Univerzitet u Novom Sadu
Tehnički fakultet „Mihajlo Pupin“
Zrenjanin

Primena MongoDB, Express i Angular Javaskript radnog okvira u izradi veb aplikacije

Using MongoDB, Express an Angular Javascript frameworks in web application development

- Diplomski rad -

Student: Jovan Đukić
Smer: Informacione tehnologije
Broj indeksa: IT 11/15

Zrenjanin 2019.

Univerzitet u Novom Sadu
Tehnički fakultet „Mihajlo Pupin“
Zrenjanin

Primena MongoDB, Express i Angular Javaskript radnog okvira u izradi veb aplikacije

Using MongoDB, Express an Angular Javascript frameworks in web application development

- Diplomski rad -

Mentor: Doc. dr Ljubica Kazi

Student: Jovan Đukić
Smer: Informacione tehnologije
Broj indeksa: IT 11/15

Zrenjanin 2019.

Sadržaj

1. UVOD

Popularnost interneta u savremenom dobu je dostigla astronomske razmere. Nekada je računar bio luksuz rezervisan za državne institucije i velike kompanije. Danas, računari ne da postoje u skoro svakom domu u svetu, nego i u džepu, pa čak i u automobilu i frižideru. I svi su povezani internetom.

I potrebe korisnika su se takođe povećale. Aplikacije na internetu moraju biti brze, uvek dostupne, da obrađuju veliku količinu podataka, da zadovolje razne potrebe savremenog čoveka, od razonode do primene u svakodnevnom poslovanju.

Sa druge strane, tradicionalnim tehnologijama postaje sve teže postići ove ciljeve. Razvoj traje dugo, skupo i teško se prilagođava čestim promenama. Rešenje za ovaj problem nalazi se u Javaskriptu, jeziku kojim je moguće podmiriti sve ove potrebe.

Ovaj radi se bavi primenom Javaskript programskog jezika u kreiranju savremenih veb aplikacija, od korisničkog interfejsa, preko biznis logike do baze podataka - sve jednim jezikom, na jednoj platformi.

Poglavlje “Teorijske osnove” se bavi nerelacionim bazama podataka, kao fleksibilnim zamenama klasičnim relacionim bazama, ulozi biblioteka i radnih okvira u razvoju aplikacija, kao i vrstama programskih jezika.

Poglavlje “Postojeća rešenja” opisuje nekoliko svetski poznatih brendova koje svoje poslovanje i rade zasnivaju na Javaskript platformi.

U četvrtom poglavlju “Opis primenjenih tehnologija” dat je sažet uvod u tehnologije koje su korištene u poglavljima 5 i 6 za kreiranje primera jedne aplikacije koristeći te tehnologije.

2. TEORIJSKE OSNOVE

2.1 Nerelacione baze podataka

Nerelacione baze podataka su nov tip baza podataka, nastale kao potreba za fleksibilnijim i dinamičnijim radom sa velikim količinama podataka. Takođe se nazivaju i NOSQL baze podataka. Termin NOSQL odnosi se na činjenicu da one ne koriste SQL za upite. [1]

Nerelaciona baza podataka ne skladišti podatke u tabelama sa redovim i kolonama, ka što to rade tradicionalne relacione baze. Umesto toga, one podatke skladište u JSON formatu (Javascript Object Notation).

Model podataka ne mora biti unapred definisan, niti da prati određenu strukturu. Baze su dovoljno fleksibilne da se struktura podataka može promeniti u svakom momentu.

Entiteti se grupišu u kolekcije, što je analogno tabelama u RDBMS. Kolekciju mogu biti registrovani korisnici, automobili u taksi kompaniji, studenti na fakultetu i sl. Jedan član kolekcije naziva se dokument, što odgovara redu u tabeli.

Umesto SQL upitom, podaci se manipulišu sa sredstvima koje sama baza pruža, a to je često prosto pozivanje funkcija i snabdevanje željenim argumentima da bi se podatak uneo, izmenio ili obrisao.

Najvažnija odlika NOSQL baza je ogromna mogućnost da se lako radi sa velikim količinama podatka, zbog mogućnosti repliciranja i granuliranja, što je kod SQL relacionih teško uraditi.

2.2 Softverski razvojni okviri

Razvojni okviri (frameworks) su alati koji se koriste za brže i jednostavnije razvijanje aplikacija. Mnoge funkcionalnosti se ponavljaju u svakoj aplikaciji, a takođe i kod njihovih implementacija. Kako ne bi došlo do ponavljanja koda, trošenje vremena na ponovno razvijanje istih funkcionalnosti koje je podložno greškama, a samim tim i novca, radni okviri služe kao apstrakcija za često ponavljan kod.

Sa velikom količinom unapred napisanog koda i implementiranih funkcionalnosti, razvoj aplikacija je brži, lakši i jeftiniji. Ukoliko na aplikaciji radi više ljudi i koriste isti radni okvir, u mogućnosti su da daleko bolje sarađuju.

Razvojni okvir je sličan biblioteci koda. Za razliku od biblioteke, okvir diktira način na koji će teći tok razvoja aplikacije. Kod samog radnog okvira se ne menja, ali se može proširiti dodavanjem novih funkcionalnosti.

Za svaki programski jezik postoje radni okvir. Za Javu je popularan “Spring”, za Rubi “Ruby On Rails”, za Pajton je “Django”. Najpoznatija Javaskript biblioteka je “Jquery”, a radni okviri su “Angular”, “ReactJS” i “VueJS”.

2.3 Skriptni programski jezici

Programski jezici predstaljaju način davanja komandi računara. Računari ne razumeju ljudski jezik, a mašinski jezik sastavljen od 0 i 1 teško je razumljiv većini ljudi. Kako bi se olakšalo pisanje programa, uvedeni su programski jezici koji ne koriste 0 i 1, već slova, koja su ljudim mnogo razumljivija. Slovne komande su bile ekvivalentne brojčanim.

Potreba za kompleksnijim programima je rasla, pa nastaju kompajlirani jezici. Ovi jezici su jednom komandom izvršavali više mašinskih instrukcija. Kompajler je program koji ove jezike prevode u mašinski, koje računari mogu da izvrše.

Skriptni jezici su dinamički jezici višeg nivoa. Oni se ne kompajliraju, već interpretiraju. Interpreter čita svaku liniju koda, i izvršava jednu po jednu. Izvršavaju se na već postojećim sistemima.

U odnosu na kompajlirane jezike, interpretirani imaju jednostavniju sintaksu, lakše se nauče, a programi pišu brže sa manje koda. Stvari poput upravljanja memorijom i slične kompleksne operacije obavlja interpreter, dok se čovek fokusira na funkcionalnost programa.

Nedostaci su im što su često sporiji, troše više memorije i podložniji su greškama.

Poznati skriptni jezici su Perl, PHP, Pajton i Javaskript.

Javaskript je skriptni jezik koji se dugo vremena koristio isključivo za dodavanje dinamičkih i interaktivnih elemenata veb stranicama. Izvršavao se samo u veb pretraživaču, pa kreiranje aplikacija kao što to rade programski jezici Java, C# ili Pajton, nije bilo moguće.

Nije posedovao podršku za objektno orijentisano programiranje kao što to imaju Java, C# ili C++, pa pojmovi poput objekta i klase u Javaskriptu imaju veoma drugačije značenje. Šta više, svaki od pretraživača je imao drugačiju implementaciju Javaskripta, pa je i sam jezik bio drugačiji od pretraživača do pretraživača. Svi ovi nedostaci su učinili da Javaskript bude samo mali skriptni jezik za prosto manipulisanje elementima na veb stranici.

Preokret doživljava nakon standardizacije jezika od strane ECMA asocijacije, koja se bavi standardizacijom informacionih i komunikacionih sistema [2]. Prva verzija ovog standarda izlazi 1997. godine, a druga 1998.

Verzija 3 iz 1999. godine je prvo veliko izdanje koje je dalo Javaskriptu ogromnu popularnost. I danas, kada kažemo Javascript, misli se upravo na ovaj standard. Svi popularni veb pretraživači podržavaju ovaj standard čak i danas. Ubrzo potom nastaje i AJAX (asynchronous javascript and xml) tehnika, koja omogućuje da se delovi veb stranice osveže novim sadržajem bez ponovnog učitavanja cele veb stranice.

Zahvaljujući popularnosti AJAX-a, nastaje JSON (Javascript Object Notation) kao format za transport podataka između klijenta i servera, istiskujući iz upotrebe do tada korišten XML.

Godine 2009. nastaje Node.js, više-platformsko Javaskript izvršno okruženje otvorenog koda[3], koje omogućava da se Javaskript izvršava van veb pretraživača. Sada je moguće pisati i klijent i serverske aplikacije u istom programskom jeziku.

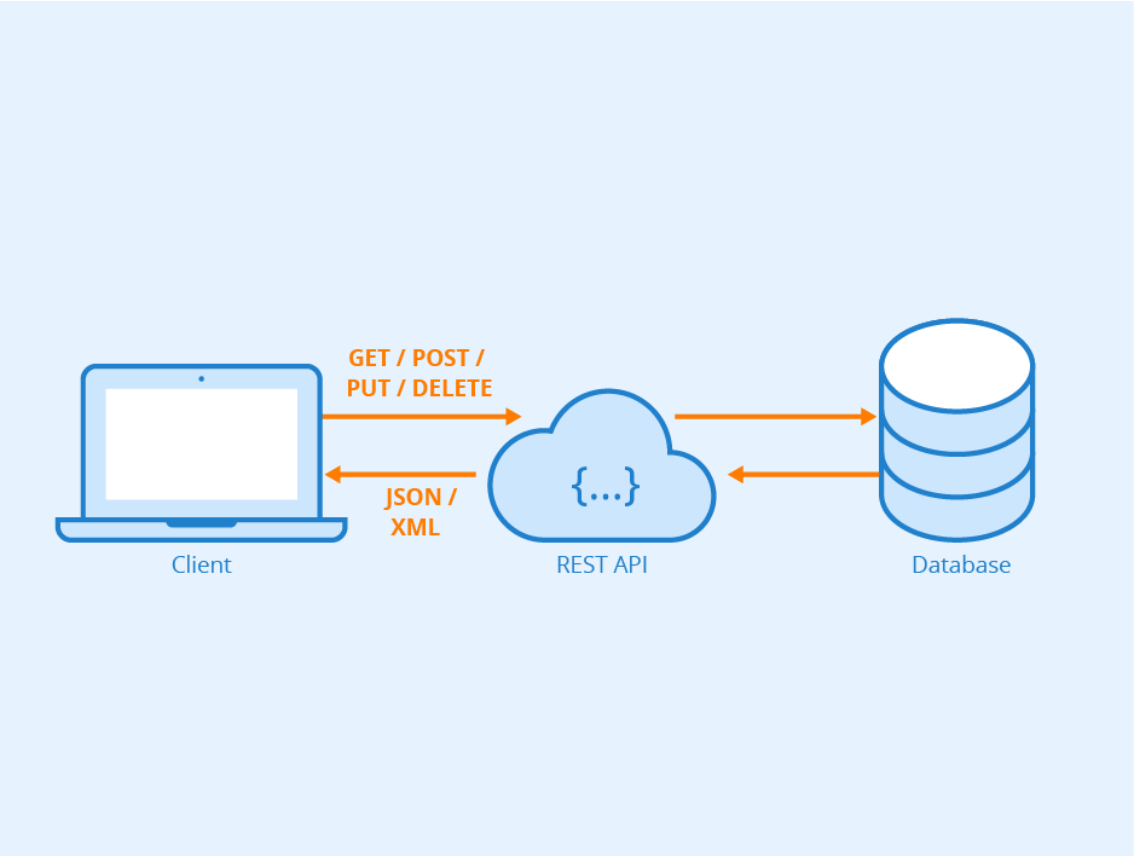
MongoDB je besplatna, nerelaciona NOSQL baza podataka, koja podatke cuva u JSON formatu, isti format koji klijentska i serverska aplikacija koriste za komunikaciju. Podaci se čuvaju u ovom obliku bez potrebe da se pretvaraju u tabele ili relacije.

Uz ove tehnologije, kao i ogroman broj radnih okvira, biblioteka i pomoćnih programa, veb aplikacije se cele pišu u samo jednom jeziku - klijent, server i baza podataka. Popularna grupa tehnologija koje se koriste u ove svrhe čine navedeni MongoDB baza podataka, Express.js serverski Javaskript radni okvir, Angular klijentski radni okvir i Node.js kao izvršno okruženje. Ove tehnologije se često zajednički nazivaju “MEAN stack”.

Javaskript jezik ne treba mešati sa jezikom Java. Java je objektno-orijentisani programski jezik kompanije Oracle, i osim imena, nema dodirnih tačaka sa Javaskriptom.

2.4 REST arhitektura

Aplikacije pisane u “MEAN” grupi tehnologija, najčešće koriste REST (Representational State Transfer) arhitekturalni stil. Ovaj stil se koristi za dizajniranje slabo povezanih aplikacija preko HTTP protokola, koji se često koristi za razvoj web servisa. REST ne podstiče nikakva pravila za implementiranje, već viši nivo smernica za dizajn koje treba pratiti tokom samostalnog implementiranja[4]. Za razliku od SOAP-a (Simple Object Access Protocol), REST nije protokol i nema nikakvih pravila i obaveza.



Slika 1. REST API Arhitektura[5]

Principi REST arhitekture:[6]

1. **Klijent - server**: odvajanje sloja korisničkog interfejsa od sloja podataka, povećava se prenosivost korisničkog interfejsa, povećava skalabilnost uprošćavanjem serverskih komponenti.
2. **Bez čuvanja stanja**: svaki zahtev klijenta upućen serveru mora da sadrži sve potrebne informacije da bi zahtev bio razumiv, i ne može koristiti informacije skladištene na serveru. Stoga se sesija čuva u potpunosti kod klijenta.
3. **Mogućnost keširanja**: Ograničenja keširanja zahtevaju da odgovor servera na zahtev klijenta mora implicitno ili eksplicitno navesti da li podaci u odgovoru imaju mogućnost keširanja. Ukoliko ima, klijentski keš ima pravo da ponovno iskoristi podatke iz odgovora za kasnije, identične zahteve.
4. **Ujednačeni interfejs**: primenjivajući princip generalnosti na interfejs komponente, cela arhitektura sistema se uprošćava i vidljivost interakcija je poboljšana. Da bi se postigao ujednačeni interfejs, više arhitekturalnih ograničenja treba upravljati ponašanjem komponente. REST je definisan sa 4 ograničenja interfejsa: identifikacija resursa, manipulacija resursa putem reprezentacije, samo-opisne poruke, i hipermedija kao sredsto za sesiju aplikacije.
5. **Slojeviti sistem**: sistem slojevitosti omogućava arhitekturi da bude sačinjena od hijerarhijskih slojeva tako što će ograničiti ponašanje komponente tako što ona neće moći da “vidi” dalje od neposrednog sloja sa kojim vrši interakciju.

6. **Kod po potrebi (nije obavezno)**: REST dozvoljava da se funkcionalnost klijenata proširi preuzimanjem i izvšavanjem koda u vidu apleta i skripti. Ovim se klijenti uprošćavaju jer imaju manji broj svojstava koje moraju implementirati.

Ključna apstrakcija informacije u REST-u je **resurs**. Resurs može biti više stvari: dokument, slika, servis, grupa drugih resurs, osoba itd. REST koristi **identifikator resursa** da odredi željeni resurs koji je u interakciji između komponenti.[6]

Stanje resursa u bilo kom datom trenutku naziva se **reprezentacija resursa**[6]. Reprezentacija sadrži podatke, metapodatke podataka i hipermedija linkove koji pomažu klijentu u potrazi ka drugim željenim stanjima.

Tri sloja REST arhitekture čine

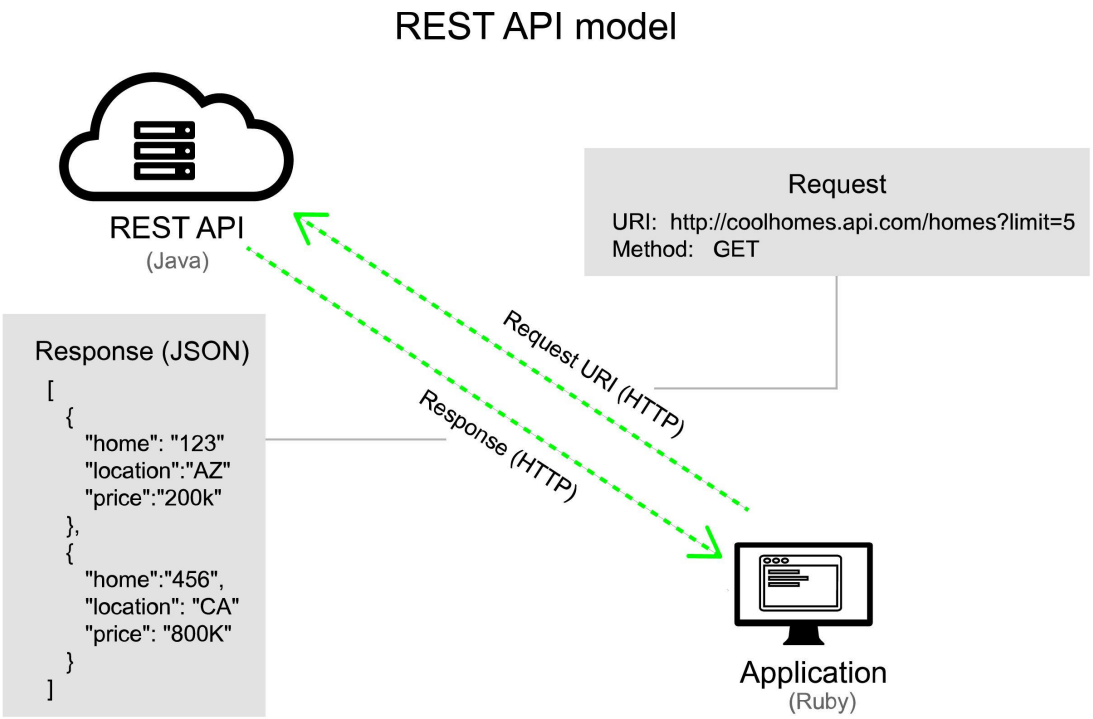
– Klijent, prezentacioni sloj
– Server API(Application Programming Interface), sloj biznis logike
– Baza podataka, perzistentni sloj podataka.

Za razliku od MVC šablona, prezentacioni sloj je samostalna aplikacija. Ona ni na koji način ne zavisi od ostalih slojeva, samostalna je, i komunicira jedino sa API serverom, od kojeg zahteva podatke za prezentaciju kranjim korisnicima, ili šalje podatke koje unose korsnici koje server obrađuje ili prosleđuje bazi podataka na čuvanje.

U REST arhitekturi ne mora postojati samo jedna klijentska aplikacija. Može ih biti više, a broj im nije ograničen. Npr. može postojati veb aplikacija, kojoj korisnici pristupaju preko veb pretraživača, mobilna aplikacija na android ili ios uređajima, i desktop aplikacija, koja radi na windows, mac ili linux operativnom sistemu. Svi ovi klijent komuniciraju sa istim serverom.

Broj servera takođe nije ograničen. Jedan klijent može komunicirati sa više različitih servera, zahtevati njihove podatke i prosleđivati drugim serverima.

Application Programming Interface je aplikacija koja se izvršava na serveru. Njen zadatak je da posreduje između klijenata i baze podataka. Ona definiše resurse i metode za pristupanje, koje klijenti zahtevaju koristeći HTTP protokol. Npr. resurs mogu biti korisnici, a identifikator tog resura može biti npr. http://www.primer.com/korisnici. Dalje, definiše metode i operacije nad ovim resursom. Najčešće operacije su operacije kreiranja, čitanja, menjanja i brisanja CRUD (Create, Read, Update, Delete).



Slika 2. Primer komunikacije klijenta i servera u REST arhitekturi [7]

HTTP metodama kao što su GET i POST, naznačava se koja se operacija izvršava nad datim resursom. Npr. klijent šalje GET zahtev za resurs http://www.primer.korisnici. Na osnovu HTTP metode i resursa, API server zna da treba izvršiti operaciju čitanja nad resursom “korisnici”. On će potom pronaći podatke o tom resursu, bilo iz baze podataka ili od nekogu drugog API servera, i poslati ga nazad klijentu.

