Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Marko Grešak

Ocena primernosti jezika Elm za razvoj spletnih aplikacij

diplomsko delo

univerzitetni študijski program
prve stopnje
računalništvo in informatika

Mentor: doc. dr. Boštjan Slivnik

Ljubljana, 2018

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

Elm je razmeroma nov funkcijski programski jezik za izdelavo spletnih aplikacij. Na primeru konkretne spletne aplikacije ugotovite, kako se Elm obnese v praksi. V ta namen izberite neko obstoječo spletno aplikacijo in po njenem vzoru izdelajte dve verziji, eno v Elmu in eno v JavaScriptu. Primerjajte način izdelave in nastali aplikaciji ter programerske izkušnje glede na uporabljen programski jezik.
Elm is a relatively young functional programming language for creating web applications. Find out how Elm works out in practice by taking an existing web application as a test case. Create two versions of the selected web applications, one written in Elm and the other in JavaScript. Compare the implementation process, the resulting applications and the programmer's experience regarding the programming language.

Iskreno se zahvaljujem svojemu mentorju, doc. dr. Boštjanu Slivniku, za vso pomoč pri izdelavi naloge in čas, ki mi ga je posvetil.
Zahvaljujem se tudi staršema, ki sta mi omogočila študij in me podpirala ter celotni družini, da so me podpirali ter mi med časom študija stali ob strani.

Kazalo

Slike

Tabele

Seznam uporabljenih kratic

kratica	angleško	slovensko
API	Application Programming Interface	aplikacijski programski vmesnik
HTML	Hyper Text Markup Language	jezik za označevanje nadbesedila
DOM	Document Object Model	objektni model dokumenta
JSON	JavaScript Object Notation	Opis JavaScript objekta
REST	REpresentational State Transfer	Predstavitveni prenos stanja
CSS	Cascading Style Sheets	predloge, ki določajo izgled spletnih strani
JSX	JavaScript XML	XML notacija v JavaScript
XML	eXtensible Markup Language	razširljiv označevalni jezik

Povzetek

Naslov: Ocena primernosti jezika Elm za razvoj spletnih aplikacij

Avtor: Marko Grešak

Funkcijski programski jezik Elm je razvit z namenom izdelave spletnih uporabniških vmesnikov. Prevede se v JavaScript in tako omogoča izvajanje v spletnih brskalnikih. Obljublja hitro delovanje brez napak med izvajanjem, hkrati pa si prizadeva razvijalcem nuditi prijazno razvojno okolje, kar, vsaj na prvi pogled, deluje kot obetavna rešitev. Omenjene obljube naj bi izpolnjeval prevajalnik, ki napake zazna že med prevajanjem kode s sklepanjem o tipih spremenljivk ter skuša razvijalcu problem opisati na čimbolj uporaben način. Namen tega diplomskega dela je preveriti navedene obljube, primerjati jezik Elm z alternativnim načinom razvoja spletnih aplikacij z uporabo JavaScript knjižnice za izdelavo spletnih uporabniških vmesnikov React ter oceniti primernost izbire jezika Elm tako, da bo razvijalcem v pomoč pri izbiri orodij za razvoj njihove naslednje aplikacije. Na podlagi meritev se jezik Elm izkaže kot boljša izbira za izdelavo zmogljive spletne aplikacije. Jezik Elm se izkaže za bolj zmogljivo rešitev in je primeren predvsem v primerih, kjer je potrebno izdelati robustno spletno aplikacijo. Tudi izkušnja med razvojem, kar je sicer subjektivna metrika, je za razvijalca prijeta.

Ključne besede: Elm, React, JavaScript, funkcijsko programiranje.

Abstract

Title: Assessing the suitability of Elm language for developing web applications

Author: Marko Grešak

Elm is a functional programming language which is compiled to JavaScript and therefore can be run in browsers. It promises no runtime errors and brings support tools developed with developers' satisfaction in mind. These promises are being realized by Elm compiler, which uses type inference to detect potential runtime errors at compile time and in addition it tries to produce a human-friendly error message. The goal of this thesis is to compare Elm language with a widely used React view library by implementing the same web application with both Elm language and React. Results of this thesis should aid the reader with the decision of whether to use Elm language as a tool for the next web application project or not. Based on the measurements, the Elm language proves as a better option for developing web applications where performance and robustness are key features. Although subjective, the developer's experience proved to be more pleasant.

Keywords: Elm, React, JavaScript, functional programming.

Poglavje 1

Uvod

1.1 Motivacija za izbrano diplomsko temo

Ideja izhaja iz želje spoznati in raziskati programski jezik Elm ter ga primerjati z obstoječimi načini izdelave spletnih aplikacij. Motivacija je med raziskavo pridobiti novo znanje na področju razvoja spletnih aplikacij, znanje o funkcijskih programskih jezikih in izkušnje z jezikom Elm.

V diplomskem delu se primerja proces izdelave spletne aplikacije z uporabo jezika Elm in na drugi strani React, JavaScript knjižnice za izdelavo uporabniških vmesnikov. Z vsako tehnologijo bo za lažjo primerjavo izdelana spletna aplikacija na način, da si bosta končni implementaciji čim bolj podobni. Primerjava bo izvedena na podlagi metrik, ki so pomembne pri spletnih aplikacijah.

Pričakovan rezultat so ugotovitve o primernosti jezika Elm. Pričakovanje je, da bodo rezultati koristili razvijalcem spletnih aplikacij, ki izbirajo orodja za izdelavo nove spletne aplikacije.

1.2 Pregled področja

1.2.1 Jezik Elm

Elm je ob času pisanja sorazmerno nov programski jezik. Edino najdeno sorodno delo, ki se navezuje na primerjavo načinom dela z jezikom Elm z načinom dela v jeziku JavaScript, je magistrska naloga „Elmulating JavaScript“ [45].

V diplomskem delu „Elm: Concurrent FRP for functional GUIs“ [43] avtor jezika Elm, Evan Czaplicki, opisuje motivacijo za razvoj jezika Elm. Czaplicki in mentor diplomskega dela Stephen Chong dodatno opisujeta ideje za jezikom Elm v publikaciji z naslovom „Asynchronous Functional Reactive Programming for GUIs“ [44].

Zanimiv vir informacij sta tudi knjigi „Elm In Action“ [48] ter „Programming Elm“ [46]. Vendar pa sta omenjeni knjigi ob času pisanja tega diplomskega dela še v nastajanju.

1.2.2 Način primerjave

Jezik Elm skuša rešiti probleme specifično na področju razvoja spletnih aplikacij, kjer prevladuje jezik JavaScript in ogrodja napisana v le-tem. Za razliko od jezika JavaScript je jezik Elm funkcijski jezik. S to razliko v okolje razvoja spletnih aplikacij vpeljuje nove ideje, ki so že znane med ostalimi splošno namenskimi funkcijskimi programskimi jeziki.

Zato je lahko primerjava izvedena na nivoju primerjave programskih jezikov ali pa se jezik Elm obravnava kot ogrodje za razvoj spletnih aplikacij in se primerja funkcionalnosti, ki jih omogoča jezik Elm z funkcionalnostmi ostalih ogrodij.

Pri primerjavi s programskimi jeziki je vredno omeniti ostale funkcijske programske jezike. Tu je pomemben predvsem jezik Haskell, ki je še najbolj podoben jeziku Elm. Vplivna jezika sta tudi Lisp in Standard ML.

Med ogrodji za razvoj spletnih aplikacij, ki so napisana v jeziku JavaScript, je veliko izbire [7]. Tukaj se za pomoč obrnemo na raziskavo o najbolj

popularnih ogrodjih za razvoj spletnih aplikacij v letu 2017 [37]. Pomembni rezultati raziskave so prikazani na grafu na sliki 1.1, kjer se knjižnica React izkaže kot najbolj popularna izbira. Za knjižnico React sledi pristop razvoja brez ogrodja, vendar ta tehnično ne sodi v kategorijo ogrodij za razvoj spletnih aplikacij. Pogosti izbiri sta tudi ogrodji Angular in Vue.js.

Slika 1.1: Rezultati ankete stanja med JavaScript ogrodji za razvoj spletnih aplikacij [37] v letu 2017.

1.2.3 Knjižnica React

Knjižnica React je, kot najbolj pogosta izbira v letu 2017 [37], zanimiva izbira za primerjavo z jezikom Elm. Če se rezultati za jezik Elm izkažejo kot boljši od najbolj pogoste izbire je to dober znak, da je jezik Elm primerna izbira za razvoj spletnih aplikacij.

Knjižnico React razvija ekipa pri podjetju Facebook. Zaradi tega dejstva je knjižnica že od začetka imela zelo visok nivo zanimanja. Na temo knjižnice React ni akademskih objav, ki bi bile zanimive v kontekstu te diplomske naloge.

Kot vir informacij je na voljo več knjig, na primer „React. js Essentials“ [47] ali „Introduction to React“ [49]. Vendar se pri omenjeni literaturi pojavi problem, saj se knjižnica React aktivno razvija in pogosto spreminja. Namreč

od konca leta 2015, ko sta bili omenjeni knjigi izdani, so razvijalci knjižnice React izdali že dve večji izdaji [32]. V obeh izdajah so za namene izboljšanja zmogljivosti spremenili zasnovo knjižnice [5, 6]. To pomeni, da informacije iz omenjenih knjig niso več ažurne za trenutno izdajo knjižnice. Zato je najbolj zanesljiv in ažuren vir informacij uradna dokumentacija knjižnice [27].

Poglavje 2

Jezik Elm

Jezik Elm [10] je strogo in statično tipiziran funkcijski programski jezik, ki je namenjen izdelavi zanesljivih spletnih aplikacij. Prevaja se v jezik JavaScript, kar omogoča izvajanje v spletnih brskalnikih.

2.1 Kaj je funkcijski programski jezik?

V programskih jezikih kot so C ali Java so programi napisani kot zaporedje ukazov. Pri funkcijskih programskih jezikih pa so programi napisani kot zaporedje matematičnih funkcij in ne dopuščajo spreminja že nastavljenih vrednosti. Iz tega izhaja lastnost, da program za enake vhode vedno vrne enak izhod.

Funkcije, katerih rezultat ni odvisen od vrednosti spremenljivk zunaj funkcije in hkrati nimajo stranskih učinkov, se imenujejo čiste funkcije (ang. pure function). To pomeni, da na rezultat funkcije ne vplivajo vrednosti izven vhodnih vrednosti ter da funkcija ne povzroča stranskih učinkov (ang. side effects). Torej funkcija med svojim izvajanjem ne more implicitno spremeniti stanja v programskem okolju, bodisi delovnem spominu ali na vhodno/izhodnih napravah [13].

Ampak v programih ne gre brez stranskih učinkov. Sploh ko govorimo o spletnih aplikacijah, pričakujemo stranske učinke, na primer podatki iz

strežnika ali iteracij uporabnika. V jeziku Elm je to rešeno z Arhitekturo Elm, ki določa kako so podatki definirani, kako se posodabljajo ter kako naj se prikažejo na uporabniškem vmesniku.

2.2 O tipizaciji v programskem jeziku Elm

Pri jeziku Elm prevajalnik zahteva informacijo o podatkovnih tipih za vse vhodne parametre in za rezultat funkcije. Poleg tega nima nične vrednosti, ki je pogosto definirana kot vrednost `null`. Odsotnost vrednosti se predstavi z vrednostjo `Nothing`, ki je sestavni del tipa `Maybe`.

Pri tipu `Maybe` gre za predstavitev spremenljivke, kjer obstaja možnost, da pričakovana vrednost ne obstaja. Prevajalnik jezika Elm pričakuje, da za vrednosti tipa `Maybe` programer definira dve pogojni poti. Prva v primeru, da vrednost obstaja in je definirana kot `Just TipVrednosti`. Druga možnost pa kot `Nothing`, kadar vrednost ne obstaja.

2.3 Sklepanje o tipih

Da pa ni potrebno programerju ročno navajati tipov za vse vrednosti, ima prevajalnik jezika Elm mehanizem za sklepanje o tipih (ang. type inference) [11]. To pomeni, da zna prevajalnik sam ugotoviti, kakšni so tipi vhodnih argumentov ter kakšni so rezultati funkcije.

Da ta mehanizem deluje zagotavlja prevajalnik, ki spremlja tipe podatkov po celotnem programu. V primerih, ko pride do nekonsistentnosti med tipi, prevajalnik od programerja zahteva, da te probleme razreši. V nasprotnem primeru se program ne prevede.

Prevajalnik jezika Elm to idejo povzema iz prevajalnika za programski jezik Haskell. Podobne lastnosti opazimo tudi v drugih programskih jezikih, kot primer v jeziku C++ (od verzije C++11 naprej) kot ključna beseda `auto` ali v jeziku C# kot ključna beseda `var`.

2.4 Prevajalnik jezika Elm

Opisani mehanizmi skupaj omogočajo, da prevajalnik jezika Elm že med prevajanjem zazna napake ter od programerja zahteva, da te probleme odpravi. Tako zagotavlja, da med delovanjem ne pride do nepričakovanih napak.

Ena glavnih lastnosti prevajalnika jezika Elm je, da je razvoj prevajalnika usmerjen k temu, da skuša biti programerju kar se da v pomoč z namigi o napakah v kodi, kot v primeru na sliki 2.1.

```
Function ‘program‘ is expecting the 2nd argument to be:
    { ..., subscriptions : ... }
But it is:
    { ..., subcsriptions : ... }
Hint: The record fields do not match up.
Maybe you made one of these typos?
    subscriptions <-> subcsriptions
```

Slika 2.1: Primer namiga, ki ga poda prevajalnik jezika Elm ob zaznani napaki.

