Primerjalna analiza: ali naj pisanje predlog CSS ostane ločeno od programske kode JavaScript?

Gregor Jošt, Luka T. Korošec, Matej Žvan

3fs, d.o.o., Kranj, Slovenija
gregor.jost@3fs.si, luka.korosec@3fs.si, matej.zvan@3fs.si

Sinopsis Definiranje stilnih predlog CSS v datotekah JavaScript, oz. CSS-v-JS (angl. CSS-in-JS) predstavlja sodoben pristop k oblikovanju spletnih strani, ki ne zahteva uporabe zunanjih datotek CSS, ampak celotno oblikovanje CSS definiramo v programskem jeziku JavaScript. Pri tem ohranimo vso sintakso, pravila in razumevanje CSS, le implementacijo prestavimo v datoteko JavaScript. Ogrodja za razvoj odjemalskega dela spletnih strani ne podpirajo CSS-v-JS pristopa, zato imamo v ta namen na voljo več kot 50 knjižnic.
Po drugi strani ima običajen pristop oblikovanja spletnih aplikacij še vedno jasne prednosti. Uporaba različnih metodologij ali tehnologij ostaja pogost pristop oblikovanja spletnih strani, ki ne zahteva učenja novega ogrodja. CSS-v-JS se vse pogosteje uveljavlja kot alternativa, saj ima CSS svojevrstne težave, ki se kažejo v prekrivanju stilov, pomanjkanju izolacije in v pomanjkanju tehnologije za odstranjevanje neuporabljenih stilov (angl. dead code elimination).
Glede na vse večjo priljubljenost in uporabnost pristopa CSS-v-JS smo se v sklopu članka osredotočili na primerjavo omenjenega pristopa z običajnim načinom. V primerjalni analizi se bomo omejili na knjižnico React, za vključitev CSS-v-JS pa bomo uporabili knjižnico styled-components, saj ta velja za eno bolj priljubljenih knjižnic za rabo v ta namen.

Ključne besede:
CSS
CSS-v-JS
React
styled-components

Univerzitetna založba Univerze v Mariboru
ISBN 978-961-286-639-6
DOI https://doi.org/10.18690/um.feri.10.2022.5

1 Uvod

Tehnika pisanja stilov CSS v jeziku JavaScript ni nova, saj se diskusija o smiselnosti in uporabnosti tovrstnega pristopa odvija že vrsto let. Spletne strani Reddit, Atlassian, IMDb in Auth0 [1] [2] CSS-v-JS (angl. CSS-in-JS) omenjeno tehniko že uporabljajo, kar zagotovo priča v prid očitnim prednostim tehnike pred bolj tradicionalnem pristopom, tj. Pred pisanjem stilov v ločenih datotekah CSS.

Pogosto se tehnika CSS-v-JS zamenjuje z dodajanjem stilov CSS neposredno na elemente DOM (stili-v-vrsti, angl. Inline-styles), vendar ta predstavlja popolnoma drugačen pristop. Kljub definiranju stilov v datoteki JavaScript večina knjižnic, ki podpira CSS-v-JS, stile CSS prevede iz kode JavaScript, s čimer ustvari unikatna, samodejno generirana imena razredov, ki jih nato pripne elementom DOM [3]. Z uporabo takšnega načina zagotovimo, da se razredi stilov CSS ne prepisujejo (ali pa, da se sploh ne uporabljajo kljub morebitni definiciji v datoteki CSS). Obenem s tem omejimo tudi apliciranje stilov zgolj na komponento, ki jo določen stil potrebuje. Ravno (pravilno/konsistentno) poimenovanje razredov in prepisovanje stilov veljata za pogosta izziva pri pisanju predlog CSS. Pomembno je izpostaviti, da tehnika CSS-v-JS običajno ne vpliva na hitrost delovanja aplikacij. Še več, CSS-v-JS lahko celo izboljša delovanje, saj spletna stran naloži le stile, ki jih posamezne komponente potrebujejo.

Tehnika CSS-v-JS ima tudi svojevrstne izzive. Dejstvo, da se pisanje stilov CSS prestavi v jezik JavaScript, lahko razvijalcem predstavlja omejitev (še posebej, če so ti vajeni pisati stile CSS v temu namenjeni datoteki). Obenem lahko tvegamo izgubo dodatne sintakse in prednosti SCSS ter podobnih generatorjev CSS, ki jih knjižnice CSS-v-JS načeloma ne podpirajo.

Literatura na področju primerjave tehnike CSS-v-JS in pristopa pisanja stilov CSS v ločeni datoteki je omejena, zato smo v sklopu tega prispevka opravili primerjalno analizo omenjenih pristopov. Omejili smo se na knjižnico React, za tehniko CSS-v-JS pa uporabili priljubljeno knjižnico styled-components.

