Atomski dizajn web aplikacija i njegova implementacija pomoću Patternlab-a, Vue.js-a i Firebase-a

Gagro, Josip

Master's thesis / Diplomski rad

2021

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Zagreb, Faculty of Organization and Informatics / Sveučilište u Zagrebu, Fakultet organizacije i informatike**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:211:529735

Rights / Prava: Attribution-NoDerivs 3.0 Unported/Imenovanje-Bez prerada 3.0

Download date / Datum preuzimanja: **2025-03-15**

SVEUČILIŠTE U ZAGREBU
FAKULTET ORGANIZACIJE I INFORMATIKE
VARAŽDIN

Repository / Repozitorij:

Faculty of Organization and Informatics - Digital Repository

zir.nsk.hr

dabar
DIGITALNI AKADEMSKI ARHIVI I REPOZITORIJI

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Josip Gagro

Atomski dizajn web aplikacija i njegova implementacija pomoću Patternlab-a, Vue.js-a i Firebase-a

DIPLOMSKI RAD

Varaždin, 2021.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Josip Gagro

Matični broj: 46417/17–R

Studij: Informacijsko i programsko inženjerstvo

Atomski dizajn web aplikacija i njegova implementacija pomoću Patternlab-a, Vue.js-a i Firebase-a

DIPLOMSKI RAD

Mentor/Mentorica:

Prof. dr. sc. Dragutin kermek

Varaždin, prosinac 2021.

Josip Gagro

Izjava o izvornosti

Izjavljujem da je moj završni/diplomski rad izvorni rezultat mojeg rada te da se u izradi istoga nisam koristio drugim izvorima osim onima koji su u njemu navedeni. Za izradu rada su korištene etički prikladne i prihvatljive metode i tehnike rada.

Autor/Autorica potvrdio/potvrdila prihvaćanjem odredbi u sustavu FOI-radovi

_____

Sažetak

Atomski sustav dizajna je pristup implementacije web aplikacije koji utječe na sve članove tima za razvoj aplikacije. U radu se opisuje kako bi članovi tima trebali surađivati ukoliko koriste atomski sustav dizajna, gdje se najviše pažnje posvećuje na programere na korisničkoj strani razvoja web aplikacije, način na koji se implementira atomski sustav dizajna i tehnologije koje je moguće koristiti za njegovu implementaciju. Nakon teorijskog opisivanja atomskog dizajna sustava opisan je praktični primjer implementacije web aplikacije. Kroz praktični primjer se objašnjava kako koristiti aplikaciju te primjeri atomskog sustava dizajna unutar aplikacije. U zaključku je obuhvaćen sveukupan prikaz rada, kratki uvid prethodno napisanog, razlozi zašto koristiti atomski sustav dizajna, njegove prednosti i mane, te preporuke za korištenje.

Ključne riječi: Web aplikacije; Razvoj web aplikacija; Sustav dizajna; Atomski sustav dizajna; Razvoj na korisničkoj strani; HTML; Javascript; CSS; Sass; Vue.js; Patternlab; Web aplikacija; Firebase; Firestore; Firebase Auth;

Sadržaj

1. Uvod

Tehnologija se iz dana u dan mijenja. Svaki dan se može čuti kako je ostvareno neko novo tehnološko postignuće koje doprinosi i olakšava život čovjeku bilo u privatnom ili poslovnom svijetu. Svako novo tehnološko postignuće potiče daljnje istraživanje i poboljšavanje već istraženog i naučenog kako bi se postojeća tehnologija usavršila ili kreirala nova tehnologija. S poboljšavanjem tehnologije povećava se njena kompleksnost, a samim tim i kompliciranost njene izrade. Sa kompleksnošću se stvaraju problemi s kojima se ljudi moraju suočiti kako bi ostvarili zadani cilj. Stoga se ljudi okreću sustavnom razvoju proizvoda kako bi izbjegli neželjene probleme prilikom razvoja i imali jasni uvid kako i na koji način razvijati željeni proizvod.

Današnji svijet se ne može zamisliti bez Interneta. Taj neiscrpni izvor informacija je jako bitan za poslovno i privatno funkcioniranje čovjeka, te današnji svijet jednostavno ne možemo zamisliti bez njega. Iako relativno mlada tehnologija s njenim se razvojem pojavio problem razvoja web aplikacija. Zahtjevi za web aplikacije su svakim danom sve veći i veći, što utječe na njihovu kompleksnost. S većom kompleksnošću potrebno je više stručnih ljudi kako bi razvilo potreban proizvod, a kako bi proizvod bio što brže i preciznije završen i isporučen korisniku potrebno je razviti neki sustav rada u kojemu će to biti moguće.

U nastavku rada će biti opisan jedan takav sustav gdje će se opisati njegove najbitnije značajke, karakteristike, prednosti i mane, gdje i zašto ga koristiti. Sustav se naziva Atomski sustav dizajna (*eng. Atomic Design System)* koji je uglavnom orijentiran na prezentacijski dio web aplikacija i način na koji bi se trebalo pristupati razvoju zaslona stranice ili web aplikacije. Atomski sustav dizajna opisuje kako bi trebala biti suradnja unutar tima, te način na koji bi se korisnički zaslon trebao razvijati. Rad će biti najviše orijentiran na razvoj na korisničkoj strani (*eng. front-end development*).

Sav teorijski dio u radu će biti potkrijepljen primjerom web aplikacije koji je razvijan po navedenom sustavu, gdje će se opisati aplikacija, navesti i opisati tehnologije korištene za implementaciju.

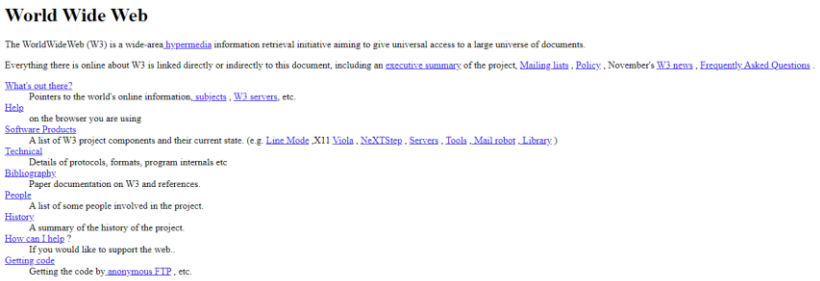
2. Web dizajn kroz povijest

U uvodu je navedeno kako ne možemo zamisliti današnji svijet bez Interneta i Weba. Za Web se može reći kako je ono jedno od najvećih revolucionarnih tehnoloških otkrića ikada u ljudskoj povijesti. Nikada čovjek nije bio toliko povezan kao sada, u vremenu kada se informacije mogu dobiti za nekoliko sekundi.

D.Kermek definira Web kao - „Kolekcija međusobno povezanih multimedijskih dokumenata koji su pohranjeni na Internetu (na jednom ili više poslužitelja) i kojima se pristupa koristeći zajednički protokol (HTTP).” [1]

Ocem Weba smatra se Tim Berns-Lee, britanski informatički znanstvenik koji je radio u švicarskom CERN-u, mjestu gdje je web i nastao. Berns-Lee je napisao dokument pod nazivom „*Information management: A proposal”* [2] u kojem se opisuje način prijenosa informacija preko Interneta korištenjem hiperteksta. U sklopu tog dokumenta dizajniran je jezik za definiranje sadržaja dokumenta HTML (eng. *HyperText Markup Language*), protokol za preuzimanje dokumenata i za interpretaciju sadržaja (HTTP) i implementiran je prvi preglednik. 1991. godine Web postaje dostupan izvan CERN-a.

U početku su web stranice bile isključivo tekstualno bazirane, odnosno onako kako zamišljamo dokumente da izgledaju, sa prvenstveno crno-bijelim tekstom. Osim crno-bijelog teksta moguće je bilo vidjeti i pojavu teksta plave boje sa crtom ispod, koja označava poveznicu na drugi dokument.



World Wide Web

The WorldWideWeb (W3) is a wide-area hypermedia information retrieval initiative aiming to give universal access to a large universe of documents.

Everything there is online about W3 is linked directly or indirectly to this document, including an executive summary of the project, Mailing lists , Policy , November's W3 news , Frequently Asked Questions .

What's out there?
Pointers to the world's online information, subjects , W3 servers, etc.
Help
on the browser you are using
Software Products
A list of W3 project components and their current state. (e.g. Line Mode ,X11 Viola , NeXTStep , Servers , Tools , Mail robot , Library)
Technical
Details of protocols, formats, program internals etc
Bibliography
Paper documentation on W3 and references.
People
A list of some people involved in the project.
History
A summary of the history of the project.
How can I help ?
If you would like to support the web..
Getting code
Getting the code by anonymous FTP , etc.

Slika 1: Prva stranica koju je napravio Tim Berns-Lee

(dostupno na http://info.cern.ch/hypertext/WWW/TheProject.html, 27.03.2020.)

Sa napretkom tehnologije, odnosno preglednika (*eng. browser*) i mogućnosti umetanja slika u stranice promijenio se i način na koji su se prezentirale informacije korisniku. Informacije su se strukturirale unutar tablica, te je programer na korisničkoj strani morao dobro razmisliti kako će organizirati strukturu dokumenta prilikom implementacije stranice. Tad se razvija popularni izraz „rezanje dizajna" (*eng. slicing design).* [3] Takva implementacija stranica je imala mnogo mana, pogotovo prilikom njihovog održavanja. Naravno, takav način implementacije omogućio je koliko-toliko pozicioniranje elemenata na stranici, odnosno glavna prednost takve implementacije je ta što je to bio najbliži način implementaciji tzv. mreži (*eng. grid)*, koja se danas koristi. Prilikom dizajniranja, ali i implementacije dizajna potrebno je bilo paziti da se ne pretjera sa slikama, jer u to vrijeme brzina interneta nije bila ni približno dobra kao danas, ukoliko je bilo previše slika na stranici ona bi se jako dugo učitavala.

Pojava Java appleta revolucionirao je kreiranje web stranica, jer je bilo moguće dodavati dinamički sadržaj unutar statičnog HTML svijeta. Appleti su male aplikacije, napisane u jeziku Java, koje se uključuju u HTML, odnosno stranicu i pokreću se unutar preglednika. Java appleti su u vrijeme svog nastanka omogućavale prve oblike interaktivnosti između korisnika i stranice, koju u to vrijeme HTML nije mogao omogućiti.

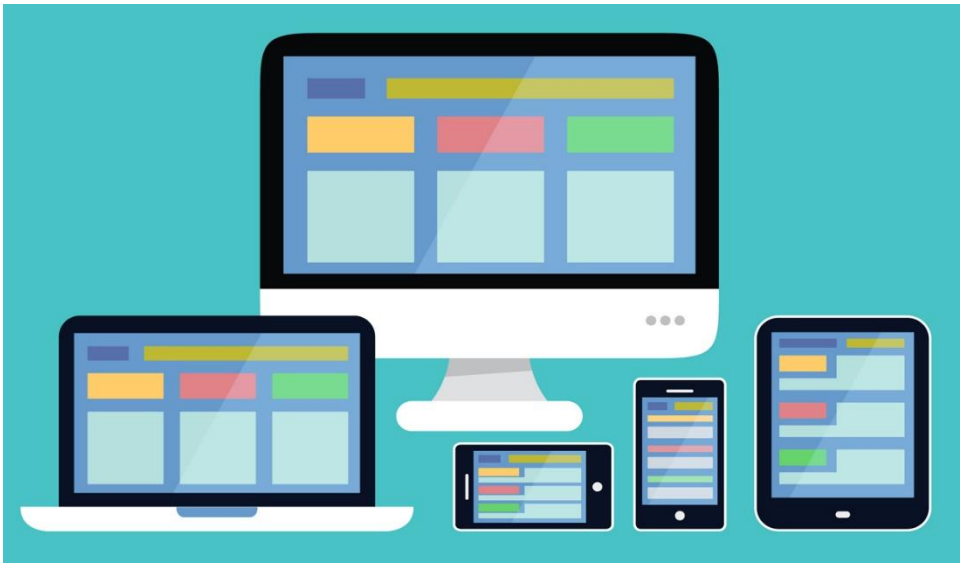
Nastankom Javascript-a i DOM-a (eng. *Document Object Model*), jezika namijenjenog za preglednike, odnosno za korisničku stranu web stranica/aplikacija je također utjecao na način na koji se radilo web stranice/aplikacije. Omogućeno je dinamičko izmjenjivanje sadržaja na stranici, izbjegavanja statičnosti HTML-a, dodavanja mogućnosti kretanja elementima, implementiranje iskakajućih prozora (eng. *pop-up*) i sl. Mana Javascript-a u početku je bila ta da je morao biti učitan na vrhu postojeće stranice što je također stvaralo poteškoće prilikom učitavanja stranice.

Sljedeća stvar koja je promijenila web i njegovo dizajniranje je Flash. Po prvi puta dizajneri su mogli kreirati elemente oblika kakve su htjeli, sa ili bez animacija. Sve to je učinilo stranice zanimljivijima i pristupačnijima za korištenje. Flash se koristio za animacije, za „uvode" u stranicu, interakcije i sl. Mana Flash-a je ta što je korisnik morao imati potrebnu verziju pregledničkog dodatka (*eng. plugin*) kako bi mogao vidjeti/koristiti Flash. Osim toga, mana Flash-a je i ta što je bilo potrebno dosta vremena da se učita i to što je koristio mnogo procesorske snage kako bi se izvodio na stranici.

Još jedna velika stvar koje se dogodila za web je pojava CSS-a. Razlog zbog kojeg je CSS nastao je bila želja da se odvoji prezentacijski dio web stranice od njenog sadržaja, što znači sav stil i pozicioniranje bi bilo smješteno u CSS datoteci, dok bi sadržaj bio smješten u datoteci HTML-a. Ovakav pristup rada je omogućio lakši rad i održavanje web stranica, ali i

mogućnost lakše zamijene stila stranice, gdje bi se samo zamijenila uključena CSS datoteka i stranica bi poprimila potpuno drugačiji izgled. Problem CSS-a u njegovom početku je bio u tome što se različito ponašao na različitim preglednicima, neki preglednici su imali određene funkcionalnosti dok drugi nisu i to je uvelike otežavalo programerima rad na stranicama. Potrebno je bilo nekoliko godina da se situacija sa podržavanjem preglednika koliko-toliko ujednači, iako i danas je situacija takva da se posebna pažnja mora posvećivati kako se određeni CSS atributi ponašaju na određenim preglednicima.

2007. godine i predstavljanjem prvog iPhone-a i okretanju pametnim telefonima krenulo je i novo razdoblja razvoja web stranica i njihovog dizajna. S pojavom pametnih telefona pojavili su se i novi izazovi koje su dizajneri i programeri morali uspješno savladati. Problem koji se javio bio je potreba prilagođavanja sadržaja i izgleda stranice na verzijama za stolna računala i za pametne telefone. Pitanja koja su morala biti riješena su da li neki sadržaj se prikazuje na svim vrstama ekrana, kako prilagoditi izgled da se stranica potpuno ne izlomi, veličina sadržaja i slika kako stranice korisniku ne bi „pojela" previše mobilnih podataka. S tim pitanjima se poboljšava i implementacija mreže (*eng. grid*) na web stranicama, počinju se standardizirati elementi stranice koji se često koriste tako da budu što fleksibilniji za korištenje što dovodi do pojave prvih programskih okvira (*eng. framework*) za rad na prezentacijskom dijelu stranice, najpoznatiji primjer su Bootstrap i Foundation. S pojavom ovog razdoblja se i stvorio termin prilagodljivi web dizajn (*eng. responsive web design*) koji je označavao mogućnost web stranice da poprima različitu prezentaciju na različitim uređajima, odnosno ekranima.



Slika 2: Prilagodljivi web dizajn

(preuzeto sa https://www.hostpapa.ca/blog/marketing/responsive-design-7-design-mistakes-can-cost-leads, 28.03.2020.)

Današnji web dizajn se odmaknuo od blještavih gumbova, sjena, šarenih formi i slično. Može se reći da se vratio počecima tj. minimalističkom pristupu sa modernim mogućnostima. Velika se pažnja posvećuje sadržaju prvenstveno, kako je on konstruiran, kako je pozicioniran na stranici, u kojem redoslijedu, hijerarhiji i slično. Pažnja se posvećuje tipografiji, slikama, ikonama. Napretkom tehnologije, odnosno preglednika i CSS jezika omogućena je veća fleksibilnost rada na stranicama. Pojava fleksibilne kutije (eng. *flexbox*) je omogućila lakši rad na pozicioniranju elemenata, te je uklonila ovisnost o programskih okvira da bi programeri lakše pozicionirali elemente. Također, moderni dizajn i razvoj web aplikacija sad je okrenut na razvoj web komponenti koje se mogu koristiti na više dijelova stranice što olakšava održavanje ali i sveukupni rad. Jedan od načina rada s web komponentama će biti opisan u nastavku.

3. Atomski sustav dizajna

3.1. Sustav dizajna

Kroz zadnja dva desetljeća može se reći kako se sve više okreće modularnom razvijanju aplikacija. Razlog tomu je taj što modularnost omogućava puno veću fleksibilnost prilikom implementiranja aplikacije. To znači da je puno lakše pronaći probleme unutar aplikacije, moguće ju je lakše održavati, proširiti, nadograditi itd. Kako se tom trendu svijet programiranja okreće, s tim trendom je zahvaćen i razvoj korisničkih sučelja. Korist modularnosti na korisničkoj strani razvoja može se vidjeti kroz razbijanje web stranica/aplikacija na male dijelove, koji se mogu ponovno koristiti na drugim mjestima. Osim navedenih razloga tu je i pojava sve većeg broja različitih uređaja sa različitom veličinom ekrana kojima je potrebno prilagoditi izgled proizvoda. Stranica se gradi tako što se najmanji dijelove stranice spajaju u veće što na kraju rezultira željenim sučeljem. Zbog velikog broja malih komponenti, velikog broja ljudi koji radi na razvoju na korisničkoj strani, od dizajnera, programera na korisničkoj strani (*eng. frontend developer*), testera, menadžera i sl. javlja se potreba za nekim sustavnim razvojem, kojim bi se smanjilo što više grešaka prilikom razvoja, rupa u komunikaciji i kako bi se proizvod, odnosno aplikacija što brže razvila i isporučila.

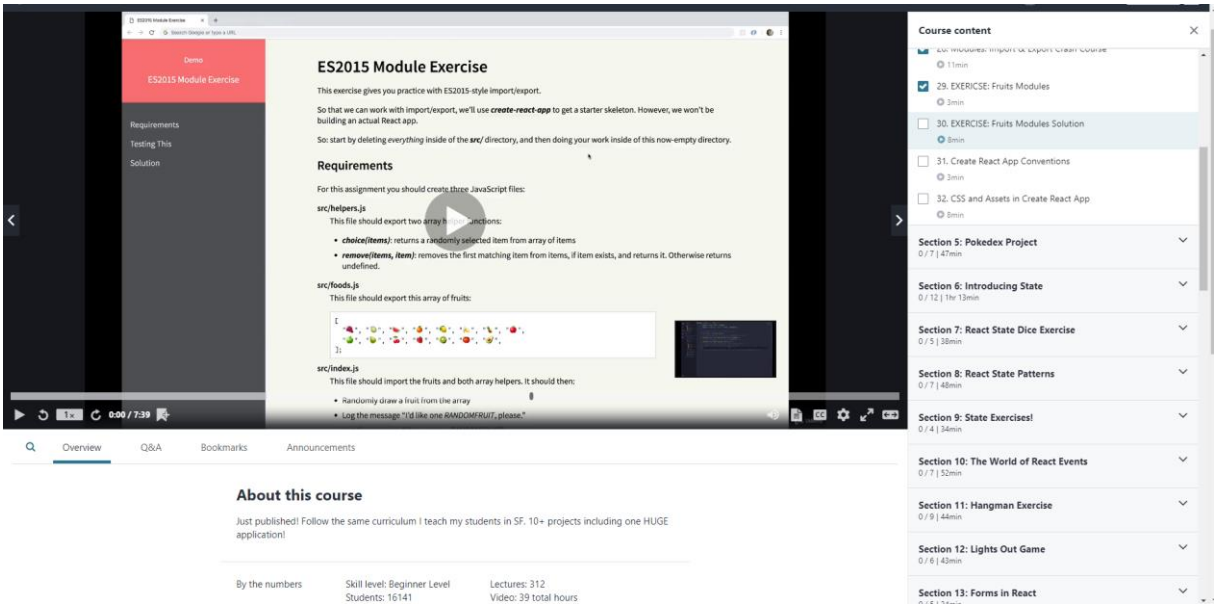
Iako sustav dizajna nema službenu definiciju, te ga svatko definira na svoj način, A. Kholmatova ga definira kao „skup međusobno povezanih uzoraka i dijeljenih vještina koje su organizirane koherentno da bi ostvarile svrhu digitalnog proizvoda.“ [4] Uzorci u ovom slučaju su ponavljajući elementi kojima se kreiraju korisnički zasloni, dok vještine odgovaraju na pitanje kako će se nešto napraviti.

A. Konaté navodi kako sustav dizajna pruža set komponenti koje se mogu koristiti kroz projekt i smjernice za donošenje odluka koji su uzorci, odnosno komponente relevantne u određenoj situaciji. [5] S dizajnom sustava dizajner se može više fokusirati na korisnika i ono što korisnik treba, umjesto na sami dizajn.

Za bilo koji digitalni proizvod uzorci međusobno rade skupa kako bi ostvarili svrhu digitalnog proizvoda. No svaki proizvod ne koristi iste uzorke kako bi ostvarilo svoju svrhu. Potrebne uzorke oblikuje proizvod i njegova svrha, tako će aplikacije koje imaju sličniju svrhu vjerojatno koristiti i slične uzorke. To se najbolje može vidjeti na sljedećem primjeru.

Za primjer je uzeta stranica za edukaciju i stranica za kupoprodaju kripto valuta. Svrha stranice za edukaciju je što jasnije i preciznije pokazati korisniku potrebne informacije kako bi što bolje shvatio i usvojio gradivo koje želi naučiti. Tako će stranice za edukacije koristiti više video elemente, strukturirane artikle, slike kojima se objašnjava neko gradivo, možda recenzije pojedine lekcije, pitanja na forumu i slično. Koristit će se boje s jakim kontrastom kako bi

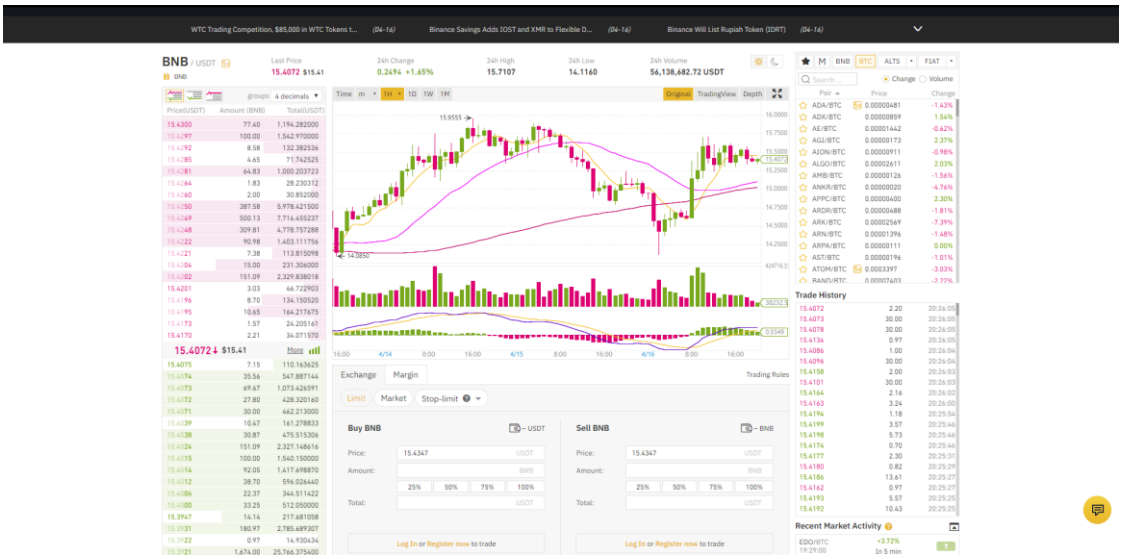
korisnik jasno mogao vidjeti sadržaj. Osim toga sadržaj će se na stranici pozicionirati na način na koji neće odvlačiti pažnju korisniku.



Slika 3: Slika edukacijske stranice Udemy.com

(preuzeto sa https://www.udemy.com/course/modern-react-bootcamp/learn/lecture/14365828?start=0#overview, 01.04.2020.)

U odnosu na edukacijsku stranicu, stranica za kupoprodaju kripto valuta se potpuno razlikuje. Svrha stranice za kupoprodaju kripto valuta je pokazati korisnicima što više relevantnih informacija za kupnju ili prodaju kripto valute. Takve stranice su strukturirane tako da se na njima može obavljati više stvari odjednom, sa zbijenim sadržajem kako bi što više stvari stalo na jednome mjestu. Od komponenti na takvim stranicama najčešće se mogu naći tablice sa podacima, razni statistički grafovi i forme za unos podataka.



Slika 4: Stranica za kupoprodaju kripto valuta

(preuzeto sa https://www.binance.com/en/markets, 01.04.2020.)

B. Frost [5] navodi kako složenost web mjesta nije u broju stranica koje se mora kreirati, nego kompliciranost web mjesta se mjeri u broju komponenti koji se nalazi na stranicama. No sam broj komponenti nije dovoljno mjerilo za odrediti složenost web mjesta. Složenost tih komponenti i njihov broj na web mjestu određuje njegovu složenost. Za primjer se mogu uzeti početne stranice. Jedna početna stranica može sadržavati odjeljak, sliku i video. Druga početna stranica se sastoji od padajućeg izbornika, klizača (*eng. slider*) i odjeljka. Jasno je kako druga stranica je puno kompliciranija za napraviti od prve iako se sastoje od jednakog broja komponenti. Također, može se postaviti i pitanje da li je web mjesto nekog fakulteta sa tisućama web stranica kompliciranije od nekog drugog web mjesta sa par stranica? Tih tisuće stranica može koristiti par komponenti i relativno je jednostavno složiti te stranice, dok npr. stranica sa mnogo komponenti može biti puno složenija za napraviti. Stoga potrebno je biti oprezan prilikom definiranja složenosti nekog web mjesta, te imati na umu da se ne dizajniraju stranice nego sustav komponenti.

Kod sustavnog pristupa rada s dizajnom potrebno je raditi iterativnim pristupom. Razlog tomu je ukoliko se događaju velike promjene na digitalnom proizvodu korisnici proizvoda mogu postati jako nezadovoljni i zbunjeni sa korištenjem novog sučelja. Stoga se proizvodi nadograđuju iterativno kako korisnik ne bi doživio šok prilikom korištenja novog sučelja, te kako ne bi gubio puno vremena učeći koristiti novo sučelje. Zbog toga se prvo izbaci proizvod sa osnovnim funkcionalnostima koji se iterativno nadograđuje. Iterativni pristup omogućuje lakše i brže otklanjanje potencijalnih problema, praćenju trendova i lakše se dobiva povratna informacija od korisnika.

3.1.1. Funkcionalni uzorci

Kholmatova definira funkcionalne uzorke kao opipljive blokove za izgradnju korisničkih sučelja. [4] Njihova svrha je omogućiti korisniku određeno ponašanje na web mjestu. Funkcionalni uzorci su oblikovani od domene web mjesta za koji su namijenjeni. To se može vidjeti na prethodnom primjeru gdje su se uspoređivale stranica za edukaciju i stranica za kupoprodaju kripto valuta. Funkcionalne komponente za edukacijsku stranicu bi bile video, slike, recenzije i sl., dok bi za kupoprodajnu stranicu to bilo grafovi, dijagrami, forme, tablice i slično.

Funkcionalni uzorci mogu biti jednostavne naravi koji se mogu kombinirati i spajati kako bi kreirali složenije uzorke. Za primjer se može uzeti heroj komponenta koja se sastoji od naslova, odjeljka i slike.

Slika 5: Primjer heroj komponente

(preuzeto sa https://helpcenter.woodwing.com/hc/en-us/articles/205114225-The-components-of-an-Inception-Story, 18.04.2020.)

Funkcionalni uzorci evoluiraju sa produktom za koji su namijenjeni. Uzorak koji je planiran na određeni način u početnoj fazi projekta može se dosta razlikovati u završnoj fazi projekta. Razlog tomu je evolucija projekta, stalno testiranje i poboljšavanje stvari. Planiranje uzoraka u početnoj fazi je jako korisno, jer može spriječiti njihovo dupliciranje. Prilikom dizajniranja komponente potrebno je znati njenu svrhu i njen sadržaj. Sadržaj i svrha kreiraju strukturu komponente. Na osnovu toga dizajneri i programeri znaju što trebaju napraviti. Kao primjer evolucije se može uzeti heroj komponenta. U nekoj fazi projekta klijent odluči da želi gumb sa nekim pozivom na akciju unutar heroj komponente ili želi uključiti klizač (*eng. slider*) kako bi mogao pokazati više informacija korisniku. Samim zahtjevima klijenta i evolucijom projekta vidi se i evolucija komponenti, u ovom primjeru heroj komponente. Ono što treba naglasiti je da se evoluiranjem uzorka ne mijenja njegova svrha postojanja. Ukoliko se doda klizač na heroj

komponentu, komponenta i dalje ima svoju svrhu, a to je da pokaže osnovne informacije kad korisnik dođe na početnu stranicu.

3.1.2. Percepcijski uzorci

Kada bi svaki proizvod izgledao isto iskustvo prilikom korištenja digitalnih proizvoda ne bi bilo jednako ugodno kao sada. Proizvodi bi jednostavno bili monotoni. Percepcijski uzorci su ono što odvaja jedan digitalni proizvod od drugog.