Najčešće korištene HTTP metode su GET, POST, PUT/PATCH i DELETE. POST služi za kreiranje novog resursa (Create), PUT za zamenu starog resursa novim, a PATCH za parcijalnu zamenu (Update) i DELETE za brisanje (delete).

Ovi nazivi metoda imaju svojstvo samo obeležja, a ne funkcionalnosti. Serveri mogu da koriste različite metode za različite operacije, i ako je preporučljivije da se to ne radi. Takođe, individualne implementacije klijenta i servera nisu bitne. Server može biti Java aplikacija, a klijent Pajton i obrnuto.

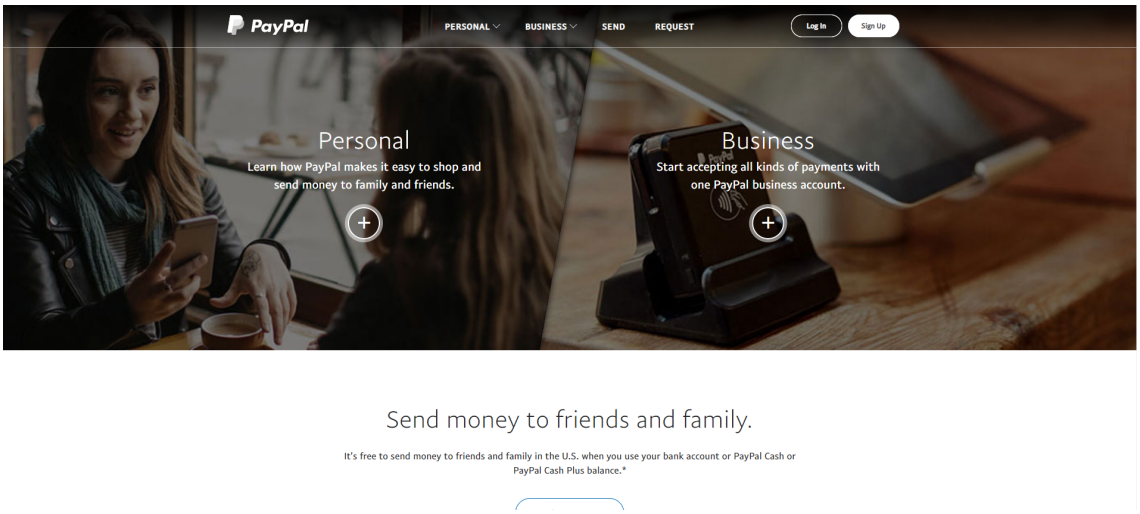
RESTful API HTTP methods

Resource	GET	PUT	POST	DELETE
Collection URI, such as `http://api.example.com/resources/`	**List** the URIs and perhaps other details of the collection's members.	**Replace** the entire collection with another collection.	**Create** a new entry in the collection. The new entry's URI is assigned automatically and is usually returned by the operation.[10]	**Delete** the entire collection.
Element URI, such as `http://api.example.com/resources/item17`	**Retrieve** a representation of the addressed member of the collection, expressed in an appropriate Internet media type.	**Replace** the addressed member of the collection, or if it does not exist, **create** it.	Not generally used. Treat the addressed member as a collection in its own right and **create** a new entry in it.[10]	**Delete** the addressed member of the collection.

Slika 3. Primeri HTTP metoda i operacije nad resursima

3. POSTOJEĆA REŠENJA

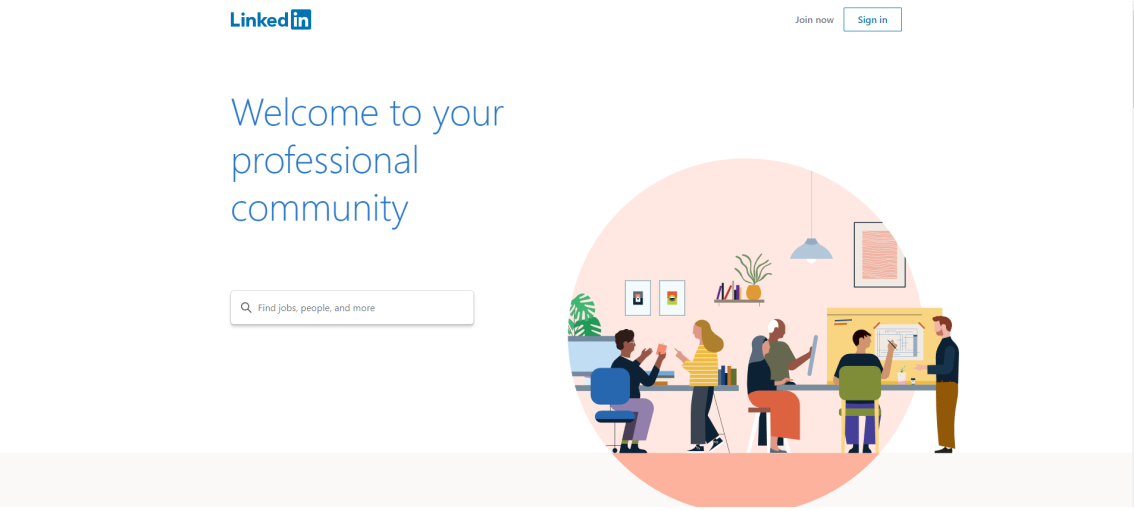
3.1 Paypal



Slika 4. Paypal [8]

Paypal je platforma za online plaćanja, koju je osnovao Elon Mask. Predstavlja najpopularniji metoda za plaćanja preko interneta i online trgovinu u čitavom svetu. Ima 220 miliona korisnika i 7.6 milijardi platežnih transakcija. Sa velikim brojem novih članova, Paypal-u je bilo potrebno rešenje za pružanje kvalitetnijih usluga svojim korisnicima. Prelaskom na NodeJS, aplikacija je napravljena duplo brže sa manje ljudi, 33% manje linija koda i 40% manji broj fajlova, u odnosu na prethodnu aplikaciju pisanu u Javi. Učitavanje stranice se ubrzalo za 200ms, i prosečno vreme odgovora smanjilo se za 35% [9].

3.2 Linkedin

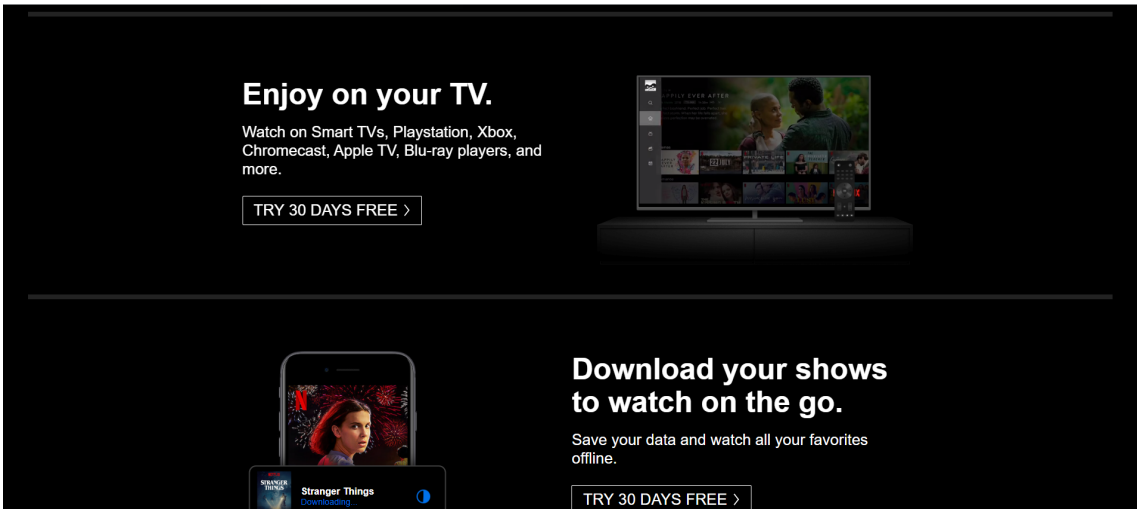


Slika 5. Linkedin [10]

Linkedin je biznis-orijentisana društvena mreža nastala 2002. godine u Kaliforniji. Omogućava korisnicima da jedni drugima postanu poslvona konekcija. Dostupan je u 24 jezika, koristi ga 400 miliona korisnika iz 200 zemalja. Koristi NodeJS. za server njihove mobilne aplikacije.

U odnosu na prethodnu verziju servera, koji je pisan u jeziku rubi i radnom okviru “ruby on rails”, mobilna aplikacija je 20 puta brža, a broj potrebnih servera smanjen je sa 30 na 3. Vreme razvoja je takođe bilo znatno brže. [11]

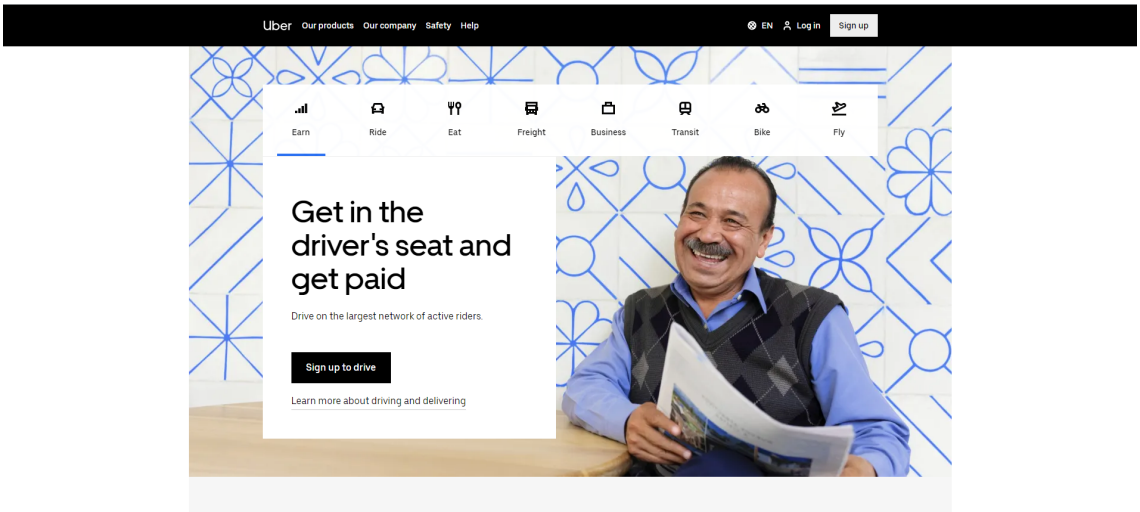
3.3 Netflix



Slika 6. Netflix [12]

Netflix je najveći svetski provajder video striming usluga. Nudi preko 5500 naslova. 120 miliona korisnika u 190 zemalja sveta gleda video sadržaj 100 miliona sati dnevno. Prelaskom na NodeJS, uočili su rast produktivnosti i performansi aplikacije. Pokretanje aplikacije se smanjilo na 1 minut, što je kod prethodne Java aplikacije trajalo 40 min [9].

3.4 Uber



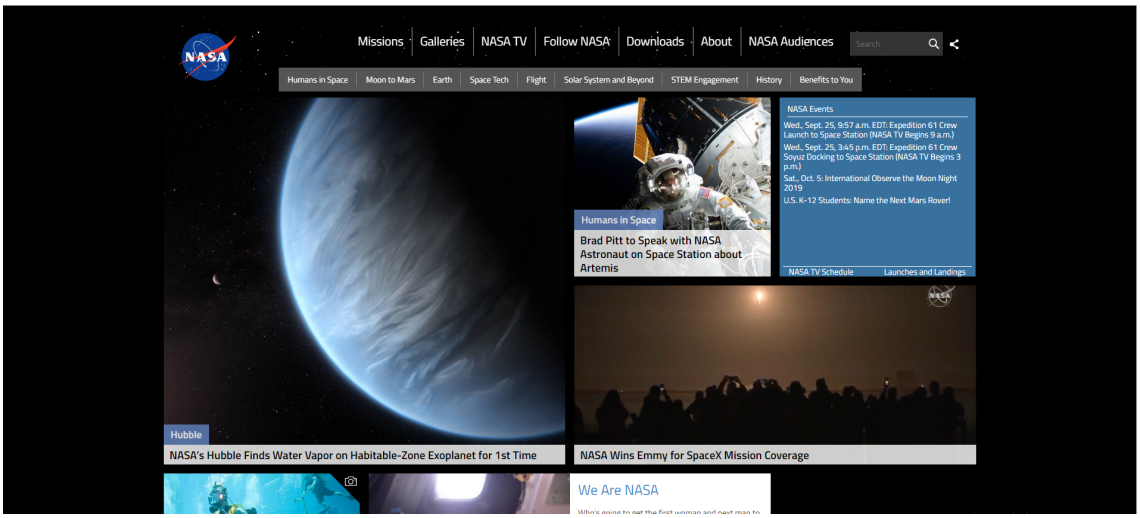
Slika 7. Uber [13]

Uber je američka kompanija koja se bavi izgradnjom transportne mreže. Operiše u preko 40 zemalja i 404 grada, gde spaja putnike sa vozačima koji koriste sopstvena vozila.

Na kraju svake vožnje, iznos se automatski naplaćuje putnikovoj kartici. Uber je među prvim kompanijama koje su iskoristile NodeJS za masivnu upotrebu, na kome je izgradila sistem spaja putnika i vozača.

Prema kompaniji, prednosti NodeJS-a su: brzo procesiranje velike količine podataka, brzo ispravljanje grešaka, mogućnost čestog proširivanja i dodavanja koda, brz razvoj i čest napredak [11].

3.5 NASA



Slika 8. NASA - National Aeronautics and Space Administration [14]

NASA je nezavisna agencija federalne vlade Sjedinjenih Američkih Država, odgovorna za civilne svemirske programe, kao i za aeronautička i vazduhoplovna istraživanja.

Tokom šetnje u otvorenom svemiru 2013. godine, jednom od astronauta je procurela voda u kacigu. Zbog gravitacionih uslova, voda je brzo došla do očiju i ušiju, i on nije mogao da vidi i čuje, a disao je otežano. Astronaut koji se nalazio u blizini je uspeo da ga odvede na sigurno.

Da bi proučili uzroke ovog incidenta, NASA je morala da prikupi i obradi ogromnu količinu podataka. Bilo je potrebno stvoriti novi sistem koji će brzlo i efikasno obraditi te podatke. Inženjeri su odlučili da taj sistem baziraju na NodeJS-u, a 3 stare baze podataka su premeštene u klaud. Taj sistem je omogućio korisnicima da na jednom mestu zahtevaju sve podatke, a vreme za dobijanje je smanjeno za 300% [15].

4. OPIS PRIMENJENIH TEHNOLOGIJA

4.1 Angular

Angular je Javaskript radni okvir za kreiranje velikih i kompleksnih jednostranih aplikacija (single page applications). Ovaj tip aplikacija karakteriše jedan HTML (Hyper Text Markup Language) fajl koji se dinamički menja pomoću Javaskripta. Sve promene se dešavaju u samoj aplikaciji bez potrebe da se aplikacija osvežava, poput menjanja stranica, validacija forme, animacije, slanja podataka serveru i prikaz novih pdataka po prijemu. Sve potrebne podatke za rad i prikaz na korisnički interfejs Angular dobija sa eksternih veb servisa, za šta u ogromnoj meri koristi AJAX tehniku.

Angular aplikacije su izutetno brze, skalabilne i dinamične. Omogućava kreiranje veoma velikih i kompleksnih korisničkih interfejsa, na kojima sa lakoćom može raditi više ljudi i timova istovremeno. Testabilnost je ugrađena u radni okvir od samoga početka, pa je veoma lako pisati automatske testove. Poseduje sve alate i pomagala za razvoj najkompleksnijih aplikacija za najzahtevnije klijente.

Angular je besplatan i otvorenog koda. Razvija ga i podržava kompanija Google, kao i veliki broj dobrovoljaca. Poseduje odličnu dokumentaciju i veliku broj entuzijasta.

Angular je nova, unapređena verzija starog radnog okvira istog imena. Zbog velikih nedostataka i zastarelih praksi u starom radnom okviru, bilo je potrebno osvežiti ga i osavremeniti. Promene su rezultirale potpuno novim, neprepoznatljivim radnim okvirom. Stoga, da ne bi dolazilo do zabune, kompanija Google je odlučila da se naziv “Angular” koristi za novi radni okvir, a za stari “Angular.js”. U ovom radu koristi se isključivo nova verzija ovog radnog okvira. Pod novim se podrazumevaju verzije 2 i više.

Sam radni okvir pisan je u jeziku Tajpskript (Typescript). Tajpskript je superset jezika Javascript, odnosno proširenje koje mu daje nove mogućnosti koje ne postoje u standardnom jeziku. Nove mogućnosti se prevashodno tiču tipova podataka, odnosno daje Javaskriptu mogućnost jakog tipiziranja (otud ime Typescript). Ostale mogućnosti tiču se objektno orijentisanog, odnosno principa i pojmova koji se nalaze u klasičnim OOP (Object Oriented Programming) jezicima poput Jave i C#, a to su klase, objekti, interfejsi, nasleđivanje, apstrakcija i sl. Programeri sa iskustvom u Javi ili C# mogu vrlo brzo preći na Tajpskript, što nije slučaj sa klasičnim Javaskriptom.

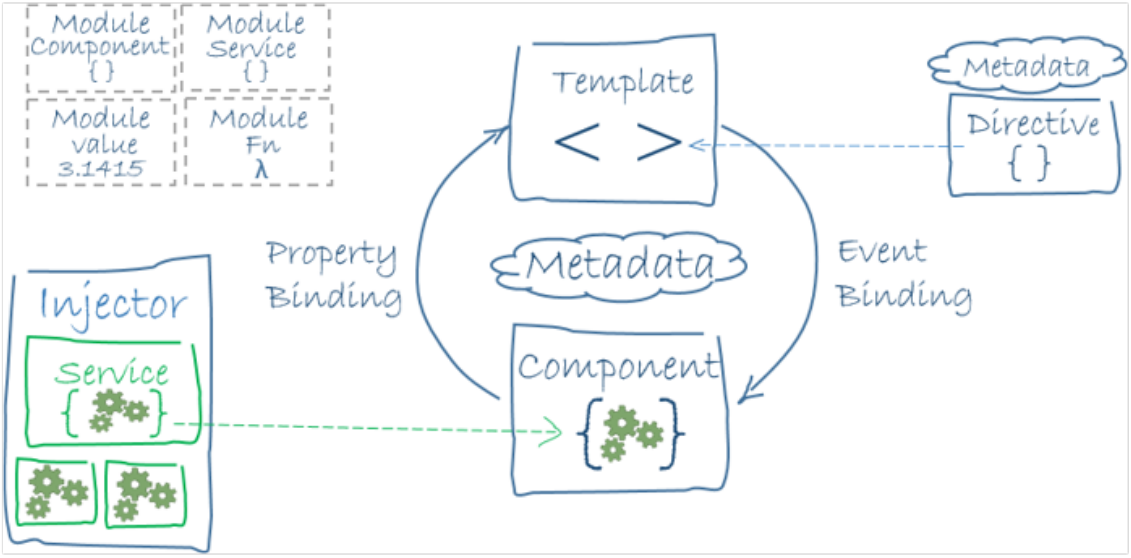
Tajpskript je takođe jezik otvorenog koda, a stvorila ga je i razvija kompanija Majkrosoft.

I sama Angular aplikacija se piše u jeziku Tajpskript. Svaki program pisan u Tajpskriptu se ne kompajlira u izvršni binarni kod, nego u neku od verzija Javaskripta, pošto pretraživači mogu izvršavati samo Javaskript.

