ANALIZA ZAHTIJEVANIH KARAKTERISTIKA FRONTEND FRAMEWORKA ZA RAZVOJ WEB APLIKACIJA

Troskot, Krešimir

VELEUČILIŠTE U RIJECI

Krešimir Troskot

ANALIZA ZAHTIJEVANIH KARAKTERISTIKA FRONTEND FRAMEWORKA ZA RAZVOJ WEB APLIKACIJA

(specijalistički završni rad)

Rijeka, 2018.

VELEUČILIŠTE U RIJECI

Poslovni odjel

Specijalistički diplomski stručni studij Informacijske tehnologije u poslovnim sustavima

ANALIZA ZAHTIJEVANIH KARAKTERISTIKA FRONTEND FRAMEWORKA ZA RAZVOJ WEB APLIKACIJA

(specijalistički završni rad)

MENTOR

Dr.sc.Marin Kaluža

STUDENT

Krešimir Troskot

MBS: 2422000117/16

Rijeka, rujan 2018.

VELEUČILIŠTE U RIJECI
Poslovni odjel
Rijeka, 07.05.2018.

ZADATAK
za specijalistički završni rad

Pristupniku: Krešimir Troskot **MBS:** 2422000117/16

Studentu specijalističkog stručnog studija informatike izdaje se zadatak za specijalistički završni rad – tema specijalističkog završnog rada pod nazivom:

Analiza zahtjevanih karakteristika front-end framework-a za razvoj web aplikacija

Sadržaj zadatka:

Deskripcijom objasniti tehnike i postupke razvoja web aplikacija. Objasniti razloge upotrebe *front-end framework*-a (FEFW) u razvoju web aplikacija. Analizirati potrebe za mogućnostima FEFW-a kojima se: osigurava jednoobraznost i strukturiranje izvornog programskog koda, omogućuje korištenje dodatnih komponenata, distribuira izvođenja prikaznih elemenata na poslužitelj, povećava brzina razvoja i usklađenost s drugim komponentama, povećava brzina izvođenja aplikacije, te olakšava upravljanje stanjem podataka. Istražiti dostupne i često korištene FEFW-ove. Analizirati odabrane FEFW-ove, te primjerima pokazati na koji se način ostvaruju zahtjevane mogućnosti FEFW-a.

Preporuka: _____

Rad obraditi sukladno odredbama Pravilnika o završnom radu Veleučilišta u Rijeci.

Zadano: 07.05.2018 **Predati do: 07.11.2018**

Mentor:
dr.sc. Marin Kaluža, v.pred.

Pročelnik odjela:
mr.sc. Marino Golob, pred.

Zadatak primio dana: 07.05.2018
K. Troskot
Krešimir Troskot

Dostavlja se:
- **mentoru**
- **pristupniku**

Prilog 5.

IZJAVA

Izjavljujem da sam specijalistički završni rad pod naslovom ANALIZA ZAHTJEVANIH KOMPETENCIJA FRONT-END FRAMEWORKA ZA RAZVOJ WEB APLIKACIJA izradio samostalno pod nadzorom i uz stručnu pomoć mentora MARIJU KALUŽU.

Ime i prezime

K. Trošelj

(potpis studenta)

Sažetak

Usporedba Javascript radnih okvira (eng. *framework* - FW) za izradu Web aplikacija je završni rad koji uspoređuje popularne Javascript frontend radne okvire (eng. *front-end framework* - FEFW). Na samom početku objasnit će se dvije osnovne vrste web aplikacija, one višestranične aplikacija (eng. *Multi Page Aplication* - MPA) i one jednostranične (eng. *Single Page Aplication* - SPA). Definirat će se glavna pitanja bitna za izradu ovih vrsta aplikacija i na temelju njih raditi usporedba tri najpoznatija FEFW – Angular, React.js, Vue.js. Na samom kraju, analizirati će se i objasniti usporedba tri FEFW.

Ključne riječi: SPA, MPA, Angular, React.js, Vue.js

SADRŽAJ

1. Uvod

Zadnjih godina postoji ogromna potražnja za vrlo sofisticiranim i složenim web aplikacijama koje žele zamijeniti staru desktop aplikaciju u skoro svim područjima. Iako se svijet sve više okreće prema mobilnim aplikacija, danas još uvijek postoje dvije glavne vrste web aplikacija tzv. Multi Page aplikacije (MPA) i Single Page aplikacije (SPA).

Razvoj web aplikacija često se temelji na korištenju nekih radnih okvira (eng. *framework* - FW). Razvojem modernog Javascripta, korištenje frontend radnih okvira (eng. *frontend framework* - FEFW) uvelike je poraslo. Motivacija za odabir ove teme proizašla je iz činjenice da postoji veći broj FEFW koji se mogu koristiti za razvoj istih ili sličnih funkcionalnosti web aplikacije. Taj veliki broj FEFW dovodi do problema u kojem korisnik ne zna koji od dostupnih odabrati, te koji je moguće lakše koristiti u razvoju specifičnih web aplikacija.

Svrha rada je upoznati se sa modernim Javascript FEFW, odgovoriti na pojedina pitanja bitna kod odabira određenog FEFW za specifične slučajeve korištenja.

Cilj je usporediti tri FEFW na svim razinama i objasniti koji od njih je najbolji za razvoj specifičnih aplikacija.

U radu će se objasniti značenja MPA i SPA, te potreba za izgradnjom takvih web aplikacija. Definirati će se pitanja koja utječu na razvoj MPA i SPA aplikacija. Istražiti će se i analizirati dostupni i često korišteni FEFW-ovi, te primjerima pokazati na koji se način ostvaruju zahtjevane mogućnosti FEFW-a.

2. SPA i MPA

Za kreiranje dobrog korisničkog iskustva važno je odabrati prikladnu arhitekturu web aplikacije. Izbor između oblika pojedinačnih (eng. *Single Page Application – SPA*) ili više stranica (eng. *Multi Page Application – MPA*) često zahtijeva detaljniju analizu mogućnosti arhitekture i potrebe korisnika. MPA su obično namjenjene za veće sustave s većim brojem različitih tipova usluga, koji preferiraju više vrsta interakcije sa svojim posjetiteljima. SPA predstavlja noviji pristup u izradi web aplikacija, a često se koristi u razvoju jednostavnijih aplikacija, s manjom količinom sadržaja (Dimi, 2017). U nastavku će se detaljnije prikazati princip rada obiju vrsta aplikacija.

2.1. MPA

MPA funkcioniraju na "tradicionalan" način. Svaka promjena u pregledniku (kao npr. prikaz ili slanje podataka) podrazumijeva dohvaćanja nove stranice sa poslužitelja (Neoteric, 2016).

Slika 1. MPA princip rada

Izvor: http://pepa.holla.cz/wp-content/uploads/2015/10/Pro-Single-Page-Application-Development.pdf (13.03.2018.)

Na slici iznad (Slika 1.) prikazan je princip rada MPA aplikacija. Na poslužiteljskoj strani registrirane su rute, a svaki zahtjev klijenta prema poslužitelju podrazumijeva vraćanje nove HTML stranice. To znači da će zahtjev poslan poslužitelju, uvijek vratiti stranicu sa prikazom rezultata u radu zahtjeva, ili grešku. Većina aplikacijske logike nalazi se na strani poslužitelja, a klijent je samo primalac dohvaćene stranice.

Ovaj pristup izradi Web aplikacije je prikladan za razvoj manjih aplikacija i kada se npr. Javascript koristi za jednostavnije akcije nad učitanom stranicom korisničkim sučeljem ili za razvoj animacija. No, ako postoji potreba za stvaranjem zahtjevnijeg korisničkog sučelja, stranica bi mogla postati vrlo složena i biti učitana s puno podataka i programskih elemenata (često se koristi Javascript) koji opterećuju preglednik i računalo klijenta. Budući da stvaranje složenih stranica na poslužitelju i njihovo prenošenje i prikazivanje klijentu zahtjeva puno vremena i degradira korisničko iskustvo, početkom 2000-ih MPA je poboljšana uvođenjem AJAX-a (eng. *Asynchronous JavaScript And XML*), koji je omogućio osvježavanje samo dijelova stranice, a ne cijele stranice. Ova tehnika je pomogla poboljšati korisničko iskustvo, ali je pritom omogućila razvoj kompleksnijih mreža stranica, pa je time i upravljanje izvornim programskim kodom postalo složenije. Ovo je jedan od razloga zašto se pojavljuju FEFW-ovi. (Shimanovsky, 2016).

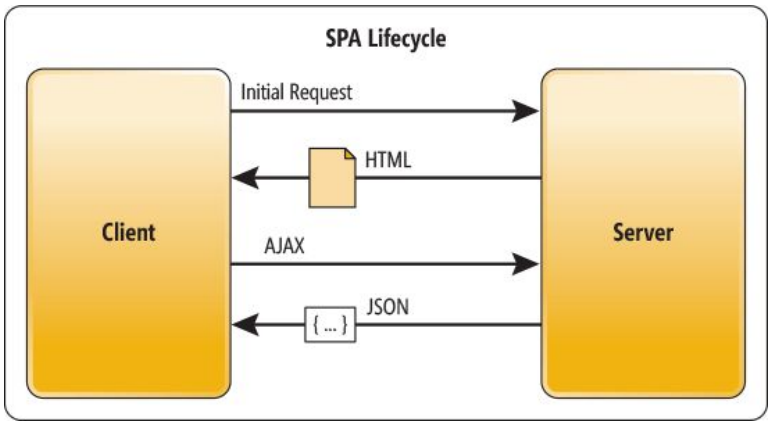
2.2. SPA

SPA se objašnjava kao kompletna web aplikacija koja ima samo jednu stranicu koja služi kao „ljuska“ za sve prikaze dijelova korisničkog sučelja. Većina resursa učitava se samo jednom tijekom cijelog radnog ciklusa aplikacije, a podaci se prenose iz prethodnih u nova stanja. Inicijalni HTML dokument učitan pri prvom otvaranju aplikacije predstavlja početnu točku za ostatak aplikacije. Svaki sljedeći dio učitava se dinamički i neovisno o „ljusci“, bez ponovnog učitavanja cijele stranice, dajući korisniku percepciju da se stranica promijenila. Ljuska je,

najčešće, minimalna u strukturi i često sadrži jednostruku, praznu oznaku ( ) koja će „ugostiti“ ostatak sadržaja aplikacije (Emmit, Scott, 2016).

Nakon početnog zahtjeva prema poslužitelju, u preglednik se učitava kompletna HTML stranica. Svaka druga interakcija klijenta i poslužitelja odvija se AJAX tehnikom, što znači da preglednik ažurira samo dio stranice koji se mijenja (podaci), bez osvježavanja kompletne stranice (Saxena, 2014.)

Slika 2. SPA princip rada



Izvor: http://www.c-sharpcorner.com/uploadfile/rahul4_saxena/single-page-application-spa-using-angularjs-web-api-and-m/ (23.03.2018.)

Na slici iznad (Slika 2.) prikazan je radni ciklus SPA web aplikacija. Ako se Slika 2. usporedi sa Slikom 1 koja prikazuje radni ciklus MPA, vidi se razlika u prirodi zahtjeva i odgovora, odnosno jedan radni ciklus MPA aplikacije završava primitkom odgovora, a radni ciklus SPA aplikacije traje i ovisi o radu korisnika kroz korisničko sučelje. SPA aplikacije koriste AJAX pri slanju zahtjeva poslužitelju, a kao odgovor primaju podatke, te primaju male dijelove HTML-a za prikaz primljenih podataka. Jednom kada podaci stignu sa poslužitelja, klijentska strana će uobličiti primljene sadržaje i prikazati na određenom mjestu.

2.3. Prednosti i nedostaci SPA u odnosu na MPA

Neke od prednosti izrade SPA u odnosu na MPA su:

- Brzo vrijeme reakcije – Budući da SPA unaprijed preuzima strukturu web stranice, ne postoji potreba za stalnim zahtjevima za dobivanjem nove stranice s poslužitelja. Kada korisnik klikne negdje, promjene se izvršavaju „trenutno“, što daje osjećaj sličan interakciji s mobilnim ili desktop aplikacijama. Korisnik ne mora čekati ili, barem, nema taj osjećaj čekanja. To je velika prednost i glavni razlog zbog kojeg su SPA aplikacije danas toliko popularne.
- Odvajanje prezentacijske logike od poslovne – Kod kojim se upravlja ponašanje korisničkog sučelja sadržan je na klijentskoj strani umjesto na poslužitelju. Ovo omogućuje da se developer može usredotočiti odvojeno na ono što je važno za korisnički doživljaj i na kritične stavke poslovne logike na poslužitelju. Dakle, miješanje prezentacijske i poslovne logike je teže, što omogućuje održavanje i ažuriranje svake strane odvojeno.
- Brži i laganiji prijenos podataka – Transakcije s poslužiteljem su lakše i brže jer se, nakon inicijalne isporuke, samo podaci šalju na ili primaju sa poslužitelja (Emmit, Scott, 2016:13).
- Moguća offline podrška – Budući da SPA koristi Javascript koji se izvodi na klijentskoj strani, teoretski, internetska veza nije potrebna cijelo vrijeme. Moguće je osigurati da aplikacija i dalje funkcionira kada korisnik nije povezan ili privremeno ne koristi internetsku vezu. U tom slučaju se lokalni podaci sinkroniziraju s onima na poslužitelju (Apps Team, 2013).

Postoje i neki nedostaci SPA u odnosu na MPA:

- Optimizacija na tražilicama (eng. *Search engine optimization* – SEO) je teško izvediv – Odnosi se na optimizaciju web stranice ili aplikacije da postigne što viši rang u rangiranju sadržaja na tražilicama (Fink, Flatow, 2014). Svaka tražilica ima tri funkcije:

1. kravling (eng. *crawling*) – odnosi se na otkrivanje podataka o web stranici. To uključuje skeniranje web lokacija i prikupljanje pojedinosti o svakoj stranici: naslovi, slike, ključne riječi, ostale povezane stranice itd.
2. indeksiranje (eng. *indexing*) – To je proces obrađivanja i smještanja podataka prikupljenih crawlingom u bazu podataka.
3. serviranje (eng. *serving*) – Cilj ove funkcije je dohvaćanje relevantnog sadržaja u trenutku korisnikovog upita tražilici (Bruce, 2016).

Kada korisnik izvodi određenu akciju na korisničkom sučelju u SPA aplikaciji, i potrebno je preuzeti nove podatke, Javascript će prilagoditi samo dio stranice bez osvježavanja i otvaranja nove. Budući da kravling podrazumijeva skeniranje sadržaja na temelju URL adresa i promjena stranica, a SPA ima samo jednu inicijalnu stranicu, tu nastaje problem. Promijenjena stranica neće biti vidljiva (Mikowski, 2014).

SEO optimizacija za SPA se svakim danom sve više unaprjeđuje i postaje bolja. Danas se tako pojavljuju alati i načini koji omogućuju predočavanje dijelova aplikacije na poslužitelju tako da kravling vidi ono što korisnik vidi, ali to je, ipak, još uvijek u nastajanju i teže izvedivo nego u drugom pristupu (Zanon, 2015).

- Javascript mora biti uključen – Ako korisnik isključi Javascript u svom pregledniku, aplikacija neće biti prikazana onako kako bi trebala. Budući da je takvih korisnika oko 1.3%, to može biti problem (Hein, 2010).
- Za sigurnost potrebno više rada – Ovaj princip nije nesiguran, no budući da je ovaj način stvaranja aplikacija relativno nov, neki sigurnosni problemi su djelomično riješeni te je potrebno uložiti više truda u proučavanje načina i najboljih praksi za rješavanje specifičnih sigurnosnih pitanja.

3. Javascript frontend FW-ovi

Uspoređivat će se tri trenutno najpopularnija Javascript frontend FW-a čija popularnost, i dalje, neprestano raste. Na slici ispod (Slika 3.) prikazana je ukupna ljestvica svih dostupnih Javascript FW-a. Budući da se ovaj rad odnosi na FW-ove za frontend stranu, izdvojeni su i analizirat će se: Angular, React i Vue.js.

Slika 3. Ljestvica popularnosti Javascript FW-ova

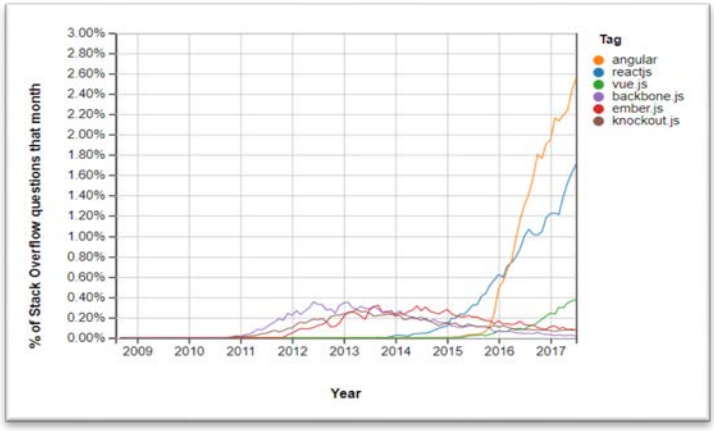
JavaScript

Framework	Score
AngularJS	96
React	93
Angular	90
Express	87
Meteor	86
Vue.js	86
Ember.js	82
Sails.js	76
Aurelia	71
Koa	68
Dojo	66
OpenUI5	64
Feathers	62

Izvor: https://hotframeworks.com/languages/javascript (23.03.2018.)

React, sa preko 82 tisuće Github zvjezdica, smatra se, vjerojatno, najpopularnijim Javascript frontend FW-om. Vue.js, sa oko 76 tisuća Github zvjezdica, jedan je od, po popularnosti, najbrže rastućih (razvijen je tek 2016. godine). S druge strane, Angular sa svojom ogromnom pozadinom, Google-om (od koga je razvijen) i Microsoftom, čiji jezik koristi (Typescript), jedan je od najstabilnijih. Među šest najpoznatijih Javascript frontend FW-a provedena je analiza te je ustanovljeno da samo ova tri navedena, trenutno bilježe rast popularnosti, dok ostali stagniraju (Slika 4.).

Slika 4. Ljestvica popularnosti 6 najpoznatijih Javascript FW-a



Izvor:
https://insights.stackoverflow.com/trends?tags=reactjs%2Cangular%2Cvue.js%2Cember.js%2Cbackbone.js%2Cknockout.js (27.03.2018.)

3.1. React.js

React je predstavljen kao Javascript knjižnica za izgradnju korisničkih sučelja. Objavljen je u ožujku 2013. godine od strane tvrtke Facebook, koja koristi React komponente na nekoliko svojih stranica, a ne kao jednu SPA (Neuhaus, 2017). Kao i Vue.js, i React koristi komponentno baziranu arhitekturu, koja omogućuje pisanje koda na laganiji i dobro održiv način (React, 2018-1). Iako nije obavezno, komponente se obično pišu u JSX-u (eng. *Javascript XML)*, React-specifičnom proširenju Javascripta, koji omogućuje korištenje HTML-a unutar Javascripta (React, 2018-2). Budući da koristi virtualni DOM, React kreira strukturu podataka u memoriji, izračunava razlike između virtualnog i stvarnog DOM-a, te nakon toga ažurira stvarni DOM u pregledniku na vrlo brz i učinkovit način (Kurian, 2017).

3.2. Angular

Angular je FW za izradu klijentskih aplikacija u HTML-u i JavaScript ili u jeziku kao što je TypeScript koji se kompajlira u JavaScript. Razvijen je od strane Google-a 2010. godine kao AngularJS, a 2014. je godine kompletno prerađen i nanovo napisan te od tada djeluje pod nazivom Angular. FW se sastoji od nekoliko knjižnica, od kojih su neke osnovne i neke opcionalne. Angular aplikacije razvijaju se pisanjem HTML predložaka pomoću angulariziranog označavanja, pisanjem klasa komponenata za upravljanje tim predlošcima, dodavanjem logike aplikacija u servise i registriranjem komponenata i servisa u module. Aplikacija se pokreće tako da učita korijenski modul. Angular preuzima prezentaciju sadržaja aplikacije u pregledniku i reagira na interakciju korisnika prema uputama koje ste unijeli (Angular, 2018-1). Za ovaj rad koristit će se trenutno zadnja dostupna verzija Angular 6 verzija.

3.3. Vue.js

Vue.js je jedan od, po popularnosti, najbrže rastućih Javascript FW-a današnjice. Na službenim stranicama se opisuje kao pristupačan, svestran i visoko performantan FW za izgradnju interaktivnih sučelja (Vue, 2018-1). Budući da je FW komponentno orijentiran i komponenta, nakon određenog internog ponašanja/računanja, vraća predložak kao izlaz, sve ove komponente ponovno su upotrebljive unutar stranice ili unutar drugih komponenata. Vue.js koristi virtualni DOM, koji ne postoji u pregledniku već u memoriji, a rezultat je puno brži pristup nego kada je riječ o stvarnom DOM-u. Na kraju se ovaj ažurirani virtualni DOM prezentira kao stvarni (Andersen, 2017). Ovaj FW je najpopularniji na istoku, a neke od kompanija koje ga koriste su Alibaba, Xiaomi, Baidu, Tencent itd. (Clockwise Software, 2017).

4. Uspoređivanje FW-a

Klasifikacija FW-a odvijat će se kroz nekoliko poglavlja. U nastavku će se objasniti pitanja koja će biti analizirana kod provjeravanja je li određeni FW optimiziran za izradu ovih vrsta aplikacija, te će se pokazati i praktična primjena.

4.1. Jednoobraznost i strukturiranje izvornog programskog koda

Strukturiranje programskog koda moguće je izvršiti konvencijski upotrebom radnih dijagrama, ili ubacivanjem skripte.

Kod MPA, kod kojih, obično, postoji jako puno stranica na kojima je kontroliranje samo malog dijela DOM-a čest slučaj, početak rada dodavanjem samo skripte najbolji je način. U ovom slučaju govorimo o izgradnji jednostavnijih komponenata, a FEFW koji ovo omogućuje smatra se upravljivijim i jednostavnijim za rukovanje (Burgess, 2016).

S druge strane, čišći, organizirani JavaScript kod i dobro modularizirana arhitektura, u kojoj je svaki dio aplikacije zasebni dio, predstavlja korak naprijed kod izgradnje skalabilne i održive SPA (Emmit, Scott, 2016). Kodiranje s modulima pomaže organizirati logiku aplikacije u male jedinice koje je lakše održavati i ažurirati. Ovo dovodi do veće ponovljivosti koda što zadovoljava princip DRY (eng. *Don't repeat yourself*). DRY je osnovni princip razvoja softvera koji podrazumijeva smanjenje ponavljanja uzoraka softvera (Baghel, 2017).

4.1.1. Script

Početak rada sa nekim FW-om dodavanjem samo skripte najbolji je način ukoliko korisnik želi kontrolirati samo jedan mali dio DOM-a.

Rad sa Vue.js FW-om u potpunosti je moguć jednostavnim dodavanjem skripte ili lokalno preuzimanjem sa službene stranice. Nije potreban nikakav radni dijagram niti moramo koristiti .vue datoteke koje dolaze s njime (Vue, 2018-2).

Slika 5. Vue.js početak rada ubacivanjem skripte

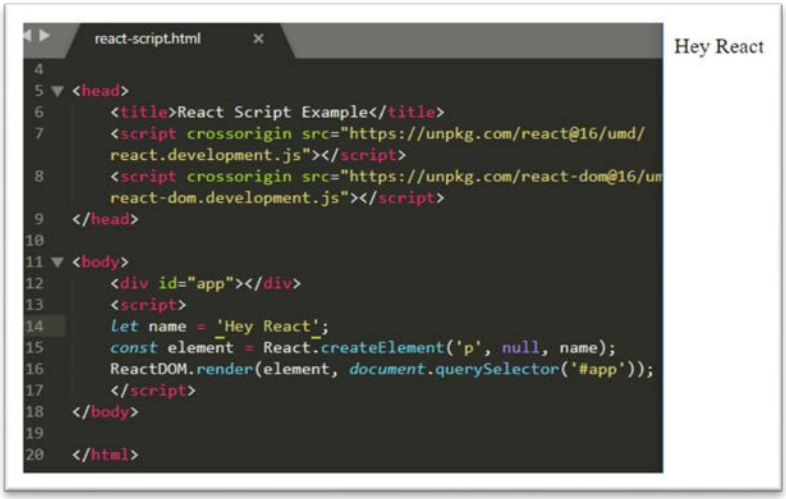
Izvor: obrada autora

Na slici iznad (Slika 5.) vidljiv je početak rada u Vue.js-u ubacivanjem skripte. Skripta je ubačena unutar <head> oznaka. Na kraju unutar <script> oznake kreirana je nova Vue instanca kojoj su prosijeđene informacije koje će se prikazivati.

U Angularu nije moguće započeti rad ubacivanjem skripte. Radni dijagram je potreban zbog samog koncepta kako ovaj FW funkcionira (modularni model temeljen na komponentama stvoren s TypeScript-om) i zbog toga su potrebni alati za izgradnju projekta. Typescript je jezik otvorenog koda razvijen od strane Microsoft-a, a predstavlja superset Javascript-a koji pruža dodatne mogućnosti kao npr. opcionalni statički tipovi, klase, moduli i slično (TypeScript, 2017). Budući da preglednici ne razumiju ovaj jezik, potrebno ga je prvo prevesti u Javascript nakon čega se može, kao i sav drugi Javascript kod, jednostavno izvršiti u pregledniku. Prevođenje u Angular-u se odvija korištenjem Typescript kompajlera koji je već ugrađen unutar Webpack alata koji dolazi sa ovim FW-om (Angular, 2018-2).

Iako se na službenim stranicama React.js-a snažno podupire početak rada izradom radnog dijagrama, to je moguće izvršiti i jednostavnim dodavanjem skripte, baš kako je to napravljeno kod Vue.js-a (React, 2018-3). U tom slučaju kod je potrebno pisati u ECMAScript 5 (ES5) budući da preglednici i dalje nisu kompatibilni sa ECMAScript 6 (ES6) (Zaytsev, 2018). ECMAScript je standard za skriptni jezik, koji specificira osnovne značajke koje skriptni jezik treba pružiti i kako ih treba implementirati.

