JavaScript i moderni razvoj aplikacija

Autori:
mag. inf. Raul Sušanj Samolov
izv. prof. dr. sc. Marina Ivašić-Kos

Sažetak

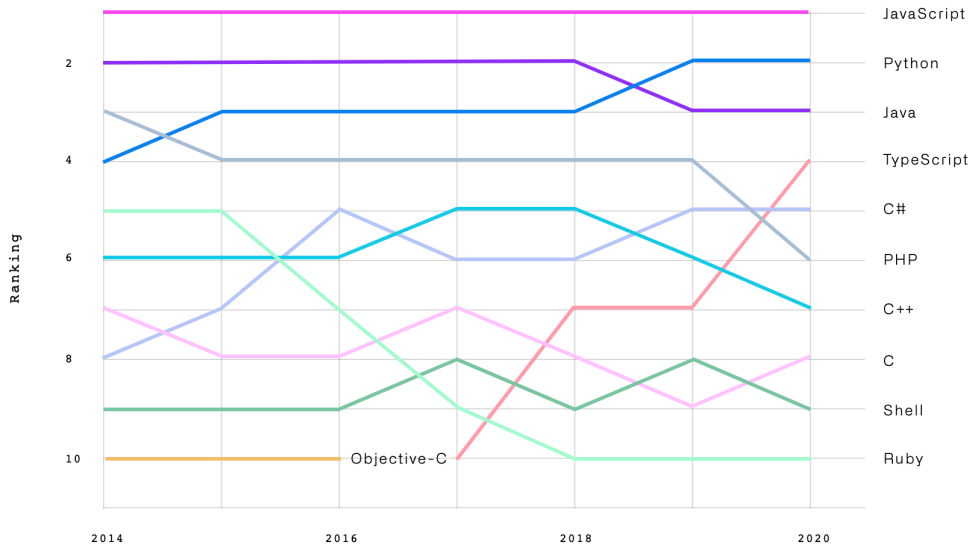
U radu je detaljan opis razvoja modernih aplikacija koje su napisane u JavaScript programskom jeziku i imaju za cilj da se mogu pokretati na raznovrsnim uređajima i platformama. Rad se bavi opisivanjem tehničkog i ekonomskog dijela razvoja aplikacija s detaljnim opisivanjem načina rada JavaScripta s naglaskom na njegove prednosti i mane te pomoćne alate koji su kompatibilni za korištenje. U zaključku se daje osvrt na trenutnu poziciju JavaScripta u IT zajednici i predviđanjima za budućnost primjene

Abstract

The paper describes in detail the development of modern applications written in the JavaScript programming language and aimed at running on a variety of devices and platforms that use the JavaScript programming language. The paper deals with the description of the technical and economic part of the application development with a detailed description of how JavaScript works, with emphasis on the advantages and disadvantages and the auxiliary tools that are compatible for its use. The conclusion gives an overview of the current position of JavaScript in the IT community and predictions for the future of the application.

Uvod

Razvoj modernih aplikacija kao što su web aplikacije je danas jedan od najprofitabilnijih poslova na svjetskom tržištu. Gotovo svaka tvrtka koja pruža neke servise ili proizvode posjeduje neku vrstu web aplikacije poput online trgovine ili mobilnu aplikaciju za program lojalnosti ili neku drugu svrhu. Razvoj web aplikacija ne zahtijeva skupu opremu ili sirovinu, već znanje programiranja, računalo s operacijskim sustavom i poslužitelj za podizanje sustava tako da trenutno najbrže posao dobivaju programeri sa znanjem JavaScript programskog jezika. JavaScript je posljednjih godina jedan od najpopularnijih programskih jezika sa velikim brojem korisnika, ne samo programera, već i tvrtki koje razvijaju aplikacije, Slika 1.



Slika 1. Najpopularniji jezici posljednjih godina na Githubu, https://octoverse.github.com/

JavaScript je omogućio razvoj na širokom spektru uređaja i platforma, a samim time je povećao vrijednost onih ljudi koji ga razumio i znaju koristiti. Tako se npr. JavaScript osim za razvoj web aplikacija može koristiti i za razvoj u mobilnih aplikacija, što je njegova velika prednost jer danas gotovo sve što želimo možemo odraditi pomoću pametnih telefona i aplikacija koje su instalirane na njima. Korištenjem mobilnih aplikacija danas nas do naručivanje hrane, odjeće i obuće, kozmetike, taksi službe, plaćanja računa i razgovora putem video poziva s prijateljima dijeli svega nekoliko dodira na ekranu.