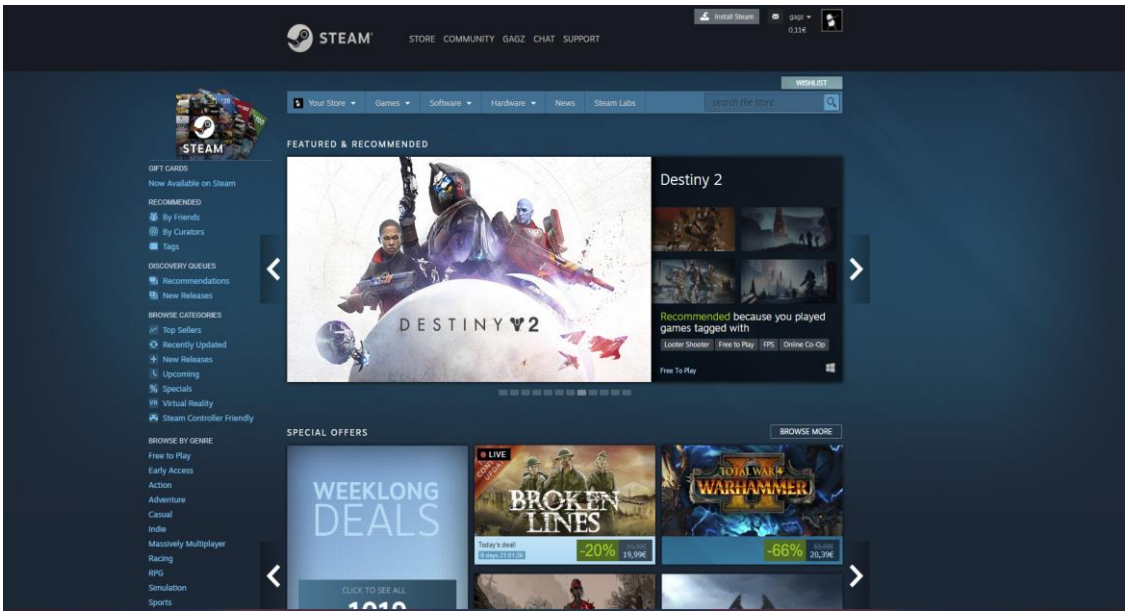
Za primjer se mogu uzeti dvije platforme za igranje video igara, Steam i Origin. Iako obje platforme imaju istu funkcionalnost, a to je da omoguće igračima kupovinu i igranje željenih naslova video igara oni se razlikuju po svom vizualnom identitetu. Pomoću percepcijskih uzoraka je moguće učiniti proizvod ili kompaniju prepoznatljivom, oni omogućuju korisniku memorizirati proizvod i istaknuti ga od drugih.

Percepcijski uzorci su uvijek prisutni, htjeli to ili ne. Prilikom dizajniranja i implementiranja funkcionalnih uzoraka nesvjesno se dizajniraju i percepcijski uzorci, koji se naravno mogu redizajnirati kasnije. Razlog automatskog dizajniranja je taj što su web i njegovi elementi napravljeni na takav način da moraju imati neki inicijalni izgled.

Neki dijelovi percepcijskih uzoraka:

- Paleta boja
- Tipografija
- Ikonografija
- Ilustracije
- Slike
- Animacije
- Smještaj elemenata

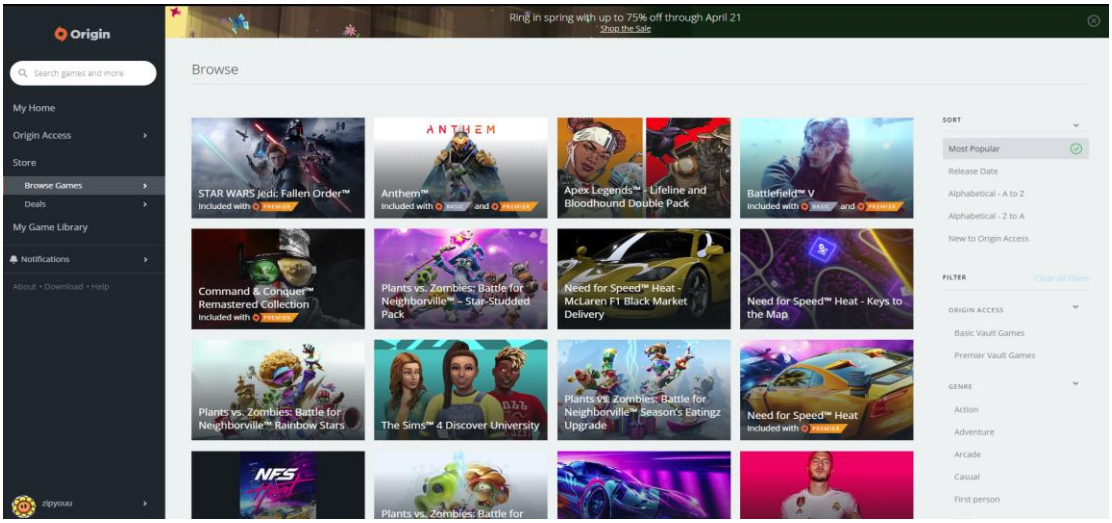
Prethodno su navedene platforme Steam i Origin kao primjer. Na slici 6. je prikazana Steam platforma gdje se može vidjeti kako koristi tamnu paletu boja. Boja teksta je kontrastna bijela kako bi korisnik lakše pronalazio željenu informaciju. Na stranici je također dosta slika koje za cilj imaju promovirati što više igara korisnicima.



Slika 6: Sučelje Steam

(preuzeto sa https://store.steampowered.com/, 19.04.2020.)

Na slici 7. je prikazana Origin platforma, gdje se može vidjeti kako je korištena, za razliku od Steam-a, svijetla paleta boja. Također različit je razmještaj elemenata u odnosu na Steam, gdje je navigacija sa lijeve strane umjesto na vrhu, a proizvodi su s desne strane navigacije umjesto ispod nje. Sve slike su ujednačene veličine te nema promotivnih slika kao što je slučaj na Steam-u.



Slika 7: Sučelje Origin

(preuzeto sa https://www.origin.com/irl/en-us/store/browse, 19.04.2020.)

3.2. Stilski priručnici

Frost navodi stilske priručnike kao temelj svakog dobrog sustava dizajna. Ono što stilski priručnici omogućavaju je dokumentiranje i organiziranje materijala koji pružaju smjernice kako koristiti sustav. [5]
Stilski priručnici mogu pokriti mnogo tema. Neke od njih su:

- Identitet proizvoda
- Glas i njegov ton
- Pisanje
- Kod
- Jezik dizajna
- Uzorci korisničkog sučelja

Identitet proizvoda odgovara na pitanje kako će se prezentirati na raznim medijima na kojima se želi prezentirati. Ono uključuje definiranje sredstava i materijala koji čine proizvod prepoznatljivim i jedinstvenim. U dokumentaciju identiteta se opisuju logo materijali, palete boja, poruke (misija, ciljevi i slično), marketinški materijali (prezentacije, poslovne kartice i slično). Ono što je bitno kod ove dokumentacije je da prezentacija bude ujednačena na svim mogućim kanalima na kojima se oni prezentiraju. [6]

Svakoj kompaniji je cilj što više sebe prezentirati kako bi što više ljudi čulo za njih i koristilo njihov proizvod. To znači da se kompanija ili proizvod osim na internetu može prezentirati na tiskanim materijalima, televiziji, radiju i ostalim medijima za protok informacija. Zbog toga je potrebno odrediti način na koji će se informacija prenijeti korisniku. U stilskim priručnicima gdje se opisuju glas i ton opisuje se način na koji način se informacija prikazuje korisniku. Taj glas (pristup) može biti neozbiljan, šaljiv, strog, formalan i sl., ovisno o kompaniji, proizvodu i njegovu sadržaju. Svatko ima glas, ali je ton ono što definira način na koji će netko shvatiti taj glas. U određenim situacijama se koristit određeni ton, te je stoga potrebno odrediti i način na koji će se informacija prenijeti u određenoj situaciji.

Sa rastom weba i sadržajno upravljanim sustavima (*eng. content-managed systems*) puno više ljudi može upravljati sadržajem koji se prikazuje na digitalnim proizvodima. Zbog toga se u stilske priručnike navode i smjernice za pisanje kako bi stil pisanja i sadržaj na proizvodima bio ujednačen.

Kako su stilski priručnici stvoreni da bi olakšali posao ljudima koji rade na projektu i kako bi se proizvod razvijao sustavno, ujednačeno tako bi i programeri trebali imati neke smjernice za pisanje koda. Smjernice za pisanje koda objašnjavaju način na koji se piše kod koji je čitljiv, skalabilan i lak za održavanje. Ukoliko ne bi postojale smjernice koje bi programeri trebali srediti nakon nekog vremena razvoja može nastati kaos iz kojega se kasnije teško izvući.

Smjernice za pisanje koda sadrže konvencije i primjere kako bi programer koji radi na projektu trebao pristupiti kodu na kojem radi. Ukoliko su smjernice dobro konstruirane tim se može koncentrirati na stvari koje će možda ubrzati aplikaciju, poboljšati performanse ili dodati novi modul, umjesto trošiti vrijeme na refaktoriranje već napisanog koda.

Jezik dizajna definira smjer u kojemu bi dizajn trebao ići, njegovu filozofiju, ciljeve i osnovne principe te način na koji bi se digitalnom proizvodu trebalo pristupiti. On je jako bitan jer na taj način se određuje kako će proizvod ujednačeno biti prikazan na svim mogućim ekranima i medijima. Kroz jezik dizajna se pokušava apstraktne koncepte objasniti na što lakši i bolji način. Jedan od najpoznatijih primjera jezika dizajna je Googleov dizajn materijala (*eng. material design*).

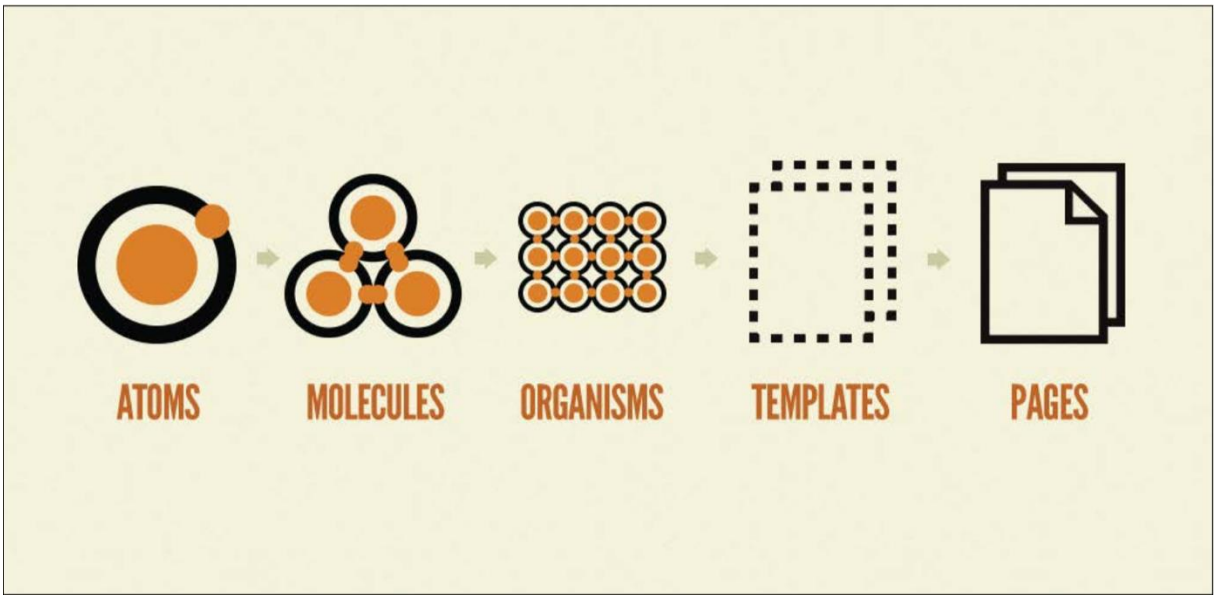
Kroz rad je već par puta istaknuto kako se razvoj sve više bazira na modularni pristup razvoju digitalnih proizvoda. Sve te module, odnosno uzorke, odnosno komponente programeri na korisničkoj strani razvoja spremaju u tzv. knjižnice uzoraka (*eng. pattern libraries*). Knjižnice uzoraka su sve popularnije u svijetu razvoja na korisničkoj strani te se može reći kako postaju jedan od temeljnih stvari kod dizajniranja modernih sučelja. Jedna od najpoznatijih tehnologija za rad s njima je Patternlab koji će biti opisan u nastavku rada.

3.3. Elementi atomskog sustava dizajna

Tvorac atomskog sustava dizajna je Brad Frost. Atomski dizajn je metodologija za kreiranje sustava dizajna. Kako i samo ime govori, inspiraciju za kreiranje atomskog dizajna Frost navodi kako ju je našao u kemiji. Povezao je prirodni svijet, koji se sastoji od atoma, molekula i organizama, sa svijetom web razvoja. Uvidio je kako je prirodni svijet napravljen na način da najmanje funkcionalne tvari (atomi) u kombinaciji stvaraju veće, kompleksnije tvari i organizme. Na sličan način to se može primijeniti i na razvoj web-a. U prirodi su temelj za kreiranje bilo čega atomi, dok su na web-u to HTML elementi. Spajanjem HTML elemenata stvaraju se kompleksnije komponente. Ono što Frost navodi je da je atomski dizajn mentalni model koji pomaže pri razmišljanju u isto vrijeme o sučeljima kao cjelini, ali i kolekciji manjih dijelova.

Kroz ovakav pristup Frost je podijelio web komponente na pet razina: [5]

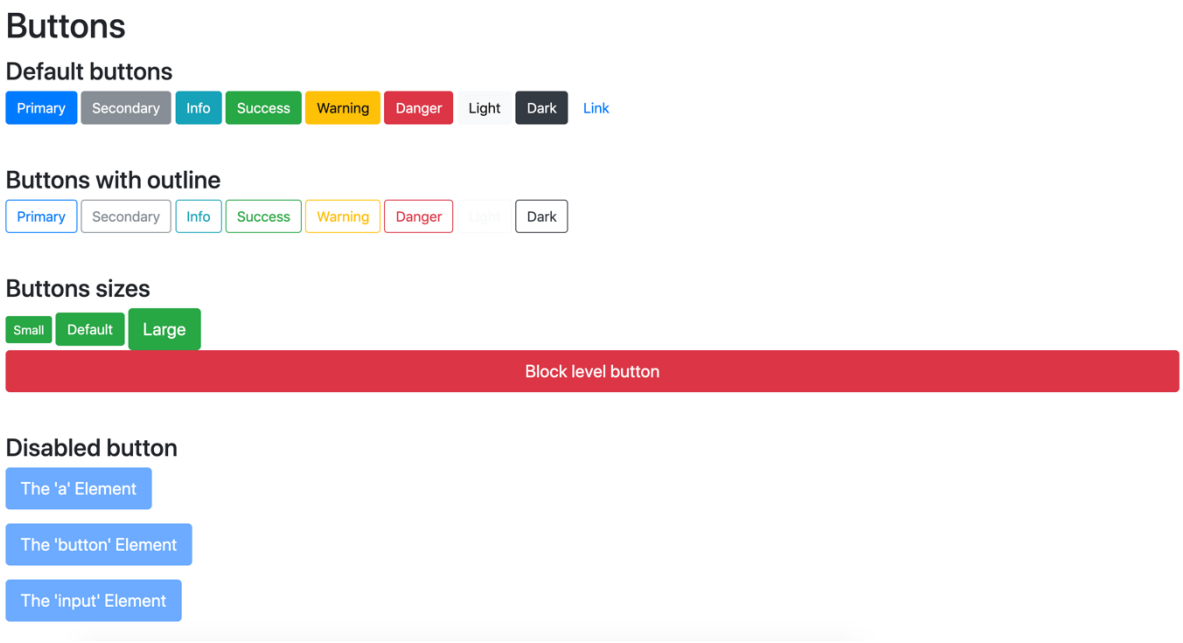
- Atomi
- Molekule
- Organizmi
- Predlošci
- Stranice



Slika 8: Elementi atomskog dizajna (Atomski dizajn, B.Frost, 2016.)

3.3.1. Atomi

Kao što je prethodno navedeno atomi su temelj za gradnju bilo koje komponente na sljedećoj razini atomskog dizajna, što znači bez njih ne bi bilo ni Web-a. U atome spadaju osnovni HTML elementi koji se ne mogu razbiti na manje dijelove. Pod tim se misli na gumbove, etikete (*eng. label*), odjeljke itd. Iako su temelj web-a, oni samo po sebi ne mogu postojati, potrebno je više njih kako bi u suradnji dali smisao digitalnom proizvodu. Svaki od njih je različit po svome ponašanju i on ima svoju svrhu. Zbog različitih funkcionalnosti koji svaki atom ima on će u kombinaciji s drugim atomima, istim ili različitim od njega, kreirati i različite komponente. Kod definiranja atoma razmišlja se na širok način, potrebno je razmišljati kako će se atom primjenjivati na cijeloj paleti komponenti. Oni demonstriraju osnovni stil web mjesta, što se lako može provjeriti, razvijati i održavati kroz knjižnice uzoraka.

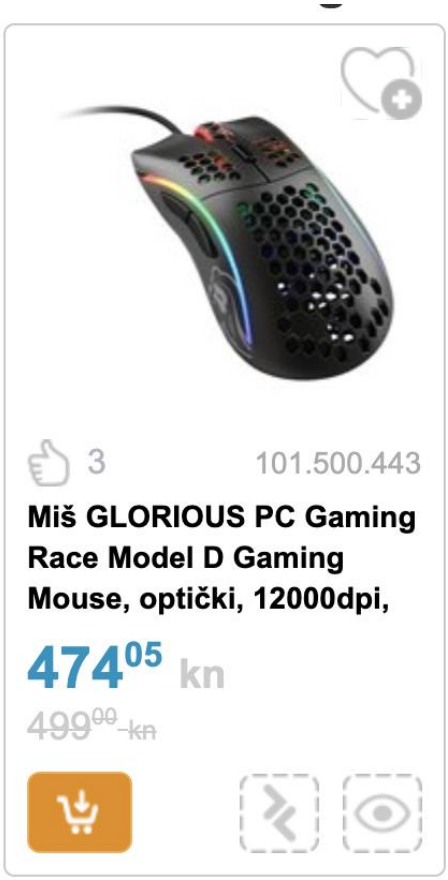


Slika 9: Primjer atoma – gumb (Bootstrap Magic)

Na slici 9. je vidljivo da se definiraju razne kombinacije gumba. Može se vidjeti da postoji primarni, sekundari stil za gumb, stilovi za razne situacije kao npr. gumb za opasnost ili upozorenje i slično. Osim toga određuje se stil za gumb kad je on onemogućen, veličine gumba, bilo da je on na blok razini ili na razini reda.

3.3.2. Molekule

Molekule su kombinacije atoma, povezanih skupa, u cjelinu koji poprimaju nova svojstva. Za njih se može reći kako su to male grupe elemenata (atoma) koji funkcioniraju zajedno kao jedna jedinica. Primjer može biti kartica produkta (*eng. product card*) koja se sastoji od atoma za odlazak na drugu poveznicu (a element), slike, naslova, opisa, gumba. S ta tri atoma koji rade kao jedna jedinica, karta proizvoda ima svoju svrhu, a to je prikazati proizvod kako izgleda, prikazati njegov naslov i ukoliko se klikne na kartu odvesti korisnika na željenu lokaciju.



Slika 10: Primjer karte produkta

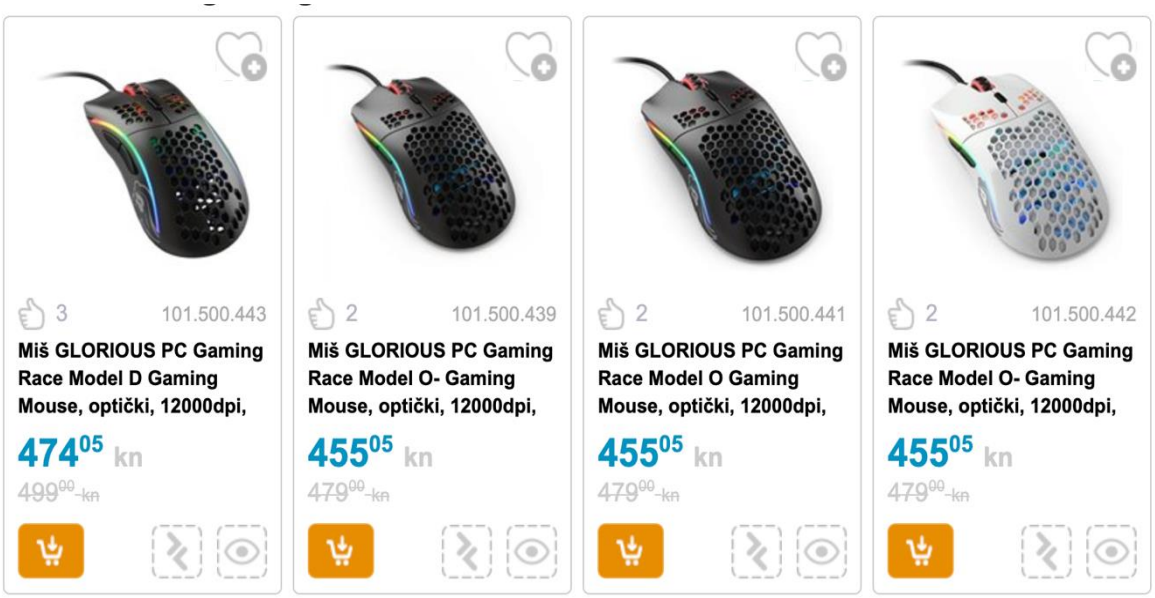
(preuzeto s https://www.links.hr/hr, 01.05.2020)

Ovakva molekula se sada može koristiti na više mjesta kao što su lista proizvoda, rezultati pretraživanja proizvod, sličnih proizvoda itd. Ono što omogućavaju molekule i pristup atomskog dizajna je pristup jedinstvene odgovornosti (*eng. single responsibility principle*), što znači svaka komponenta bi trebala raditi jednu stvar, stvar za koju je predviđena i to je to. Takav pristup omogućava lakše testiranje, održavanje, ponovno korištenje, te potiče konzistentnost na cijelom digitalnom proizvodu.

3.3.3. Organizmi

Frost organizme definira kao relativno složene komponente sučelja sastavljene od grupa molekula i/ili atoma i/ili drugih organizama. [7] Ti organizmi formiraju različite sekcije unutar sučelja proizvoda. Organizmi mogu sadržavati različite ali i iste molekule, ovisno o funkciji koju organizam treba ispuniti. Kroz izradu komponenti od najmanjih dijelova do organizama dizajneri i programeri dobivaju osjećaj konteksta u kojem smjeru bi proizvod trebao ići. Oni

demonstriraju komponente koje su u akciji, te služe kao uzorci koji će se koristiti na digitalnom proizvodu.



Slika 11: Primjer liste karata produkta

(preuzeto s https://www.links.hr/hr, 01.05.2020.)

Na slici 11. se može vidjeti organizam liste karata produkta. Taj organizam se sastoji od više istih molekula i njegova svrha je prikazati korisniku popis proizvoda koje može kupiti. Ovaj organizam se može koristiti na više mjesta, osim što se može na početnoj stranici kako bi korisniku se naglasili neki artikli, on se može koristiti i na stranici za pretraživanje rezultata gdje će se korisniku ispisati artikli koji zadovoljavaju uvjete pretrage kroz listu karata produkta. Osim na pretrazi, kao što je u radu prethodno navedeno, ovaj se organizam može koristiti i za predlaganje sličnih proizvoda korisniku.

3.3.4. Predlošci

Na predlošcima završava kemijska terminologija, unutar atomskog sustava dizajna, koja se do sada koristila za komponente kako bi se komponente lakše prezentirale klijentu i svim ostalim ljudima koji sudjeluju na dostavljanju proizvoda klijentu. Daljnje korištenje kemijske terminologije bi moglo samo izazvati nepotrebnu konfuziju između sudionika. Zbog toga su tu predlošci. Predlošci su objekti na razini stranice na koje se smještaju komponente kako bi pobliže dočarali kako će određena stranica izgledati. Na taj način se može prezentirati na koji

način komponente surađuju, kako funkcioniraju skupa, što još više pojašnjuje kontekst u kojemu su prethodno apstraktni elementi bili predstavljeni.

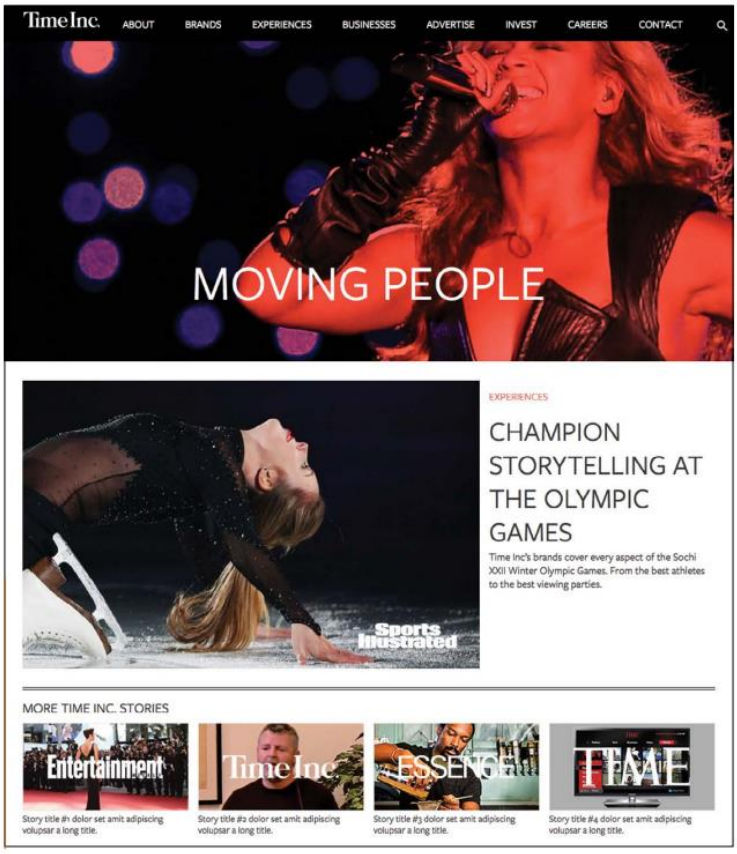
Slika 12: Primjer predloška (Atomski dizajn, B.Frost, 2016.)

Ono na što se predlošci fokusiraju nije na finalni izgled stranice nego se fokusiraju na sadržajnu strukturu stranice. Može se reći da se fokusira na prethodno opisane funkcionalne uzorke. Svaki sustav dizajna mora biti prilagodljiv sadržaju koji treba prezentirati te se na predlošcima sastavlja kostur stranice koji će se moći prilagođavati potrebnom sadržaju. Unutar predložaka se definiraju predlošci na koji način se upravlja sa sadržajem, kao što su veličina slika, dužina teksta i slično.

3.3.5. Stranice

Frost stranice definira kao posebne instance predložaka koje prikazuju kako sučelje izgleda sa realnim sadržajem. [7] Ukoliko je predložak kostur stranice, onda se može reći kako su stranice meso koje se lijepi na taj kostur. Stranice se formiraju na način da se na predloške stavlja sadržaj koji će danas-sutra ići na produkciju, odnosno sadržaj koji će korisnik vidjeti i biti u interakciji s njim. S poslovne i korisničke strane se može reći kako je ovo najvažniji element atomskog sustava dizajna.

Na ovom elementu svi elementi se sjedinjuju i tvore funkcionalno korisničko sučelje. Jedna od važnijih stvari koje stranice predstavljaju je to što su one bitna stavka prilikom testiranja. Na osnovu stranica se testira uspješnost prethodnih elemenata. Ukoliko neki prethodni element izaziva problem, tako da se ne prilagođava potrebnom sadržaju, grešku je lako identificirati zbog modularnosti. Na stranicama je najbolje vidljivo kako modularni način i široko razmišljanje prilikom izrade manjih elemenata je jako bitno. S takvim komponentama izrada stranice je relativno stabilna i jednostavna.



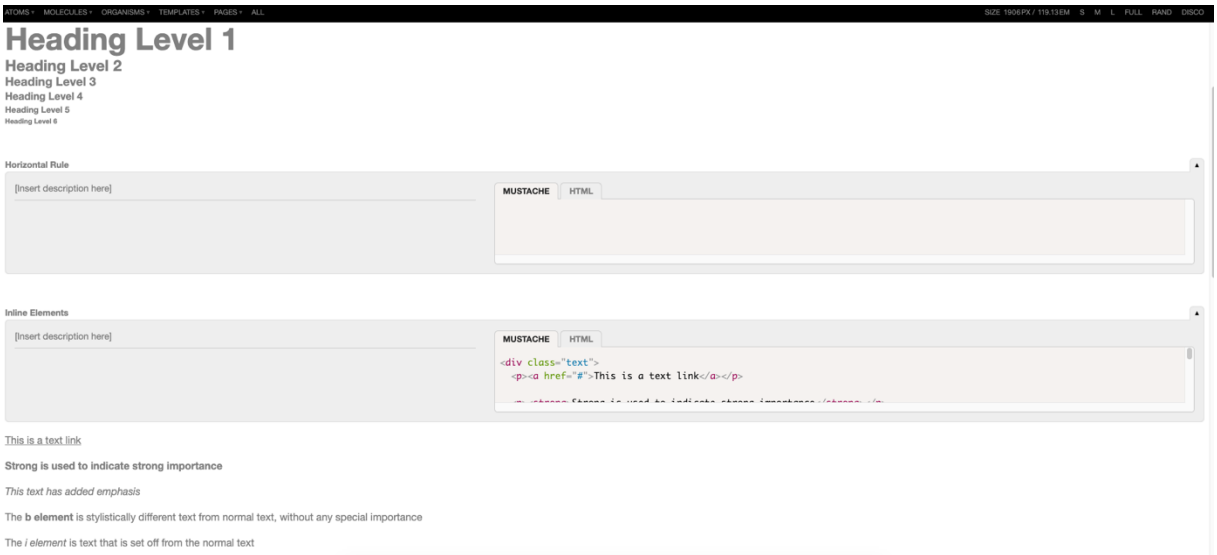
Slika 13: Primjer stranice (Atomski dizajn, B.Frost, 2016.)