Problem v kodi je, da se je programer zmotil in napisal `subcsriptions`, prevajalnik pa pričakuje ime `subscriptions`. Prevajalnik je v tem primeru opisal zaznano napako in tudi predlagal možnost za popravek. V orodjih za urejanje programske kode obstajajo tudi vtičniki, ki omogočajo avtomatsko popravilo tovrstnih napak. Ta orodja delujejo s pomočjo predlogov, ki jih poda prevajalnik.

Prevajalnik tako omogoča programerju, da tipkarske napake in druge manjše napake odpravi hitro. Zato programerju pogosto ni potrebno za rešitev brskati po dokumentaciji ali drugje na spletu. To programerja razbremeni in omogoča večjo produktivnost med delom.

2.5 Arhitektura Elm

Arhitektura Elm (ang. The Elm Architecture) je ena od glavnih lastnosti jezika Elm. Definira vzorec toka podatkov skozi aplikacijo. Ker jezik Elm

ne dovoljuje neposrednega spreminjanja vrednosti, se uporabi funkcija, ki vrne novo stanje aplikacije. Arhitektura Elm za ta namen definira tri dele aplikacije: „Model“, „Update“ in „View“.

2.5.1 Model

Sestavljena spremenljivka `model` hrani trenutno stanje podatkov v aplikaciji. V vsaki ne-trivialni aplikaciji so vrednosti strukturirane v obliki, ki spominja na drevesno strukturo. Na vrhnjem nivoju so podatki za določene segmente aplikacije, navadno ena stran ali skupek povezanih strani. Znotraj vrednosti na vrhnjem nivoju so dodatne sestavljene spremenljivke, katere hranijo vrednosti za komponente znotraj segmentov.

```
type alias Model =
    { route : Route
    , posts : WebData (List Post)
    , sessionUser : Maybe Session
    , loginData : LoginModel
    }
```

Slika 2.2: Primer sestavljenega tipa za model.

Na sliki 2.2 je prikazan primer sestavljenega tipa za model. Tip modela je za konsistentnost in lažje delo definiran z drugim imenom z uporabo ključne besedne zveze `type alias`. V primeru je `loginData` vrednost, kjer se hranijo dodatni podatki, opisani s tipom `LoginModel` na sliki 2.3.

```
type alias LoginModel =
    { username : String
    , password : String
    , rememberMe : Bool
    , errors : List String
    }
```

Slika 2.3: Primer sestavljenega tipa za model.

2.5.2 Update

Funkcija `update` skrbi za posodobitev stanja aplikacije. Kot argument prejme sporočilo oziroma izvedeno akcijo ter trenutni model, vrne pa posodobljeno različico vrednosti za model. Prevajalnik pričakuje, da funkcija vedno vrne novo vrednost za model, ki je enakega tipa. To mora programer zagotoviti tudi, če za dano akcijo ne želi spremeniti stanja v modelu. S tem prevajalnik zagotavlja predvidljivost delovanja.

Ker funkcija prejme celotno vrednost za model, je ob vsaki akciji dostopno celotno stanje aplikacije. To pomeni, da lahko vsaka akcija spreminja katerikoli del aplikacije. Tak način omogoča deljenje podatkov med deli aplikacije, brez tesne povezanosti različnih delov aplikacije ali drugih pristopov za komunikacijo med različnimi segmenti aplikacije.

```
update : Msg -> Model -> ( Model, Cmd Msg )
```

Slika 2.4: Definicija funkcije `update`.

Kot je prikazano na sliki 2.4, funkcija `update` prejme dva argumenta, prvi tipa `Msg` in drugi tipa `Model`. Vrne n-terko vrednosti novega modela ter naslednjega ukaza s sporočilom. Tip za ukaz `Cmd Msg` se preslika v naslednji klic funkcije `update`, za kar poskrbi standardna knjižnica znotraj jezika Elm. Posebna vrednost ukaza je `Cmd.none`, ki opisuje, da ni naslednjega sporočila. Tako funkcija `update` omogoča rekurzivne klice in posodabljanje stanja aplikacije z zaporedjem akcij.

Na sliki 2.5 je primer tipa `Msg` ter funkcije `update`. Prevajalnik jezika Elm zahteva, da za prejeto vrednost `msg` obravnavamo vse možne vrednosti, saj se aplikacija v nasprotnem primeru ne bi prevedla.

V tem primeru sporočilo `NavigateTo` opisuje akcijo, ki sproži navigacijo v aplikaciji. Za vrednost `msg = NavigateTo` funkcija update vrne nespremenjeno vrednost za model ter naslednji ukaz `Navigation.newUrl route`. Ta ukaz interno sproži akcijo o spremembi lokacije, ki je opisana z vrednostjo `OnLocationChange`. Za to akcijo se v vrnjeni vrednosti za model spremeni

vrednost `route` na prejeto vrednost ob akciji `OnLocationChange`. Vrednost ukaza `Cmd.none` pa pomeni, da ni naslednjega ukaza funkcijo `update`.

```
type Msg
    = NavigateTo String
    | OnLocationChange Route
    | OnFetchPosts (WebData (List Post))
update : Msg -> Model -> ( Model, Cmd Msg )
update msg model =
    case msg of
        NavigateTo route ->
            ( model, Navigation.newUrl route )

        OnLocationChange route ->
            ( { model | route = route }, Cmd.none )

        OnFetchPosts response ->
            ( { model | posts = response }, Cmd.none )
```

Slika 2.5: Primer tipa `Msg` ter funkcije `update`.

2.5.3 View

Funkcija `view` je namenjena prikazu stanja aplikacije na uporabniškem vmesniku. Kot je razvidno na sliki 2.6, funkcija kot argument prejme stanje aplikacije, vrne pa virtualni objektni model dokumenta. Med izvajanjem v brskalniku se virtualni objektni model dokumenta pretvori v DOM elemente brskalnika, saj je to edina oblika, ki jo spletni brskalnik zna prikazati. Za to poskrbi del kode, ki jo prevajalnik jezika Elm doda v prevedeno kodo.

```
view : Model -> Html Msg
view model =
    div []
        [ text ("Hi " ++ model.loginData.username) ]
```

Slika 2.6: Definicija funkcije `view`.

2.5.4 Način in potek delovanja

Aplikacija v jeziku Elm se začne s klicem funkcije glavnega programa `main`. Ker se v aplikaciji pogosto uporablja modul za upravljanje navigacije `Navigation`, je potrebno uporabiti tudi glavni program iz tega modula. To se stori s klicem `Navigation.programWithFlags`, kot je prikazano na sliki 2.7.

```
main : Program Value Model Msg
main =
    Navigation.programWithFlags (Route.parseLocation >> OnLocationChange)
        { init = init
        , view = view
        , update = update
        , subscriptions = subscriptions
        }
```

Slika 2.7: Klic glavnega programa iz modula `Navigation`.

1. Glavni program na začetku nastavi privzeto vrednost za spremenljivko `model`. Uporabi vrednost, ki jo vrne podana funkcija `init`.

2. Kliče se funkcija `view`, ki ustvari predstavitev uporabniškega vmesnika za trenutno, torej začetno stanje aplikacije. Na uporabniškem vmesniku se navadno pojavijo uporabniške akcije, kot na primer klik na gumb. Te akcije sprožijo klic funkcije `update`.

3. Funkcija `update` za prejeto akcijo vrne posodobljeno stanje. Ko se sekvenca ukazov `Cmd` za sporočila zaključi, se ponovno kliče funkcija `view`. Ta prikaže posodobljeno vrednost modela na enak način, kot v točki 2. Ko se zgodi nov dogodek, se opisani cikel ponovi.

2.5.5 Vplivi v skupnosti razvijalcev spletnih aplikacij

Arhitektura Elm je, kot vzorec načrtovanja toka podatkov, med programerji vzbudila veliko zanimanja. Ideja je bila med drugim tudi navdih za razvoj knjižnice Redux.js [31].

Omembe vreden je tudi vpliv v knjižnici Vuex [39]. To je pomožna knjižnica za upravljanja stanja v ogrodju Vue.js. Kot že opisano v uvodu in predstavljeno na sliki 1.1 je ogrodje Vue.js eno izmed bolj popularnih ogrodij za razvoj spletnih aplikacij v letu 2017.

2.6 Uporaba funkcij v brskalniku preko jezika JavaScript

Prevedena programska koda Elm že vsebuje funkcije, ki jih jezik kot ogrodje potrebuje za delo s standardnimi knjižnicami znotraj brskalnika. Primeri uporabe standardnih knjižnic:

- komunikacija s strežnikom,
- prestrezanje dogodkov, kot na primer klik na gumb ali spreminjanje vrednosti v vnosnem polju,
- pretvarjanje niza JSON v objekt jezika JavaScript oziroma obratno,
- delo z datumi in
- spreminjanje stanja prikazanega uporabniškega vmesnika.

Za uporabo funkcij jezika JavaScript, ki jih jezik Elm ne podpira, je programer primoran sam napisati vmesnik med funkcijami v programski kodi JavaScript ter funkcijo, ki se kliče v jeziku Elm. Za oznako funkcij, ki delujejo kot vmesnik med jezikom Elm ter JavaScript, se uporabi ključna beseda `port`.

Primer nepodprte funkcionalnosti je objekt za obstojno shrambo podatkov v brskalniku `localStorage`. Pogost primer so tudi zunanje knjižnice, ki so napisane v jeziku JavaScript in nimajo alternative v jeziku Elm.

Pri tem delu mora biti programer pazljiv, saj dela z dvema različnima jezikoma. Poleg tega pa pri jeziku JavaScript ni preverjanja tipov ob času prevajanja ter zaščite pred napačno uporabo podatkovnih tipov, tako kot to zagotavlja prevajalnik pri jeziku Elm.

2.6.1 Pošiljanje podatkov iz funkcije jezika Elm v funkcijo jezika JavaScript

Za pošiljanje vrednosti iz jezika Elm v programsko kodo funkcije jezika JavaScript programer definira funkcijo z uporabo ključna besede `port`, kot prikazano v primeru na sliki 2.8.

```
port imeIzhodneFunkcije : a -> Cmd msg
```

Slika 2.8: Definicija funkcije z uporabo ključne besede `port`.

V deklaraciji tipa `a -> Cmd msg` vrednost `a` predstavlja tip, ki je podprt s strani ključne besede `port`. Prevajalnik ne dovoljuje poljubnih tipov, zato sintaksa na sliki 2.8 vrne napako na sliki 2.9.

```
Port ‘imeIzhodneFunkcije‘ is trying to communicate an unsupported type.
    port imeIzhodneFunkcije : a -> Cmd msg
    ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
The specific unsupported type is the following free type variable:
    a
The types of values that can flow through in and out of Elm include:
    Ints, Floats, Bools, Strings, Maybes, Lists, Arrays, Tuples, Json.
        Values, and concrete records.
```

Slika 2.9: Napaka prevajalnika v primeru uporabe prostega tipa za vhodno vrednost.

Napaka na sliki 2.9 opisuje, da je funkcija `imeIzhodneFunkcije` definirala izhodni tip kot prost tip `a`, ključna beseda `port` pa pričakuje enega od naštetih tipov. Torej je potrebno navesti konkretni podatkovni tip, kot v primeru na sliki 2.10, kjer se kot vhodni tip določi niz znakov.

```
port imeIzhodneFunkcije : String -> Cmd msg
```

Slika 2.10: Primer funkcije z uporabo ključne besede `port`, ki prejme vrednost tipa `String`.

Popravljena funkcija na sliki 2.10 prejme argument tipa `String` oziroma niz znakov, vrne pa tip `Cmd msg`. Kot predstavljeno v opisu funkcije `update`, je tip `Cmd msg` podatkovni tip za opis ukaza znotraj aplikacije Elm. To pomeni, da klic zunanje funkcije ni navaden klic, kjer se takoj vrne rezultat. Vrne se le akcija klica funkcije, prejem dejanskega rezultata pa je potrebno izvesti na poseben način, ki je opisan v naslednji sekciji.

Pomembno je tudi, da prevajalnik pričakuje točno določeno definicijo funkcije. Funkcija vedno prejme le en argument, vrne pa ukaz tipa `Cmd msg`. Če želimo poslati več podatkov, jih lahko pošljemo kot seznam, objekt ali pa kot `Json.Value`, ki predstavlja objekt jezika JavaScript.

Funkcijo kličemo enako kot druge funkcije znotraj jezika Elm. Ključna beseda `port` pa pove, da se bodo argumenti prenesli v funkcijo v kodi JavaScript, torej se za klicano funkcijo ne izvede koda jezika Elm. Tu prevajalnik jezika Elm izgubi nadzor nad programsko kodo, zato je naloga programerja, da zagotovi pravilnost delovanja kode JavaScript.

Za prejem klica funkcije `port` v programski kodi JavaScript je potrebno definirati še povratno funkcijo, ki se kliče ob klicu funkcije znotraj jezika Elm, kot je prikazano na primeru na sliki 2.11.

```
// Deklaracija zacetka programa, ‘Main‘ predstavlja modul glavnega
    programa, ki se nahaja znotraj datoteke Main.elm.
var app = Elm.Main.fullscreen();
app.ports.imeIzhodneFunkcije.subscribe(function(data) {
  // Argument ‘data‘ predstavlja podatke, ki so bili podani kot argument
      klica funkcije v jeziku Elm.
});
```

Slika 2.11: Funkcija v kodi JavaScript, ki se kliče ob klicu funkcije, deklarirane s ključno besedo `port`.

