SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU

FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH ZNANOSTI

Sveučilišni studij

CSS procesori i metodologije

Diplomski rad

Sebastijan Dumančić

Osijek, 2017.

SADRŽAJ

1. UVOD CSS PROCESORE I METODOLOGIJU

CSS ili *Cascading Style Sheets* stilski je jezik koji se koristi za opisivanje izgleda dokumenata većinom napisanih pomoću HTML-a (engl. *HyperText Markup Language*), odnosno kako bi se elementi trebali prikazivati na ekranu, papiru, u govoru ili drugim medijima, prema [1]. Zajedno s već navedenim HTML-om te programskim jezikom *Javascript* čini osnovu današnjeg modernog interneta. Osim samih funkcionalnosti, CSS omogućuje odvajanje prezentacije dokumenta od njegovog sadržaja i strukture čime se postiže lakše održavanje, proširivanje i pristupačnost dokumenata na internet, te visoka razina fleksibilnost prilikom prikazivanja dokumenata na različitim uređajima.

CSS je od svog nastanka 1996. godine prošao kroz nekoliko revizija, ali je svojom svrhom ostao isti. Upravo zbog vremena u kojem je nastao i ubrzanog razvoja interneta koje je zatim uslijedilo, CSS je zaostajao za razvojem internetskih stranica i aplikacija. Zbog potrebe za standardizacijom CSS koda, organizacije poput W3C[1] tijekom godina radile su na razvoju CSS-a te proširivanju funkcionalnosti objavljivanjem preporuka za implementaciju CSS-a od kojih posljednja nosi oznaku *CSS Level 3* [2]. Nove preporuke dolazile su s novim funkcionalnostima i mogućnostima koje su olakšavale napredni prikaz sadržaja ovisno o tome što se u struci pokazalo potrebnim u posljednje vrijeme. Ipak, usprkos novim specifikacijama jezika—njegovo korištenje i implementacija još uvijek ovisi o proizvođačima Internet preglednika kao što su *Google, Microsoft* i *Mozilla* te često zaostaje za preporukama W3C-a. Osim toga, implementacija novih standarda ovisi i o krajnjima korisnicima, kojima preostaje osvježiti verzije svojih preglednika, što je još jedan korak koji ograničava razvojne timove da koriste pune mogućnosti nove specifikacije CSS-a.

Upravo zbog nabrojanih problema i još bržeg razvoja mogućnosti prikaza u posljednjem desetljeću, došlo je do razvoja alata za procesiranje[2] CSS-a. Prema [3], alati za procesiranje nadogradnja su postojećeg CSS-a i omogućuju korištenje varijabli, operatora, funkcija, *mixin*-a i ostalih naprednih funkcionalnosti. Iako među sobom imaju razlike, svi alati za procesiranje uglavnom rade tako da omogućuju pisanje naprednih funkcionalnosti u svojem modificiranom,

[1] *World Wide Web Consortium* – vidi poglavlje 2.2
[2] Procesiranje (engl. *processing)* – proces obrađivanja koda kako bi se omogućile nove funkcionalnosti koje su inače nedostupne u standardnom jeziku

službeno neispravnom, obliku CSS jezika koji zatim prevode (kompajliraju)[3] (engl. *to compile*) u standardni CSS kod i time omogućuju implementaciju, za standardni CSS kod, još nedostupnih mogućnosti.

U ovom radu bit će riječi o povijesti i razvoju CSS-a, njegovim prednostima i nedostatcima te razvoju alata za procesiranje CSS-a. U četvrtom poglavlju detaljnije će se proći kroz trenutačno najkorištenije alate, razlike među njima i njihove prednosti i nedostatke. Peto poglavlje će se baviti praktičnom implementacijom nekih od nabrojanih alata i metodologija te usporedbom njihovih brzina. Zatim će se u šestom poglavlju obraditi trenutačno aktualne metodologije pisanja CSS koda. Na posljetku, na temelju trenutačnog stanja i razvoja CSS jezika i popratnih alata, u sedmom poglavlju analizirat će se položaj CSS-a u industriji i mogući smjer u kojemu će se jezik dalje razvijati. Zaključak donosi pregled pokrivenih tema i završnu riječ.

2. POVIJEST I RAZVOJ CSS-A

Devedesete godine prošlog stoljeća vrijeme su eksplozivnog razvoja interneta i vezanih tehnologija. Usprkos tome što ~~su~~ u to vrijeme mrežno povezivanje i udaljena digitalna razmjena informacija već neko vrijem bili poznati koncept, tek je krajem prošlog stoljeća došlo do širokopojasnog prihvaćanja računalnih tehnologija i prodora računala i interneta u kućanstva diljem svijeta. Kolijevkom interneta kakvog danas poznajemo smatra se CERN, odnosno Europsko vijeće za nuklearna istraživanja (franc. *Conseil européen pour la recherche nucléaire*). Unutar CERN-a, koji je od svog osnutka 1954. godine bio mjesto mnogobrojnih otkrića u području znanosti, svoje mjesto pronašli su Sir Tim Berners Lee, kreator HMTL-a te Håkon Wium Lie, norveški znanstvenik zaslužan za prvu verziju CSS-a [4].

2.1. Povijest

Kada je Håkon Wium Lie, 1994. godine, predstavio svoj prijedlog za stilski jezik nazvan CSS, koncept stilskih tablica nije bio potpuno nov – raniji oblici takvih tablica postojali su za stiliziranje dokumenata još u 80-tim godinama prošlog stoljeća, ali bile su ograničene, teško proširive i neiskoristive za okružje interneta. Čak je i sam Tim Berners Lee, u svom rudimentarnom

[3] Kompajliranje – proces prevođenja iz jednog programskog jezika u drugi. Izvorno, prevodio se programski jezik u strojni jezik, razumljiv računalu, dok se danas termin koristi za prevođenje bilo kojeg programskog jezika u bilo koji drugi programski jezik. U slučaju CSS-a, proces bi se točnije trebao nazvati *transpajliranje* (engl. *transpiling)* jer su izvorišni i odredišni jezici iste razine apstrakcije, što nije bio slučaj s programskim i strojnim jezicima u prošlosti kada se pojam kompajliranje prvi puta pojavio.

internetskom pregledniku, imao jedan oblik lokalnih stilskih tablica koje nije bilo moguće povezati s dokumentima na internetu pa zbog toga nisu mogle biti korištene u novom okruženju [5].
Iz tog razloga, jedna od glavnih mogućnosti CSS-a, zahvaljujući kojoj je i do danas ostao relevantan, je mogućnost primjenjivanja više različitih tablica na isti dokument koji će biti stiliziran prema pravilima kaskade (poglavlje 3.3). Usporedno s prijedlogom Lieja, osam drugih prijedloga pristiglo je na evaluaciju W3C-a od čega je, uz *Cascading Style Sheets*, prijedlog Berta Bosa pod nazivom *Stream-based Style Sheets Proposal (SSP)* bio na ozbiljnijem razmatranju. Håkon Wium Lie je radio s Yvesom Lafonom kako bi implementirali svoj prijedlog, CSS, u *Arena browser* – internetski pretraživač tada korišten u svrhe testiranja novih standarda [6].

Usporedno s njima, Bert Bos je pokušao implementirati SSP u svoj browser nazvan *Argo*. Kako je ubrzo CSS prihvaćen kao superiornije rješenje, Bos se pridružio Lieju u razvoju CSS-a. Krajem 1996. godine, razvoj prve verzije, nazvane CSS Level 1, bio je gotov i stilski je jezik objavljen u prosincu 1996. godine [6]. Ubrzo nakon toga, do tada jedno tijelo zaduženo za dotadašnji razvoj HTML-a, CSS-a i DOM-a (*Document Object Model*) razdvojeno je u tri posebna odbora, među kojima se nalazio i onaj zadužen za CSS. Već dvije godine kasnije, sredinom 1998., objavljena je osvježena verzija nazvana *CSS Level 2* koja je dugo vremena bila relevantna verzija u skoro svim modernim internetskim pretraživačima. Iako se počelo s razvojem treće verzije CSS-a odmah nakon objavljivanja CSS2, ona službeno i dalje nije dovršena. W3C je modularizirao pristup razvoju CSS-a, pa su tako neki dijelovi davno dovršeni, dok se neki tek dodaju. Danas se pod *CSS Level 3* svrstavaju sve nove funkcionalnosti iako nikada nisu izašle iz razvoja pod zajedničkim imenom kao cjelokupni prijedlog W3C-a.

2.2. W3C

World Wide Web Consortium ili W3C međunarodna je organizacija koja za zadaću ima uvođenje i održavanje standarda na internetu. Organizaciju je osnovao, i trenutačno ju vodi, Tim Berners Lee, kreator HTML-a. Prema [7], organizacija broji preko 400 članova: od organizacija, udruga, tvrtki, fakulteta, državnih vlada i drugih. W3C nadzire i vodi razvoj novih standarda i izdaje nove verzije postojećih – između ostalih i CSS3. *World Wide Web Consortium* je za CSS vrlo bitan jer definira nove standarde sukladno trenutačnim potrebama koje zatim proizvođači internetskih

pretraživača mogu, ali i ne moraju uvrstiti u svoje preglednike. Cjelokupni rad radne grupe W3C-a koja se bavi *Cascading Style Sheets*-ima je dostupan na njihovoj internetskoj stranici[4].

[4] W3C CSS radna grupa (engl. *Working Group*) - https://www.w3.org/TR/tr-groups-all#tr_Cascading_Style_Sheets__CSS__Working_Group

2.3. Internet preglednici

Kao što je spomenuto u poglavlju 2.2, internetski preglednici, odnosno njihovi proizvođači, odlučuju koje će dijelove specifikacija izdanih od W3C-a uključiti u sljedeće verzije svog proizvoda. Upravo je to jedan od fundamentalnih problema koji usporavaju razvoj CSS-a, jer u potpunosti ovisi o implementaciji nezavisnih tvrtki i potrebno ih je osvježiti na svakom korisničkom računalu. Detaljnije o varijablama i njihovoj implementaciji u standardnom CSS-u vidi poglavlje 4.2. W3C je varijable, odnosno prema [8], *Custom properties*, kako se nazivaju u njihovoj specifikaciji, dovršio i objavio kao preporuku za korištenje za sve internetske pretraživače. Usprkos tome, široko se prihvaćanje te preporuke tek događa, pa je prema stranici *Caniuse*[5] moguće vidjeti da trenutačne verzije *Microsoft*-ovih pretraživača *Internet Explorer* te *Edge*, uz *Operu Mini* te ugrađeni pretraživač na operativnom sustavu *Android* verzije 4.4 i niže, uopće ne podržavaju propisane varijable.

Povrh tih problema, odgovornost za pravovremeno ažuriranje preglednika tek je na krajnjem korisniku pa je tako između nove mogućnosti i njenog neometanog korištenja još jedan korak koji je potrebno savladati. Vrlo je izgledno da bi alati za procesiranje CSS-a bili mnogo manje zastupljeni kada bi se nove preporuke W3C-a mogle uvoditi neovisno o proizvođačima internetskih preglednika. Kako to nije slučaj, alati su se pozicionirali kao srednji sloj koji dopušta korištenje novih funkcionalnosti koje se zatim prevode u standardni CSS kod koji je razumljiv i starijim pretraživačima.

[5] www.caniuse.com - Internetska stranica koja daje pregled svih verzija glavnih internetskih pretraživača i stupanj podrške za određenu funkcionalnost.