Povijest JavaScripta “od nastanka do danas”

JavaScript (https://www.javascript.com/) je razvijen 1995. godine od strane programera Brendan Eicha koji je radio za Netscape Communications, tvrtku koja je tada imala monopol nad tržištem web preglednika. Eich je kreirao prvu verziju programskog jezika u svega desetak dana, a inspiraciju za strukturu i specifikaciju jezika dobio je od Java programskog jezika. Sredinom 1995. godine, Microsoft započinje preuzimanje tržišta web preglednika sa svojim web preglednikom pod nazivom Internet Explorer. Netscape Communications i Microsoft su tada postali glavni suparnici u utrci za najkvalitetnijim i najbržim web preglednikom pa je Netscape u nastojanju da spriječi sustizanje Microsoftovog Internet Explorer poduzeo slijedeće korake. Prvo su krenuli s procesom standardizacije programskog jezika kako Microsoft ne bi preuzeo kontrolu nad jezikom koji mu je tada bio interesantan kao već gotovo i provjereno rješenje za web preglednike. Standardizacija je nazvana ECMAScript te s koristi i danas u svrhu generalnog programskog standarda za razvoj jezika poput JavaScripta, a propisuje ga organizacija Ecma International (European Computer Manufacturers Association). Osim procesa standardizacije, Netscape ulazi u partnerstvo s tvrtkom Sun koja je od 1990. godine razvijala i promovirala Java programski jezik kao najbolji alat za softverska rješenja pametnih kućanskih aparata, a kasnije kao najbolje rješenje za web platformu, dijelila je zajednički interes s Netscapeom da spriječi Microsoftov monopol. Sun je pomoću partnerstva s Netscapeom imao benefit promocije svojeg programskog jezika u kombinaciji s još uvijek najpopularnijim web preglednikom, dok je Netscape imao benefit promocije JavaScripta kao službenog kompatibilnog programskog jezika s Java programskim jezikom, drugim riječima JavaScript je dobivao na popularnosti zbog postojećeg popularnog jezika. Iz istog razloga je JavaScript i preimenovan iz originalnog naziva Mocha u JavaScript.

Netscape je nudio i svoj proizvod pod nazivom LiveWire koji je omogućavao razvoj u istom programskom jeziku na klijentskoj i poslužiteljskoj strani sustava, funkcionalnost koju je i Sun razvijao s Javom. S obzirom na limitiranost tadašnjeg weba u usporedbi s kompleksnom Javom, Sun nije gledao na JavaScript kao stvarnu prijetnju te se partnerstvo između te dvije tvrtke i dalje nastavilo. Iako se počelo “čuti” za JavaScript i dalje je bio jedan od nepoželjnih jezika za programere. Kratko vrijeme inicijalnog razvoja jezika, slaba promocija, novi i prvi put viđen dizajn

programskog jezika i generalno slab interes za razvoj sustava za web preglednik, rangirali su JavaScript daleko niže od ostalih programskih jezika na ljestvici. Averzija i ismijavanje prema jeziku nije postojala samo od strane programera već i od njegovog začetnika i kreatora Brendan Eicha koji je priznao da mu je žao što se odlučio na neke elemente i dizajn programskog jezika.

Kroz godine JavaScript više nije bio u središtu pozornosti sve dok se nije pojavio sintaktički podskup samog jezika pod nazivom JSON (JavaScript Object Notation) kojeg je razvio Douglas Crockford te je samim time osvježio interes za JavaScriptom. Web je postajao sve veći i interesantniji populaciji, a samim time je bila sve veća potreba za razvojem web platforma tj. web stranica i programera koji će razvijati u JavaScriptu. U 2000-tim godinama popularnost i korištenje JavaScripta postaje sve veća, a jedan od svojih vrhunaca dostiže 2006. godine kada se pojavljuje jedna od najpopularnijih biblioteka za JavaScript, a to je jQuery (https://jquery.com/), biblioteka koja je pomogla velikom broju programera da razumije do tada slabo shvaćen i proučen programski jezik.

Potencijal ovog programskog jezika su primijetile i druge vodeće tvrtke te su pokušale razvijati svoje podskupe i supersetove jezika, međutim većina ih nije dospjela do većeg broja programera osim Typescript (https://www.typescriptlang.org/). Typescript je razvio Microsoft i postavio ga je kao softver otvorenog koda kako bi ga popularizirao i pridobio zainteresirane za supersetom JavaScripta koji se danas koristi u velikom broju mobilnih i web aplikacija, ali opet ne koliko i sam JavaScript. Najveći porast korištenja JavaScripta se pojavljuje 2009. godine kada je objavljena peta verzija ECMAScripta, a najveće izmjene i nadopune se događaju 2015. godine kada izlazi šesta verzija ECMAScripta poznatija kao ES6 (te pod novim nazivljem ECMAScript 2015). Kako se standard nije izmjenjivao gotovo šest godina ECMA international odlučuje u buduće objavljivati svake godine novu verziju ECMAScripta kako ne bi opet došlo do velikog broja izmjena i nadopuna koje otežavaju učenje postojećim, ali i novim JavaScript programerima.

Danas JavaScript pod standardom ECMAScript 2020 (ES11) stoji kao jedan od najpopularnijih i najkorištenijih programskih jezika u kojem su napisane brojne moderne web i mobilne aplikacije, te se svake godine nadograđuje s novim funkcionalnostima i unaprjeđuje dizajn [1].