Primer na sliki 2.11 se začne s klicem funkcije `Elm.Main.fullscreen()`. Ta zažene glavni program aplikacije Elm, vrne pa instanco aplikacije. Instanca vsebuje tudi lastnost, ki hrani funkcije `port`. Imena funkcij na objektu `ports` se ujemajo z imeni za funkcije `port` v programski kodi Elm. S klicem funkcije `subscribe` dodamo novo povratno funkcijo. Ob vsakem klicu

funkcije `port` znotraj jezika Elm se kličejo vse povratne funkcije, ki so bile dodane v jeziku JavaScript.

Pri delu z vrednostjo `data` se lahko zanašamo, da bo vrednost enakega tipa, kot je določeno v jeziku Elm. Potrebno pa je paziti na napake v kodi JavaScript, saj le-te lahko povzročijo prenehanje delovanja aplikacije.

2.6.2 Prenos vrednosti iz JavaScripta v Elm

```
port imeVhodneFunkcije : (String -> msg) -> Sub msg
```

Slika 2.12: Deklaracija vhodne funkcije preko ključne besede `port`.

V primeru vhodne funkcije na sliki 2.12 je v opisu funkcije določeno, da prejme vrednost tipa niz znakov. Tako kot pri izhodnih funkcijah, tudi tu prevajalnik pričakuje točno določen tip funkcije.

Vhodni tip funkcije je vedno funkcija z deklaracijo `(a -> msg)`. To je funkcija, ki določen tip preslika v prosti tip sporočila, označen kot `msg`. V primeru na sliki 2.12 se pričakuje funkcija, ki tip `String` preslika v `msg`.

Funkcija `port` vrne tip `Sub msg`, ki opisuje naročnino na določeno sporočilo. Tip sporočila, ki ga vrne funkcija za preslikovanje, se ujema s tipom sporočila v rezultatu.

Za ustrezno posredovanje dogodkov iz jezika JavaScript je potrebna še funkcija `subscriptions`, kot v primeru na sliki 2.13. Koda v primeru določa, da se vsak dogodek funkcije `imeVhodneFunkcije` preslika v sporočilo tipa `OnInput`. To sporočilo se posreduje v funkcijo `update`, kjer se ustrezno obravnava. Funkcija `subscriptions` se poda ob klicu za glavni program, kot je prikazano v primeru na sliki 2.7.

V jeziku JavaScript je vhodna funkcija dostopna na enak način kot v primeru za izhodno funkcijo na sliki 2.11. Za pošiljanje podatkov se uporabi funkcija `send`, kot je prikazano v primeru na sliki 2.14.

```
subscriptions : Model -> Sub Msg
subscriptions model =
    Sub.batch
        [ Sub.map OnInput imeVhodneFunkcije
        -- ...
        ]
```

Slika 2.13: Deklaracija funkcije za preslikavo naročnin v sporočila.

```
var app = Elm.Main.fullscreen();
app.ports.imeVhodneFunkcije.send(value);
```

Slika 2.14: Koda JavaScript, ki pošlje vrednost v kodo Elm.

2.7 Orodja za pomoč pri razvoju

2.7.1 `elm-make`

Orodje `elm-make` vsebuje prevajalnik, ki preveri vso Elm kodo ter skrbi za prevajanje aplikacije iz jezika Elm v kodo JavaScript. Prevajalnik pred dejanskim prevajanjem izvorne kode preveri pravilnost kode ter v primeru zaznanih napak le-te javi programerju. Omogoča tudi opozorila za dele kode, kjer je zaznana priložnost za izboljšavo. Pogosto so to primeri, kjer za funkcijo ali vrednosti manjka oznaka podatkovnega tipa vrednosti. Opozorila se vklopi z uporabo stikala `--warn`.

2.7.2 `elm-package`

Orodje `elm-package` pomaga pri upravljanju zunanjih knjižnic znotraj projekta v jeziku Elm. Skrbi za dodajanje in ustrezno posodabljanje uporabljenih knjižnic.

Stroga statična tipizacija jezika Elm omogoča zaznavanje sprememb pri javnem delu programskega vmesnika. S tem znanjem lahko orodje `elm-package` zagotavlja semantično označevanje različic [35]. S tem prepreči primere, kjer se pri posodobitvi knjižnice pojavi nepričakovana napaka v aplikaciji.

Za prenos novejše različice knjižnice, kjer orodje `elm-package` sklepa, da lahko predstavlja probleme, mora programer eksplicitno spremeniti omejitve za oznako različice ter ponovno pognati ukaz za prenos knjižnic.

2.7.3 `elm-repl`

Orodje `elm-repl` ponuja ukazno lupino, ki zajema in interpretira ukaze v jeziku Elm ter prikaže rezultat danega izraza. Orodje se lahko uporabi za hitro preverjanje delovanja določenega dela kode.

2.7.4 `elm-reactor`

Orodje `elm-reactor` služi kot razvojno orodje, ki programerju omogoča ogled prevedenih datotek jezika Elm iz spletnega brskalnika. Združuje prevajalnik za jezik Elm ter spletni strežnik, ki prevedeno kodo streže kot HTML datoteko, ki jo lahko brskalnik prenese in interpretira ter tako v uporabniški vmesnik brskalnika naloži aplikacijo.

2.7.5 `elm-format`

Orodje `elm-format` skrbi za ustrezno stilsko oblikovanje izvorne kode. S stikalom `--validate` omogoča preverjanje, če je koda ustrezno oblikovana, brez posodabljanja izvorne kode.

Za razliko od ostalih, to orodje ob času pisanja diplomske naloge ni uradno podprto. Vendar je v skupnosti razvijalcev zelo razširjeno, saj poenostavi konsistentno oblikovanje izvorne kode.

Poglavje 3

Spletna aplikacija

Za namene primerjave jezika Elm s pristopom razvoja z uporabo jezika JavaScript, skupaj s knjižnico React, je izdelana ista spletna aplikacija v obeh jezikih[1].

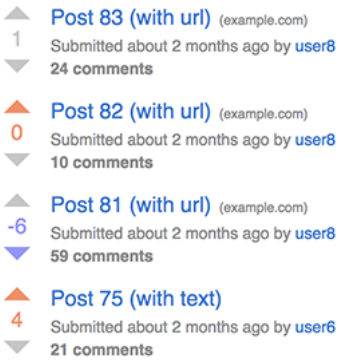
3.1 Predstavitev

Ideja izdelane spletne aplikacije je prikaz najbolj pogostih operacij, ki se pojavijo v tipični spletni aplikaciji. Te operacije so navigacija po aplikaciji, prenos in prikaz podatkov iz spletnega strežnika, dodajanje novih podatkov, urejanje ter brisanje, pogojno stilsko oblikovanje uporabniškega vmesnika, prijava uporabnika, omejevanje dostopa do določenih delov aplikacije le prijavljenim uporabnikom ter uporaba komponent aplikacije na različnih delih aplikacije. Za konec pa tudi postopek za objavo kode, optimizirane za produkcijsko okolje.

Izdelana spletna aplikacija temelji na načinu delovanja spletnega portala Reddit. Vsi obiskovalci vidijo vse objave, podrobnosti in komentarje pod določeno objavo ter ocene za objav in komentarjev. Za določenega uporabnika lahko vidijo vse objave in komentarje, poleg tega pa se lahko prijavijo oziroma registrirajo. Prijavljeni uporabniki pa imajo dodatno še možnost

[1]Uporabniški vmesnik spletne aplikacije je v celoti v angleškem jeziku.

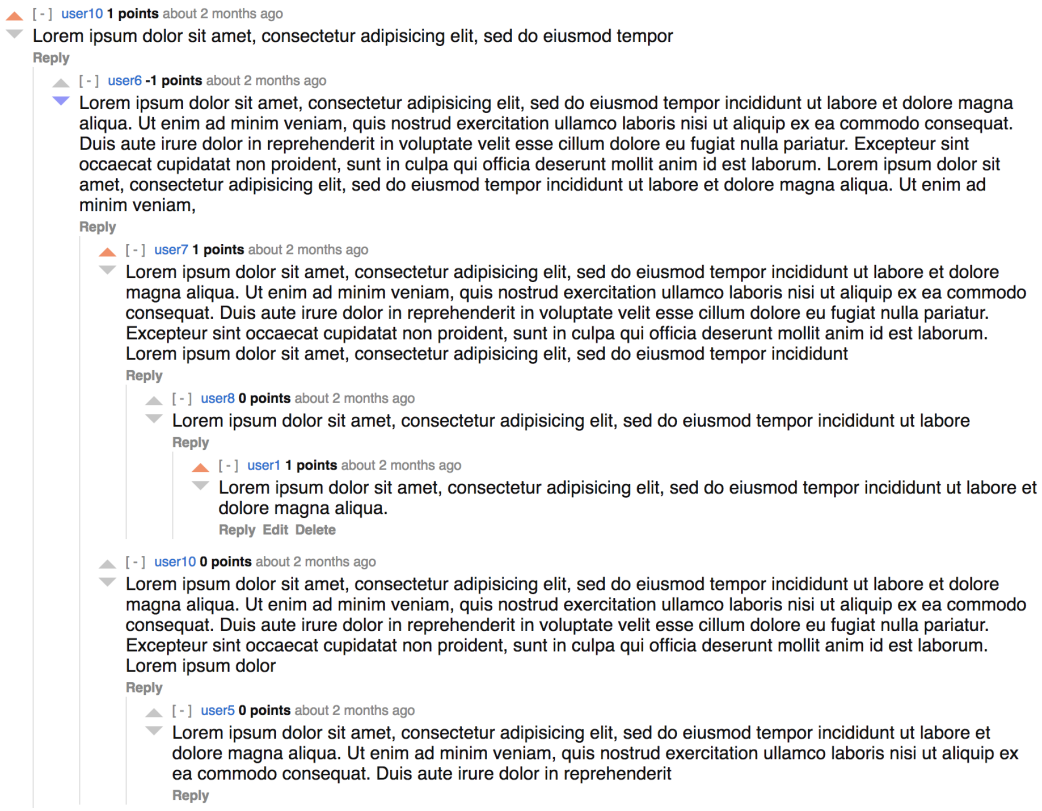
dodajanja novih objav, katere se delijo na tekstovno objavo ali pa objavo povezave na nek spletni naslov. Na vse objave lahko podajo komentar, z možnostjo gnezdenja komentarjev oziroma komentiranje komentarja. Svoje komentarje lahko tudi uredijo ali izbrišejo. Poleg tega lahko objavo ali komentar ocenijo s pozitivno ali negativno oceno, kar je tudi vizualno ponazorjeno z oranžno oziroma vijolično barvo.



Slika 3.1: Prikaz objav na seznamu vseh objav.

Za vsako objavo uporabnik vidi naslov objave, za objavo na povezavo pa tudi domeno povezave, kot je razvidno na sliki 3.1. Poleg naslova so še podatki o tem, koliko časa nazaj je bila objava dodana, število komentarjev, ime avtorja objave ter oceno objave. Ocena ima barvno ponazorjeno uporabnikovo oceno, prav tako je obarvan pripadajoči gumb za oceno. Siva barva besedila predstavlja, da uporabnik za objavo ni glasoval, oranžna barva predstavlja pozitivno, vijolična pa negativno oceno. Število komentarjev je hkrati tudi povezava na pregled komentarjev za objavo. Ime uporabnika pa služi kot povezava na profil avtorja objave.

Slika 3.2 pa predstavlja pregled komentarjev s ponazorjeno gnezdeno strukturo komentarjev. Za vsak komentar je razvidno kdo je avtor, podatek koliko časa nazaj je bil komentar dodan, skupna ocena komentarja ter vsebina komentarja. Pod komentarjem so gumbi za dodajanje odgovora na komentar. Za lasten komentar pa ima uporabnik na voljo še gumba za urejanje in brisanje. Tako kot pri objavi, tudi tukaj ime uporabnika deluje



Slika 3.2: Prikaz gnezdenih komentarjev v pregledu podrobnosti objave.

kot povezava na profil avtorja komentarja. Poleg komentarja sta še gumba za pozitivno oziroma negativno oceno, ki sta obarvana na enak način kot pri objavi. Gumb, ponazorjen z znakom `[+]` oziroma `[-]`, omogoča strnitev oziroma razširitev komentarja, skupaj z vsemi gnezdenimi komentarji.