3. Cascading Style Sheets

CSS ili Cascading Style Sheets je u svom osnovnom obliku relativno jednostavan jezik koji se strukturom i načinom rada nije puno mijenjao od svojih početaka sredinom 90-tih godina prošlog stoljeća. Već postojećim HTML elementima moguće je dodati klase i identifikatore kojima se potom dodjeljuju atributi unutar css datoteke i taj proces je zadnjih 20 godina ostao nepromijenjen. U ovom poglavlju kratko će bit opisan osnovni način rada CSS-a te struktura njegovih atributa, što se događa u posebnim situacijama kada više klasa ili identifikatora pokušava primijeniti svoje atribute na isti HTML element te kako na ispravan način iskoristiti te mogućnosti jezika.

3.1. Klase, identifikatori i atributi

Međupovezanost HTML-a i CSS postiže se uporabom klasa te identifikatora koji se postavljaju u svaki *html* element, nakon čega je taj element moguće stilizirati unutar .css datoteke[6]. Definiranoj se klasi zatim pristupa koristeći točku, a identifikatoru pomoću znaka `#`. Primjer u nastavku je HTML kod s definiranom klasom i identifikatorom[7]:

```
<p class="klasa" id="identifikator">
   sadržaj paragrafa
</p>
```

Dok je u CSS-u pristup tom elementu preko klase i identifikatora prikazan u sljedećem primjeru[8]:

```
.klasa {
// atributi klase
}

#identifikator {
// atributi identifikatora
}
```

Prema [9] klasa i identifikator se razlikuju u tome što je identifikator unikatni pokazivač na samo jedan element unutar dokumenta dok klasa može označavati više elemenata. Svakom od njih se pak mogu aplicirati atributi koji su osnovni elementi rada s CSS-om. Atributima se postižu sve

[6] Kako bi bilo moguće pisati .css kod u izdvojenoj datoteci potrebno je `<link>` atributom povezati vanjsku .css datoteku s *html* datotekom koju se stilizira.
[7] Primjer napisao Sebastijan Dumančić
[8] Primjer napisao Sebastijan Dumančić

funkcionalnosti koje CSS posjeduje, od boje i veličine elemenata do ponašanja na određenim rezolucijama i prijeloma teksta. Ponašanje atributa i njihovo pravilno korištenje propisano je W3C specifikacijom i podložno je interpretaciji internet preglednika, zbog čega je potrebno poznavati kako se njihove implementacije razlikuju među najpopularnijim preglednicima, kako bi krajnji rezultat bio najbliže željenom na svim platformama za koji se sustav razvija.

3.2. Sintaksa

CSS je u svojem osnovnom obliku vrlo striktan jezik što se tiče sintakse i načina pisanja i kao takav je ograničen u rasponu grešaka koje je moguće napraviti prije nego napisani kod više nije ispravan i postane nečitljiv internet pregledniku. Prema [10], svako CSS pravilo sastoji se od selektora i deklaracije unutar koje postoji atribut i vrijednost koju on poprima, kao što je prikazano na Slici 3.1.

Slika 3.1 – Struktura CSS pravila

CSS nije validan ukoliko bilo koji od elemenata navedenih na Slici 3.1. nedostaje za pojedino pravilo, iako neke implementacije alata za procesiranje CSS-a, poput *Stylus*-a, prema [11], omogućuju izostavljanje pojedinih elemenata.

3.3. Kaskade

Kaskada je osnovno svojstvo CSS-a koje omogućuje kombiniranje deklaracija stila iz različitih datoteka, a njena se važnost krije i samom imenu CSS-a – *Cascading Style Sheets*. Atributi unutar CSS datoteke strukturirani su tako da je moguće jednim CSS selektorom promijeniti stil drugog elementa ukoliko je, prema pravilima, taj selektor važniji. U svojem osnovnom obliku, kaskada definira redoslijed primjenjivanja CSS svojstava na element, odnosno koji će se od atributa u

konačnici primijeniti na element ako postoji konflikt. Prema [12] tri su principa prema kojima se određuje kaskada:

1. Važnost
2. Specifičnost
3. Redoslijed

Važnost (engl. *importance*) određuje koliko je određeni atribut bitan za stil elementa prema tome gdje se on nalazi. Primjerice, postoji unaprijed postavljeni stil internetskog pretraživača koji će svaku hipervezu stilizirati tako da bude podvučena i plave boje. CSS kod kojim se to definira nije nigdje napisan od strane razvojnog programera, ali ipak, internetski pretraživači dolaze sa setom unaprijed postavljenih vrijednosti poput stiliziranja linkova ili tipki kako bi se dokumenti bez stilskih tablica ispravno prikazali. Ako HTML dokumentu pridružimo .css datoteku u kojoj je za element `<a/>`[9] definirana druga boja, hiperveza će poprimiti taj stil, zbog veće važnosti vanjske .css datoteke. Svojstvo nasljeđivanja omogućuje zadržavanje već definiranih stilova za dijelove sučelja te dodatno stiliziranje onoga što je potrebno. Također, krajnji korisnik u internet pregledniku može primijeniti svoj stil koji će imati veću važnost i primijenit će se umjesto vrijednosti definirane od razvojnog programera. U tom slučaju, promjena je vidljiva isključivo korisniku koji ju je promijenio.

Specifičnost (engl. *Specificity*) – mjeri koliko je selektor precizan u svojoj deklaraciji, odnosno na koliko bi se elemenata on mogao odnositi. Primjerice, selektor elementa paragrafa `p` može se odnositi na mnogo elemenata na stranici i zbog toga ima nisku specifičnost. Selektor klase ima nešto višu, dok ID selektor ima najvišu specifičnost. Tablica 3.1[10] detaljno prikazuje razine specifičnosti koji mogu postojati za dani element:

[9] HTML element a ili *anchor* označava hipervezu.
[10] Tablica 3.1. preuzeta je s https://developer.mozilla.org/en-US/docs/Learn/CSS/Introduction_to_CSS/Cascade_and_inheritance

Tablica 3.1 – Prikaz nivoa specifičnosti pojedinih CSS selektora

selektor	tisuće	stotine	desetke	jedinice	ukupna specifičnost
`h1`	0	0	0	1	0001
`#important`	0	1	0	0	0100
`h1 + p::first-letter`	0	0	0	3	0003
`li > a[href=*"en-US"] > .inline-warning`	0	0	2	2	0022
`#important div > div > a:hover, inside a <style> element`	1	1	1	3	1113

Svaki selektor ima svoju težinu koja se dodjeljuje ukupnom selektoru kako bi se odredila njegova ukupna specifičnost. U Tablici 3.1, primjerice, zadnji redak ima sljedeću vrijednost:

- specifičnost 1000 ako je stil elementa definiran u HTML dokumentu unutar `<style>` oznaka. Upravo zbog svoje visoke specifičnosti često se izbjegava,
- specifičnost 100 za svaki ID selektor (`#important`) u ukupnom selektoru,
- specifičnost 10 za svaki selektor klase, selektor atributa ili pseudo-selektor (`:hover`) u ukupnom selektoru,
- specifičnost 1 za svaki selektor elementa u ukupnom selektoru.

Zbrojem vrijednosti specifičnosti od 1113 dobije se mjera važnosti posljednje navedenog selektora u Tablici 3.1. što znači da se njegova svojstva primjenjivati sve dok ne postoji drugi selektor s većom specifičnošću koji definira ista svojstva.

Na posljetku, redoslijed (engl. *Source order*) određuje koji stil kojeg selektora će se primijeniti na element ukoliko su dva selektora jednake specifičnosti pokušala primijeniti iste atribute. Primjerice u sljedećem kodu:

```
header > p {
    color: red;
}

header > p {
    color: blue;
}
```

U gornjem primjeru paragrafu `p` dodijeljene su različite vrijednosti koristeći iste selektore iste važnosti. Stil paragrafa odgovarat će zadnje navedenom pravilu, odnosno paragraf će biti plave, a ne crvene boje.

3.4. Nasljeđivanje

Nakon kaskade, nasljeđivanje je posljednji koncept koji određuje stil nekog elementa. Primjerice, prema [12] smisleno je da se atribut `font-family` nasljeđuje među elementima, tako da je jednostavno aplicirati određeni *font* na cijeli dokument postavljanjem `font-family` atributa na `<html>` element preko kojeg će svi ostali elementi naslijediti to svojstvo. S druge strane, logično je da se atributi poput `margin`, `padding`, `border` i `background-image` ne nasljeđuju, jer bi ih bilo potrebno resetirati na svim elementima koji su potomci elementa za koji su inicijalno definirani. W3C je u svojoj specifikaciji definirao za svaki pojedini atribut nasljeđuje li se njegova vrijednost ili ne i ta informacija nalazi se na njihovim internetskim stranicama.
Pored osnovnih vrijednosti za nasljeđivanje, svim atributima moguće je dodijeliti tri dodatne vrijednosti koje mijenjaju kako se nasljeđuju: *inherit, initial* te *unset.*

`Inherit` – atribut koji govori elementu da naslijedi vrijednost elementa unutar kojeg se nalazi. Ako prvi element iznad nema definiran stil koji se treba naslijediti, gleda se sljedeći element iznad i tako sve do glavnog `<html>` elementa.
`Initial` – vraća CSS svojstvo elementa na početno svojstvo definirano od internet preglednika
`Unset` – daje mogućnost resetiranja vrijednosti elementa na početnu vrijednost nakon što je već naslijeđena neka druga vrijednost.

W3C se pobrinuo da je rijetko potrebno modificirati ponašanje nasljeđivanja među elementima, ali ako je to i potrebno, s tri nabrojana atributa vrlo se lako postiže željeni rezultat.

4. CSS PROCESORI

Kao što je ranije objašnjeno (vidi poglavlje 1), CSS je za današnje potrebe donekle ograničen jezik. Kada ga se usporedi s pravim programskim jezicima (poput C#, PHP-a ili *Java*-e), ubrzo se primjećuju problemi koji dovode do teško održivog, nepotrebno ponavljajućeg i nejasnog koda. Nedostatak varijabli, ugnježđivanja, funkcija i drugih osnovnih funkcionalnosti cjelovitih programskih jezika čini CSS jezikom kojim se teško radi na većim projektima zbog teške preglednosti, brzog rasta broja linija i traženja pojedinih elemenata unutar koda. Primjerice, nije neuobičajeno da .css datoteka generirana koristeći alate za procesiranje CSS-a ima nekoliko tisuća linija koda, što datoteku čini izuzetno nepreglednom za uređivati i održavati koristeći samo osnovni CSS kod.

Zbog tih nedostataka je u zadnjih nekoliko godina došlo do ubrzanog razvoja i šireg prihvaćanja alata za procesiranje CSS-a, kao što je vidljivo u statistikama korištenja pojedinih alata za procesiranje prema [13] i [14]. Najpoznatiji predstavnici ove specifične skupine jezika su SASS (*Syntactically Awesome Stylesheets*), LESS, Stylus te PostCSS. Svima njima zajedničko je proširivanje standardnog CSS-a sa dodatnim funkcionalnostima koje su, zatim, u svakom jeziku na svoj način, prevedene u standardni CSS kod, čime se postiže održivost, skalabilnost i veća iskoristivost koda koji je bliži cjelovitim programskim jezicima. U ovom poglavlju usporedno će se analizirati najzastupljeniji alati za procesiranje (CSS-a, SASS, LESS, Stylus te PostCSS) i njihove prednosti i nedostatci te će se izvršiti detaljnija analiza svakog pojedinog alata i njegovih specifičnosti.

4.1. Način rada alata za procesiranje CSS-a

Alati nabrojani u prethodnom poglavlju imaju zajedničkih funkcionalnosti, ali se razlikuju u načinu implementacije u projekt i u načinu rada. SASS je tako pisan u programskom jeziku Ruby, koji dolazi unaprijed instaliran na MacOS operativnom sastavu, dok ga je potrebno dodatno instalirati na Windows i Linux računala. Ovisnost SASS-a predstavlja svojevrsnu prepreku s obzirom da Ruby nije uobičajen jezik za veliki dio razvojnih programera pa je iz tog razloga SASS prepisan i u razne druge programske jezike, od kojih je najpoznatija *libSass*, njegova implementacija u C/C++ programski jezik.

