SVEUČILIŠTE U ZAGREBU

GRAFIČKI FAKULTET

MARIO LONČAREK

RAZVOJ PARAMETARSKI PRILAGODLJIVOG GRAFIČKOG KORISNIČKOG SUČELJA WEB APLIKACIJE

DIPLOMSKI RAD

Zagreb, 2020.

SVEUČILIŠTE U ZAGREBU

GRAFIČKI FAKULTET

MARIO LONČAREK

RAZVOJ PARAMETARSKI PRILAGODLJIVOG GRAFIČKOG KORISNIČKOG SUČELJA WEB APLIKACIJE

DIPLOMSKI RAD

Mentor: doc. dr. sc. Tibor Skala

Student: Mario Lončarek

Zagreb, 2020.

Rješenje o odobrenju teme diplomskog rada

SAŽETAK

Prosječni korisnik stekao je naviku kako bi neko korisničko sučelje trebalo izgledati. Iako se korisnička sučelja možda vizualno razlikuju, ona više manje prate iste principe i najbolje prakse. O tome ovisi i uspjeh nekog softvera, jer korisničko sučelje mora korisniku olakšati korištenje, bez da ga tjera da razmišlja. No korisnici mogu biti različiti i ne moraju se nužno ista pravila primjenjivati na sve korisnike. Pod pojmom korisnika smatraju se različite ciljane skupine (dob, spol, zanimanje, hobi,…) a svaki od njih drugačije koristi i očekuje drugačije ishode koristeći grafičko korisničko sučelje. To dovodi do toga da univerzalno rješenje sučelja koje je prilagođeno samo jednoj ciljanoj skupini automatizmom otežava orijentaciju unutar sučelja ostalim skupinama. Iz ovoga se može zaključiti da na tržištu postoji prostor i potreba da se korisničko sučelje prilagođava svakom korisniku zasebno, temeljeno na dobi, spolu, zanimanju ali i osobnim preferencijama korisnika. Ne postoji proizvod koji je napravljen za apsolutno svakoga, ali se korisnicima može uvelike olakšati korištenje proizvoda, u ovom slučaju web aplikacije. Stoga ovaj rad istražuje i kako bi se korisnicima moglo omogućiti da personaliziraju svoje iskustvo svojim potrebama te kako bi funkcionirao koncept koji bi korisniku to omogućio. Koncept je napravljen u tehnologijama za razvijanje web aplikacija, pokretan React.js-om kao jezgrom aplikacije.

Ključne riječi: Prilagodljivo grafičko sučelje, Javascript, React.

ABSTRACT

The average user has become accustomed to what a user interface should look like. Although user interfaces may be visually different, they all more or less follow the same principles and best practices. The success of a software also depends on it, because the user interface has to be easy to use, without making the user think. But users can be different and not all rules have to apply for all the users. The term user is considered to be different target groups (age, gender, occupation, hobby, ...) and each of them uses and expects different outcomes using a graphical user interface. This leads to the fact that a universal interface solution that is adapted to only one target group automatically makes it difficult for other groups to navigate within the interface. From this it can be concluded that there is space in the market and the need to adapt the user interface to each user separately, based on age, gender, occupation or personal preferences of the user. There is no product that is made for absolutely everyone, but users can be greatly facilitated by using the product in this case web applications. Therefore, this paper also explores how users could be able to personalize their experience to their needs in order for the concepts that the user could enable to function. The concept is made in web application development technology, running React.js as the core of the application.

Keywords: Customizable graphical interface, Javascript, React.

SADRŽAJ

1. UVOD

Koristeći razne aplikacije u svakodnevnom okruženju, korisnici su naviknuti na grafičko korisničko sučelje (GUI) koji se od sustava do sustava po određenim zahtjevima razlikuje, ali se u principu drži osnovnih standarda arhitekture sučelja. Sami uspjeh softverskog proizvoda ovisi upravo o tome kako GUI komunicira s korisnikom i olakšava korištenje njegovih različitih značajki. Dodirne točke grafičkog sučelja su prostor u kojem se odvija interakcija čovjeka i računala. Pod pojmom korisnici ulaze sve dobne skupine; djeca, mladi, odrasli i stariji. Sučelje univerzalnog sadržaja koje je prilagođeno samo jednoj ciljanoj skupini automatizmom otežava orijentaciju unutar sučelja drugim skupinama. Stoga se javlja potreba korisničkog sučelja temeljenog na dobi korisnika u mobilnom operativnom sustavu koje bi se sastojalo od početnog zaslona za odabir preferencija koji se dalje usmjerava na sučelje prilagođeno dobi korisnika. Dizajn treba korisniku omogućiti fluidnu, prirodnu interakcije, ne ometajući iluziju djelovanja u prostoru i vremenu. Ako je rezultat dizajna uspješan, onda se može manipulirati korisničkim doživljajem usporavanjem i ubrzavanjem protoka vremena.

Gradeći prilagodljivo korisničko sučelje pomoću tehnologije Javascript (React) u procesu oblikovanja trebaju se u obzir uzeti određeni principi i mehanizmi vizualne percepcije. Idejno rješenje se temelji na mogućnosti dizajnera da po parametrima koje je prethodnim istraživanjem prikupio, definira granice varijabilnih dijelova dizajna. Vrijednost koja se nalazi između tih gabarita dizajna, određuje korisnik. Ne postoji proizvod koji je napravljen za apsolutno svakoga, ali se korisnicima može uvelike olakšati korištenje proizvoda, u ovom slučaju web stranice odnosno web aplikacije**.**

2. KORISNIČKA SUČELJA

Prema Bachelardu, mijenjajući prostor, a ostavljajući svoje uobičajene osjetilnosti, mi ulazimo u komunikaciju s prostorom koji je fizički zanimljiv i nov. No, ne mijenjamo mjesto, već mijenjamo vlastitu prirodu.[1] *33 str* Mobilni telefoni postaju nezamjenjivi dio svakog kućanstva. Služe kao zidni sat, budilica, kalkulator, kalendar i još mnogo toga, te svaka od tih višenamjenskih funkcija ima svoje sučelje nove generacije mobilnih telefona. Mladi korisnici su se dobro prilagodili višestrukim funkcionalnim grafičkim korisničkim sučeljem, ali se s vremenom pokazalo da sučelje mora biti jednako dobro funkcionalno za dvije dobne skupine, tj. za starije osobe i djecu. U ovom istraživanju se uzima više dobnih skupina s obzirom na opseg dobi uzorka ispitanika. Sučelje kao takvo može od mobitela nove generacije napraviti beskoristan ili vrlo malo koristan uređaj starijim osobama i djeci. To dovodi do sljedećeg zaključka - potreba korisničkog sučelja temeljenog na dobi u mobilnom operativnom sustavu koje će se sastojati od početnog zaslona za odabir sučelja koji se dalje usmjerava na sučelje prilagođeno dobi korisnika.

Internet kao globalna mreža je promijenila polje djelovanja ljudskog faktora i postala je nezaobilazni dio svakidašnjeg života, bilo da se koristi TV, mobilni telefon, računalo ili neki drugi oblik multimedijalnog sustava.
Prema knjizi *Mismatch* autorice Kat Holmes, razlaže se kontroverzna tema o ljudima koji oblikuju dodirne točke društva koji zapravo određuju tko može sudjelovati i tko je izostavljen, često nesvjesno. Isto to se preslikava na dizajn dodirnih točaka u komunikacijskom kanalu između čovjeka i medija, gdje grafičko sučelje postaje površina koja predstavlja niz dodirnih točaka. Grafičko korisničko sučelje predstavlja prostor u kojem se odvija interakcija čovjeka i računala. Je li dizajn dovoljno prilagođen svima? Na svijetu ne postoji proizvod koji je napravljen za svakoga.

[1] "Poetika prostora - Gaston Bachelard - Google Books." https://books.google.com/books/about/Poetika_prostora.html?id=gt7kAAAACAAJ. Pristupano 4 srpnja. 2020.

Proizvod se dizajnira isključivo za ljude kakvi jesu, a ne kakve dizajneri žele da budu [2]

2.1 DIZAJN KORISNIČKIH SUČELJA

Postoje dva svojstva digitalnog okruženja koji omogućavaju dinamički doživljaj informacije. To su interaktivnost i osjećaj uživljavanja *(immersion).* Sposobnost korisnika da aktivno sudjeluju u radnjama koje se odvijaju u prezentacijskom okružju elektronskoga područja nazivamo interaktivnost. Uska povezanost i orkestracija između vizualnih, kinestetičkih i auditivnih modaliteta ključna je za osjećaj uživljavanja i interaktivnosti. Ta svojstva zajedno čine računala snažnim sredstvom za prezentaciju i preuzimanje informacije. Također daju korisniku osjećaj navigacije prostorom, osjećaj blizak osjećaju naseljavanja i uporabe prostora. Cilj unapređenja računala kao medija, po mišljenju nekih, predstavljaju nastojanja da korisnik postane ne svjestan činjenice da se uopće služi medijem kako bi doživljaj postao što neposredniji i autentičniji. 216

Dizajn sučelja treba korisniku omogućavati fluidnu i prirodnu interakciju tako da ne ometa iluziju djelovanja u prostoru i vremenu. Sučelje koje je uspješno razrađeno, izaziva i manipulira korisnikovim doživljajem usporavanjem i ubrzavanjem protoka vremena.[3] S obzirom na to, dolazi se do pojma *usability designa* odnosno korisničkog iskustva.

Tehnologije današnjice omogućavaju lako ostvarivanje interakcije s korisnikom putem zaslona na dodir kao i putem već poznatih ulaznih jedinica (miš, tipkovnica, joystick itd.). Prednost interakcije putem zaslona na dodir jest intuitivnost i jednostavnost koju takav način interakcije pruža jer se upravljanje ICT (engl. “Information And Communication Technology”) proizvodom ili uslugom obavlja

[2] "ZAVRŠNI RAD." http://eprints.grf.unizg.hr/2744/1/Z821_Mareni%C4%87_Iva.pdf. Pristupano 4 srpnja. 2020.
[3] "DIZAJN DIGITALNOG PROSTORA, Jasenka Pribernik | Hrvatska ...." http://www.hsn.hr/product/184/dizajn-digitalnog-prostora-jasenka-pribernik. Pristupano 3 srp. 2020.

pomoću prstiju ili posebne olovke što je korisniku mnogo jednostavnije i prihvatljivije. Mogućnosti koje pruža zaslon na dodir najviše dolaze do izražaja kod malih, pokretnih uređaja gdje je ulazna jedinica integrirana u sam uređaj, čime se izmijenila uloga korisničkog sučelja koje postaje ulazna jedinica. Promjenom funkcije sučelja dolazi do potrebe za redizajnom kako bi se ostvarile njegove zadaće – funkcija prikaza i funkcija ulazne jedinice. Ovim redizajnom korisničko sučelje trebalo bi biti prilagođeno za prikaz na raznim platformama i uređajima kako bi se na svakom od njih prikazivalo jednako, odnosno zadržalo funkcionalne komponente jednakim.

To dovodi do zaključka da bi sučelje za starije trebalo sadržavati minimalističku funkcionalnost, ikone i tekst zbog slabog vida.

Vizualna percepcija je jedan od najproduktivnijih i najbržih načina na koji ljudi mogu dobiti informacije i dobiti ih obrađeni od strane mozga. To utječe na toliko aspekata života da bi zanemarivanje problema pri stvaranju proizvoda za korisnike bilo krajnje nepromišljeno. Zbog toga je aspekt primjene vizualnih elemenata visoke funkcionalnosti u sučeljima, kao što su ikone i njihov utjecaj na opću učinkovitost proizvoda, već dugo aktualna tema u globalnoj zajednici dizajna.