3.2 Lastnosti spletne aplikacije

Pogled za vse uporabnike:

- navigacijska vrstica, ki je vidna na vseh straneh aplikacije, kjer ima uporabnik na voljo:
  - povezavo na domačo stran

  - za neprijavljene uporabnike povezavi na stran za registrirajo ter stran za prijavo
  - za prijavljene uporabnike se kaže besedilo „Logged in as ⟨ime trenutnega uporabnika⟩“ (ime trenutnega uporabnika je povezava na uporabnikov profil) in gumb za odjavo

- na domači strani je viden pregled vseh objav, kjer uporabnik za vsako objavo vidi:
  - naslov (če je objava tipa povezava, naslov deluje kot povezava dano povezavo, sicer povezava pelje na podroben pogled objave)
  - če je objava tipa povezava, je poleg naslova tudi ime domenskega naslova povezave
  - ime avtorja objave (hkrati kot povezava na stran s podatki o avtorju)
  - čas, ki je pretekel od objave
  - število komentarjev (hkrati služi kot povezava na podroben pogled objave)
  - skupna ocena objave
  - gumba za pozitivno in negativno oceno, ki sta onemogočena za neprijavljene uporabnike

- pregled podrobnosti objave, kjer uporabnik vidi:
  - podatke o objavi, enako kot opisano zgoraj
  - če je tekstovna objava, se pod podatki objave pokaže vsebina objave
  - seznam komentarjev, ki so prikazani v gnezdeni obliki
  - za vsak komentar se prikaže:
    * vsebina komentarja

    - ime avtorja objave (hkrati tudi povezava na stran s podatki o avtorju)
    - skupna ocena komentarja
    - čas, ki je pretekel od objave komentarja
    - gumb za strnitev oziroma razširitev drevesa komentarja
    - gumba za pozitivno in negativno oceno, ki sta onemogočena za neprijavljene uporabnike

- pregled podatkov o uporabniku:
  - naslov „User ⟨ime uporabnika⟩“
  - dva zavihka, prvi za seznam vseh objav, drugi za seznam vseh komentarjev tega uporabnika
  - tako objave kot komentarji so prikazani na enak način, kot opisano zgoraj

- stran za registrirajo:
  - uporabnik vnese uporabniško ime, geslo ter potrditev gesla
  - v primeru napake se ta prikaže uporabniku, sicer se uporabnika preusmeri na stran s prijavo s sporočilom o uspešni registraciji

- stran za prijavo:
  - uporabnik vnese uporabniško ime in geslo
  - v primeru neuspešne prijave se prikaže napaka, sicer se uporabnika preusmeri na domačo stran

Prijavljeni uporabniki imajo poleg zgoraj opisanih opcij na voljo:

- gumbi za oceno objave ter komentarjev so omogočeni, za že ocenjene objave so ustrezno obarvani
- na domači strani se pojavita gumba za dodajanje nove besedilne objave in objave s povezavo

- na strani s komentarji se pojavi vnosno polje za nov komentar
- pod vsakim komentarjem se pojavi gumb „Reply“, ki omogoča dodajanje odgovora na komentar
- za lasten komentar ima uporabnik na voljo tudi:
  - gumb „Edit“ za urejanje komentarja, kjer se pojavi enako polje kot za dodajanje komentarja, ki je vnaprej napolnjeno s trenutno vsebino komentarja
  - gumb „Delete“ za brisanje komentarja, kjer se pred brisanjem pojavi še potrditveno okno s pozivom, če uporabnik res želi izbrisati komentar

3.3 Zaledni del

Zaledni del je enak za obe izvedbi aplikacije. Služi kot vir podatkov in način izvajanja operacij nad podatki. Primer operacij so dodajanje novih objav, urejanje in brisanje komentarjev, prijava uporabnika in druge. Do zalednega dela se dostopa izključno preko spletnega aplikacijskega programskega vmesnika, kjer komunikacija poteka preko splete storitve REST. Podatki pa se prenašajo v obliki JSON sporočil.

Zaledni del je razvit z uporabo jezika Elixir [9]. Za pomoč pri razvoju spletnega aplikacijskega programskega vmesnika pa je uporabljeno še ogrodje Phoenix [25].

Jezik Elixir je, tako kot Elm, funkcijski programski jezik. Za razliko od jezika Elm pa je dinamično tipiziran, po sintaksi pa je še najbolj podoben programskem jeziku Ruby. Deluje na navideznem stroju jezika Erlang (BEAM VM) [9]. Jezik je zanimiv predvsem za porazdeljene sisteme, ki so odporni na napake in delujejo zahtevajo nizek čas zakasnitve. Za izdelavo spletnih storitev je jezik Elixir zanimiv predvsem zaradi zadnje lastnosti, saj so hitri odgovori zelo zaželeni.

Poglavje 4

Primerjava jezika Elm in razvoja z uporabo knjižnice React

Izdelani spletni aplikaciji sta vizualno povsem enaki. Edina zaznavna razlika je pri akciji za odjavo. Pri aplikaciji, izdelani v jeziku Elm, se posodobitev stanja v prikaz za neprijavljenega uporabnika zgodi hitro in skoraj nezaznavno. Aplikacija s knjižnico React pa po odjavi preskoči na prazen zaslon in prične s ponovnim nalaganjem podatkov in šele nato prikaže stanje odjavljenega uporabnika.

4.1 Način ocenjevanja

Spletni aplikaciji sta primerjani po metrikah zmogljivosti, velikost prevedene kode, številu vrstic izvorne kode ter opis izkušnje med razvojem spletne aplikacije.

4.1.1 Zmogljivost

Za ocenjevanje zmogljivosti se uporablja orodje Lighthouse [18]. Orodje deluje kot vtičnik za brskalnike ali kot orodje v ukazni vrstici. Z izdajo brskal-

nika Google Chrome 60 je na voljo kot vgrajeno razvojno orodje za analizo zmogljivosti spletnih aplikacij [41].

Orodje Lighthouse meri lastnosti spletne aplikacije, kot so čas čakanja do interaktivne vsebine (ang. first interactive), indeks hitrosti zaznavanja (ang. Perceptual Speed Index), čas prenosa izvorne kode in podatkov s strežnika in druge. Poda oceno zmogljivosti spletne aplikacije z oceno od 0 do 100, kjer višja ocena pomeni boljšo zmogljivost. Indeks hitrosti zaznavanja je vrednost, ki ponazarja kako hitro se vsebina na strani naloži in prikaže uporabniku. Nižja vrednost pomeni boljši rezultat. Poleg podane ocene orodje predlaga tudi izboljšave.

Zajeto poročilo orodja Lighthouse se lahko izvozi v obliki JSON. Le-to se lahko kasneje uvozi v pregledovalnik na uradni spletni strani [19]. V pregledovalniku je ponujena tudi opcija ustvarjanja poročila v obliki pdf.

Na dodatku A.1 je primer poročila, kjer je razviden način ocenjevanja orodja Lighthouse. Poročilo je ustvarjeno za spletno stran `https://reddit.com`, podana skupna ocena je 43.

Aplikaciji nimata svojih specifičnih optimizacij za namene izboljšanja rezultatov v poročilu. Vse izvedene optimizacije so pri obeh aplikacijah opravljene po istem postopku in z enakimi orodji. Prikazani rezultati so dosegljivi brez posebnega poudarka na optimizacijah za zmogljivost med časom razvoja.

4.1.2 Velikost prevedene kode

Za delo v spletnih brskalnikih mora biti naložena programska koda v jeziku JavaScript. Prevajalnik jezika Elm že poskrbi za pravilno in optimalno prevajanje programske kode Elm v ciljno kodo JavaScript.

Pri razvoju s knjižnico React pa se uporablja poseben zapis kode v sintaksi JSX [15], ki privzeto ne deluje v spletnih brskalnikih. Zato je potrebno orodje, ki prevede sintakso JSX v kodo JavaScript, da kodo spletni brskalniki znajo interpretirati. Za ta namen se uradno priporoča [17] prevajalno orodje Babel. Poleg sintakse JSX se v uradni primerih za knjižnico React pojavlja sintaksa novejših standardov ECMAScript. Ta sintaksa ni podprta v

nekaterih starejših različicah brskalnikov. Orodje Babel poskrbi tudi za pretvarjanje tovrstne sintakse in tako zagotovi, da je končna programska koda združljiva s širokim naborom spletnih brskalnikov.

Za namene prevajanja se, v obeh različicah aplikacije, uporablja orodje Webpack [40]. Nastavljeno je tako, da na izhod zapiše optimizirano različico prevedene programske kode.

Izvorna koda spletne aplikacije je shranjena na spletni shrambi Amazon S3 [3] ter gostovana preko Amazon CloudFront [2]. Prenos podatkov za obe aplikaciji je nastavljen kar se da optimalno, za čim boljše rezultate. Tako imata obe aplikaciji imata povsem enake skupne nastavitve, zato meritve izhajajo iz enakih predpogojev.

4.1.3 Število vrstic izvorne kode

Za štetje vrstic izvorne kode se uporablja orodje `cloc`. Orodje na ukazni vrstici omogoča štetje vrstic za podano pot na disku. Izdelano je namensko za štetje vrstic v izvorni kodi, torej prepozna prazne vrstice ter komentarje in le-te šteje ločeno od vrstic z ukazi. S tem je končni rezultat bolj točen, saj so v različnih programskih jezikih pravila za oblikovanje programske kode drugačna. Zaradi teh odstopanj bi število vrstic v datoteki lahko pripeljalo do večjih odstopanj v končnem rezultatu.

4.1.4 Izkušnja med razvojem spletne aplikacije

Kot izkušnje med razvojem spletne aplikacije štejejo čas učenja, hitrost postavitve novega projekta, problemi med razvojem ter porabljen čas za razrešitev problemov. Šteje pa tudi pomoč iz obstoječih rešitev in splošna ocena produktivnosti med delom.

Ta način ocenjevanja je subjektivne narave, saj je izkušnja odvisna od vsakega posameznika. Poleg ocene izkušnje so podani tudi argumenti za oceno. Vendar pa tovrstna ocena ni ponovljiva po znanstveni metodi, zato je ne moremo šteti kot zanesljiv del ocene pri primerjanju.

4.2 Rezultati za knjižnico React

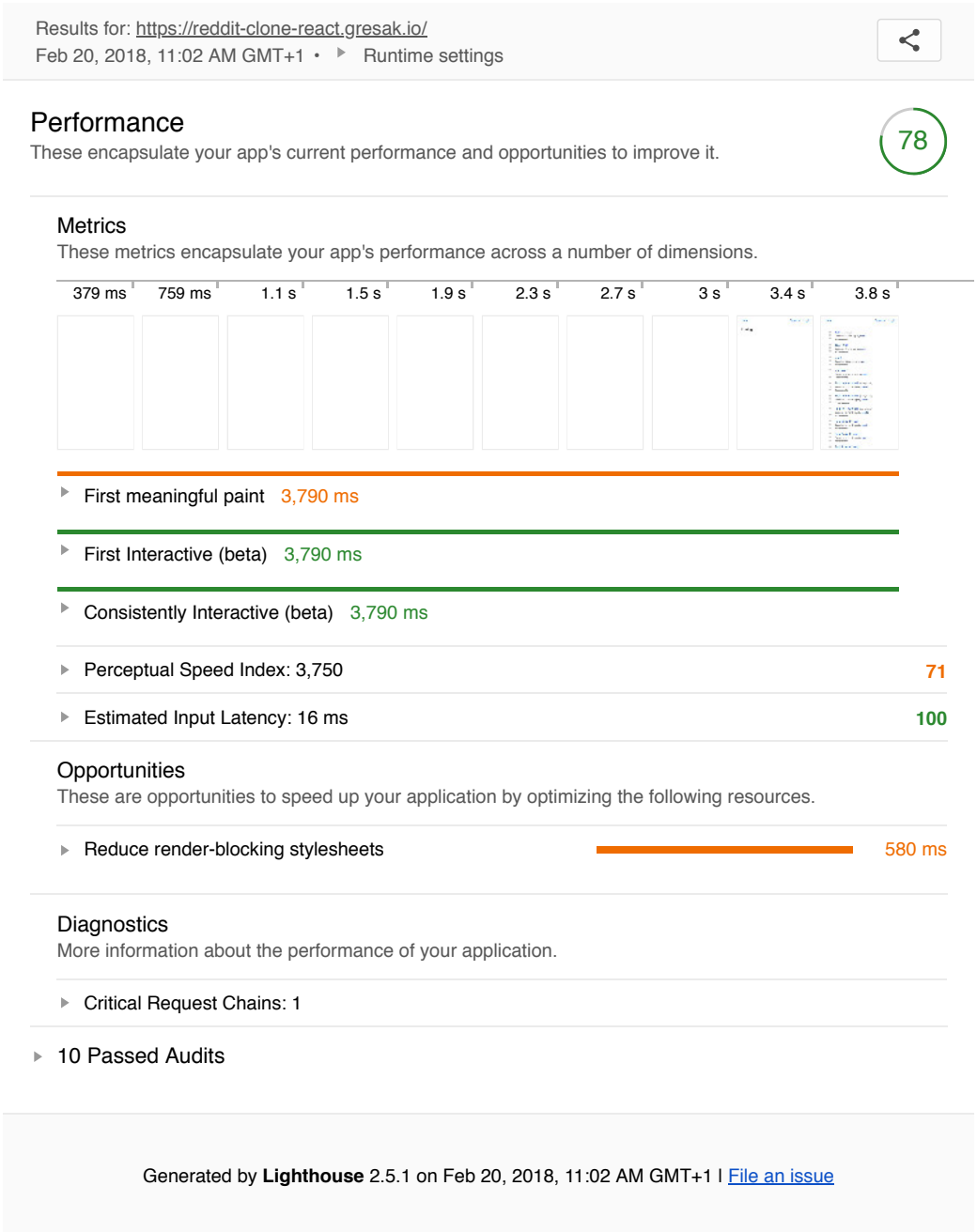
4.2.1 Zmogljivost

Iz poročila na sliki 4.1 je razvidno, da je orodje Lighthouse ocenilo zmogljivost spletne aplikacije v jeziku JavaScript z uporabo knjižnice React z 78 točkami. Za aplikacijo v knjižnici React je vrednost indeks hitrosti zaznavanja 3750, podana ocena za pa je 71.

Ocenjen čas, ko mora uporabnik čakati na prikaz vsebine (ang. first meaningful paint) in čas do interaktivne vsebine je 3790ms. To pomeni, da mora uporabnik čakati skoraj 4 sekunde, preden se prikaže vsebina. V tem času aplikacija čaka 2270ms za prenos JavaScript kode ter podatkov preko omrežja. Šele za tem brskalnik dobi zahtevane datoteke za prikaz spletne aplikacije. Te informacije so razvidne v razširjenem poročilu v dodatku A.2.

