UNIVERZA V MARIBORU
FAKULTETA ZA ELEKTROTEHNIKO,
RAČUNALNIŠTVO IN INFORMATIKO

Alen Karahasanović

SPLETNA APLIKACIJA ZA ANALIZO TELEVIZIJSKE GLEDANOSTI Z UPORABO OGRODIJ ANGULARJS IN D3.JS

Diplomsko delo

Maribor, avgust 2016

SPLETNA APLIKACIJA ZA ANALIZO TELEVIZIJSKE GLEDANOSTI Z UPORABO OGRODIJ ANGULARJS IN D3.JS

Diplomsko delo

Študent:	Alen Karahasanović
Študijski program:	Visokošolski študijski program Računalništvo in informatika
Smer:	Informatika
Mentor:	doc. dr. Borko Bošković
Somentor:	doc. dr. Boštjan Šumak
Lektor:	Tomaž Kurež, prof. slov.

Univerza v Mariboru

Fakulteta za elektrotehniko,
računalništvo in informatiko
Smetanova ulica 17
2000 Maribor, Slovenija

Številka: 93593253
Datum in kraj: 20. 04. 2016, Maribor

Na osnovi 330. člena Statuta Univerze v Mariboru (Ur. l. RS, št. 44/2015)
izdajam

SKLEP O DIPLOMSKEM DELU

1. **Alenu Karahasanoviću,** študentu visokošolskega strokovnega študijskega programa RAČUNALNIŠTVO IN INFORMATIKA, smer Informatika, se dovoljuje izdelati diplomsko delo.

2. **MENTOR:** **doc. dr. Borko Bošković**
   SOMENTOR: **doc. dr. Boštjan Šumak**

3. **Naslov diplomskega dela:**
   SPLETNA APLIKACIJA ZA ANALIZO TELEVIZIJSKE GLEDANOSTI Z UPORABO OGRODIJ ANGULARJS IN D3.JS

4. **Naslov diplomskega dela v angleškem jeziku:**
   WEB APPLICATION FOR THE ANALYSIS OF TELEVISION VIEWING USING ANGULARJS AND D3.JS FRAMEWORKS

5. Diplomsko delo je potrebno izdelati skladno z "Navodili za izdelavo diplomskega dela" in ga oddati v treh izvodih (dva trdo vezana izvoda in en v spiralo vezan izvod) ter en izvod elektronske verzije do 31. 08. 2016 v referatu za študentske zadeve.

Pravni pouk: Zoper ta sklep je možna pritožba na senat članice v roku 3 delovnih dni.

Dekan:
red. prof. dr. Borut Žalik

UNIVERZA V MARIBORU FAKULTETA ZA ELEKTROTEHNIKO, RAČUNALNIŠTVO IN INFORMATIKO 8

Obvestiti:
- kandidata,
- mentorja,
- somentorja,
- odložiti v arhiv.

ZAHVALA

Zahvaljujem se mentorju doc. dr. Borku Boškoviču in somentorju doc. dr. Boštjanu Šumaku za pomoč in vodenje pri pisanju diplomskega dela.

Za podporo bi se prav tako lepo zahvalil vsem najbližjim, posebno pa dekletu, ki mi je bila v podporo pri dokončanju študija.

SPLETNA APLIKACIJA ZA ANALIZO TELEVIZIJSKE GLEDANOSTI Z UPORABO OGRODIJ ANGULARJS IN D3.JS

Ključne besede: AngularJS, D3.js, analiza televizijske gledanosti

UDK: 621.397.13:005.52(043.2)

Povzetek

V diplomski nalogi smo implementirali prototipno aplikacijo za analizo televizijske gledanosti z uporabo ogrodij AngularJS in D3.js in tehnologij Node.js, Express.js ter MongoDB. V okviru tega smo preučili metode merjenja televizijske gledanosti in raziskali prednosti merjenja podatkov na podlagi tehnologije IPTV. Uporabljena ogrodja so nam omogočila hiter in učinkovit razvoj aplikacije, ki je uporabniku prijazna. Aplikacija omogoča uporabniški vmesnik za kreiranje poizvedb in prikazovanje podatkov gledanosti v obliki grafov in tabel.

WEB APPLICATION FOR THE ANALYSIS OF TELEVISION VIEWING USING ANGULARJS AND D3.JS FRAMEWORKS

Keywords: AngularJS, D3.js, Analysis of television viewing

UDK: 621.397.13:005.52(043.2)

Abstract

In our thesis we implemented a prototype application for the analysis of television viewing using AngularJS and D3.js frameworks and Node.js, Express.js and MongoDB technologies. In this context we studied the methods of television viewing measurement and explored the advantages of measuring data based on the IPTV technology. The used frameworks enabled us a fast and efficient development of a user friendly application. The application offers a user interface that enables the creation of queries and displaying viewing data in the form of charts and tables.

KAZALO

KAZALO SLIK

KAZALO PROGRAMSKE KODE

KAZALO TABEL

SEZNAM KRATIC

SPA - Single Page Application
API - Application Programming Interface
NoSQL – "non SQL" ali "non relational"
JSON – JavaScript Object Notation
BSON – Binary JSON
DOM – Document Object Model
SQL – Structured Query Language
MVC – Model View Controller
MVVC – Mode View-View Controller
HTML – Hyper Text Markup Language
CSS – Cascading Style Sheets
SVG – Scalable Vector Graphics
URL – Uniform Resource Locator
IPTV – Internet Protocol Television
VOD – Video on Demand
EPG – Electronic Program Guide

1 UVOD

Tehnologije za razvoj spletnih aplikacij so se v zadnjem desetletju drastično izboljšale. Hiter razvoj je seveda posledica vse večje uporabe interneta, ki je dandanes postal nekakšna razširitev človeške realnosti. Danes si sveta brez interneta skoraj ne znamo predstavljati, saj le-ta vsebuje zajetno količino informacij in služi kot vmesnik do vseh stvari, ki so nekdaj obstajale le v fizični realnosti. Spletne aplikacije vsakodnevno uporabljamo za komunikacijo, kupovanje, poslovanje itd. Da bi lahko to počeli hitro in enostavno, potrebujemo intuitivne uporabniške vmesnike, s katerimi so ta opravila brezhibna. Seveda so v ozadju kompleksni in obširni sistemi, ki znajo z uporabniškega vidika navidezno enostavne interakcije interpretirati ter pretvoriti v zapletene procese nad podatki, sistemi itd. Za razliko od statičnih in funkcionalno omejenih aplikacij v preteklosti smo z novimi tehnologijami prešli v dobo, kjer lahko spletne aplikacije razvijamo za vsakdanjega uporabnika. Danes morajo spletne aplikacije seveda podpirati delovanje na globalni ravni, z namenom, da jih uporablja veliko število ljudi.

Zbiranje in analiziranje podatkov v namen analiziranja obnašanja uporabe obstaja že pred samim obstojem interneta. Kar se je včasih počelo s telefonskimi raziskavami, se danes z lahkoto implementira, ko imamo natančne in velike količine podatkov, kot je razvidno pri aplikacijah. Iz omenjenega vzroka se je kot primer v televizijski industriji, ki je v tem smislu tehnološko nazadovala, pojavila potreba po modernih metodah merjenja gledanosti in obnašanja vsebin, tako linearne in nelinearne televizije kot tudi televizije na zahtevo.

V drugem poglavju pričujočega diplomskega dela bomo opisali zgodovino analize televizijske gledanosti, kjer bomo navedli metode merjenja gledanosti iz preteklosti, kot tudi tiste, ki so prisotne še danes. V nadaljevanju bomo obravnavali vprašanje, kako se je z razvojem tehnologije v televizijski industriji pojavila možnost uporabe sodobnejših metod merjenja, ki na internetu že vrsto let veljajo za standard.

V tretjem poglavju bomo navedli in preučili tehnologije, kot so ogrodja AngularJS in D3.js, ki se uporabljajo za razvoj sodobnih spletnih aplikacij. Prav tako bomo omenili še tehnologije, ki so potrebne za celovito rešitev posameznih spletnih aplikacij.

V četrtem poglavju bomo opisali in prikazali razvoj naše spletne aplikacije z uporabo preučenih ogrodij in tehnologij. Razvili smo spletno aplikacijo za analizo televizijske gledanosti po vzoru spletne aplikacije podjetja TVBeat. Aplikacija omogoča grajenje poizvedb na podatkih televizijske gledanosti in njihovo prikazovanje v obliki grafov in tabel. Prikazali bomo strukturo aplikacije, vrsto podatkov, ki se uporablja, prav tako tudi komponente, s katerimi je zgrajena aplikacija. Na podlagi primerov bomo opisali, kako so določene komponente implementirane z uporabljenimi ogrodji.

V petem poglavju bomo opravili preizkus, kjer bomo primerjali časovno zahtevnost funkcije za pripravo podatkov ter opazovali, kako se časovna zahtevnost spreminja glede na količino podatkov, ki jih vrnejo poizvedbe.

V končnem poglavju bomo podali ugotovitve pri preučevanju spletne aplikacije in navedli sklepe, ki nakazujejo, kakšne so prednosti ali slabosti pri uporabi izbranih tehnologij.

2 ANALIZA TELEVIZIJSKE GLEDANOSTI

Leta 1939 so v New Yorku začeli oddajati prvi redni televizijski program, svetovni vzpon pa je televizija doživela šele po letu 1945 [20]. Z nadaljnjim tehnološkim napredkom na tem področju je sledil razvoj barvne televizije, ki je v šestdesetih letih prejšnjega stoletja dokončno izpodrinila črno-belo, leta 1965 pa se je s prvim komercialnim satelitom, imenovanim Intelsat I, pričel še razvoj satelitske televizije [21].

Tako je televizija danes eno najbolj razširjenih medijskih občil v gospodarsko razvitih deželah sveta, ki v veliki meri vpliva na oblikovanje javnega mnenja in način življenja. To potrjuje tudi podatek, da je v letu 2013 že 79 % vseh gospodinjstev sveta imelo televizijski sprejemnik [22].