Veliki udio korisnika su po prirodi vizualna stvorenja, pa se često slijede mehanizmi vizualne percepcije koje treba uzeti u obzir u procesu oblikovanja:

2.2 POTREBA ZA DOBRIM KORISNIČKIM DOŽIVLJAJEM

Zahvaljujući današnjim tehnologijama, web stranice i mobilne aplikacije postaju sve složenije i bogatije funkcionalnostima stoga se stvara potreba za dobrim korisničkim doživljajem. Korisnicima je omogućen pristup internet stranicama s različitih uređaja. Kako bi se web stranice, a kasnije i mobilne aplikacije istaknule nakon svih tih promjena, morale su biti ugodne za korištenje, odnosno omogućiti pozitivno iskustvo korisniku koji ih koristi.

Uspjeh softverskog proizvoda uvelike ovisi o tome kako GUI komunicira s korisnikom i olakšava korištenje njegovih različitih značajki. Ako korisnici ne provedu mnogo vremena u učenju i razumijevanju kako funkcionira GUI-a, smatra se da je informacijski sustav ima vrlo učinkovit GUI. Kvalitetan GUI mora moći pristupiti zahtjevima korisnika i zadovoljiti njihove zahtjeve.

Tijekom života čovjek prikupi ogromnu količinu iskustva i razvije trenutno razumijevanje za neke simbole, oblike i boje. Potrebno je koristiti tu sposobnost prepoznavanja kod ljudi da se jasnije prenese poruka kroz sučelje. Ako postoji potreba za priopćavanjem neke važne poruke, koristit će se stvari koje ljudi razumiju bez da im se to objašnjava. Budući da smo svi toliko ovisni o našem iskustvu, sjećanju, prisjećanju i prepoznavanju, prirodno je shvatiti da ono što je razumljivo i trivijalno za jednu osobu može značiti nešto potpuno drugačije od nekoga drugoga. Razumijevanje i kombinacija ovih već ustaljenih obrazaca može povećati učinkovitost korištenja neke aplikacije, smanjiti troškove podrške, smanjiti broj prototipnih iteracija i smanjiti troškove testiranja upotrebljivosti.

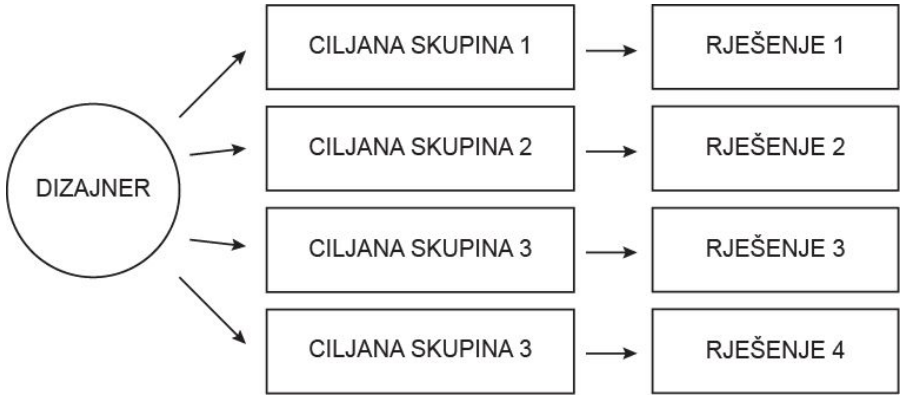
Digitalna pristupačnost odnosi se na praksu izgradnje digitalnog sadržaja i aplikacija koje mogu koristiti široki krug ljudi, uključujući pojedince koji imaju vizualne, motorne, slušne, govorne ili kognitivne poteškoće.[4] S obzirom na to da se radi o velikom broju ljudi, svaka od prethodno spomenutih aplikacija je uzela jednu skupinu ljudi s poteškoćama kao ciljanu skupinu. Ovaj rad će rješenje temeljiti na široj ciljanoj skupini te se neće ograničavati na njihove poteškoće već će se njima prilagođavati bez obzira o kojoj poteškoći se radi. Ovdje bi naglasak s korisnika i programa prešao na korisnika i dizajnera. Dizajner bi u ovom slučaju odredio maksimalne i minimalne granice dizajna koje bi odgovarale dizajnu sučelja, a program bi se prilagođavao korisniku unutar predodređenih granica.

Ciljana publika je specifična grupa ljudi kojima se proizvod, usluga ili brand prodaje na različite načine. U većini aspekata organizacije ili poslovanja, poznavanje vaše

[4] https://uxdesign.cc/designing-for-accessibility-is-not-that-hard-c04cc4779d94 Pristupano 6. srpnja 2020.

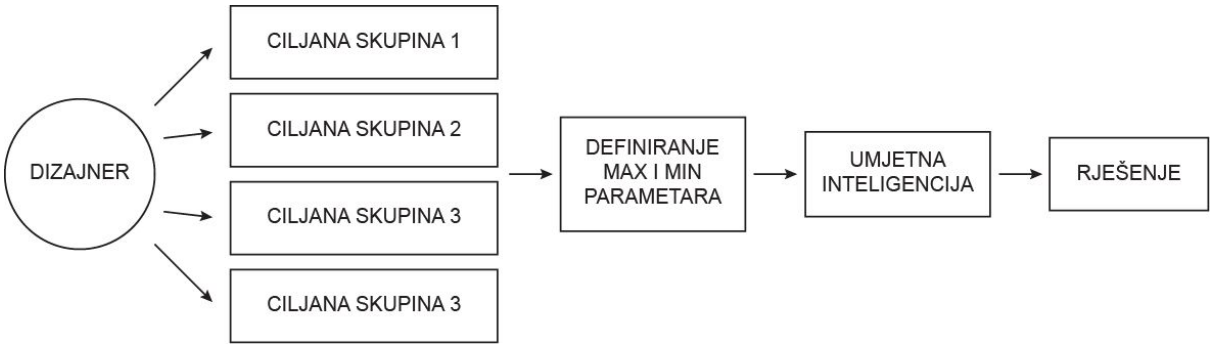
ciljane publike je ključno za isporuku odgovarajuće poruke ili rješenja. To je jednako važno i kreativnim i vizualnim odjelima. Stvaranje proizvoda za klijenta s raznolikom ciljanom publikom može biti izazov. Brojne su dobne skupine, nacionalnosti i spolovi koje treba uzeti u obzir, uz geografske razlike i mnoge druge brige.
Istraživanje ciljane skupine je lakše kada se uspostavi baza klijenata. Vjerni klijenti vjerojatno će biti spremni ispuniti kratku anketu o tome što funkcionira, a što ne, s vašim trenutnim ili budućim dizajnom. Međutim, postavljanje pravih pitanja je ključno. Potrebno je pitati trenutnu publiku o određenim dijelovima dizajna koji im govore i tako ih privući. Ako već nemate uspostavljene publike, onda se može istraživati slične proizvode i raditi stvari poput beta testiranja.

Otvaranje proizvoda globalnoj publici je odlična ideja, ali važno je prvo razmotriti kako će dizajn utjecati na tu publiku na temelju toga gdje se nalaze širom svijeta. S obzirom na to dizajner će se naći u neugodnoj situaciji gdje je primoran napraviti jednu od dvije stvari. Prvo je kompromis u dizajnu što negativno utječe na dizajnera koji gradi svoju reputaciju kroz radove koje objavljuje. Jednom kada loš dizajn izađe na vidjelo digitalnog svijeta, ne može se vratiti. Kao ni obraz dizajnera koji je dugo gradio. Druga opcija je da dizajner radi poseban dizajn za svaku ciljanu skupinu koja se nalazi u širokoj ciljanoj publici. Ovo rješenje zahtjeva mnogo vremena te nije garantirano da će biti izvedivo.



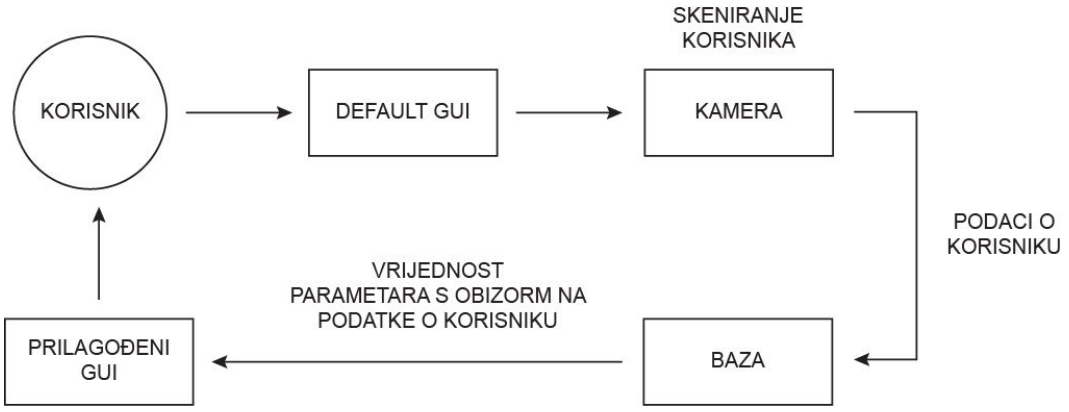
Slika br. 1: Dizajniranje posebno za svaku ciljanu skupinu

Dakle, oba rješenja imaju negativan ishod s obzirom na dizajnera. U ovom radu se pokušava pomoći dizajneru kako bi njegov proces dizajna bio učinkovit i ne toliko zahtjevan. Zbog čega se koncentrira na izradu rješenja gdje će dizajner moći na što jednostavniji način ugoditi širokoj ciljanoj skupini. **Idejno rješenje se temelji na mogućnosti dizajnera da po parametrima koje je prethodnim istraživanjem prikupio, definira granice varijabilnih dijelova dizajna.**



Slika br. 2: Dizajniranje za sve ciljane skupine

Svaki od tih parametara će idealno funkcionirati za svaku skupinu ovisno o tome iz koje skupine je trenutni korisnik.



Slika br. 3: Prilagodba GUI-a

Glavni problem današnjih dizajnera je olakšano korištenje pametnih uređaja svim korisnicima. Kreiranje dizajna kojeg mogu koristiti široke skupine korisnika, uključujući one koji imaju sve mogućnosti ili one koji imaju privremeno ili trajno oštećenje koje im otežava interakciju, je i dalje diskurs na kojem se radi.

3. UVOD U EKSPERIMENTALNI DIO RADA

Koncept ovog rada zamišljen je kao web aplikacija napisan u jeziku “Javascript” odnosno njegovoj biblioteci (engl. “Javascript library”) koji je napisan oko Javascripta a zove se React.js.
Aplikacija se može isprobati na sljedećoj poveznici:
https://mario-loncarek-diplomski.netlify.app/
Ova tehnologija izabrana je jer omogućuje jako brzu izradu koncepta uz najbolje performanse za korisnika, odnosno koristeći ovu tehnologiju i ove okvire korisniku je omogućeno da bez slanja zahtjeva prema poslužitelju korisnik može imati interakciju s ovom web aplikacijom, odnosno korisnik može mijenjati parametre te u realnom vremenu promatrati rezultate. Aplikacija se sastoji od početne te glavne stranice. Glavna se stranica aplikacije sastoji od modula koji imaju razne vrste različitog sadržaja kako bi se moglo što bolje promatrati promjene, te od samog konfiguratora kojeg korisnik može pozivati i micati s ekrana.
Osim React.js-a koji je glavni pokretač ove aplikacije korištene su i druge moderne tehnologije:

- Git/Github za kontrolu verzije i distribuiranje koda
- Netlify za posluživanje aplikacije korisnicima
- Create-react-app kao React.js starter za aplikaciju
- B-creative kao CSS starter za aplikaciju
- ESLint za analizu koda protiv grešaka
- Prettier za održavanje standardnog izgleda koda
- ES6 verzija Javascripta
- SCSS pretprocesor za CSS s uključenom funkcijom modula
- GSAP za animacije

3.1 ŠTO JE REACT.JS

React.js je JavaScript biblioteka otvorenog koda koja se koristi za izgradnju korisničkih sučelja, a posebno se koristi za SPA (engl. “single-page applications”) tip aplikacija, odnosno aplikacija s jednom stranicom.

