Razvojni okviri u klijentskom dijelu web aplikacije

Pavlinić, Robert

SVEUČILIŠTE U ZAGREBU
FILOZOFSKI FAKULTET
ODSJEK ZA INFORMACIJSKE I KOMUNIKACIJSKE ZNANOSTI
SMJER ISTRAZIVAČKA INFORMATIKA
Ak. god. 2020./2021.

Robert Pavlinić

Razvojni okviri u klijentskom dijelu web aplikacije

Diplomski rad

Mentor: doc. dr. sc. Vedran Juričić

Zagreb, travanj 2021.

Izjava o akademskoj čestitosti

Izjavljujem da je ovaj rad rezultat mog vlastitog rada koji se temelji na istraživanjima te objavljenoj i citiranoj literaturi. Izjavljujem da nijedan dio rada nije napisan na nedozvoljen način, odnosno da je prepisan iz necitiranog rada, te da nijedan dio rada ne krši bilo čija autorska prava. Također izjavljujem da nijedan dio rada nije korišten za bilo koji drugi rad u bilo kojoj drugoj visokoškolskoj, znanstvenoj ili obrazovnoj ustanovi.

Sadržaj

1. Uvod

Web aplikacije danas imaju za cilj pružati gotovo iste mogućnosti kao i nativne aplikacije, zbog čega razvoj klijentskog dijela web aplikacija postaje sve kompleksniji. Kako bi se olakšao i ubrzao razvoj web aplikacija, pojavljuju se razni razvojni okviri. Iako postoji velik broj razvojnih okvira za klijentski dio aplikacije te se čini kao da se svaki dan pojavljuje novi, već nekoliko godina zaredom tri najpopularnija su *React*, *Angular* i *Vue.js*. Sva tri za razvoj koriste paradigmu komponenti – enkapsulacija izgleda, strukture i logike ponovno iskoristivog dijela korisničkog sučelja koje se zatim slažu u veće komponente. Unatoč tome, u samoj filozofiji razvojnih okvira i njihovom ekosustavu postoje velike razlike. Danas se od programera koji se bave razvojem korisničkih sučelja očekuje da mogu lako i brzo preći s jednog na drugi razvojni okvir, stoga vrijedi istražiti koje su im sličnosti, a koje razlike.

2. Web aplikacije

Web aplikacije su programi s kojima korisnici imaju interakciju putem sučelja web preglednika. Iako ne postoji konkretno pravilo prema kojem možemo razlikovati web stranice od web aplikacija, generalno se web aplikacijama smatraju programi koji su dinamičke naravi te oni putem kojih korisnici mogu obaviti neku radnju, a za web stranice možemo reći da su statičke i svrha im je prikaz informacija i podataka[1]. Web aplikacije i stranice se sastoje od poslužiteljskog i klijentskog dijela. Web preglednik od poslužitelja preuzima *HTML, CSS i JavaScript* datoteke putem *HTTP* zahtjeva čiji sadržaj tada interpretira. Taj skup datoteka jest klijentski dio neke web aplikacije ili stranice. *HTML* služi za definiranje strukture elemenata, *CSS* za upravljanje izgledom elemenata, a *JavaScript* za dodavanje interakcija.

2.1. Povijest Web aplikacija

1989. godine, Tim Berners-Lee u *CERN*-u razvija World Wide Web. Prvotno zamišljen kao sustav za razmjenu dokumenata među znanstvenicima, moderni web se sa svojim web-dućanima, alatima za kolaboraciju, igrama i drugim raznim aplikacijama uvelike razlikuje od tadašnjeg weba.

Teško je odrediti točan vremenski period koji bismo mogli smatrati začetkom razvoja modernog weba i modernih web aplikacija. Jedan od bitnijih događaja je zasigurno osnivanje *Web Hypertext Application Technology Working* grupe (engl. kratica *WHATWG*) 2004. godine. *WHATWG* je radna skupina koju su osnovali zaposlenici Mozille, Applea i Opera Softwarea kako bi nastavili razvoj web tehnologija i surađivali u izradi web standarda nakon stagnacije u razvoju *HTML*-a i neslaganja s World Wide Web konzorcijem (engl. kratica *W3C*) oko njegove budućnosti. *HTML* standard je zanemarivan, a *W3C* je kao glavno tijelo za definiranje web standarda radio na tehnologijama poput *XHTML*-a koji nije rješavao stvarne probleme na webu i pomagao u njegovom razvoju[2]. Osnivanjem *WHATWG*-a započeo je razvoj *HTML5* standarda koji je službeno objavljen 2008. godine. *HTML* je u *HTML5* specifikaciji

1 Shklar, L., & Rosen, R. (2009). Web Application Architecture: Principles, Protocols and Practices (2nd edition). Wiley.

2 *FAQ — WHATWG*. (bez dat.). Preuzeto 07. ožujak 2021., od https://whatwg.org/faq

opisan kao jezik za označavanje koji je prvotno bio dizajniran kao jezik za semantičko opisivanje znanstvenih dokumenata, no s godinama se razvio u jezik za opisivanje drugih tipova dokumenata, pa čak i aplikacija[3]. 2019. godine *W3C* i *WHATWG* potpisuju sporazum o zajedničkom razvijanju *HTML* i *DOM* standarda i specifikacije[4].

Još jedan bitan događaj u razvoju weba kao platforme je i objavljivanje *ECMAScript* 2015 specifikacije (kratica *ES6*) koja je dodala mnoge nove značajke u standard. ES6 specifikacija je vrlo bitna za razvoj modernih web aplikacija jer donosi mnogo promjena i novih značajki od prvog izdanja specifikacije 1997. godine[5] poput novih struktura podataka te nove sintakse. *JavaScript* – programski jezik koji je najpopularnija implementacija *ECMAScript* specifikacije je do *ES6* specifikacije bio zastario u usporedbi s ostalim programskim jezicima, a od tada svake godine izlazi nova *ECMAScript* specifikacija. Trenutna verzija standarda je 11 – *ECMAScript* 2020.

Iako su web aplikacije postojale i prije *HTML5* i *ES6* specifikacija, bilo ih je znatno teže razvijati, a još teže prilagoditi za razne web preglednike i uređaje. Također, takve su aplikacije većinom bile ekstenzija postojećeg modela web stranica gdje se za svaku korisničku akciju učitavala neka druga stranica kako bi se prikazao rezultat te akcije, stoga nisu imale puno zajedničkog s nativnim (desktop ili mobilnim) aplikacijama koje su pružale daleko bolju responzivnost na korisničke akcije te generalno bolje korisničko iskustvo. Pomoću *JavaScripta* se tada takvim aplikacijama dodavalo malo interaktivnost. S vremenom dolazi do razvoja tehnologija koje malo-pomalo mijenjaju način izrade web aplikacija te se sve više logike prebacuje na klijentski dio. Jedan od naziva za taj skup tehnologija je *AJAX*[6] - Asinkroni *JavaScript* i *XML*. Svrha *AJAX* tehnologija jest omogućiti razvijateljima web aplikacija stvaranje korisničkih iskustava koja se sve više približavaju nativnim aplikacijama. Jedna od najbitnijih tehnologija *AJAX-a* je aplikacijsko programsko sučelje *XMLHttpRequest*[7] (kratica *XHR*) koje omogućava dohvaćanje podataka sa servera bez ponovnog učitavanja web stranice.

---

3 HTML Standard. (bez dat.). Preuzeto 04. siječanj 2021., od https://html.spec.whatwg.org/#background

4 *W3C and the WHATWG signed an agreement to collaborate on a single version of HTML and DOM | W3C News*. (bez dat.). Preuzeto 07. ožujak 2021., od https://www.w3.org/blog/news/archives/7753

5 ECMAScript 2015 Language Specification. (2015). 566.

6 Garrett, J. J. (bez dat.). Ajax: A New Approach to Web Applications. 5.

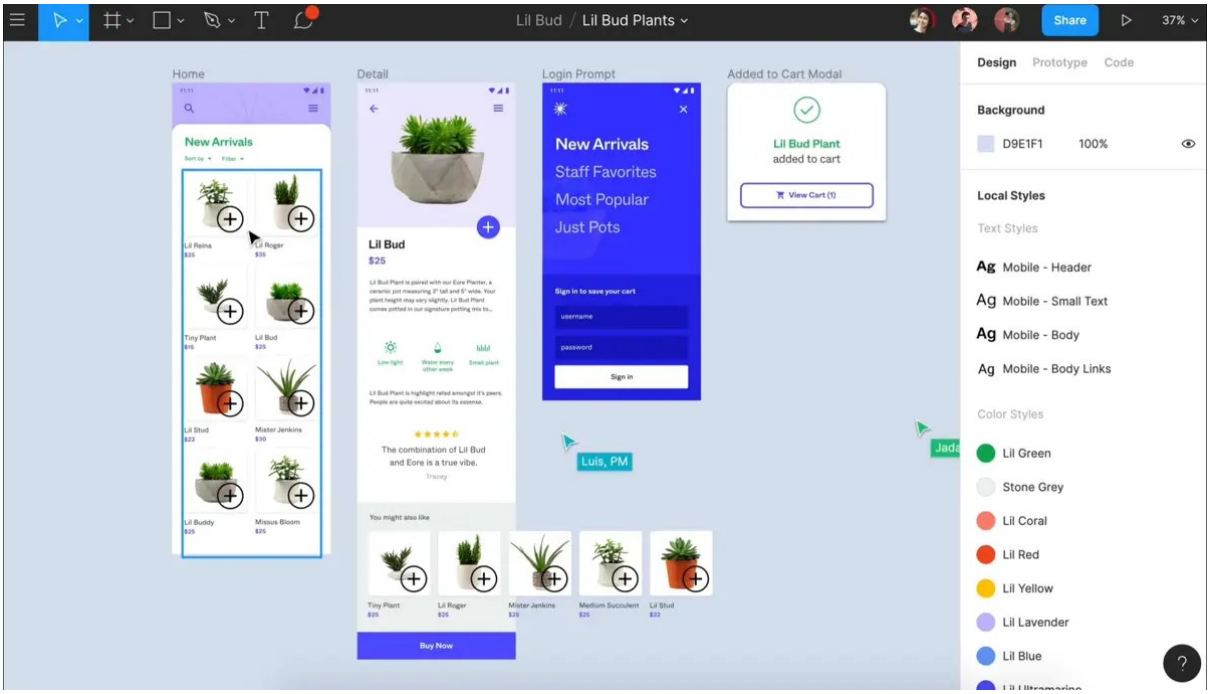
7 XMLHttpRequest—Web APIs | MDN. (bez dat.). Preuzeto 31. siječanj 2021., od https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest

Web se tako kontinuiranim razvojem standarda, iz platforme za pregledavanje dokumenata, pretvorio u pravu platformu za razvoj naprednih aplikacija.

2.2. Web aplikacije danas

Web aplikacije danas mogućnostima gotovo da pariraju tradicionalnim nativnim aplikacijama. Neki od primjera naprednijih web aplikacija su:

- Figma[8] - alat za dizajniranje korisničkih sučelja. Posebno impresivna značajka je suradnja više korisnika u stvarnom vremenu na istom dizajnu

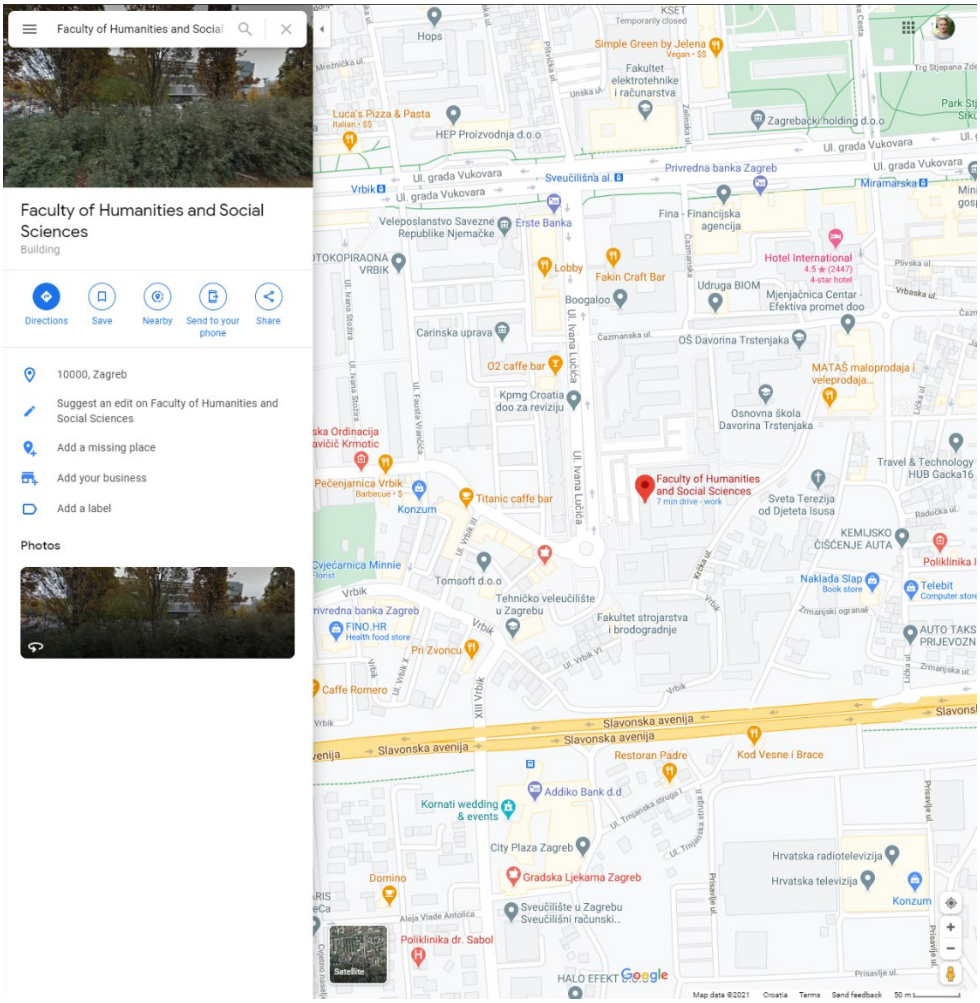


Slika 1 – korisničko sučelje Figme

- Google Maps[9] - Googleova besplatna aplikacija za prikaz zemljopisnih karata

[8] Figma: The collaborative interface design tool. (bez dat.). Figma. Preuzeto 17. siječanj 2021., od https://www.figma.com/

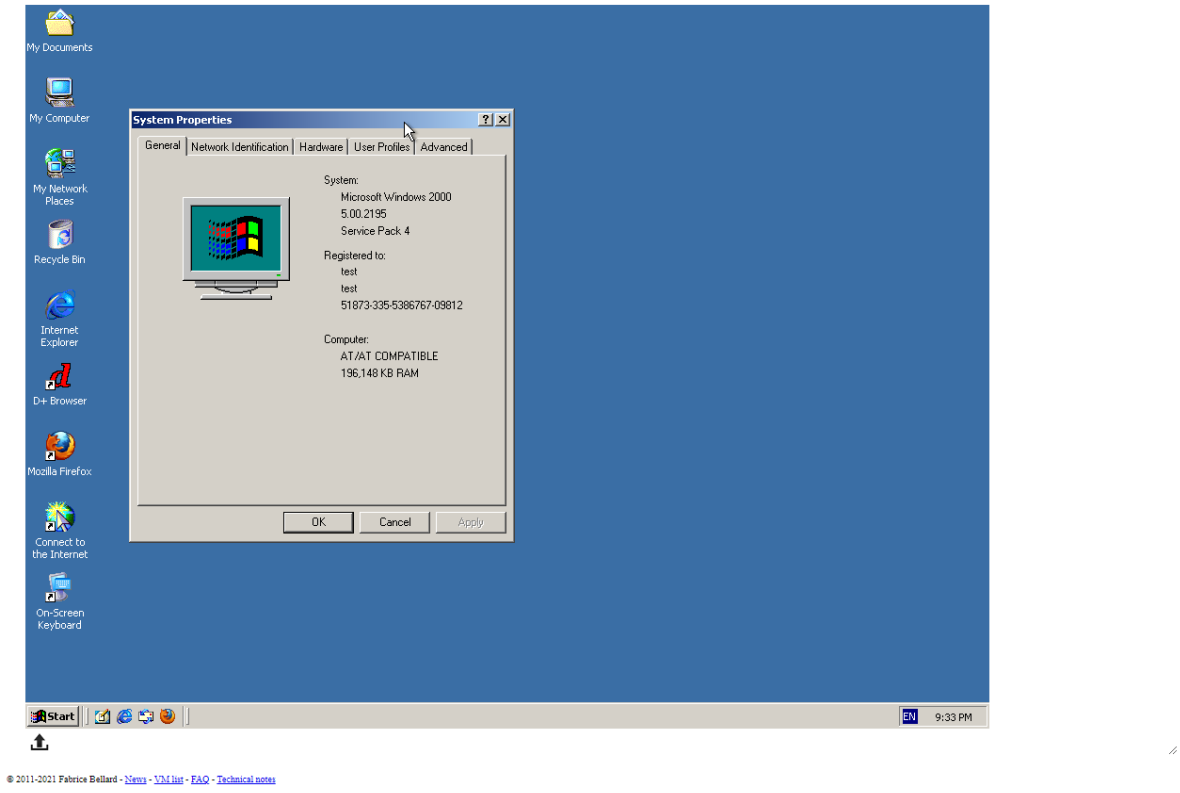
[9] Google Maps. (bez dat.). Google Maps. Preuzeto 17. siječanj 2021., od https://www.google.hr/maps



Slika 2 – korisničko sučelje Google Mapsa

- JSLinux[10] - emulatori nekoliko operativnih sustava poput Windows 2000

[10] Bellard, F. (bez dat.). JSLinux. Preuzeto 17. siječanj 2021., od https://bellard.org/jslinux/



Slika 3 – Windows 2000 emulator unutar web preglednika

Neke od prednosti web aplikacija u usporedbi s nativnim aplikacijama:

- Web aplikacije se ne trebaju instalirati te je njihova veličina generalno znatno manja od veličine nativnih aplikacija jednakih funkcionalnosti. Primjerice, Twitter web aplikacija prilikom prvog učitavanja preuzme oko 6 MB resursa[11], Android verzija Twitter aplikacije na Android 10 operativnom sustavu je veličine oko 20 MB[12], a iOS verzija je veličine čak 132.6 MB[13].
- Web aplikacije su podržane gdje god je podržan web preglednik kojim im korisnik pristupa, dok se nativne aplikacije moraju razvijat zasebno za svaku

[11] Twitter. It's what's happening. (bez dat.). Twitter. Preuzeto 11. siječanj 2021., od https://twitter.com/

[12] Twitter—Apps on Google Play. (bez dat.). Preuzeto 11. siječanj 2021., od https://play.google.com/store/apps/details?id=com.twitter.android&hl=en_US&gl=US

[13] Twitter. (bez dat.). App Store. Preuzeto 11. siječanj 2021., od https://apps.apple.com/us/app/twitter/id333903271

platformu (iznimke su tehnologije poput *Fluttera*[14] koji omogućuje razvoj aplikacija za više platformi u isto vrijeme s istim izvornim kodom).