Mogućnosti JavaScripta

JavaScript se godinama tretirao kao skriptni jezik namijenjen za razvoj klijentskih aplikacija za okruženje web preglednika, međutim kroz vrijeme se njegova primjena proširila i na razvoj poslužiteljskih aplikacija te na veći broj okruženja i uređaja.

Istina je da se i danas većinom JavaScript koristi za razvoj klijentskih aplikacija, ali se sve veći broj IT tvrtki i njihovi razvojni inženjeri odlučuju za JavaScript kao jezik za razvoj poslužiteljskih servisa poput API-a (Application Programming Interface), CRON poslova i sl. Nakon Netscapeovog programa Live Wire koji je pružao mogućnost pisanja JavaScripta na poslužiteljskoj strani softvera, 2009. godine se pojavljuje Node.js (https://nodejs.org/en/) tzv. JavaScript runtime koji omogućava isto, ali dobiva na sve većoj popularnosti te se danas tretira kao glavno okruženje za pisanje JavaScripta bez potrebe web preglednika. Nakon popularizacije i sve većem broju napisanih servisa u Node.js-u 2010. godine se pojavljuje i upravitelj paketima pod nazivom npm (https://www.npmjs.com/get-npm) koji je olakšao programerima objavljivanje i preuzimanje dijelova otvorenog koda za Node.js.
Danas npm broji preko 350.000 paketa otvorenog koda te olakšava razvoj klijentskih i poslužiteljskih aplikacija

Komponente i način rada Javascript-a

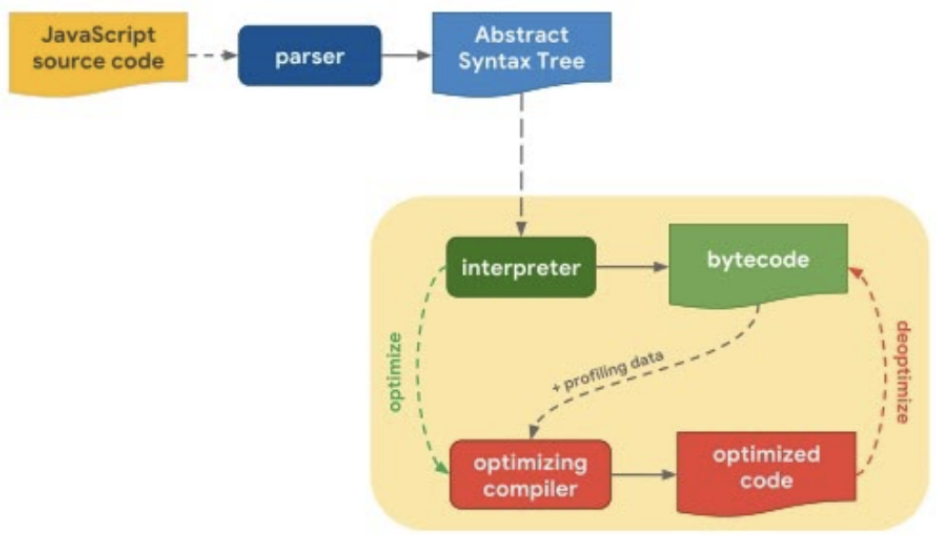
JavaScript se kroz godine značajno unaprijedio, unaprijedili su mu se koncepti i dizajni jezika. Kao i svaki programski jezik, JavaScript se sastoji od nekoliko glavnih komponenata i često ga se u IT zajednici tretira kao poprilično neobičan programski jezik jer ponekad dozvoljava kršenje nekih pravila što zna rezultirati s uspješnim kodom, ali zna i predstavi problem programerima. Kako bi se izbjegle greške i kako ne bi došlo do nelogičnosti potrebno je dobro poznavanje JavaScripta kao programskog jezika, programiranja općenito ali i znanje o funkcioniranju programskog jezika “ispod haube” tj. kako JavaScript procesuira kod koji pišemo u njemu kako bi postigli određenu funkcionalnost.

Javascript Engine (JavaScript pogon)

Prilikom kompajliranja JavaScript programskog jezika dolazi u uporabu tzv. JavaScript Engine koji je sastavni dio web preglednika, ali ga možemo pronaći i u prije navedenom runtime okruženju Node.js. Proces je sličan u oba slučaja pa ćemo ovdje dati primjer kako funkcionira JavaScript Engine Googleovog Chrome web preglednika pod nazivom V8 Engine jer se on koristi i kao JavaScript Engine u Node.js-u. Drugi web preglednici imaju svoje JavaScript Engine koji se minimalno razlikuju, neki od njih su npr. Spider Monkey za Firefox Mozillu, JavaScript Core za Safari, Chakra za Microsoft Edge.