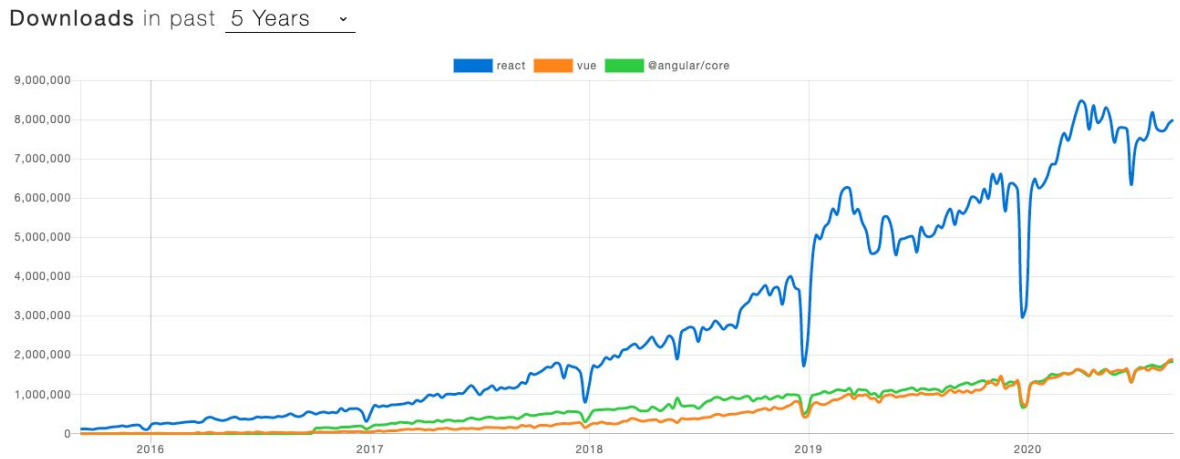
Aplikacija s jednom stranicom (SPA) je web aplikacija ili web stranica koja komunicira s web preglednikom dinamičkim prepisivanjem trenutne web stranice s novim podacima s web poslužitelja, umjesto standardnog načina na koji preglednik učitava cijele nove stranice na svaki zahtjev. Aplikacije s jednom stranicom (SPA) nastale su s ciljem da pružaju jako velike brzine učitavanja i tranzicija između stanja ili stranica. Pojednostavljeno, kod aplikacija s jednom stranicom kada korisnik potražuje novi dio ili cijelu novu stranicu, ne šalje se HTTP (engl. “Hypertext Transfer Protocol”) zahtjev prema web poslužitelju, već su sva stanja ili stranice zapisane u Javascriptu, uključujući njihov HTML (engl. “Hypertext Markup Language”) i CSS (eng “Cascading Style Sheets”) te se na zahtjev korisnika dijelovi ili cijela trenutna stranica zamjenjuje novim podacima, bez zahtjeva prema web poslužitelju (engl. “server”). U SPA-u preglednik dohvaća sav potreban HTML, JavaScript i CSS kod s jednim, početnim učitavanjem stranice ili se odgovarajući resursi dinamički učitavaju i dodaju na stranicu po potrebi, obično kao odgovor na zahtjeve korisnika. Stranica se ne treba ponovno učitati niti u jednom trenutku procesa. Hash lokacije ili API (engl. “application programming interface”) za povijest HTML5 mogu se koristiti za pružanje percepcije i navigacije odvojenih logičkih stranica u aplikaciji.[5]

React.js je nastao u timu developera iz Facebooka koji su ga koristili na Facebook i Instagram feedu, dok je inicijalna distribucija za javnost 2013 godine.[6] Održavaju ga Facebook i zajednica pojedinačnih programera i tvrtki. Ono što je zanimljivo u priči

[5] SPA (Single-page application) - https://developer.mozilla.org/en-US/docs/Glossary/SPA Pristupano 2 kolovoza 2020.
[6] https://en.wikipedia.org/wiki/React_(web_framework) Pristupano 2 kolovoza 2020.

oko React.js-a je to da on danas dominira Javascript svijetom, od popularnosti do traženosti poslova.

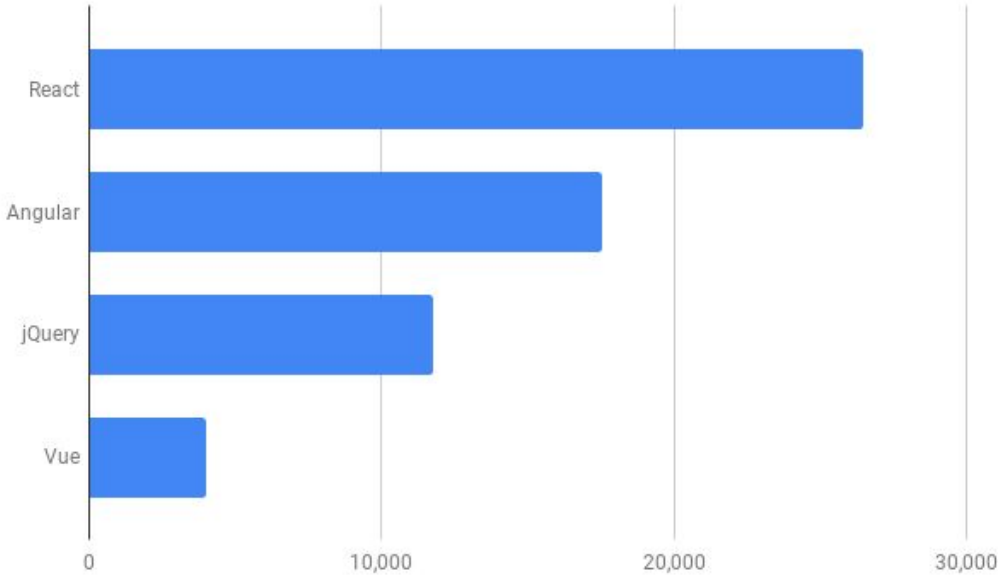
To govore i brojke pojedinačnih dnevnih preuzimanja React.js-a s NPM-a (engl. “Node package manager”) ako se usporedi s konkurencijom od kojih su najveći Vue.js te Googlov Angular.js.



Slika br. 4: Broj pojedinačnih skidanja React-a i konkurentskih biblioteka

Preuzeto: https://www.npmtrends.com/react-vs-vue-vs-@angular/core

Iz tog razloga se ovakva statistika preslikala i na brojke oglasa za posao na jednom od najvećih oglasnika za poslove “Indeed” ako ga se opet usporedi s glavnim konkurentima.



Slika br. 5: Broj oglasa za posao za React.js

Preuzeto:

https://medium.com/javascript-scene/top-javascript-frameworks-and-topics-to-learn-in-2020-and-the-new-decade-ced6e9d812f9

React.js danas koriste vjerojatno sve najveće i najbolje firme koje svoje poslovanje baziraju na web aplikacijama od kojih su neki: Netflix, Facebook, Instagram, Airbnb, Cloudflare, Dropbox, BBC, Flipboard, Imgur, Postmates, Reddit…

Njegova popularnost te konstantni rast popularnosti nije bez razloga, a neke od najvažnijih karakteristike koje ga čine popularnim su:[7]

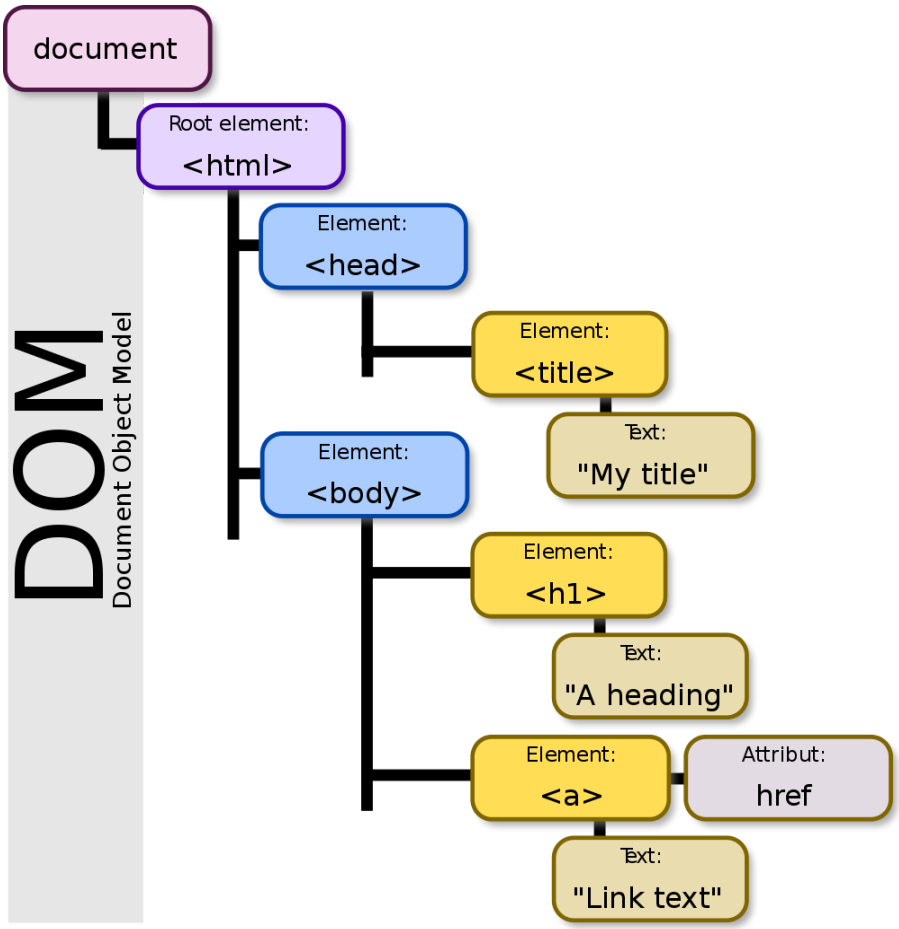
- jednostavno stvaranje dinamičkih aplikacija
- odvajanje koda u komponente i/ili module za višekratnu upotrebu
- brzina rada i izvršavanja
- mala brzina učenja (ako se razumije Javascript)
- deklarativno pisanje koda
- velika zajednica developera
- može se koristiti i za web i za mobilne aplikacije
- razvijeni alati za otklanjanje pogrešaka.

[7] https://www.simplilearn.com/what-is-react-article Pristupano 2 kolovoza 2020.