Unatoč navedenim prednostima, web aplikacije imaju i mane u odnosu na nativne aplikacije. Neke od njih su:

- Web aplikacije često imaju slabije performanse jer se izvršavaju u web pregledniku koji je još jedna razina apstrakcije
    - *WebAssembly* je tehnologija kojoj je cilj omogućiti razvoj aplikacija nativnih performansi na webu te je dostupna u novijim verzijama popularnih preglednika – *Firefox*, *Chrome*, *Safari* te *Microsoft Edge*[15]
- Web aplikacije nemaju pristup nekim uređajima koji su priključeni na računalo, npr. USB memorijski štapić
    - *WebUSB*[16] i slične specifikacije pokušavaju riješiti izazove pristupa perifernim uređajima putem web preglednika. Implementacija *WebUSB* aplikacijskog programskog sučelja je trenutno dostupna samo u *Chrome* pregledniku[17].
- Za pristup web aplikacijama je potrebna internetska veza
    - Progresivne web aplikacije (kratica *PWA)* su web aplikacije koje imaju neke od bitnih karakteristika nativnih aplikacija – mogu se dodati na početni zaslon uređaja, tj. instalirati kao i nativne aplikacije, rade čak i (do neke mjere) u slučaju nestanka veze te mogu slati notifikacije korisnicima[18]. Razvojem PWA tehnologija, web aplikacije prestaju biti vezane za web preglednike te dobivaju vlastite prozore, kao i nativne aplikacije. Neke od najpoznatijih web aplikacija, poput Twittera i

[14] Flutter—Beautiful native apps in record time. (bez dat.). Preuzeto 11. siječanj 2021., od https://flutter.dev/

[15] WebAssembly. (bez dat.). Preuzeto 11. siječanj 2021., od https://webassembly.org/

[16] WebUSB API. (bez dat.). Preuzeto 11. siječanj 2021., od https://wicg.github.io/webusb/

[17] Can I use... Support tables for HTML5, CSS3, etc. (bez dat.). Preuzeto 11. siječanj 2021., od https://caniuse.com/webusb

[18] What are Progressive Web Apps? (bez dat.). Web.Dev. Preuzeto 11. siječanj 2021., od https://web.dev/progressive-web-apps/

Spotifya[19], već pružaju *PWA* mogućnosti. *Chrome* i *Firefox* preglednici trenutno podržavaju najviše aplikacijskih programskih sučelja koja pružaju *PWA* mogućnosti kao što su instalacija na početni zaslon i slanje notifikacija[20]

- Web aplikacije nemaju lak pristup datotečnom sustavu računala jer se web stranice izvršavaju u kontekstu izoliranom od ostatka sustava te međusobno jedna od druge
    - Način na koji web aplikacije imaju interakciju s datotekama na korisnikovom računalu jest putem `<input type="file">` polja za unos datoteka te kasnije eksplicitnim pohranjivanjem neke datoteke pomoću iskočnog prozora. *FileSystem*[21] aplikacijsko programsko sučelje je pokušaj da se web stranicama pruži siguran pristup datotečnom sustavu računala. Kako se pregledavanjem web stranica izvršava nepoznat kod na računalu, potrebno je poduzeti stroge mjere kako bi se zaštitili korisnici. Prijedlog *FileSystem* sučelja nema slobodan pristup datotečnom sustavu, već samo datotekama kojima je korisnik eksplicitno dao pristup web aplikaciji. *FileSystem* sučelje je trenutno dostupno samo u *Chrome* i *Edge* preglednicima[22].

19 Spotify - Web Player: Music for everyone. (bez dat.). Spotify. Preuzeto 31. siječanj 2021., od https://open.spotify.com/

20 Can I use... Support tables for HTML5, CSS3, etc. (bez dat.). Preuzeto 11. siječanj 2021., od https://caniuse.com/web-app-manifest

21 *The File System Access API: Simplifying access to local files*. (bez dat.). Web.Dev. Preuzeto 07. travanj 2021., od https://web.dev/file-system-access/

22 *Can I use... Support tables for HTML5, CSS3, etc*. (bez dat.). Preuzeto 07. travanj 2021., od https://caniuse.com/filesystem

3. Moderni razvoj klijentskog dijela Web Aplikacija

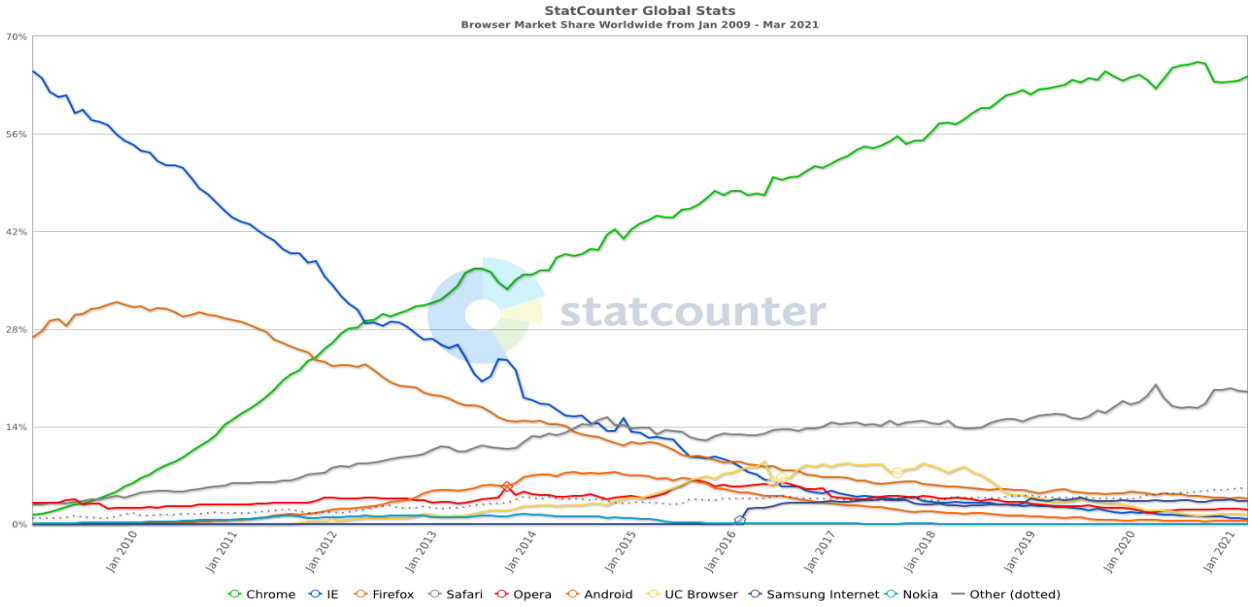
Razvoj klijentskog dijela web aplikacija podrazumijeva rad s tehnologijama *JavaScript*, *HTML* i *CSS* kako bi se prikazalo grafičko sučelje aplikacije. Web standardi se razvijaju velikom brzinom te web aplikacije dobivaju sve više i više mogućnosti kako bi korisnicima pružile najbolje iskustvo. Cijena tih mogućnosti je mnogo veća kompleksnost web aplikacija te pri razvoju većih aplikacija dolazi do mnogih izazova za čije efikasno rješavanje postojeće web tehnologije ne pružaju odgovarajuću razinu apstrakcije. Potrebu za strukturom i strukturiranim rješenjima u razvoju kompleksnih sučelja web aplikacija pokušavaju zadovoljiti mnogi razvojni okviri.

Preteča aktualnim razvojnim okvirima bile su programske knjižnice poput *jQueryja*[23] – izdanog još 2006. godine s ciljem lakšeg upravljanja *HTML* dokumentom i lakšeg izvršavanja čestih operacija poput dohvaćanja podataka sa servera i slično. *jQuery* i dalje uživa popularnost među razvijateljima sa preko 3.5 milijuna tjednih preuzimanja s najpopularnijeg registra programskih knjižnica za JavaScript, *npma*[24]. Također, jedna od bitnih stvari koje je *jQuery* omogućio razvijateljima jest neovisnost o platformi, tj. web pregledniku u kojem se kod izvršavao. U godinama koje su uslijedile, došlo je do velikih promjena u dostupnim web preglednicima na tržištu te je bilo potrebno rješenje koje će omogućiti razvoj aplikacija koje će raditi jednako u svim preglednicima. Kao što je vidljivo iz slike 4, u razdoblju nakon 2012. godine, web preglednik *Chrome* preuzima vodstvo kao najkorišteniji web preglednik gdje ostaje i danas. *Chrome* web preglednik je baziran na web pregledniku otvorenog koda *Chromiumu*[25] na kojemu su bazirani i mnogi drugi aktualni web preglednici poput *Bravea*, *Edgea*, *Opere* i drugih. Iako se čini da danas postoji mnogo web preglednika te bi problemi s interoperabilnosti i razvojem aplikacija koje rade jednako u svim preglednicima postojali i danas, to nije slučaj jer *Chromium* projekt prati standarde te

[23] js.foundation, J. F.-. (bez dat.). *JQuery*. Preuzeto 07. travanj 2021., od https://jquery.com/
[24] Npm | build amazing things. (bez dat.). Preuzeto 30. siječanj 2021., od https://www.npmjs.com/

[25] *The Chromium Projects*. (bez dat.). Preuzeto 07. travanj 2021., od https://www.chromium.org/

redovito implementira nove značajke i aplikacijska programska sučelja.



Slika 4 – postotak korištenja web preglednika od 2009. godine, preuzeto sa statcounter.com

Mnoge ideje koje su se tada pojavile u *jQueryju* su se kasnije pojavile i u samim web standardima. Današnje `fetch` aplikacijsko programsko sučelje za dohvaćanje podataka sa servera je puno sličnije *jQuery* funkciji `$.ajax()` nego `XMLHttpRequest` sučelju koje se do tada koristilo za istu svrhu. Razvojem standarda zadnjih godina te prestankom podržavanja starijih web preglednika poput *Internet Explorera, jQuery* više nije toliko potreban za razvoj aplikacija. Kao posljedica toga, pojavljuju se i projekti poput stranice *You might not need jQuery*[26] kojima je svrha educirati razvijatelje o mogućnostima u web preglednicima koje su dostupne i bez instaliranja zasebnih knjižnica.

Nakon *jQueryja*, kada se prepoznao potencijal weba i web aplikacija, pojavljuju se *MVC*[27] (engl. *model-view-controller*) razvojni okviri kako bi unijeli potrebnu strukturu u projekte. *MVC* kao obrazac je postojao već neko vrijeme na serverskoj strani, no kako je klijentski dio aplikacije do tada bio relativno jednostavan, za njime i sličnim obrascima nije bilo potrebe na klijentskoj strani. Najpopularniji predstavnici razvojnih

[26] *You Might Not Need jQuery*. (bez dat.). Preuzeto 07. travanj 2021., od http://youmightnotneedjquery.com/
[27] *MVC - MDN Web Docs Glossary: Definitions of Web-related terms | MDN*. (bez dat.). Preuzeto 07. travanj 2021., od https://developer.mozilla.org/en-US/docs/Glossary/MVC

okvira tog doba su *Backbone.js*[28], *Angular.js*[29] i *Ember*[30]. *MVC* kao obrazac nalaže razdvajanje dijelova aplikacije u 3 zasebna dijela:

- Model – služi za upravljanje podacima te implementira domensku logiku
- Pogled (engl. *view*) – služi za prikazivanje podataka koristeći model. Razdvajajući pogled kao zaseban dio moguće je implementirati nekoliko vrsta pogleda na temelju istog modela - ovisno o kontekstu
- Upravljač (engl. *controller*) – služi za povezivanje modela i pogleda te prosljeđivanje korisnikovih interakcija i pozivanje potrebnih metoda nad modelom

Noviji razvojni okviri ne nalažu razdvajanje dijelova aplikacije kao što je to slučaj kod *MVC* obrasca, već ih objedinjuju pod nazivom komponenti.

Tri trenutno najpopularnija razvojna okvira prema broju tjednih preuzimanja s *npm* registra su: *React* sa 10,589,492 preuzimanja, *Angular* sa 2,634,189 preuzimanja i *Vue.js* sa 2,332,770 preuzimanja (brojke označuju tjedna preuzimanja od 14.3.2021. – 21.3.2021.).

React je programska knjižnica za izradu korisničkih sučelja koju održava Facebook. Prva verzija objavljena je 2013. godine, a aktualna verzija je 17.0.0. U odnosu na *Angular i Vue.js*, *React* se zbog svog relativno malenog aplikacijskog programskog sučelja često naziva i knjižnicom te tako dolazi sa najmanje gotovih rješenja za izradu kompletnog klijentskog dijela[31]. Pri radu sa *Reactom*, često se koriste i dodatne knjižnice kao *react-router* za upravljanje navigacijom, *Redux* ili *MobX* za rad sa stanjem, *react-transition-group* za animacije te druge knjižnice za ostale funkcionalnosti koje ne dolaze s *Reactom*. Malo aplikacijsko programsko sučelje *Reacta* razlog je što postoji mnogo knjižnica inspiriranih *Reactom* i mnogo klonova *Reacta* s istim aplikacijskom programskim sučeljem s nekim drugim karakteristikama

[28] *Backbone.js*. (bez dat.). Preuzeto 07. travanj 2021., od https://backbonejs.org/
[29] *AngularJS — Superheroic JavaScript MVW Framework*. (bez dat.). Preuzeto 07. travanj 2021., od https://angularjs.org/
[30] *Ember.js—A framework for ambitious web developers*. (bez dat.). Preuzeto 07. travanj 2021., od https://emberjs.com/
[31] React – A JavaScript library for building user interfaces. (2021, January 10). https://reactjs.org/

– znatno manjom veličinom same knjižnice u slučaju *Preacta*[32] te većom brzinom izvođenja čestih operacija u slučaju *Infernoa*[33]

Angular je razvojni okvir za izradu aplikacija kojeg održava Google. *Angular* je novo izdanje razvojnog okvira *Angular.js*, no osim imena nema puno zajedničkog te je implementacija potpuno drugačija. Prva verzija objavljena je 2016. godine, a aktualna verzija je 11.0.5. U usporedbi s *Reactom* i *Vue.jsom*, *Angular* ima najveću površinu aplikacijskog programskog sučelja te dolazi s najviše paketa. Filozofijom se potpuno razlikuje od *Reacta* te mu je cilj pružati korisnicima sve potrebne alate za izradu kompleksnih web aplikacija. Tako uz *Angular* dolaze i paketi za navigaciju, animacije, izradu formi, testiranje, prevođenje i sl.[34]

Vue.js je razvojni okvir za izradu aplikacija čiji je autor Evan You. *Vue.js* se naziva progresivnim razvojnim okvirom jer ima sličnosti s *Reactom* i *Angularom*. Prva verzija objavljena je 2014. godine, a aktualna verzija je 3.0.5. Uz *Vue.js* ne dolaze dodatni paketi, no postoje službeno podržani paketi – *vue-router* za upravljanje navigacijom i *vuex* za upravljanje stanjem koje također održava *Vue.js* tim[35]. *Vue.js* uzima ono najbolje iz *Reacta* – jednostavnost učenja i započinjanje projekta s minimalnom količinom koda i ono najbolje iz *Angulara* – službeno podržane dodatne pakete.

U sljedećim će se poglavljima proći kroz izazove i probleme u razvoju kompleksnih web aplikacija koje razvojni okviri pokušavaju riješiti. Ovisno o opsegu, neki okviri rješavaju više, a neki manje izazova, no detekcija promjena i definiranje strukture sučelja su dva najčešća izazova koja se javljaju pri razvoju većih aplikacija te se mogu smatrati najmanjim zajedničkim nazivnikom navedenih razvojnih okvira – rješenja koja sva tri navedena okvira pružaju bez potrebe instaliranja dodatnih knjižnica kao što je to slučaj za rješenja za neke druge izazove.