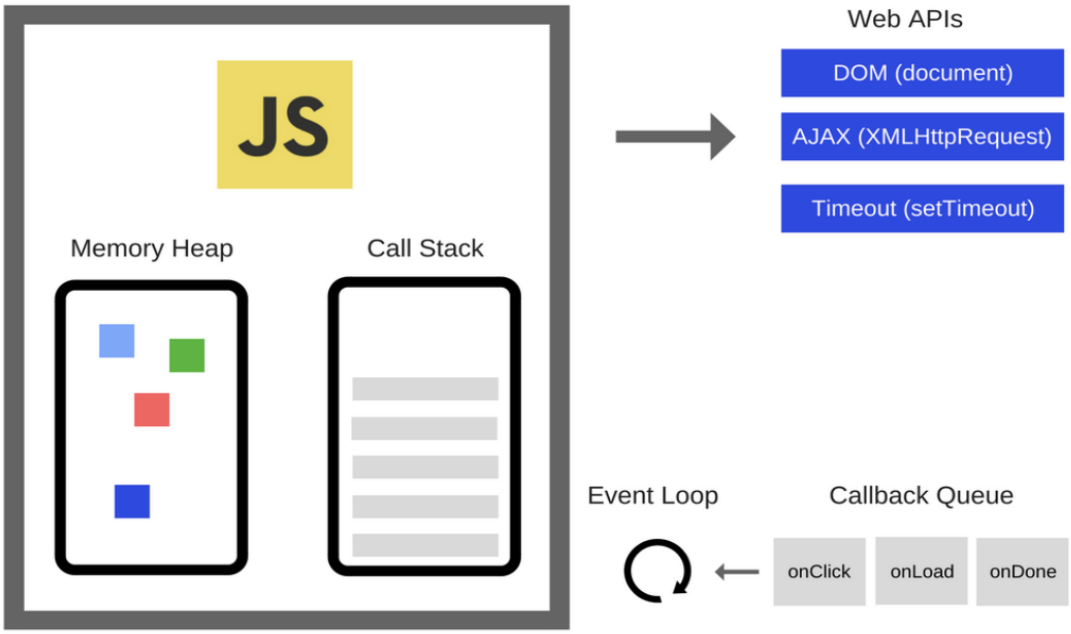
JavaScript Engine može čitati i pokretati kod, zato je prvi korak čitanje izvornog koda koji je programer napisao, taj dio procesa još nazivamo parsiranje. Prilikom parsiranja koda, JavaScript engine ga pretvara u Stablo Apstraktne Sintakse (Abstract Syntax Tree, AST) te ga prosljeđuje dalje u alatni lanac kompajlera (Slika 2). Neki od poznatijih kompajlera jesu Acron, Esprima i cheroot. Prvo kroz što se provodi AST, u daljnjem tekstu kod, je interpreter koji pretvara kod na dva sljedeća načina. Prvi način je da uzima jednostavne dijelove koda tj. one dijelove koji su optimizirani i one dijelove koda koji se ne ponavljaju više od jednog puta te ih direktno pretvara u binarni kod, a drugi način je da uzima složenije dijelove koda i tzv. vruće kodove tj. kodove koji su identični i primaju iste parametre (npr. funkcija koja se poziva na više mjesta), zatim ih optimizira pomoću TurboFan kompajlera (https://v8.dev/docs/turbofan) za optimizaciju uz pomoć mapiranih podataka prethodnog koda koji je već pretvoren prethodno u binarni kod te na kraju takav optimizirani kod pretvara u binarni kod.

Proces je vrlo jednostavan te ga se skraćeno može opisati bez dodatnih pojašnjenja oko pojedinih pojmova. Pojednostavljeno bi JavaScript Engine opisali kao alat koji prihvaća izvorni kod, zatim ga pretvara u stablo apstraktne sintakse i šalje interpreteru koji jednostavni kod pretvara u binarni kod, a složeniji i ponavljajući kod šalje najprije kompajleru za optimizaciju, a ovaj ga nakon optimizacije pretvori u binarni kod. Cijeli opisani proces se u V8 JavaScript enginu naziva JIT (justi-in-time compilation) [2].



Slika 2. Prikaz optimizacije izvornog koda (https://mathiasbynens.be/notes/prototypes)

Cjelokupan proces rada JavaScripta ne oslanja samo na JavaScript Engine, već su potrebni dodatni alati koje pružaju web preglednici kao što su hrpa memorije, stog poziva i slično, Slika 3.



Slika 3. JavaScript runtime (https://thecodest.co/blog/asynchronous-and-single-threaded-javascript-meet-the-event-loop/)

Memory Heap (Hrpa memorije)

Svaka varijabla, funkcija ili objekt koji definiramo mora se pohraniti negdje u radnu memoriju. Memory Heap je komponenta u web pregledniku koja je zadužena za alociranje memorije u JavaScript kodu te ona za programera odrađuje taj dio posla. Ono na što moramo paziti kada je riječ o memoriji, je to da ju ne preopteretimo Memory Heap, tj. da ne dođe do curenja memorije (engl. memory leak) jer je memorija uvijek limitirana. Prednost JavaScript jest ta da nam pruža ugrađeni kolektor smeća (garbage collector) koji je zadužen da oslobodi dijelove radne memorije od neiskorištenog koda, odnosno od onih varijabli, funkcija i objekata čija se referenca više ne koristi u programu [3].