Za razliku od SASS-a, LESS, Stylus te PostCSS su pisani u Javascript-u, odnosno serverskoj implementaciji jezika – *NodeJS*. Uz NodeJS dolazi i *Node Package Manager* odnosno *npm* – repozitorij paketa za NodeJS. Već kratkim pregledom tehnologija potrebnih za razvoj, koristeći alate za procesiranje CSS-a, dolazi se do osnovnog problema korištenja alata za procesiranje – potrebna je dodatna instalacija ili podešavanje koje u svojoj kompleksnosti varira od jednog do drugog jezika, kako bi bilo moguće koristiti sve što on pruža.

Iako sintaksom vrlo sličan standardnom CSS kodu, kod napisan u bilo kojem od gore navedenih alata za procesiranje CSS-a nije validan i internet preglednik ga ne može pročitati primijeniti na HTML dokument. Kako bi internet preglednici znali pročitati taj kod, potrebno ga je obraditi, odnosno procesirati (od čega i dolazi naziv alata koji se koriste) kako bi ponovno postao validan CSS kod. Upravo zbog toga što nije potrebno pisati validan CSS kod, moguće je koristiti napredne funkcionalnosti koje pružaju alati za procesiranje CSS-a s obzirom da će se u konačnici prevesti u trenutačno ispravan i validan CSS kod.

Kao što je već spomenuto, svaki od alata radi obradu koda na svoj način i međusobno se razlikuju u potrebnim koracima prilikom postavljanja projekta, načina procesiranja te brzine kojom se samo procesiranje izvodi.

4.2. Glavne mogućnosti alata za procesiranje CSS-a

Svi do sad nabrojani alati donose slične mogućnosti na sebi svojstven način, ovisno o sintaksi i načinu pisanja. Jedan od glavnih koncepata svih programskih jezika – varijabla, (do nedavno nepostojeća u CSS-u) dolazi tek s alatima za procesiranje CSS-a. Iako je koncept varijable osnovni dio svih programskih jezika, varijable sum prema [8], tek nedavno uvrštene u preporuke CSS-a od strane W3C-a. U slučaju drugih programskih jezika (poput *Javascipt*-a), puno je jednostavnije uvođenje i prihvaćanje novih funkcionalnosti, dok je u slučaju CSS-a, prihvaćanje novih funkcionalnosti dugotrajan proces koji ovisi o proizvođačima internetskih preglednika i korisnicima (vidi poglavlje 2.3).

Prema tome, moguće je zaključiti da će službena podrška CSS-a barem još neko vrijeme biti u zaostatku zbog dodatnih koraka potrebnih za predstavljanje i prihvaćanje novih funkcionalnosti.

Sljedeći primjer prikazuje kako se problem varijable rješava koristeći SASS-u:

```
$primary-color: #333;

h1 {
    color: $primary-color;
}
```

Navedeni će se primjer koda procesirati[11] u standardni CSS kod u nastavku:

```
h1 {
    color: #333;
}
```

Gdje je varijabla `$primary-color` unutar `h1` elementa zamijenjena vrijednošću varijable, odnosno `#333`. Iako vrlo jednostavan, ovaj primjer demonstrira kako se osnovne vrijednosti u projektu mogu postaviti unaprijed i zatim se koristiti gdje god je potrebno. Osim toga, najvažnije i glavno svojstvo varijabli mogućnost je promjenjivosti njezine vrijednosti koja se mijenja na jednom mjestu, a njena generirana vrijednost se mijenja svugdje gdje je korištena.

Osim podrške za varijable, svi alati za procesiranje CSS-a dijele i mogućnost ugnježđivanja (engl. *nesting*). Prema [15], ugnježđivanje je pisanje bloka koda unutar drugog bloka za koji je semantički vezan. CSS u svojem osnovnom obliku podržava semantičko ugnježđivanje, kao primjerice u sljedećem kodu[12]:

```
section h1 {
    color: red;
}
```

čime se atribut primjenjuje samo na `h1` naslove postavljene unutar `section` *tag-a* HTML-a. Problem nastaje kada je potrebno aplicirati više atributa na više specifičnih elemenata ugniježđenih unutar drugih elemenata. U nastavku je primjer takve situacije napisan u običnom CSS-u[13]:

```
section {
    width: 100%;
}

section header {
    text-align: center;
}
```

[11] Procesiranje koda pojašnjeno je detaljnije u poglavlju 4.1
[12] Primjer napisao Sebastijan Dumančić
[13] Primjer napisao Sebastijan Dumančić

```
section item {
    float: left;
    width: 33%;
}

section item img {
    max-height: 300px;
}
```

i zatim u PostCSS[14]-u:

```
section {
    width: 100%;

    & header {
text-align: center;
}

& item {
    float: left;
    width: 33%;

            &:img {
                    max-height: 300px;
            }
    }
}
```

U primjeru je jasno vidljiva razlika između dva stila pisanja. Ugnježđivanjem se dobije jasna semantička struktura elemenata i očito je vidljivo koji element je pozicioniran unutar kojeg u HTML kodu. Pisanjem na ovaj način izbjegava se nepotrebno ponavljanje istih klasa (`section`) nego se s znakom `&` ponavlja bilo koja klasa unutar koje se on nalazi.

Treća funkcionalnost je *mixin*, koja bi se, prema [16], najbliže opisala funkcijama u klasičnim programskim jezicima poput *Javascript*-a. *Mixin*-i su setovi pravila koji se mogu pozvati unutar drugih pravila sa proslijeđenim parametrom čiju vrijednost poprima generirani CSS. Primjer u nastavku demonstrira ovu funkcionalnost napisanu u *LESS*-u[15]

[14] U osnovnim funkcionalnostima alata za procesiranje CSS-a, sintaksa se vrlo malo razlikuje pa je moguće naizmjence koristiti jezike uz vrlo malo promjena.

[15] LESS se u ovom primjeru razlikuje sintaksom od ostalih alata za procesiranje CSS-a tako da se *mixin* u drugim alatima poziva s naredbom @include

```
.bordered(@width) {
    border: @width solid #333;

    &:hover {
    border-color: #999;
}
}

section {
    .bordered(5px);
}
```

Što se u generiranom CSS prikazuje kao:

```
section {
    border: 5px solid #333;
}

section:hover {
    border-color: #999;
}
```

Gore navedeni primjer demonstrira princip korištenja *mixin*-a i koncept ugnježđivanja. Iako su *mixin*-i najlakše opisani kao funkcije, njihova je specifičnost u tome što ispisuju više unaprijed napisanih vrijednosti unutar selektora u kojem su pozvani. Osim njih, postoje i CSS funkcije koje se razlikuju po tome što vraćaju samo jednu vrijednost kao rezultat funkcije. Primjer u nastavku napisan je u SASS-u[16]:

```
@function zbroj-brojeva($prvi-broj, $drugi-broj) {
    @return $prvi-broj + $drugi-broj
}

section {
    padding: zbroj-brojeva (20px, 10px);
}
```

Što se generira u CSS kod u nastavku:

```
section {
    padding: 30px;
```

[16] Primjer napisao Sebastijan Dumančić

```
}
```

U prethodnom su primjeru u SASS kodu atributu `padding` proslijeđene dvije vrijednosti – `10px` i `20px` koje su se, koristeći funkciju `zbroj-brojeva` ispisale kao jedinstvena vrijednost od `30px` u selektoru `section`. Funkcijama se na taj način vrlo lako mogu aplicirati različiti stilovi istog tipa umjesto generiranja velikog broja sličnih klasa unutar kojih se samo iznos istih atributa razlikuje.

4.2.1. Prednosti i nedostatci alata za procesiranje CSS-a

Prednosti alata za procesiranje CSS-a su velike, a prema [17] sežu od već nabrojanih mogućnosti kao što su varijable, funkcije i *mixin*-i do preglednije strukture .css datoteke zahvaljujući ugnježđivanju. Samim time što su alati za procesiranje CSS-a poprimili na važnosti u kratkom periodu od kada su se pojavili te činjenicom da se koriste među najvećim timovima koji razvijaju internet aplikacije (*Facebook, Google, GitHub,* itd.) jasno je da imaju svoje mjesto u modernom okruženju programera. Ipak, sa sobom donose neke poteškoće koje donekle kompliciraju rad, prvenstveno u pogledu dodatnih programa, paketa i ovisnosti (engl. *dependancy*). Česti primjer toga je *node package manager* (NPM) spomenut u potpoglavlju 4.1 koji služi za korištenje gotovih dijelova koda kako bi se izbjeglo ponovno pisanje kompleksnijih stvari koje su već negdje riješene. Nije neuobičajeno za *npm* mapu da sadrži više desetak tisuća datoteka koje su međusobno ovisne jedna o drugoj čime jednostavni projekt može postati velik i ponekad suviše kompliciran. Osim toga, potrebno je podesiti i samo okruženje u kojem radi pojedini alat za procesiranje, bio to NodeJS, Ruby ili nešto treće, što zahtjeva barem djelomično poznavanje jezika ili, u najmanju ruku, postojanje detaljnijih uputa za postavljanje projekta.

Zaključno, iako alati za procesiranje CSS-a unose mnoštvo prednosti, strukture i preglednosti u velike projekte, sa sobom donose veću kompleksnost projekta, tako da je na posljetku na razvojnom programeru za odlučiti vrijede li sve navedene prednosti vrijeme potrošeno za podešavanje i održavanje takvog projekta.

4.3. SASS, LESS, Stylus

Alati za procesiranje, CSS-a, SASS, LESS, Stylus i PostCSS, danas su uglavnom u razvijenim i stabilnim inačicama. Primjerice SASS, koji se pojavio 2006. godine, ulazi u svoje drugo desetljeće postojanja. Kao što je bilo vidljivo u primjerima u poglavlju 4.2, glavni predstavnici su Sass, Less te Stylus, uz PostCSS koji je nešto noviji – o čemu će biti više riječi u poglavlju 4.4 Svi navedeni alati dijele glavne funkcionalnosti uz nešto drugačiji pristup sintaksi, načinu instalacije i korištenju. Tvorci *Stylus*-a su, primjerice, smatrali da je nepotrebno ponavljati zagrade i točke-zarez (;) prilikom deklariranja atributa, što je u drugim alatima, te standardnom CSS-u obavezno. U sljedećim poglavljima bit će opisane osnovne funkcionalnosti koje nabrojane alate čine drugačijima.

4.3.1. SASS

SASS ili *Syntactically Awesome Stylesheets* alat je za procesiranje CSS-a koji je najstariji među nabrojanima, prema [18], prvi puta se pojavio 2006. godine i do sada je prošao 3 glavne verzije. U svojoj prvoj verziji, SASS je izgledao dosta drugačije od standardnog CSS-a, tj. pisao se bez nekih elemenata, primjerice:

```
section
    width: 100%
    color: red
```

Način pisanja u gornjem primjeru, isti kojeg koristi i *Stylus* u svojoj posljednjoj verziji, bio je jedini način pisanja SASS koda sve do verzije 3, kada je uvedena SCSS sintaksa, koja je zapravo identična običnom CSS-u uz dodatke koji omogućuju napredne funkcionalnosti SASS-a. Trenutačno su obje sintakse podržane i razvijaju se.

Nadalje, SASS je u originalu pisan u programskom jeziku *Ruby* koji je i dalje glavna metoda pokretanja i *kompajliranja* .scss koda u standardni .css. Programski jezik *Ruby* objektno je orijentirani program koji je nastao u 90-tim godinama prošlog stoljeća i u ovom slučaju se koristi se za instalaciju i procesiranje .scss koda.