Članek je sestavljen iz naslednjih poglavij. Najprej bomo v sklopu drugega poglavja predstavili sodobne načine pisanja predlog CSS, pri čemer se bomo osredotočili na module CSS, metodologije, SCSS in spremenljivke. V tretjem poglavju bomo predstavili tehniko CSS-v-JS in si podrobneje pogledali knjižnico styled-components. Sledili bosta poglavji primerjave stilov CSS s tehniko CSS-v-JS, kjer bomo najprej opredelili primer in ga razvili z uporabo obeh pristopov, opisali orodja, s katerimi si bomo pomagali pri pridobivanju podatkov, potrebnih za primerjavo (četrto poglavje) in nato podali še rezultate analize (peto poglavje). V šestem poglavju bomo povzeli ugotovitve in podali zaključne misli.

2 Sodobni načini pisanja predlog CSS

Predloge CSS v osnovi vsebujejo skupek deklaracijskih blokov. Pri pisanju uporabljamo selektorje in jim nastavljamo različne vrednosti za lastnosti, pri tem pa moramo biti pozorni na specifičnost selektorjev in njihov vrstni red. Selektorji lahko poljubno spreminjajo lastnosti, dodajajo nove ali spreminjajo obstoječe, ne morejo pa izbrisati že napisanih lastnosti.

Brskalniki pri izrisu elementov najprej upoštevajo specifičnost selektorjev. V kolikor se pojavita dva identična selektorja, brskalnik uporabi tistega, ki je napisan nižje (angl. Cascading-style-sheet). Pri tem velja omeniti, da obstajajo manjše razlike kako različni brskalniki implementirajo jezik CSS.

Upoštevajoč ta pravila ugotovimo, da lahko implementacija in vzdrževanje predlog CSS predstavljata številne izzive. Slednji pridejo še posebej do izraza, če razvijamo obsežnejše aplikacije v skupinah več razvijalcev. Pri tem so nam lahko v pomoč različna orodja, s katerimi predlog CSS ne pišemo, ampak generiramo. Tak pristop posledično omogoči hitrejši razvoj z manj napakami.

2.1 Generatorji CSS

Generatorji CSS (angl. CSS preprocessors) omogočajo, da namesto neposredne definicije predlog CSS pišemo kodo, ki se bo prevedla v ustrezno predlogo CSS [4]. Med najbolj razširjena generatorja CSS štejemo SCSS in LESS.

Generatorji CSS omogočajo uporabo pogojnih stavkov, zank, funkcij, predpripravljenih blokov (angl. Mixinov) in spremenljivk za optimizirano pisanje predlog CSS. Pri tem lahko poljubno nastavimo spremenljivke, ki omogočijo bolj poenoten končni izdelek. Z uporabo spremenljivk, funkcij in predpripravljenih blokov lahko kodo napišemo enkrat in jo nato večkrat uporabimo (princip DRY), kar omogoča lažje vzdrževanje in spreminjanje predlog.

Številne prednosti generatorjev lahko po drugi strani vodijo v površnost in zlorabo moči generatorjev. Velikokrat se izkaže, da je generirana predloga večja od zapisa brez generatorjev, saj ti omogočajo gnezdenje selektorjev. Slednje lahko hitro privede do nujnega upoštevanja preveč specifičnih pravil, ki jih moramo ponoviti, ne moremo pa jih uporabiti drugje.

2.2 Metodologije

Glavni izziv pri pisanju predlog CSS predstavlja razumevanje zapisa. Običajno ne želimo, da potrebuje uporabnikov brskalnik več različnih predlog, zato te združimo v čim manj dokumentov, kar lahko zaradi velikosti datotek vodi v nepreglednost in težjo razumljivost predlog CSS. V praksi so se zato uveljavile različne metodologije, ki poizkušajo olajšati razumevanje napisanih predlog. Med slednje štejemo BEM, OOCSS, AMCSS, SMACSS in SUITCSS [5]. Z uporabo omenjenih metodologij imamo definirana jasna pravila reševanja problemov. Velikokrat metodologija določi tudi način poimenovanja selektorjev in prav z uporabo strukturiranega pristopa pridobimo na razumevanju napisanega.

Če se osredotočimo na BEM (angl. Block-element-modifier), lahko opazimo, da je osrednji cilj metodologije ponovna uporaba kode (angl. Code reuse). Slednje dosežemo s konsistentnim poimenovanjem in omejitvijo obsega. Po metodologiji BEM ni nič globalno, vsaka komponenta je namreč zapisana v svojem dokumentu in nima vpliva na druge komponente.

Obseg predloge CSS omejimo tako, da uporabljamo izključno razrede CSS, pri čemer vsak razred dobi imenski prostor. BEM predpiše poimenovanje selektorjev kot »blok__element–različica«, vsak izmed treh elementov pa ima svoj pomen. Blok predstavlja imenski prostor komponente, zato mora vsak selektor na začetku vsebovati ime bloka, za katerega nastavlja pravila. Element je podenota komponente, ki ne more delovati samostojno, različica pa je sprememba na bodisi bloku bodisi elementu. V kolikor se striktno držimo metodologije, lahko komponente premikamo med različnimi projekti in ohranjamo enoten izgled.

