Remix: zmogljivo meta-ogrodje za razvoj spletnih aplikacij

Gregor Jošt,[1] Miha Kočevar[2]

[1] 3fs Nordic SI, d.o.o., Maribor, Slovenija
gregor.jost@3fs.si

[2] AGILEDROP d.o.o., Celje, Slovenija
miha.kocevar@agiledrop.si

V hitro spreminjajočem se svetu spletnega razvoja imajo ogrodja pomembno vlogo pri poenostavljanju in pospeševanju razvoja zmogljivih in uporabniku prijaznih spletnih aplikacij. Med številnimi ogrodji za razvoj odjemalskega dela spletnih aplikacij se vse bolj uveljavlja Remix, zmogljivo meta-ogrodje, ki temelji na knjižnici React. Medtem ko se knjižnica React odlično obnese pri prikazu elementov uporabniškega vmesnika in njihovih posodobitev, Remix kot ogrodje ponuja celovit pristop razvoja spletnih aplikacij. To vključuje usmerjanje (angl. routing), pridobivanje podatkov, optimizacijo delovanja in boljšo izkušnjo za razvijalce. Med pomembnejše lastnosti ogrodja Remix štejemo generiranje spletne strani na strežniku (angl. server-side rendering), kar izboljšuje učinkovitost, omogoča optimizacijo SEO (angl. search engine optimization) in zagotavlja hitro in učinkovito nalaganje začetne strani. V prispevku se bomo osredotočili na več vidikov ogrodja Remix. Najprej bomo podrobneje predstavili samo ogrodje, raziskali njegovo ozadje in poudarili glavne lastnosti. Poleg tega se bomo posvetili tudi primerjavi ogrodja Remix z nekaterimi obstoječimi meta-ogrodji, narejenimi na podlagi knjižnice React. Sledila bo implementacija majhne aplikacije, ki bo omogočila praktičen prikaz določenih prednosti, ki jih ogrodje prinaša. Skozi ta proces bomo raziskali, kako ogrodje omogoča dinamičnost, pridobivanje podatkov z uporabo gnezdenih poti in kako v splošnem izboljša delovanje spletnih aplikacij.

Ključne besede:
Remix
React
meta-ogrodje
uporabniški vmesniki
spletne aplikacije

Univerzitetna založba Univerze v Mariboru
ISBN 978-961-286-772-0
DOI https://doi.org/10.18690/um.feri.8.2023.8

1 Uvod

Pri razvoju spletnih aplikacij imajo ogrodja in knjižnice pomembno vlogo, saj zagotavljajo razvijalcem že vnaprej pripravljena orodja in abstrakcije, ki poenostavljajo postopek razvoja. S tega vidika lahko ogrodje definiramo kot celovit nabor orodij, smernic in konvencij, ki omogočajo strukturiran pristop k razvoju aplikacij, medtem ko so knjižnice zbirke modulov kode, ki jih razvijalci selektivno uporabljajo za lažji razvoj. Knjižnice so torej osredotočene na zagotavljanje specifičnih funkcionalnosti, ogrodja pa ponujajo širšo strukturo in bolj strog pristop k razvoju [1].

Ogrodja za razvoj spletnih aplikacij razvijalcem ponujajo zbirko ponovno uporabljivih komponent, orodij in funkcionalnosti, ki poenostavljajo pogoste naloge in zagotavljajo dosledno strukturo za gradnjo aplikacij. Nekatera najbolj znana in pogosto uporabljena ogrodja za razvoj spletnih aplikacij vključujejo Angular, Vue.js ali Svelte [2]. Ponujajo funkcionalnosti, kot so arhitektura na osnovi komponent, vezava podatkov (angl. data binding), upravljanje stanja in učinkovito izrisovanje (angl. rendering), kar omogoča razvijalcem gradnjo razširljivih in odzivnih spletnih aplikacij.

Po drugi strani pa je knjižnica React v zadnjih letih postala eno izmed najbolj priljubljenih orodij za razvoj spletnih vmesnikov [2], saj zagotavlja deklarativen, komponentno-orientiran pristopu k razvoj. Kljub temu pa se z naraščajočo kompleksnostjo spletnih aplikacij pojavlja potreba po dodatnih orodjih in konvencijah. To je privedlo do pojava meta-ogrodij za React, ki temeljijo na omenjeni knjižnici in zagotavljajo celovito izkušnjo razvoja. V tem kontekstu so meta-ogrodja ogrodja, ki združujejo obstoječa ogrodja ali knjižnice z dodatnimi funkcionalnostmi in orodji ter ponujajo celovito rešitev za razvoj spletnih aplikacij. Pogosto vključujejo funkcionalnosti, kot so usmerjanje, izvajanje na strežniku (SSR), upravljanje stanja in optimizacijo izgradnje [3]. Ta meta-ogrodja, kot je na primer Next.js, so postala priljubljena zaradi svoje sposobnosti poenostavljanja in izboljšanja učinkovitosti razvoja.

V tem prispevku se bomo osredotočili na (meta-)ogrodje Remix, ki je vse bolj priljubljeno ogrodje za razvoj spletnih aplikacij. Ogrodje Remix temelji na knjižnici React in zagotavlja celovit pristop k razvoju spletnih aplikacij, vključno z usmerjanjem, pridobivanjem podatkov, optimizacijo delovanja in izboljšano izkušnjo za razvijalce. S poudarkom na generiranju spletne strani na strežniku (SSR) Remix izboljšuje učinkovitost, omogoča optimizacijo SEO in zagotavlja hitro in učinkovito nalaganje začetne strani [4].