Kako je bio prvi, SASS je postavio dosta standarda za ostale alate koji su dolazili nakon njega - većina funkcionalnosti nabrojanih u ovom poglavlju prvo su se pojavile u SASS-u. Ipak, upravo zbog toga što je prvi, alat ima nekoliko problema koji su i uzrokovali daljnje širenje drugih alata. Najvažnija stvar oko SASS-a je da mu je potreban *Ruby* kako bi se inicijalizirao i *kompajlirao*, barem u svojem osnovnom obliku. *Ruby* ne dolazi unaprijed instaliran na Windows računalima, zbog čega je potrebno instalirati dodatne alate što dodaje kompleksnost i nepoznanicu u inače poznato razvojno okruženje za razvojne programere koji se do sada nisu susretali sa *Ruby*jem.

Jedna od najvrjednijih mogućnosti SASS-a su, već spomenute, funkcije. Osim što podržava kreiranje novih funkcija, SASS dolazi s već unaprijed postavljenim setom često korištenih funkcija. Primjerice, ako je u standardnom CSS kodu potrebno posvijetliti hipervezu kada se pokazivačem pređe preko nje, sljedeći kod je potreban:

```
a {
    color: #333;
```

```
}

a:hover {
    color: #999;
}
```

Odnosno potrebno je poznavati koja je točno trenutačna broja hiperveze i potrebno je odabrati svjetliju boju i nju dodijeliti koristeći pseudoselektor *hover*. Isti primjer u SASS-u izgleda ovako:

```
$hyperlink = #333;

a {
    color: $hyperlink;

    &:hover {
           color: lighten($hyperlink, 20%);
    }
}
```

Primjerom u SASS-u generira se (približno) jednaki rezultat, ali nije potrebno birati i poznavati novu boju koja se dodjeljuje *hover* stanju. Za jedan element prednost možda i nije očita, ali na velikim projektima u kojima ima na desetke *hover* stanja na desecima različitih boja, ušteda vremena na inicijalnom postavljanju je velika. Do pravog problema u velikim projektima dolazi kada je potrebno promijeniti boju nekog elementa koji ima *hover* stanje jer bi bilo potrebno generirati i novu svjetliju (ili tamniju) boju za i tako za svaki element kojem se boja mijenja. Korištenjem varijabli i funkcija, taj cijeli proces se smanjuje na promjenu vrijednosti boje na mjestu varijable i ostalo se odvija automatski.

4.3.2. LESS

LESS je nastao 2009. godine i bio je među prvim alternativama do tada već raširenom SASS-u. Prema [19] u svojim prvim verzijama bio je također pisan u *Ruby*ju, ali je ubrzo prebačen na, danas sve više relevantan, Javascript[17]. Iako je prvenstveno zamišljen za korištenje unutar internetskog preglednika, Javascript se koristi za instalaciju i *kompajliranje* LESS-a putem svoje serverske varijante nazvane *NodeJS*, odnosno njegovog paketnog organizatora *node package manager* ili *npm*. Od SASS-a se razlikuje u sintaksi tako što varijable označava s simbolom @, uz ostale manje razlike. S bržim *kompajlerom* u vidu *Javascript*-a i manje glavnih funkcionalnosti, LESS je brži u

[17] *Javascript* je skriptni programski jezik čija je inicijalna svrha bila uvođenje interaktivnosti u *web* sučelja. U posljednjem desetljeću doživljava izuzetni rast s razvojem alata za pisanje cijelih aplikacija kao što su *AngularJS, ReactJS* i slični.

istim stvarima, dok onemogućava korištenje nekih naprednih funkcionalnosti, kao što su logičke petlje.

4.3.3. Stylus

Stylus se smatra četvrtim najkorištenijim alatom za procesiranje CSS-a, uz ostale nabrojane u poglavlju 4.3 i 4.4 Najviše se razlikuje od ostalih prema sintaksi jer dopušta izostavljanje zagrada prije nabrajanja atributa, točke-zarez na kraju pojedinog atributa pa i dvotočku prilikom deklaracije atributa. Zbog sveg tog nije neobično vidjeti kod kao u nastavku koji je neispravan u većini drugih jezika koji se koriste u alatima:

```
h1
    font-size 2em
    color red
```

Osim toga, varijable se u Stylusu deklariraju na sljedeći način:

```
primaryColor = "#f0f0f0"

h1
    color primaryColor
```

Ideja kod ispuštanja nabrojanih oznaka je da se isto postiže podvlačenjem atributa pod selektor te da je nepotrebno pisati iste stvari kod svakog atributa. S jedne strane, ostali alati za procesiranje prihvatili su to kao dobru ideju pa postoje proširenja koja omogućavaju slične funkcionalnosti u PostCSS-u i SASS-u. S druge je strane, otežano je traženje grešaka (engl. *debugging*) i teži je pregled većih dijelova koda jer je manje očito gdje jedan atribut, odnosno selektor, završava, a drugi počinje.

4.4. PostCSS

Nakon što su SASS, LESS i Stylus prihvaćeni kao korisni, a ne samo prolazni alati, bilo je očito da će se novi alati nastaviti pojavljivati. Kako je sve više ljudi počelo koristiti spomenute alate, pojavila se potreba za prilagodbom pojedinačnim skupinama razvojnih programera kojima je potreban specifični set funkcionalnosti. Tako se 2013. godine pojavio PostCSS kao rješenje koje je prilagodljivo pojedinačnom projektu. PostCSS se temelji na proširenjima (engl. *plugin*) za gotovo sve funkcionalnosti koje su standardne u SASS-u i drugim alatima. Taj se pristup

opravdava time da si svaka osoba, tim ili tvrtka može prilagoditi sustav za procesiranje CSS-a upravo prema onome što je njima potrebno. Iako najnoviji, pa stoga nešto manje prihvaćen od ostalih, može se reći da je PostCSS najnapredniji alat za procesiranje CSS-a zbog svoje proširivosti koja mu daje skoro neograničeni set funkcionalnosti. Prema [20] PostCSS trenutačno broji preko 200 paketa za proširenje koji pokrivaju i nadmašuju funkcionalnosti ostalih alata za procesiranje.

Jedno od najkorištenijih proširenja – *Autoprefixer* – dobar je primjer jednostavnog proširenja koje znatno olakšava pisanje koda. *Autoprefixer* omogućuje pisanje samo jednog tipa atributa za atribute koje je inače potrebno dodati prefikse kako bi bili validni za sve internet preglednike.

```
a {
    display: flex;
}
```

se ispisuje u validni CSS u nastavku koji je potreban kako bi kod radio u ispravno u *Internet Explorer*-u, *Microsoft Edge*-u, *Mozilli Firefox* i drugima:

```
a {
    display: -webkit-box;
    display: -webkit-flex;
    display: -ms-flexbox;
    display: flex;
}
```

Osim *autoprefixer*-a, neki od najkorištenijih proširenja su:
`lost-grid` koji omogućuje pisanje naprednog sustava ćelija za strukturiranje sadržaja na stranici baziranog na `calc` funkciji koja računa postotke elementa unutar kojeg se nalazi pa se time skalira s veličinom ekrana na kojem se nalazi,
`stylelint` – proširenje koje prati napisani kod i upozorava na greške prilikom pisanja,
`colorfunction` – koji omogućava modificiranje boja prema intenzitetu, zasićenosti, prozirnosti itd, te mnogi drugi dostupni na *GitHub*[18] stranicama projekta.

PostCSS je pisan u Javascript-u i razlikuje se od ostalih jer sam po sebi nije alat za procesiranje CSS-a, nego je razvojni okvir koji sadrži *parser*[19] i omogućuje pisanje proširenja koji zapravo vrše

[18] https://github.com/postcss/postcss
[19] Parser – dio programa koji razdvaja set informacija u dijelove koji se potom lakše mogu obraditi drugim programima (http://searchmicroservices.techtarget.com/definition/parser)

pojedinačne funkcionalnosti ostalih alata za procesiranje CSS-a. Upravo zato što omogućuje korištenje samo onoga što je potrebno u pojedinom projektu, PostCSS postiže zavidne brzine izvođenja i veličine samog paketa alata. Prema [21] veličine pojedinih alata su sljedeće:

- *SASS (libSass)* – 110 datoteka, 21 300 linija koda,
- *Stylus:* 72 datoteke, 7 900 linija koda,
- *LESS:* 105 datoteka, 9 800 linija koda.

U usporedbi s tim, PostCSS raste kako rastu proširenja koja se uključe u nekom projektu. Primjerice, mogućnost ugnježđivanja postiže se uključivanjem proširenja `postcss-nested` koje je, prema [21] napisano u 68 linija koda. Varijable su definirane u proširenju `postcss-simple-vars` koje sadrži 74 linije koda, dok su *mixin*-i sadržani u 147 linija koda paketa `postcss-mixins`. Iako je ukupan broj dostupnih proširenja u ovom trenutku preko 300, već se s 20-ak postiže set funkcionalnosti ekvivalentan drugim alatima za procesiranje, uz puno manji broj linija koda i brže izvođenje jer se izostavljaju funkcionalnosti koje se ne koriste u pojedinom projektu. Također prema [21], brzine procesiranja su sljedeće[20]:

```
PostCSS:   45 ms
libsass:   101 ms  (2.2 puta sporije)
Less:      136 ms  (3.0 puta sporije)
Stylus:    175 ms  (3.9 puta sporije)
Ruby Sass: 1731 ms (38.7 puta sporije)
```

Iz sveg navedenog vidljivo je da je PostCSS najfleksibilniji, najlakše proširiv i, u nekim slučajevima, najbrži alat koji će zasigurno u sljedećih nekoliko godina dobiti na popularnosti.

[20] Druga, nezavisna, testiranja osporavaju ovoliku razliku u brzini na temelju toga što je za test korišten limitirani set funkcionalnosti za koje je poznato da vrlo brzo rade u PostCSS-u. Stoga, ovaj primjer se treba uzimati kao idealni slučaj, a ne pravilo.

5. PRAKTIČNI RAD

Kao što je bilo moguće zaključiti iz prethodnih poglavlja, postoje razlike između alata za procesiranje CSS-a i svaki od njih ima razlog kada bi se trebao koristiti. Primjerice, PostCSS se pokazao kao prikladan kada nije potrebno puno mogućnosti koje pružaju ostali alati nego samo pojedine funkcionalnosti, potrebne za neki specifični projekt.

Pored funkcionalnosti koje alati za procesiranje CSS-a pružaju, načina njihove implementacije te održivosti projekata temeljenih na pojedinim alatima, brzina obrađivanja koda često je kriterij kojim razvojni programeri pokušavaju odlučiti koji je alat *bolji* od drugog. Brzina se, u ovom slučaju, odnosi na vrijeme potrebno da se generira krajnja CSS datoteka od datoteke (ili više njih) u kojoj se piše kod za specifični alat za procesiranje CSS-a. Kao što je detaljnije objašnjeno u poglavlju 4.1, CSS kod pisan u nekom od alata potrebno je prvo kompajlirati kako bi postao validnim kodom kojeg internetski pretraživači mogu pročitati, a upravo je za taj korak potrebno određeno vrijeme koje se razlikuje između pojedinih alata. Iako se u pravilu radi o promjenama reda veličine ispod jedne sekunde, prilikom razvoja stranice se ta akcija izvršava pri svakom spremanju datoteke i osvježavanju CSS koda u internetskom pretraživaču, na što se zna čekati i više stotina puta u razvoju jednog sustava.