Glavni gradivni blokovi svake Angular aplikacije čine:

- Moduli
- Komponente

– Servisi
– Direktive
– Cevi (Pipes)



Slika 9. Arhitektura Angular aplikacije [16]

4.1.1 Moduli

Angular aplikacije su modularne i Angular ima svoj sistem modula. Moduli u angular predstavljaju kontejnere za sav srodan kod koji se tiče jedne oblasti ili dela aplikacije. Oni mogu sadržati komponente, servise, direktive i sve druge fajlove koji zajedno čine jednu logičku celinu.[17] Npr. jedan modul može grupisati kod koji čini “landing stranicu”, dok bi drugi mogao biti “dashboard”. Moduli takođe mogu importovati i druge module, kao i eksportovati neke od svojih delova kako bi drugi moduli mogli da ih koriste. Ovo jse najčešće koristi kod biblioteka koda.

```
import { NgModule }      from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
@NgModule({
  imports:      [ BrowserModule ],
  providers:    [ Logger ],
  declarations: [ AppComponent ],
  exports:      [ AppComponent ],
  bootstrap:    [ AppComponent ]
})
export class AppModule { }
```

Slika 10. Jednostavni Angular modul[17]

Modul je zapravo klasa, koja je opisana Angularovim dekoraterom `@NgModule()`. Ovaj dekorater daje Angularu informacije o modulu kako bi mogao kasnije da izvrši kompilaciju.

Svaka Angular aplikacija ima makar jedan modul, koji se po konvenciji zove "AppModule". Može imati i neograničen broj pomoćnih modula. Svi ovi pomoćni moduli su u stvari moduli koji se importuju u glavni "AppModule". Pri kompilaciji i pokretanju aplikacije, Angular uvek prvo kreće od "AppModule"-a.

AppModule mora imati i makar jednu komponentu, koja se po konvenciji zove AppComponent. Ovo su jedina dva uslova potrebna za kreiranje najprostije Angular aplikacije.

4.1.2 Komponente

Komponente su gradivni elementi korisničkog interfejsa u Angular aplikaciji. Komponenta predstavlja deo ekrana koji korisnik vidi i ima kontakt sa njim. Npr. komponente mogu biti navigacioni meni, lista korisnika, kontakt forma, footer i sl. Ovi delići ekrana, sa svojim HTML-om i CSS-om (Cascading Style Sheets) kodom i komponentom zajedno čine pogled (view).

Svaka komponenta je zatvoren sistem. Sav HTML, CSS i Tajpskript kod jedne komponente se ne mogu mešati sa kodom druge komponente. Stilovi pisani u jednoj komponenti se primenjuju samo na nju i ne može se preliti na druge. Komponente je moguće ugnežđavati, a broj komponenti u aplikaciji nije ograničen. Teži se što manjim i jednostavnijim komponentama, kako bi bile lakše za rad i održavanje.

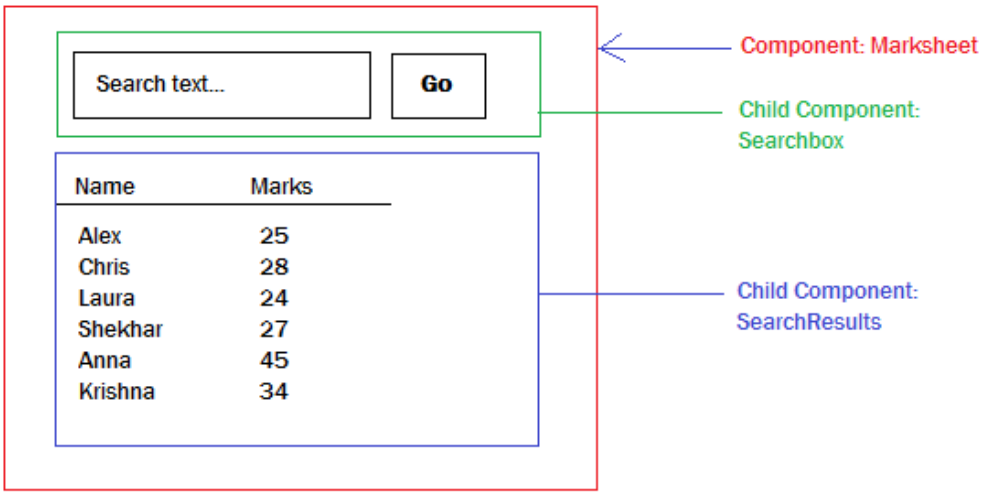
Svaka Angular aplikacija mora imati najmanje jednu komponentu, i ona se po konvenciji naziva "AppComponent". Moguće je da čitava aplikacija bude jedna velika komponenta, međutim, kako ona raste postaje sve teže za rad i održavanja. Sve ostale komponente se ugnežđavaju u ovu glavnu.

Slično kao i kod modula, komponenta je takođe klasa, koja je opisana Angularovim

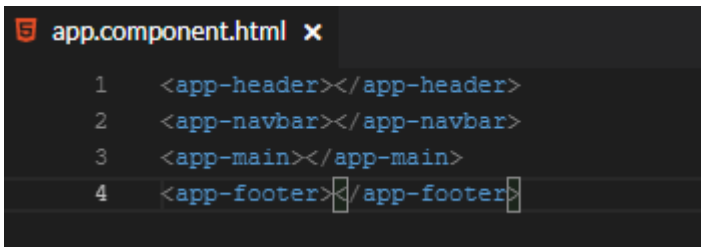
dekoraterom `@Component()`. Ovaj dekorater pruža informacije o komponenti, a najbitnije su putanje do HTML i CSS fajlova i selektor. Na ovaj način Angular zna koji HTML i CSS kod pripada određenoj komponenti.

Selektor komponente je njena oznaka koja se koristi pri pozivanju komponente u korisničkom interfejsu. Kada se naziv selektora u postavi na korisnički interfejs u obliku HTML elemnta, Angular će na tom mestu postaviti sav HTML i prateći CSS i Tajpskript kod na to mesto.

Npr. komponenta se zove “NavigationComponent”. Njen selektor bi bio “app-navigation”, a HTML element bi bio `<app-navigation></app-navigation>`. Ovako napravljena komponenta se ponaša isto kao i standardni HTML elementi. Prefiks “app” se dodaje da ne bi došlo do kolizije u slučaju da dve različite komonente imaju isto ime. Takođe, ovaj prefiks se može proizvoljno menjati.



Slika 11. Primer ugnežđavanja komponenti [18]



Slika 12. Primer podele korisničkog interfejsa na komponente

4.1.3 Direktive

Angular aplikacije su izuzetno dinamične. Elementi korisničkog interfejsa se menjaju na osnovu instrukcija koje dobijaju od direktiva. Direktive su klase se `@Directive()`

dekoraterom.[19]

Direktiva je u suštini isto što i komponenta, sa jednom glavnom razlikom: nema svoj HTML i CSS. Obzirom da je komponenta ključni pojam u radnom okviru, dobila je svoj dekorater.[19]

Postoje dve vrste direktiva: strukturne i atributske.[19] Radni okvir poseduje ugrađene direktive, ali je moguće napraviti i sopstvene po potrebi.

Strukturne direktive menjaju strukturu korisničkog interfejsa. One mogu ponavljati određeni element željen broj puta, ili da na osnovu nekog uslova brišu ili ubacuju element na korisnički interfejs. Najčešće korištene strukturne direktive su "*ngFor" i "*ngIf". Ove direktive su analogne petljama i selekcijama koje postoje u svim programskim jezicima.

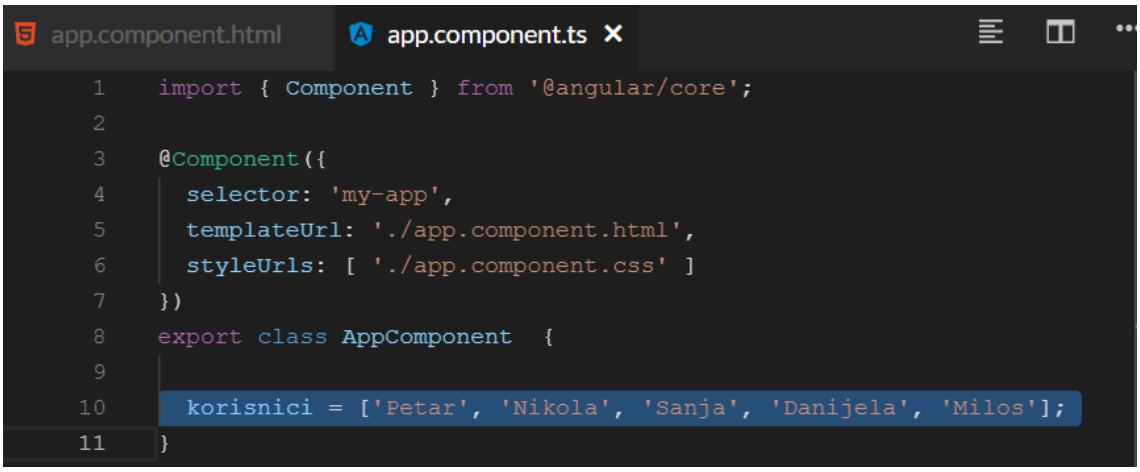
Kod direktive "*ngIf", ukoliko je postavljen uslov tačan, element se izbacuje iz DOM-a (Document Object Model). Ukoliko nije, on se briše.

app.component.html

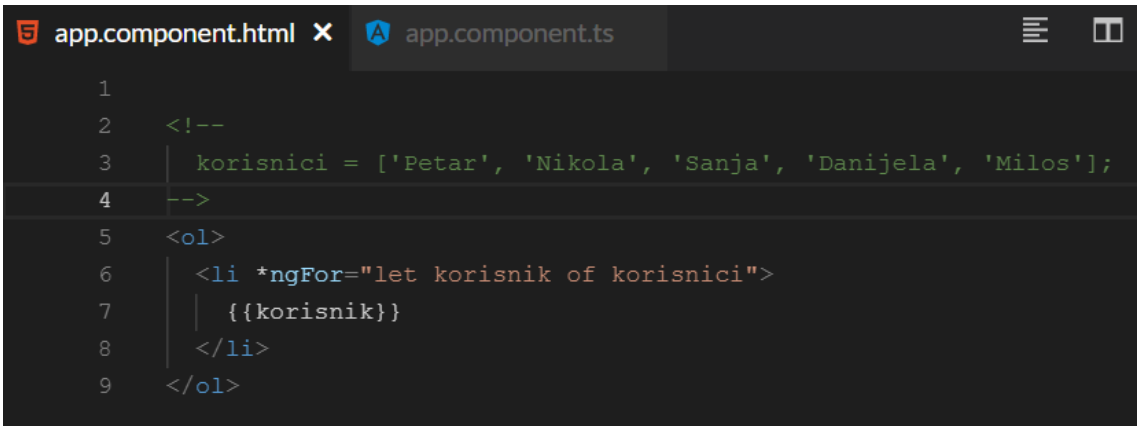
```
1  <section *ngIf="korisnkJePrijavljen === true">
2    <h1>Welcome</h1>
3  </section>
4
5  <section *ngIf="korisnikJePrijavljen === false">
6    <h1>Please log in.</h1>
7  </section>
```

*Slika 13. Primer "*ngIf" direktive*

Direktiva "*ngFor" ponavlja elemente onoliko puta koliko ima članova u datoj kolekciji. Kolekciju predstavlja niz nekih podataka koje treba prikazati na korisničkom interfejsu. Npr. treba napraviti listu korisnika i ispisati njihova imena, a ima ih pet. Direktiva će u tom slučaju napraviti pet elemenata liste i daje nam mogućnost da podatke o korisnicima ubacimo u DOM.



Slika 14. Definicija niza korisnika u komponenti



*Slika 15. Upotreba "*ngFor" direktive nad nizom korisnika*

*Slika 16. Rezltat "*ngFor" direktive je lista korisnika*

Znak "*" je deo sintakse vezane za strukturne direktive. On govori da će ova direktiva imati za rezultat brisanje ili kreiranje elementa na kome se nalazi. Bez ovog znaka, strukturna direktiva ne bi radila. Ova sintaksa ne postoji kod atributskih direktiva.

Atributske direktive dodaju ili menjaju neka od svojstava elementa na kome se nalaze, najčešće onih što se tiču izgleda. Angular pruža nekoliko atributskih direktiva, a najčešće se koriste “ngStyle” i “ngClass”.

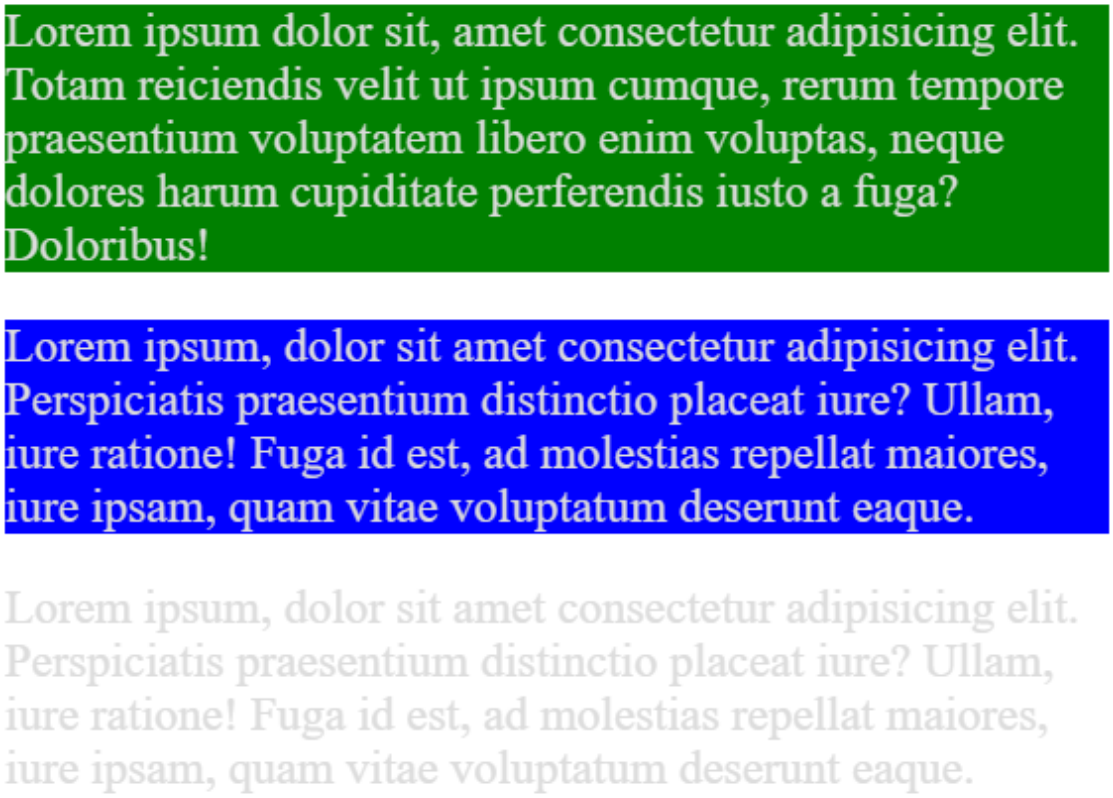
“ngStyle” i “ngClass” dodaju ili sklanjaju CSS pravilo i CSS klasu sa elementa na kom se nalaze, respektivno. Ukoliko je ispunjen logički uslov, pravilo ili klasa se dodaju elementu, u suprotnom se ne dodaju.

app.component.html

```
1  <p [ngClass]="{'green-bg': true}">Lorem ipsum dolor sit, amet
   consectetur adipisicing elit. Totam reiciendis velit ut ipsum cumque,
    rerum tempore praesentium voluptatem libero enim voluptas, neque
   dolores harum cupiditate perferendis iusto a fuga? Doloribus!</p>
2
3  <p [ngClass]="{'blue-bg': true}">
4    Lorem ipsum, dolor sit amet consectetur adipisicing elit.
     Perspiciatis praesentium distinctio placeat iure? Ullam, iure
     ratione! Fuga id est, ad molestias repellat maiores, iure ipsam,
     quam vitae voluptatum deserunt eaque.
5  </p>
6
7  <p [ngClass]="{'blue-bg': false}">
8    Lorem ipsum, dolor sit amet consectetur adipisicing elit.
     Perspiciatis praesentium distinctio placeat iure? Ullam, iure
     ratione! Fuga id est, ad molestias repellat maiores, iure ipsam,
     quam vitae voluptatum deserunt eaque.
9  </p>
```

Slika 17. Primer “ngClass” direktive na paragrafima

Prva dva paragrafa ispunjavaju uslov, stoga će prvi dobiti zelenu pozadinu, a drugi plavu. Treći paragraf ne ispunjava uslov i neće dobiti plavu pozadinu, već će boja njegove pozadine ostati nepromenjena.



Slika 18. Dinamičko dodeljivanje CSS klase pomoću "ngStyle"

4.1.4 Cevi (Pipes)

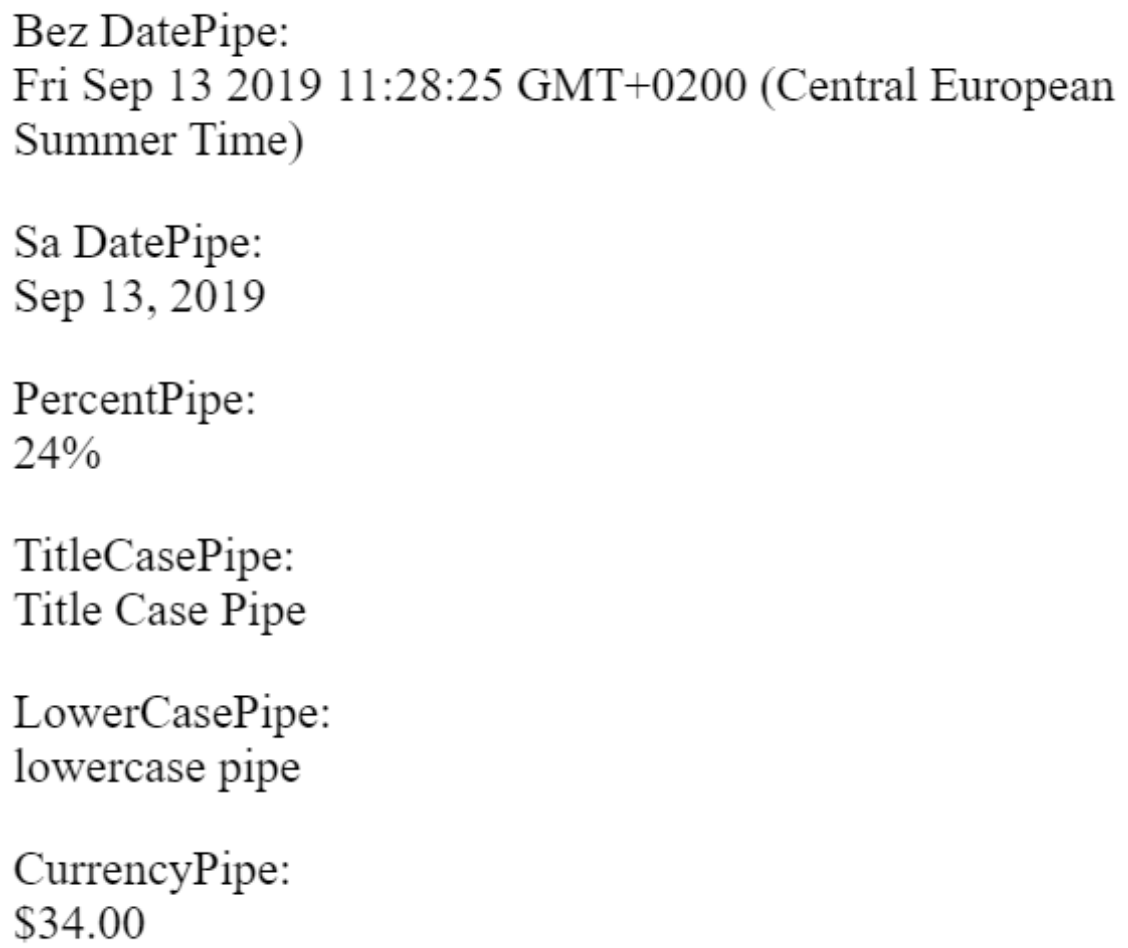
Cevi su proste klase dekorisane `@Pipe()` dekoraterom. One imaju samo jedan metod, i cilj im je transformacija vrednosti iz korisničkog interfejsa, koje ovaj metod vraća interfejsu neposredno pred prikazivanjem korisniku.