4.2.2 Velikost prevedene kode

Velikost prevedene kode se izmeri kot velikost izhodne datoteke za prevajanje aplikacije z nastavitvami za produkcijsko izdajo. Poleg tega se za namene hitrejšega prenosa preko omrežja vsebino stisne z algoritmom gzip, ki je podprt v vseh modernih spletnih brskalnikih [8]. Osnovna velikost je 760KB, stisnjena pa 196KB. Torej se velikost kode za prenos zniža za približno 74%. Ta razlika precej izboljša ocene orodja Lighthouse o zaznani hitrosti spletne strani. Vendar pa je tudi 196KB še vedno precej velika datoteka, kar se opazi na povečanem času prenosa po omrežju in za tem še na času interpretiranja programske kode. Opazi se tudi, da je bilo pri delu s knjižnico React potrebnih še nekaj dodatnih knjižnic, ki dodajo svojo težo na končno velikost prevedene kode.



Slika 4.1: Poročilo o zmogljivosti spletne aplikacije izdelane s knjižnico React.

4.2.3 Število vrstic izvorne kode

```
-------------------------------------------------------------------------------
Language          files         blank       comment          code
-------------------------------------------------------------------------------
JavaScript           37           363           111          1997
HTML                  1             1             0            11
-------------------------------------------------------------------------------
SUM:                 38           364           111          2008
-------------------------------------------------------------------------------
```

Slika 4.2: Statistika o številu vrstic v projektu s knjižnico React.

Iz podatkov na sliki 4.2 o številu vrstic kode ima projekt v jeziku JavaScript s knjižnico React 1997 vrstic programske kode JavaScript v 37 datotekah. Ker so stili CSS za aplikacijo napisani znotraj JavaScript kode, je tudi število vrstic za stile všteto med številom vrstic kode JavaScript. Poleg kode JavaScript je še ena datoteka HTML z 11 vrsticami, kjer je definirana osnovna označevalna struktura aplikacije skupaj z metapodatki za pravilen prikaz na mobilnih napravah.

4.2.4 Opis izkušnje med razvojem spletne aplikacije

Utemeljitev za izbiro knjižnice React

Prvi zaplet se je pojavil že pred začetkom razvoja, pri izbiri ogrodja. Obstaja veliko alternativ ter seveda nasprotujočih si mnenj, kar oteži izbiro. Z zbranimi statistikami je knjižnica React izbrana zaradi največjega nivoja prepoznavnosti in uporabe. Ker je le knjižnica za pomoč pri ustvarjanju uporabniških vmesnikov in ne celotno ogrodje, je učenje veliko bolj preprosto.

Knjižnice, ki dopolnjujejo razvoj s knjižnico React

Vendar pa se zaplete pri bolj kompleksnih primerih uporabe, kot je izdelana aplikacija. Pojavi se problem upravljanja stanja v aplikaciji. Knjižnica React sicer rešuje problem z upravljanjem stanja na nivoju posamezne komponente,

vendar pa se v večjih aplikacijah pogosto pojavi potreba po deljenju stanja med različnimi deli aplikacije, kjer ta pristop postane težko obvladljiv.

Pogosto uporabljeni rešitvi za upravljanje stanja sta knjižnici Redux [30] ali MobX [21]. Odločitev ponovno sledi večji popularnosti, ki jo ima knjižnica Redux. Knjižnica Redux zahteva nespremenljivo strukturo podatkov, zato se za te namene uporablja knjižnica seamless-immutable [34], ki omogoča lažje delo z nespremenljivo objektno strukturo v jeziku JavaScript.

Za delo s podatki v jeziku JavaScript se pogosto uporablja zunanja knjižnica, kot na primer lodash [20]. Ta ponuja širok nabor pogosto potrebnih funkcij pri delu s podatki, ki olajšajo delo. Za olajšano delo z datumi pa je dodana še knjižnica Moment.js [22].

Poleg tega pa je za navigacijo po spletni aplikaciji potrebna knjižnica react-router [29]. To je pomožna knjižnica, ki ponuja komponente za prikaz navigacijskih elementov v brskalniku. Ponuja tudi dodatek, ki poveže akcije navigacije po spletni aplikaciji s shrambo stanja v knjižnici Redux.

Za dodajanje stilov CSS se uporablja knjižnica styled-components [36]. Tu je izbira osnovana na želji po testiranju nove tehnologije in novega pristopa. Sicer pa knjižnica React prav tako omogoča uporabo standardnih stilov CSS ali stilov preko predprocesorja, kot na primer Sass [33].

Za namene podpiranja delovanja v širšem naboru brskalnikov se uporabljata knjižnici babel-polyfill [4] ter whatwg-fetch [42]. Omenjeni knjižnici v brskalniku nadomestita manjkajoče funkcionalnosti iz novejšega standarda ECMAScript s kodo, ki simulira funkcionalnost, opisano v standardu.

Skupno je za namene dela s knjižnico dodanih kar 18 modulov preko orodja za upravljanje odvisnosti npm [24], kar nekoliko pojasni velikost prevedene kode.

Razvojno okolje in preverjanje za napake

Za razvojno okolje ni uradno določenega načina. Obstajajo le priporočilo [1] za uporabo začetnega kompleta (ang. starter kit), ki že vključuje eno od najpogosteje uporabljenih orodij [23], Webpack [40].

Vendar pa privzete nastavitve definirajo le najbolj osnovna orodja za razvoj, zato je bilo potrebno dodati še nastavitve za produkcijsko izdajo. Za vzpostavitev razvojnega okolja z orodjem Webpack je dodanih 15 modulov preko orodja za upravljanje odvisnosti npm [24].

Pri delu z jezikom JavaScript se lahko hitro zgodi, da v izvorni kodi ostane napaka, ki je zaznana šele med delovanjem. Čeprav ni nujno potrebno, se za namene zaznavanja napak in hkrati tudi za namene stilskega oblikovanja kode priporoča uporaba orodja ESLint [12]. To orodje preverja izvorno kodo ter opozarja na dele, za katere predvideva, da lahko povzročajo napake in hkrati predlaga izboljšave. Kljub uporabi tega orodja je, zaradi dinamične tipizacije jezika JavaScript, še vedno težko zaznati velik del napak. Torej je še vseeno potrebno obsežno preveriti delovanje kode med delovanjem. Poleg orodja ESLint se uporablja še orodje Prettier [26], ki omogoča avtomatsko oblikovanje izvorne kode glede na vnaprej določene standarde. To zagotavlja bolj konsistentno oblikovanost izvorne kode. Za te namene je dodanih 9 modulov preko orodja npm [24].

Podatki o odvisnostih v projektu

Skupno je v projekt odvisen od 42 modulov, ki zavzamejo 606MB prostora na disku. Pomembno je tudi dejstvo, da je razvoj projekta po najboljših močeh sledi priporočilom skupnosti in ne vključuje odvisnosti, ki niso zares potrebne. Torej je ta projekt precej podoben resničnemu projektu, kjer se razvijalci trudijo ustvariti čim bolj optimalen projekt.

Tako veliko število odvisnosti odpre veliko možnosti za probleme v prihodnosti. To pa zato, ker programer nima neposrednega nadzora nad razvojem modulov, od katerih je projekt odvisen, kar lahko povzroči probleme pri posodabljanju. Zaradi vseh odvisnosti mora razvijalec že na začetku sprejeti veliko odločitev. To pomeni, da porabi veliko časa za raziskavo, primerjavo alternativ ter vpeljavo izbrane rešitve v projekt.

Delo s knjižnico React

Učenje knjižnice React in tudi sintakse JSX je s predznanjem jezika JavaScript ter označevalnega jezika HTML zelo preprosto. Uradna dokumentacija [27] dobro razloži osnovne koncepte na primerih, zato za učenje osnov ni potrebnih ostalih virov.

Ker je jezik JavaScript dinamično tipiziran, je sklepanje o tipih in zaznavanje napačnega podatkovnega tipa težko in v nekaterih primerih celo neizvedljivo. Zato se je potrebno zanašati na preverjanje tipov med izvajanjem.

Knjižnica React omogoča preverjanje tipov podanih atributov na nivoju komponente preko lastnosti `propTypes` [38]. Podatkovni tipi se preverjajo med izvajanjem in v primeru neujemanja javijo napako v JavaScript konzoli.

Hkrati pa specifikacija o pričakovanih podatkovnih tipih omogoča lažje delo s posredovanimi vrednostmi v komponenti, saj vsak razvijalec lahko prebere ta del kode in lahko sklepa o kontekstu uporabljenih spremenljivk.

Delo s knjižnico Redux

Knjižnica Redux skrbi za hranjenje stanja aplikacije v enem samem globalnem, nespremenljivem objektu. Ideja, ki jo knjižnica implementira je povzeta po ideji arhitekture Elm [31].

Delo s knjižnico je preprosto. V komponenti sprožimo akcijo, ki se posreduje v reduktor (ang. reducer). Reduktor je funkcija, ki prejme trenutno stanje aplikacije ter podatke o akciji in pričakuje se, da vedno vrne pa novo stanje. Ko se stanje posodobi, pomožna knjižnica react-redux [28] poskrbi, da se uporabniški vmesnik ustrezno posodobi.

Izkušnja razvijalca

Po začetnem delu z nastavitvijo razvojnega okolja je izkušnja dela s knjižnico React prijetna, saj omogoča produktivnost in hitro odkrivanje ter odpravljanje napak. Zaradi velike popularnosti obstaja širok nabor knjižnic, ki rešujejo

določene pogoste probleme ter pohitrijo delo.

Še vedno pa se občuti hibe jezika JavaScript, ki postajajo bolj očitne s povečevanjem projekta. Pri večjih projektih delo zavira dinamična tipizacija, saj je potrebno za del izvorne kode ugotoviti kakšne so pričakovane vrednosti, da ne pride do napake. Dinamična tipizacija pa ni absolutno slaba lastnost, saj v nekaterih primerih omogoča hitrejšo rešitev. Jezik JavaScript pa dodatno omogoča hitre rešitve, saj se lahko tip vrednosti poljubno spreminja in se pri nekaterih operacijah tip prisili, da se pretvori v drug tip (ang. type coercion) [16]. Vendar pa je naloga razvijalca, da se odloči za take rešitve, saj lahko povzročijo precej preglavic.

4.3 Rezultati za jezik Elm

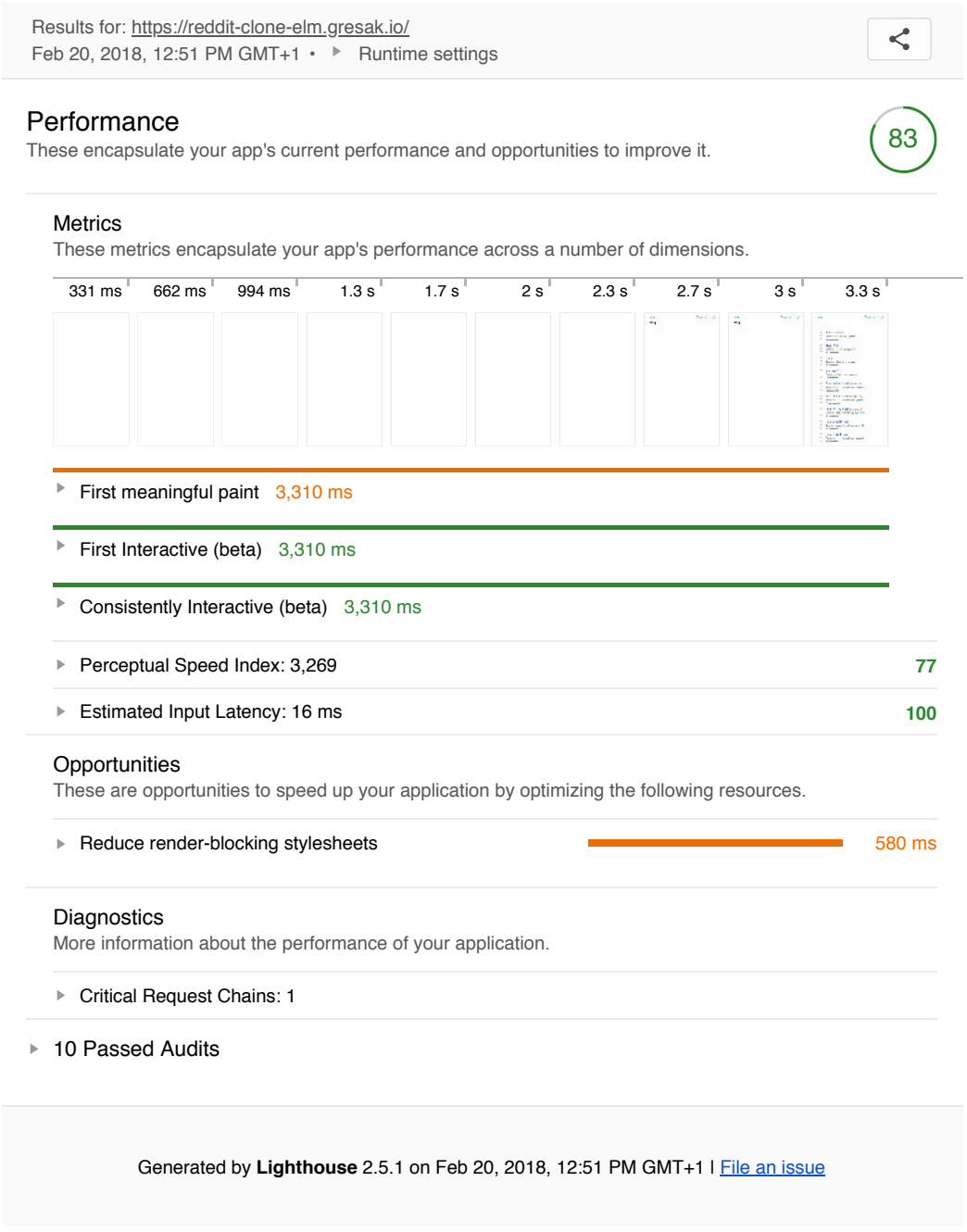
4.3.1 Zmogljivost

Iz poročila na sliki 4.3 je razvidno, da je orodje Lighthouse ocenilo zmogljivost spletne aplikacije v jeziku Elm nekoliko bolje kot prejšnjo aplikacijo, s 83 točkami. Indeks hitrosti zaznavanja je 3269, ocena indeksa pa je s 77 od 100 točk.