2 Ogrodje Remix

2.1 Pregled glavnih lastnosti in funkcionalnosti

Ogrodje Remix temelji na filozofiji *postopnega izboljšanja* (angl. Progressive Enhancement) [4]. V osnovi postopno izboljšanje pomeni, da zagotavljamo osnovno vsebino in funkcionalnosti za čim več uporabnikov, hkrati pa omogočamo najboljšo možno izkušnjo uporabnikom novejših brskalnikov, ki lahko izvajajo vse sodobne funkcionalnosti spletnih aplikacij. Podrobneje, pristop se osredotoča na gradnjo spletne strani ali aplikacije tako, da najprej zagotovi osnovno funkcionalnost in vsebino, ki deluje na vseh vrstah brskalnikov in naprav, vključno s starejšimi brskalniki ali tistimi z omejenimi zmogljivostmi. Ta osnovna različica mora biti uporabna in zagotoviti osnovne informacije in funkcionalnosti. Nato se postopno izboljšuje izkušnja tako, da se uporabnikom z novejšimi in zmogljivejšimi brskalniki ponudi izboljšan uporabniški vmesnik, animacije, interaktivne elemente in druge dodatne napredne funkcionalnosti [5]. V skladu s tem Remix omogoča razvijalcem, da začnejo zgraditi preprost HTML na strežniku in nato postopoma nadgrajujejo funkcionalnost z odjemalskim JavaScriptom in interaktivnimi elementi. To omogoča hitro začetno fazo razvoja in nato postopno dodajanje naprednih funkcionalnosti, ko jih dejansko potrebujemo.

Remix prav tako podpira koncept *predpomnjenja*. S pomočjo te funkcionalnosti se vse potrebne datoteke, kot so moduli JavaScript in CSS, predpomnijo v brskalniku, kar omogoča, da se strani hitreje naložijo, zlasti v primeru počasnejšega omrežja. To izboljšuje uporabniško izkušnjo in zmanjšuje čas nalaganja strani.

Ker Remix temelji na knjižnici *React Router*, vključuje prilagodljiv in zmogljiv sistem za *usmerjanje* (angl. routing), ki zagotavlja gradnjo kompleksnih spletnih aplikacij z več (pod)stranmi. To omogoča, da se aplikacija razdeli na več delov, pri čemer ima vsaka svojo URL pot in ločeno logiko. S tem se zagotovi boljša organizacija aplikacije in lažje upravljanje navigacije med stranmi. Takšna optimizacija navigacije na strani odjemalca omogoča, da ogrodje ugotovi, kateri deli strani se oz. se ne spreminjajo med prehodi med URL-ji. Posledično aplikacija pridobiva podatke samo za tiste poglede, ki vključujejo spremembe, kar zmanjšuje nepotrebne zahtevke in izboljšuje odzivnost aplikacije. Takšne *gnezdene poti* (angl. nested routes) olajšajo gradnjo kompleksnih spletnih aplikacij, ki vsebujejo več nivojev poti.

Remix vsebuje tudi vse potrebno za enostavno *pridobivanje podatkov* iz oddaljenih virov. Z vgrajeno podporo za pridobivanje podatkov lahko spletno aplikacijo opremimo z vsemi potrebnimi informacijami za prikazovanje vsebine. To velja tudi za upravljanje z obrazci (angl. forms) spletne strani. Remix v ta namen zagotavlja dve funkciji, prejemnik (loader) in akcijo (action), ki omogočata operacije na strežniški strani z enostavnim dostopom do podatkov. Posledično ni potrebe, da bi na strani odjemalca imeli posebej logiko JavaScript za upravljanje z obrazci. Za razliko od drugih ogrodij, kjer bi morali vključiti JavaScript za izvedbo klicev preko fetch ali Axios, v Remixu to ni potrebno, saj delo z obrazci temelji na njihovem osnovnem delovanju.

Ker gre za ogrodje, ki temelji na knjižnici React, vključuje tudi funkcionalnosti, potrebne za preprosto *upravljanje stanja* (angl. state management). Poleg že poznanih konceptov Reacta, Remix vključuje tudi svoje dodatne funkcionalnosti za upravljanje stanja s podatki, pridobljenimi na strani strežnika. Na ta način lahko učinkovito obvladujemo stanje aplikacije, kar omogoča boljšo uporabniško izkušnjo in lažje upravljanje s podatki aplikacije.

Prilagodljiva arhitektura je še ena prednost ogrodja. Z njeno pomočjo lahko razvijalci enostavno organizirajo kodo in ločijo funkcionalnosti, kar olajša vzdrževanje in nadgradnjo aplikacij. Ogrodje je zasnovano tako, da podpira enostavno povezovanje med strežnikom in odjemalcem, kar omogoča dodajanje odjemalske logike brez obsežnih sprememb v obstoječi kodi na strežniku.

V grobem so razvijalci Remixa strnili filozofijo ogrodja v štiri točke [4]:

1. Sprejmi model strežnik/odjemalec, vključno z ločitvijo izvorne kode od vsebine/podatkov.
2. Delaj s temelji spleta, ne proti njim: izkoristi moč in funkcionalnosti, ki jih že ponujajo osnovni gradniki spleta, kot sta protokol HTTP in jezik HTML.
3. Uporabljaj JavaScript za nadgradnjo uporabniške izkušnje z oponašanjem vedenja brskalnika: pri uporabi JavaScripta v spletnih aplikacijah je smiselno dodajati nove funkcionalnosti, ki izboljšajo uporabniško izkušnjo. Vendar pa se ne bi smeli truditi ponovno implementirati že obstoječe funkcionalnosti, ki jo brskalniki že sami ponujajo.
4. Ne pretiravaj z abstrakcijo osnovnih tehnologij.

2.2 Podroben pregled ogrodja

V osnovi je ogrodje Remix sestavljeno iz štirih komponent [4]:

1. Prevajalnik (angl. Compiler)
2. Upravljavec strežniških HTTP zahtevkov (angl. server-side HTTP handler)
3. Strežniško ogrodje (angl. server framework)
4. Ogrodje za brskalnik (angl. browser framework)

Za razvoj aplikacij v ogrodju Remix je ključnega pomena uporaba *prevajalnika*. Prevajalnik, ki temelji na **esbuild** orodju, omogoča ustvarjanje različnih delov aplikacije. Med temi deli je tudi strežniški upravljavec HTTP zahtevkov, ki vključuje vse podprte poti spletne strani in ustrezne module. Slednje omogoča izrisovanje na strežniku in obdelavo zahtevkov za različne vire. Poleg tega prevajalnik ustvari tudi vse, kar je potrebno za izvajanje aplikacije v brskalniku. To vključuje samodejno deljenje programske kode glede na pot aplikacije in uvoz virov, na primer stilske datoteke CSS ali slike. Prevajalnik vzpostavi tudi manifest virov (angl. asset manifest), ki ima ključno vlogo pri pred-nalaganju virov med prvotnim izrisom na strežniku in pred-pridobivanju virov za prehode med stranmi na strani odjemalca. Vse tri gradbene artefakte lahko nato naložimo na katerikoli strežnik, ki podpira izvajanje programskega jezika JavaScript.