3.2 PRINCIP RADA REACT.JS-A

Za razvoj nekog softvera obično se koristi obično se koristi MVC (engl. “Model – View – Controller”) obrazac dizajna softverske arhitekture. Takva softverska arhitektura sastoji se od tri međusobno povezana elementa: model, pogled te kontroler. Model je mjesto gdje se pohranjuju podaci koje softver konzumira. Pogled je ono što se predstavlja korisnicima i kako korisnici komuniciraju s aplikacijom. Kontroler je donosilac odluka i spoj između modela i pogleda.[8] React.js možemo shvatiti kao element pogleda (engl. “view”) kada se shvati MVC arhitektura softvera. To znači da njega ne možemo zvati niti MVC niti Javascript okvirom (engl. “framework”). On jednostavno služi za manipulaciju DOM-a (engl. “Document Object Model”) odnosno sadržaja koji se prezentira korisniku. DOM je kolekcija objekata koje je preglednik stvorio od HTML-a te se njime onda može manipulirati. DOM Manipulacija je način na koji neka stranica ili aplikacija može postati interaktivna za korisnika.

[8] https://developer.chrome.com/apps/app_frameworks Pristupano 4 kolovoza 2020.



Slika br. 6: Vizualni prikaz DOM-a (engl. Document Object Model)

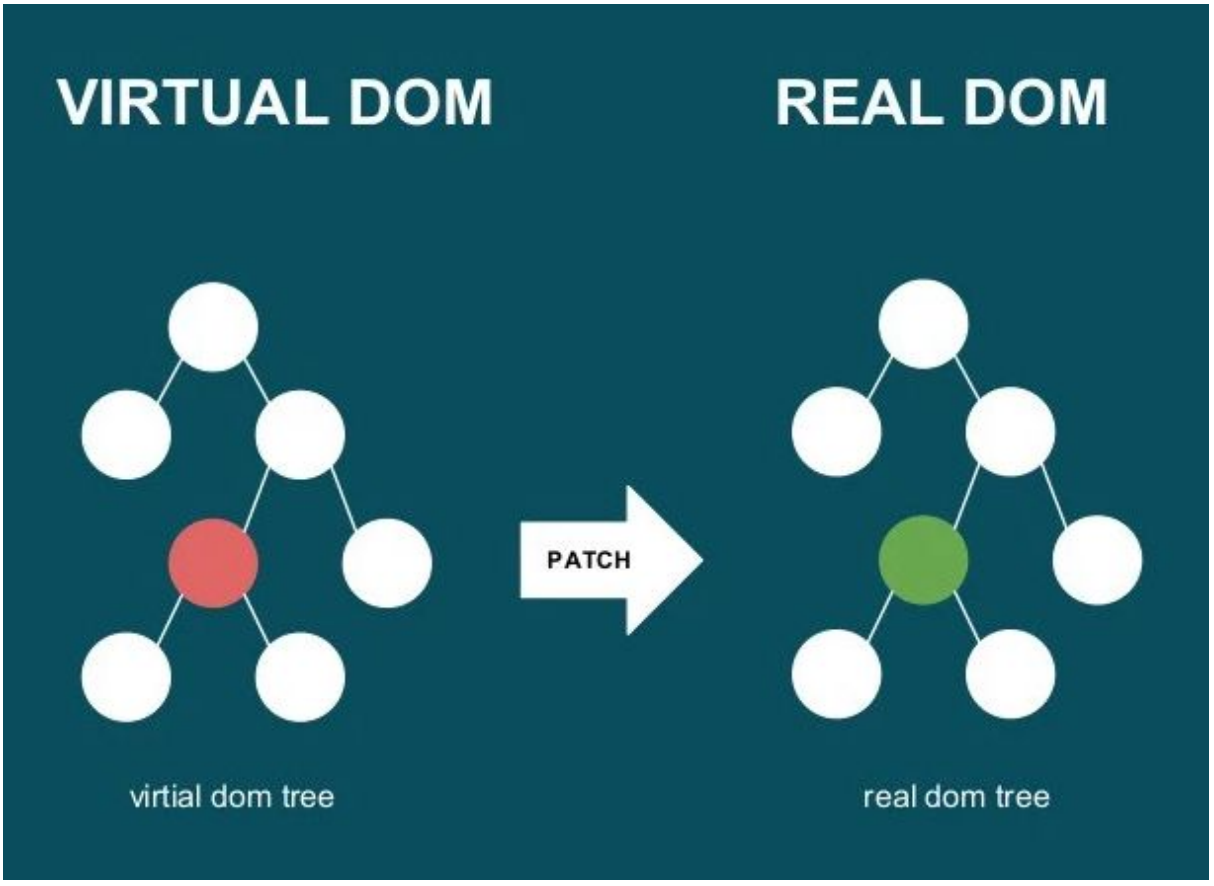
Preuzeto:

https://levelup.gitconnected.com/new-to-react-you-need-to-understand-these-key-concepts-before-anything-else-2247efc1eaac

No takva manipulacija je puno sporija od većine JavaScript operacija. Kako bi riješili ovaj problem, React.js implementira nešto što se naziva virtualnim DOM-om. Pošto je glavna značajka React-a upravo njegova brzina, React se razlikuje po tome što implementira virtualni DOM koji je u osnovi prikaz stabla DOM-a u Javascriptu. Odnosno svaki DOM objekt ima odgovarajući virtualni DOM objekt. Dakle, kada treba čitati ili pisati u DOM, on će zapravo koristiti svoj virtualni DOM.[9] Tada će virtualni DOM pokušati pronaći najučinkovitiji način za ažuriranje pravog DOM-a preglednika. Za razliku od DOM elemenata preglednika, React elementi su obični objekti i jeftini su za stvaranje. Virtualni DOM brine se o ažuriranju DOM-a kako bi

[9] https://reactjs.org/docs/faq-internals.html Pristupano 4 kolovoza 2020.

odgovarao React elementima. Upravo iz tog razloga ima smisla imati virtualni DOM jer Javascript može brže obraditi podatke te je zbog istog razloga React izuzetno brz i za korisnika neprimjetan. To znači da se aplikacija učitava samo jednom na inicijalnom učitavanju, dok nadalje nema HTTP zahtjeva već se sve izvršava na strani klijenta pomoću Javascripta. Osim izvršavanja na korisničkoj strani, React podržava i izvršavanje na strani poslužitelja.



Slika br. 7: Virtualni DOM

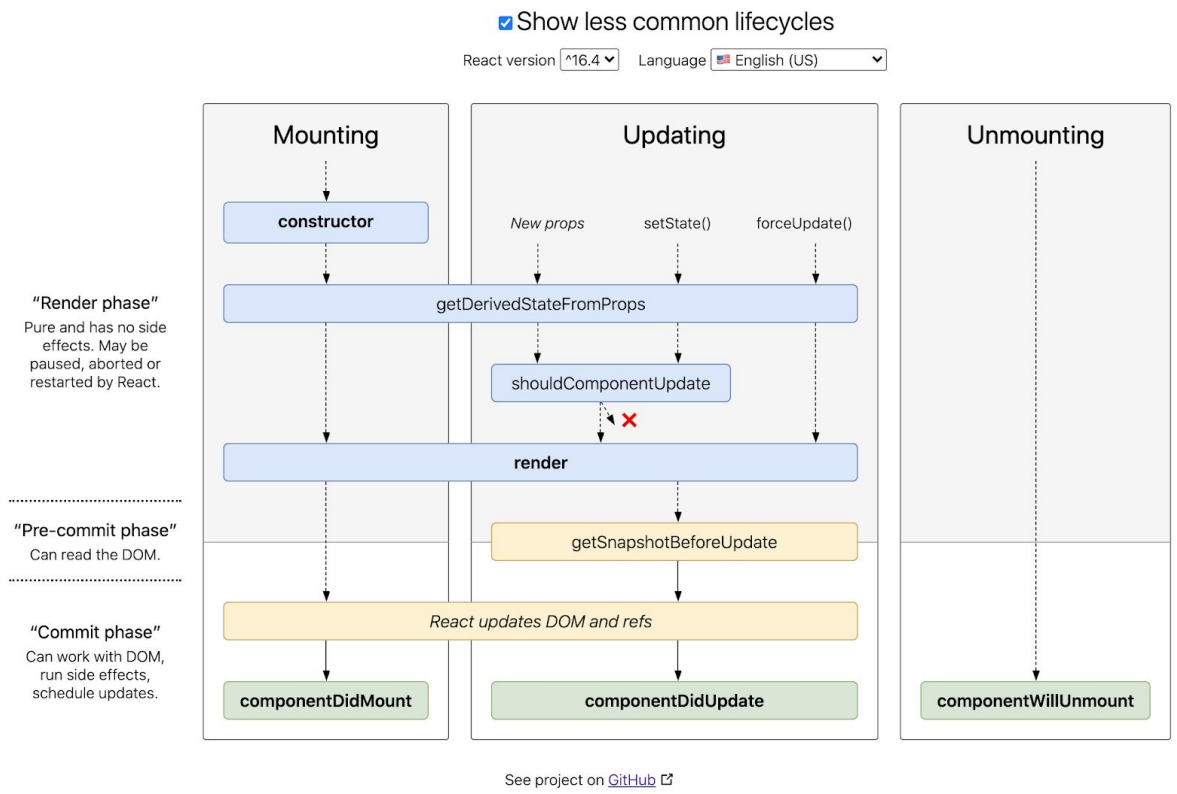
Preuzeto: https://we-are.bookmyshow.com/at-the-of-react-3636e293e09

3.3 KOMPONENTE

React se razvija pomoću komponenata. Komponente omogućuju da se korisničko sučelje podijeli na neovisne dijelove koji se mogu ponovno upotrijebiti i izolirati. Komponente su gradivni elementi bilo koje React aplikacije, a tipična React aplikacija će se sastojati od velikog broja komponenti. Jednostavnije, komponenta je JavaScript klasa ili funkcija koja po želji prihvaća proizvoljne ulaze koji se u reactu zovu svojstva (props) te vraćaju React elemente koji opisuju što bi se trebalo pojaviti na ekranu. Iako React aplikacija može raditi s globalnim CSS stilovima, najbolja je praksa i njih imati odvojeno po komponentama. Komponente u React-u sadrže sav Javascript potreban za izvršavanje te komponente te njen prezentacijski opis ali ne u obliku HTML-a jer zapravo pišemo u Javascript okruženju pa se u Reactu koristi opisni dodatak za Javascript zvan JSX koji sliči na HTML ali su oni zapravo vrlo drugačije tehnologije no njihov rezultat korisniku izgleda jednako. Ujedno je to i jedna od glavnih razlika konvencionalnom pisanju aplikacija, koncept pisanja HTML-a u Javascript-u.[10]

Svaka komponenta u React-u ima svoj životni ciklus predstavljen kao niz metoda. Metode životnog ciklusa React komponenti smatramo nizom događaja koji se događaju od stvaranja jedne komponente do njenog izbacivanja. Svaki od tih događaja može biti praćen u komponenti. Komponente također imaju i stanja u kojima se nalaze što je jedan od ključnih koncepata u React-u.[11]

[10] https://reactjs.org/docs/components-and-props.html Pristupano 5 kolovoza 2020.
[11] https://reactjs.org/docs/state-and-lifecycle.html Pristupano 5 kolovoza 2020.



Slika br. 8: Prikaz metoda životnog ciklusa React komponente

Preuzeto: https://projects.wojtekmaj.pl/react-lifecycle-methods-diagram/

Komponente se dijele prema načinu kako su pisani u Javascript-u: klasne te funkcijske komponente. Klasne i funkcijske komponente jednake su s React-ovog gledišta. Na programeru je da izabere najbolju praksu ili uzorak.