Slika 6. Početak rada ubacivanjem skripte u React.js-u



Izvor: obrada autora

Slika iznad (Slika 6.) prikazuje početak rada ubacivanjem skripte u React-u. Kreirana je varijabla name i novi element koji prima neke od argumenata. Prvi argument je onaj u koji će biti nešto upisano, drugi su propsi koji su ovdje null i koji će biti objašnjeni kasnije, i treći je nekakav podatak. Na samom kraju DOM je renderiran sa tim elementom. U ovom slučaju React će raditi bez JSX-a i bez Babela ili nekog sličnog prevodioca. Babel je alat koji omogućuje prevođenje jedne verzije koda u drugu, onu podržanu od strane preglednika.

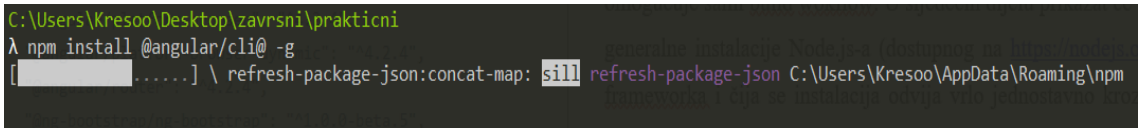
I Typescript i ES6 mogu biti kompajlirani u pregledniku s određenim paketima, što omogućuje slanje nekompajliranog koda pregledniku. U tom slučaju dolazi do slanja puno nepotrebnog koda, točnije, cijelog kompajlera. Osim toga kompajliranje bi se izvršavalo pri inicijalnom učitavanju aplikacije, što bi rezultiralo lošim korisničkim iskustvom. Zbog navedenih razloga, ova se opcija neće razmatrati (Angular University, 2016).

4.1.2. Radni dijagram

Za kreiranje jednostavnijih komponenti, bilo bi poželjno koristiti se jednostavnim ubacivanjem skripte. S vremenom, kada se dodaje više funkcija kao, primjerice, klijentski usmjerivač, upravljač stanjem i http modul, ipak, je potreban radni dijagram koji će sve to, na sveobuhvatan način, povezati.

Kod Angulara je radni dijagram zaista potreban. Jedan od razloga zašto je potreban je taj što, kako je ranije navedeno, Typescript kao jezik ne razumiju preglednici te ga je potrebno kompajlirati. Osim toga, potrebno je skupiti sve te fajlove i potrebno je puno optimizirati, a to sve omogućuje sami radni dijagram za izgradnju. U sljedećih dijelu prikazat će se postavljanje radnog dijagrama. Nakon generalne instalacije Node.js-a (dostupnog na https://nodejs.org/en/), koji je potreban za sva tri FW-a i čija se instalacija odvija vrlo jednostavno kroz nekoliko klikova i zbog istog se neće prikazati u radu, potrebno je instalirati Angular CLI (eng. *Command Line Interface*) globalno naredbom „npm install @angular/cli -g“ kao na slici ispod (Slika 7.).

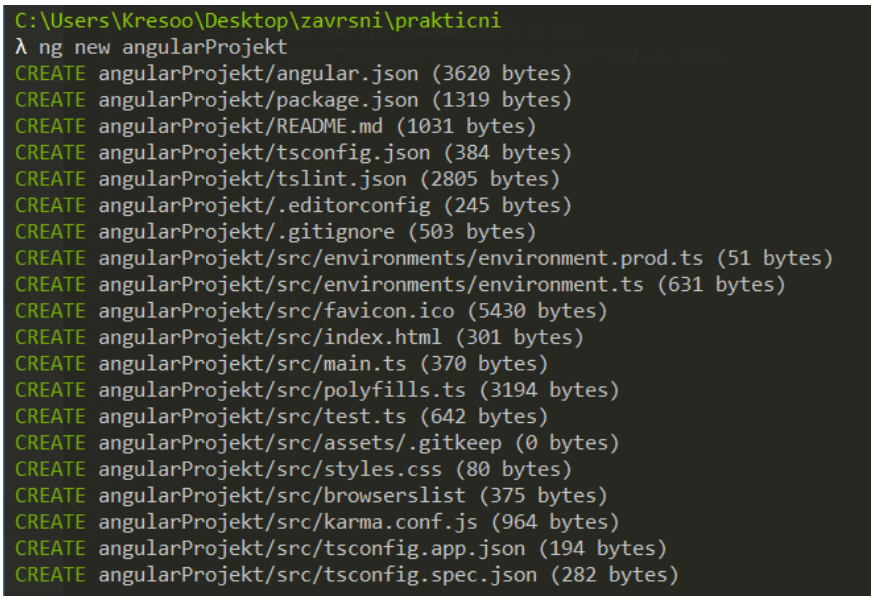
Slika 7. Angular CLI instalacija



Izvor: obrada autora

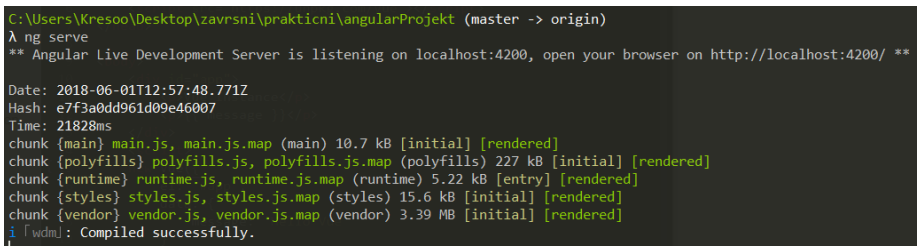
Nakon toga, naredbom „ng new imeProjekta“ kreira se novi projekt (Slika 8.). Na kraju, kako bi testiranje i izrada aplikacije bilo uspješno, potrebno je pokrenuti razvojni poslužitelj koji će ugostiti aplikaciju u lokalnom okruženju naredbom „ng serve“ (Slika 9.).

Slika 8. Stvaranje novog Angular projekta



Izvor: obrada autora

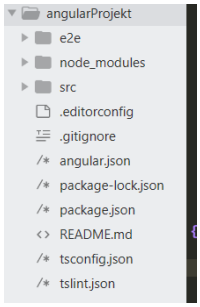
Slika 9. Pokretanje Angular razvojnog poslužitelja



Izvor: obrada autora

Struktura radnog dijagrama izgleda kao na slici ispod (Slika 10.).

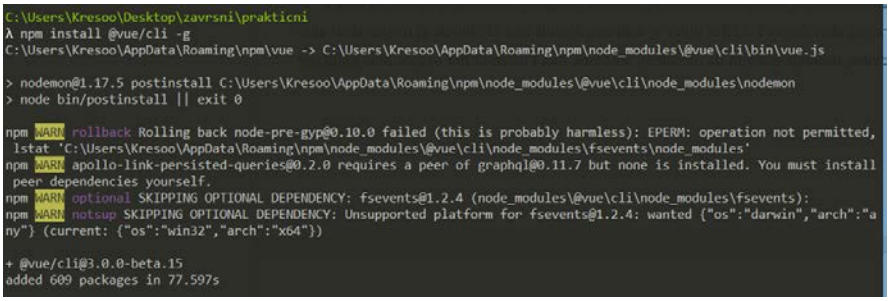
Slika 10. Struktura Angular radnog dijagrama



Izvor: obrada autora

Vue.js ne stavlja naglasak na radni dijagram i on nije potreban. Omogućen je jednostavan početak rada dodavanjem js skripte. U tom slučaju potrebno je raditi u ES5. Početak rada postavljanjem radnog dijagrama bi mogao biti koristan i dati određene prednosti, ali nije u potpunosti potreban (Vue, 2018-3). Za početak rada sa radnim dijagramom, prvo je potrebno globalno instalirati Vue CLI. To se izvršava naredom „npm install -g @vue/cli“ (Slika 11.).

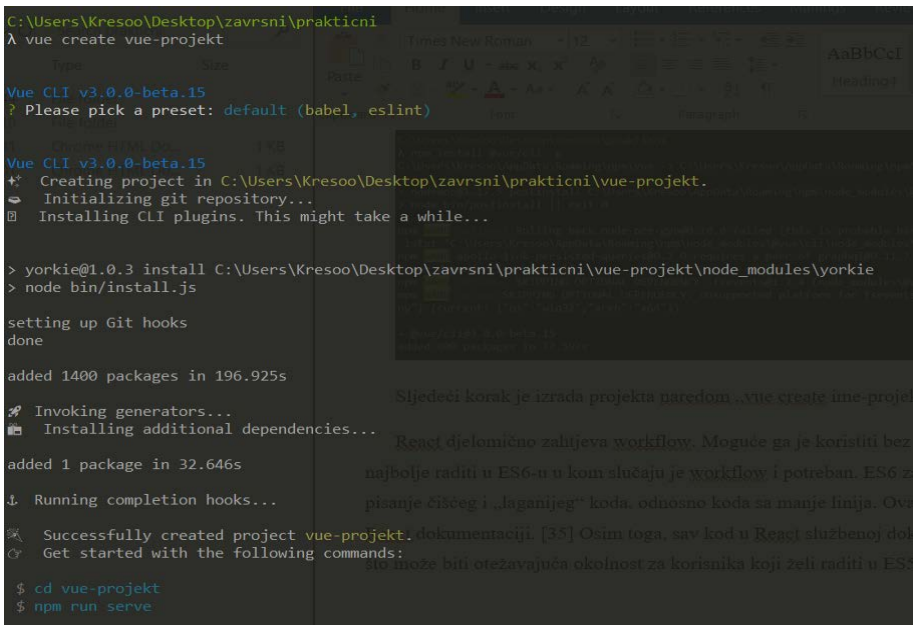
Slika 11. Vue CLI instalacija



Izvor: obrada autora

Sljedeći korak je izrada projekta naredbom „vue create ime-projekta“ (Slika 12.).

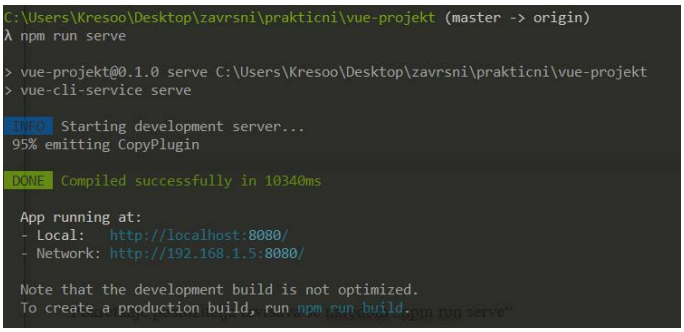
Slika 12. Stvaranje projekta u Vue.js-u



Izvor: obrada autora

Pokretanje poslužitelja izvršava se naredom „npm run serve“, a aplikacija će biti poslužena na zadanom 8080 portu (Slika 13.).

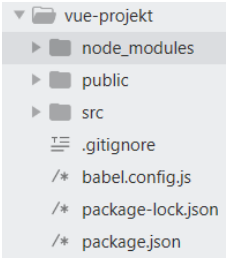
Slika 13. Vue.js pokretanje razvojnog poslužitelja



Izvor: obrada autora

Struktura radnog dijagrama izgleda kao na slici ispod (Slika 14.).

Slika 14. Struktura radnog dijagrama

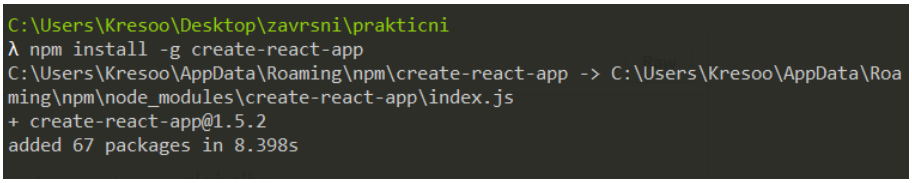


Izvor: obrada autora

React djelomično zahtjeva radni dijagram. Moguće ga je koristiti bez toga, ali sa React-om je najbolje raditi u ES6-u u kom slučaju je radni dijagram zaista potreban. ES6 za razliku od ES5 omogućuje pisanje čišćeg i „laganijeg“ koda, odnosno koda sa manje linija. Ovaj dio

dokazan je i u službenoj React dokumentaciji (React, 2018-4). Osim toga, sav kod u React službenoj dokumentaciji napisan je u ES6, što može biti otežavajuća okolnost za korisnika koji želi raditi u ES5. Prije kreiranja radnog dijagrama potrebno je naredbom „npm install -g create-react-app“ globalno instalirati React CLI (Slika 15.).

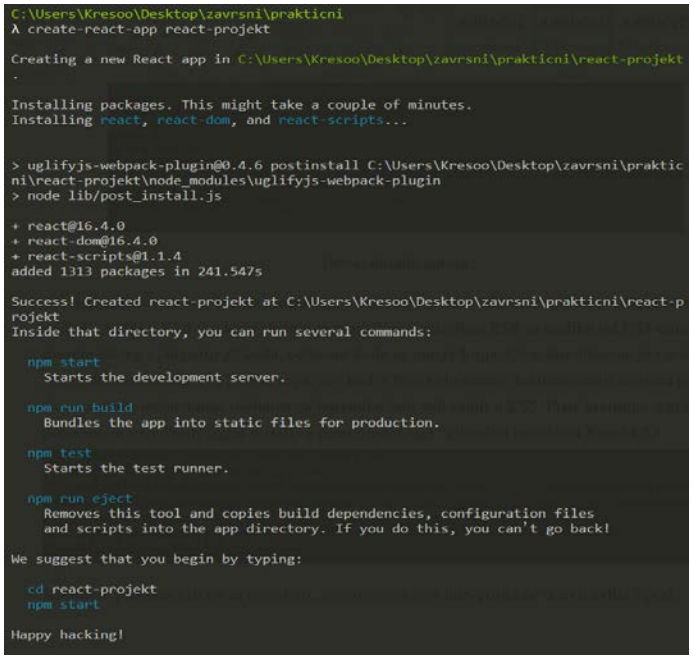
Slika 15. React.js globalna instalacija



Izvor: obrada autora

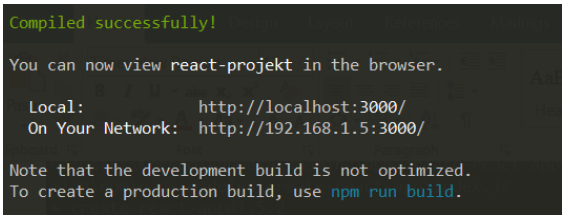
Kreiranje projekta odvija se naredom „create-react-app ime-projekta“ (Slika 16.), a pokretanje razvojnog poslužitelja naredom „npm start“ (Slika 17.).

Slika 16. React.js stvaranje projekta



Izvor: obrada autora

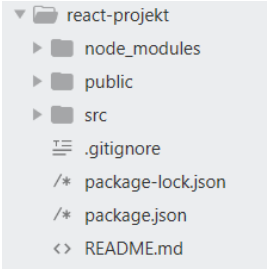
Slika 17. React.js pokretanje razvojnog poslužitelja



Izvor: obrada autora

Radni dijagram izgleda kao na slici ispod (Slika 18.).

Slika 18. React.js radni dijagram



Izvor: obrada autora

4.2. Povećanje brzine razvoja i usklađenosti s drugim komponentama

Korištenje modula također pomaže kod ostvarivanja integriteta podataka, organizacije koda i izbjegavanja ponavljajućih naziva. Bez ovakvog pristupa, posebice, u izgradnji SPA rad sa, prije svega, globalnim varijablama i funkcijama bi ubrzo postao neupravljiv (Takada, 2013)(Bruce, 2016). Kako bi ovo bilo izvedivo vrlo je bitno da postoji najbolja praksa napisana u

dokumentaciji u vezi s imenovanjem datoteka i direktorija jer će jedino na taj način aplikacija moći biti održiva. Također, u svrhu dodatnog povećanja brzine razvoja aplikacije bilo bi dobro da FEFW ima mogućnost automatskog generiranja pojedinih dijelova.

4.2.1. Konvencija pisanja naziva datoteka i direktorija

Kada se govori o SPA podrazumijeva se aplikacija koja sadrži jako puno koda i, vjerojatno, jako puno datoteka koje sadrže sav taj kod. Kako bi sve to bilo održivo vrlo je bitno da postoji i objavljena najbolja praksa imenovanja datoteka i direktorija. Kako bi sve to bilo održivo vrlo je bitno da postoji objavljena najbolja praksa imenovanja datoteka i direktorija.

Angular omogućuje pisanje i održavanje puno kompleksnog koda. Postoje mnoge najbolje prakse kako bi trebali imenovati datoteke te kako ih treba strukturirati i sve je to objavljeno na Angular-ovoj službenoj stranici u posebnoj rubrici styleguide. Ova rubrika sadrži osmišljen vodič za Angular sintaksu, konvencije i strukturu aplikacije, s detaljnim objašnjenjima svrhe. (Angular, 2018-4).

Kod Vue.js-a nema jasne najbolje prakse. Ako se prati pristup za registriranje komponenata, prikazan u službenoj dokumentaciji, vidljivo je da su dopušteni kebab-case, camelCase ili PascalCase nazivi, što može otežati održavanje većih aplikacija (Vue, 2018-6). To ne znači da nije izvedivo i da korisnik ne može koristiti jedan specificiran pristup, nego nije eksplicitno navedeno i korisniku je omogućen odabir. To zahtjeva promišljanje u početku razvoja.

Kod Reacta je ista situacija kao kod Vue.js-a. Najbolje prakse su u zajednici što znači da je na developeru da ih nauči i da pronađe one koje odgovaraju njegovim potrebama. Nisu striktno navedene kao što je to kod Angulara, što može biti problem (Abranov, 2016).

4.2.2. Mogućnost automatskog generiranja dijelova aplikacije

Kako bi se povećala brzina izgradnje same aplikacija neki FW-i često imaju mogućnost automatskog generiranja aplikacije. Ipak, kada se govori o frontend FW-ima ova funkcionalnost je dosta rjeđa. Sljedeći tekst će pokazati koji to od pojedinačnih Javascript FW-a imaju, a koji nemaju ovu funkcionalnost.

Angular dolazi sa radnim dijagramom i CLI-em koji omogućuje postavljanje radnog dijagrama, procesa za izgradnju (objedinjavanjem svega) na vrlo jednostavan način, ispisivanjem naredbi u konzolu (Angular, 2018-4). Sa Angular CLI -em, osim kreiranja radnog dijagrama, moguće je i kreirati i ostale sastavne dijelove projekta kao što su nova komponenta, smjernica, servis, klasa, modul i još mnogo toga. Naredba je „ng generate component onoStoSeZeliKreirati naziv“. U nastavku će biti prikazan najčešći slučaj korištenja ove naredbe, a to je stvaranje nove komponente. Nakon početnog kreiranja projekta, kreirana je nova komponenta „home“ (Slika 19.).

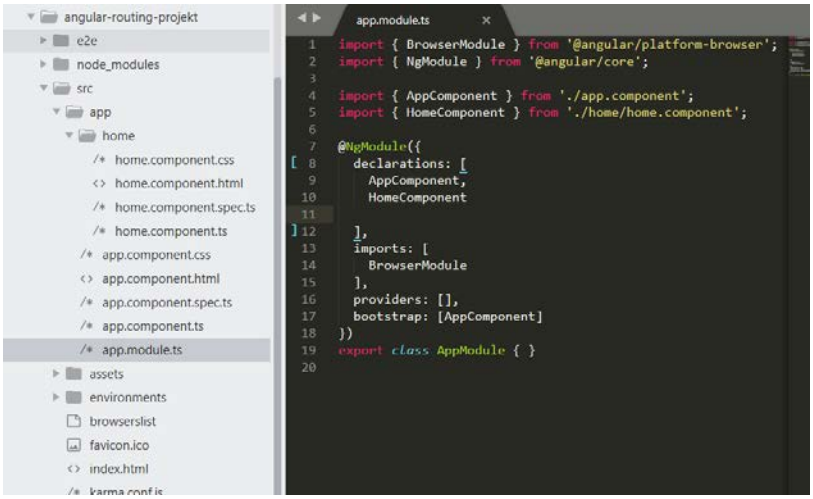
Slika 19. Angular kreiranje nove komponente

```
C:\Users\Kresoo\Desktop\zavrsni\prakticni\angular\angular-routing-projekt (master -> origin)
λ ng generate component home
CREATE src/app/home/home.component.html (23 bytes)
CREATE src/app/home/home.component.spec.ts (614 bytes)
CREATE src/app/home/home.component.ts (261 bytes)
CREATE src/app/home/home.component.css (0 bytes)
UPDATE src/app/app.module.ts (390 bytes)
```

Izvor: obrada autora

Automatski je generirana nova komponenta unutar radnog src/app direktorija. Također, komponenta je automatski registrirana u glavnu app.module.ts datoteku što je vidljivo na slici ispod (Slika 20.).

Slika 20. Glavna app.module.ts datoteka



Izvor: obrada autora

Što se tiče radnog dijagrama, Vue.js, kao i Angular, omogućuje njegovo stvaranje kroz CLI, ali ne pruža mogućnost generiranja pojedinih zasebnih dijelova aplikacije. To je prikazano ranije u poglavlju radni dijagram.

React omogućuje stvaranje radnog dijagrama od šestog mjeseca 2016. godine kroz službeni create-react-app paket, no, kao i Vue, nema opcije automatskog generiranja pojedinih dijelova aplikacije (Abranov, 2016).

4.3. Korištenje dodatnih modula

Način korištenja dodatnih modula, posebice onih bitnih u razvoju SPA, jedan je od ključnih faktora. Ovo poglavlje odnosi se na to da li je FEFW kompletan tj. da li su svi moduli, potrebni za izgradnju SPA, već ugrađeni u njemu ili postoji službeno izdan paket, ili se ipak potrebno osloniti na zajednicu kako bi dobili prikladne pakete, što bi onda mogao biti problem (npr. ne možemo biti sigurni da li paket ostaje ažuriran). Vjerojatnije je da će softver izgrađen od strane

neke veće firme biti pregledavan na redovitijoj bazi, jer, ipak, više ljudi koristi softver, a i ako se problem otkrije, to će biti riješeno puno brže (Wagner, 2014). Ovo je vrlo bitno kada se govori o izgradnji SPA aplikacija kod koje će, budući da se radi o većoj i kompleksnijoj Javascript aplikaciji, svi ovi paketi biti zaista potrebni, a činjenica je da bi trebalo postojati što manje brige o pronalaženju drugih paketa za nedostajuće značajke. To je samo više ovisnosti o kojima treba brinuti i više potencijalnih točaka neuspjeha (Köhr, Schlaudraff, 2017).

S druge strane, odabir određenog FW-a često za sobom može povlačiti velik broj već ugrađenih dodataka (paketa) sadržanih u njemu samome, tako da se ovo poglavlje odnosi i na mogućnost FW-a da developeru pruži viši stupanj kontroliranja veličine aplikacije tj. FEFW, odabirom samo modula koji su uistinu potrebni. Ukoliko je cilj izgradnja jednostavnije komponente tj. kontrola samo sitnog dijela DOM-a, kompletan FEFW sa svim modulima s kojima dolazi apsolutno nije potreban, jer bi u tom slučaju to moglo rezultirati problemom kod optimizacije (Neuhaus, 2017).

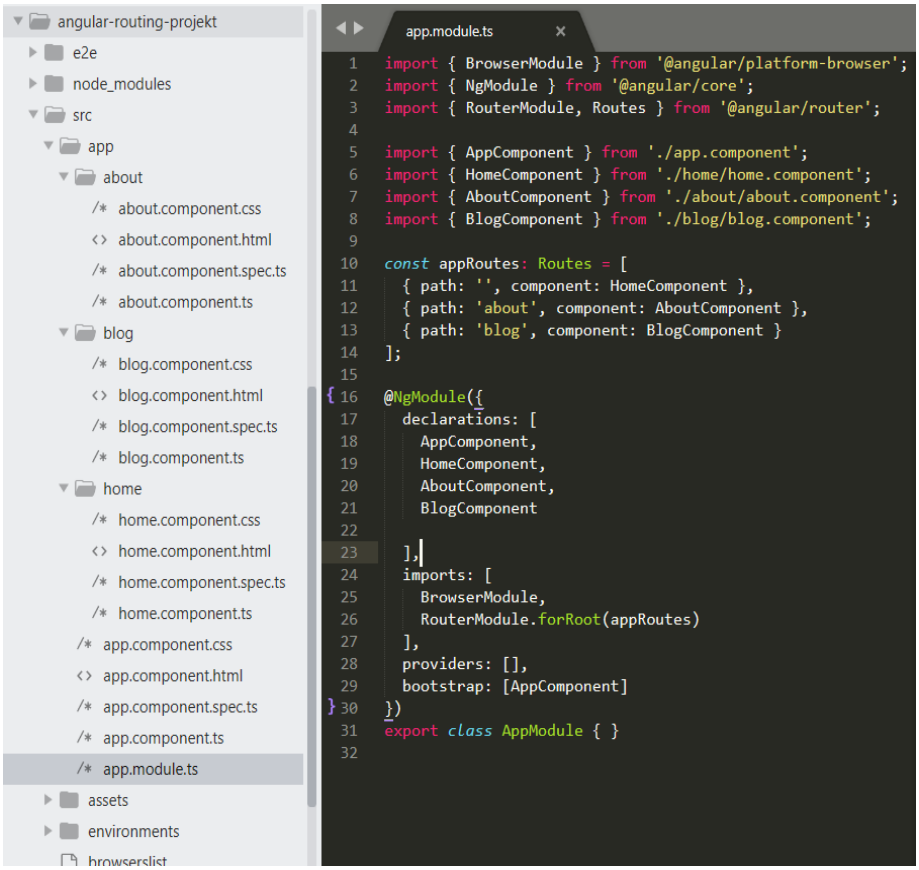
U ovom radu naglasak će staviti na osnovne module koje nemaju službeno izdane svaki od pojedinačnih FW-a. To su modul za klijentsko usmjeravanje (eng. *Client-side routing*), modul za validaciju formi i modul za stilove.

