Moderni Web development koncepti u e-commercu na primjeru Sylius platforme

Omrčen, Luka

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU

FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

Sveučilišni studij

MODERNI WEB DEVELOPMENT KONCEPTI U E-COMMERCU NA PRIMJERU SYLIUS PLATFORME

Diplomski rad

Luka Omrčen

Osijek, 2018

SADRŽAJ

1. UVOD

Većina modernih web stranica današnjice ima složeni dizajn, privlačan i interaktivan sadržaj, koji uključuje mnoge značajke i usluge, uklapajući se pritom na svim trenutnim zaslonima i web preglednicima. Razvijanje takvih web stranica i aplikacija zahtjeva dublje poznavanje web elemenata i implementaciju složenih koncepata koji se pojavljuju na svakodnevnoj bazi. U današnje vrijeme razvojni programer mora biti dovoljno kompetentan i posjedovati široki spektar tehničkih znanja kako bi mogao razumjeti i implementirati nove tehnike unutar tehnološkog procesa izrade web aplikacije/stranice. Razvoj suvremenog web-a počeo je prije deset godina kada su se pojavili „pametni telefoni“ koji su svojom pojavom doveli do ubrzanog rasta komercijalnih i nekomercijalnih aplikacija i usluga koje su postale dio svakodnevice. Jedna od takvih usluga koja je doživjela ubrzani rast i nije dosegla svoj vrhunac je i e-trgovina (eng. E-commerce). E-trgovina predstavlja pojam za elektroničku trgovinu koja omogućuje kupcu naručivanje proizvoda preko interneta. U današnje vrijeme trgovina putem interneta smatra se jednom od najprofitabilnijih i najefektivnijih oblika trgovine zbog jednostavnosti izvršenja prodajnog procesa kao i niskih troškova poslovanja. Među najpoznatijim i najdominantnijim platformama e-trgovine nalazi se „Magento“, ali u današnje vrijeme pojavljuju se i nove konkurentne platforme kao što je „Sylius“.

1.1. Zadatak diplomskog rada

Cilj ovog diplomskog rada je opisati proces planiranja i definiranja sadržaja i dizajna web stranice ili aplikacije kako bi se prilagodio svim uređajima i preglednicima današnjice. Definiraju se pojmovi kao što su „mobile-first“ i „desktop-first“ te se objašnjava razlika između njih. Opisuje se razlika između responzivnog i adaptivnog web dizajna, te općenita razlika između web dizajna i UI/UX dizajna. Uvode se koncepti i nove tehnologije u izradi kao što su predprocesori SASS, LESS i trenutno novi PostCSS, te ES6+ javascript i CSS4 koji predstavljaju tehnologije weba u budućnosti. Definira se pojam e-trgovina na primjeru Sylius platforme. Potrebno je opisati Sylius-ovu arhitekturu te način rada razvojnog programera (frontend) na navedenoj platformi. Definiraju se sve točke procesa tehnološke izrade počevši od kreiranja vlastite teme od navedenog dizajna pa sve do realiziranog konkretnog rješenja.

2. MODERNI WEB DIZAJN I DEVELOPMENT KONCEPTI

Zahtjevi i dizajn trenutno modernih web stranica i aplikacija razlikuju se od onoga što su bili prije svega nekoliko godina. Iako je njihova svrha i funkcionalnost ostala ista, znatno je promijenjen pristup informiranja i pozornost krajnjem korisniku. Moderne web stranice i aplikacije na prvi pogled odlikuju kreativnošću i raznolikošću, kako u dizajnerskom tako i u tehnološkom smislu. Moderni web alati, metode i tehnike razvoja weba razvili su se i evoluirali u onom smjeru kako bi ispunjavali sve tehnološke i dizajnerske zahtjeve današnjice. U navedenom poglavlju opisan je proces planiranja i definiranja sadržaja i dizajna web stranice i aplikacije na svim raspoloživim dimenzijama današnjice. Definiraju se metodologije „mobile-first“ i „desktop-first“ te se objašnjava razlika između njih. Opisuje se razlika između responzivnog i adaptivnog web dizajna, te općenita razlika između web dizajna i UI/UX dizajna.

2.1. Definiranje strategije i prilagodba sadržaja na višestrukim uređajima

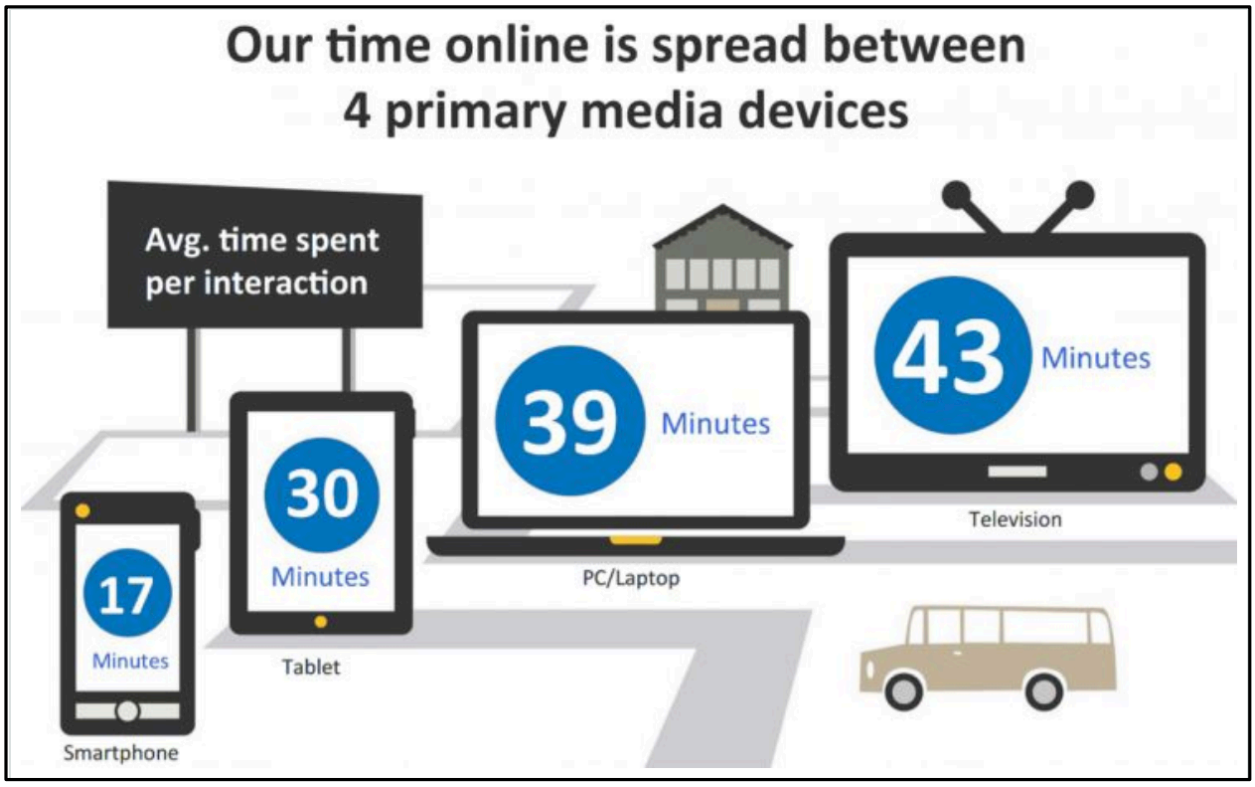
Način na koji obrađujemo sadržaj mijenja se brže nego u bilo kojem trenutku u povijesti. Tehnološki pomak od tiska do digitalnog doba bio je seizmički gledano sa strukturalne perspektive, stvari se nisu prestale kretati sve od tada. Razlog tomu predstavlja rast korištenja mobitela, a sada konkretno i upotreba tableta koji daju novi dizajnerski spektar i dodaju još jedan dodatni sloj složenosti o kojem tvrtke do sada nisu morale razmišljati. Godine 2014 Facebook je predstavio svoj financijski izvještaj prihoda od oglašavanja. Prihodi od 2,5 milijarde dolara za tromjesečje od siječnja do ožujka bili su, po prvi puta, većinski udio zarade od oglašavanja na mobitelu, što daje jasan znak promjena navika kod korisnika. Također Facebook je napravio dobar posao stvaranja velike strategije sadržaja koja je mnogo složenija, a "pukotina" koja se iskazala zahtijeva strukturirani pristup koji počinje razumijevanjem načina kako vodimo interakciju s našim svakodnevnim uređajima.

Upoznavanje sa sadržajem koji se mora adekvatno prilagoditi i smjestiti predstavlja ključan aspekt procesa izgradnje strategije. Prvi korak u definiranju strategije predstavlja stvaranje istinski informirane slike, a to se postiže definiranjem određenih pitanja kao što su:

1. Koji uređaj se koristi i kada tijekom dana.
2. Redoslijed uređaja u klasičnom toku kupovine (e-trgovina).
3. Koji uređaji se upotrebljavaju za pristup ključnim sadržajnim platformama.
4. Vremenski period upotrebe svakog uređaja i što se gleda na njima.

Odgovorima na navedena pitanja olakšava se strukturiranje konkretne i ciljane strategije za definiranje sadržaja. Razumijevanje onoga što korisnici traže pri korištenju različitih uređaja pomaže u pisanju, pakiranju i distribuiranju sadržaja na različitim kanalima prema krajnjem korisniku. Povrh samog razumijevanja, također se osigurava ključan element u bilo kojoj strategiji: varijacija. Različiti pristup stvaranja sadržaja ne samo da će "zabaviti" i informirati ciljanu publiku, nego će i na ispravan način poboljšati razinu korisničkog iskustva na višestrukim uređajima i samim time zadržati postojeću publiku te privući novu.

Način na koji se vrši interakcija sa sadržajem mijenja se tijekom dana. Prema *slici 2.1* iz navedene Google-ove studije 2012 godine, vrijeme provedeno nad mobilnim sadržajem ističe se rano ujutro, oko podneva za vrijeme ručka te na putu prema kući s posla. Večernji sati ističu više korištenje tableta ili istodobno korištenje više zaslona (osobno računalo, televizor itd.) u vlastitim domovima.



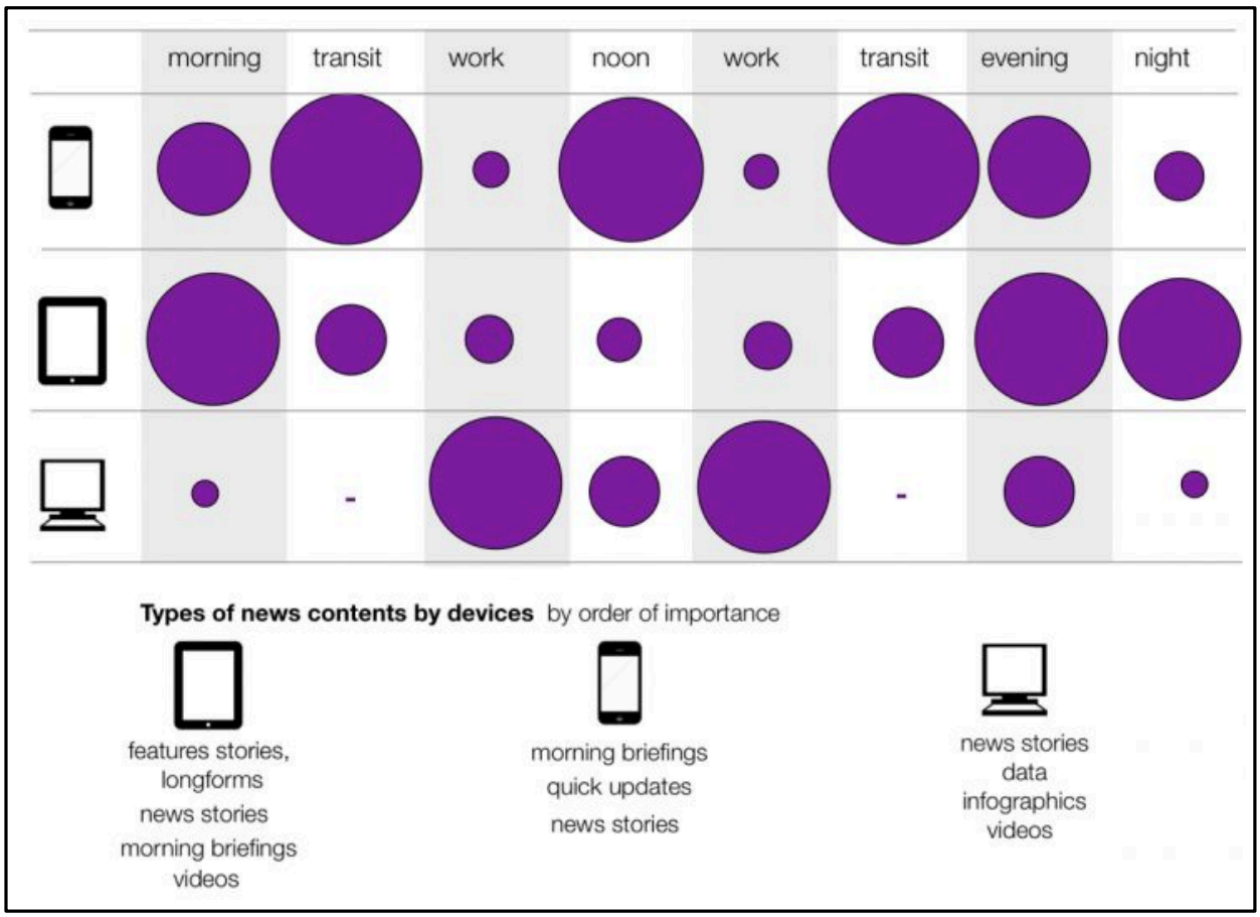
Sl 2.1. Prikaz prosječnog vremena interakcije na određenim uređajima

Sa stajališta e-trgovine Google-ovo istraživanje jasno pokazuje sve veće korištenje mobilnih i tablet uređaja prvi uvjetima konverzije kupovine, što ukazuje potrebu za responzivnim dizajnom i sadržajem koji se lako prilagođava na različitim zaslonima.

