Komparativna analiza prednosti i nedostataka javascript radnih okolina

Potlimbrzović, Domagoj

Undergraduate thesis / Završni rad

2014

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **Josip Juraj Strossmayer University of Osijek, Faculty of Humanities and Social Sciences / Sveučilište Josipa Jurja Strossmayera u Osijeku, Filozofski fakultet**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:142:979796

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-03-28**

FILOZOFSKI FAKULTET
SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU

Repository / Repozitorij:

FFOS-repository - Repository of the Faculty of Humanities and Social Sciences Osijek

zir.nsk.hr

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU

dabar
DIGITALNI AKADEMSKI ARHIVI I REPOZITORIJI

Sveučilište Josipa Jurja Strossmayera u Osijeku

Filozofski fakultet

Preddiplomski studij Informatologije

Domagoj Potlimbrzović

Komparativna analiza prednosti i nedostataka JavaScript radnih okolina

Završni rad

Mentor: doc. dr. sc. Boris Badurina

Sumentor: Tomislav Jakopec, asistent

Osijek, 2014.

SADRŽAJ

1. UVOD 1

2. HYPERTEXT MARKUP LANGUAGE (HTML) 2

2.1. KRATKI PREGLED POVIJESTI HTML-A 2

2.2. STATIČNOST HTML-A 2

3. JAVASCRIPT 4

4. JAVASCRIPT RADNE OKOLINE 5

4.1. ANGULAR 5

4.2. BACKBONE 6

4.3. EMBER 6

4.4. PROTOTYPE 8

4.5. MOOTOOLS 9

4.6. NODE.JS 10

5. STRUKTURA KOMPARATIVNE ANALIZE 12

5.1. KOMPARATIVNA ANALIZA 13

6. ZAKLJUČAK 18

LITERATURA 19

SAŽETAK

Ovaj rad donosi kratak prikaz popularnih radnih okolina Angular, Backbone, Ember, Prototype, MooTools i Node.js. Drugi dio rada bavit će se komparativnom analizom tri najpopularnije i najzanimljivije spomenute radne okoline: Angular, Backbone i Ember. Komparativna analiza temeljit će se na sljedećim kriterijima: dizajn službenog mrežnog mjesta, jednostavnost snalaženja po mrežnom mjestu, savladavanje radne okoline, složenost implementacije u HTML, popularnost i budućnost te će se nakon tabličnog prikaza ukratko sažeti prednosti i nedostaci pojedinih radnih okolina u odnosu na ostale u obliku zaključka.

S obzirom da HTML, o kojem će također biti ponešto riječi na samom početku rada, nije programski jezik, niti se s njim mogu izrađivati dinamična mrežna mjesta, trebalo je pronaći način koji će učiniti statične mrežne stranice dinamičnima te koje će dati mogućnost korisnicama za interakciju s tim istim stranicama. Danas je interakcija korisnika s mrežnim stranicama gotovo nezaobilazna komponenta na Internetu, stoga ne čudi da je i JavaScript postala nezaobilazna komponenta Interneta. U tom smislu, radne okoline JavaScripta sve su češće ne samo izbor za one naprednije, već i nužnost za sve koji se misle baviti bilo kakvom vrstom programiranja. Ipak, nisu sve radne okoline niti jednako zahtjevne, niti funkcioniraju na isti način, stoga će, nakon kratkog predstavljanja samih radnih okolina, fokus rada biti na uspoređivanju odabranih kriterija triju izabranih radnih okolina. Tada će se na samom kraju rada napraviti sinteza svih prednosti i nedostataka pojedinih radnih okolina, kako bi se dobio sistematičan dojam i pregled o tome koja je radna okolina pogodnija za što, kako im izgledaju mrežne stranice, jesu li funkcionalne i lake za navigaciju, koliko su popularne te kakva im se predviđa budućnost.

Ključne riječi: JavaScript, Angular, Backbone, Ember, Prototype, MooTools, Node.js

1. UVOD

Cilj je ovog rada dati kratak prikaz šest popularnih JavaScript radnih okolina te analizirati njihove karakteristike i značajke kako bi se mogao dati prikaz o tome koja je radna okolina u čemu bolja, a koja je u čemu lošija.

Prije samih radnih okolina, u svrhu stvaranja konteksta za ovaj rad, bit će ukratko predstavljen *HyperText Markup Language*, tj. HTML. Osim osnovnih karakteristika, vrlo kratke povijesti i značajki HTML-a, s par rečenica dotaknut će se problematika statičnosti HTML-a za koju se smatra da je izravan uzrok nastanka radnih okolina. U biti, može se reći da su radne okoline, barem one opisane u ovom radu, ustvari izravna posljedica statičnosti HTML-a, odnosno odgovor na taj problem.

Nadalje, rad nastavlja s opisom JavaScripta – najpopularnijeg skriptnog jezika na Internetu, kojeg podržavaju gotovo svi mrežni preglednici.

U drugom dijelu rada, prvo će biti sažeto predstavljeno šest radnih okolina, od kojih će tri poslužiti za daljnju komparativnu analizu na samom kraju rada. Radne okoline koje će biti opisane u ovom radu su: Angular, Backbone, Ember, Prototype, MooTools i Node.js. Svaka od navedenih radnih okolina biti će predstavljena opisom najvažnijih značajki te podatcima o tome kada je nastala, tko ju je osmislio te gdje i u kolikoj mjeri se danas koristi, a Angular, Backbone i Ember će biti i tablično uspoređene.

Nakon tablične usporedbe, tri navedene radne okoline bit će sažeto uspoređene jedna s drugom, uz zaključak o tome koja je radna okolina najbolja za koju svrhu te u kojem bi slučaju programeri trebali posegnuti za implementacijom i korištenjem npr. Angular radne okoline, u kojem bi se slučaju trebali opredijeliti za Ember, a u kojem za Backbone.