4.3.1. Klijentski usmjerivač

Usmjerivač (eng. *Routing*) je u suštini način na koji se korisnik kreće kroz web stranicu ili web aplikaciju. Klikom na vezu mijenjaju se URL-ovi koji korisnicima pružaju neke nove podatke ili novu web stranicu. Promjena rute kod klijentskog usmjerivanja odvija se interno Javascriptom. Kada korisnik klikne na vezu, URL se promjeni, ali je zahtjev ka poslužitelju spriječen (Schepenaar, 2017).

Angular ovaj modul već ima ugrađenog u njemu samome. Nije potrebna nikakva instalacija i dovoljno je samo uvesti ono što je potrebno iz router modula. Router modul je opcionalni servis koji omogućuje prikaz specifične komponente na promjeni URL-a. Nije dio Angular jezgre, ali je službeno izdan (Angular, 2018-7). Nakon kreiranja komponenata na način kako je gore prikazano i na koje će se usmjeravati, potrebno je u app.module.ts datoteci uvesti RouterModule i Routes. RouterModule daje korisniku funkcionalnosti usmjeravanja, dok je Routes tip podatka koji će imati niz ruta. Sada kod u app.module.ts izgleda kao na slici ispod (Slika 21.).

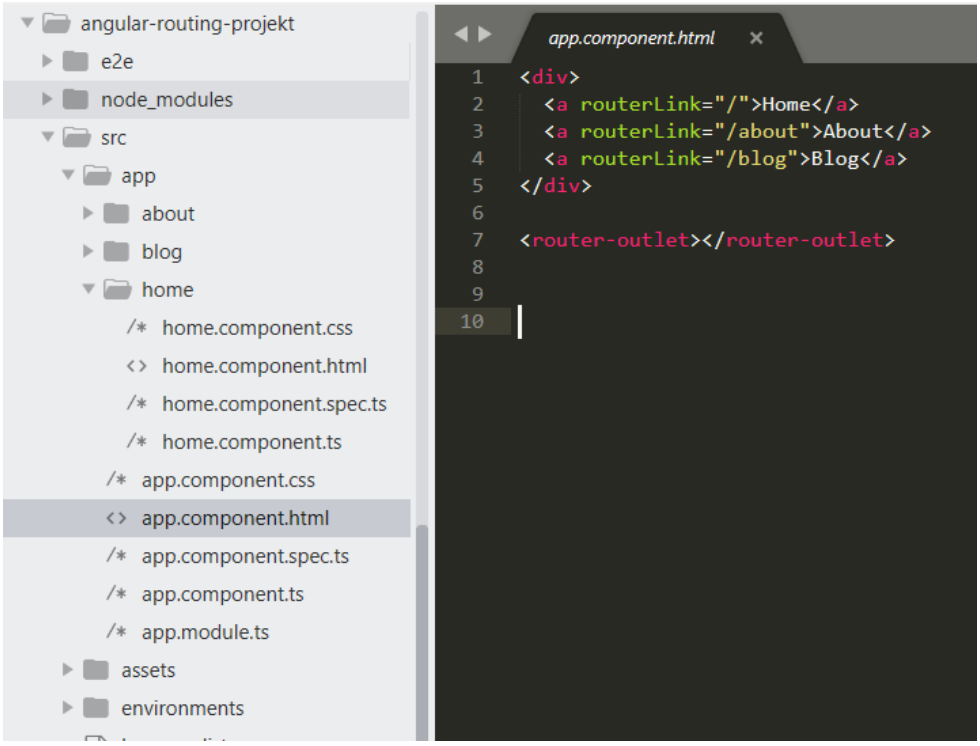
Slika 21. Kreiranje putanja i uvođenje objekata iz Angular usmjerivača



Izvor: obrada autora

Uvedeni su minimalni dijelovi router modula potrebni za usmjeravanje, kreiran je niz appRoutes u kojem su navedene putanja i komponenta na koju vodi ta putanja. Također, u imports je uveden RouterModule sa metodom forRoutes() te mu je u konfiguraciji dodijeljen ranije kreiran niz. Nakon toga, potrebno je u glavnoj HTML datoteci dodati router-outlet atributnu smjernicu i linkove koji će voditi na pojedinačne rute (Slika 22.).

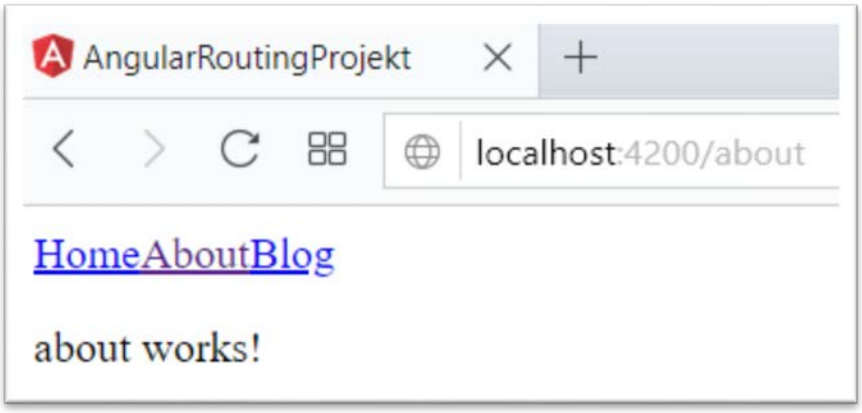
Slika 22. Dodavanje linkova u Angular predložak



Izvor: obrada autora

Atributne smjernice mijenjaju izgled ili ponašanje elementa, komponente ili druge smjernice. Ova smjernica pokazuje gdje će se prikazati trenutno učitana ruta. Linkovi su obične <a> oznake ali bez href atributa. Umjesto toga Angular pruža routerLink smjernicu koja omogućuje usmjeravanje na klijentskoj strani. Aplikacija nakon pokretanja razvojnog poslužitelja izgleda kao na slici ispod (Slika 23.).

Slika 23. Angular aplikacija za klijentsko umjeravanje

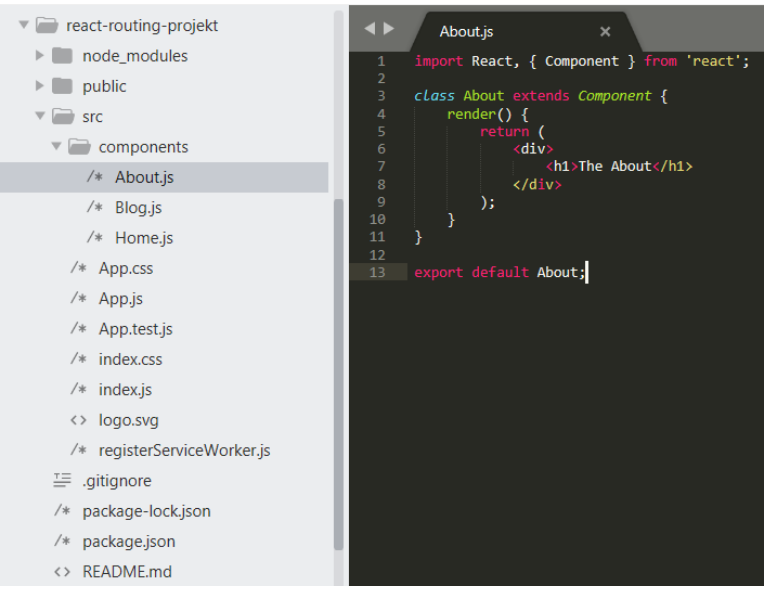


Izvor: obrada autora

Vidljivo je da je promjenom putanje na /about učitana About komponenta koja prikazuje tekst „about works“.

React.js nema službeno izdan ovaj modul. Ipak, postoji paket razvijen od strane zajednice koji je vrlo popularan i ima preko 30000 Github zvjezdica. Prije samog početka potrebno je kreirati nekoliko novih komponenti. React nema mogućnost kreiranja automatskim generiranjem kao što je to slučaj kod Angulara, no sami proces kreiranja je jednostavniji. Dovoljno je kreirati novu Javascript datoteku i uvesti objekte React i Component iz osnovog React modula, te nakon toga ispisati novu klasu ili funkciju (Slika 24.).

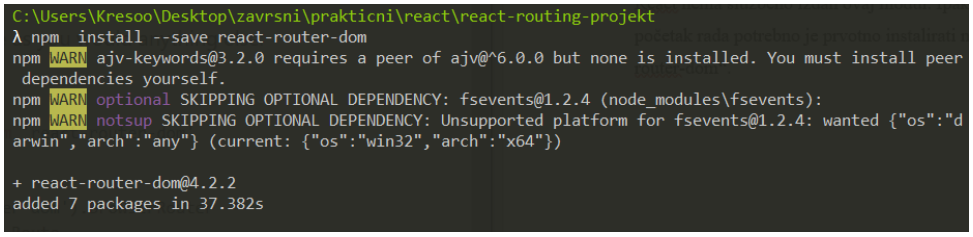
Slika 24. React.js komponenta na koju će se usmjeravati



Izvor: obrada autora

Prije korištenja klijentskog usmjerivača potrebno je prvotno instalirati novi modul naredbom „npm install –save react-router-dom“ (Slika 25.).

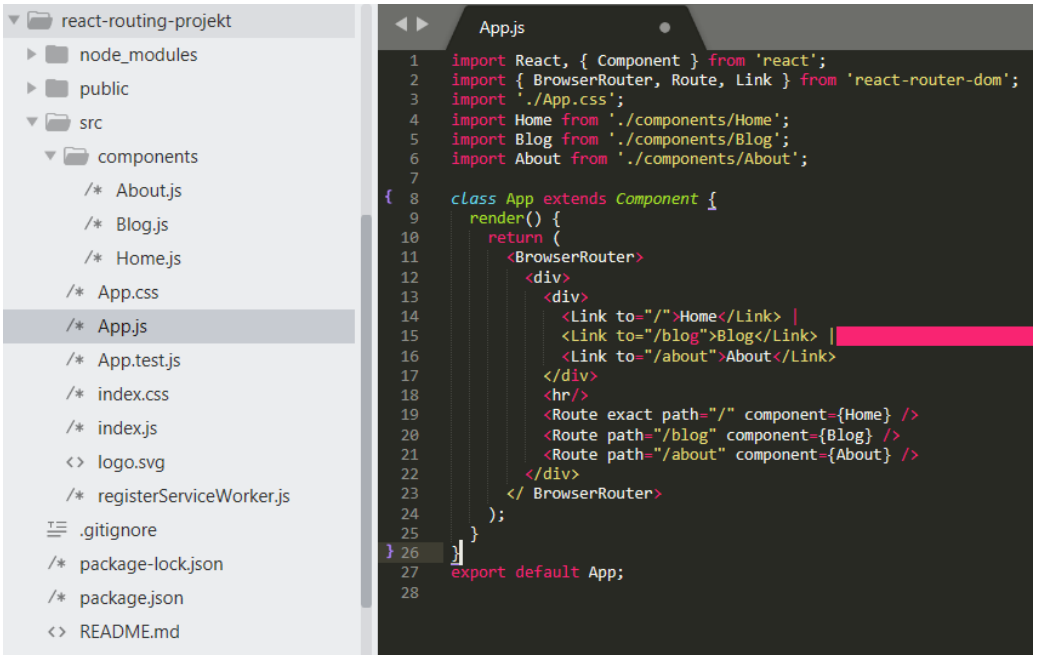
Slika 25. React.js instalacija klijentskog usmjerivača



Izvor: obrada autora

Sljedeći korak je uvođenje objekata iz novog modula. Ono što je minimalno potrebno uvesti su BrowserRouter, Route i Link objekte. BrowserRouter je objekt unutar kojeg će živjeti podaci o putanji, Route je objekt unutar kojeg se specificiraju moguće putanje u aplikaciji, a Link je klijentska zamjena za <a> oznaku (Slika 26.).

Slika 26. React.js uvođenje objekata i kreiranje putanja



Izvor: obrada autora

Nakon pokretanja razvojnog poslužitelja naredbom „npm start“ rezultat je prikazan na slici ispod (Slika 27.).

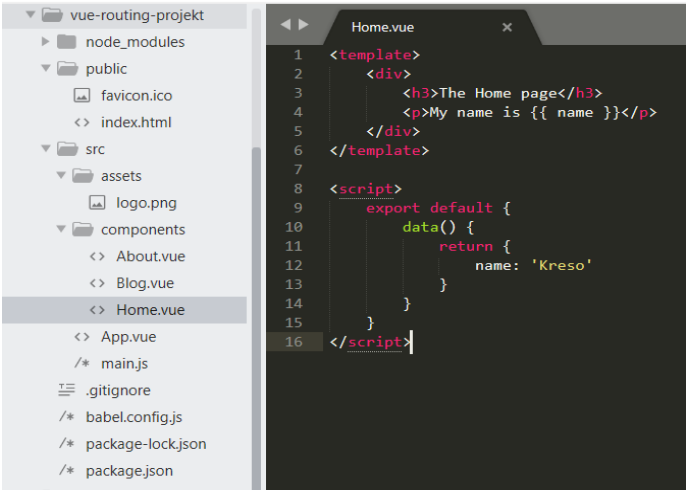
Slika 27. React.js aplikacija za klijentsko usmjeravanje

Izvor: obrada autora

Vidljivo je da je promjena putanje na /about dohvatila About komponentu kako je i specificirano Route objektom.

Usmjerivač kod Vue.js-a je službeno izdan modul. Prije svega kreirane su nove komponente kao na slici ispod. Vidljivo je da Vue svoj predložak ne pruža odvojeno kao što je to kod Angulara, a sva logika se jednostavno nalazi unutar <script> oznaka (Slika 28.).

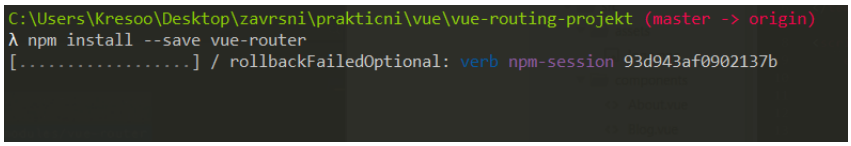
Slika 28. Vue.js komponenta na koju će se usmjeravati



Izvor: obrada autora

Sljedeći korak je instalacija modula naredbom „npm install --save vue-router“ (Slika 29.).

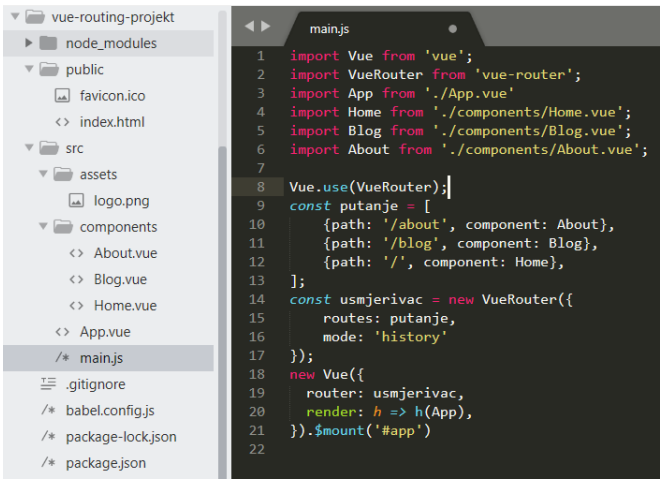
Slika 29. Vue.js instalacija klijentskog usmjerivača



Izvor: obrada autora

Nakon toga, u main.js dokumentu je uveden objekt VueRouter, pozvana je funkcija Vue.use koja omogućuje da Vue.js počne koristiti prethodni objekt. Stvorena je „putanje“ konstanta koja predstavlja niz objekata. U svakom od objekata ispisana je putanja u jednom polju i komponenta na koju vodi putanja u drugom polju. Nakon toga kreirana je nova instanca VueRouter-a naziva „usmjerivac“ koji prima Javascript objekt sa „putanje“ konstantom. Na kraju, u Vue instancu polju naziva „router“ je dodana „usmjerivac“ konstanta (Slika 30.).

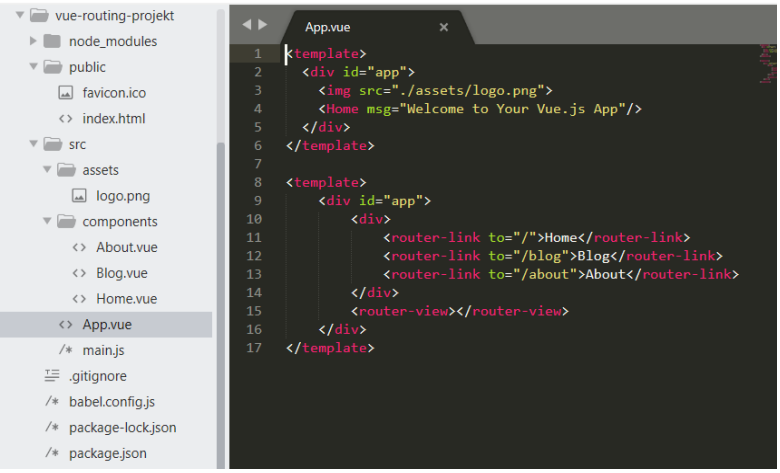
Slika 30. Vue.js uvođenje usmjerivača, kreiranje njegove instance i putanja



Izvor: obrada autora

Nadalje, u App.vue dokumentu, dodana je router-view oznaka koja pokazuje mjesto gdje će Vue.js učitati putanje. Također, unutar router-link komponenata, koje dolaze sa vue-router modulom, specificirani su linkovi na putanje (Slika 31.).

Slika 31. Vue.js linkovi na putanje u predlošku



Izvor: obrada autora

4.3.2. Modul za validaciju formulara

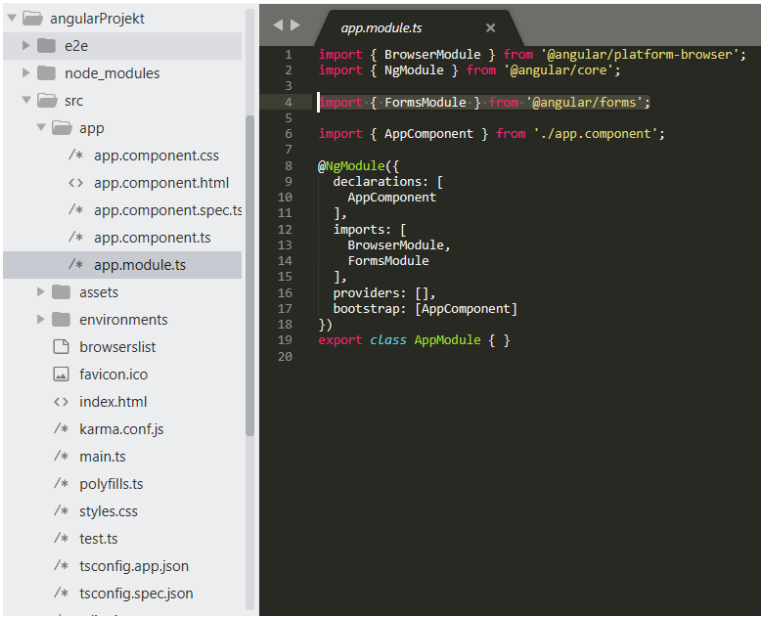
Svaka web stranica mora poticati korisnika da ispuni prikazane formulare na najbolji mogući način ili na zahtjevani način. U takve namjene često se koriste dodatni moduli.

React takav modul službeno izdan nema. Ipak se potrebno osloniti na zajednicu u kojoj često postoji više alternativa što čini težak odabir za nove korisnike. Također, iz razloga što često postoji više alternativa potrebno je određeno iskustvo u odabiru između svih tih ponuđenih modula, često vrlo sličnih namjena (Npmjs, 2013-2017). Alternativno, potrebno je pisati vlastiti kod ili pronaći neki od dostupnih modula izgrađenih od strane zajednice.

Vue.js, također, nema takav modul službeno izdan i to je nešto što nije uključeno u ovaj FW. Dakle, potrebno je koristiti paket treće strane koji, možda, nije gotov ili koji je razvijen od strane jedne osobe, koja, možda, nema dovoljno vremena da ga održi ažuriranog. Ovo bi mogao biti problem, naravno, ovisno o tome koje će biti funkcionalnosti razvijane aplikacije. Ovaj FW, ipak, na službenim stranicama pokazuje kako izvesti jednostavnije provjere valjanosti formulara i preporuča neke knjižnice kao dodatne module (Vue, 2018-9).

Angular ovaj modul apsolutno ima. Prije početka rada potrebno je uvesti FormsModule iz Angular jezgre u app.module.ts datoteku što je vidljivo na slici ispod (Slika 32.).

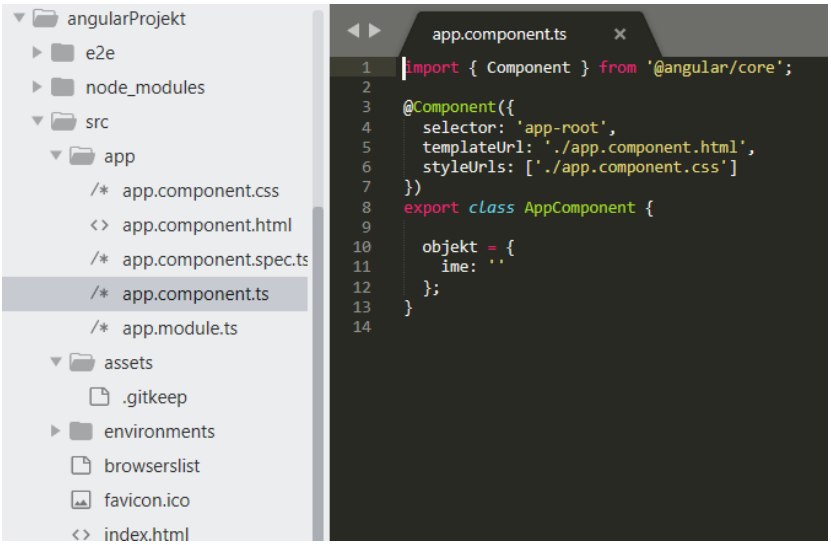
Slika 32. Angular uvođenje modula za formulare



Izvor: obrada autora

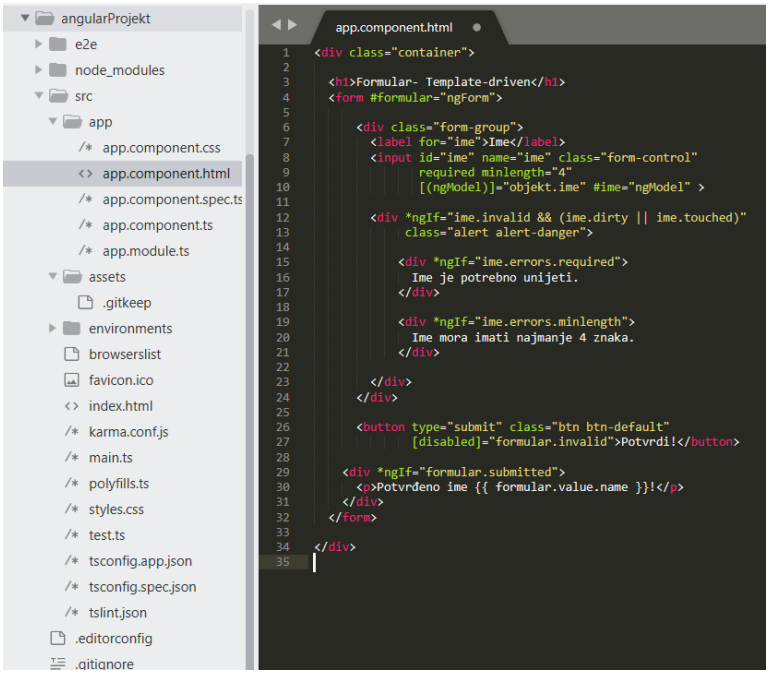
Sljedeći korak je definiranje formulara i validacije u predlošku, te kreiranje objekta u Typescript dokumentu. Na slici ispod (Slika 33.) kreiran je objekt sa poljem ime te mu je inicijalizirana vrijednost na null.

Slika 33. Angular kreiranje objekta



Izvor: obrada autora

Slika 34. Angular kreiranje formulara u predlošku



Izvor: obrada autora

Na slici iznad (Slika 34.) kreiran je formular s jednom <input> oznakom naziva „formular“ te mu je dodijeljena direktiva ngForm koja omogućuje korištenje polja invalid, dirty, touched itd. Invalid polje provjerava da li je polje nevažeće, dok dirty i touched sprečavaju prikazivanje pogrešaka dok korisnik ne učini jednu od dvije stvari:

- promijeni vrijednost inputa, postavljajući kontrolu na dirty;
- ili zamagli (fokusira) element formulara klikom, postavljajući kontrolu na touched.

Također, svakom elementu formulara potrebno je dodati ngModel direktivu koja omogućuje dvosmjerno vezivanje. To znači da se na svaku promjenu vrijednosti u elementu formulara pokreće događaj koji šalje novu vrijednost polju objekta na koji se odnosi. Rezultat je prikazan u sljedećim slikama.

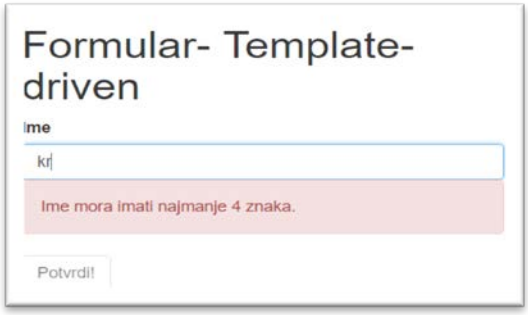
Početno stanje formulara (Slika 35.).

Slika 35. Angular formular aplikacija 1

Izvor: obrada autora

Tek kada korisnik klikne na element formulara i počne pisati, pojavit će mu se poruka da mora unijeti minimalno 4 znaka (Slika 36.). Ta poruka će nestati tek kad se zadovolji kriterij.