Mobilni Internet vratio je kontrolu natrag čitatelju ili kupcu, a način na koji se konzumira sadržaj ili vrše konverzije je pod korisničkim uvjetima. S takvom moći u rukama oni koji

konzumiraju sadržaj ili kupuju, navedeni i ciljani sadržaj koji se prezentira korisniku mora biti dostupan i učinkovit kako bi se privukla njegova pažnja. Kako bi se osigurao određeni fokus i strategija usmjerila u pravom smjeru prema ciljanoj publici, potrebno je izvršiti analizu i određeno segmentiranje nad već postojećim podacima o postojanim korisnicima ili kupcima. Potrebno je definirati ključne pojedinosti onih koje se želi obuhvatiti sa ciljanim sadržajem, koji osiguravaju ključnu ulogu u osiguravanju usluga koje se nude ciljanoj publici prema njihovim potrebama, pa se tako samim time stavljaju u središnju točku procesa.

Nakon što se dobije jasan uvid koja se željena publika cilja, sljedeću točku procesa predstavlja kada i gdje korisnik stupa u interakciju s određenim uređajima i sadržajem u svakodnevnom životu *(sl 2.2.)*. Takvi podatci nam generalno trebaju pri svakoj analizi profiliranja korisničkog ponašanja.



Sl 2.2. Prikaz interakcije sa sadržajem tijekom dana na određenim uređajima

Vrsta sadržaja koja se konzumira na svakom uređaju varira široko te zahtjeva sistematski pristup stvaranju sadržaja kako bi služio svojoj svrsi. Drugi vrlo bitan faktor je korisnikova pozornost te njegovo vrijeme koje je usmjereno na ciljani sadržaj. Kako bi cilj bio ostvaren, potrebno je podijeliti sadržaje u ovisnosti o vrsti uređaja na kojemu se on prikazuje.

Stolna računala i zasloni visokih rezolucija jedni su od najjednostavnijih uređaja pri osiguravanju korisničkog iskustva i prezentiranju ciljanog sadržaja. Razlog tome je taj što se oni koriste najveći vremenski period kroz povijest te se većina korisnika zna služiti njima i svi oblici sadržaja mogu se ostvariti na njima. Na ovakvim vrstama uređaja može se prikazati bogatiji i interaktivni sadržaj koji na najbolji način iskorištava mogućnosti modernog web preglednika i većih zaslona. Primjer tome može biti bogata i interaktivna demonstracija nekog proizvoda popraćena s raznim mogućnostima interakcije korisnika sa sadržajem koji ga zanima na određenoj stranici.

Za razliku od stolnih računala, mobilni uređaji i tableti zadržavaju svoj fokus usmjeren na brzim i konkretnim pregledima sadržaja. Samim time potrebno je olakšati korisniku da se jednostavno vodi između različitih sadržaja unutar stranice na intuitivan i jednostavan način, u cilju poboljšanja vremena provedenog na stranici i konzumiranju sadržaja. Veličina navedenog sadržaja mora biti u mobilnim normama prijenosa podataka, tematika mora biti lako probavljiva te se temeljiti na različitim značajkama, popraćenim s konkretnim slikama i ostalim interaktivnim medijima. Osim navedenih pravila, ključno je osigurati kvalitetno reagiranje i navigaciju koja podržava protokole na dodirnim zaslonima, kao što je npr. "povlačenje dodirom" preko zaslona za novi članak ili funkcionalnost za jednostavno dijeljenje sadržaja preko društvenih mreža.

Kako bi se proces planiranja i definiranja sadržaja proveo u potpunosti, potrebno je dodati određeni nivo strukture planiranja u kojem postoji mogućnost provjere jesu li sve definirane stavke ispunjene ili ne. Taj proces započinje, kao i većina strateških planova, s pitanjima na koja se prilikom izrade plana mora odgovoriti. Neka od navedenih pitanja dana u sljedećem primjeru:

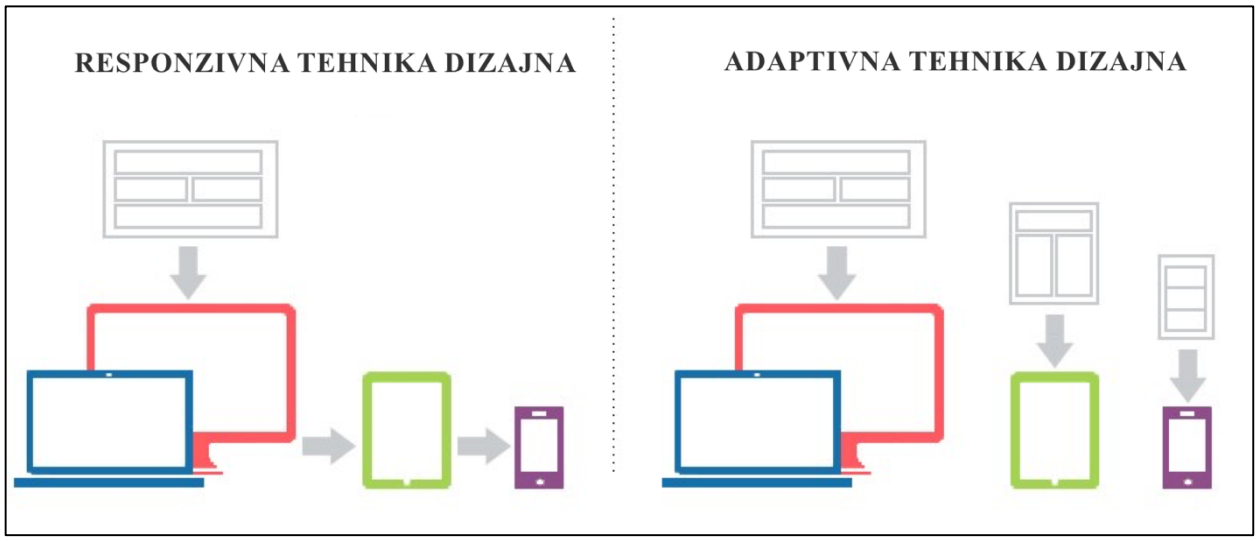
1. Postoji li sadržaj koji odgovara svakoj od osoba koje se nalaze u trenutnoj bazi korisnika?
2. Je li svaka ideja za svaki tip sadržaja relevantna za ciljanu publiku ili robnu marku (e-knjige, članci..)?
3. Postoji li ideja koja će istovremeno odgovoriti potrebama mobilnih, tablet i stolnih uređaja?
4. Je li sadržaj dizajniran za sve vrste konzumiranja (audio, vizualno..)?

Ključ strukturnog planiranja osigurava pravu kombinaciju sadržaja za strukturirani pristup stvaranja određenih ideja.

2.2. Responzivna i adaptivna tehnika dizajna

Responzivne web stranice i adaptivne stranice strukturalno su slične u tome što oboje mijenjanju izgled na temelju preglednika i uvjetima na kojima se gledaju (najčešće to predstavlja širinu preglednika).

Responzivne stranice reagiraju na veličinu preglednika u bilo kojem trenutku. Bez obzira na definiranu širinu preglednika, stranica prilagođava svoj izgled pregledniku (nekada i funkcionalnosti) *(sl 2.3)*. Specifičnost koja ističe ovu tehniku je ta da se pisani izvorni kod ne mora mijenjati specifično niti za jednu veličinu zaslona ili uređaja, on je ujedno pregledniji i prilagodljiviji u svim situacijama.



Sl 2.3. Responzivna i adaptivna tehnika dizajna

Dizajniranje responzivne stranice predstavlja izazov jer se mora obuhvatiti što više veličina zaslona. Neke od prednosti takvog dizajna su:

1. Ugodan i fleksibilan izgled na svim veličinama zaslona.
2. Jedinstveno vizualno i operativno iskustvo s vrlo niskim troškovima za održavanje.
3. SEO jedinstven jer ne postoje različite verzije stranica.
4. Veza između mobilne i stolne verzije stranice može se koristiti bez preusmjeravanja.

Adaptivna tehnika prilagođava se specifičnoj okolini preglednika i može i ne mora uzeti u obzir trenutnu širinu preglednika. Sastoji se od višestrukih prikaza istog dizajna, općenito jedan za svaku određenu veličinu, koje definira klijent ili programer. Svakoj verziji dizajna (mobilno, tablet, stolno računalo..) dodijeljena je određena širina, nazivajući je prijelomna točka (*sl 2.3*).

Umjesto korištenja postotaka kao kod responzivne tehnike, adaptivni dizajn koristi točno jedan statički raspored po svakoj definiranoj prijelomnoj točki. Neke od prednosti adaptivnog dizajna su:

1. Kompatibilan čak i s vrlo složenim web stranicama
2. Može se implementirati po nižoj cijeni od responzivnog dizajna
3. Kodiranje je vremenski brže u odnosu na responzivni dizajn
4. Testiranje je puno lakše te projekt može biti relativno točniji jer je podijeljen u različite cjeline.

2.3. „Mobile-first" i „Desktop-first" metodologija

Mobilni uređaji imaju najviše ograničenja, prvenstveno veličinu i širinu zaslona koje obuhvaćaju. Projektiranje unutar tih parametara prisiljava dizajnere da pažljivo odrede prioritet sadržaja koji će se prikazivati.

„Mobile first" metodologija pristupa temelji se na tome da mobilna verzija web stranice treba biti u središtu strategije razvoja i dizajna, uzimajući u obzir ograničenja i ponašanje raznih mobilnih preglednika koji korisnici koriste. Ponekad mobilni korisnici zahtijevaju različiti sadržaj od korisnika stolnih računala ili drugih uređaja. Specifičnost sadržaja za određeni uređaj može se mjeriti s obzirom na kontekst, što u određenim situacijama poboljšava korisničko iskustvo. Prvi korak u procesu razvoja „Mobile-first" metodologije predstavlja stvaranje korisničkih scenarija, odnosno definiranje bitnih sadržaja koji se pažljivo analizira i strukturira na temelju važnosti kako bi se osiguralo da cjelokupni dizajn bude konkretan i precizan (*sl 2.4.*). Povećavanjem dimenzija dodatni slojevi sadržaja mogu se dodavati postojećima ili izmijeniti strukturu kako bi korisnicima koji gledaju pomoću većih zaslona poboljšali korisničko iskustvo.

Sl 2.4. Mobile-first metodologija pristupa

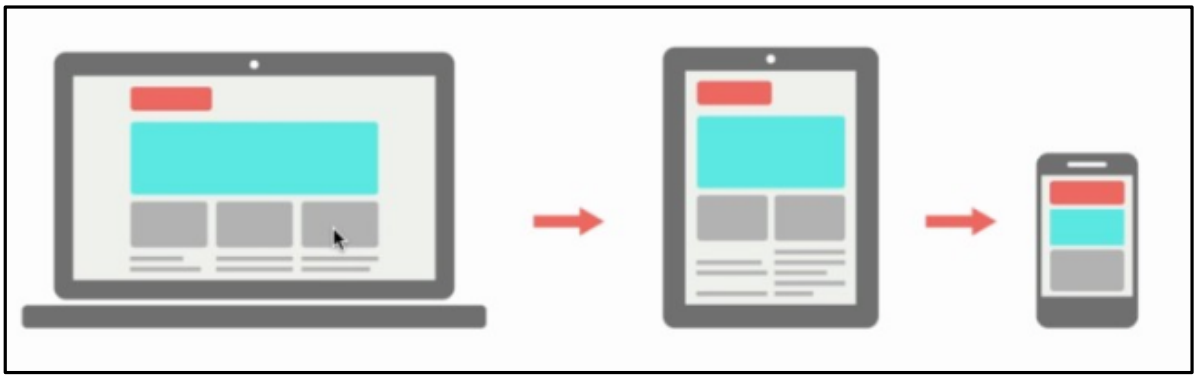
„Mobile-first" strategija može biti neizmjerno uspješna u svim aspektima, međutim ponekad ne odgovara za svako poslovanje. Ovakav pristup može biti puno skuplji zbog produljene faze izrade kao i stručnosti potrebne za stvaranje svestranosti određene platforme, što na kraju ne mora značiti da će generirati bolje rezultate. Idealno rješenje biranja ove metode ovisi o klijentu kao i o njegovim potrebama i pitanjima koje je potrebno postaviti, npr.:

1. Preferiraju li korisnici pretraživanje preko stolnog računala ili mobilnog uređaja?
2. Koriste li klijenti većinom mobilne uređaje za pregledavanje weba?
3. Ukoliko je to slučaj, zašto klijenti izbjegavaju posjetiti web stranicu s mobilnim uređajem?
4. Hoće li više klijenata posjetiti web stranicu putem mobilnog uređaja ako je sučelje prilagodljivo i namijenjeno mobilnim uređajima?

Osim na navedena pitanja, istraživanja su pokazala da se „Mobile-first" pokazao kao najisplativije rješenje pri sljedećim stavkama:

1. Mobilni promet iznosi 50% ili više, te prodaja se znatno povećava na mobilnim uređajima.
2. Vrsta proizvoda ili usluga koje se nudi spadaju u kategorije: zabava, stil života, umrežavanje, vijesti ili bilo koju drugu pokretnu kategoriju.
3. Dizajn platforme je jednostavan i minimalistički bez puno detalja.
4. Jednostavnost upotrebe, bez kompliciranih značajki pri korištenju.

S druge strane, „Desktop-first" metodologija usmjerava se na dizajn zaslona punih veličina popraćen raznim detaljima i interaktivnošću. S ovim tipom strategije nije bitno mobilno iskustvo nego se prioritet usmjerava na korisničko iskustvo sa stolnim računala koje mora biti detaljno i dinamično. Česti uzorak pri smanjivanju rezolucije kod ove metodologije predstavlja skrivanje ili premještanje sadržaja kako bi se zadržao prepoznatljivi koncept i što više originalnih značajki koje se nalaze na platformi (*sl 2.5.)*. Mobilno iskustvo postoji, ali ne i razrađeno u onoj mjeri kao i „Mobile-first" metodologija. Ova metodologija prigodna je za one složene web platforme koje su bogate raznim značajkama i imaju sučelja koja ciljaju na korisnike stolnih računala ili ostalih zaslona visokih rezolucija, posebno ako postoje Android ili iOs aplikacije navedene platforme koje bi poslužile kao poboljšanje korisničkog iskustva na mobilnim uređajima.