Z vse večjim naraščanjem števila televizijskih postaj in vse večjo konkurenco na tem področju se je pojavila tudi potreba po merjenju gledanosti televizije. Ta je kmalu postala ključno orodje za vse vidike televizijske industrije.

Danes informacije o številčnosti in strukturi gledalcev ter njihovem mnenju o televizijskem programu zanimajo predvsem programske oddelke televizijskih hiš (za učinkovito oblikovanje programske sheme), filmske producente (ugotovijo, kaj imajo gledalci radi in kaj ne; tako dobijo smernice za kreiranje prihodnje vsebine), kot tudi tržnike oglaševalskega časa pri televizijah (z visoko stopnjo gledanosti morajo prepričati oglaševalce in jim tako prodati oglaševalski čas), oglaševalce kot kupce oglaševalskega časa (v kolikšni meri so dosegli ciljno publiko) in nazadnje še oglaševalske agencije, medijske svetovalce in tržno-raziskovalne organizacije [23].

Z vse večjo potrebo po čim bolj natančnih meritvah gledanosti televizije so se spreminjale in izboljševale tudi metode merjenja. Ena izmed prvih metod je bila opravljena s pomočjo telefonskih raziskav, sledila ji je dnevniška metoda. Danes je najbolj razširjena elektronska metoda ali telemetrija, ki jo je že začela izpodrivati najnovejša metoda merjenja z tehnologijo IPTV.

2.1 Telefonska metoda merjenja televizijske gledanosti

Merjenje televizijske gledanosti s pomočjo telefonske metode je zelo enostavno, preprosto za izvedbo in cenovno ugodno. Omenjena metoda sicer zahteva urbano okolje z dobro razvitim telefonskim omrežjem, zaradi česar so izvzeti ruralni predeli z maloštevilčnim prebivalstvom.

Telefonska raziskava se izvaja tako, da se dnevno po telefonu anketira določen vzorec anketirancev. Vprašanja se nanašajo na njihovo gledanje televizije prejšnji dan, pri čemer se posamezen anketiranec zanaša na lasten spomin. Vzorec anketirancev je vsak dan sestavljen iz drugih oseb. Tako v daljšem časovnem obdobju pridemo do rezultata, ki je reprezentativen za celotno populacijo [24].

Navedena metoda ima to slabost, da se zanaša na spomin anketirancev, ki si večinoma zapomnijo bolj znane televizijske programe in pogosto predvajane oddaje v osrednjem času gledanosti. Na rezultat vpliva tudi kakovost vprašalnika. Takšne meritve so smiselne in učinkovite le pri omejeni programski shemi.

2.2 Dnevniška metoda merjenja televizijske gledanosti

V primerjavi s telefonsko metodo je dnevniška metoda natančnejša. Namenjena je merjenju televizijske gledanosti točno določenega števila programov.

Metoda se izvaja tako, da panelisti prejmejo dnevnik, v katerega vpisujejo podatke, kaj so gledali v časovnem obdobju enega tedna. Dnevnik že vsebuje spored televizijskih programov, ki je prilagojen različnim regijam, težava pa nastopi, če se spremeni televizijski program. Tudi ta metoda ima več slabosti, saj jo, prav tako kot telefonsko, spremlja faktor spomina in se zanaša na disciplino panelistov [25].

2.3 Elektronsko merjenje gledanosti televizije

Zaradi razvoja novih tehnologij, kot sta satelitska in kabelska televizija, posledično pa tudi zaradi večjega števila programov, ki se predvajajo 24 ur dnevno, se je pojavila potreba po natančnejših podatkih televizijske gledanosti. Prav tako se je razmahnil oglaševalski trg,

pri čemer je oglaševalce zanimala natančna statistika gledanosti njihovih komercialnih sporočil. Tako se je razvila elektronska metoda ali telemetrija (angl. *Peoplemeter*), ki je v primerjavi s predhodnimi metodami merjenja veliko bolj natančna.

Telemetrija deluje tako, da anketiranec sklene pogodbo o namestitvi merilne opreme, ki na osnovi avtomatiziranega zbiranja podatkov o gledanju televizije kvantificira in kvalificira gledanost televizijskih programov ter oglasnih blokov, pri tem pa je edina dejavnost panelista prijavljanje in odjavljanje [24].

Zgoraj navedeni način merjenja televizijske gledanosti nam da bolj objektivne podatke, saj se le-ti avtomatsko beležijo in se več ne zanašajo na subjektivni dejavnik spomina. Težava nastopi pri nedoslednosti prijavljanja in odjavljanja panelistov iz sistema, kar je še posebej značilno za otroke in starejše.

2.4 Tehnologija IPTV in prihodnost merjenja televizijske gledanosti

V zadnjem desetletju se je z razvojem in pokritostjo hitrejših internetnih povezav močno razširila tehnologija IPTV. Prav tako se je na razvitih trgih uveljavila uporaba digitalnih sprejemnikov (angl. *Set-top box*), ki omogočajo veliko število prednosti pri sprejemanju televizijskega signala, kot so večja kakovost slike, uporabniški vmesniki, nelinearna vsebina itd. Tehnologija IPTV je relativno nova in zahteva sodobnejšo infrastrukturo, zato so danes v določenih državah specifična slabše pokrita območja še vedno odvisna od kabelske, satelitske ali ponekod tudi terestrialne povezave (preko radijskih valov) za prenos televizijskega signala.

IPTV za razliko od kabelske in terestrialne tehnologije omogoča dvosmerno povezavo. To pomeni, da uporabnik ni omejen samo na sprejem signala, ampak lahko komunicira tudi s strežniki ponudnika (preko uporabniškega vmesnika). Ti omogočajo določene funkcionalnosti, kot so vsebine na zahtevo (angl. *Video on demand*), snemalnik, gledanje linearnih vsebin z zamikom (angl. *Time-shift*), pregled elektronskega programskega vodiča (angl. EPG - *Electronic program guide*) itd. Prav tako se vsi dogodki, ki jih uporabnik pošilja preko upravljalnika, lahko beležijo na strežnikih ponudnika in se uporabljajo za merjenje gledanosti.

Trenutna podjetja, kot so Nielsen in Kantar Media, ki dominirajo globalni trg merjenja gledanosti televizije, se še vedno v večini zanašajo na obstoječe načine elektronskega merjenja (t. i. *"peoplemeters")* z omejenim številom gospodinjstev. Le-ti zajemajo zelo majhne vzorce prebivalstva. V Sloveniji npr. podjetje Nielsen pokriva okoli 400 gospodinjstev in podatke le-teh usmerja na celotno populacijo Slovenije. Težava, ki se pojavi pri majhnih vzorcih, je dejstvo, da lahko še tako majhna sprememba v vedenju gledanosti povzroči nekonsistentne rezultate (to je najbolj opazno pri manj gledanih televizijskih kanalih). Prav tako ne pokrivajo nišnih kanalov, ki imajo relativno majhno gledanost. V primeru takšnega načina merjenja pa so to t. i. *"zero rating"* rezultati (ko ni podatkov ali pa so tako majhni, da projekcije ni mogoče izvajati).

Za tehnologijo IPTV se odpira možnost zajemanja velikega vzorca uporabnikov. Ta vzorec lahko zajema vse od 30 tisoč do več milijonov uporabnikov, odvisno od velikosti ponudnika storitev in države. Pri takšnih vzorcih govorimo o količini dogodkov nad 100 milijonov dnevno, kar pomeni, da preidemo v domeno t. i. velikih podatkov (angl. *Big data*). S takšno količino podatkov, upoštevajoč dejstvo, da se podatki beležijo v realnem času in z možnostjo merjenja alternativnih vsebin, so prednosti omenjenega načina merjenja naslednje:

- Vpogled v podatke v realnočasovnem urejanju programa v živo, kot so dnevno informativni programi, kjer lahko podaljšujemo, skrajšujemo in vrivamo prispevke med izvajanjem programa.
- Bolj podrobna, hitrejša in enostavna post-analiza gledanosti.
- Merjenje in vpogled v gledanost vsebin na zahtevo in gledanost z zamikom.
- Večja ločljivost podatkov za nišne (manj gledane) kanale.

Glede na infrastrukturo internetnih in televizijskih ponudnikov lahko razlikujemo način zbiranja podatkov ali dogodkov. Dogodke, ki so potrebni za merjenje televizijske gledanosti, lahko razvrstimo v naslednje postavke:

- Anonimen ID uporabnika – gre za sekljan (angl. *Hash*) identifikator, ki se uporablja zgolj za razločevanje posameznih uporabnikov ali sprejemnikov. Uporabnik je tako popolnoma anonimen na strani sistema za merjenje gledanosti.
- ID kanala ali vsebine na zahtevo.

- Mode - ta dogodek opisuje tri stanja: izklop ali vklop sprejemnika in preklop kanala.
- Časovna oznaka (angl. *Timestamp*) - označuje točen čas, kadar je uporabnik sprožil določeno akcijo.

Na podlagi navedenih podatkov lahko torej ugotovimo, na kak način uporabnik preklaplja med televizijskimi kanali, katere vsebine si je ogledal in koliko časa je to počel.

Takšne in podobne dogodke ovrednotimo ter obogatimo s pomočjo podatkov elektronskega programskega vodiča (angl. *Electronic program guide - EPG*). Iz podatkov, ki jih prikazuje EPG, tako dobimo metapodatke vsebin, kot so naziv, čas predvajanja, žanri itd. Prav tako lahko združujemo podatke gledanja z demografskimi podatki in na podlagi teh delamo bolj podrobne analize.

V sklopu obravnavane spletne aplikacije se bomo posluževali podatkov, ki so pridobljeni na osnovi metode merjenja na osnovi tehnologije IPTV.

3 TEHNOLOGIJE SPLETNE APLIKACIJE

V sklopu spletne aplikacije za analizo televizijske gledanosti smo uporabili vrsto tehnologij, ki temeljijo na programskem jeziku JavaScript in vsaka po sebi pokrivajo določen del celovite rešitve. Seveda je ob tem potrebno izpostaviti dejstvo, da, ko govorimo o celoviti rešitvi, govorimo o spletni aplikaciji, ki predstavlja sprednji del (angl. *Front-end*) celotne arhitekture sistema za analizo televizijske gledanosti. V ozadju oz. v zadnjem delu (angl. *Back-end*) celotnega sistema obstaja še kompleksen sistem zbiranja, formatiranja, mapiranja, procesiranja, shranjevanja in priprave podatkov.

