo mogoče nadgraditi v več smereh. Prihodnje raziskave bi lahko vključevale razvoj enake testne aplikacije v obeh ogrodjih in izvedbo lastnih meritev zmogljivosti v nadzorovanem okolju. Tak pristop bi omogočil neposredno primerjavo obeh ogrodij ob enakih pogojih. To bi tudi pomagalo pri odpravi omejitve glede različnih metodologij, ki se pojavljajo pri obstoječih virih. Prav tako bi bila smiselna dolgoročna študija vzdrževanja aplikacij, razvitih v obeh ogrodjih. Zajemala bi dejavnike, kot so čas za implementacijo novih funkcionalnosti, pogostost napak in stroški vzdrževanja. Zanimivo bi bilo primerjavo razširiti na dodatna meta ogrodja, kot so Nuxt ali Astro, kar bi omogočilo širši vpogled v ekosistem sodobnih spletnih ogrodij. Na področju varnosti bi nadaljnje raziskave lahko vključevale praktično testiranje ranljivosti in analizo pogostosti varnostnih posodobitev. Prav tako bi bilo smiselno raziskati, kako izbira ogrodja vpliva na širše vidike razvoja, kot so stroški vzdrževanja, razpoložljivost razvijalcev na trgu dela in dolgoročna stabilnost projekta.

# 8 SKLEP

Namen diplomskega dela je bil izvesti primerjalno analizo spletnih ogrodij Next.js in SvelteKit ter na podlagi pregleda strokovne literature, uradne dokumentacije in obstoječih raziskav ugotoviti njune ključne značilnosti, prednosti, omejitve in področja uporabe. Raziskava je bila usmerjena v primerjavo arhitekture, razvijalske izkušnje, zmogljivosti, varnosti in praktične uporabnosti obeh ogrodij. Cilj diplomskega dela je bil oblikovati celovit pregled obeh tehnologij in na podlagi ugotovitev pripraviti priporočila za njuno uporabo pri različnih vrstah spletnih projektov.

Z analizo strokovne literature, uradne dokumentacije in rezultatov obstoječih raziskav so bili zastavljeni cilji diplomskega dela doseženi. Primerjava je omogočila sistematičen vpogled v razvoj sodobnih spletnih ogrodij in pokazala, kako se Next.js in SvelteKit razlikujeta v pristopu k razvoju spletnih aplikacij. Analiza je pokazala, da obe ogrodji sledita sodobnim smernicam razvoja spletnih tehnologij in ponujata učinkovite rešitve za izdelavo sodobnih spletnih aplikacij, vendar pri tem uporabljata različne arhitekture pristope in razvojne koncepte.

Raziskava je pokazala, da je ena izmed najpomembnejših razlik med obema ogrodjema njuna arhitektura. Next.js temelji na knjižnici React in razvijalcem omogoča uporabo naprednih načinov upodabljanja ter dostop do zelo razvitega ekosistema knjižnic, orodij in skupnosti. Zaradi velike razširjenosti predstavlja uveljavljeno rešitev za razvoj večjih in bolj kompleksnih spletnih aplikacij. SvelteKit pa uporablja drugačen pristop, saj komponente prevede v čisti JavaScript že med procesom gradnje aplikacije. Tak način razvoja zmanjšuje količino kode, ki se izvaja v brskalniku in omogoča preprostejšo strukturo aplikacije in manjšo obremenitev sistema.

Pomembne razlike so bile ugotovljene tudi pri razvijalski izkušnji. Analiza je pokazala, da je SvelteKit zaradi uporabe standardnih spletnih tehnologij, preglednejše sintakse in manjše količine potrebne kode primernejši za razvijalce z manj izkušnjami ali za ekipe, ki želijo hitro razviti sodobno spletno aplikacijo. Next.js zaradi tesne povezanosti z Reactom od razvijalcev zahteva več predznanja, hkrati pa omogoča dostop do obsežnega ekosistema, bogate dokumentacije in aktivne skupnosti, ki olajša reševanje razvojnih izzivov. Obe ogrodji torej ponujata kakovostno razvijalsko izkušnjo, vendar je ta prilagojena različnim potrebam in profilom uporabnikov.