Sl 2.5. Desktop-first metodologija pristupa

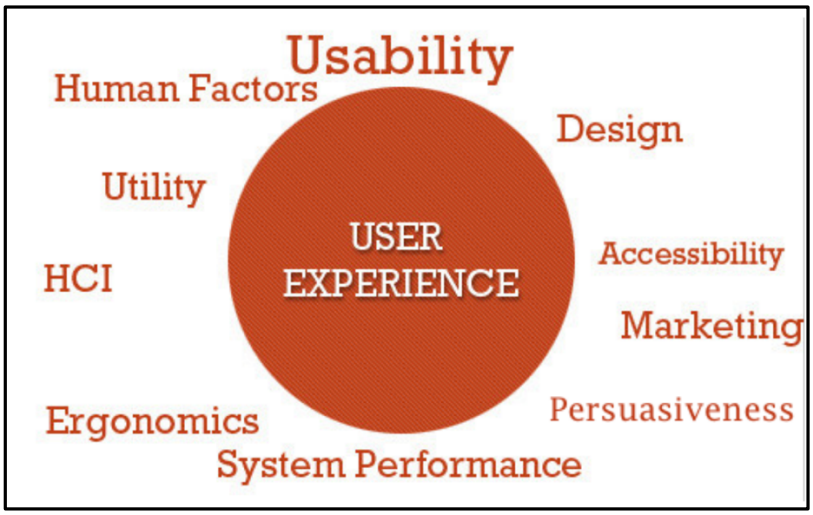
Neke od stavaka u kojemu se pokazalo da je „Desktop-first" strategija najisplativije rješenje:

1. Više od 50% prometa na platformi dolazi s radne površine te se prodaja znatno povećava na radnim površinama.
2. Web sučelje je složeno te je bogato raznim značajkama.
3. Postoji dodatna podrška u vidu Android ili iOs aplikacija za mobilne korisnike
4. Detaljni i složeni vizualni identiteti koji su dio marketinške strategije.
5. Za navedene funkcionalnosti potrebna je visoka rezolucija radne površina te jednostavna mobilna interakcija.

2.4. UX (korisničko iskustvo) i UI (korisničko sučelje) tehnike dizajna

Dizajn korisničkog iskustva (UX) predstavlja skup tehnologija i alata kojima je cilj povećanje korisničkog zadovoljstva poboljšavanjem uporabe sustava i koncepata vezanih uz interakciju između korisnika i platforme. Korisničko iskustvo predstavlja značajan aspekt u stvaranju različitih vrsta proizvoda i usluga. UX se usredotočuje na to kakav dojam cjelokupni dizajn ostavlja na korisnika i njegove osjećaje. Kako bi se potakli pozitivni osjećaji kod korisnika, tijekom korištenja platforme ili sustava, dizajneri trebaju shvatiti korisnikove ciljeve, želje, strahove, ponašanje te ambicije koje ih okružuju.

UX dizajn nije ograničen samo na vizualnu domenu nekog sučelja ili proizvoda. UX predstavlja koncept koji ima različite komponente koje utječu na UX domenu. Neki od značajnijih komponenti su: upotrebljivosti, korisnost, ljudski čimbenici, marketing, pristupačnost, uvjerljivost itd. (*sl 2.6.*). Glavni cilj UX dizajnera predstavlja isporuka proizvoda ili određene usluge koja će zadovoljiti potrebe kupaca i omogućiti im nesmetano postizanje željenog ishoda. Kako bi to postigli potrebno je provesti korisnička istraživanja kako bi se skupilo što više korisnih informacija o potrebnom cilju, a potom navedene informacije koristili za izradu prototipa.



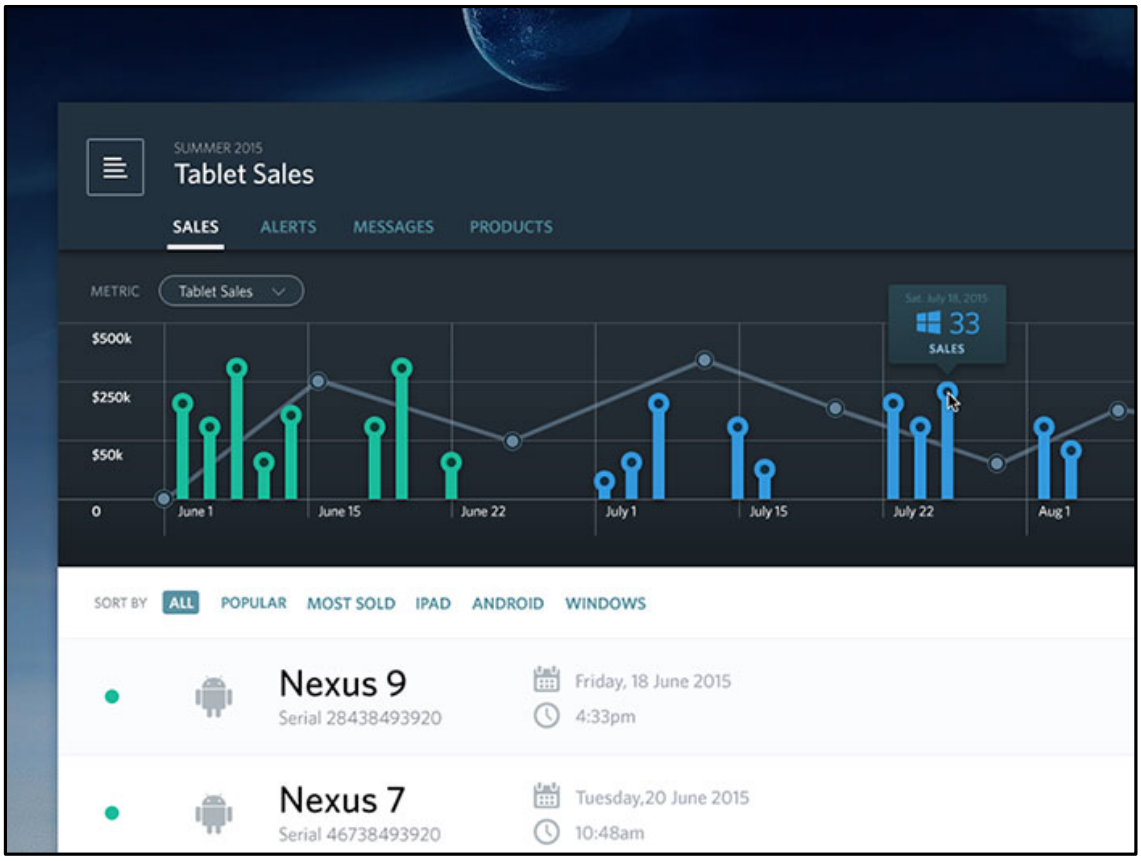
Sl 2.6. UX komponente koje utječu na UX domenu

Za razliku od dizajna korisničkog iskustva, dizajn korisničkog sučelja (UI) usmjerava se na dizajniranje i povezivanja sučelja kao most koji povezuje sustav s korisnikom. Općenito, cilj dizajna korisničkog sučelja predstavlja korisnikovu mogućnost da ostvari svoj cilj s minimalnom količinom otpora ili frustracije prilikom obavljana određenog rada na sustavu. Ukoliko je sučelje dobro osmišljeno korisnik ne mora razmišljati o onome što trenutno radi, nego je u stanju postići ono što je želio, a da izričito ne razmišlja o tome kako to napraviti. Dobar dizajn korisničkog sučelja predstavlja jedan široki spektar znanja, a postoje određene smjernice, odnosno pravila kojih se treba pridržavati pri dizajniranju dobrog korisničkog sučelja, a neka od njih su:

1. Sustav uvijek treba informirati korisnike o tome što se trenutno događa putem povratnih informacija u razumnim vremenskim razmacima.
2. Sustav treba biti lokaliziran jeziku korisnika, s riječima, izrazima ili konceptima koji su općenito poznati korisniku, a ne sustavno orijentiranim pojmovima.
3. Korisnici često pogrešno odabiru funkcionalnosti unutar sustava te je potrebno omogućiti ,,izlaz" kako bi napustili trenutno stanje, ne prolazeći kroz produžene dijaloge pri povratnim informacijama. Primjer tih izlaza predstavljaju ,,Poništi" i ,,Ponovi".
4. Korisnici se ne trebaju pitati predstavljaju li različite riječi, situacije ili određene radnje istu stvar. Potrebno je slijediti definirane standarde.
5. Smanjenje opterećenja korisnikovog pamćenja informacijama uvođenjem objekata, radnji i dodatnih opcija. Korisnik ne smije pamtiti informacije iz jednog dijaloga u drugi. Upute za uporabu sustava trebaju biti vidljive i lako dostupne u bilo kojem trenutku i lako vizualno dostupne.

6. Estetski i minimalistički dizajn. Dijalozi ne smiju sadržavati informacije koje su nevažne za korisnike ili rijetko potrebne. Sve dodatne informacije u dijalogu spajaju se s relevantnim informacijama i smanjuju njihovu relativnu vidljivost.
7. Sustav mora omogućiti korisniku da prepozna, dijagnosticira i oporavi se od pogrešaka. Poruke pogrešaka trebaju biti jasno izražene na prepoznatljivom jeziku (bez stručnih termina ili kodnih grešaka). Sustav mora precizno definirati problem i konstruktivno predložiti daljnje rješenje.
8. Potrebno je posjedovati i definirati dokumentaciju. Sve informacije unutar dokumentacije trebaju biti jednostavne za pretraživanje, usredotočene na korisnički zadatak te trebaju sadržavati popis konkretnih koraka kako ostvariti određeni cilj.

Dobar primjer dizajna korisničkog sučelja koji prati gore navedene smjernice prikazan je na *slici 2.7*.



Sl 2.7. Primjer dobro definiranog UI dizajna

3. NOVE WEB TEHNOLOGIJE

3.1. Pretprocesori SASS i LESS

Pretprocesori su programi koji primaju jednu vrstu podatka, obrađuju ga i pretvaraju u drugu vrstu podatka. U slučaju CSS-a, neki od najpopularnijih pretprocesorskih jezika su SASS i LESS. Pretprocesori dodaju nove funkcionalnosti CSS-u, smanjujući vrijeme rada i napore razvoja programeru. Pomoću pretprocesora moguće je poboljšati funkcionalnosti CSS-u sa sredstvima kao što su varijable, operatori, interpolacije, funkcije, miksini itd. Sva ova sredstva integriraju se u regularnu sintaksu CSS, pritom povećavajući učinkovitost i produktivnost.

Jedno od značajnijih svojstava predprocesora je da slijede DRY načelo(„Dont repeat yourself"), a ne WET(„Write everything twice") prilikom pisanja programskog koda, kako bi uštedjeli vrijeme razvojnim programerima iz razloga što ne moraju brinuti o ponavljanju pisanja programskog koda. Strukturalni elementi unutar stranice ostaju isti, a pretprocesuiranje, pogotovo u LESS-u, olakšava izradu modularnih modula. Razvojni programeri koriste varijable za definiranje boja, vizuala ili fontova te ih postavljaju na određene vrijednosti, bez potrebe za ponovnim definiranjem vrijednosti na svakom mjestu gdje je to predviđeno. Korištenjem pretprocesora CSS kod je organiziraniji, lako se održava te ga je moguće ponovo koristiti. Sve promjene se uređuju iz jednog mjesta.

SASS je izuzetno bogat i vrlo kompatibilan CSS pretprocesor, napisan u Ruby-u te je kompatibilan u radu na više operativnih platformi i integriran s raznim mogućnostima koje pružaju CSS pretprocesori. Brojni programski okviri („framework") su pisani u SASS-u a neki od najpopularnijih su Compass, Bourbon i Susy. SASS ima razne značajke kao što su korištenje varijabli, miksini, učitavanje SASS datoteka iz jedne u drugu, roditeljsko nasljeđivanje selektora itd. Postoje i ostale napredne značajke kao što su kontrolne direktive za biblioteke. SASS nudi mogućnost korištenja dvije vrste sintakse – SCSS i SASS. SASS sintaksa je starija sintaksa koja je poznatija kao i „pojednostavljena sintaksa". U navedenoj sintaksi nije potrebno pisati vitičaste zagrade niti je potrebno stavljati na kraj reda točku zarez. Kako bi se definirala svojstva za određeni selektor potrebno je uvući u redak napisana svojstva pod definirani selektor. Ova sintaksa prikladnija je za programere koji dolaze iz pozadine kao što je Ruby on rails ili Python. SCSS sintaksa je slična CSS-ovoj prirodnoj sintaksi. Svojstva određenog selektora pišu se unutar vitičastih zagrada, te se svaki redak svojstava s vrijednosti mora završiti s točka zarezom. SCSS sintaksa postaje sve više popularnija i istaknutija. Razlog tomu je što je navedena sintaksa

preglednija, nije potrebna prilagodba kao što je to u slučaju s SASS sintaksom. Većina razvojnih programera koristi SCSS sintaksu pri izradi raznih alata i biblioteka.

Inspiriran SASS-om, LESS se pojavio 2009. godine kao alternativa SASS-u. Pisan u Javascriptu, dok prva izvorna verzija je pisana u Ruby-u. Kao i Sass, LESS posjeduje mogućnosti varijabli, funkcija, operatora te miksina. Budući da je integriran zajedno s dinamičkim značajkama, razvojni CSS okviri, kao Bootstrap, napisani su u LESS-u. Zanimljiva značajka LESS-a je ta što se pokreće unutar NODE.js-a, preglednika i Rhino.js. LESS koristi alate trećih strana za pretvorbu izvornih datoteka u CSS datoteke. U usporedbi s ostalim pretprocesorima, LESS nudi jednostavniju i čistu sintaksu nalik CSS-u, što omogućuje lakši prijelaz na pisanje u pretprocesoru.