Stog poziva (Call Stack)

Stog poziva je važna komponenta JavaScript obzirom da se radi o tzv. programskom jeziku s jednom niti (engl. one threaded programming language) koji izvršava liniju po liniju koda tako da se sljedeća linija koda se ne može pozvati niti izvršiti dok god prethodna linija nije izvršena. Takve programske jezike i samo programiranje još nazivamo sinkrono. Kako bi se osiguralo da se kod izvršava sinkrono, koristi se Stog poziva kao sastavni dio runtime memorije koji se sastoji od okvira za pozive. Svaki okvir za poziv se sastoji od lokalnih varijabli, argumenata i parametara te povratne adrese rezultata. Kada želimo izvršiti neku funkciju, Stog poziva postavlja okvir na vrh te ga nakon izvršenja izbacuje van. Stog poziva koristi metodu pozivanja FIFO, prvi unutra prvi van.

Ono što ne želimo da se dogodi je tzv. Stack overflow koji je u pojednostavljenim riječima događaj kada se Stog poziva napuni s pozivima preko svog limita što dovodi do rušenja aplikacije. Stack overflow možemo demonstrirati s neispravno napisanom rekurzivnom funkcijom koja nema uvjet prekida izvršavanja [2].

Web API

Kako bi se izbjeglo ograničenje JavaScripta da se naredbe ne mogu izvršavati istovremenu, uvedena je još komponenta koja ne pripada sastavno JavaScript programskom jeziku već web preglednicima, a to je Web API. Bez Web API-a većina aplikacija bi danas bila vrlo spora ili se neke funkcionalnosti ne bi mogle niti implementirati pa tako npr. ne bi mogli poslati korisniku

obavijest dok on pretražuje neku listu u aplikaciji koja dohvaća podatke s poslužitelja, već bi morali čekati da korisnik prekine pretraživanje ili bi ga prekinuli u akciji slanjem obavijesti i prikazom iste na sučelju aplikacije.

Web API sadrži većinu funkcionalnosti koje zahtijevaju asinkronu izvedbu te dok se JavaScript program izvršava Web API preuzima asinkroni dio programa. Neke od najkorištenijih funkcija Web API-a jesu fetch, setTimeout, setInterval i potrebna je implementacija istih od strane programera.

Cjelokupni tok neke asinkrone radnje se odvija tako da se definirane asinkrone funkcije pozivaju i obrađuju od strane Web API-a, tako da Web API-i izvršava funkciju, dok se ostatak funkcija koje su sinkrone dalje šalju u Red događanja (Event queue). Kada se funkcija iz Web API-a izvrši, tada šalje funkciju isto u Red događanja [2]. Jedan primjer bi bio s funkcijom setTimeout, koja prima dva parametra, prvi parametar je povratna funkcija, a drugi je vrijeme čekanja za poziv povratne funkcije u milisekundama. Kada engine primijeti da se radi o asinkronoj funkciji, automatski ju šalje Web API-u koji ju izvršava na zasebnoj niti i to u C++ programskom jeziku.

Red događanja, red poslova i petlja događanja

Red događanja (Event queue) je red u kojem čekaju funkcije za slanje u Stog poziva na izvršavanje. Od ECMAScript 2015 uvedena je dodatna komponenta tzv. red poslova (Job queue) koji sadrži JavaScript Promise sa resolverima i rejectorima koji imaju prednosti izvršenja nad ostatkom funkcija, što znači da će petlja događanja slati prvo funkcije iz reda poslova u Stog poziva, a onda tek iz reda događanja. Petlja događanja (Event loop) je petlja koja se beskonačno izvršava i provjerava red poslova i događanja te Stog poziva [2]. Petlju događanja možemo zamisliti kao petlju koja konstantno pita da li je Stog poziva prazan, da li ima kakva funkcija u redu poslova i događanja, te ako je stog prazan, šalje funkciju iz redova u stog. Nakon cijelog procesa se funkcije pozivaju u Stogu poziva.

Razvoj poslužiteljskih aplikacija

Pojavom Node.js-a i sve češćim korištenjem u razvoju modernih web aplikacija kao alternativa za poslužiteljske servise, povećao se i broj JavaScript programera zbog toga što se pružila mogućnost razvoja vještina programiranja u domeni klijentskog i poslužiteljskog razvoja.

Veliki broj ozbiljnih, produkcijskih poslužiteljskih servisa se danas razvija u Node.js-u iako po performansama nije najbolji izbor [4]. Pravi razlog korištenja JavaScripta na poslužiteljskoj strani je ušteda resursa i radne snage, jer dok je prije bilo obavezno imati jednog programera za klijentsku stranu i jednog programera za poslužiteljsku stranu, danas ta dva posla može odraditi jedan programer u JavaScriptu. Svakako je preporučljivo u industriji da se ipak zapošljava programere za svako područje zasebno, ali tada imamo drugu prednost JavaScripta u razvoju cjelokupnog sustava, a to je da oni programeri koji rade na klijentskoj strani aplikacije uvijek mogu pomoću programerima na poslužiteljskoj strani i obrnuto. Osim navedenog, postoji visoka razina kompatibilnosti poslužiteljske i klijentske strane te veliki broj dodatnih biblioteka i alata koji jednako dobro funkcioniraju.