Na samom kraju rada, biti će predstavljen sistematičan prikaz svih zaključaka na temelju kriterija koji su bili postavljeni za usporedbu, a to su: dizajn službenog mrežnog mjesta, jednostavnost snalaženja po mrežnom mjestu, savladavanje radne okoline, složenost implementacije u HTML, popularnost i budućnost.

2. HYPERTEXT MARKUP LANGUAGE (HTML)

HTML je kratica za *HyperText Markup Language*, što znači prezentacijski jezik za izradu web stranica. Temeljna zadaća HTML jezika jest uputiti web preglednik kako prikazati hipertekst dokument. Valja napomenuti da HTML nije programski jezik niti su ljudi koji ga koriste i njime barataju programeri.

2.1. KRATKI PREGLED POVIJESTI HTML-A

Prvi javno dostupan opis HTML-a je dokument zvan HTML oznake (eng. *tags*), a prvi put se spominje na Internetu od strane Tim Berners-Leeja krajem 1991. godine. Taj opis se sastoji od 20 elemenata početnog, relativno jednostavnog dizajna HTML-a. Trinaest tih elemenata još uvijek postoji u HTML4. Od samih početaka do četvrte verzije HTML-a, modifikacije su se uglavnom svodile na definiranje, specifikaciju ili pročišćavanje raznih oznaka unutar HTML-a, a potonje se odnosi na one oznake koje su postale suvišne, jer su postojale dvije oznake za istu funkciju. Nakon toga, dolazi do HTML5 koji donosi brojne nove mogućnosti koje HTML 4.01 i XHTML 1.x nisu imali, kao što su mogućnost reprodukcije videa na stranicama bez korištenja *Adobe Flasha* ili *Microsoftovog Silverlighta*, mogućnost upravljanja pomoću tipkovnice i opcijama za bilo koju vrstu manipulacija, *drag and drop* kao i ostali novi elementi.[1]

2.2. STATIČNOST HTML-A

Da bi se bolje razumjela potreba za stvaranjem nečega kao što su radne okoline, treba se vratiti korak unazad i objasniti koji je uzrok njihova stvaranja, tj. kakvog su oni problema rješenje. Na tragu tog problema, dolazimo do problema iz naslova ovog poglavlja, a to je statičnost HTML-a. Međutim, postavlja se pitanje – što su uopće statične mrežne stranice?

Najbolje bi bilo reći kako su statične mrežne stranice one koje je stvaratelj same stranice napravio takvima kakve jesu i one se nikada neće promijeniti. Dakle, njihov autor je napisao neki tekst, tj. postavio određeni sadržaj i takva stranica bi mogla ostati u takvom obliku još

[1] CARNet Loomen. URL: https://loomen.carnet.hr/mod/book/view.php?id=116203&chapterid=26734 (24.08.2014.)

godinama, što i ne stvara veliki problem kod stranica koje služe isključivo kao svojevrsne brošure, tj. koje daju nepromjenjive podatke.[2]

Stoga, može se sumirati da su statične stranice one čiji je sadržaj nepromjenjiv, dok su dinamične stranice isprogramirane nekim programskim jezikom i služe kako bi u određenom trenutku (ili aktivacijom nekog elementa na stranici) prikazale potreban sadržaj ili na drugi način dale relevantnu informaciju.[3]

Osim navedene, postoji još jedna vrlo važna razlika između statične i dinamičnih mrežnih stranica, a to ekstenzija samih stranica. Statične su stranice definirane ekstenzijom *.html* i njih možemo pokrenuti neovisno o lokaciji na kojoj se nalazimo, bilo na mreži ili lokalno na računalu. S druge strane, dinamične mrežne stranice obično imaju ekstenziju onog programskog jezika ili skripte koji ju pokreće, primjerice *.php* ili *.asp* ekstenzije.[4]

S obzirom na raznolikost današnjeg svjetskog mrežnog prostora, možemo reći da gotovo sve stranice, počevši od *Facebooka*, *Twittera* pa preko raznovrsnih blogova, ali i portala svih vrsta, imaju potrebu za dinamičnosti koja će omogućiti posjetiteljima interakciju. Ta interakcija može biti u vidu ostavljanja komentara na mrežnoj stranici, ali i u bilo kojem drugom obliku.

[2] Statične i dinamične Internet stranice, 12.04.2012. URL: http://www.web-skola.info/tecaj/itemlist/tag/internet (25.08.2014.)
[3] Isto.
[4] Isto.

3. JAVASCRIPT

JavaScript je najpopularniji skriptni jezik na Internetu kojeg podržavaju svi poznatiji preglednici (*Mozilla Firefox*, *Google Chrome*, *Opera*, *Netscape*, *Internet Explorer*). JavaScript je javno raspoloživ skriptni jezik, koji je stvoren s ciljem dodavanja interaktivnosti HTML stranicama. Skriptni jezici su programski jezici manjih mogućnosti, koji se sastoje od od izvršnog računalnog kôda, obično ugrađenog u HTML stranice. Mogućnosti JavaScripta su mnogobrojne, primjerice moguće je postaviti da se skripta izvršava kada se izvrši neka radnja, npr. kada se stranica učita, ili kada korisnik klikne na određeni gumb ili drugi HTML element. Nadalje, JavaScript može pročitati i promijeniti sadržaj nekog HTML elementa te validirati podatke prije nego se pošalju na server, čime se server oslobađa dodatne obrade.[5]

[5] Java. URL: http://www.mathos.unios.hr/wp/wp2009-10/P8_Java.pdf (25.08.2014.)