Pri analizi zmogljivosti in varnosti se je pokazalo, da imata obe ogrodji svoje prednosti. SvelteKit dosega zelo dobre rezultate pri velikosti paketov, času do interaktivnosti in porabi sistemskih virov, medtem ko je Next.js bolj konsistenten pri večjih obremenitvah in večjem številu sočasnih uporabnikov. Podobno velja za področje varnosti. Obe ogrodji vključujeta sodobne varnostne mehanizme, vendar se razlikujeta predvsem v načinu implementacije. Next.js ponuja več vgrajenih funkcionalnosti na ravni ogrodja, SvelteKit pa več odločitev glede implementacije varnostnih mehanizmov prepušča razvijalcu. Raziskava je zato pokazala, da stopnja varnosti ni odvisna zgolj od vgrajenih mehanizmov ogrodja, temveč tudi od njihove pravilne implementacije s strani razvijalca.

Na podlagi vseh analiz je mogoče zaključiti, da med Next.js in SvelteKit ni mogoče določiti univerzalno boljšega spletnega ogrodja. Izbira je odvisna predvsem od zahtev projekta, velikosti razvojne ekipe, izkušenj razvijalcev in pričakovanih funkcionalnosti oziroma zmogljivosti aplikacije. Next.js predstavlja primerno izbiro predvsem za večje projekte z visokimi zahtevami glede optimizacije za iskalnike, dolgoročne vzdržljivosti in razširljivosti, medtem ko je SveleteKit zaradi svoje preprostosti, hitrosti razvoja in učinkovite porabe virov posebej primeren za manjše in srednje velike projekte ter razvoj prototipov in MVP-jev.

Diplomsko delo združuje ugotovitve različnih strokovnih virov in jih povezuje v celovit primerjalni pregled, ki predstavlja koristno izhodišče za študente, razvijalce in organizacije pri izbiri ustreznega spletnega ogrodja. Zaradi hitrega razvoja spletnih tehnologij se obe ogrodji nenehno nadgrajujeta, zato bodo nekatere ugotovitve v prihodnje zahtevale ponovno preverjanje in posodobitev. Kljub temu rezultati diplomskega dela predstavljajo aktualne pregled ključnih razlik med Next.js in SvelteKit ter poudarjajo dejavnike, ki jih je smiselno upoštevati pri izbiri tehnologije za razvoj sodobnih spletnih aplikacij, Predstavljena primerjalna analiza torej predstavlja uporabno podlago za nadaljnje raziskave in pomoč pri izbiri ustreznega spletnega ogrodja v praksi.

# 9 VIRI, LITERATURA


Ayebola, J. (1. Maj 2025). *How Incremental Static Regeneration (ISR) Works in Next.js*. Pridobljeno iz freecodecamp: https://www.freecodecamp.org/news/how-incremental-static-regeneration-isr-works-in-nextjs/

Bernstein, M. (2010). Origins of the JavaScript Ninja: John Resig. *XRDS: Crossroads, The ACM Magazine for Students, 17*(1), 46-47.

Bratslavsky, P. (2025). *SvelteKit Explained: Svelte's Full-Stack Framework*. Pridobljeno iz Strapi: https://strapi.io/blog/sveltekit-explained-full-stack-framework-guide

Brown, A. (8. April 2025). *Svelte vs Next.js: Understand the Differences*. Pridobljeno iz descope: https://www.descope.com/blog/post/nextjs-vs-reactjs-vs-sveltekit

Chidi. (2025). *Sveltekit vs. Next.js: A side-by-side comparison*. Pridobljeno iz Hygraph: https://hygraph.com/blog/sveltekit-vs-nextjs

conductor. (2025). *The Ultimate Guide to Core Web Vitals*. Pridobljeno iz conductor: https://www.conductor.com/academy/core-web-vitals/

developyn. (24. Julij 2024). *The history of Svelte*. Pridobljeno 2026 iz developyn: https://developyn.com/the-history-of-svelte/

DevMorph Team. (7. Februar 2026). *SvelteKit vs Next.js 16: 2026 Performance Benchmarks*. Pridobljeno iz DevMorph: https://www.devmorph.dev/blogs/sveltekit-vs-nextjs-16-performance-benchmarks-2026