2.3 Moduli CSS

Moduli CSS se uporabljajo v okoljih JavaScript oz. TypeScript, kjer v kodo vključimo predlogo CSS. Cilj modulov je omejitev obsega delovanja napisanih stilov CSS, pri čemer se zanašamo na to, da bo orodje naše selektorje preimenovalo tako, da ne bodo imeli zunanjega vpliva [6].

2.4 Orodje Postcss

Orodje Postcss služi transformaciji predlog CSS z uporabo jezika JavaSript, kjer lahko CSS poljubno spreminjamo, dodajamo ali brišemo. V praksi se uporabljajo različni, že napisani vtičniki za postcss, na primer postcss-inline-svg[1], ki vse slike SVG doda neposredno v predlogo kot »data:image…«. Vtičnik autoprefixer[2] pa stile CSS transformira tako, da določenim lastnostim doda predpone za brskalnike (npr. -ms, -webkit).

2.5 Spremenljivke CSS

Spremenljivke CSS predstavljajo entitete, ki jih definirajo razvijalci, vsebujejo pa lahko različne vrednosti [7]. Te entitete so nato dosegljive kjerkoli v stilih CSS, običajno pa se uporabljajo, ko moramo določene vrednosti lastnosti spreminjati med delovanjem spletnih strani, na primer pri izdelavi različnih tematik. Obenem tudi omogočajo, da se izognemo ponavljanju določenih vrednosti.

Spremenljivke CSS nastavimo s specifično notacijo (primer: »main-color: black;«) in do njih dostopamo z rezervirano besedo »var« (primer: »color: var(--main-color);«).

3 Definicija stila CSS v jeziku JavaScript (CSS-v-JS)

CSS-v-JS predstavlja tehniko, s katero se oblikovanje komponent definira v kodi JavaScript (oz. TypeScript) in ne v zunanjih datotekah CSS [8]. S poznavanjem sintakse JavaScript torej ustvarimo, dodajamo in upravljamo s stilom CSS. Kljub temu omenjena tehnika ne nadomesti poznavanja sintakse CSS; še vedno moramo razumeti načine dodajanja stilov na elemente DOM, dedovanje in poznati vrste lastnosti, ki jih podpira sintaksa itd.

Kljub dejstvu, da imamo trenutno za razvoj CSS-v-JS na voljo več kot 50 JavaScript knjižnic [3], vse izmed njih posedujejo nekatere skupne značilnosti:

- Omejen doseg CSS, kar podpirajo tudi moduli CSS, omenjeni v prejšnjem poglavju. Knjižnice CSS-v-JS običajno ustvarijo unikatna imena razredov, ki se dodajo na element DOM [3]. Na takšen način je doseg stilov CSS omejen (običajno na eno komponento), prav tako pa je odveč skrb, da bi prišlo do prekrivanja imen razredov CSS.
- Knjižnice skrbijo za samodejno dodajanje predpone (angl. Vendor prefix) lastnostim CSS, ki so še v fazi razvoja ali niso podprte na vseh brskalnikih (primer: -webkit-transition).
- Vse sodobne knjižnice ne dodajajo stilov CSS neposredno na elemente DOM (stili-v-vrsti, angl. Inline-styles), saj ima tovrstno dodajanje stilov precej omejitev, npr. Pomanjkanje podpore psevdo-razredom, psevdo-elementom in medijskim poizvedbam (angl. Media-query) [9], zato je večina knjižnic takšen pristop v celoti umaknila.
- V povezavi z zgornjo točko, knjižnice za razvoj CSS-v-JS podpirajo celoten nabor sintakse CSS, vključno s psevdo-razredi, medijskimi poizvedbami itd., saj se zanašajo na dinamično ustvarjanje razredov.

Knjižnice za razvoj CSS-v-JS se medsebojno razlikujejo, saj so nekatere namenjene specifično določenemu ogrodju JavaScript, medtem ko druge niso odvisne od ogrodja [9]. Med slednje spadajo Emotion, Treat in JSS, medtem ko sta knjižnici styled-components in Stitches namenjeni ogrodju oz. Knjižnici JavaScript (v tem primeru gre za knjižnico React). V prispevku smo se omejili na knjižnico React, zato smo za potrebe raziskave izbrali eno bolj priljubljenih knjižnic, in sicer styled-components.

[1] https://github.com/TrySound/postcss-inline-svg
[2] https://github.com/postcss/autoprefixer

3.1 Knjižnica styled-components

Knjižnica styled-components je namenjena razvoju aplikacij React oz. React Native in skladno z definicijo CSS-v-JS omogoča definiranje stilov na nivoju komponente. Ta knjižnica predstavlja kombinacijo jezika JavaScript (oz. TypeScript) in CSS. Osnovni koncept za pisanje stilov predstavljajo označevalne predloge (angl. Tagged Template Literals) ECMAScript 2016 [2].