Razlog za boljšo oceno je nižji čas, ko mora uporabnik čakati na prikaz vsebine, pri tej aplikaciji je izmerjen čas 3310ms. Krajši čas je predvsem zasluga nižje velikosti datoteke, kar zagotovi prevajalnik jezika Elm. Celoten čas prenosa zahtevanih datotek je 1775ms, kot je razvidno v razširjenem poročilu v dodatku A.3.

4.3.2 Velikost prevedene kode

Tako kot pri spletni aplikaciji izdelani s knjižnico React se velikost meri za aplikaciji prevedeni s produkcijskimi nastavitvami. Osnovna velikost celotne aplikacijske kode je 220KB, velikost datoteke stisnjene z gzip pa 60KB, torej se velikost kode za prenos zniža za okoli 72%.



Slika 4.3: Poročilo o zmogljivosti spletne aplikacije v jeziku Elm.

4.3.3 Število vrstic izvorne kode

```
-------------------------------------------------------------------------
Language          files         blank       comment          code
-------------------------------------------------------------------------
Elm                  20           474             1          2123
JavaScript            1             7             0            30
HTML                  1             1             0            10
-------------------------------------------------------------------------
SUM:                 22           482             1          2163
-------------------------------------------------------------------------
```

Slika 4.4: Poročilo o zmogljivosti spletne aplikacije v jeziku Elm.

V podatkih na sliki 4.4 razberemo, da spletna aplikacija v jeziku Elm vsebuje 2123 vrstic izvorne kode jezika Elm v 20 datotekah. To nekoliko več kode kot JavaScript v aplikaciji s knjižnico React, vendar v manj datotekah. Prav tako kot v prejšnji aplikaciji so tudi tu stili vključeni znotraj programske kode Elm.

Koda v jeziku JavaScript služi za povezovanje jezika Elm s funkcijami brskalnika za shrambo podatkov `localStorage` oziroma `sessionStorage` ter za funkcijo `confirm`, ki uporabniku pokaže pozivno okno za potrditev ali preklic akcije. Prav tako kot pri prejšnji aplikaciji datoteka HTML služi za osnovno strukturo ter metapodatke, ki omogočajo ustrezen prikaz v brskalnikih.

4.3.4 Opis izkušnje med razvojem spletne aplikacije

Začetki

V primerjavi z izkušnjo med razvojem projekta s knjižnico React so začetki pri jeziku Elm lažji, saj ni potrebno sprejemati toliko odločitev. Priporočena orodja za delo so bolj jasno določena, hkrati obstaja tudi eno glavno orodje in ne več alternativ, kjer se mora razvijalec odločiti za pravo.

Tudi delo z odvisnostmi je poenostavljeno, predvsem pa bolj zanesljivo, saj odvisnosti vedno sledijo ustrezni semantični oznaki različice. Zato se lažje

prepreči posodobitev, ki povzroči napako v aplikaciji.

Za celoten projekt se uporablja 12 modulov dodanih z orodjem npm [24]. V večini so to moduli za vzpostavitev delovnega okolja preko orodja Webpack [40] in orodja za prevajanje in stilsko oblikovanje izvorne kode jezika Elm. Poleg teh je dodanih še 14 odvisnosti preko orodja `elm-package`, ki dodajajo funkcionalnosti kot so delo z zahtevki na strežnik, dodajanje stilov CSS in navigacija po spletni aplikaciji.

Učenje jezika Elm

Čeprav jezik Elm cilja specifično na razvoj spletih aplikacij, je podobnosti z jezikom JavaScript malo. Pri učenju je glavno zavedanje, da je pristop programiranja s funkcijskim programskim jezikom drugačen od bolj znanega imperativnega programiranja. Poleg tega je celotna programska koda oblikovana okoli arhitekture Elm, zato je pomembno poznavanje le-te in sledenju predpisanim dobrim praksam.

O tipih

Od programerja se pričakuje poznavanje konvencij jezika. Da se imena tipov vedno začnejo z veliko črko, spremenljivke pa z malo črko. Na to se opira tudi prevajalnik, saj ob nespoštovanju opisane konvencije javi napako.

Poleg osnovnih tipov so tudi tipi z vrednostmi, kot na primer `List String`, ki je seznam nizov. Pogost tip v projektu jezika Elm je `Html msg`. To je tip, ki ga vrne funkcija, ki predstavlja komponento v aplikaciji. Ta tip opisuje virtualni HTML element, na katerega je vezano sporočilo `msg`. Iz dejstva, da se spremenljivke vedno začnejo z malo črko izhaja, da `msg` ni konkreten tip, ampak je prost tip, ki se določa glede na kontekst uporabe.

Preslikava sporočil

Sporočila opisujejo nek dogodek v aplikaciji, ki jim lahko pripišemo vrednosti, ki opisujejo dogodek. Pogosto poimenovanje tipa za sporočila na nivoju celotnega projekta je `Msg`.

Za večje aplikacije je en sam globalen tip za sporočila `Msg` težko obvladljiv. Zato se za določene dele aplikacije definira nov tipe za sporočila, ki prenašajo podatke o akciji za ta del aplikacije. Ta sporočila se združijo v enem samem sporočilu glavnega dela aplikacije. Ker pa jezik Elm pričakuje enak tip sporočila za celotno drevo strukture aplikacije, je potrebno tip ustrezno preslikati v glavni tip, kar omogoča funkcija `Html.map`.

```
type Msg
  = OnSubMsg SubMsg
  | ...
type SubMsg
  = ...
```

Slika 4.5: Primer tipa sporočila na nivoju aplikacije.

V primeru na sliki 4.5 je `OnSubMsg` ena od možnosti za vrednost tipa `Msg`, ki ima vrednost sporočila `SubMsg`. Definicija tipa `SubMsg` v tem primeru ni pomembna, saj se vrednost le prenese naprej k funkciji, ki ustrezno obdela sporočilo. Sporočilo `SubMsg` se uporabi v pod-komponenti `subView`, kot je prikazano na sliki 4.6.

```
subView : Html SubMsg
subView = ...
```

Slika 4.6: Primer komponente, ki vrne sporočilo tipa `SubMsg`.

Sporočilo komponente na sliki 4.6 se ustrezno preslika v vrednost `OnSubMsg` za tip glavnega sporočilo `Msg`. Za preslikavo se uporabi funkcija `Html.map`, kot je prikazano na primeru 4.7.

```
subView
|> Html.map OnSubMsg
```

Slika 4.7: Primer preslikave sporočila tipa `SubMsg`.

Pogojni stavki v jeziku Elm

Prevajalnik jezika Elm zahteva, da so pri pogojnih stavkih pokrite vse možne vhodne vrednosti [14].

```
type Options
    = Option1
    | Option2
    | Option3
```

Slika 4.8: Primer tipa s tremi možnimi vrednostmi.

Ko za tip `Options` s tremi različnimi vrednostmi, kot na sliki 4.8, programer skuša v pogojnem stavku `case` za vrednost `opt` tipa `Options` pokriti le opcijo `Option1`, kot v primeru na sliki 4.9, prevajalnik javi napako na sliki 4.10.

```
case opt of
    Option1 -> 1
```

Slika 4.9: Poskus kode, kjer je za tip `Option` iz slike 4.8 pokrita pot za le eno od možnih vrednosti.

```
This ‘case‘ does not have branches for all possibilities.
23|>   case opt of
24|>       Option1 -> 1
You need to account for the following values:
    Option2
    Option3
Add branches to cover each of these patterns!
```

Slika 4.10: Napaka v primeru, ko v stavku `case` niso pokrite vse možne vrednosti.

Napaka na sliki 4.10 pravi, da pogojni stavek na vrstici 23 ne pokriva vseh možnih opcij za vrednost `opt` in tudi pove, katere vrednosti še pričakuje.

Obstajajo primeri, kjer programer želi pokriti le nekaj primerov, za ostale pa uporabiti privzeto vrednost. Za tovrstne primere se uporabi nadomestni

vzorec. Nadomestni vzorec je kar ime spremenljivke za prejeto vrednost. Za neuporabljeno vrednost se po konvenciji uporabi znak _.

Na sliki 4.11 je prikazan popravek za primer na sliki 4.9. Uporabi se nadomestni vzorec s spremenljivko _, ki za vrednost `Option1` vrne 1, sicer pa 0.

```
case opt of
    Option1 -> 1
    _ -> 0
```

Slika 4.11: Popravljena koda, kjer je za tip `Option` iz slike 4.8 pokrita le pot za vrednost `Option1`, sicer pa se uporabi nadomesti vzorec.

Tak pristop zagotavlja, da je kontekst vrednosti vedno poznan. To prevajalniku omogoča sklepanje o tipih in zaznavanje napak. To je pomembno predvsem za funkcijo `update` pri arhitekturi Elm. Le-ta kot argument prejme sporočilo, za katerega se vedno pokrijejo vse možne vrednosti. Tako se zagotavlja, da v aplikaciji ni nepredvidenega stanja.

Delo z JSON iz strežnika

Ker je prevajalnik jezika Elm zasnovan na način, da mora na vsaki točki poznati vse možne vrednosti za podatkovne tipe, je delo z objekti JSON nekoliko oteženo. Prevajalnik ne more predvidevati strukture objekta iz strežnika, zato je potrebno eksplicitno dekodirati vse objekte.

Če strežnik za podatke o komentarju vrne strežnik vrednost JSON, kot je v primeru na sliki 4.12, je v jeziku Elm potrebno napisati kodo za dekodiranje, kot prikazuje primer na sliki 4.13[1].

V primeru na sliki 4.13 so s funkcijo `required` označena polja, ki so obvezna v objektu JSON. Če vrednosti ni, pride do napake pri dekodiranju. Tovrstne napake je potrebno ustrezno obravnavati pred uporabo vrednosti.

[1]Pri funkciji za dekodiranje je pomemben vrstni red atributov. Ta se mora ujemati z zaporedjem vrednosti v tipu, ki se dekodira, v zgornjem primeru je to `Comment`.

```
{
  "id": 1,
  "text": "besedilo",
  "rating": 3,
  "user_rating": null
}
```

Slika 4.12: Primer vrednosti JSON iz strežnika.

```
import Json.Decode as Decode
import Json.Decode.Pipeline exposing (decode, optional, required)
type alias Comment =
    { id : Int
    , text : String
    , rating : Int
    , userRating : Int
    }
commentDecoder : Decode.Decoder Comment
commentDecoder =
    decode Comment
        |> required "id" Decode.int
        |> required "text" Decode.string
        |> required "rating" Decode.int
        |> optional "user_rating" Decode.int 0
```

Slika 4.13: Primer dekodiranja objekta JSON v jeziku Elm.

Funkcija `required` kot prvi argument prejme ime polja v objektu JSON ter funkcijo za dekodiranje kot drugi argument. Funkcije za dekodiranje primitivnih vrednosti ponuja modul `Json.Decode`.

Funkcija `optional` za razliko od funkcije `required` prejme še privzeto vrednost. Dana privzeta vrednost se uporabi v primeru, ko vrednost ne obstaja v objektu JSON oziroma je vrednost enaka `null`. Z uporabo vrednosti `optional` ni možnosti za napako pri dekodiranju, ampak hkrati tudi ni varnosti pri zagotavljanju pravilnega dekodiranja.

Izkušnja razvijalca

Začetna izkušnja dela z jezikom Elm je pozitivna, saj so vse pomembne stvari že uradno določene. Zato razvijalec ne rabi sprejeti pomembnih odločitev o

projektu še preden začne z delom.

Delo s funkcijskim jezikom je nova izkušnja v primerjavi z ostalim programiranjem med študijem. Za to paradigmo programiranja se težko trdi, da je boljša ali slabša od ostalih paradigem. Vpliv pa je odvisen predvsem od posameznikove dojemljivosti in strinjanja s koncepti funkcijskega jezika.

Zanimivi koncepti jezika so predvidljiv tok podatkov skozi funkcije, preprečevanje spreminjanja že nastavljenih vrednosti in onemogočanje slučajnih stranskih učinkov. Vsi ti koncepti omogočajo razvoj bolj varne aplikacije.

Prevajalnik jezika Elm skuša biti v pomoč programerju. Zato so sporočila ob napaki oblikovana z namenom, da so enostavno berljiva in da hkrati ponujajo namig za rešitev zaznanega problema. Pri razvoju spletne aplikacije se je to izkazalo kot zelo uporabna lastnost. To pa je dobrodošlo predvsem začetnike, ki se še spoznavajo z jezikom Elm.