[32] *Preact*. (bez dat.). Preuzeto 28. ožujak 2021., od https://preactjs.com/

[33] *Inferno*. (bez dat.). Inferno.js. Preuzeto 28. ožujak 2021., od https://www.infernojs.org/

[34] *Angular. (2021, siječanj 10). https://angular.io/*

[35] *Vue.js*. (bez dat.). Preuzeto 01. veljača 2021., od https://vuejs.org/

3.1. Komponente

Korisnička sučelja velikih web aplikacija sadrže brojne elemente koji se koriste u raznim kontekstima. Kako se kod ne bi duplicirao, elementi se apstrahiraju u zasebne, ponovno iskoristive dijelove. Sve tri tehnologije koriste istu paradigmu za izradu ponovno iskoristivih dijelova koji se zovu komponente. One apstrahiraju sva tri ključna dijela nekog elementa korisničkog sučelja: izgled, logiku i strukturu.

Komponente se zatim kompozicijom spajaju u veće komponente te hijerarhijski tvore stablo. Na vrhu stabla je najčešće komponenta koja sadrži cijelu aplikaciju. Iako je cijela aplikacija sačinjena od komponenti, to ne znači da su sve jednake. Česta podjela komponenti je na takozvane „pametne“ i prezentacijske komponente. Pametne komponente su kontejneri za prezentacijske komponente te one odrađuju posao dohvaćanja podataka, bave se obradom podataka koje je korisnik unio te reagiraju na korisnikovu interakciju s aplikacijom. Također, iz gledišta stiliziranja i slaganja komponenti u korisničkom sučelju (engl. *layout*), njihova je svrha pozicionirati komponente. Prezentacijske komponente prezentiraju podatke koje im pametne komponente proslijede. One su najčešće zasebne kontrole poput polja za unos podataka ili neke druge zasebne komponente koje su ponovno iskoristive u različitim kontekstima – npr. kartice, liste, gumbi i slični elementi korisničkog sučelja. Iz gledišta stiliziranja, idealno bi bilo kad komponente ne bi imale stilove koji određuju margine i ostale pozicijske stilove kako bi bile maksimalno iskoristive u različitim kontekstima. Ako su prezentacijske komponente polja za unos podataka, one najčešće neće same obrađivati unesene podatke, već će ih samo proslijediti prema komponentama iznad u hijerarhiji dok ne dođu do pametne komponente koja će ih obraditi.

Komponente također često imaju i tzv. metode životnog vijeka (engl. *lifecycle methods*) – funkcije koje se pozivaju u određenom trenutku života komponente. Metode životnog vijeka omogućuju razvijateljima određivanje vremena kad će se njihov kod izvršavati. Česta vremena su: prije inicijalizacije komponente, nakon što se komponenta osvježi s novim podacima te neposredno pred uništavanje komponente, tj. micanja iz dokumenta.

Neke od najčešćih operacija koje se izvode u metodama životnog vijeka su iniciranje mrežnih poziva za dohvaćanje podataka, čišćenje resursa koji su bili

alocirani za vrijeme rada komponente te izračun novih vrijednosti koje ovise o podacima koje su proslijeđene komponenti.

Komponente u razvojnim okvirima najčešće implementiraju model jednosmjernog toka podataka gdje podaci teku isključivo od vrha stabla komponenata prema dnu. Tako komponente iznad u hijerarhiji prosljeđuju komponentama ispod podatke potrebne za rad. Ako komponente niže u hijerarhiji žele promijeniti podatke u komponentama iznad u hijerarhiji, neće to učiniti izravno mijenjajući vrijednosti, već će emitirati događaj koji će sadržavati promjenu ili pozvati funkciju koju im je proslijedila komponenta iznad. Jedan od projekata koji je popularizirao ovaj model prosljeđivanja podataka u razvojnim okvirima na klijentskom dijelu jest *Flux* – *Facebookov* preporučeni obrazac za izradu aplikacija s *React* knjižnicom[36].

Jedna od početnih kritika upućenih komponentama, a pogotovo komponentama u *Reactu*, je bila što se tom paradigmom ne održava granica između zasebnih dijelova aplikacije, kao što je to slučaj s *MVC* arhitekturom, te se ne poštuje princip *SOC* (engl. *separation of concerns*). Iako neki razvojni okviri poput *Angulara* i dalje održavaju fizičku granicu između izgleda, logike i strukture stavljajući te dijelove u zasebne datoteke, aplikacije se i dalje razvijaju u smislu komponenata te su upravo one osnovne gradivne jedinice nekog korisničkog sučelja.

Osim kao dio raznih razvojnih okvira, paradigma komponenti se pojavljuje i kao dio web standarda. Web komponente[37] se sastoje od nekoliko standarda:

- Prilagođeni elementi (engl. *Custom elements*) – omogućuju definiranje novih DOM elemenata po uzoru na postojeće *HTML* elemente. Novi elementi definiraju se implementirajući novu podklasu *HTMLElement* klase te pozivajući *define* metodu nad `CustomElementsRegistry` objektom kako bi se definirala oznaka kojom će se anotirati novi prilagođeni element. Prilagođeni elementi također imaju i metode životnog vijeka kao i komponente u razvojnim okvirima koje se pozivaju u određenom trenutku izvršavanja koda prilagođene komponente.
- *Shadow DOM* – standard koji omogućuje enkapsulaciju među komponentama. Jedna od bitnijih stvari koju *Shadow DOM* omogućuje jest enkapsulacija i

[36] *Flux | Flux*. (bez dat.). Preuzeto 14. veljača 2021., od https://facebook.github.io/flux/
[37] *WICG/webcomponents*. (2021). [HTML]. Web Incubator CG. https://github.com/WICG/webcomponents (Original work published 2013)

ograničavanje opsega stilova gdje se stilovi ograničuju i primjenjuju samo na elemente unutar određenog *Shadow roota* – tj. korijena stabla *shadow* elemenata.

- *HTML* predlošci – definiranje sučelja unutar `<template>` oznaka koje se ne renderiraju odmah, već najčešće služe za definiranje sučelja u kontekstu neke komponente, slično kao i komponente u razvojnim okvirima

Iako su web komponente dio standarda te su ti standardi podržani u većini novijih web preglednika, razvojni okviri i dalje uživaju veću popularnost među razvijateljima. Jedna od kritika web komponenti jest da neki standardi ili ne rade u starijim preglednicima ili zahtijevaju korištenje dodatnih nestandardnih knjižnica kako bi se dodao dio nepodržane funkcionalnosti. Za razliku od web komponenti koje koriste funkcionalnost ugrađenu u web preglednik, razvojni okviri implementirani su u tzv. *user-spaceu,* tj. implementirani su koristeći samo aplikacijska programska sučelja dostupna i razvijateljima. Tako je većina razvojnih okvira podržana i u starijim preglednicima poput Internet Explorera 11 – izdanog 2013. godine.

3.2. Detekcija promjena

Kako sinkronizirati stanje aplikacije sa sučeljem kako se ne bi prikazale zastarjele vrijednosti stanja ili pak one koje su netočne je velik izazov pri razvoju web aplikacija. Mehanizam kojim se sučelje usklađuje sa stanjem naziva se detekcija promjena. Detekciju promjena možemo podijeliti u dvije skupine:

- ručna – gdje programer sam signalizira da je došlo do promjene stanja te se sučelje treba osvježiti.
- automatska – sučelje se osvježava kao reakcija na korisnikovu interakciju s aplikacijom, npr. klikom na dugme ili promjenom neke varijable u aplikaciji na neki drugi način, npr. uzrokovanom brojačem ili nekom sličnom zakazanom akcijom.

3.2.1. React

Jedan od najbitnijih koncepata u razvoju aplikacija sa *Reactom* je *state* – koncept koji označava trenutno stanje aplikacije. *State* sadrži trenutne vrijednosti o kojima ovisi sučelje te se sučelje smatra funkcijom stanja. Promjenom stanja, sučelje se osvježava te se prikazuju aktualne vrijednosti. Detekcija promjena se u *Reactu* izvršava ručno - pozivanjem funkcije za ažuriranje stanja `setState`. Nakon pozivanja funkcije za ažuriranje stanja, *React* zakazuje kreiranje novog stabla elemenata u memoriji koje zatim uspoređuje s trenutnim stablom elemenata. Takvo stablo elemenata u memoriji naziva se Virtualni *DOM* te mu je svrha izbjeći neefikasne promjene u pravim *HTML* elementima jer može doći i do nekoliko promjena u Virtualnom *DOM* stablu prije nego se one reflektiraju u *HTML*-u.

Nakon usporedbe dvaju stabala Virtualnog *DOM-a*, osvježavaju se samo dijelovi sučelja koji su promijenjeni. Trenutno stablo je zatim pohranjeno kako bi se moglo usporediti s novim stablom nakon iduće promjene stanja. Podstablo se smatra potpuno različitim ako je rezultat izvršavanja funkcije renderiranja različit *HTML* element te se tada u potpunosti zamjenjuje. U slučaju razlike u atributima jednakih *HTML* elemenata, vrijednosti atributa se osvježuju[38]. Proces usporedbe se tada ponavlja za svako daljnje podstablo. Sama detekcija promjena i usporedba stabala nalazi se u zasebnom paketu *react* te ne ovisi o kontekstu izvođenja – bio to web, mobilni OS ili nešto treće. Zasebni rendereri – poput *react-dom* i *react-nativea* odgovorni su za samo renderiranje i generiranje elemenata sučelja ovisno o kontekstu izvođenja.

3.2.2. Angular

Angular detekciju promjena izvršava tako da uspoređuje vrijednosti svih *bindinga* (interpoliranih, tj. umetnutih vrijednosti i evaluiranih varijabli u predlošku) i varijabli koje su proslijeđene kao ulazne vrijednosti (engl. *input*) prije i poslije pokretanja detekcije promjena. *Angular* zna koje varijable treba usporediti jer su predlošci statički definirani te se prilikom njihove kompilacije detektira koje varijable su

[38] Reconciliation – React. (bez dat.). Preuzeto 11. siječanj 2021., od https://reactjs.org/docs/reconciliation.html

interpolirane. Ako se vrijednost varijable promijenila, potrebno je osvježiti sučelje kako bi se prikazale nove vrijednosti[39].

Kako bi se automatiziralo pokretanje ciklusa detekcije promjena nakon korisnikove interakcije s aplikacijom, koristi se *zone.js. Zone.js* je knjižnica koja mijenja implementaciju standardnih asinkronih *DOM* funkcija poput `setTimeout`, `addEventListener`, `XMLHttpRequest` i drugih kako bi se mogao promatrati tok njihovog izvršavanja[40]. Nakon što se funkcija izvrši, *Angular* poziva funkciju `tick` `ApplicationRef` sučelja koja pokreće detekciju promjena na vrhu stabla komponenata. Zadana strategija detekcije promjena uspoređuje sva polja nekog objekta koji je proslijeđen kao ulazna vrijednost, no postoji i opcija `onPush` detekcije promjena koja uspoređuje samo reference objekata te tako optimizira cijeli proces.

Također, programerima je na raspolaganju i ručna detekcija promjena korištenjem servisa koji nasljeđuje klasu `ChangeDetectorRef` i metodama poput `detectChanges`. Moguće je tako razviti aplikaciju bez *zone.js* knjižnice te ručnom detekcijom promjena implementirati strategiju sličnu onoj u *Reactu*. Jedna od prednosti takvog pristupa je znatno manja veličina aplikacije jer je *zone.js* knjižnica veličine 49.5 kB nakon minificiranja (proces smanjivanja veličine datoteke brisanjem znakova koji ne mijenjaju značenje koda, poput znakova novog retka, tabulatora ili razmaka na nekim mjestima), no to nije česta praksa te *zone.js* knjižnica dolazi instalirana uz *Angular*.

3.2.3. Vue.js

Vue.js koristi moderne *ECMAScript* značajke kako bi automatski osvježio sučelje nakon što je došlo do promjene stanja u aplikaciji. Takozvana reaktivnost je u *Vue.jsu* postignuta uz pomoć *Proxy* strukture podataka[41]. *Proxy* struktura podataka služi kako bi se presrela interakcija s ciljanim objektom te omogućuje metaprogramiranje u *JavaScriptu*. Najčešće se presreće dohvaćanje vrijednosti

[39] Angular Change Detection—How Does It Really Work? (2016, lipanj 21). Angular University Blog. https://blog.angular-university.io/how-does-angular-2-change-detection-really-work/

[40] Zone.js. (bez dat.). npm. Preuzeto 11. siječanj 2021., od https://www.npmjs.com/package/zone.js

[41] Reactivity in Depth | Vue.js. (bez dat.). Preuzeto 17. siječanj 2021., od https://v3.vuejs.org/guide/reactivity.html#what-is-reactivity

svojstva objekta s *get* svojstvom *Proxyja* te postavljanje vrijednosti svojstva objekta sa *set* svojstvom *Proxyja*[42]. Konkretno, u slučaju *Vue.jsa*, uz pomoć *Proxyja* se presreće interpolacija svojstava data objekta neke komponente te se označava koji dio koda je pristupio tom svojstvu kako bi se, kada se vrijednost svojstva promijeni, taj dio koda mogao ponovno izvršiti, tj. sučelje osvježiti. Prednost takvog pristupa u odnosu na eksplicitni način, kao primjerice u *Reactu,* jest prirodno korištenje *JavaScript* programskog jezika te manja površina aplikacijskog programskog sučelja same knjižnice.

Nakon što su detektirane promjene i poznate su komponente koje se trebaju osvježiti, *Vue.js* koristi istu tehniku generiranja novog Virtualnog *DOM* stabla kao i *React* koje se uspoređuje s prethodnim kako bi se osvježilo sučelje.

3.3. Definiranje strukture grafičkog sučelja komponenti

Korisnička sučelja velikih web aplikacija sadrže brojne elemente koji se koriste u raznim kontekstima. Kako se kod ne bi duplicirao, elementi se apstrahiraju u već navedene komponente. Unatoč istoj paradigmi, sama implementacija i korištenje komponenti se znatno razlikuju. Definicija strukture komponenti nalazi se na spektru od potpuno dinamičkog do potpuno statičkog. Medu navedenim okvirima, *React* je najdinamičniji, *Angular* najstatičniji, a *Vue.js* je na sredini [43].

3.3.1. React

Za definiranje izgleda komponenata u *Reactu,* koristi se *JSX* koji je jezik sličan *HTML-u*, no dopušta ugnježđivanje proizvoljnih JavaScript izraza što omogućuje maksimalnu slobodu u definiranju sučelja[44]. *JSX* se kompilacijom pretvara u pozive funkcija `createElement` te je tako potpuno dinamičan. Iako dinamičnost pruža slobodu i ekspresivnost, pozive funkcija je vrlo teško optimizirati u usporedbi sa statički definiranim sučeljima - što pri većim aplikacijama može dovesti do problema s

[42] Proxy—JavaScript | MDN. (bez dat.). Preuzeto 17. siječanj 2021., od https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Proxy

[43] JSConf. (2019, srpanj 3). Evan You on Vue.js: Seeking the Balance in Framework Design | JSConf.Asia 2019. https://www.youtube.com/watch?v=ANtSWq-zI0s

[44] *Introducing JSX – React*. (bez dat.). Preuzeto 01. veljača 2021., od https://reactjs.org/docs/introducing-jsx.html

performansama. Rezultat izvršavanja funkcija je već navedeni virtualni *DOM* – reprezentacija *HTML* elemenata u memoriji. Kako bi se optimizirao ovakav tip pogleda, koristi se tehnika zvana memoizacija[45] – spremanje rezultata izvršavanja funkcije te ponovne uporabe tog rezultata ako se argumenti funkcije nisu promijenili.

3.3.2. Angular

Za definiranje izgleda komponenti u *Angularu,* koriste se predlošci (engl. *template*). *Angular* predložak je super-set *HTML*-a koji sadrži tzv. direktive i interpolirane vrijednosti.

Direktive su objekti koji se koriste kao atributi *HTML* elemenata u predlošcima kako bi im dali nove mogućnosti. Direktive dolaze u dvije varijante:

- Atributne direktive – mijenjaju atribute *HTML* elemenata na kojima su definirane kao atributi
- Strukturalne direktive – mijenjaju strukturu *HTML* podstabla nad čijim su roditeljima definirane kao atributi

Predlošci su ograničeniji u ekspresivnosti nego potpuni programski jezici, ali se zbog toga mogu puno bolje optimizirati[46]. Konkretno, *Angular* kompilira predloške u instrukcije koje se izvršavaju tijekom rada aplikacije kako bi se osvježio pogled. Prilikom kompilacije predložaka, poznato je koji dijelovi sučelja su potpuno statički, pa se ne moraju iznova izvršavati funkcije koje ih generiraju kao što je to slučaj u *Reactu*.

3.3.3. Vue.js

Vue.js spaja pristupe *Reacta* i *Angulara* za definiranje izgleda komponenti te tako pruža korisnicima mogućnost korištenja predložaka, koji se, za razliku od *Angulara* ne pretvaraju u instrukcije, već se kompiliraju u pozive funkcija koje generiraju virtualni *DOM* - kao što je to slučaj kod *Reacta*. Korisnici također mogu, ako im je potrebno, definirati izgled komponenti potpuno dinamički, ručnim pozivanjem tih

[45] *Memoization and Laziness*. (bez dat.). Preuzeto 01. veljača 2021., od https://www.cs.cmu.edu/~rwh/introsml/techniques/memoization.htm
[46] *Google/incremental-dom*. (2021). [TypeScript]. Google. https://github.com/google/incremental-dom (Original work published 2015)

funkcija te tako iskoristiti svu moć i ekspresivnost proizvoljnih *JavaScript* izraza. Prednost korištenja predložaka je što ih *Vue.js* kompajler pretvara u optimiziranije pozive funkcija jer prepoznaje koji su dijelovi sučelja statički definirani te se neće mijenjati kroz rad aplikacije.