Varijable predstavljaju glavnu funkcionalnost svakog pretprocesora. U SASS-u deklaracija svake varijable započinje sa „ $ ” simbolom, dok kod LESS-a varijablu deklariramo s „@”. Kod SASS-a i LESS-a potrebno je nakon deklaracije varijable s dvotočkom (:) deklarirati vrijednost navedene varijable (*blok kod 3.1*). Definirana varijabla može se koristiti u svim pretprocesorskim datotekama u koje je uvezena umjesto klasičnog načina deklariranja entiteta u svakom dijelu CSS-a.

```
1.  // SASS
2.  $font-size: 16px;
3.  div {
4.      font-size: $font-size;
5.  }
6.
7.  // LESS
8.  @font-size: 16px;
9.  div {
10.     font-size: @font-size;
11. }
12.
13. // CSS prevedeni
14. div {
15.     font-size: 16px
16. }
```

Blok kod 3.1. Deklaracija varijabli pretprocesora

Miksini u pretprocesorima omogućuju ponovnu upotrebu stilskih blokova. Razvojni programeri ne moraju prolaziti kroz dugačke linije CSS-a kako bi izvršili ili dodali određene promjene. Sve promjene mogu se napraviti direktno unutar miksina. Miksini unutar SASS-a deklariraju se početnim simbolom „@” te mogu sadržavati parametre i definiraju se naredbom „import” unutar željenog selektora, dok se kod LESS-a miksini definiraju kao CSS klase koje se postavljaju unutar željenog selektora (*blok kod 3.2*).

```
// SASS
@mixin huge($width, $fnt-size) {
    border: $width solid #ddd;
    font-size: $fnt-size;

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

h1 {
    @include bordered(5px, 48px);
}

// LESS
.huge (@width,@fnt-size) {
    border: @width solid #ddd;
    font-size: @fnt-fize;
    &:hover {
        border-color: #999;
    }
}

h1 {
    .huge(5px,48px);
}

// CSS prevedeni

h1 {border: 5px solid #ddd;font-size: 48px;}
h1:hover {border-color: #999;}
```

Blok kod 3.2. Deklaracija miksina u pretprocesorima

CSS ne posjeduje mogućnost hijerarhije tijekom rada s roditelj-dijete selektorima. Kako bi dohvatili dijete selektor unutar roditelja, potrebno je pisati kombinacije selektora odvajajući ih razmacima. Prilikom korištenja pretprocesora to svojstvo postoji pružajući time vizualnu hijerarhiju na način kako je definirana i unutar HTML-a (*blok kod 3.3*). Potrebno je voditi brigu tijekom korištenja hijerarhije jer može doći do grešaka unutar koda ukoliko se koristi pretjerana količina ugnježđivanja unutar roditelj-dijete selektora.

```
ul {
    margin: 0;
    li {
        float: left;
    }
}

// LESS
ul {
    margin: 0;
    li {
        float: left;
    }
}

// CSS prevedeni
ul {margin: 0;
```

```
ul li {float: left;}
```

Blok kod 3.3. Korištenje ugnježđivanja unutar pretprocesora

Osim korištenja miksina, pretprocesori posjeduju mogućnost i nasljeđivanja. Nasljeđivanje se koristi u onim slučajevima gdje su definirana jedinstvena svojstva određenih selektora, koja mogu naslijediti ostali selektori umjesto njihovog kopiranja (*slika 3.4*).

```
// SASS
.block {
    margin: 10px 5px;
}

p {
    @extend .block;
    border: 1px solid #eee;
}

ul,
ol {
    @extend .block;
    color: #333;
    text-transform: uppercase;
}

// LESS
.block {
    margin: 10px 5px;
}

p {
    &:extend(.block);
    border: 1px solid #eee;
}

ul,
ol {
    &:extend(.block);
    color: #333;
    text-transform: uppercase;
}

//  CSS prevedeni
.block, p, ul, ol { margin: 10px 5px; }
p { border: 1px solid #eee; }
ul, ol { color: #333; text-transform: uppercase; }
```

Blok kod 3.4. Nasljeđivanje unutar pretprocesora

SASS i LESS imaju funkcionalnost razdvajanja programskog koda na manje, upravljive dijelove u smislu lakšeg održavanja i kontroliranja razvojnog koda. Moguće je grupirati slične dijelove i mape programskog koda te sve uključiti u glavnu datoteku CSS-a (*blok kod 3.5*).

```
@import "library";
@import "mixins/mixin.scss";
@import "reset.css";
```

Blok kod 3.5. Uključivanje dijelova programskog koda pretprocesora

Dok SASS podržava normalno korištenje if/else uvjetnih grananja, u LESS-u se to postiže pomoću CSS „čuvara". Pretprocesori koriste petlje kako bi iterirali kroz navedene uvijete kreirajući pritom niz stilova (*blok kod 3.6*).

```
// SASS
@if lightness($color) > 30% {
    background-color: black;
}
@else {
    background-color: white;
}

// LESS
.mixin (@color) when (lightness(@color) > 30%) {
    background-color: black;
}
.mixin (@color) when (lightness(@color) =<; 30%) {
    background-color: white;
}
```

Blok kod 3.6. Uvjetno grananje kod pretprocesora

Kao i if/else uvjetno grananje, SASS koristi normalan pristup za iteraciju petlje. LESS koristi css „Čuvare" i rekurzivni miksin za iteriranje petlje, gotovo sličan pristup kao i korištenje if/else uvjetnog grananja (*blok kod 3.7*).

```
// SASS
@for $i from 1px to 3px {
    .border-#{i} {
        border: $i solid blue;
    }
}

// LESS
.loop(@counter) when (@counter > 0){
    .loop((@counter - 1));
    .border-@{counter} {
        border: 1px * @counter solid blue;
}
```

Blok kod 3.7. Iteriranje petlje kod pretprocesora

3.2. PostCSS

Za razliku od pretprocesora SASS-a i LESS-a, PostCSS je relativno nova tehnologija koja želi promijeniti ekosustav pisanja CSS stilova putem prilagođenih datoteka i alata. Za razliku od SASS-a i LESS-a, PostCSS koristi Javascript (točnije Node.js okvir) koji ima sposobnost brzog

transformiranja stilova u odnosu na ostale pretprocesore. Pomoću alata za automatizaciju kao što su Gulp i Grunt, stilovi se mogu transformirati kroz „build“ proces izgradnje, na sličan način kao što se SASS i LESS izgrađuju. PostCSS je početno lansiran kao metoda za korištenje Javascript-a kao podloge za obradu CSS-a. PostCSS po sebi predstavlja vrlo jednostavan API, koji se koristi svojim ogromnim ekosustavom dodataka, samim time pružajući široki spektar funkcionalnosti i raznolikosti kako bi korisnik odabrao onu kombinaciju dodataka koja mu je potrebna za obavljanje ciljanog zadatka. Zbog upotrebe API-a, PostCSS omogućuje stvaranje prilagođenih dodataka i alata za ona svojstva koja su potrebna. Ovakav modularan pristup kreiranja funkcionalnosti čini alat neutralnim sredstvom te glavni fokus stavlja na zahtjeve projekta koje je potrebno obaviti. PostCSS predstavlja agnostički jezični format koji omogućava pisanje sintakse različitih pretprocesora, kao što su SASS i LESS, ukoliko je to potrebno.

Kako bi instalirali PostCSS model u projektu, potrebno je imati instaliran Node.Js okvir kako bi mogli preuzeti modul naredbom: npm install gulp-postcss –D. Unutar Gulp automatizacijske datoteke *gulpfile.js* navodimo instalirani PostCSS modul i koristimo ga unutar definiranog automatizacijskog zadatka. Primjer zadatka definiran je u *blok kodu 3.8.* Zadatak definira ulazne puteve koje će dohvatiti PostCSS, generirati izlaznu datoteku te spremiti na putanju koja je definirana u automatizacijskom zadatku.

```
1. gulp.task('styles', function () {
2.     return gulp.src('dev/style.css')
3.         .pipe(postcss([]))
4.         .pipe(gulp.dest(path/to/prod))
5. });
```

Blok kod 3.8. Definiranje PostCSS modula unutar gulpfile.js datoteke

Kako bi pokrenuli automatizacijski zadatak, potrebno je unutar konzole upisati naredbu: *gulp styles.* Upotreba PostCSS-a sama po sebi nije dovoljna, njegova snaga leži u instaliranju i definiranju raznih dodataka. Prilikom definiranja automatizacijskog zadatka sve unutar vitičastog polja predstavlja alat unutar kojega se mogu definirati razne funkcionalnosti iz instaliranih dodataka.

Kako bi instalirali dodatak potrebno je unijeti sljedeću naredbu: *npm install (naziv-dodatka).* Za navedeni primjer koristi se dodatak *Autoprefixer. Autoprefixer* je dodatak koji omogućuje pisanje novih CSS značajki, bez potrebe za ručno definiranje prefiksa navedenih svojstava za starije pretraživače koji ne podržavaju sami po sebi nove značajke. Za instaliranje Autoprefixer-a potrebno je unijeti naredbu: *npm install autoprefixer –D*. Nakon instalacije dodatka potrebno je unutar gulpfile.js datoteke izmijeniti definirani kod prema *blok kodu 3.9*. Ukoliko je

definirano i instalirano više od jednog dodatka, dobra je praksa stvoriti novo polje unutar kojega su definirani dodatci. U tom slučaju unutar automatizacijskog zadatka potrebno je kao parametar predati definirano polje.

```
1.  var plugins = [
2.    require('autoprefixer')({ browsers: ['last 2 versions', 'ie 6-
    8', 'Firefox > 20']  })
3.  ];
4.
5.  gulp.task('styles', function () {
6.      return gulp.src('dev/style.css')
7.          .pipe(postcss(plugins))
8.          .pipe(gulp.dest(path/to/prod))
9.  });
```

Blok kod 3.9. Izmijenjeni automatizacijski zadatak unutar gulpfile.js

Navedeni prefiksi biti će proslijeđeni svakom CSS svojstvu kojemu je to potrebno, prema predviđenom objektu koji je poslan (*blok kod 3.10*).

```
1.  // Definirani stil (ulaz)
2.  .item {
3.    display: flex;
4.    flex-flow: row wrap;
5.  }
6.
7.  // Izlaz definiranog stila poslije automatizacijskog zadatka
8.  .item {
9.    display: -webkit-flex;
10.   display: -ms-flexbox;
11.   display: -webkit-box;
12.   display: flex;
13.   -webkit-flex-flow: row wrap;
14.       -ms-flex-flow: row wrap;
15.           flex-flow: row wrap;
16. }
```

Blok kod 3.10. Definirani ulaz i izlaz poslije izvršenog automatizacijskog zadatka

3.3. Javascript ES6

Javascript je programski jezik napravljen u nadi da će kapitalizirati uspjeh Java programskog jezika. Tvrtka Netscape je priložila Javascript ECMA internacionalna organizacija za standardizaciju programskih jezika. Upravo to je rezultiralo novim jezičnim standardom poznatijim kao ECMAScript. ES predstavlja jednostavnu kraticu za ECMAScript. Uz svaku ES kraticu prilaže se određeni broj koji pokazuje na određenu verziju ECMAScript-a. ES6, također poznat kao i ECMAScript 2015, trenutno je najnovija i radna verzija ECMAScript standarda, također i prvo ažuriranje jezika od ES5 standarda koje je postavljeno 2009 godine. Neke od značajnih funkcionalnosti ES6 standarda predstavljaju funkcijske strelice, klase, objektni literali, destrukturiranje, moduli itd.

Strelice predstavljaju skraćenu deklaraciju za funkcije definiranu sa simbolom „ =>”. Imaju slične značajke kao strelice u programskim jezicima C#, Java i Coffescript. Sintaksa sa strelicama podržava blok izraze kao i uvjetne blokove koji vraćaju određenu vrijednost izraza. Za razliku od standardnih funkcija, deklaracija strelicama dijeli isti leksički „this” izraz kroz njihov definirani okolni kod (*blok kod 3.11*).

```
1.  // Blok izrazi
2.  var odds = evens.map(v => v + 1);
3.  var nums = evens.map((v, i) => v + i);
4.  var pairs = evens.map(v => ({even: v, odd: v + 1}));
5.
6.  // Uvjetni izrazi
7.  nums.forEach(v => {
8.    if (v % 5 === 0)
9.      fives.push(v);
10. });
11.
12. // Lexical this
13. var bob = {
14.   _name: "Bob",
15.   _friends: [],
16.   printFriends() {
17.     this._friends.forEach(f =>
18.       console.log(this._name + " knows " + f));
19.   }
20. }
```

Blok kod 3.11. Korištenja ES6 strelica prilikom deklaracije funkcija

ES6 klase prate pojednostavljeni zapis prototipa temeljen na objektno-orijentiranim uzorcima. Klase su ustvari „posebne funkcije” koje se mogu definirati kao funkcijski izrazi i kao deklaracija funkcija, a klasna sintaksa ima dvije glavne komponente: klase definirane izrazom i klase definirane deklaracijom (*blok kod 3.12*). Klase podržavaju objektne prototipe koje imaju svojstvo nasljeđivanja, pozivanje roditeljskih funkcija, kreiranje instanci, te implementiranje konstruktora i statičkih metoda.

```
1.  // Klasa definirana deklaracijom
2.  class Rectangle {
3.    constructor(height, width) {
4.      this.height = height;
5.      this.width = width;
6.    }
7.  }
8.
9.  // Klasa definirana izrazom
10. let Rectangle = class {
11.   constructor(height, width) {
12.     this.height = height;
13.     this.width = width;
14.   }
15. };
```

