Preizkušeni pristopi pri upodabljanju spletnih aplikacij na strežniku

Manica Abramenko, Žiga Lah, Nejc Hauptman, Jani Šumak

Inova IT d.o.o., Maribor, Slovenija
manica.abramenko@inova.si, nejc.hauptman@inova.si,
ziga.lah@inova.si, jani.sumak@inova.si

V prispevku predstavljamo, kakšne prednosti prinaša upodabljanje spletnih aplikacij na strežniku (ang. Server-Side Rendering, v nadaljevanju SSR) v nasprotju s prikazovanjem pri odjemalcu (ang. Client-Side Rendering, v nadaljevanju CSR). SSR je tehnika spletnega razvoja, pri kateri strežnik ustvari celotne strani, prilagojene za vsako zahtevo uporabnika, in jih dostavi brskalniku uporabnika. Ta pristop ponuja mnogo prednosti, vključno s hitrejšim začetnim nalaganjem strani, izboljšano optimizacijo za iskalnike in boljšo zmogljivostjo na napravah z manjšo procesorsko močjo. V uvodu bomo izpostavili, s kakšnimi težavami se srečujemo kot razvijalci spletnih aplikacij s prikazovanjem pri odjemalcu in kako bi se jih lahko lotili z uporabo spletnih aplikacij na strežniku. Podrobneje bomo predstavili, kaj je SSR, in pregledali ogrodja, ki jih lahko uporabimo za generiranje vsebine spletnih aplikacij na strežnikih, kot sta Remix in Next.js. Na praktičnem primeru bomo prikazali rezultate posameznih metrik, ki so relevantne pri prikazovanju, nalaganju in delovanju spletne aplikacije. Prispevek zaključimo z razmišljanjem o časovnih, prostorskih in omrežnih obremenitvah, ki jih prinese razvijanje dotičnih spletnih aplikacij.

Ključne besede:
spletni razvoj
SSR
SEO
JS ogrodja
React
Remix
Next.js

Univerzitetna založba Univerze v Mariboru
ISBN 978-961-286-894-9
DOI https://doi.org/10.18690/um.feri.4.2024.20

1 Uvod

V sodobnem razvoju spletnih aplikacij se soočamo z različnimi izzivi, ki vplivajo na uporabniško izkušnjo, zmogljivost aplikacij in njihovo optimizacijo za iskalnike. Ena izmed ključnih odločitev pri razvoju je izbira med upodabljanjem pri odjemalcu (ang. Client Side Rendering, v nadaljevanju CSR) in upodabljanjem na strežniku (ang. Server Side Rendering, v nadaljevanju SSR). Vsaka od teh tehnik ima svoje prednosti in slabosti, vendar je v zadnjem času SSR pridobil veliko pozornosti zaradi svojih številnih koristi. Tradicionalno se za upodabljanje na strežniku uporabljajo jeziki, kot so PHP, ASP.NET, Java in Ruby. V svetu JavaScript-a pa so priljubljena ogrodja, kot so Next.js, Remix, Astro, Gatsby, Nuxt in SvelteKit, ki razvijalcem omogočajo preprosto implementacijo SSR v njihovih spletnih aplikacijah.

V tem prispevku bomo raziskali tehniko SSR ter delovanje, njene prednosti in slabosti, načine implementacije in različna ogrodja, ki so na voljo za spletni razvoj z uporabo SSR, kot so Remix, Next.js in 'lastna implementacija'. Prav tako bomo predstavili primere uporabe in analizirali, kako lahko SSR izboljša učinkovitost in optimizacijo spletnih aplikacij.

2 Kaj je SSR?

SSR je tehnika spletnega razvoja, pri kateri strežnik obdela in upodobi HTML vsebino spletne strani, preden jo dostavi odjemalcu. To je glavna razlika v primerjavi s CSR, ki s pomočjo JavaScirpta po prenosu potrebnih sredstev izriše vsebino strani. V praksi to pomeni, da za upodabljanje vsebine pri CSR potrebujemo prenos datotek za stile in skripte ter samih podatkov, ki jih želimo prikazati, medtem ko SSR vrne HTML dokument z vsebino. Ker tudi spletni iskalniki dobijo dejansko HTML vsebino, lahko tako stran indeksirajo na višje mesto, kar izredno izboljša optimizacijo za brskalnike (ang. Search Engine Optimization, v nadaljevanju SEO).

SSR lahko implementiramo na dva načina: SSR ob poslani zahtevi (ang. SSR-at request time), ki generira HTML ob vsaki spremembi enotnega naslova vira (ang. Uniform Resource Locator, v nadaljevanju URL), ali pa SSR ob izgradnji. Pri slednji se HTML vsebina generira vnaprej in jo strežnik pošlje brskalniku ob zahtevi) [1], [2].

2.1. Delovanje SSR

Življenjski cikel SSR spletne aplikacije vsebuje korake, ki so porazdeljeni med strežnik in odjemalca. Najzahtevnejše operacije (pridobivanje in procesiranje podatkov) se dogajajo na strežniku, medtem ko za interaktivnost skrbi odjemalčev del na brskalniku. Glavna prednost tega pristopa je, da lahko vsebino pokažemo, preden stran postane interaktivna. Cikel je sestavljen iz naslednjih korakov (Slika 1):

- Začetna zahteva: Ko uporabnik prispe na spletno stran preko določenega URL-ja, se sproži zahtevek na strežnik z vsemi podatki, ki jih strežnik potrebuje (piškotki, glave zahtevka, parametri v URL-ju).
- Procesiranje in pridobivanje podatkov: Ob prejemu zahtevka na strežniku se izvedejo vse potrebne operacije. V začetku je poskrbljeno za avtorizacijo in avtentikacijo, s katero preverimo, ali ima uporabnik pravice, ki jih potrebuje za želeno vsebino, prav tako se preveri žeton seje, ki zagotavlja uporabnikovo sejo na brskalniku. Obenem se zabeležijo (ang. logging) vsi zahtevki za potrebe analitike in odpravljanje napak (ang. debugging). Glede na zahtevek iz brskalnika strežnik izvede pridobivanje podatkov iz različnih virov, ki jih potem lahko obdelamo, da jih je mogoče uporabiti pri prikazu strani uporabniku. V tem koraku lahko implementiramo predpomnjenje (ang. caching), ki zmanjšuje obremenitev strežnika. V tem koraku poskrbimo za lokalizacijo, vsa poslovna pravila in vsa ostala preoblikovanja, da ustrezajo uporabniku, ki je poslal zahtevek.