Naslednja komponenta, ki je sestavni del ogrodja Remix, je *Upravljavec strežniških HTTP zahtevkov*. Čeprav se Remix izvaja na strežniku, dejansko ne predstavlja strežnika. Je le upravitelj (angl. handler), ki se uporablja znotraj dejanskega strežnika JavaScript. Ogrodje je namreč zgrajeno na osnovi spletnega Fetch API-ja. Slednje omogoča, da Remix deluje na različnih strežnikih, kot so Vercel, Netlify, Architect, Cloudflare Workers ali Deno Deploy.

Programski kod spodaj prikazuje, kako je videti Remix, ko se izvaja v ogrodju Express.

```
const remix = require("@remix-run/express");
const express = require("express");

const app = express();

app.all("*", remix.createRequestHandler({ build: require("./build") }));
```

Express (oz. Node.js) je dejanski strežnik, medtem ko je Remix zgolj upravitelj (angl. handler) na tem strežniku. Paket **@remix-run/express**, ki se nahaja v prvi vrstici programskega koda zgoraj, imenujemo adapter. Ker so upravitelji neodvisni od strežnikov, za delovanje potrebujemo takšne adapterje.

V osnovi adapterji pretvorijo vmesnik zahtev in odgovorov določenega strežnika v standardizirani Fetch API na vhodu, ko zahteva pride do Remixa. Nato pa prilagodijo Fetch odgovor, ki ga vrne Remix, nazaj v vmesnik odgovora strežnika. S tem pristopom Remix postane prilagodljiv in sposoben delovati na različnih strežnikih, saj adapterji poskrbijo za medsebojno razumevanje med Remixom in posameznimi strežniki. Adapterji omogočajo, da Remix izkoristi prednosti in funkcionalnosti različnih strežnikov, hkrati pa ohranja enoten in dosleden način delovanja.

Komponenta *Strežniško ogrodje* predstavlja poglavitni del ogrodja. Remix je v osnovi strežniško ogrodje, ki sledi arhitekturi MVC (Model-View-Controller). V primerjavi s tradicionalnimi ogrodji, kot je npr. ASP.NET Core MVC, se v Remixu vlogi pogleda (View) in krmilnika (Controller) združita, medtem ko je model v celoti prepuščen razvijalcem. Namesto da bi bili omejeni na določene tehnologije, lahko izkoristimo celoten ekosistem JavaScripta, ki ponuja številne podatkovne baze, ORM (angl. Object-Relational Mapping) in podobna orodja. Poleg tega Remix vključuje funkcionalnosti za upravljanje piškotkov (angl. cookies) in sej (angl. sessions), kar olajša obvladovanje stanja med zahtevami.

Za razliko od običajnih strežniških ogrodij je Remix osredotočen na uporabniški vmesnik in ne na modele. V Remixu imajo poti (angl. routes) možnost obdelave celotnega spletnega naslova (URL) ali samo njegovega dela. Tako vsaka postavitev (pogled) postane svoj krmilnik, pri čemer ogrodje združuje podatke in komponente za izgradnjo uporabniškega vmesnika. Ena izmed ključnih prednosti tega pristopa je, da ne potrebujemo ločenih krmilnikov za vsak URL. Namesto tega lahko vsaka postavitev (pogled) vsebuje svojo logiko za obdelavo zahtev, kar omogoča, da so strani, ustvarjene z Remixom, bolj dinamične in interaktivne.

Pogosto poti Remixa vključujejo tako uporabniški vmesnik kot interakcije z modeli kar v isti datoteki. Datoteke imajo tri glavne izvoze: prejemnik (loader), akcijo (action) in privzet izvoz (dejanska komponenta). Primer je podan v programskem kodu spodaj.

```
// Prejemniki (loaders) se izvajajo le na strežniku in
// zagotavljajo podatke komponentam med zahtevami GET
export async function loader() {
  return json(await db.projects.findAll());
}

// Akcije (actions) se izvajajo le na strežniku in obdelujejo zahteve POST, PUT, PATCH in DELETE.
// Prav tako lahko zagotovijo podatke komponentam
export async function action({ request }: ActionArgs) {
  const form = await request.formData();

  await createProject({ title: form.get("title") });

  return json({ ok: true });
}

// Izvoz privzete vrednosti je komponenta, ki bo prikazana, ko pot ustreza URL-ju.
// To deluje tako na strežniku kot na odjemalcu.
export default function Projects() {
  const projects = useLoaderData<typeof loader>();
  const actionData = useActionData<typeof action>();

  return (
    <div>
      {projects.map((project) => (
        <Link key={project.slug} to={project.slug}>
          {project.title}
        </Link>
      ))}

      <Form method="post">
        <input name="title" />
        <button type="submit">Create New Project</button>
      </Form>
      {actionData?.errors ? <ErrorMessages errors={actionData.errors} /> : null}

      <Outlet />
    </div>
  );
}
```

Remix lahko uporabljamo tudi izključno kot strežniško ogrodje, brez kakršne koli uporabe JavaScripta na strani odjemalca. Za nalaganje podatkov se lahko uporablja funkcija **loader**, spremembe lahko upravljamo s funkcijo **action**, medtem ko obrazci HTML in komponente zagotavljajo osnovne funkcionalnosti spletne aplikacije.