Prema [21], PostCSS se pokazao kao alat za procesiranje CSS-a s najbržim vremenom obrade CSS koda, što je s druge strane osporavano, prvenstveno u testovima koji uključuju C++ implementaciju SASS-a, nazvanu *libSass*, koja se često uzima kao najbrže rješenje. Iz tog razloga, u ovom će poglavlju brzina obrađivanja koda pojedinih alata za procesiranje CSS-a biti testirana na nekoliko načina. U prvom testu će se obrađivati CSS kod za internetsku stranicu u prilogu, koja je napisana za potrebu ovog testa. Zatim će se obrađivati automatski generirani kod koji bi trebao izdvojiti jake i slabe strane kod pojedinih alata za procesiranje CSS-a. Bitno je za napomenuti da postoji nekoliko specifičnosti u testovima u nastavku koje za svrhu imaju standardizaciju testova kako bi rezultati bili relevantni.

Prvo, u CSS kodu pisanom za prvi test nisu korištene napredne funkcionalnosti koje pružaju alati za procesiranje CSS-a. Svrha prvog, kontrolnog testa, bila je simulirati manji projekt i predstaviti kako se pojedini alati nose s malim i nekompliciranim kodom. Također, prvim testom se prikazuju razlike u brzini obrade jednostavnog i relativno malog koda kako bi se uspostavile razlike između efikasnosti svakog alata. Osim toga, izbacivanje naprednih funkcija omogućava korištenje najbrže

verzije PostCSS-a – one bez dodatnih mogućnosti, kako bi se provjerilo isplati li se koristiti upravo PostCSS i njegovu mogućnost obrade koda sa specifičnim funkcionalnostima koje su potrebne pojedinom projektu.

Drugo, najveći utjecaj na brzinu obrade koda ima veličina datoteke, odnosno broj linija. Kod projekta u prilogu se sastoji od ~600 linija koda, što je manje od velikih, dugogodišnje održavanih projekata u kojima korištenje alata za procesiranje dolazi do velikog izražaja. Iako manja veličinom, priložena CSS datoteka i dalje prikazuje razlike u brzini obrade koda između pojedinih alata, uz očekivano skaliranje pri većim projektima. Kako bi se simulirao veliki projekt čija je veličina bitna za prikazati performanse alata, a pisati projekt te veličine čisto za test je izvan obima ovog rada, test će se ponoviti i na druga dva seta podataka koji čine automatski generirane klase kojima je moguće generirati dokument proizvoljne veličine. Koristeći programski jezik Python[21] kreirana je skripta koja generira različite setove podataka nad kojima se kasnije vrši obrada koristeći pet različitih alata za procesiranje CSS-a.
Na posljetku, u projektu je korištena BEM (*engl.* Block Element Modifier) metodologija pisanja CSS-a, a koja je detaljnije opisana u poglavlju 6.2.

5.1. Metoda testiranja

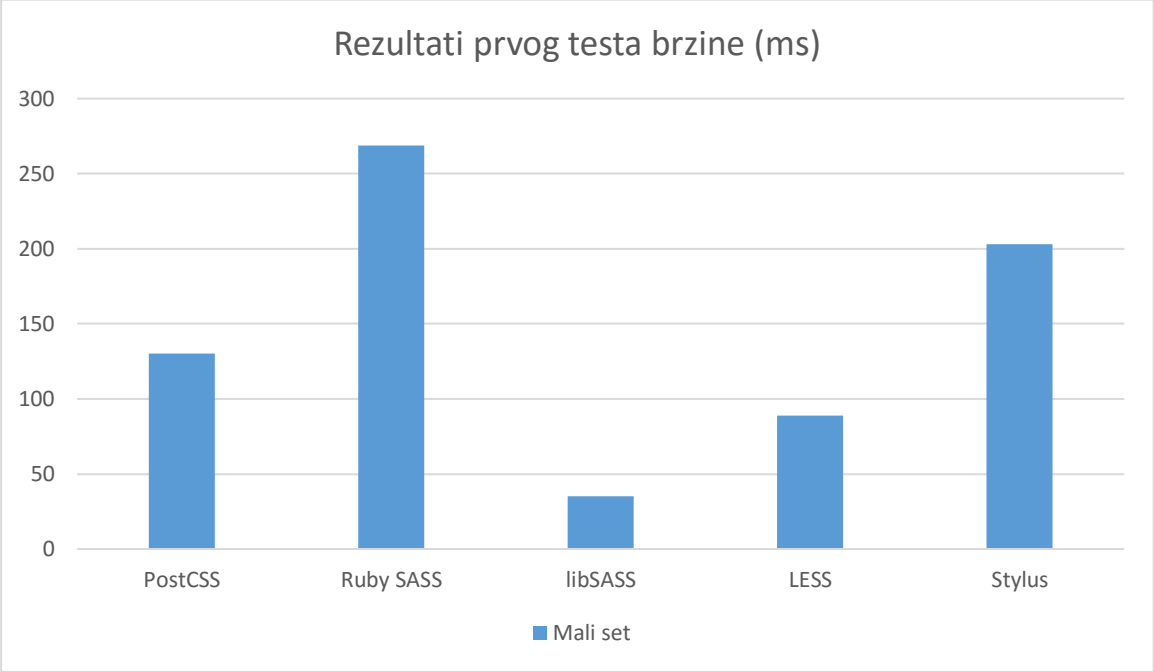
Kako bi se dobili konzistentni rezultati, napisani kodovi, pravi i generirani, obrađivati će se 10 puta za redom, iz čega će se izdvojiti prosječna brzina za pojedinačni alat, zaokružena bez decimala[22]. U testu su korišteni Ruby Sass, libsass, LESS, Stylus te PostCSS. Za obradu i vršenje mjerenja koristi se alat za automatizaciju *Gulp*[23] koji nakon svakog uspješnog zadatka ispiše u konzolu vrijeme koje je bilo potrebno za provedbu tog zadatka, a to je vrijeme zatim prebačeno u stupčasti prikaz s vremenom naznačenim u milisekundama. Korištena skripta se također nalazi u projektu pod imenom *gulpfile.js*.

[21] Programski jezik otvorenog koda nastao u 1990-tim godinama.
Više informacija dostupno na službenim internetskim stranicama projekta: https://www.python.org/

[22] Detaljni rezultati dostupni su u Prilogu 1.
[23] Alat i pripadajuća dokumentacija su dostupni na https://github.com/gulpjs/gulp

5.2. Rezultati

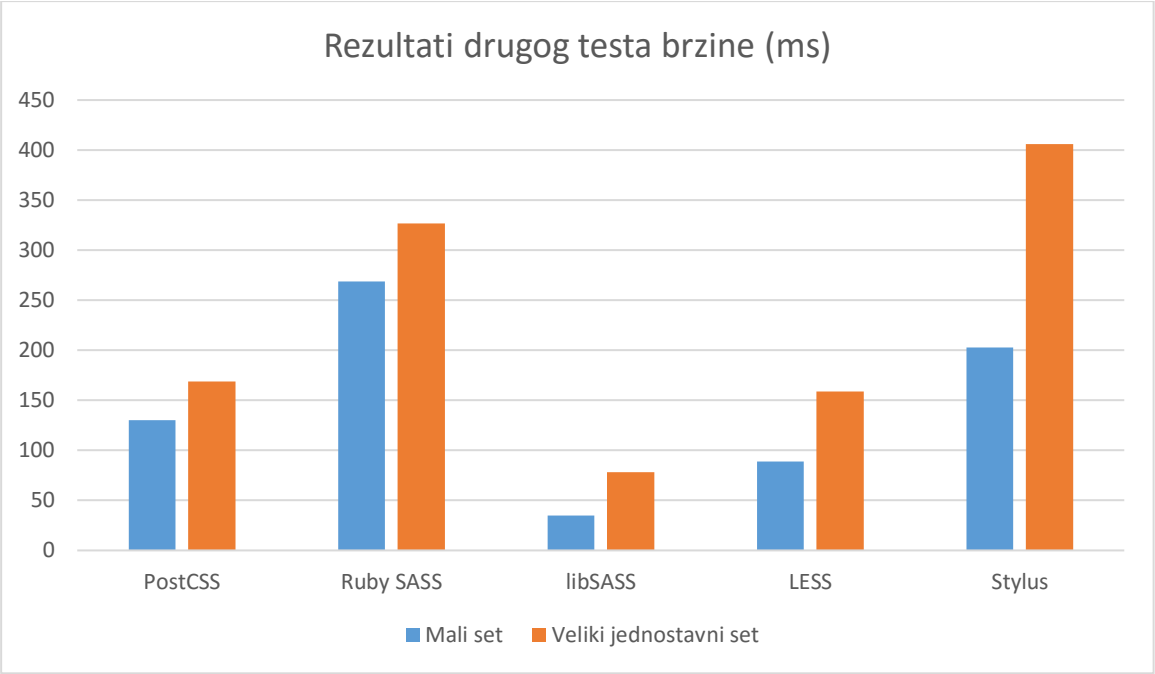
Prvo, bilo je bitno pronaći početne vrijednosti za brzine alata prilikom obrade jednostavnog i kratkog koda. Slika 5.1. prikazuje rezultate prvog mjerenja:



Slika 5.1. Rezultati prvog testa brzine

Analizom Slike 5.1. može se zaključiti nekoliko stvari:

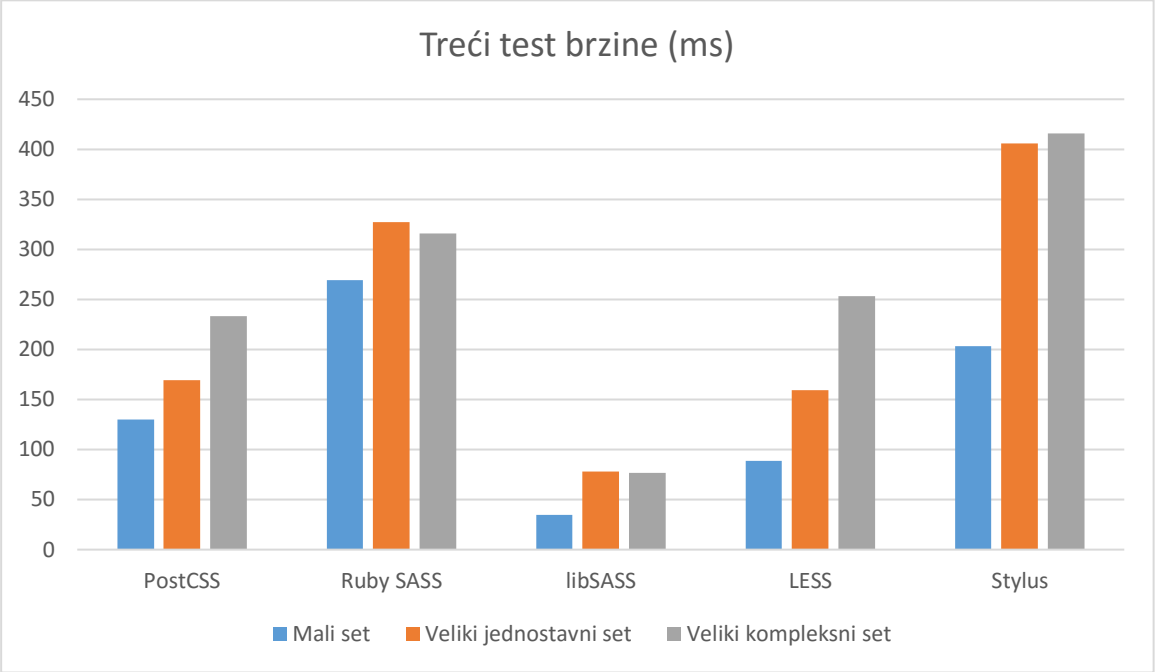
Prvo, libSass je očito najbrži, pogotovo ako se uspoređuje s originalnom implementacijom SASS-a pisanoj u Rubyju. Drugo, Ruby Sass i Stylus se mogu izdvojiti kao očito sporiji od pet testiranih alata, s vremenima preko 200ms i za vrlo malu datoteku bez dodatnih funkcionalnosti. Ovaj kontrolni test dobro prikazuje razlike između pojedinih alata koje u ovom slučaju ovise isključivo o njihovoj implementaciji, jeziku u kojem su pisani i okruženju u kojem se izvode.
Rezultati sljedećeg testa prikazuju brzinu pojedinih alata prilikom obrade koda sličnog prethodnome, uz razliku što je automatski generiran i dužine ~6500 linija koda, odnosno desetak puta više od prvog primjera. U ovom testu također nisu korištene napredne funkcionalnosti alata, nego se testira skaliranje alata s povećanjem broja linija koda u datoteci koju obrađuju. Slika 5.2. prikazuje rezultate:



Slika 5.2. - Rezultati drugog testa brzine

Iz drugog testa očit je porast vremena obrade koda sa porastom broja linija koda. Porast je linearan i više-manje jednak uz iznimku Stylusa koji je prilikom obrade manjeg seta bio brži od Ruby Sass-a, dok je u ovome testu vidljivo sporiji. Libsass i dalje zadržava prednost pred svima, čak toliko da je u manje vremena obradio 10 puta veću datoteku od drugih alata. Ova dva testa prikazuju najbitniju stavku kod obrade CSS koda – povećanjem broja linija koda direktno utječemo na brzinu obrade čime dolazimo do zaključka da je iznimno bitno pisati koncizan i što jasan kod.