Također, sa stranicama je moguće prikazati razne varijacije sadržaja. Primjer može biti početna stranica korisnika koji se prijavio i onog kojeg nije, stranica sa proizvodima u košarici ili bez, stranica sa administrativnim pravima ili bez i slično. Sa svim tim prilagodbama koriste

se iste komponente, ali se mijenja izgled stranice, odnosno sadržaj koji će korisniku biti prikazan. Na ovakve varijacije komponente moraju biti spremne. Komponente koje su podložne varijacijama osiguravaju stabilnost sustava.

3.4. Pattern Lab

Pattern Lab je jedan od alata koji omogućava sve ono što je navedeno prethodno vezano za atomski sustav dizajna. Jedan od tvoraca, Brian Muenzenmeyer [8] ga definira kao alat otvorenog koda koji pomaže pri stvaranju i održavanju komponenti unutar sustava dizajna. To je u biti knjižnica uzoraka koji se koriste kroz projekt. Pattern Lab su razvili Dave Olsen, Brian Muenzenmeyer i Brad Frost. Može se koristiti uz PHP jezik ili Node.js, ali stranice ne moraju biti napravljene u ovim jezicima kako bi se koristile, ovi jezici služe kao motor (eng. *engine*) za pokretanje. Ono što omogućuje ovaj alat je generiranje statičnih stranica atomskog sustava, te dokumentiranje i dodavanje bilješki. Način na koji radi Pattern Lab je taj da uzme izvorni kod i popratne dodatke (slike, video materijali itd.) te ih pretvori u funkcionalne uzorke unutar knjižnice uzoraka. Odlika Pattern Lab-a je ta što on daje slobodu razvojnom timu. To znači da tim sam odlučuje kako će strukturirati projekt, kako će se sintaksa koda pisati, koje tehnologije će se koristiti, programski okviri, biblioteke, skripte itd.

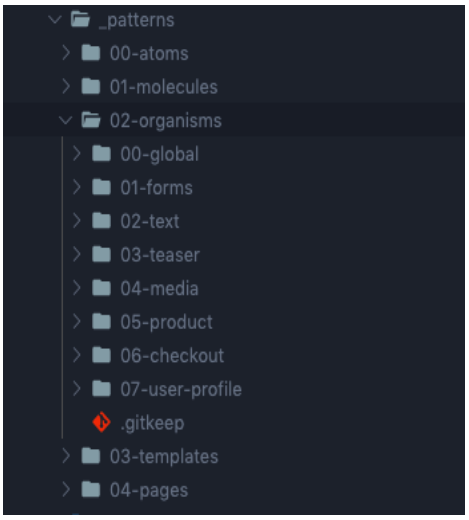


Slika 14: PatternLab

Kako se Pattern Lab temelji na principima atomskog dizajna, tako se i rad s njim zasniva na tim principima. Prvo se kreiraju manji dijelovi, odnosno atomi, koji se uključuju u veće dijelove, odnosno molekule, pa molekule u organizme i tako dok ne dobijemo stranicu. Krasi ga DRY (*eng. don't repeat yourself – nemoj se ponavljati*) princip, što dosta olakšava stvari,

skraćuje posao i ne zagađuje kod. S ovakvim temeljima rada kroz Pattern Lab je moguće promijeniti neku komponentu te će se promjena primijeniti na svim komponentama na kojim je ona bila uključena.

Za instalaciju je potrebno imati Node.js, te s naredbom `npm create pattern-lab` se pokreće instalacijski izbornik s kojim se može izabrati direktorij gdje će sve biti smješteno, jezik za predloške, koji će se uzorci inicijalno kreirati. Nakon toga je moguće konfigurirati projekte te kreirati strukturu direktorija, gdje opet Pattern Lab omogućuje slobodu. Inicijalno direktoriji se nazivaju kao i elementi atomskog sustava, ali tim može ih preimenovati kako hoće. Bitno je samo da nazivi ono bude konstantni kroz cijeli projekt.



Slika 15: Primjer strukture direktorija

Prethodno je objašnjeno kako se Pattern Lab temelji na atomskom dizajnu i kako uključuje manje komponente u veće. To se postiže sa Mustache jezikom za predloške. Naziv je dobio što se koriste {{}} koji izgledom podsjećaju na brkove. Sve što je unutar tih zagrada se interpretira kao jezik uzoraka. Komponente se uključuju na ovaj način:

```
{{> molecules-search}}
```

Znak > označava da se uključuje neka komponenta unutar datoteke, u ovom slučaju polje za pretraživanje koje pripada elementu molekula. Ovakav princip se koristi i na manjim i na većim komponentama od molekula.

S Pattern Lab-om je moguće dinamički manipulirati s podacima koji se koriste na komponentama. Na taj način je moguće zamijeniti podatke sa predloška sa stvarnim podacima koji će se prikazati korisniku. S ovim svojstvom je lako moguće preći iz predloška na stranice sa stvarnim sadržajem. Na taj način je moguće lakše prezentirati klijentu kako će se proizvod ponašati i kako će izgledati, te uz to omogućuje lakše testiranje i otkrivanje grešaka ukoliko postoje. Za dodavanje dinamičkog sadržaja koriste se obično JSON datoteke, ali Pattern Lab isto kao i prije daje slobodu timu da izaberu drugi format za strukturiranje podataka. Inicijalni

podaci koji se koriste na predlošcima su definirani u datoteci *data.json*. Da bi se ispravni podaci koristili na razini stranice potrebno je nadjačati podatke sa predloška. To se postiže tako da se u direktoriju stranica za dati uzorak kreira datoteka sa željenim podacima. Ta datoteka mora se nazivati isto kao i uzorak. Uz dinamičko mijenjanje podataka moguće je koristiti i varijable, petlje itd.

Kroz rad se već par puta promijenilo kako se s atomskim dizajnom može lakše testirati. To se najbolje može vidjeti upravo s Pattern Lab-om jer on omogućava varijaciju sadržaja, odnosno dijelova na komponenti koji će se prikazati ili sakriti u nekim slučajevima. Na taj način se skraćuje posao dizajnerima, ljudima zaduženim za sadržaj, klijentu i drugima. Kroz tzv. pseudo-uzorke moguće je simulirati različite scenarije u kojima će se komponenta ponašati drugačije. Za primjer se može uzeti navigacija, gdje će se neregistriranom korisniku prikazati jedan dio navigacije, registrirani korisnik će imati više opcija dok recimo administrator će možda moći vidjeti sve opcije. Da bi se to postiglo potrebno je kreirati datoteku za pseudo-uzorak npr. *navigacija~admin.json* gdje će se nalaziti JSON objekt sa željenim podacima. Znak ~ označava da će ta datoteka naslijediti sve podatke iz *navigacija.json* te dodati ono što je u administratorskoj datoteci. S ovakvim pristupom da se dinamički podaci mogu mijenjati unutar *json* datoteka više nije potrebno kontaktirati stalno programere na korisničkoj strani da naprave sitnu promjenu kako bi onda bila vidljiva drugima. Sada to može uraditi i širi krug ljudi na jednostavan način.

Pattern Lab kao alat je svjestan da se sadržaj treba prilagođavati medijima na kojima će se on prikazati te u sebi ima ugrađeno svojstvo da se sadržaj lako može prikazati na uređaju određene širine. Moguće je gledati kako se komponenta ponaša na više preglednika i uređaja u isto vrijeme, potrebno je samo biti spojen na istu mrežu sa svim uređajima. Uz to postoji opcija gdje se može vidjeti dokumentacija komponente. Kroz tu dokumentaciju se može opisati komponenta, čemu služi, koje elemente uključuje i u koje je elemente i ona sama uključena. Uz sve to moguće je vidjeti i sam *HTML* i *Mustache* kod bez uporabe dodatnih alata.

3.5. Prednosti i mane atomskog sustava dizajna

Prednosti atomskog sustava dizajna su:

- **Sustavni pristup dizajnu**

  Ono što atomski dizajn omogućava je sustavni pristup dizajniranju proizvoda i njegovog implementiranja. Na taj način se smanjuje broj grešaka, poboljšava komunikaciju i povećava brzinu izvođenja projekta.

- **Modularnost**

  Kroz rad se nekoliko puta spomenula modularnost kao prednost kod razvoja programskih proizvoda. Atomski dizajn nije izuzetak. Modularnost je razrađena na najmanje elemente koji se kasnije uključuju u veće elemente što na kraju rezultira stranicom.

- **Mogućnost ponovnog korištenja komponenti**

  Atomski sustav se temelji na svojim elementima (atomi, molekule, organizmi, predlošci, stranice). Komponente kreirane na takav način je moguće koristiti više puta kroz projekt, gdje će se skratiti proces kreiranja složenijih komponenti jer se na kraju kreiranje veće komponente svodi na povezivanje manjih dijelova (elemenata). R.F. Augusdi u svom istraživanju navodi da korištenje atomskog sustava dizajna, odnosno ponovno korištenje već kreiranih komponenti ubrzava proces kreiranja dizajna za nova sučelja za 49.9%, te da je smanjuje troškove interakcije i dogovaranja za 72.7%. [9] Konaté također navodi kako sustavi dizajna ubrzavaju vrijeme za razvoj komponenti za programere za kreiranje novog sučelja za 20%. [6]

- **Odvajanje prezentacijskog od funkcionalnog dijela**

  Manji elementi atomskog sustava dizajna se fokusiraju na funkcionalnost i strukturu komponente. Na taj način programeri se fokusiraju da jezgra komponente bude dobro implementira, te kasnije kad se rade završni elementi odnosno stranice fokusira se na sadržaj i prezentacijski dio koji će korisnik na kraju vidjeti.

- **Testiranje**

  Atomski sustav dizajna omogućava lakše testiranje proizvoda, lakše identificiranje problema što omogućuje brži razvoj aplikacije. Ako se problem pronađe na manjoj komponenti i riješi se, problem je automatski riješen i na većim komponentama. L. Nghi predlaže TDD metodologiju (eng. *Test Driven Development*) rada sa atomskim sustavom dizajna, kako bi se kroz iterativnost razvoja i kroz temeljitost testiranja manjih komponenti razvijali stabilni kompleksniji sustavi. [10]

- **Povezuje razvojni tim**

  Atomski sustav dizajn povezuje više sfera razvojnog tima te uspostavlja zajednički jezik pri razvoju. Na taj način smanjuje se broj problema u komunikaciji, svi su na istoj valnoj

duljini i znaju što trebaju raditi te se stvara empatija među ljudima, jasno im je što mogu očekivati i zašto svaki njihov zahtjev ne može proći.

- **Susretljiv i jednostavan**
  Principe atomskog dizajna sustava je relativno jednostavno savladati i shvatiti te novi ljudi koji se uključe u tim vrlo brzo pohvataju način rada koji zahtjeva atomski sustav dizajna.

Mane atomskog sustava dizajna su:

- **Vrijeme i novac**
  Za provedbu atomskog sustava dizajna potrebno je uložiti dodatno vrijeme, novac i ljude kako bi se on kvalitetno proveo, na što klijent nije baš uvijek spreman.
- **Ponekad neisplativ**
  Atomski dizajn sustava se isplati provoditi na većim sustavima i aplikacijama, jer vrijeme provedeno stvarajući elemente atomskog dizajna sustava može se provesti na realizaciji kompletnog projekta. On se neće isplatiti na sustavima koji nemaju ponavljajuće elemente ili ako je aplikacija jednostavnog dizajna.

4. Opis ideje web aplikacije

Za praktični dio diplomskog rada je napravljena web aplikacija kroz koju je implementiran atomski sustav dizajna. Aplikacija je namijenjena nogometnim zaljubljenicima kako bi im praćenje nogometnih utakmica bilo još zanimljivije. Inspirirana je već postojećom aplikacijom *Fantasy Premier League,* gdje se u radu umjesto engleske Premier lige pokriva 1. Hrvatska nogometna liga, što znači da su ciljani korisnici hrvatskog tržišta. Ideja je nastala zbog toga što još ne postoji ovakva aplikacija na hrvatskom tržištu, dok je HNL dosta popularan i praćen.

U *1. Hrvatskoj Fantasy ligi* postoje četiri vrste korisnika, gdje svaki korisnik ima određene vrste prava:

- **Neregistrirani korisnik**
  - može vidjeti osnovne informacije na početnoj stranici
  - može se registrirati popunjavanjem forme i slanjem zahtjeva za registraciju
- **Registrirani korisnik**
  - može se prijaviti popunjavanjem forme i slanje zahtjeva za registraciju
  - može obnoviti lozinku popunjavanjem forme i slanja zahtjeva za obnovu lozinke
  - može vidjeti popis liga
  - može se registrirati i sudjelovati u kreiranoj ligi
  - može sudjelovati u selektiranju ekipe
  - može upravljati ekipom
  - može pregledavati statistiku igrača
  - može uređivati profil
  - može odjaviti se
- **Moderator**
  - ima sva prava kao registrirani korisnik
  - može kreirati ligu
  - može izbrisati ligu
  - može kreirati selekciju igrača
  - može upravljati ligom
- **Administrator**
  - ima sva prava kao moderator
  - može upisivati rezultate i bodove nogometašima
  - brisati igrače iz aplikacije

U ligi fantazija (eng. *fantasy league*) potrebno je kreirati svoju ekipu, gdje se mogu birati nogometaši iz cijele lige uz ograničen budžet. Svaki sudionik lige može izabrati 15 igrača, gdje mora imati:

- dva golmana
- pet obrambenih igrača
- pet veznih igrača
- tri napadača

Svaki nogometaš se razlikuje po pozicijama i po cijeni. Bodovanje se vrši na osnovu učinka nogometaša gdje se boduju njegovo igranje utakmica, golovi (ovisno o poziciji), asistencije, žuti i crveni kartoni. Svaka ta kategorija donosi određeni broj bodova te je potrebno dobro razmisliti koje igrače odabrati uz limitiran budžet. Korisnik ima mogućnost iskoristiti do tri žetona s kojima mu se udvostručuju bodovi za određeno kolo. Osim toga korisnik može birati koju formaciju želi koristiti određeno kolo ili koji igrač će igrati određeno kolo, sve u cilju kako bi osvojio što više bodova. Pobjednik je onaj korisnik koji na kraju prvenstva ima najviše bodova.

Tablica 1: Bodovanje igrača

Kategorija	**Bodovi**
Igranje utakmice	1
Gol – obrambeni igrač, golman	6
Gol – vezni igrač	5
Gol - napadač	4
Asistencija	3
Žuti karton	-2
Crveni karton	-4

Za izradu aplikacije korišteno je više tehnologija koje će detaljno biti opisane u nastavku diplomskog rada.

5. Opis korištenih tehnologija

U sklopu implementacije praktičnog dijela diplomskog rada su korištene sljedeće tehnologije:

- **HTML**
- **CSS**
    - Sass
- **Vue.js**
- **Firebase**

5.1. HTML

HTML (*Hyper Text Markup Language*) je jezik koji je razvio Tim Berns Lee 1989. godine. To je jednostavni jezik oznaka (eng. *tags*) koji služi za izradu hipertekstualnih dokumenata. Te oznake su temelj svakog hipertekstualnog dokumenta, te one predstavljaju semantičku, strukturnu i prezentacijsku informaciju. Može se reći kako je HTML jedan od najrasprostranjenijih i najkorištenijih jezika, jer bez njega danas se ne može zamisliti web. HTML datoteke se mogu prepoznati po tome što završavaju sa *.html* oznakom.

Odlika HTML-a kao jezika je njegova jednostavnost i lakoća njegova savladavanja. Svaka oznaka označava određeni element, koji ima svoje semantičko značenje te svoj prezentacijski dio. Na osnovu tih elemenata računalo, odnosno preglednik, zna što taj element predstavlja i kako će ga prikazati krajnjem korisniku. Taj proces se još i naziva označavanje (eng. *markup*). HTML dokumenti se pokreću u preglednicima koji prezentiraju dokument korisnicima na prigodan i ujednačen način, što znači da oznaka za naslov će jednako biti interpretirana na različitom pregledniku.

Još jedna odlika ovoga jezika što on nije ovisan o platformi, odnosno može se odvijati u bilo kojem pregledniku, na bilo kojem operacijskom sustavu. Nisu potrebni nikakvi dodatni alati za izradu takvog dokumenta, dovoljan je program za pisanje tekstualnih datoteka. Za razliku od drugih jezika koji se kompiliraju, preglednici interpretiraju sadržaj HTML datoteke na jednostavan i logičan način, od vrha prema dnu te od lijeva prema desnu. [11]
Osnovna sintaksa HTML oznake izgleda ovako:

```
<p>HTML oznaka</p>
```

Dakle, svaka oznaka započinje sa znakom „<“. Iza nje ide ime elementa, konkretno u ovom primjeru to je oznaka za odjeljak, te nakon imena slijedi znak „>“. Na osnovu početne oznake preglednik zna o kojem se elementu radi te će ga prezentirati korisniku na odgovarajući način. Zatim se stavlja tekst koji autor želi prikazati. Nakon toga je potrebo oznaku zatvoriti.

Procedura je ista kao i sa otvaranjem oznake, s tim da prije imena oznake je potrebno dodati znak „/“. Postoje oznake koje nije potrebno zatvarati, jer ne postoji sadržaj koji bi išao između oznaka. Takve oznake se nazivaju samo zatvarajuće oznake (eng. *self closing tags*). Primjer takve oznake je oznaka za sliku (<img>).
HTML elementima se mogu proširiti njihove značajke i dati dodatne informacije o njima samima. To se može postići sa atributima. Preko atribute je moguće uređivati stilove elementa, veličinu, širinu, dodati klasu, dodati identifikacijsko ime elementu, dodati atribute kojim se olakšava korištenje dokumenta osobama sa poteškoćama. Jako se puno koriste i njihovo poznavanje je jednako važno kao i poznavanje imena oznaka. Primjer elementa sa atributima:

```
<p id="moj-id" class="moja-klasa" data-moj-atribut="mojAtribut">
      Sadržaj ovog odjeljka.
</p>
```

Da bi preglednik mogao prepoznati i prikazati HTML kako treba potrebno je započeti dokument sa <html> oznakom. Između html oznaka mogu doći sekcije zaglavlja (<head>) i sekcija tijela (<body>).
U zaglavlju se navode meta podaci, gdje se mogu dodati dodatne informacije o dokumentu, uključiti stilovi za cijeli dokument, uključiti skripte s kojom se dobije interaktivnost između korisnika i web stranice/aplikacije.
U sekciji tijela se dodaju sve ono što se želi prezentirati korisniku dokumenta, naslovi, odjeljci, slike itd.

```
<html>
      <head>
            <title>Ime dokumenta</title>
      </head>
      <body>
            <h1>Početni naslov</h1>
      </body>
</html>
```

Kako bi HTML funkcionirao jednako neovisno od platforme, stvoreni su standardi o kojima brine W3C (*World Wide Web Consortium*). Njihov cilj je učiniti Web što više dostupnim, na što više uređaja, kako bi informacije i sadržaj se širile što brže i što lakše. Sa novim standardima je evoluirao i HTML te je kroz povijest imao više verzija, s kojima bi se predstavljale nove stvari i mogućnosti na Web-u.

Poznatije verzije kroz povijest su: [12]

- **HTML 1** – prva verzija nastala 1989. godine. Mogućnost jezika su bile male, gdje se mogao prikazivati samo jednostavni tekst. Podržavali su ga samo preglednici *Lynx* i *Mosaic*.
- **HTML 2** – druga verzija je nastala 1995. godine. U toj verziji se HTML širi i na ostale preglednike. Predstavljeni su neki elementi za forme. S HTML 2 verzijom su definirani osnovni standardi HTML, jer prije toga još nije postojao *W3C*.
- **HTML 3.2** – verzija nastala 1997 godine. S ovom verzijom su došle tablice te je dodano još elemenata za rad s formama. HTML 3 uvodi mogućnost stiliziranja stranica s CSS-om.
- **HTML 4.01 –** verzija nastala 1998. S ovom verzijom došlo je do evolucije u razvoju i dizajniranju web stranica zbog mogućnosti dodavanja stilskih listova (eng. *style sheets*). To znači da se mogao odvojiti prezentacijski od sadržajnog dijela. S tim je bilo moguću isti sadržaj prikazati na jako puno različitih načina.
- **HTML 5 –** verzija nastala 2014. godine, koja se i danas koristi. S ovom verzijom su došli semantički elementi, prošireni su elementi forme, pojednostavila se sintaksa i izbačeni su elementi koji se nisu pretjerano primjenjivali.

5.2. CSS

CSS (*Cascading Style Sheets*) je jezik za prezentaciju sadržaja na web stranicama i aplikacijama. Kreiran je 1996. godine. S njim možemo mijenjati boje koje želimo prikazati, raspored elemenata na stranici, visinu, širinu i oblik elemenata, omogućava prilagođavanje sadržaja raznim širinama i tipovima ekrana. Ideja koja se krije za kreiranje ovog jezika je bila ta da se odvoji prezentacijski od sadržajnog dijela web stranica. Kao rezultat toga je lakša izrada web stranica/aplikacija i njihovo održavanje. [13]

CSS je moguće pisati direktno unutar HTML datoteke ili u zasebnu *.css* datoteku. CSS sintaksa koda je jako jednostavna, jer koristi engleski jezik. Sintaksa se sastoji od selektora i deklaracijskog bloka. Pomoću selektora se određuje koji elementi se žele urediti. Selektor može biti ime elementa, ime klase, id elementa itd. Selektori se razlikuju po jačini. Ime elementa je najslabije, zatim ide ime klase, pa id elementa. Unutar deklaracijskog bloka se navode svojstva koja se žele dati elementu.

Primjer CSS sintakse za klasu:

```
.moja-klasa {
      color: blue;
}
```

Ukoliko se stilovi pišu unutar HTML datoteke koristi se ključna riječ *style*. Ona može biti korištena kao atribut unutar HTML oznake ili kao element unutar zaglavlja dokumenta. Ako se stilizira preko atributa, navedeni stilovi će se primijeniti samo za taj element. Primjer stiliziranja elementa preko atributa:

```
<p style="color: red;">Stilizirani odjeljak</p>
```

Ako se stilovi primjenjuju preko zaglavlja potrebno je koristiti <*style*> element, gdje se između oznaka piše CSS kod. S tim načinom se može uređivati više elemenata s jednog mjesta. Primjer stila iz zaglavlja:

```
<head>
      <style type="text/css">
            p {
                  color: red;
            }
      </style>
</head>
```

Ako se stilovi žele uključiti u HTML datoteku sa zasebnom CSS datotekom potrebno ih je povezati. Na ovaj način se izbjegava dupliranje koda dok se radi web mjesto. Izmjenu je potrebno raditi samo na jednom mjestu, a ne na svakom dokumentu gdje se prikazuje određeni element. Povezivanje dokumenata se radi sa <link> elementom i potrebnim atributima. Primjer:

```
<head>
      <link rel="stylesheet" type="text/css" href="stilovi.css">
</head>
```

CSS se također mijenjao kroz povijest i ima tri verzije: [14]

- **CSS 1** – verzija nastala 1996. godine. Kao što je navedeno nastao je kako bi se stranice mogle uljepšati i personalizirati. U početku je bilo problematično uopće koristiti CSS jer mnogi preglednici nisu uopće podržavali CSS.
- **CSS 2** – verzija nastala 1998. godine. U ovoj verziji su dodana nova svojstva, prvenstveno vezanja na pozicioniranje elementa.
- **CSS 3** – verzija nastala 1999. godine. Ova verzija se još naziva i modularna, jer joj se organizacija temelji na modulima. Moduli su proširili mogućnosti CSS-a u odnosu na prethodnu verziju. Iako postoji veliki broj tih modula samo četiri su preporučljiva za korištenje (boje, imenska mjesta, selektori i medijski upiti).

5.2.1. Sass

Sass (*Syntatically Awesome Stzle Sheets*) je CSS pre-procesor. Pre-procesori su manji programi koji procesiraju podatke tako da stvaraju podatke koji koriste drugi programi. Sa Sass-om se reducira ponavljanje CSS koda i samim time vrijeme potrebno za razvoj web stranica i aplikacija. Omogućava modularno razvijanje stilova i pomaže lakšem organiziranju stilova na velikim projektima. Za njega se još kaže kako je to CSS sa super moćima. [15]
Ono što odlikuje Sass je što je neovisan od verzije korištenog CSS, može se koristiti sa bilo kojom CSS knjižnicom (eng. *library).* Neki od najpoznatiji CSS programskih okvira (eng. *framework)* su izgrađeni sa Sass-om (*Compass, Bourbon, Susy* itd.). Jako je popularan i ima ogromnu zajednicu koji ga podržavaju i razvijaju.

Sass omogućuje stvari koje CSS nema. To su npr. varijable, ugnježđivanje, funkcije, miksini itd. Sass se može prepoznati po dvjema ekstenzijama, .scss i .sass. Razlika između te dvije ekstenzije je u sintaksi. Prva ekstenzija (.scss) je slična klasičnoj CSS verziji, dok .sass ekstenzija se razlikuje po tome što ne koristi zagrade za deklaracijske blokove već umetanje, i ne koristi točku zarez na kraju svakog svojstva.

Primjer .sass sintakse:

```
.moja-klasa
    height: 200px
    width: 50%
    margin: 0 auto

.moja-klasa2
    color: yellow
    font-size: 12px
```

Sass nudi mogućnost kreiranja varijabli u kojim se spremaju određene vrijednosti. Sa varijablama se olakšava kreiranje stilova, jer promjena na jednom mjestu će utjecati na sve ostale element na kojima se koristi. Varijabla se definira sa znakom $ nakon čega ide ime varijable i onda njezina vrijednost.

```
$primarna-boja: #ccc;
$sekundarna-boja: #eee;
```

Primjena varijabla je jako velika, gdje je najbolji primjer definiranje paleta boja koje će se koristiti na nekom velikom projektu. Osim boja mogu se definirati točke za širine ekrana, vrste slova itd. Za Sass je karakteristično da podržava i različite tipove vrijednosti. U Sass-u je moguće dodavati brojeve, stringove, boolean vrijednosti, polja, mape, liste itd.

Sa Sass-om je moguće koristit i logičke uvjete u kojima će se koristiti neki stilovi. To se postiže sa ključnim riječima @if i @else. Primjer:

```
@if $moja-varijabla == „crna“ {
    color: black;
} @else {
    color: white
}
```

Kako bi se što više smanjilo pisanje koda, Sass ima takozvane miksine koji omogućavaju definiranje stilova koji mogu biti korišteni kroz dokumente. Definiraju se sa ključnom riječi @mixin, nakon čega slijedi ime i nakon toga deklaracijski blok. Za korištenje miksina koristi se ključna riječ @include i nakon nje ime miksina. Miksinima se također mogu prosljeđivati i argumenti koji se koriste unutar deklaracijskog bloka. Primjer miksina:

```
@mixin reset($boja) {
    margin: 0;
    padding: 0;
    color: $boja;
}

@include reset(red);
```

Kako bi se ubrzalo pisanje koda, Sass je uveo ugnježđivanje. Ugnježđivanjem se omogućava pisanje klasa unutar deklaracijskog bloka, što znači da nije uvijek potrebno prepisivati neki naziv klasik kako bi se npr. dodao modifikator klase ili kreirao novi element koji nasljeđuje dio imena neke klase. Nasljeđivanje se koristi sa znakom &. Primjer nasljeđivanja:

```
.moja-klasa {
      color: green;
      font-size: 15px;

      &_modifikator {
            margin: 20px;
            padding: 0 20px;
      }
}
```

Iz ovoga primjera će pre-procesor kreirati dvije klase (moja-klasa i moja-klasa_modifikator), gdje se kako uz puno manje koda dobije ista stvar koju bi morali pisati u jednostavnom CSS-u.

Sass podržava i funkcije koje su sintaksom slične miksinima. Funkcije omogućavaju definirati kompleksne operacije prilikom definiranja vrijednosti nekih svojstava na određenim

elementima. Definiraju se sa ključnom riječi @function, nakon čega slijedi ime funkcije i opcionalno argumenti. Na ovaj način se opet smanjuje ponavljajući kod koji bi se dogodio da se stilovi pišu u jednostavnom CSS-u.

Primjer funkcije:

```
@function uduplaj($broj) {
      return $broj * 2;
}

.moja-klasa {
      width: uduplaj(12345);
}
```

Osim mogućnosti da se kreiraju funkcije, Sass ima gotove funkcije kojima se može npr. mijenjati tamnoća ili svjetlina neke boje, funkcije za minimalnu ili maksimalnu vrijednost i slično.