Zadnja komponenta, *Ogrodje za brskalnik*, pride do izraza, ko Remix brskalniku posreduje dokument HTML. Takrat ogrodje poskrbi, da se stran "hidrira" (angl. hydrate). V kontekstu razvoja spletnih strani hidracija označuje postopek pretvorbe statične vsebine HTML-ja v interaktivno aplikacijo. Ko se spletna stran sprva izriše na strežniku in pošlje odjemalcu, vsebuje statično vsebino in nima dinamičnih funkcionalnosti. Hidracija je nadaljnji korak, ko brskalnik naloži JavaScript kodo in doda upravljavce dogodkov (angl. event handlers), kar naredi stran interaktivno [4].

Remix vključuje tudi nekaj optimizacij za navigacijo med stranmi na strani odjemalca. Ogrodje ve, katere postavitve (angl. layout) se bodo ohranile med dvema URL-jema, zato pridobiva podatke le za tiste, ki se spreminjajo. Ko uporabnik klikne povezavo, namesto da bi na strežnik pošiljali zahtevo za celoten dokument in vse vire, Remix pridobi podatke za prikaz želene strani in posodobi uporabniški vmesnik. Slednje izboljša zmogljivosti v primerjavi z zahtevo za celoten dokument.

Pomembna lastnost ogrodja je tudi *predpomnjenje* (angl. Caching), kar je podprto na naslednji način. Ko uporabnik z miško prehaja preko povezave, Remix že vnaprej predpomni vse potrebne vire za naslednjo stran. To pomeni, da se podatki, moduli JavaScript in celo datoteke CSS za novo stran že naložijo, še preden uporabnik dejansko

klikne na povezavo. To stori tako, da brskalniku pošlje manifest sredstev, ki vsebuje seznam vseh virov, ki jih potrebuje stran. Brskalnik nato te vire predpomni v ozadju, preden jih potrebuje, kar zmanjša čas nalaganja strani.

Prednalaganje virov v Remixu poteka na naslednji način:

1. Remix ustvari manifest sredstev, ki vsebuje seznam vseh virov, ki jih potrebuje stran.
2. Remix pošlje manifest sredstev brskalniku.
3. Brskalnik lahko nato te vire predpomni v ozadju.
4. Ko uporabnik klikne povezavo, bo stran že naložena in pripravljena za uporabo.

Ta pristop k predpomnjenju virov je izredno učinkovit, še posebej v primerih počasne spletne povezave. Ker se večina virov že nahaja v predpomnilniku brskalnika, se stran naloži hitro in brez čakanja, kar uporabnikom omogoča odzivno izkušnjo.

2.3 Generiranje spletne strani na strežniku (SSR) v ogrodju Remix

Client-Side Rendering (CSR) je pristop, ki se pogosto uporablja pri izdelavi spletnih aplikacij. Pri tem pristopu se HTML oz. uporabniški vmesniki oblikuje na strani odjemalca, s pomočjo JavaScripta. Ko uporabnik zahteva določeno spletno stran, se na strani odjemalca najprej naloži osnovni HTML skupaj z datotekami JavaScript. Nato se na strani odjemalca s pomočjo JavaScripta pridobijo podatki iz strežnika in dinamično oblikuje vsebina HTML [6]. CSR pristop omogoča bolj interaktivno uporabniško izkušnjo, saj se vsebina in vmesnik prilagajata uporabnikovim interakcijam brez potrebe po osveževanju celotne strani. Vendar pa CSR prinaša tudi nekatere izzive, kot so počasnejši prvi prikaz strani, saj je potrebno najprej naložiti osnovno strukturo in nato pridobiti in oblikovati podatke z JavaScriptom. Poleg tega je zaradi obsežne uporabe JavaScripta obremenitev na strani odjemalca večja, kar lahko vpliva na učinkovitost in odzivnost spletnih aplikacij.

Kot alternativo imamo že nekaj časa na voljo pristop SSR (*Server-Side Rendering*). Za razliko od CSR se strani HTML ustvarijo in oblikujejo na strežniku, preden se pošljejo na naprave odjemalca. Ko uporabnik zahteva določeno spletno stran, spletni strežnik sestavi potrebno HTML vsebino, vključno s podatki iz baze ali drugih virov, nato pa jo pošlje odjemalcu. Ta pristop omogoča, da se uporabniku prikaže popolnoma oblikovana spletna stran že ob prvem naloženem odgovoru strežnika, saj ni potrebno, da bi se spletna stran generirala šele na strani odjemalca s pomočjo JavaScripta [6]. Ko se spletna stran naloži, pa preko že prej omenjenega postopka hidracije za vso nadaljnjo izvajanje poskrbi koda JavaScript. SSR prinaša nekatere prednosti, kot so boljša uporabniška izkušnja zaradi hitrejšega prikaza začetne strani, boljša optimizacija za iskalnike (SEO) in boljša dostopnost.

Remix privzeto uporablja SSR kot osrednji mehanizem izrisovanja (angl. rendering), vendar vključuje tudi CSR za usmerjanje na strani odjemalca in dinamično posodabljanje strani po prvotnem nalaganju. Ta hibridni pristop združuje prednosti SSR za prvotno izrisovanje z ugodnostmi CSR za boljše interakcije na strani odjemalca.

2.4 Optimizacija SEO in hitro nalaganje začetne strani

SEO (ang. Search Engine Optimization) pomeni optimizacija spletnih strani za iskalnike. Gre za prakso optimizacije spletne strani ali spletišča za izboljšanje njene vidnosti in uvrstitve na straneh iskalnikov (ang. SERP-ji oz. Search Engine Result Pages). Glavni cilj SEO-ja je privabiti več organskega (neplačanega) prometa iz iskalnikov, kot so Google, Bing, Yandex ali Yahoo [7].

Remix dosega boljšo optimizacijo za iskalnike z naslednjimi mehanizmi [4].

Kot že omenjeno, se Remix razlikuje od običajnih enostranskih aplikacij (Single-page application), saj privzeto podpira SSR. Ko obiskovalec zahteva spletno stran, se s tem pristopom na strežniku generira začetni HTML s predhodno vključeno vsebino. To zagotavlja, da so spletni iskalniki sposobni takoj dostopati do celotne vsebine, kar olajša njihovo indeksiranje.