Postoji nekoliko oblika poslužiteljskih servisa, ali onaj najčešći u razvoju web aplikacija je API (Application Programming Language). API služi za dohvaćanje, slanje i manipulaciju podataka te je glavna poveznica između baze podataka i klijentske strane aplikacije. API najčešće odrađuje funkcije kreiranja, čitanja, izmjene i brisanja zapisa iz ili u bazu podataka ali može sadržavati i dodatne funkcionalnosti ne vezane za bazu podataka [5].

Razvoj klijentskih aplikacija

Istina je da razvoj poslužiteljskih servisa zahtjeva veliko znanje poznavanja principa rada baze podataka, mrežnog sustava i protokola, kibernetičke sigurnosti i drugih složenijih koncepata i dugi niz godina se smatralo navedene vještine i razvoj poslužiteljskih servisa zahtjevnijim i ozbiljnijim dijelom razvoja aplikacija. Danas je situacija nešto drugačija i kompleksnost razvoja klijentskih aplikacija se gotovo pa izjednačio s razvojem poslužiteljskih servisa, moglo bi se ponekad zaključiti da je čak krivulja učenja teža u razvoju klijentskih aplikacija nego poslužiteljskih. Kompleksnost razvoja na klijentskim aplikacijama možemo zahvaliti novim i modernim bibliotekama i razvojnim okruženjima čija je svrha ubrzati i pojednostaviti razvoj. Iako smo naveli

kako je kompleksnost veća, ta kompleksnost rješava mnoge probleme s kojima su se programeri prije susretali, a kompleksnost je sve veća samo zato što se svakoga dana pronalaze nova i bolja rješenja, nove biblioteke i alati. JavaScript u ovom području dominira u usporedbi s drugim programskim jezicima i pruža najveću paletu alata za razvoj klijentskih aplikacija. Gotovo je moguće svaku aplikaciju danas napisati u JavaScriptu, radilo se o web, mobilnoj ili nekoj drugoj platformi. JavaScript se u razvoju klijentskih aplikacija toliko razvio da je u nekim oblicima čak zamijenio HTML i CSS te je moguće cijelu aplikaciju napisati samo u JavaScriptu [6]. Danas pomoću JavaScripta možemo kreirati HTML elemente i manipulirati s njima, a isto tako možemo dodavati stilove i stilske klase koje definiramo kao JavaScript objekte. Iako nam je potreban samo JavaScript prilikom razvoja, na kraju se ipak kod napisan u JavaScript translatira u dobri stari HTML i CSS, tako da je ključno i jako dobro poznavanje tih dviju tehnologija kako bi se mogle razvijati moderne i kvalitetne klijentske aplikacije.

React.js

U prošlosti je najpopularnija JavaScript biblioteka bila jQuery koja je omogućavala olakšanu manipulaciju nad DOM-om. DOM je stablasta struktura koja opisuje izgled i cjelokupnu strukturu aplikacija, a sastoji se od HTML elemenata. React.js biblioteka, (u daljnjem tekstu samo React), je preuzela ulogu jQueryja nakon 2013. godine kada je objavljen kao otvoreni kod, a do tada je bila korištena samo interno od strane Facebooka gdje je i razvijena. Razvio ju je programer Jordan Walke, po uzoru na XHP biblioteku za komponente u PHP programskom jeziku. Ciljevi Reacta su olakšati manipulaciju nad DOM-om i ubrzani razvoj korisničkog sučelja. Osnovni princip rada Reacta je tzv. *Lego princip* razvijanje korisničkog sučelja od komponenata (Components). Moguće je ugnježđivati već zadane komponente međusobno, te se tako dobivaju nove kompleksnije komponente. Zadane komponente primaju vrijednosti za zadana svojstva (props) te se putem njih proširuje funkcionalnost same komponente, a komponente koje smo sami razvili primaju samo svojstva koja definiramo. Princip prosljeđivanja podataka iz komponente u komponentu se odvija kroz svojstva, a privremeno memoriranje neke vrijednosti može se ostvariti drugim konceptom pod nazivom stanje (state). Stanje je poseban JavaScript objekt, specifičan za React u kojem se mogu pohranjivati vrijednosti u obliku ključ-vrijednost. Izmjene nad stanjem su moguće samo uz pomoć jedne funkcija setState(), koja za argument prima objekt sa sadržajem željenog ključa koji

želimo izmijeniti i pripadajuću novu vrijednost. Moguće je razvijati komponente sa stanjem kao što su komponenta razreda (Class component) i bez stanja, funkcionalne komponente.
Sredinom 2018. godine uvedene su dodatne metode u React koje su omogućile postavljanje stanja i u funkcionalne komponente, takve metode se nazivaju kuke (hooks). Jedan od dodatnih koncepata Reacta jesu metode životnog ciklusa (lifecycle methods) koje isto tako nisu bile dostupne u funkcionalnim komponentama do pojave kuka. Metode životnog ciklusa jesu metode koje se pokreću u određenom trenutku životnog ciklusa aplikacije te unutar njih možemo izvršiti neku željenu funkciju. Neke od njih se pokreću prije prikaza komponente, neke prilikom izmjene na komponenti, a neke kada se komponente više ne prikazuje. Kuka useEffect je uvela jedinstvenu metodu koja objedinjava sva tri scenarija, samo je potrebno definirati da li ta metoda ovisi o nekom parametru komponente u kojoj je definirana [8].