Često korištene cevi koje Angular pruža su "PercentPipe", "DatePipe", "TitleCasePipe" i "LowerCasePipe" i "CurrencyPipe". [20]

app.component.html

```
1
2   <p>Bez DatePipe: <br> {{datum}}</p>
3
4   <p>Sa DatePipe: <br> {{datum | date}}</p>
5   <p>PercentPipe: <br> {{0.24 | percent}}</p>
6   <p>TitleCasePipe: <br> {{'tiTlE caSE PiPe' | titlecase}}</p>
7   <p>LowerCasePipe: <br> {{ 'LOWERCASE PIPE' | lowercase}}</p>
8   <p>CurrencyPipe: <br> {{ 34 | currency}}</p>
```

Slika 19a. upotreba cevi u Angularu



Slika 19b. Rezultat rada cevi

Cevi predstavljaju most između formata podataka koji se skladište u računaru, i formata koji je razumljiv čoveku. Datum u formatu dan-nedelja-godina je čoveku mnogo više razumljiviji, ali se taj datum mnogo lakše skladišti u nekom drugom formatu. Ovim se korisnicima pruža mnogo bolje korisničko iskustvo.

Kao i ostali gradivni elementi Angular radnog okvira, i cevi se mogu graditi ispočetka.

4.1.5 Servisi

Servisi si su najprostiji element Angular arhitekturi, ali i najvažniji.

Servis je obična klasa, kao što su klase u Javi ili C#. Može se nasleđivati, implementirati interfejse, deklarisati javne, privatne i zaštićene metode itd. On ima specifičnu namenu i jasno definisanu svrhu, i manji broj odgovornosti koje mora obavljati na najbolji način. Na ovaj način postiže se bolja modularnost i ponovna iskoristivost koda. [22]

Cilj servisa je definisanje biznis logike za određeni domen aplikacije. Omogućava da kod komponente bude jednostavniji i fokusiran na prezentacionu logiku. Sve kompleksne operacije, računanja i logike se izvršavaju u servisu. U servisu se najčešće pozivaju

eksterni servisi i dobijaju podaci, koji se prosleđuju komponentama, a komponente ih koriste da stvore korisnički interfejs.

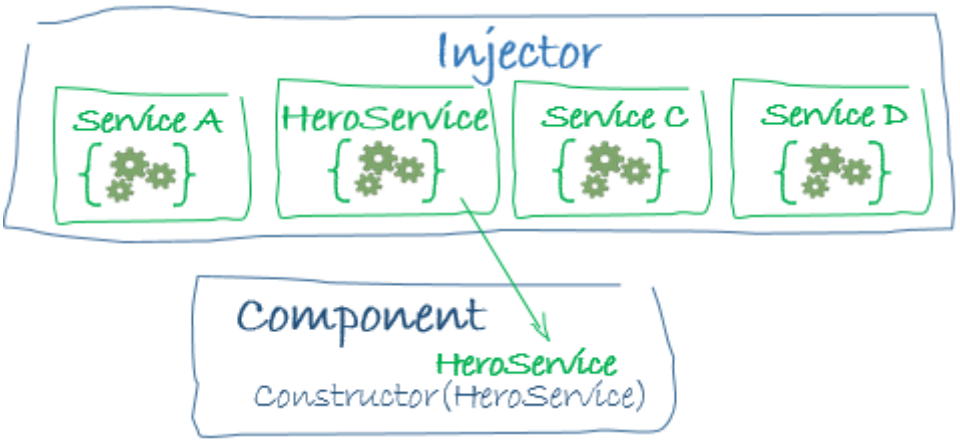
Iako je moguće svu biznis logiku pisati direktno u komponenti, ovakva praksa se smatra anti-paternom, stoga se izbegava. Komponenti pripada jednostavna prezentaciona logika, a u servisu sva druga.

Angular poseduje mehanizam koji omogućava jednostavno ubacivanje servisa u komponente. Ovaj mehanizam naziva se "Dependency Injection". Zbog ovog mehanizma, servisi imaju decorater `@Injectable()`, što označava da se ovaj servis može dodeljivati komponentama, ali i drugim servisima.

Slika 20a. Dependency Injection[21]

"Dependency Injection" mehanizma stvara kontejner u kome čuva sve instance servisa. Ove instance su najčešće Singleton, tj. postoji samo jedna instanca svakog servisa. Potom svakoj komponenti dodeljuje pristup servisu koji komponenta zahteva. Komponenta željeni servis definiše u konstruktoru kao privatni argument, nakon čega radni okvir obavlja sve potrebne korake u pozadini koji su potrebni da komponenta dobije pristup servisu.

Kada komponenta dobije pristup servisu, ona može slobodno koristiti sve njene metode i atribute.



Slika 20b. Mehanizam ubacivanja servisa iz kontejnera u komponentu[21]

Za razliku od ostalih elemenata, servis ne mora biti importovan u modul. U novijim verzijama Angular radnog okvira, moguće je konfigurisati ga tako da je dostupan na globalnom nivou cele aplikacije. Ovo je ekvivalentno importovanju servisa u "AppModule".

```
import { Injectable } from '@angular/core';

@Injectable({
  providedIn: 'root',
})
export class UserService {
}
```

Slika 20.c Primer globalnog servisa[22]

app.component.ts

```
1   import { Component } from '@angular/core';
2   import { UsersService } from './users.service';
3
4   @Component({
5     selector: 'my-app',
6     templateUrl: './app.component.html',
7     styleUrls: [ './app.component.css' ]
8   })
9   export class AppComponent  {
10
11    constructor(private usersService: UsersService) {
12      this.usersService.getUsers();
13    }
14  }
```

Slika 20d. primer komponente sa ubačenim servisom

4.1.6 Angular CLI

Angular CLI (Command Line Interface) je sredstvo pomoću kog se Angular aplikacije brzo kreiraju, razvijaju, generišu fajlovi i održavaju pomoću komandi iz komandne linije[23]. Pri inicijalizaciji Angular projekta, CLI takođe vrši svu konfiguraciju kako bi razvoj aplikacije tekao brzo, efikasno i bez puno smetnji.

Neke od poslova koje obavlja CLI je:

- Inicijalizacija projekta
- Inicijalizacija git repozitorijuma
- Konfiguriše razvojni http server sa automatskim osvežavanjem
- Kompajlira SASS / SCSS fajlove

– Kompajlira Tajpskript fajlove
– Generiše komponente, servise, module itd.

Na kraju razvoja aplikacije, CLI priprema aplikaciju za produkciono okruženje. Tokom ove pripreme, vrši veliki broj optimizacija i kompresovanje koda. Na kraju se dobija jedan HTML fajl, nekoliko JS fajlova i CSS fajlova, sa propratnim statičkim fajlovima (slike, fontovi itd.). Zatim se oni mogu korisnicima servirati preko HTTP servera, i oni će moći tada da koriste Angular aplikaciju.

```
ng build my-app -c production
```

Slika 21. komanda za kreiranje produkcione verzije aplikacije [23]

4.1.7 Typescript

Typescript (Tajpskript) je programski jezik nastao kao potreba da se isprave nedostaci koji se nalaze u Javaskript jeziku. Pojavljivala se sve veća potreba za velikim i kompleksnim Javaskript aplikacijama. Međutim, sa rastom aplikacije, postajalo je sve teže proširiti je, bagovi su se javljali sve češće i postajali se teži za otkrivanje i popravljanje.

Tajpskript je superset Javaskripta. To znači da je svaki validan Javaskript kod, ujedno i validan Tajpskirpt kod. Javaskript kodu se mogu dodati mnoge funkcionalnosti, koje ne postoje u originalnom jeziku, a Tajpskript kompajler (tačnije, transpajler) će generisati implementaciju tih funkcionalnosti. Na ovaj način pišemo daleko manje i jednostavnijeg koda.

Typeskript je jako tipiziran objektno orijentisan jezik, što Javaskript nije, pravljen po uzoru na Javu i C#. Omogućava pisanje Javaskripta objektno orijentisanim principa, što je moguće u klasičnom Javaskriptu, ali teško.

Glavna prednost Tajpskripta je tipiziranje podataka i varijabli. Eksplicitnim navođenjem tipova sprečavaju se greške izazvane pogrešnim navođenjem tipova podataka da dođu do krajnjih korisnika. Ovakve greške uočava kompajler istog momenta kada se pojave.

Sve odlike objektno orijentisanih jezika, nalaze se takođe u Tajpskript, među kojima su: apstrakcija, enkapsulacija, nasleđivanje, polimorfizam, obeleživači dostupnosti (public, protected, private) itd.

Tajpskript fajlovi imaj u ekstenziju .ts, a rezultat kompajliranja je fajl ekstenzije .js (običan Javaskript fajl).

```
1   class Animal {
2     constructor(public name: string) {}
3     move(distanceInMeters: number = 0) {
4       console.log(`${this.name} moved ${distanceInMeters}m.`);
5     }
6   }
7
8   class Dog extends Animal {
9     constructor(name: string) {
10      super(name);
11    }
12    move(distanceInMeters = 3) {
13      super.move(distanceInMeters);
14    }
15  }
16
17  let zuca = new Dog("Zuca");
18
19  zuca.move();
20
```

Slika 22a. primer nasleđivanja i instanciranja u Tajpskriptu

```
"use strict";
var __extends = (this && this.__extends) || (function () {
    var extendStatics = function (d, b) {
        extendStatics = Object.setPrototypeOf ||
            ({ __proto__: [] } instanceof Array && function (d, b) { d.__proto__ = b; }) ||
            function (d, b) { for (var p in b) if (b.hasOwnProperty(p)) d[p] = b[p]; };
        return extendStatics(d, b);
    };
    return function (d, b) {
        extendStatics(d, b);                (parameter) d: any
        function __() { this.constructor = d; }
        d.prototype = b === null ? Object.create(b) : (__.prototype = b.prototype, new __());
    };
})();
var Animal = /** @class */ (function () {
    function Animal(name) {
        this.name = name;
    }
    Animal.prototype.move = function (distanceInMeters) {
        if (distanceInMeters === void 0) { distanceInMeters = 0; }
        console.log(this.name + " moved " + distanceInMeters + "m.");
    };
    return Animal;
}());
var Dog = /** @class */ (function (_super) {
    __extends(Dog, _super);
    function Dog(name) {
        return _super.call(this, name) || this;
    }
    Dog.prototype.move = function (distanceInMeters) {
        if (distanceInMeters === void 0) { distanceInMeters = 3; }
        _super.prototype.move.call(this, distanceInMeters);
    };
    return Dog;
}(Animal));
var zuca = new Dog("Zuca");
zuca.move();
```

Slika 22b. Javaskript kod dobijen kompilacijom koda sa slike 18. a

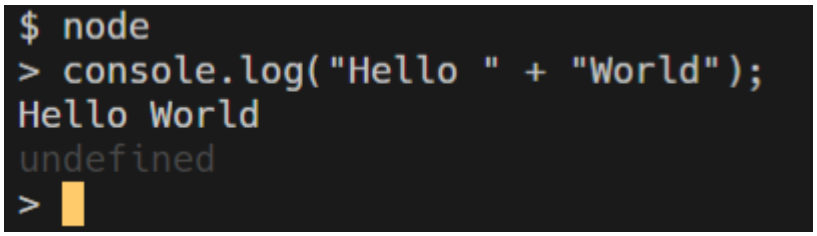
Zahvaljujući Tajpskriptu, u mogućnosti smo da pišemo kod sa slike 18 a. U suprotnom, bili bismo primorani da pišemo kod sa slike 18 b. koji je ekvivalentan.

4.2 NodeJS i ExpressJS

NodeJS je platforma za izršavanje Javaskripta na serveru, nastao 2009. godine. Zahvaljujući njemu, Javaskript je postao moćno oruđe za pisanje izuzetno brzih i efikasnih aplikacija.

NodeJS predstavlja sloj iznad Javaskript interpretera “V8”, koji se nazali u “Google Chrome” pretraživaču i održava ga kompanija Google. [24] Poseduje bogatu standardnu biblioteku koja omogućava jednostavan rad fajl sistemom, operativnim sistemom, kriptografijom, a akcenat se stavlja na biblioteke za mrežnu komunikaciju, sa čime jednostavno kreirati veb server i slične aplikacije.

Besplatan je i otvorenog koda. Potrebno ga je preuzeti sa sajta https://nodejs.org/en/, instalirati, i odmah postaje dostupan kroz komandnu liniju. U komandnoj liniji se poziva rečju “node”, i otvara se interfejs za pisanje Javaskripta.



Slika 23. izvršavanje Javaskripta na serveru

Zajedno sa NodeJS-om, prilikom instalacije dolazi i NPM (Node Package Manager). Pored bogate standardne biblioteke, postoji i veliki broj biblioteka i radnih okvira pisanih od strane trećih lica. Ove biblioteke je vrlo lako uključiti u projekat pomoću NPM. Najpopularnija od ovih biblioteka je Express.JS.

Experss.JS je mala, brza i minimalistička Javaskript biblioteka za pisanje efikasnih i fleksibilnih veb servera u Javaskriptu. Poseduje mnoštvo metoda vezanih za HTTP protokol, što ga čini idealnim za pisanje API-a brzo i lako, koje su samo tanak sloj iznad standardnih NodeJS biblioteka za rad sa HTTP protokolom [25]

Za razliku od Angulara, Express nema nikakvu konvenciju kako pisati aplikacije pomoću njega. Programer ima slobodu da struktuira aplikaciju na način kakav njemu odgovara, ili kako zahteva projekat. Takođe, on je zamišljen da radi jednu stvar, i da je radi dobro, stoga drugi aspekti veb aplikacije moraju biti posebno implementirani, ili iskoristi druge biblioteke koje se bave time. Express nema podršku za oblasti poput autorizacije ili baze podatake. One se moraju posebno implementirati. Srećom, lako je integrisati druge biblioteke sa Express-om.

Aplikacije pisane pomoću Express biblioteke su najčešće API, koji slušaju na određenom portu. Ukoliko detektuje HTTP poziv na nekoj od ruta, on će na njih odgovoriti, a odgovor poslati u JSON formatu.

```
1   const express = require('express')
2   const app = express()
3
4   app.get('/', (req, res) => res.send('Hello World!'))
5
6   app.listen(3000)
```

Slika 24. najprostiji Express API

Za kreiranje prostog API servera, potrebno je prvo uključiti biblioteku u fajl, potom ga instancirati. Metoda “get” očekuje HTTP pozivom sa GET metodom, na samom korenu domena. Kada se to desi, izvršava se funkcija sa argumentima “req” i “res”. Ovi argumenti predstavljaju objekte koji sadrže informacije o zahtevu koji je stigao na server, i odgovoru, koji će biti poslat nazad. Metoda “send” na “res” objektu šalje odgovor na zahtev sa prostim “Hello World” stringom. Metoda “listen” pokreće server koij sluša zahteve na portu 3000.

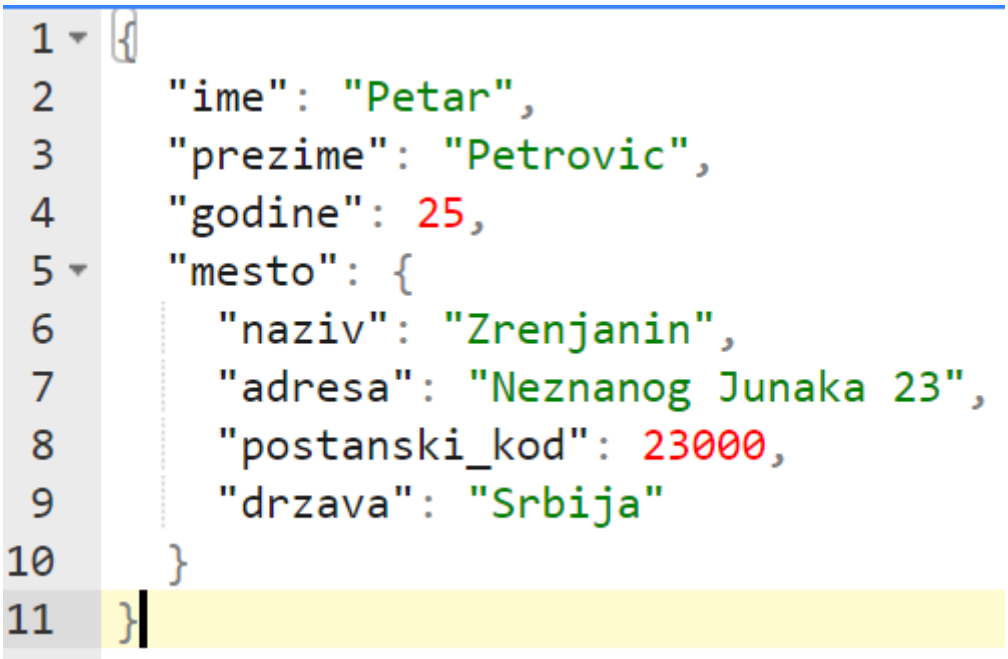
4.3 MongoDB

4.3.1 Osnovni pojmovi

MongoDB (ime izvedeno od reči “humongous” - ogromno), od istoimene kompanije, je relativno nova vrsta baza podataka u kojoj ne postoje pojmovi poput tabela, šema, SQL--a, redova, spajanja (JOIN), stranih ključeva i sl [29]. On je NOSQL, nerelaciona baza podataka u kojoj su tabele zamenjene dokumentima.[1]

Cilje ove baze podataka je da radi sa dokumentima, umesto tabelama, da bude brza, skalabilna i laka za korišćenje. Sve podatke skladišti u jednom dokumentu u JSON formatu. JSON svojom strukturom jasno opisuje podatke koje sadrži, tako da nema potrebe da se unapred definišu podaci koji će se skladištiti. JSON nema striktno definisanu šemu podataka, jer se dokumenti mogu jednostavno menjati bez uticaja na druge dokumente. Šta više, JSON poboljšava performanse jer sve podatke čuva u jednom dokumentu, pa nema potrebe za komplikovanim i sporim upitima za spajanje (JOIN).

Tačnije, format skladištenja podataka je BSON (Binary JSON). BSON, kao binarna verzija JSON-a, je format razvijen za potrebe Mongo baze podataka. Binarni oblik je pogodniji računarima, što dodadne daje na bazi podataka na brzini. BSON ima svoje dodatne prednosti, među kojima su tipovi za rad sa binarnim podacima[29].



Slika 25. Mongo dokument

Kada se unutar jednog dokumenta nalaze kompleksne strukture poput objekta i niza, takav dokument se naziva “embedded document” (ugrađen dokument). Dokument “Osoba” ima ime i prezime, ali i mesto, koje ima naziv, adresu itd. što njega samim čini dokumentom.

[1] U novijim verzijama se pojavljuje podrška i za transakcije i ACID upite.

Dokumenti se mogu neograničeno ugrađivati jedni u druge, bez ograničenja broja dokumenata i dubine.

Dokument nema unapred određenu i fiksnu strukturu. Svaki dokument može imati bilo koju kombinaciju ključa i podataka. Ključ deluje kao oznaka podatka, a vrednost sadržaj tog podatka npr. “mesto”: “Zrenjanin”. Oni se u svakom trenutku mogu menjati. Neka polja se mogu dodati, a neka obrisati u bilo kom delu dokumenta, integritet baze podataka neće bit narušen. Ukoliko neko polje ostane nepopunjeno, ono jednostavno ostaje prazno.

Kao i kod relacionih baza podataka, i nerelacione moraju jedinstveno identifikovani svaki podatak. Tabela u relacionoj bazi mora unapred imati definisano polje, koje se uzima za jedinstveni identifikator. Kod Mongo baze podataka ovaj proces je automatizovan. Svakom dokumentu Mongo dodeljuje polje “_id”, ukoliko prethodno nije eksplicitno definisan. Ova fleksibilnost omogućuje pojedincu da odluči da li će sam definisati primarne ključeve za dokumente, ili će dozvoliti bazi da to čini umesto njega.