Ena izmed prednosti Remixa je tudi nadzor nad metapodatki za SEO. Ponuja orodja in tehnike za prilagajanje naslovov strani, meta-opisov, kanoničnih URL-jev in strukturiranih podatkov. Pravilno upravljanje metapodatkov pripomore k boljšemu razumevanju vsebine s strani spletnih iskalnikov, kar izboljšuje možnosti za višje uvrščanje v iskalnih rezultatih.

Pomembno je tudi izpostaviti, da Remix skrbi za čiste URL-je. To obenem olajša navigacijo obiskovalcem in hkrati pripomore k boljšemu indeksiranju vsebine s strani iskalnih botov.

Nenazadnje pa ogrodje tudi stremi k manjši odvisnosti od JavaScripta na strani odjemalca, kar pripomore k boljšemu razumevanju in sledenju vsebine s strani spletnih iskalnikov. Z uporabo SSR in postopnega izboljševanja, Remix omogoča enostavno dostopanje do vsebine, kar je ključnega pomena za SEO.

Kombinacija vseh omenjenih pristopov, torej SSR, postopnega izboljševanja, nadzora metapodatkov in drugih funkcij, usmerjenih v SEO, naredi Remix skladen z najboljšimi praksami za SEO.

3 Primerjava z obstoječimi meta-ogrodji

Remix ni edino meta-ogrodje, ki temelji na knjižnici React. Med alternativami se pogosto omenjata tudi Next.js in Gatsby. Čeprav imajo nekaj podobnosti, imajo tudi različne značilnosti in primere uporabe, kar bomo predstavili v nadaljevanju.

3.1 Next.js

Next.js je priljubljeno odprtokodno ogrodje za izdelavo naprednih spletnih aplikacij s pomočjo jezika JavaScript. Temelji na knjižnici React in ponuja napredne funkcionalnosti, ki olajšajo razvoj in izboljšajo uporabniško izkušnjo. Glavna značilnost Next.js je njegova sposobnost podpiranja tako SSR kot tudi CSR [8].

Next.js ponuja preprost pristop k usmerjanju, ki temelji na sistemu datotek. To pomeni, da lahko razvijalci organizirajo in upravljajo poti v svoji aplikaciji preprosto z uporabo imenikov in datotek, kar olajša vzdrževanje in razširljivost.

Poleg tega Next.js vključuje tudi podporo za pridobivanje podatkov. Na voljo imamo različne metode, kot so pridobivanje podatkov na strani strežnika, statična generacija spletnih strani ali pridobivanje podatkov na strani odjemalca.

3.2 Gatsby

Gatsby je še eno priljubljeno odprtokodno ogrodje za izdelavo spletnih aplikacij, ki temelji na Reactu. Ogrodje je namenjeno predvsem izdelavi statičnih spletnih strani.

Glavna značilnost Gatsbyja je njegova sposobnost pred-generiranja statičnih HTML strani med izdelavo aplikacije (angl. build time). To pomeni, da se vsebine in podatki pridobijo med gradnjo aplikacije, nato pa se izdelajo pred-izrisane (angl. pre-rendered) strani HTML. Te strani se nato posredujejo odjemalcu kot običajne statične datoteke, kar omogoča izjemno hitro nalaganje in odzivnost spletnega mesta [9].

Gatsby omogoča razvijalcem enostavno upravljanje vsebine s pomočjo datotek Markdown ali drugih formatov. Poleg tega Gatsby uporablja GraphQL za pridobivanje podatkov, kar olajša integracijo s CMS platformami in drugimi zunanjimi viri podatkov. Podatki se torej pridobivajo in uporabljajo na učinkovit način, kar prispeva k hitrosti in zmogljivosti spletne strani. Gatsby ponuja tudi bogat ekosistem vtičnikov, ki omogočajo razvijalcem dodajanje funkcionalnosti, kot so optimizacija zmogljivosti, analitika, SEO in še več.

Gatsby se predvsem osredotoča na statično generacijo spletnih strani (SSG) in je primeren za izdelavo spletnih strani, osredotočenih na vsebino, kot so blogi.

3.3 Rezultati primerjave

Tabela 1 prikazuje rezultate primerjave ogrodij Remix, Next.js in Gatsby. Pri primerjavi smo se osredotočili predvsem na načine izrisa, SEO, pridobivanja podatkov, usmerjanja in podpori obrazcev.

Tabela 1: Primerjava meta-ogrodij React.

	Remix	Next.js	Gatsby
Izris	Server-side rendering (SSR)	Server-side rendering (SSR), Static Site Generation (SSG)	Static Site Generation (SSG)
SEO	Zelo dobra podpora zaradi SSR	Zelo dobra podpora zaradi SSR	Odlična podpora zaradi SSG
Skupnost	Naraščа, vendar ni primerljiva z Next.js in Gatsby	Obsežna, dejavna	Obsežna, dejavna
Pridobivanje podatkov	Uporablja React Suspense za asinhrono pridobivanje podatkov	Uporablja getStaticProps ali getServerSideProps za statično pridobivanje podatkov	Uporablja GraphQL za pridobivanje podatkov
Usmerjanje	Uporablja Remix Router ze upravljanje s potmi, uporablja strukturo projekta za ustvarjanje poti	Uporablja Next.js Router, uporablja strukturo projekta za ustvarjanje poti	Uporablja Gatsby Router
Upravljanje z obrazci	Uporablja privzeto obnašanje obrazcev na strani odjemalca za pošiljanje podatkov	Uporablja prilagojene (angl. custom) obrazce	Uporablja mutacije GraphQL

Kot je razvidno iz tabele, so Remix, Next.js in Gatsby izvrstna meta-ogrodja za razvoj spletnih aplikacij. Vsak od njih ima svoje edinstvene prednosti in značilnosti, zato je ključno izbrati ogrodje, ki najbolje ustreza specifičnim potrebam projekta.