Slika 36. Angular formular aplikacija 2



Izvor: obrada autora

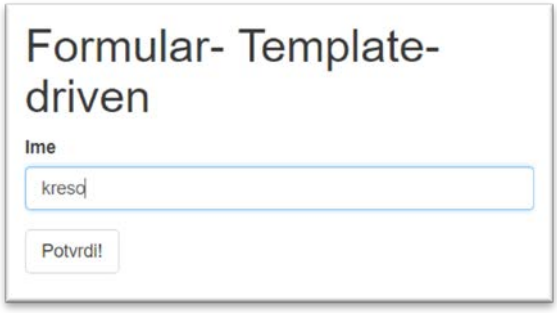
Ukoliko korisnik obriše poruku pojavit će mu se poruka da je ime potrebno unijeti (Slika 37.).

Slika 37. Angular formular aplikacija 3

Izvor: obrada autora

Također, gumb potvrde će biti isključen sve dok postoji ijedan kriterij koji nije zadovoljen (Slika 38.).

Slika 38. Angular formular aplikacija 4



Izvor: obrada autora

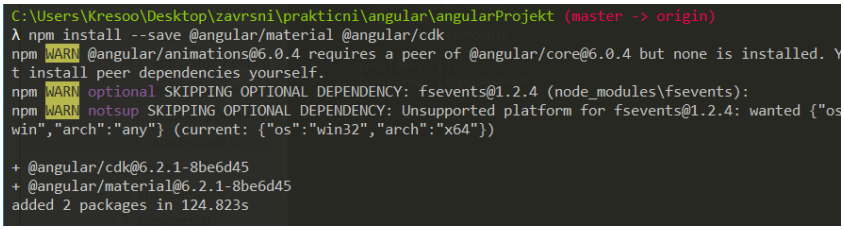
Vidljivo je da ovaj modul znatno olakšava validaciju formulara.

4.3.3. Modul za dizajnerske komponente

Iako nije neophodan, modul za dizajnerske komponente može znatno pomoći ubrzati sami proces izrade aplikacije. Radi se o gotovim komponentama koje je moguće koristiti i konfigurirati prema vlastitim zamislima.

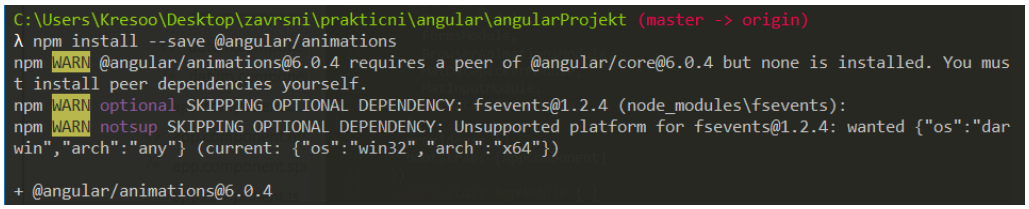
Angular ovaj modul ima službeno, a za početak rada dovoljno je instalirati material modul naredom „npm install --save angular/material2-builds angular/cdk-builds“ (Slika 39.). Kako određene komponente zahtjevaju i animacijski dio, potrebno je i instalirati angular/animations naredbom „npm install --save @angular/animations“ (Slika 40.).

Slika 39. Angular instalacija modula za dizajnerske komponente 1



Izvor: obrada autora

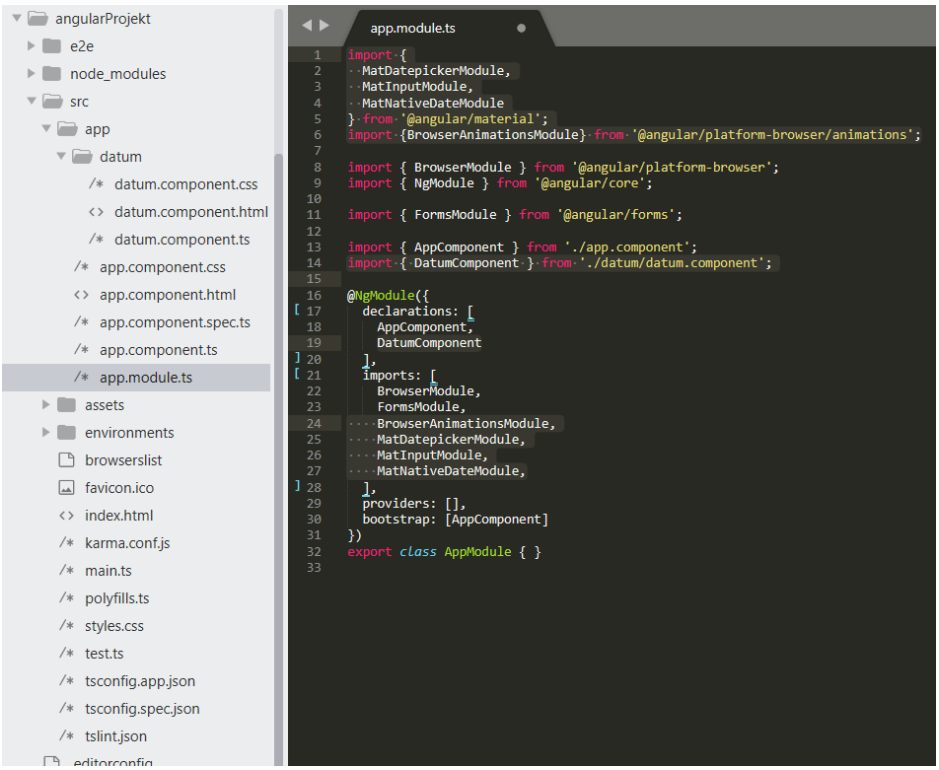
Slika 40. Angular instalacija modula za dizajnerske komponente 2



Izvor: obrada autora

Nakon toga, potrebno je iz material modula i animation modula uvesti one komponente koje su potrebne za specifičnu funkcionalnost. Budući da se u ovom radu prikazuje funkcionalnost uvođenja gotove komponente datuma potrebno je uvesti komponente kao na slici ispod (Slika 41.).

Slika 41. Angular uvođenje komponente iz modula za dizajnerske komponente



Izvor: obrada autora

Sljedeći korak je ispisivanje komponente sa željenim konfiguracijski postavkama u predlošku (Slika 42.).

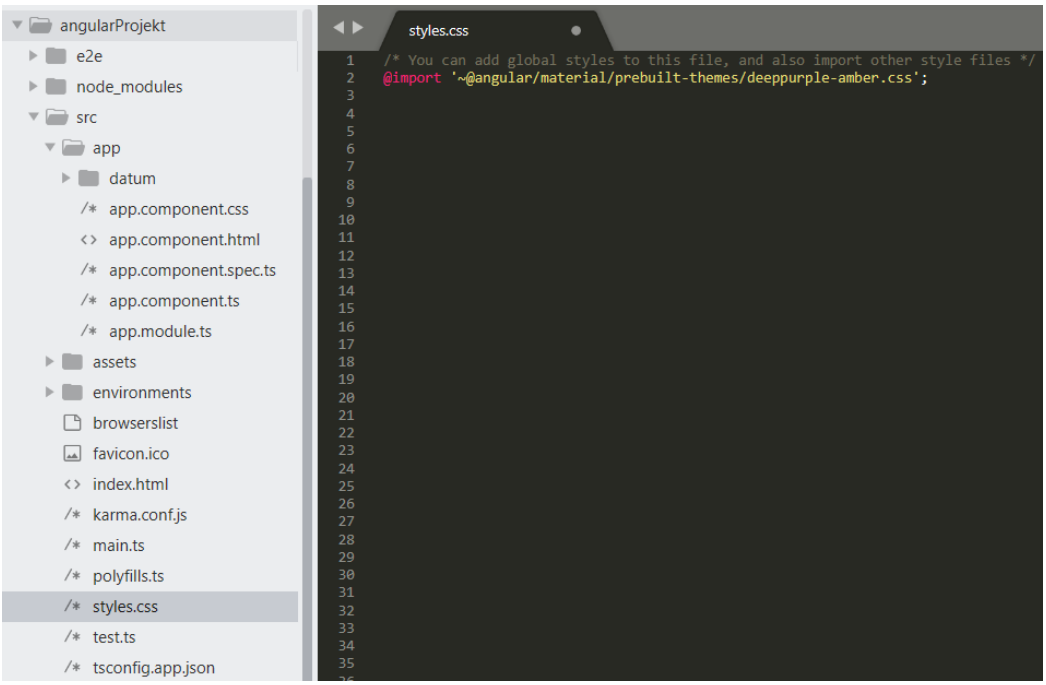
Slika 42. Angular ispisivanje komponente datuma u predlošku

Izvor: obrada autora

Na kraju, moguće je dodati i predefinirane teme koje Angular nudi.

Za ovu funkcionalnost dovoljno je glavni dokument za stilove uvesti „@import '~@angular/material/prebuilt-themes/naziv-teme.css';“ kao na slici ispod (Slika 43.).

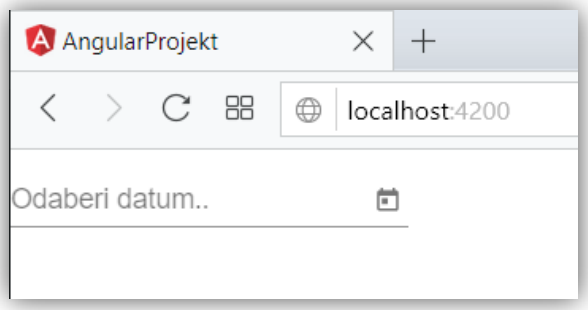
Slika 43. Angular mogućnost definiranja teme za stilove



Izvor: obrada autora

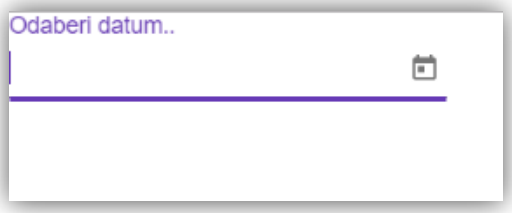
Nakon pokretanja poslužitelja rezultat izgleda kao na slikama ispod (Slika 44., Slika 45., Slika 46., Slika 47.).

Slika 44. Angular aplikacija za datum 1



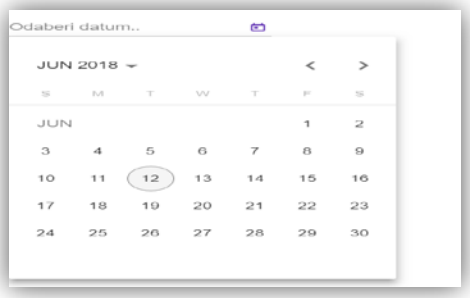
Izvor: obrada autora

Slika 45. Angular aplikacija za datum 2



Izvor: obrada autora

Slika 46. Angular aplikacija za datum 3



Izvor: obrada autora

Slika 47. Angular aplikacija za datum 4

Izvor: obrada autora

React.js i Vue.js službeno izdan modul za dizajerske komponente nemaju, iako postoji jako puno npm paketa u zajednici izgrađenih od strane trećih osoba.

4.4. Distribuiranje izvođenja prikaznih elemenata sa poslužitelja

Ukoliko je cilj izgradnja Multi-Page aplikacije, jedan od razloga zasigurno može biti i bolja optimiziranost aplikacije na tražilicama. SSR (eng. *Server-side rendering*) odnosi se na pokretanje aplikacije na poslužitelju što omogućuje kravlerima da vide sadržaj. Kravler (eng. *crawler*) je vrsta robota koja automatski pretražuje Web u svrhu indeksiranja na web tražilicama. Ovo je bitan zahtjev ako je cilj web aplikacije visoka optimiziranost na tražilicama. Budući da MPA može biti sastavljena od više manjih SPA, dobro je da FW ima i ovu opciju koja će omogućiti renderiranje svake od tih SPA na poslužitelju (React Community, 2016).

4.4.1. Službena podržanost

Ovaj dio pokazat će koji od FW-a imaju, a koji nemaju službenu podržanost modula za izvođenje prikaznih elemenata na poslužitelju. Na praktičan način prikazat će se uvođenje i konfiguracija postavki potrebna za početak rada.

Angular ima službeno izdan vodič za server-side rendering Angular Universal. To je središnji dio koji povezuje Node.js s Angular-om na način da pokreće istu aplikaciju na poslužitelju i u pregledniku. On generira statične stranice aplikacije na poslužitelju putem SSR procesa. Može generirati i posluživati te stranice kao odgovor na zahtjeve preglednika. Također, može unaprijed generirati stranice kao HTML datoteke koje se kasnije poslužuju (Angular, 2018-3). Prije samog početka potrebno je instalirati nekoliko paketa naredbom “npm install --save @angular/platform-server @nguniversal/module-map-ngfactory-loader ts-loader @nguniversal/express-engine” (Slika 48.).

Slika 48. Angular instalacija modula za SSR

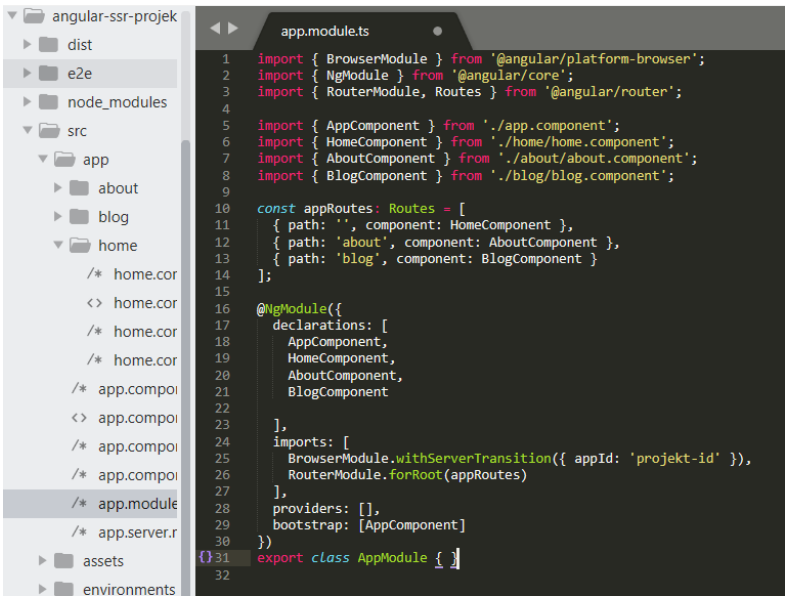
```
C:\Users\Kresoo\Desktop\zavrsni\prakticni\angular\angular-ssr-projekt (master -> origin)
λ npm install --save @angular/platform-server @nguniversal/module-map-ngfactory-loader ts-loader @nguniv
ersal/express-engine
npm WARN rollback Rolling back readable-stream@2.3.6 failed (this is probably harmless): EPERM: operatio
n not permitted, scandir 'C:\Users\Kresoo\Desktop\zavrsni\prakticni\angular\angular-ssr-projekt\node_mod
ules\fsevents\node_modules'
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@1.2.4 (node_modules\fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@1.2.4: wanted {"os":"dar
win","arch":"any"} (current: {"os":"win32","arch":"x64"})

+ @angular/platform-server@6.0.5
+ @nguniversal/module-map-ngfactory-loader@6.0.0
+ @nguniversal/express-engine@6.0.0
+ ts-loader@4.4.1
added 7 packages in 38.001s
```

Izvor: obrada autora

U app.module.ts potrebno je dodati withServerTransition metodu sa argumentom appId. Angular time dodaje odgovarajuću vrijednost na nazive stila stranica prikazanih na poslužitelju, tako da se mogu identificirati i ukloniti kada započne aplikacija klijenta (Slika 49.).

Slika 49. Angular dodavanje withServerTransition metode



Izvor: obrada autora

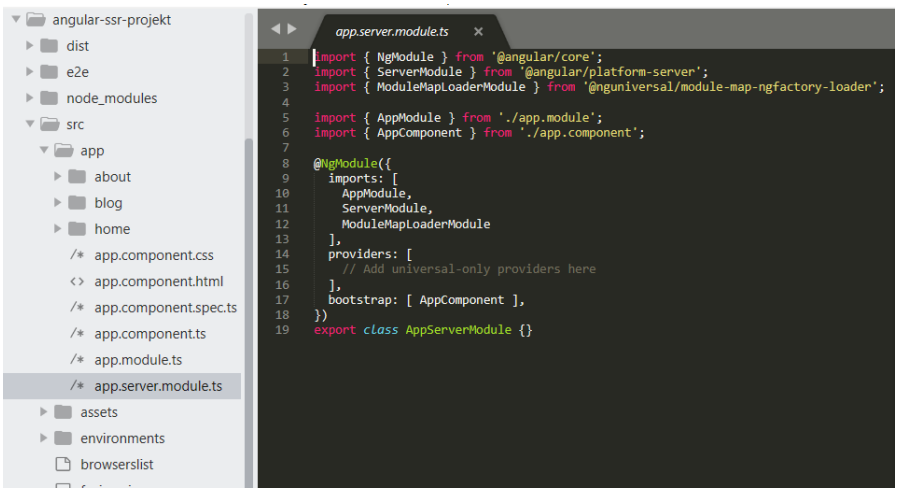
Angular univerzalna aplikacija raspoređena je u dva dijela. To su kod na strani poslužitelja koji služi početnoj aplikaciji i kod na strani klijenta koji se dinamički učitava. Angular prema zadanim postavkama izdaje kod na strani klijenta u dist direktoriju, tako da je potrebno izmijeniti outputPath u angular.json kako bi zadržali izlazne rezultate na strani klijenta odvojeno od koda na strani poslužitelja. To je prikazano na slici ispod (Slika 50.).

Slika 50. Angular izmijena outputPath u angular.json

Izvor: obrada autora

Sljedeći korak je kreiranje poslužiteljskog modula kao na slici ispod (Slika 51.). To je Angular modul koji "omotava" korijenski modul aplikacije (AppModule) tako da Angular Universal može posredovati između aplikacije i poslužitelja. Potrebno je redom uvesti modul aplikacije klijenta (AppModule), poslužiteljski modul (ServerModule) i ModuleMapLoaderModule koji omogućuje lagano učitavanje ruta.

Slika 51. Angular kreiranje poslužiteljskog modula za SSR



Izvor: obrada autora

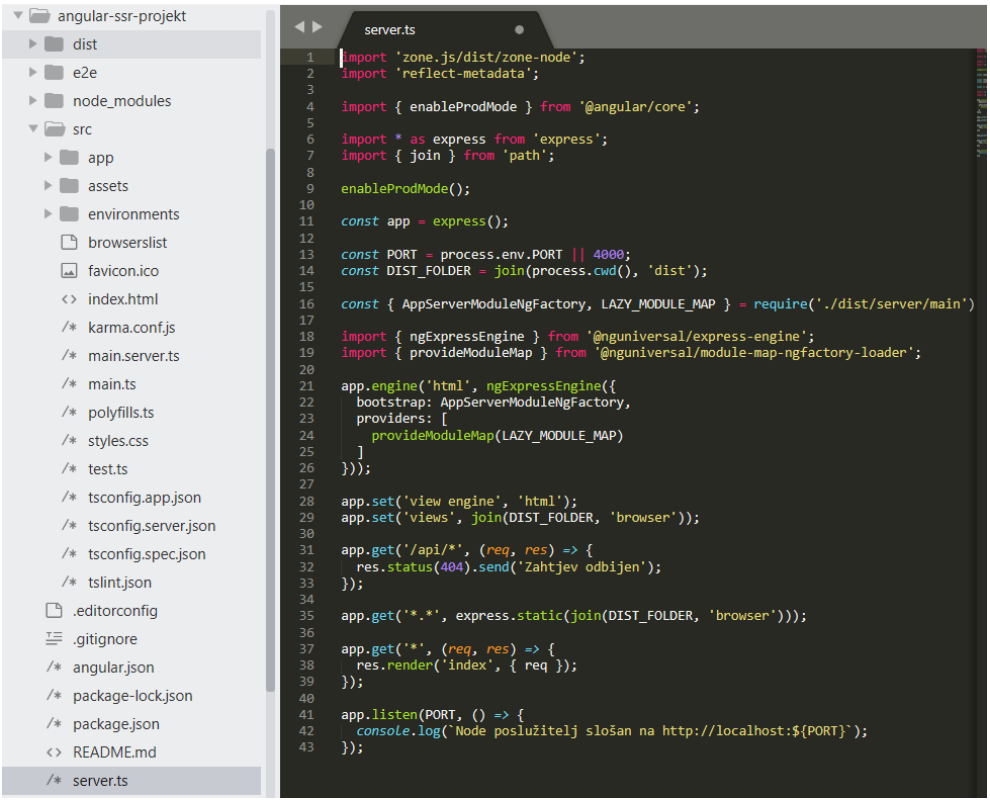
Nakon toga potrebno je kreirati datoteku main.server.ts koja izvozi AppServerModule i koja će kasnije biti navedena kod dodavanja cilja poslužitelja u konfiguraciji (Slika 52.).

Slika 52. Angular kreiranje datoteke main.server.ts

Izvor: obrada autora

Nadalje, potrebno je izgraditi univerzalni web poslužitelj (Slika 53.) koji će reagirati na zahtjeve aplikacije sa statičkim HTML-om kojeg definira na temelju predloška definiranog u Angular aplikaciji. Također, on poslužuje i statičke dijelove kao što su skripte, CSS i slike. Web poslužitelj u ovoj aplikaciji je izgrađen uz pomoć Express.js FW, iako može biti izgrađen u bilo kojem drugom.

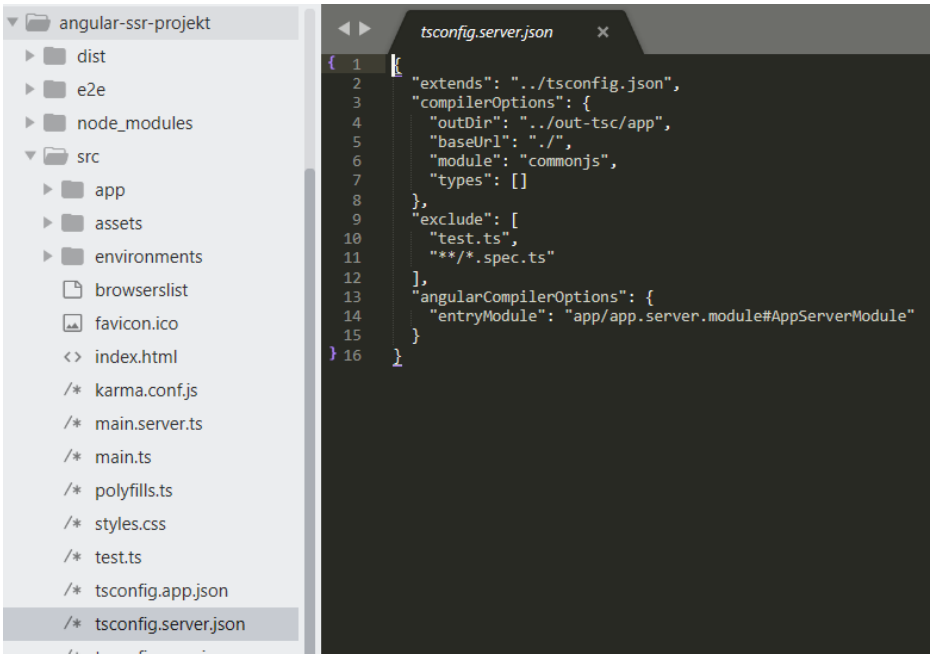
Slika 53. Izgradnja univerzalnog web poslužitelja



Izvor: obrada autora

Nakon toga potrebno je kreirati novi dokument kojim se zadaju postavke za TypeScript i kompilaciju univerzalne aplikacije (Slika 54.).

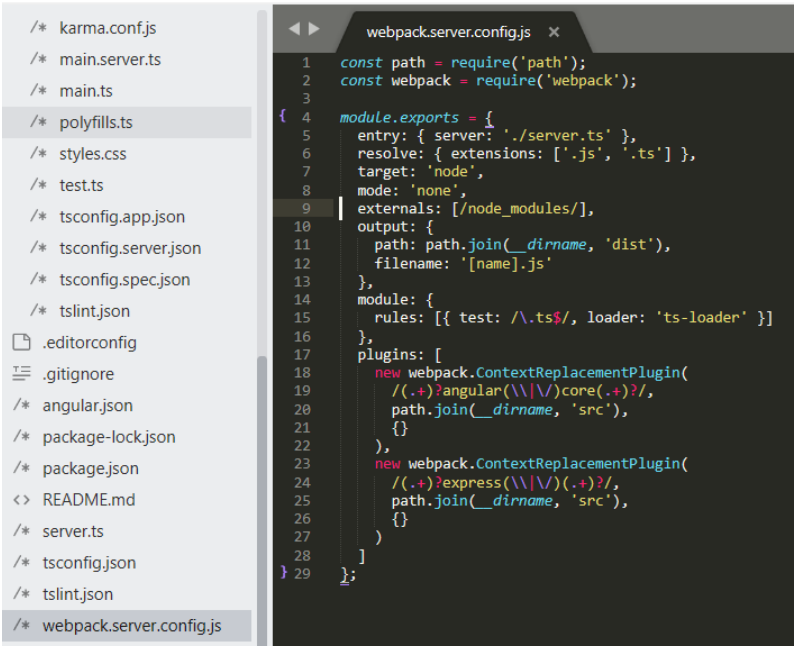
Slika 54. Angular konfiguracija za TypeScript i kompilaciju univerzalne aplikacije



Izvor: obrada autora

Univerzalne aplikacije ne trebaju nikakvu dodatnu konfiguraciju Webpack-a, CLI to odrađuje automatski. Ipak, budući da je poslužitelj također Typescript aplikacija, potrebno je koristiti Webpack za prevođenje. Potrebno je kreirati dokument naziva webpack.server.config.js u korijenskom direktoriju projekta sa kodom kao na slici ispod (Slika 55.).