V obravnavani aplikaciji se bomo osredotočili na naslednji tehnologiji:

- **AngularJS** - predstavlja glavni del spletne aplikacije in omogoča povezovanje s poslovno logiko, kot tudi prikazovanje in manipuliranje podatkov ter oblik v uporabniškem vmesniku.
- **D3.js** – omogoča grajenje najbolj pomembnega vizualnega dela aplikacije, kot so različni grafi. Le-ti manifestirajo podatke v obliko, ki je smiselna za končnega uporabnika.

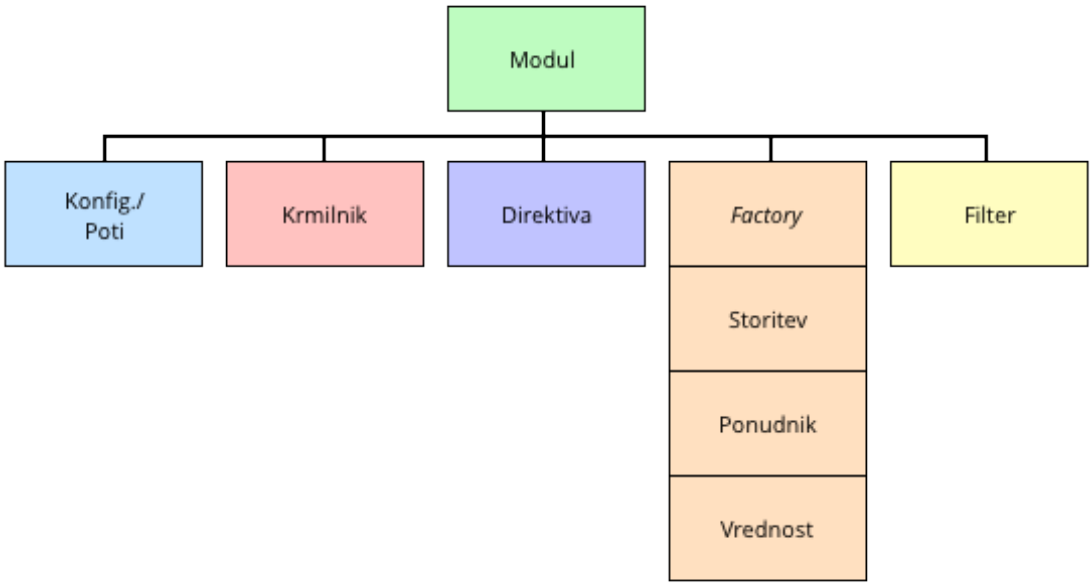
3.1 AngularJS

AngularJS je ogrodje za razvoj dinamičnih spletnih aplikacij, napisano v programskem jeziku JavaScript. Je odprtokodno ogrodje, v glavnem podprto in vzdrževano s strani podjetja Google in ostale skupnosti posameznikov, ki prispevajo tako k izboljšanju jedra kot tudi k razvoju knjižnic za dodatne funkcionalnosti.

Glavna ideja ogrodja AngularJS je omogočanje lažjega in hitrejšega razvoja enostranskih spletnih aplikacij (angl. *Single-page application - SPA*), in sicer tako, da ponudi višji nivo abstrakcije med razvijalcem in najbolj pogostimi nalogami, s katerimi se razvijalec ubada med razvojem spletnih aplikacij. Ogrodje AngularJS deluje v konceptu arhitekture MVC (angl. *Model view controller*), v smislu "*scope*" objekta pa je bolj podobna arhitekturi MVVM (angl. *Model-view-viewmodel*), kjer "*scope*" objekti predstavljajo "*pogled-model*" (angl. *ViewMode*l) [1].

Najbolj atraktivna in zanimiva funkcionalnost ogrodja AngularJS je, da omogoča razširitev sintakse HTML, in sicer tako, da same elemente HTML označimo s posebnimi oznakami ali direktivami. Z omenjenimi direktivami lahko elemente HTML definiramo kot nove prirejene vtičnike ali pa jim dodamo specifična vedenja in možnost manipuliranja z elementi DOM (angl. *Document object model*).

Komponente in funkcionalnosti ogrodja AngularJS. so razvidne iz slike 3.1:

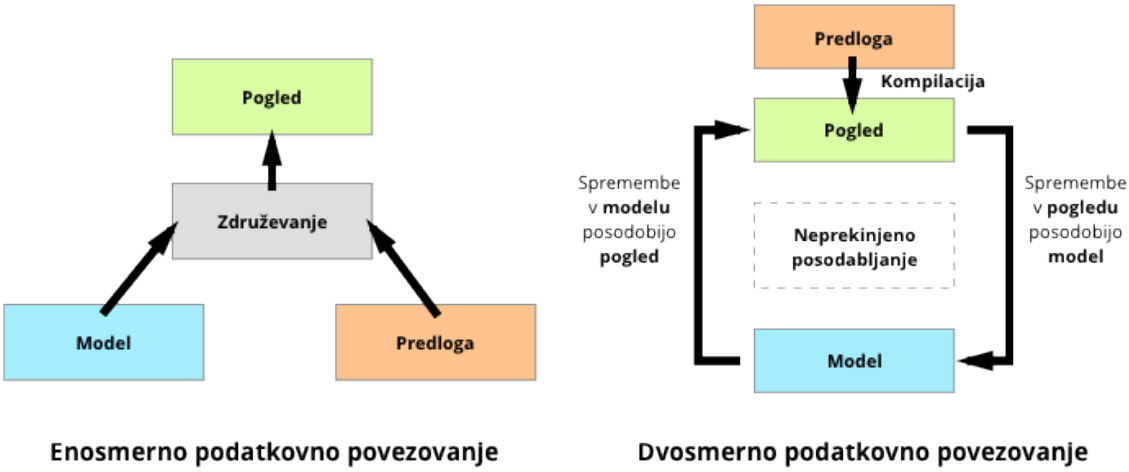


Slika 3.1: AngularJS komponente [2].

3.1.1 Podatkovno povezovanje

Običajno podatkovno povezovanje (angl. *Data-binding*) v ostalih sistemih predlog poteka enosmerno, kar pomeni, da se podatki iz modela na podlagi predloge združijo v poglede. Pomanjkljivost takšnega povezovanja je, da se po tej združitvi nadaljnje spremembe, ki se zgodijo na modelu, ne posodobijo samodejno v pogledu. Enako velja v primeru, ko uporabnik opravlja spremembe v pogledu, saj se le-te ne posodobijo v modulu. V tem primeru je razvijalec primoran napisati posebno logiko za manipuliranje elementov DOM in atributov, da omogoči sinhronizacijo med modelom in pogledom [3]. Ta pristop lahko postane zelo naporen v smislu razvoja in vzdrževanja kode.

Prednost ogrodja AngularJS je v tem, da implementira dvosmerno podatkovno povezovanje, kjer sta model in pogled v sinhronizaciji. Ko pride do spremembe v modelu, se primerno posodobi pogled in obratno. Na sliki 3.2 je razvidna razlika med opisanimi podatkovnimi povezovanji.



Slika 3.2: Potek enosmernega in dvosmernega povezovanja [6].

3.1.2 Moduli

Moduli predstavljajo posamezne enote, znotraj katerih združimo določeno funkcionalnost aplikacije. Modul združuje različne dele aplikacije, kot so krmilniki, storitve, filtri, direktive itd. Aplikacija lahko združuje tudi več modulov, od katerih lahko vsak vsebuje specifično kodo za določeno funkcionalnost aplikacije. Prednosti uporabe modulov se kažejo v dejstvu, da omogočajo lažje pisanje testov, uporaba modulov pa zaživi tudi med aplikacijami in ob nalaganju različnih predelov kode v kakršnem koli zaporedju [4].

Modul deklariramo s posebno metodo "*.module()*", prikazano v programski kodi 3.1. Metodi moramo podati dva parametra. Prvi parameter je ime ustvarjenega modula, drugi parameter pa vsebuje seznam odvisnosti (to so lahko npr. drugi moduli, od katerih je ta modul odvisen).

```
angular.module('myModule',  []);
```

Programska koda 3.1: Deklaracija modula.

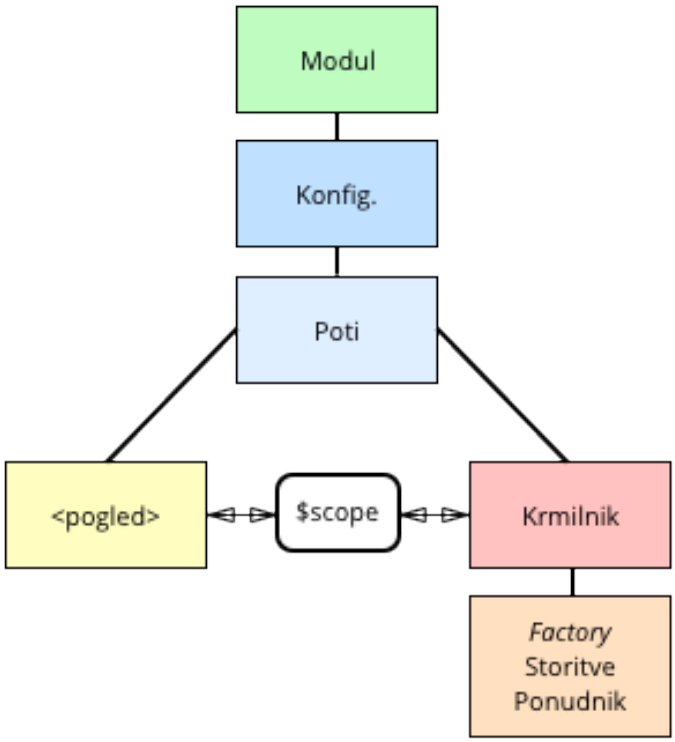
Ko smo modul deklarirali, ga uporabimo v predlogi z direktivo "*ng-app*" prav tako pa moramo skripto, ki vsebuje deklaracijo modula, vključiti v predlogo (programska koda 3.2).

```
<div ng-app="myModule">
  ...
</div>
```

Programska koda 3.2: Definicija modula v direktivi.