React Native

Ideja hibridnih aplikacija postoji već dugi niz godina i postojale su neke biblioteke i neka razvojna okruženja koja su omogućavala razvoj mobilnih aplikacija pisanjem izvornog koda u web tehnologijama, konkretno u JavaScriptu. Nažalost rijetko su se programeri usudili eksperimentirati s takvim alatima sve dok se 2015. godine nije pojavio React Native. Dodatna biblioteka koja radi u kombinaciji sa Reactom je zainteresirala širu JavaScript zajednicu i s obzirom na to da je React do tada poprimio poprilično dobru reputaciju nije postojalo toliko straha da se isproba nešto novo. Prednost koju je većina web programera vidjelo u React Nativu jest mogućnost razvoja mobilnih aplikacija u programskom jeziku kojeg već dobro poznaju. Osim navedene prednosti dodatna prednost je bila što se pisanjem jednog izvornog koda pokrivalo razvoj aplikacije za čak tri platforme Android, iOS, Windows, a danas broji i četvrtu Symphony OS. Implementacija izvornog koda u React Nativu je identična onoj u React.js-u s nekoliko razlika. Ključne razlike su da React Native ne koristi iste elemente kao React, npr. dok u Reactu koristimo div u React Nativu se koristi View, a za elemente poput p, h1, h2 i sl. za prikaz teksta se koristi komponenta Text. React Native ima prednost pred drugim sličnim bibliotekama, da može pristupiti matičnim modulima (native modules) poput kamere, bluetootha, nfc-a i cijelom skupu senzorike pametnih mobitela.
Osim razvoja mobilnih aplikacija, React Native je moguće koristiti za razvoj aplikacija za stolna i prijenosna računala, nosive uređaje poput pametnih satova i narukvica, pa čak i pametne televizije [9].

Testiranje

Jedan od najvrjednijih procesa prilikom razvoja modernih aplikacija je testiranje. Segment testiranja je često zanemaren tako da se samo površno testiraju aplikacija od strane programera, a ako neka tvrtka ima djelatnike samo za testiranje, QA inženjere, često se upotrebljava samo koncept E2E (end to end) testiranja. E2E testiranje je kada netko iz razvojnog tima testira aplikaciju tako da ju koristi kako bi je i krajnji korisnik koristio. Osim takve verzije E2E testiranja, moguće je napisati automatske E2E testove. Postoji nekoliko vrsta testiranja kao što su E2E, unit testovi, integracijski testovi i produkcijski testovi.

Kključni testovi su E2E i unit testovi, dok E2E simulira ponašanje krajnjeg korisnika, unit testovi testiraju tj. provjeravaju da li neka funkcija vraća ispravan rezultat u nekom određenom scenariju korištenja aplikacije. Preporuka je da se uz E2E testiranje s ljudskim faktorom implementiraju i automatizirani E2E i unit testovi. Osim što je preporuka pisanje testova, postoji i princip razvoja aplikacija vođen testovima, što znači da se prvo pišu testovi za potencijalnu funkciju, a zatim funkcija koja mora za svaki dati test vratiti željeni rezultat. Zbog kompleksnosti nekih testova i velikog broja linija koda koje je potrebno napisati, koriste se neka razvojna okruženja za testiranje poput MochaJS-a, Jesta, Jasmine, Karme i drugih [7].

Podizanje sustava

Pod sustav se smatra cjelokupna aplikacija koja se može sastojati od baze podataka, poslužiteljskih servisa i klijentskih aplikacija. Sustav možemo podignuti na više različitih okruženja, a podignuti sustav se ovisno o okruženju može razlikovati u verziji. Najčešće razvojne agencije podižu sustav na tri različita okruženja. Postoji testno okruženje koje se nalazi na udaljenom poslužitelju i dostupno je samo razvojnim programerima, zatim se postavlja *stagging* okruženje koje stoji na raspolaganju voditeljima projekta i klijentima i zadnje okruženje koje se podiže je produkcijsko. Prije podizanja sustava potrebno je podesiti okruženja prema kriterijima koje zahtjeva JavaScript aplikacija. Potrebno je prvo podignuti bazu podataka za koju je preporučljivo da se podiže na zaseban poslužitelj zbog zauzeća memorije i dodatne sigurnosti. Nakon uspješno podignute baze podataka, podižu se poslužiteljski servisi i klijentske aplikacije. Oboje se može nalaziti na istome poslužitelju, ali i ne mora. Kada je riječ o JavaScriptu najbolji izbor kreiranja softverskog