Na posljetku, u trećem se testu uvodi dodatna kompleksnost u vidu korištenja naprednih funkcionalnosti koju pružaju svi alati. Koristeći skriptu iz Priloga, programskim jezikom Python generiran je set klasa jednake dužine kao u prethodnom primjeru uz iznimku da postoji pet slučajeva; od jednostavne klase s jednim atributom do varijabli, *mixin*-a, tri nivoa ugnježđivanja te korištenja *autoprefixer*-a. Uz nabrojano, koristeći petlje, svaka generirana klasa ima jedinstveno ime čime se izbjeglo eventualno preskakanje ili ubrzavanje obrade koda koje bi alati koristili ukoliko prepoznaju ponavljajuće pravilo. Krajnje generirani kod sastoji se od jednakog broja svih vrsta klasa, od jednostavnih pa do najkompleksnije generiranih. Rezultati ovog testa nalaze se na Slici 5.3.



Slika 5.3 – rezultati su iskazani u milisekundama (*ms*)

Slika 5.3. donosi neke očekivane, ali i neke neočekivane rezultate. Očekivano je dakako daljnji rast vremena potrebnog za obradu koda koristeći PostCSS. Dok se u prva dva testa koristila osnovna verzija ovog alata, za posljednju je, zbog dodanih funkcionalnosti, bilo potrebno uključiti četiri dodatna proširenja – `autoprefixer`, `mixins`, `simplevars`, te `nested`, koji dodaju funkcionalnosti automatskog prefiksiranja određenih atributa, korištenja mixina, varijabli te mogućnost ugnježđivanja. Svako od tih proširenja usporava rad samog alata pa je tako krajnje vrijeme obrade 60ms duže nego na prethodnom testu. Također očekivani rezultat su jednaki (ili unutar margine pogreške) rezultati između drugog i trećeg testa za Ruby, libSass te Stylus zbog toga što alati teoretski ne bi trebali duže obrađivati kod koji ima dodatne funkcionalnosti jer kod za njihovu obradu već sadrže, koristile se te funkcionalnosti ili ne. S druge strane, iznenađujući je rezultat LESS-a koji pokazuje povećanje od 96ms, odnosno oko 60% nakon dodavanja ugnježđivanja, varijabli i ostalih stvari nabrojanih na početku poglavlja. Iz trećeg testa možemo zaključiti kako su PostCSS i LESS dobar izbor alata za obradu koda koji ne koristi puno naprednih funkcionalnosti, dok je libSass, barem brzinom, odličan za sve slučajeve. S druge strane, Ruby Sass te Stylus pokazuju očito sporije vrijeme obrade koda.

Zaključno, testovi pokazuju većinom očekivane stvari poput linearnog rasta vremena potrebnog za obradu sve većih datoteka ili najsporijeg vremena Ruby Sass-a što je očekivano s obzirom na

korištenje Ruby-ja koji se pokazao manje efikasnim od implementacije istog alata u C-u (Libsass). Ipak, testom je prikazan utjecaj kompleksnosti koda na vrijeme obrade koristeći LESS te nagli porast vremena obrade koda koristeći Stylus u drugom testu – oba neočekivana rezultata. Osim toga, zaključuje se da vrijeme obrade koda koristeći sve navedene alate ne bi trebalo biti ograničavajući faktor u njihovom korištenju, s obzirom da su u svim slučajevima kod obradili za manje ili jednako od 400ms što je dovoljno kratko vrijeme da se uglavnom ne primijeti.

6. CSS METODOLOGIJE

Godinama je glavna primjedba CSS-u bila njegova skalabilnost. Čak i s uvođenjem alata za procesiranje koji su detaljno opisani u poglavlju 4 u velikim sustavima s više stotina modula, stranica i još više elemenata, jednostavne stvari poput imenovanja elemenata postanu problematične ako ne postoji standardizirani način imenovanja novih elemenata [22].
Jedna od osnovnih funkcionalnosti programskih jezika lokalno je deklariranje varijabli, odnosno njihov *namespacing*, koji u CSS-u jednostavno ne postoji. Selektori su u CSS-u, neovisno o korištenom alatu za procesiranje, globalni, što znači da se njihovoj definiciji može pristupiti sa cijele stranice. Primjerice, kodom u nastavku:

```
p {
    color: red;
}
```

Dodijeljen je stil svim paragrafima na cijeloj stranici. Ukoliko se koristi selektor sa visokom specifičnošću[24] dolazi se do situacije u kojoj se teško može dodijeliti novi stil paragrafu negdje dalje na stranici. Ubrzo postane potrebno koristiti `!important` atribut koji mijenja standardnu strukturu kaskade i time poništava jedno od glavnih načela CSS-a.
CSS metodologije, za razliku od alata za procesiranje, nisu alati koji se instaliraju i potom koriste, već skup smjernica za pisanje koda na skalabilan i održiv način. Osim toga, među sobom imaju mnogo više razlika nego alati za procesiranje, te skup pravila koji definiraju pojedinu metodologiju nije strog i može se prilagoditi prema potrebi projekta i/ili programera.

Neovisno o korištenoj metodologiji, implementacija bilo koje od njih potiče na pisanje modularnog koda koji se zatim koristi tako da se kombiniraju moduli unutar jednog elementa kojima se potom stvaraju cijele stranice, odnosno sustavi. Moduli su zasebni dijelovi koji su odvojivi od ukupnog koda i ne ovise o okruženju u kojem se nalaze. Pravilno modulariziranim kodom stvara se sustav koji u cijelosti ne ovisi o pojedinom modulu te, ako je potrebno, moguće ga je izdvojiti bez potrebe za popravljanjem strukture preostale stranice.

[24] Specifičnost (engl. *specificity*) se odnosi na specifičnost kaskade CSS selektora. Primjerice selektor div.span ima veću specifičnost od samo span i jedino će se njegovi atributi uzeti u obzir za prikaz tog elementa.

6.1. OOCSS

Prva metodologija koja je bila naširoko prihvaćena je OOCSS (engl. *Object Oriented CSS*) koja je predstavljena 2009. godine. Glavna karakteristika OOCSS-a odvajanje je strukture od stila – nešto što se već radi podjelom na HTML i CSS. Razlika je u tome što se i u CSS-u definira stil koji definira strukturu stranice, ponašanje njenih dijelova na različitim rezolucijama i slično te stil koji stvara samo sučelje, njegove boje, interaktivnost i cjelokupni izgled. Dvije strane koje u kodu dijele mjesto zapravo služe dosta različitim svrhama, pa je iz tog razloga logično na neki način odvojiti njihovu deklaraciju i pisanje. Odvajanje stila i strukture elemenata bi na primjeru izgledale ovako[25];
Struktura tipke i njena visina i širina definirani su klasom *button*:

```
.button {
    box-sizing: border-box;
    height: 100px;
    width: 100%;
}
```

Stil tipke je potom definiran dodatnom klasom, npr. *primary-button*:

```
.primary-button {
    background: $primary-color;
    border: 1px solid gray;
    color: white;
}
```

U HTML kodu se zatim tipka generira koristeći te dvije klase

```
<button class="button primary-button">
```

Time se izbjegava nepotrebno dupliciranje CSS koda što bi bilo potrebno kada bi se trebala generirati tipka druge boje, odnosno *secondary button* unutar kojeg bi se ponovno morali pisati atributi već definirani u klasi *button*. Na ovaj način se samo HTML elementu *button* pridruži klasa *secondary-button* koja sadrži samo promjene za taj stil tipke, a zadržava dijeljeni stil svih *button* elemenata definiranih u klasi *.button*.

[25] Primjer preuzet i prilagođen s http://sixrevisions.com/css/css-methodologies/

Kako bi se koncepti OOCSS-a uspješno primijenili na velike sustave s mnogo modula i elemenata, kritično je prepoznati ponavljajuće stilove i apstrahirati ih u izdvojene klase kako bi se izbjeglo dupliciranje CSS koda. Ipak, pri tome se mora obratiti pažnja na rastući broj klasa kako se kompleksnost građenog modula povećava. Koristeći OOCSS nije neuobičajeno vidjeti element s pet i više klasa od kojih svaka dodjeljuje dio stila ili strukture što se rijetko susreće u standardnom CSS-u.

OOCSS ne donosi samo jednostavnost u pisanju i održavanju koda, nego i ubrzanje korištenja internetskih stranica. Koristeći iste klase više puta unutar stranice umjesto ponovnog pisanja koda, OOCSS smanjuje efektivni broj linija koda koje internetski pretraživač mora preuzeti na korisničko računalo i time cijeli sustav ima kraće vrijeme učitavanja. Osim očite prednosti u brzini učitavanja, modularni pristup pisanju CSS koda omogućuje daljnji razvoj sustava uz pisanje vrlo malo novog koda - već se veliki dio potrebnih funkcionalnosti može generirati od postojećih modula koji sadržavaju sve potrebne dijelove. Često se takvi elementi smatraju *besplatnima* jer je moguće kreirati cijeli novi modul kombiniranjem već postojećih klasa čime se bez povećavanja veličine CSS dokumenta stilizira novi modul.

6.2. SMACSS, BEM, Atomic CSS

Kao i kod alata za procesiranje, nakon što je prvi alat, odnosno prva metodologija pisanja CSS koda, poprimila na popularnosti, pojavile su se alternative i drugačiji pristupi. Jedan od alternativnih principa imenovanja klasa naziva se SMACSS (engl. *Scalable and Modular Architecture for CSS*). Prema [23] SMACSS kategorizira klase u 5 kategorija; osnovne, raspored, moduli, stanja i teme.