Blok kod 3.12. ES6 vrste definiranja klasa

ES6 nudi poboljšanu mogućnost rada s tekstualnim i znakovnim tipovima podataka. Tekstualni literarni predlošci nalaze se unutar (``) navodnika umjesto dosad standardnih jednostrukih ili dvostrukih navodnika. Literarni izrazi mogu sadržavati i rezervirana mjesta za deklariranje varijabli. Izraz se definira znakom dolara i zatim unutar vitičastih zagrada postavi naziv varijable ili funkcije - (${naziv}). Izraz definiranog teksta i varijable prenosi se zajedno u funkciju. Zadana funkcija zatim povezuje sve dijelove u jedan string. Kada dođe do određenog izraza koji je definiran kao literarni izraz, poziva se funkcija koji obrađuje definirani izraz, vraća vrijednost toga izraza i nastavlja dalje s radom (*blok kod 3.13*).

```
1.  // Obična deklaracija literarnog teksta
2.  `Javascript 123456`
3.
4.  // Deklaracija višerednog literarnog teksta
5.  `Javascript
6.  in a new row`
7.
8.  // Interpolacija teksta s literarnim izrazima
9.  var name = "Bob", time = "today";
10. `Hello ${name}, how are you ${time}?`
```

Blok kod 2.13. ES6 definiranje teksta s literarnim izrazima

ES6 Javascript pruža dva nova tipa podatka – *let* i *const*. *Let* dopušta deklaraciju varijabli koje su ograničene samo na programski blok, stanje ili izraz u kojem se upotrebljavaju, za razliku od dosadašnje *Var* deklaracije koja definira varijablu na globalnoj ili lokalnoj razini na cijelu funkciju bez obzira na programski blok u kojem se nalazi. Za razliku od *Let* deklaracije, deklaracija *Const* varijable definira referencu na vrijednost koja se može samo čitati. To ne znači da se vrijednost koju ona drži onda ne može promijeniti, nego da identifikator varijable ne može biti ponovno dodijeljen (*blok kod 3.14*).

```
1.  function f() {
2.    {
3.      let x;
4.      {
5.        // ok
6.        const x = "sneaky";
7.        // greška, x ne može mjenjati vrijednosti
8.        x = "foo";
9.      }
10.     // greška, x je već deklariran unutar bloka
11.     let x = "inner";
12.   }
13. }
```

Blok kod 3.14. Deklariranje let i const varijable

ES6 pruža podršku na razini jezika za module i definiranje komponenti. Ponašanje prilikom pokretanju definirano je od strane izvora koji definira učitavanje komponenti. Navedeni pristup predstavlja implicitno asinkroni model – kod se ne može izvršiti sve dok zahtjev za modul nije dostupan i procesuiran. Postoje dva stanja – *import* i *export*. *Import* se koristi za uvoz komponenti iz drugih modula. Uvozni moduli su uvezeni u *strict(strogom)* načinu rada bez obzira jesu li deklarirani kao takvi ili ne. *Import* se ne može koristit u ugrađenim skriptama osim ako skripta nema tip oznake „modul". Export se koristi prilikom izrade modula pri čemu je glavni cilj izvoz funkcija, klasa, objekata ili primitivnih tipova vrijednosti iz modula kako bi ih drugi programi mogli uvesti u svoj programski blok (*blok kod 3.15*).

```
1.  // lib/math.js
2.  export function sum(x, y) {
3.    return x + y;
4.  }
5.  export var pi = 3.141593;
6.
7.  // app.js
8.  import * as math from "lib/math";
9.  alert("2π = " + math.sum(math.pi, math.pi));
10.
11. // sandbox.js
12. import {sum, pi} from "lib/math";
13. alert("2π = " + sum(pi, pi));
```

Blok kod 3.15. Deklariranje import i export modula u ES6

3.4. CSS4 (Selectors level 4)

CSS4 (Selectors level4) ne predstavlja veliku tehnološku nadogradnju na postojeći CSS3. CSS4 predstavlja postepenu nadogradnju i poboljšanje postojećih funkcionalnosti i noviteta CSS definicija koje trebaju doći u budućnosti. Trenutačna specifikacija CSS4 je nova te trenutno ne postoji preglednik koji zapravo podržava nova pravila [10].

Glavna značajka koja se trenutno ističe u CSS4 specifikaciji predstavlja podršku za dohvaćanje roditeljskog selektora. Na primjer, ako postoji sljedeći vezani niz selektora: *body header h1* – CSS pravila se odnose samo na onaj selektor koji je zadnji u nizu. Problem nastaje kada želimo dodati nešto u roditeljski selektor elementa koji ima točno određeni dijete selektor. CSS4 rješava problem tako da pored imena roditeljskog selektora doda oznaku „ ! " što znači da se roditeljski selektor *header* dohvaća samo ukoliko postoji njegov dijete selektor *h1* koji ga veže u nizu (*blok kod 3.16*).

```
1.  // CSS3
2.  body header{
3.    background: red;
```

```
}
body header h1 {
    font-size: 16px;
}

// CSS4
body header! h1 {
  background: red;
}
body header h1 {
    font-size: 16px;
}
```

Blok kod 3.16. CSS4 dohvaćanje roditeljskog selektora

Osim roditeljskih selektora, unutar CSS4 specifikacije definiraju nove pseudo klase. Jedna od novih pseudo klasa je logička pseudo klasa *:matches* i *:not*. S logičkim pseudo klasama, moguće je grupirati različite predmete unutar CSS dokumenta tako da se uvede uvjetna logika unutar CSS-a (*blok kod 3.17*). U CSS3 za svaki selektor i njegovo stanje potrebno je eksplicitno definirati pseudo klasu koju želimo koristiti dok je u CSS4 s uvjetnom pseudo klasom moguće kombinirati uvjetnu logiku određenog selektora.

```
// CSS3
ul.menu li a:link,
ul.menu li a:hover,
ul.menu li a:visited,
ul.menu li a:focus {
  color: red;
}

// CSS4
ul.menu li a:matches(:link, :hover, :visited, :focus) {
    color: red;
}

p:not(.active, .visible) {
  color: red;
}
```

Blok kod 3.17. Logičke pseudo klase

Osim logičkih postoje i lokacijske pseudo klase *:any-link* i :*local-link* koje daju mnogo veću kontrolu prilikom stiliziranja linkova na lokalne ili eksterne poveznice. CSS3 trenutno posjeduje pseudo klase *:link* i *:visited* koje nova lokacijska pseudo *:any-link* klasa kombinira u jednu pseudo klasu (*blok kod 3.18*). Za razliku od *:any-link* pseudo klase, *:local-link* omogućava stiliziranje linkova koji vode lokalnu poveznicu unutar stranice time dajući kontrolu stiliziranja na linkovima koje vode izvan i unutar stranice.

```
// CSS3
a:link,
a:visited {
  color: red;
```

```
}

// CSS4
a:any-link {
  color: red;
}

nav :local-link {
  text-decoration: none;
}
```

Blok kod 3.18. Lokalizacijske pseudo klase

Trenutno unutar CSS3 postoje tri pseudo klase za kontroliranje UI stanja: *:enable, :disabled* i *:checked.* CSS4 nudi mogućnost stiliziranja UI ulaza ukoliko oni nisu bili aktivirani ili su izgašeni. Za navedene radnje koristi se pseudo klasa :indeterminate (*blok kod 3.19*).

```
// CSS3
input.checkbox:checked {
   background: #f00;
}

// CSS4
input.checkbox:indeterminate {
   background: #ccc;
}
```

Blok kod 3.19. Definiranje pseudo klase za promjenu UI stanja

Osim lokacijskih, logičkih i pseudo klasa promjena stanja, CSS4 definira nove pseudo klase za višestruko strukturiranje selektora *:nth-match i :nth-last-match*. :nth-match pseudo klase rješavaju problem nemogućnosti ispitivanja dodatnih logičkih uvjeta trenutnih pseudo klasa *:nth-child* i *:nth-of-type.* Prema primjeru *blok koda 3.20* pokazuje se rješavanje problema selektiranja samo onih elemenata koji su parni u nizu strukture te posjeduju klasu *.active.* Ne može se uzeti u obzir logička pseudo klasa *:matches(.active)* jer bi se tada selektirali svi elementi koji imaju klasu *.active.*

```
// CSS4
li a:nth-match(even of .active) {
   color: red;
}
```

Blok kod 3.20. Definiranje strukturne psuedo klase :nth-match

4. E-TRGOVINA NA PRIMJERU SYLIUS PLATFORME

Mnogi digitalni startup-i suočavaju se s istim izazovima prilikom pokretanja poslovanja – odlučiti se za već postojeće poslovno rješenje ili napraviti prilagođeni softver koji će odgovarati točnim potrebama njihovog poslovanja i samim time dati veću kontrolu fleksibilnosti softvera. Prvo rješenje predstavlja jeftiniju soluciju i predstavlja čestu praksu u smislu eksperimentiranja i pokretanja projekta u produkciju što je prije moguće, ali ponekad to donosi komplikacije u dugoročnom pogledu. Izrada prilagođenog softvera „otvorenog koda" predstavlja u današnje vrijeme kompromis. Glavna rješenja su rijetko prihvatljiva za razvojne programere. Velika i monolitna softverska rješenja dobro obavljaju posao za standardne poslovne modele i operacije, ali ne i za poduzeća koja u svome poslovanju žele doprinijeti inovacije i nove poslovne trendove. U navedenom poglavlju definiraju se pojmovi e-trgovina i Sylius platforma za e-trgovinu kao i analiza arhitekture i način rada Sylius platforme za e-trgovinu.

4.1. E-trgovina i njene prednosti

Pojam e-trgovina možemo definirati kao skup svih komercijalnih aktivnosti koje se izvode putem interneta. Sve poslovne aktivnosti odvijaju se unutar prostora električnog poslovanja koje se definira kao e-trgovina. E-poslovanje se pojavljuje sve češće, a obuhvaća sve vrste komercijalnih djelatnosti koje se obavljaju putem interneta, kao što je Internet bankarstvo, trgovanje dionicama i obveznicama, aukcije, promet nekretninama, rezerviranje avionskih i ostalih prometnih karata, najam prostora itd. E-poslovanje predstavlja neovisnu, sofisticiranu platformu koja obuhvaća veliki operativni dio svjetskog gospodarstva. e-trgovina pruža velike mogućnosti i prilike za vlasnike stvarnih poslovanja koji se žele istaknuti kao uspješni poduzetnici, koristeći neograničene informacije i dostupne alate koji su im omogućeni preko interneta. Navedene tehnologije koriste se danas za razne vrste transakcija, uključujući one koje se ne temelje samo na internetu. Transakcije putem kreditnih kartica, provjere podataka, poslovne komunikacije, marketinške strategije i aktivnosti, istraživačke aktivnosti itd. temelje se na navedenim tehnologijama.

Svaka komercijalna djelatnost izvodi se putem interneta, elektronskog instrumenta ili putem umreženih uređaja te predstavlja neovisnu i jedinstvenu vrstu aktivnosti unutar e-trgovine. Postoje određene vrste e-trgovina koje je moguće kategorizirati prema primarnim komercijalnim aktivnostima koje određena tvrtka obuhvaća:

- B2B(Business to Business) - Tvrtka se bavi elektroničkim komercijalnim aktivnostima, obavljajući poslovne aktivnosti e-trgovine s drugom tvrtkom koja predstavlja kupca.
- B2C(Business to Customer) - Tvrtka se bavim određenom vrstom elektroničkih komercijalnih aktivnosti, obavljajući poslovne aktivnosti e-trgovine s kupcima, odnosno fizičkim osobama.
- C2B(Customer to Business) – Određeni kupac bavi se elektroničkim komercijalnim aktivnostima, obavljajući poslovne aktivnosti e-trgovine s jednom ili više tvrtki.
- C2C(Customer to customer) – Određeni kupac bavi se elektroničkim komercijalnim aktivnostima, obavljajući poslovne aktivnosti e-trgovine s kupcima, odnosno fizičkim osobama.

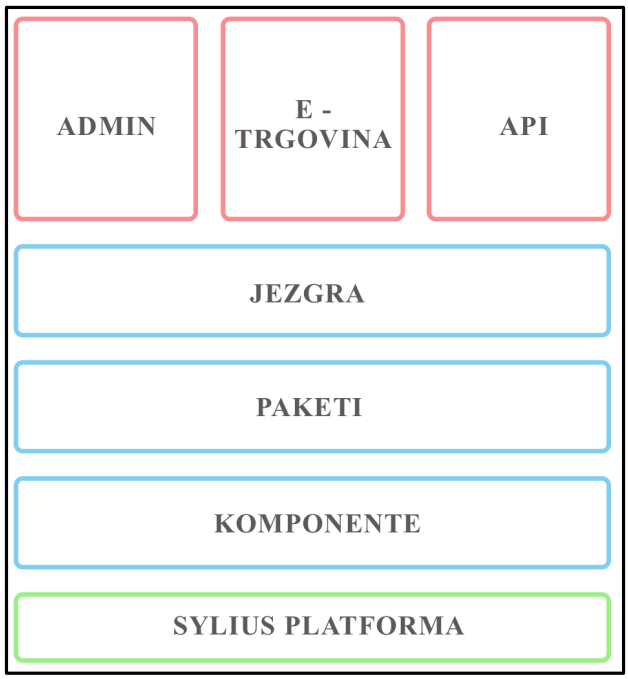
E-trgovina nudi mnoge prednosti tvrtkama. Jedna od prednosti predstavlja online marketing. Efektivniji marketing znači i veći profit. Definiranje online marketinga od strane tvrtke omogućuje prezentiranje proizvoda i usluga u svim dijelovima svijeta neprekidno u bilo kojem vremenskom trenutku. Tvrtke imaju veću priliku povećati vlastiti profit kao i priliku pristupa većem broju ciljanih korisnika. Proces pripreme i održavanja web platforme za online trgovinu jeftiniji je od iznajmljivanje ili konstruiranja maloprodajnih i veleprodajnih trgovina. Tvrtke ne trebaju trošiti ogromne proračunske novce na promotivne aktivnosti ili instalirati skupu opremu koju bi korisnici koristili. Opseg online trgovine nije određen udaljenostima ili granicama, on je neograničen, omogućujući učinkovitu komunikaciju s partnerima i kupcima, ali i priliku za tvrtke da iskoriste prednosti svojih dobara i usluga u odnosu na svoje konkurente. Osim za tvrtke, postoje i razne prednosti za kupce kao što je ušteda vremena i truda da se pronađe ciljani proizvod, imajući u vidu slobodu izbora, snižene cijene itd. E-trgovina ne predstavlja još jedan poslovni poduhvat za tvrtke, on predstavlja cijeli jedan novi sustav poslovanja svjetskog gospodarstva. Trenutno zauzima značajan dio svjetskog tržišta i generira ogromne prihode. Profit ne smije predstavljati glavni cilj pokretanja elektroničkog poslovanja ili usvajanja online trgovine, već cilj treba predstavljati tehnološki napredak, razumijevanje tržišnog razvoja te zdravo poslovanje tvrtke.