3.3.1 FUNKCIJSKE KOMPONENTE

Najjednostavniji način definiranja komponente je pisanje JavaScript funkcije. Funkcija je valjana kao React komponenta jer prihvaća pojedinačni argument ili svojstvo (prop) s podacima i vraća React element. Takve komponente nazivamo "funkcijskim komponentama" jer su doslovno JavaScript funkcije. Nazivaju se i komponentama bez stanja jer jednostavno prihvaćaju podatke i prikazuju ih u nekom obliku odnosno one su uglavnom odgovorne za prikazivanje korisničkog sučelja. Metode životnog ciklusa komponente ne postoje u funkcijskoj komponenti, jer je

funkcijska komponenta samo obična JavaScript funkcija te unutar nje ne možemo koristiti metode. Zato se i nazivaju funkcijskim komponentama bez stanja.
No novije verzije Reacta (od React 16.8 nadalje) uvode “React Hook”, odnosno način kako i funkcijska komponenta može na svoj način imati životni ciklus.

```
// Pozivanje React-a
import React from "react";
// Pozivanje stilova
import styles from "./Button.module.scss";

// Sintaksa funkcijske komponente
const Button = ({ children, modifier, onClick }) => ( // svojstva (props)
   <button // Prikazani element
       type="button"
       className={`${styles.button} ${modifier || ""} u-fw-700
u-uppercase`}
       onClick={onClick}
   >
       {children}
   </button>
);

// Izvoz komponente kako bi se mogla ponovno upotrijebiti
export default Button;
```

Isječak koda 1: Osnovni primjer funkcijske komponente koja predstavlja gumb kao dio korisničkog sučelja

3.3.2 KLASNE KOMPONENTE

Klasne komponente su React komponente pisane pomoću Javascript klasa iz ES6 verzije Javascripta. Klase su najčešći načina definiranja React komponente. Iako je sintaksa opširnija od funkcijske sintakse, nudi veću kontrolu nad komponentom s metodama životnog ciklusa (iako funkcijske komponente sada nude Hook-ove da bi postigli slicne efekte). Svaka klasna komponenta obavezno mora imati Render() metodu koja prikazuje neku komponentu kao React element korisniku. To je slično “return-u” u funkcijskim komponentama. U klasama je jako lako raditi s metodama životnog ciklusa. Neke od tih metoda su metode za postavljanje komponente (mount) kao na primjer “render(), “componentDidMount()” koji nam šalju događaje

vezane za metodu i na temelju toga programer može izvršavati stvari u trenucima u kojima je potrebno a ne samo na inicijalnom učitavanju aplikacije. Slične su i metode za ažuriranje, hvatanje grešaka te micanje komponente. Osim toga u klasnim komponentama najlakše se kontrolira stanjem aplikacije. Stanje aplikacije je koncept koji je objašnjen u jednom od sljedećih poglavlja ali može biti definirano kao objekt skupa uočljivih svojstava koja kontroliraju ponašanje komponente. Drugim riječima, stanje komponente je objekt koji sadrži neke informacije koje se mogu promijeniti tijekom životnog ciklusa komponente.

Iako su u suštini klasne i funkcijske komponente Reactu iste, one ipak imaju neka različita svojstva. Klasne Komponente malo su složenije od funkcionalnih komponenata. Funkcionalne komponente nisu svjesne ostalih komponenti u aplikaciji, one jednostavno primaju i prikazuju podatke. Klasne komponente mogu međusobno surađivati i biti svjesne toga gdje se nalaze. Također Možemo prosljeđivati podatke i svojstva iz jedne komponente klase u druge komponente klase.

```jsx
// Pozivanje React-a
import React from "react";
// Pozivanje stilova
import Button from "../../common/Button/Button";

// Sintaksa klasne komponente
class ClassButton extends React.Component {

    constructor(props) {
        super(props);

        // Stanje aplikacije
        this.state = {
            isAnimating: false
        };
    }

    // Metoda životnog ciklusa
    componentDidMount() {
        console.log("Komponenta je učitana");
    }

    // Metoda koja kontrolira klik na gumb
    handleClick() {
        console.log("Gumb je pritisnut");
    }

    // Render metoda koja prikazuje komponentu
    render() {
        return (
            // Prikazani element
            <Button onClick={this.handleClick}>
                <p>Pokreni aplikaciju</p>
            </Button>
        );
    }
}

// Izvoz komponente kako bi se mogla ponovno upotrijebiti
export default ClassButton;
```

Isječak koda 2: Osnovni primjer klasne komponente koja predstavlja gumb kao dio korisničkog sučelja

3.4 SVOJSTVA (ENGL. “REACT PROPS”)

React nam omogućuje prosljeđivanje podataka u komponentu pomoću nečega što se nazivaju svojstva (“props”, skraćeno od “properties”). Svojstva su u osnovi vrsta globalne varijable ili objekta. Svojstva možemo proslijediti bilo kojoj komponenti dok deklariramo atribute za bilo koju HTML oznaku. [12]

```
<Configurator
   ref={this.configuratorRef}
   state={this.state}
   closeConfigurator={this.closeConfigurator}
   fontSizeChange={this.fontSizeChange}
   letterSpacingChange={this.letterSpacingChange}
   lineHeightChange={this.lineHeightChange}
   informationTypeChange={this.informationTypeChange}
   menuPositionChange={this.menuPositionChange}
   color1Change={this.color1Change}
   color2Change={this.color2Change}
   color3Change={this.color3Change}
   color4Change={this.color4Change}
   color5Change={this.color5Change}
/>
```

Isječak koda 3: Prosljeđivanje svojstva komponenti

U ovom primjeru ovoj se komponenti prosljeđuju svojstva, koja su u ovom slučaju funkcije koje će pozvati komponenta, no svojstva mogu biti bilo što, brojevi, nizovi, funkcije… Tada u komponenti možemo pristupiti određenom svojstvu te na temelju toga izvršavati manipulaciju komponente.

Svojstvu se u klasnoj komponenti pristupa na sljedeći način:

```
this.props.fontSizeChange
```

Isječak koda 4: Pristupanje svojstvu u klasnoj komponenti

[12] https://reactjs.org/docs/components-and-props.html Pristupano 5 kolovoza 2020.

a u funkcijskoj komponenti na sljedeću način:

```
props.fontSizeChange
```

Isječak koda 5: Pristupanje svojstvu u funkcijskoj komponenti

React-ov protok podataka između komponenata je jednosmjeran i ide od višljeg prema nižem u hijerarhiji (od roditelja do djeteta).

Bilo da deklarirate komponentu kao funkciju ili klasu, ona nikada ne smije mijenjati vlastita svojstva. React je prilično fleksibilan, ali ima jedno striktno pravilo: sve komponente React-a moraju se ponašati kao čiste funkcije s obzirom na svoja svojstva. To znači da su svojstva komponente samo za čitanje, dok pisanje nije dozvoljeno.

Naravno, korisnička sučelja aplikacije su dinamična i mijenjaju se tijekom vremena. Rješenje tog problema donose nam stanja React komponente. Stanje je poseban objekt koji je ugrađen u React, a koji komponentama omogućuje stvaranje i upravljanje vlastitim podacima. Dakle, za razliku od svojstva, komponente ne mogu prosljeđivati podatke stanja, ali ih mogu stvoriti i njima upravljati interno.

3.5 STANJA (ENGL. “REACT STATE”)

Tehnički gledano svojstva (engl. “props”) i stanja su obični JavaScript objekti. Iako oboje sadrže informacije koje utječu prikazivanje komponente, one se razlikuju po tome što se svojstva prosljeđuju komponenti (slično kao što se parametri prosljeđuju funkcijama u svim programskim jezicima), dok se stanjem upravlja unutar komponente (slično varijablama deklariranim unutar funkcije). Ako pojednostavimo definiciju, stanja su Javascript objekt u kojem se pohranjuju vrijednosti svojstva koje

pripadaju komponenti.[13] Kada se bilo koja vrijednost stanja promjeni, React će osvježiti izgled komponente tako što će ju ponovno prikazati s novim vrijednostima stanja. Taj koncept stanja je zapravo i "najteži" koncept za razumijevanje principa rada Reacta. Iako se funkcijske komponente nazivaju komponentama bez stanja i nisu namijenjene da imaju stanje ona također u novim verzijama Reacta mogu imati stanje s implementacijom "React Hook-ova".

```
// Pozivanje React-a
import React from "react";
// Sintaksa klasne komponente
class Name extends React.Component {
   constructor(props) {
       super(props);
       // Stanje aplikacije
       this.state = {
           name: "Mario Lončarek"
       };
   }
   // Render metoda koja prikazuje komponentu
   render() {
       return (
           //Ispisivanje trenutnog stanja
           <div>
               {this.state}
           </div>
       );
   }
}
// Izvoz komponente kako bi se mogla ponovno upotrijebiti
export default Name;
```

Isječak koda 6: Tipičan izgled klasne komponente s zadanim stanjem

Kada bi se u bilo kojem trenutku na bilo kakvu željenu interakciju korisnika stanje promijenilo ponovno bi se izvršila "render()" metoda te prikazala komponentu s novim stanjem.

Stanje se mijenja koristeći "setState()" metodu, te se ona u praksi zapravo poziva unutar neke funkcije koja se izvršila na neku interakciju korisnika. Na primjeru ovog diplomskog rada:

[13] https://reactjs.org/docs/state-and-lifecycle.html Pristupano 6 kolovoza 2020.

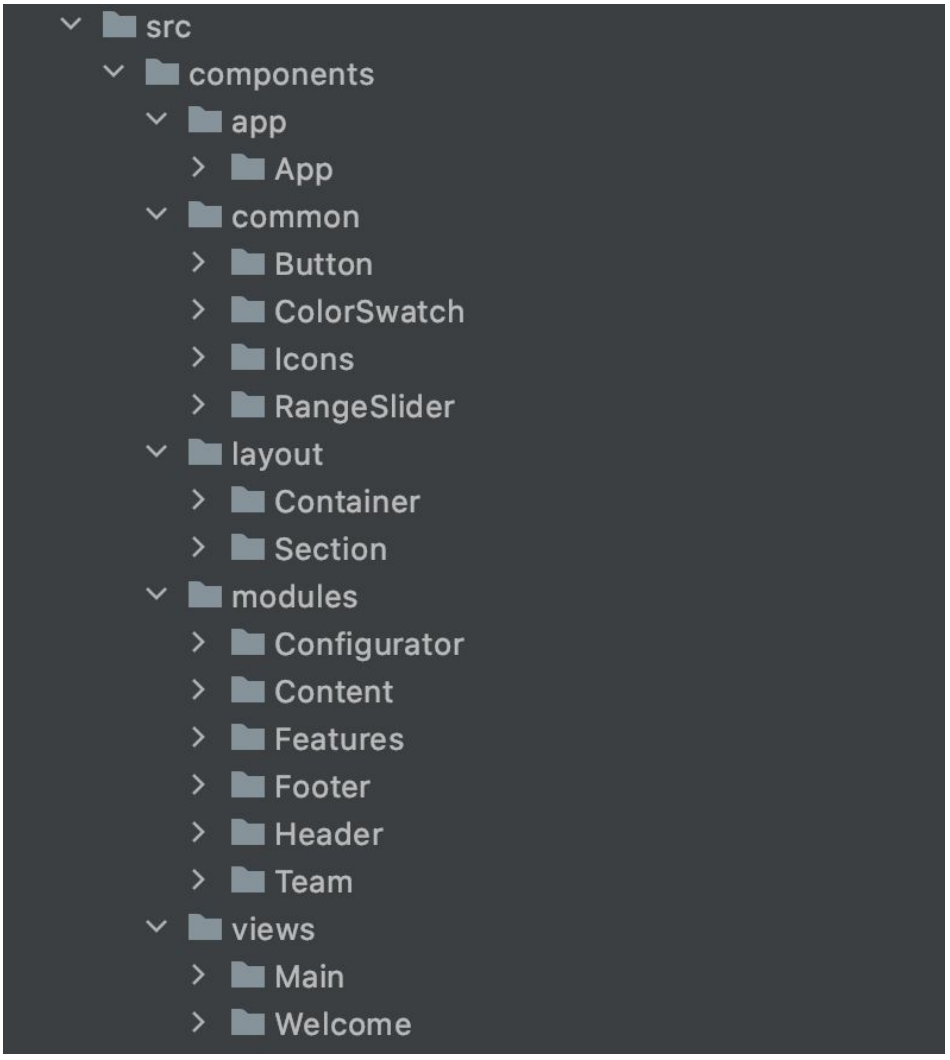
```
menuPositionChange(event) {
    this.setState({
        menuPosition: event.target.value,
    });
}
```

Isječak koda 7: Promjena stanja komponente

Funkcija se izvršila u trenutku kada je korisnik u konfiguratoru promijenio lokaciju gumba za navigaciju, u kojoj se pozvala metoda koja mijenja trenutno stanje. Stanje se promijenilo te se komponenta ponovno prikazala te je korisnik u realnom vremenu vidio promjenu i sada vidi gumb navigacije na novoj poziciji koju je odabrao. Na ovom se konceptu također bazira i aplikacija za ovaj diplomski rad.