Osnovna ideja styled-components je v odstranjevanju preslikave med komponentami in stili. Namesto predloge CSS tako implementiramo komponento React, znotraj katere s pomočjo označevalne predloge definiramo stile CSS. Knjižnica nato ustvari predlogo CSS in razrede pripne vozliščem DOM preko »className« lastnosti elementov React. Predlogo CSS nato vključi na konec elementa »head« spletne strani med izvajanjem (angl. Runtime). Osnovna sintaksa komponent, implementiranih z razširitvami knjižnice styled-components, je naslednja:

```
const ComponentName = styled.element`
      css_property: css_value;
`;
```

Iz zgornjega izseka kode je razvidno, da objekt »styled« podpira vse elemente HTML, znotraj označevalne predloge (vse, kar je znotraj dveh ostrivcev, ``) pa lahko stile CSS definiramo enako kot bi jih v običajni datoteki CSS. Specifičen primer takšne komponente prikazuje naslednji izsek programskega koda:

```
import styled from 'styled-components';

const Container = styled.div`
  margin: 2.5em auto;
  border: 1px solid #dadada;
  padding: 1em;
  min-height: 25em;
`;

const Button = styled.button`
  border: 0.1em solid #999999;
  cursor: pointer;
`;
```

Kot je razvidno, smo definirali dve komponenti; prvo, poimenovano »Container« (predstavlja vsebnik za spletno stran, razširi obstoječ element HTML »div«) in drugo z imenom »Button« (predstavlja generičen gumb, razširi element HTML »button«). Obe komponenti lahko uporabimo na enak način kot običajno komponento React, na primer:

```
function App() {
 return (
   <Container>
     <Button>Hello world</Button>
   </Container>
 );
}
```

Knjižnico styled-components odlikujejo tudi druge lastnosti in funkcionalnosti:

- Ob možnosti razširitve obstoječih elementov DOM imamo tudi možnost razširitve lastnih, obstoječih komponent React.

- Dostop do lastnosti »props« znotraj stilov CSS, kar velja za eno ključnih prednosti knjižnice styled-components. Na takšen način lahko glede na lastnosti komponente ustrezno (in dinamično) definiramo stil CSS. Znotraj označevalne predloge imamo namreč dostop do lastnosti »props«, ki so lahko kakršnega koli tipa.
- Podpora temam: knjižnica podpira definiranje različnih tem in v ta namen ponuja specifično komponento. Vsi elementi znotraj komponente imajo nato dostop do stilov, opredeljenih v temi. Na takšen način je zelo preprosto definirati na primer svetlo oz. Temno temo spletne strani.
- Knjižnica podpira definicijo globalnih stilov s pomočjo temu namenjene funkcije. Komponento, ki predstavlja globalne stile CSS, enkrat vključimo na najvišjem nivoju strukture aplikacije, podobno kot v primeru globalne datoteke CSS.

Podobno kot razvoj običajnih aplikacij React tudi v primeru styled-components struktura ni vnaprej definirana in strogo predpisana. Običajno imamo dva pristopa, in sicer: 1) komponente styled-components so v istem imeniku kot običajne komponente React (npr. Komponenta »Header.tsx« bi imela v istem imeniku še komponento »Header.styled.tsx«) ali 2) komponente styled-components so znotraj istega imenika (običajno poimenovanega »styles«) razdeljene na več datotek (npr. »Header.styled.tsx«, »Footer.styled.tsx« in »Button.styled.tsx«).

Kljub jasnim prednostim, kot so ponovna uporaba kode, dostop do lastnosti »props«, dinamično oblikovanje, podpora temam in zmanjšanju napak med razvojem, ima uporaba knjižnice tudi določene slabosti oz. Izzive. Med slednje lahko štejemo težje razhroščevanje (zaradi potencialnega gnezdenja stilov znotraj ene komponente), povečanje velikosti virtualnega DOM-a knjižnice React, prav tako pa izgubo podpore razširitvam CSS, kot je npr. SASS.

4 Primerjava stilov CSS s pristopom CSS-v-JS

Za relevantno primerjavo različnih stilov pisanja CSS smo potrebovali dve primerljivi aplikaciji. Za namen objektivnosti primerjave smo se najprej odločili za implementacijo aplikacije, ki uporablja CSS, in jo nato prepisali v obliko CSS-v-JS.

Za primerjavo bomo razvili preprosto aplikacijo, ki bo vsebovala dovolj funkcionalnosti, da bomo zajeli različne lastnosti CSS, saj želimo ugotoviti, ali sta pristopa pisanja CSS primerljiva. V ta namen bomo razvili skrajševalnik URL-jev, aplikacija pa bo vsebovala več strani:

- stran za vnos URL-ja in njegovo okrajšavo,
- stran za prikaz okrajšanih URL-jev,
- stran, s predstavitvijo podjetja 3fs, na kateri bo obsežno besedilo in nekaj slik,
- nalagalnik (angl. Loader) in
- stran za primer napake (404).