3.1.3 Scope

Scope je vezni element ali "lepilo", ki povezuje poglede in krmilnik. Pogled in krmilnik med sabo ne komunicirata direktno, ampak preko objekta, na katerega se sklicujeta. Navedeno je razvidno iz slike 3.3, ki prikazuje arhitekturo delovanja ogrodja AngularJS.



Slika 3.3: Arhitektura delovanja ogrodja AngularJS [2].

Scope objekti se ob začetku aplikacije povežejo na elemente DOM in na direktive, določene v predlogah. Tako poznamo "*$rootScope*" objekt, ki je oče vseh "*$scope*" objektov in se veže na začetni element DOM aplikacije, "*$scope*" objekti posameznih krmilnikov pa na primerne otroke v drevesu DOM. Na koncu dobimo drevo scope objektov, ki predstavljajo samo drevo elementov DOM [7].

Objekt "*$scope*" predstavlja trenutno stanje podatkov in funkcij med predlogo ter krmilnikom. V krmilniku ga podamo kot argument, nato pa je le-ta dosegljiv samo znotraj tega krmilnika. Iz programske kode 3.3 je razvidno, kako določamo vrednost isto imenovane spremenljivke v dveh različnih krmilnikih. Prav tako je razvidno, da "*$scope*" objekt poskrbi, da se izpiše prava vrednost znotraj definiranega prostora določenega krmilnika.

```
<div ng-app="myModule">
  <div ng-controller="firstController">
    {{ message }}
  </div>
  <div ng-controller="secondController">
    {{ message }}
  </div>
</div>
...
<script>
  var module = angular.module('myModule', []);
  module.controller('firstController', function($scope) {
    $scope.message = "Apples";
  });
  module.controller('secondController', function($scope) {
    $scope.message = "Oranges";
  });
</script>
```

Programska koda 3.3: Prikaz "*$scope*" objekta v dveh različnih krmilnikih.

3.1.4 Krmilniki

Krmilniki v ogrodju AngularJS so posredniki med pogledi in modeli. Služijo obogatenju pogledov oz. natančneje - scope pogledu aplikacije. Krmilnik uporabimo za nastavljanje začetnega stanja in dodajanje obnašanja objektu "*$scope*".

Krmilnike določimo tako, da elemente v predlogah označimo z direktivo "*ng-controller*", nakar moramo imeti še primerno konstruktorsko funkcijo, kjer določimo začetno stanje in specifično obnašanje. V fazi, ko se krmilnik pripne na drevo DOM z ustrezno direktivo, se ustvari "*$scope*" objekt, ki ga tudi podamo v argumentu konstrukcijske funkcije. V programski kodi 3.4 je prikazan primer, kako definiramo enostavni krmilnik.

```
...
<div ng-app="myModule">
  <div ng-controller="myController">
    ...
  </div>
</div>
...
<script>
  var module = angular.module('myModule', []);
  module.controller('myController', function($scope) {
    ...
  });
</script>
```

Programska koda 3.4: Definicija krmilnika.

Krmilnike lahko tudi gnezdimo in s tem omogočimo, da se objekti "*$scope*" med očetom in otrokom dedujejo. V pogledih lahko omogočimo, da krmilnike prikličemo ob posebnih akcijah, kot npr. klik na gumb, in sicer tako, da uporabimo direktivo "*ng-click*" in v njej navedemo klic funkcije, ki je definirana znotraj krmilnika [8]. V programski kodi 3.5 je razvidna uporaba gnezdenih krmilnikov in klic funkcije. Prvo definiramo kot objekt "*user*" v očetu, nato pa v otroku definiramo funkcijo, ki sproži dodajanje priimka objektu "*user*".

```
<script>
        var app = angular.module('myModule',[]);
        app.controller('parentController', function($scope) {
          $scope.user = { name: "John" }
        });
        app.controller('childController', function($scope) {
          $scope.getLastName = function() {
                $scope.user.lastname = "Doe";
          }
        });
</script>

<div ng-app="myModule">
  <div ng-controller="parentController">
    <div ng-controller="childController">
      <button ng-click="getLastName()">Click to add last name</button>
    </div>
    {{ user.name }} {{ user.lastname}}
  </div>
</div>
```

Programska koda 3.5: Gnezdeni krmilniki in klic funkcije v krmilniku.

3.1.5 Izrazi

Z izrazi lahko uporabimo vrednosti spremenljivk znotraj določenega dosega (angl. *Scope*), in sicer v odvisnosti, kje so definirane v predlogi. V izrazu se lahko nahajajo števila, nizi, vrednosti spremenljivk ali pa rezultat na podlagi matematične operacije le-teh. Zapisujemo jih z dvojnimi zavitimi oklepaji {{ izraz }} [9].

Kot je razvidno iz programske kode 3.6, lahko izraze uporabljamo med elementi, in sicer v atributih, ali pa tudi v direktivah.

```
<div title="{{ title }}"
     ng-class="{{ active }}">{{ content }}</div>
<span>Value: {{ 2 + 3 }}</span>
```

Programska koda 3.6: Primer različnih izrazov.

3.1.6 Filtri

Kadar želimo podatke, ki jih prikazujemo, formatirati, uporabljamo filtre. Filtre lahko uporabljamo skupaj z izrazi, in sicer tako, da jih ločimo z navpičnico {{ izraz | filter }} (programska koda 3.7).

```
<!-- Izpisal se bo naslov z veliki črkami -->
<div>{{ title | uppercase }}</div>
```

Programska koda 3.7: Uporaba filtra v izrazu.

Filtre lahko v kodi uporabimo tudi takrat, ko želimo manipulirati s podatki ali z vrednostmi objektov. Takrat uporabimo "*$filter*" storitev, ki jo moramo primerno podati v argumentu konstrukcijske funkcije [10]. Iz programske kode 3.8 je razvidno, kako znotraj krmilnika uporabimo filter, da uredimo zbirko imen.

```
var app = angular.module('myModule',[]);
app.controller('myController', ['$scope', '$filter',
  function($scope, $filter) {
    $scope.names = $filter('orderBy')(['David', 'Alan', 'Bill',
'John']);
  }
]);
```

Programska koda 3.8: Uporaba filtra v krmilniku.

Filtre lahko zgradimo sami ali pa uporabimo že vgrajene filtre, ki omogočajo:

- Formatiranje oblike datuma in vseh časovnih enot, ki jih ta vsebuje.
- Filtriranje zbirke elementov.
- Pretvarjanje števil v nize.
- Urejanje zbirke elementov itd.

3.1.7 Direktive

Direktive so ključni gradniki ogrodja AngularJS in omogočajo razširitev statičnih elementov HTML s posebnimi oznakami, ki elementom dodajo določeno funkcionalnost. Oznake direktiv so lahko definirane kot imena elementov, atributi, razredi CSS ali komentarji v predlogi. Ogrodje AngularJS ima vrsto vgrajenih direktiv, te pa so zapisane tako, da so predznačene s predpono "*ng-"* ali "*data-ng-"* [11].

V programski kodi 3.9 lahko vidimo primer direktiv, ki so zapisane kot atributi elementov HTML.

```
<div ng-app="myModule">...</div>
...
<div ng-controller="myController">...</div>
...
<input type="input" ng-model="name">
...
<button ng-click="saveName()">...</button>
```

Programska koda 3.9: Primeri direktiv v predlogi.

Vgrajenih direktiv je nemalo, skozi potek diplomskega dela pa se bomo osredotočili in na kratko opisali nekaj najbolj koristnih in uporabljanih, in sicer:

- ng-app,
- ng-model,
- ng-bind,
- ng-click in
- ng-repeat.

Direktiva **ng-app** določa korenski element aplikacije in omogoča, da se aplikacija avtomatično inicializira, kadar se naloži spletna stran. Z "*ng-app*" običajno označimo nek začetni element HTML, kot je "*<body>"* ali *"<html>"* [11].

Direktiva **Ng-model** poveže vrednost kontrol HTML, kot so vnosna polja in izbirni seznami ("*<input>",* "<*select*>", "<*textarea*>") s podatki. Iz programske kode 3.10 je razviden primer, kjer je vnosno polje povezano z objektom, ki je definiran v krmilniku. Vrednost se

bo torej prikazovala v vnosnem polju, prav tako tudi na mestu izraza *"{{ name }}*". Če spremenimo vrednost v vnosnem polju, se bo prav tako primerno posodobila vrednost na mestu izraza. Na tem mestu je razvidno delovanje dvosmernega povezovanja.

```
<script>
var app = angular.module('myModule',[]);
app.controller('myController', function($scope) {
   $scope.name = "John Doe";
});
</script>
<div ng-app="myModule" ng-controller="myController">
  <input type="input" ng-model="name">
  {{ name }}
</div>
```

Programska koda 3.10: Primer uporabe direktive "ng-model".

Direktiva **Ng-bind** poveže element HTML na objekt na enak način kot pri uporabi izraza. `<div ng-bind="name"></div>`. Vrednost se veže na element "*innerHTML*".

Direktiva **Ng-click** omogoča določitev posebnega obnašanja na pritisk elementa, na katerem je direktiva, npr. klic funkcije ali pa izraz. `<button ng-click="saveName()">...</button>.`

Direktiva **Ng-repeat** omogoča izpisovanje elementov glede na niz vrednosti. V programski kodi 3.11 lahko vidimo primer, kjer izpišemo seznam imen v posamezne "*
*" elemente HTML.

```
<ul ng-repeat="myModule">
  <li ng-repeat="name in ['Alan', John, 'David']">
    {{ name }}
  </li>
</ul>
```

Programska koda 3.11: Primer izpisa seznama z direktivo "ng-repeat".