- Izris: Po prejemu vseh podatkov se s pomočjo različnih predlog (ang. template) ali ogrodij na strežniku sestavi celotna HTML struktura strani.
- Odziv (ang. response): Zraven vsebine pošiljamo še vrsto vsebine, politike predpomnjenja in druge metapodatke, ki jih potrebujejo moderni brskalniki. Da bi zmanjšali velikost poslanih datotek, uporabljamo tehnike stiskanja, ki jih omogočajo programi, kot sta Gzip in Brotli, kar izboljša delovanje strani v okoljih, ki uporabljajo slabšo internetno povezavo. Omrežja za dostavo vsebine (ang. Content Delivery Network, v nadaljevanju CDN) uporabljamo za predpomnjenje vsebin na različnih geografskih območjih, da zmanjšamo zamude pri prenosu podatkov.
- Hidracija (ang. hydration): Del, ki se začne dogajati na brskalniku takoj po prejemu HTML datotek s strani strežnika. Poleg same vsebine se začnejo prenašati vsi JavaScript ter CSS viri, ki jih potrebujemo za interaktivnost same strani. Ob prenašanju se vzpostavljajo poslušalci dogodkov (ang. event listeners), inicializacirana sta upravljanje stanja (ang. state management) in povezava z vsemi dinamičnimi elementi. Ko so vsi koraki končani, postane stran za uporabnika interaktivna. Za upravljanje navigacije in sprememb URL-jev uporabljamo mehanizme na strani odjemalca, da ne potrebujemo ob vsaki stvari novih zahtevkov za dodatne vire.

Slika 1: Prikaz delovanja SSR cikla.

SSR ponuja številne prednosti, vendar ima tudi določene pomanjkljivosti, ki jih morajo spletni razvijalci skrbno pretehtati, preden se odločijo za dotični pristop [1], [2].

2.2. Prednosti SSR

- Hitrejše začetno nalaganje strani: Začetni čas nalaganja je lahko hitrejši, saj strežnik pošlje popolnoma upodobljeno stran brskalniku, ki jo lahko takoj prikaže. SSR posodablja le dele HTML-ja, ki jih je treba posodobiti, zato ustvarja hitrejše prehode strani med stranmi in veliko hitrejše prvo barvanje vsebine (FCP).
- Izboljšan SEO: Indeksiranje spletnih mest, ki uporabljajo SSR, je za iskalnike veliko lažje kot indeksiranje spletnih mest, upodobljenih na strani odjemalca. Vsebina je upodobljena, preden se stran naloži, zato jim ni treba zagnati JavaScripta, da bi jo prebrali in indeksirali.
- Boljša performančnost na napravah z nižjo zmogljivostjo: Uporabniki s počasno internetno povezavo ali s starejšimi napravami lahko takoj vzpostavijo interakcijo s spletnimi stranmi. Upodabljanje vsebine poteka na strežniku, kar zmanjša obremenitev procesiranja na strani odjemalca.
- Konstanta dostava vsebine: Ker je vsebina generirana na centralnem strežniku, vsi odjemalec vidijo enako vsebino spletne strani.
- Dinamična dostava vsebine: SSR omogoča dinamično dostavo vsebin z upodabljanjem vsebine na strežniku in pošiljanjem v brskalnike uporabnikov. To zagotavlja hitrejši prikaz dinamične vsebine, kar pozitivno vpliva na angažiranost uporabnikov in uspešnost SEO [3], [4], [5].

2.3. Pomanjkljivosti SSR

- Povečana obremenitev strežnika: : SSR lahko dodatno obremeni strežnik, saj vsaka zahteva terja upodabljanje nove strani. To je mogoče ublažiti s strategijami predpomnjenja, vendar še vedno predstavlja potencialno težavo glede razširljivosti.
- Počasnejša interaktivnost: Medtem ko je začetno nalaganje strani hitrejše, se lahko naslednje interakcije počasnejše v primerjavi z upodabljanjem na strani odjemalca, saj lahko uporabnikove interakcije terjajo dodatne zahteve strežnika, ki mora ponovno dostaviti vsebino.
- Kompleksnost razvoja: Implementacija SSR je lahko bolj zapletena in zahteva dodatno orodje in konfiguracijo. Morda bodo potrebne tudi spremembe v pristopu razvijalcev, kar zadeva upravljanje stanja in usmerjanja. V primeru statičnih strani teh težav ni.
- Omejena interaktivnost: Zapletene interakcije na strani odjemalca bodo zahtevale dodaten trud za implementacijo, saj niso tako dinamične kot upodabljanje na strani odjemalca.
- Zakasnitev (ang. latency): Odvisno od lokacije strežnika lahko pride do večje zakasnitve, zlasti za uporabnike, ki so geografsko oddaljeni od strežnika.
- Težave z združljivostjo: Številne knjižnice in orodja tretjih oseb niso združljive z upodabljanjem na strani strežnika [3], [4], [5].

3 Analiza priljubljenih React rešitev

3.1. Remix

Remix je celovito spletno ogrodje (ang. framework), ki za svoje delovanje uporablja knjižnico React. Gre za odprtokodno ogrodje, razvito v skladu s sodobnimi spletnimi praksami in standardi. Omogoča izdelavo naprednih in učinkovitih spletnih vmesnikov, ki zadovoljujejo širok spekter uporabniških potreb. Poseben poudarek dajejo strežniškemu upodabljanju in predpomnjenju, kar bistveno pripomore k hitrejšemu nalaganju in izboljšani uporabniški izkušnji [6]. Z drugimi besedami, gre za strežniški in brskalniški izvajalec kode (ang. runtime), ki omogoča hitro nalaganje strani in instantne tranzicije z uporabo vgrajenih funkcionalnosti brskalnikov in porazdeljenih sistemov. Temelji na uporabi Fetch API-ja, zato ga lahko poganjamo praktično kjerkoli. Deluje na tradicionalnih Node.js okoljih in tudi na brezstrežniških [7].

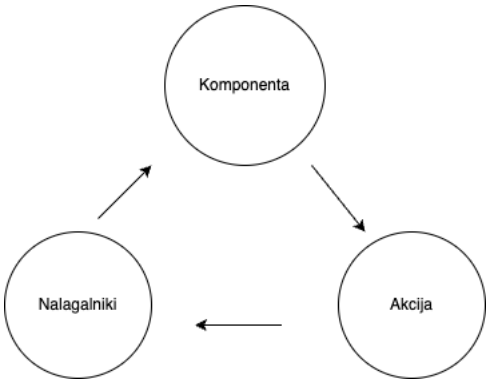
Ogrodje Remix se lahko predstavi kot prevajalnik (ang. compiler), upravljalec strežniških zahtev (ang. server-side HTTP handler), strežniško ogrodje (ang. server framework) ali kot ogrodje za odjemalca (ang. client framework) [7].