4. JAVASCRIPT RADNE OKOLINE

U ovom dijelu rada, ukratko će biti predstavljeno šest radnih okolina, od kojih će tri najpopularnije kasnije biti uvrštene u tablicu za komparativnu analizu. Nakon opisa samih radnih okolina te uvrštavanja u tablicu tri najpopularnije, s odabranim kriterijima za usporedbu, slijedi kratka analiza prema podatcima iz tablice, a koja ujedno služi i kao sinteza te zaključak o prednostima i nedostatcima odabranih radnih okolina. Za potrebe ovog rada, opisane radne okoline su: Angular, Backbone, Ember, Prototype, MooTools i Node.js; a za komparativnu analizu upotrebljene su: Angular, Backbone i Ember.

4.1. ANGULAR

Angular je JavaScript radna okolina otvorenog kôda koji se izvršava na strani klijenta te je održavan od strane *Google*-a. Nastao je 2010. godine, a razvili su ga *Brat Tech LCC, Google and Community*. [6]

Angular je, prije svega, popularan među programerima, a i dizajnerima, zbog svojih 5 svojstava, a to su: *two-way data-binding*, predlošci, MVC, implementacija ovisnosti (eng. *independency injection*) i smjernice (eng. *directives*).[7]

Two-way data-binding je najkorisnije svojstvo Angulara, a ono programeru pomaže na način da značajno smanjuje količinu potrebnog kôda za pisanje. Uobičajene mrežne aplikacije mogu sadržavati do 80% osnovnog (eng. *core*) kôda, a *data-binding* prikriva taj kôd, tako da se programer može usredotočiti na samu aplikaciju.[8]

Ova se radna okolina općenito smatra "tvrđim orahom" za ljude koji nisu upoznati sa MVC arhitekturom, a čak niti znanje o MVC-u nije nužno garancija da će rad s Angularom teći glatko. U takvim slučajevima bolje se opredijeliti se za nešto mnogo jednostavnije, kao što je npr. Backbone radna okolina, o kojem će nešto više biti riječi u sljedećem poglavlju.

[6] AngularJS. URL: https://angularjs.org/ (25.08.2014.)
[7] Ruebbelke, Lukas. 5 Awesome AngularJS Features. URL: http://code.tutsplus.com/tutorials/5-awesome-angularjs-features--net-25651 (25.08.2014.)
[8] Isto.

4.2. BACKBONE

Backbone je krajem 2010. godine osmislio Jeremy Ashkenas, koji je također poznat i po *CoffeeScriptu*, tj. programskom jeziku koji se dekompilira u JavaScript jezik.[9] Backbone je JavaScript biblioteka koja se zasniva na MVP (eng. *model-view-presenter*) paradigmi aplikacijskog dizajna, koji je projektiran kako bi se olakšalo automatizirano testiranje i poboljšalo uklanjanje poteškoća u prezentacijskoj logici. Neke od zanimljivih mrežnih aplikacija napravljenih pomoću Backbone-a su *USAtoday.com*, *WordPress.com*, *Sony Entertainment Network*, *LinkedIn Mobile* i sl.

Backbone je poznat po tome što se njime lako rukuje, budući da se oslanja jedino na JavaScript biblioteku. Osmišljen je za razvoj jednostavnih mrežnih aplikacija i za održavanje raznih dijelova mrežnih aplikacija (npr. višestruki klijenti i server) sinkroniziranim. Na mrežnim stranicama Backbone-a stoji da je taj projekt postavljen na *GitHub*, uz koji stoji u ponudi i izvorni kôd skupa s bilješkama, kao i testni obrazac te primjer aplikacije, lista tutorijala, te lista projekata diljem svijeta koji se koriste Backbone-om.[10]

S Backbone-om, podatci se predstavljaju kao modeli koji se mogu stvarati, kontrolirati, uništavati ili spremati na server poslužitelja. U gotovoj Backbone aplikaciji ne mora se ručno ažurirati HTML ili pisati kôd koji će tražiti element sa specifičnim ID-om, jer kada se model promijeni, pregled se ažurira jednostavno sâm od sebe.[11]

Backbone je, ustvari, pokušaj otkrivanja minimalnog seta podataka za strukturiranje (modela i kolekcija) i korisničkog sučelja (izgleda i URL-ova) koji su obično korisni kada se izrađuju mrežne aplikacije s JavaScriptom. U ekosustavu gdje su radne okoline uglavnom sveobuhvatne te ne daju posebnog prostora korisniku da bude kreativan, nego sve podliježe njihovom uobičajenom ponašanju i izgledu - Backbone pokušava i dalje biti alat koji daje korisniku slobodu stvaranja potpuno osobnog iskustva vlastite mrežne aplikacije.

4.3. EMBER

Ember je radna okolina otvorenog kôda, dakle besplatna JavaScript mrežna aplikacija na klijent strani temeljen na MVC softverskom arhitektonskom obrascu. Omogućava programerima izraditi skalabilne jednostavne mrežne aplikacije. Jednostavna aplikacija,

[9] Backbone.js. URL: http://backbonejs.org/ (27.08.2014.)
[10] Isto.
[11] Isto.

također poznata kao jednostavno sučelje, je mrežna aplikacija smještena u okviru jedne mrežne stranice sa ciljem pružanja lakše interakcije, slične onoj računalima bez pristupa Internetu.[12]

Osnovni koncepti Ember radne okoline su rute, modeli, kontroleri, predlošci, pogledi te komponente ili sastavni dijelovi. O svakom od njih biti će ponešto riječi.[13]

Ruter (eng. *route*) je temeljni Emberov koncept koji naglašava važnost URL-a u upravljanju stanja aplikacije. Objekt rute odgovara URL-u i on u nastavcima emitira trenutno stanje aplikacije. Rute su tako definirane u jedinstvenom ruter objektu.[14] Ember koristi ruter kako bi definirao aplikacijsku logiku s točno određenim stanjima. Svaka od tih ruta može imati proizvoljan broj dodatnih ruta koje mogu odrediti stanje aplikacije. Ruter je također mehanizam uz pomoću kojeg Ember može osvježavati URL aplikacije te pratiti promijene na njemu. Glavna prednost rutera je što omogućuje strukturiranje aplikacijskih stanja prema željenoj hijerarhijskoj strukturi koja sliči dijagramu stanja.[15]