Dutton, S. (brez datuma). *Use tools to measure performance*. Pridobljeno 2026 iz wed.dev: https://web.dev/articles/performance-audit-tools

Ekpobimi, H. O. (2024). Building high-performance web applications with Next.js. *Computer Science & IT Research Journal, 5*(8).

freeCodeCamp. (2017). *Angular*. Pridobljeno 2026 iz https://github.com/freeCodeCamp/freeCodeCamp/blob/50224b424324130cef6a1cba02db3234b6934251/guide/english/angular/index.md

Garrett, J. J. (2005). *Ajax: New Approach to Web Applications.* Pridobljeno iz https://designftw.mit.edu/lectures/apis/ajax_adaptive_path.pdf

GeeksForGeeks. (1. September 2025). *Next.js Data Fetching*. Pridobljeno iz GeeksForGeeks: https://www.geeksforgeeks.org/nextjs/next-js-data-fetching/

Jamstack. (brez datuma). *What is Jamstack?* Pridobljeno 2026 iz Jamstack: https://jamstack.org/what-is-jamstack/

Kazanivska, O., & Muzyka, R. (2024). *Overview of Next.js for Modern Web Apps: Pros, Cons, and Use Cases*. Pridobljeno iz Leobit: https://leobit.com/blog/overview-of-next-js-for-modern-web-apps-pros-cons-and-use-cases/

Lucky, O. (2023). *Next.js vs SvelteKit: Which One is Better for Your Next Strapi App?* Pridobljeno iz trapi: https://strapi.io/blog/nextjs-vs-sveltekit-which-one-is-better-for-your-next-strapi-app

Maurya, P., Keshari, P., Kumar, P., & Katiyar, D. (2023). Web Development Using ReactJS. *Journal of Current Research in Engineering and Science, 6*(2), Paper 58. Pridobljeno iz https://www.psvpec.in/jcres/2023_2/58.pdf

Mika, A., & Vasylenko, J. (2025). *What is Next.js?* Pridobljeno iz Ramotion: https://www.ramotion.com/blog/what-is-next-js/

Mozilla. (brez datuma). *Attacks*. Pridobljeno 2026 iz MDN Web Docs: https://developer.mozilla.org/en-US/docs/Web/Security/Attacks

Mozilla. (brez datuma). *Content delivery networks (CDN)*. Pridobljeno 2026 iz MDN Web Docs: https://developer.mozilla.org/en-US/docs/Glossary/CDN

Mozilla. (brez datuma). *Website security*. Pridobljeno 2026 iz MDN Web Docs: https://developer.mozilla.org/en-US/docs/Learn_web_development/Extensions/Server-side/First_steps/Website_security

Next.js. (brez datuma). *Next.js Documentation*. Pridobljeno 2026 iz Next.js: https://nextjs.org/docs

Ornbo, G. (2012). *Sams Teach Yourself Node.js in 24 Hours.* Sams Publishing.

Osmani, A., & Miller, J. (5. Januar 2026). *Rendering on the Web*. Pridobljeno 2026 iz wed.dev: https://web.dev/articles/rendering-on-the-web

OWASP. (2021). *AO6:2021 - Vulnerable and Outdated Components*. Pridobljeno 2026 iz OWASP Top 10: https://owasp.org/Top10/2021/A06_2021-Vulnerable_and_Outdated_Components/index.html

OWASP. (brez datuma). *HTTP Headers Cheat Sheet*. Pridobljeno 2026 iz OWASP Cheat Sheet Series: https://cheatsheetseries.owasp.org/cheatsheets/HTTP_Headers_Cheat_Sheet.html

OWASP. (brez datuma). *Secrets Management Cheat Sheet*. Pridobljeno 2026 iz OWASP Cheat Sheet Series: https://cheatsheetseries.owasp.org/cheatsheets/Secrets_Management_Cheat_Sheet.html

Pati, S., & Zaki, Y. (2025). *Evaluating the Efficacy of Next.js: A Comparative Analysis with React.js on Performance, SEO, and Global Network Equity*.