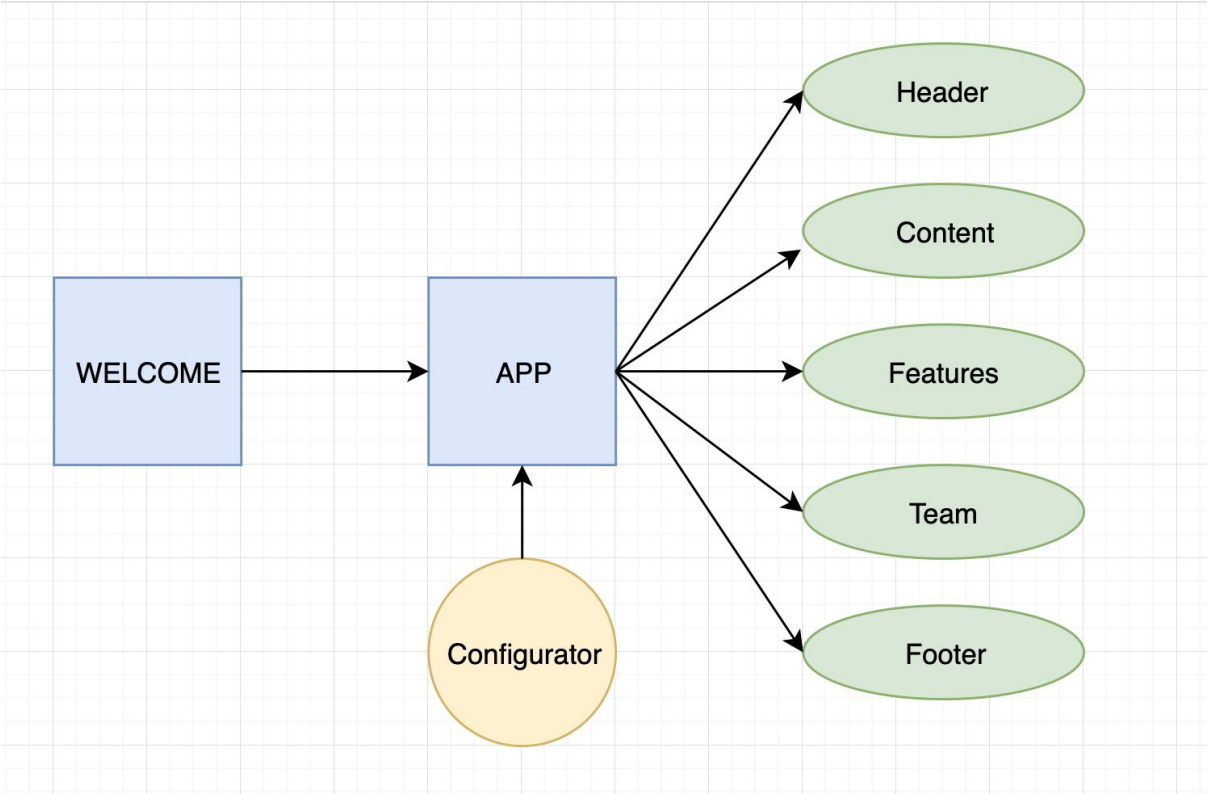
4. IZRADA EKSPERIMENTALNE APLIKACIJE I ARHITEKTURA APLIKACIJE

Aplikacija se sastoji od dvije glavne stranice - početne te stranice koja predstavlja glavni prikaz aplikacije odnosno od same aplikacije. Korisničko sučelje organizirano je na komponente i na module. Komponente u ovoj aplikaciji predstavljaju elemente sučelja koji se ponavljaju na modulima te ne ovise o stanjima već samo ispisuju odnosno prikazuju neki element. Moduli su također komponente ali sadrže tekstove, slike, komponente forme ili sve slično što se koristi za demonstraciju rada ove aplikacije. Modul komponente ovisne su o stanju njegove roditeljske komponente. Roditeljske komponente modula su zapravo stranice aplikacije ili glavni pogledi - početni i aplikacijski pogled. Aplikacijski pogled također sadrži modul konfiguratora koji mu mijenja stanje.



Slika br. 9: Arhitektura eksperimentalne aplikacije

Dijagramom komponenti aplikacije možemo bolje dočarati logiku arhitekture te kako komponente međusobno komuniciraju:



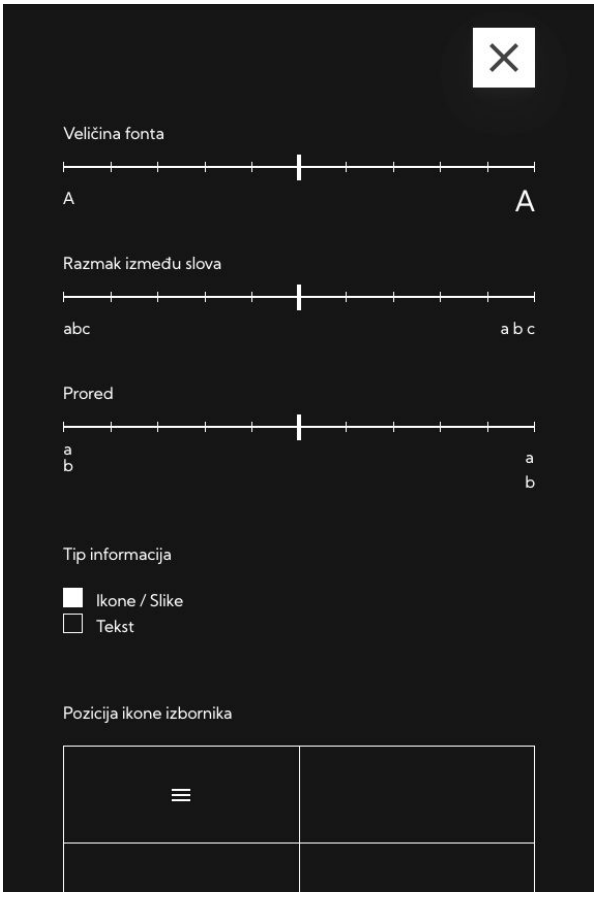
Slika br. 10: Dijagram komponenti eksperimentalne aplikacije

4.1 PRINCIP RADA APLIKACIJE

U ovom radu nisu objašnjeni svi koncepti niti značajke React.js biblioteke. No zapravo se cijela komponenta bazira na promjeni stanja glavnog pogleda aplikacije, a promjene stanja se pokreću promjenom parametara u konfiguratoru, te prosljeđivanju svojstva u komponente. Promjene stanja zatim pokreću ponovno prikazivanje svih komponenti u toj roditeljskoj komponenti na koje je to stanje utjecalo i to se prosljeđuje na komponente uz pomoć svojstva.

Glavni pogled aplikacije služi kao roditeljska komponenta svih modula i komponenti. On također u sebi sadrži i stanje svih komponenti odnosno u ovom slučaju to su stanja svih početnih vrijednosti koje su zadane za ovaj koncept. Aplikacija upravlja parametrima fonta (veličina, prored, razmak između slova), tipom informacija

(tekstualni ili slike/ikone), pozicijom gumba navigacije, te zasićenosti boje pomoću konfiguratora.



Slika br. 11: Izgled konfiguratora aplikacije

Sve te vrijednosti moraju imati zadanu početnu vrijednost koja se prikazuje na početku stranice. Te početne vrijednosti su stanja u React komponenti:

```
this.state = {
   fontSize: "1",
   letterSpacing: "1",
   lineHeight: "1.5",
   class1Current: 16,
   class1Default: 16,
   class1Max: 27,
   class1Min: 12,
   class2Current: 18,
   class2Default: 18,
   class2Max: 30,
   class2Min: 14,
   // ... ostatak klasa fontova
   informationType: "icons",
   menuPosition: "1",
   color1: "#93b7be",
   color2: "#f1fffa",
   color3: "#d5c7bc",
   color4: "#785964",
   color5: "#454545",
   configuratorOpen: false,
};
```

Isječak koda 8: Objekt stanja eksperimentalne aplikacije

4.2 LOGIKA PRILAGODBE PARAMETARA FONTA

Fontovi u aplikaciji su podijeljeni na 7 klasa koje predstavljaju 7 različitih veličina fonta. Dizajner web stranice ili aplikacije mora dostaviti minimalne, početne i maksimalne vrijednosti za svaku od tih veličina. Dizajner ograničava korisnika da parametrima mijenja svoje korisničko sučelje ali samo unutar granica koje su dozvoljene. Time se ipak zadržava neka razina kontrole te predvidljivosti.

```
class1Current: 16,
class1Default: 16,
class1Max: 27,
class1Min: 12,
```

Isječak koda 9: Zadane vrijednosti jedne klase

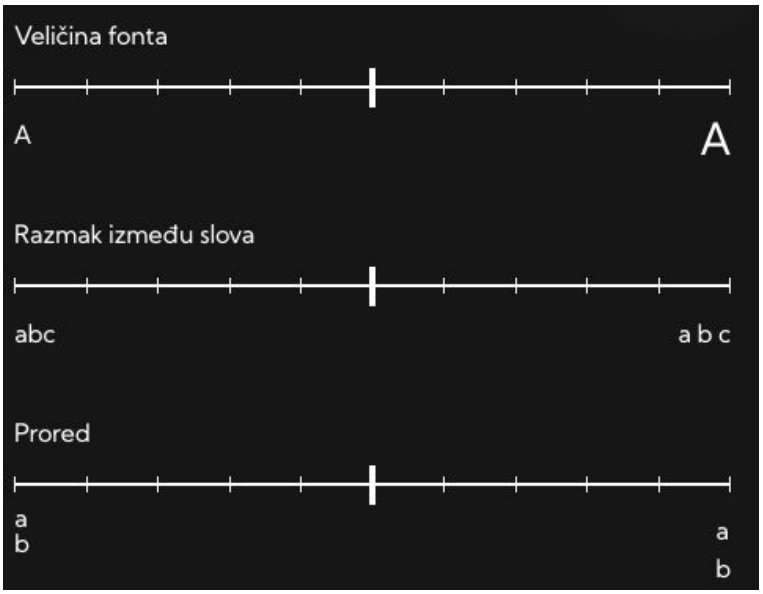
Na primjeru gornjeg dijela koda možemo zaključiti da klasa 1 ima početnu vrijednost 16px, maksimalnu vrijednost 27px a minimalnu vrijednost 12px.