Hkrati pa vsa varnost, ki jo prevajalnik zagotavlja, programerja omejuje in zavira delo. Končni rezultat je sicer bolj varen, verjetnost za napake je manjša, popravilo napake pa lažje. Ampak v primerih, kjer je namen izdelati prototip v čim krajšem času, se jezik Elm zaradi omejevanja za ceno varnosti ne izkaže kot najboljša izbira.

Poleg tega je zaznati, da je skupnost razvijalcev še vedno majhna. Obstoječih rešitev v obliki knjižnic je še vedno veliko manj v primerjavi z rešitvami, ki so ponujene za knjižnico React, sploh pa za jezik JavaScript.

4.4 Primerjava

V primerjavi meritev hitrosti spletne aplikacije, povzetih v tabeli 4.1, se jezik Elm izkaže za boljšo rešitev kot rešitev s knjižnico React. Razlog za hitrejše delovanje je predvsem veliko manjša velikost prevedene programske kode. To pa zagotavlja hitrejši prenos zahtevanih datotek preko spleta.

Pri primerjavi števila vrstic programske kode je jezik Elm nekoliko slabši. Vredno pa je omeniti dejstvo, da je v jeziku Elm velik poudarek na berljivosti kode, kar se v projektu zagotavlja avtomatsko z orodjem `elm-format`. Poleg

metrika	React	Elm
Čas nalaganja strani	3790ms	3310ms
Čas prenosa kode	2270ms	1775ms
Indeks hitrosti zaznavanja	3750	3269
Ocena za indeks	71	77
Skupna ocena (Lighthouse)	78	83
Velikost kode	760KB	220KB
Velikost kode (gzip)	196KB	60KB
Število vrstic kode	2008	2163

Tabela 4.1: Povzetek rezultatov meritev.

tega je v jeziku Elm nekaj dodatnih vrstic zaradi vseh definicij tipov. Soditi to metriko v prid prvemu oziroma drugemu pristopu bi bilo subjektivno. Razlika je dokaj majhna in predvsem odvisna od preference za stil pisanja kode vsakega posameznika.

Povzeto je bila izkušnja pri jeziku Elm razvijalcu bolj prijazna kot celoten pristop s knjižnico React. Vendar pa je zaradi narave funkcijskih jezikov, med katere sodi Elm, potrebno poznati nekatere koncepte ter sprejeti drugačen način programiranja. Sicer se zahtevnost učenja drastično poveča, hkrati pa tudi prijetnost izkušnje med razvojem upade.

Jezik Elm skuša zagotoviti čim lažji način za razvoj varnih in hitrih aplikacij, vendar pa za ceno tega pride večji čas razvoja. To pomeni, da bi bil jezik Elm slaba izbira v primerih, kjer je cilj čim hitrejši razvoj prototipa. Če nekoliko daljši čas razvoja ni prevelika ovira, se jezik Elm izkaže za zelo dobro izbiro, saj omogoča lažji razvoj varnih spletnih aplikacij.

Poglavje 5

Sklepne ugotovitve

Na podlagi meritev se jezik Elm izkaže kot boljša izbira za izdelavo zmogljive spletne aplikacije.

Hkrati tudi olajša razvoj varnih spletnih aplikacij, saj prevajalnik od programerja pričakuje, da ustrezno ravna za vsako možno stanje v aplikaciji in tako bolj razmisli o delovanju aplikacije. Cena takšnega načina zagotavljanja varnosti pa je povečan čas razvoja, kar pomeni, da jezik Elm ne bi bil primeren za hiter razvoj prototipov. Ker gre za drug jezik, ki je hkrati funkcijski programski jezik, je za namene razvoja spletnih aplikacij učenje jezika Elm bolj zahtevno kot učenje ogrodja za razvoj spletnih aplikacij, ki deluje v jeziku JavaScript.

Ocena primernosti jezika Elm za razvoj spletnih aplikacij je odvisna predvsem od ciljev in omejitev za posamezno spletno aplikacijo. Potrebno je upoštevati, da je za jezik Elm potrebno več časa za načrtovanje in razvoj po pričakovanjih prevajalnika. Iz zbranih rezultatov se jezik Elm izkaže za bolj zmogljivo rešitev in je primeren predvsem v primerih, kjer je potrebno izdelati robustno spletno aplikacijo. Tudi izkušnja med razvojem, kar je sicer subjektivna metrika, je za razvijalca prijeta.

Literatura

[1] Add React to a New Application - React documentation. https://web.archive.org/web/20180301233539/https://reactjs.org/docs/add-react-to-a-new-app.html. [Dostopano 4. 3. 2018].

[2] Amazon CloudFront. https://aws.amazon.com/cloudfront/. [Dostopano 2. 3. 2018].

[3] Amazon S3. https://aws.amazon.com/s3/. [Dostopano 2. 3. 2018].

[4] babel-polyfill. https://babeljs.io/docs/usage/polyfill/. [Dostopano 4. 3. 2018].

[5] Breaking changes - React v15.0. https://web.archive.org/web/20180119140135/https://reactjs.org/blog/2016/04/07/react-v15.html. [Dostopano 26. 2. 2018].

[6] Breaking changes - React v16.0. https://web.archive.org/web/20180113230857/https://reactjs.org/blog/2017/09/26/react-v16.0.html. [Dostopano 26. 2. 2018].

[7] Collection: Front-end JavaScript frameworks. https://web.archive.org/web/20180227022606/https://github.com/collections/front-end-javascript-frameworks. [Dostopano 26. 2. 2018].

[8] Compressing with gzip - MDN. https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Content-Encoding#Compressing_with_gzip. [Dostopano 2. 3. 2018].

[9] Elixir language home page. https://elixir-lang.org/. [Dostopano 2. 3. 2018].

[10] Elm language home page. https://web.archive.org/web/20180202012552/http://elm-lang.org/. [Dostopano 2. 2. 2018].

[11] Elm types. https://web.archive.org/web/20180302024018/https://guide.elm-lang.org/types/. [Dostopano 2. 3. 2018].

[12] ESLint. https://eslint.org/. [Dostopano 4. 3. 2018].

[13] Functional programming: Purity - HaskellWiki. https://wiki.haskell.org/Functional_programming#Purity. [Dostopano 26. 2. 2018].

[14] Hints for Missing Patterns - Elm compiler. https://github.com/elm-lang/elm-compiler/blob/0.18.0/hints/missing-patterns.md. [Dostopano 5. 3. 2018].

[15] Introducing JSX - React documentation. https://web.archive.org/web/20180302081617/https://reactjs.org/docs/introducing-jsx.html. [Dostopano 2. 3. 2018].

[16] JavaScript type coercion explained. https://web.archive.org/web/20180305022050/https://medium.freecodecamp.org/js-type-coercion-explained-27ba3d9a2839?gi=e06b25a6016d. [Dostopano 5. 3. 2018].

[17] JSX Compiler Service - React documentation. https://web.archive.org/web/20180302081139/https://reactjs.org/jsx-compiler.html. [Dostopano 2. 3. 2018].

[18] Lighthouse - Tools for Web Developers. https://developers.google.com/web/tools/lighthouse/. [Dostopano 2. 3. 2018].

[19] Lighthouse Report Viewer. https://googlechrome.github.io/lighthouse/viewer/. [Dostopano 2. 3. 2018].

[20] Lodash. https://lodash.com/. [Dostopano 4. 3. 2018].

[21] MobX. https://mobx.js.org/. [Dostopano 4. 3. 2018].

[22] Moment.js. http://momentjs.com/. [Dostopano 4. 3. 2018].

[23] "Most React apps will have their files "bundled" using tools like Webpack" - React documentation. https://web.archive.org/web/20180304035506/https://reactjs.org/docs/code-splitting.html. [Dostopano 4. 3. 2018].

[24] npm. https://www.npmjs.com/. [Dostopano 4. 3. 2018].

[25] Phoenix framework home page. http://phoenixframework.org/. [Dostopano 2. 3. 2018].

[26] Prettier. https://prettier.io/. [Dostopano 4. 3. 2018].

[27] React documentation. https://web.archive.org/web/20180129022736/https://reactjs.org/docs/hello-world.html. [Dostopano 3. 2. 2018].

[28] react-redux. https://github.com/reactjs/react-redux. [Dostopano 4. 3. 2018].

[29] react-router. https://reacttraining.com/react-router/. [Dostopano 4. 3. 2018].

[30] Redux. https://redux.js.org/. [Dostopano 4. 3. 2018].

[31] Redux.js Prior Art - Elm influence. https://web.archive.org/web/20180302022819/https://redux.js.org/introduction/prior-art#elm. [Dostopano 2. 3. 2018].

[32] Releases - React. https://web.archive.org/web/20180226223051/https://github.com/facebook/react/releases. [Dostopano 26. 2. 2018].

[33] Sass: Syntactically Awesome Style Sheets. https://sass-lang.com/. [Dostopano 4. 3. 2018].

[34] seamless-immutable. https://github.com/rtfeldman/seamless-immutable. [Dostopano 4. 3. 2018].

[35] Semantic versioning. https://web.archive.org/web/20180202151449/https://semver.org/. [Dostopano 2. 2. 2018].

[36] styled-components. https://www.styled-components.com/. [Dostopano 4. 3. 2018].

[37] The State of JavaScript 2017: Front-end Frameworks – Results. https://web.archive.org/web/20180121011429/https://stateofjs.com/2017/front-end/results/. [Dostopano 26. 2. 2018].

[38] Typechecking With PropTypes - React documentation. https://web.archive.org/web/20180301234010/https://reactjs.org/docs/typechecking-with-proptypes.html. [Dostopano 4. 3. 2018].

[39] Vue.js scaling up - Vuex Elm influence. https://web.archive.org/web/20180302023115/https://vuejs.org/v2/guide/comparison.html#Scale. [Dostopano 2. 3. 2018].

[40] Webpack. https://webpack.js.org/. [Dostopano 2. 3. 2018].

[41] What's New In DevTools (Chrome 60). https://web.archive.org/web/20180302054246/https://developers.google.com/web/updates/2017/05/devtools-release-notes. [Dostopano 2. 3. 2018].

[42] whatwg-fetch. https://github.com/github/fetch. [Dostopano 4. 3. 2018].

[43] Evan Czaplicki. Elm: Concurrent FRP for functional GUIs. *Senior thesis, Harvard University*, 2012.

[44] Evan Czaplicki and Stephen Chong. Asynchronous functional reactive programming for GUIs. In *Proceedings of the 34th ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 411–422, New York, NY, USA, June 2013. ACM Press.

[45] Nils Eriksson and Christofer Ärleryd. Elmulating JavaScript, 2016.

[46] Jeremy Fairbank. *Programming Elm*. The Pragmatic Bookshelf, 2018.

[47] Artemij Fedosejev. *React.js Essentials*. Packt Publishing Ltd, 2015.

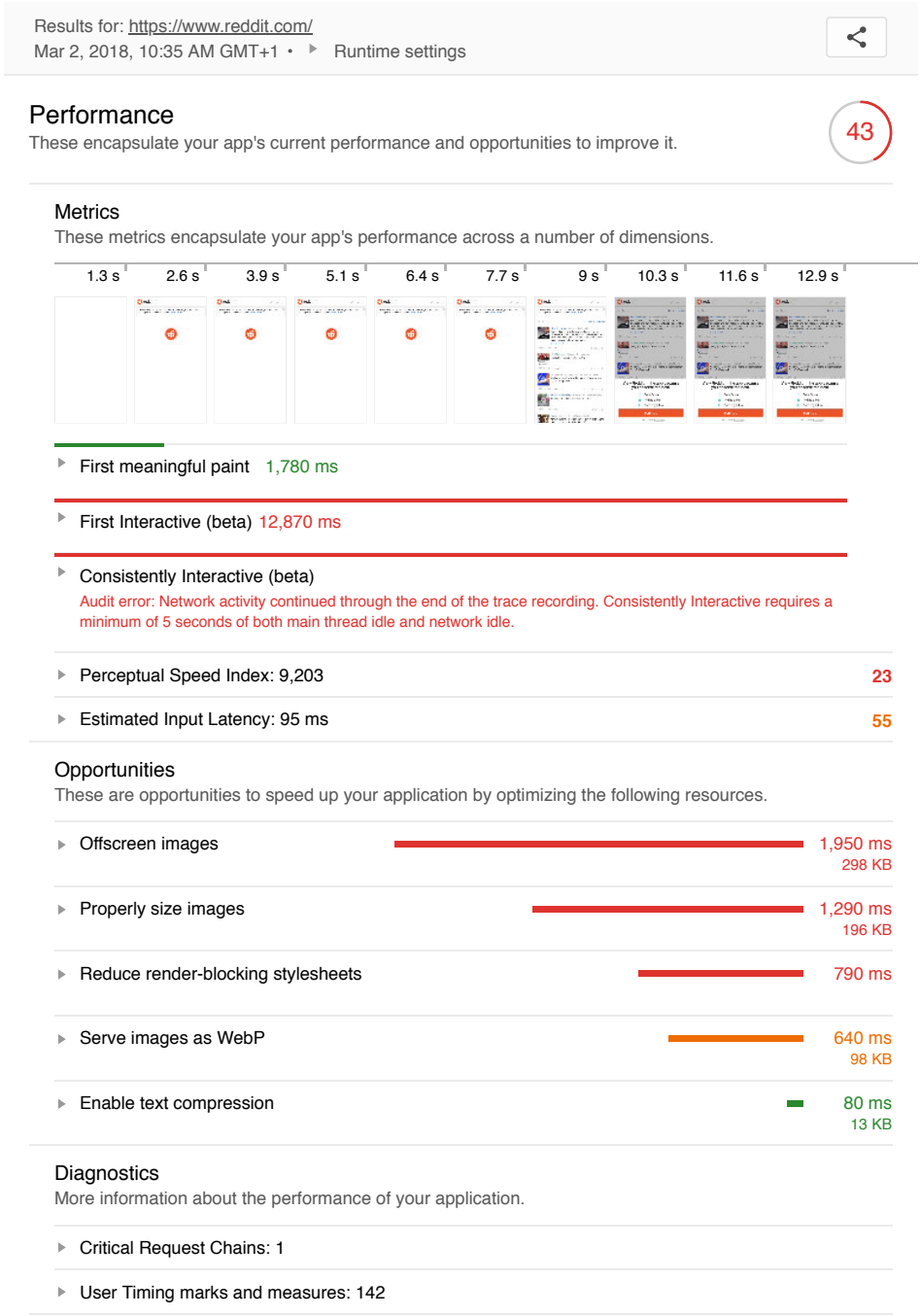
[48] Richard Feldman. *Elm in Action*. Manning, 2017.