Kadar želimo posebne direktive, lahko le-te napišemo v kodi JavaScript. Deklariramo jih lahko s konstrukcijsko funkcijo "*directive()*", ki jo, podobno kot krmilnik, definiramo na določenem modulu. Iz programske kode 3.12 je razviden primer deklaracije direktive v jeziku JavaScript, kot tudi primer, kako lahko direktivo v pogledu definiramo kot element HTML.

```
<script>
var app = angular.module('myModule',[]);
app.directive('customDirective', function() {
  return {
    template: "<h1>Title</h1>"
  };
});
</script>

<div ng-app="myModule">
  <customDirective></customDirective>
</div>
```

Programska koda 3.12: Primer deklaracije posebne direktive.

3.1.8 Storitve

Ker so krmilniki omejeni na delovanje znotraj svojega dosega, potrebujemo še metode, ki bodo podatke obdržale čez celoten čas delovanja aplikacije. Te metode imenujemo storitve, uporabljamo jih za hranjenje podatkov in poslovno logiko, ki jo lahko uporabljamo kjerkoli v aplikaciji. Krmilniki lahko uporabljajo podatke in funkcije storitev tako, da ob deklaraciji krmilnika kot argument podamo ime storitve.

Ogrodje AngularJS vsebuje vrsto že vgrajenih storitev, kot so "*$http*", "*$location*", "*$route*" itd. Vsaka omogoča določeno funkcionalnost. Npr. "*$http*" omogoča dostop do brskalnikovega "*XMLHttpRequest*" objekta. Namesto da bi pisali vso potrebno logiko za interakcijo s tem objektom, lahko to počnemo preko storitve "*$http*". Omenjena storitev je najbolj pogosta, uporabljamo jo npr. kadar želimo zahtevati ali pošiljati podatke na strežnik.

Kadar potrebujemo svojo posebno storitev, jo lahko tudi deklariramo z metodo "*factory*". Obstajajo še druge metode, kot so "*service*" in "*provider*", ki služijo istemu namenu, razlikujejo se zgolj v načinu definicije objektov.

Iz programske kode 3.13 je razviden način, kako definiramo novo metodo "*factory*", ki vsebuje funkcijo za množenje. V krmilniku v argumentu podamo ukaz "*factory*" in kličemo funkcijo za množenje.

```
var app = angular.module('myModule', []);

app.factory('MathService', function() {
  var factory = {};

  factory.multiply = function(a, b) {
    return a * b;
  }
  return factory;
})

app.controller('myController', function($scope, MathService) {
  $scope.sum = MathService.multiply($scope.a, $scope.b);
});
```

Programska koda 3.13: Primer storitve [12].

3.1.9 Usmerjanje

Usmerjanje (angl. *Routing*) potrebujemo takrat, kadar želimo za različne vsebine imeti poseben naslov URL. To omogoča, da imajo različni pogledi ali predloge svoj naslov, s tem pa dosežemo lažjo navigacijo po aplikaciji. Tako lahko glede na naslov URL določimo, katera predloga se bo naložila in kateri krmilnik bo zadolžen za inicializacijo tega pogleda [16].

Modul za usmerjanje moramo naložiti kot dodatno skripto v "head" sekciji glavnega dokumenta HTML. Tega tudi dodamo v deklariran modul aplikacije, in sicer kot izjemo "*ngRoute*". Nato kličemo "*config()*" metodo, v kateri ustvarimo objekt "*$routeProvider*" in

metodo. Z metodama "*when()*" in "*otherwise()*" določimo, na katerih naslovih se bosta naložila predloga in krmilnik. V programski kodi 3.14 vidimo primer usmerjanja v ogrodju AngularJS.

```
var app = angular.module('myModule', ['ngRoute']);
app.config(['$routeProvider', function($routeProvider){
  $routeProvider
    .when('/', {
      templateUrl: 'views/home.html',
      controller: 'HomeController'
    })
    .when('/login', {
      templateUrl: 'views/login.html',
      controller: 'LoginController'
    })
    .when('/about, {
      templateUrl: 'views/about.html',
      controller: 'AboutController'
    })
    ...
}]);
```

Programska koda 3.14: Usmerjanje v ogrodju AngularJS.

3.2 D3.js

D3.js je knjižnica jezika JavaScript, ki omogoča manipulacijo podatkov in prikazovanje le-teh s pomočjo že vgrajenih tehnologij v brskalniku, kot so HTML, SVG in CSS. Omogoča napredno in prilagodljivo vizualizacijo podatkov v kakršnikoli obliki, ki jo lahko prikažejo sodobni brskalniki [18].

D3.js je odprtokodna knjižnica, ki je napisal Mike Bostock v času svojega doktorskega študija na Stanfordu. Prav tako je projekt sponzoriran s strani revije New York Times, kjer je v mnogih člankih knjižnica uporabljena tudi za vizualizacijo zanimivih statističnih analiz [19].

Omenjena odprtokodna knjižnica se osredotoča na povezovanje podatkov na elemente drevesa DOM in je zaradi svoje zasnove zelo fleksibilna pri dodajanju novih funkcij ter pri ponovni uporabi kode. Zaradi svoje fleksibilnosti se je razvila tudi skupnost, kjer posamezniki razvijajo bolj specifične komponente in vtičnike na osnovi ogrodja D3.js. Najbolj pogosta uporaba je posebna grafična vizualizacija velike količine podatkov.

Izpostavimo še nekatere njene najpogostejše funkcionalnosti:

- **Selekcije** (angl. *Selections*) – izbiranje posameznih skupin ali elementov in prirejanje različnih lastnosti le-teh.
- **Dinamične lastnosti** (angl. *Dynamic properties*) – stile, atribute in ostale lastnosti lahko definiramo kot funkcije podatkov, kar omogoča, da prirejene vrednosti niso le konstante.
- **Vstopne in Izhodne selekcije** – lahko kontrolirajo dodajanje in brisanje elementov glede na količino dodanih podatkov, ko ti niso enaki količini izbranih elementov.
- **Tranzicije** – uporabljajo se za animirano spreminjanje elementov. To je uporabno, ko se posodabljajo podatki in takrat, kadar želimo posodobiti npr. graf, kjer se podoba vizualnih elementov prilagodi le-tem podatkom. To želimo narediti na vizualno bolj privlačen in intuitiven način.

3.2.1 Dimple.js

Obstaja vrsta knjižnic za prikazovanje grafov, pri katerih smo omejeni na funkcionalnost, ki jo že podpirajo, spreminjanje ali dodajanje novih funkcij pa je zelo naporno in potratno.

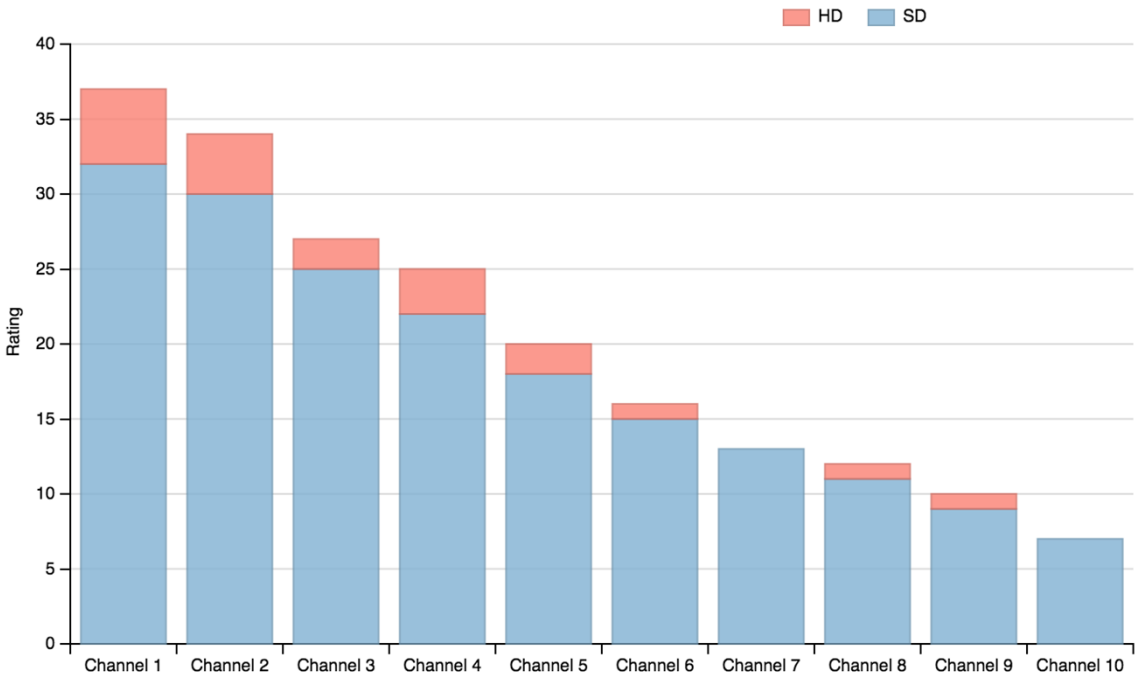
V obravnavani aplikaciji bomo za vizualizacijo podatkov v obliki različnih grafov uporabili knjižnico Dimple.js, ki je zgrajena na osnovi ogrodja D3.js in še vedno omogoča fleksibilnost in moč, ki jo vsebuje ogrodje D3.js. Vsebuje že definirane funkcije za enostavnejše in hitrejše kreiranje najbolj pogostih grafov. V sledečem poglavju bomo bolj podrobno opisali, kako uporabimo knjižnico Dimple.js, in sicer v povezavi z ogrodjem AngularJS.

Za primer si v spodaj prikazani programski kodi 3.15 poglejmo, kako enostavno in hitro lahko ustvarimo palični graf iz podatkov datoteke JSON. Kot je razvidno iz kode, lahko, ko dobimo podatke, ustvarimo nov objekt z metodo "*dimple.chart()*", nakar lahko nastavljamo vse od velikosti grafa do kategorij, x/y osi, legend itd.

```
var svg = dimple.newSvg("#chartContainer", 700, 450);
d3.json("/data/video_resolution_by_channels.json", function (data) {
  var myChart = new dimple.chart(svg, data);
  myChart.setBounds(60, 30, 620, 350)
  myChart.addCategoryAxis("x", ["Channel"]);
  myChart.addMeasureAxis("y", "Rating");
  myChart.addSeries("Video Resolution", dimple.plot.bar);
  myChart.addLegend(65, 10, 510, 20, "right");
  myChart.draw();
});
```

Programska koda 3.15: Ustvarjanje grafa s knjižnico Dimple.js.