3.4. Strategija renderiranja

Iako je glavna primjena razvojnih okvira u početku bila izrada aplikacija jednog dokumenta (engl. *Single Page Application* – kratica *SPA*), danas postoji nekoliko strategija renderiranja[47] koje se razlikuju po mjestu i načinu renderiranja sučelja:

- *Server Side Rendering* (kratica *SSR*), tj. renderiranje na serveru je najstarija strategija. Svaka korisnikova akcija koja zahtjeva promjenu stanja u aplikaciji izvršava se preko navigacije što uzrokuje ponovno učitavanje aplikacije. Rezultat navigacije je novi *HTML* dokument gdje je najčešće prikazana posljedica korisnikove akcije – bila to greška, ili novo stanje. Ovakav pristup je najjednostavniji za implementaciju, ali nije prikladan za vrlo interaktivne aplikacije zbog vremena koje korisnik mora provesti čekajući nakon svake akcije. Osim jednostavnosti implementacije, prednost ovog pristupa je i to što pretraživači lako mogu indeksirati sadržaj aplikacije jer je sadržaj prisutan već u samom *HTML* dokumentu.
- *Client-Side Rendering* (kratica *CSR*), tj. renderiranje na klijentu je strategija gdje ne dolazi do ponovnog učitavanja aplikacije nakon neke korisnikove akcije koja mijenja stanje u aplikaciji. Takve aplikacije se nazivaju *Single Page* aplikacije, tj. aplikacije jednog dokumenta. Rezultat mrežnih zahtjeva koji su posljedica korisnikovih akcija nije potpuno novi *HTML* dokument već su to podaci, najčešće u *JSON* ili *XML* formatu, a može biti i samo dio *HTML* dokumenta koji se treba osvježiti[48]. Ovakav pristup koristi se za interaktivne aplikacije gdje se skraćuje vrijeme između korisnikove akcije i vidljivog rezultata, no na slabijim uređajima može uzrokovati probleme s performansama. Također, mana ovakvog pristupa jest neprilagođenost web

[47] Rendering on the Web. (bez dat.). Google Developers. Preuzeto 17. siječanj 2021., od https://developers.google.com/web/updates/2019/02/rendering-on-the-web

[48] HTML Over The Wire | Hotwire. (bez dat.). Preuzeto 10. siječanj 2021., od https://hotwire.dev/

aplikacija tražilicama. Kako u početnom *HTML* dokumentu najčešće nema stvarnog sadržaja stranice, da bi dobile sadržaj koji mogu indeksirati, tražilice bi trebale izvršavati *JavaScript* kod. Neke tražilice poput Googleove upravo to i rade[49], no postoji i mnogo drugih tražilica koje još uvijek indeksiraju samo statički sadržaj. Kod ovakvog pristupa inicijalni *HTML* dokument se sastoji samo od nekoliko *HTML* oznaka čija je svrha biti kontejner gdje će se renderirati ostatak aplikacije.

- *SSR* + hidracija je strategija gdje se inicijalno renderiranje stranice odvija na serveru u obliku statičkog *HTML* dokumenta kojeg zatim preuzima klijent – proces koji se naziva hidracijom (engl. *hydration*) te se sve daljnje dinamičke promjene izvršavaju na klijentu bez ponovnog učitavanja stranice[50]. Prednosti ove strategije renderiranja je što pretraživači mogu lako indeksirati sadržaj aplikacije jer je on prisutan u *HTML* dokumentu već pri prvom zahtjevu, a korisnik nakon kasnijih interakcija ne mora čekati vrijeme kako bi se aplikacija ponovno učitala zbog koraka hidracije. Nedostaci ove strategije su što se sadržaj efektivno renderira dva puta – jednom na serveru, a drugi puta na klijentu te je zbog procesa hidracije *Time to Interactive* (kratica *TTI*), koji je jedna od bitnijih metrika pri mjerenju performansa web aplikacija[51], nešto dulji jer korisnik vidi renderirano sučelje s kojim ne može odmah imati interakciju.
- *SSR* + *static prerender* je novija strategija koju pružaju neki razvojni okviri poput *Next.jsa*[52] koji u pozadini koristi *React* kako bi se dodatno optimizirale neke stranice. Analizom ili posebnim označavanjem određenih stranica, one se renderiraju prilikom kompilacije aplikacije. Podaci se dinamički dohvaćaju iz baze ili nekog drugog mjesta te se dobiva statički *HTML* dokument koji se prilikom korisničkog zahtjeva ne treba iznova generirati. Iako ovakav pristup sam po sebi nije nov, posebnost je njegova integracija s ostalim načinima

[49] *Understand the JavaScript SEO basics | Google Search Central*. (bez dat.). Google Developers. Preuzeto 01. veljača 2021., od https://developers.google.com/search/docs/guides/javascript-seo-basics

[50] Client Side Hydration | Vue SSR Guide. (bez dat.). Preuzeto 10. siječanj 2021., od https://ssr.vuejs.org/guide/hydration.html

[51] Time to Interactive. (bez dat.). Web.Dev. Preuzeto 10. siječanj 2021., od https://web.dev/interactive/

[52] *Advanced Features: Automatic Static Optimization | Next.js*. (bez dat.). Preuzeto 21. ožujak 2021., od https://nextjs.org/docs/advanced-features/automatic-static-optimization

renderiranja što omogućava razvijateljima da što vise optimiziraju različite dijelove aplikacije ovisno o njihovom načinu korištenja.

3.5. Navigacija

Osim poveznica koje se u web aplikacijama definiraju *anchor* (*<a>)* oznakama, moderne web aplikacije pružaju i programatske načine navigacije kroz aplikaciju. Osim samog načina navigiranja kroz aplikaciju, razvojni okviri često pružaju i dodatne mogućnosti usko vezane za navigaciju. Dio razvojnog okvira koji upravlja navigacijom naziva se usmjerivač (engl. *router*). Usmjerivač može dinamički učitati potrebne podatke, primjerice sa poslužitelja, za prikaz neke komponente prije nego je ta komponenta prikazana u korisničkom sučelju kako korisnici ne bi nepotrebno čekali osvježavanje sučelja. Jednostavnije vrijednosti se također mogu prosljeđivati i prilikom navigacije, najčešće koristeći putanju. *URL* (engl. *Uniform Resource Locator*) je jedinstvena adresa do nekog sadržaja na webu[53]. Parametri su dodatni podaci u *URL-u* koji se koriste kako bi se vratio ispravan resurs te se dodaju na kraju *URL-a*, nakon znaka `?` u obliku `ključ=vrijednost`. Pohranom trenutnog stanja aplikacije u putanju, korisnike prilikom navigiranja na neku stranicu dočekuje spremno korisničko sučelje te se takve putanje mogu dijeliti među korisnicima. Osim ovakvog načina prosljeđivanja podataka među rutama, podaci se često mogu prosljeđivati i na način koji neće biti vidljiv u putanji. Naravno, takvo prosljeđivanje podataka funkcionira samo ako se navigira unutar same aplikacije gdje usmjerivač ima kontrolu nad njom.

Usmjerivač također često služi kako bi se ograničio pristup određenim stranicama. Primjerice, koristeći usmjerivač može se ograničiti pristup korisničkim stranicama ako korisnik nije prijavljen ili pak ograničavanje sljedećeg koraka neke forme ako prethodni korak nije ispravno popunjen.

Također, usmjerivač često ima i ulogu u strukturiranju aplikacije. Hijerarhijskim ugnježđivanjem ruta navigacije i korištenjem komponenti koje određuju raspored (engl. *layout*) izbjegava se duplikacija. Primjerice, navigacijska traka i podnožje se mogu prikazivati na nekim rutama, a sakriti na drugima.

[53] *What is a URL? - Learn web development | MDN*. (bez dat.). Preuzeto 04. travanj 2021., od https://developer.mozilla.org/en-US/docs/Learn/Common_questions/What_is_a_URL

3.6. Stiliziranje i animiranje

Pri razvoju velikih aplikacija koristeći klasično stiliziranje web aplikacija putem *CSS-a*, dolazi do problema s održavanjem i dodavanjem novih stilova. Najveći problem je kako ograničiti opseg (engl. *scoping*) nekog *CSS* pravila kako bi ono stiliziralo samo željene elemente bez da uzrokuje neželjene promjene u postojećim elementima. Tradicionalno se taj problem rješavao posebnim pravilima imenovanja klasa koje se dodjeljuju kao atributi *HTML* elementima, poput npr. *BEM*[54] metodologijom koja koristi strukturu *blok__element—modifikator.* Prilikom označavanja komponenti *BEM* klasama – komponenta bi bila blok, manji dijelovi sučelja koji čine komponentu bi bili element, a posebne opcije za mijenjanje komponente, npr. tamna tema, bile bi modifikator. Nedostaci ovakvog pristupa su: veliki broj klasa koji neki element može imati dodijeljeno te osmišljavanje imena koja se ne dupliciraju samo kako bi se izbjegao utjecaj stilova na neki drugi element. Razvojni okviri često pružaju vlastito rješenje za ograničavanje opsega stilova na način da ograniče opseg stilova samo na komponentu unutar koje su definirani. Jedan od načina kako se to može izvesti jest da se za svaku komponentu generira jedinstveni identifikator. On se tada postavlja kao prefiks *CSS* klasa dodijeljenih toj komponenti što uzrokuje izolaciju stilova.

Osim statičkog stiliziranja, elementi sučelja na webu se mogu i animirati. Razlika između animacija i tranzicija je broj stanja između kojih se animira promjena. Tranzicije su animirane promjene između dva stanja, dok animacije podržavaju više stanja. Animacije se također mogu izvršavati u petlji i unatrag. Kompleksnije web aplikacije mogu imati mnogo animacija i tranzicija te razvojni okviri često pružaju jednostavniji način za definiranje stanja i ponovnu iskoristivost kako bi se izbjegla duplikacija. One u pozadini mogu koristiti *CSS* animacije ili web animacijsko aplikacijsko programsko sučelje.

3.7. Upravljanje stanjem

U web aplikaciji može biti mnogo stanja na klijentskom dijelu. Primjeri nekih stanja su:

[54] contributors, V. S., Vladimir Starkov and. (bez dat.). *BEM — Block Element Modifier*. Preuzeto 27. ožujak 2021., od http://getbem.com/

- stanje učitavanja prilikom dohvaćanja podataka sa servera
- stanje pogreške prilikom dohvaćanja podataka ili neke druge pogreške
- stanje kada su podaci uspješno dohvaćeni i željena operacija je uspješno izvršena.

Navedena stanja može prouzročiti i samo jedan mrežni zahtjev, a kompleksna moderna aplikacija, poput npr. Facebooka[55], ih tijekom rada napravi i stotine. Razvojni okviri pružaju rješenja za upravljanje stanjem do neke mjere, no u kompleksnijim aplikacijama je potrebno specijalizirano rješenje. Specijalizirane knjižnice za upravljanje stanjem često imaju i adaptere za najpopularnije razvojne okvire kako bi se što lakše integrirale s njima. Jedan od najvećih problema s velikim objektima stanja jest mogućnost praćenja odakle dolaze promjene te međuovisnost promjena – kada promjena stanja u jednom dijelu aplikacije ovisi o promjeni stanja u drugom dijelu aplikacije. Primjerice, dolazeća poruka u *chat* aplikaciji može utjecati na broj nepročitanih poruka, broj notifikacija, redoslijed prikaza razgovora (razgovori s najnovijim porukama na vrhu) i slično. Jedno od rješenja za taj problem jest centralizacija stanja i ograničavanje načina na koji se stanje može promijeniti. Konkretno, u slučaju knjižnice za upravljanje stanjem *Redux*[56], stanje je definirano kao jedan veliki *JavaScript* objekt. Promjena stanja se vrši isključivo putem tzv. akcija koje imaju jasno definirano sučelje. Centralizacijom i standardizacijom stanja i načina promjene stanja se tada omogućuje razvoj specijaliziranih alata koji olakšavaju pregledavanje stanja, načina kako je došlo do trenutnog stanja te simuliranja budućih stanja. Jedan od takvih alata za knjižnicu *Redux* jest *redux-devtools*[57]

3.8. Sigurnost i testiranje

Kako bi se aplikacije osigurale od napada i spriječile greške, tj. *bugovi*, preporuča se implementiranje testova. Testovi su programi koji izvršavaju druge programe i validiraju rezultat njihovog izvršavanja. Velik broj komponenti i veća

[55] *Facebook—Log In or Sign Up*. (bez dat.). Facebook. Preuzeto 01. veljača 2021., od https://www.facebook.com/
[56] *Redux—A predictable state container for JavaScript apps.* | *Redux*. (bez dat.). Preuzeto 14. ožujak 2021., od https://redux.js.org/
[57] *Reduxjs/redux-devtools*. (2021). [TypeScript]. Redux. https://github.com/reduxjs/redux-devtools (Original work published 2015)

količina izvornog koda znači i više mogućnosti za pojavu grešaka[58]. Razvojni okviri često pružaju načine kako najbolje testirati velik broj komponenti izrađenih tim razvojnim okvirom koje imaju velik broj stanja i ovise o velikoj količini ostalog izvornog koda aplikacije. *Angular* tako pruža razvijateljima napredne načine testiranja koristeći `TestBed`[59] klasu koja koristi *Angularov* injektor ovisnosti kako bi se kod komponente izvršavao sa testnim vrijednostima. Komponente razvijene nekim od razvojnih okvira najčešće se testiraju na način da se komponentama kao ulazni argument proslijedi neka vrijednost te se zatim u testu provjerava je li renderirano sučelje sa željenom strukturom. Također, s komponentama se često izvršava interakcija iz korisnikove perspektive gdje se programatski upravlja kontrolama u sučelju.

Prema *OWASP-u* (Open Web Application Security Project), *XSS*[60] (Cross-Site Scripting) je jedan od deset najčešćih načina sigurnosnih napada na webu. Najčešći način izvršavanja *XSS* napada je umetanjem malicioznog koda u kod aplikacije, primjerice koristeći polja za unos podataka. Web aplikacije često u korisničkom sučelju prikazuju neki sadržaj koji je generirao neki drugi korisnik. Prilikom prikazivanja korisnički generiranog sadržaja bitno je sanirati taj sadržaj kako ne bi došlo do sigurnosnog propusta te curenja korisničkih podataka ili neke druge neželjene posljedice slučajnog izvršavanja umetnutog malicioznog koda. Razvojni okviri često zadano prikazuju sadržaj saniran, a za prikazivanje nesaniranog sadržaja potrebno je koristiti posebne metode. Primjerice, u *Reactu* je nesanirani sadržaj moguće prikazati samo prikladno imenovanim svojstvom komponente – `dangerouslySetInnerHTML`.

3.9. Ostali izazovi

Neki od ostalih izazovi koje rješavaju razvojni okviri su:

- Forme - razvojni okviri često olakšavaju definiranje i upravljanje kompleksnim formama. Validacija, grupiranje te međuovisnost kontrola – kad promjena

[58] Tanenbaum, A. S., & Wetherall, D. J. (2012). Computer Networks (5th edition). Pearson.

[59] *Angular—TestBed*. (bez dat.). Preuzeto 05. travanj 2021., od https://angular.io/api/core/testing/TestBed

[60] *OWASP Top Ten Web Application Security Risks | OWASP*. (bez dat.). Preuzeto 05. travanj 2021., od https://owasp.org/www-project-top-ten/

vrijednosti jedne kontrole utječe na vrijednost, samo su neki od izazova pri razvoju kompleksnih formi

- Prevođenje - velike web aplikacije poput globalnih web dućana, npr. *Ebaya*[61] i *Amazona*[62] pružaju lokalizirane verzije svojeg sučelja korisnicima diljem svijeta. Razvojni okviri i njihove pripadajuće knjižnice često pružaju gotova rješenja za lokalizaciju aplikacija.

[61] *Electronics, Cars, Fashion, Collectibles & More*. (2021, siječanj 30). EBay. https://www.ebay.com

[62] *Amazon.com: Online Shopping for Electronics, Apparel, Computers, Books, DVDs & more*. (2021, siječanj 30). https://www.amazon.com/

4. Demo aplikacija

U sklopu diplomskog rada izrađena je jednostavnija jednaka aplikacija u sva tri razvojna okvira kako bi se analizirale njihove razlike i sličnosti. Aplikacija se naziva *GH-Demo* i služi za prikaz informacija o *GitHub*[63] korisnicima i njihovim repozitorijima. *GitHub* pruža korisnicima usluge upravljanja izvornim kodom te uslugu pohrane izvornog koda u repozitorije. Demo aplikacija sadrži elemente koji demonstriraju neke od najčešćih funkcionalnosti web aplikacija – navigaciju, rad s formama, povezivanje s drugim mrežnim servisima putem *HTTP* zahtjeva i animacije. Također, demonstrira već navedene koncepte razvojnih okvira - detekciju promjena i apstrakciju sučelja.

Aplikacija je strukturirana kao dvije zasebne stranice: pretraga korisnika po imenu – prikazana na slici 16 u prilogu te informacije o pojedinom korisniku i njegovim repozitorijima – prikazana na slici 17 u prilogu.

Kako bi se dobile informacije o repozitorijima, koristi se *GitHub API* servis koji pruža *REST* aplikacijsko programsko sučelje za dohvaćanje podataka[64].