5.3. Vue.js

Vue.js je Javascript programski okvir (eng. *framework*) otvorenog koda (eng. *open-source*) za izgradnju korisničkih sučelja i jednostraničnih aplikacija (eng. *signle page appliations*). Odlikuje ga jednostavnost, lakoća integriranja sa drugim tehnologijama i mala veličina. Kreirao ga je Evan You 2014. godine, koji ga uz svoj tim i danas održava i unaprjeđuje. Baziran je na MVVM arhitekturi. Moguće ga je koristiti u projektu kroz CDN skriptu ili kao npm paket, ovisno o tipu projekta. Zadnja verzija je Vue3, no Vue2 je i dalje jako popularan i korišten. Radi na principu komponenti, što znači da se od manjih dijelova stvaraju kompleksnije komponente. Komponentu koja se poziva u drugoj komponenti se naziva dijete, dok komponenta koja poziva drugu komponentu se naziva roditelj. [16]
U Vue datoteci postoji HTML dio koji se naziva predložak, logički dio i prezentacijski dio. Ovakva struktura dokumenta omogućava imati sve vezano za komponentu na jednom mjestu no ovakva struktura može biti i problematična ukoliko se radi o velikoj komponenti, gdje je puno koda što otežava čitanje i održavanje komponente. U nastavku će biti objašnjena prva dva dijela, prezentacijski dio je objašnjen u sekciji za Sass.

5.3.1. Predlošci

Vue koristi sintaksu predložaka čija je osnova HTML. Ovakva sintaksa omogućava deklarativno spajanje DOM-a i podataka iz Vue logike. Element koji označava predložak je <template>. Ono što Vue radi sa predloškom je da ga pretvara u virtualni DOM što omogućava

Vue-u da nadogradi izgled zaslona sa minimalnim brojem promjena. Za povezivanje podataka sa DOM-om se koristi „Mustache“ sintaksa sa duplim znakom {{}}. Primjer:

```
<p>Moja poruka glasi: {{ poruka }}</p>
```

U primjeru iznad poruka unutar zagrada će se dinamički mijenjati ovisno o logici unutar komponente. Vue će je ponovno skicirati tek onda kada se njena vrijednost promjeni.

Vue u sebi sadrži direktive s kojima se može kontrolirati kako će se sadržaj prikazati. Direktivama je moguće selektivno prikazati neki sadržaj, proslijediti dinamičke vrijednosti HTML elementima, ubaciti dinamički HTML itd. Direktive u Vue-u počinju sa prefiksom **v-** nazivDirektive.

Umjesto sintakse sa zagradama moguće je koristiti direktivu v-text za dodavanje teksta elementu. Primjer:

```
<p>{{ poruka }}</p>
<!—isto kao -->
<p v-text="poruka"></p>
```

Za selektivno prezentiranje elemenata koriste se v-if i v-else direktive. Primjer:

```
<div v-if="broj === 1"></div>
<div v-else-if="broj === 2"></div>
<div v-else></div>
```

Kako bi se smanjio broj linija koda Vue je uveo i petlje kojima se prikazuju slični elementi. Na taj način kroz par linija koda se napravi isto što bi nam trebalo možda par stotina linija koda. Za petlju se koristi v-for direktiva. Bitna stavka korištenja v-for je dodavanje ključa svakom elementu, jer na osnovu toga ključa Vue zna o kojem se elementu radi. Tako ukoliko se element briše ili mijenja svoje podatke Vue zna točno o kojem elementu se radi i samo će na njemu izvesti promjene. V-for direktiva se može koristiti na podacima kroz koje se moguće iterirati (polja, liste, mape).

Primjer v-for direktive:

```
<ul>
      <li v-for="korisnik of korisnici" :key="korisnik.id">
            {{ korisnik.ime }}
      </li>
</ul>
```

Umjesto dohvaćanja elementa u logičkom dijelu i dodavanju mu akcije na uobičajen Javascript način, Vue omogućava dodavanje akcija direktno u predlošku. To se postiže sa prefiksom v-on. Nakon prefiksa ide događaj (eng. *event*) koji će okinuti akciju i poslije toga ide ime akcije. Skraćena verzija ove sintakse je umjesto v-on prefiksa dodaje se znak „@“. Primjer dodavanja akcija elementu:

```
<button v-on:click="dodajKorisnika">Klikni me</button>
<!—isto kao -->
<button @click="dodajKorisnika">Klikni me</button>
```

Također je moguće dodati modifikacije događajima da se npr. izvrše jednom, da se ne izvršava nativno ponašanje elementa i slično. Primjer:

```
<button @submit.prevent="dodajKorisnika">Dodaj korisnika</button>
```

5.3.2. Logički dio

Kao što je prethodno navedeno u radu Vue datoteke sadrže i logički dio koji se nalazi unutar <script> elementa. Unutar njega se nalaze podaci koji se koriste u predlošcima, metode koje se pozivaju, metode koje prate promjene, metode životnog ciklusa komponente i slično.
Primjer logičkog dijela:

```
<script>
      export default {
            name: „moja-komponenta“,
            props: [„prop1“, „prop2“],
            data: function() {
                  return {
                        data1: false,
                        data2: []
                  }
            },
            computed: { ... },
            watch: { ... },
            methods: { ... }
      }
</script>
```

U primjeru su navedeni neki od atributa koji se mogu nalaziti unutar Vue objekta. Atribut „name“ je atribut koji se koristi za imenovanje komponente te ono olakšava posao prilikom traženja grešaka, ukoliko ne postoji navedeni atribut Vue će sam dati ime komponenti tj. uzet će ime datoteke kao ime komponente.
„props“ atribut su svojstva, odnosno podaci koji se prosljeđuju od roditelja prema djetetu. Svojstvima se može definirati tip podatka, da li je svojstvo obavezno ili ne, njegovu standardnu

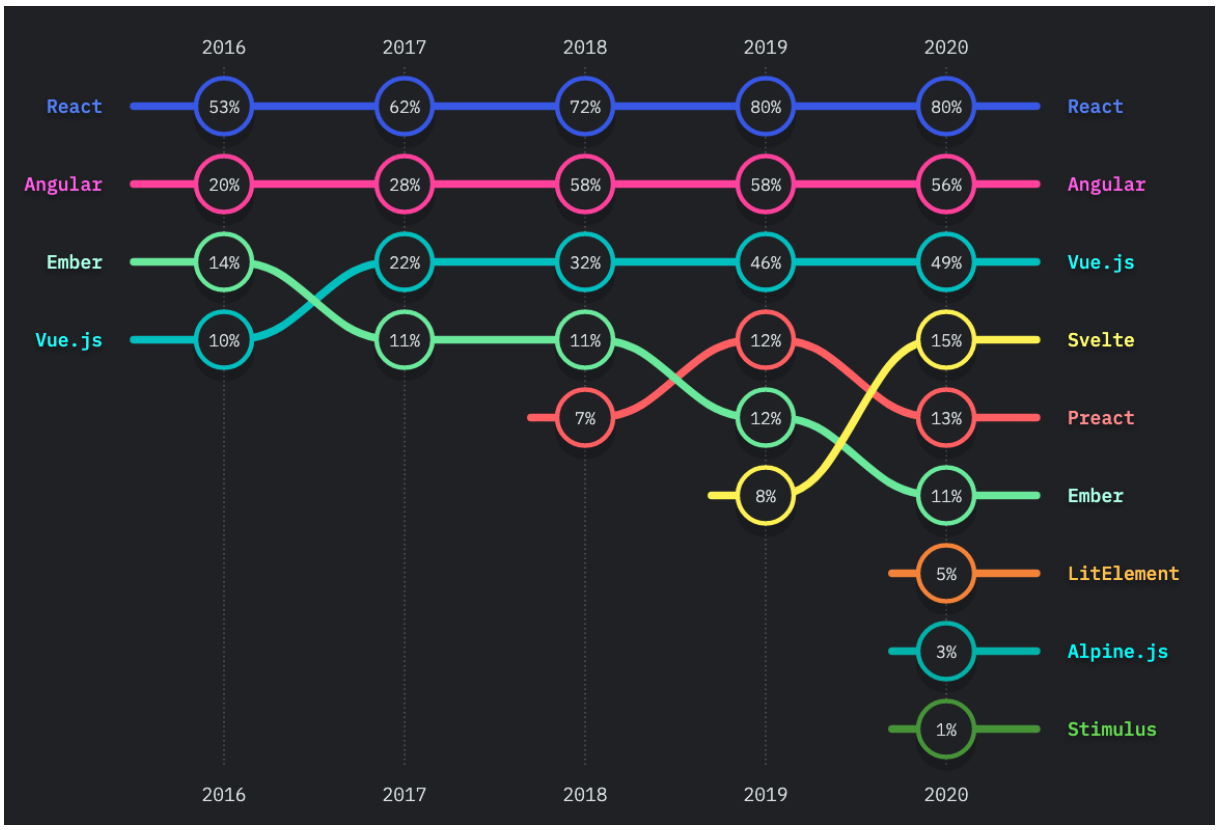
vrijednost i sl. Svojstva unutar djece nije moguće modificirati, za to je potrebno pozivati metodu koja je proslijeđena od roditelja prema djetetu i koristiti ključnu riječ „$emit“.
„data“ atribut je funkcija koja vraća objekt sa podacima koje će komponenta koristiti lokalno ili ih prosljeđivati djeci. Treba napomenuti da ovaj atribut ne mora biti funkcija već objekt sa podacima, u tom slučaju podaci koji se nalaze unutar komponente će biti korišteni za sve komponente sa istim nazivom. Svaka promjena za podatke unutar „data“ atributa će izazvati ažuriranje komponente na korisnikovom zaslonu.
„computed“ atribut je atribut također za podatke, no razlika je u tome što se podaci spremljeni u „computed“ spremaju u cache memoriju i izazivaju ažuriranje prikaza komponente samo u slučaju njihove promjene. Korištenje ovakvih podataka pridonosi brzini i performansama stranice/aplikacije.
„watch“ atribut je atribut koji služi za slušanje promjena unutar komponente. Ukoliko se prosljeđeni podatak iz „data“ ili „computed“ atributa promijeni, pozvati će se logika unutar „watch“ atributa.
U atribut „methods“ se spremaju metode koje se koriste kroz komponentu, bilo u predlošku komponente ili njenom logičkom dijelu.

Osim navedenih atributa postoje još i metode životnog ciklusa komponente. Svaka Vue instanca prolazi kroz nekoliko koraka: kreiranja (eng. *created*), montiranja (eng. mounted), ažuriranje (eng. *updated*) i uništavanja (eng. *destroyed*). Ove metode se koriste kada se želi izvršiti neka radnja u specifičnom trenutku života komponente npr. dohvatiti podatke sa servera prilikom kreiranja komponente i slično. Metode životnog ciklusa su: [17]

- **beforeCreate** – poziva se odmah nakon što se instanca inicijalizirala
- **created** – poziva se nakon što se instanca kreirala. U ovoj fazi su postavljeni podaci i njihovo promatranje.
- **beforeMount** – funkcija koja se poziva prije prve „render“ funkcije. Render funkcija obavlja radnje u pozadini koje pretvaraju sadržaj iz predloška i logike u ono što klijent vidi na zaslonu.
- **mounted –** funkcija koja se poziva nakon što se instanca montirala. Nakon montiranja moguće je pristupiti podacima komponente. Ova funkcija ne garantira da su se montirale djeca komponente.
- **beforeUpdate –** poziva se prije nego se podaci promijene, odnosno prije nego je DOM ažuriran
- **updated –** poziva se nakon što su se podaci promijenili
- **beforeDestroy –** poziva se prije nego što će instanca biti uništena
- **destroyed –** poziva se nakon što je komponenta uništena. Kad se ova metoda pozove više nije moguće pristupiti djeci i podacima komponente

5.3.3. Usporedbe s drugim programskim okvirima

Trenutno u svijetu razvoja na korisničkoj strani postoji jako veliki broj programskih okvira za razvoj sučelja. Prema podacima sa StateOfJS [18], koja prati trendove u razvoju na korisničkoj strani, najpoznatiji i najkorišteniji od njih su React koji je kreirao Facebook, Angular koji je kreirao Google i Vue koji je nezavisan. U nastavku će biti opisane sličnosti i razlike Vue-a s druga dva navedena programska okvira.



Slika 16: Popularnost programskih okvira

(preuzeto sa https://2020.stateofjs.com/en-US/technologies/front-end-frameworks, na dan 01.09.2021.)

Veličina Vue-a (oko 30KB) je manja u odnosu na Angular (oko 65KB) i React (oko 40KB). Vue kao i React je jako fleksibilan programski okvir, koji se može prilagoditi gotovo svakoj strukturi aplikacije, dok kod Angulara to nije slučaj. Angular npr. zahtjeva korištenje TypeScript jezika, dok je kod Vue-a korištenje TypeScripta opcionalno.

Razlike u brzini između ovih programskih okvira su neznatne. Valja napomenuti da kod React-a ukoliko dođe do promjene podataka unutar instance se ažurira cijela komponenta sa svojom djecom. To se može spriječiti sa dodatnim metodama, gdje Vue to radi na pametan

način, gdje automatski, bez dodatnih metoda, ažurira samo one dijelove koji su se promijenili. To uvelike poboljšava performanse aplikacije. I React i Vue koriste virtualni DOM.

React koristi JSX sintaksu koja je slična XML sintaksi, ali koja funkcionira sa Javascript jezikom. Takva sintaksa omogućava iskoristiti punu moć Javascript jezika prilikom izgradnje korisničkog sučelja. Za razliku od React-a Vue koristi predloške koji možda ne mogu iskoristiti puni potencijal Javascript jezika, ali je sintaksa puno sličnija HTML jeziku što ga čini puno pristupačnijim za koristiti.

Velika prednost React programskog okvira je njegova popularnost. On je trenutno najpopularniji i najkorišteniji programski okvir te iz tog razloga postoji jako puno knjižnica (eng. *libraries*) koje podržavaju React i olakšavaju njegovo korištenje. Uz to React je osnovan od strane Facebooka te ima veliku podršku. Kod Vue-a to nije slučaj, jer je to programski okvir iza kojeg ne stoji ni jedna velika korporacija i još nije toliko popularan kao React, stoga ne postoji ni toliko knjižnica koje olakšavaju rad s njim.

Vue kao programski okvir je jako pristupačan i jednostavan za naučiti. Potrebno je poznavati osnove HTML i Javascript jezik da bi se izgradile jednostavne stranice/aplikacije. Kod Angular-a to nije slučaj i krivulja učenja je jako strma. Angular sadrži mnoge API-je koje je potrebno poznavati da bi se mogao koristi, te uz to zahtjeva poznavanje TypeScript-a. [19]

5.4. Firebase

Firebase je Google-ova platforma za kreiranje mobilnih i web aplikacija. [20] U početku Firebase je bila nezavisna kompanija osnovana 2011. godine. Osnovali su je James Tamplin i Andrew Lee pod nazivom Envolve. Envolve se bavila sa pružanjem API-ja programerima za integriranje komunikacije u stvarnom vremenu na web stranicama. Iako je to bila ideja kompanije, programeri su Envolve koristili na drugi način. Envolve je korišten kako bi se potrebni podaci usklađivali u stvarnom vremenu za npr. igre. Nakon što su Tamplin i Lee vidjeli što se događa odlučili su odvojiti arhitekturu i API jedno od drugog. Nakon odvajanja je nastao Firebase, koji je Google otkupio 2014. godine.

Danas Firebase pruža mnogo usluga, koje uvelike olakšavaju posao programerima. Neke od usluga Firebase su:

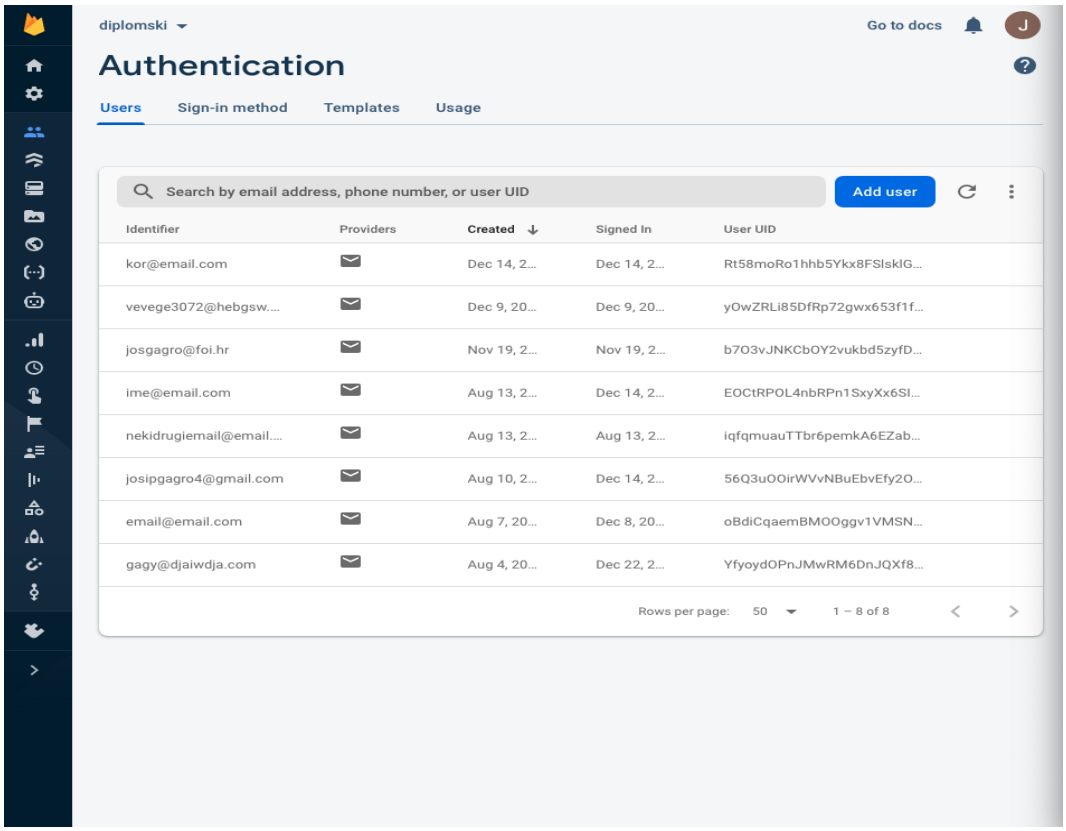
- Baza podataka u stvarnom vremenu (eng. *Real time database*) – omogućava kreiranje podataka bez servera (eng. *serverless*) gdje se podaci spremaju kao JSON
- Firestore – podaci se spremaju u oblak (eng. *cloud*) kao JSON
- Autentikacija – autentikacija korisnika za prijavu unutar aplikacije
- Poruke u oblaku – mogućnost slanja obavijesti iz oblaka

U nastavku će biti opisani Autentikacija i Firestore, jer su korišteni za implementaciju praktičnog dijela aplikacije.

5.4.3. Autentikacija

Firebase pruža svoju uslugu autentikacije, što znači da korisnici se mogu prijaviti, odjaviti u aplikaciju, obnoviti šifru ili povezati više računa s jednim. Sve ove radnje su izazov za implementaciju i sigurnost same aplikacije. Firebase autentikacije omogućava integriranje autentikacije unutar aplikacije jednostavno i brzo. Sa Firebase autentikacijom programeri i vlasnici aplikacija mogu biti sigurni da koriste najmodernije pristupe za autentikaciju, što vašu aplikaciju čini jako sigurnom i otpornom na različite vrste napada. [21]

Firebase autentikaciju, kao i ostale usluge je moguće koristiti preko njihovog web sučelja. Za Firebase autentikaciju je moguće preko sučelja upravljati korisnicima, odabrati metode za prijavu unutar aplikacije, kreirati predloške za email obnovu lozinke, provjeravati statistiku prijava korisnika i sl.



Slika 17: Primjer popisa registriranih korisnika aplikacije u Firebase-u

Firebase autentikacija je bliska ostalim Firebase tehnologijama, ali ne isključuje povezivanje ove autentikacije sa vlastitom logikom na poslužiteljskoj strani aplikacije (eng. *backend*). Moguće ju je koristiti sa više različitih tehnologija i različitih platforma, od iOS, Android-a do web tehnologija.

Nakon autentikacije, odnosno prijave korisnika u aplikaciju stvara se token koji se vraća od Firebase-a prema aplikaciji koji sadrži sve podatke o identitetu korisnika. Prijavljeni korisnik onda može čitati i pisati podatke iz baze podataka, te koristiti ostale značajke aplikacije.

Autentikaciju je jako jednostavno uključiti unutar aplikacije, gdje se unutar Firebase konfiguracije pozove komanda:

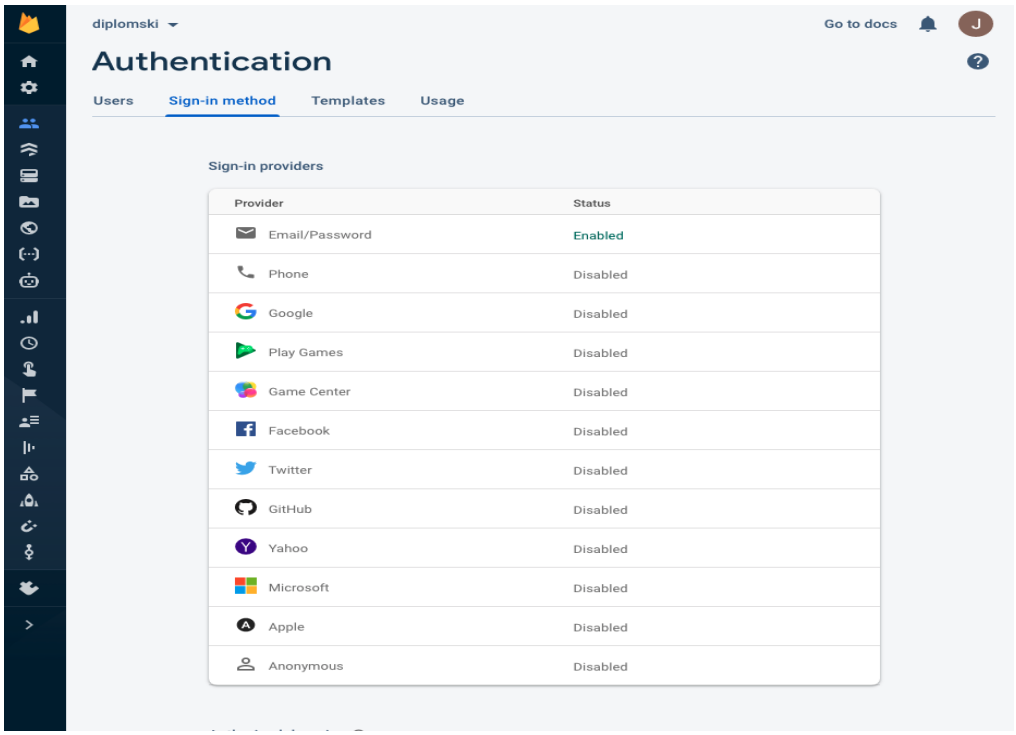
```
const auth = firebase.auth();
```

Progamer dalje iz varijable auth može pozivati metode autentikacije kao što su:

- createUserWithEmailAndPassword – kreira korisnika sa email adresom i lozinkom
- signInWithEmailAndPassword – prijavljuje korisnika sa email adresom i lozinkom
- signOut – odjavljuje korisnika
- sendPasswordResetEmail – pošalje email korisniku za obnovu lozinke
- updateEmail – ažurira email adresu
- updatePassword – ažurira lozinku

Prijavu, odnosno autentikaciju moguće je izvršiti sa:

- Email računom i lozinkom
- SMS autentikacija
- Google računom
- Facebook računom
- Twitter računom
- GitHub računom



Slika 18: Lista svih mogućnosti za autentikaciju unutar Firebase-a

5.4.4. Firestore

Firestore je NoSQL baza podataka pohranjena na oblaku na kojem se pohranjuju i sinkroniziraju podaci za razvoj aplikacija na korisničkoj i poslužiteljskoj strani. [22] Firestore kao i ostale Firebase usluge se može integrirati sa Google alatima te raznim jezicima kao što su:

- Java
- Python
- Node.js
- C++
- Go

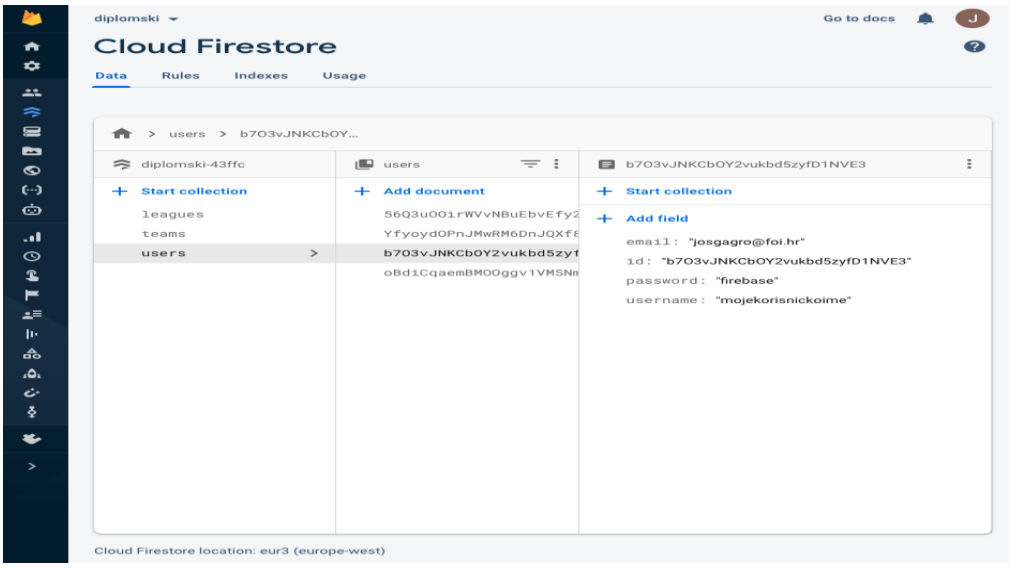
Ono što odlikuje Firestore je fleksibilnost, odnosno njegov model podataka. Podaci se pohranjuju u dokumente koji se pohranjuju u kolekcije. Kolekcije služe kao kontejneri za

dokumente, dok dokumenti služe za zapisivanje i organiziranje podataka. Dokumenti podržavaju razne tipove podataka, od jednostavnih kao što su brojevi i riječi (string), pa sve do kompliciranih objekata. Struktura dokumenata je takva da i oni mogu sadržavati svoje kolekcije, odnosno podkolekcije.

Primjer jedne strukture Firestore kolekcije:

- Studenti (kolekcija)
    - student_1 (dokument)
        - ime (podatak)
        - prezime (podatak)
        - datum_rođenja (podatak)
        - …
        - hobiji (podkolekcija)
            - hobij_1 (dokument)
                - podatak_1 (podatak)
                - …

Podaci se dohvaćaju upitima, kojima se mogu dohvatiti zasebni dokumenti ili cijela kolekcija. Pomoću upita također je moguće filtrirati, sortirati podatke ili postaviti limit koliko podataka se želi dohvatiti. Performanse upita u Firestore-u ovise o broju dobivenih setova rezultata, a ne o samim podacima rezultata. Sa Firestore-om njegovi korisnici mogu biti sigurni da dobivaju jaku infrastrukturu koju podržava Google, konzistenciju i mogućnost obrađivanja velikog broja podataka i upita koji dolaze od strane aplikacije.



Slika 19: Primjer zapisa u Firestore-u

Za rad sa Firestore-om i njegovom integracijom u aplikaciji potrebno je prvo kreirati projekt unutar Firebase konzole. Nakon kreiranja potrebno je povezati aplikaciju s Firestore-om. Za web aplikacije to je moguće preko skripte ili preko npm paketa. Prednosti povezivanja putem paketa je ta što je moguće uključiti samo one stvari koje su potrebne unutar aplikacije, dok sa skriptom se uključuje sve inačice Firestore-a. U nastavku će biti objašnjeno kako je Firestore povezan sa aplikacijom ovog diplomskog rada, te kako je korišten Firestore.

Kada se aplikacija poveže sa Firestore-om potrebno je kreirati njegovu instancu kako bi se dohvaćale metode iz objekta, s kojima se kasnije upravlja podacima. Da bi se aplikacija inicijalizirala potrebno je prosljediti API ključ koji je generirao Firebase za kreirani projekt, autentikacijsku domenu i identifikacijski broj projekta. Ovi podaci su dostupni unutar projektne konfiguracije unutar Firebase-a, gdje je Firebase čak i pripremio sav kod koji treba aplikaciji, te ga je potrebno samo kopirati i zalijepiti unutar aplikacije.

Kod za inicijalizaciju Firebase-a unutar web aplikacije:

```
import { initializeApp } from "firebase/app";

const firebaseConfig = {
  apiKey: "AIzaSyBH8TzUxK76zm8mFgVCTz4DwNN2f8N8z7w",
  authDomain: "diplomski-43ffc.firebaseapp.com",
  databaseURL: "https://diplomski-43ffc.firebaseio.com",
  projectId: "diplomski-43ffc",
  storageBucket: "diplomski-43ffc.appspot.com",
  messagingSenderId: "1040606113704",
  appId: "1:1040606113704:web:5f6ff3fd70638d5075f79d"
};

const app = initializeApp(firebaseConfig);
```

Prilikom kreiranja je moguće također osigurati podatke, odnosno konfigurirati tko može pisati i čitati podatke, te na kojim dokumentima može pisati i čitati podatke unutar Firestore-a.

Primjer koda koji dozvoljava pisanje i čitanje unutar baze podataka do kraja 2021. godine:

```
rules_version = '2';
service cloud.firestore {
  match /databases/{database}/documents {
    match /{document=**} {
      allow read, write: if request.time < timestamp.date(2021, 12, 31);
    }
  }
}
```

6. Implementacija aplikacije

6.1. Priprema projekta

Kako bi se implementirala prethodno opisana aplikacija potrebno je kreirati i konfigurirati projekt. U nastavku će biti opisano kako je postavljen projekt i proces instalacije i konfiguracije projekta. Pošto su korištene tehnologije prethodno opisane u ovom dijelu će biti opisana samo njihova instalacija, konfiguracija i struktura projekta.

Kako bi bilo moguće kreirati projekt potrebno je imati Node.js instaliran jer preko njega se koristi alat *npm* koji je potreban za instalaciju paketa, odnosno alata za implementaciju i realizaciju praktičnog dijela diplomskog rada.