Remix deluje kot *prevajalnik* z uporabo Vita. Vite na strežniku ustvari HTTP upravljalca, ki skrbi za upodabljanje in izvajanje strežniških zahtev za podprte poti. Na odjemalcu ustvari gradnjo (ang. build), ki za avtomatsko deljenje zagotavlja kode glede na pot, uvoz slik in virov ter izvedbo drugih optimizacij, potrebnih za delovanje aplikacije v brskalniku. Prevajalnik generira manifest virov (ang. asset manifest), ki se uporabljajo pri uvodnem upodabljanju na strežniku (ang. Initial server render) ali pri prednalaganju (ang. prefetch) na strani odjemalca. Remix ni server, četudi se izvaja na strežniku, temveč gre za upravitelja HTTP zahtev. Ker temelji na uporabi JavaScripta, ga lahko namestimo praktično kamorkoli, kjer podpirajo JavaScript okolja. Zgrajen je bil okrog Fetch API-ja in za obdelovanje HTTP zahtev uporablja različne pakete, ki služijo kot adapterji. Eden takšnih je paket @remix-run/express. Kot večina *strežniških ogrodij* tudi Remix uporablja arhitekturni pristop MVC (Model-View-Controller). *Pogled* in *krmilnik* opravljata enako vlogo, kot to poteka znotraj drugih strežniških ogrodij, npr. Laravel, vendar razvijalcu ni potrebno pisati ločenih krmilnikov in pogledov, saj za abstrakcijo na podlagi procesiranja URL poti poskrbijo moduli Remix Route. Načine in pristope pri implementaciji modela ogrodje prepusti razvijalcu. Remix lahko uporabljamo kot strežniško ogrodje brez uporabe JavaScript-a v brskalniku. Na podlagi definiranih poti na strežniku naložimo podatke, jih spreminjamo z uporabo akcij in HTML obrazcev ter prikažemo spremembe na

uporabniškem vmesniku z zgolj osnovnim delovanjem brskalnikov. *Brskalniško ogrodje* deluje z uporabo hidracije, kar pomeni, da Remix hidrira stran z uporabo JavaScript modulov. Remix z vgrajenimi optimizacijami zazna, v katerem delu aplikacije so se zgodile spremembe, in na podlagi tega pošlje le spremembe za tiste dele aplikacije, ki so se spreminjali. To pomeni, da ni treba zahtevati celotnega dokumenta, kar bi povzročilo večjo porabo virov in slabšo uporabniško izkušnjo. Remix lahko prednaloži podatke in različne druge vire (CSS in JavaScript module) pred izvedbo uporabnikove akcije.

Ogrodje vsebuje že vgrajen sistem za usmerjanje (ang. routing), ki poleg podpore statičnih poti podpira tudi gnezdeno usmerjanje (ang. nested routes). Vgrajen je sistem za usmerjanje temelji na datotečnem sistemu, kar pomeni, da strani ustvarjamo s kreacijo map in podmap, pri čemer imena predstavljajo segmente poti. Modularnost, ki jo zagovarja pristop gnezdenega usmerjanja, zagotavlja, da je vsaka pot osredotočena zgolj na dodeljeni segment v URL-ju in pripadajoči del uporabniškega vmesnika [8]. Uporaba gnezdenega usmerjanja vpliva tudi na čas nalaganja. Remix paralelno nalaga vse potrebne vire, ko se URL ujema z več potmi. V nasprotju z enostranskimi aplikacijami, kjer imamo sekvenčno nalaganje virov, paralelno izvajanje bistveno izboljša perfomanco aplikacije.

Avtomatsko ohranjanje usklajenosti uporabniškega vmesnika s stanjem serverja predstavlja eno izmed jedrnih funkcionalnosti Remixa. Podatkovni tok je sestavljen iz treh gradnikov; komponente, akcije in nalagalnikov (ang. loaders) (Slika 2) [9].



Slika 2: Prikaz podatkovnega toka.

Znotraj usmerjevalne datoteke (ang. route file) definiramo *nalagalno funkcijo*, ki poskrbi za pridobitev podatkov in jih nato posreduje *komponenti*. Komponenta predstavlja del uporabniškega vmesnika in uporablja podatke, pridobljene iz nalagalne funkcije. Ko uporabnik znotraj forme napravi določeno spremembo in izvede akcijo (npr. stisne na gumb), se izvede akcija na pot, definirano znotraj atributa *action* (Slika 3).

```
routes/user-details.tsx

...

export async function loader({ request }: LoaderFunctionArgs ) {
   const { name, email } = await getUserDetails(request);

   return json({ name, email});
}


export default function UserDetails() {
   const { name, email } = useLoaderData<typeof loader>();

   return (
      <Form method="post" action="/user-details">
         <span>Uporabnik: ${name} </span>
         <input name="ime" defaultValue={name}  />
         <input name="email" defaultValue={email} />
         <button type="submit">Shrani</button>
      </Form>
    );
}


export async function action({ request }: ActionFunctionArgs) {
   const formData = await request.formData();
   const { id } = await getUserDetails(request);

   await updateUser(id, {
       email: formData.get("email"),
       displayName: formData.get("displayName"),
   });

   return json({ ok: true });
}
```

Slika 3: Odsek kode, ki prikazuje uporabniški vmesnik in funkcijo *action*.

Ena izmed temeljnih smernic spletnega ogrodja Remix je minimalna količina JavaScripta in progresivna izboljšava (ang. progressive enhancement). Ta pristop je usmerjen v zagotavljanje optimalne uporabniške izkušnje za vse uporabnike, ne glede na zmogljivost njihovih naprav, različico brskalnika ali hitrost internetne povezave [10]. Progresivna izboljšava pomeni, da aplikacija zagotovi osnovno funkcionalnost vsem uporabnikom, kar vključuje prikaz osnovne HTML strani, napolnjene z vsebino. Uporabnikom z dobro internetno povezavo in sodobnejšimi brskalniki pa omogoči naprednejšo izkušnjo z uporabo stilov, animacij in drugih naprednih funkcionalnosti Ta strategija pospešuje nalaganje spletne strani in olajša indeksiranje, saj se vsebina naloži takoj preko HTML izvorne kode, ne da bi bilo treba predhodno naložiti JavaScript [11].

V nasprotju s SPA (ang. Single Page Application) aplikacijami, kjer SPA ob zahtevi vrne prazen dokument in na to sinhrono nalaga vire ter izvaja upodabljanje (ang. rendering) ko se naloži JavaScript, aplikacije, napisane z ogrodjem Remix, ob prvi zahtevi paralelno izvedejo nalaganje potrebnih virov. To pomeni, da je čas do prvega prikaza (ang. time to first paint) izrazito krajši, saj Remix lahko zagotovi HTML, preden se JavaScript naloži [12].