V primeru programske kode 3.15 tako ustvarimo graf, ki prikazuje 10 kanalov, razdeljenih po video resoluciji (SD in HD). Vrednost Y osi predstavlja metrika "*Rating*". Rezultat je prikazan na sliki 3.4.



Slika 3.4: Primer grafa knjižnice Dimple.js.

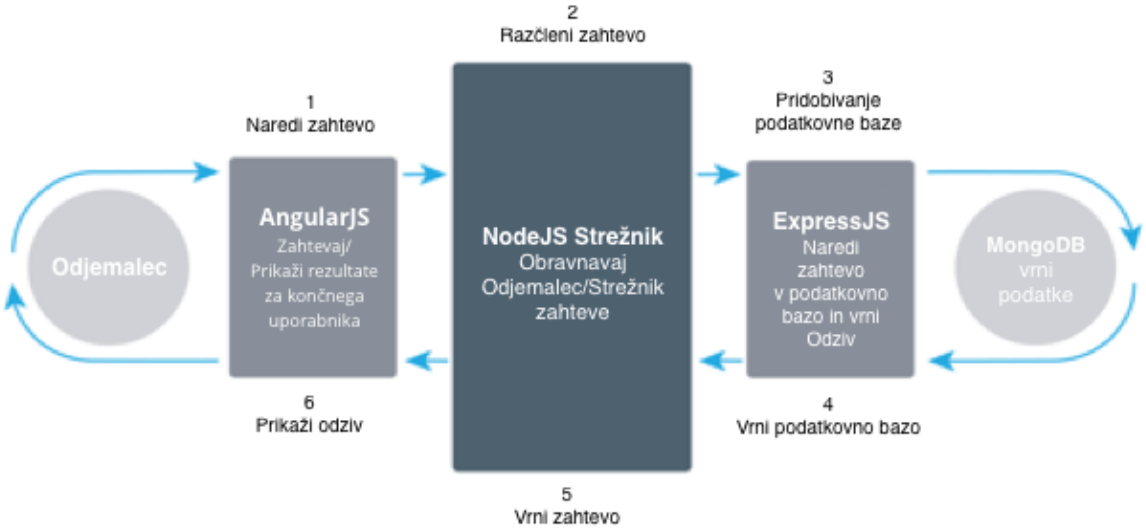
3.3 Ostale tehnologije

Kot smo že omenili, je "*client-side*" ogrodje AngularJS tisto ogrodje, ki ponuja večino rešitev, ki jih potrebujemo za SPA (angl. *Single-page application*). Kadar se kompleksnost aplikacije povečuje, je pomembno, da je začetna arhitektura aplikacije in tehnologij (uporabljenih v aplikaciji) zgrajena tako, da omogoča neboleče skaliranje funkcij aplikacije.

V tem smislu je danes postala zelo popularna kombinacija ogrodij ali tehnologij, ki se kot osnova uporablja pri gradnji modernih spletnih aplikacij, imenovana "***MEAN stack***" [13]. To ogrodje vsebuje 4 osnovne tehnologije, ki pokrivajo celoten spekter spletne aplikacije, in sicer od podatkov do samega prikazovanja vsebine:

- **M**ongoDB
- **E**xpress.js
- **A**ngularJS
- **N**ode.js

Na sliki 3.5 nazorno prikažemo, kako so posamezne tehnologije razdeljene v celotni arhitekturi spletne aplikacije.



Slika 3.5: "MEAN stack" diagram delovanja [5].

Spletno aplikacije delimo na **strežniški del** (angl. *Server-side*) in **odjemalčev del** (angl. *Client-side*). Na strežniškem delu se nahaja poslovna in podatkovna logika, ki komunicira z zunanjim vmesnikom (preko katerega dobivamo glavne podatke), čisti in formatira podatke, skrbi za uporabniško upravljanje ter opravlja še vrsto ostalih specifičnih nalog. Na odjemalčevem delu se izvaja prikazovanje dobljenih podatkov in komunikacija z uporabnikom.

3.3.1 Node.js

Node.js je odprtokodno JavaScript izvajalno okolje, zgrajeno na osnovi Chrome V8 JavaScript pogona. Služi kot ogrodje za razvoj spletnih aplikacijskih strežnikov v jeziku JavaScript. Zaradi svoje dogodkovno vodene in asinhrone I/O arhitekture omogoča razvoj zelo hitrih in skalabilnih spletnih strežnikov. Le-ti omogočajo delovanje spletnih aplikacij, ki delujejo v realnem času in obdelujejo ogromno količino podatkov. Osnovni moduli ogrodja Node.js so v veliki večini napisani v jeziku JavaScript, kot dodatek pa prednaloženi upravljalec paketov "*npm*" omogoča dostop do velikega števila programov Node.js tretjih oseb, ki služijo kot dodatki na že obstoječe module [14].

Upravljalec paketov "*npm*" omogoča razvijalcu, da brska skozi spletna skladišča knjižnic ali modulov, ter skozi ukazno vrstico namesti le-te.

```
npm install <ime_paketa>
```

Ta služi kot osnovno orodje pri začetnem grajenju aplikacije, kot tudi pri kasnejšem posodabljanju obstoječih knjižnic in dodajanju novih. S knjižnicami si lahko bistveno olajšamo razvoj kompleksnih aplikacij in pohitrimo razvoj, in sicer tako, da se osredotočimo na razvoj specifičnih funkcionalnosti aplikacije.

3.3.2 Express.js

Express.js je spletno aplikacijsko ogrodje za okolje Node.js, ki omogoča funkcionalnosti grajenja spletnih aplikacij in vmesnikov. Namesti se kot "*npm*" modul in razširi ali doda funkcionalnosti k okolju Node.js. V sklopu obravnavane aplikacije ga uporabljamo za poganjanje aplikacije in komunikacijo z vmesnikom zunanjega analitičnega strežnika, ki vsebuje že interpretirane podatke o gledanosti glede na podane parametre. V osnovi ogrodje Express.js uporabljamo za komunikacijo z vmesniki in podatkovno bazo ter izvrševanje poslovne logike nad podatki.

V programski kodi 3.16 navajamo primer, v katerem z ogrodjem Express.js napišemo programski vmesnik (angl. *Middleware*), s katerim lahko upravljamo z zahtevami (angl. *Request*) in odzivi (angl. *Response*). V spodnjem primeru programskega vmesnika pošljemo glavo zahtevi in vsebino "*Hello world!*", ki se izpiše na spletni strani [15].

```
var express = require("express");
var http = require("http");
var app = express();

// programski vmesnik
app.use(function(request, response) {
  response.writeHead(200, { "Content-Type": "text/plain" });
  response.end("Hello world!\n");
});
http.createServer(app).listen(3000);
```

Programska koda 3.16: Primer kreiranja strežnika in uporabe programskega vmesnika z ogrodjem Express.js.

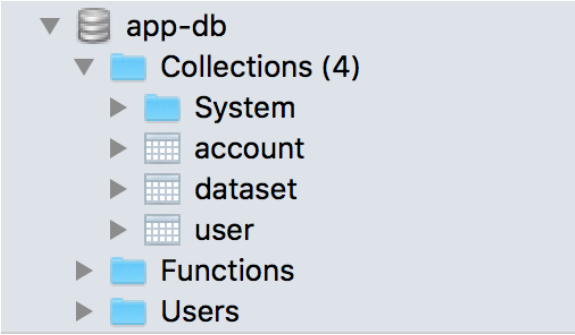
Programske vmesnike lahko uporabljamo za usmerjanje aplikacij glede na naslov URL ali pa za komunikacijo med odjemalcem in strežnikom.

3.3.3 MongoDB

Čeprav analitične podatke dobivamo iz zunanjega vmesnika, ki je ločen od obravnavane spletne aplikacije, potrebujemo podatkovno bazo, kjer bomo shranjevali informacije in nastavitve uporabniških računov in aplikacije. V ta namen potrebujemo enostavno in dovolj učinkovito podatkovno bazo.

MongoDB je dokumentno orientirana, odprtokodna podatkovna baza. Gre za vrsto podatkovne baze NoSQL, ki ni zasnovana na osnovi tabel in relacij, kot velja za tradicionalne in najbolj običajno uporabljene, relacijske baze. Struktura podatkovne baze MongoDB temelji na zbirkah in dokumentih v formatu BSON (binarni JSON) z dinamičnimi shemami. Prednost podatkovne baze MongoDB je, kot v mnogih podatkovnih bazah NoSQL, hitrost, skalabilnost in dinamičnost. Zaradi enostavne podatkovne strukture in načina poizvedovanja je zelo popularna izbira med razvijalci sodobnih spletnih aplikacij [17].

V podatkovni bazi MongoDB ustvarjamo zbirke, v katere napolnimo dokumente v obliki formata JSON. Za primer navedimo osnovno bazo, ki je prikazana na sliki 3.6, kjer najdemo tri zbirke, v katerih se nahajajo dokumenti.



Slika 3.6: Primer zbirk v podatkovni bazi MongoDB.

Če pogledamo v zbirko "*user*", kot prikazuje programska koda 3.17, zasledimo dokumente, ki predstavljajo posamezen vpis za vsakega uporabnika v zapisu JSON.

```
{
    "_id" : ObjectId("57ae24441c1fe2ceee17b23a"),
    "name": "John",
    "lastname": "Doe",
    "city": "New York",
    "addresses": [
        {"street": "125 Wilson St"},
        {"street": "21 Something St"}
    ]
}
...
```

Programska koda 3.17: Primer dokumenta v podatkovni bazi MongoDB.