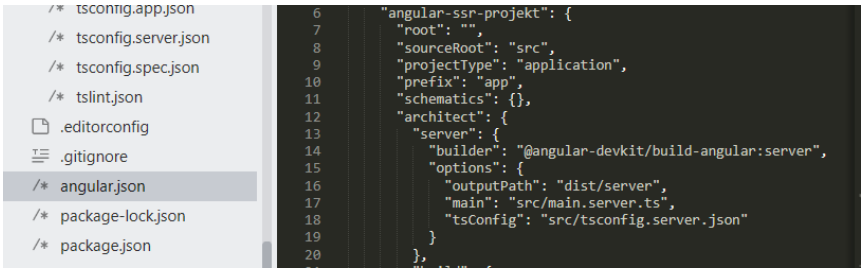
Slika 55. Angular webpack.server.config.js datoteka



Izvor: obrada autora

Nakon toga, u dokumentu angular.json potrebno je dodati pod objekt “arhitect” novi objekt naziva “server” (Slika 56.). U njemu su definirani mjesto smještanja kompajliranog rezultata, mjesto glavne ulazne točke i mjesto konfiguracijskog dijela TypeScript-a.

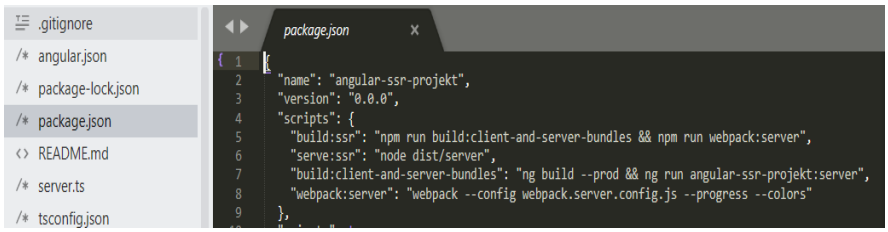
Slika 56. Angular angular.json datoteka



Izvor: obrada autora

Prije izgradnje projekta potrebno je definirati skripte u package.json dokumentu kao na slici ispod (Slika 57.).

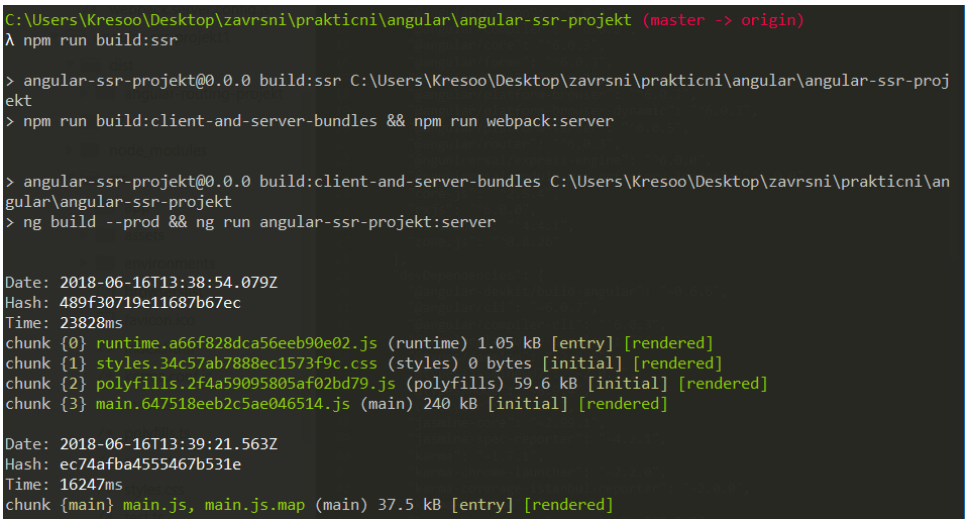
Slika 57. Angular package.json datoteka



Izvor: obrada autora

Naredba „npm run build:ssr“ kreira projekt sa odredištem u dist mapi (Slika 58.).

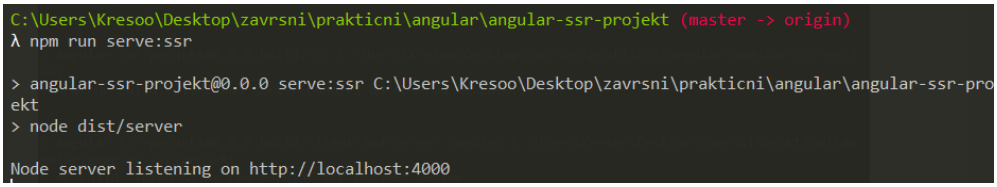
Slika 58. Angular kreiranje odredišnog SSR projekta



Izvor: obrada autora

Naredba “npm run serve:ssr” poslužuje aplikaciju kao na slici ispod. Aplikacija će biti dostupna na http://localhost:4000/ (Slika 59.).

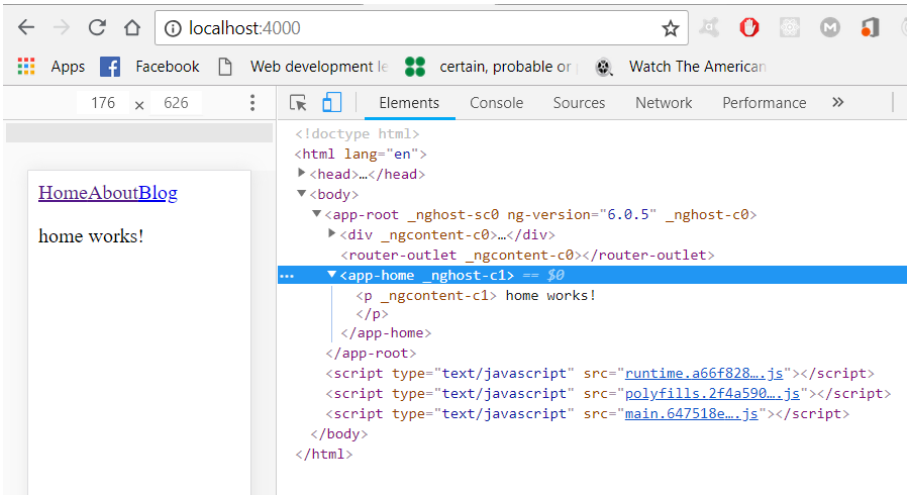
Slika 59. Angular pokretanje SSR projekta



Izvor: obrada autora

Angular Universal će generirati odredišne stranice aplikacije koja izgleda kao cjelovita aplikacija. Ipak, stranice su čisti HTML i mogu se prikazati čak i ako je JavaScript onemogućen. Statična inačica odredišne stranice ili putanje biti će prikazana kako bi zadržali pozornost korisnika. U isto vrijeme učitavati će se cijela Angular aplikacija, ali u pozadini. Na slici ispod (Slika 60.) vidljivo je da je učitan HTML tekst, a ne samo prazna oznaka.

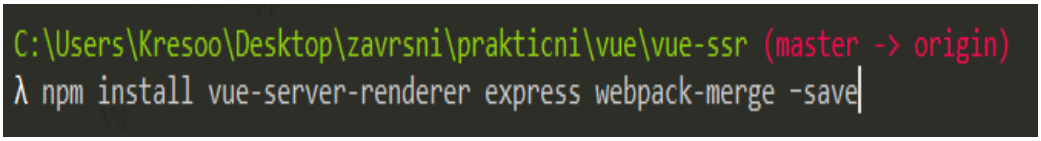
Slika 60. Angular SSR aplikacija



Izvor: obrada autora

Vue.js također ima službeno izdan vodič za izgradnju Vue aplikacije renderirane na serveru. Vodič se nalazi na posebnoj domeni, odvojenoj od Vue dokumentacije. Vrlo je detaljan i pretpostavlja da je korisnik već upoznat s Vue.js FW-om, kao i da ima pristojno znanje Node.js-a i Webpack-a. Isto tako, u dokumentaciji je navedena referenca na Nuxt.js, FW izdan od zajednice a koji se predstavlja kao rješenje višeg stupnja (Vue, 2018-4). Pruža neke dodatne mogućnosti, no isto tako ograničava developerovu direktnu kontrolu nad strukturom aplikacije (Vue, 2018-5). Prije samog početka potrebno je instalirati nekoliko modula naredom "npm install vue-server-renderer express webpack-merge –save" (Slika 61.). Modul vue-server-render je Vue modul za izvođenje prikaznih elemenata na poslužitelju, express je potreban jer je potreban nekakav Node poslužitelj i, budući da će biti potrebno postaviti konfiguracijske postavke i za klijentsku i za poslužiteljsku stranu, potreban je webpack-merge modul koji omogućuje spajanje konfiguracijskih postavki webpack-a.

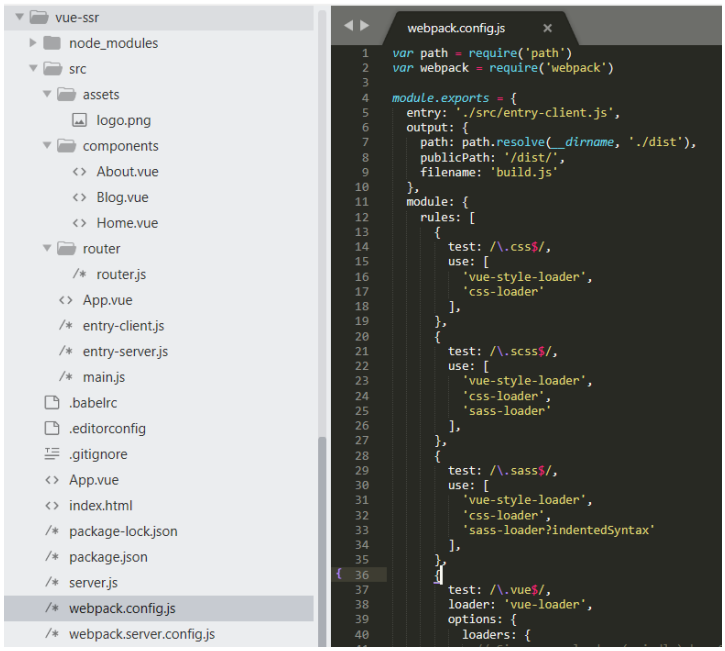
Slika 61. Vue.js instalacija modula za SSR



Izvor: obrada autora

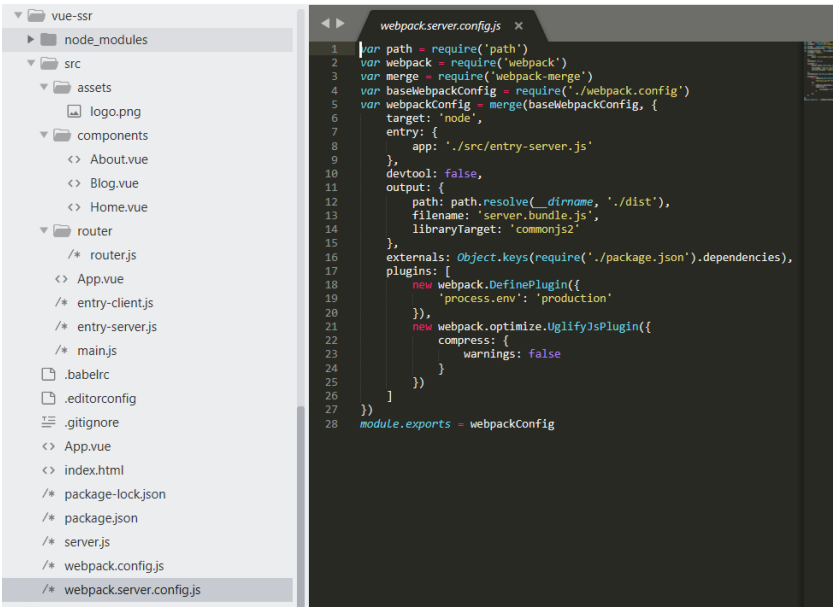
Sljedeći korak je kreiranje nova dva dokumenta za konfiguraciju postavki na strani klijenta (Slika 62.) i na strani poslužitelja (Slika 63.). To je vidljivo na slikama ispod. Najbitniji dio je postaviti "entry" kao ulazne točke za klijenta na prvoj slici i za poslužitelja na drugoj. Ostatak je kod dostupan na stranicama dokumentacije.

Slika 62. Vue.js webpack.config.js datoteka



Izvor: obrada autora

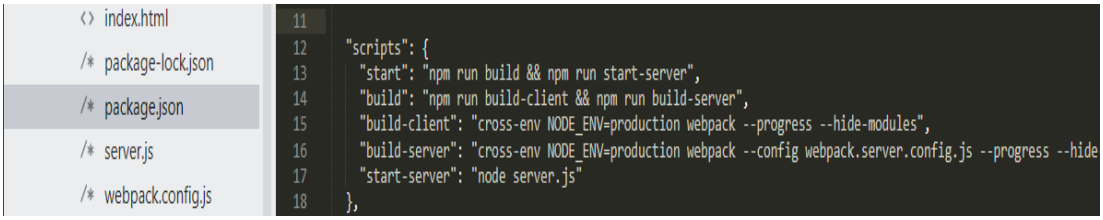
Slika 63. Vue.js webpack.server.config.js datoteka



Izvor: obrada autora

Nakon toga, slijedi definiranje skripte tj. načina izgradnje klijenta, poslužitelja i pokretanje poslužitelja. U konfiguraciji se koristi skripta “start” koja će pokrenuti tri koraka koji su prethodno spomenuti. No, postavljene su sve skripte da se odvojeno izvode, ako bude potrebno zbog nekog budućeg razloga (Slika 64.).

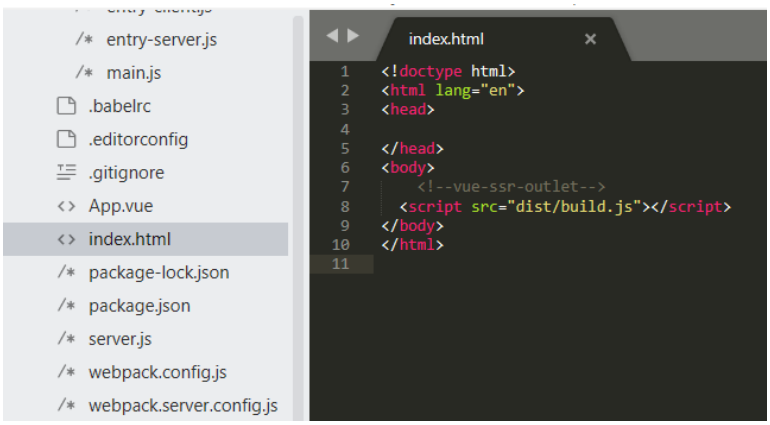
Slika 64. Vue.js package.json datoteka



Izvor: obrada autora

Sljedeći korak je definiranje glavnog HTML dokumenta (Slika 65.). Nakon što se projekt izgradi na poslužitelju, klijentu će biti poslana početna stranica sa punim HTML sadržajem. Nakon toga poslužitelj će nastaviti dohvaćati dokumente, a klijentska skripta će preuzeti upravljanje usmjeravanjem i ostalim prikaznim dijelovima.

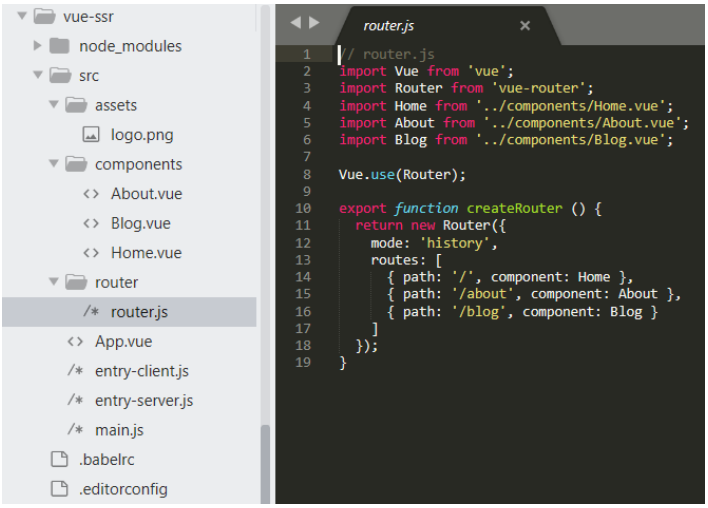
Slika 65. Vue.js definiranje glavnog HTML dokumenta za SSR



Izvor: obrada autora

Sljedeće je potrebno definirati putanje, ovoga puta u posebnom dokumentu po konvenci navedenoj u dokumentaciji posebno namjenjenoj ovom poglavlju. Budući da će aplikacija započeti s poslužiteljem, potrebno je osigurati novu instancu usmjerivača za svaki zahtjev poslužitelja (Slika 66.).

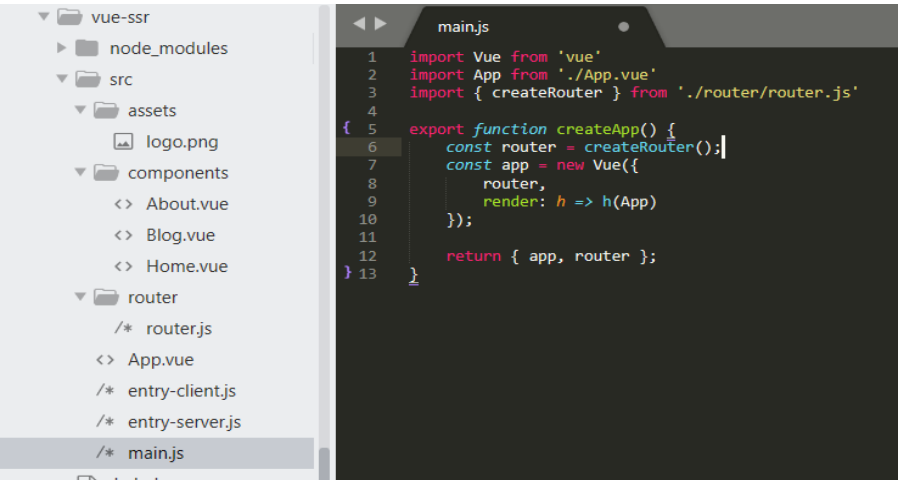
Slika 66. Vue.js konfiguracija usmjerivača



Izvor: obrada autora

Iz istog razloga zbog kojeg je potrebno pružiti novu instancu usmjerivača, potrebno je i pružiti novu instancu aplikacije. Ova datoteka ima odgovornost za pokretanje usmjerivača i komponente korijenske aplikacije. Oba, ulazna točka poslužitelja i ulazna točka klijenta, koriste ovu datoteku, što je vidljivo na slici ispod (Slika 67.).

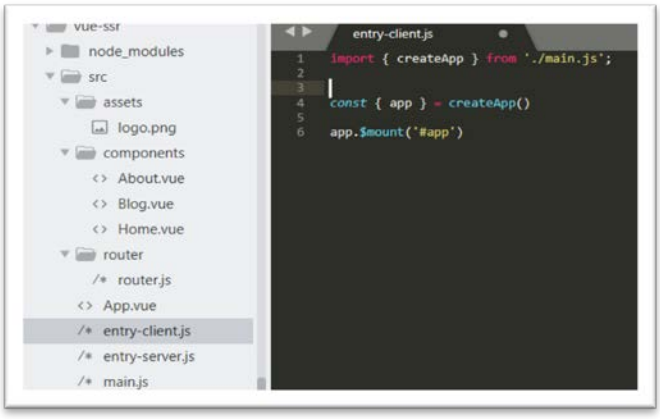
Slika 67. Vue.js kreiranje nove instance aplikacije sa usmjerivačem



Izvor: obrada autora

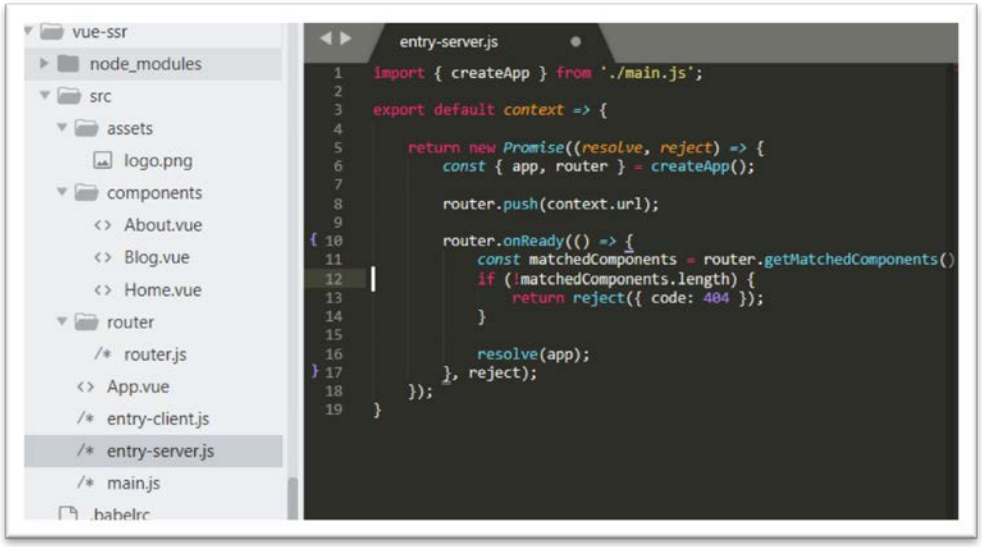
Slijedi kreiranje dva nova dokumenta, odnosno ulazne datoteke za konfiguraciju konfiguracije klijenta (Slika 68.) webpacka i poslužitelja (Slika 69.), što je vidljivo na slikama ispod.

Slika 68. Vue.js konfiguriranje webpacka klijenta



Izvor: obrada autora

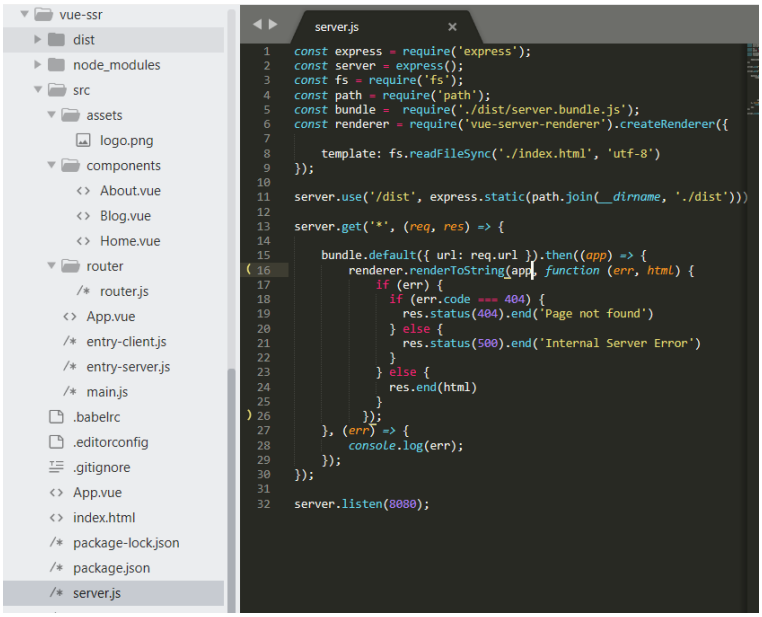
Slika 69. Vue.js konfiguriranje webpack poslužitelja



Izvor: obrada autora

Na kraju, jedino što nedostaje je konfiguracija i pokretanje “express” poslužitelja, što je vidjivo na slici ispod (Slika 70.). Kod prikazuje uvođenje određenih modula i dohvaćanje dokumenta koji će biti izgrađen od strane webpacka za poslužitelj. Tako će poslužitelj moći vidjeti sav kod i prenijeti ga na klijenta po njegovom zahtjevu.

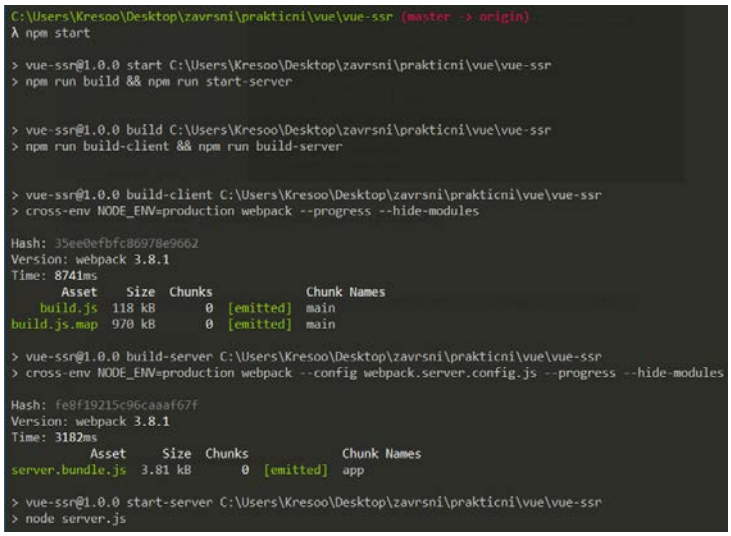
Slika 70. Vue.js konfiguracija express poslužitelja



Izvor: obrada autora

Naredbom “npm start” aplikacije će biti uspješno pokrenuta (Slika 71.).

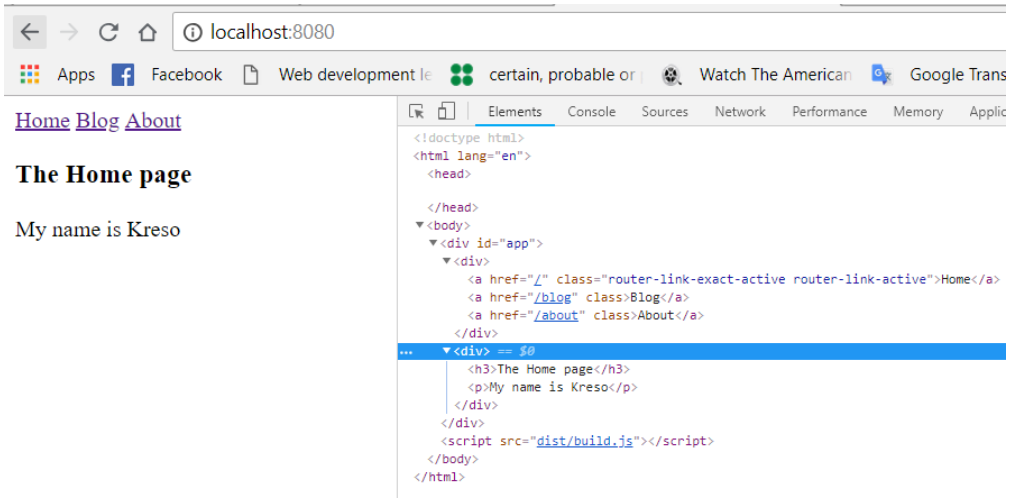
Slika 71. Vue.js pokretanje SSR aplikacije



Izvor: obrada autora

Aplikacija će biti pokrenuta na 8080 portu, a izvođenje prikaznih elemenata biti će vidljivo u Chrome alatima kada se klikne na inicijalni dokument (Slika 72.). Vidljivo je da je izbačen HTML sadržaj, a ne prazan kao što je to kod standarnog dohvaćanja SPA. Sljedeći dio klijentske strane preuzima skripta, a stranice se izvode na poslužitelja samo na zahtjev.