poslužitelja je *nginx* poslužitelj (https://www.nginx.com/) unutar čije se konfiguracije mora podesiti na kojim ulazima će se pokretati poslužiteljski servis, a na kojem klijentska aplikacija. Unutar nginx konfiguracije se definira i domena putem koje će se moći pristupiti aplikaciji i servisima. Izvorni kod sustava se može ručno prenijeti na poslužitelje ili putem git alata dohvatiti iz repozitorija, što je i preporučeni način. Nakon cijelog postupka potrebno je pokrenuti nginx poslužitelj, zatim poslužiteljske servise, a klijentsku aplikaciju se treba prvo "izgraditi" (build) i zatim pokrenuti uz pomoć pripadajućih skripti. Kako se gotovo uvijek mora s vremena na vrijeme ponovo pokrenuti poslužitelj zbog izmjena u izvornom kodu ili ako dođe do neke greške u sustavu ili samom poslužitelju, dobra ideja je postaviti alate za automatsko ponovno pokretanje sustava u slučaju greške i praćenje sustava. Jedan od najjednostavnijih u osnovnoj verziji besplatnih alata je PM2 (https://pm2.io/docs/runtime/guide/startup-hook/). Kada se radi o mobilnim klijentskim aplikacijama proces podizanja je drugačiji. Mobilne aplikacije je potrebno podignuti na trgovine za aplikacije za svaku platformu zasebno, tako imamo Google Play za Android platformu, App Store za iOS platformu i App Gallery za Symphony OS. Pomoću zadanih naredbi koje pokrećemo unutar projekta kreiramo uvez aplikacije (app bundle) koji je potrebno podignuti na navedene trgovine putem koje krajnji korisnici mogu preuzeti aplikaciju. U trgovinama aplikacija nije potrebno nikakvo dodatno konfiguriranje osim unosa osnovnih podataka o aplikaciji [4].

Dokumentacija razvoja i projekta

Ključan dio nakon svake završene faze razvoja, da li se radilo o krajnjoj fazi ili nekoj ranijoj je dokumentacija razvoja i projekta. Dokumentacija razvoja dokumentira tehničke aspekte sustava kao što su model baze podataka, model entiteta i veza, strukturu projekta, biblioteke koje se koriste, nejasne dijelove izvornog koda i sl., dok se dokumentacija projekta bavi dokumentiranjem sustava, najčešće klijentske aplikacije. Dokumentacija projekta služi kao uputstvo za krajnjeg korisnika i sadrži tko je sudjelovao na projektu, koliko je sati na kojem djelu sustava utrošeno, kako se koristi sustav i druge korisne informacije.

Zaključak

S JavaScript programskim jezikom danas možemo razvijati moderne aplikacije za gotovo sve uređaje iako je nastao od skriptnog jezika napisanog u svega 10 dana. Prednosti JavaScripta jesu jednostavna implementacija, brzo učenje jezika, pokretanje na velikom broju uređaja i platforma, velika raznovrsnih biblioteka i razvojnih okruženja i velika zajednica programera.

Danas kad spomenemo modernu web ili mobilnu aplikaciju, možemo s velikom sigurnošću reći da je barem klijentska aplikacija razvijena u JavaScript programskom jeziku. Najčešći oblik implementacije JavaScripta na klijentskim aplikacijama je razvojno okruženje React.js i React Native, dok poslužiteljske servise implementiramo pomoću Node.js-a.

Svaki programski jezik ima svoje uspone i padove u popularnosti, pa tako i JavaScript kojem popularnost trenutno još uvijek raste, ali se usprkos tome on se neprestano razvija, prilagođava novim tehnologijama i unaprjeđuje.

Literatura

1. Ben Aston, “A brief history of JavaScript”, https://medium.com/@_benaston/lesson-1a-the-history-of-javascript-8c1ce3bffb17 (pristupljeno 08.07.2020.).

2. Alexander Zlatkov , “How JavaScript works: an overview of the engine, the runtime, and the call stack”, https://blog.sessionstack.com/how-does-javascript-actually-work-part-1-b0bacc073cf (pristupljeno 11.08.2020).

3. Marijn Haverbeke, *Eloquent JavaScript, 3rd edition*, No Starch Press, San Francisco 2018.

4. Brad Traversy, “Node.js Deployment”, https://gist.github.com/bradtraversy/cd90d1ed3c462fe3bddd11bf8953a896 (pristupljeno 21.11.2020).

5. Adrian Senecki, “GraphQL vs REST – the battle of API designs”, https://tsh.io/blog/graphql-vs-rest/ (pristupljeno 21.08.2020).

6. Jon Ducket, *HTML & CSS, Design and Build Websites*, John Wiley & Sons, Inc. Indianapolis 2011.

7. Jash Unadkat, “Top 5 JavaScript Testing Frameworsk”, https://www.browserstack.com/guide/top-javascript-testing-frameworks (pristupljeno 12.11.2020).

8. Alex Banks, Eve Porcello, *Learning React, Functional web development with React and Redux*, O’Reilly Media, Inc., Sebastopol 2017.

9. Bonnie Eisenman, *Learning React Native, Building native mobile apps with JavaScript*, O’Reilly Media, Inc., Sebastopol 2016.