3.2. Next.js

Next.js je eno izmed najbolj priljubljenih JavaScript ogrodij za razvoj celovitih spletnih rešitev. Enako kot Remix Next.js za razvoj spletnih aplikacij uporablja knjižnico React. Sovpada z arhitekturnim stilom Jamstack, ki predstavlja aplikacije, razvite s pomočjo programskega jezika JavaScript. Premika zadolžitve na stran odjemalca, ki na podlagi API (ang. Application Programming Interface) zahtev pridobivajo podatke in komunicirajo z različnimi servisi. Nato strani, zapisane z označevalskim jezikom (ang. Markup) ob gradnji, servira na mrežo za dostavo vsebine (ang. Content Delivery Network ali CDN) [13]. Next.js vsako stran predhodno tudi upodobi (ang. pre-render) in strežnik odjemalcu vrne zgolj potrebno HTML vsebino z minimalno količino JavaScripta, kar pripomore k boljši performanci in SEO (ang. Search engine optimization) analizi.

Glavne značilnosti ogrodja Next.js so upodabljanje na strežniku, samodejno deljenje kode, statično generiranje strani in razvoj javnega API-ja [14]. Next.js podpira več pristopov za serviranje vsebine:

- Statično generiranje (SSG - Static Site Generation): strani se generirajo ob času gradnje in se shranijo kot statične datoteke. Ta pristop se pogosto uporablja kadar so spremembe strani zelo redke.
- SSR: nalogo generiranja strani prevzame strežnik in se zgodi ob vsakem odjemalčevem zahtevku. Pristop omogoča tudi serviranje dinamične vsebine, vendar se lahko čas dostave podaljša.
- Inkrementalna statična regeneracija (ISR - Incremental Static Regeneration): gre za kombinacijo med pristopoma SSG in SSR, kjer se strani generirajo statično, vendar se lahko določeni deli vsebine občasno osvežijo na strežniku brez potrebe po ponovnem gradbenem postopku celotne strani.
- Renderiranje na strani odjemalca (CSR): vsebina se naloži in upodobi na strani odjemalca. Ta pristop se je uveljavil z vpeljavo enostranskih aplikacij.

Next.js ima 2 tipa usmerjanja - tradicionalni Pages in nov pristop, imenovan App Router. Pri uporabi Pages usmerjevalnika razvijalec z uporabo datotečnega sistema definira strani tako, da jih doda oz gnezdi kot datoteke pod mapo *pages*. Vsaka datoteka v tej mapi postane dostopna preko URL-ja. Na primer:

Na primer:

- *pages/index.tsx* postane domača stran *(/)*
- *pages/uporabniki.tsx* postane dostopna na */uporabniki*
- *pages/uporabniki/zahteve.tsx* postane dostopna na */uporabniki/zahteve*

V vsaki datoteki lahko z uporabo metode *getStaticPaths* definiramo katere dinamične poti naj se vnaprej generirajo. Z uporabo metod *getStaticProps* ali *getServerSideProps* pa lahko pridobimo določene podatke. Izvajata se na serverju.

3.2.1. Dinamično usmerjanje

Za dinamične poti Next.js uporablja oklepaje v imenih datotek. Npr: *pages/uporabniki/[id].tsx*, id je dinamičen šifrant, kar pomeni, da bo, ko uporabnik navigira na stran */uporabniki/1,* pridobil in prikazal podatke uporabnika z ID-jem ena (Slika 4) [15].

```
pages/uporabniki/[id].tsx

import { useRouter } from 'next/router'


export default function Uporabnik() {
   const router = useRouter()
   return <p>ID: {router.query.id}</p>
}
```

Slika 4: Odsek kode, ki prikazuje dinamično usmerjanje.

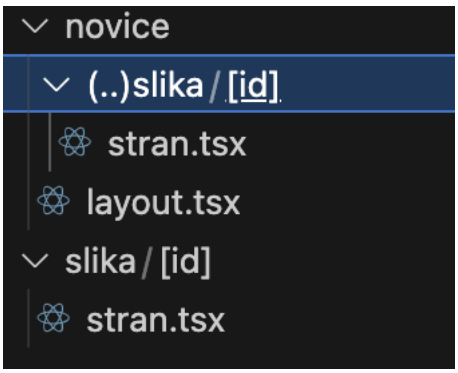
Nextov usmerjevalnik podpira lovljenje dinamičnih segmentov. Z uporabo lovljenja dinamičnih segmentov lahko ulovimo segmente neke URL poti. Datoteko, poimenovano */pages/uporabniki/**[...id]**,* pri čemer [...šifrant] predstavlja dinamične segmente, pove Nextu, da mora omogočiti lovljenje dinamičnih segmentov za ta celoten URL [16]. Z uporabo dvojnih zavitih *oklepajev (/pages/uporabniki/**[[...id]]**)* pa gre za opcijsko lovljenje dinamičnih segmentov.

Tabela 1: Uporaba dinamičnih segmentov.

Pot	URL	Parametri
pages/uporabniki/[...id].tsx	/uporabniki/1	{ slug: [‘1’] }
pages/uporabniki/[...id].tsx	/uporabniki/1/zahteve	{ slug: [‘1’, 'zahteve'] }
pages/uporabniki/[[...id]].tsx	/uporabniki/1/zahteve	{ slug: [] }

Next.js od različice 13 vpeljuje nov koncept usmerjanja imenovan *"App Router",* ki prinaša večjo fleksibilnost, zmogljivost in modularno izdelavo sodobnih spletnih aplikacij [16].

- Struktura in osnovno delovanje: Usmerjanje temelji na strukturi datotek znotraj mape *app*. Enako kot pri tradicionalnem pristopu, tudi tukaj struktura map določa URL-je. Na primer, datoteka *app/uporabniki/konfiguracija.tsx* postane dostopna na URL poti */uporabniki/konfiguracija*. Ta primer predstavlja gnezdeno usmerjanje, ki je prav tako podprto znotraj App Routerja ter temelji na hirarhiji komponent.
- Strežniške komponente: Omogoča uporabo strežniških komponent, ki se renderirajo na strežniku, kar omogoča boljšo zmogljivost in lažje upravljanje stanja. Nudi tudi kombiniranje strežniških in odjemalskih komponent.
- Izboljšano pridobivanje podatkov: Poenostavljeno pridobivanje podatkov z integracijo v strežniške komponente in podpora za pretakanje (ang. streaming), kar omogoča postopno renderiranje in hitrejšo dostavo vsebine.
- API Routes: API poti so definirane v mapi *app/api*, kar omogoča poenostavljeno organizacijo kode znotraj iste strukture map.
- Dodatne funkcionalnosti: Boljše upravljanje stanja in optimizacija zmogljivosti z uporabo strežniških komponent in podpora jedernih React komponent, kot je na primer Suspense.
- Vzporedne poti (ang. parallel routes): S pristopom vzporednih poti lahko naložimo več strani znotraj določenega dela aplikacije in je uporaben, kadar so ti deli dinamični. Za uporabo tega načina navigaranja moremo ime direktorija začeti z afno - *@datoteka*.
- Prestrezanje poti (ang. intercepting routes): Gre za pristop, ki podpira nalaganje poti iz drugih delov aplikacije [17]. Kadar uporabljamo ta pristop moramo imenik poimenovati na podlagi *oklepaj pika* konvencije, pri čemer vsaka dodatna pika znotraj oklepaja definira kolko segmentov želimo upoštevati (Slika 5).
    - (.), zadetki zgolj na istem nivoju
    - (..), zadetki na nivoju višje
    - (..)(..), zadetki dva nivoja višje
    - (...), vsi zadetki od “app”-a naprej



Slika 5: Poimenovanje imenika.