Kolekcije su analogne tabelama u RDMS, dok su dokumenti analogni bazi podataka. Svaki dokument se sastoji od jedne ili više kolekcije, koje su mnogo fleksibilnije od tabela. U kolekciju je dozvoljeno ubaciti bilo koje podatke. Npr. u kolekciju filmova sačinjavaju naziv filma, reditelj i godina nastanka. I ako nas ništa ne sprečava da u ovu kolekciju stavimo ime kućnog ljubimca, zbog indeksiranja i upita je bolje držati srodne podatke u istoj kolekciju.

Kolekcije se kreiraju po potrebi. Nije potrebno deklarisati ni nazive. Umesto toga, Mongo će ih kreirati onda kada prvi put bude upisivao nov dokument ili kolekciju. Npr. kolekcija filmova biće kreirana tek kada se bude uneo prvi film. U suštini, baza podataka je kolekcija kolekcija.

4.3.2 Manipulacija podacima

Slično kao NodeJS, i MongoDB-u se pristupa preko komandne linije. Mongo-v komandni interfejs dolazi sa velikim brojem metoda i mehanizama za rad sa podacima. Mnoge uobičajene operacije su predefinisane, kao što su čitanje, pisanje, menjanje i brisanje podataka i njih nije potrebno posebno pisati.

```
> use people
switched to db people
> db.people.insert({name: "Petar Petrovic", age: 25})
WriteResult({ "nInserted" : 1 })
> db.people.insert({name: "Nikola Nikolic", age: 34})
WriteResult({ "nInserted" : 1 })
> db.people.insert({name: "Marko Markovic", age: 26})
WriteResult({ "nInserted" : 1 })
>
```

Slika 26. upisivanje podataka u prethodno odabranu bazu

Ključna reč “use” se koristi da odaberemo bazu podataka na kojoj želimo da radimo. Ta baza se smešta u “db” objekat, u kome se nalaze kolekcije. Sintaksa i način pisanja upita je identičan onom u jeziku Javaskript. Pošto nije neophodno eksplicitno kreirati kolekciju, u mogućnosti smo da odmah ubacimo dokumente u kolekcije pomoću ugrađene metode “insert”. Ova metoda prihvata JSON objekat, i smešta je u bazu podataka onako kako joj je i prosleđeno.

```
> db.people.find()
{
        "_id" : ObjectId("5d7d3d83f467b5b2089bf7f0"),
        "name" : "Petar Petrovic",
        "age" : 25
}
{
        "_id" : ObjectId("5d7d3d94f467b5b2089bf7f1"),
        "name" : "Nikola Nikolic",
        "age" : 34
}
{
        "_id" : ObjectId("5d7d3da5f467b5b2089bf7f2"),
        "name" : "Marko Markovic",
        "age" : 26
}
```

Slika 27. selektovanje cele kolekcije

Metoda “find” pronalazi sve dokumente iz željene kolekcije. Bez argumenata, analogna je upitu “SELECT * FROM” u relacionim bazama. Ovoj metodi se može proslediti bilo koji uslov, i ona će vratiti sve rezultate koji ispunjavaju taj uslov. Npr. `db.people.find({age: 25})` bi vratila samo osobe koje imaju tačno 25. godina.

```
> db.people.find({age: {$lt: 30}})
{
        "_id" : ObjectId("5d7d3d83f467b5b2089bf7f0"),
        "name" : "Petar Petrovic",
        "age" : 25
}
{
        "_id" : ObjectId("5d7d3da5f467b5b2089bf7f2"),
        "name" : "Marko Markovic",
        "age" : 26
}
>
```

Slika 28. selektovanje osoba sa manje od 30 godina.

Mongo poseduje veliki broj operatora. Jedan od njih je `$lt` (less than), koji selektuje količinu manju od navedene.

4.3.3 Skaliranje baze podataka

Strategija za skaliranje relacionih baza podatka je jasna: kupiti veći i brži server. Međutim, ne može se večno primenjivati jer će eventualno nastati situacija u kojoj bolji i brži server nije moguće kupiti (preskup, ili čak - ne postoji). Tada je jedino logičko rešenje: kupiti 2 servera.

I ako zvuči jednostavno, kod relacionih baza podataka je teško podeliti bazu podataka na više fizičkih mašina. Npr. MySQL i PostgresSQL ne mogu da upravljaju bazom na više mašina koje mogu istovremeno i čitati i pisati u bazu[21]. Oracle DBMS podržava ovu funkcionalnost, ali je ona veoma kompleksna za implementiranje, a troškovi su veoma veliki.

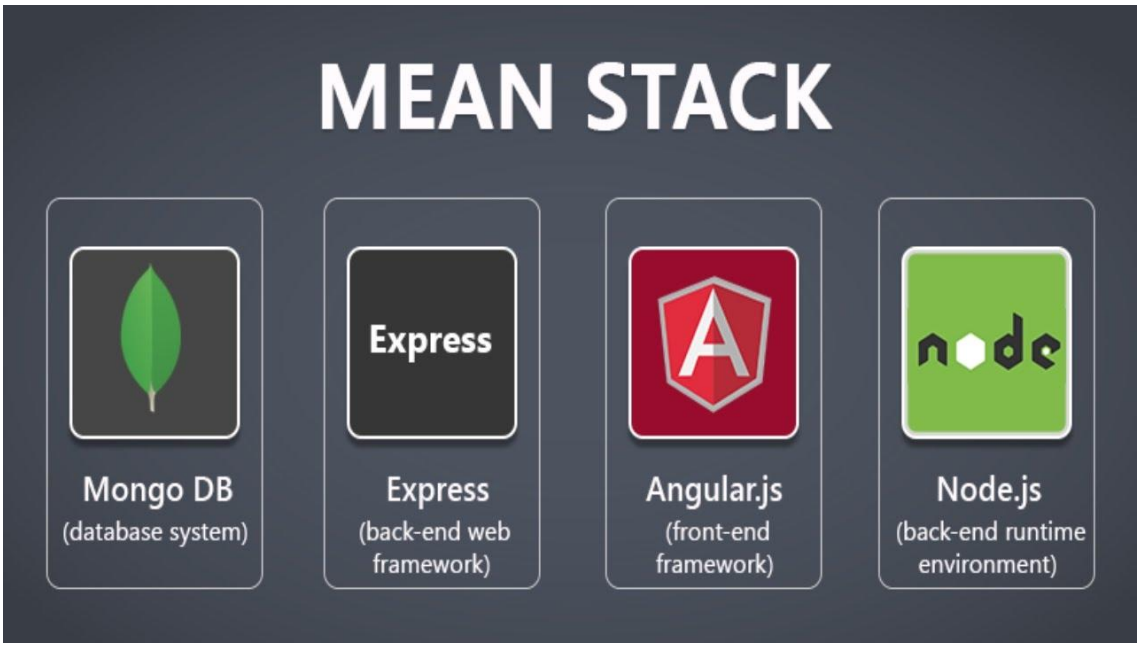
Kada se vrše upiti ka bazi podataka, ona mora pronaći sve podatke, obradi po zahtevu upita, i vrati rezultate pretrage. Međutim, baza je u velikom problemu ako se deo podataka nalazi na drugoj mašini.

Ukoliko BP skladišti malu količinu podataka i često manipuliše njima, potrebno je rasteretiti bazu na više servera. Sada BP dolazi do novog problema: izmene na jednom server moraju biti dostupne i na drugom. Dodatni problemi se javljaju ako se različite promene na dva servera dese u isto vreme.

Mongo rešava ove probleme tako što ih u potpunosti zaobilazi. Shodno tome da se podaci skladište u obliku dokumenta u BSON formatu, podaci se nalaze na jednom mestu i nisu ispreplitani relacijama. Upiti Mongo bazi se traže specifične parove ključa i vrednosti u dokumentima, a ovu informaciju je lako rasporediti na više servera. Svaki server proverava podatke koje ima i vraća ih.

Dva odvojena Mongo servera mogu podeliti podatke nad kojim operišu (sharding). Podaci se mogu raspodeliti na više servera, gde će svaki od njih manipulisati samo jednim određenim delom podataka. Npr. jedan server vrši upite vezane za korisnike biblioteke, a druga za knjige. Na ovaj način Mongo baza podataka se jednostavno može skalirati na stotine na servera, kao i na dva.

5. OPIS POSTUPKA IZRADE



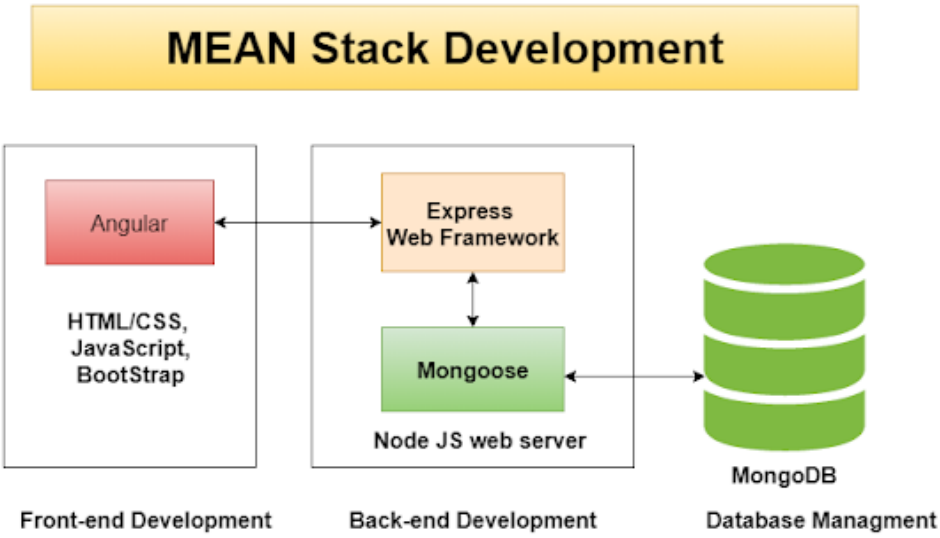
Slika 29. Grupa tehnologoija za izradu Javaskript aplikacija [27]

Pre početka izrade aplikacije, neophodno je instalirati NodeJS i MongoDB. Oni se mogu besplatno preuzeti sa svojih sajtova https://nodejs.org/en/download/ i https://www.mongodb.com/download-center/community.

Frontend aplikacija i backend aplikacija (korisničiki interfejs i sloj biznis logike) se razvijaju odvojeno. Na njima može raditi više ljudi ili timova paralelno i nezavisno jedni od drugih. Svaki tim samostalno radi svoju aplikaciju, i ne mora imati niti znanje niti uvid u rad one druge. U ranijim fazama razvoja, aplikacije neće imati nikakvog dodira ni kontakta, sve dok ne budu došle do faze kada mogu razmenjivati podatke.

Moguće i da jedna osoba radi na obema aplikacijama. Takav vid rada naziva se “Full stack development”.

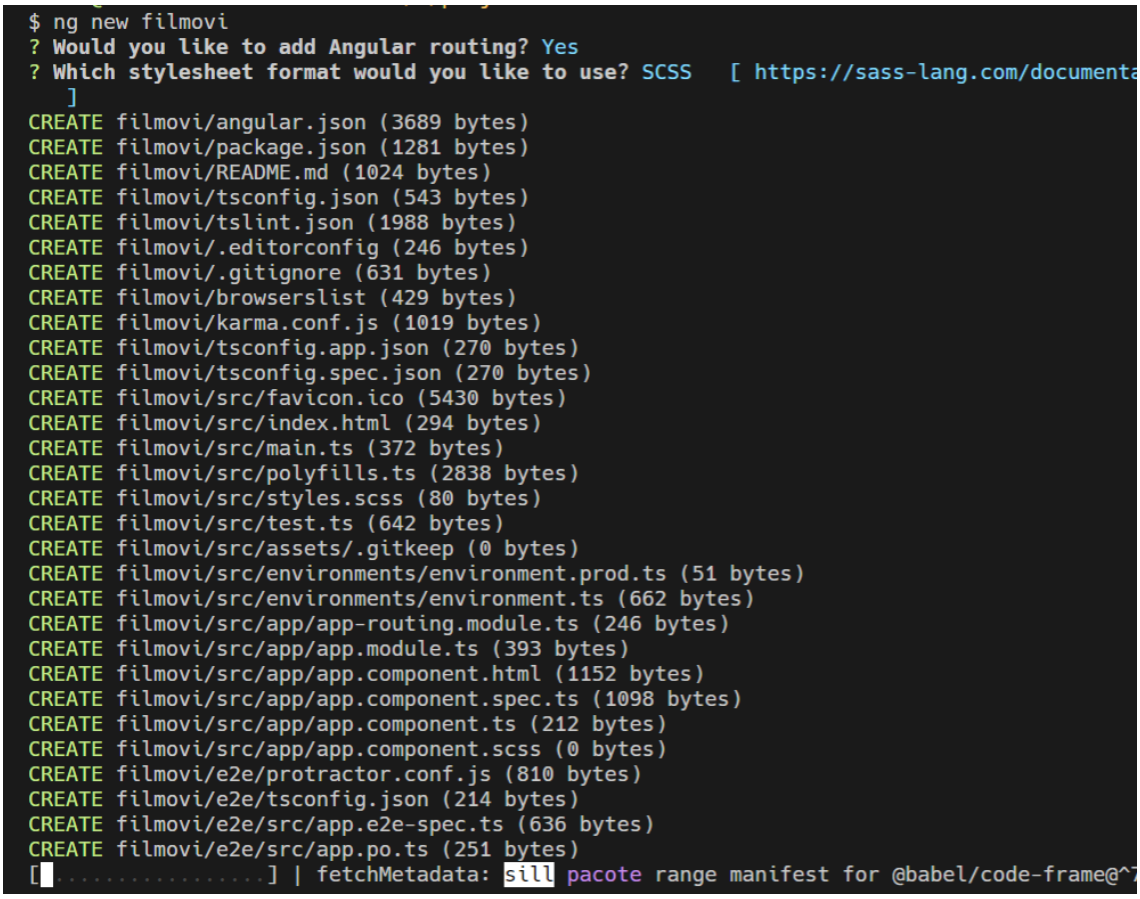
Zajednički alat za obe aplikacije je NodeJS, bez koga razvoj ne bi bio moguć. Menadžer paketa NPM koriste obe aplikacije za instaliranje biblioteka i radnih okvira.



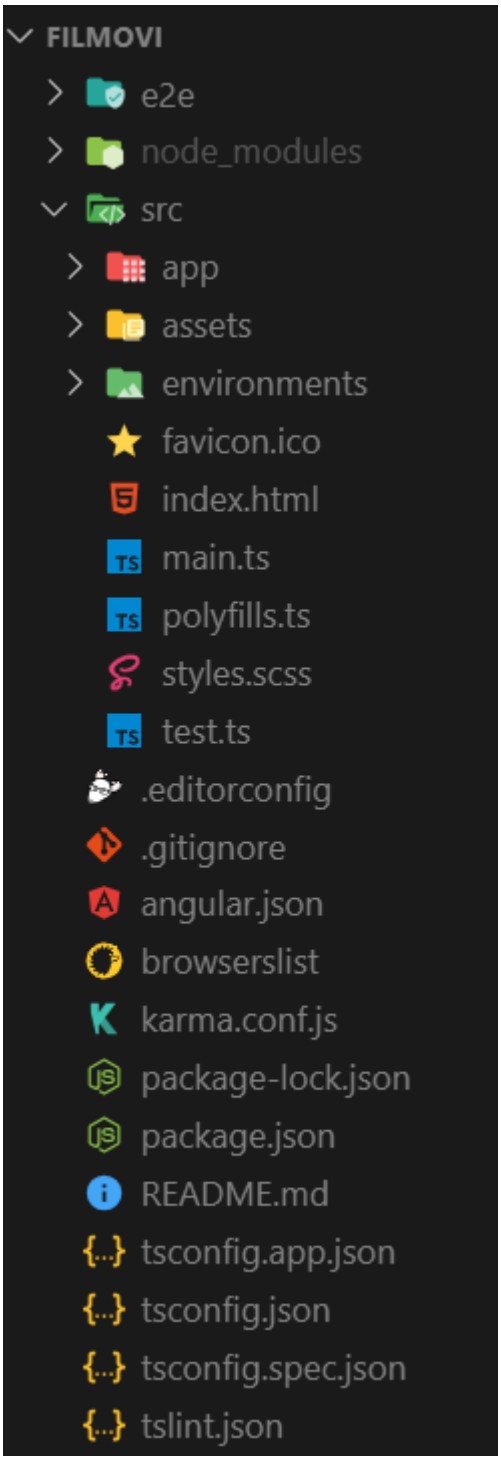
Slika 30. Arhitektura aplikacije u MEAN grupi alata [28]

5.1 Postavljanje strukture

Za izradu angular aplikacije, prvo treba instalirati Angular CLI, kojim će se upravljati aplikacijom. U komandnoj liniji, pišemo komandu “ng new filmovi”, koja će kreirati novi Angular projekat.



Slika 31. kreiranje novog Angular projekta

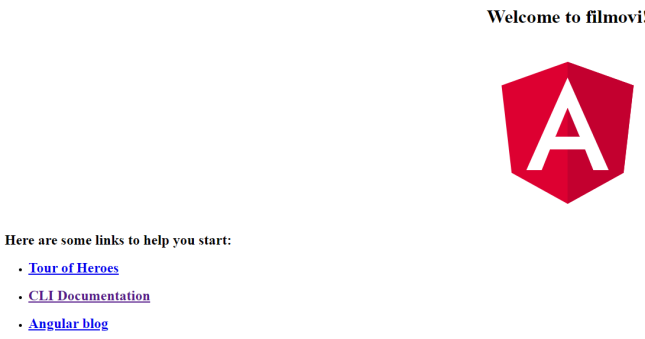


Slika 32. Struktura fajlova Angular aplikacije.

Komadnom “ng s”, aplikacija se pokreće i otvara u pretraživaču na lokaciji http://localhost: 4200.

```
$ ng s
 10% building 3/3 modules 0 activei [wds]: Project is running at http://localhost:4200/webpack-dev-server/
i [wds]: webpack output is served from /
i [wds]: 404s will fallback to //index.html

chunk {main} main.js, main.js.map (main) 11.5 kB [initial] [rendered]
chunk {polyfills} polyfills.js, polyfills.js.map (polyfills) 251 kB [initial] [rendered]
chunk {runtime} runtime.js, runtime.js.map (runtime) 6.09 kB [entry] [rendered]
chunk {styles} styles.js, styles.js.map (styles) 16.6 kB [initial] [rendered]
chunk {vendor} vendor.js, vendor.js.map (vendor) 4.1 MB [initial] [rendered]
Date: 2019-09-15T09:59:43.320Z - Hash: 06cc2c85db7a563be03b - Time: 5722ms
** Angular Live Development Server is listening on localhost:4200, open your browser on http://localhost:4200/
 **
i [wdm]: Compiled successfully.
```

Slika 33. Pokretanje Angular aplikacije

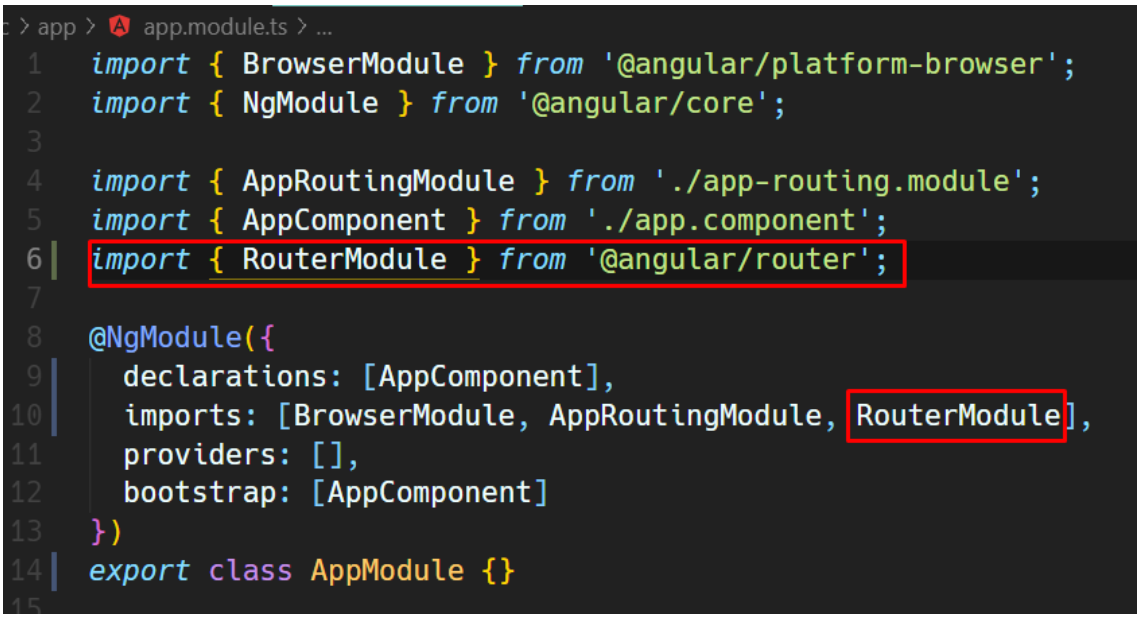


Slika 34. Uspešno kreirana i pokrenuta Angular aplikacija.