Što se modela tiče, bitno je naglasiti da svaka ruta sadrži povezan model, koji nadalje sadrži podatke povezane s trenutnim stanjem aplikacije.[16] Nadalje, kontroleri se koriste kako bi se modeli učinili logičnijim u prikazu[17], dok su predlošci zapisani pomoću semantičkog mrežnog sustava za predloške zvanog *Handlebars*. Ovaj jezik za predloške koristi se za opis korisničkog sučelja, a predlošci se koriste kako bi se napravio HTML.[18] Kao predložak za prikaz Ember koristi *Handlebars.js*, koji je zadužen za dinamičko generiranje web stranice. Zbog toga većina Ember aplikacija koristi te predloške za prikaz korisničkog sučelja. Svaki pogled će iskoristiti točno jedan predložak koji će koristiti za prikaz elemenata na zaslonu.

Pogledi (eng. v*iews*) se koriste kako bi se dodali profinjeniji, tj. sofisticirani načini za rukovanje korisničkim događajima. Nadalje, njime se dodaju i prilagođene grafike napravljenom bez CSS-a, JavaScript animacije ili se uvodi višekratnost upotrebe u ponašanje predložaka.[19] On je također zadužen za prikaz elemenata aplikacije na zaslon. Pogledi mogu sadržavati više stanja i to obično ovisi o kontroleru koji je vezan za njih. Po osnovnim postavkama svakom pogledu je pridružen točno jedan kontroler i njegov sadržaj. Pogled

[12] Ember.js. URL: http://emberjs.com/ (01.09.2014.)
[13] Isto.
[14] Ember.js: Routing. URL: http://emberjs.com/guides/routing/ (01.09.2014.)
[15] Isto.
[16] Ember.js: Models. URL: http://emberjs.com/guides/models/ (01.09.2014.)
[17] Ember.js: Controllers. URL: http://emberjs.com/guides/controllers/ (01.09.2014.)
[18] Ember.js: Handlebars. URL: http://emberjs.com/guides/templates/handlebars-basics/ (01.09.2014.)
[19] Ember.js: Views. URL: http://emberjs.com/guides/views/ (02.09.2014.)

koristi kontrolera kako bi dohvatio podatke, ali će ga koristiti i za bilo kakve druge akcije koje korisnik napravi na trenutnom zaslonu.

Komponente su specijalizirani izgled za stvaranje prilagođenih elemenata koji mogu biti lako ponovno upotrebljeni u predlošcima. Implementacija Ember komponenti prilagođava se što je više moguće specifikacijama W3C mrežnim komponentama.[20]

Ember je radna okolina koja omogućava korisnicama da, zahvaljujući sustavu *Handlesbars,* s puno manje kôda postignu jednak učinak na stranicu. Kao što je već spomenuto, pomoću *Handlebars* predložaka, sve se automatski ažurira čim se podatci u pozadini promjene. Nadalje, ova se radna okolina diči činjenicom da se s njim ne trebaju raditi trivijalni izbori, budući da on sam ugrađuje uobičajeno korištene idiome, pa se pažnja korisnika može preusmjeriti na ono što će aplikaciju činiti posebnom. Naposljetku, *Emberovci* naglašavaju da je sam sustav izgrađen zbog produktivnosti i učinkovitosti. Oblikovan uzimajući u obzir ergonomiju programera, njegov pristupačan API pomaže programeru da posao napravi kvalitetno i brzo.[21]

4.4. PROTOTYPE

Prototype je JavaScript radna okolina kojeg je stvorio Sam Stephenson u veljači 2005. godine kao dio osnove za AJAX podršku u *Ruby on rails*. Implementiran je kao jedinstven dokument JavaScript kôda obično pod nazivom prototype.js. Distribuira se kako samostalno, tako i kao dio većih projekata, kao što su *Ruby on rails, script.aculo.us* i *Rico*.

Karakteristika *Prototypea* je pružanje raznovrsnih funkcija za programiranje JavaScript aplikacija. Značajke sežu od raznih programskih kratica pa sve do rješavanja *XMLHttpRequesta*. Također, pruža usluge biblioteke za podršku klasa i objekata temeljenih na klasama, što ga čini vrlo korisnim jer upravo to nedostaje JavaScript jeziku. U JavaScriptu stvaranje objekata je temeljeno na *prototype*-u: funkcija za stvaranje objekata može sadržavati *prototype* svojstvo i bilo koji objekt dodijeljen tom svojstvu, bit će iskorišten kao prototip za objekte stvorene tom funkcijom.

Prototype smanjuje ili gotovo izuzima kompleksnost mrežnog programiranja na klijent strani. Stvoren je za rješavanje problema iz realnog života, dodaje korisne ekstenzije

[20] Ember.js: Components. URL: http://emberjs.com/guides/components/ (02.09.2014.)
[21] Ember.js. URL: http://emberjs.com/ (02.09.2014.)

skriptiranom okruženju preglednika i pruža elegantna sučelja za programiranje aplikacija oko nezgrapnih AJAX sučelja i objektnih modela dokumenata.