Osnovne klase su najčešće HTML elementi poput *body*, *html*, *form*, *section*, itd. Klase rasporeda, najčešće s prefiksom s *l-* ili *layout-* označavaju klase kojima se stvara struktura i raspored elemenata na stranici. Moduli su komponente koje se mogu koristiti više puta i nalaze se unutar rasporednih klasa. Klase stanja definiraju kako će moduli izgledati u različitim uvjetima – aktivni, neaktivni, skriveni, mobilne ili desktop verzije modula. Na posljetku, tematska kategorija CSS-a objašnjava alternativnu verziju modula ili rasporeda.
U primjeru HTML koda, to bi izgledalo kao u nastavku[26]:

[26] Primjer koda preuzet s http://sixrevisions.com/css/css-methodologies/

```
<section class="l-footer">
  <form class="search is-submitted">
    <input type="search" />
    <input type="button" value="Search">
  </form>
</section>
```

SMACSS je nešto fleksibilniji i podrazumijeva manje pravila od BEM-a (engl. *Block-Element-Modifier*). BEM je način pisanja CSS koda koji su razvili programeri *Yandex-a*, ruskog internetskog pretraživača. Danas možda i najkorištenija metodologija pisanja CSS koda, BEM je u svojem nastojanju za stvaranjem više strukturiranog HTML koda za posljedicu stvorio semantički kvalitetnije stranice i jasno odvojene dijelove koji se lakše ponovno koriste. Prema [24] U samom imenu krije se način rada – Blok, element i modifikator. Blok je osnovni element, odnosno modul koji se stilizira, primjerice *input* polje za unos. Blok može sadržavati više elemenata koji definiraju njegovu strukturu i izgled, primjerice naslov input polja koji je imenovan s *input__title*, a modifikator je alternativno stanje osnovnog element bloka i označava se s --, primjerice *input--disabled* bi označavao polje za unos koje je deaktivirano.

U primjeru HTML tako BEM izgleda:

```
<section class="main">
    <h1 class="main__title>
    Naslov odjeljka
</h1>
<div class="main__content">
    Sadržaj odjeljka
</div>
<div class="main__content main__content--disabled">
    Neaktivni sadržaj odjeljka
</div>
</section>
```

BEM metodologija je specifična prema tome što daje jasne upute za imenovanje klasa ovisno o njihovoj HTML strukturi pa je tako moguće iz CSS datoteke za prethodni primjer saznati da se *main_content* nalazi unutar odjeljka *main* te da je *main__content—disabled* jednako stilizirani odjeljak unutar većeg odjeljka *main* te da je deaktiviran. Takva razina ovisnosti strukture CSS i HTML dokumenata nije moguća bez uvođenja strogih pravila imenovanja modula. Isti dio koda napisan u standardnom CSS-u izgledao bi ovako:

```
<section class="main">
    <h1 class="title>
    Naslov odjeljka
</h1>
<div class="content">
    Sadržaj odjeljka
</div>
<div class="content disabled">
    Neaktivni sadržaj odjeljka
</div>
</section>
```

Što samo po sebi ne izgleda pogrešno, ali do problema dolazi u velikim sustavima zbog globalne dostupnosti atributa poput *title* ili *disabled*, koji će onda kaskadom aplicirati jednaka svojstva u cijelom sustavu. Osim toga, iz CSS datoteke nije moguće zaključiti gdje se u strukturi nalazi *title* odnosno *main*, te nije moguće saznati njihovu korelaciju. Kao i kod svih metodologija, ovakve napredne funkcionalnosti ovise o kvaliteti implementacije metodologije jer njena polovična i neispravna implementacija dovodi od više problema nego što ih rješava.

6.3. Atomic CSS

Atomic CSS predstavlja nešto veći odmak od ostalih CSS metodologija tako što primjenjuje koncept više klasa s manje atributa koji se pokazao efikasnim u SMACSS-u, ali se koristi do krajnosti. Prema [25] Atomic CSS pristupa pisanju stilova u CSS-u tako da jednoj klasi daje jednu zadaću, odnosno da svaki atribut koji se može aplicirati jednom elementu ima sam svoju klasu. Za razliku od ostalih alata, Atomic CSS se može nazvati CSS arhitekturom zbog toga što dolazi s unaprijed postavljenim klasama koje se potom apliciraju na željene HTML elemente.

Kako bi bilo moguće koristiti Atomic CSS, potrebno je poznavati njegove mogućnosti, odnosno unaprijed pripremljene klase i kako se koriste. U dokumentaciji alata[27] nalazi se popis svih klasa koje se mogu koristiti pri stvaranju elemenata. Uzmimo za primjer kvadrat određene veličine s nekom pozadinom i rubom; koristeći se dokumentacijom dolazi se do sljedećih klasa:

```
<div class=”W(100) H(100) Bgc(#f0f0f0.5) Bd(1px, solid, #333)>
    Sadržaj unutar kvadrata
```

[27] AtomicCSS dokumentacija nalazi se na https://acss.io/reference

```
</div>
```

Koristeći funkcije, Atomic CSS iz gornjih klasa generira element visine `100px`, širine `100px`, pozadinske boje `#F0F0F0` s faktorom prozirnosti 0.5 te punim rubom od `1px` boje `#333`.
Na isti način se primjenjuju sva ostala svojstva – potrebno je pronaći u dokumentaciji ime klase koja sadrži željeni atribut, primjerice `background-color` – `Bgc`, te proslijediti parametar vrijednosti boje – `#F0F0F0`.

Prema [26] prednost ovog pristupa je da se kod nigdje ne koristi dva puta, tako što uvijek postoji samo jedna klasa s jednim specifičnim atributom. Nedostatak je pak taj što HTML elementi poprimaju izuzetno veliki broj klasa koje u nekim slučajevima mogu otežati čitljivost samog HTML dokumenta.

7. BUDUĆNOST CSS-A I ALATA ZA PROCESIRANJE CSS-A

U zadnjih nekoliko godina došlo je do ubrzanog razvoja alata za rad s klijentskim tehnologijama – HTML, CSS i *Javascript*. Uz nabrojane alate i metodologije vezane za CSS, gotovo se na mjesečnoj bazi pojavljuje novi *Javascript framework*[28] (poput *AngularJs, ReactJS, VueJS* itd.) koji obećava promijeniti i olakšati rad s tim programskim jezikom. U sličnoj se fazi razvoja trenutačno nalaze alati za procesiranje CSS-a, gdje ih ponekad ima i nekoliko koji služe istoj svrsi. Takav razvoj događaja je normalan u ranim fazama razvoja novih tehnologija i alata pa se kroz sličan period prošlo i s razvojem programskih jezika kojih je bilo mnoštvo u 90-im godinama prošloga stoljeća te početkom 21. stoljeća, dok se razvoj novih programskih jezika usporio u posljednjem desetljeću. Kako se u sferi alata za procesiranje CSS-a radi o izuzetno manje kompleksnim jezicima i rješenjima od programskih jezika poput *Javascript-a* ili PHP-a, za očekivati je da će se i njihov razvoj puno brže dovršiti. Iako nisu do sada spomenuti, postoji još alata za procesiranje CSS-a, kao što su *Rework, Myth, CSS-Crush* i drugi, koji su većinom kreirani na dobrim osnovama, ali nisu šire prihvaćeni. PostCSS, kao predstavnik novijeg vala alata, daje naslutiti smjer u kojem bi se industrija mogla kretati – modularni, skalabilni i visoko prilagodljivi alati koji mogu odgovarati malim i agilnim timovima, ali i velikim, korporativnim okruženjima koji imaju sasvim drugačije zahtjeve.

Osim toga, internetski pretraživači, odnosno njihovi proizvođači – poglavito Google i Microsoft, internetski pretraživač sve više smatraju operativnim sustavom unutar operativnog sustava do te mjere da je Google to i službeno pokušao sa svojim operativnim sustavom *ChromeOS* koji je u potpunosti baziran na njihovom pretraživaču *Google Chrome*-u. Daljnjim razvojem u tom smjeru, možemo očekivati veću kontrolu samih proizvođača nad verzijama svojih proizvoda te posljedično bržu implementaciju W3C preporuka. Time bi se na posljetku omogućilo ranije prihvaćanje novih standarda i smanjila potrebu za izdvojenim alatima za procesiranje CSS. Prednosti koje donose spomenute metodologije pisanja CSS koda lako su primjenjive na bilo koje okruženje i smjer u kojem se CSS bude kretao u sljedećih nekoliko godina, dok su alati poprilično zaključani unutar svog ekosustava – bio to *Ruby, NodeJS* ili nešto treće. Njihova specifičnost sama po sebi ne znači da su spomenuti alati loši ili nefleksibilni, nego upravo suprotno – njihovi tvorci sami su odgovorni za razvoj i kontroliraju okruženje u kojem se izvode, upravo ono zbog čega su tako brzo i prihvaćeni. Usprkos tome, svi alati su tu zbog nedostataka u samom jeziku i predstavljaju dodatni

[28] *Framework* (hrv. razvojni okvir) je razvojno okruženje koje sadrži najčešće korištene funkcionalnosti kako bi se ubrzao razvoj u programskom jeziku za koji je razvojni okvir pisan.

sloj između razvojnih programera i krajnjeg koda koji se generira na, ponekad ne potpuno transparentan način. Idealni dugoročni razvoj CSS-a uzrokovao bi potpuno napuštanje alata i svih međukoraka te jednu, unificiranu i kompletnu implementaciju CSS-a u svim internetskim pretraživačima – nešto na što se još neko vrijeme mora pričekati.

8. ZAKLJUČAK

CSS, odnosno *Cascading Style Sheets*, stilski je jezik koji je prvenstveno bio zamišljen za osnovno stiliziranje tekstualnih, a ubrzo zatim i multimedijalnih, HTML dokumenata. Nastao je sredinom 90-tih godina prethodnog stoljeća i kao jezik se nije puno mijenjao. Suprotno njemu, internet i njemu popratne tehnologije napredovale su više no što je itko mogao očekivati u vrijeme kada je jezik bio napisan, pa je tako sredinom prethodnog desetljeća, svega 10tak godina nakon njegovog uvođenja, CSS počeo pokazivati teškoće prilikom stiliziranja modernih sučelja. Daljnjim razvojem kompleksnosti sučelja koja se koriste u modernim internetskim aplikacijama te jednostavno porastom veličine samih aplikacija koje su stilizirane koristeći CSS, postajalo je sve teže kvalitetno skalirati stilski kod. Problemi globalne dostupnosti CSS pravila te posljedično neželjenog nasljeđivanja koda uzrokovali su nepotrebno povećanje kompleksnosti i veličine koda koji je u svojim počecima trebao biti vrlo jednostavan. Alati za procesiranje CSS-a su od svog pojavljivanja unijeli revoluciju u način pisanja te vrijednost CSS koda u projektu i time pokazali da jezik sam po sebi nije manjkav već da mu nedostaje malo fleksibilnosti kako bi bilo moguće napraviti sve što se od njega može zahtjevati u današnjem, i budućem, okruženju. Automatiziranjem često pisanih naredbi, ograničavanjem globalne dostupnosti i uvođenjem varijabli i sličnih proširenja, stilski jezik CSS približio se punokrvnim programskim jezicima zadržavajući pritom svoju deklarativnu prirodu i jednostavnost čitanja. Osim toga, alati za procesiranje, odnosno metodologije pisanja CSS-a nabrojane u ovom radu uvelike su olakšali rad razvojnim programerima koji pišu i održavaju kod zbog poboljšanja koje donose u vidu skalabilnosti i proširivanja funkcionalnosti koda. Zbog inherentnih problema jezika CSS-a i načina njegove implementacije u internetske pretraživače, može se očekivati da će procesiranje CSS-a, odnosno svih jezika koji su prošireni setovi standardnog jezika, još neko vrijeme biti relevantno prilikom izgradnje i održavanja sučelja na internetu kakvog danas poznajemo. Proširivanje standardnog seta funkcionalnosti jezika i njegovo prilagođavanje potrebama industrije, omogućilo je kraće vrijeme razvoja uz manji broj grešaka te veći fokus na izgradnju kvalitetnih i pristupačnih sučelja. Posebno u posljednjih nekoliko godina, kada se sve veći fokus stavlja na korisničko iskustvo prilikom korištenja digitalnih sučelja, postalo je bitno pisati kvalitetan CSS kod, a upravo to omogućuju spomenuti alati. Neovisno koji se alat ili metodologija naposljetku koristili, zajedno su donijeli revoluciju u način pisanja CSS-a te se sa sigurnošću može reći da je CSS ponovno adekvatno rješenje koje može pratiti potrebe današnjeg interneta.