Slika 72. Vue.js SSR aplikacija



Izvor: obrada autora

React-u nedostaje službene dokumentacije o SSR-u. React API uključuje objekt nazvan ReactDOMServer, a njegova je svrha prikazivanje komponenata u HTML kod. Također, potrebno je naučiti koristiti druge pakete kao što su React Router i Redux, kako bi prikazivanje na poslužitelju i klijentu funkcioniralo uredno. Iz React tima je najavljeno da će u skorije vrijeme ovaj dio dobiti službenu podršku. Postoji i FW koji se može koristiti za izradu React SSR aplikacija, a naziva se Next.js. Prema tomu React omogućuje SSR, ali bez službene podrške i uz upotrebu dodatnih paketa treće strane (Bejar, 2017).

4.5. Povećanje brzine izvođenja aplikacije

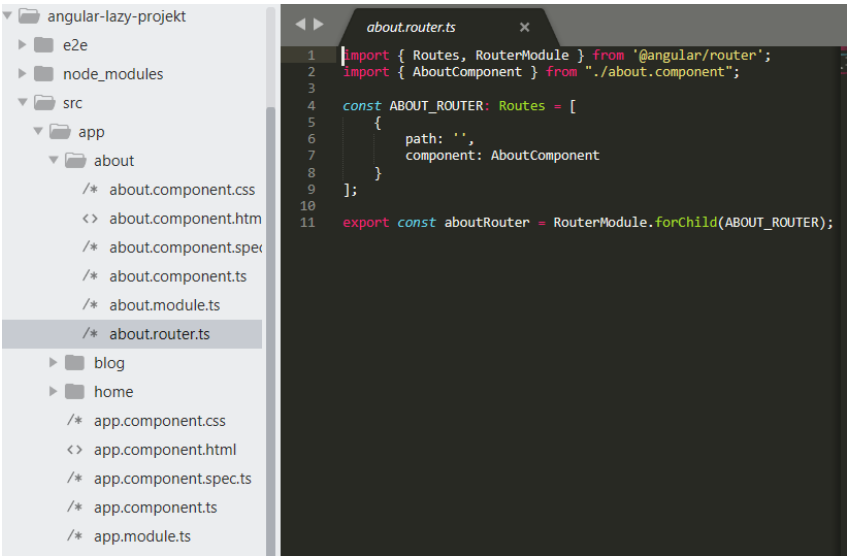
Najčešći uzrok loše optimizirane SPA je veličina inicijalne skripte. To je datoteka koja se mora preuzeti kod inicijalnog učitavanja stranice. To uzrokuje sporo učitavanje stranice i iz tog razloga njegovu je veličinu potrebno smanjiti što je više moguće. Također, brzinu izvođenja aplikacije, kod svakog je FW moguće povećati na njima specifičan način.

4.5.1. Podijela koda i lijeno učitavanje

Kako bi se izbjeglo sporo učitavanje aplikacije sa velikom bundnle datotekom, kod srednjih i velikih aplikacija često se izvršava podjela (eng. *code splitting*) te datoteke na više manjih i lijeno učitavanje (eng. *lazy loading*) aplikacije. Primjerice na aktivaciju specifične putanje, asinkrono se učitavaju pojedine komponente. Ova funkcionalnost može značajno povećati performanse inicijalnog učitavanja aplikacije (Delaney, 2017).

Za postavljanje ove funkcionalnosti kod Angular-a potrebno je stvoriti novi modul i putanje djece za svaku komponentu koja će se lijeno učitati. U sljedećem primjeru prikazat će se lijeno učitavanje „About“ komponente. Na slici ispod (Slika 73.) uvedena je jedina dijete komponenta, a to je „About“. Na kraju koda koristi se funkcija forChild za izvoz usmjerivača, jer to nije korijen usmjerivača aplikacije.

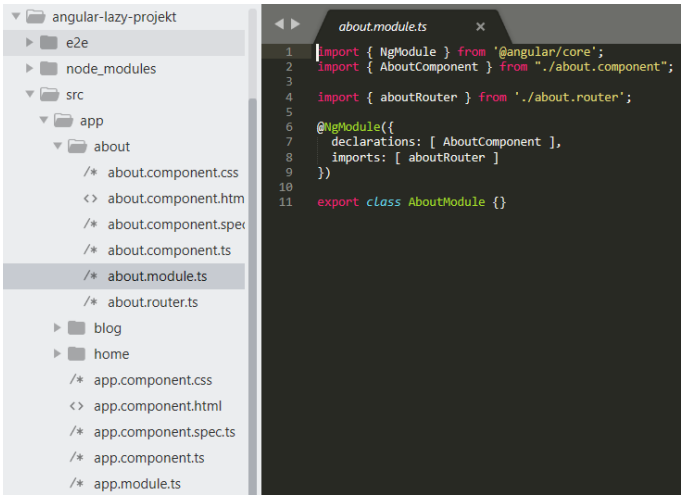
Slika 73. Angular about.router.ts datoteka



Izvor: obrada autora

Slijedi kreiranje nove module.ts datoteke (Slika 74.). Vrlo je slična kao app.module.ts datoteka. Uvedene su sve povezane komponente u polje deklaracije i na kraju je cjelokupna datoteka izvedena, kao jedan modul, za daljnje korištenje.

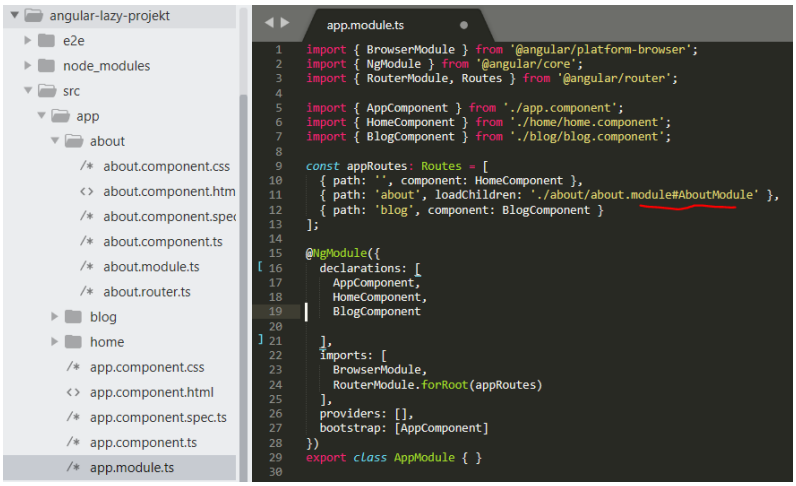
Slika 74. Angular kreiranje modula za About komponentu



Izvor: obrada autora

Nakon toga, u korijenskoj module.ts datoteci potrebno je buduću lijeno učitanu putanju navesti specifično posebnim usmjerivačem „loadChildren“ (Slika 75.). Putanja se sastoji od cijele relativne putanje datoteke, znaka „#" i, zatim, naziv klase izvoznog modula.

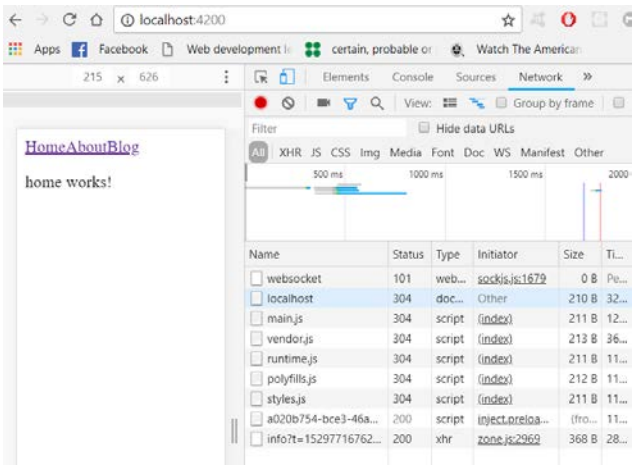
Slika 75. Angular dodavanje putanje posebnim usmjerivačem „loadChildren“



Izvor: obrada autora

Na slici ispod vidljiva je početno učitana aplikacija (Slika 76.).

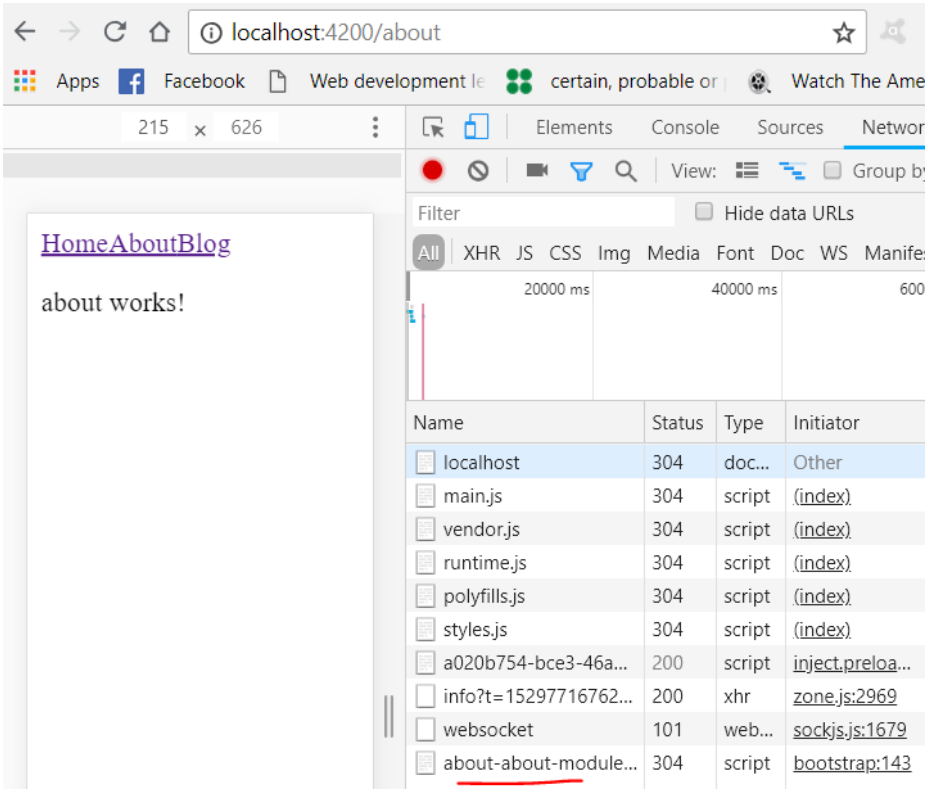
Slika 76. Angular aplikacija za lijeno učitavanje



Izvor: obrada autora

Aktivacijom /about putanje učitan je poseban komad koda koji nije ranije, odnosno komponenta „About“, što je vidljivo na slici ispod (Slika 77.).

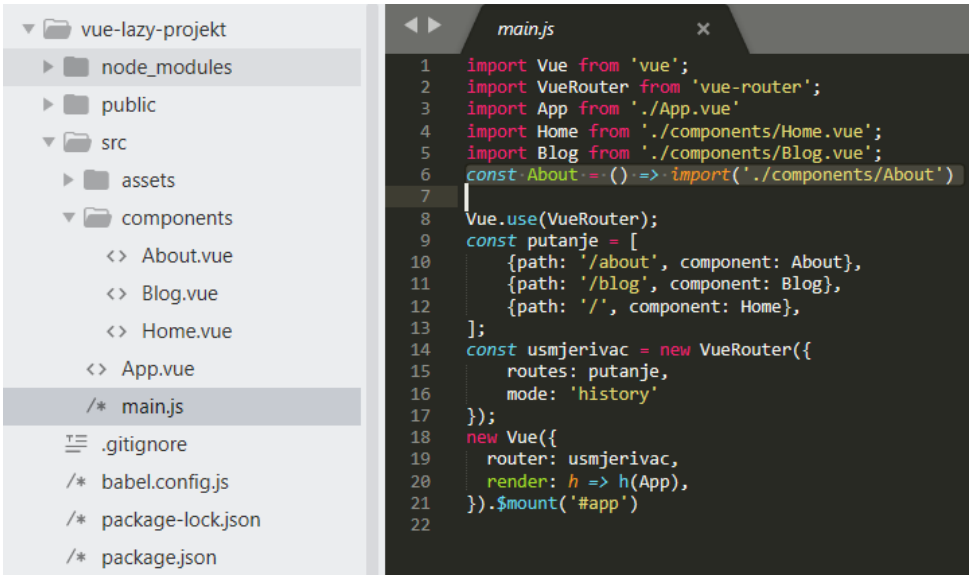
Slika 77. Angular lijeno učitavanje samo jedne komponente



Izvor: obrada autora

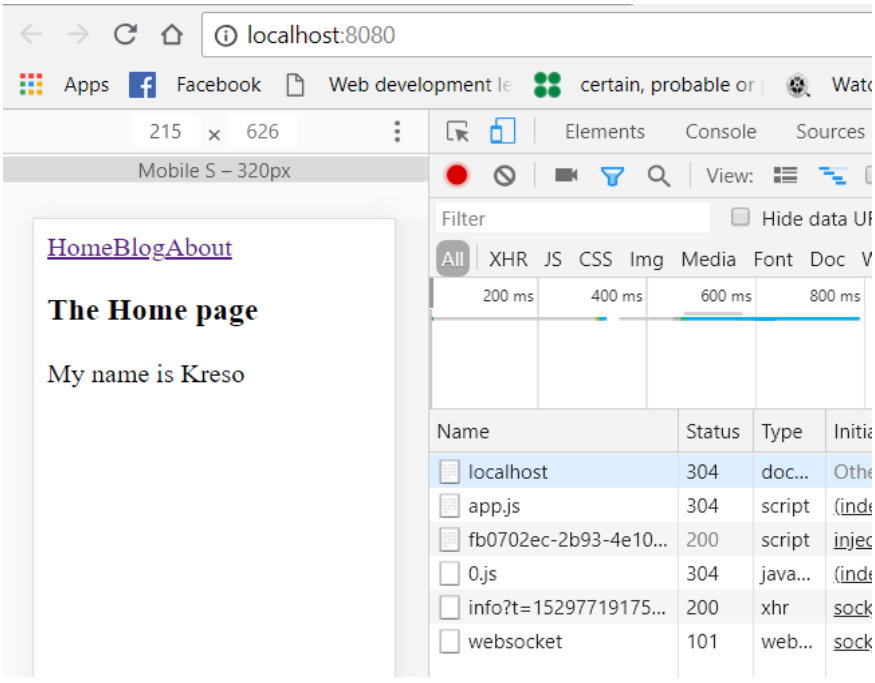
Vue.js ovu funkcionalnost izvodi prilično elegantno. Modul za klijentsko usmjeravanje ovu funkcionalnost već ima sadržanu u njemu samome. Dovoljno je samo učitati komponentu na specifičan način prikazan na slici ispod (Slika 78.). Sve ostalo ostaje kao i ranije (Slika 79., 80.).

Slika 78. Vue.js uvođenje komponente za lijeno učitavanje



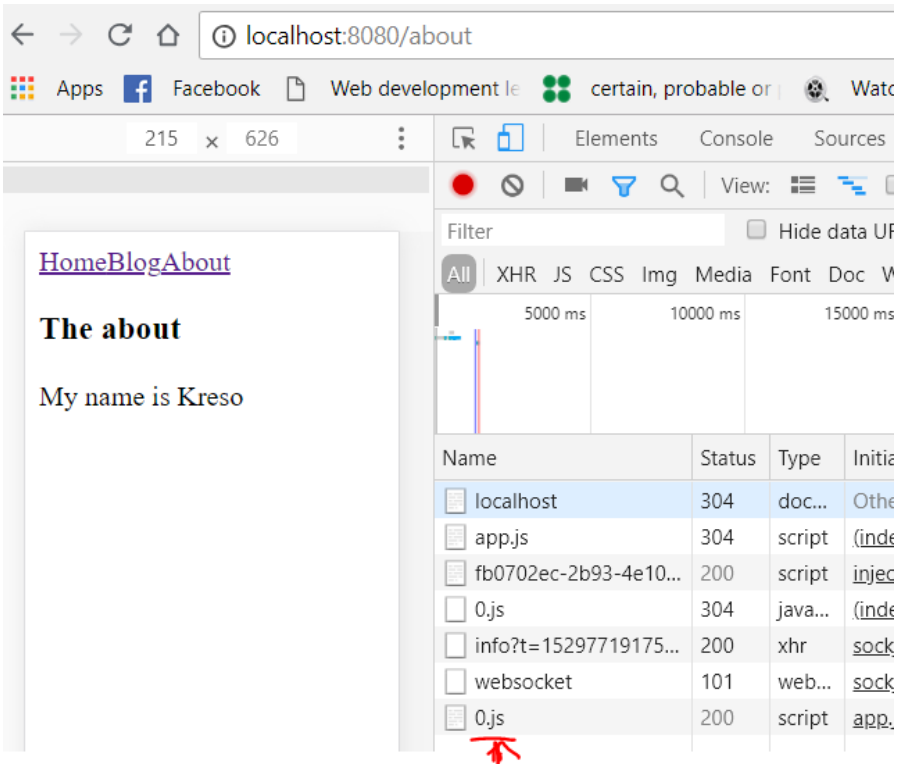
Izvor: obrada autora

Slika 79. Vue.js aplikacija za lijeno učitavanje



Izvor: obrada autora

Slika 80. Vue.js lijeno učitavanje samo jedne komponente



Izvor: obrada autora

React ovu funkcionalnost ne može izvesti ovako jednostavno kao što je to slučaj kod Vue.js-a, ali zato ima popularni modul razvijen od treće strane. Za njegovu instalaciju potrebno je pokrenuti naredbu kao na slici ispod (Slika 81.).

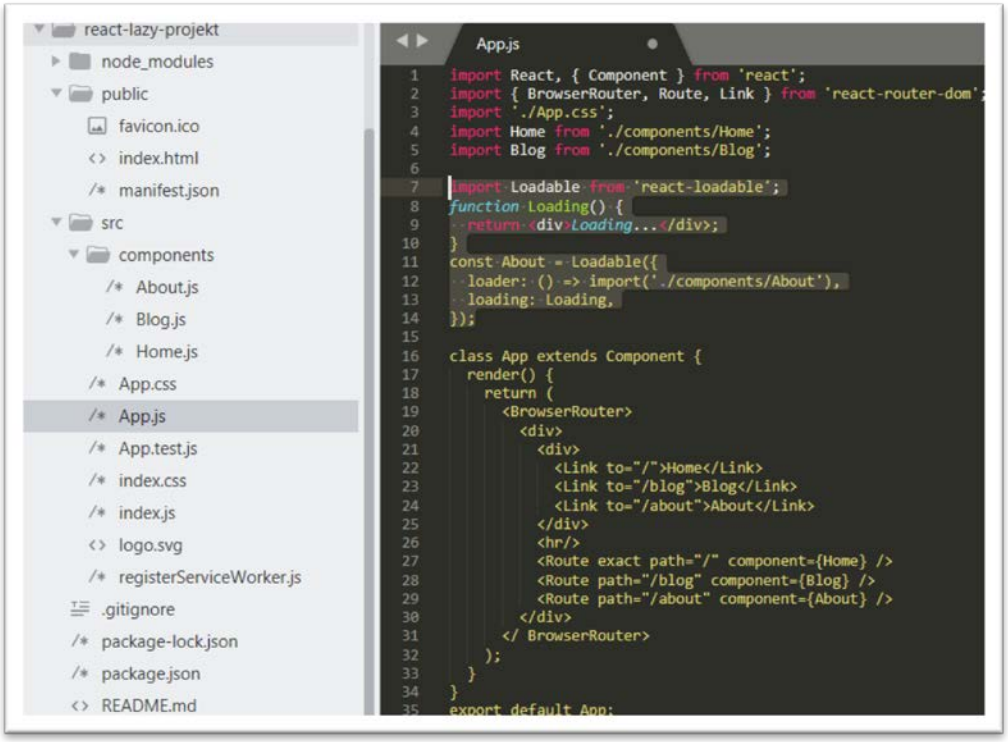
Slika 81. React.js instalacija modula za lijeno učitavanje

```
C:\Users\Kresoo\Desktop\zavrsni\prakticni\react\react-lazy-projekt (master -> origin)
λ npm install react-loadable --save
```

Izvor: obrada autora

Nakon toga uveden je novi modul, te je uvedena komponenta na modulov specifičan način. Lijeno učitavanje kao i u prethodnim primjerima uveden je samo na „About“ komponentu. Za vrijeme učitavanja novog komada koda prikazivat će se tekst „Loading“ sve dok komponenta nije učitana. Sve ovo vidljivo je na slici ispod (Slika 82).

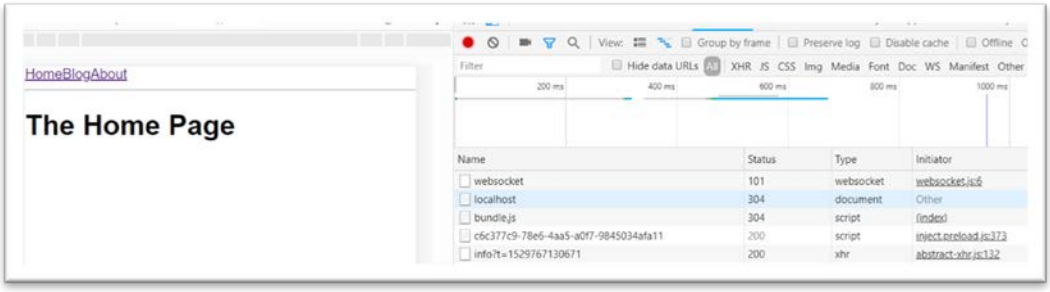
Slika 82. React.js uvođenje komponente za lijeno učitavanje



Izvor: obrada autora

Nakon otvaranja razvojnih alata u Chrome pregledniku vidljivo je da je na inicijalnom učitavanju učitana osnovna „bundle“ datoteka sa popratnom imovinom (Slika 83).

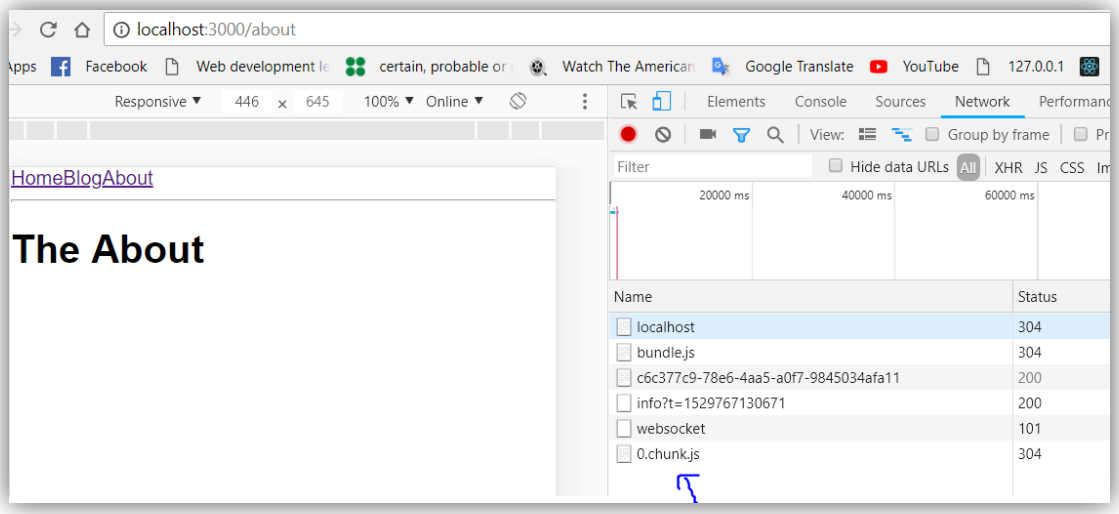
Slika 83. React.js aplikacija za lijeno učitavanje



Izvor: obrada autora

Nakon klika na „About“ učitan je novi komad koda, odnosno nova komponenta, što je vidljivo na slici ispod (Slika 84).

Slika 84. React.js lijeno učitavanje samo jedne komponente

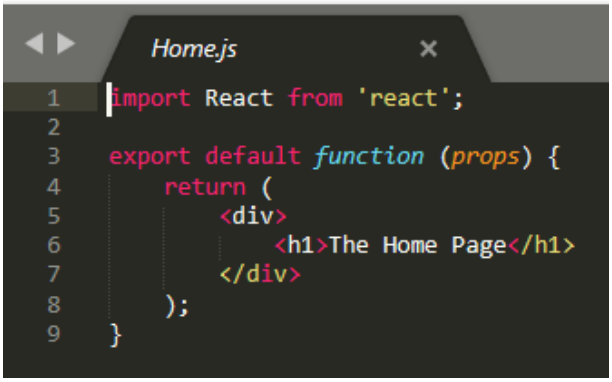


Izvor: obrada autora

4.5.2. Mogućnost pisanja posebnih komponenata koje štede resurse

U React-u postoje dvije osnovne vrste komponenata koje je moguće pisati. To su one koje upravljaju stanjem, odnosno komponente klase, i one koje ne upravljaju stanjem, odnosno komponente funkcije. Komponente klase upravljaju stanjem i svaka njegova promjena rezultirati će ponovnim dohvaćanjem te komponente u virtualni DOM po kojem principu React funkcionira. Kada se stanje neke komponente promijeni, aktivira se ponovno prikazivanje cjelokupnog podstabla komponenti, počevši od te komponente kao korijena. React na svaku promjenu stanja dohvaća tu komponentu ponovno, provjerava različitosti između stvarnog i virtualnog DOM-a, te na kraju ažurira samo različitosti u stvarni DOM. Da bi se izbjeglo nepotrebno ponovno prikazivanje komponenti djece, potrebno je koristiti komponente funkcije ili prezentacijske komponente. To su obične Javascript funkcije koje vraćaju određeni HTML sadržaj. Ne upravljaju stanjem i nemaju render() metodu što znači da React ne pokreće ponovno tu komponentu i svu njenu djecu kako bi osvježio svoj virtualni dom, čime se znatno povećava prostor za povećanjem performansi. React u svojoj dokumentaciji preporuča korištenje ovakvih komponenata gdje god je to moguće. Primjer ovakve komponente je prikazan na slici ispod (Slika 85).