4.5. MOOTOOLS

MooTools je objektno-orijentirana, JavaScript radna okolina. Pušten je na tržište pod besplatnom, slobodnom MIT licencom. Prvi autor ove radne okoline bio je Valerio Proietti, koji ga je pustio u javnost 2006. godine, a kao inspiracija poslužio mu je *Prototype*. MooTools je zapravo nastao iz popularnog *plugin*-a moo.fx kojeg je Proietti stvorio za *Prototype* 2005. godine, a koji se i dalje održava i koristi. I dok je *Prototype* proširio mnoge JavaScript originalne nizove i funkcije objekata dodatnim metodama, Proietti je želio radnu okolinu koju će, osim toga, također proširivati i originalne elemente objekta. MooTools uključuje brojne komponente, međutim, ne treba svaka od tih komponenti biti učitana za svaku aplikaciju. Neke od značajnijih kategorija komponenti u MooTools bit će predstavljene u sljedećem paragrafu.[22]

Jezgra MooToolsa je zbirka korisnih funkcija koju zahtijevaju sve ostale komponente, a nju proširuje i pruža joj unaprjeđenu funkcionalnost službena kolekcija dodataka u vidu komponente *More*.[23] Zatim, postoji komponenta klasa kao temeljna biblioteka za utjelovljenje klase objekta. Komponenta *Element* sadrži veliki broj poboljšanja i standardizacije kompatibilnosti objekta HTML elementa. *Fx* je napredan API efekt za animaciju elemenata na stranici, dok *Request* uključuje XHR sučelje, kolačiće (eng. *cookies*), te JSON i HTML alate za precizno aportiranje koje služi programerima za iskorištavanje. Na kraju, tu je i *Window* komponenta koja pruža *cross-browser* sučelje za specifičnu klijent informaciju, kao što je npr. dimenzija prozora. Sam *cross-browser* pojam rabi se za opisivanje mogućnosti mrežne stranice, mrežne aplikacije, HTML-a ili skripte na klijent strani da funkcionira u povoljnim uvjetima, tj. onim uvjetima gdje su sadržane sve potrebne značajke za funkcioniranje aplikacije te da se elegantno povuče kada značajke više nisu na raspolaganju.[24]

Sve u svemu, MooTools je kompaktna, modularna i objektno orijentirana JavaScript radna okolina dizajnirana za JavaScript programere na srednjoj ili naprednoj razini. Omogućava pisanje moćnog, fleksibilnog i *cross-browser* kôda sa svojim elegantnim, dobro dokumentiranim i koherentnim aplikacijskim programskim sučeljem. MooTools kôd poštuje

[22] MooTools. URL: http://mootools.net/ (21.09.2014.)
[23] MooTools: Core. URL: http://mootools.net/docs/core (21.09.2014.)
[24] Isto.

stroge standarde i ne izbacuje nikakva upozorenja a opsežno je dokumentiran te ima smislena imena varijabli.[25]

4.6. NODE.JS

Posljednja u nizu radnih okolina, obrađenih za potrebe ove teme jest Node.js, a od nabrojenih jedna je od najbrže rastućih, barem po popularnosti. Prvotno ga je proizveo Ryan Dahl 2009. godine, dok je njegov razvoj i održavanje podržala tvrtka Joyent, gdje je Dahl radio.[26] Inspiracija za Dahla je bila nemogućnost preglednika da vidi koliko je posto datoteke bilo *uploadano*, jer je preglednik tu informaciju morao zatražiti od mrežnog poslužitelja. Dahl je odlučio to promijeniti, te je tako nastao Node.js.

Node.js je višeplatformsko izvršno okruženje namijenjeno poslužiteljskoj strani i mrežnim aplikacijama. Node.js aplikacije su pisane JavaScript jezikom te mogu biti pokrenute unutar Node.js-a na operativnim sustavima *MAC, Microsoft Windows* i *Linux* bez ikakvih izmjena. Njegove aplikacije su dizajnirane kako bi maksimizirale propusnost i učinkovitost. Zbog svoje asinkrone prirode, uobičajeno se koristi za aplikacije u realnom vremenu (eng. *real-time apps).* Node.js sadrži ugrađene HTTP serverske datoteke tako da je moguće pokrenuti web stranicu bez korištenja nekog vanjskog softvera (npr. *Apache*) te tako dopušta veću kontrolu nad postavljanjem samog web poslužitelja. Također omogućuje web programerima da cijelu web aplikaciju naprave koristeći JavaScript jezik na klijentskoj i poslužiteljskoj strani. Node.js se prihvaća kao platforma na poslužiteljskoj strani iznimno visokih performansi te se njime sve značajnije počinju koristiti *Groupon*, SAP, *LinkedIn*, *Microsoft*, *Yahoo*, *Walmart* i *PayPal*.[27]

Još jedna o velikih prednosti Noda.js su njegova proširenja, koja se jednostavno instaliraju uz pomoć ugrađenog upravitelja proširenjima.[28] Iako vrlo moćan i koristan, treba napomenuti da Node.js nije rješenje za sve probleme. Međutim, s druge strane, postoje određene vrste problema u kojima se njegove prednosti dodatno ističu. Računalne programe možemo generalno podijeliti na one koji su više CPU ovisni te na one koji su više vezani uz operacije čitanja i pisanja. Problem CPU ovisnosti može riješiti povećanje broja ciklusa samog procesora. Primjer takvog problema je računanje prostih brojeva. Node.js nije namijenjen za

[25] Isto.
[26] Handy, Alex. Node.js pushes JavaScript to the server-side, 24.06.2011. URL: http://sdtimes.com/node-js-pushes-JavaScript-to-the-server-side/ (04.09.2014.)
[27] Node.js. URL: http://nodejs.org/ (04.09.2014.)
[28] Isto.

rad s takvim problemima. Problem s operacijama čitanje i pisanja se može ublažiti povećanjem propusnosti diska, memorije, mrežne propusnosti te poboljšanog načina spremanja podataka u pričuvni spremnik procesora. Upravo je to područje u kojem Node.js pokazuje svoje pravu snagu, jer je zbog toga i dizajniran. Zbog takvog dizajna može bez problema savladati desetke tisuća istodobnih konekcija, gdje bi druge platforme bile neučinkovite.