Kako bi se izjednačile mogućnosti razvojnih okvira, u *React* verziji aplikacije korištene su dodatne programske knjižnice koje nisu dio samog *Reacta* – *react-router-dom* za navigaciju i *react-transition-group* za animacije. U *Vue.js* verziji aplikacije je također korištena dodatna knjižnica – *vue-router* koja je službeno podržana knjižnica za navigaciju koju održava *Vue.js* tim. Uz *Angular* nisu instalirane dodatne knjižnice jer su sve navedene funkcionalnosti već podržane u knjižnicama priloženim uz sam *Angular*.

4.1. Inicijalizacija projekta

Za prvotno postavljanje i inicijalizaciju projekta sa svakim okvirom korišteni su službeni alati za komandne linije: *create-react-app*[65] za React, *angular-cli*[66] za Angular

[63] Build software better, together. (bez dat.). GitHub. Preuzeto 25. siječanj 2021., od https://github.com

[64] GitHub REST API - GitHub Docs. (bez dat.). Preuzeto 25. siječanj 2021., od https://docs.github.com/en/rest

[65] Create React App. (bez dat.). Preuzeto 25. siječanj 2021., od https://create-react-app.dev/

[66] Angular CLI. (bez dat.). Preuzeto 25. siječanj 2021., od https://cli.angular.io/

i *vue-cli*[67] za *Vue.js*. Navedeni alati znatno olakšavaju i ubrzavaju proces inicijalizacije novih projekata. Za optimalno iskustvo razvoja web aplikacija s navedenim razvojnim okvirima, alati komandne linije koriste knjižnice poput *webpacka*[68] i *Babela* za:

- Automatsko osvježavanje aplikacije u pregledniku nakon promjene u izvornom kodu kako bi se ubrzao iterativni razvoj aplikacije.
- Spajanje više datoteka izvornog koda u jednu (engl. *bundling)* zbog efikasnijeg slanja datoteka preko HTTP 1.1 protokola.
- Razdvajanje koda prema navigacijskoj strukturi aplikacije kako bi se optimiziralo učitavanje aplikacije, tj. kako bi se kod potreban za prikaz neke stranice učitao tek kad korisnik navigacijom dođe na tu stranicu.
- Micanje koda koji se neće izvršiti tijekom rada aplikacije (engl. *tree-shaking*). Statičkom analizom izvornog koda utvrđuje se koji se dio koda referencira iz nekog drugog te je do njega moguće doći, a koji dio ne. Dio koji se nigdje drugdje ne referencira se tada ne uključuje u završni kod aplikacije te se tako smanjuje veličina.
- Kompilaciju novijih značajki *JavaScripta* u ekvivalentni kod starije verzije radi bolje podrške u web preglednicima i kompilaciju jezika poput *TypeScripta*[69] i već navedenog *JSX-a* u *JavaScript* koji web preglednici mogu izvršiti
- Uvoz i referenciranje datoteka koje inače ne bi moglo biti moguće referencirati u kodu poput slika u *SVG* formatu
- Kompilaciju različitih *CSS* pred procesora

4.2. Razvoj projekta

Kako sva tri razvojna okvira koriste paradigmu komponenti za apstrakciju ponovno iskoristivih dijelova korisničkih sučelja, sama struktura aplikacije je vrlo slična. Jedna od najvećih razlika u razvoju aplikacije je u ekosustavima razvojnih okvira. Primjerice, u radu s *Angularom* je obavezno korištenje *TypeScript* programskog jezika.

[67] Vue CLI. (bez dat.). Preuzeto 25. siječanj 2021., od https://cli.vuejs.org/

[68] Webpack. (bez dat.). Webpack. Preuzeto 25. siječanj 2021., od https://webpack.js.org/

[69] Typed JavaScript at Any Scale. (bez dat.). Preuzeto 25. siječanj 2021., od https://www.typescriptlang.org/

- *TypeScript* je programski jezik koji je super set *JavaScripta* te mu dodaje statički definirane tipove podataka. *TypeScript* pruža veću sigurnost pri razvoju većih aplikacija jer statički definirani tipovi pretvaraju greške koje bi se pojavile prilikom izvođenja aplikacije u greške koje se pojavljuju prilikom kompilacije – npr. pogrešan tip parametra proslijeđen nekoj funkciji ili pozivanje nepostojeće metode nad nekim objektom – operacije koje je moguće izvesti u *JavaScriptu* zbog njegove dinamičke naravi. *TypeScript* se kompilacijom prevodi u *JavaScript* koji je tada moguće izvršavati u web preglednicima. *TypeScript* se u početku uspoređivao sa programskim jezikom *CoffeeScript*[70] koji je jedan od prvih programskih jezika koji se koristio na webu kompilacijom u *JavaScript*, no velika razlika između *CoffeeScripta* i *TypeScript* je što *CoffeeScript* ima vrlo različitu sintaksu od *JavaScripta*, dok je *TypeScript* super set *JavaScripta* što znači da je svaki validan *JavaScript* program također i validan *TypeScript* program.

S ostalim okvirima, *TypeScript* podrška se može dodati ručno ili korištenjem pripadajućih alata komandne linije, no on nije nužan za njihovo korištenje. Također, *Angular* se razlikuje od ostalih okvira po uskoj integraciji s knjižnicom *RxJS* koja uvelike mijenja način rada s *Angularom* koristeći paradigmu funkcijsko-reaktivnog programiranja (kratica *FRP*).

- *FRP* modelira promjene u aplikaciji kao događaje na koje se drugi dio koda može pretplatiti, tj. pratiti njihovo emitiranje. *RxJS* sadrži mnoštvo operatora – funkcija koje se primjenjuju kako bi se pretvorili podaci koje neki *Observable* (ime za objekt koji proizvodi i emitira događaje) emitira. Operatori se zatim nižu jedan za drugim kompozicijom funkcija koristeći *pipe* operator. Neki od *Rx* operatora su:
    - *map* – operator koji pretvara jedan podatak u drugi
    - *filter* – operator koji miče vrijednosti koje ne zadovoljavaju funkciju predikata
    - *first* – operator koji prestaje s pretplatom na neki *Observable* nakon sto je emitirana prva vrijednost koja zadovoljava funkciju predikata

[70] *CoffeeScript*. (bez dat.). Preuzeto 07. travanj 2021., od https://coffeescript.org/#introduction

Prednost korištenja *RxJSa* i *FRP* paradigme jest što dio aplikacije koji konzumira neki *Observable* ne mora brinuti odakle podaci zapravo dolaze – njihov izvor može biti neka asinkrona ili sinkrona operacija, neki događaj ili nešto treće.

4.2.1. GitHub API

Za spajanje na *GitHubovo* aplikacijsko programsko sučelje i dohvaćanje i pretragu podataka o korisnicima i njihovim repozitorijima koriste se servisi `SearchService` i `UserService`. U *React* i *Vue.js* okvirima korišten je jednak kod za definiranje klase servisa, dok se u *Angularu* koristi zadana klasa `HttpClient` koja pruža integraciju s *RxJS* knjižnicom pretvarajući odgovore na HTTP zahtjeve u *Observable* vrijednosti. Svaki od razvojnih okvira pruža mehanizam prosljeđivanja vrijednosti komponentama koje su nekoliko razina ispod komponente gdje su one definirane. U *Angularu* je to injektor ovisnosti, u *Reactu* `Context`, a u *Vue.jsu* `provide` i `inject` svojstva komponenata. Takav način prosljeđivanja zajedničkih ovisnosti u daljnjem razvoju aplikacije olakšava i testiranje jer je ovisnosti moguće lako zamijeniti prilikom pisanja testova sa servisima koji pozivaju neko drugo programsko sučelje ili vraćaju testne vrijednosti. U protivnom, ako bi komponente same dohvaćale servise koji su im potrebni za rad, testiranje bi bilo otežano jer bi se u izvornom kodu komponente izravno referencirao kod servisa te bi ga prilikom testiranja bilo teže izmijeniti.

4.2.2. Definiranje komponenti

Pri radu s *Angularom*, komponente se definiraju pomoću *JavaScript* klasa i *TypeScript* dekoratora. Dekoratori su eksperimentalna značajka *TypeScripta*, dok su u *JavaScriptu* u procesu standardizacije. Njihova svrha je lako modificiranje postojećih klasa i njihovih članova – metoda i svojstava. U *Angularu* se dekorator `@Component` koristi kako bi se *JavaScript* klasi dodali metapodaci koji su potrebni za rad s komponentama – koji predložak komponenta koristi (bilo to izravno definiranje predloška ili putanja do predloška), koji stilovi se primjenjuju te neke dodatne opcije poput odabir tipa detekcije promjena. Izvršavanje koda u nekom trenutku života komponente u *Angularu* se izvodi implementiranjem metoda raznih zadanih sučelja.

Tako sučelje `onInit` definira metodu `ngOnInit` koju će *Angular* pozvati pri inicijalizaciji komponente. Često korištena sučelja su i `OnDestroy` koje definira metodu `ngOnDestroy` koja se poziva prilikom micanja komponente te `OnChanges` koje definira metodu `ngOnChanges` koja se poziva prilikom promjene u vrijednostima koje su proslijeđene komponenti. Za pohranu stanja o repozitorijima i informacijama korisnika, koriste se *RxJS* klase `ReplaySubject` koje svim novim pretplatnicima pružaju prethodno emitirane vrijednosti do *n* posljednjih emitiranja. Za pohranu stanja korisničkog sučelja – stanje učitavanja i stanje potencijalne greške, koristi se *RxJS* klasa `BehaviorSubject` koja je slična `ReplaySubjectu` sa *n=1*, no nužno je definiranje i zadane vrijednosti. Koristeći *Angularov* injektor ovisnosti kroz konstruktor klase komponente, dohvaća se instanca servisa `UserDetailsService` kojom se obavljaju mrežni zahtjevi. Prosljeđivanjem novih vrijednosti korisničkih detalja i informacija o repozitoriju unutar `subscribe` metode `Observable` sučelja, sve komponente koje su pretplaćene bit će osvježene.

Dio koda komponente `UserDetailsComponent` čija je svrha dohvaćanje informacije korisnika:

```
@Component({
  selector: 'gh-user-details',
  templateUrl: './user-details.component.html',
  styleUrls: ['./user-details.component.scss'],
  animations: [listAnimation],
})
export class UserDetailsComponent {
  public repos$ = new ReplaySubject<Array<Repo>>();
  public userDetails$ = new ReplaySubject<User>();
  public loading$ = new BehaviorSubject<boolean>(true);
  public error$ = new BehaviorSubject<string>('');

  constructor(private route: ActivatedRoute, private userService: UserService) {
    this.route.paramMap
      .pipe(
        first((params) => typeof params.get('username') === 'string'),
        switchMap((params) => {
          const username = params.get('username')!;

          return forkJoin([
            this.userService.getUser(username),
            this.userService.getUserRepos(username),
          ]);
        }),
        delay(1000),
        finalize(() => {
          this.loading$.next(false);
        })
      )
      .subscribe({
        next: ([userDetails, repos]) => {
          this.userDetails$.next(userDetails);
          this.repos$.next(repos);
        },
        error: () => this.error$.next('An error has occured'),
      });
  }
}
```

Slika 5 – `UserDetailsComponent` komponenta u *Angularu*

Komponenta `UserDetailsComponent` zatim prosljeđuje informacije o pojedinom repozitoriju komponenti `RepositoryCardComponent` koja ih prima kao ulaznu vrijednost koristeći `@Input` dekorator nad nekim poljem komponente – u ovom slučaju polju `repo`, gdje će se pohraniti podaci o proslijeđenom repozitoriju. Također, koristi se i `onPush` vrijednost `ChangeDetectionStrategy` enumeracije kako bi se optimizirao proces detekcije promjena. Dio koda s definicijom `RepositoryCardComponent` komponente:

```
@Component({
  selector: 'gh-repository-card',
  template: `
    <h3>{{ repo.name }}</h3>
    <ul>
      <li>Language: {{ repo.language || 'None' }}</li>
      <li>Stars: {{ repo.stargazers_count }}</li>
      <li>Forks: {{ repo.forks }}</li>
    </ul>
  `,
  styleUrls: ['./repository-card.component.scss'],
  changeDetection: ChangeDetectionStrategy.OnPush,
})
export class RepositoryCardComponent {
  @Input() repo!: Repo;
}
```

Slika 6 – `RepositoryCardComponent` komponenta u *Angularu*

Pri radu s *Reactom*, komponente se mogu definirati na nekoliko načina. Za razvoj demo aplikacije korišten je noviji način definiranja komponenti gdje se komponente definiraju kao funkcije koje kao vrijednost vraćaju definiciju sučelja – najčešće koristeći *JSX* jezik. Moguće je koristiti i stariji način definiranja komponenti koji je sličan onome u *Angularu*, gdje se komponente definiraju uz pomoć *JavaScript* klasa, a metode životnog vijeka kao metode te klase. U novijem načinu, funkcionalnost metoda životnog vijeka se postiže korištenjem tzv. kuka (engl. *hooks*) koje su funkcije koje *React* poziva u određenim trenucima prilikom renderiranja – slično kao i s metodama životnog vijeka. Jedna od najvećih razlika između kuka i metoda životnog vijeka je način grupiranja funkcionalnosti. Kod metoda životnog vijeka, kod se grupira ovisno o vremenu kad se koji dio koda treba izvršiti. Tako bi se funkcionalnost, npr. prisluškivanja događaja, razdvojila kroz nekoliko metoda životnog vijeka – pokretanje prisluškivanja pozivanjem metode `addEventListener` u metodi `componentDidMount`, a prestanak prisluškivanja u metodi `componentWillUnmount`. Kod kuka se kod grupira ovisno o funkcionalnosti te se sinkronizira s životnim vijekom komponente. Tako bi se isti primjer prisluškivanja događaja definirao unutar jedne kuke, umjesto kroz nekoliko metoda životnog vijeka.

Za pohranu stanja koristi se kuka `useState` čija je povratna vrijednost uređen par – trenutna vrijednost stanja i funkcija za promjenu stanja. Slično kao i kod *Angulara,* instanca servisa `UserService` se dohvaća za obavljanje mrežnih zahtjeva

koristeći sličan mehanizam za dijeljenje vrijednosti između različitih podstabala u hijerarhiji komponenata - `Context`. Samo pozivanje metode servisa koje dohvaća informacije i korisnicima i njihovim repozitorijima obavlja se unutar `useEffect` kuke. Kuki su kao dodatni parametri proslijeđene vrijednosti korisničkog imena – `username` i samog servisa – `userService` s čijim promjenama zatim *React* sinkronizira izvršavanje kuke – u ovom slučaju, ako se promjeni vrijednost `username` varijable, ponovno će se dohvatiti informacije o novom korisniku s tim korisničkim imenom. Pozivanjem funkcija `setLoading`, `setUser` i `setRepos` u `useEffect` kuki, pokreće se novi ciklus detekcije promjena te će se osvježiti sve komponente koje primaju navedene vrijednosti kao parametre. Također, osvježit će se i sama komponenta `UserDetails` koja pohranjuje vrijednosti u stanju.

Dio koda komponente `UserDetails` čija je svrha dohvaćanje informacije korisnika:

```
export default function UserDetails({ username }) {
  const [loading, setLoading] = useState(true);
  const [repos, setRepos] = useState([]);
  const [user, setUser] = useState(null);

  const { userService } = useContext(ServicesContext);

  useEffect(() => {
    async function getData() {
      const [user, repos] = await Promise.all([
        userService.getUser(username),
        userService.getUserRepos(username),
      ]);

      if (user) {
        setLoading(false);
        setUser(user);
        setRepos(repos);
      }
    }

    getData();
  }, [username, userService]);
```

Slika 7 – `UserDetails` komponenta u *Reactu*

Komponenta `UserDetails` zatim prosljeđuje informacije o pojedinom repozitoriju komponenti `RepositoryCard` koja te informacije prima kao ulazni podatak putem *propsa*[71], koji su u slučaju komponenti definiranih kao funkcije parametri te funkcije. Kod komponente `RepositoryCard`:

```
export function RepositoryCard({ repo }) {
  return (
    <div className="repository-card">
      <h3>{repo.name}</h3>
      <ul>
        <li>Language: {repo.language || "None"}</li>
        <li>Stars: {repo.stargazers_count}</li>
        <li>Forks: {repo.forks}</li>
      </ul>
    </div>
  );
}
```

Slika 8 – `RepositoryCard` komponenta u *Reactu*

Pri radu s *Vue.jsom*, preporučeni način definiranja komponenti jest kao komponente jedne datoteke (engl. *Single File Components*) gdje se definicija sučelja nalazi između oznaka `<template>`, funkcije i svojstva koje pripadaju komponenti definiraju se kao svojstva objekta unutar `<script>` oznaka, a stilovi se definiraju unutar `<style>` oznaka. *Vue.js* također ima nekoliko metoda životnog vijeka koje se mogu koristiti i izvan samih komponenti kroz aplikacijsko programsko sučelje kompozicije (engl. *Composition API*). *Vue.js* koristi reaktivne koncepte slične *RxJSu* kako bi se automatski izvršio kod koji ovisi o nekom drugom kodu – konkretno u slučaju *Vue.jsa* je to pomoću funkcija `computed` i `watch` čiji se kod izvršava kad se vrijednost neke od reaktivnih varijabla koje se koriste unutar samih funkcija promijeni. `Computed` se koristi kako bi se izračunale vrijednosti, a `watch` najčešće kako bi se pokrenule asinkrone operacije, npr. mrežni pozivi. *Vue.js* također pruža mehanizam sličan injektoru ovisnosti u *Angularu* i `Contextu` u *Reactu* putem `inject` svojstva komponente. Dio koda komponente `UserDetails` čija je svrha dohvaćanje informacije korisnika:

[71] *Components and Props – React*. (bez dat.). Preuzeto 28. ožujak 2021., od https://reactjs.org/docs/components-and-props.html

```
<script>
import RepositoryCard from "./components/RepositoryCard";

export default {
  components: {
    RepositoryCard,
  },
  inject: {
    userService: "userService",
  },
  data() {
    return {
      loading: true,
      user: null,
      repos: [],
    };
  },
  async mounted() {
    const [user, repos] = await Promise.all([
      this.userService.getUser(),
      this.userService.getUserRepos(),
    ]);

    if (user) {
      this.loading = false;
      this.user = user;
      this.repos = repos;
    }
  },
};
</script>
```

Slika 9 – `UserDetails` komponenta u *Vue.jsu*

`UserDetails` komponenta zatim podatke o pojedinom repozitoriju prosljeđuje komponenti `RepositoryCard`, kao i u *React* i *Angular* verziji aplikacije. Komponenta `RepositoryCard` ulazne parametre prima definiranjem objekta `props` (istog imena i semantičkog značenja kao i `props` koncept u *Reactu*) te definiranjem svojstva tog objekta gdje je vrijednost svojstva tip ulaznog podatka – u ovom slučaju svojstvo `repo` s vrijednošću `Object`. Kod komponente `RepositoryCard`:

```
<template>
  <div class="repository-card">
    <h3>{{ repo.name }}</h3>
    <ul>
      <li>Language: {{ repo.language || "None" }}</li>
      <li>Stars: {{ repo.stargazers_count }}</li>
      <li>Forks: {{ repo.forks }}</li>
    </ul>
  </div>
</template>

<script>
export default {
  props: {
    repo: Object,
  },
};
</script>
```

Slika 10 – `RepositoryCard` komponenta u *Vue.jsu*

4.2.3. Definiranje korisničkog sučelja

Pri radu s *Angularom,* korisničko sučelje je definirano pomoću predložaka. Predlošci se najčešće definiraju u zasebnim `.html` datotekama te su zatim referencirani u glavnoj datoteci komponente. Predlošci podržavaju sintaksu sličnu podskupu *JavaScript* izraza unutar duplih vitičastih zagrada za interpolaciju vrijednosti izraza. Većina operacija u predlošcima se izvodi tzv. cijevima (engl. *pipe*) i direktivama. Cijevi služe za transformaciju podataka, dok se direktive koriste za upravljanje strukturom *DOM* elemenata (strukturalne) i njihovim vrijednostima (atributne) koji će se generirati prema predlošku.