Aplikaciji je potrebno da ima dve stranice: jedna za izlistavanje filmova, druga za izlistavanje detalja jednog filma. S obzirom da je Angular aplikacija “Single Page”, kreiranje novih HTML fajlova nije moguće.

Angular poseduje modul za za rad sa rutama i stranicama, koj se zove “RouterModule”. Ovaj modul poseduje sve potrebne alate za rutiranje u aplikaciji.

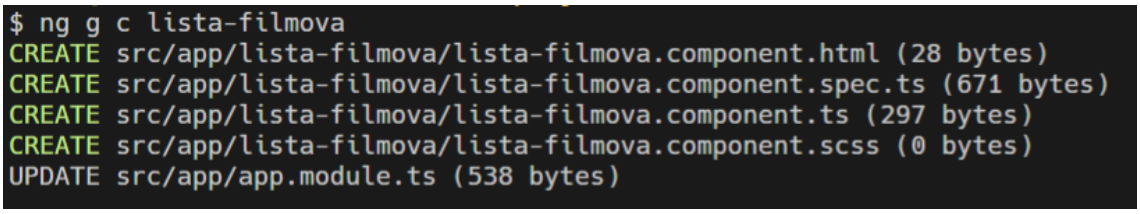
Modul je potrebno prvo importovati, potom u app.component staviti `<router-outlet></router-outlet>` direktivu. Ova direktiva dolazi i Router modula, koja se ponaša kao komponenta. Ona služi kao pokazatelj na mesto gde treba dinamički stavljati i sklanjati komponente, što simulira menjanje stranica u tradicionalnim aplikacijama [29].



Slika 35. Importanje Router modula

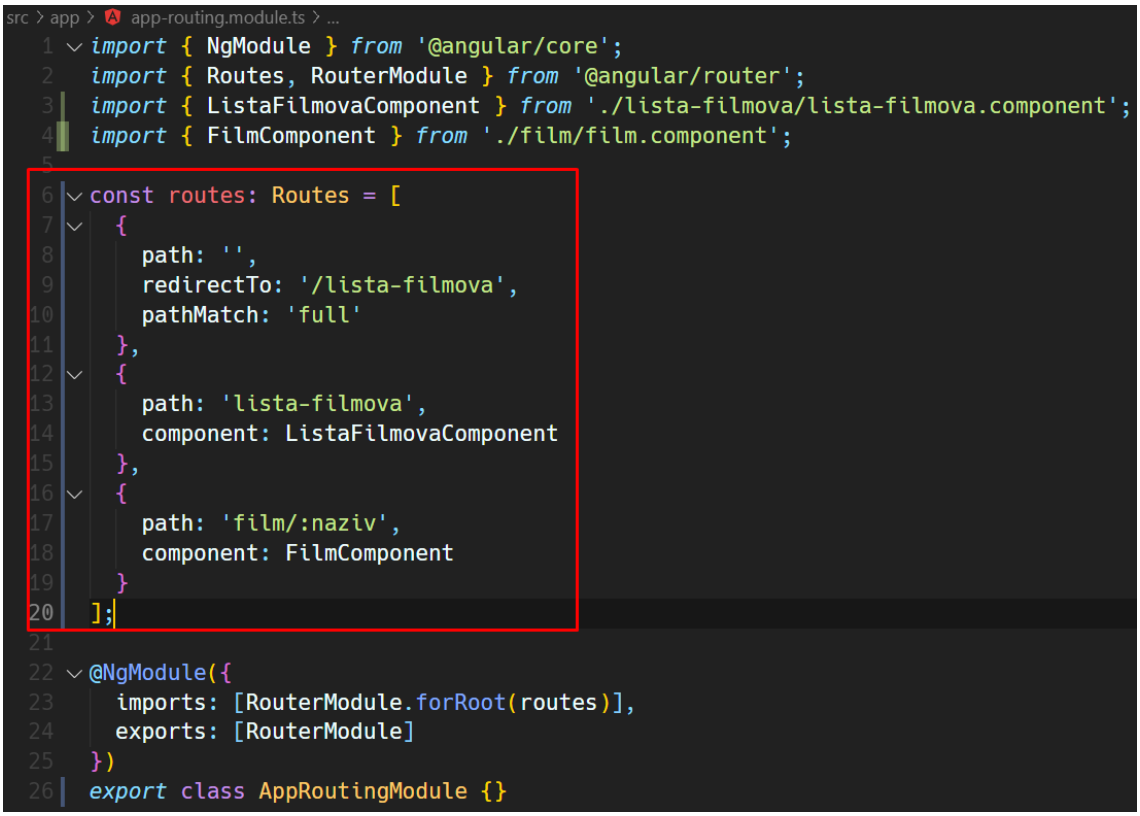
Slika 36. Označavanje mesta za rutiranje komponenti.

Stranice za film i listu filmova generisaće se kao komponente, koje će ruter smenjivati po potrebi. Komanda za kreiranje komponente je “ng g c lista-filmova”. Isto važi i za komponentu “film”.



Slika 37. Generisanje komponente

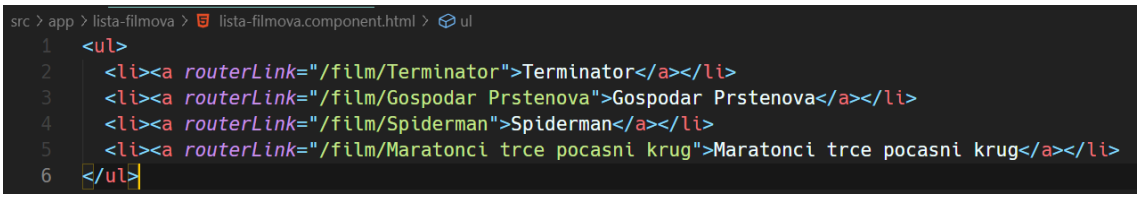
Podešavanje ruta vrši se u fajlu “app-routing.module”, u kome se definišu rute koje će se koristiti u tom modulu. Nazivi ruta će biti “/lista-filmova” i “/film”. Ruta “film” će imati i parametar, koji će označavati ime filma.



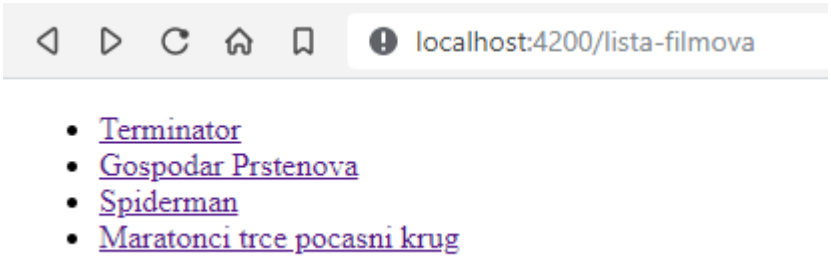
Slika 38. Definisanje ruta

U komponent list filmova, dodaćemo listu filmova sa statičnim podacima. Kasnije će ova lista biti popunjavana podacima sa servera.

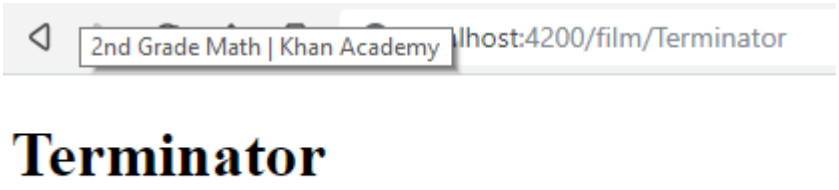
Na svakoj listi postojaće link do njegove stranice. Za linkove se koristi direktiva “routerLink”, umesto uobičajenog “href” atributa. Atribut “href” bio naložio pretraživaču da otvori neku drugu stranicu sa tom oznakom, čine bi se napustila trenutna aplikacija. Klikom na link, otvoriće se stranica tog konkretnog filma.



Slika 39a. Lista filmova sa routerLink direktivom



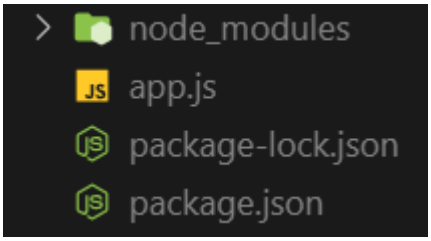
Slika 39b. Lista filmova u pretraživaču



Slika 39c. Otvaranje stranice filma

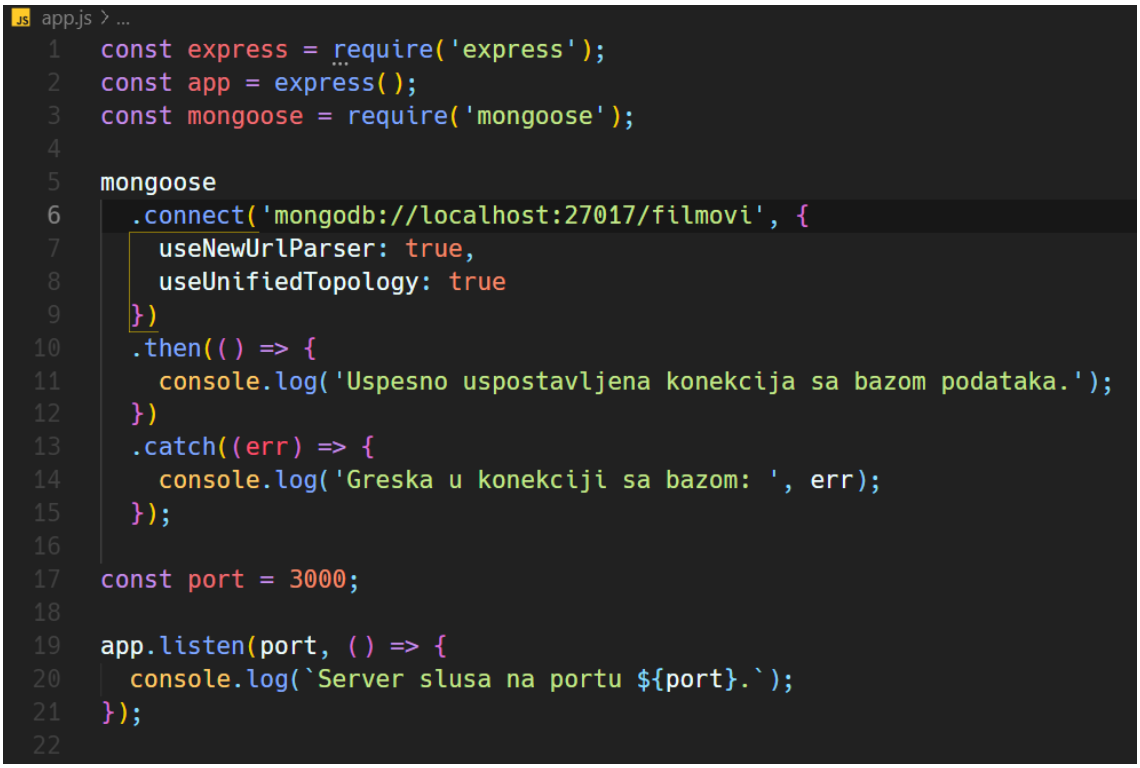
5.2 API

API je posebna aplikacija, koja se radi u posebnom folderu. Komandom “npm init -y” inicijalizuje NodeJS projekat, a “npm install express” preuzima i instalira express biblioteku.



Slika 40. Početna struktura API-a

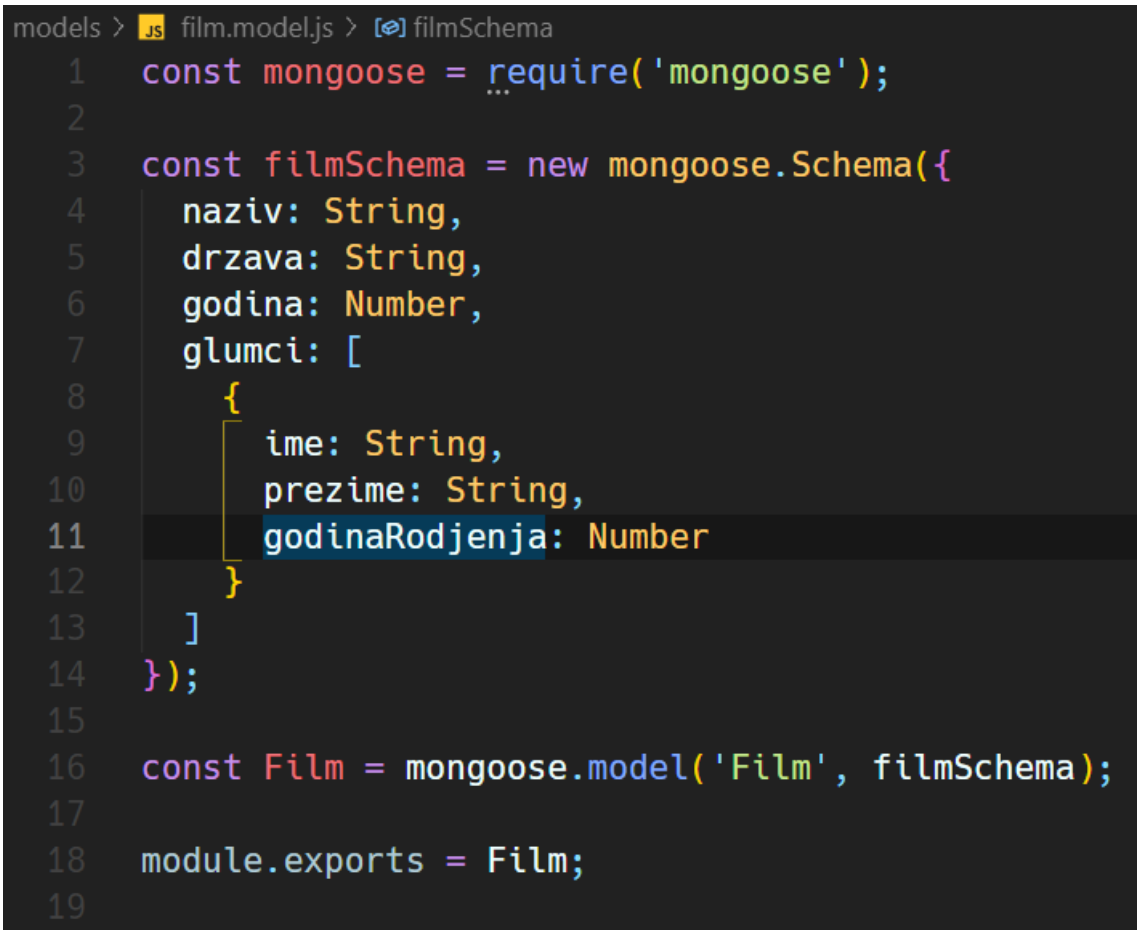
Fajl “app.js” sadrži svu konfiguraciju za pokretanje servera i konekciju sa bazom podataka. Za lakši rad sa bazom podataka, koristi se ORM (Object Relationship Mapper) “mongoose”, koja olakšava uspostavljanje konekcije sa bazom, daje mogućnost pisanja modela podataka, i pruža više mogućnosti nego zvanični drajver. Bazu nije potrebno kreirati, jer to čini mongo automatski po upisu u nju.



Slika 41. API server i baza podataka.

Iako kod nerelacione baze nije potrebno definisati strukturu podataka, korisno je znati kojim podacima aplikacija raspolaže, kako bi se olakšala komunikacija između baze podataka i servera. Između njih stoji ORM, koji šemu podataka pretvara u objekat sa atributima i metodama koji se koriste u biznis logici.

Takođe nije potrebno praviti dve šeme za dva entiteta. Upiti će biti brži ukoliko se svi podaci nalaze u jednoj kolekciji, stoga se dokument glumac ugrađuje (“embeduje”) u dokument filmova.



Slika 42. Šema i model podataka sa 2 entiteta

Zadatak servera je pronađe podatke u bazi podataka i prosledi klijentu. Takođe, mora na zahtev klijenta podatke uneti u bazu, obrisati ili izmeniti. Stoga se kreiraju rute, na koje će klijenti slati zahteve i dobijati odgovore.

```
router.get('/', async (req, res) => {
  const godina = req.query.godina;

  try {
    let filmovi = [];

    if (godina) {
      filmovi = await Film.find({ godina });
    } else {
      filmovi = await Film.find();
    }

    res.status(200).json(filmovi);
  } catch (error) {
    res.status(500).json({ greska: 'Greska pri pronalazenju filmova' });
  }
});
```

Slika 43. Ruta za čitanje filmova

```
33  router.post('/', async (req, res) => {
34    try {
35      const noviFilm = new Film(req.body);
36      await noviFilm.save();
37
38      res.status(201).json(noviFilm);
39    } catch (error) {
40      res.status(500).json({ greska: error });
41    }
42  });
```

Slika 44. Ruta za kreiranje novog filma i smestanje u bazu

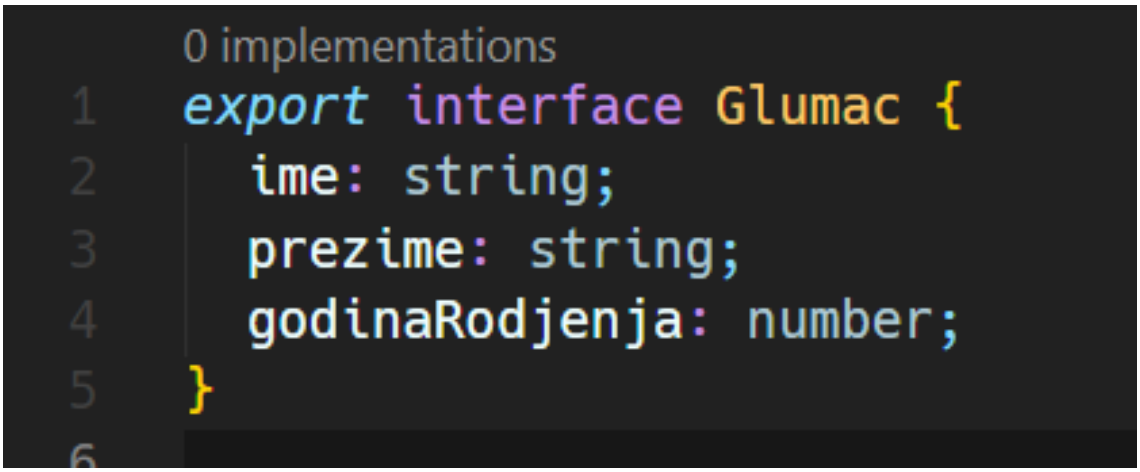
5.3 Servis i modeli podataka

Kako bismo iskoristili prednosti statičkog tipiziranja koje omogućava jezik tajpskript, kreiraćemo model za filmove sa podacima koji će se primati sa servera i prikazitai na korisničkom interfejsu.

Ovi modeli pisaće se u obliku interfejsa, kao što bi i u drugom objektno orijentisanim jezicima.