Iz primerjave lahko razberemo, da Remix omogoča razvoj odzivnih aplikacij in zagotavlja podporo za optimizacijo spletnega iskanja (SEO). Po drugi strani Next.js poleg SSR podpira tudi izris statičnih spletnih strani. Močno podporo SEO pa zagotavlja Gatsby, ki je primarno namenjen implementaciji statičnih spletnih strani.

4 Implementacija aplikacije

V tem poglavju bomo predstavili proces implementacije spletne aplikacije z uporabo ogrodja Remix. V osnovi bo aplikacija omogočala uporabnikom, da vnesejo kakršne koli ideje, drugi uporabniki bodo imeli možnost »všečkati« posamezno idejo.

4.1 Namestitev osnovne aplikacije

Za izdelavo aplikacije Remix smo uporabili naslednje:

- Node.js v18.16.0 (okolje za izvajanje kode JavaScript),
- npm v9.5.1 (upravitelj paketov za aplikacije Node.js) in
- VS Code (urejevalnik kode).

Skladno z navodili se razvoj začne z izvedbo ukaza **npx create-remix@latest** v ukazni vrstici. Ob sami vzpostavitvi se moramo opredeliti glede določenih aspektov aplikacije, kar bomo predstavili v nadaljevanju.

Na začetku smo morali izbrati, kateri tip aplikacije želimo razviti. Na voljo imamo dve možnosti, za voljo razvoja tega primera pa smo se odločili za preprosto aplikacijo (možnost **just the basics**).

```
? What type of app do you want to create? (Use arrow keys)
> Just the basics
A pre-configured stack ready for production
```

Remix prav tako ponuja različne ponudnike za namestitev aplikacije. V našem primeru smo izbrali **Remix App Server**.

```
? Where do you want to deploy? Choose Remix App Server if you're unsure; it's easy to change
deployment targets.
> Remix App Server
  Express Server
  Architect
  Fly.io
  Netlify
  Vercel
  Cloudflare Pages
```

Ko potrdimo vse možnosti vzpostavitve aplikacije Remix, se v izbranem imeniku na podlagi šablone (angl. Template) ustvari naslednja struktura [4]:

- app/ - Imenik, kjer se nahaja celotna aplikacija Remix.
- app/entry.client.tsx – Predstavlja kodo JavaScript, ki se izvede, ko se aplikacija naloži v brskalniku. Uporablja se za hidracijo komponent React.
- app/entry.server.tsx – Predstavlja kodo JavaScript, ki se izvede, ko zahtevek doseže strežnik. Remix obdela vse potrebne podatke, razvijalci aplikacije pa implementiramo ustrezni odgovor (angl. Response). Programski kod v tej datoteki skrbi, da se aplikacija React pretvori v niz, ki se potem pošlje odjemalcu.
- app/root.tsx – V tej datoteki se nahaja korenska komponenta aplikacije, ki izriše element <html>.
- app/routes/ - Tu se nahajajo vsi moduli poti (angl. route modules). Remix na podlagi imena datotek v tej mapi ustvari URL poti za aplikacijo.
- public/ - Tu se nahajajo statični viri (slike, pisave itd.).
- remix.config.js – Vsebuje možnosti konfiguracije aplikacije.

V nadaljevanju bomo predstavili pomembnejše vidike aplikacije in podali primer programskega koda.

4.2 Podatkovna baza

Za upravljanje podatkovne baze smo uporabili SQLite v povezavi s Prisma. Prisma je odprtokodni objektno-relacijski preslikovalnik (ORM) za Node.js in TypeScript ter se uporablja kot alternativa pisanju običajnih poizvedb SQL. Prisma trenutno podpira različne podatkovne baze, med drugim PostgreSQL, MySQL, SQL Server, SQLite, MongoDB in CockroachDB [10].

S pomočjo Prisma Migrate smo ustvarili tabelo v podatkovni bazi, ki bo služila shranjevanju idej. Shema tabele je prikazana v naslednjem programskem kodu.

```
model Idea {
  id          String   @id @default(uuid())
  createdAt   DateTime @default(now())
  updatedAt   DateTime @updatedAt
  title       String
  description String
  likes       Int      @default(0)
}
```

4.3 Korenska komponenta

Kot že rečeno, korenska komponenta aplikacije izriše element <html>, skrbi pa tudi, da se naložijo metapodatki strani (<Meta />), povezave do stilov CSS (<Links />) in prostor, kamor se glede na naslov URL-ja izpiše vsebina aplikacije (<Outlet />). Če želimo izvajati kodo JavaScript tudi na strani odjemalca, pa je potrebno vključiti še <Scripts />.

Primer korenske komponente za potrebe aplikacije je predstavljen v naslednjem izseku programskega koda.

```
import globalStylesUrl from "~/styles/global.css";

export const links: LinksFunction = () => [
  { rel: "stylesheet", href: globalStylesUrl },
];

export default function App() {
  return (
    <html lang="en">
      <head>
        <meta charSet="utf-8" />
        <meta name="viewport" content="width=device-width,initial-scale=1" />
        <Meta />
        <Links />
      </head>
      <body>
        <Outlet />
        <Scripts />
        <LiveReload />
      </body>
    </html>
  );
}
```

4.4 Spletne poti

Aplikacija bo nudila možnost pregleda seznama idej, pregleda posamezne ideje in dodajanje nove ideje. Kot že rečeno, usmerjanje v aplikacijah Remix temelji na osnovi datotek, kjer so poti določene na podlagi strukture. Za naše potrebe bi potrebovali naslednje poti:

- /
- /ideje
- /ideje/:idejaId
- /ideja/dodaj-novo

Da dosežemo takšne poti, moramo ustrezno poimenovati datoteke. Glede na konvencijo poimenovanja poti in datotek, zgoraj opredeljene poti postanejo datoteke znotraj imenika **app/routes**:

- _index.tsx
- ideja.tsx
- ideja.$idejaId.tsx
- ideja.dodaj-novo.tsx

4.4.1 Glavna pot

V datoteki **_index.tsx** se nahaja glavna pot, ki se izvede, ko uporabnik obišče osnovni URL aplikacije. Ta pot predstavlja torej začetno stanje aplikacije, ki se prikaže ob prvem obisku. Programski kod glavne poti je naslednji.

```
export const links: LinksFunction = () => [
  { rel: "stylesheet", href: stylesUrl },
];

export default function IndexRoute() {
  return (
    <div className="container">
      <div className="content">
        <h1>Aplikacija za beleženje seznama idej</h1>
        <nav>
          <ul>
            <li>
              <Link to="ideja">Preglej seznam idej</Link>
            </li>
          </ul>
        </nav>
      </div>
    </div>
  );
}
```

Kot je razvidno, je v datoteki definirana komponenta **IndexRoute**, ki predstavlja začetno stanje aplikacije. Gre za preprosto komponento React, brez kakršnih koli dodatkov ogrodja Remix.