Z zgoraj omenjenimi stranmi bomo pokrili animacije, vnosna polja in gumbe, tabele, besedilo, različne pisave ter rastrske in vektorske fotografije. Obenem bomo poskrbeli, da bodo vse strani prilagodljive tako majhnim, vmesnim in velikim zaslonom.

Prepis aplikacije v CSS-v-JS bo potekal sistematično, da bosta končna produkta ekvivalentna tako v smislu uporabljenih stilov kot v izgledu spletne aplikacije. Za sledenje spremembam v kodi bomo uporabili GIT, za prepis pa ustvarili novo vejo. Obe aplikaciji bomo namestili na spletno platformo Heroku, kjer bomo kasneje izvedli analizo.

Pri primerjavi obeh pristopov smo se osredotočili na merljive in neposredno primerljive metrike. Najprej bomo primerjali število uporabljenih knjižnic in velikost imenika »node_modules«, za kar bomo uporabili orodje za upravljanje paketov yarn in klasično orodje Linux du. Nato se bomo z odprtokodnim orodjem cloc[3] posvetili razliki v številu datotek in vrstic kode, saj to orodje v svojem poročilu prikaže tudi tip datoteke in tip vrstice v datotekah (tj. Prazna, komentar, koda). Sledi merjenje hitrosti izgradnje (angl. Build) celotne aplikacije z orodjem yarn in klasičnim orodjem Linux time[4] ter primerjava velikosti končnega programskega paketa. Za konec si bomo prihranili primerjavo hitrosti nalaganja obeh spletnih strani, ki jo bomo izmerili z orodjema Google Chrome Lighthouse in SiteSpeed.io.

Analiza različnih metrik se bo izvajala na istem računalniku ob istem času, s čimer zmanjšamo vpliv zunanjih dejavnikov na rezultate analize. V izogib enkratnim odstopanjem v merjenju bomo opravili več meritev in nato izračunali povprečje ter mediano, kjer je to primerno (npr. Pri primerjavi števila knjižnic ni potrebno opraviti več meritev).

5 Rezultati analize

Skladno z opredeljenim v prejšnjem poglavju bomo v tem sklopu izvedli primerjavo in podali rezultate analize.

5.1 Velikost imenika »node_modules« in število uporabljenih knjižnic

Velikost imenika »node_modules« in število uporabljenih knjižnic vpliva na diskovni prostor, zato smo ti dve metriki združili v skupno podpoglavje. Pri izbiri načina pisanja CSS je namreč vredno upoštevati tudi ta vidik.

Pri sodobnem pisanju spletnih aplikacij, ki temeljijo na jeziku JavaScript, je veliko kode, ki poganja aplikacijo, že spisane v obliki različnih knjižnic. Z uporabo orodij za upravljanje paketov, kot sta prej omenjena Yarn in NPM (Node Package Manager), na enostaven način pridobimo knjižnice in jih uporabimo v svoji aplikaciji. Dodane knjižnice tudi same pogosto uporabljajo kakšno drugo knjižnico, tako da lahko število vseh uporabljenih knjižnic naše aplikacije hitro naraste. Vse te knjižnice so lokalno nameščene v imenik »node_modules«.

5.1.1 Velikost imenika »node_modules«

Z uporabo prej omenjenega orodja du lahko hitro ugotovimo velikost imenika »node_modules«. Z ukazom `rm -rf node_modules && yarn install && du -hd 0 node_modules` najprej izbrišemo celoten imenik, nato namestimo vse potrebne knjižnice, na koncu pa pogledamo še velikost novonastalega imenika. Zastavica `-h` pomeni, da je izpis, ki je privzeto v bajtih, pretvorjen v človeku prijazno obliko (megabajti), medtem ko zastavica `-d 0` orodju pove, da nas zanima samo velikost imenika na nivoju 0 (tj. »node_modules«). Postopek ponovimo na obeh vejah, pri CSS in CSS-v-JS.

Rezultat postopka je pokazal, da je velikost imenika na veji CSS 362 Mb, medtem ko je imenik na veji CSS-v-JS velik 373 Mb. Slednji je tako za 3,0 % ali 11 Mb večji.

5.1.2 Število uporabljenih knjižnic

Pri pridobivanju števila uporabljenih knjižnic na prvem nivoju (angl. Direct dependencies) si lahko pomagamo z vpogledom v datoteko package.json, ki se uporablja za beleženje dodanih knjižnic. Upoštevali bomo tako tiste, ki

[3] https://github.com/AlDanial/cloc
[4] https://www.gnu.org/software/time/

so uporabljene samo pri razvijanju aplikacije (razdelek »devDependencies«), kot tudi tiste, ki so v programski paket vključene kasneje (angl. Bundle) (razdelek »dependencies«).