3.2.2. Predpomnjenje in optimizacije

Next.js vključuje več funkcij za optimizacijo in predpomnjenje, ki pomagajo pri izboljšanju zmogljivosti aplikacij, poskrbi pa tudi za granjenje in dostavo aplikacij z uporabo njihovega oblaka.

- Predpomnjenje strani: Statično generirane strani in API poti se predpomnijo za hitrejšo dostavo.
- Optimizacija slik: Vgrajena podpora za optimizacijo slik, kompresijo in prilagodljivost ločljivosti. Vključuje vgrajeno komponento, ki poskrbi za samodejno optimizacijo in predpomnjenje slik, kar izboljša zmogljivost nalaganja slik.
- Samodejno deljenje kode: Next.js samodejno deli kodo v manjše dele, ki se nalagajo po potrebi, kar zmanjšuje čas začetnega nalaganja strani.
- Prednalaganje: Next.js omogoča prefetching povezave, kar pomeni, da se povezane strani vnaprej naložijo, ko uporabnik pride v njihovo bližino, kar izboljšuje hitrost navigacije po spletni strani.
- Optimizacija razvijalske izkušnje s postavitvijo aplikacij na Vercelov oblak: Podpora za samostojno skaliranje aplikacij in optimizacijo dostave aplikacij, kadar je aplikacija postavljena na Vercelu.

3.3. React Core - Lastno ogrodje po meri

Danes na spletu lahko najdemo ogromno različnih ogrodij, ki jih lahko uporabimo za razvoj kompleksnih in preprostih aplikacij. Ogrodja poskrbijo, da se lahko razvijalec v večini primerov osredotoča zgolj na razvoj aplikacije, za vse optimizacije, mejne pogoje in podporo na različnih brskalnikih pa je zadolženo ogrodje. Najdejo se tudi primeri, ko zahtev ne morejo zadovoljiti že preizkušena ogrodja in je potrebo razviti ogrodje po meri. Z lastnim ogrodjem tako avtor sam definira arhitekturo, navigacijo, pristope in konvencije. Prav tako je fleksibilen pri izbiri programskega jezika, načina namestitve, uporabe knjižnic ter drugih orodij.

Kadar gradimo ogrodje po meri, moramo upoštevati imeti tri komponente:

- strežnik (Node.js)
- renderiranje na strežniku
- hidracija

Naloga strežnika je da poskrbi za podatke in generiranje HTML vsebine. Server z uporabo *renderToPipeableStream* funkcije iz paketa *react-dom/server* generira HTML vsebino v Node.js stream. *renderToPipeableStream* (Slika 6) kot

parameter funkcije sprejme React vozlišče (ang. React Node), v tem primeru gre za App komponento (Slika 7), ki je glavna komponenta aplikacije [18].

```
import { renderToPipeableStream } from 'react-dom/server';
…


app.use('/', (request, response) => {
 const { pipe } = renderToPipeableStream(<App />, {
   bootstrapScripts: ['/main.js'],
   onShellReady() {
     response.setHeader('content-type', 'text/html');
     pipe(response);
   }
 });
});
```

Slika 6: Prikaz renderToPipeableStream funkcije.

```
export const App = () => {
 return (
   <html>
     <head>
       <meta charSet="utf-8" />
       <meta name="viewport" content="width=device-width, initial-scale=1" />
       <link rel="stylesheet" href="/styles.css"></link>
       <title>OTS APP</title>
     </head>
     <body>
       <Router />
     </body>
   </html>
 );
}
```

Slika 7: Prikaz App komponente.

React bo sam poskrbel, da vstavi doctype HTML značko, ko servira vsebino odjemalcu.

Na strani odjemalca nato pokličemo *hydrateRoot* funkcijo (Slika 8), ki bo na prejet HTML dokument dodala dogodkovne poslušalce. To pripomore, da stran postane interaktivna.

```
import { hydrateRoot } from 'react-dom/client';
import { App } from './App;

hydrateRoot(document, <App />);
```

Slika 8: Funkcija hydrateRoot.

4 Primerjava

Primerjava ogrodij za strežniško upodabljanje spletnih ogrodij predstavlja dvojen izziv. Če bi merili zgolj hitrost strežnika, bi lahko merili odzivni čas, odzivni čas s serializacijo, odzivni čas s poizvedbo v bazi ipd. in pri tem upoštevali strojno opremo in operacijski sistem. Če bi, po drugi strani, merili samo čelni del, bi se osredotočili na hitrost upodabljanja v brskalniku - več o tem v nadaljevanju.

Ker prispevek govori o upodabljanju na strežniku, se nam je zdelo primerno primerjati oboje, tako hitrost zalednega sistema kot čelnega dela. Pri prvem bo večji poudarek seveda na hitrosti upodabljanja tistega deleža aplikacije, ki bi bil sicer upodobljen v odjemalcu, če bi razvijali SPA aplikacijo.

Ob zavedanju, da je tovrstnih primerjav veliko, so primerjave, predvsem tiste, ki se nanašajo na merjene čelnega in zalednega sistema, izvedene predvsem za demonstracijo. Izsledke je potrebno analizirati v luči razvijalske izkušnje.

4.1. Začetni projekt

Osnovne nastavitve in dobra praksa pri orodjih, potrebnih za generiranje in zagon projekta, postajajo del ogrodji, zato bomo uvodoma primerjali generatorje kode za Next.js, Remix in samostojno rešitev za React z SSR [19].

Prenovljena dokumentacija Reacta priporoča uporabo ogrodij Next.js in Remix. Razvijalci orodja so torej mnenja, da je za večino primerov bolje uporabiti vnaprej določene rešitve za pogoste težave. Po njihovem mnenju je pomembneje, da projekti, ne glede na čelno ogrodje, rešujejo podobne težave za uvajanje, kar je robustno in razširljivo [19].

Glede na povzeto nas torej ne preseneča, da imata omenjeni ogrodji vsako svoj program za generiranje začetnega projekta, in sicer create-next-app in create-remix [20].