Jedna od najkorištenijih cijevi jest `async` koja se koristi za automatsko osvježavanje sučelja kad se promijeni vrijednosti `Observablea` koji je proslijeđen kao parametar pružajući tako razvijateljima usku integraciju s *RxJs* knjižnicom i u predlošcima. `async` cijev je preporučeni način konzumiranja `Observable` vrijednosti u komponentama jer, osim što brine o pravovremenom osvježavanju sučelja, brine i o prestanku osluškivanja novih vrijednosti, tj. odjavljivanju (engl. *unsubscribe)* koje, ako se ne obavi, može prouzrokovati curenje memorije (engl. *memory leak*). Curenje memorije može postati problem u velikim aplikacijama gdje se koristi velik broj *Observable* vrijednosti jer će se funkcije koje su se pretplatile na *Observable* i dalje

pozivati te ih neće biti moguće dealocirati, tj. maknuti iz radne memorije. Sama async cijev u implementaciji koristi već navedeno `ChangeDetectorRef` sučelje i njegovu metodu `markForCheck` kako bi označila komponentu u kojoj se koristi za sljedeći ciklus detekcije promjena.

Neke od najkorištenijih zadanih strukturalnih direktiva su:

- `*ngFor` – direktiva koja se koristi za iteraciju i prikaz lista elemenata
- `*ngIf` – kondicionalna direktiva za prikazivanje ili skrivanje nekog elementa ovisno o proslijeđenoj vrijednosti

Neke od najkorištenijih zadanih atributnih direktiva su:

- `ngClass` – direktiva koja se koristi za kondicionalno postavljanje neke CSS klase nad elementom
- `ngStyle` – direktiva koja se koristi za direktno postavljanje stilova na željeni element u predlošku u obliku JavaScript objekta

Kod predloška komponente `UserDetailsComponent`:

```
<h1>GH User Repos</h1>

<h2 *ngIf="loading$ | async; else userDetails">Loading...</h2>

<ng-template #userDetails>
  <div *ngIf="userDetails$ | async as user" class="user">
    <h2>
      <img [src]="user.avatar_url" [alt]="user.login + ' avatar'" />
      {{ user.login }}
    </h2>
  </div>

  <ng-container *ngIf="repos$ | async as repos">
    <ul [@listAnimation]="repos.length">
      <li *ngFor="let repo of repos">
        <gh-repository-card [repo]="repo"></gh-repository-card>
      </li>
    </ul>
  </ng-container>
</ng-template>
```

Slika 11 – predložak `UserDetailsComponent` komponente u *Angularu*

Pri radu s *Reactom,* korisničko sučelje je najčešće definirano pomoću *JSX-a*. U usporedbi s predlošcima *Angularom* i *Vue.jsom*, *JSX* je vjerojatno najlakši za savladati zbog sintakse najsličnije *HTML-u* kojeg većina web razvijatelja već poznaje. Unutar vitičastih zagrada *JSX* podržava proizvoljne *JavaScript* izraze te zbog toga nije potrebna dodatna sintaksa za upravljanje strukturom i vrijednostima korisničkog sučelja. Dio koda komponente `UserDetails` koji sadrži *JSX* definiciju sučelja:

```
return (
  <div className="user-details">
    <h1>GH Demo - User Repos</h1>
    {loading && <h2>Loading...</h2>}
    {user && (
      <h2>
        <img src={user.avatar_url} alt={`${user.login} avatar`} />
        {user.login}
      </h2>
    )}
    <TransitionGroup className="repos" component="ul">
      {repos.map((repo, i) => (
        <CSSTransition
          key={repo.id}
          timeout={150 * i + 150}
          classNames="repo"
        >
          <li className="repo">
            <RepositoryCard repo={repo} />
          </li>
        </CSSTransition>
      ))}
    </TransitionGroup>
  </div>
);
```

Slika 12 – definicija sučelja `UserDetails` komponente u *Reactu*

Čest obrazac prilikom rada s *JSX*-om je korištenje kondicionalnog operatora `? :` i Booleovih operatora `&&` i `||` za prikazivanje ili skrivanje elemenata jer se vrijednosti `false` i `true`, `null` i `undefined` ne prikazuju. U komponenti `UserDetails` se na taj način kondicionalno prikazuju stanje učitavanja ili stanje kada su podaci o korisniku dostupni. U *JSX-u* se za iteraciju i prikaz liste elemenata često koristi zadana *JavaScript* funkcija višeg reda `map`. `map` je funkcija višeg reda jer kao parametar prima funkciju koju poziva za svaki element niza. U komponenti `UserDetails` se tako za

svaki korisnikov repozitorij poziva funkcija koja kao vrijednost vraća drugu komponentu, `RepositoryCard`, koja prikazuje informacije o tom repozitoriju.

Također, iako je *JSX* (osim mogućnosti interpoliranja *JavaScript* izraza) većinom sličan *HTML-u*, postoje neke razlike u čestim atributima na koje valja obratiti pozornost. Tako se u *JSX-u*, u odnosu na *HTML*, klase nad elementima postavljaju ključnom riječi `className`, a ne `class`, te se oznake povezuju s poljima pomoću ključne riječi `htmlFor`, a ne `for` kao što je slučaj kod *HTML-a*.

Rad s *Vue.jsom* je vrlo sličan radu s *Angularom* jer *Vue.js* koristi predloške, no također i *Reactu* jer *Vue.js* podržava i izražavanje korisničkog sučelja kao funkcije uz mogućnost postavki za korištenje *JSX* sintakse. Interpolacija vrijednosti se kao i u *Angularu* vrši stavljanjem izraza unutar duplih vitičastih zagrada.

Dio koda komponente `UserDetails` koji sadrži definiciju sučelja:

```
<template>
  <div class="user-details">
    <h1>GH User Repos</h1>
    <h2 v-if="loading">Loading...</h2>
    <h2 v-else-if="user">
      <img :src="user.avatar_url" :alt="user.login + ' avatar'" />
      {{ user.login }}
    </h2>

    <transition-group appear name="repo-list" tag="ul" class="repos">
      <li
        v-for="(repo, i) in repos"
        :key="repo.id"
        :style="{ '--i': i }"
        class="repo"
      >
        <RepositoryCard :repo="repo"></RepositoryCard>
      </li>
    </transition-group>
  </div>
</template>
```

Slika 13 – predložak `UserDetails` komponente u *Vue.jsu*

U isječku koda vidljive su neke od najčešće korištenih strukturalnih direktiva u *Vue.js* predlošcima:

- `v-for` za iteraciju i prikaz liste elemenata
- `v-if` i `v-else if` za kondicionalno prikazivanje elemenata
- `v-bind` direktiva za reaktivno vezivanje podataka sa sučeljem, tj. *binding.* Korištenjem `v-bind` direktive, sučelje se automatski osvježuje nakon što se promjeni vrijednost – u ovom slučaju `repo` varijable. Zbog čestog korištenja `v-bind` direktive u *Vue.js* aplikacijama, moguće je i korištenje kraće sintakse - dvotočke prije imena atributa čija je vrijednost vezana – npr. `:src` i `:alt` atributa <img> oznake

5. Usporedba razvojnih okvira

5.1. Opseg okvira

Jedna od najvećih razlika u razvoju aplikacija s navedena tri razvojna okvira jest koliko gotovih rješenja pružaju bez instaliranja dodatnih knjižnica i paketa. Dakako, uvijek postoji mogućnost vlastite implementacije potrebne funkcionalnosti, no zbog uštede vremena i kvalitete implementacije, preporuča se korištenje poznatijih knjižnica. *Angular* tako pruža najviše gotovih rješenja te za razvoj aplikacija s *Angularom* gotovo da i nije potrebno instalirati dodatne knjižnice. *React* se u potpunosti razlikuje od *Angulara* jer rješava samo probleme apstrakcije sučelja i detekcije promjena, dok je za funkcionalnosti poput stiliziranja, animacija, navigacije i efikasnijeg upravljanja formama i stanjem potrebno instalirati zasebne knjižnice od kojih ni jedna nije službeno podržana od strane *React* tima. *Vue.js* se nalazi između *Angulara* i *Reacta* te dolazi s minimalnom funkcionalnošću - kao i *React*, no također nudi korisnicima i službeno podržane knjižnice za navigaciju i upravljanje stanjem – *vue-router* i *vuex*[72]. Niti jedan od ovih pristupa nije idealan te svi imaju neke mane i prednosti. Neke od karakteristika razvojnog okvira većeg opsega poput *Angulara*:

- Ustaljene konvencije omogućuju lakše prilagođavanje novim projektima jer većina projekta ne odstupa u većoj mjeri od konvencija
- Bolja integracija službeno podržanih paketa s ostatkom aplikacije jer isti tim koji razvija razvojni okvir, razvija i dodatne pakete te je moguća bolja kolaboracija medu odgovornim razvijateljima
- Veći broj prijavljenih problema u implementaciji (u trenutku pisanja rada oko 2.4 tisuće prijavljenih problema na *Angular* GitHub repozitoriju[73]) zbog veće količine koda u razvojnom okviru

Neke od karakteristika okvira manjeg opsega poput *Reacta*:

- Nedostatak službeno podržanih knjižnica pozitivno utječe na broj neslužbenih knjižnica te pruža veći izbor za razvijatelje (npr. za *React* postoji nekoliko

[72] What is Vuex? | Vuex. (bez dat.). Preuzeto 25. siječanj 2021., od https://vuex.vuejs.org/

[73] Angular/angular. (bez dat.). GitHub. Preuzeto 25. siječanj 2021., od https://github.com/angular/angular

knjižnica za efikasnije upravljanje formama: *Formik*[74], *react-hook-form*[75] i *react-final-form*[76] od kojih se svaka preuzme preko 200 000 puta tjedno s registra *npm*) zbog potrebe zajednice korisnika razvojnog okvira za efikasnijim rješenjima

- S druge strane, nedostatak službeno podržanih knjižnica za rezultat može imati zastarjele knjižnice s kojih je teško migrirati na alternative zbog količine postojećeg koda gdje se koriste.
- Manji broj prijavljenih problema u implementaciji (u trenutku pisanja oko 500 prijavljenih problema na *React* GitHub repozitoriju[77]) zbog manje količine koda u razvojnom okviru ili knjižnici

Jedna od bitnijih stvari prilikom rada s nekim razvojnim okvirom, a na koju utječe i opseg okvira je i krivulja učenja. Iako je krivulja učenja stvar subjektivne prirode, pa je teško reći s kojim okvirom je lakše naučiti raditi, veličina površine aplikacijskog programskog sučelja nekog razvojnog okvira dobar je indikator strmosti krivulje učenja. Prema toj metrici, *Angular* ima najstrmiju krivulju učenja te bi se moglo reći da je najteži za naučiti, dok *React* ima najblažu krivulju učenja te bi bio najlakši. Međutim, pri razvoju većih i kompleksnijih aplikacija, funkcionalnosti koje pruža okvir s manjom površinom aplikacijskog programskog sučelja mogu biti neadekvatne za efikasan razvoj te je potrebno naučiti raditi s nekoliko dodatnih programskih knjižnica kako bi se postigao paritet s većim okvirima. Primjerice*, react-router je* knjižnica koja se u praksi vrlo često koristi pri razvoju s *Reactom*, a čija površina aplikacijskog programskog sučelja ne bi bila uključena u navedenu metriku. Mogli bismo stoga reći kako se krivulje učenja razvojnih okvira sve više izjednačavaju što je kompleksnija aplikacija koja se s njima razvija.

[74] Formik. (bez dat.). npm. Preuzeto 25. siječanj 2021., od https://www.npmjs.com/package/formik

[75] React-hook-form. (bez dat.). npm. Preuzeto 25. siječanj 2021., od https://www.npmjs.com/package/react-hook-form

[76] React-final-form. (bez dat.). npm. Preuzeto 25. siječanj 2021., od https://www.npmjs.com/package/react-final-form

[77] Facebook/react. (bez dat.). GitHub. Preuzeto 25. siječanj 2021., od https://github.com/facebook/react

Veličina aplikacijskog programskog sučelja utječe i na način razvoja funkcionalnosti u aplikaciji, čega su primjer tranzicije u aplikaciji *GH-Demo*. U sučelje aplikacije su dodane tranzicije kako bi se demonstrirala razlika između načina razvoja u *Reactu* i sličnim okvirima i *Angularu*. Kod *Reacta*, kako do nedavno nije bilo ostalih načina apstrakcije osim komponenti, kompleksnija funkcionalnost se postiže kompozicijom komponenti. Konkretno, u primjeru komponente `UserDetails`, ulazna tranzicija kartica s detaljima korisnikovih repozitorija postignuta je kompozicijom komponenti `TransitionGroup` i `CSSTransition` s domenskom komponentom `RepositoryCard`. Posljedica takvog ograničenja – malog broja apstrakcija koje okvir pruža, knjižnice su koje imaju slično sučelje kao i ostatak koda u aplikaciji, drugim riječima, generalno ih je lakše za naučiti zato što koriste slične koncepte i već poznate razvojne obrasce. U slučaju *Angulara*, za definiranje animacija i tranzicija koristi se zadana knjižnica *@angular/animations* koja pruža funkcije kao `trigger, animate, style` i `transition` koje u pozadini koriste *CSS* animacije kako bi animirale elemente sučelja. Na slici 14 prikazana je definicija ulazne animacije elemenata liste u *Angularu*. Animacije u *Angularu* su zaseban koncept te aplikacijskim programskim sučeljem odskaču od ostatka koda okvira što znači da su još jedan dodatan koncept koji je potrebno naučiti kako bi se savladao rad s okvirom.

```
export const listAnimation = trigger('listAnimation', [
  transition('* => *', [
    query(
      ':enter',
      [
        style({ transform: 'translateY(50px)', opacity: 0 }),
        stagger(150, [
          animate('150s', style({ transform: 'translateY(0px)', opacity: 1 })),
        ]),
      ],
      { optional: true }
    ),
  ]),
]);
```

Slika 14 – definiranje ulazne animacije elementa liste u *Angularu*

5.2. Veličina aplikacije

Kriterij pomoću kojeg možemo usporediti razvojne okvire je i veličina jednakih aplikacija razvijenih pomoću različitih okvira. Nakon razvijanja aplikacije *GH-demo* te kreiranja optimiziranih paketa (engl. *bundle*), dobivene su sljedeće veličine aplikacija:

	Inicijalna veličina paketa	Ukupna veličina paketa	Inicijalna veličina paketa s gzip kompresijom
React	165.24 kB	187.35 kB	56.25 kB
Vue	128.63 kB	132.11 kB	58.6 kB
Angular	386.81 kB	402.03 kB	117.11 kB

Tablica 1 – veličina optimiziranih paketa za svaki razvojni okvir

Optimizirani paketi su kreirani pomoću sljedećih naredbi:

- *Angular*: `ng build –prod –build-optimizer`
- *React*: `react-scripts build`
- *Vue*: `vue-cli-service build`

U tablici je također navedena i inicijalna veličina paketa s *gzip*[78] kompresijom jer je to najčešći način posluživanja komprimiranih web stranica danas[79].