4.2. Sylius platforma za e-trgovinu

Sylius predstavlja novi pristup u razvoju e-trgovina baziranoj na Symfony razvojnom okviru. Sylius omogućava standardno B2C sučelje i upravljačku ploču, što u stvarnosti predstavlja samo jedan dio strukture jezgre sustava. Za razliku od drugih platformi za e-trgovinu, Sylius-ov koncept zamišljen je da bude podloga za razvoj prilagođenog softvera koji pruža neograničene mogućnosti. Neke od mogućnosti su:

1. Kreiranje vlastitog identiteta e-trgovine kao i administratorskog sučelja.
2. Izrada vlastitog korisničkog sučelja od nule s mogućnošću upotrebe Sylius-ove poslovne logike.
3. Dodavanje novih i upotreba komponenti iz Sylius biblioteke.
4. Dodavanje Sylius-a kao komponente u vlastiti Symfony projekt.
5. Korištenje Sylius-a kao API.

Sylius-ova podloga temelji se na Symfony okviru, koji je vodeći PHP okvir za izradu web aplikacija. Korištenja Symfony okvira omogućuje razvojnim programerima da rade efektivnije pružajući im sigurnost u razvoju aplikacije koja je strukturirana, lako održiva i omogućuje kraći vremenski period izrade korištenjem ponovno upotrebljivih modula. Osim Symfony-a, Sylius koristi Doctrine razvojni okvir za pružanje funkcionalnosti rada sa složenim slojevima podataka. Jedna od ključnih značajki Doctrine-a predstavlja mogućnost pisanje upita u posebnom Doctrine razvojnom jeziku – objektno orijentiranom dijalektu SQL-a, koji omogućava lakše korištenje i manipulaciju nad slojevima podataka unutar baze podataka. Za obradu pogleda na zaslonu Sylius koristi Twig, koji predstavlja moderan predložak koji posjeduje vlastiti jezik koji je brz, siguran i fleksibilan za pisanje PHP-a unutar HTML-a. Na *slici 4.1* definiran je simboličan prikaz Sylius-ove arhitekture.



Sl 4.1. Prikaz Sylius-ove arhitekture

Sylius platforma sastoji se od nekoliko ključnih elemenata. Svaka pojedina komponenta unutar Syliusa može se koristiti samostalno u odnosu na druge komponente. Primjer tomu je

komponenta za porez. Zadatak komponente za porez predstavlja izračun poreza bez obzira hoće li to biti porezi za određene proizvode ili za nešto drugo, te predstavlja samostalnu funkcionalnu cjelinu koja obavlja taj zadatak. Kako bi omogućili komponentu za oporezivanje da radi nad bilo kojim objektom unutar web aplikacije potrebno je implementirati odnosno pozvati komponentu unutar programskog sučelja (interface). Pored komponenti, paketi predstavljaju drugu važnu stavku unutar Sylius-ove arhitekture. Oni predstavljaju grupaciju manjih gotovih komponenti unutar Sylius-a kao što su servisi, modeli, konfiguracije itd. koje su spremne za korištenje. Ukoliko razvojni programer nema vremena za konfiguriranje i definiranje servisa modela i sl. potrebnih za rad komponente za oporezivanje prilagođen njegovim potrebama, može pozvati Sylius-ov paket za oporezivanje s minimalno ili bez ikakve konfiguracije potrebne za rad. Sylius platforma je aplikacija koja nudi potpunu uslugu i funkcionalnosti koji su potrebni jednoj e-trgovini. Prije početka rada na Sylius-u, Sylius nudi mogućnosti biranja platforme sa svim značajkama ili samo s odabranim paketima i komponentama kako bi se razvilo prilagođeno rješenje. Sylius platforma je sama po sebi vrlo fleksibilna i nudi mogućnost lake prilagodbe svim definiranim poslovnim zahtjevima koje su potrebni na platformi. Jezgra predstavlja glavnu komponentu unutar koje su integrirane sve ostale komponente. Sylius unutar jezgre ima potpuno integrirani koncept svega što mu je potrebno za pokretanje jedne e-trgovine.

U svakom sustavu s definiranim sigurnosnim slojem funkcionalnosti administracije mora biti ograničen samo na određene korisnike s određenom ulogom – administratore. Admin sustav definiran je Admin paketom unutar Syliusa. E-trgovina je kategorija koja je definirana Shop paketom, predstavlja standardno B2C sučelje za sve što se događa unutar sustava. Uglavnom je izrađena od YAML konfiguracije i predložaka. Pogledi su generirani s Twigom, a sami dizajn definiran je SemanticUI CSS bibliotekom. Sylius API upotrebljava REST pristup manipuliranja podatcima. Definirani kontroleri unutar Syliusa definirani su tako da se mogu koristiti i upotrijebiti unutar API-a. Ukoliko se koristi Sylius API na nekom drugom razvojnom okruženju, potrebno je koristiti točno onu akciju koja je definirana unutar zahtjeva API-a.

Osim navedenih kategorija Sylius-ove arhitekture, Sylius ima mogućnost korištenja biblioteka treće strane za različite zadatke koji nisu implementirani unutar standardnog Sylius-ovog okruženja:

1. KNP Menu – Za napredne postavke trgovine i administratorskog sučelja.
2. Gaufrette – pohranjivanje slika i medija lokalno ili preko vanjskih poslužitelja.
3. Imagine – biblioteka za obradu i generiranje slika.

4. Pagerfanta – napredna paginacija.

4.3. Kreiranje Sylius projekta

Sylius može poslužiti kao aplikacija za krajnjeg korisnika, kao i temelj za prilagođenu aplikaciju e-trgovine. Kako bi kreirali novi Sylius-ov projekt potrebno je instalirati Composer, alat za instalaciju PHP paketa. Composer prilikom Sylius instalacije također instalira sve biblioteke treće strane koje Sylius i Symfony koristi u svome radu bez da korisnik ručno instalira bilo koju biblioteku.

Kako bi kreirali novi Sylius projekt potrebno je unutar terminala upisati naredbu prema *blok kodu 4.1.*

```
1. $ composer create-project sylius/sylius-standard hajduk
```

Blok kod 4.1. Kreiranje Sylius projekta

Navedena naredba kreira novi Symfony projekt unutar hajduk direktorija. Kada sve biblioteke potrebe Symfony-u budu instalirane, pokrenut će se parametarska skripta *parameters.yml* koju je potrebno ispuniti. Nakon što instalacija završi potrebno je upisati naredbu prema *blok kodu 4.2.* Navedena naredba instalira sve navedene pakete koje čine Sylius unutar Symfony-evog *Vendor* direktorija te predstavlja odvojenu funkcionalnu cjelinu kako bi glavni fokus ostao na prilagođenom razvoju projekta.

```
1. $ cd hajduk # Ulazimo u direktorij nazvan hajduk
2. $ php bin/console sylius:install
```

Blok kod 4.2. Instaliranje Sylius-a

Nakon instaliranja Sylius-a potrebno je pokrenuti skripte za generiranje resursa koji čine Sylius-ov prikaz na frontend-u prema *blok kodu 4.3.*

```
1. $ yarn install
2. $ yarn build
```

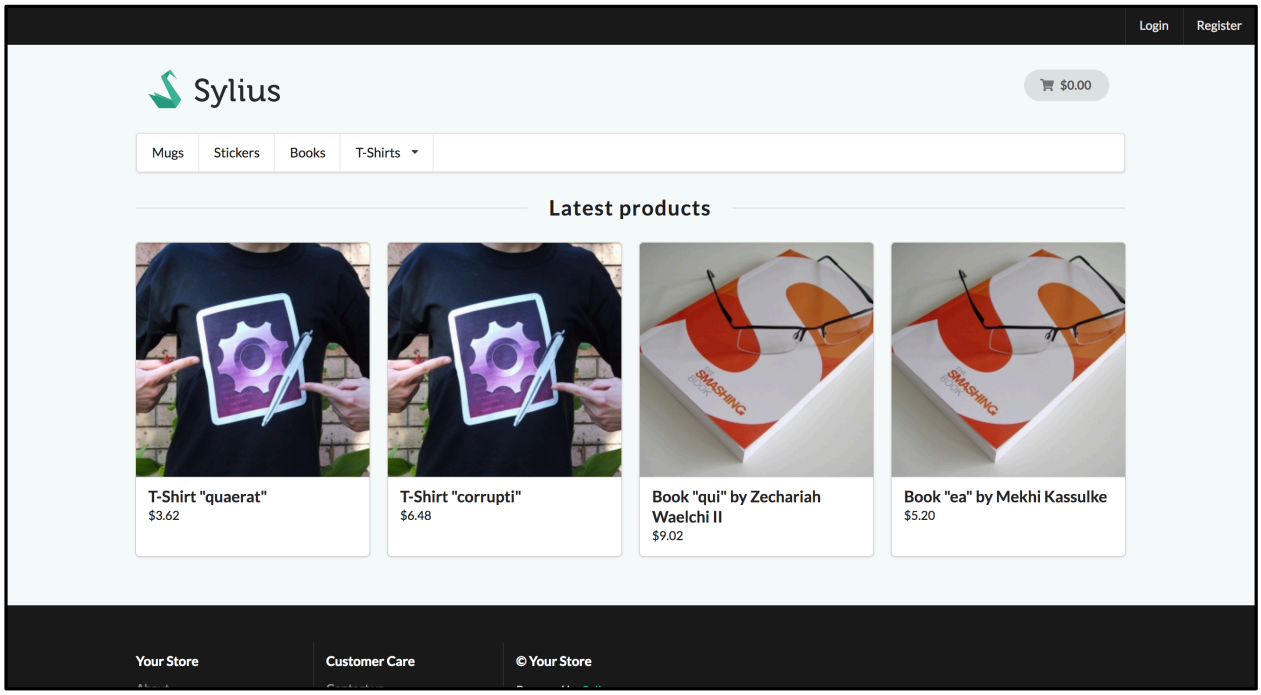
Blok kod 4.3. Generiranje Sylius frontenda

Nakon instalacije Sylius-ovog backenda i frontenda, potrebno je upaliti Symfony-ev ugrađeni web server naredbom *prema blok kodu 4.4.* Pristup Sylius-ovoj e-trgovini moguće je

otvoriti na linku: http://127.0.0.1:8000. Instalirani Sylius projekt s generiranim frontendom prikazan je na *slici 4.2.*

```
1.  php bin/console server:start --docroot=web 127.0.0.1:8000
```

Blok kod 4.4. Pokretanje Symfony-evog web servera



Sl 4.2. Generirani Sylius-ov standardni projekt

Nakon uspješnog instalacijskog procesa Sylius-a moguće je početi s razvojem unutar Sylius okvira. Unutar korijenskog direktorija projekta nalaze se sljedeći poddirektoriji:

1. *app/config* – direktorij u kojemu se nalazi .yaml konfiguracijske datoteke koje uključuju usmjeravanje, sigurnost, procesna stanja itd.
2. *var/logs* – direktorij unutar kojega se nalaze svi zapisi aktivnosti.
3. *var/cache* – direktorij unutar kojeg je definirana predmemorija projekta.sr
4. *src* – prilagođena razvojna logika definirana unutar App paketa
5. *web* – multimedijska sredstva korištena od strane Sylius-a (css, js, slike itd).

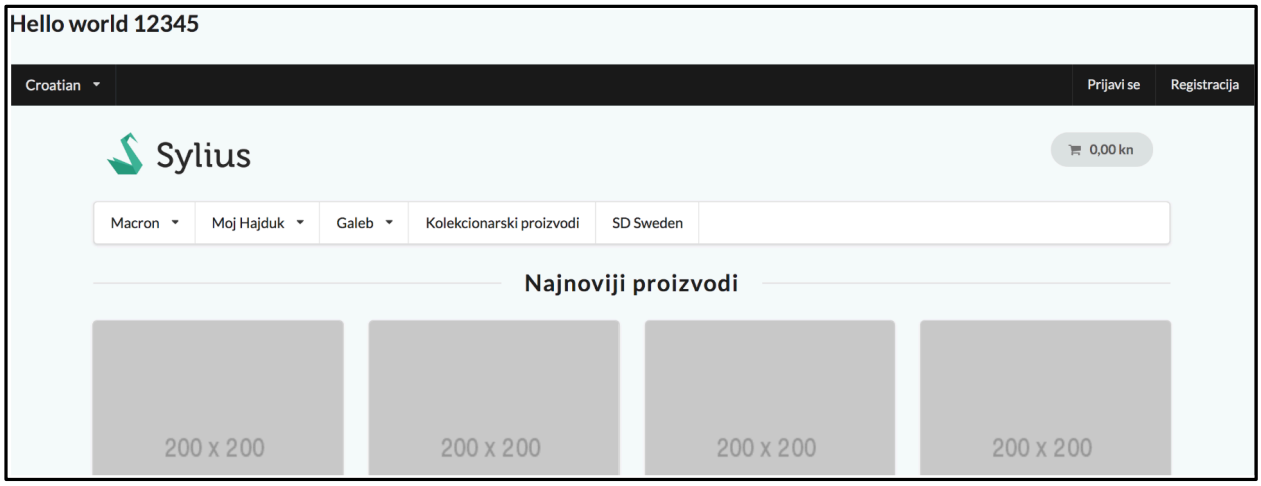
4.4. Kreiranje Sylius prilagođene teme

Sylius ima dvije vrste osnovnih tema. Prva je Sylius trgovina koja predstavlja frontend stranu i Sylius admin koji predstavlja predstavlja backend dio platforme. Moguće je prilagoditi standardnu Sylius temu ili napraviti novu temu za backend i frontend zasebno.