5. STRUKTURA KOMPARATIVNE ANALIZE

Komparativna analiza se temelji na pet proizvoljnih kriterija, koji su jednaki za svaku od odabranih radnih okolina. Prvi odabrani kriterij je dizajn službenog mrežnog mjesta, tj. kombinacija vizualnog aspekta i funkcionalnosti određenog mrežnog mjesta. Vrednovanje ovog kriterija očituje se ocjenom od jedan (1) do pet (5), gdje jedinica znači vrlo loš dizajn i nefunkcionalne mrežne stranice, dok petica znači suprotno, dakle oku ugodan dizajn i visoka funkcionalnost mrežnih stranica. Drugi kriterij odnosi se na jednostavnost snalaženja po mrežnom mjestu. Iako donekle povezano za prvi kriterij, ovo je ipak različita komponenta, a opet važna u komparativnoj analizi, budući da neko mrežno mjesto može biti loše estetski uređeno i nefunkcionalno, ali jednostavno za navigaciju kroz stranicu. Vrednovanje ovog kriterija bit će jednako vrednovanju prvog, dakle ocjenama od jedan (1) do pet (5), gdje je petica najbolja, a jedinica najlošija ocjena.

Sljedeći kriterij odnosi se na savladavanje radne okoline i složenost njegove implementacije u HTML. Iako je ovo vrlo kompleksan kriterij, jer ono dakako ovisi o sposobnostima pojedinca, odnosno programera, uvjetima zadatka i konteksta korištenja pojedine radne okoline, u najširem smislu te riječi – za potrebe ovog rada ostat će se i dalje pri metodi ocjenjivanja od jedan do pet, kako zbog dosljednosti, tako i zbog lakšeg uspoređivanja podataka u analizi. Jedinica znači teško savladavanje i implementaciju radne okoline, a petica lako savladavanje i implementaciju iste.

Što se popularnosti tiče, jedinica će značiti malu popularnost, a petica najveću, dok će zadnja stavka, pod nazivom "budućnost" sadržavati vrlo kratki opis (u dvije do tri riječi) o tome kakva se popularnost radnoj okolini predviđa.

5.1. KOMPARATIVNA ANALIZA

RADNA OKOLINA	Dizajn službenog mrežnog mjesta	Jednostavnost snalaženja po mrežnom mjestu	Savladavanje radne okoline i implementacija u HTML	Popularnost	Budućnost
Angular	4	2	4	4	Vrlo rastuća popularnost
Backbone	3	4	3	5	Stagnacija popularnosti u budućnosti
Ember	5	4	2	2	Pad ili stagnacija popularnosti

Najljepši dizajn mrežnog mjesta definitivno ima Ember, čija stranica osim usklađenih boja i oku ugodnih slika, nije pretrpana nepotrebnim sadržajem, a dizajn uz to odiše jednostavnošću dok u isto vrijeme djeluje bogato sadržajem. Što se Angulara tiče, situacija ni tu nije loša, međutim različiti logotipi na početnoj stranici, bez obzira koliko opravdani bili mogu poslati zbunjujuću poruku korisnicima, a gornji dio početne stranice je možda i odveć jednostavno dizajniran, pa djeluje pomalo neprofesionalno. Backbone je u ovom segmentu definitivno zakazao, s obzirom na to da njihovo mrežno mjesto nalikuje djelu autora koji nije previše mario za ostavljanje prvog dojma, a za koji je opće poznato da može bitno utjecati na odluke korisnika i njegovo ponašanje uopće. Stranica djeluje monotono i prazno, a logotip nezgrapno i neelegantno.

Nakon što pristupi mrežnoj stranici i dogodi se prvi dojam, korisnik neminovno kreće s proučavanjem pa sljedeće što može popraviti ili pokvariti prvi stečeni dojam jest lakoća, odnosno složenost snalaženja na mrežnom mjestu. U tom smislu, Angular definitivno ostavlja dojam mjesta na kojem se nešto teže snalazi, s obzirom na niz elemenata koji, ako ništa drugo, nepotrebno zaokupljaju pažnju korisnika te ometaju koncentraciju korisnika koji bi na brz i jednostavan način htio doći do relevantnih uputa i dokumenata koji će mu pomoći u početničkom korištenju radne okoline. S druge strane, Backbone i Ember, upravo zbog svoje jednostavnosti, koja u dizajnerskom smislu možda i nije neki plus, korisniku omogućuju pregledniji pristup sadržaju stranica, što također može utjecati na zadovoljstvo korisnika koje za posljedicu može imati motivaciju korisnika da se duže pozabavi upravo tom radnom okolinom.

Kada se korisnik, na temelju prvih dojmova koje na njega mrežne stranice ostave, odluči za pojedinu radnu okolinu, tu nastupa treći važan element na temelju kojeg se radne okoline mogu uspoređivati, a to je savladavanje radne okoline i složenost implementacije u HTML. Kritički gledano, naravno da će one lakše savladive radne okoline najčešće značiti i manje mogućnosti barem u nekom segmentu, a one teže savladive otvoriti korisniku više mogućnosti u obliku varijacija u pisanju kôda, stoga se niti ne može govoriti u strogom smislu te riječi o dobrim ili lošim radnim okolinama. Jednako vrijedi i za složenost implementacije u HTML, gdje gotovo u pravilu možemo govoriti o tome da teže implementacije često znaju biti bogatstvo u smislu funkcionalnosti i mogućnosti koje sa sobom nose, dok lakše implementacije mogu dovesti do ograničavajućih faktora kao što su vrlo uzak okvir u kojeg se malo toga vanjskog može ugraditi (npr. nepostojanje mogućnosti ugradnje *plugin*-ova). Angular pruža gomilu inovacija primjenjivih na proširenje HTML-a te je savršen izbor za one istinske programere koji su se programiranju posvetili svim svojim bićem.[29] S druge strane, Backbone sam po sebi teži minimalizmu, stoga je malen, brz i lagan za naučiti, a pruža minimum, ili u mnogim slučajevima i mnogo manje onoga što je potrebno da se s njim krene. Upravo je ovdje vidljivo da jednostavnost implementacije može značiti zapravo otvorenost za mnoge nadogradnje (eng. *plugins*) koji mogu obogatiti projekt, dok je Ember teško savladiv za početnike, a ukoliko se radi o programerima upoznatim s MVC pozadinom programiranja u *Rubyju*, *Pythonu*, *Javi* ili bilo kojem drugom objektno orijentiranom jeziku, tada i Ember može biti lagan za savladati.[30]