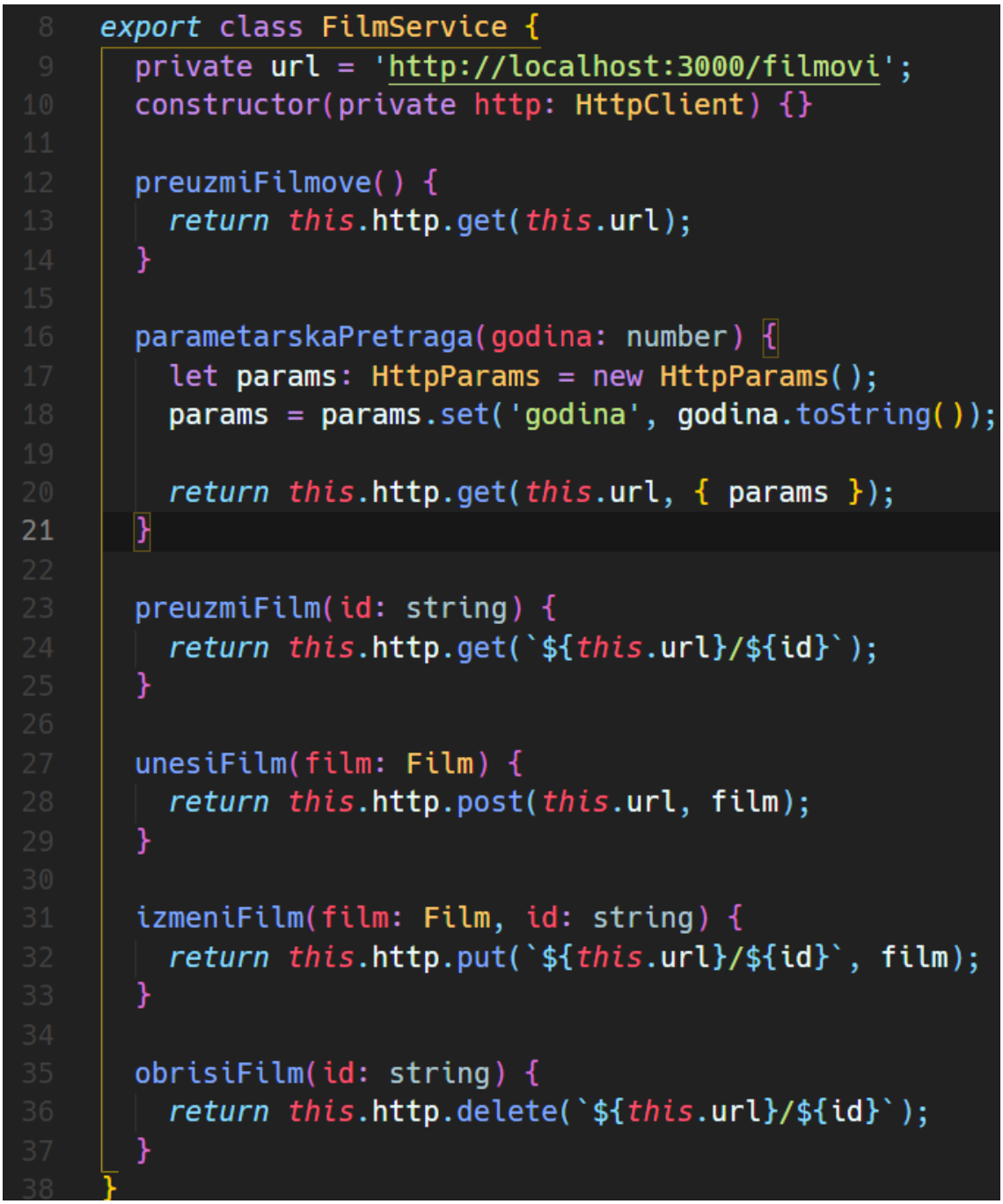
```
1   import { Glumac } from './glumac.model';
2
    1 implementation
3   export interface Film {
4     _id: string;
5     naziv: string;
6     godina: number;
7     drzava: string;
8     glumci: Glumac[];
9   }
10
```

Slika 45. Model filma



Slika 46. Model glumca

Sva komunikacija sa serverom, obavlja se u servisu. U njemu se definišu metode koje će slati HTTP zahteve serveru sa željenim parametrima, a povratne informacije će proslediti komponenti, koje će ih renderovati u komponenti.



```
export class FilmService {
  private url = 'http://localhost:3000/filmovi';
  constructor(private http: HttpClient) {}

  preuzmiFilmove() {
    return this.http.get(this.url);
  }

  parametarskaPretraga(godina: number) {
    let params: HttpParams = new HttpParams();
    params = params.set('godina', godina.toString());

    return this.http.get(this.url, { params });
  }

  preuzmiFilm(id: string) {
    return this.http.get(`${this.url}/${id}`);
  }

  unesiFilm(film: Film) {
    return this.http.post(this.url, film);
  }

  izmeniFilm(film: Film, id: string) {
    return this.http.put(`${this.url}/${id}`, film);
  }

  obrisiFilm(id: string) {
    return this.http.delete(`${this.url}/${id}`);
  }
}
```

Slika 47. Servis za kokmunikaciju sa serverom

5.4 Komponente i prezentaciona logika

Metode iz servisa poziva komonenta pri inicijalizaciji kako bi dobila filmove za prikaz. Takođe, unosom godine u polje, biće poslat zahtev za filtriranjem filmova po godini.

```
export class ListaFilmovaComponent implements OnInit {
  filmovi: Film[] = [];
  pretraga: FormGroup;

  constructor(private filmService: FilmService, private fb: FormBuilder)
    this.pretraga = this.fb.group({
      godina: ['']
    });
  }

  parametarskaPretraga() {
    this.filmService
      .parametarskaPretraga(this.pretraga.get('godina').value)
      .subscribe((filmovi: Film[]) => {
        this.filmovi = filmovi;
      });
  }

  ngOnInit() {
    this.filmService.preuzmiFilmove().subscribe((filmovi: Film[]) => {
      console.log(filmovi);
      this.filmovi = filmovi;
    });
  }
}
```

Slika 48. Komponenta liste filmova, inicijalni podaci i filtriranje

```
<table class="table">
  <thead>
    <tr>
      <th scope="col">#</th>
      <th scope="col">Naziv</th>
      <th scope="col">Godina</th>
      <th scope="col">Drzava</th>
    </tr>
  </thead>
  <tbody>
    <tr *ngFor="let film of filmovi; let i = index">
      <th scope="row">{{ i + 1 }}</th>
      <td
        ><a [routerLink]="['/film', film._id]">{{ film.naziv }}</a></td
      >
      <td>{{ film.godina }}</td>
      <td>{{ film.drzava }}</td>
    </tr>
  </tbody>
</table>
```

Slika 49. Tabela za dinamičko izlistavanje filmova

Unos filma vrši se putem forme, u koju se unose željeni podaci. Angular će unete podatke

pretvoriti u JSON format, koji će se proslediti putem servisa serveru na skladištenje.

Izmena filma je skoro identična. Razlika je u tome što će forma za izmenu prvo zahtevati željeni film, i popuniti formu njegovim podacima. Zatim se oni mogu menjati po želji, potom proslediti serveru na obradu.

```
export class UnosComponent {
  unosForma: FormGroup;

  constructor(private fb: FormBuilder, private filmService: FilmService) {
    this.unosForma = this.fb.group({
      naziv: [''],
      godina: [''],
      drzava: [''],
      glumci: this.fb.array([])
    });
  }

  unesiFilm() {
    this.filmService.unesiFilm(this.unosForma.value).subscribe(
      () => {
        alert('Film uspesno unet');
      },
      () => {
        alert('Greska pri unosenju filma');
      }
    );
  }
```

Slika 50. Komponenta za unos filma i kreiranje forme

Svaki film ima svoju stranicu, u kojoj pokazuje više detalja, nego u tabelarnom prikazu. Tu se nalaze dugme za brisanje, koje će proslediti serveru id filma koji je potrebno obrisati.

```
29    obrisiFilm() {
30      this.filmService.obrisiFilm(this.id).subscribe(
31        () => {
32          alert('Film uspesno obrisan.');
33          this.router.navigate(['/lista-filmova']);
34        },
35        () => {
36          alert('Greska pri brisanju');
37        }
38      );
39    }
```

Slika 51. Funkcija za brisanje filma

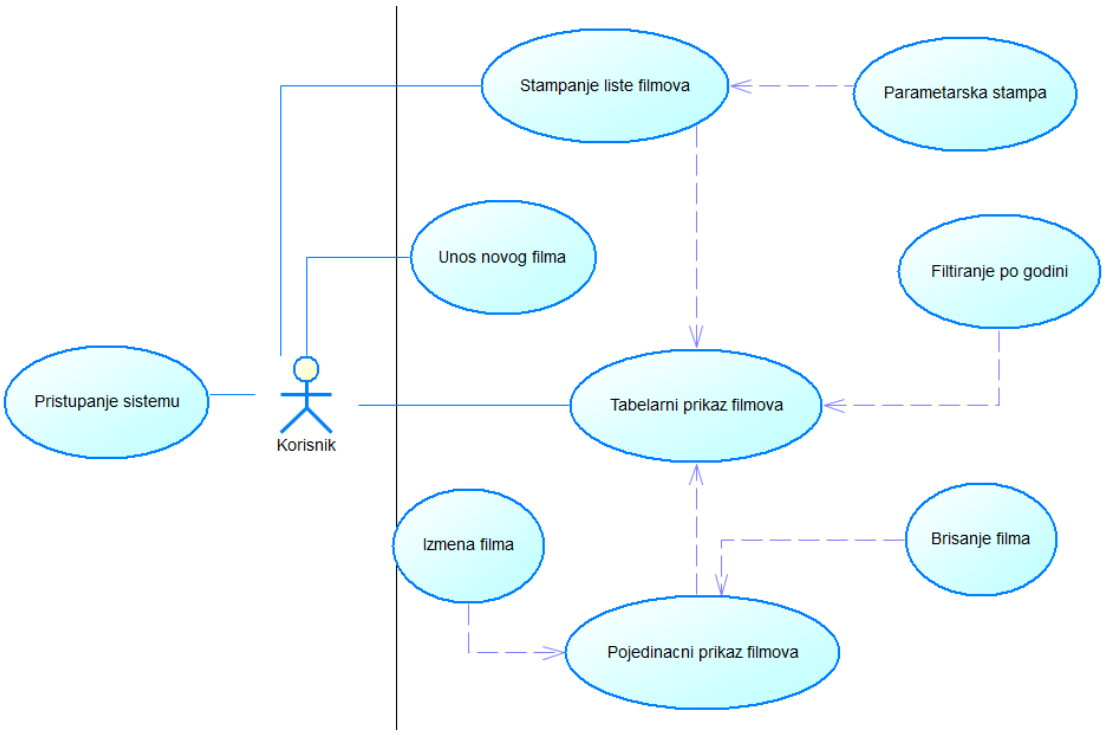
6. REALIZOVAN PRIMER

6.1 Modeli

Ovo poglavlje predstavlja use case dijagram, konceptualni model, dijagram komponenti, dijagram razmeštaja i dijagram sekvenci za jedan slučaj korišćenja za primer opisan u u poglavlju 5.

6.1.1 Use case dijagram

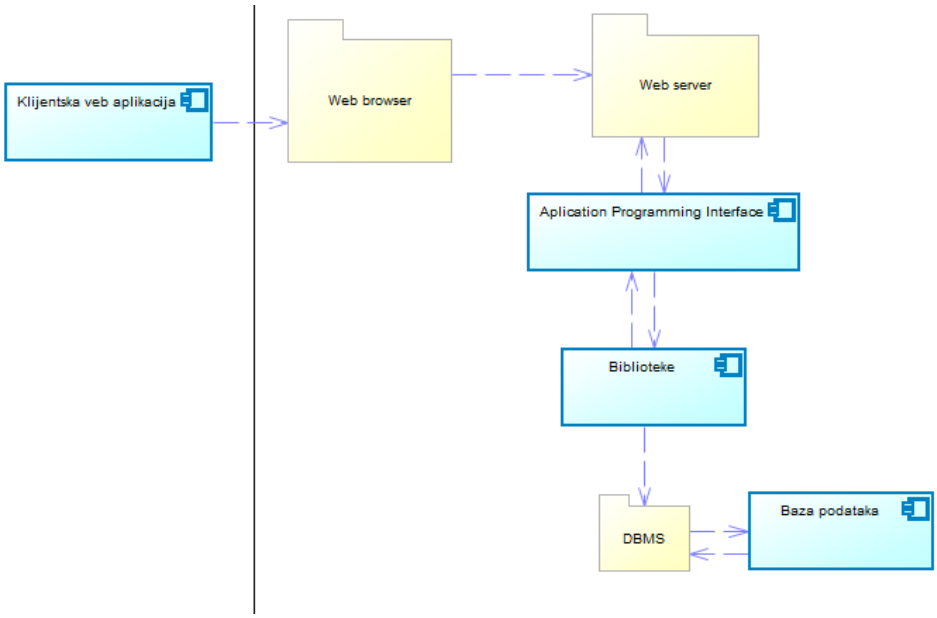
Na slici je predstavljen use case dijagram.



Slika 52. USE CASE dijagram

6.1.2 Dijagram komponenti

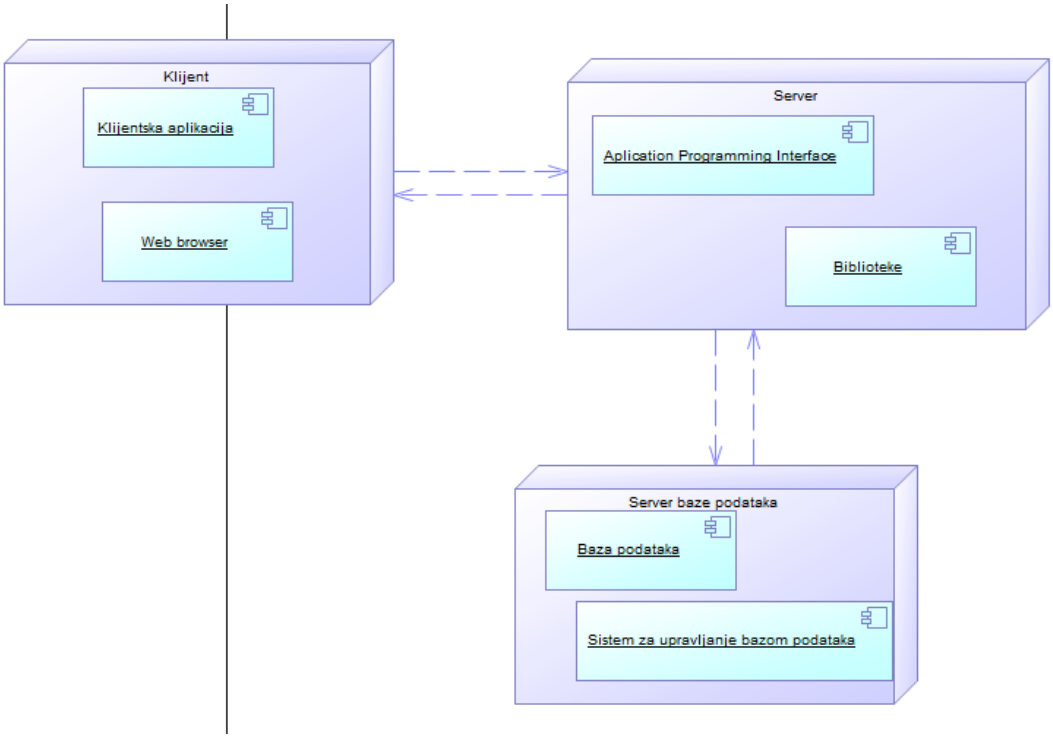
Na sledećoj slici prikazan je dijagram komponenti.



Slika 53. Dijagram komponenti

6.1.3 Dijagram razmeštaja

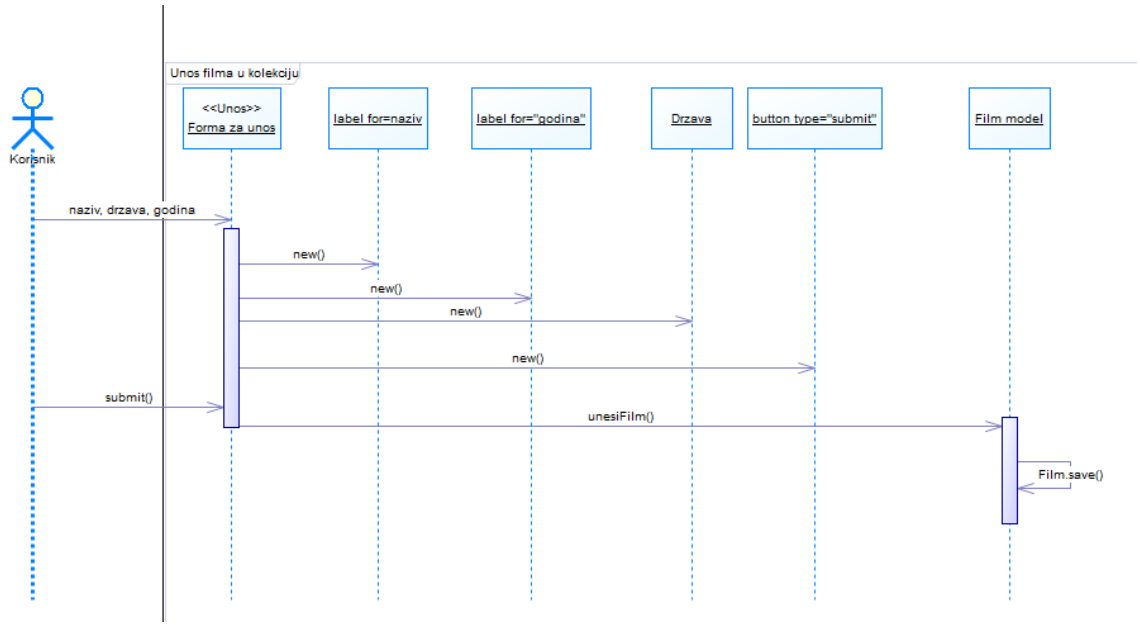
Sledeća slika predstavlja dijagram razmeštaja.



Slika 54. Dijagram razmeštaja

6.1.4 Dijagram sekvenci za jedan slučaj korišćenja

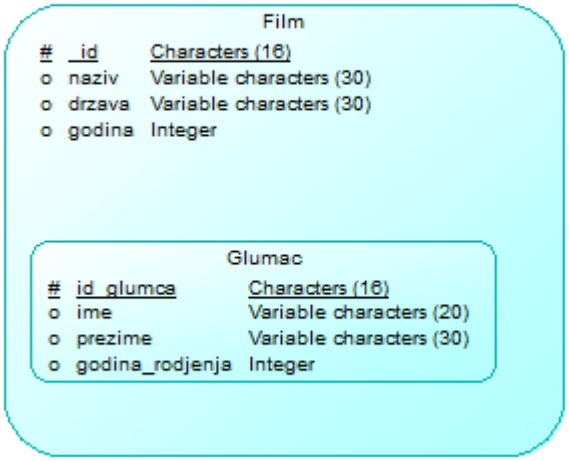
Na narednoj slici je predstavljen dijagram sekvenci za slučaj korišćenja – unos novog filma u kolekciju.



Slika 55. Dijagram sekvenci

6.1.5 Konceptualni dijagram

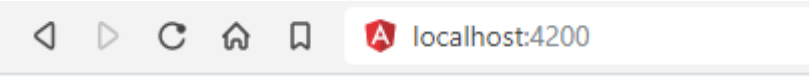
Na sledećoj slici prikazan je konceptualni dijagram.



Slika 56. Konceptualni dijagram

6.2 Korisničko uputstvo

Potrebno pokrenuti Angular aplikaciju komandom “ng s”, i Express aplikaciju komandom “node app”. Aplikaciju potom otvoriti u veb pretraživaču na adresi http://locahost:4200.



Slika 57. URL veb aplikacije

6.2.1 Tabelarni prikaz i filtriranje

Po otvaranju aplikacije, prikazaće se tabelarno podaci o postojećim filmovima. U polje “pretraga po godini” ukucati godinu nastanka filma za listu svih filmova nastalih te godine.