Pentest Wizard. (brez datuma). *Understanding the Risks of Storing Secrets on the Frontend Storage*. Pridobljeno 2026 iz Pentest Wizard: https://pentestwizard.com/the-risks-of-storing-secrets-on-the-frontend/#:~:text=Despite%20the%20convenience%20it%20may,can%20be%20exploited%20by%20cybercriminals.

Saboori, S. (3. September 2024). *Comprehensive Guide to Data Fetching in Next.js*. Pridobljeno iz Fishtank: https://www.getfishtank.com/insights/comprehensive-guide-to-data-fetching-in-nextjs

Sanchez, P. (15. Junij 2024). *Frontend SSR frameworks benchmarked: Angular, Nuxt, NextJs and SvelteKit*. Pridobljeno iz pausanchez: https://pausanchez.com/en/articles/frontend-ssr-frameworks-benchmarked-angular-nuxt-nextjs-and-sveltekit/

Stack Overflow. (2026). *Stack Overflow - Tags*. Pridobljeno iz Stack Overflow: https://stackoverflow.com/tags

State of JS. (2025). *State of JS Survey 2025 - Front-end Frameworks*. Pridobljeno iz stateofjs: https://2025.stateofjs.com/en-US/libraries/front-end-frameworks/

State of JS. (2025). *State of JS Survey 2025 - Meta-frameworks*. Pridobljeno 2026 iz stateofjs.

State of JS Survey. (2024). *State of JavaScript 2024*. Pridobljeno 2026 iz State of JS: https://2024.stateofjs.com/

Svelte. (14. December 2022). *Announcing SvelteKit 1.0*. Pridobljeno 2026 iz svelte: https://svelte.dev/blog/announcing-sveltekit-1.0

Svelte. (brez datuma). *sveltejs/kit*. Pridobljeno 2026 iz https://github.com/sveltejs/kit

SvelteKit. (2026). *SvelteKit Documentation*. Pridobljeno iz Svelte: https://svelte.dev/docs

UIC. (brez datuma). *The Transition from Client-Side to Server-Side Rendering*. Pridobljeno 2026 iz CS 484: Secure Web Application Development: https://www.cs.uic.edu/~ckanich/cs484/f24/readings/chapter-3-server-side-web-development/full-stack-react/index.html

Ulili, S. (2025). *SvelteKit vs Next.js*. Pridobljeno iz BetterStack: https://betterstack.com/community/guides/scaling-nodejs/sveltekit-vs-nextjs/#bundle-size-and-performance

Umeh, V. (30. Maj 2025). *Next.js Rendering Strategies and how they affect core web vitals*. Pridobljeno iz This Dot Labs: https://labs.thisdot.co/blog/next-js-rendering-strategies-and-how-they-affect-core-web-vitals

Vercel. (2016). *Next.js First Release*. Pridobljeno 2026 iz GitHub: https://github.com/vercel/next.js/releases/tag/1.0.0

Vercel. (brez datuma). *SvelteKit on Vercel*. Pridobljeno 2026 iz Vercel: https://vercel.com/docs/frameworks/full-stack/sveltekit

Vercel. (brez datuma). *vercel/next.js*. Pridobljeno 2026 iz https://github.com/vercel/next.js

Vue.js. (2014). *Vue.js 0.10 is here!* Pridobljeno 2026 iz https://v2.vuejs.org/2014/03/22/vuejs-010-release/

W3Techs. (2025). *Usage statistics of JavaScript for website*. Pridobljeno iz W3Techs.com: https://w3techs.com/technologies/details/cp-javascript

Wirfs-Brock, A., & Eich, B. (2020). JavaScript: The first 20 years. *Proceedings of the ACM on Programming Languages, 4*, 1-189. Pridobljeno iz https://doi.org/10.1145/3386327

Zaidan, A. J., & Suyatno, D. F. (2025). Rendering Performance Analysis of Astro JS, Next JS, Nuxt JS, and SvelteKitFrameworks Using Google Lighthouse, PageSpeed Insight, and JMeter. *Journal of Emerging Information System and Business Intelligence*. Pridobljeno iz https://ejournal.unesa.ac.id/index.php/JEISBI/article/view/64283/49659