Savladavanje i složenost implementacije radne okoline u HTML je za potrebe ovog rada testirana uobičajenim *Hello World* primjerom, na kojem se jasno mogu nazrijeti razlike. Da bi se što vjerodostojnije obrazložili komentari i prosudbe o savladavanju i složenosti implementacije odabranih triju radnih okolina, primjer *Hello World* napravljen je sa sve tri radne okoline te je nakon opisa samog procesa implementacije dodan zaključak o bitnim karakteristikama svih pojedinih radnih okolina.

Prvi, i nešto složeniji, primjer je onaj radne okoline Ember. Za korištenje Embera, trebalo bi se svakako prvenstveno upoznati s MVC modelom, o kojem je bilo ranije riječi u radu. Za sâm početak potrebno je stvoriti HTML dokument:

[29] Shaked, Uri. AngularJS vs. Backbone.js vs. Ember.js. URL: http://www.airpair.com/js/javascript-framework-comparison (10.09.2014.)
[30] Isto.

```
<html>
<head>
<script
src="https://ajax.googleapis.com/ajax/libs/jquery/1.7.2/jquery.min.js"
type="text/javascript">
</script>
<script type="text/javascript"
src="http://www.codeproject.com/scripts/ember.min.js"></script>
</head>
<body>
<script type="text/x-handlebars">
{{#view App.UserView }}
<b>Name:</b> {{name}}
{{/view}}
</script>

<button onclick="btnTest_OnClick()" id="btnTest">Klikni za
promjenu</button>

</body>
</html>
```

Nakon toga, stvara se MVC model, a budući da Ember zahtjeva aplikacijske imenske prostore, prvo se unese sljedeća naredba:

```
window.App = Ember.Application.create();
```

Zatim, dolazi model, instanca i kontroler, kao u primjeru:

```
App.Person = Ember.Object.extend({
  id: 0,
  name: ""
});

var person = App.Person.create();
person.name = "Domagoj";
person.id = 0;

App.userController = Ember.Object.create({
  content: person,
  changeModel: function () {
    this.content.set('name', 'Ivan');
  }
});
```

Dogovor je da kontroler ima polje nazvano *sadržaj* koje služi kao referenca podatcima koje kontrolira, tj. svojem modelu. U tu svrhu dodaje se funkcija promjene modela koja mijenja ime, npr. u Ivan.

Nakon što smo uspostavili kontrolera i model, slijedi preostala komponenta MVC-a, a to je klasa pogleda:

```
App.UserView = Ember.View.extend({
  nameBinding: 'App.userController.content.name'
});
```

Ovdje se uspostavila veza, tako da ako se polje kontrolera ikad promjeni, tada bi se promjene automatski osvježile u pogledu, a važno je paziti i na imenovanje tog vezivanja, tj. ono mora biti povezano s poljem *{{name}}* koji se stvorio u predlošku pogleda.

Na kraju je potrebno deklarirati i gumb iz HTML dokumenta, da bi se klikom na njega osvježili podatci:

```
function btnTest_OnClick() {
  App.userController.changeModel();
}
```

Sljedeći primjer je nešto jednostavniji. Za početak potrebno je preuzeti predložak u obliku HTML dokumenta.[31] Ovaj dokument sadrži knjižnice koje su potrebne i rezervirana mjesta za HTML i JavaScript kôd. Nakon toga, kreće se s Backbone pogledom. On je ekvivalent kontroleru u MVC-u, povezuje korisničke događaje (npr. klikove i sl.) i uzvraća HTML predloške te dolazi u kontakt s modelima koje sadrže podatke o aplikaciji. Primjer gore navedenoga:

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge,chrome=1">
  <title>Hello World in Backbone.js</title>
</head>
<body>
  <!-- Your HTML -->

  <div id="container">Loading...</div>

  <!-- Libraries -->

  <script src="http://ajax.googleapis.com/ajax/libs/jquery/1.7.2/jquery.min.js" type="text/javascript"></script>
  <script src="http://cdnjs.cloudflare.com/ajax/libs/underscore.js/1.3.3/underscore-min.js" type="text/javascript"></script>
  <script src="http://cdnjs.cloudflare.com/ajax/libs/backbone.js/0.9.2/backbone-min.js" type="text/javascript"></script>
  <script src="http://cdnjs.cloudflare.com/ajax/libs/backbone-localstorage.js/1.0/backbone.localStorage-min.js" type="text/javascript"></script>

  <!-- Javascript code -->

  <script type="text/javascript">
    var AppView = Backbone.View.extend({
      el: $('#container'),
      template: _.template("<h3>Hello <%= who %><h3>"),
      initialize: function(){
        this.render();
      },
      render: function(){
        this.$el.html(this.template({who: 'Domagoj!'}));
      }
    });
    var appView = new AppView();
  </script>

</body>
</html>
```

[31] URL: https://raw.githubusercontent.com/amejiarosario/Backbone-tutorial/439ff34409dfc01adca7f9f96efcd726295f1aac/backbone-tutorial.html (23.09.2014.)

Najjednostavniji od uspoređenih radnih okolina definitivno je Angular. Kôd potreban za primjer *Hello World* je sljedeći:

```
<!doctype html>
<html ng-app>
<head>
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.0-
rc.2/angular.min.js"></script>
</head>
<body>
<div>
<label>Ime:</label>
<input type="text" ng-model="ime" placeholder="Upišite svoje ime">
<hr>
<h1>Hello {{ime}}!</h1>
</div>
</body>
</html>
```

I iz ovih jednostavnih primjera, kao što je već spomenuto na početku ovog poglavlja, može se zaključiti da su neke radne okoline jednostavnije za implementaciju u HTML od drugih. Dok se kod Angulara sve svodi na par instanci, kod Embera je priča znatno drugačija. Kako god bilo, iskustvom iz prve ruke na jednostavnom *Hello World* primjeru, korisnik dobiva dojam o tome koliko je implementacija pojedine radne okoline komplicirana te na temelju toga može lakše donijeti odluku o tome koju radnu okolinu i u koje svrhe koristiti.

Zadnja dva kriterija iz tablice se odnose na popularnost radnih okolina te na izglede i popularnost u budućnosti. Ovim podatcima dobiva se slika o tome kojim su od navedenih radnih okolina korisnici i programeri najzadovoljniji, a trenutno najpopularnija od analiziranih radnih okolina je upravo Backbone, ali s trendom usporavanja rasta popularnosti, odnosno stagnacije, dok u tome napreduje Angular, koji je iako trenutno iza Backbone-a po broju stranica koje ga koriste, to se vrlo brzo može promijeniti, s obzirom na tendenciju vrlo rastuće popularnosti, koju dodatno podupire *Google* zajednica. Što se Embera tiče, ova trenutno najmanje zastupljena od navedenih radnih okolina i nema neku tendenciju promjene na pozitivno, što ne mora značiti da svojim inovativnim akcijama u budućnosti to ne može promijeniti.[32]

[32] SimilarTech. URL: https://www.similartech.com/categories/javascript (11.09.2014.)

6. ZAKLJUČAK

Kako bi se virtualni prostor učinio dinamičnijim i time prikladnijim za upotrebu sve zahtjevnije mrežne zajednice diljem svijeta, bilo je nužno utjecati na statičnu prirodu HTML-a. To se i dogodilo uvođenjem JavaScript koncepta, čije su biblioteke nažalost često previše opširne da bi ju programeri, a pogotovo početnici, mogli savladati, njima se funkcionalno koristiti ili unaprjeđivati.

Ipak, s obzirom na različite potrebe raznih programera i zadataka koje njihove aplikacije i mrežne stranice trebaju obavljati, nije se moglo stati samo na jednoj radnoj okolini, jer ona ni izbliza ne bi mogla odgovoriti na sve izazove koje pred nju stavljaju potrebe već spomenutih faktora. Stoga smatram da sva raznolikost radnih okolina nije samo pûko dobro u kontekstu stvaranja dinamičkog Interneta, već i nužnost ukoliko se želi odgovoriti na sve zahtjeve suvremenih aplikacija i ispuniti sve ideje današnjih programera.

Stoga, od ključne je važnosti paralelno postojanje međusobno različitih radnih okolina, iz razloga da bi svaki korisnik mogao izabrati onaj koncept koji najbolje odgovara zamišljenoj ideji. Tako će kreativni programer odabrati radnu okolinu koja mu pruža mogućnost da se maksimalno izrazi te ga neće opterećivati ograničavajućom i već postojećem formom, dok će s druge strane, programer orijentiran k funkcionalnosti težiti odabrati stabilnu radnu okolinu s kvalitetnim *templating*-om, tj. setom predložaka, uz koju će se on moći pozabaviti nekim drugim aktivnostima, njemu važnijim aktivnostima.

LITERATURA

1. AngularJS. URL: https://angularjs.org/ (25.08.2014.)
2. Backbone.js. URL: http://backbonejs.org/ (27.08.2014.)
3. CARNet Loomen. URL: https://loomen.carnet.hr/mod/book/view.php?id=116203&chapterid=26734 (24.08.2014.)
4. Ember.js. URL: http://emberjs.com/ (01.09.2014.)
5. Ember.js: Components. URL: http://emberjs.com/guides/components/ (02.09.2014.)
6. Ember.js: Controllers. URL: http://emberjs.com/guides/controllers/ (01.09.2014.)
7. Ember.js: Handlebars. URL: http://emberjs.com/guides/templates/handlebars-basics/ (01.09.2014.)
8. Ember.js: Models. URL: http://emberjs.com/guides/models/ (01.09.2014.)
9. Ember.js: Routing. URL: http://emberjs.com/guides/routing/ (01.09.2014.)
10. Ember.js: Views. URL: http://emberjs.com/guides/views/ (02.09.2014.)
11. Handy, Alex. Node.js pushes JavaScript to the server-side, 24.06.2011. URL: http://sdtimes.com/node-js-pushes-JavaScript-to-the-server-side/ (04.09.2014.)
12. Java. URL: http://www.mathos.unios.hr/wp/wp2009-10/P8_Java.pdf (25.08.2014.)
13. MooTools. URL: http://mootools.net/ (21.09.2014.)
14. MooTools: Core. URL: http://mootools.net/docs/core (21.09.2014.)
15. Node.js. URL: http://nodejs.org/ (04.09.2014.)
16. Ruebbelke, Lukas. 5 Awesome AngularJS Features. URL: http://code.tutsplus.com/tutorials/5-awesome-angularjs-features--net-25651 (25.08.2014.)
17. Shaked, Uri. AngularJS vs. Backbone.js vs. Ember.js. URL: http://www.airpair.com/js/javascript-framework-comparison (10.09.2014.)
18. SimilarTech. URL: https://www.similartech.com/categories/javascript (11.09.2014.)
19. Statične i dinamične Internet stranice, 12.04.2012. URL: http://www.web-skola.info/tecaj/itemlist/tag/internet (25.08.2014.)
20. URL: https://raw.githubusercontent.com/amejiarosario/Backbone-tutorial/439ff34409dfc01adca7f9f96efcd726295f1aac/backbone-tutorial.html (23.09.2014.)