Če želimo lastno rešitev za SSR, je na voljo nekaj generatorjev kode, toda odločitev je na strani razvijalca. RAzvali Reacta nas z opozorilo, da če "ima [naša] aplikacija nenavadne omejitve, ki jih [ta] ogrodja ne rešujejo [...]", uporabimo generator Vite ali Parcel [21].

Generirana koda se v omenjenih primerih ne razlikuje veliko, če seveda odmislimo posebnosti ogrodij in generatorjev kode. Z generatorji za lastno rešitev imamo pričakovano več dela, saj moramo poiskati primerno predlogo za SSR. V primeru uporabe orodja Vite, je potrebno izbrati možnost create-vite-extra in nato želeno predlogo za react-ssr [22]. Ker gre za rešitev po meri, je na razvijalcih, da se odločijo, ali želijo imeti predlogo z TypeScriptom in predlogo s pretakanjem, medtem ko je pri ogrodjih Remix in Next.js pretakanje vgrajeno v ogrodje [23], [24].

4.2. Uvajanje

Kar je bilo rečeno za generiranje začetnega projekta, je v veliki meri moč trditi tudi za uvajanje aplikacije na strežnik.

Generatorji kode za aplikacije, pripravljene z generatorji kode za Next.js oziroma Remix, in dokumentacija omenjenih ogrodji vključujejo navodila in rešitve za uvajanje na strežniku. Next.js celo ponujajo lastno rešitev za uvajanje [25].

Pri lastni rešitvi takšnih, vnaprej pripravljenih rešitev seveda ni, zato se od razvijalcev pričakuje, da sami vzpostavijo procese uvajanja in sami izberejo strežniško okolje.

4.3. Primerjava zalednega sistema

Pri primerjavi zalednih sistemov se bomo osredotočili na hitrost upodabljana strani. Seveda bi lahko merili tudi čas obdelave zahtevkov, čas odziva, odzivnost strežnika ob visokem prometu, porabo strojnih virov ipd., vendar so avtorji mnenja, da bi takšne primerjave zastrle jedro prispevka in nehote sprožile razprave o optimizacijah.

Za potrebe pričujoče primerjave bomo torej vzeli vnaprej pripravljene rešitve brez optimizacij. Vprašanje, ki si ga bomo zastavili, pa bo, kako hitro lahko ogrodja upodobijo in pošljejo daljši članek. Za merjenje hitrosti bomo uporabili orodje Jmeter [26].

Koda za dotično primerjavo in primerjavo čelnega dela je prosto dostopna na Github [27].

Tabela 2: JMeter rezultati zalednih sistemov (10 niti)

Ogrodje	Povprečni odzivno čas	Najhitrejši odzvini čas	Najpočasnejši odzivni čas	KB/s
React SSR	14	9	128	0.01
Next.js	29	9	128	0.10
Remix	17	9	128	0.08

4.4. Primerjava čelnega sistema

Za primerjavo čelnih sistemov se vse bolj uveljavlja Web Vitals. Web Vitals je Googlova iniciativa za merjenje kakovosti spletne uporabniške izkušnje [28].

Jedrne metrike so:

- Izris največjega vsebinskega dela (LCP) meri hitrost nalaganja strani, ko se stran prvič zažene. Za dobro uporabniško izkušnjo naj bo vrednost LCP 2.5 sekund ali manj. Meritev se nanaša na izris največjega elementa v vidnem področju uporabniškega očesa [29].
- Interakcija do naslednjega izrisa (INP) meri interaktivnost. Za dobro uporabniško izkušnjo naj bo vrednost INP 200 milisekund ali manj. Metrika meri zamik med uporabniško akcijo in odzivom strani [30].
- Celoten zamik postavitve (CLS) meri stabilnost strani. Za dobro uporabniško izkušnjo naj bo vrednost

Tabela 3: Core web vitals

Ogrodje	LCP	INP	CLS
Reactt SSR	0.4 s	0.48 s	0.001
Next.js	0.5 s	0.16 s	0.001
Remix	0.5 s	0.16 s	0.001

5 Razvijalska izkušnja

Čeprav se zavedamo, da je težko govoriti o razvijalski izkušnji uporabe in dela z zgoraj naštetih ogrodji, je ta pomemben dejavnik razvoja [31]. Ker bi celovita primerjava razvijalske izkušnje terjala daljši samostojni prispevek, se bomo omejili na pregled izsledkov dveh večjih spletnih raziskav, State of JavaScript in Stack Overflow developer survey.

V nobeni od navedenih raziskav sicer ne bomo zasledili vprašanj, ki bi se nanašala na uporabo lastnih rešitev za upodabljanje na strežniku, zato se bomo osredotočili samo na uporabi Reacta brez ogrodja in ogrodij Next.js in Remix. Prav tako se bomo pri analizi anket omejili na ugotovitve izvajalcev anket, saj je poglobljena analiza izven dometa tega prispevka.

Ob upoštevanju dejstva, da je bilo ogrodje Remix objavljeno 1. novembra 2020, se bomo v okviru tega prispevka osredotočili na raziskave, izvedene od leta 2022.

5.1. Stack Overflow developer survey

Stack Overflow je spletna stran z vprašanji in odgovori za programerje. Ponuja vprašanja in odgovore na določene teme s področja računalniškega programiranja. Ustvarjena je bila leta 2008 kot alternativa konkurenčni Expert exchange [so] in je eden najbolj priljubljenih virov spletnega iskanja za programerje [32].

Stack Overflow developer survey je vsakoletna spletna raziskava, ki poteka od leta 2010 in se je letno udeleži do 90 tisoč razvijalcev z vsega sveta. Gre za splošno raziskavo o uporabi programskih jezikov, ogrodij in podatkovnih baz. Rezultati in analize raziskav so objavljeni na njihovih spletnih straneh [33].

Raziskava je razdeljena na več sekciji, od splošnih vprašanj o anketirancih do povratnih vprašanj o sami raziskavi. Vprašanja, ki se nanašajo na uporabo tehnologij, so v istoimenski sekciji, ki je razdeljena na podsekcije Najbolj priljubljene tehnologije, Zadovoljstvo in želje, Uporabljene ali bodo uporabljene in Najbolje plačane.

Zaradi manjšega deleža uporabnikov orodja Remix bomo preskočili tretje vprašanje, ki ga je tudi sicer teže prikazati v tabelarni obliki, saj išče povezave med uporabljenimi tehnologijami in tehnologijami, ki jih anketiranci želijo uporabljati v prihodnje. Ker je teh povezav veliko, tabelarni prikaz odgovorov ni najbolj pregleden.

Ravno tako ne bomo uporabili analiz vprašanja glede najbolje plačanih tehnologij, ker vprašanje ni razdeljeno na ogrodja.