Slika 85. React.js komponenta funkcije



Izvor: obrada autora

Vuejs ovu funkcionalnost nema, no budući da ovaj FW niti nema render() metodu, to nije niti potrebno. Dovoljno je samo izostaviti stanje u komponenti i Vue.js će takvu komponentu automatski smatrati prezentacijskom.

Angular funkcionira totalno drugačije od Reacta i Vue.js-a. ovaj FW u cijeloj aplikaciji osluškuje na promjene i shodno tome izvršava akcije. Nema poseban način pisanja komponenata i to je striktno definirano, ali zato pruža mogućnosti nadogradnje postojećih sa posebnim metodama, što će biti objašnjeno u sljedećem poglavlju.

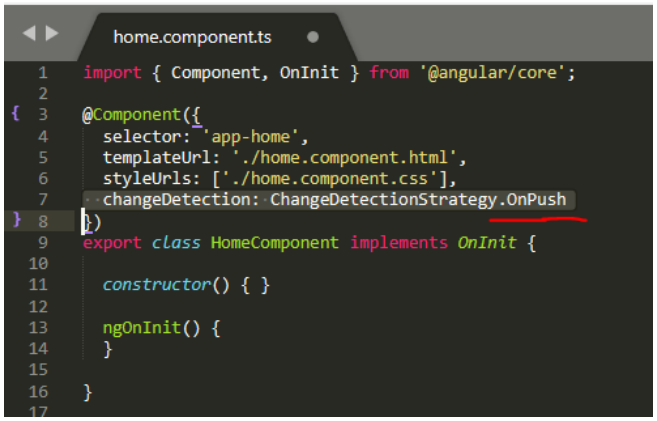
4.5.3. Korištenje metode za povećanje brzine

React, kako je ranije rečeno, kod promjene stanja komponente aktivira ponovno prikazivanje cjelokupnog podstabla komponenti, počevši od te komponente kao korijena. Iako se ažuriraju samo promijenjeni DOM čvorovi, ponovno prikazivanje traje još neko vrijeme. U mnogim slučajevima to nije problem, ali ako je vidljivo značajno usporavanje, sve to moguće je ubrzati korištenjem funkcije životnog ciklusa shouldComponentUpdate, koja se pokreće prije početka procesa ponovnog prikazivanja. Zadana implementacija ove funkcije postavljena je na „true“, ostavljajući React da izvršava ažuriranja. U situacijama kada nema potrebe da se komponenta ažurira, moguće je metodu shouldComponentUpdate postaviti da vraća „false“. Time će komponenta preskočiti cijeli proces renderiranja (React, 2018-5).

Prema zadanim postavkama, Angular pokreće svoju detekciju promjena na svim komponentama svaki put kada se nešto promijeni u aplikaciji. Iako je ovaj FW dosta brz, kako aplikacija raste, morati će ulagati jako puno napora da prati sve promjene. Iz tog razloga Angular omogućuje davanje ručnog pokazatelja kada provjeriti komponentu. To se odrađuje “OnPush” metodom unutar “ts” datoteke pojedine komponente, što je prikazano na slici ispod (Slika 86). U

ovom slučaju, Angular će detekciju promjena nad ovom komponentom pokrenuti samo u slučaju promjene stanja neke roditeljske komponente ili kod pokrenutog događaja te komponente ili pak njenog djeteta.

Slika 86. Angular komponenta sa OnPush metodom



Izvor: obrada autora

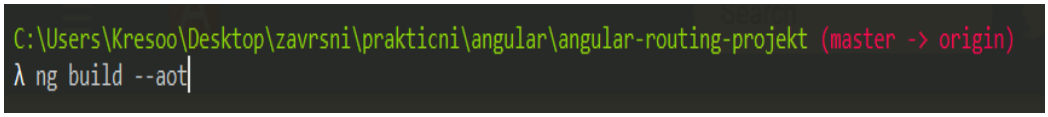
Vue.js ovu funkcionalnost nema, niti mu je potrebna. Kako se navodi u službenoj dokumentaciji ovisnosti među komponentama automatski se prate tijekom njhovog prikazivanja, tako da sustav zna točno koje komponente zapravo trebaju ponovo prikazati kada se stanje mijenja. Može se smatrati da svaka komponenta već u sebi ima automatski implementiranu shouldComponentUpdate metodu.

4.5.4. Automatska optimizacija konzolarnim naredbama

U razvoju aplikacije upotrebom Angular FW-a, korisnik piše svoj razvojni (development) kod koji se pokreće u pregledniku na development poslužitelju. U tom slučaju predlošci će biti

takvi isporučeni u preglednik, što znači da neće biti kompajlirani. HTML kod unutar predložaka gdje se koristi Angular sintaksa (npr. ngFor, ngIf, itd.) bit će dostavljen pregledniku koji to ne razumije. Iz tog razloga Angular uključuje kompajler koji je zadužen za „razumijevanje“ takvog HTML koda napisanog u komponentama, te kompajliranje tijekom izvođenja (eng. *Just-in-Time* - JIT). To će povećati veličinu bundle datoteke i umanjiti performanse aplikacije u izvođenju. Da bi se performanse povećale, potrebno je pokrenuti AOT (eng. *Ahead-of-Time*) kao dio radnog dijagrama. Ova vrsta kompajliranja omogućava da se sav kod kompajlira u Javascript već u radnom dijagramu, što znači da će pregledniku biti dostavljen samo Javascript kod bez Angular kompajlera i bez potrebe za kompajliranjem i parsiranjem predložaka tijekom izvođenja. Ovo nije početna opcija Angular-a i potrebno ju je uključiti kroz CLI, a najčešće se uključuje prelaskom u produkcijski način rada aplikacije (Angular, 2018-6)(Yiang, 2016)(Zia, 2017). Naredba izgleda kao na slici ispod (Slika 87.).

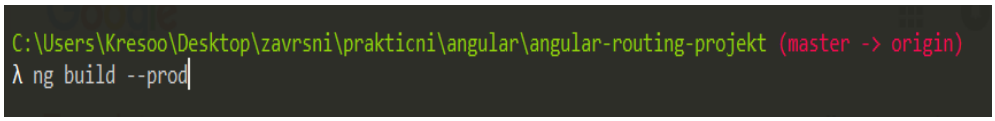
Slika 87. Angular pokretanje AOT



Izvor: obrada autora

Kod pripremanja aplikacije za produkciju preporuča se optimiziranje aplikacije naredbom kao na slici ispod (Slika 88.). Ova naredba, osim što automatski pokreće ranije prikazan AOT, uklanja višak razmaka, komentare, prepisuje kod za korištenje kratkih, kriptiranih varijabli i imena funkcija, te uklanja mnogo nekorištenog koda.

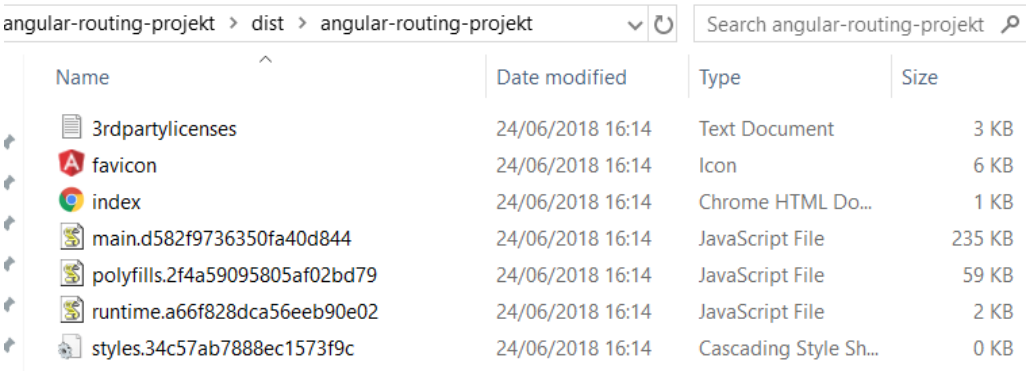
Slika 88. Angular kreiranje projekta uz automatsku optimizaciju



Izvor: obrada autora

Optimizirana aplikacija biti će stvorena u dist mapi kao na slici ispod (Slika 89.).

Slika 89. Angular kreirana aplikacija u dist mapi



Izvor: obrada autora

Iako se preporučuje gore navedena naredba, Angular naredbeni interfejs omogućuje cijeli niz naredbi kojima je moguće optimizirati samo pojedine dijelove, no to se ipak koristi u specifičnim slučajevima.

Kod React-a proces optimizacije je minimalan. Cijeli programski kod je već u Javascript-u i ne postoji nikakav HTML kod kojeg je potrebno parsirati. Za pripremanje aplikacije za produkciju i izvršavanje finalne optimizacije dovoljno je pokrenuti naredbu kao na slici ispod (Slika 90.).

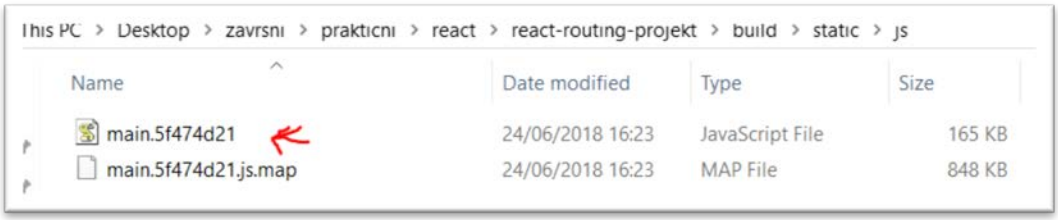
Slika 90. React.js kreiranje projekta uz optimizaciju

```
C:\Users\Kresoo\Desktop\zavrsni\prakticni\react\react-routing-projekt (master -> origin)
λ npm run build
```

Izvor: obrada autora

Da bi aplikacija radila dovoljno je postavljanje index.html datoteke i čiste js datoteke prikazane na slici ispod (Slika 91.).

Slika 91. React.js bundle datoteka



Izvor: obrada autora

Vue.js omogućuje odabir između dva pristupa: runtime-only i runtime+compiler. Runtime-only pristup podrazumijeva ne isporuku kompajlera. To znači da je umjesto predloška unutar instance potrebno koristiti .vue datoteke koji će biti kompajlirani u Javascript. Te datoteke uz normalan HTML kod sadrže i vue-specifičan kod koji se kompajlira kao dio radnog dijagrama, tako da je ono što se na kraju dostavi pregledniku optimizirani kod koji je samo Javascript i ne uključuje bilo koji prevodilac ili bilo koji HTML kod. Runtime+compiler pristup omogućuje kontrolu sitnih dijelova DOM-a korištenjem predložaka unutar instance, odnosno korištenjem komponenti. Tada nije potrebno korištenje .vue datoteka, ali pritom se ne koristi visoka optimiziranost bundle datoteke pa je ona veća i aplikacija ima oslabljene performanse (Vue, 2018-7). Vue.js konzolarno naredbom „npm run build“ (Slika 92.) priprema aplikaciju za produkciju s runtime-only pristupom. Ovom naredom odrađuje se sve tako da nije potrebno dodatno optimizirati.

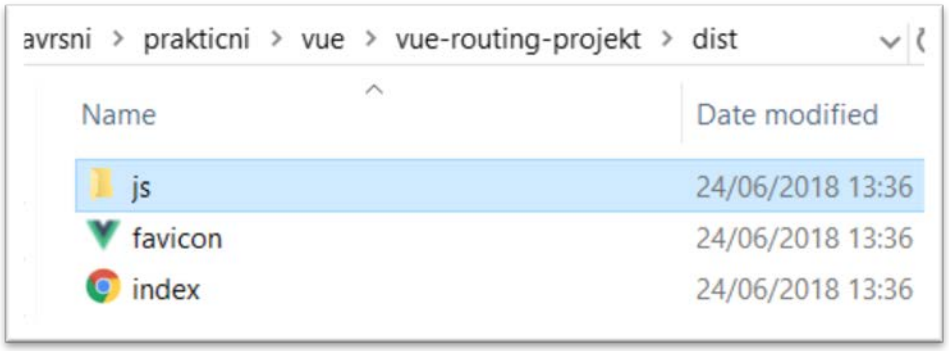
Slika 92. Vue.js kreiranje projekta uz optimizaciju

```
C:\Users\Kresoo\Desktop\zavrsni\prakticni\vue\vue-routing-projekt (master -> origin)
λ npm run build
```

Izvor: obrada autora

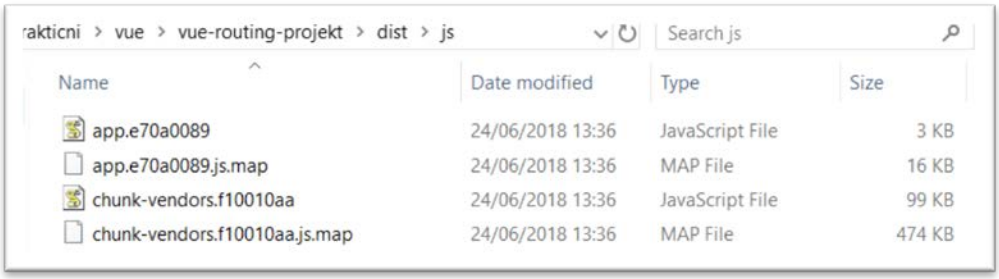
Ova naredba stvara dist mapu, a za postavljanje na poslužitelj dovoljna je „index.html“ (Slika 93.) datoteka i bundle datoteke u „js“ mapi (Slika 94.).

Slika 93. Vue.js kreirani optimizirani projekt – index.html datoteka



Izvor: obrada autora

Slika 94. Vue.js kreirani optimizirani projekt 2 – skripte



Izvor: obrada autora

4.6. Upravljanje stanjem podataka

Kako aplikacija raste, znati da se promjena stanja u jednom modulu konstantno i točno odražava u drugim modulima vrlo je težak zadatak za svakog developera. Upravljanje stanjem jedan je od najtežih dijelova razvoja SPA aplikacije u sigurnom i, kroz vrijeme, održivom okruženju. Kao što je već navedeno, ovo pitanje odgovorit će postoji li jasan način i službeno izdan paket kako upravljati stanjem ili je to, ipak, na developeru.

4.6.1. Mogućnost odvajanja logike u posebne datoteke

Odvajanje logike u posebne module, osim što povećava održivost i strukturalnost aplikacije, značajno olakšava i proces razvoja aplikacije. Primjerice, stanje neke vrijable, HTTP poziv ili, pak, metoda, koja se treba dijeliti na više komponentama, u nekim fw mogu biti smješteni u posebnim i samo za tu svrhu namjenjenim datotekama.

U Angular FW-u, u paketima koje pruža službeni tim, za ovu funkcionalnost uključeni su servisi. Da bi ih se koristilo dovoljno je stvoriti novu komponentu, ručno pisanjem ili automatski konzolarno. Na slici ispod (Slika 95.) prikazano je kreiranje servisa automatskim generiranjem.

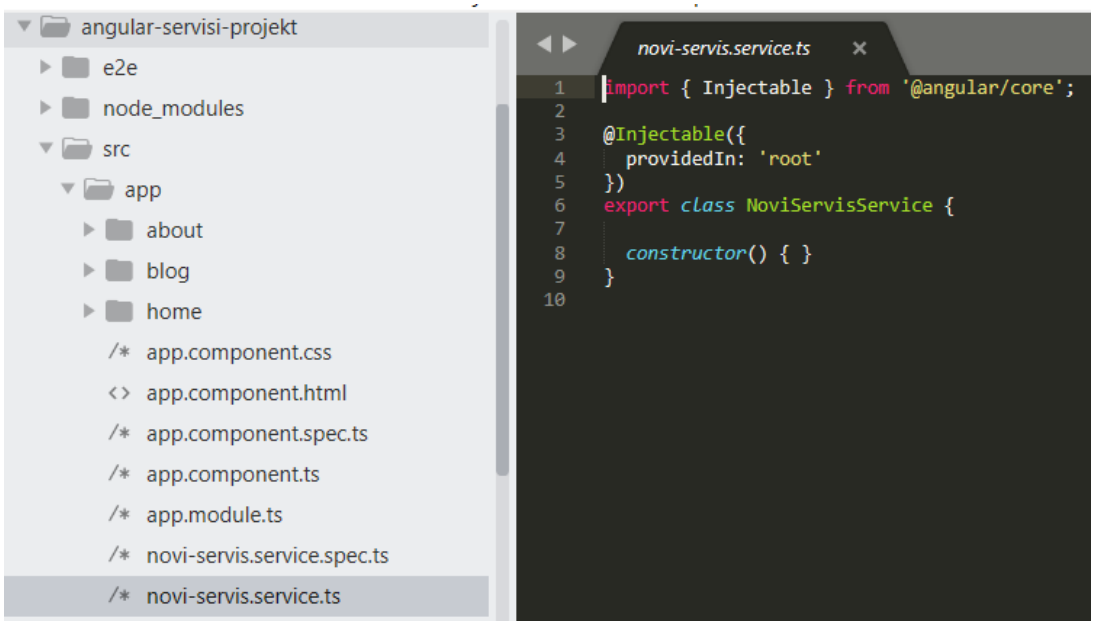
Slika 95. Angular kreiranje servisa

```
C:\Users\Kresoo\Desktop\zavrsni\prakticni\angular\angular-servisi-projekt (master -> origin)
λ ng generate service novi-servis
CREATE src/app/novi-servis.service.spec.ts (399 bytes)
CREATE src/app/novi-servis.service.ts (139 bytes)
```

Izvor: obrada autora

Servis će biti kreiran u korijenskom direktoriju, što je prikazano na slici ispod (Slika 96.). Izgleda slično kao komponentna datoteka, ali koristi dekorator „@Injectable ()“, što znači da ga se može uvesti u druge komponente i pristupiti njegovim svojstvima i metodama.

Slika 96. Angular uvođenje dekoratora u kreirani servis



Izvor: obrada autora

Za veće Angular aplikacije, s puno asinkronih aktivnosti i gdje postoji mnogo stanja koja će biti dijeljena i manipulirana kroz više komponenti i modula, upravljanje stanjem korištenjem servisa može biti prilično zahtjevno. U tom slučaju možda će zatrebati neki od paketa za upravljanje stanjem kao što je to ngrx/store (Rangle, 2017).

React i Vue.js nemaju ovu funkcionalnost, nego se ovo odrađuje na način da se izolira stanje komponente na najvišem nivou i onda dobacuje dolje u one na razini ispod, dopuštajući da stanje teče od vrha prema dolje. U većini slučajeva, to je izvedeno prilično elegantno. No, problemi se javljaju kada stablo komponenti naraste, i kada su komponente daleko jedna od druge,

ili kada jedna komponenta nije potomak druge, i obje komponente ovise o istom stanju. Primjer, imamo navbar i komponenta koja se javlja kada imamo novu poruku. Unutar navbara je komponenta koja će se pojaviti ako imamo novu poruku. U ovom scenariju, i navbar komponenta i message komponenta ovise o istom stanju. Ovdje, nijedna komponenta nije potomak druge, što čini upravljanje stanjem mnogo izazovnije. Kako bi se izbjegli "višestruke izvore istine" ispravna stvar za napraviti u ovom slučaju bila bi pohranjivanje ovog zajedničkog stanja u najbližem zajedničkom predaku ili bilo kojem drugom međusobnom predaku i poslati stanje dolje potomku. Za male do srednje aplikacije to je dobar pristup, ali je nezgrapan s velikim aplikacijama, a mogu se pojaviti i problemi u performansama (Crawford, 2016).

4.6.2. Mogućnost korištenja dodatnog modula

Korištenjem dodatnog modula rješavaju se navedeni problemi upravljanja stanjem aplikacije na način da se uvodi središnji spremnik podataka aplikacije. On će sadržavati stanje aplikacije i izvor je istine za komponente. Korištenjem ovog koncepta, uklanja se potreba za ručnom sinkronizacijom stanja između komponenti.

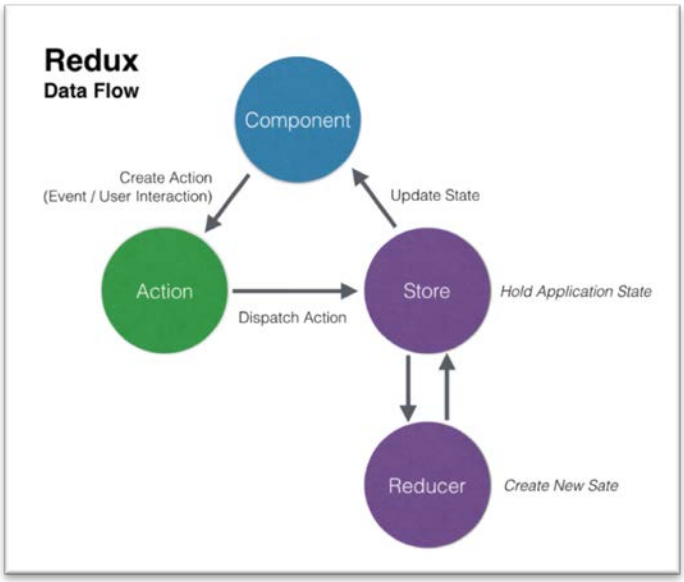
React nema službeni modul, no postoje dva koja se obično upotrebljavaju, a to su Redux i Flux. To znači da prva stvar koju treba odlučiti je koji od ova dva paketa, odnosno pristupa, koristiti. Postoji jako puno razmišljanja na ovu temu i odabrati pravi pristup i integrirati ga, u početku, može biti zaista teško. Ipak, jednom kada se postavi radi izvrsno, a sami paket redovito je održavan. U ovom radu prikazat će postavljanje Redux-a u aplikaciju. Prije početka samog rada sa Redux-om potrebno je instalirati dva modula naredom: „npm install redux react-redux --save“ (Slika 97.).

Slika 97. React.js instalacija modula za upravljanje stanjem

Izvor: obrada autora

Redux je modul koji omogućuje spremanje svih stanja u centralno mjesto tj. jedan Javascript objekt, dok react-redux modul omogućuje spajanje aplikacije izrađene u React-u sa Redux-om. Kontrola tijeka stanja upotrebom Redux-a sastoji se se od akcija, reduktora i skladišta, što je prikazano na slici ispod (Slika 98.).

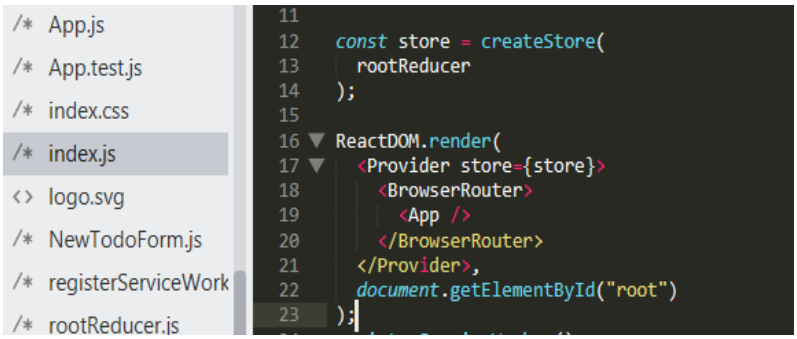
Slika 98. Redux kontrola tijeka stanja



Izvor: https://medium.com/codingthesmartway-com-blog/learn-redux-introduction-to-state-management-with-react-b87bc570b12a

Akcije se koriste za slanje informacija iz aplikacije u skladište. One sadrže opis buduće radnje koju će reduktor izvršiti i, ako je potrebno, dodatnih parametara koje zahtjeva specifična funkcija. Sve te informacije poslane u skladište potrebne su za promjenu stanja aplikacije nakon interakcije korisnika, internih događaja ili API poziva. Reduktori su čiste JavaScript funkcije koje uzimaju trenutačno stanje aplikacije i akcijski objekt i vraćaju novo stanje aplikacije. Skladište predstavlja središnji objekt koji drži stanje aplikacije. Skladište se kreira pomoću metode createStore iz biblioteke Redux kao na slici ispod (Slika 99.).

Slika 99. Redux skladište u React.js aplikaciji



Izvor: obrada autora

Kao glavni parametar skladištu je potrebno proslijediti glavni reduktor. Također, preporuča se korištenje <Provider> komponente kao omotača korijenske komponente aplikacije. Ona omogućuje da skladište bude dostupno svim komponentama u aplikaciji bez dodatnih postavki za svaku komponentu.

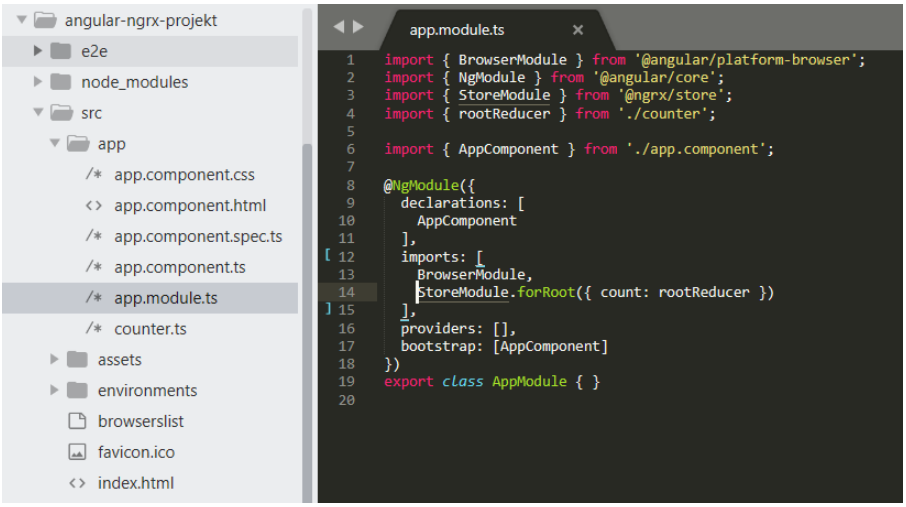
Angular, također, nema službeni modul za upravljanje stanjem. Ipak, moguće je uključiti modul razvijen od treće strane koji omogućuje upravljanje stanjem aplikacije putem observable-a. Observable-si pomažu u upravljanju asinkronim podacima, poput podataka koji dolaze sa backend-a. Ovaj paket se dobro integrira sa Angular FW-om upravo iz razloga što jedan i drugi koriste observable-se. Ngrx paket developeru daje jasan način kako upravljati stanjem, no problem leži upravo u činjenici da je ovo paket treće strane razvijen od skupine developera iz zajednice. Može postati zastarjel, rijetko održavan ili se, pak, cijelo vrijeme mijenjati (Looper, 2017). Ovaj modul je vrlo sličan Redux-u, a za početak rada, najprije je potrebno instalirati modul naredbom kao na slici ispod (Slika 100.).