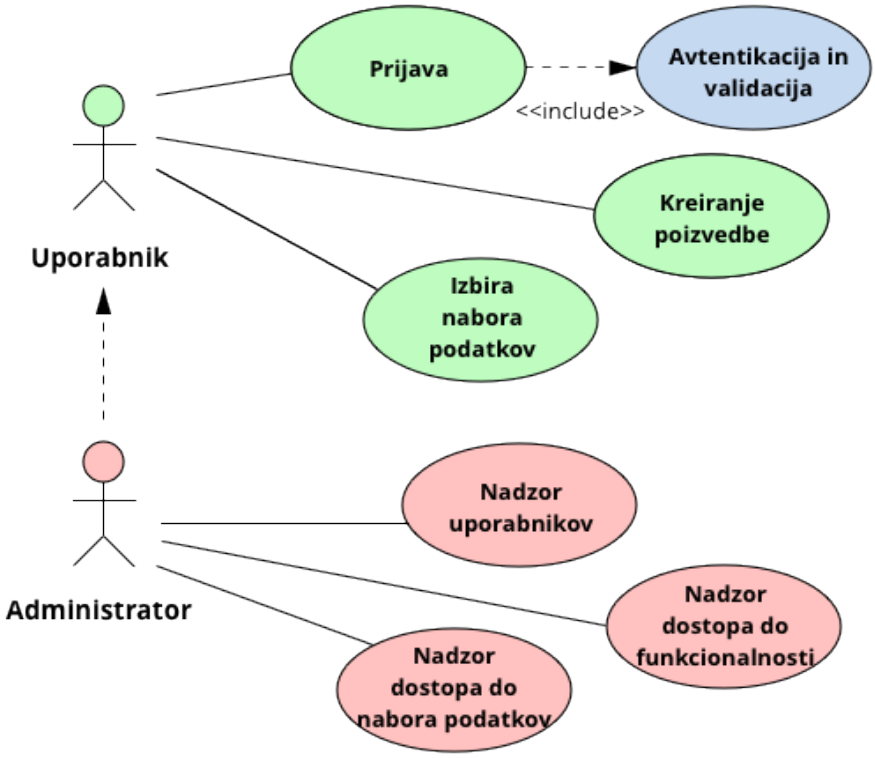
4 NAČRTOVANJE IN RAZVOJ SPLETNE APLIKACIJE ZA ANALIZO TELEVIZIJSKE GLEDANOSTI

V pričujočem poglavju bomo prikazali razvoj in načrtovanje spletne aplikacije za analizo televizijske gledanosti, ki smo jo razvili za potrebe diplomskega dela. Aplikacijo smo razvili po vzoru spletne aplikacije podjetja TVBeat. Zgrajena je na osnovi ogrodij Node.js, AngularJS in D3.js.

Namen spletne aplikacije za analizo televizijske gledanosti je s sodobnimi spletnimi tehnologijami, kot sta ogrodji AngularJS in D3.js, uporabniku omogočiti enostavno in močno orodje, s katerim lahko analizira veliko količino podatkov gledanosti. Spletna aplikacija služi kot prezentacijski nivo v celotnem ekosistemu podatkov o televizijski gledanosti. Obstoječi standardi industrije zahtevajo vrsto orodij ali funkcionalnosti za popoln vpogled v podatke televizijske gledanosti. Glede na vejo industrije (oglaševalci, televizijske hiše, ponudniki TV-storitev) se te funkcionalnosti medsebojno razlikujejo.

V spletni aplikaciji, ki smo jo razvili, omogočimo prijavljenemu uporabniku kreiranje poizvedb nad podatki gledanosti. Uporabnik v pogledu Analyzer z različnimi kontrolami sestavi poizvedbo tako da določi časovno obdobje, primerjalno dimenzijo in filtre. Uporabniki lahko pregledujejo gledanost iz različnih naborov podatkov (angl. *Dataset*). Podatki gledanosti in vsebin (za izbran "dataset"), katere bomo uporabljali v naši spletni aplikaciji, se pridobivajo iz zunanjega analitičnega strežnika podjetja TVBeat. Analitični strežnik hrani testne podatke in jih glede na poizvedbo spletne aplikacije primerno sestavi in pošlje nazaj. Ko spletna aplikacija prejme podatke, jih pripravi in prikaže uporabniku v obliki grafa in tabele. Prikaz teh podatkov lahko uporabnik prav tako spreminja tako, da še določi tip grafa in ureja podatkovne točke po izbrani metriki. Podatki uporabnikov so shranjeni v podatkovni bazi MongoDB, te pa uporabljamo pri avtentikaciji in avtorizaciji. Aplikacija prav tako omogoča, da administrator vrši nadzor nad uporabniki, uporabniškimi računi in nastavitvami dostopa do funkcionalnosti aplikacije za uporabnika.

Na sliki 4.1 je razviden diagram primerov uporabe:



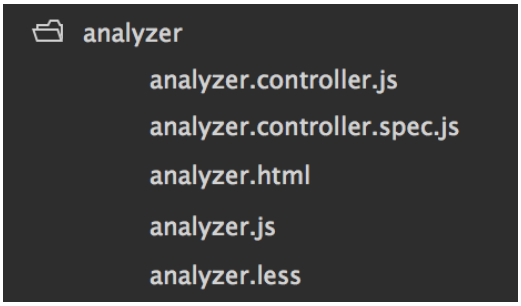
Slika 4.1: Diagram primerov uporabe.

4.1 Struktura aplikacije

V smislu strukture aplikacije je kot glavno ogrodje uporabljeno ogrodje Node.js, ki s svojim upravljalnikom paketov "*npm*" omogoča namestitev vseh potrebnih orodij za razvoj in delovanje aplikacij, ki so povezani z ogrodjem Node.js. V strukturi aplikacije uporabimo še dodaten upravljalnik paketov Bower, ki, za razliko od "*npm*", skrbi za nalaganje paketov, ki niso odvisni od ogrodja Node.js, ampak se izvršujejo na strani odjemalca. To so v našem primeru ogrodja AngularJS, D3.js, Dimple.js in prav tako začetno ogrodje za oblikovanje strani Bootstrap. Bootstrap omogoči začetno strukturo HTML in CSS, s katero hitreje in lažje razvijemo končno obliko uporabniškega vmesnika. V aplikaciji bomo za vse glavne poglede in komponente uporabljali "*./less*" datoteke. LESS je predprocesorski jezik stila CSS, ki dodaja dinamično obnašanje v obliki spremenljivk, operacij, funkcij itd.

Datotečna struktura aplikacije je v osnovi razdeljena na strežniški del in odjemalski del. Na strežniškem delu imamo tako programske vmesnike Node.js ali Express.js. Tukaj se nahajajo funkcije, ki skrbijo za sprejemanje in pošiljanje podatkov na zunanje strežnike. Prav tako vsebujejo funkcije za avtentikacijo in avtorizacijo uporabnika.

Na odjemalskem delu imamo definirane poglede, krmilnike in storitve ogrodja AngularJS. Te so ločene glede na glavne poglede, kjer je vsak razdeljen na osnovni krmilnik, predlogo in datoteko LESS, ki definira stile CSS za te poglede ali predloge. Dodatno še uporabljamo datoteke z oznako "*.spec*", ki vsebujejo teste za posamezne krmilnike ali ostale vrste datotek (slika 4.2).



Slika 4.2: Primer razdelitve datotek posameznega pogleda.

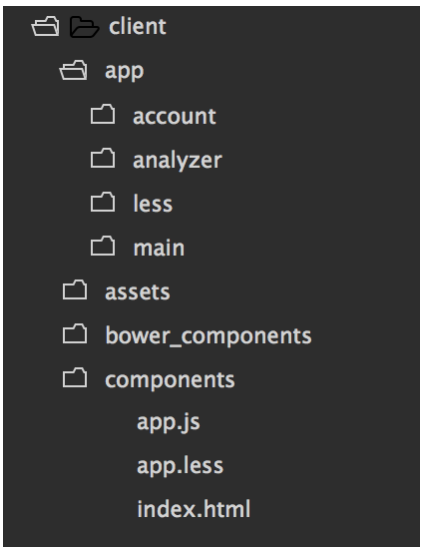
V "*.js*" datoteki še nastavimo usmerjanje za določen pogled (programska koda 4.1).

```
.config(function($stateProvider) {
  $stateProvider
    .state('main.analyzer', {
      data: {pageTitle: 'Analyzer'},
      url: '/analyzer?
chartType&dimension&displayType&filters&interval&metric&offset&sort&time&
' + 'widget',
      templateUrl: 'app/analyzer/analyzer.html',
      controller: 'AnalyzerCtrl'
    })
});
```

Programska koda 4.1: Primer usmerjanja za pogled Analyzer.

Začetna datoteka, ki inicializira aplikacijo je "*app.js*". "*Index.html*" je glavni dokument HTML, v katerega se naložijo vse komponente Bower in primerne predloge po incializaciji.

Storitve in posamezne ločene komponente se nahajajo v mapi "*components*", kjer so v večini definirane tudi vse potrebne storitve aplikacije. Komponente so potrebne za boljšo abstrakcijo posameznih kontrol, ki jih uporabimo na več mestih v aplikaciji. Primer datotečne strukture odjemalskega dela aplikacije je viden na sliki 4.3.



Slika 4.3: Datotečna stuktura spletne aplikacije.

4.2 Podatki

V aplikaciji uporabljamo 4 vrste podatkov za prikazovanje gledanosti in upravljanje uporabniških računov:

- Podatki gledanosti (angl. *Viewing data*)
- Podatki vsebin (angl. *Content data*)
- "*Dataset*" podatki
- Podatki uporabnikov in uporabniških računov

Podatki gledanosti in vsebin se hranijo na analitičnem strežniku TVBeat, od katerega preko poizvedb zahtevamo podatke.

Podatki gledanosti

Glavne podatke predstavljajo podatki gledanosti, ki jih pridobivamo iz zunanjega vmesnika. Ta vmesnik je dostopna točka analitičnega strežnika TVBeat. Analitični strežnik

je ločen del sistema in je zadolžen, da glede na podano zahtevo spletne aplikacije vrne podatke gledanosti v že določenem formatu.