Da bi se Vue instalirao potrebno je u terminal upisati:

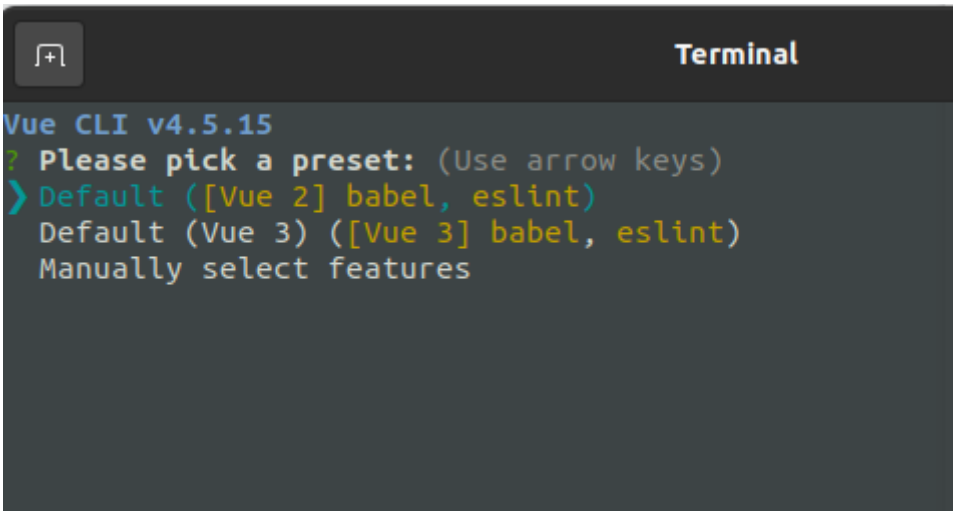
```
npm install -g @vue/cli @vue/cli-service-global
```

Sa ovom naredbom se instalira Vue sučelje naredbi (eng. *CLI - Client Line Interface*) pomoću kojeg korisnik putem terminala može izvršavati mogućnosti Vue-a, kao što je npr. kreirati projekt, pokrenuti projekt, izgraditi (eng. *build*) projekt i slično.

Nakon što se paket instalirao potrebno je kreirati projekt. Vue projekt se kreira sa naredbom:

```
vue create naziv-projekta
```

Kad se upiše spomenuta naredba otvori se spomenuti CLI. Na početku se otvara izbornik gdje se može odabrati već predefinirane instalacije ili kreirati svoju instalaciju sa određenim značajkama. U sklopu ovog diplomskog rada je uzeta zadnja opcija.

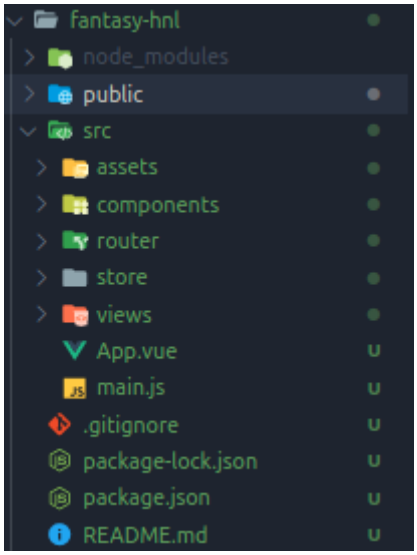


Slika 20: Izbornik Vue instalacije

Kada se odabere opcija za ručno biranje značajki otvara se izbornik alata koji se želi instalirati. U sklopu ovoga rada su odabrane opcije:

- Vue 2 - korištena verzija unutar diplomskog rada
- Vue usmjerivača (eng. *router*) - biblioteka koji se koristi za putanje unutar aplikacije
- Vuex - biblioteka za praćenje globalnog stanja aplikacije
- Sass - CSS pre-procesor

Osim ovih biblioteka instalirana je dodatno i biblioteka Vuelidate, koja se koristi za validaciju formi unutar aplikacije. Nakon toga CLI nudi mogućnost spremanja svih odabranih opcija kao predložak za buduće projekte, te nakon toga kreće instalacija svega odabranog. Kada se sve instalira projekt je spreman za rad sa Vue-om. CLI kreira predložak i strukturu datoteka koji može olakšati prilikom početka rada na projektu.



Slika 21: Struktura datoteka nakon instalacije Vue-a

U mapi *node_modules* se smještaju alati i paketi koji se instaliraju putem npm-a. Mapa *public* je mapa u koju se spremaju kompiliraju datoteke. Ta mapa se koristi za produkciju, odnosno ide na poslužitelj. Kompiliranje se izvodi naredbom:

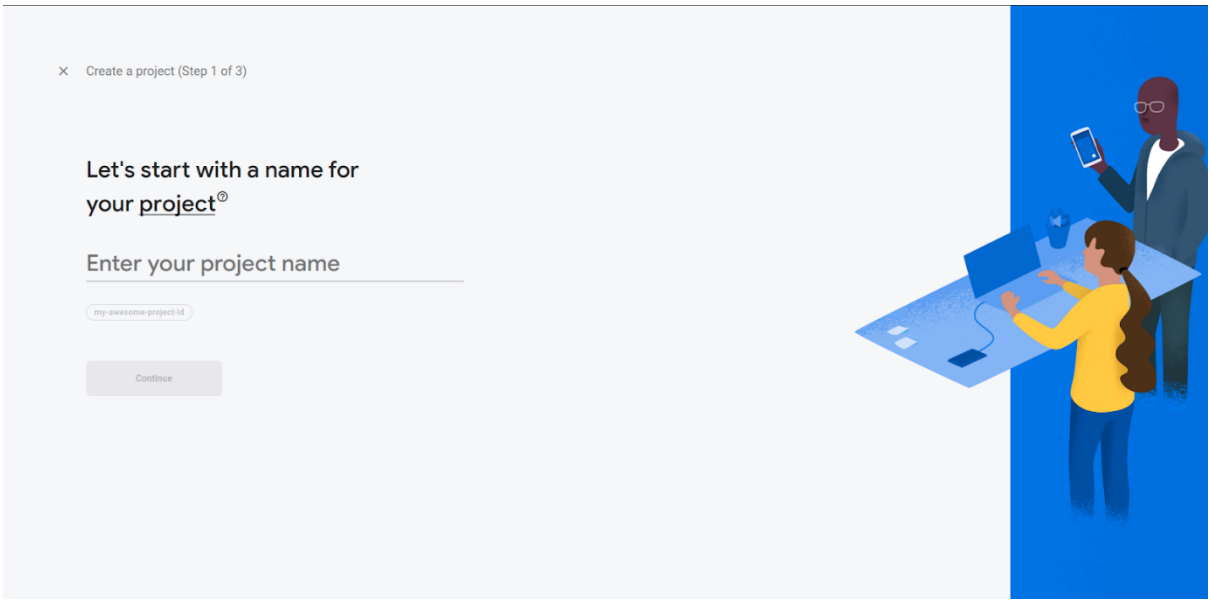
```
npm run build
```

U mapi *src* su datoteke za lokalni rad. U njoj se nalaze slike, tipovi slova, stilovi, kreirane komponente, logika za putanje, globalno stanje aplikacije, prikazi i sve ostalo potrebno za implementaciju i logiku aplikacije. Za izdvojiti su datoteke *main.js* i *App.vue. Main.js* služi za inicijalizaciju Vue instance i uključivanja ostalih alata u aplikaciju, dok je *App.vue* glavna

komponenta za sve ostale komponente, što znači da se unutar te datoteke pozivaju ostale Vue datoteke, odnosno manje komponente.
Za pokretanje Vue aplikacije lokalno koristi se komanda:

```
npm run serve
```

S ovom komandom se pokrene lokalni poslužitelj koji pokreće aplikaciju i moguće joj je pristupiti na određenim vratima (eng. *port*). Obično je to vrijednost 8080, no moguće ju je konfigurirati.
Nakon kreiranja Vue projekta potrebno je kreirati Firebase projekt i uključiti ga unutar aplikacije. Firebase projekt se kreira na Firebase stranici, odnosno u Firebase konzoli. Prilikom kreiranja potrebno je dati ime projektu te odabrati opcije koje se želi uključiti unutar projekta, npr. Google analitike i sl.



Slika 22: Kreiranje projekta u Firebase-u

Nakon što se projekt kreira potrebno je instalirati Firebase lokalno. Instalacija se vrši također preko npm-a sa komandom:

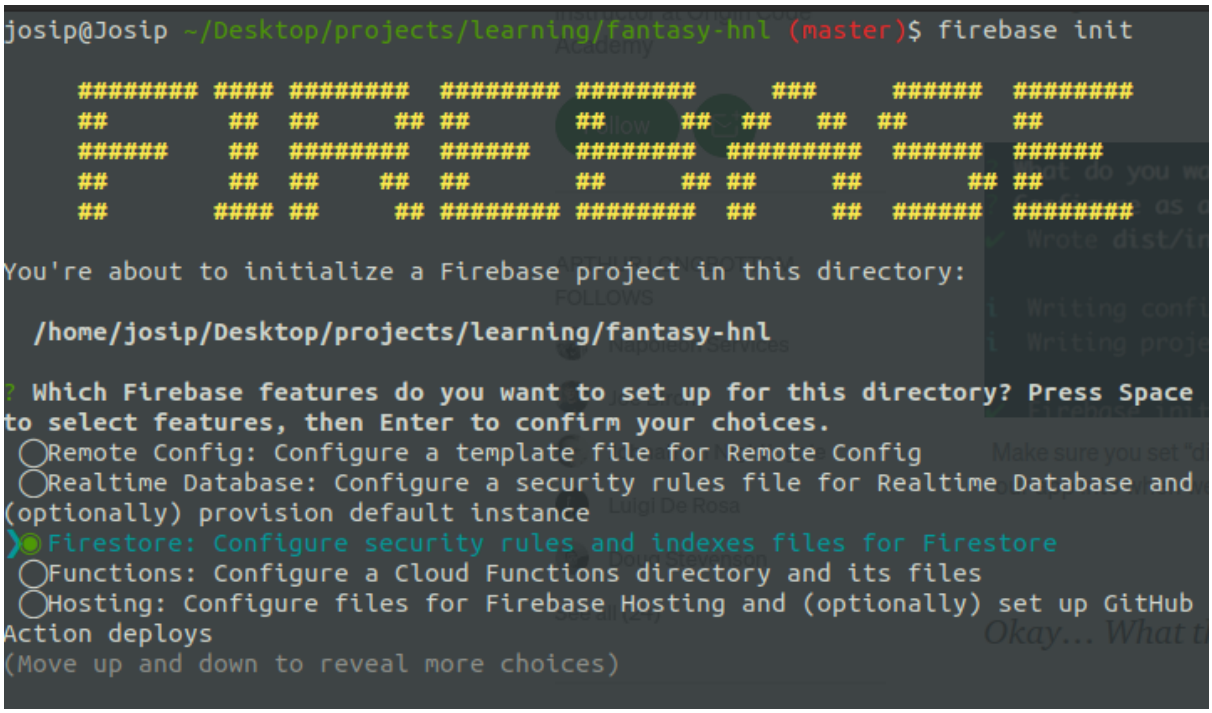
```
npm install firebase
```

Osim Firebase potrebno je instalirati i njegov CLI kako bi se mogle koristiti njegove mogućnosti iz terminala, kao što je povezivanje računa i povezivanje projekata. Za to se koristi naredba:

```
npm install -g firebase-tools
```

Nakon instalacije potrebno se prijaviti sa svojim računom i inicijalizirati Firebase projekt. To se radi sa naredbom:

```
firebase init
```



Slika 23: Instalacija Firebase-a

Slično kao i prilikom instalacije Vue-a otvara se izbornik gdje korisnik može izabrati što se sve želi uključiti u aplikaciju te nakon toga mogućnost kreiranja novog ili povezivanja već kreiranog Firebase projekta. Kada se projekt inicijalizira potrebno je konfigurirati Firebase lokalno i uključiti ga u aplikaciju. Konfiguraciju je jednostavno postaviti jer prilikom kreiranja projekta Firebase sam generira kod koji je potrebno uključiti u konfiguraciju. Kod se može pronaći unutar konzole projekta pod postavkama projekta.

Konfiguracija aplikacije diplomskog rada je smještena u datoteci *firebaseConfig.js* koja se kasnije uključuje u *main.js*. Osim dodavanja konfiguracije potrebno je i inicijalizirati Firebase instancu te uključiti Firestore i Autentikaciju. Iz ove datoteke se također izvoze kolekcije iz baze podataka kako bi bile dostupne unutar ostalih datoteka.

Kod konfiguracije Firebase-a:

```
import firebase from 'firebase';
import 'firebase/firestore';

var firebaseConfig = {
  apiKey: "AIzaSyBH8TzUxK76zm8mFgVCTz4DwNN2f8N8z7w",
  authDomain: "diplomski-43ffc.firebaseapp.com",
  databaseURL: "https://diplomski-43ffc.firebaseio.com",
  projectId: "diplomski-43ffc",
```

```
  storageBucket: "diplomski-43ffc.appspot.com",
  messagingSenderId: "1040606113704",
  appId: "1:1040606113704:web:5f6ff3fd70638d5075f79d"
};

firebase.initializeApp(firebaseConfig);

const db = firebase.firestore();
db.settings({
  host: 'localhost:5004',
  ssl: false
});
const auth = firebase.auth();
auth.useEmulator('http://localhost:9099');
const trenutniKorisnik = auth.currentUser;

const kolekcijaKorisnika = db.collection('korisnici');
const kolekcijaTimova = db.collection('timovi');
const kolekcijaLiga = db.collection('lige');
const kolekcijaKola = db.collection('kola');

export default {
  firebase,
  db,
  auth,
  trenutniKorisnik,
  kolekcijaKorisnika,
  kolekcijaTimova,
  kolekcijaLiga,
  kolekcijaKola
};
```

Kod *main.js* datoteke:

```
import Vue from 'vue';
import App from '@/App.vue';
import FormGroup from '@/components/FormGroup.vue'
import router from '@/router';
import store from '@/store/store';
import firebase from '@/firebaseConfig';
import Vuelidate from 'vuelidate';
import '@/assets/Styles/app.scss';
import '@/filters/index';

Vue.use(Vuelidate);
Vue.config.productionTip = false;
Vue.config.devtools = true;

Vue.component('FormGroup', FormGroup);

let app;
firebase.auth.onAuthStateChanged(korisnik => {
      if (!app) {
            app = new Vue({
                  el: '#app',
                  router,
                  store,
                  render: h => h(App)
            });

            if (!korisnik) {
```

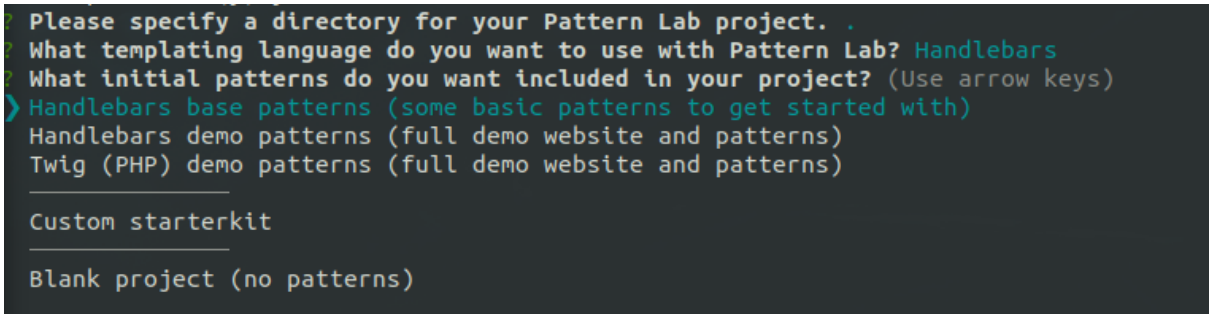
```
                window.sessionStorage.removeItem('hnlUser');
            }
        }
});
```

U datoteci se može vidjeti kako se uključuju prethodno navedene tehnologije. Nakon uključivanja Vue-u se prosljeđuje Vuelidate i na ovaj način je dostupan kroz cijelu Vue aplikaciju umjesto da se uključuje posebno u svaku komponentu. Nakon toga se inicijalizira aplikacija, ako nije već inicijalizirana, te ukoliko se promijeni status kod autentikacije brise se korisnik iz sesije.

Sljedeće što je potrebno instalirati je Patternlab. Za Patternlab jer kreirana nova mapa u koju se sprema instalacija. Kako bi se kreirao Patternlab potrebno je upisati naredbu:

```
npm create pattern-lab
```

Kada se upiše i potvrdi naredba otvara se izbornik u kojem je moguće izabrati koji jezik za predloške se želi koristiti, koje inicijalne predloške se želi uključiti te u koju mapu se Patternlab želi uključiti, ukoliko se ne želi uključiti u trenutnu mapu. Pošto je prethodno kreirana Patternlab mapa u ovom slučaju ostaje se u trenutnoj mapi.



Slika 24: Instalacija Patternlaba-a

6.2. Baza podataka

Kako je prethodno u radu navedeno baza podataka je napravljena u Firestore-u. Baza podataka se sastoji od četiri kolekcije:

- kolekcija liga
- kolekcija kola
- korisnika
- kolekcija ekipa

U kolekciju liga se spremaju kreirane lige u kojima se korisnici mogu natjecati. Svakoj ligi se pristupa preko jedinstvenog identifikacijskog ključa. Liga se sastoji od sljedećih atributa:

- *vrijemeDrafta* - vrijeme početka odabira igrača
- *ime* - naziv lige
- *datumDrafta* - datum početka odabira igrača
- *maxBroj*- maksimalan broj igrača koji može pristupiti ligi
- *korisnici* - lista igrača koji sudjeluju u ligi

U kolekciju kola se spremaju informacije o svakom odigranom kolu. Svako kolo sadrži podkolekciju utakmica, u koju se spremaju dokumenti, odnosno podaci za svaku odigranu utakmicu toga kola. Utakmica se sastoji od sljedećih atributa:

- *domaciTim* - domaća ekipa
- *gostujuciTim* - gostujuća ekipa
- *rezultat* - rezultat utakmice
- *datumUtakmice* - datum utakmice
- *vrijemeUtakmice* - vrijeme početka utakmice
- ***igraci*** - kolekcija igrača koja je igrala utakmicu, spremljeni atributi za igrača:
    - *tim_id* - identifikacijska oznaka ekipe
    - *id* - identifikacijska oznaka igrača
    - *asistencije* - broj asistencija na utakmici
    - *golovi* - broj golova na utakmici
    - *zutiKarton* - broj žutih kartona na utakmici
    - *crveniKarton* - broj crvenih kartona na utakmici
    - *bodovi* - broj bodova na utakmici

U kolekciju ekipa su spremljene ekipe koje se natječu unutar 1. HNL lige. Sastoji se od 10 ekipa. Kolekcija se sastoji od četiri atributa i pod kolekcije igrača. Popis atributa:

- *id* - identifikacijska oznaka ekipe
- *logo* - logo ekipe
- *ime* - ime ekipe
- ***igraci*** - kolekcija igrača ekipe, spremljeni atributi za igrače:
    - *id* - identifikacijska oznaka igrača
    - *tim_id* - identifikacijska oznaka ekipe igrača
    - *ime* - ime igrača
    - *pozicija* - pozicija igrača
    - *cijena* - vrijednost igrača
    - *odigraoUtakmica* - ukupan broj odigranih utakmica
    - *golovi* - ukupan broj golova
    - *asistencije* - ukupan broj asistencija

    - *zutiKartoni* - ukupan broj žutih kartona
    - *crveniKartoni* - ukupan broj crvenih kartona
    - *bodovi* - ukupan broj bodova

U kolekciju korisnika spremaju se registrirani korisnici aplikacije. Atributi kolekcije su:

- *id* - identifikacijska oznaka korisnika
- *email* - mail adresa korisnika
- *korisnickoIme* - korisničko ime korisnika
- *lozinka* - lozinka korisnika
- *imgUrl* - logo korisnika
- *tip* - vrsta korisnika
- ***lige*** - podkolekcija liga u kojima korisnik sudjeluje:
    - *liga_id* - identifikacijska oznaka lige
    - *budzet* - budžet s kojim igrač raspolaže za odabir sastava ekipe
    - *tokeni* - broj tokena za iskoristiti
    - *iskoristeniTokeni* - lista kola u kojima je iskorišten token
    - ***igraci*** - podkolekcija igrača, atributi:
        - *id* - identifikacijska oznaka igrača
        - tim_id - identifikacijska oznaka ekipe
        - ***kola*** - kolekcija kola
            - *kolo_id* - identifikacijska oznaka kola
            - *naKlupi* - informacija da li je igrač bio na klupi određeno kolo

Za pristupanje podacima koristi se instanca baze podataka iz konfiguracijske datoteke Firebase-a. Kolekcije se izvoze iz te datoteke i dostupne su kroz cijeli projekt. Izvoz kolekcija:

```
const kolekcijaKorisnika = db.collection('korisnici');
const kolekcijaTimova = db.collection('timovi');
const kolekcijaLiga = db.collection('lige');
const kolekcijaKola = db.collection('kola');

export default {
  firebase,
  db,
  auth,
  trenutniKorisnik,
  kolekcijaKorisnika,
  kolekcijaTimova,
  kolekcijaLiga,
  kolekcijaKola
};
```

Primjer dohvaćanja kolekcije igrača korisnika:

```
const korisnikoviIgraci = await
firebase.kolekcijaKorisnika.doc(korisnik.uid).collection('igraci').get();
```

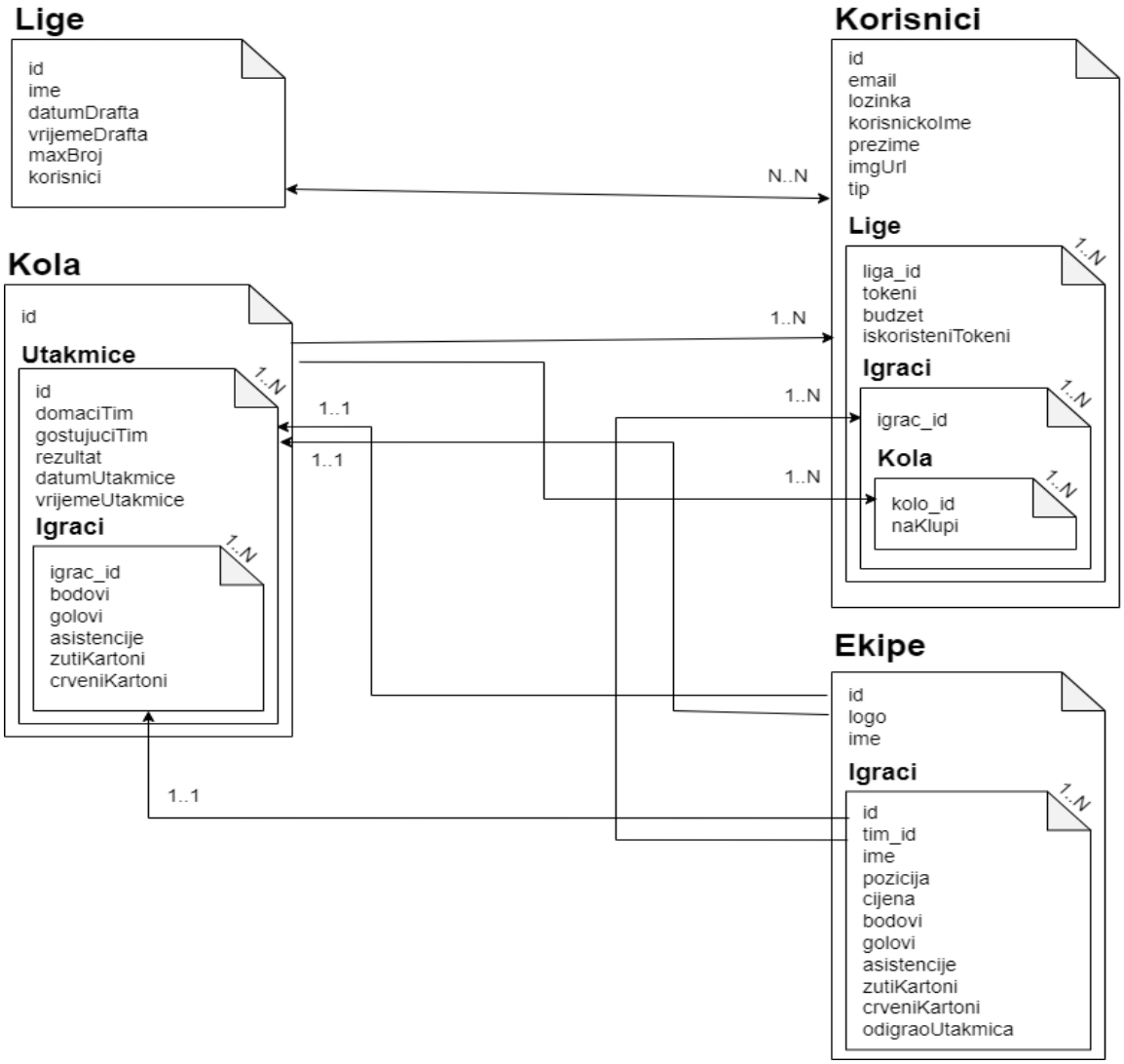
Primjer dohvaćanja dokumenta:

```
const korisnikoveInformacije = await
firebase.kolekcijaKorisnika.doc(korisnik).get();
```

Primjer spremanja podataka:

```
await firebase.kolekcijaKorisnika.doc(korisnik.uid).set({
    id: user.uid,
    korisnickoIme: this.korisnickoIme,
    email: this.email,
    lozinka: this.lozinka,
    budzet: 100,
    tip: 1
});
```

Već je prethodno navedeno da je Firebase NoSQL baza, koja pripada dokumentno orijentiranim bazama podataka (eng. *document oriented database*). Kako i ime govori baza podataka se bazira na dokumentima i kolekcijama, gdje su glavni formati za spremanje u obliku JSON-a, BSON-a ili XML-a. Dokumenti se spremaju u kolekcije, gdje se dokumenti mogu međusobno referencirati. Za razliku od relacijskih baza podataka koji imaju definiranu strukturu podataka i relacija između tablica, čiji se model opisuje ER dijagramom, kod NoSQL baze podataka fleksibilnost, koju ova baza podataka omogućava, stvara problem prilikom modeliranja. Model baze podataka projekta koji je prikazan na slici 25. je utemeljen na prijedlozima H.Vere, gdje se navodi da se koncepti o normalizaciji odbacuju zbog toga što se radi o ne relacijskog bazi podataka, dok su koncepti kardinalnosti između objekata (dokumenata) slični onima kod relacijskih baza podataka. [23]



Slika 25: Model baze podataka

6.3. Patternlab

Kao što je prethodno navedeno u radu za implementacijski dio sustava dizajna korišten je Patternlab. Također, prethodno je objašnjena njegova instalacija, te će se ova sekcija više bazirati na samu strukturu sustav dizajna i način na koji je on implementiran.
Sustav dizajna za aplikaciju 1. HNL Fantasy Liga se sastoji od:

- atoma
- molekula
- organizama
- predložaka
- stranica

Mapa (eng. *folder*) *source* je glavna mapa za rad sa Patternlab-om. Svi dijelovi sustava dizajna su smješteni u *_patterns* mapu koja je dio *source* mape. Unutar *source* mape nalazi se mapa *_data* u koju se spremaju *.json* datoteke i služe za sadržaj koji se želi prikazivati globalno, kako se ne bi svaki put kreirao isti sadržaj na različitim dijelovima sustava dizajna. U toj mapi se nalaze datoteke za sadržaj za gumbe, forme, tekstualne elemente i slično. U mapi *_meta* se nalaze datoteke za zaglavlje i podnožje (mjesto u kojem se uključuju meta podaci unutar HTML jezika), u koju se mogu uključiti skripte ili stilske datoteke kako bi Patternlab mogao prikazati element na željeni način. Te datoteke Patternlab generira prilikom instalacije i ne smije ih se brisati, moguće ih je samo ažurirati na navedeni način, da se uključe dodatne skripte, stilske datoteke ili drugi meta podaci.

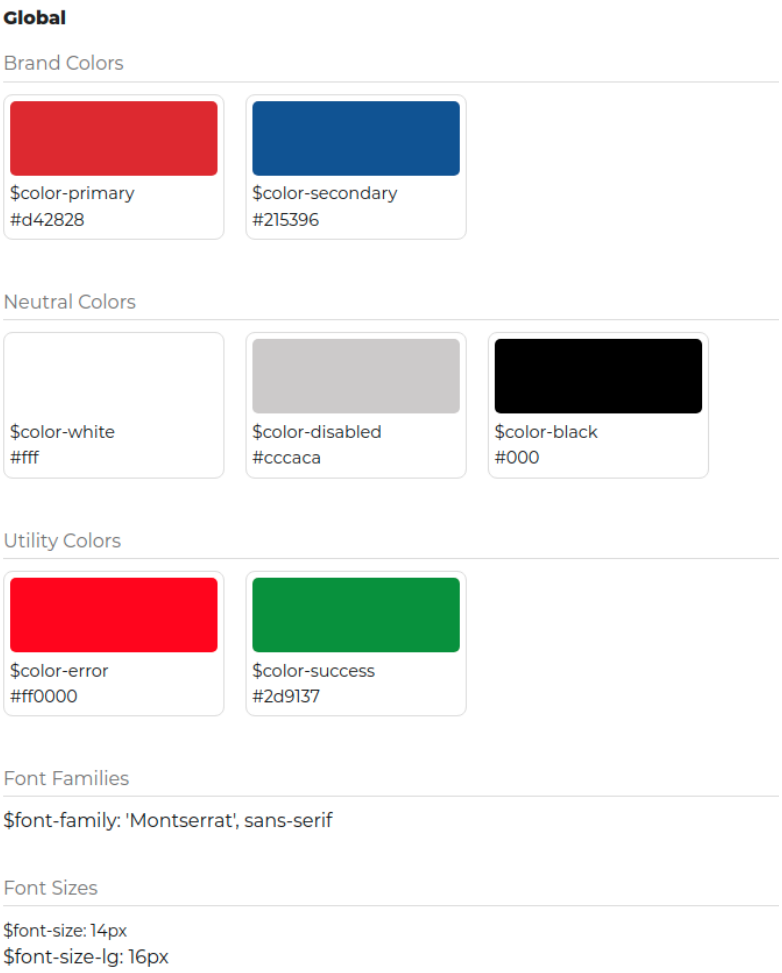
6.3.1. Atomi

Atomi su smješteni unutar *patternlab/source/_patterns/atomi* mape. Projekt se sastoji od šest grupa atoma a to su:

- globalni
- gumbi
- forme
- slike
- liste
- tekstualni atomi

Globalni atomi su atomi koji se koriste globalno kroz sve ostale dijelove atomskog sustava dizajna. U projektu u te atome ulaze boje, od kojih imamo neutralne boje, boje identiteta i

pomoćne boje. Osim boja u globalne atome ulaze i veličina slova i tip slova koji se koristi u projektu.