U konfiguratoru postoje tri elementa forme klizača vrijednosti koji zapravo vraćaju neki broj koji se kasnije koristi u izračunima.

```
<RangeSlider
   label={"Veličina fonta"}
   minValue={0.5}
   minLabel={Parser("<span class='u-b0'>A</span>")}
   maxValue={1.5}
   maxLabel={Parser("<span class='u-a4'>A</span>")}
   initialValue={1}
   step={0.1}
   id={"font-size"}
   onChange={fontSizeChange}
/>
```

Isječak koda 10: Primjer komponente klizača

Dakle klizači za veličinu fonta, prored i razmak između rade na skali od 0.5 do 1.5, s početnom vrijednosti od 1 (u slučaju klase 1 to bi bilo 16px), te se klizači pomiču korakom od 0.1. Tako klizači vraćaju broj koji se množi u izračunima i zapravo se tako dobivaju vrijednosti u pikselima.



Slika br. 12: Elementi za prilagodbu parametara fonta

Svaki puta kada se bilo koji od klizača promjeni pozove se metoda koja upravlja stanjem koje predstavlja taj klizač, neka za primjer bude samo veličina fonta kao najkompleksnija kalkulacija u aplikaciji. Kada se promijeni klizač pozove se metoda “fontSizeChange” :

```
fontSizeChange(event) {
    let oldValue = this.state.fontSize;
    this.setState(
        {
            fontSize: event.target.value,
        },
        () => {
            if (oldValue >= this.state.fontSize && oldValue <= 1) {
                this.setState({
                    class1Current:
                        this.state.class1Default -
                        (1 - this.state.fontSize) *
                            10 *
                            this.class1NegativeChange,
                });
            } else {
                this.setState({
                    class1Current:
                        this.state.class1Default +
                        (1 - this.state.fontSize) *
                            -10 *
                            this.class1PositiveChange,
                });
            }
        }
    );
}
```

Isječak koda 11: Metoda za mijenjanje stanja vrijednosti fonta

Ovaj kod će prvo zapamtiti staru vrijednost koja je bila zadana u klizaču vrijednosti, a zatim će u stanje spremiti novu vrijednost klizača, a kada se to izvrši radi se usporedba starog i trenutnog stanja na klizaču da bi se detektirao pomak. Iz tog razloga moramo za svaku klasu izračunati njen pozitivni i negativni pomak:

```
this.class1PositiveChange =
   (this.state.class1Max - this.state.class1Default) / 5;
this.class1NegativeChange =
   (this.state.class1Default - this.state.class1Min) / 5;
```

Isječak koda 12: Izračun pozitivnog i negativnog pomaka za pojedinu klasu

Dakle količina pozitivnog pomaka za pojedinu klasu dobije se ako se razlika između maksimalnog i početnog stanja podijeli s 5, jer imamo 5 mogućih pozitivnih pomaka klizača. Negativni se pomak dobiva tako da se razlika između početnog i minimalnog pomaka podijeli s 5, opet jer imamo 5 mogućih negativnih pomaka.

Kada razumijemo za koliko se pomičemo u pozitivnom i negativnom smjeru onda možemo i ažurirati novo trenutno stanje te klase ovisno o smjeru pomaka te količini pomaka koje smo dobili od klizača vrijednosti.

Na samom elementu koji mora promijeniti svoje parametre fonta vrlo je jednostavno napraviti promjenjivi element pošto svakom promjenom klizača zapravo pozivamo ponovno prikazivanje komponenti. Na primjeru klase 1 za neki element to izgleda ovako:

```
<p
   style={{
       fontSize: state.class1Current,
       letterSpacing:
           state.letterSpacing - 1 + "px",
       lineHeight: state.lineHeight,
   }}
>
```

Isječak koda 13: Izračun proreda pojedine klase

Dakle veličina fonta je zapravo trenutno stanje veličine fonta klase 1. Trenutna veličina se izračunava svakom promjenom klizača, a svakom promjenom se i ta komponenta ponovno prikaže se novim vrijednostima stanja.

Prored je puno jednostavniji matematički pošto možemo jednostavno kopirati vrijednost klizača jer u CSS-u možemo primijeniti vrijednost proreda bez mjerne jedinice. U tom slučaju vrijednost proreda zapravo postaje konstanta koja se množi s veličinom fonta. Dakle ako je vrijednost klizača 1.5, prored će biti preslikana vrijednost klizača odnosno također 1.5. Ta vrijednost se množi s bilo kojom veličinom fonta, za primjer neka bude 16 i dobije se vrijednost proreda od 24px. Isto vrijedi i u negativnom smjeru.

Razmak između slova je vrlo precizna mjera u CSS-u pa se vrijednost klizača umanjuje za 1 i pretvara u pixele. Pa ako imamo vrijednost klizača 0.5, vrijednost razmaka između slova bit će -0.5px. Isto vrijedi i u pozitivnom smjeru.

4.3 LOGIKA PROMJENE TIPA INFORMACIJA

Korisnik u aplikaciji pomoću konfiguratora može promijeniti tip informacije koja mu se prikazuje, dali samo tekstualno ili želi ikone/slike. U konfiguratoru je to predstavljeno kao izborni gumb (radio button) element forme. Taj element vraća vrijednost koja je proizvoljno nazvana ovisno o tome koji element je obilježen. U slučaju ove aplikacije vrijednosti su “icons” te “text”.

Slika br. 13: Promjena tipa informacija

Također u stanju glavnog pogleda postoji početno stanje za tip informacija. Kada se promjeni obilježeni radio gumb, on također šalje novo stanje u glavnu aplikaciju koja zatim ponovno učitava komponente na koje je to stanje utjecalo.

```
informationTypeChange(event) {
    this.setState({
        informationType: event.target.value,
    });
}
```

Isječak koda 14: Promjena stanja tipa informacija

Kada imamo stanje izbornog gumba, uz pomoć React-a možemo vrlo lako izvesti uvjetni prikaz elemenata:

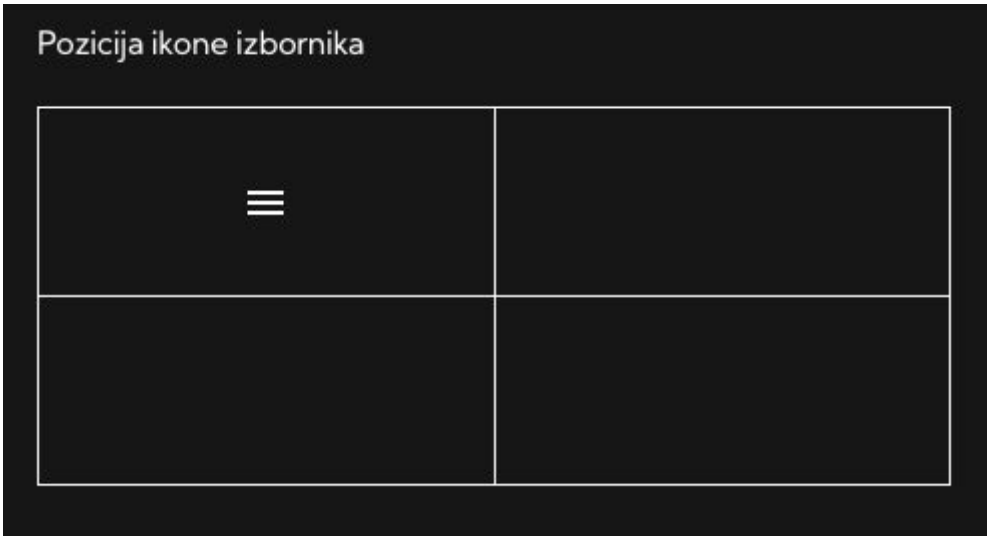
```
{state.informationType === "icons" ? (
    <Icon
        name={"facebook"}
        style={{
            fontSize: state.class7Current,
            letterSpacing: state.letterSpacing - 1 + "px",
            lineHeight: state.lineHeight,
        }}
    />
) : (
    <p
        style={{
            fontSize: state.class3Current,
            letterSpacing: state.letterSpacing - 1 + "px",
            lineHeight: state.lineHeight,
        }}
    >
        Facebook
    </p>
)}
```

Isječak koda 15: Uvjetni prikaz elemenata

U ovom primjeru ispituje se trenutno stanje prikaza informacija koje se proslijedio iz roditeljske komponente. Ako je vrijednost “icons” prikazuje se ikona, no ako je vrijednost neka druga prikazuje se tekstualna reprezentacija te ikone odnosno tekstualna informacija. Pošto izborni gumb forme ima samo dva stanja, redundantno je ispitivati dali je stanje jednako bilo kojem drugom stanju, jer ako nije prvo, onda je sigurno drugo stanje.

4.4 LOGIKA PROMJENE POZICIJE GUMBA NAVIGACIJE

Promjena pozicije gumba navigacije također je izvedena s izbornim gumbom (engl. “radio button”). Ali u ovom slučaju gumbi su stilizirani tako da predstavljaju određenu poziciju na ekranu korisnika. Predefinirane su 4 pozicije na kojima gumb navigacije može biti: gornji lijevi, gornji desni, donji lijevi te donji desni.



Slika br. 14: Promjena pozicije gumba navigacije

Izborni gumbi vraćaju broj pozicije na koja je trenutno izabrana, te kao i ostalo imaju svoju početnu vrijednost zapisanu u stanju glavne aplikacije. Promjenom tih vrijednosti također se izvršava promjena stanja te samim time novi prikaz komponenti s novim stanjima.

Na samom elementu kojeg mijenjamo (gumb navigacije) stanje je povezano s CSS klasom:

```
<div
   className={cx(
       styles.configuratorTrigger,
       `position-${state.menuPosition}`
   )}
>
```

Isječak koda 16: Povezanost stanja sa CSS klasom

Ovisno o stanju na element se aplicira druga klasa, dok su sve 4 CSS klase za pozicije unaprijed predefinirane:

```
&.position-1 {
   top: get-vw(60px);
   left: get-vw(60px);
}

&.position-2 {
   top: get-vw(60px);
   right: get-vw(60px);
}

&.position-3 {
   bottom: get-vw(60px);
   left: get-vw(60px);
}

&.position-4 {
   bottom: get-vw(60px);
   right: get-vw(60px);
}
```

Isječak koda 17: CSS klase koje upravljaju pozicijom gumba navigacije

4.5 LOGIKA PROMJENE ZASIĆENOSTI BOJA

Promjene zasićenosti boja u aplikaciji oslanja se na koncept saturacije boja. Kao i kod parametara za promjenu vrijednosti fontova, sve boje moraju biti unaprijed predefinirane i zapravo korisnik odabire ono što je dizajner postavio kao granice.

```
color1: "#93b7be",
color2: "#f1fffa",
color3: "#d5c7bc",
color4: "#785964",
color5: "#454545",
```

Isječak koda 18: Početne vrijednosti stanja pojedinih boja

Promjena zasićenosti boje također koristi izborne gumbe (“radio button”), od kojih svaki predstavlja jednu nijansu i predefiniranim bojama.