Inicijalna veličina paketa dobivena je oduzimanjem veličine dinamički učitanih dijelova aplikacije (engl. c*hunk)* od ukupne veličine aplikacije. Dinamički učitanim dijelovima se smatraju oni koji se učitavaju tijekom rada aplikacije, a ne umetanjem u izvorni *HTML* dokument. U slučaju *GH-Demo* aplikacije su to dijelovi koji se prikazuju ovisno o trenutnoj lokaciji u navigaciji.

Veliku razliku izmedu *Angulara* i *Reacta i Vue.jsa* uzrokuju knjižnice koje dolaze s *Angularom* poput *RxJS* i *zone.js*. Micanjem *zone.jsa,* inicijalna veličina paketa u *Angular* aplikaciji se smanjuje za 35.6 kB, no žrtvuje se lakoća razvijanja aplikacije jer se tada detekcija promjena mora izvršavati ručno u kodu. Iz tablice 1 vidljivo je kako je *Angular* verzija aplikacije znatno veća od *React i Vue.js* verzije. Jedan od uzroka tome je i količina samog koda koja je potrebna za definiciju komponenti i

[78] The gzip home page. (bez dat.). Preuzeto 25. siječanj 2021., od https://www.gzip.org/

[79] Compression in HTTP - HTTP | MDN. (bez dat.). Preuzeto 25. siječanj 2021., od https://developer.mozilla.org/en-US/docs/Web/HTTP/Compression

predložaka/definicija sučelja. Iz priloženih slika koda `UserDetails` komponenti (slike 4, 6 i 8), vidljivo je kako *Angular* verzija aplikacije zahtjeva najveću količinu koda, dok *React* i *Vue.js* zahtijevaju znatno manje te se tako te količine koda podudaraju i sa veličinama iz tablice 1.

5.3. Performanse

Kako u izrađenoj demo aplikaciji nema čestih promjena stanja koje bi dovele do česte detekcije promjena i renderiranja sučelja, da bi se dobio uvid u performanse razvojnih okvira može se iskoristiti web alat *JS framework benchmark* koji uspoređuje razvojne okvire u nekoliko kategorija. Jedna od zanimljivijih kategorija vezana za performanse jest brzina ažuriranja i renderiranja *DOM* elemenata, konkretno u slučaju ove usporedbe: umetanje, zamjena, brisanje, označavanje i dodavanje redaka tablice. Usporedbom navedena tri razvojna okvira, dobiveni su sljedeći rezultati:

Operacija	Vue 3.0.2	React 17.0.1	Angular 11.0.2
Kreiranje 1000 redaka	131.8 ms	155.8 ms	161.4 ms
Zamjena 1000 redaka	116.1 ms	128 ms	130.9 ms
Osvježavanje svakog 10. retka u tablici od 1000 redaka (16x usporenje procesora)	171.6 ms	172 ms	158.2 ms
Označavanje 1 od 1000 redaka (16x usporenje procesora)	161.9 ms	80 ms	78.2 ms
Zamjena 2 od 1000 redaka (4x usporenje procesora)	54.2 ms	414.3 ms	422.3 ms
Dodavanje 1000 redaka tablici od 10000 redaka (2x usporenje procesora)	256.6 ms	283.6 ms	299.3 ms
Brisanje 1 od 1000 redaka	23.2 ms	22.2 ms	25.4 ms

Brisanje svih (1000) redaka (8x usporenje procesora)	127.7 ms	136.5 ms	246.9 ms

Tablica 2 – rezultati *JS framework benchmarka*[80], preuzeto i prilagođeno s https://krausest.github.io/js-framework-benchmark/

Iz tablice rezultata je vidljivo da, osim u rijetkim slučajevima – kao npr. pri zamjeni redaka, među razvojnim okvirima nema većih razlika u performansama.

Stoga bismo mogli reći da pri razvoju većih aplikacija eventualni problemi s performansama vrlo vjerojatno nisu rezultat pogrešno odabranog razvojnog okvira, već uzroke valja tražiti u korisničkom dijelu izvornog koda. Također, razvojni okviri pružaju i mogućnosti izmjene uobičajenog procesa detekcije promjena ako je potrebno kako bi se poboljšale performanse određenih komponenti. Primjerice, u *Angularu* postoji funkcija `runOutsideAngular` čije pozivanje neće uzrokovati proces detekcije promjena, dok u *Reactu* postoji funkcija `useMemo` koja koristi već navedenu tehniku memoizacije kako bi se izbjegla ponovljena računanja istih vrijednosti i renderiranje neke komponente. Međutim, preporuka autora razvojnih okvira je usredotočiti se na razvoj aplikacije koristeći idiomatske razvojne obrasce svakog okvira kako bi izvorni kod aplikacije bio što razumljiviji, a tehnike optimizacije ostaviti tek za trenutak kad se pojave problemi s performansama.

[80] Za neke operacije dodano je simulirano usporenje procesora nekoliko puta koje je dostupno u Chrome pregledniku kako bi se dobilo prikladnije trajanje operacije. Mjerenja su izvršena automatiziranjem preglednika koristeći *webdriver* knjižnicu i *Selenium* (https://www.npmjs.com/package/selenium-webdriver) razvojni okvir za automatizaciju preglednika.

6. Kritike i budućnost razvojnih okvira

Iako razvojni okviri u klijentskom dijelu rješavaju mnoge navedene probleme te znatno olakšavaju razvoj kompleksnijih aplikacija, nekad i sami budu uzrok problema. Jedna od najčešćih kritika upućenih razvojnih okvirima jest njihova upitna potreba i dodatna kompleksnost koju dodaju tijekom razvoja. Većina razvojnih okvira je proizašla iz velikih tehnoloških kompanija koje su, rješavajući svoje probleme, generalizirano rješenje kasnije ponudile široj zajednici. Tako je primjerice *React* nastao u *Facebooku* te se koristi za izradu same *Facebook* aplikacije. Stoga se nekad dovodi u pitanje nužnost korištenja istog razvojnog okvira za izradu aplikacija koje nemaju iste zahtjeve te nisu kompleksne kao i aplikacije velikih tehnoloških kompanija.

Osim povećanja kompleksnosti u samom razvoju aplikacije, dodavanje razvojnih okvira u neki projekt često podrazumijeva i povećanje kompleksnosti u alatima koji se koriste za razvoj. Primjerice, za rad sa *Angularom* nije dovoljno samo poznavati njegovo aplikacijsko programsko sučelje, već i novi programski jezik – *TypeScript*, poprilično različitu paradigmu – funkcijsko-reaktivno programiranje sa *RxJS-om*, alate za pred procesiranje kod aplikacije poput *webpacka* te njihove razne opcije i slično. Iako razvojni okviri često pružaju alate komandne linije kako bi što više olakšali taj proces te sakrili implementacijske i konfiguracijske detalje, razni razvijatelji smatraju kako je količina tehnologija prevelika te dolazi do zasićenosti, popularno nazvano *JavaScript* umorom, (engl. *JavaScript* fatigue[81]) što je razumljivo, uzevši u obzir da je web kao platforma unazad nekoliko godina izgledao vrlo različito te je izrada stranica bila jednostavnija.

Jedan od protuargumenata kritikama o kompleksnosti jest da, osim olakšavanja samog procesa razvoja web aplikacije, rad s razvojnim okvirima olakšava i zapošljavanje web razvijatelja. Netko tko već poznaje neki popularan razvojni okvir, brže će postati produktivan te pridonijeti razvoju aplikacije u odnosu na nekoga tko, kako bi pridonio razvoju, mora prvo savladati neki prilagođen okvir koji se koristi isključivo u toj tvrtki. Takva ponuda i potražnja na tržištu zasigurno pridonosi rasprostranjenosti razvojnih okvira na webu.

[81] *JavaScript Fatigue*. (2020, kolovoz 6). CSS-Tricks. https://css-tricks.com/javascript-fatigue/

Osim kritika o kompleksnosti razvoja, jedna od kritika odnosi se na dodatnu količinu koda koji je potreban za izvršavanje aplikacija koje su razvijene koristeći razvojne okvire. Iako je metrika koja se rijetko ili uopće ne razmatra prilikom razvoja web aplikacija – izvršavanje dodatne količina koda povećava potrošnju električne energije. Također, na slabijim uređajima poput mobilnih telefona, korištenje zahtjevnih web aplikacija znatno povećava potrošnju baterije uređaja, a veća količina koda koja se šalje preko mreže troši veću količinu mobilnog podatkovnog prometa. Neki od novijih razvojnih okvira poput *Sveltea*[82] odgovor su na takve kritike te imaju za cilj isporučiti što manje koda krajnjim korisnicima. Konkretno, *Svelte* smanjuje veličinu aplikacije prethodnom kompilacijom aplikacijskog koda u običan *JavaScript* koji se može izvršiti bez zasebnog dijela koji pokreće aplikaciju (engl. *runtime environment*).

Iako su *React*, *Angular* i *Vue.js* već neko vrijeme najpopularniji razvojni okviri, čini se kako se svakodnevno pojavljuju novi. Ranije navedeni *JS framework benchmark* bilježi rezultate performansi za preko 100 različitih razvojnih okvira. Ranije navedeni *Svelte*, čija popularnost svakodnevno raste, predstavlja jedan od mogućih budućih trendova razvoja razvojnih okvira. Kompilacijom *JavaScripta* otvaraju se razne mogućnosti optimizacije poput povećanja brzine izvođenja ili smanjenja veličine same aplikacije. Također, neki od razvojnih okvira idu i korak dalje te koriste nove jezike posebno prilagođene za razvoj korisničkih sučelja. Kako je web prvotno bio zamišljen kao tehnologija za razmjenu znanstvenih dokumenata, a danas se koristi kao platforma za razvoj aplikacija, neke od ideja koje su se tada činile korisnima, danas više nisu u upotrebi. Iako je normalno za očekivati da će se neka platforma s godinama razvijati te će neke tehnologije postati zastarjele, a zamijenit će ih nove i prikladnije tehnologije, u slučaju weba je ta promjena bila drastična. Također, za razliku od nekih drugih platformi za razvoj aplikacija, količina web stranica koja postoji danas onemogućuje micanje nekih zastarjelih aplikacijskih programskih sučelja jer bi u protivnom one prestale raditi. Stoga se može pretpostaviti da bi glavne web tehnologije – *HTML*, *CSS* i *JavaScript* danas postojale u nekom poprilično drugačijem obliku ako kompatibilnost sa starijim tehnologijama ne bi predstavljala problem i ako bi se platforma od početka razvijala s podrškom za razvoj aplikacija, a ne samo

[82] *Svelte • Cybernetically enhanced web apps*. (bez dat.). Preuzeto 06. travanj 2021., od https://svelte.dev/

uređivanje dokumenata. Neki od primjera takvog pristupa su *marko*[83] i *imba*[84], dva razvojna okvira koja koriste programske jezike koji se prije izvršavanja u pregledniku kompiliraju u *JavaScript* te čija je ideja da budu što prikladniji za razvoj specifično korisničkih sučelja.

Također, jedna od zanimljivih tehnologija u budućnosti bi mogao biti i *WebAssembly* – tehnologija koja u novijim web preglednicima omogućava kompilaciju različitih programskih jezika u binarni format koji oni tada mogu izvršavati. *JavaScript* tako prestaje biti jedini programski jezik koji je moguće izvršavati u web pregledniku te se otvaraju mogućnosti za druge programske jezike. Jedan od takvih razvojnih okvira je *Microsoftov Blazor WebAssembly*[85] koji koristi *C#* programski jezik za razvoj klijentskog dijela aplikacije.

Bilo da se nastave razvijati unutar okvira i ograničenja trenutnih web tehnologija ili pak razvijati na potpuno novim temeljima, sigurno je da će se razvojni okviri u klijentskom dijelu web aplikacija i dalje razvijati i unaprjeđivati velikom brzinom kao i do sad.

[83] *Marko*. (bez dat.). Preuzeto 07. travanj 2021., od https://markojs.com/
[84] *Imba—The friendly full-stack language*. (bez dat.). Preuzeto 07. travanj 2021., od https://imba.io
[85] guardrex. (bez dat.). *Introduction to ASP.NET Core Blazor*. Preuzeto 07. travanj 2021., od https://docs.microsoft.com/en-us/aspnet/core/blazor/

7. Zaključak

Izrada kompleksnijih web aplikacija danas podrazumijeva korištenje nekog razvojnog okvira kako bi se olakšao njihov razvoj. Korištenjem razvojnih okvira povećava se brzina razvijanja i sigurnost te se olakšava rješavanje kompleksnih problema u izradi korisničkih sučelja. Iako postoji velik broj razvojnih okvira u klijentskom dijelu web aplikacija, njihovu srž cine dvije stvari – detekcija promjena i apstrakcija definicije sučelja. U ovom su radu obrađena tri trenutno najpopularnija razvojna okvira – *Angular, React i Vue.js* te je kroz izradu jednake aplikacije sa sva tri razvojna okvira napravljena njihova usporedba. Kroz usporedbu i priložene isječke koda vidljivo je da se sama struktura i raslojavanje aplikacije na komponente ne razlikuje uvelike između okvira. Najveću razliku čini ekosustav i dodatni česti problemi u razvoju koje razvojni okviri imaju za cilj riješiti. Tako je, primjerice, *Angular* razvojni okvir koji sa zadanim postavkama i paketima rješava najviše problema u razvoju, a *React* okvir koji rješava samo one najnužnije, a ostatak prepušta široj zajednici – razvijateljima paketa otvorenog koda. Svi razvojni okviri su dostatni za razvoj većine modernih web aplikacija te njihov odabir najviše ovisi o preferencijama i znanju projektnog tima – preferiraju li ustaljene konvencije i stabilnost ili pak žele slobodu izbora knjižnica za rješavanje određenih problema. Iako su razvojni okviri generalno korisni jer rješavaju mnoge probleme u razvoju kompleksnijih aplikacija, prije posezanja za nekim od popularnih razvojnih okvira valja razmisliti je li on uopće potreban za razvoj željene aplikacije i hoće li njegovo korištenje na kraju uštedjeti vrijeme ili će produljiti razvoj aplikacije, te ono najbitnije - hoće li korištenje razvojnog okvira poboljšati korisničko iskustvo.

8. Literatura

1. *Advanced Features: Automatic Static Optimization | Next.js*. (bez dat.). Preuzeto 21. ožujak 2021., od https://nextjs.org/docs/advanced-features/automatic-static-optimization
2. *Amazon.com: Online Shopping for Electronics, Apparel, Computers, Books, DVDs & more*. (2021, siječanj 30). https://www.amazon.com/
3. *Angular*. (2021, siječanj 10). https://angular.io/
4. *Angular Change Detection—How Does It Really Work?* (2016, lipanj 21). Angular University Blog. https://blog.angular-university.io/how-does-angular-2-change-detection-really-work/
5. *Angular CLI*. (2021, siječanj 24). https://cli.angular.io/
6. *Angular/angular*. (2021, siječanj 25). GitHub. https://github.com/angular/angular
7. *Angular—ApplicationRef*. (2021, siječanj 11). https://angular.io/api/core/ApplicationRef#tick
8. *AngularJS — Superheroic JavaScript MVW Framework*. (bez dat.). Preuzeto 07. travanj 2021., od https://angularjs.org/
9. *Angular—NgZone*. (2021, siječanj 31). https://angular.io/api/core/NgZone#runOutsideAngular
10. *Angular—TestBed*. (bez dat.). Preuzeto 05. travanj 2021., od https://angular.io/api/core/testing/TestBed
11. ardalis. (bez dat.). *Overview of ASP.NET Core MVC*. Preuzeto 07. travanj 2021., od https://docs.microsoft.com/en-us/aspnet/core/mvc/overview
12. *Babel · The compiler for next generation JavaScript*. (2021, siječanj 24). https://babeljs.io/
13. *Backbone.js*. (bez dat.). Preuzeto 07. travanj 2021., od https://backbonejs.org/
14. Bellard, F. (2021, siječanj 17). *JSLinux*. https://bellard.org/jslinux/
15. *Build software better, together*. (2021, siječanj 24). GitHub. https://github.com
16. *Can I use... Support tables for HTML5, CSS3, etc*. (bez dat.). Preuzeto 07. travanj 2021., od https://caniuse.com/filesystem
17. *Can I use... Support tables for HTML5, CSS3, etc*. (2021a, siječanj 11). https://caniuse.com/webusb
18. *Can I use... Support tables for HTML5, CSS3, etc*. (2021b, siječanj 11). https://caniuse.com/web-app-manifest
19. *Change And Its Detection In JavaScript Frameworks*. (2021, siječanj 9). https://teropa.info/blog/2015/03/02/change-and-its-detection-in-javascript-frameworks.html