[49] Cory Gackenheimer. *Introduction to React*. Apress, 2015.

Dodatek A

Poročila orodja Lighthouse

A.1 Poročilo za spletno stran reddit.com



Slika A.1: Poročilo o zmogljivosti spletnega portala Reddit.

A.2 Razširjeno poročilo za projekt s knjižnico React

Results for: https://reddit-clone-react.gresak.io/
Feb 20, 2018, 11:02 AM GMT+1 • Runtime settings

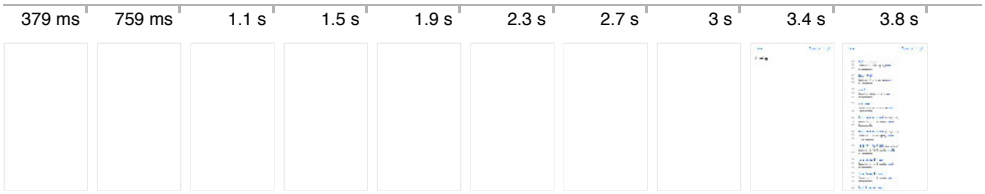
Performance

These encapsulate your app's current performance and opportunities to improve it.

78

Metrics

These metrics encapsulate your app's performance across a number of dimensions.



- First meaningful paint — 3,790 ms
  First meaningful paint measures when the primary content of a page is visible. Learn more.
- First Interactive (beta) — 3,790 ms
  First Interactive marks the time at which the page is minimally interactive. Learn more.
- Consistently Interactive (beta) — 3,790 ms
  Consistently Interactive marks the time at which the page is fully interactive. Learn more.
- Perceptual Speed Index: 3,750 — 71
  Speed Index shows how quickly the contents of a page are visibly populated. Learn more.
- Estimated Input Latency: 16 ms — 100
  The score above is an estimate of how long your app takes to respond to user input, in milliseconds. There is a 90% probability that a user encounters this amount of latency, or less. 10% of the time a user can expect additional latency. If your score is higher than Lighthouse's target score, users may perceive your app as laggy. Learn more.

Opportunities

These are opportunities to speed up your application by optimizing the following resources.

- Reduce render-blocking stylesheets — 580 ms

  Link elements are blocking the first paint of your page. Consider inlining critical links and deferring non-critical ones. Learn more.

  View Details

URL	Size (KB)	Delayed Paint By (ms)
/ (reddit-clone-react.gresak.io)	0.71 KB	577 ms

Diagnostics

More information about the performance of your application.

▼ Critical Request Chains: 1
The Critical Request Chains below show you what resources are required for first render of this page. Improve page load by reducing the length of chains, reducing the download size of resources, or deferring the download of unnecessary resources. Learn more.
Longest chain: **2,270.5ms** over **2** requests, totalling **196.41 KB**

▼ View critical network waterfall:

Initial Navigation
/ (reddit-clone-react.gresak.io)
/app-a7b3d88….js (reddit-clone-react.gresak.io) **- 1,671.4ms, 196.41 KB**

▼ 10 Passed Audits

▼ Reduce render-blocking scripts
Script elements are blocking the first paint of your page. Consider inlining critical scripts and deferring non-critical ones. Learn more.

▼ Properly size images
Serve images that are appropriately-sized to save cellular data and improve load time. Learn more.

▼ Offscreen images
Consider lazy-loading offscreen images to improve page load speed and time to interactive. Learn more.

▼ Optimize images
Optimized images load faster and consume less cellular data. Learn more.

▼ Serve images as WebP
WebP provides better lossy and lossless compression than PNG or JPEG, which means faster downloads and less data consumption. Learn more.

▼ Enable text compression
Text-based responses should be served with compression (gzip, deflate or brotli) to minimize total network bytes. Learn more.

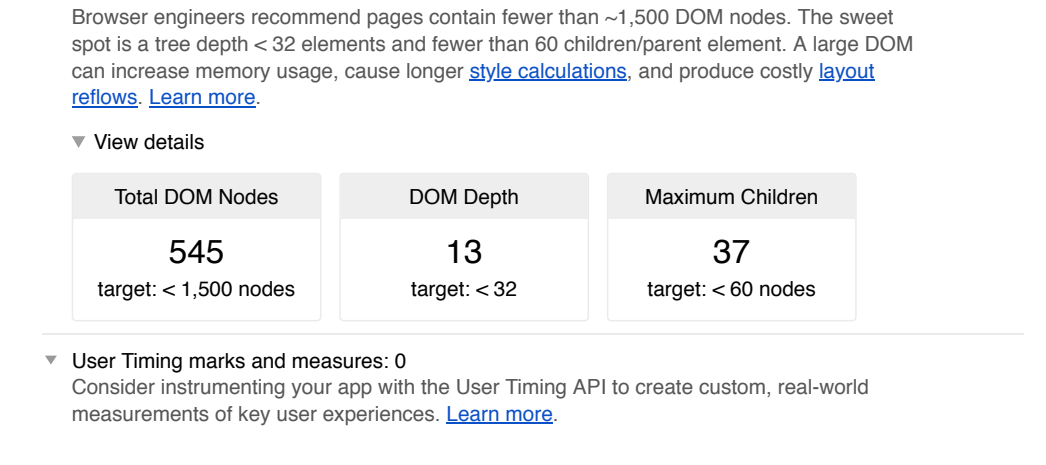
▼ Keep server response times low (TTFB): 570 ms
Time To First Byte identifies the time at which your server sends a response. Learn more.

▼ Avoids enormous network payloads: Total size was 199 KB 100
Network transfer size costs users real money and is highly correlated with long load times. Try to find ways to reduce the size of required files.

▼ View Details

URL	Total Size	Transfer Time
/app-a7b3d88….js (reddit-clone-react.gresak.io)	196 KB	1,180 ms
/api/posts (reddit-eu.herokuapp.com)	1 KB	10 ms
/ (reddit-clone-react.gresak.io)	1 KB	0 ms
/favicon.ico (reddit-clone-react.gresak.io)	0 KB	0 ms

▼ Avoids an excessive DOM size: 545 nodes 100



Slika A.2: Razširjen pogled poročila o zmogljivosti spletne aplikacije izdelane s knjižnico React.

A.3 Razširjeno poročilo za projekt v jeziku Elm

Results for: https://reddit-clone-elm.gresak.io/
Feb 20, 2018, 12:51 PM GMT+1 • Runtime settings

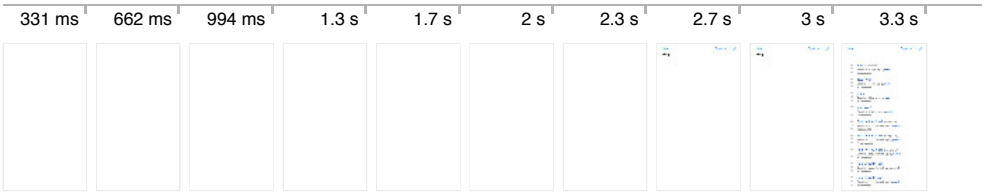
Performance

These encapsulate your app's current performance and opportunities to improve it.

83

Metrics

These metrics encapsulate your app's performance across a number of dimensions.



- First meaningful paint — 3,310 ms
  First meaningful paint measures when the primary content of a page is visible. Learn more.
- First Interactive (beta) — 3,310 ms
  First Interactive marks the time at which the page is minimally interactive. Learn more.
- Consistently Interactive (beta) — 3,310 ms
  Consistently Interactive marks the time at which the page is fully interactive. Learn more.
- Perceptual Speed Index: 3,269 — 77
  Speed Index shows how quickly the contents of a page are visibly populated. Learn more.
- Estimated Input Latency: 16 ms — 100
  The score above is an estimate of how long your app takes to respond to user input, in milliseconds. There is a 90% probability that a user encounters this amount of latency, or less. 10% of the time a user can expect additional latency. If your score is higher than Lighthouse's target score, users may perceive your app as laggy. Learn more.

Opportunities

These are opportunities to speed up your application by optimizing the following resources.

- Reduce render-blocking stylesheets — 580 ms

  Link elements are blocking the first paint of your page. Consider inlining critical links and deferring non-critical ones. Learn more.

  View Details

URL	Size (KB)	Delayed Paint By (ms)
/ (reddit-clone-elm.gresak.io)	0.67 KB	576 ms

Diagnostics

More information about the performance of your application.

▼ Critical Request Chains: 1
The Critical Request Chains below show you what resources are required for first render of this page. Improve page load by reducing the length of chains, reducing the download size of resources, or deferring the download of unnecessary resources. Learn more.
Longest chain: **1,775.4ms** over **2** requests, totalling **57.69 KB**

▼ View critical network waterfall:

Initial Navigation
/ (reddit-clone-elm.gresak.io)
/app-5e5bf3a….js (reddit-clone-elm.gresak.io) **- 1,178.1ms, 57.69 KB**

▼ 10 Passed Audits

▼ Reduce render-blocking scripts
Script elements are blocking the first paint of your page. Consider inlining critical scripts and deferring non-critical ones. Learn more.

▼ Properly size images
Serve images that are appropriately-sized to save cellular data and improve load time. Learn more.

▼ Offscreen images
Consider lazy-loading offscreen images to improve page load speed and time to interactive. Learn more.

▼ Optimize images
Optimized images load faster and consume less cellular data. Learn more.

▼ Serve images as WebP
WebP provides better lossy and lossless compression than PNG or JPEG, which means faster downloads and less data consumption. Learn more.

▼ Enable text compression
Text-based responses should be served with compression (gzip, deflate or brotli) to minimize total network bytes. Learn more.

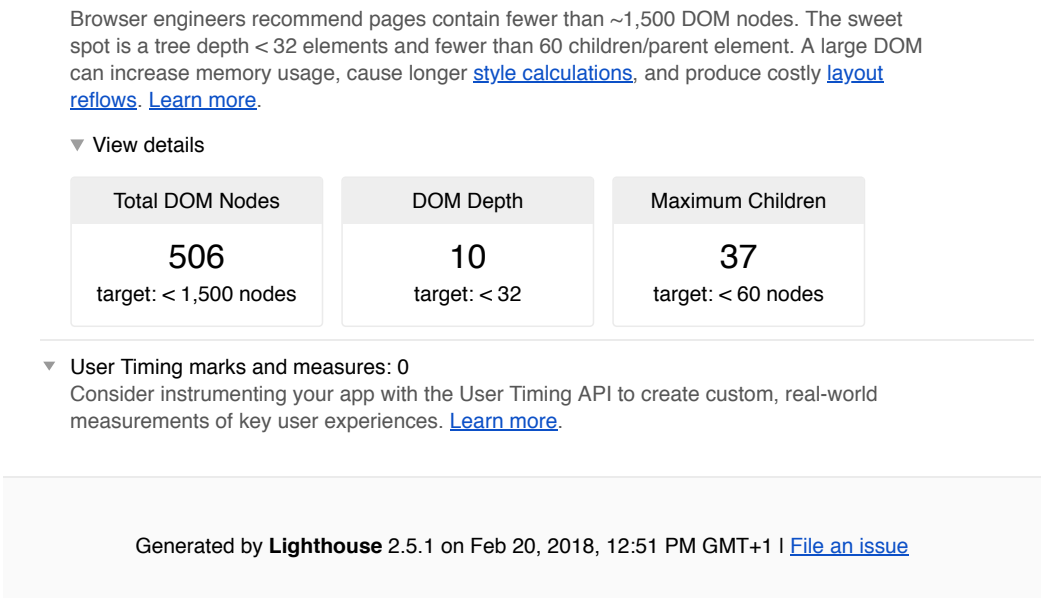
▼ Keep server response times low (TTFB): 570 ms
Time To First Byte identifies the time at which your server sends a response. Learn more.

▼ Avoids enormous network payloads: Total size was 60 KB 100
Network transfer size costs users real money and is highly correlated with long load times. Try to find ways to reduce the size of required files.

▼ View Details

URL	Total Size	Transfer Time
/app-5e5bf3a….js (reddit-clone-elm.gresak.io)	58 KB	860 ms
/api/posts (reddit-eu.herokuapp.com)	1 KB	20 ms
/ (reddit-clone-elm.gresak.io)	1 KB	10 ms
/favicon.ico (reddit-clone-elm.gresak.io)	0 KB	0 ms

▼ Avoids an excessive DOM size: 506 nodes 100



Slika A.3: Razširjen pogled poročila o zmogljivosti spletne aplikacije v jeziku Elm.