Slika 26: Globalni atomi

Gumbi su atomi koji se koriste u različitim molekulama i organizmima. Postoje više varijanti gumbova. Implementacija je napravljena tako da postoji jedna datoteka koju ostale varijante nasljeđuju. Kod glavne datoteke:

```
<{{#if buttonTag }}button{{/ if }}{{#if linkTag }}a href="{{ url }}"{{/ if
}} class="{{ class }}" {{# buttonTitle}}title="{{ buttonTitle }}"{{/
buttonTitle}} style="{{style}}" type="button">{{#if buttonText }}{{
buttonText }}{{/ if }}</{{#if buttonTag }}button{{/ if }}{{#if linkTag
}}a{{/ if }}>
```

Kroz varijable je moguće je upravljati koji će se HTML element koristiti za gumb, koja adresa će se koristiti ako se radi o poveznici, moguće je upravljati stilovima, tekstom koji se nalazi unutar gumba, dodavanje klasa i slično.
Varijante gumba uključe taj atom u svojoj datoteci i ažuriraju *json* po potrebi varijante. Primjer za mali gumb:

1. uključivanje atoma

```
{{> atomi-gumb}}
```

2. ažuriranje *json* datoteke

```
{
      "class": "c-btn -small"
}
```

Ukoliko u *json* datoteci nisu navedene varijable koje se nalaze unutar atoma, onda se koriste varijable iz globalnih *json* datoteka. U ovom slučaju se koriste globalne varijable, gdje se samo mijenja varijabla za klasu tj. uključuje se modifikator za mali gumb.
Atomi iz gumb skupine:

- *gumb* - osnovni gumb
- *gumb-zamijenjen* - osnovni gumb sa zamjenutim bojama
- *gumb-sporedni* - sekundarni gumb
- *gumb-sporedni-zamijenjen* - sekundarni gumb sa zamjenutim bojama
- *gumb-puna-sirina* - gumb pune širine kontejnera
- *gumb-mali* - gumb sa manjom veličinom slova
- *gumb-tekst* - gumb koji izgleda kao tekst
- *gumb*-onemogućen - gumb na koji nije moguće kliknuti
- *gumb-zuti-karton* - gumb za dodavanje žutih kartona
- *gumb-crveni-karton* - gumb za dodavanje crvenih kartona
- *gumb-karton*-onemogućen - gumb za dodavanje kartona kada je onemogućen
- *gumb-ikona* - gumb sa ikonom umjesto teksta

Slika 27: Atomi gumbova

Atomi iz skupine formi su atomi koji se odnose na elemente forme. To su prvenstveno elementi za unos podataka od strane korisnika aplikacije. Ta skupina se sastoji od tri atoma, a to su:

- *input*
- *select*
- *radio*

Kod za atom *select*:

```
<div class="form_select-container">
  <div class="form-group">
    <select id={{ id }} placeholder="{{ placeholder }}" {{# multiple
}}multiple{{/ multiple }} {{# disabled }}disabled{{/ disabled }}>
      {{# selectOptions }}
        <option value="{{ value }}">{{ option }}</option>
      {{/ selectOptions }}
      {{# optionGroups }}
        <optgroup label="{{ optGroupLabel }}">
        {{# selectOptions }}
          <option value="{{ value }}">{{ option }}</option>
        {{/ selectOptions }}
          </optgroup>
      {{/ optionGroups }}
    </select>
 </div>
</div>
```

Skupina atoma slika se sastoji od dva atoma:

- *ikona*
- *pejsazna-slika*

Primjer koda za atom *pejsazna-slika:*

```
<img src="{{ src }}" alt="{{ alt }}" class="{{ klase }}" />
```

Skupina atoma za liste se također sastoji od dva atoma:

- *lista definicija*
- *nesortirana lista*

Primjer koda za atom *nesortirana lista*:

```
<ul class="point-list">
  {{#each nesortiranaLista}}
    <li>{{ pojam }}</li>
  {{/each}}
</ul>
```

Skupina atoma za tekst se sastoji od tri atoma a to su:

- *naslov*
- *odjeljak*
- *unutarnji-elementi*

Slika 28: Atomi naslova

6.3.2. Molekule

Molekule kao i atomi su smještene unutar svoje mape na putanji *patternlab/source/_patterns/molekule.* Molekule se sastoje od sedam grupa, a to su:

- blokovi
- gumbovi
- formacija
- forme
- globalni
- liste

- navigacija

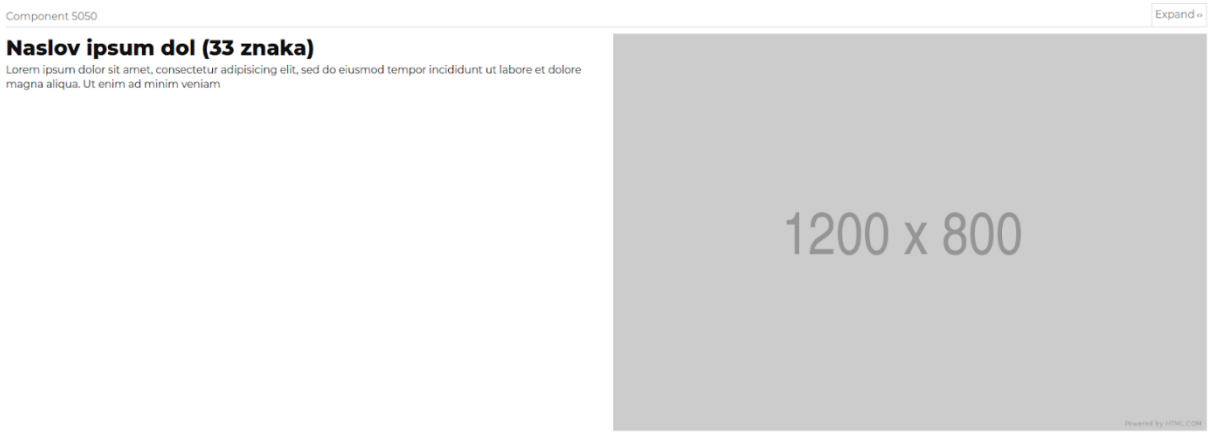
Blokovi se sastoje od tri molekule, to su:

- *Komponenta 50-50*
- *Komponenta greške*
- *Naslovna komponenta*

Komponenta 50-50 je komponenta koja se koristi za prezentaciju sadržaja koji se upotpunjuje slikom. Redoslijed koji dio će biti prvo prikazan je volja autora, odnosno autor može na jednom mjestu imati tekst na lijevoj strani a sliku na drugoj, dok na drugom mjestu može biti obrnuta situacija. Zbog toga se u ovoj molekuli koriste varijable kojima je moguće upravljati sadržajem kada se molekula uključuje u složenije elemente atomskog sustava. Komponenta greške se koristi ukoliko je došlo do pogreške npr. na formi, gdje se korisnika informira o nastaloj greški, te ima stil kojim bi dodatno naznačilo na važnost te poruke. Naslovna komponenta je komponenta koja se koristi za uvod u stranicu. Sastoji se od dva atoma - pejzažne slike i naslova.

Primjer koda komponente 50-50:

```
<div class="component-50-50">
  <div style="{{ lijeviStilovi }}">
    {{{ sadrzajLijevo50 }}}
  </div>
  <div style="{{ desniStilovi }}">
    {{{ sadrzajDesno50}}}
  </div>
</div>
```



Slika 29: Molekula komponente 50-50

Grupa gumbova se sastoji samo od jedne molekula, a to je kontejner za atome i njegov kod izgleda:

```
<div class="btn-container">
```

```
  {{#each listaGumbova}}
    {{> atomi-gumb}}
  {{/each}}
</div>
```

Kroz varijable je moguće dodavati atome koji će se prikazati unutar molekule, koji se kroz petlju prikažu unutar Patternlab-a.

Grupa formacija se sastoji od dvije molekule, igrača i terena formacije. Igrač je molekula koja se služi za prikaz igrača na formaciji, te s njom se prikazuju informacije o igraču, odnosno ime i gumb za brisanje igrača.

Primjer koda:

```
<div class="formation_field-list-item">
  <img src="../../../images/dres.png" alt="">
  <p>{{ naziv }}</p>
  {{#if gumbText}}
    <button>{{ gumbText }}</button>
  {{/if}}
</div>
```

Teren formacije je složenija molekula koja uključuje molekulu igrač iz ove grupe molekula. S njom se prikazuje cijela formacija koju korisnik izabere. Za tu molekulu se definiraju varijable pozicija kroz koje će se petlja vrtiti kako bi se igrači prikazali na pravilan način.

Primjer koda za molekulu teren formacije:

```
<div class="formation_field">
  <ul class="formation_field-list -attackers">
    {{#each napadaci}}
      <li class="formation_field-list-item">
        {{> molekule-igrac}}
      </li>
    {{/each}}
  </ul>
  <ul class="formation_field-list -midfielders">
    {{#each veznjaci}}
      <li class="formation_field-list-item">
        {{> molekule-igrac}}
      </li>
    {{/each}}
  </ul>
  <ul class="formation_field-list -defenders">
    {{#each braniči}}
      <li class="formation_field-list-item">
        {{> molekule-igrac}}
      </li>
    {{/each}}
  </ul>
  <ul class="formation_field-list -goalkeeper">
    {{# golman}}
      <li class="formation_field-list-item">
        {{> molekule-igrac}}
      </li>
    {{/ golman}}
  </ul>
</div>
<div class="formation_raw-list-container">
```

```
  <p>{{ naslovListe}}</p>
  <ul class="player-list">
    {{#each igrači}}
      {{> molekule-kartica-igraca}}
    {{/each}}
  </ul>
</div>
```

Slika 30: Molekula terena formacije

Molekule forme su molekule raznih tipova elemenata forme za unošenje podataka zajedno sa etiketom (eng. *label)*. Sastoji se od *input* elemenata za unošenje teksta, brojeva, datuma, vremena te *radio* i *select* elemenata sa svojom etiketom.

Primjer koda za element kojim se unosi datum:

```
<div class="form-group">
    {{#if label}}
    <div class="form-group_label-container">
        <label id="{{ id }}">{{ label }}</label>
    </div>
    {{/if}}
    <input type="date" placeholder="{{ placeholder }}" id="{{ id }}">
</div>
```

Pod globalne molekula spada samo logo koji se koristi na navigaciji. Sastoji se od slike koja je ujedno i poveznica, koja u slučaju aplikacije ovog diplomskog rada vodi na početnu stranicu.

Primjer koda molekule logo:

```
<a href="{{ url }}" class="navigation_logo router-link-active">
    <img            src="../../../images/cro-1-hnl-logo-png-transparent.png"
alt="hnl_logo">
</a>
```

Molekula navigacije je molekula koja se koristi unutar zaglavlja za navigiranje unutar aplikacije. To je također dinamična molekula koja sadrži petlju kroz koju se mogu dodavati elementi navigacije putem *json* datoteke.

Primjer koda navigacije:

```
<ul class="navigation_list">
  {{#each stavkeNavigacije}}
      <li>
        <a href="{{ url }}">{{ naslov }}</a>
      </li>
  {{/each}}
</ul>
```

Molekule grupe lista su molekule kojima se ispisuju informacije o ligi, igraču i kolu. Sastoji se od četiri molekule, a to su:

- *kartica lige*
- *kartica igrača*
- *kartica kola*
- *kartica igrača podliste (eng. sublist)*

Molekula kartica lige služi za ispisivanje informacija o ligi te je dio organizma popisa liga. Informacije koje se prikazuju korisniku su naziv lige, vrijeme odabira igrača, trenutni broj prijavljenih korisnika za ligu i maksimalni broj korisnika koji se može prijaviti za ligu. Osim toga prikazuju se dva gumba, jedan kojim se korisnik prijavljuje za ligu, a drugi za izmjenu podataka o ligi. Kartica igrača je molekula koja je dio organizma popisa igrača te se u njoj prikazuju informacije o igraču, kao što su njegovo ime i cijena. Uz to se nalazi gumb kojim je moguće izvršiti akciju ovisno gdje je molekula, odnosno organizam uključen unutar aplikacije. Kartica kola je isto tako molekula koja prikazuje informacije o kolu. Informacije koje se prikazuju su početak kola, ekipe koje igraju te rezultat. Kartica igrača podliste prikazuje ime igrača koji je igrao utakmicu, te uključuje molekulu kontejner gumba s kojom se dodaju ili oduzimaju asistencije i golovi igrača.

Primjer koda molekule kartice lige:

```
<div class="league-card">
  <a href="#">
    <h3>{{ naslov }}</h3>
  </a>
  <dl>
    <dt>Draft:</dt>
    <dd>{{ vrijemeDrafta }}</dd>
  </dl>
  <dl>
    <dt>{{ brojIgracaNaslov }}</dt>
    <dd>{{ trenutniBrojIgraca }}/{{ maxBroj }}</dd>
  </dl>
  {{# gumbJedan}}
```

```
    {{> atomi-gumb buttonTag=true}}
  {{/ gumbJedan}}
  {{# gumbDva }}
    {{> atomi-gumb buttonTag=true}}
  {{/ gumbJedan}}
</div>
```

6.3.3. Organizmi

Organizmi se sastoje od pet skupina organizama. To su:

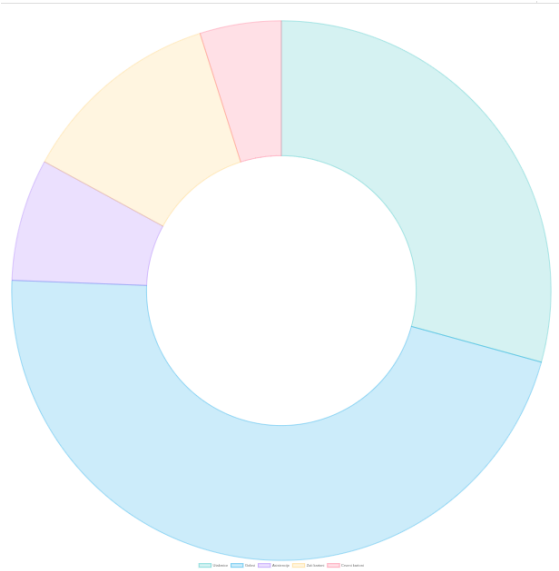
- globalni
- grafikoni
- formacija
- forma
- liste

U globalne organizme spadaju zaglavlje, podnožje i notifikacija. Ovi organizmi se koriste na svakoj ili gotovo na svakoj stranici. Zaglavlje uključuje molekulu navigacije, molekulu loga i atom ikone za mobilnu verziju. Podnožje i notifikacija su jednostavni organizmi koji su prisutni na gotovo svim stranicama. Podnožje nije uključeno samo na stranicu autentikacije. Kod zaglavlja:

```
<header>
    <nav class="navigation">
      {{> molekule-logo}}
      <button class="navigation_burger js-navigation" type="button">
        {{> atomi-ikona}}
      </button>
        {{> molekule-navigacija}}
    </nav>
</header>
```

Slika 31: Globalni organizmi

Grafikoni se sastoje od jednog organizma, a to je kružni organizam. To je organizam koji se sastoji od jednom HTML elementa u koji se putem Javascripta inicijalizira grafikon. Kružni grafikon se koristi kod pregleda statistike igrača.



Slika 32: Organizam grafikon

Formacija se također sastoji od jednog organizma a to je tijelo formacije. Ovaj organizam se sastoji od molekule terena formacije i organizma liste igrača. Koristi se prilikom odabira igrača i prilikom ažuriranja formacije odnosno ekipe tijekom određenog kola. Kod ovog organizma:

```
<div class="formation_container">
  {{> molekule-teren-formacije}}

  <div class="formation_bench">
    <p>{{ naslov }}</p>
    {{> organizmi-lista-igraca}}
  </div>
</div>
```

Slika 33: Organizam tijelo formacije

Pod organizme grupe forme spadaju općenita forma i forma za pretraživanje. Forma kao organizam je skup elemenata za unos podataka od strane korisnika. U tom organizmu se nalazi petlja koja prolazi kroz definirano polje elemenata i uključuje ih ukoliko se nalazi u podatkovnoj datoteci. Sastoji se od molekula iz skupine formi. Na taj način prilikom uključivanja organizma mogu se mijenjati podaci, odnosno polja za unos ovisno o mjestu u kojem se uključuju. Forma za pretraživanje se koristi prilikom pretraživanja liga i igrača. Uključuje molekulu za tekstualni unos podataka te atom gumba sa ikonom. Primjer koda forme za pretraživanje:

```
<div class="search-bar">
  <form class="form">
    <div class="form-group">
      {{> molekule-unos-tekst label=false }}
      {{> atomi-gumb-ikona class="c-btn -secondary"}}
    </div>
  </form>
</div>
```

Slika 34: Organizam forme za pretraživanje

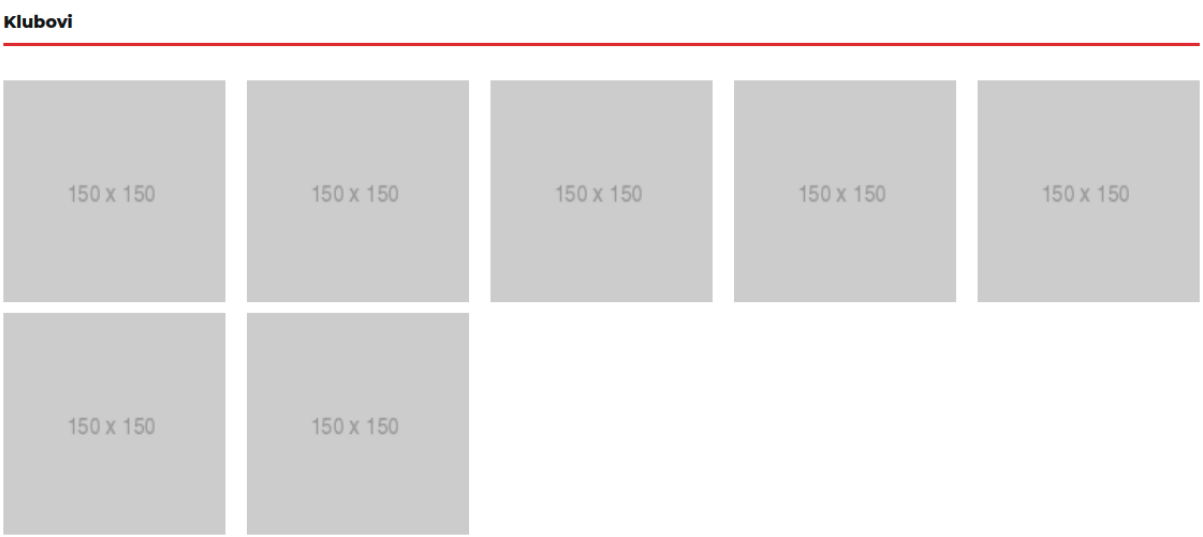
Skupina lista je skupina organizama koja uglavnom uključuje molekule kartica kojima se ispisuju informacije o igračima, kolima i klubovima. Organizmi iz ove skupine su među organizmima koji su najviše bili uključeni u stranice. Skupina se sastoji od šest organizama. To su:

- *lista klubova*
- *lista liga*
- *lista igrača*

- *lista kola*
- *tablica bodova*
- *podlista igrača kola*

Lista klubova je organizam koji se koristi prilikom odabira igrača. Organizam uključuje molekulu radio-slika, gdje postoji petlja kroz koju se uključuju podaci za molekulu. Kod za ovaj organizam:

```
{{# listaKlubova}}
  <p class="draft_clubs-title">{{title}}</p>
  <ul class="draft_clubs-list">
    {{#each klubovi}}
     {{> molekule-radio-slika}}
    {{/each}}
  </ul>
{{/ listaKlubova}}
```



Slika 35: Organizam liste klubova

Lista liga se koristi na stranici Popis liga. To je organizam u kojem se kroz petlju ispisuju odnosno uključuju molekule kartice lige. Kod liste lige:

```
{{# listaLiga}}
<ul class="league-listing_list">
  {{#each lige}}
    <li>
      {{> molekule-kartica-lige}}
    </li>
  {{/each}}
</ul>
{{/ listaLiga}}
```

Slika 36: Organizam lista liga

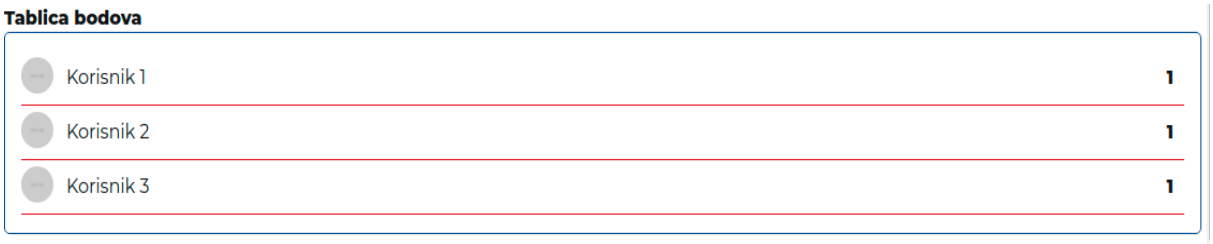
Lista igrača je jedan od najčešćih korištenih organizama unutar ovog sustava dizajna. Koristi se prilikom odabira igrača, kod statistike igrača, ažuriranja sastava ekipe, ažuriranja kola. Organizam također ima petlju kroz koju uključuje molekulu kartice igrača. Kod ovog organizma:

```
{{# listaIgraca}}
  <ul class="player-list">
    {{#each igraci}}
      {{> molekule-kartica-igraca}}
    {{/each}}
  </ul>
{{/ listaIgraca}}
```

Organizam tablica bodova informira korisniku o rangu igrača lige i broju bodova koji su skupili sudjelovanjem u određenoj ligi. Kroz ovaj organizam se prikazuje profilna slika korisnika, njegovo korisničko ime te broj bodova. Ovaj organizam se uključuje na početnoj stranici ukoliko je korisnik prijavljen, te na stranici za upravljanje ekipom.

```
{{# listaKorisnika}}
<div class="points-table_container">
  <strong>{{ naslov }}</strong>
  <ol class="points-table {{modifier}}">
    {{#each korisnici}}
    <li>
      <div>
        {{#if img}}
        <img src="{{img}}">
        {{/if}}
        {{^ img}}
        <img src="../../images/placeholder150.png">
        {{/ img}}
        <span>{{korisnickoIme }}
        </span>
      </div>
      {{#if bodovi}}
      <div>
```

```
        <strong>{{ bodovi }}</strong>
      </div>
      {{/if}}
    </li>
    {{/each}}
  </ol>
</div>
{{/ userList}}
```



Slika 37: Organizam tablice bodova

6.3.4. Predlošci i stranice

Predlošci i stranice kao i ostali dijelovi atomskog sustava imaju svoje mape unutar Patternlab projekta. Ova dva dijela atomskog sustava su jako povezana te zbog toga će biti objašnjeni zajedno u ovoj sekciji. Kako je prethodno u radu objašnjeno, ovim dijelovima atomskog sustava se opisuje kako izgledaju stranice aplikacije. Projekt se sastoji od jedanaest predložaka, odnosno stranica. Sadržaj se kao i prethodno manipulira preko dodijeljenih *json* datoteka. Kod stranica izgled datoteke je skoro pa jednak u svakoj od njih. U stranicu se uključuje samo predložak koji odgovara stranici i nakon toga se u dodijeljenoj *json* datoteci se izmjeni sadržaj, koji odgovara realnom izgledu stranice. Primjer koda unutar stranice:

```
{{> templates-naziv-predloška}}
```

I predlošci i stranice se sastoje od jedanaest dijelova, gdje će neki od njih biti opisani u nastavku. Predlošci i stranice se sastoje se od:

- početna stranica
- autentikacija
- draft
- uređivanje profila
- upravljanje ligom
- upravljanje kolima
- upravljanje ekipom
- upravljanje korisnicima
- stranica nije pronađena
- popis liga
- informacije o ligi

Početna stranica je predložak koji uključuje organizam zaglavlja, molekulu naslova, atom odjeljka, molekulu 50-50 komponentu i organizam podnožja. Njezina *json* datoteka sadrži varijable kojima će se nadjačati sadržaj iz globalnih *json* datoteka i koje će se kasnije koristiti u slučaju stranice. Kod predloška početne stranice:

```
<div id="app">
      <div class="container">
            {{> orgranizmi-zaglavlje }}
            <main class="c-container">
                  {{> molekule-komponenta-naslova}}
                  <div class="row">
                        <div class="col-lg-7 text-justify">
                              <p>{{{ tekst }}}</p>
                        </div>
                  </div>
                  {{> molekule-5050-komponenta}}
            </main>
            {{#with podnozje}}
                  {{> organizmi-podnozje }}
            {{/with}}
      </div>
</div>
```

Slika 38: Predložak početne stranice

Autentikacija je predložak koji uključuje atom naslova, organizam forme, atom guba te molekulu greške. Za ovaj predložak postoji i posebna Javascript datoteka koja simulira logiku promjene forme između prijave i registracije.