Za pridobivanje števila knjižnic na vseh nivojih (angl. Indirect dependencies) moramo uporabiti orodji yarn in wc[5]. Ukaz, ki ga poženemo v ukazni vrstici, je `yarn list | wc -l`. Prvi del izriše drevesno strukturo vseh knjižnic na način, da je v vsaki vrstici izpisana po ena knjižnica. S preštevanjem števila vseh vrstic dobimo skupno število vseh uporabljenih knjižnic. V ta namen uporabimo orodje wc z zastavico `-l`, ki orodju pove, naj prešteje vse vrstice. Nato odštejemo 3 vrstice, ki jih doda yarn, in dobimo realno število. Tabela 1 prikazuje rezultat analize.

Tabela 1: Primerjava števila uporabljenih knjižnic.

	Število uporabljenih knjižnic na prvem nivoju	Število knjižnic na vseh nivojih
CSS	20	21
CSS-v-JS	35	40
Razlika	21,7 %	2,8 %

Kot je razvidno iz zgornje tabele, uporabljamo 23 knjižnic pri osnovnem pristopu CSS, medtem ko pri CSS-v-JS uporabljamo 28 knjižnic. Število le-teh se tako poveča za 21,7 %.

Število vseh uporabljenih knjižnic pri CSS pristopu je 3926, medtem ko jih pri pristopu CSS-v-JS uporabljamo 4035, kar je za 2,8 % več.

5.2 Število datotek in vrstic kode

Z večanjem števila datotek se obvladljivost projekta otežuje, zato želimo obdržati kar se da nizko število, podobno je tudi s številom vrstic kode. Če bi za ekvivalenten rezultat potrebovali dvakrat več kode, je tak pristop manj uporaben. V tem poglavju bomo analizirali, kako se število datotek in vrstic kode spremeni, ko aplikacijo prepišemo iz CSS v ekvivalenten pristop CSS-v-JS.

Uporabili bomo odprtokodno orodje cloc (angl. Count lines of code). Le-to bo z uporabo zastavice `--vcs git` preštelo in analiziralo le datoteke, ki so vključene v sistem sledenja spremembam git, ignoriralo pa bo datoteke, ki jih ta sistem ignorira (npr. Imenik »node_modules«).

Z vpisom ukaza `cloc --vcs git .` v ukazno vrstico pridobimo tabelo, ki število datotek in vrstice kode združi glede na programski jezik (npr. TypeScript, SCSS, JSON, itd). Za vsak jezik tako pridobimo število datotek, praznih vrstic, komentarjev in število vrstic dejanske kode.

Analiza je pokazala, da se s prepisom aplikacije v pristop CSS-v-JS pravzaprav zmanjša število datotek iz 85 na 70, kar predstavlja 17,6 % razliko. Slednje lahko pripišemo dejstvu, da v primeru CSS-v-JS ne potrebujemo predpripravljenih blokov (angl. Mixin), saj imamo možnost pisanja funkcij neposredno v komponenti CSS-v-JS. Obenem se je število vseh vrstic dejanske kode s prepisom zmanjšalo za 0,7 %, iz 3389 (veja CSS) na 3367 (veja CSS-v-JS). Velikost razlike priča o tem, da moramo v obeh pristopih še vedno pisati isti CSS, spremeni se le lokacija.

Rezultati povedo, da lahko z uporabo pristopa CSS-v-JS pričakujemo manjše število datotek, pri čemer se število potrebnih vrstic kode bistveno ne spremeni.

[5] https://linux.die.net/man/1/wc

5.3 Hitrost graditve aplikacije

Aplikacijo je pred namestitvijo na strežnik potrebno zgraditi. Za gradnjo poskrbi orodje react-scripts, ki je ključen del razvojnega okolja React, aktiviramo pa ga preko ukaza `yarn build`. Za meritev bomo upoštevali čas, ki ga poroča orodje time. Prav tako bomo opravili 5 meritev in izračunali povprečje, s tem pa zmanjšali vpliv ostalih procesov na računalniku na hitrost gradnje. Za meritev bomo v ukazno vrstico vpisali ukaz `rm -rf build && time yarn build`. Ukaz bo najprej odstranil rezultat gradnje, tako da bo morala vsaka gradnja aplikacijo zgraditi od začetka, nato pa bo orodje time gradnjo zagnalo in merilo.

Povprečje petih meritev na veji CSS znaša 5,52 sekund, na veji CSS-v-JS pa 4,81 sekund. Vidimo lahko, da se z uporabo pristopa CSS-v-JS čas gradnje zmanjša za 12,7 % ali za 0,71 sekunde. Predpostavljamo, da je čas krajši zaradi manjšega števila datotek kot tudi zaradi tega, ker obdelovanje datotek CSS ni potrebno, potrebujemo le obdelati datoteke JavaScript, kar zmanjša število korakov.

5.4 Hitrost delovanja in velikost spletne strani

Ko je stran zgrajena, jo namestimo na strežnik in tako postane dostopna vsem na spletu. Hitrost je pri tem ključnega pomena. Aplikacija se mora naložiti dovolj hitro, saj to vpliva na uporabniško izkušnjo. V tem poglavju bomo primerjali rezultate testov, opravljenih z orodjema Lighthouse in Sitespeed.io.