Postoje dva načina rada pri prilagođavanju teme. Prvi način prilagodbe teme predstavlja nadogradnju postojeće teme tako da se unutar sylius-ovog ”app/Resources“ definiraju određene .twig datoteke iz pojedinih dijelova Sylius-a koje je potrebno prilagoditi. Prilikom obrade, Sylius-ove postojeće datoteke su ignorirane ukoliko se unutar navedene putanje nalaze datoteke koje su izmijenjene. Unutar `./vendor/sylius/src/Sylius/Bundle` se nalaze svi Sylius-ovi paketi. Za nadogradnju vlastite teme potrebno je pronaći Shop paket koji sadrži sve elemente potrebne za prikaz frontend-a trgovine. Unutar paketa nalazi se direktorij *Resources/views* koji sadrži sve poglede prikaza na frontendu Sylius trgovine. Unutar *app/resources* potrebno je dodati novi direktorij s nazivom paketa uz pravilo da ispred imena paketa bude prefiks „Sylius”. Npr. ukoliko je naziv paketa „ShopBundle”, potrebno ga je nadograditi s nazivom „SyliusShopBundle”. Unutar novokreiranog paketa potrebno je kopirati sve ili određene poglede iz originalnog „ShopBundle” paketa kako bi nadogradili postojeći okvir. Prema blok kodu 4.5. unutar *layout.html.twig* modificiramo postojeći kod. Rezultat promjene prikazan je *na slici 4.3.*

```
1.  {% block top %}
2.      <h2> Hello world 12345</h2>  // Dodajemo h2 oznaku ispred menu-a
3.      <div id="menu" class="ui large sticky inverted stackable menu">
4.          {{ sonata_block_render_event('sylius.shop.layout.before_currency_switcher') }}
5.
6.          {{ render(controller('sylius.controller.shop.currency_switch:renderAction')) }}
7.          {{ render(controller('sylius.controller.shop.locale_switch:renderAction')) }}
8.
9.          {{ sonata_block_render_event('sylius.shop.layout.before_security_widget') }}
10.
11.         {{ render(controller('sylius.controller.shop.security_widget:renderAction')) }}
12.
13.         {{ sonata_block_render_event('sylius.shop.layout.after_security_widget') }}
14.     </div>
15. {% endblock %}
```

Blok kod 4.5. Modificirana layout.html.twig datoteka



Sl 4.3. Promjene na Sylius-ovoj početnoj stranici

Drugi način prilagodbe predstavlja kreiranje vlastite Sylius teme. Tema predstavlja složeniji oblik od nadogradnje iz prvog koraka. Vlastita tema pogodna je za korištenje u situacijama kada je potrebno izmijeniti ili kreirati veliki broj vlastitih stavki i funkcionalnosti koje ne obuhvaća Sylius-ova standardna tema. Prije same izrade nove teme potrebno je postaviti putanju gdje će se nalaziti novokreirana tema. Unutar *app/config/config.yml* potrebno je dodati stavke prema *blok kodu 4.6.*

```
1.  sylius_theme:
2.      sources:
3.          filesystem: ~
```

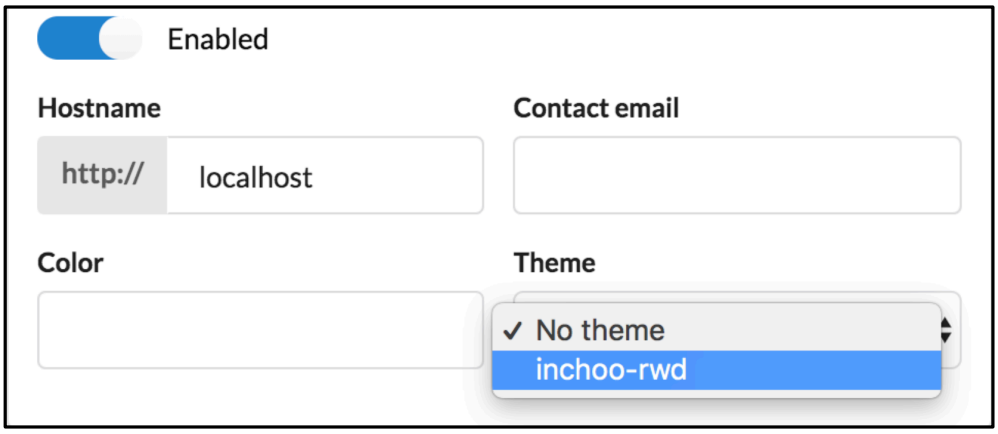
Blok kod 4.6. Konfiguriranje putanje nove Sylius teme

Nakon konfiguriranja, unutar *app* direktorija potrebno je kreirati novi direktorij naziva *themes* u kojemu će se nalaziti direktorij s nazivom teme po vlastitom izboru. Unutar direktorija vlastite teme potrebno je kreirati composer.json konfiguracijsku datoteku s podatcima navedenim prema *blok kodu 4.7.*

```
1.  {
2.    "name": "sylius/inchoo-rwd",
3.    "authors": [
4.      {
5.        "name": "Luka Omrčen",
6.        "email": "luka.omrcen18@gmail.com"
7.      }
8.    ],
9.    "extra": {
10.     "sylius-theme": {
11.       "title": "inchoo-rwd"
12.     }
13.   }}
```

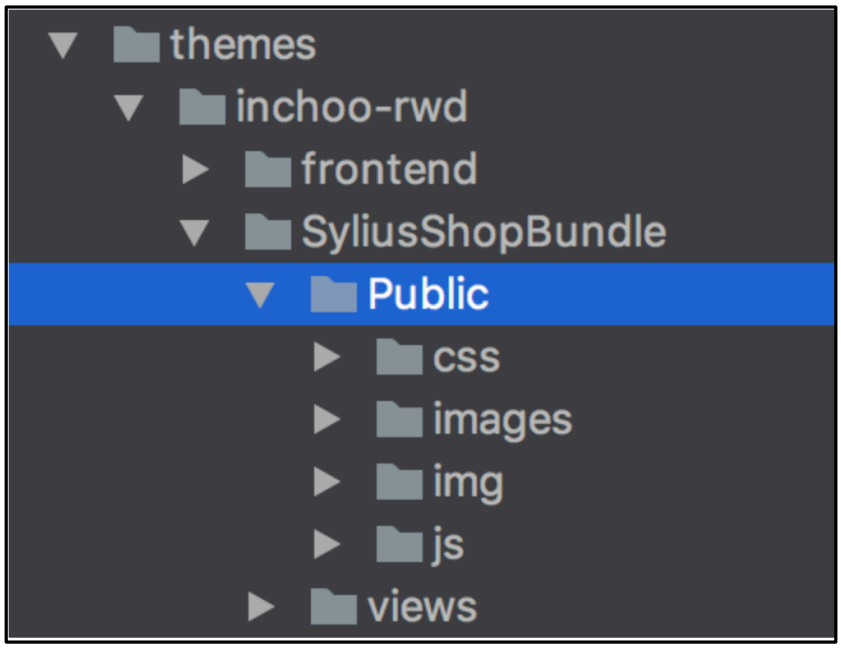
Blok kod 4.7. Konfiguriranje composer.json konfiguracijske datoteke

Name parametar predstavlja strukturu direktorija koji će se kreirati unutar web direktorija pod strukturom paketa, dok *sylius-theme* predstavlja naziv teme koja će se prikazati unutar Sylius-ovog admin sučelja pri konfiguraciji teme. Ukoliko su svi koraci valjano izvršeni, unutar admin sučelja pod kategorijom *channels* u opciji *themes* prikazana je novokreirana tema *(slika 4.4)*.



Sl 4.4. Novokreirana Sylius tema

Novokreirana tema funkcionira na isti način kao i *app/resource* direktorij koji služi kao nadogradnja. Unutar direktorija teme možemo dodavati nove ili izmjenjivati postojeće poglede iz različitih paketa. Osim paketa koji sadrže poglede za modifikaciju trenutne strukture stranice, moguće je i definirati *public* direktorij unutar *SyliusShop* paketa koji će dodati korisnikove CSS, JS datoteke kao i multimedijske sadržaje (*slika 4.5*).



Sl 4.5. Prikaz novokreirane Sylius teme na frontendu

Kako bi se korisnikova podatkovna struktura prikazivala na frontendu, potrebno je prema *bloku 4.8* koristiti naredbu za instaliranje podataka unutar Sylius-ovog *web* direktorija.

```
1.  $ php bin/console sylius:theme:assets:install
```

Blok kod 4.8. Instaliranje podatkovne strukture vlastite Sylius teme

Nakon instaliranja podatkovne strukture unutar Sylius *web* direktorija pojavljuje se direktorij s nazivom vlastite teme. Unutar strukture vlastite teme definiran je *SyliusShop* direktorij koji predstavlja poveznicu na *SyliusShopBundle* paket unutar novokreirane teme, te su kreirani svi direktoriji koji se nalaze unutar *public* direktorija vlastite teme. Unutar *SyliusShopBundle/views/layout.twig.html* direktorija teme definiramo poveznice na vlastite CSS i JS datoteke (*blok kod 4.9*). Kako je unutar admin sučelja označena *inchoo-rwd* tema sustav automatski zna koja je tema trenutno aktivna i koja je navedena putanja unutar direktorija *themes* gdje se nalaze korisnikovi definirani podatci.

```
1.  <head>
2.      <meta charset="utf-8">
3.      <meta http-equiv="X-UA-Compatible" content="IE=edge">
4.      <title>{% block title %}Sylius{% endblock %}</title>
5.      <meta content="width=device-width, initial-scale=1, maximum-scale=1, user-
    scalable=no" name="viewport">
6.
7.      {% block metatags %}
8.      {% endblock %}
9.
10.     {% block stylesheets %}
11.         <link rel="stylesheet" href="{{ asset('bundles/syliusshop/css/style.css') }}">

12.         <script src="{{ asset('bundles/syliusshop/js/vendor/modernizr-
    2.8.3.min.js') }}"></script>
13.         {{ sonata_block_render_event('sylius.shop.layout.stylesheets') }}
14.     {% endblock %}
15.
16.     {{ sonata_block_render_event('sylius.shop.layout.head') }}
17. </head>
```

Blok kod 4.9. Dodavanje poveznica na vlastite CSS i JS datoteke

Ukoliko nakon izmjene programskog koda se ne vide promjene, potrebno je obrisati predmemoriju sustava prema *blok kodu 4.10.* Konačan izgled teme koja je prilagođena programeru za daljnji proces razvoja prikazana je na *slici 4.6.*

```
1.  $ php bin/console cache:clear
```

Blok kod 4.10. Brisanje predmemorije Sylius-ovog sustava

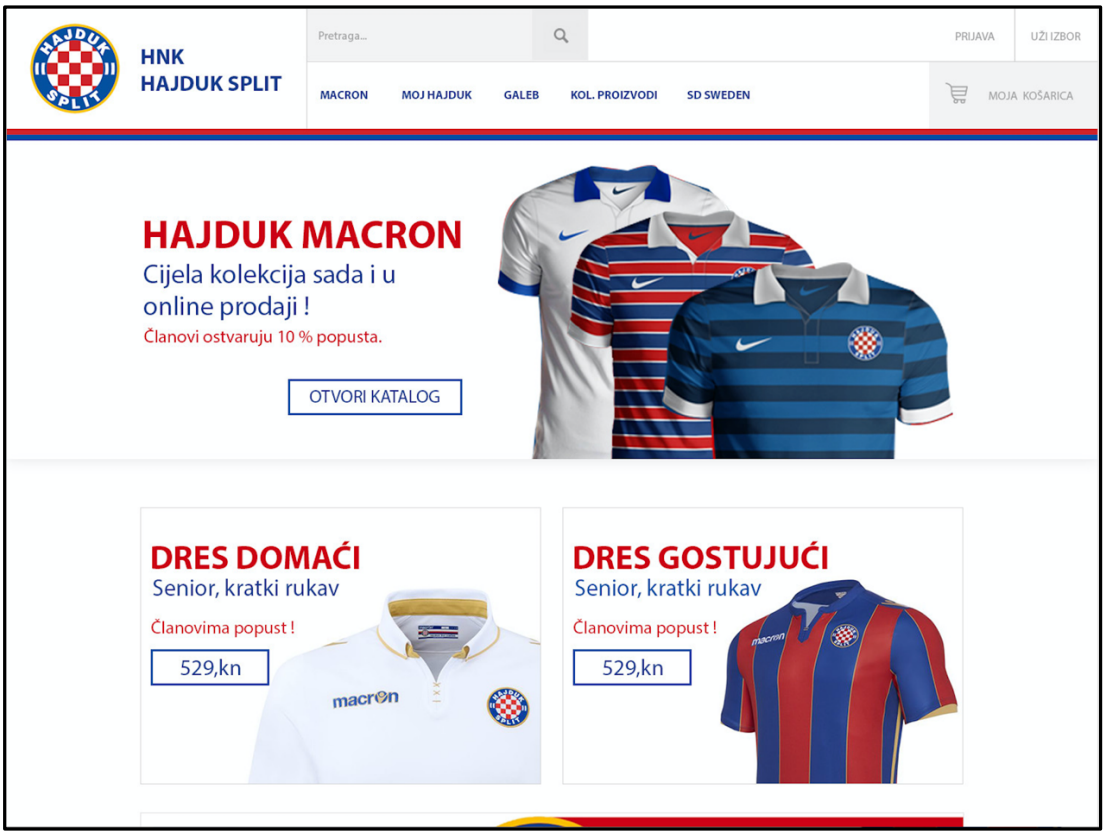
Sl 4.6. Prikaz teme za prilagođena rješenja

Sylius-ova trenutna literatura navodi da svaki put kada se naprave određene izmjene unutar *public* foldera teme, potrebno je svaki puta pozvati naredbu koja će taj sadržaj generirati u Sylius-ov *web* direktorij tijekom razvojnog perioda. S pogledom na frontend okolinu rada, to je vrlo nezgodno i troši puno vremena. Rješenje ovog problema temelji se na kreiranju posebnog direktorija unutar novokreirane teme s bilo kojim nazivom (u ovom slučaju naziv direktorija je *frontend – sl 4.5*). Unutar navedenog direktorija razvojni programer definira svoje razvojno okruženje (SASS, LESS, PostCSS, ES6 javascript itd). Nakon definiranja odgovarajućeg razvojnog rješenja, potrebno je unutar direktorija teme kreirati novi *gulpfile.js* kao i izlazno odredište svakog zadatka unutar Sylius *web* direktorija u kojemu se nalazi traženi paket. Primjer definiranja gulpfile.js zadatka da izvrši kompresiju i spoji sve Javascript datoteke definiran je u *blok kodu 4.11*. Nakon definiranja automatizacijskog zadatka, potrebno je instalirati sve dodatke koji su potrebni za automatizirani rad naredbom unutar terminala: *npm install*.