Leta 2024 je na vprašanje o spletnih tehnologijah v sekciji Najbolj priljubljene tehnologije odgovorilo 48.503 udeležencev.

Tabela 4: Najbolj priljubljene tehnologije 2024

Ogrodje	Vsi udeleženci	Zaposleni	Izobražujejo	Ostali
React	39.5%	41.6%	36.6%	27.7%
Next.js	17.9%	18.6%	17.9%	12.7%
Remix	1.6%	1.6%	1,3%	1.5%

V sekciji Spoštovane in zaželene je 47.707 anketirancev odgovorilo na vprašanje, s katerimi tehnologijami so bili zadovoljni, katere tehnologije se jim zdijo zanimive oziroma bi jih ponovno uporabili.

Tabela 5: Uporabljene tehnologije 2024

Ogrodje	Zadovoljstvo	Želja
React	62.2%	33.4%
Next.js	59.5%	18.2%
Remix	56.7%	2.8%

Leta 2023 je na vprašanje o spletnih tehnologijah v sekciji Najbolj priljubljene tehnologije odgovorilo 71.802 udeležencev.

Tabela 6: Najbolj priljubljene tehnologije 2023

Ogrodje	Vsi udeleženci	Zaposleni	Izobražujejo	Ostali
React	40.58%	42.87%	36.6%	30.13%
Next.js	16.67%	17.3%	15.12%	13.64%
Remix	1.27%	1.37%	0.76%	1.03%

V sekciji Spoštovane in zaželene je 70.637 anketirancev odgovorilo na vprašanje s katerimi tehnologijami do bili zadovoljni, katere tehnologije se jim zdijo zanimive oziroma bi jih ponovno uporabili.

Tabela 7: Uporabljene tehnologije 2023

Ogrodje	Zadovoljstvo	Želja
React	63.6 %	35.2 %
Next.js	65.9 %	20.3 %
Remix	57 %	3.1 %

Leta 2022 je na vprašanje o spletnih tehnologijah v sekciji najbolj priljubljene tehnologije odgovorilo 58.743 udeležencev.

Tabela 8: Najbolj priljubljene tehnologije 2022

Ogrodje	Vsi udeleženci	Zaposleni	Izobražujejo
React	42.62 %	44.31 %	42.81 %
Next.js	13.52 %	13.93 %	12.32 %
Remix	/	/	/

V sekciji *Spoštovane in zaželene* je 57.654 anketirancev odgovorilo na vprašanje, s katerimi tehnologijami radi delajo, katerim se poskušajo izogniti in s katerimi si želijo delati.

Tabela 9: Uporabljene tehnologije 2022

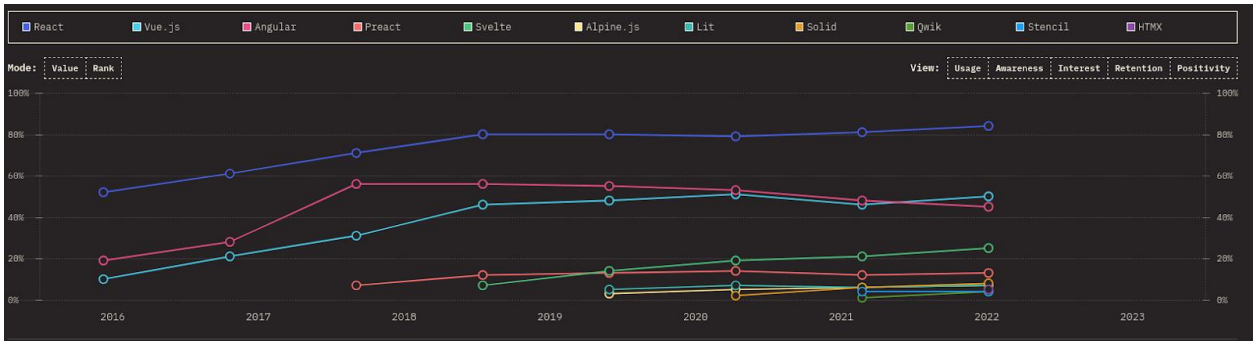
Ogrodje	Zadovoljstvo	Izogibanje	Želja
React	68.19 %	31.81 %	22.54 %
Next.js	69.23 %	30.77 %	11.28 %
Remix	/	/	/

5.2. Stanje JavaScripta (ang. State of JavaScript)

V primerjavi z zgornjo raziskavo je State of JavaScript mlajša spletna raziskava, ki se osredotoča na uporabo programskega jezika JavaScript. Kot prejšnja je tudi State of JavaScript vsakoletna spletna raziskava, ki se je udeleži okoli 30 tisoč razvijalcev z vsega sveta. Raziskava poteka vse od leta 2016, njeni izsledki pa so objavljeni na spletnih straneh [34].

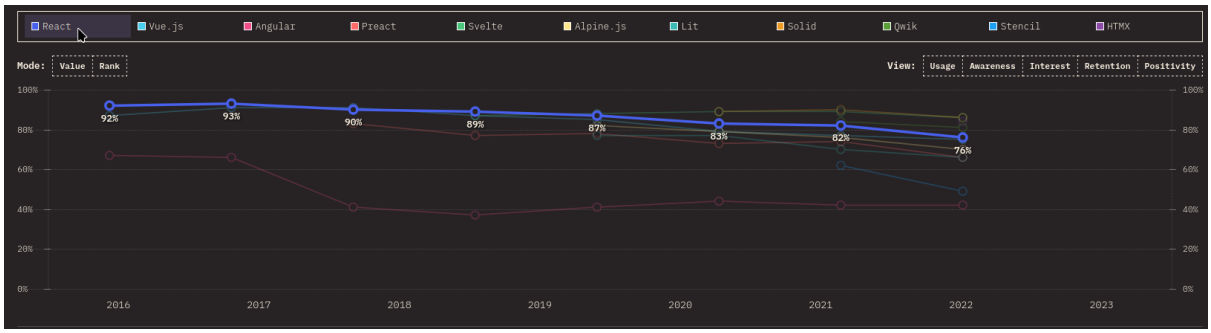
Za razliko od prejšnje ankete State of JavaScript strožje ločuje med ogrodji (čelna ali zaledna ogrodja) in knjižnicami. Tudi demografika se razlikuje, vendar bi analiza presegla okvire tega prispevka.

Iz odgovorov je moč razbrati, da je React trdno zasidran med najbolj priljubljenimi ogrodji, saj mu priljubljenost ne upada (Slika 9).



Slika 9: Uporaba čelnih ogrodji

In čeprav zanimanje za obstoječa orodja in orodja med razvijalci pregovorno upada, ostaja React med najbolj priljubljenimi ogrodji, ki ga razvijalci želijo uporabljati tudi v bodoče (Slika 10).