Slika br. 15: Promjena zasićenosti boja aplikacije

Još jedna sličnost s promjenom parametara fonta je da se mijenja samo trenutna vrijednost, dok su predefinirane vrijednosti samo granice u kojima se korisnik može kretati. Svaki izborni gumb ima vrijednost koja odgovara heksadecimalnom kodu za tu boju. Boje su predstavljene kao niz heksadecimalnih kodova određenih principom

saturacije. Taj niz se kasnije provlači kroz petlju koja generira onoliko izbornih gumbiju kolika je veličina niza kojeg smo mu proslijedili u svojstvima.

```
<ColorSwatch
   uniqueName={"color1"}
   colors={["#939dbe", "#93a9be", "#93b7be"]}
   isSelected={state.color1}
   onChange={color1Change}
/>
```

Isječak koda 19: Komponenta izbornih gumbiju za promjenu stanja boje

Na samom elementu koji prihvaća te boje, boja se aplicira na taj element kao CSS boja pozadine.

```
<footer
   ref={ref}
   className={styles.footer}
   style={{
       backgroundColor: state.color3,
   }}
>
```

Isječak koda 20: Promjena pozadine ovisna o stanju boje

Promjenom izbornih gumbiju poziva se funkcija koja mijenja trenutno stanje određenih boja koje korisnik želi mijenjati, te se poziva ponovno.

5. REZULTATI I RASPRAVA

Rješenje koje nudi ovaj rad je modularno. Broj i vrsta parametara koje koristi je varijabilna, te je podložno promjenama. Promjene mogu biti s obzirom na potvrdu ispravnosti metode koja je do sada korištena jer se testiranje provodi na temelju iterativnog procesa ispitivanja uzoraka koji poboljšava rad same aplikacije, što je riješeno automatski.

Neke od ideja za daljnji razvoj je uvrštavanje komponente vremena u prilagođavanje GUI-a. Uređaj bi mjerio pauze u korištenju aplikacije, kao i ukupno vrijeme korištenja aplikacije. Pretpostavlja se da će osobe koje su mlađe brže koristiti aplikaciju, dok starije duže. S obzirom na to da je ciljana skupina ovog rada široke mase, potrebno je obuhvatiti što više posebnih potreba, pa je tako došlo za potrebom i za mogućnošću prepoznavanja gluhonijemog jezika ili slabovidnih osoba. Glavni problem današnjih dizajnera je olakšano korištenje pametnih uređaja svim korisnicima. **Kreiranje dizajna kojeg mogu koristiti široke skupine korisnika, uključujući one koji imaju sve mogućnosti ili one koji imaju privremeno ili trajno oštećenje koje im otežava interakciju, je i dalje diskurs na kojem se radi.**

Mlađa dobna skupina, ispod 14 godina, nije uključena u studiju zbog etičkog kodeksa istraživanja s djecom. Međutim, prema istraživanju s korisnicima u dobi od 3 do 12 godina pokazuje se da su djeca stekla značajno znanje o korištenju web stranica i aplikacija, iako većina dizajna još uvijek nije optimizirana za mlađe korisnike. Projektiranje za djecu zahtijeva različite pristupe upotrebljivosti, uključujući ciljanje sadržaja usko za djecu različite dobi.[14] Dakle, mladi se danas puno više nalaze u digitalnom svijetu, stoga je njima ova prilagodba potrebna.

Kako računala postaju manja i u konačnici uglavnom virtualna, znanstvenici moraju razmišljati o drugim načinima stupanja u interakciju s njima, pa bi korištenje

[14] https://www.nngroup.com/articles/childrens-websites-usability-issues/ Pristupano 7. srpnja 2020.

tipkovnice moglo biti dio ovog istraživanja u budućnosti. Tehnologija koja dopušta ljudima da jednostavno upisuju slova ne postoji. *Early eye-gaze* tipkanje, razvijeno za odrasle s dobrim kognitivnim funkcijama, ali smanjenom pokretljivošću, nekad je bilo sporo i naporno, ali je sada modernizirano. Na nedavnoj konferenciji o umjetnoj inteligenciji u Cambridgeu, Kristensson je predstavio metodu tipkanja koja kombinira praćenje očiju s predviđanjem, što znači da korisnik može jednostavnije i bezbrižnije komunicirati.[15] Dakle, sve više se radi na tome da tipkovnica u budućnosti ne bude sastavni dio digitalnog svijeta.

Sustavi za praćenje očiju upotrebljavani su u mnogim različitim područjima, uključujući psihologiju, kognitivnu znanost, rehabilitaciju invaliditeta istraživanja i interakcije između čovjeka i računala (HCI). Smatra se da bi korištenjem Tobii uređaja ili bilo kojeg drugog uređaja za praćenje očiju poboljšalo i olakšalo komunikaciju između korisnika i uređaja zbog čega bi ovo bila jedna od budućih nadogradnji ovog rada.

Umjetna inteligencija (engl. *artificial intelligence*, u daljnjem tekstu AI) je napravila veliki utjecaj posljednjih godina. Unatoč pojavi početnih pesimističkih stavova o njegovoj uporabi, AI se drži obećanja da će računalo upotrijebiti za razbijanje novih temelja, za rješavanje inače neprohodnih problema i za poboljšanje kvalitete postojećih računalnih sustava.[16] S obzirom na to, postoji mogućnost da osim postojećih varijabilnih parametara, AI koristi više od toga, pa tako sučelja prilagođava još više raznolikim ljudskim potrebama.

Jedna od mogućih dodatnih varijabilnih komponenata je praćenje pokreta lica. Moderni mobilni telefoni opremljeni su procesorom velike snage, zaslonom visoke razlučivosti, internetskom vezom i integriranim snimanjem podataka putem mikrofona i fotoaparata. Ukratko, oni imaju sav hardver koji im je potreban za aplikacije kao što su biometrijska autentifikacija, napredna sučelja i telekonferencije

[15] https://qz.com/1468577/the-future-of-typing-doesnt-involve-a-keyboard/ Pristupano 7. srpnja 2020.
[16] Using Artificial Intelligence to Automatically Test GUI, Abdul Rauf, Mohammad N. Alanazi, College of Computer and Information Sciences Al-Imam Mohammad Ibn Saud Islamic University (IMSIU) Riyadh, Saudi Arabia J_045

niske propusne moći što omogućava implementaciju modela *Active Appearance* koji je u stanju pratiti lice na mobilnom uređaju u realnom vremenu.[17]

Rezultat eksperimentalnog dijela pokazao se kao jako dobar koncept prilagodljivog grafičkog korisničkog sučelja web aplikacije. React.js se zbog svih svojih karakteristika i značajki pokazao kao najbolji mogući alat s kojim se ovakav rezultat može postići za malo vremena. Za korisnika osigurava iskustvo promjene u realnom vremenu te osjećaj pozitivne interakcije s aplikacijom ili web stranicom zbog svoje brzine. React.js je napravljen baš za ovakve situacije, brza manipulacija onime što korisnik vidi. Razvijeni su i svi logički algoritmi za promjenu određenih parametara te se ovakav koncept sigurno može upotrijebiti u praksi za slučajeve navedene i istražene u ovom radu.

[17] Real-Time Facial Feature Tracking on a Mobile Device, P.A. Tresadern, M.C. Ionita, T.F. Cootes J_047

6. ZAKLJUČAK

Ako korisnici ne provedu mnogo vremena u učenju i razumijevanju kako funkcionira GUI-a, smatra se da je informacijski sustav ima vrlo učinkovit GUI. Kvalitetan GUI mora moći pristupiti zahtjevima korisnika i zadovoljiti ih. Kako računala postaju manja i u konačnici uglavnom virtualna, znanstvenici moraju razmišljati o drugim načinima stupanja u interakciju s njima. Sve više se radi na tome da nadolazeći i napredniji digitalni svijet što manje opterećuje korisnika. Jedinstveni dizajn koji se može prilagoditi pojedincu ovisno o njegovim potreba i željama je jedno od rješenja, a doprinos toga je i olakšavanje procesa izrade dizajna za različite ciljane publike. Sakupljajući povratne informacije o korisničkom sučelju te njegova implementacija u ostale sustave je budućnost prilagodljivog dizajna koji sam po sebi nije prepreka inovacijama. Razmišljajući o svim karakteristikama određenih ciljanih skupina, od kojih jedan dio njih ima neki oblik hendikepa, dizajn napokon može postati dostupan i onima za koje konvencionalna pravila dizajna jednostavno ne vrijede.

7. LITERATURA

1. "Poetika prostora - Gaston Bachelard - Google Books." https://books.google.com/books/about/Poetika_prostora.html?id=gt7kAAAACAAJ. Pristupano 4. srpnja 2020.
2. "ZAVRŠNI RAD." http://eprints.grf.unizg.hr/2744/1/Z821_Mareni%C4%87_Iva.pdf. Pristupano 4. srpnja 2020.
3. "DIZAJN DIGITALNOG PROSTORA, Jasenka Pribernik | Hrvatska ...." http://www.hsn.hr/product/184/dizajn-digitalnog-prostora-jasenka-pribernik. Pristupano 3. srp. 2020.
4. https://uxdesign.cc/designing-for-accessibility-is-not-that-hard-c04cc4779d94 Pristupano 6. srpnja 2020.
5. SPA (Single-page application) - https://developer.mozilla.org/en-US/docs/Glossary/SPA Pristupano 2. kolovoza 2020.
6. https://en.wikipedia.org/wiki/React_(web_framework) Pristupano 2.kolovoza 2020.
7. https://www.simplilearn.com/what-is-react-article Pristupano 2. kolovoza 2020.
8. https://developer.chrome.com/apps/app_frameworks Pristupano 4. kolovoza 2020.
9. https://reactjs.org/docs/faq-internals.html Pristupano 4. kolovoza 2020.
10. https://reactjs.org/docs/components-and-props.html Pristupano 5. kolovoza 2020.
11. https://reactjs.org/docs/state-and-lifecycle.html Pristupano 5. kolovoza 2020.
12. https://reactjs.org/docs/components-and-props.html Pristupano 5. kolovoza 2020.
13. https://reactjs.org/docs/state-and-lifecycle.html Pristupano 6. kolovoza 2020.
14. https://www.nngroup.com/articles/childrens-websites-usability-issues/ Pristupano 7. srpnja 2020.
15. https://qz.com/1468577/the-future-of-typing-doesnt-involve-a-keyboard/ Pristupano 7. srpnja 2020.

16. Using Artificial Intelligence to Automatically Test GUI, Abdul Rauf, Mohammad N. Alanazi, College of Computer and Information Sciences Al-Imam Mohammad Ibn Saud Islamic University (IMSIU) Riyadh, Saudi Arabia J_045
17. Real-Time Facial Feature Tracking on a Mobile Device, P.A. Tresadern, M.C. Ionita, T.F. Cootes J_047

8. POPIS SLIKA

9. POPIS MANJE POZNATIH RIJEČI

API	Aplikacijsko programsko sučelje (engl. “Application Programming Interface”)
CSS	Cascading Style Sheets
DOM	Document Object Model
GUI	Grafičko korisničko sučelje (engl. graphical user interface)
HTML	Hyper Text Markup Language
HTTP	Hypertext Transfer Protocol
ICT	Information And Communication Technology
MVC	Model - View - Controller
NPM	Node package manager
SPA	Single-page application

10. POPIS ISJEČAKA KODA