Najprej smo obe veji, CSS in CSS-v-JS, naložili na spletno platformo Heroku. Vsaka aplikacija je nato dostopna na svojem spletnem naslovu, kjer sta neodvisni ena od druge. V podpoglavjih sledi merjenje hitrosti in velikosti spletne strani s prej omenjenima orodjema.

5.4.1 Analiza z Lighthouse

Lighthouse je orodje, vgrajeno v spletni brskalnik Google Chrome. Za izvedbo testa zadostuje obisk spletne strani, ki jo želimo analizirati z zagonom analize Lighthouse (najdemo jo med zavihki v razvijalskih orodjih brskalnika).

Na obeh vejah so vsi glavni indikatorji (hitrost, dostopnost, dobre prakse in optimizacija za iskalnike) pokazali popolno oceno (vrednost 100/100), zato lahko trdimo, da med stranema ni bistvene razlike.

Manjše razlike se kažejo v podrobnostih. Čas do interaktivnosti (angl. Time to interactive) je na veji CSS znašal 0,5 sekunde, na veji CSS-v-JS pa 0,3 sekunde. Prav tako je razlika vidna pri največjem vsebinskem prikazovanju (angl. Largest contentful paint), in sicer se le-ta zgodi pri 0,8 sekunde na veji CSS in pri 0,6 sekunde na veji CSS-v-JS. Slednje pripisujemo manjšemu številu datotek, ki jih mora brskalnik prenesti za ogled strani, saj datotek CSS ne rabi prenašati. Z uporabo http/2 protokola bi se zahtevki za datoteke izvajali vzporedno, kar bi pospešilo delovanje. Predpostavljamo, da bi to vplivalo na rezultate v prid pristopa CSS.

5.4.2 Analiza s Sitespeed.io

SiteSpeed.io je odprtokodno orodje za analizo in spremljanje spletnih strani. V primerjavi z Lighthouse zajame več metrik, hkrati pa podpira analizo več strani hkrati in zaporednih testiranj, ki jih uporabi za izračun povprečij.

Uporaba orodja je nekoliko bolj zahtevna kot pri Lighthouse. Najlažje ga je zagnati z uporabo virtualizacijskega orodja Docker, za katerega SiteSpeed.io ponuja vsebnik (angl. Container). Orodje zaženemo z vpisom ukaza `docker run –rm -v »$(pwd):/sitespeed.io« sitespeedio/sitespeed.io:25.5.1 https://<naslov-nase-spletne-strani>/`. Po zagonu orodje v trenutnem imeniku ustvari poročilo HTML, ki ga lahko odpremo v brskalniku in na tak način preverimo rezultate. Na voljo imamo širok nabor metrik, zato izberemo tiste, ki so najbolj relevantne in ilustrativne za prikaz razlik med različnima načinoma pisanja CSS, ali tiste, ki so se med vejama najbolj razlikovale.

Tabela 2 prikazuje rezultate analize, v kateri so zbrane povprečne vrednosti tekom treh meritev na petih podstraneh naše spletne aplikacije in absolutna in relativna razlika med povprečjema na različnih vejah.

Tabela 2: Rezultati analize s Sitespeed.io.

Angleško ime metrike	CSS	CSS-v-JS	Absolutna razlika	Razlika v odstotkih
Coach performance score	81	84	+3	+3,7%
Best Practice score	99	100	+1	+1,0 %
Total requests	12	10	-2	-16,7 %
HTML size	1,3 KB	1,1 KB	-0,2 KB	-15,4 %
CSS size	3,2 KB	0 b	-3,2 KB	-100 %
Javascript size	70,2 KB	86,3 KB	+16,1 KB	+22,9 %
Size SUM	74,7 KB	87,2 KB	+12,5 KB	+16,7 %
First Paint	396 ms	402 ms	+6 ms	+1,5 %
Fully Loaded	1557 ms	1286 ms	-270 ms	-17,4 %
Largest Contentful Paint	718 ms	683 ms	-35 ms	-4,9 %
First Contentful Paint	396 ms	402 ms	+6 ms	+1,5 %
First Visual Change	421 ms	417 ms	-4 ms	-1,0 %
Speed Index	696 ms	664 ms	-32 ms	-4,6 %
CPU Style Layout	19 ms	20 ms	+1 ms	+5,3 %
CPU Script Evaluation	33 ms	45 ms	+12 ms	+36,4 %

Vidimo lahko, da je celotna ocena strani za 3,7 % višja pri pristopu CSS-v-JS, kar predstavlja povprečje vseh meritev na spletni strani. Število zahtevkov za datoteke je pričakovano višje pri pristopu CSS. Prav tako je pričakovana velikost CSS datotek pri pristopu CSS-v-JS 0 b, medtem ko se velikost datotek JavaScript poveča za kar 22,9 %. V povprečju je velikost vseh datotek pri pristopu CSS-v-JS večja za 16,7 %. Posledično (a morda ne intuitivno) je tudi čas nalaganja spletne strani pri CSS-v-JS manjši, saj mora ta kljub večji velikosti datotek le-teh prenesti več, kar prinese 17,4 % pospešitev. Indeks hitrosti kaže, da je pristop CSS v povprečju za 4,6 % počasnejši od pristopa CSS-v-JS. Pričakovano se poveča čas, ki ga procesor porabi za evaluacijo datotek JavaScript, in sicer kar za 36,4 %.