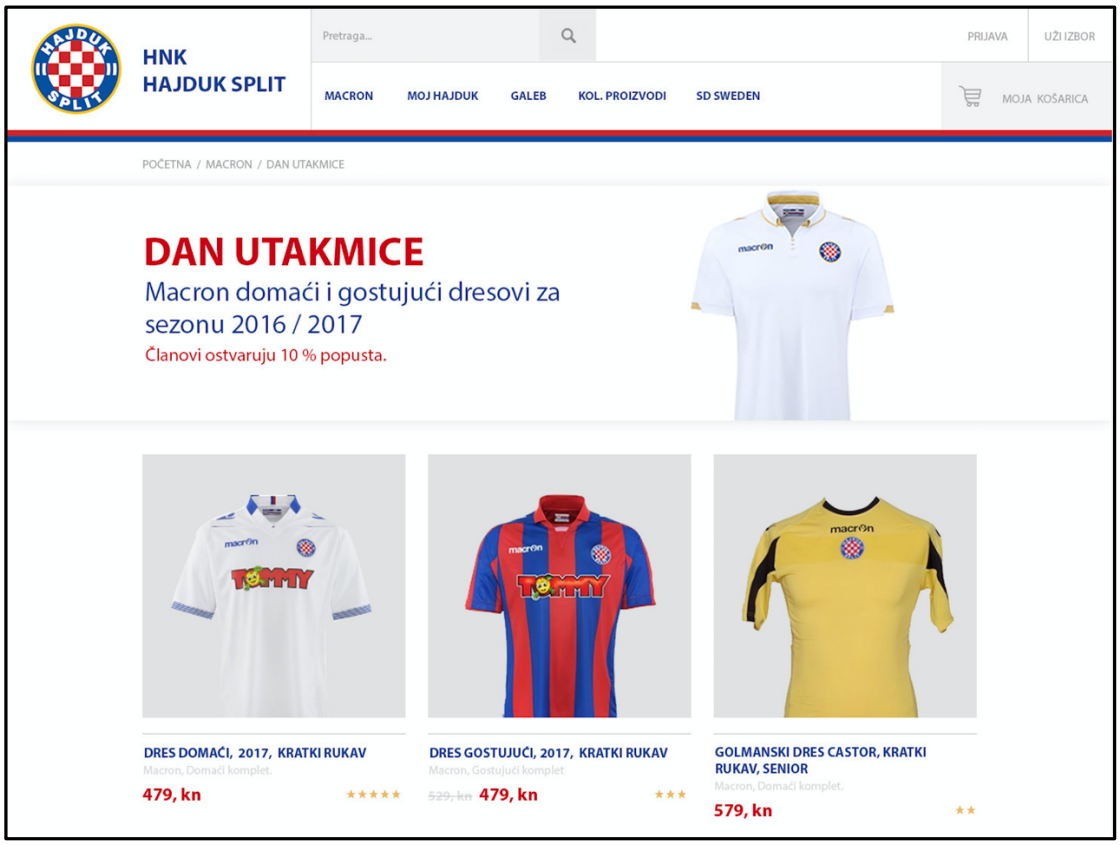
```
1.  gulp.task('uglify', function (cb) {
2.      gulp.src(['./frontend/js/vendor/*.js', './frontend/js/*.js'])
3.          .pipe(uglify())
4.          .pipe(concat('style.js'))
5.          .pipe(gulp.dest('./../../../web/bundles/_themes/sylius/inchoo-
    rwd/syliusshop/js/'))
6.  });
```

Blok kod 4.11. Definiranje automatizacijskog zadatka unutar gulpfile.js

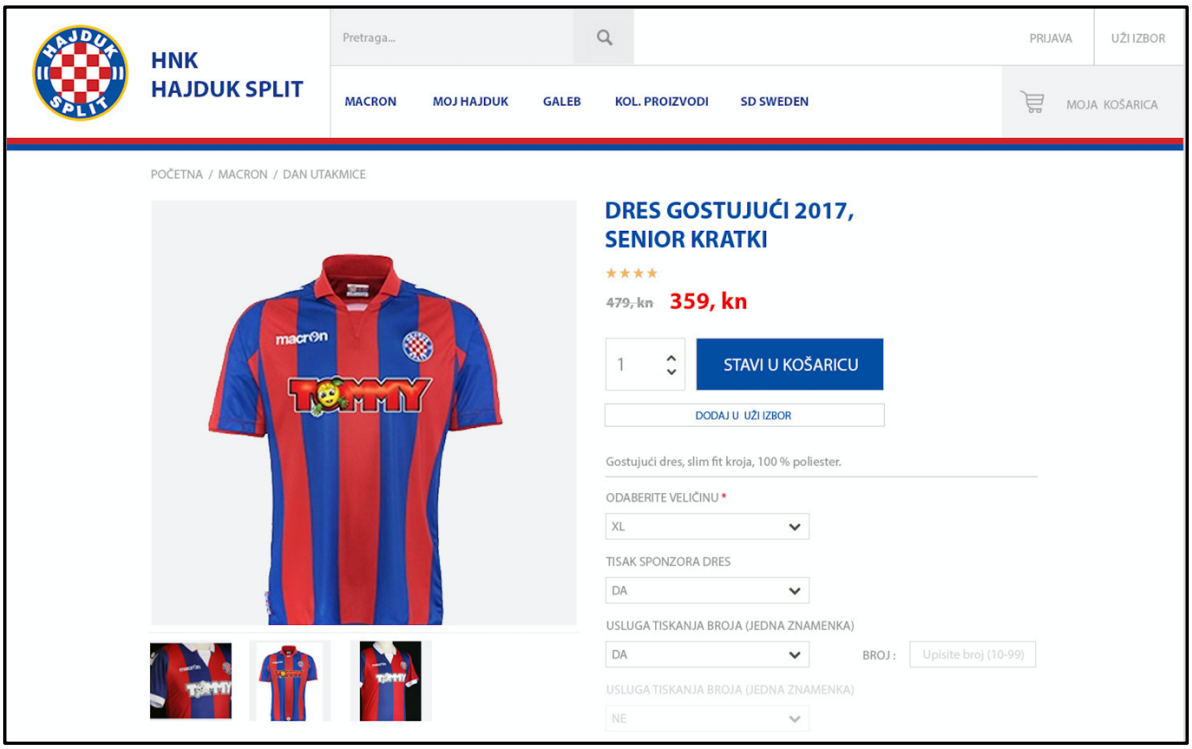
Primjer funkcionalno gotove Sylius teme prikazan je na slikama *4.7, 4.8 i 4.9.*



Sl 4.7. Prikaz vlastite Sylius teme (početna stranica)



Sl 4.8. Prikaz vlastite Sylius teme (katalog)



Sl 4.9. Prikaz vlastite Sylius teme (pojedini proizvod)

5. ZAKLJUČAK

Trend i razvoj web tehnologija neprestano se mijenja. Izgled modernih stranica i način na koji se informacije prezentiraju daleko su tehnički superiornije unazad deset godina. Kako se zahtjevi i značajke modernih web stranica sve više povećavaju pojavljuju se nove tehnologije i alati koje napreduju u skladu s tim zahtjevima. U suvremenom razvoju weba uvedeni su raznovrsni web elementi koji poboljšavaju izgled i općeniti dojam web stranica. Web dizajn predstavlja važan aspekt prezentacije koji se koristi za izražavanje i naglašavanje značenja određenih web sadržaja. Dobra organizacija i prezentiranje sadržaja predstavlja ključ uspjeha projektiranja suvremenih web stranica, dodajući sloj složenosti cjelokupnom procesu razvoja. Današnji razvojni programeri trebaju biti svjesni zahtjeva i mogućnosti korištenja novih tehnologija kao i mogućnosti njihove implementacije gdje je to potrebno. E-trgovina i dalje predstavlja jednu od poslovnih metoda iskorištava prednost nadolazeće tehnološke ere. Ukoliko prodajna moć padne, E-trgovina dalje ima mogućnost ostvarivanja visokih primanja i transakcija u odnosu na stare tehnike prodaje. E-trgovina je nedvojbeno postala važan član društva današnjice. Uspješne tvrtke počinju graditi poslovanje u tom smjeru dajući dovoljno sredstava za njegov razvoj. E-trgovina ne predstavlja generalno IT problem, već poslovnu priliku. E-trgovina predstavlja korisnu tehnologiju koja korisnicima i poslodavcima omogućuje lak pristup poslovanju i tvrtkama širom svijeta.

LITERATURA

[1] Google multiscreen world - prezentacija (2012) http://services.google.com/fh/files/misc/multiscreenworld_final.pdf (12.08.2018)

[2] Halvorson, K. (2012). "Content Strategy for the Web: Content Strategy Web"

[3] Knight, K. (2011). "Responsive Web Design: What It Is and How To Use It"

[4] Itzkovitch, A. (2012). "Creating an Adaptive System To Enhance UX".

[5] Cao, J. (2015). "Responsive vs. Adapative Design: What's the Best Choice for Designers?"

[6] SASS službena dokumentacija https://sass-lang.com/documentation/file.SASS_REFERENCE.html (14.08.2018)

[7] LESS službena dokumentacija http://lesscss.org/features/ (14.08.2018)

[8] PostCSS github stranica https://github.com/postcss/postcss (10.08.2018)

[9] Haverbeke, M. (2014) Eloquent JavaScript, 2nd Ed

[10] Selectors level 4 – službena dokumentacija https://www.w3.org/TR/selectors-4/ (18.08.2018)

[11] Larsson, T.(2017) "Ecommerce Evolved: The Essential Playbook To Build, Grow & Scale A Successful Ecommerce Business"

[12] Sylius službena dokumentacija https://docs.sylius.com/en/1.2/book/ (24.08.2018)

SADRŽAJ

Većina modernih web stranica današnjice ima složeni dizajn, privlačan i interaktivan sadržaj, koji uključuje mnoge značajke i usluge, uklapajući se pritom na svim trenutnim zaslonima i web preglednicima. Jedna od takvim usluga koja je ostvarila ubrzani rast i nije dosegla svoj vrhunac je i „E-trgovina". E-trgovina predstavlja jednu od poslovnih metoda koja iskorištava prednost nadolazeće tehnološke ere. Navedeni diplomski rad u prvom dijelu opisuje proces planiranja i definiranja sadržaja i dizajna web stranice ili aplikacije kako bi se prilagodio svim uređajima i preglednicima današnjice. Objašnjavaju se pojmovi „mobile-first" i „desktop-first" te razlika između njih. Opisuje se razlika između responzivnog i adaptivnog web dizajna, te općenita razlika između web dizajna i UI/UX dizajna. Uvode se koncepti i nove tehnologije u izradi kao što su predprocesori SASS, LESS i trenutno novi PostCSS, te ES6+ javascript i CSS4 koji predstavljaju tehnologije weba u budućnosti. U drugom dijelu diplomskog rada definira se pojam e-trgovina na primjeru Sylius platforme. Analizira se Sylius-ova arhitektura te način rada razvojnog programera (frontend) na navedenoj platformi. Definiraju se sve točke procesa tehnološke izrade počevši od kreiranja vlastite teme pa sve do realiziranog konkretnog rješenja.

Ključne riječi: adaptivni dizajn, CSS4, „Desktop-first", e-trgovina, ES6 Javascript, LESS, „Mobile-first", platforma za e-trgovinu, PostCSS, responzivni dizajn, SASS, Sylius, UI dizajn, UX dizajn

ABSTRACT

Modern web development concepts in e-commerce on Sylius platform

Most modern web sites today have complex design, appealing and interactive content, which includes many features and services, adaptable to all current screens and web browsers. "E-commerce" is the one of those services that has rapid growth and has not reached its culmination. E-commerce is one of the business methods that takes advantage of the upcoming technological era. This graduate paper describes the process of planning and defining the content and design of a website or application to adapt to all devices and browsers of today. The terms "mobile-first" and "desktop-first" are explained and the difference between them. Main difference between terms responsive and adaptive web design, and the general difference between web design and UI / UX design. Explanation of concepts and new technologies that are being developed with the predprocessors like SASS, LESS and the current new PostCSS, ES6 + javascript and CSS4, which represent the technology of the web in the future. In the second part of this graduate paper, the term e-commerce is defined with the practical example of Sylius e-commerce platform. Analysis of Sylius architecture and workflow of web developer is being described. All points process of technological development is defined starting from creating own unique theme until final project realization.

Keywords: Adaptive Design, CSS4, "Desktop-first", E-commerce, E-commerce platform, ES6 Javascript, LESS, "Mobile-first", PostCSS, Responsive Design, SASS, Sylius, UI Design, UX Design

ŽIVOTOPIS

Luka Omrčen rođen je 20. rujna 1994. godine u Osijeku. Pohađao je osnovnu školu Josipovac u Josipovcu. Nakon završenog osnovnoškolskog obrazovanja, upisao je Elektrotehničku i prometnu školu Osijek, smjer tehničar za računarstvo gdje je maturirao 2013 godine. Iste godine upisuje Fakultet elektrotehnike, računarstva i informacijskih tehnologija u Osijeku. Nakon završenog preddiplomskog studija 2016 godine, upisuje diplomski studij računarstva, smjer programsko inženjerstvo. Stručnu praksu odradio je u osječkoj tvrtki Inchoo u kojoj je zatim nastavio istraživanje Sylius platforme u trajanju od pet mjeseci. Trenutno je zaposlen u tvrtki Inchoo.

Vlastoručni potpis:

_____

Luka Omrčen