LITERATURA

[1] Mozilla Developer Network, »CSS,« Mozilla Developer Network, [Mrežno]. Available: https://developer.mozilla.org/en-US/docs/Web/CSS. [Pokušaj pristupa 19 2 2017].

[2] World Wide Web Consortium, »Descriptions of all CSS specifications,« World Wide Web Consortium, [Mrežno]. Available: https://developer.mozilla.org/en-US/docs/Web/CSS. [Pokušaj pristupa 2 18 2017].

[3] B. Cinarli, »An Introduction to CSS Pre-Processors: Sass, LESS and Stylus,« 2014. [Mrežno]. Available: https://htmlmag.com/article/an-introduction-to-css-preprocessors-sass-less-stylus. [Pokušaj pristupa 18 02 2017].

[4] European Organization for Nuclear Research, »The birth of the web,« [Mrežno]. Available: https://home.cern/topics/birth-web. [Pokušaj pristupa 19 2 2017].

[5] B. Hidoine, »The World Wide Web Consortium announces Web Style Sheets,« 4 1996. [Mrežno]. Available: http://www.ercim.eu/publication/Ercim_News/enw25/hidoine.html. [Pokušaj pristupa 20 2 2017].

[6] World Wide Web Consortium, »A brief history of CSS until 2016.,« [Mrežno]. Available: www.w3.org/Style/CSS20/history.html. [Pokušaj pristupa 19 2 2017].

[7] World Wide Web Consortium, »About W3c,« 19 2 2017. [Mrežno]. Available: www.w3.org/Consortium/.

[8] World Wide Web Consortium, »CSS Custom Properties for Cascading Variables Module Level 1,« 20 2 2017. [Mrežno]. Available: www.w3.org/TR/css-variables-1/.

[9] Mozilla Developer Network, »Learn Web Development,« 20 2 2017. [Mrežno]. Available: https://developer.mozilla.org/en-US/docs/Learn/CSS/Introduction_to_CSS/Selectors.

[10] W3Schools, »CSS Syntax and Selectors,« 20 2 2017. [Mrežno]. Available: http://w3schools.com/css/css_syntax.asp.

[11] Stylus, »CSS Style Syntax,« 18 3 2017. [Mrežno]. Available: http://stylus-lang.com/docs/css-style.html.

[12] Mozilla Developer Network, »Cascade and Inheritance,« 20 2 2017. [Mrežno]. Available: developer.mozilla.org/en-US/docs/Web/CSS/Cascade.

[13] Drupal, »Usage statistic for SASS/SCSS,« [Mrežno]. Available: https://www.drupal.org/project/usage/less. [Pokušaj pristupa 18 3 2017].

[14] Codepen, »Popularity of Preprocessors on CodePen,« [Mrežno]. Available: https://blog.codepen.io/2016/02/24/popularity-of-preprocessors-on-codepen/. [Pokušaj pristupa 18 3 2017].

[15] Computer Hope, »Nest,« [Mrežno]. Available: http://www.computerhope.com/jargon/n/nesting.htm. [Pokušaj pristupa 18 3 2017].

[16] I. Gerchev, »A Comprehensive Introduction to LESS: Mixins,« [Mrežno]. Available: https://www.sitepoint.com/a-comprehensive-introduction-to-less-mixins/. [Pokušaj pristupa 18 3 2017].

[17] B. Simpson, »LESS/SASS: The Advantages of CSS Preprocessing Explained,« [Mrežno]. Available: http://blog.blakesimpson.co.uk/read/37-less-sass-the-advantages-of-css-preprocessing-explained. [Pokušaj pristupa 18 3 2017].

[18] Sass-lang, »Sass Changelog,« [Mrežno]. Available: https://sass-lang.com/documentation/file.SASS_CHANGELOG.html. [Pokušaj pristupa 18 3 2017].

[19] Lesscss.org, »About,« [Mrežno]. Available: lesscss.org/about/. [Pokušaj pristupa 19 2 2017].

[20] PostCSS, »GitHub Documentation,« [Mrežno]. Available: https://github.com/postcss/postcss#usage. [Pokušaj pristupa 18 3 2017].

[21] Evil Martians, »About PostCSS,« [Mrežno]. Available: https://ai.github.io/about-postcss/en/. [Pokušaj pristupa 18 3 2017].

[22] K. Potts, »A Look at Some CSS Methodologies,« [Mrežno]. Available: sixrevisions.com/css/css-methodologies/. [Pokušaj pristupa 20 2 2017].

[23] J. Snook, »Categorizing CSS Rules,« [Mrežno]. Available: https://smacss.com/book/categorizing. [Pokušaj pristupa 18 3 2017].

[24] H. Roberts, »MindBEMding – getting your head 'round BEM syntax,« [Mrežno]. Available: https://csswizardry.com/2013/01/mindbemding-getting-your-head-round-bem-syntax/. [Pokušaj pristupa 20 2 2017].

[25] Atomic CSS, »Thinking in Atomic,« [Mrežno]. Available: https://acss.io/thinking-in-atomic.html. [Pokušaj pristupa 20 2 2017].

[26] R. Bray, »Is 2016 the year of Atomic CS?,« [Mrežno]. Available: http://www.creativebloq.com/css3/atomic-css-11619006. [Pokušaj pristupa 20 2 2017].

SAŽETAK

Rad CSS procesori i metodologije za cilj ima približiti svrhu postojanja alata za procesiranje CSS-a početniku u svijetu razvoja internetskih aplikacija. Opisana su četiri glavna predstavnika alata za procesiranje CSS-a – SASS, LESS, *Stylus* te PostCSS. Uz to, dan je kratki pregled povijesti razvoja *Cascading Style Sheets*-a i njegovih ograničenja koja su posljedično dovela do razvoja alata za procesiranje CSS-a. Prikazani su osnovni primjeri koristeći navedene alate te su izdvojene razlike i prednosti, odnosno nedostatci među njima. Osim toga, opisane su metodologije pisanja CSS koda koje doprinose čitljivosti i pravilnoj strukturi koda. Uz sve nabrojano, izvršena su mjerenja brzine obrađivanja koda koristeći svaki od obrađenih alata te su izdvojene njihove prednosti i nedostatci. Ovaj rad pruža generalni pregled glavnih dostupnih alata koji omogućavaju lakši rad s CSS-om, ali za detaljniju se analizu preporuča iscrpna dokumentacija svakog pojedinačnog alata od navedenih. Na posljetku, poglavljem *Budućnost CSS-a i alata za procesiranje CSS-a* pokušava se predvidjeti daljnji razvoj alata odnosno hoće li u budućnosti uopće biti i potrebni.

KLJUČNE RIJEČI

Cascading Style Sheets, internet preglednici, CSS, HTML, SASS, LESS, PostCSS, specifičnost, kaskada, alati za procesiranje CSS-a, klasa, CSS metodologije

CSS preprocessors and methodologies

ABSTRACT

Thesis “CSS processors and methodologies” aims to familiarize beginners in a world of web development with CSS processors and their purpose. Four of the main representatives have been described, namely SASS, LESS, *Stylus*, and PostCSS. Also, CSS’s history has been summarized together with its shortcomings which ultimately led to development of CSS processors. Simple code examples written in each of the aforementioned tools are provided which highlight differences, advantages and disadvantages among them. Main methodologies for writing CSS code have been described, all of which help with readability and proper code structure. Furthermore, using each of the covered tools, speed test for compiling code have been created which showcase their strong and weak points. This thesis’ main goal is a high-level overview of all main CSS processors which can help with writing CSS, but for a deeper insight in the each tool a look in its detailed documentation is advised. Finally, the chapter titled *Future of CSS and CSS processing tools* tries to paint a picture of a further development of the CSS processing tools and will their assistance even be necessary in the near future.

KEYWORDS

Cascading Style Sheets, internet browsers, CSS, HTML, SASS, LESS, PostCSS, specificity, cascade, CSS processing tools, class, CSS methodologies

ŽIVOTOPIS

Sebastijan Dumančić rođen je 23.01.1993. godine u Osijeku. Pohađao je Osnovnu školu Frana Krste Frankopana te I. Gimnaziju, također u Osijeku. Preddiplomski studij računarstva započinje 2011. godine na Elektrotehničkom Fakultetu u Osijeku te ga uspješno završava 2014. godine. Tijekom preddiplomskog studija sudjeluje u radu udruge IEEE te volontira kao voditelj studentske radijske emisije *Geeks on a Plane*, jedne od prvih emisija na studentskom radiju UNIOS. U međuvremenu upisuje i uspješno završava tečaj Web programiranja u školi informatike i menadžmenta *Edunova*, nakon čega se intenzivnije nastavlja fokusirati na razvoj internetskih stranica i sustava sve do danas. Diplomski studij procesnog računarstva upisuje 2014. godine na, tada, Elektrotehničkom fakultetu, a danas Fakultetu elektrotehnike, računarstva i informacijskih tehnologija Osijek. Tijekom diplomskog studija provodi jedan semestar kao gostujući student u sklopu CEEPUS programa mobilnosti na Tehničkom sveučilištu u Grazu (Technische Universität Graz), Austrija. Po povratku nastavlja volontirati u udruzi *Erasmus Student Network* te u suradnji s Uredom za međunarodnu suradnju Sveučilišta Josipa Jurja Strossmayera i Agencijom za mobilnosti i programe EU radi na poboljšanju programa mobilnosti u gradu i državi. Kao dugogodišnji volonter i zagovaratelj internacionalizacije visokog obrazovanja vjeruje u koncept cjeloživotnog učenja.

PRILOZI

Prilog 1 – detaljni rezultati mjerenja brzine

Mali set (vrijeme u ms)					
Alat	PostCSS	Ruby SASS	libSASS	LESS	Stylus
1.	130	246	51	84	199
2.	129	245	32	93	205
3.	122	247	32	88	197
4.	120	244	33	85	196
5.	132	259	32	99	200
6.	138	267	33	90	203
7.	147	249	34	88	201
8.	122	267	36	86	215
9.	131	256	36	84	216
10.	126	413	35	93	197
Prosječno	129,7	269,3	35,4	89	202,9

Prilog 1.1. – Detaljni rezultati mjerenja brzine na malom setu pravila

Jednostavni veliki set (vrijeme u ms)					
Alat	PostCSS	Ruby SASS	libSASS	LESS	Stylus
1.	192	318	79	147	484
2.	170	320	79	139	372
3.	160	319	89	156	400
4.	172	327	74	260	377
5.	183	329	76	141	384
6.	176	327	76	158	387
7.	156	335	76	141	375
8.	157	326	77	163	414
9.	165	330	73	143	390
10.	157	341	79	137	472
Prosječno	168,8	327,2	77,8	158,5	405,5

Prilog 1.2. – Detaljni rezultati mjerenja brzine na velikom, jednostavnom setu pravila

Kompleksni veliki set (vrijeme u ms)					
Alat	PostCSS	Ruby SASS	libSASS	LESS	Stylus
1.	227	300	82	235	419
2.	221	308	72	215	410
3.	227	306	90	264	411
4.	224	317	74	357	439
5.	230	317	72	230	410
6.	267	322	77	216	438
7.	249	324	75	290	425
8.	234	317	73	264	404
9.	221	331	73	218	410
10.	234	318	83	246	399
Prosječno	233,4	316	77,1	253,5	416,5

Prilog 1.3. – Detaljni rezultati mjerenja brzine na velikom, kompleksnom setu pravila