Slika 10: Priljubljenost členih ogrodji - React

Next.js in Remix sta med razvijalci sicer poznana, ampak v pričujočih anketah ne zasedata vidnejših mest in sta omenjena samo med ostalimi ogrodji za čelne sisteme.

6 Zaključek

Prispevek je primerjal pristope strežniškega upodabljanja za ogrodje React. Avtorji so izpostavili prednosti in slabosti strežniškega upodabljanja, kjer so kot največjo prednost izpostavili izboljšan SEO, ki prinese kompleksnost razvoja, saj je implementacija bolj zapletena. Opisni primerjavi je sledila primerjava generiranja, uvajanja in odzivnosti osnovnih različic, kjer ugotovijo, da obstoječa orodja (Next.js in Remix) ponujajo funkcionalnosti, ki zadostujejo potrebam večine razvijalcev. Na koncu prispevka so avtorji dodali izsledke dveh vidnejših razvijalskih spletnih anket o razvijalskih izkušnjah. V primerjavah so avtorji potrdili priporočila razvijalcev ogrodja React, da je za "rešitev po meri" potrebno več razvijalskega časa in da vsaj v osnovi ni prednosti v primerjavi z ogrodji, kot sta Next.js in Remix. Tudi povzetki iz raziskav, predvsem Stack Overflow developer survey, nakazujejo, da razvijalci vse bolj posegajo po omejenih ogrodjih, medtem ko interes za samostojne rešitve rahlo upada. Slednje namiguje, da so ogrodja, zgrajena na Reactu, dosegla določeno mero stabilnosti in da razvijalcev ne omejujejo pri razvoju spletnih aplikacij. Avtorji ne zanikajo upravičenosti in prednosti razvoja rešitev po meri, vendar dodajo, naj pri odločanju za implementacijo uvajanja na strežniku pristavimo še prednost razvijalske izkušnje, ki kaže v prid ogrodjem in ne rešitvi po meri.

Literatura

[1] https://www.heavy.ai/technical-glossary/server-side-rendering, Server-Side Rendering Definition, obiskano 14. 7. 2024.

[2] https://medium.com/@prashantramnyc/server-side-rendering-ssr-vs-client-side-renderin g-csr-vs-pre-rendering-using-static-site-89f2d05182ef, Server Side Rendering (SSR) vs. Client Side Rendering (CSR) vs. Pre-Rendering using Static Site Generators (SSG) and client-side hydration, obiskano 14. 7. 2024.

[3] https://www.searchenginejournal.com/server-side-rendering/481581/, Server-Side Rendering: The Pros & Cons To Consider For SEO, obiskano 15. 7. 2024.

[4] https://solutionshub.epam.com/blog/post/what-is-server-side-rendering, What is server-side rendering: definition, benefits and risks, obiskano 15. 7. 2024.

[5] https://prismic.io/blog/what-is-ssr, What is Server-side Rendering (SSR)?, obiskano 15. 7. 2024.

[6] https://remix.run/docs/en/main/discussion/introduction, Introduction, technical explanation, obiskano 16. 7. 2024.

[7] https://remix.run/docs/en/main/discussion/introduction, Remix, obiskano 16. 7. 2024.

[8] https://remix.run/docs/en/main/discussion/routes, React Router, obiskano 16. 7. 2024.

[9] https://remix.run/docs/en/main/discussion/data-flow, Fullstack Data Flow, obiskano 16. 7. 2024.

[10] https://developer.mozilla.org/en-US/docs/Glossary/Progressive_Enhancement, Progressive enhancement, obiskano 16. 7. 2024.

[11] https://en.wikipedia.org/wiki/Progressive_enhancement, Progressive enhancement, obiskano 16. 7. 2024.

[12] https://remix.run/docs/en/main/discussion/progressive-enhancement, Progressive enhancement, obiskano 16. 7. 2024.

[13] https://umbraco.com/knowledge-base/jamstack/#:~:text=Jamstack%20is%20a%20term%20that,%2C%20and%20Markup%20(JAM), What is Jamstack?, obiskano 16. 7. 2024.

[14] https://nextjs.org/docs, Introduction, obiskano 16. 7. 2024.

[15] https://nextjs.org/docs/pages/building-your-application/routing/dynamic-routes, Dynamic Routes, obiskano 16. 7. 2024.

[16] [https://nextjs.org/docs/app/building-your-application/routing, Routing Fundamentals, obiskano 16. 7. 2024.

[17] https://nextjs.org/docs/app/building-your-application/routing/intercepting-routes, Intercepting Routes, obiskano 16. 7. 2024.

[18] https://react.dev/reference/react-dom/server/renderToPipeableStream, renderToPipeableStream, obiskano 16. 7. 2024.

[19] https://react.dev/learn/start-a-new-react-project#production-grade-react-frameworks, Production-grade React frameworks, obiskano 29. 7. 2024.

[20] https://nextjs.org/docs/getting-started/installation#automatic-installation, Automatic Installation, obiskano 29. 7. 2024.

[21] https://remix.run/docs/en/main/start/quickstart#installation, Installation, obiskano 29. 7. 2024.

[22] https://react.dev/learn/start-a-new-react-project#can-i-use-react-without-a-frameworkm, Start a New React Project, obiskano 29. 7. 2024.

[23] https://remix.run/docs/en/main/guides/streaming, Streaming, obiskano 29. 7. 2024.

[24] https://nextjs.org/docs/app/building-your-application/rendering/server-components#streaming, Streaming, obiskano 29. 7. 2024.

[25] https://vercel.com/docs/deployments/overview, Deploying to Vercel, obiskano 29. 7. 2024.

[26] https://jmeter.apache.org/, Apache JMeter, obiskano 29. 7. 2024.

[27] https://github.com/janiSumak/ots-2024, OTS 2024, obiskano 5. 8. 2024.

[28] https://web.dev/articles/vitals, Web vitals, obiskano 29. 7. 2024.

[29] https://web.dev/articles/lcp, Largest Contentful Paint (LCP), obiskano 29. 7. 2024.

[30] https://web.dev/articles/inp, Interaction to Next Paint (INP), obiskano 29. 7. 2024.

[31] https://www.gartner.com/en/software-engineering/topics/developer-experience, Developer Experience as a Key Driver of Productivity, obiskano 29. 7. 2024.

[32] N. Rao, C. Bansal, T. Zimmermann, A. H. Awadallah and N. Nagappan, "Analyzing Web Search Behavior for Software Engineering Tasks," 2020 IEEE International Conference on Big Data (Big Data), Atlanta, GA, USA, 2020, str.. 768-777.

[33] https://survey.stackoverflow.co/, Stack Overflow Annual Developer Survey, obiskano 29. 7. 2024.

[34] https://stateofjs.com/en-US, State of JavaScript, obiskano 29. 7. 2024.