Ko se ta pot izvede, se prikaže naslov strani, navigacijski meni z gumbom za pregled seznama idej in potrebni slogi za oblikovanje. Ta komponenta se nato izriše v <Outlet /> korenske komponente aplikacije. Kot že omenjeno, je <Outlet /> posebna oznaka, ki določa mesto, kamor se vstavi vsebina (izrišejo komponente), povezane s trenutno potjo.

4.4.2 Pot, ki prikaže seznam idej

Za izpis seznama idej, ki se nahaja na poti **/ideje**, potrebujemo na strežniku dostop do podatkovne baze. Obenem želimo tudi omejiti, katere podatke posredujemo odjemalcu. Vse to podpira ogrodje Remix tako, da izvozimo funkcijo **loader**. Ta funkcija omogoča pridobivanje in obdelavo podatkov preden se prikažejo odjemalcu. Vsebino funkcije predstavlja naslednji programski kod.

```
export const loader = async () => {
  const ideas = await db.idea.findMany({
    orderBy: { createdAt: "desc" },
    select: { id: true, title: true, likes: true },
    take: 5,
  });

  return json({ ideas });
};
```

Iz primera zgoraj je razvidno, da se funkcija loader uporablja za pridobivanje idej iz podatkovne baze s pomočjo funkcije **db.idea.findMany**. S to funkcijo lahko določimo različne parametre, kot je ureditev po času ustvarjanja ideje, izbira določenih polj tabele in omejitev števila idej, ki jih bomo prikazali.

Podatki so nato na voljo na strani odjemalca preko **useLoaderData** kavlja (angl. hook), ki ga ponuja ogrodje Remix. Kavelj vrne vsebino, ki jo nato s pomočjo standardne sintakse Reacta prikažemo v obliki seznama, kot je razvidno iz spodnjega izseka programskega kode.

```
export default function IdeasRoute() {
  const data = useLoaderData<typeof loader>();

  return (
    <main className="ideas-main">
      <div className="ideas-list">
        <p>Seznam idej</p>
        <ul>
          {data.ideas.map(({ id, title, likes }) => (
            <li key={id}>
              <Link to={id} prefetch="intent">
                {title}
              </Link>
              {likes}👍
            </li>
          ))}
        </ul>
        <Link to="new" className="button">
          Dodaj svojo idejo
        </Link>
      </div>
      <div className="ideas-outlet">
        <Outlet />
      </div>
    </main>
  );
}
```

4.4.3 Pot za dodajanje nove ideje

Za dodajanje nove ideje bomo potrebovali obrazec, ki se bo ob potrditvi na strani odjemalca shranil v podatkovno bazo. V ta namen ogrodje Remix ponuja funkcijo **action**. Funkcija action omogoča, da opravimo obdelavo obrazcev in shranjevanje podatkov neposredno na strežniku. Primer funkcije za shranjevanje novih idej predstavlja naslednji programski kod.

```
export const action = async ({ request }: ActionArgs) => {
  const form = await request.formData();

  const title = form.get("title");
  const description = form.get("description");

  if (typeof title !== "string" || typeof description !== "string") {
    return badRequest({
      fieldErrors: null,
      fields: null,
      formError: "Form not submitted correctly.",
    });
  }

  const newIdea = await db.idea.create({
    data: { description, title },
  });

  return redirect(`/ideje/${newIdea.id}`);
};
```

Za vnos podatkov potrebujemo še obrazec, ki je implementiran v obliki komponente React. V primeru omenjene komponente ne potrebujemo specifičnih funkcionalnosti ogrodje Remix, saj gre za osnovne funkcionalnosti ogrodja React, kar je razvidno iz programskega koda spodaj.

```
export default function NewIdeaRoute() {
  return (
    <div>
      <p>Dodaj svojo idejo</p>
      <form method="post">
        <div>
          <label>
            Naziv:
            <input type="text" name="title" required />
          </label>
        </div>
        <div>
          <label>
            Opis:
            <textarea name="description" required />
          </label>
        </div>
        <div>
          <button type="submit" className="button">
            Dodaj idejo
          </button>
        </div>
      </form>
    </div>
  );
}
```

Čeprav gre za preprosto komponento React, je pomembno poudariti, da ob potrditvi obrazca ogrodje Remix prevzame nadzor nad izvedbo in samodejno kliče zgoraj definirano funkcijo action. To pomeni, da se ob oddaji obrazca ne izvede le osnovna funkcionalnost komponente React, temveč ogrodje Remix poskrbi za izvedbo dejanj, kot sta obdelava obrazca in shranjevanje podatkov. S tem pristopom je doseženo tesno usklajevanje med obrazcem na odjemalcu in obdelavo na strežniku.

4.4.4 Pot za pregled posamezne ideje

Pot, ki omogoča tako pregled posamezne ideje glede na njeno vrednost ID kot »všečkanje« ideje, združuje oba koncepta, predstavljana v prejšnjih podpoglavjih, torej funkciji loader in action.

Funkcija loader bo poskrbela, da se na podlagi trenutnega ID-ja v naslovu URL izpišejo ustrezni podatki pripadajoče ideje. Uporabniku se bo torej prikazal naslov, opis in število »všečkov« ideje.