20. *Client Side Hydration | Vue SSR Guide*. (2021, siječanj 10). https://ssr.vuejs.org/guide/hydration.html
21. *CoffeeScript*. (bez dat.). Preuzeto 07. travanj 2021., od https://coffeescript.org/#introduction
22. *Components and Props – React*. (bez dat.). Preuzeto 28. ožujak 2021., od https://reactjs.org/docs/components-and-props.html
23. *Compression in HTTP - HTTP | MDN*. (2021, siječanj 25). https://developer.mozilla.org/en-US/docs/Web/HTTP/Compression
24. contributors, V. S., Vladimir Starkov and. (bez dat.). *BEM — Block Element Modifier*. Preuzeto 27. ožujak 2021., od http://getbem.com/
25. *Create React App*. (2021, siječanj 24). https://create-react-app.dev/
26. *Document Object Model (DOM)—Web APIs | MDN*. (bez dat.). Preuzeto 01. veljača 2021., od https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model
27. *Documentation—Decorators*. (bez dat.). Preuzeto 14. veljača 2021., od https://www.typescriptlang.org/docs/handbook/decorators.html
28. *ECMAScript 2015 Language Specification*. (2015). 566.
29. *Electronics, Cars, Fashion, Collectibles & More*. (2021, siječanj 30). EBay. https://www.ebay.com
30. *Ember.js—A framework for ambitious web developers*. (bez dat.). Preuzeto 07. travanj 2021., od https://emberjs.com/
31. *Facebook—Log In or Sign Up*. (bez dat.). Facebook. Preuzeto 01. veljača 2021., od https://www.facebook.com/
32. *Facebook/react*. (2021, siječanj 25). GitHub. https://github.com/facebook/react
33. *FAQ — WHATWG*. (bez dat.). Preuzeto 07. ožujak 2021., od https://whatwg.org/faq
34. *Figma: The collaborative interface design tool.* (2021, siječanj 17). Figma. https://www.figma.com/
35. *Flutter—Beautiful native apps in record time*. (2021, siječanj 11). https://flutter.dev/
36. *Flux | Flux*. (bez dat.). Preuzeto 14. veljača 2021., od https://facebook.github.io/flux/
37. *Formik*. (2021, siječanj 25). npm. https://www.npmjs.com/package/formik
38. *Functional Reactive Programming*. (bez dat.). Manning Publications. Preuzeto 01. veljača 2021., od https://www.manning.com/books/functional-reactive-programming
39. Garrett, J. J. (bez dat.). *Ajax: A New Approach to Web Applications*. 5.
40. *Gatsby*. (2021, siječanj 10). Gatsby. https://www.gatsbyjs.com/
41. *GitHub REST API - GitHub Docs*. (2021, siječanj 24). https://docs.github.com/en/rest
42. *Glossary | Redux*. (bez dat.). Preuzeto 14. ožujak 2021., od https://redux.js.org/understanding/thinking-in-redux/glossary
43. *Google Maps*. (2021, siječanj 17). Google Maps. https://www.google.hr/maps

44. *Google/incremental-dom*. (2021). [TypeScript]. Google. https://github.com/google/incremental-dom (Original work published 2015)
45. guardrex. (bez dat.). *Introduction to ASP.NET Core Blazor*. Preuzeto 07. travanj 2021., od https://docs.microsoft.com/en-us/aspnet/core/blazor/
46. *Hooks API Reference – React*. (2021, siječanj 31). https://reactjs.org/docs/hooks-reference.html
47. *HTML Over The Wire | Hotwire*. (2021, siječanj 10). https://hotwire.dev/
48. *HTML Standard*. (2021, siječanj 4). https://html.spec.whatwg.org/#background
49. *Imba—The friendly full-stack language*. (bez dat.). Preuzeto 07. travanj 2021., od https://imba.io
50. *Inferno*. (bez dat.). Inferno.js. Preuzeto 28. ožujak 2021., od https://www.infernojs.org/
51. *Interactive Results*. (2021, siječanj 31). https://krausest.github.io/js-framework-benchmark/current.html
52. *Introducing Hooks – React*. (bez dat.). Preuzeto 14. veljača 2021., od https://reactjs.org/docs/hooks-intro.html
53. *Introducing JSX – React*. (bez dat.). Preuzeto 01. veljača 2021., od https://reactjs.org/docs/introducing-jsx.html
54. *JavaScript Fatigue*. (2020, kolovoz 6). CSS-Tricks. https://css-tricks.com/javascript-fatigue/
55. *Jquery*. (bez dat.). Npm. Preuzeto 07. travanj 2021., od https://www.npmjs.com/package/jquery
56. JSConf. (2019, srpanj 3). *Evan You on Vue.js: Seeking the Balance in Framework Design | JSConf.Asia 2019*. https://www.youtube.com/watch?v=ANtSWq-zI0s
57. js.foundation, J. F.-. (bez dat.). *JQuery*. Preuzeto 07. travanj 2021., od https://jquery.com/
58. *Marko*. (bez dat.). Preuzeto 07. travanj 2021., od https://markojs.com/
59. Max, K. (2021, siječanj 10). *What every front-end developer should know about change detection in Angular and React—Angular inDepth*. inDepthDev. https://indepth.dev/posts/1064/what-every-front-end-developer-should-know-about-change-detection-in-angular-and-react
60. *Memoization and Laziness*. (bez dat.). Preuzeto 01. veljača 2021., od https://www.cs.cmu.edu/~rwh/introsml/techniques/memoization.htm
61. *MVC - MDN Web Docs Glossary: Definitions of Web-related terms | MDN*. (bez dat.). Preuzeto 07. travanj 2021., od https://developer.mozilla.org/en-US/docs/Glossary/MVC
62. ng-conf. (2015, ožujak 6). *Change Detection Reinvented Victor Savkin*. https://www.youtube.com/watch?v=jvKGQSFQf10

63. NG-NL. (2016, ožujak 14). *NG-NL 2016: Pascal Precht - Angular 2 Change Detection Explained*. https://www.youtube.com/watch?v=CUxD91DWkGM
64. *Npm | build amazing things*. (2021, siječanj 30). https://www.npmjs.com/
65. *OWASP Top Ten Web Application Security Risks | OWASP*. (bez dat.). Preuzeto 05. travanj 2021., od https://owasp.org/www-project-top-ten/
66. *Preact*. (bez dat.). Preuzeto 28. ožujak 2021., od https://preactjs.com/
67. *Proxy—JavaScript | MDN*. (2021, siječanj 17). https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Proxy
68. *React – A JavaScript library for building user interfaces*. (2021, siječanj 10). https://reactjs.org/
69. *React Router: Declarative Routing for React*. (2021, siječanj 24). ReactRouterWebsite. https://reacttraining.com/react-router
70. *React Transition Group*. (2021, siječanj 24). https://reactcommunity.org/react-transition-group/
71. *React-final-form*. (2021, siječanj 25). npm. https://www.npmjs.com/package/react-final-form
72. *React-hook-form*. (2021, siječanj 25). npm. https://www.npmjs.com/package/react-hook-form
73. *Reactivity in Depth | Vue.js*. (2021, siječanj 17). https://v3.vuejs.org/guide/reactivity.html#what-is-reactivity
74. *Reconciliation – React*. (2021, siječanj 11). https://reactjs.org/docs/reconciliation.html
75. *Redux—A predictable state container for JavaScript apps. | Redux*. (bez dat.). Preuzeto 14. ožujak 2021., od https://redux.js.org/
76. *Reduxjs/redux-devtools*. (2021). [TypeScript]. Redux. https://github.com/reduxjs/redux-devtools (Original work published 2015)
77. *Rendering on the Web*. (2021, siječanj 17). Google Developers. https://developers.google.com/web/updates/2019/02/rendering-on-the-web
78. *RxJS - BLACK LIVES MATTER*. (2021, siječanj 25). https://rxjs-dev.firebaseapp.com/
79. Savkin, V. (2020, veljača 12). *Understanding Angular Ivy: Incremental DOM and Virtual DOM*. Medium. https://blog.nrwl.io/understanding-angular-ivy-incremental-dom-and-virtual-dom-243be844bf36
80. Shklar, L., & Rosen, R. (2009). *Web Application Architecture: Principles, Protocols and Practices* (2nd edition). Wiley.
81. *Spotify - Web Player: Music for everyone*. (2021, siječanj 30). Spotify. https://open.spotify.com/

82. *StatCounter Global Stats—Browser, OS, Search Engine including Mobile Usage Share*. (bez dat.). StatCounter Global Stats. Preuzeto 07. travanj 2021., od https://gs.statcounter.com/
83. *Svelte • Cybernetically enhanced web apps*. (bez dat.). Preuzeto 06. travanj 2021., od https://svelte.dev/
84. Tanenbaum, A. S., & Wetherall, D. J. (2012). *Computer Networks* (5th edition). Pearson.
85. *The Chromium Projects*. (bez dat.). Preuzeto 07. travanj 2021., od https://www.chromium.org/
86. *The File System Access API: Simplifying access to local files*. (bez dat.). Web.Dev. Preuzeto 07. travanj 2021., od https://web.dev/file-system-access/
87. *The gzip home page*. (2021, siječanj 25). https://www.gzip.org/
88. *The Official Pokémon Website | Pokemon.com | Explore the World of Pokémon*. (2021, siječanj 17). https://www.pokemon.com/us/
89. *Time to Interactive*. (2021, siječanj 10). Web.Dev. https://web.dev/interactive/
90. *Twitter*. (2021, siječanj 11). App Store. https://apps.apple.com/us/app/twitter/id333903271
91. *Twitter. It's what's happening*. (2021, siječanj 10). Twitter. https://twitter.com/
92. *Twitter—Apps on Google Play*. (2021, siječanj 10). https://play.google.com/store/apps/details?id=com.twitter.android&hl=en_US&gl=US
93. *Typed JavaScript at Any Scale.* (2021, siječanj 24). https://www.typescriptlang.org/
94. *Understand the JavaScript SEO basics | Google Search Central*. (bez dat.). Google Developers. Preuzeto 01. veljača 2021., od https://developers.google.com/search/docs/guides/javascript-seo-basics
95. *Vue CLI*. (2021, siječanj 24). https://cli.vuejs.org/
96. *Vue Router*. (2021, siječanj 24). https://router.vuejs.org/
97. VueConf Toronto. (2019, prosinac 13). *Design Principles of Vue 3.0 by Evan You*. https://www.youtube.com/watch?v=WLpLYhnGqPA
98. *Vue.js*. (bez dat.). Preuzeto 01. veljača 2021., od https://vuejs.org/
99. *W3C and the WHATWG signed an agreement to collaborate on a single version of HTML and DOM | W3C News*. (bez dat.). Preuzeto 07. ožujak 2021., od https://www.w3.org/blog/news/archives/7753
100. *WebAssembly*. (2021, siječanj 11). https://webassembly.org/
101. *WebAssembly/interface-types*. (2021). [WebAssembly]. WebAssembly. https://github.com/WebAssembly/interface-types (Original work published 2017)
102. *Webpack*. (2021, siječanj 24). Webpack. https://webpack.js.org/
103. *WebUSB API*. (2021, siječanj 11). https://wicg.github.io/webusb/

104. *What are Progressive Web Apps?* (2021, siječanj 11). Web.Dev. https://web.dev/progressive-web-apps/

105. *What is a URL? - Learn web development | MDN*. (bez dat.). Preuzeto 04. travanj 2021., od https://developer.mozilla.org/en-US/docs/Learn/Common_questions/What_is_a_URL

106. *What is Vuex? | Vuex*. (2021, siječanj 25). https://vuex.vuejs.org/

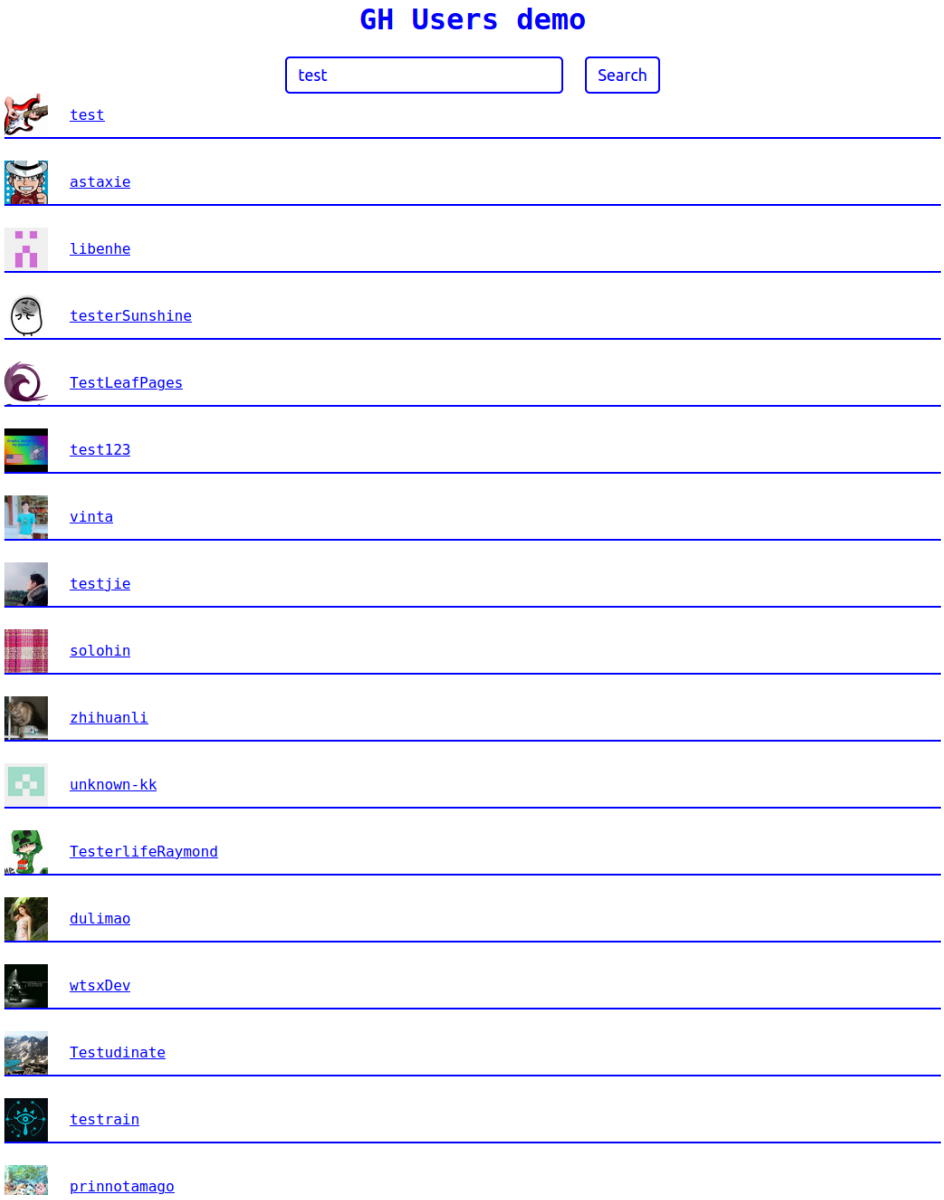
107. *WICG/webcomponents*. (2021). [HTML]. Web Incubator CG. https://github.com/WICG/webcomponents (Original work published 2013)

108. *XMLHttpRequest—Web APIs | MDN*. (2021, siječanj 30). https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest

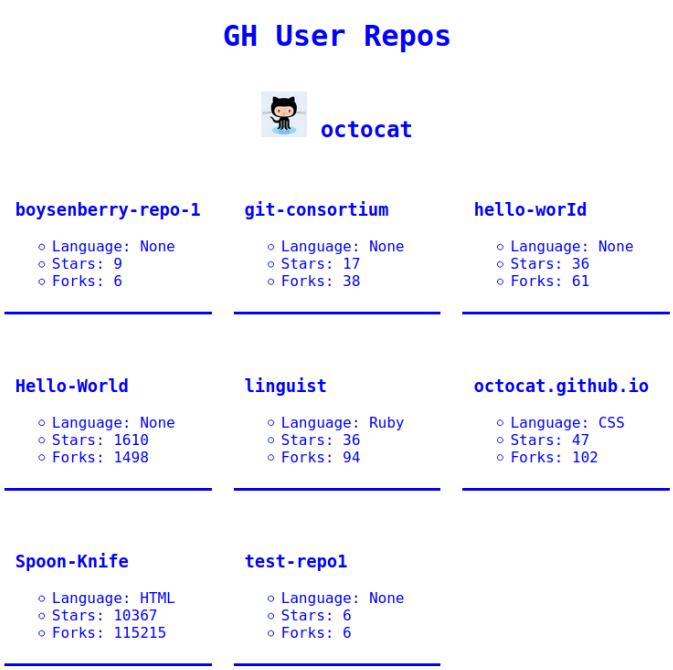
109. *You Might Not Need jQuery*. (bez dat.). Preuzeto 07. travanj 2021., od http://youmightnotneedjquery.com/

110. *Zone.js*. (2021, siječanj 11). npm. https://www.npmjs.com/package/zone.js

9. Prilozi



Slika 15 – Početni ekran aplikacije GH Demo



Slika 16 – Prikaz korisnikovih repozitorija

Razvojni okviri u klijentskom dijelu web aplikacije

Sažetak

U ovom radu analizirani su razvojni okviri u klijentskom dijelu aplikacije. Nakon uvoda gdje se ukratko prolazi kroz povijest i sadašnjost izrade web aplikacija, navedeni su glavni izazovi koji se pojavljuju pri izradi klijentskog dijela većih i kompleksnijih aplikacija te pregled kako su ti izazovi riješeni u trenutno tri najpoznatija razvojna okvira – *Reactu*, *Vue.jsu* i *Angularu*. Nakon teorijskog dijela, razvijena je jednaka aplikacija sa sva tri razvojna okvira s ciljem uvida u proces razvoja aplikacija s razvojnim okvirima i njihove usporedbe. Također, navedene su i neke kritike upućene razvojnim okvirima te kratak osvrt na budućnost razvojnih okvira.

Ključne riječi: web aplikacije, razvojni okviri, *React*, *Vue.js*, *Angular*, *SPA*

Client-side web application frameworks

Summary

This paper analyzes client-side web application development frameworks. After a brief introduction and a look at the history and the current state of web application development, the paper outlines the main challenges when developing big and complex web applications and how are those challenges solved in three currently most popular frameworks – React, Vue.js and Angular. The paper also goes through the development of three equal web applications, each developed with one of the mentioned frameworks in order to have a better insight in the process of application development with a framework and to compare them. Also, some critiques of the frameworks are mentioned, as well as a brief discussion of their possible future.

Key words: web applications, frameworks, *React*, *Vue.js*, *Angular*, *SPA*