Slika 100. Angular instalacija modula za upravljanje stanjem

Izvor: obrada autora

Sljedeći korak je registriranje modula u app.module.ts. Također, uveden je i glavni reduktor, a u imports je uveden novi objekt StoreModule te mu je proslijeđen glavni reduktor kao parametar (Slika 101.).

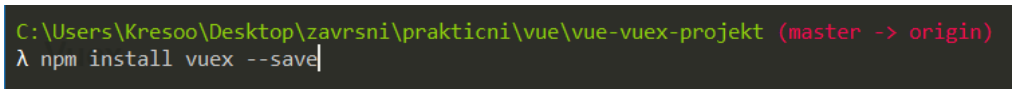
Slika 101. Angular registriranje modula za upravljanje stanjem



Izvor: obrada autora

Što se tiče Vue.js-a, ipak je drugačije. Ovaj FW ima Vuex, službeni paket koji se dobro uklapa u Vue.js i koji olakšava upravljanje stanjem i većim aplikacijama. Predstavlja centralnu pohranu za sve komponente u aplikaciji s pravilima koja osiguravaju da stanje može mutirati samo na predvidljiv način. Vuex ne ograničava način strukturiranja koda, tako da je potrebno naučiti kako pravilno upravljati slijedeći najbolje prakse. Ipak, ovo je službeni paket i iz tog razloga ovaj FW zaista pruža najbolji način upravljanja stanjem (Vue, 2018-8). Prije početka korištenja ovog modula potrebno ga je prvotno instalirati naredbom kao na slici ispod (Slika 102.).

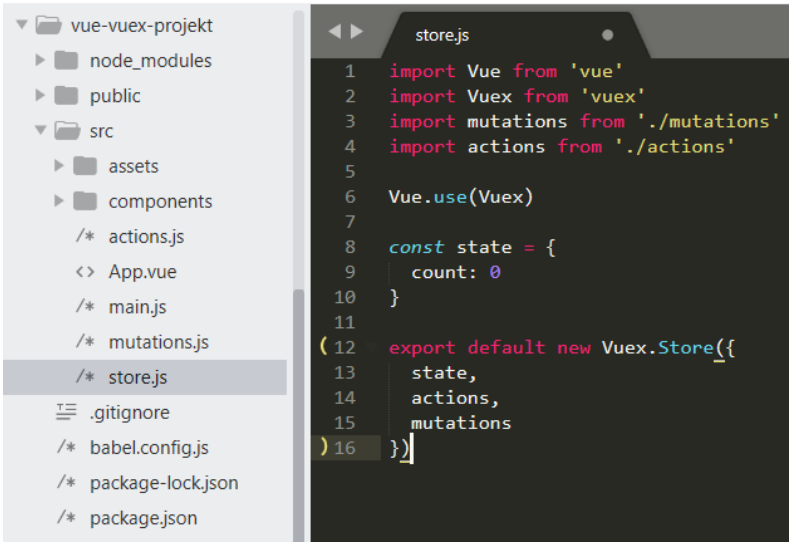
Slika 102. Vue.js instalacija modula za upravljanje stanjem



Izvor: obrada autora

Na sljedećoj slici (Slika 103.) uveden je Vuex te je na kraju izveden sa prosljeđenim parametrima, da se može koristiti u cijeloj aplikaciji.

Slika 103. Vuex skladište u Vue.js aplikaciji



Izvor: obrada autora

5. Diskusija analize

U prethodnom poglavlju provedena je usporedba triju FEFW. U prvom dijelu prethodnog poglavlja naglasak je stavljen na jednoobraznost i strukturiranje programskog koda. Vidljivo je da bez ikakavih problema jednostavan početak rada ubacivanjem skripte omogućuje Vue.js. Dovoljno je ubaciti jednu skriptu i započeti rad. FEFW s ovakvom funkcionalnošću u velikoj je prednosti kada je cilj upravljanje manjim dijelovima stranice. Također, takav početak rada omogućuje i React, no, ovaj FW ipak najbolje radi kada se koristi radni dijagram i ES6 sintaksa. Angular ne dopušta početak rada na ovakav način. Radni dijagram neophodan je u razvoju SPA aplikacije, a njegovu upotrebu danas omogućuju sva tri FEFW.

Drugi dio četvrtog poglavlja uspoređuje brzinu razvoja i usklađenosti komponenata u FEFW-ovima. Kada se radi o velikim aplikacijama, u kojima Javascript kontrolira većinu frontenda, ovaj dio je vrlo bitan. Jedino Angular ima jasno navedenu konvenciju pisanja datoteka i direktorija. React i Vue ipak nemaju, te je najbolje prakse potrebno pronaći u zajednici. Također, Angular jedini omogućuje automatsko generiranje pojedinih dijelova aplikacije, no budući da je način na koji funkcionira ovaj FW kompleksniji od ostala dva, to se ne može smatrati velikom prednošću. Ostala dva FW-a omogućuju automatsko generiranje samo početnog radnog dijagrama.

Treći dio uspoređuje mogućnost korištenja dodatnih modula. Ogromna zajednica i veliki broj dostupnih modula, zapravo, je najveća prednost Node.js-a u cijelini. Bez ovih modula stvaranje SPA nebi bilo moguće. S druge strane, prenatrpan FEFW nije prikladan za kontroliranje samo sitnog dijela DOM-a. Angular ima službeno izdane gotovo sve module, među kojima klijentski usmjerivač, modul za validaciju formi, pa čak i modul za dizajnerske komponente. Vue od ovih modula službeno izdan ima jedino klijentski usmjerivač, dok React nema niti jedan. Ipak, u zajednici postoji jako puno modula izgrađenih od strane trećih osoba, a koji su danas vrlo popularni, čime se znatno umanjuje mogućnost njihove zastare. React je, zasigurno, broj jedan

kada se radi o broju dostupnih modula, budući da je razvijen ranije, a pozadina mu je poznatija kompanija.

U četvrtom dijelu prethodnog poglavlja provjerava se da li FEFW službeno podržava distribuiranje izvođenja prikaznih elemenata na poslužitelju, koji je bitan faktor kod razvoja MPA. Angular i Vue imaju službeni vodič kako to izvesti. Proces postavljanja vrlo je sličan kod oba FEFW i uključuje postavljanje dosta konfiguracijskih postavki putem Webpack alata, koji je odgovoran za skupljanje komponenata, modula, ovisnosti i prevođenje koda. React još nema službenu podršku ove funkcionalnosti.

Peti dio ulazi u mogućnosti povećanja brzine pojedinačnih FEFW. Podijelu koda i lijeno učtavanje omogućuju sva tri FEFW. Za razliku od ostalih FEFW Angular ovo izvodi podosta komplicirano. Potrebno je kreirati novi usmjerivač za komponente djece kojeg treba izvesti i uvesti u korijenski usmjerivač. Zatim, potreban je novi modul za registraciju poseban za tu komponentu i njenu djecu, te je u korijenskom modulu potrebno uvesti komponentu na specifičan način. Kod Vue.js-a dovoljno je samo uvesti komponentu na specifičan način s Vue klijentskim usmjerivačem, dok React ovo omogućuje kroz paket treće strane, nakon čega je proces vrlo jednostavan, sličan onome u Vue.js-u. Dakle, Angular-u treba najviše koraka za postizanje ove funkcionalnosti, dok Vue.js-u najmanje. Mogućnost pisanja posebnih komponenata koje štede resurse ima samo React. Vue.js-u to nije ni potrebno jer automatski prepoznaje što je potrebno renderirati, a što ne. Angular funkcionira drugačije i podrazumijeva striktno definirano pisanje komponente. Korištenje metode za povećanje brzine imaju i React i Angular, dok Vue.js nema, a niti mu je potrebna, budući da sam prati ovisnosti među komponentama. Optimizaciju konzolarnim naredbama uspješno odrađuju sva tri FEFW. Vrlo je bitno izvršiti izgradnju projekta već u radnom dijagramu, kako nebi bilo potrebe slati kod zajedno s kompajlerom. Razlika je u tome što Angular kroz svoj komandni interfejs pruža najveći broj opcija i naredbi za specifične slučajeve.

Zadnji dio prethodnog poglavlja pokazuje kako svaki od uspoređivanih FEFW upravlja stanjem. Angular jedini omogućuje odvajanje logike u posebne, za tu funkcionalnost

specificirane, datoteke. Ovo može bitno olakšati nadzor nad stanjem aplikacije, no, ipak, ne mora u potpunosti riješiti ovaj problem ukoliko se radi o velikim i vrlo velikim aplikacijama, kada stablo komponenata i servisa naraste. Dodatni modul za upravljanje stanjem imaju sva tri FEFW, iako Vue.js jedini to omogućuje službeno. Dovoljno je instalirati modul, te ga, sukladno njegovoj arhitekturi i načinu upravljanja, početi koristiti.

Tablica 1. Odgovori na analizirana pitanja i podpitanja

Pitanja za analizu	**Podpitanja**	**Angular**	**React.js**	**Vue.js**
Jednoobraznost i strukturiranje izvornog programskog koda	Script	Ne	Da	Da
	Radni dijagram	Obavezan	Preporučljiv	Moguće sa i bez
Povećanje brzine razvoja i usklađenosti s drugim komponentama	Konvencija pisanja naziva datoteka i direktorija	Da	Ne	Djelomično
	Mogućnost automatskog generiranje dijelova aplikacije	Da	Djelomično	Djelomično
Korištenje dodatnih modula	Klijentski usmjerivač	Službeno izdan	Neslužbeno izdan u zajednici	Službeno izdan
	Modul za validaciju formulara	Službeno izdan	Neslužbeno izdan u zajednici	Neslužbeno izdan u zajednici

	Modul za dizajnerske komponente	Službeno izdan	Neslužbeno izdan u zajednici	Neslužbeno izdan u zajednici
Distribuiranje izvođenja prikaznih elemenata sa poslužitelja	Službena podržanost	Da	Ne	Da
Povećanje brzine izvođenja aplikacije	Podijela koda i lijeno učitavanje	Da (kompleksno)	Da uz modul treće strane	Da
	Mogućnost pisanja posebnih komponenata koje štede resurse	Ne	Da	Ne (nije ni potrebno)
	Korištenje metode za povećanje brzine	Da	Da	Ne (nije ni potrebno)
	Automatska optimizacija konzolarnim naredbama	Da (najviše opcija)	Da	Da
Upravljanje stanjem podataka	Mogućnost odvajanja logike u posebne datoteke	Da	Ne	Ne
	Mogućnost korištenja dodatnog modula	Neslužbeni modul	Neslužbeni modul	Službeni modul

Izvor: obrada autora

U tablici iznad (Tablica 1.) prikazani su odgovori na analizirana pitanja i podpitanja za svaki od uspoređivanih FEFW. Vidljivo je da Angular, za razliku od ostalih, ne podržava početak rada uvođenjem skripte, te je radni dijagram zaista potreban. S druge strane, ovaj FW ima izvrsno definiranu konvenciju pisanja naziva datoteka i direktorija. Jedini ima mogućnost automatskog generiranja pojedinih dijelova aplikacije, te još automatsku optimizaciju, izgrađen sustav upravljanja stanjem i sve bitne službeno izdane module. React.js, omogućuje početak rada ubacivanjem skripte, no pisanjem u ES5, a radni je dijagram preporučljiv. Ima vrlo malo službenih modula, te nema definiranu konvenciju imenovanja datoteka i direktorija. Za većinu toga potrebno je proučavati i istraživati. Vrlo je optimiziran već sam po sebi, no omogućuje još i podijelu koda, lijeno učitavanje, mogućnost pisanja posebnih komponenata i korištenje dodatnih metoda za optimizaciju. Vue.js je prilagodljiv, te pruža potpunu mogućnost izbora početka rada ubacivanjem skripte ili, pak, kroz radni dijagram. Nema striktno definiranu konvenciju pisanja naziva datoteka i direktorija. Ima službeno izdane klijentski usmjerivač i modul za upravljanje stanjem, te vodič za distribuiranje prikaznih elemenata sa poslužitelja. Gotovo da nije potrebno dodatno optimizirati, a proces izvođenja zadanih funkcionalnosti, u većini slučajeva, jednostavniji je nego kod ostala dva promatrana FEFW.

6. Zaključak

Moderne web-aplikacije, zbog funkcionalnosti koje omogućuju u korisničkom sučelju, imaju složenu programsku strukturu. Ručno pisanje programskog koda zbog složenosti cijele aplikacije može rezultirati neujednačenom kvalitetom i sadržajem pojedinih aplikacijskih djelova. Održavanje tako razvijanih aplikacija otežano je. Zbog toga se web-aplikacije često razvijaju korištenjem različitih FW. FW omogućuje strukturiranje, jednostavnije i ujednačenije pisanje programskog koda, te time olakšava održavanje web-aplikacije. Postoji puno FW koji se mogu koristiti u razvoju web-aplikacija, i to za različite dijelove aplikacije, a oni analizirani u ovom radu koriste se u razvoju front end dijela web-aplikacije. Prema načinu izvođenja web-aplikacije mogu biti višestranične (MPA) ili jednostranične (SPA). Budući da SPA još ne može zamijeniti tradicionane MPA u pogledu optimizacije na tražilicama, bilo bi idealno kada bi postojao FEFW koji je optimiziran za izgradnju i jednih i drugih vrsta aplikacija. U radu su uspoređivana tri trenutno napoznatija FEFW: Angular, React.js i Vue.js.

Angular FEFW prvi je razvijen od sva tri promatrana. Ovaj FW, zbog svoje veličine, kompleksnosti i potrebe za visokom optimizacijom (koja se izvršava u radnom dijagramu), nije prilagođen za kontrolu sitnih dijelova DOM-a, tj. za razvoj MPA aplikacija. S druge strane, sadrži gotovo sve službeno izdane i održavane module za izgradnju kompletne SPA. React.js, prvotno izgrađen kao knjižnica, no kasnije razvijen uz pomoć zajednice u kompletan FEFW, ima vrlo malo službenih paketa i za većinu toga potrebno je istraživati. Ipak, zbog svoje iznimne popularnosti, ima vrlo velik broj modula treće strane koji su danas već postali popularni i vođeni od zajednice, čime se mogućnost zastare značajno smanjiva. Vue.js, najnoviji je od uspoređivanih FEFW, te, osim što su većina modula dostupni i službeno izdani, zapravo, spaja najbolje iz Angular-a i React.js-a, kao što su prilagodjiviji početak rada, više službenih modula, pa čak i oni za upravljanje stanjem i SSR. Manje je potrebno optimizacije jer je gotovo sve automatizirano, podjela koda i lijeno učitavanje je izvedeno jednostavnije, a većina toga se izvodi kroz manje koraka i manje toga treba učiti. Prilagodljiv je, te omogućuje optimiziranu izgradnju i MPA i SPA, što mu u ovom radu daje prednost nad ostala dva promatrana FEFW.

Popis kratica

FW - Framework

FEFW - Frontend framework

MPA - Multi Page Aplication

SPA - Single Page Aplication

HTML - Hypertext Markup Language

AJAX - Asynchronous JavaScript And XML

DOM - Document Object Modelž

SEO - Search engine optimization

JSX - Javascript XML

DRY - Don't repeat yourself

ES5 - ECMAScript 5

ES6 - ECMAScript 6

CLI - Command Line Interface

URL - Uniform Resource Locator

CSS - Cascading Style Sheets

SSR - Server Side Rendering

JIT - Just-in-Time

AOT - Ahead-of-Time

HTTP - Hypertext Transfer Protocol

Popis literature

1. Abranov, D. (2016) *Create Apps with No Configuration* https://reactjs.org/blog/2016/07/22/create-apps-with-no-configuration.html (17.12.2018.)
2. Andersen, B. (2017) *Understanding the Virtual DOM* https://codingexplained.com/coding/front-end/vue-js/understanding-virtual-dom (13.12.2017.)
3. Angular (2018-1) *Architecture Overview* https://angular.io/guide/architecture (23.12.2017.)
4. Angular (2018-2) *Webpack* https://angular.io/guide/webpack(23.12.2017.)
5. Angular (2018-3) *Angular Universal: server-side rendering* https://angular.io/guide/universal (19.12.2017.)
6. Angular (2018-4) *Style Guide* https://angular.io/guide/styleguide (17.12.2018.)
7. Angular (2018-5) *Npm Packages* https://angular.io/guide/npm-packages (22. 12.2017.)
8. Angular (2018-6) *The Ahead-of-Time (AOT) Compiler* https://angular.io/guide/aot-compiler (11.01.2018.)
9. Angular (2018-7) *Routing & Navigation* https://angular.io/guide/router Schepenaar (11.03.2018.)
10. Angular University. (2016) *Angular 2 vs React: The Ultimate Dance Off* https://medium.com/javascript-scene/angular-2-vs-react-the-ultimate-dance-off-60e7dfbc379c (23.12.2017.)
11. Apps Team (2013) *An Intro Into Single Page Applications (SPA)* https://blog.4psa.com/an-intro-into-single-page-applications-spa/ (10.12.2017.)
12. Baghel, A. S. (2017) *Software Design Principles DRY, KISS, YAGNI* https://www.c-sharpcorner.com/article/software-design-principles-dry-kiss-yagni/ (10.02)
13. Bejar, J. (2017) *A Review Of Server Side Rendering In Javascript Frameworks* https://wyeworks.com/blog/2017/9/6/a-review-of-ssr-in-javascript-frameworks (17.12.2018.)

14. Bersling (2017) *State Management: ngrx/store vs Angular services* https://www.bersling.com/2017/06/05/state-management-ngrxstore-vs-angular-services/ (11.01.2018.)
15. Bruce, J. (2017) *How Do Search Engines Work?* http://www.makeuseof.com/tag/how-do-search-engines-work-makeuseof-explains/ (13.12.2017.)
16. Burgess, M. (2016) *JavaScript Frameworks Are Great* https://medium.com/@mattburgess/javascript-frameworks-are-great-2df4a3f0b24d(10.12.2017.)
17. Clockwise Software (2017) *Frontend frameworks showdown: Angular vs. React vs. Vue* https://clockwise.software/blog/angular-vs-react-vs-vue/ (13.12.2017.)
18. Cordle, C. (2017) *Why Angular 2/4 Is Too Little, Too Late* https://medium.com/@chriscordle/why-angular-2-4-is-too-little-too-late-ea86d7fa0bae (13.01.2018.)
19. Crawford, C. (2016) *What the Flux? An Overview of the React State Management Ecosystem* https://thenewstack.io/flux-overview-react-state-management-ecosystem/ 13.01.2018.)
20. Delaney, J. (2017.) *How to Lazy Load Components in Angular 4 in Three Steps* https://angularfirebase.com/lessons/how-to-lazy-load-components-in-angular-4-in-three-steps/ (13.03.2018.)
21. Dias, A. (2016) *Challenges in a (really) large single-page application - how to optimize script downloads* http://alvarodias.org/articles/challenges-in-a-really-large-single-page-application-how-to-optimize-script-downloads (10.12.2017.)
22. Dimi (2017) *Discover Single-Page Vs. Multipage Design: Pros & Cons (2017)* https://themefuse.com/single-page-vs-multipage-design/ (20.12.2017.)
23. Emmit, A., Scott, Jr., (2016.) *SPA Design and Architecture: Understanding single-page web applications* https://github.com/transidai1705/javascript-ebooks/blob/master/%5BSPA%20Design%20and%20Architecture%20Understanding%20Single%20Page%20Web%20Applications%201st%20Edition%20by%20Emmit%20Scott%20-%202016%5D.pdf (20.11.2017.)

24. Fink, G., Flatow, I., (2014.) *Pro Single Page Application Development: Using Backbone.js and ASP.NET* http://pepa.holla.cz/wp-content/uploads/2015/10/Pro-Single-Page-Application-Development.pdf (15.11.2017.)
25. Hein, R. (2010) *How Many Users Have JavaScript Disabled* https://www.searchenginepeople.com/blog/stats-no-javascript.html (13.12.2017.)
26. Köhr, J., Schlaudraff, J. (2017) *Angular 2 in a multi-page application* https://blog.novatec-gmbh.de/angular-2-in-a-multi-page-application/ (20.12.2017.)
27. Kurian, G., G. (2017) *How Virtual-DOM and diffing works in React* https://medium.com/@gethylgeorge/how-virtual-dom-and-diffing-works-in-react-6fc805f9f84e (13.12.2017.)
28. Looper, J. (2017) *An Introduction to Observables for Angular Developers* (11.01.2018.) https://developer.telerik.com/topics/web-development/introduction-observables-angular-developers/
29. Mikowski, M. (2014) *How to optimize single-Page sites for search engines* https://www.webdesignerdepot.com/2013/10/how-to-optimize-single-page-sites-for-search-engines/ (13.12.2017.)
30. Neoteric (2016) *Single-page application vs. multiple-page application* https://medium.com/@NeotericEU/single-page-application-vs-multiple-page-application-2591588efe58 (20.12.2017.)
31. Neuhaus, J. (2017) *Angular vs. React vs. Vue: A 2017 comparison* https://medium.com/unicorn-supplies/angular-vs-react-vs-vue-a-2017-comparison-c5c52d620176 (10.12.2017.)
32. Ngrx (2017) *Ngrx Store* https://github.com/ngrx/store (05.01.2018.)
33. Npmjs (2013-2017) *React Packages* https://www.npmjs.com/search?q=react&page=1&ranking=popularity (22. 12.2017.)
34. Osbourne, S. (2017) *JavaScript Framework Battle: 'Hello World' in each CLI* https://medium.com/dailyjs/javascript-framework-battle-hello-world-in-each-cli-cfdba8bf5e4b (05.01.2018.)
35. Rangle (2017) *Angular 2 Training Book* https://angular-2-training-book.rangle.io/handout/state-management/ (05.01.2018.)

36. React (2018-1) *React.js* https://reactjs.org/ (13.12.2017.)
37. React (2018-2) *Introducing JSX* https://reactjs.org/docs/introducing-jsx.html (13.12.2017.)
38. React (2018-3) *Try React* https://reactjs.org/docs/try-react.html (23.12.2017.)
39. React (2018-4) *React Without ES6* https://reactjs.org/docs/react-without-es6.html (19.12.2017.)
40. React (2018-5) *Optimizing Performance* https://reactjs.org/docs/optimizing-performance.html (01.07.2018.)
41. React Community (2017) *Server Rendering* https://github.com/reactjs/redux/blob/master/docs/recipes/ServerRendering.md (19.12.2017.)
42. Saxena, R. (2014) *Single Page Application (SPA) Using AngularJS, Web API and MVC 5* http://www.c-sharpcorner.com/uploadfile/rahul4_saxena/single-page-application-spa-using-angularjs-web-api-and-m/ (10.12.2017.)
43. Schepenaar, W. (2017) *Server-side vs Client-side Routing* https://medium.com/@wilbo/server-side-vs-client-side-routing-71d710e9227f
44. Shimanovsky, S. (2016) *How is PowerApps giving control back to the business?* http://www.eikospartners.com/blog/multi-page-web-applications-vs.-single-page-web-applications (10.12.2017.)
45. Takada, M. (2013) *Single page apps in depth* http://singlepageappbook.com/index.html (15.11.2017.)
46. TypeScript (2017) *Typescriptlang* https://www.typescriptlang.org/index.html (23.12.2017.)
47. Vue (2018) *Plugins* https://vuejs.org/v2/guide/plugins.html (22. 12.2017.)
48. Vue (2018-1) https://vuejs.org/ (13.12.2017.)
49. Vue (2018-2) *Explanation of Different Builds* https://vuejs.org/v2/guide/installation.html#Explanation-of-Different-Builds (23.12.2017.)
50. Vue (2018-3) *Introduction* https://vuejs.org/v2/guide/ (19.12.2017.)
51. Vue (2018-4) *Server-Side Rendering* https://vuejs.org/v2/guide/ssr.html (19.12.2017.)
52. Vue (2018-5) *Vue.js Server-Side Rendering Guide* https://ssr.vuejs.org/en/ (17.12.2018.)

53. Vue (2018-6) *Component Naming Conventions* https://vuejs.org/v2/guide/components.html#Component-Naming-Conventions (17.12.2018.)
54. Vue (2018-7) *Runtime + Compiler vs. Runtime-only* https://github.com/vuejs/vue/tree/dev/dist#runtime--compiler-vs-runtime-only (27.12.2018.)
55. Vue (2018-8) *State Management* https://vuejs.org/v2/guide/state-management.html (13.01.2018.)
56. Vue (2018-9) *Form Validation* https://vuejs.org/v2/cookbook/form-validation.html (13.03.2018.)
57. Yiang, Y., Z. (2016) *Angular 2 is terrible* https://meebleforp.com/blog/36/angular-2-is-terrible (19.12.2017.)
58. Zanon, D. (2015) *AngularJS: How to create a SPA crawlable and SEO friendly?* https://zanon.io/posts/angularjs-how-to-create-a-spa-crawlable-and-seo-friendly (13.12.2017.)
59. Zaytsev, J. (2018) *Combat Table* https://kangax.github.io/compat-table/es6/ (23.12.2017.)

Popis slika

Popis tablica