Draft je predložak koji uključuje organizam zaglavlja, atom naslova, organizam liste igrača, atom gumba, organizam formacije i organizam podnožja. Kao i u prethodnim primjerima i ovaj predložak ima svoju *json* datoteku kojima nadjačava iz globalnih *json* datoteka i oblikuje sadržaj po svojoj potrebi. Kod predloška:

```
<div id="app">
      <div class="container">
            {{> organizmi-zaglavlje }}
            <main class="c-container draft">
      <h1>{{ naslov }}</h1>
      <div class="row">
        <div class="col-12 col-md-6">
          <div class="draft_budget-container">
            <p class="draft_budget">{{ budzet }}</p>
            <p class="draft_budget-total">{{ budzetTotal }}</p>
          </div>
          <div class="draft_positions">
            {{#each pozicije}}
              <input type="radio" id="{{pozicija}}" name="position">
              <label     for="{{pozicija}}"     class="c-btn     -small     -
reversed">{{pozicija}}</label>
            {{/each}}
          </div>
          {{> organizmi-lista-igraca}}
          {{> atomi-gumb buttonTag=true class="c-btn -full-width"}}
        </div>
        <div class="col-12 col-md-6">
          {{> organisms-lista-klubova}}
          {{#with formacija}}
            {{> organizmi-tijelo-formacije}
          {{/with}}
        </div>
      </div>
            </main>
            {{#with podnozje}}
                  {{> organizmi-podnozje }}
            {{/with}}
      </div>
</div>
```

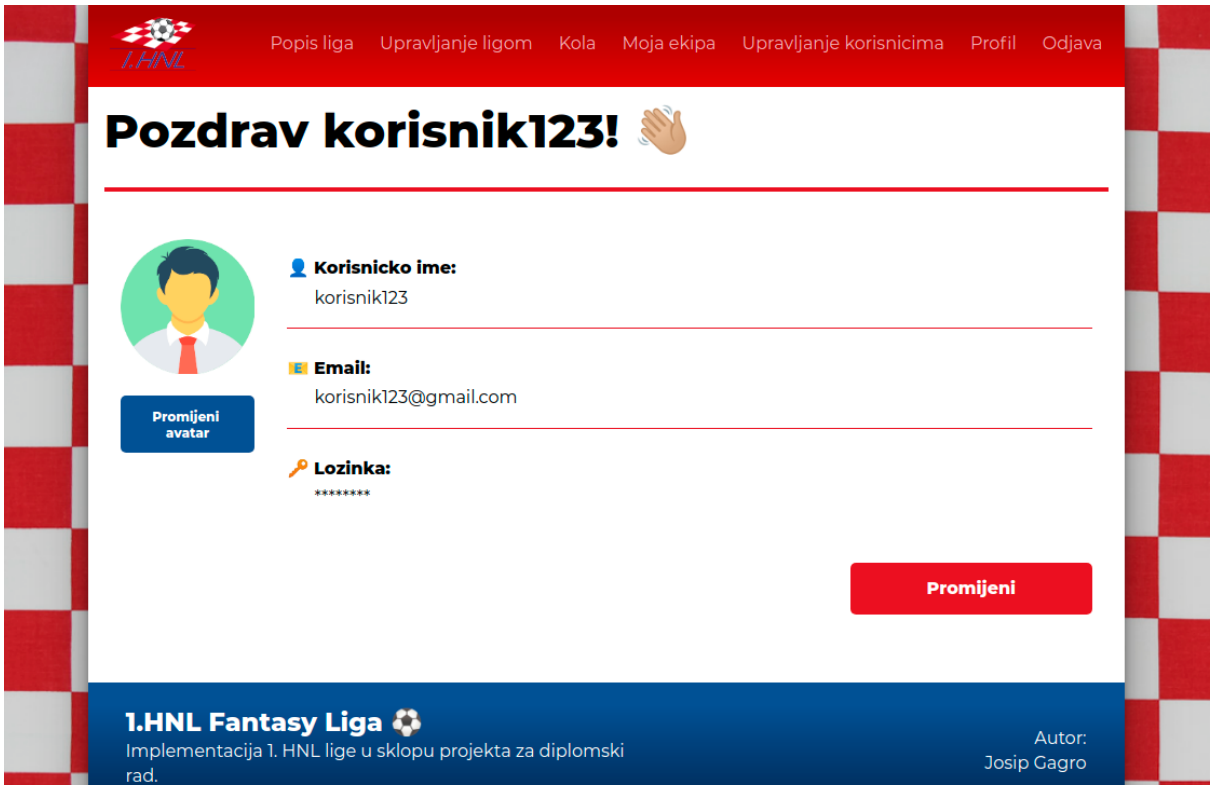
Predložak za uređivanje profila se sastoji od organizma zaglavlja, atoma gumba, organizma forme, atoma liste definicija i organizma podnožja. Ovaj predložak također sadrži logiku u zasebnoj Javascript datoteci, koja služi za prikaz i skrivanje forme za ažuriranje profila. Logika za to:

```
window.addEventListener('DOMContentLoaded', () => {
  const infoBtn = document.querySelector('.js-infos');
  const infoContainer = document.querySelector('.user-profile_info');

  if(infoBtn) {
    infoBtn.addEventListener('click', () => {
      if(infoContainer) {
        infoContainer.classList.add('d-none');
        const form = document.querySelector('.user-profile .js-form');
        form.classList.remove('d-none');

        const closeBtn = form.querySelectorAll('button');
```

```
        if(closeBtn.length > 0) {
          closeBtn.forEach(btn => {
            btn.addEventListener('click', () => {
              form.classList.add('d-none');
              infoContainer.classList.remove('d-none');
            });
          })
        }
      }
    })
  }
});
```



Slika 39: Stranica upravljanja profila

6.4. Aplikacija

6.4.1. Početna stranica

Početna stranica je stranica koja dočekuje korisnika prilikom korištenja aplikacije. Na njoj se nalaze osnovne informacije o stranici i pravilima igre. Ova Vue komponenta, odnosno stranica uključuje dvije manje komponente a to su Navigacija i Tablica bodova. Neprijavljeni korisnik na početnoj stranici na navigaciji može vidjeti samo poveznicu na Prijavu, koja će ga odvesti na stranicu autentikacije, gdje se može prijaviti ili registrirati. Također, neprijavljeni korisnik ukoliko želi posjetiti neku drugu stranicu aplikacije će biti preusmjeren na stranicu autentikacije. Tom logikom upravlja Vue-ov usmjerivač, gdje se provjerava da li je autentikacija potrebna za određenu putanju te postoji li trenutni korisnik. Ukoliko ne postoji trenutni korisnik usmjerivač ga preusmjerava na stranicu za autentikaciju, suprotno ga vodi na željeno odredište. Kod za to je:

```
usmjerivac.beforeEach((to, from, next) => {
  const auth = to.matched.some(x => x.meta.requiresAuth);
  const trenutniKorisnik = firebase.auth().currentUser;

  if (auth && !trenutniKorisnik) {
    next({
      path: '/auth',
      query: { redirect: to.fullPath }
    });
  } else {
    next();
  }
});
```

Ukoliko je korisnik prijavljen, dohvaćaju se lige koje se prosljeđuju komponenti Tablice bodova kao svojstvo (eng. *property*). Korisnik će moći odabrati jednu od tih liga i prikazati će se bodovi sudionika lige. Metoda za dohvaćanje liga se poziva nakon što se komponenta Početne stranice kreira. Prilikom kreiranja se provjerava da li postoji korisnik u Vuex-u te ukoliko postoji poziva metoda. Kod za metodu:

```
methods: {
dohvatiLige: async function() {
  try {
    this.$store.dispatch('postaviCekanje', true);
    await this.trenutniKorisnik.lige.forEach(async liga => {
    const podaciLige = await firebase.kolekcijaLiga.doc(liga.id).get();
    this.lige.push({...podaciLige.data(), id: liga.id});
    });
    this.$store.dispatch('postaviCekanje', false);
  } catch(error) {
    this.$store.dispatch('postaviCekanje', false);
    this.$store.dispatch('pokaziObavijest', {naslov: 'Dogodila se greska!',
info: 'Molimo pokusajte ponovno', uspjeh: false});
    throw new Error(error);
  }
```

```
}},
created() {
  if(this.trenutniKorisniK) {
      this.dohvatiLige();
  }
}
```

Slika 40: Početna stranica

6.4.2. Autentikacija

Autentikacija je dio aplikacije na kojem se neprijavljeni korisnik može prijaviti, registrirati ili zatražiti obnovu lozinke ukoliko ju je zaboravio. Autentikacija se sastoji od četiri komponente a to su komponenta za prijavu, komponenta za registraciju, komponenta za obnavljanje lozinke i komponenta za informacije. Može se reći da je Autentikacija kontejner ove četiri komponente, koja upravlja koja će se komponenta prikazati korisniku. Za to ima posebna varijabla unutar komponente, te na osnovu njene vrijednosti prikazan je određen sadržaj korisniku. Inicijalnu vrijednost ima 0, što prikazuje formu prijave. Komponenta prijave je forma u koju korisnik unosi svoje informacije te se klikom na gumb prijavljuje u aplikaciju. Svaki element forme ima i implementiranu validaciju, gdje se provjerava da li je email unesen u točnom formatu te da li lozinka sadrži bar osam znakova. Validacija forme je izrađena pomoću Vue-ove biblioteke Vuelidate. Ukoliko su uneseni podaci točni klikom na gumb prijava korisnik se prijavljuje u

aplikaciju. U slučaju pogrešne email adrese ili netočne lozinke ispisuje se prigodna poruka korisniku. Kod za prijavu korisnika:

```
prijaviKorisinka: async function() {
  this.$v.$touch();
  if(!this.$v.$invalid) {
    try {
      this.$store.dispatch('postaviCekanje', true);
      const { user } = await
firebase.auth.signInWithEmailAndPassword(this.email, this.lozinka);
      this.porukaGreske = null;
      if(user) {
        this.$store.dispatch('postaviCekanje', false);
        this.$store.dispatch('dohvatiKorisnikovProfil', user);
      }
    } catch(error) {
      this.$store.dispatch('postaviCekanje', false);
      switch(error.code) {
        case 'auth/user-not-found': this.porukaGreske = 'Korisnik nije
pronaden. Molimo obavite registraciju.'; break;
        case 'auth/wrong-password': this.porukaGreske = 'Pogresna
lozinka.';
      }
    }
  }
}
```

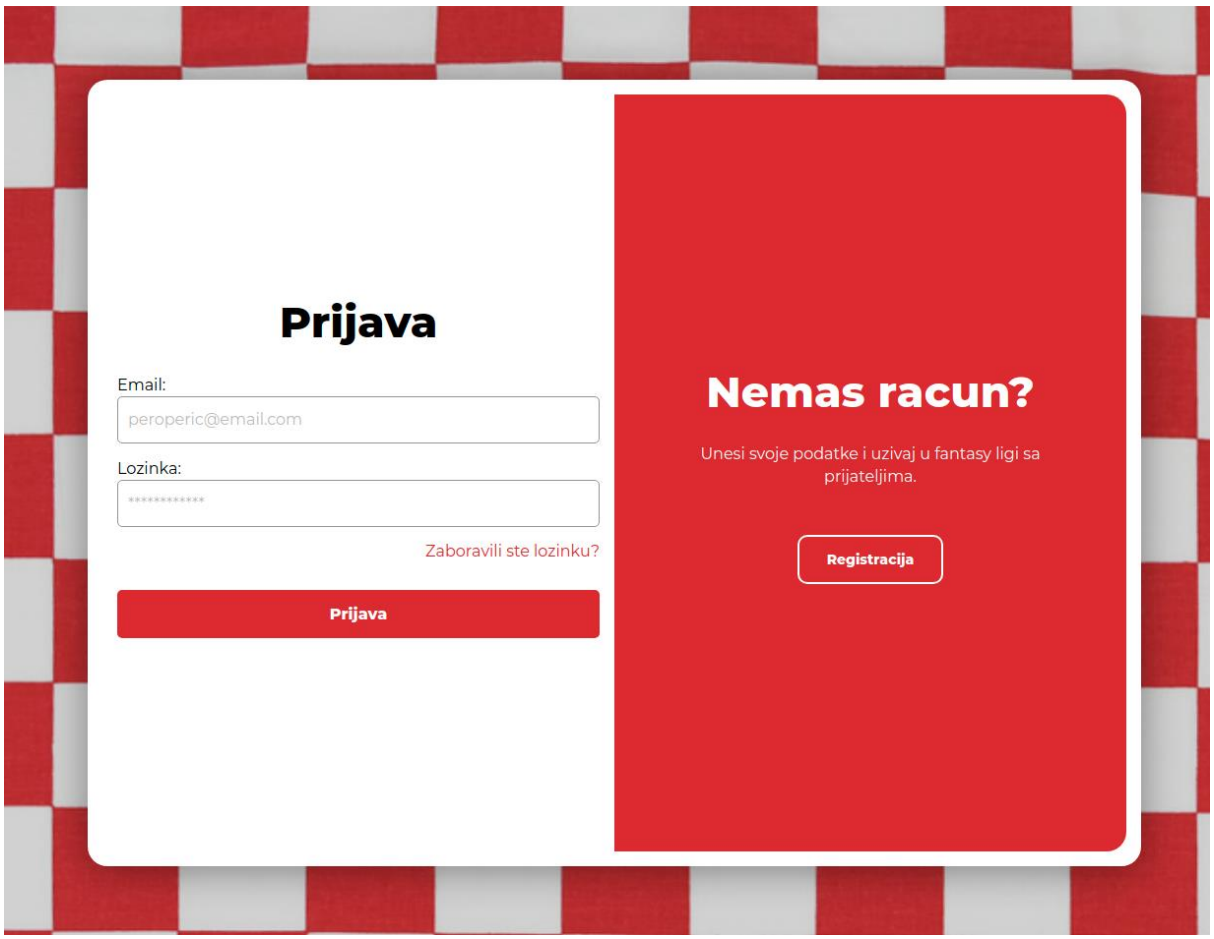
Komponenta Registracija je također komponenta forme, gdje također svaki element za unos podataka ima validaciju. Svaki od elemenata je obavezan za unijeti, email mora biti u točnom formatu, lozinka mora imati minimalno osam znakova i ponovljena lozinka mora biti jednaka kao i lozinka. Klikom na gumb Registracija poziva se Firebase metoda za kreiranje korisnika sa email-om i lozinkom te se njegovi podaci spremaju u bazu podataka. Kod za registraciju:

```
registrirajKorisnika: async function() {
  this.$v.$touch();
  if(!this.$v.$invalid) {
    try {
      this.porukaGreske = null;
      this.$store.dispatch('postaviCekanje', true);
      const { user } = await
firebase.auth.createUserWithEmailAndPassword(this.email, this.lozinka);
      await firebase.kolekcijaKorisnika.doc(user.uid).set({
        id: user.uid,
        username: this.korisnickoIme,
        email: this.email,
        password: this.lozinka,
        type: 1
      });
      this.$store.dispatch('dohvatiKorisnikovProfil', user);
    } catch(error) {
      switch(error.code) {
        case 'auth/email-already-in-use': this.porukaGreske = 'Email je vec
u upotrebi.'; break;
        case 'auth/weak-password': this.porukaGreske = 'Lozinka je
prekratka'; break;
      }
    }
```

```
    }
}
```

Do forme za obnovu lozinke korisnik dolazi tako da na formi za prijavu klikne na poveznicu “Zaboravili ste lozinku?”. Otvara se forma sa jednim elementom za unos podataka u koji korisnik unosi svoj email. I ovdje je implementirana validacija za obavezno polje i email format. Klikom na gumb Pošalji korisniku se na popunjenu adresu, ukoliko korisnik postoji, šalje email na kojem dobiva poveznicu za obnovu lozinke. Kod za obnovu lozinke:

```
obnoviLozinku: async function() {
      try {
        await firebase.auth.sendPasswordResetEmail(this.email);
        this.promijeniPogled();
      } catch (error) {
        switch(error.code) {
          case "auth/invalid-email": this.porukaGreske = 'Email je pogresno
formatiran.'
        }
      }
    }
```



Slika 41: Autentikacija

6.4.3. Popis liga

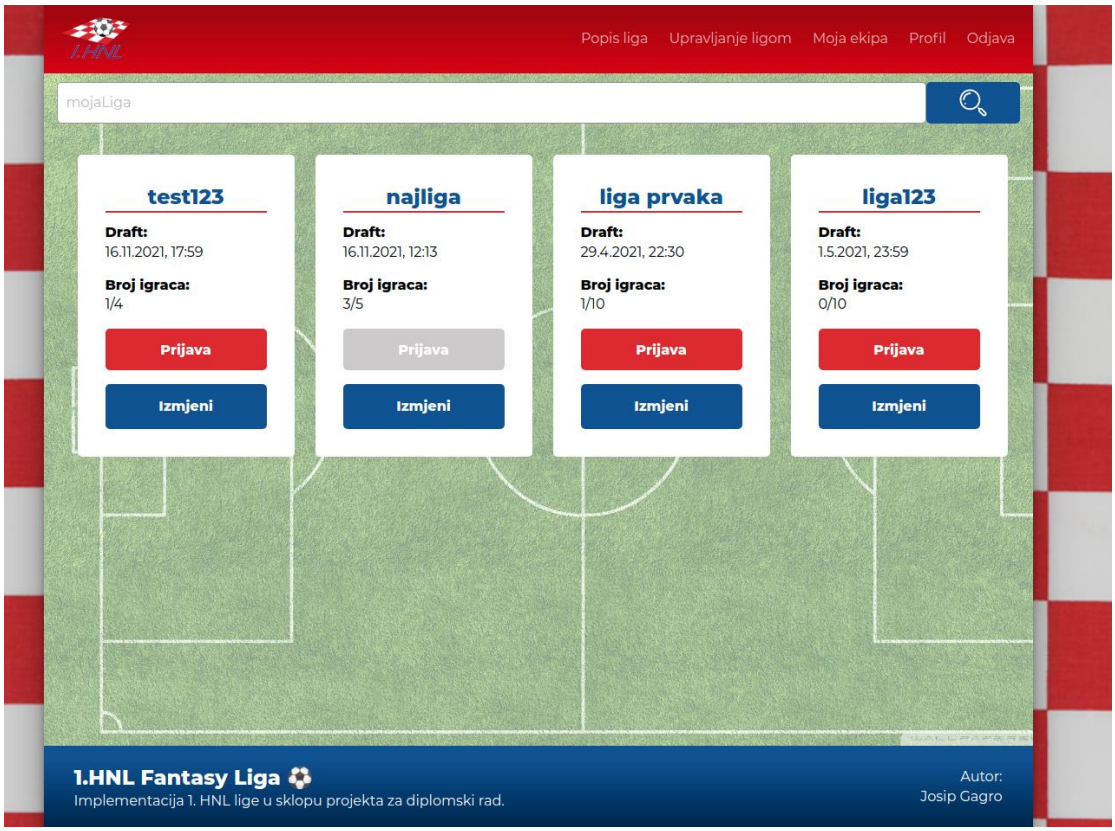
Popis liga je komponenta kojoj mogu pristupiti svi prijavljeni korisnici. Na njoj se izlistavaju kreirane lige na koje se korisnik može prijaviti. Moderatoru je vidljiva dodatna opcija, odnosno gumb koji ga vodi na izmjenu podataka o ligi. Komponenta uključuje dvije manje komponente, a to su komponenta kartice lige i komponenta za pretragu. Svaka kartica prikazuje naziv lige, vrijeme početka odabira igrača i broj prijavljenih igrača. Dohvaćene lige iz baze podataka se spremaju u lokalnu varijablu unutar komponente te se dalje prosljeđuju karticama lige kao svojstvo. Dohvaćanje liga je napravljeno s paginacijom. Gumb za dohvaćanje novih liga se sakrije ukoliko nema više liga za dohvatiti. Ključne Firebase metode za paginaciju su *startAfter i limit.* Metoda *startAfter* govori bazi podataka koji dokument će uzimati kao početak kolekcije, dok *limit* funkcija govori bazi podataka koliko dokumenata će poslati korisničkoj strani. Kod za dohvaćanje liga s paginacijom:

```
_lige = await firebase.kolekcijaLiga
                    .startAfter(this.zadnjaVidljiva)
                    .limit(this.brojPaginacija)
                    .get();
```

Kako bi se paginacija dogodila potrebno je pamtiti zadnji dokument koji se učitao putem metode za dohvaćanje liga. U ovom slučaju taj dokument se sprema u varijablu *zadnjaVidljiva*. Nove lige se priključe ostalim ligama unutar lokalne varijable lige, gdje se kasnije izlistaju na ekranu.

Pretraživanje liga se izvodi preko komponente za pretraživanje. Kad se unese naziv ili dio naziva lige u polje za unos i klikne na gumb za pretraživanje izlistati će se samo lige koje počinju sa unesenom vrijednošću. Ukoliko se izvrši pretraživanje sa praznim poljem, izlistati će se sve kreirane lige. Za pretraživanje se koriste Firebase upiti, za koje se koristi funkcija *where.* Kod za pretraživanje:

```
if(this.vrijednostPretrazivanje.length > 0) {
  const krajRijeci = this.vrijednostPretrazivanje.replace(/.$/, c =>
String.fromCharCode(c.charCodeAt(0) + 1));
  lige = await firebase.kolekcijaLiga
                  .where('ime', '>=', this.vrijednostPretrazivanje)
                  .where('ime', '<', krajRijeci)
                  .orderBy('ime')
                  .get();
} else {
  lige = await firebase.kolekcijaLiga.get();
}
```



Slika 42: Popis liga

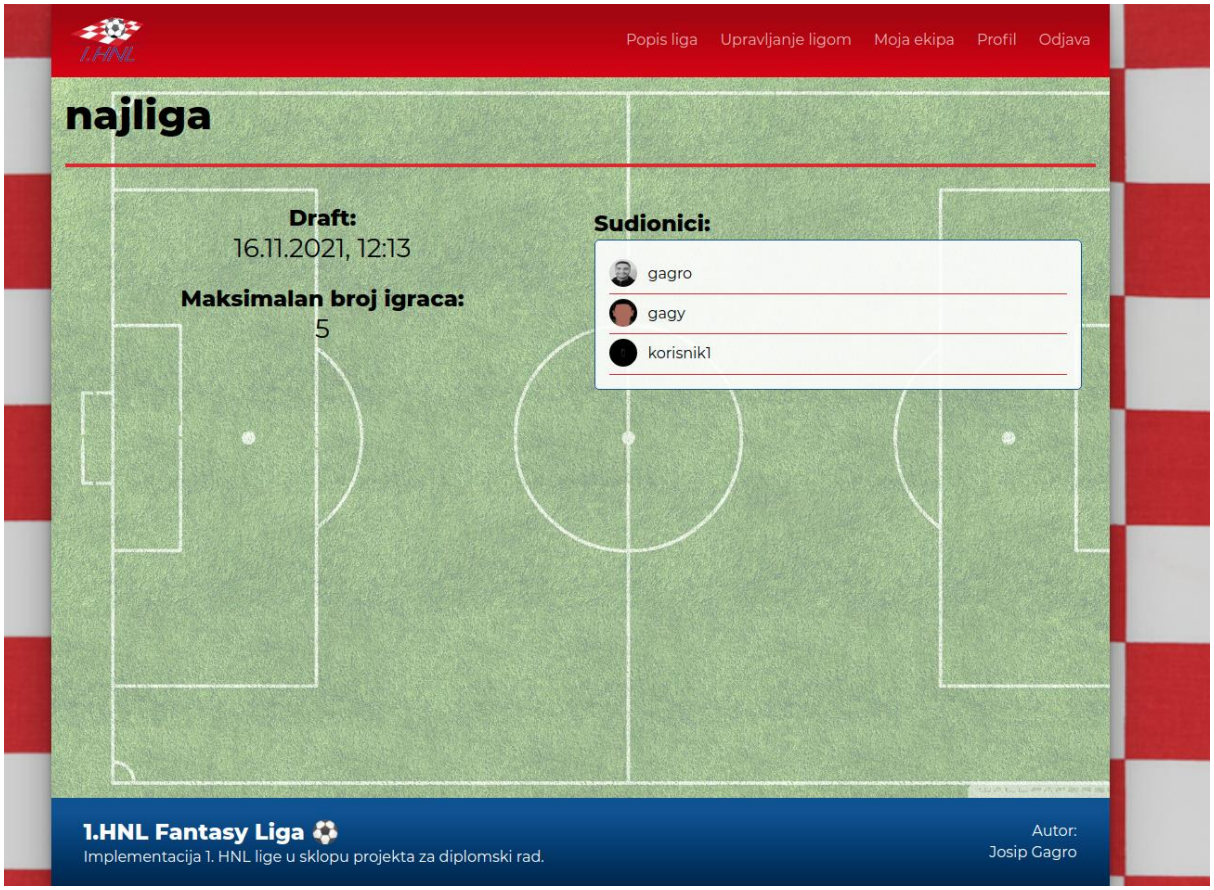
6.4.4. Informacije lige

Klikom na naslov lige kartice lige otvara se nova stranica - informacije lige. Informacije lige su dostupne svim prijavljenim korisnicima aplikacije. Prilikom kreiranja komponente dohvaćaju se informacije o ligi, gdje se identifikacijska oznaka lige dohvaća iz putanje. Identifikacijskoj oznaci iz putanje se pristupa iz *$route* objekta koji je dio Vue-a. Na ovoj komponenti za dohvaćanje podataka se ne koristi metoda *get*, već se koristi metoda *onSnapshot* koja prati promjene na dokumentu, te ukoliko se dogodi promjena ona obavještava aplikaciju i ažurira podatke u stvarnom vremenu. Tako se u stvarnom vremenu može vidjeti novi korisnik koji je pristupio ligi ili promjenu neke od informacija vezanih za ligu. Primjer:

```
this.snimakIgraca = await
firebase.kolekcijeLiga.doc(this.ligaId).onSnapshot(doc => {
  this.igraci = doc.data().igraci;
  this.$nextTick(() => {
    this.$store.dispatch('postaviIgraceLige', this.igraci || []);
  })
});
```

Pošto je ovo jednostranična aplikacija (eng. *Single Page Application - SPA*) slušanje dokumenta je potrebno isključiti kako se ne bi nastavilo slušati na promjene ukoliko korisnik ode na drugu stranicu/komponentu. Isključivanje se radi prije uništavanja trenutne komponente pozivom varijable *snimakIgraca* kao funkcije - *snimakIgraca().*

S ove stranice korisnici također mogu pristupiti i odabiru igrača. Gumb koji vodi na odabir igrača je vidljiv samo ako je korisnik prijavljen za sudjelovanje u ligi i ako je trenutni datum u rasponu od početka odabira i sedam dana nakon tog, što znači da korisnik ima tjedan dana izabrati ekipu za sezonu u ligi u kojoj sudjeluje.



Slika 43: Informacije o ligi

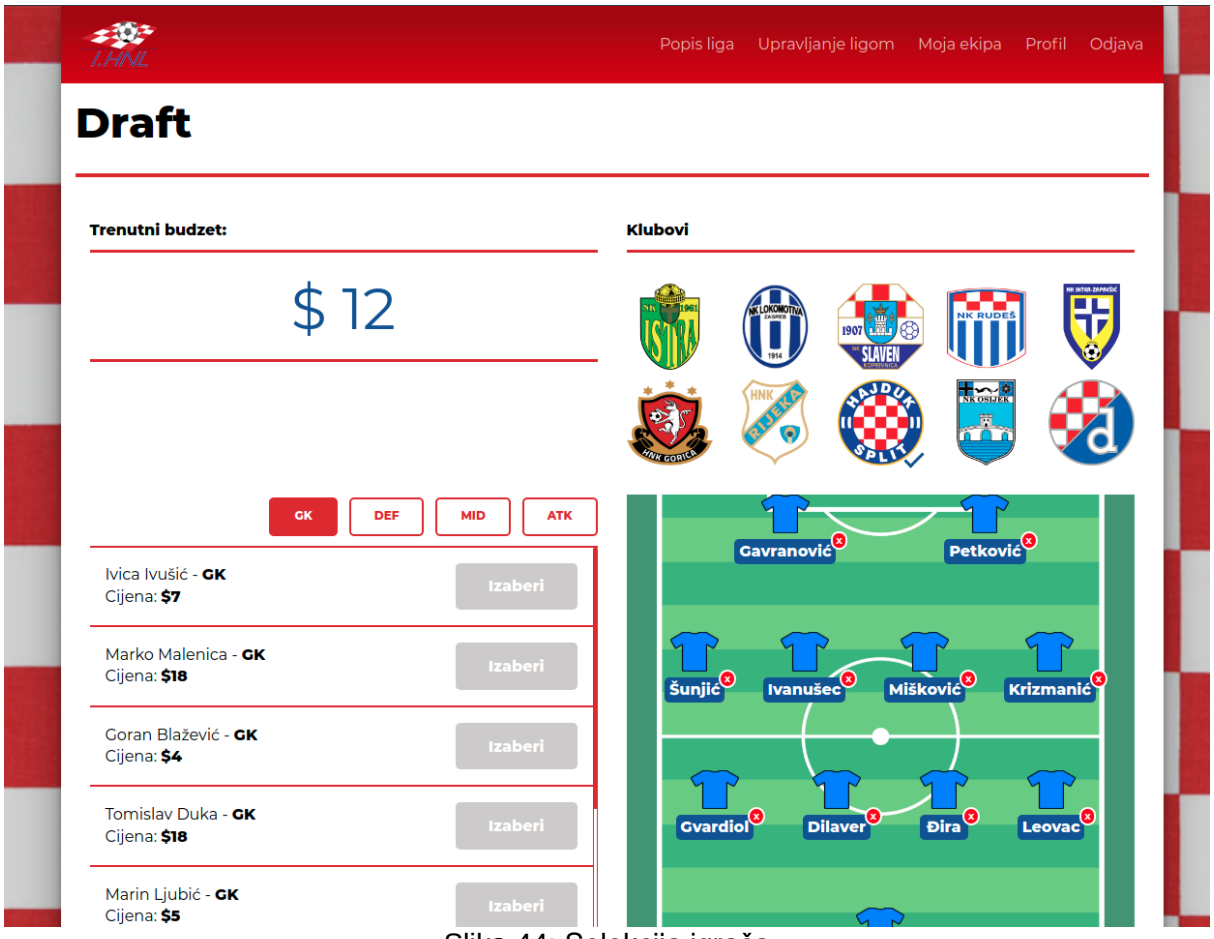
6.4.5. Selekcija igrača

Selekciji igrača mogu pristupiti korisnici koji su prijavljeni u ligu. Prilikom kreiranja komponente dohvaćaju se informacije o klubovima i informacije o korisnikovim podacima unutar odabrane lige, što podrazumijeva preostali korisnikov budžet i igrače koje je korisnik do tog trenutka izabra. Klubovi su prikazani sa svojim logom, te klikom na njih se učitavaju nogometaši odabranog kluba. Nogometaš može biti izabran ako korisnik ima dovoljno novca u budžetu za njega i ako ima mjesta za poziciju koju igrač igra. Prilikom izlistavanja nogometaša napravljena je paginacija koja učitava po pet nogometaša. Također igrače je moguću filtrirati po poziciji.

Kod za dohvaćanje igrača po poziciji:

```
const dohvaceniIgraci = await
firebase.kolekcijaTimova.doc(this.odabraniTim.toString())
                        .collection('igraci')
                        .where('pozicija', '==', e.target.value)
                        .get();
```

Kada se nogometaš doda u korisnikovu ekipu budžet se umanjuje za njegovu cijenu, te se provjerava koliko igrača korisnik ima na toj poziciji. Ukoliko korisnik ima manje od dopuštenog broja igrača na toj poziciji na terenu odabrani nogometaš se sprema u prvih jedanaest igrača. Ukoliko korisnik ima popunjenu liniju na toj poziciji odabrani igrač se sprema na klupu. Izabrani igrač se može izbrisati iz formacije ili klupe. Nakon brisanja ažurira se budžet za cijenu nogometaša koji je izbrisan. Nogometaša koji je bio u prvih jedanaest, a korisnik ga je izbrisao, će zamijeniti igrač s klupe na toj poziciji, ako postoji.