Funkcija action pa bo skrbela za posodabljanje števila »všečkov« ob kliku na pripadajoči gumb.

Implementacija obeh funkcij je razvidna iz naslednjega programskega koda.

```
export const loader = async ({ params }: LoaderArgs) => {
  const idea = await db.idea.findUnique({
    where: { id: params.ideaId },
    select: { title: true, description: true, likes: true },
  });

  if (!idea) {
    throw new Error("Idea not found");
  }

  return json({ idea });
};

export const action = async ({ params }: ActionArgs) => {
  const idea = await db.idea.findUnique({
    where: { id: params.ideaId },
  });

  if (!idea) throw new Error("Idea not found");

  const updatedLikes = idea?.likes + 1;

  await db.idea.update({
    where: { id: params.ideaId },
    data: {
      likes: updatedLikes,
    },
  });

  return redirect(".");
};
```

V primeru komponente, ki prikazuje podatke o posamezni ideji, se poleg osnovnih funkcionalnosti React poslužujemo tudi dodatnih funkcionalnosti, ki jih nudi Remix. Programski kod prikazuje izpis podrobnosti posamezne ideje in gumb za »všečkanje«.

```
export default function IdeaRoute() {
  const data = useLoaderData<typeof loader>();

  return (
    <div>
      <p>Idea:</p>
      <h1>{data.idea.title}</h1>
      <p>{data.idea.description}</p>
      <p>Všečki: {data.idea.likes}</p>
      <Form method="post">
        <button className="button">👍</button>
      </Form>
    </div>
  );
}
```

Kavelj useLoaderData smo že predstavili v prejšnjem podpoglavju, v zgornjem primeru pa je potrebno izpostaviti tudi komponento **<Form>**.

V ogrodju Remix je uporaba velike začetnice pri komponenti <Form> namenjena razlikovanju med običajnimi elementi HTML in prilagojenimi komponentami, ki jih zagotavlja ogrodje Remix. Ključna razlika med komponento <Form> in običajnim elementom HTML <form> je, kako ogrodje upravlja oddajo obrazcev. Medtem ko običajen

HTML obrazec običajno sproži ponovno naložitev strani, komponenta <Form> ogrodja Remix uporablja navigacijo na strani odjemalca, kar prepreči ponovno nalaganje strani.

V zgornjem primeru smo preko implementacije prikazali, kako preprosto je implementirati interaktivne spletne aplikacije s pomočjo ogrodja Remix. Zahvaljujoč intuitivni naravi in naprednim funkcionalnostim, ogrodje omogoča razvijalcem hiter in učinkovit razvoj dinamičnih aplikacij s poudarkom na uporabniški izkušnji.

5 Zaključek

V prispevku smo predstavili ogrodje Remix in podrobno opisali njegove lastnosti in filozofijo. Remix se je izkazal za inovativno meta-ogrodje, ki postavlja v ospredje uporabniško izkušnjo, lažje vzdrževanje in zmogljivost spletnih aplikacij. Opravili smo tudi primerjavo z drugimi podobnimi meta-ogrodji, da bi bolje razumeli, kako se Remix uvršča v ekosistem razvoja spletnih aplikacij. Medtem ko je na trgu več rešitev, Remix izstopa po svoji edinstveni kombinaciji pristopov, ki združujejo najboljše lastnosti obstoječih ogrodij. Na koncu smo še predstavili praktičen primer uporabe ogrodja, kjer smo ustvarili spletno aplikacijo, ki omogoča uporabnikom dodajanje idej. Implementacija je pokazala, da ogrodje dobro podpira zahteve dinamičnih spletnih aplikacij.

Ogrodje Remix je nedvomno obetavno ogrodje za razvoj spletnih aplikacij, ki odpira vrata inovacijam in učinkovitemu razvoju. S svojo osredotočenostjo na razvijalsko izkušnjo in podporo spletnim standardom zagotavlja, da bo imel pomembno vlogo v prihodnosti razvoja spletnih aplikacij.

Literatura

[1] S. Roy, „The Difference Between a Framework and a Library,“ 23 12 2022. [Elektronski]. Dosegljivo: https://www.baeldung.com/cs/framework-vs-library.

[2] Stack Overflow, „Stack Overflow Developer Survey 2023,“ 2023. [Elektronski]. Dosegljivo: https://survey.stackoverflow.co/2023/.

[3] Codemotion, „Here to Stay: All About Meta-Frameworks,“ 20 4 2023. [Elektronski]. Dosegljivo: https://www.codemotion.com/magazine/languages/here-to-stay-all-about-meta-frameworks/.

[4] Remix Software, Inc., „Remix Docs,“ 2023. [Elektronski]. Dosegljivo: https://remix.run/docs/en/main.

[5] MDN contributors, „Progressive Enhancement,“ 2023. [Elektronski]. Dosegljivo: https://developer.mozilla.org/en-US/docs/Glossary/Progressive_Enhancement.

[6] P. Ram, „Server Side Rendering (SSR) vs. Client Side Rendering (CSR) vs. Pre-Rendering using Static Site Generators (SSG) and client-side hydration,“ 19 10 2021. [Elektronski]. Dosegljivo: https://medium.com/@prashantramnyc/server-side-rendering-ssr-vs-client-side-rendering-csr-vs-pre-rendering-using-static-site-89f2d05182ef.

[7] D. Goodwin, „What Is SEO – Search Engine Optimization?,“ 2023. [Elektronski]. Dosegljivo: https://searchengineland.com/guide/what-is-seo.

[8] D. Niżyński, „What is next js and why should you use it in 2023?,“ 13 7 2023. [Elektronski]. Dosegljivo: https://pagepro.co/blog/what-is-nextjs/.

[9] L. D., „What Is Gatsby and How It Works,“ 13 4 2023. [Elektronski]. Dosegljivo: https://www.hostinger.com/tutorials/what-is-gatsby.

[10] Prisma Data, Inc., „Prisma,“ 2023. [Elektronski]. Dosegljivo: https://www.prisma.io/docs/concepts/overview/what-is-prisma.