5.4.3 Velikost programskega paketa

Ob gradnji aplikacije orodje react-scripts priročno prikaže velikost izdelanega programskega paketa. Ta paket ne vsebuje slik, tekstovnih (razne licence) in razhroščevalnih datotek, ki pomagajo pri razvoju aplikacije. Paket smo pretvorili v obliko gzip.

Na veji CSS velikost programskega paketa znaša 89 KB, medtem ko je na veji CSS-v-JS paket velik 104 KB, kar je za 16,9 % ali 15 KB več od paketa na veji CSS.

6 Zaključek

V prispevku smo opravili primerjavo pristopa CSS-v-JS in pisanja stilov CSS v ločeni datoteki. Najprej smo predstavili sodobne pristope pisanja običajnih stilov CSS, nato smo opisali pristop CSS-v-JS, ki združuje prednosti komponentnega razvoja in pisanja stilov CSS.

Sledila je primerjalna analiza obeh pristopov na preprostem primeru, ki je vključeval različne aspekte spletnih aplikacij. Ugotovili smo, da sta pristopa z vidika hitrosti delovanja precej primerljiva. Pričakovana odstopanja smo zaznali pri velikosti programskega paketa, vendar le-ta ne predstavlja razloga za uporabo enega pristopa pred drugim. Po drugi strani je število paketov pri nalaganju spletne strani v primeru pristopa CSS-v-JS nižje kot v primeru pristopa CSS, kar ni bilo pričakovano.

Pri interpretaciji rezultatov moramo upoštevati nekaj omejitev. Kompleksnost in velikost aplikacije, ki smo jo razvili, ni primerljiva z običajnimi poslovnimi aplikacijami. Poudarek smo dali tudi čim bolj podobnemu razvoju obeh aplikacij, zato nismo posebej optimizirali določenega pristopa. Obenem smo izpustili tudi subjektivne metrike, predvsem hitrost pisanja kode, preglednost, težavnost vzdrževanja, razširljivost itd. Nenazadnje se omejitev pojavi tudi v izbiri knjižnice za pristop CSS-v-JS, pri čemer smo v našem primeru izbrali styled-components.

Očitnih prednosti tehnike CSS-v-JS ali slabosti običajnega pristopa pisanja stilov CSS ni zaznati, zato je končna odločitev o izbiri pristopa odvisna zgolj od preferenc razvojne skupine, znanja jezika JavaScript in nenazadnje tudi od obsega projekta, ki ga razvijamo.

Literatura

[1] C. Meade, "A Lukewarm Approval of CSS-in-JS," 20 10 2021. [Spletna stran]. Dosegljivo na: https://sparkbox.com/foundry/css_in_js_overview_css_in_js_pros_and_cons.

[2] styled-components. [Spletna stran]. Available: https://styled-components.com/.

[3] A. Pfeiffer, "A Thorough Analysis of CSS-in-JS," 3 6 2021. [Spletna stran]. Dosegljivo na: https://css-tricks.com/a-thorough-analysis-of-css-in-js/.

[4] MDN, "CSS preprocessor," 10 8 2021. [Spletna stran]. Dosegljivo na: https://developer.mozilla.org/en-US/docs/Glossary/CSS_preprocessor.

[5] "BEM," [Spletna stran]. Dosegljivo na: https://en.bem.info/methodology/faq/#how-does-bem-differ-from-oocss-amcss-smacss-suitcss.

[6] R. Rendle, "What are CSS Modules and why do we need them?," 10 8 2021. [Spletna stran]. Dosegljivo na: https://css-tricks.com/css-modules-part-1-need/.

[7] MDN, "Using CSS custom properties (variables)," 10 7 2022. [Spletna stran]. Dosegljivo na: https://developer.mozilla.org/en-US/docs/Web/CSS/Using_CSS_custom_properties.

[8] A. Rudenko, "CSS-in-JS support in DevTools," [Spletna stran]. Dosegljivo na: https://developer.chrome.com/blog/css-in-js/. [Accessed 26 2 2021].

[9] A. Monus, "An Introduction to CSS-in-JS: Examples, Pros, and Cons," 9 9 2019. [Spletna stran]. Dosegljivo na: https://webdesign.tutsplus.com/articles/an-introduction-to-css-in-js-examples-pros-and-cons--cms-33574.