Slika 44: Selekcija igrača

6.4.6. Upravljanje ekipom

Svaki korisnik koji se prijavio u bar jednu ligu može pristupiti upravljanju svoje ekipe. Na ovom dijelu aplikacije korisnik ima mogućnost ažuriranja formacije po kolu, mijenjanje koji igrač će igrati a koji biti na klupi, pregled statistike igrača, pregled tablice bodova u izabranoj ligi i mogućnost korištenja žetona za duple bodove za određeno kolo. Komponenta uključuje manje komponente kao što su statistika igrača, formacija i tablica bodova. Prilikom kreiranja komponente dohvaćaju se lige u kojima se korisnik natječe. Kada se dohvate lige onda se dohvaćaju podaci o korisnikovim odabranim igračima za tu ligu i na osnovu toga se slaže formacija. Korisnik ima dva padajuća izbornika, jedan za mijenjanje liga, a drugi za mijenjanje kola. Ako korisnik promijeni ligu, aplikacija šalje zahtjev na bazu podataka da dohvati podatke o nogometašima koje je korisnik odabrao za tu ligu. Kada se učitaju podaci o nogometašima ažurira se i formacija za odabranu ligu. Ako korisnik promijeni kolo komponenta formacije provjerava koji klupi su postavljeni na klupi a koji ne, te na osnovu toga ažurira formaciju. Ukoliko korisnik odluči promijeniti formaciju za to kolo to može odabrati iz ponuđenih opcija ispod formacije. Klikom na neku od formacija se pokreće akcija koja postavlja igrače na klupu ukoliko je to potrebno i ubacuje igrače s klupe u prvih jedanaest. Kod za promjenu formacije:

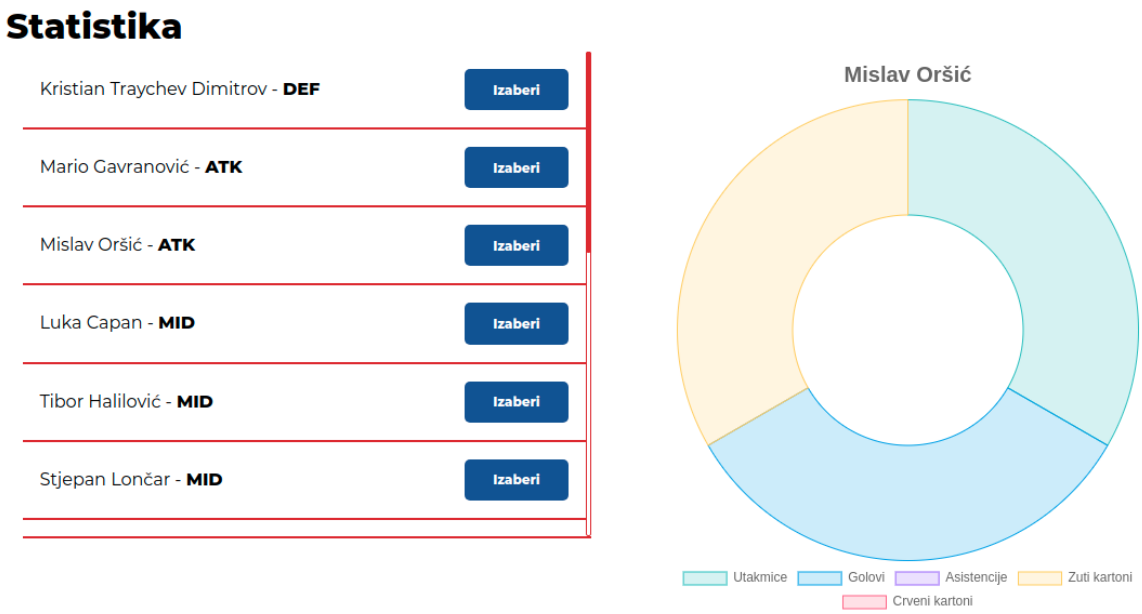
```
promijeniFormaciju: function(e) {
  const formacija = e?.target.value || this.korisnikovaFormacija;
  this.dohvatiStrukturuFormacije(formacija);

  this.$nextTick(() => {
    this.$store.dispatch('postaviFormaicju', {
      formacija,
      kolo: this.kolo.id,
      liga: this.liga.id,
      korisnikoviIgraci: this.korisnikoviIgraci }).then(() => {
        this.$forceUpdate();
    });
  })
}
```

Osim promjene formacije korisnik može promijeniti i igrača koji će igrati određeno kolo. Klikom na igrača s klupe pojavljuje se gumb kod igrača koji igraju u prvih jedanaest na toj poziciji. Klikom na gumb “Zamijeni” odabrani igrači će se zamijeniti. Korisnik također može iskoristiti žeton za duple bodove za odabrano kolo, što znači da igrači koji nisu bili na klupi to kolo će korisniku donijeti duple bodove. Korisnik ima pravo koristiti tri žetona na tri različita kola. Osim navedenih mogućnosti, korisnik prilikom odabira liga dobija opciju pregleda statistike igrača koje je odabrao. Igrači su prikazani u listi, gdje su prikazane informacije nogometaša, ime i pozicija koju igra te pokraj toga gumb za odabir igrača. Kada se igrač izabere pojavljuje se grafikon sa informacijama o nogometašu. Prikazuje se koliko je izabrani nogometaš odigrao utakmica, koliko je zabio golova, koliko je imao asistencija, koliko je dobio žutih i crvenih kartona. Ako nogometaš nema zabilježenih statističkih podataka ispisuje se prikladna poruka.

Za prikazivanje statistike korištena je biblioteka *Chart.js*. Za rad sa grafikonom potrebno je kreirati objekt sa klasom iz biblioteke. Konstruktoru je potrebno proslijediti objekt s informacijama o vrsti grafikona, etiketama koje će se koristiti, naslovu, bojama koje će se koristiti, informacije o pozicioniranju, legendi i slično. Kod za grafikon u projektu:

```
this.grafikon = new Chart(this.$refs.chart, {
  type: 'doughnut',
  data: {
    labels: ['Utakmice', 'Golovi', 'Asistencije', 'Zuti kartoni', 'Crveni
kartoni'],
    datasets: [{
        data: [0, 0, 0, 0, 0],
        backgroundColor: [
          'rgba(75, 192, 192, 0.2)',
          'rgba(54, 162, 235, 0.2)',
          'rgba(153, 102, 255, 0.2)',
          'rgba(255, 206, 86, 0.2)',
          'rgba(255, 99, 132, 0.2)'
        ],
        borderColor: [
          'rgba(75, 192, 192, 1)',
          'rgba(54, 162, 235, 1)',
          'rgba(153, 102, 255, 1)',
          'rgba(255, 206, 86, 1)',
          'rgba(255, 99, 132, 1)'
        ],
        borderWidth: 1
    }]
  },
  options: {
    responsive: true,
    plugins: {
      legend: {
        position: 'bottom'
      },
      title: {
        text: '',
        align: 'center',
        display: true,
        font: {
          size: 20
        }
      }
    }
  }
});
```



Slika 45: Statistika igrača

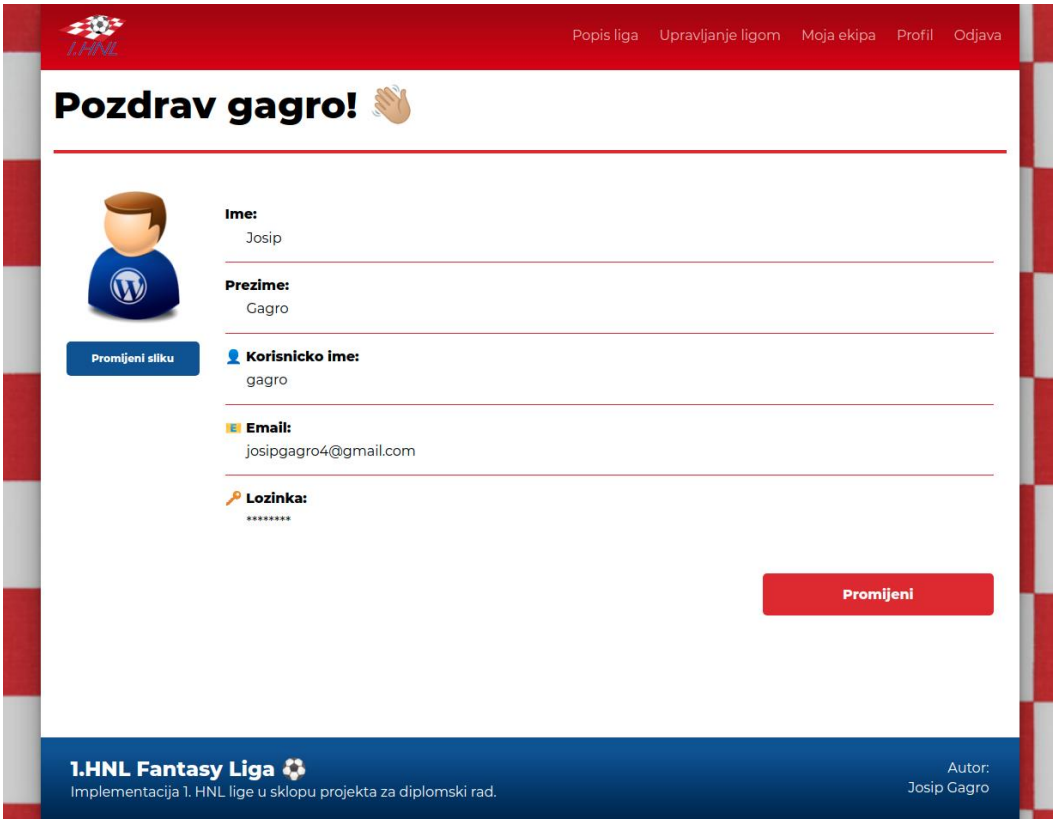
6.4.7. Uređivanje profila

Svaki prijavljeni korisnik ima pravo pristupa uređivanju profila. Na toj stranici korisnik može ažurirati podatke o sebi, kao što su ažuriranje imena i prezimena, korisničkog imena, mijenjanje lozinke ili ažuriranje profilne slike. Komponenta se sastoji od dva stanja, prezentacijskog i funkcionalnog. Prezentacijski dio komponente, kako i samo ime govori, prikazuje informacije o korisniku. Prezentacijsko stanje je stanje koje dočekuje korisnika prilikom posjećivanja stranice uređivanja profila. Funkcionalni dio se otvara klikom na gumb “Promijeni”. Nakon toga se otvara forma za unos podataka, gdje su uneseni korisnički podaci. Kao i kod prethodno opisanih formi i ova forma sadrži validaciju elemenata forme. Nakon što korisnik unese podatke koje želi ažurirati korisnik mora unijeti i lozinku kako bi potvrdio promjene. Ukoliko korisnik ne unese lozinku dobiti će obavijest i promjene se neće spremiti. Prilikom promjene lozinke korisnik mora unijeti staru i novu lozinku. Uvjet za lozinku je minimalno osam znakova, kao i kod registracije. Metoda za ažuriranje podataka:

```
await firebase.kolekcijaKorisnika.doc(this.trenutniKorisnik.id).update({
  ime: this.ime ? this.ime : '',
  prezime: this.prezime ? this.prezime : '',
  korisnickoIme: this.korisnickoIme,
  email: this.email,
 lozinka: this.novaLozinka.length > 0 ? this.novaLozinka :
this.trenutniKorisnik.lozinka
});
```

Korisnik također može promijeniti profilnu sliku. Klikom na gumb “Promijeni sliku” korisniku se otvara izbornik, gdje može odabrati sliku sa računala. Nakon što potvrdi odabir ažurira se korisnikova profilna slika. Kod za promjenu slike:

```
ucitajSliku: async function(e) {
  try {
    this.$store.dispatch('postaviCekanje', true);
    const file = e.target.files[0];
    const profilSlikaRef =
firebase.storage.ref().child(`${this.trenutniKorisnik.id}/profilePic`);
    await profilSlikaRef.put(file);
    const url = await profilSlikaRef.getDownloadURL();
    await
firebase.kolekcijaKorisnika.doc(this.trenutniKorisnik.id).update({
      imgUrl: url
    });
    const korisnik = await
firebase.kolekcijaKorisnika.doc(this.trenutniKorisnik.id).get();
    this.$store.dispatch('postaviKorisnikovProfil', {korisnickiPodaci:
korisnik.data(), korisnikoveLige: this.trenutniKorisnik.lige});
    this.$store.dispatch('postaviCekanje', false);
  } catch(error) {
    this.$store.dispatch('pokaziObavijest', {naslov: 'Dogodila se greska',
info: 'Molimo pokusajte ponovno', uspjeh: false});
    this.$store.dispatch('postaviCekanje', false);
  }
}
```

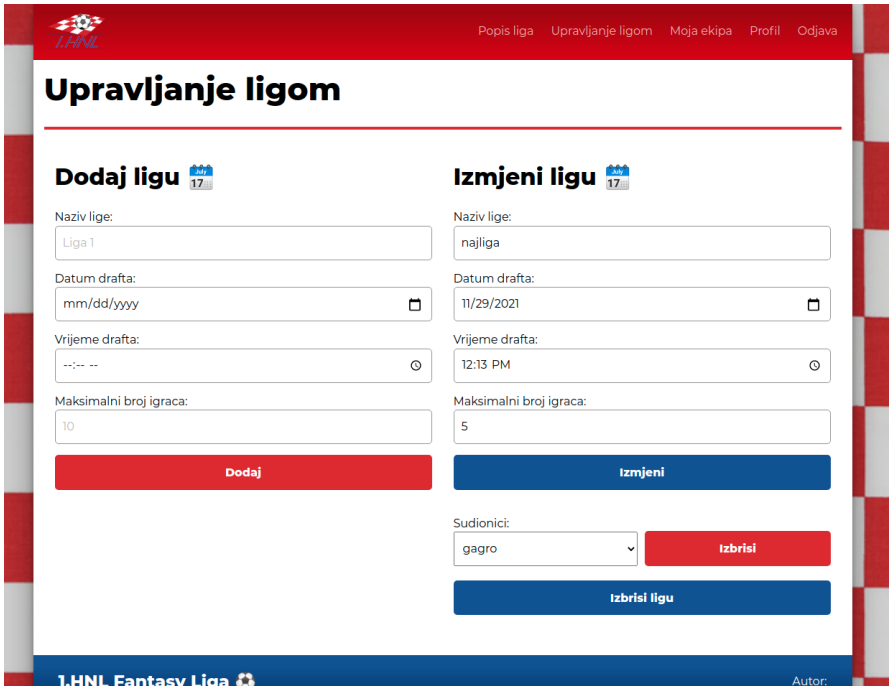


Slika 46: Upravljanje profilom

6.4.8. Upravljanje ligom

Upravljati ligom mogu moderatori i administratori. Stranica se sastoji od dvije forme. Prva forma je za dodavanje lige i onda je uvijek vidljiva. Druga forma je vidljiva ukoliko postoji parametar identifikacijske oznake u poveznici, odnosno ukoliko se izmjenjuju podaci već kreirane lige. Za svaki element forme je implementirana validacija, gdje svako polje je obavezno te dodatna provjera za broj igrača koji ograničava korisnika da unese broj veći od deset. Osim što se mogu ažurirati podaci o ligi također je moguće izbrisati sudionike lige, te je moguće izbrisati i samu ligu. Ukoliko postoji parametar identifikacijske oznake lige prilikom kreiranja komponente ona se koristi za dohvaćanje svih podataka o ligi, sve informacije o njoj i njenim sudionicima. Postoje tri gumba od kojih svaki ima svoju akciju, a to su gumb za izmjenu lige, gumb za brisanje sudionika gdje se briše sudionik izabran iz padajućeg izbornika i gumb za brisanje lige. Kod za ažuriranje lige:

```
azurirajLigu: async function(podaci) {
  try {
    this.$store.dispatch('postaviCekanje', true);
    await firebase.kolekcijaLiga.doc(this.prosljedenaLiga).update(podaci);
    this.$store.dispatch('postaviCekanje', false);
    this.$store.dispatch('pokaziObavijest', {naslov: 'Obavijest', info:
'Uspjesno ste azurirali ligu!', uspjeh: true});
  } catch (error) {
    this.$store.dispatch('postaviCekanje', false);
    this.$store.dispatch('pokaziObavijest', {naslov: 'Dogodila se greska',
info: 'Molimo pokusajte ponovno', uspjeh: false});
  }
}
```



Slika 47: Upravljanje ligom

6.4.9. Upravljanje korisnicima

Upravljati korisnicima može samo administrator. Ovo je relativno jednostavna stranica koja se sastoji izbornika i elementa za unos brojeva i dva gumba. Prilikom kreiranja stranice dohvate se korisnici aplikacije i spremaju se u listu izbornika. Kada se odabere korisnik koji se želi ažurirati pojavi se gumb za brisanje korisnika i polje za unos podataka i njegov gumb za ažuriranje vrste korisnika. Klikom na gumb izvodi se navedena akcija na gumbu. Kod za ažuriranje vrste korisnika:

```
await firebase.kolekcijaKorisnika.doc(this.odabraniKorisnik).update({
  vrsta: this.vrsta
}).then(() => {
  this.$store.dispatch('pokaziObavijest', {naslov: 'Akcija uspjesna!', info:
'Korisnik je azuriran', uspjeh: true});
})
```

Slika 48: Upravljanje korisnicima

6.4.10. Upravljanje kolima

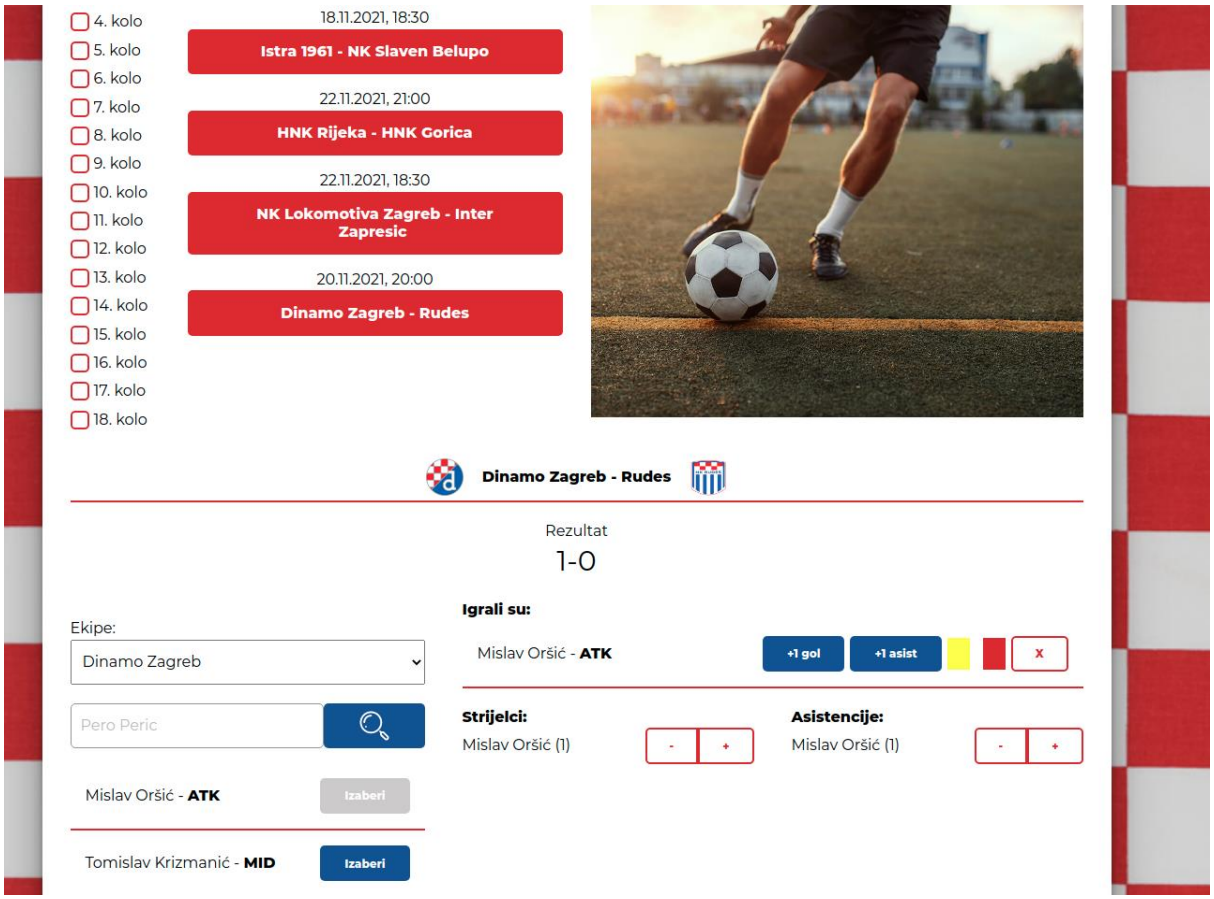
Upravljanje kolima je vidljivo i dostupno samo administratorima. Tu administrator evidentira rezultate utakmica i podatke o nogometašima. Podaci koji se bilježe su igranje utakmice, golovi, asistencije, žuti i crveni kartoni. Kada administrator dođe na stranicu vidljivo mu je sedamnaest kola. Klikom na kolo prikazuju mu se utakmice koje se igraju to kolo, ekipe koje igraju utakmicu i vrijeme utakmice. Klikom na utakmicu otvara se podsekcija koja sadrži padajući izbornik za odabir kluba i lista igrača odabranog kluba. Promjenom kluba mijenja se i lista igrača. Iz liste je moguće odabrati igrača koji je igrao utakmicu. Za listu igrača postoji opcije pretraživanja igrača po imenu. Kada se igrač izabere, prikazuje se u listi igrača koji su igrali utakmicu. Gumb za dodavanje kod odabranog igrača je onesposobljen. U listi odabranih igrača svakom igraču se pojavljuju gumbovi za dodavanje golova, asistencija, žutih i crvenih kartona, te gumb za brisanje igrača iz liste. Ako korisnik doda igraču gol ili asistenciju igrač se dodaje u posebnu listu za golove i asistencije, gdje korisnik ima mogućnost povećavanja ili smanjivanja broja golova i asistencija. U toj listi kraj imena igrača stoji i informacija koliko je igrač dao golova ili asistencija. Kada administrator doda gol igraču, automatski se ažurira rezultat utakmice. Kod za dodavanje bodova:

```
igrac.bodovi = igrac.bodovi ? igrac.bodovi : 0;

switch(kategorija) {
  case 'asistencije': igrac.bodovi += 3; break;
  case 'golovi':
    switch(this.odabraniTimIgraci.find(_igrac => _igrac.id ==
igrac.id).pozicija) {
      case 'Veznjak':
        igrac.bodovi += 5; break;
      case 'Napadac':
        igrac.bodovi += 4; break;
      default:
        igrac.bodovi += 6; break;
    } break;
}

igrac[kategorija] = igrac[kategorija] ? igrac[kategorija] + 1 : 1;
this.$set(listaIgraca, listaIgraca.indexOf(igrac), igrac);
this.zapisiPodatakeUFirebase(igrac);
```

Kao što je navedeno prethodno administrator ima mogućnost dodavanja žutih i crvenih kartona. Ako korisnik doda žuti karton igraču, prikazuje se informacija na gumbu da je već dodan žuti karton. Ukoliko korisnik klikne još jednom na taj gumb dodaje se crveni karton, a žuti karton se onesposobljava. Crveni karton je moguće igraču dati i direktno. Crveni karton je moguće poništiti. Svakim dodavanjem ili oduzimanjem navedenih stavki se ažuriraju i bodovi i statistika igrača, koja se kasnije koristi za bodovanje korisnika koji sudjeluju u ligama.



Slika 49: Upravljanje kolima

7. Zaključak

Atomski sustav dizajna je sustavni pristup razvoja komponenti na korisničkoj strani koji ubrzava i olakšava rad timovima koji rade na razvoju proizvoda. Njegova prisutnost pozitivno utječe na timove raznih opisa odgovornosti, od dizajnera, programera pa do menadžera. Odlikuje ga modularnost, skalabilnost i iterativni pristup razvoju, te sa svojom sveprisutnošću smanjuje šum u komunikaciji između timova.

U radu je opisan atomski sustav dizajna i njegova implementacija putem Patternlab-a, Vue.js-a i Firebase-a. Navedene tehnologije se odlikuju jednostavnošću učenja i uporabe, odličnom dokumentacijom i zajednicom programera koja je uvijek voljna pomoći i odgovoriti na postavljena pitanja. Još jedna odlika ovih tehnologija je odlična kompatibilnost jedne s drugom. S pomoću opisanih tehnologija opisan je proces nastanka dizajna sustava i način na koji se sustav gradio od atoma, molekula i organizama do predložaka i stranica, pa sve do gotove aplikacije.

Iz svega navedenog u diplomskom radu može se zaključiti kako je atomski sustav dizajna jako koristan i poželjan za uključiti u projekte koji imaju veliku kompleksnost i na kojem radi veliki broj ljudi. Razbijanjem kompleksnosti na manje dijelove pojednostavljuje se dizajniranje i razvoj proizvoda. S razbijanjem na manje dijelove dizajneri se mogu fokusirati na korisnike i njihove potrebe i želje, a programeri se mogu fokusirati na čist kod i slijediti standarde, a ne žuriti isporučiti kod što prije, gdje svaki programer radi na svoj način. S načinom rada koji osigurava atomski sustav dizajna puno brže se dogovaraju, dizajniraju i kreiraju nova sučelja, što današnje vrijeme očekuje – brže, veće, bolje. Osim toga sa sustavnim načinom rada i postavljenim pravilima rada puno je manja šansa za grešku, jer omogućava razvoj vođen testovima, te osim toga greške se mogu puno prije i puno lakše otkriti. Sa svim pozitivnim stvarima javlja se problem resursa i vremena koje je potrebno uložiti u njegov razvoj. Planiranje atomskog sustava dizajna može biti dugotrajan proces, kojim je potrebno pokriti veliki broj scenarija gdje i kako se komponente mogu koristiti. Osim planiranja potrebno je izdvojiti ljude i resurse koji će raditi na sustavu dizajna i njegovu održavanju, te zbog toga mali projekti koji se sastoje od nekoliko komponenti i na kojem radi nekoliko ljudi vjerojatno neće uključiti atomski dizajn sustava jer omjer dobivenog i uloženog nije velik i vidljiv kao kod većih i kompleksnijih projekata.

8. Popis literature

[1] D. Kermek, »elfarchive1617.foi.hr,« 2017. [Mrežno]. Available: https://elfarchive1617.foi.hr/pluginfile.php/3320/mod_resource/content/7/predavanja/Kermek_WebDiziProg_01.pdf. [Pokušaj pristupa 28. Travanj 2020].

[2] T. Berners-Lee, »Information Management: A Proposal,« Ožujak 1989. [Mrežno]. Available: https://cds.cern.ch/record/369245/files/dd-89-001.pdf. [Pokušaj pristupa 21 Listopad 2021].

[3] S. Rulkus, »Froont.com,« 4 Prosinac 2014. [Mrežno]. Available: https://blog.froont.com/brief-history-of-web-design-for-designers/. [Pokušaj pristupa 28 Ožujak 2020].

[4] A. Kholmatova, Design Systems, Freiburg: Smashing Media AG, 2017.

[5] B. Frost, Atomic Designs, Pittsburgh: Brad Frost, 2016.

[6] A. Konaté, »Design Systems at Work: Optimizing design processes and aligning design work to company identity,« Aalto University, 2018.

[7] B. Frost, »Youtube,« 08 11 2016. [Mrežno]. Available: https://www.youtube.com/watch?v=W-h1FtNYim4&t=46s&ab_channel=AnEventApart. [Pokušaj pristupa 14 Veljača 2021].

[8] B. Muenzenmeyer, »Smash Magazine - How To Make And Maintain Atomic Design Systems With Pattern Lab 2,« 13 07 2016. [Mrežno]. Available: https://www.smashingmagazine.com/2016/07/building-maintaining-atomic-design-systems-pattern-lab/. [Pokušaj pristupa 07 Svibanj 2020].

[9] R. F. Augusdi, A. A. Yunanto i D. I. Permatasari, »Development of Sandbox English Conversation Training Applications with Atomic Design,« *2021 International Electronics Symposium (IES)*, pp. 55-61, 2021.

[10] N. Le, »Creating software component using atomic design and test-driven development,« 2017.

[11] MDN Web Docs, »HTML: HyperText Markup Language,« Mozilla, [Mrežno]. Available: https://developer.mozilla.org/en-US/docs/Web/HTML. [Pokušaj pristupa 08 Rujan 2021].

[12] P. Pedamkar, »Versions of Html,« [Mrežno]. Available: https://www.educba.com/versions-of-html/. [Pokušaj pristupa 14 Rujan 2021].

[13] MDN Web Docs, »CSS: Cascading Style Sheets,« Mozilla, [Mrežno]. Available: https://developer.mozilla.org/en-US/docs/Web/CSS. [Pokušaj pristupa 17 Rujan 2021].

[14] B. Bos, »Brief history of CSS,« W3C, 17 Prosinac 2016. [Mrežno]. Available: https://www.w3.org/Style/CSS20/history.html. [Pokušaj pristupa 01 Prosinac 2021].

[15] Sass, »Sass: Syntactically Awesome Style Sheets,« [Mrežno]. Available: https://sass-lang.com/guide. [Pokušaj pristupa 15 Rujan 2021].

[16] E. You, »Vue.js,« [Mrežno]. Available: https://vuejs.org/v2/guide/. [Pokušaj pristupa 12 Rujan 2021].

[17] E. You, »The Vue instance,« [Mrežno]. Available: https://vuejs.org/v2/guide/instance.html#Lifecycle-Diagram. [Pokušaj pristupa 20 Rujan 2021].

[18] »State of JS,« 2021. [Mrežno]. Available: https://2020.stateofjs.com/en-US/technologies/front-end-frameworks/. [Pokušaj pristupa 23 Rujan 2021].

[19] E. You, »Comparison with Other Frameworks,« [Mrežno]. Available: https://vuejs.org/v2/guide/comparison.html. [Pokušaj pristupa 15 Rujan 2021].

[20] Google, »Firebase Documentation,« [Mrežno]. Available: https://firebase.google.com/docs/. [Pokušaj pristupa 03 Listopad 2021].

[21] Google, »Firebase Authentication,« [Mrežno]. Available: https://firebase.google.com/docs/auth/. [Pokušaj pristupa 06 Listopad 2021].

[22] Google, »Cloud Firestore,« [Mrežno]. Available: https://firebase.google.com/docs/firestore/. [Pokušaj pristupa 04 Listopad 2021].

[23] H. Vera, W. Boaventura, M. Holanda, V. Guimaraes i F. Hondo, »Data Modeling for NoSQL Document-Oriented Databases,« *CEUR Workshop Proceedings*, svez. 1478, pp. 129-135, 2015.

[24] »Web Design Museum,« [Mrežno]. Available: https://www.webdesignmuseum.org/web-design-history. [Pokušaj pristupa 28 Ožujak 2020].

[25] J. Talkar, »Code Project,« 25 Svibanj 2000. [Mrežno]. Available: https://www.codeproject.com/Articles/571/Java-to-JavaScript-Communication. [Pokušaj pristupa 01 Prosinac 2021].

9. Popis slika

10. Popis tablica