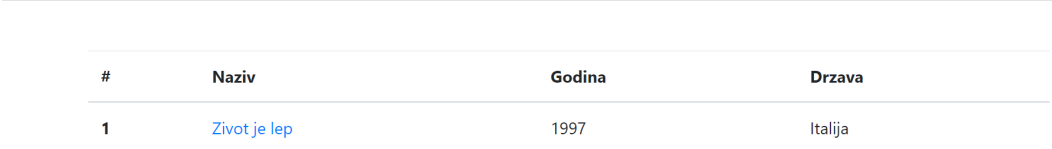
Slika 58. Stranica tabelarni prikaz

Slika 59. Filtriranje filmova po godini nastanka

6.2.2 Štampa i parametarska štampa

Na stranici tabelarnog prikaza, ispod polja za unos godine. Nalazi se dugme “Stampaj”. Klikom na ovo dugme, otvoriće se stranica prilagođena štampanju sa listom svih filmova.

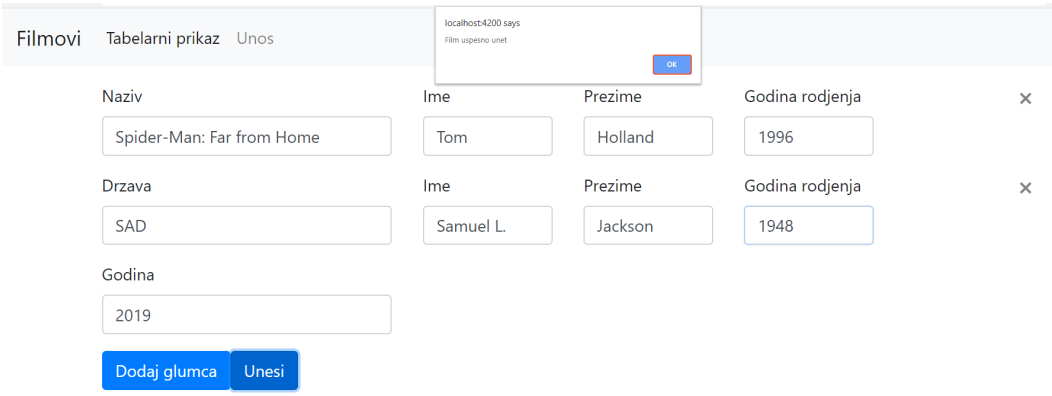
Ukoliko je prethodno popunjeno polje za godinu, lista za štampanje će sadržati samo filtrirane podatke. Navigacioni meni i polje za filtriranje su uklonjeni sa stranice za štampu.



Slika 60. Parametarska štampa.

6.2.3 Unos novog filma

Klikom na dugme “Unos” na navigacionom meniju, otvara se stranicu sa formom za unos novog filma. Klikom na dugme “Dodaj glumca”, pojavljuju se dodatna polja za unos podataka o glumcu. Broj glumaca nije ograničen, a konkretan glumac se briše klikom na dugme “x” sa njegove desne strane. Klikom na “Unesi”, aplikacija prikazuje povratnu poruku o uspešnosti unosa.



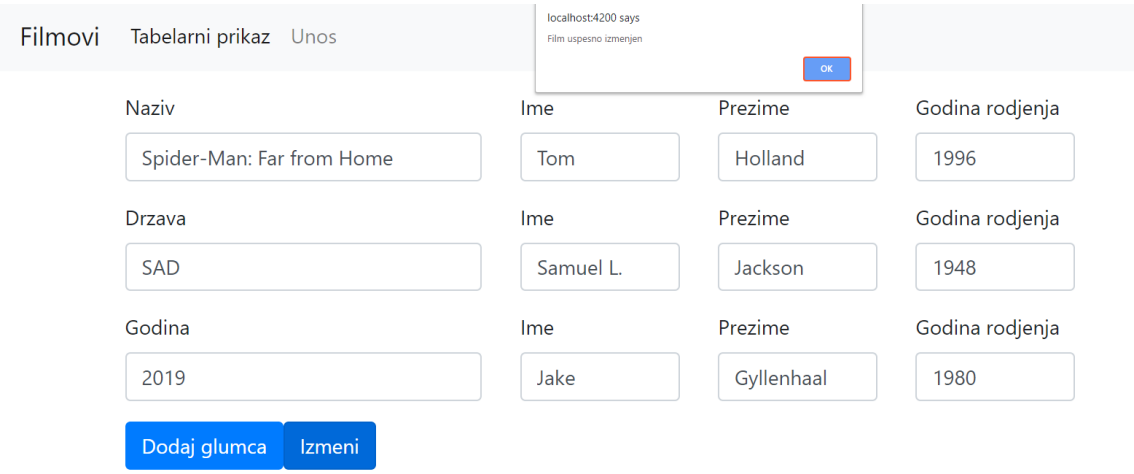
Slika 61. Unos novog filma sa glumcima

6.2.4 Izmena i brisanje filma

Svaki od filmova u tabelarnom prikazu može biti kliknut, kako bi se otvorila stranica sa više detalja u tom filmu. Takođe, tu se nalaze i tasteri za izmenu ili brisanje tog filma.

Slika 62. Pojedinačni prikaz filma sa tasterima za izmenu i brisanje

Klikom na taster “Izmeni”, otvara se ista forma kao i za unos, sa tim da je ova unapred popunjena postojećim podacima. Podaci se mogu izmeniti, potom potvrditi tasterom “Izmeni”.



Slika 63. Dodavanje glumca već postojećem filmu

Klikom na dugme “Obriši”, film se briše iz baze podataka i ne pojavljuje se više u tabelarnom prikazu.

6.3 Tabelarni prikaz slojeva i podslojeva aplikacije

Glavni sloj	**Rest Arhitektura**	**Podsloj**	**Tehnološka implementacija**	

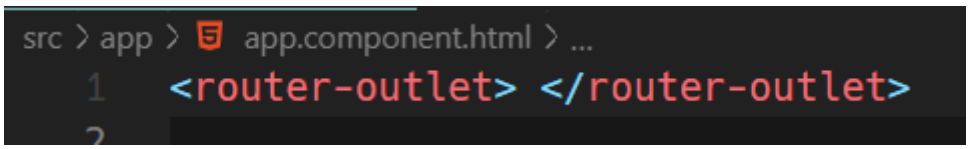
Prezentacioni sloj	Klijent	Korisinčki interfejs	Angular	Komponente
Sloj servisa	Klijent	Prezentaciona logika	Angular	Anguar servisi
Sloj poslovne logike	Server API		Express REST server	
Sloj za rad sa podacima	Server baze podataka	Sloj modela	Mongoose ORM	
Baza podataka		Nerelaciona	MongDB baza podataka	

Tabela 1. Slojevi i podslojevi aplikacije

6.4 Delovi programskog koda sa objašnjenjem

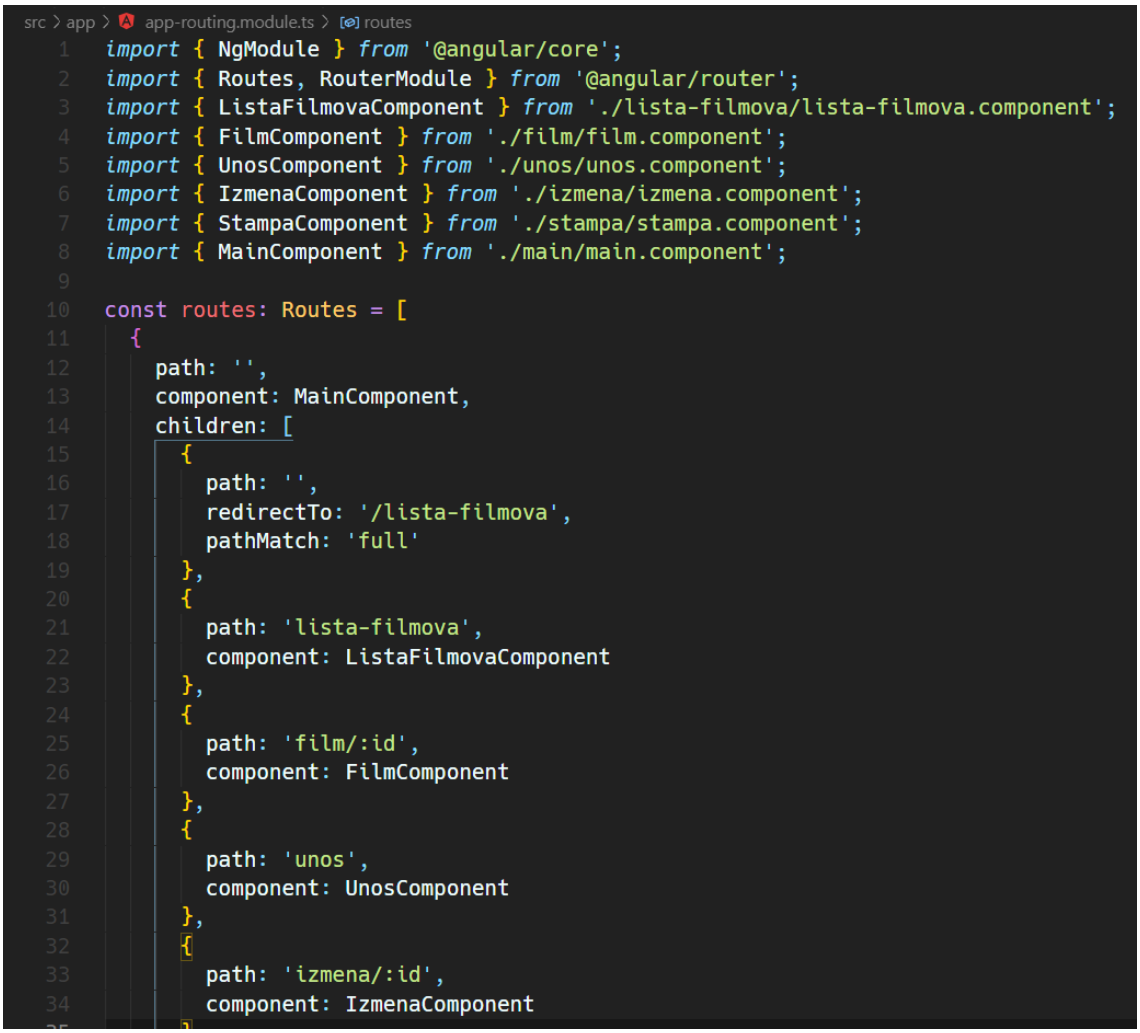
6.4.1 Sloj prezentacione logike i korisnički interfejs

Angular aplikacija ima samo jedan index.html, koji se dinamički menja kako bi postigao efekat menjanja stranica.



Slika 64. Router outlet

Router outlet je direktiva zadužena za učitavanje i sklanjanje komponenti tokom navigacije, a same komponente za navigaciju i njihove rute definišu se u routing modulu.



Slika 65. Komponente i rute na kojima se nalaze.

6.4.2 Sloj servisa i prezentaciona logika.

Servisi služe da se komponente rasterete teške logike i fokusiraju na samu prezentaciju. Sloj servisa poziva eksterne servere, ponašajući se kao spona između korisničkog interfejsa i biznis logike. Logika u servisu se piše jednom, i koristi svim komponentama kojima je to potrebno.

```
export class FilmService {
  private url = 'http://localhost:3000/filmovi';
  constructor(private http: HttpClient) {}

  preuzmiFilmove() {
    return this.http.get(this.url);
  }

  parametarskaPretraga(godina: number) {
    let params: HttpParams = new HttpParams();
    params = params.set('godina', godina.toString());

    return this.http.get(this.url, { params });
  }

  preuzmiFilm(id: string) {
    return this.http.get(`${this.url}/${id}`);
  }

  unesiFilm(film: Film) {
    return this.http.post(this.url, film);
  }

  izmeniFilm(film: Film, id: string) {
    return this.http.put(`${this.url}/${id}`, film);
  }

  obrisiFilm(id: string) {
    return this.http.delete(`${this.url}/${id}`);
  }
```

Slika 66. Servis kao most između interfejsa i servera

6.4.3 Sloj biznis logike

Na ovom sloju se nalazi API koji na definisanim rutama osluškuje zahteve klijenata, i ispunjava ih kad se dese. Ovi zahtevi se tiču manipulacijom i upitima baze podataka, pisanjem i čitanjem sa diska ili kalkulacijama i proračunima Svaka od ruta je zadužena za jednu konkretnu operaciju.

```
router.put('/:id', async (req, res) => {
  const id = req.params.id;
  const izmene = req.body;

  try {
    let film = await Film.findByIdAndUpdate(id, izmene, { new: true });

    res.status(200).json(film);
  } catch (error) {
    res.status(500).json({ greska: error });
  }
});

router.delete('/:id', async (req, res) => {
  const id = req.params.id;

  try {
    await Film.findByIdAndRemove(id);

    res.status(200).json({ message: 'Film obrisan' });
  } catch (error) {
    res.status(500).json({ greska: error });
  }
});
```

Slika 67. Rute za izmenu i brisanje iz baze podataka.

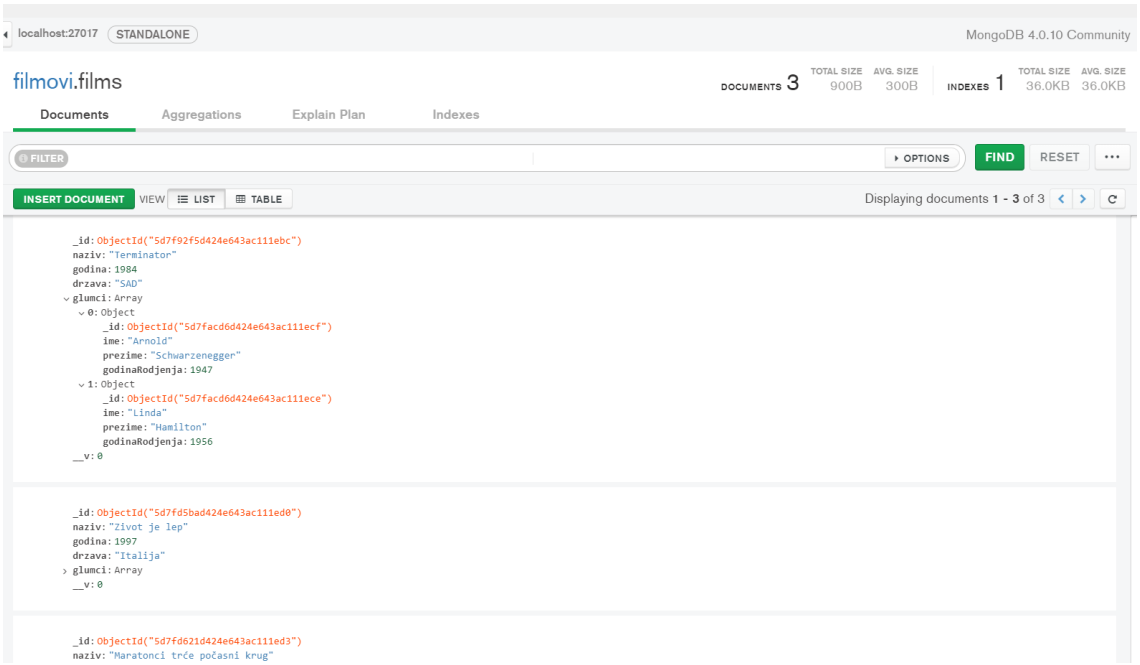
6.4.4 Sloj modela i baze podataka

Sloj modela je međusloj između baze podataka i biznis logike. Definiše modele podataka sa atributima i metodama koje koristi sloj biznis logike, a iste te modele prosleđuje direktno bazi podataka.

```
const mongoose = require('mongoose');

const filmSchema = new mongoose.Schema({
  naziv: String,
  drzava: String,
  godina: Number,
  glumci: [
    {
      ime: String,
      prezime: String,
      godinaRodjenja: Number
    }
  ]
});

const Film = mongoose.model('Film', filmSchema);

module.exports = Film;
```

Slika 68. Šema podataka pretvorena u model

Sloj baze podataka je instanca MongoDB servera. Na njemu se vrši direktna manipulacija celom bazom, čitanje, pisanje, ali i adminsistriranje i skaliarnje. Serveru se može pristupiti komandnom linijom i zadavati instrukcije, ili koristiti neki od grafičkih interfejsa.



Slika 69. Grafički prikaz baze podataka

7. ZAKLJUČAK

U ovom radu opisan je postupak izrade brzih i skalabilnih aplikacija, koristeći Javaskript jezik u svim nivoima aplikacije, sa primerom datim u kolekciji filmova i glumaca.

Postupak korišćenja ovih tehnologija preuzet je sa zvaničnih prezentacija. Predstavljeno je 5 svetski poznatih organizacija koje svoje poslovanje zasnivaju na ovim tehnologijama.

Postupak izrade je objašnjen za svaki deo aplikacije, kreirane su dve odvojene aplikacije koje tesno sarađuju i komuniciraju. Dobijena aplikacije je opisana je sa 5 vrsta dijagrama: USE CASE dijagram, dijagram komponenti, dijagram razmeštaja, CDM i dijagram sekvenci za jedan slučaj korišćenja.

Kako bi se lakše stekao uvid u strukturu aplikacije, tabelarno je predstavljen svaki sloj i podsloj i način njihove implementacije, i dato je uputstvo za korišćenje izrađene aplikacije.

Izrađena aplikacija je vrlo jednostavna i minimalistična, ali je izuzetno prilagodljiva i podobna za mnoga proširenja.

Moguća proširenja su:

- Redizajniranje boljim dizajnim
- Ubaciti animacije
- Unaprediti performanse tehnikama reaktivnog programiranja
- Dodati mogućnost za unošenje više podataka vezanih za film, kao što je žanr, režiser, a za glumce njihove biografije i dostignuća
- Kreirati mobilnu aplikaciju koristeći Ionic radni okvir
- Kreirati desktop aplikaciju koristeći Electron radni okvir

8. LITERATURA

1. https://docs.microsoft.com/en-us/azure/architecture/data-guide/big-data/non-relational-data
2. https://auth0.com/blog/a-brief-history-of-javascript/
3. Osnove NodeJS-a, https://nodejs.dev/introduction-to-nodejs
4. Arhitektura REST API, https://restfulapi.net/rest-architectural-constraints/
5. https://www.seobility.net/en/wiki/REST_API
6. https://restfulapi.net/
7. https://idratherbewriting.com/learnapidoc/docapis_what_is_a_rest_api.html
8. https://www.paypal.com/us/home
9. https://naturaily.com/blog/node-js-applications
10. https://www.linkedin.com/
11. https://brainhub.eu/blog/9-famous-apps-using-node-js/
12. https://www.netflix.com
13. https://www.uber.com/
14. https://www.nasa.gov/
15. https://foundation.nodejs.org/wp-content/uploads/sites/50/2017/09/Node_CaseStudy_Nasa_FNL.pdf
16. Angular arhitektura, https://angular.io/guide/architecture
17. Arhitektura Angula modula, https://angular.io/guide/architecture-modules
18. https://vitalflux.com/wp-content/uploads/2016/02/components.png
19. Angular direktive, https://angular.io/guide/architecture-components#directives
20. Angular ugrađene cevi, https://angular.io/guide/pipes#built-in-pipes
21. Aritektura Angular servisa, https://angular.io/guide/architecture-services
22. Osnove Angular provajderi, https://angular.io/guide/providers
23. Osvnoce Angular CLI, https://angular.io/cli
24. Tehnoglogije koje koristi NodeJS, https://nodejs.org/en/docs/meta/topics/dependencies/#v8
25. Express zvanični sajt, https://expressjs.com/
26. Eelco Plugge, Peter Membrey, Tim Hawkins – The Definitive Guide to MongoDB (2010)
27. https://i.ytimg.com/vi/Lzi2xYQdwWc/maxresdefault.jpg
28. https://4.bp.blogspot.com/-zQRHoujtNTQ/WiK1d8AqKOI/AAAAAAAAF_U/FPhQIEebkmM9gp6TYxrADhSFugqmxPNVwCLcBGAs/s1600/mean%2Bstack%2Bdevelopment.png
29. Osnove Angular router outlet-a, https://angular.io/guide/router#router-outlet