Ko uporabnik sproži zahtevo po specifični poizvedbi, se izvrši klic na strežniško pot, definirano na strežniškem delu:

GET /:id/data

Programski vmesnik prestreže ta klic in glede na parametre, podane v klicu, sestavi primeren naslov URL ter pokliče vmesnik analitičnega strežnika. Spodaj navajamo primer kode za klic na vmesnik analitičnega strežnika, ki je razvidna iz programske kode 4.2.

```
function getData(dataset, parameters) {
  return
datasetService.getDataset(dataset).then(function(datasetDefinition) {
    return getDataResponse(_.extend({}, dataset, datasetDefinition),
parameters);
  });
}

function getDataResponse(dataset, parameters) {
  return request.getAsync(getRequestUrl(dataset, parameters), {timeout:
TIMEOUT}).spread(function(response, body) {
    if (response.statusCode != 200) {
      throw new Error('Unsuccessful attempt. Code: ' +
response.statusCode);
    }
    return JSON.parse(body);
  }).then(function(data) {
    data.intervals = _.mapValues(data.intervals, function(items) {
      return items.length && getTransformedData(dataset, items,
parameters) || [];
    });
    return data;
  });
}
```

Programska koda 4.2: Funkcije za klic na analitični strežnik.

Oblika končnega naslova:

GET	/api/dataset1/query?all=true&dimensions=channel.id&filter[channel.id][$in]=1 &filter[time][$gte]=2015-09-05T03:00:00+01:00&filter[time][$lt]=2015-09-06T03:00:00+01:00&intervals=total&limit=30&offset=0&sort[metric]=reachPercent &sort[order]=descending

V naslovu podamo ime "*dataseta*", za katerega želimo podatke. Prav tako pa moramo podati določene parametre, kot so:

- Dimenzija (angl. *Dimension*) - vrsta podatka gledanosti.
- Časovni filter ali časovno obdobje (angl. *Date range*) - določa časovno obdobje podatkov.
- Urejanje po določeni metriki, naraščajoče ali padajoče.
- Časovni interval, po katerem želimo dobiti razčlenjene podatke.

Na sliki 4.4 si še oglejmo primer vrnjenih podatkov, ki jih dobimo glede na zahtevo. Kot je razvidno iz prikaza, dobimo rezultat v formatu JSON, kjer so nanizani (v tem primeru) rezultati za vsak dan znotraj časovnega obdobja.


```
▼ intervals: {total: [{rating: 22080.7828949743, ratingPe
  ▼ day: [{rating: 22392.8458311537, ratingPercent: 0.223
    ▶ 0: {rating: 22392.8458311537, ratingPercent: 0.22359
    ▶ 1: {rating: 760.0018889335, ratingPercent: 0.0075888
    ▶ 2: {rating: 21720.1496360214, ratingPercent: 0.21688
    ▼ 3: {rating: 805.2534385976, ratingPercent: 0.0080406
        averageDuration: 89.56244315907334
        avgDurAll: 11.578559935096667
        channel.id: 1598
        label: "Channel 5"
        rating: 805.2534385976
        ratingPercent: 0.0080406666
        reach: 12947
        reachPercent: 0.1292791881
        share: 0.0370740281
        time.interval: "2016-08-12T02:00:00Z"
        totalDuration: 1159564.951580523
```


Slika 4.4: Primer podatkov gledanosti.

Podatki vsebin

Vsakič, kadar zahtevamo podatke gledanosti, simultano iz analitičnega strežnika še na osnovi istih parametrov prikličemo podatke vsebin. Podatki vsebin vsebujejo informacije elektronskega programskega vodiča, ki povedo, katere vsebine so se v določenem časovnem obdobju predvajale. Strežniška pot za klic podatkov vsebin je naslednja:

GET /:id/content

Zahteva za podatke vsebin je oblikovana identično kot za podatke gledanosti, s to razliko, da se v naslovu URL poda še "/content" na mesto "/query". Začetek naslova torej izgleda tako:

/api/dataset1/content?[parametri]

Za primer si še poglejmo rezultat podatkov vsebin:


```
...
  {
    "end": "2016-08-11T02:10:00Z",
    "image": "",
    "value": 276801,
    "label": "Breaking Bad (Channel X)",
    "start": "2016-08-11T00:20:00Z",
    "channel.id": 1598
  },
  {
    "end": "2016-08-11T03:00:00Z",
    "image": "",
    "value": 276802,
    "label": "Above & Beyond (Channel X)",
    "start": "2016-08-11T02:10:00Z",
    "channel.id": 1598
  }
...
```


"Dataset" podatki

Kot že omenjeno, poznamo več naborov podatkov glede na ponudnika. Med temi lahko v aplikaciji preklapljamo, s tem pa gledamo tudi drugačne podatke. Vsak "*dataset*" vsebuje specifične dimenzije in vrednosti dimenzij. Prav tako vsebuje nabor metrik, ki jih podpira.

Za primer vzemimo dimenzijo kanal ali "*Channel*", ki vsebuje vse kanale, ki se pojavijo v določenem naboru podatkov.

Preden lahko prikažemo podatke, moramo dobiti vse dimenzije (in vrednosti dimenzij), kot tudi metrike. Dimenzija je vrsta podatka gledanosti. To so lahko kanali, naprave, regije, vsebina itd. Vsaka dimenzija vsebuje vrednosti, po katerih bodo podatki razdeljeni. Npr. dimenzija "*regija*" ima vrednosti: podravska, notranjska, primorska itd. Tako z dimenzijo določimo vrednosti, po katerih se bodo vrnjeni podatki gledanosti razdelili. V opisanem primeru bi dobili podatke razdeljene po vseh regijah. Dimenzije uporabljamo tudi za filtriranje istih podatkov.

Strežniška pot na vmesnik, kjer kličemo "*dataset*" podatke, je tako sledeča:

GET /api/datasets/:id

Podatki "*dataset*" so opisni podatki za posamezne nabore podatkov, ki jih hranimo v podatkovni bazi MongoDB, in sicer v zbirki "*dataset*".

Z vrednostmi metrik napolnimo kontrole za izbiro metrik, v katerih se bodo prikazovali podatki v grafih in tabelah. Vrednosti dimenzij se napolnijo v kontrolah za izbiro primerjav in filtrov nad podatki. Omenjene kontrole bomo podrobneje obravnavali v poglavjih, ki sledijo.

Primer podatkov dimenzij in metrik z analitičnega strežnika:

```
{
  "dimensions": [
    {
      "plural": "Channels",
      "singular": "Channel",
      "value": "channel.id"
      "values": [
        {
          "value": 1, "label": "Channel 1"
        },
        {
          "value": 2, "label": "Channel 2"
        }
       ]
    },
```

```
    {
      "plural": "Device types",
      "singular": "Device type",
      "value": "device.useragents"
      "values": [...]
    },

   ]
  "metrics": [
    {"value": "ratingPercent", "type": "float", "label": "Rating %"},
    ...
  ]
 }
```

Podatki uporabnikov in uporabniških računov

Podatki uporabniških računov so, za razliko od zgoraj opisanih, shranjeni v posebno podatkovno bazo. Uporablja se podatkovna baza MongoDB, kjer se za potrebe verodostojnosti in nastavitev uporabniških računov nahajata dve zbirki (angl. *Collections*) podatkov:

- Accounts
- Users

V zbirki "*Accounts*" so podatki računa stranke, ki lahko ima več uporabnikov. Vsak račun stranke vsebuje nastavitve ali omejitve za:

- Dostop do posameznih pogledov in funkcionalnosti.
- Dostop do posameznih naborov podatkov.
- Dostop do posameznih dimenzij, metrik itd.
- Seznam uporabnikov, ki so del tega računa.

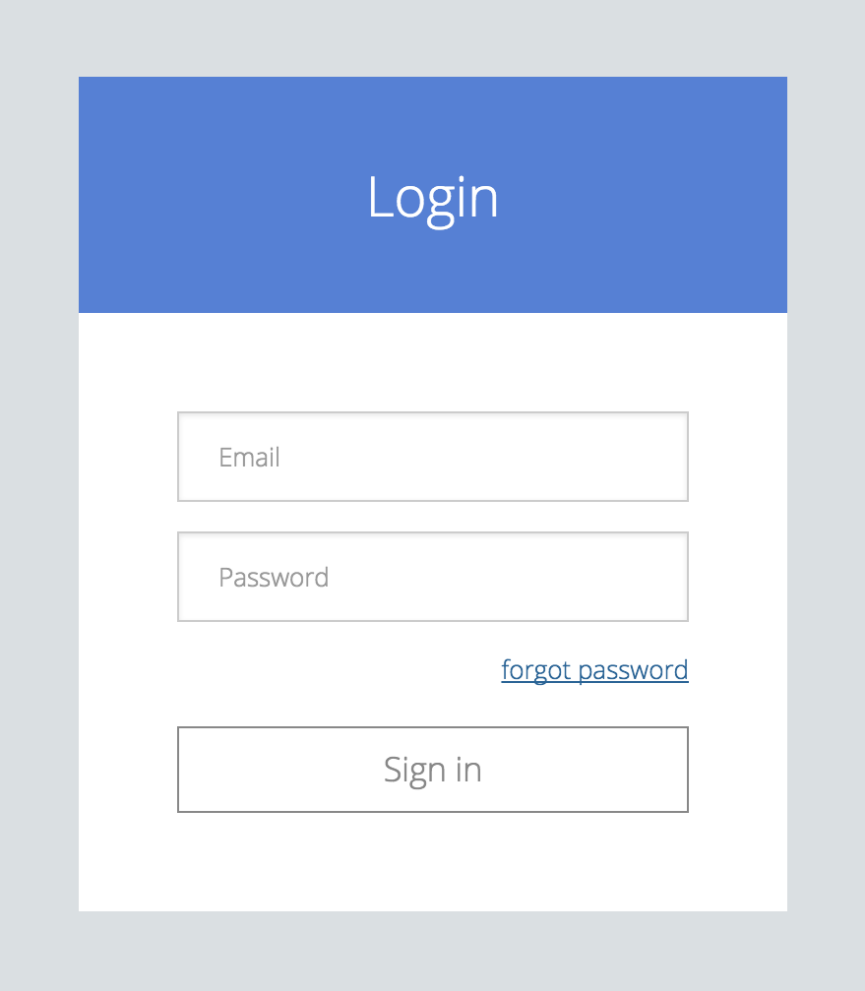
Zbirka "*Users*" vsebuje osebne in avtentikacijske podatke o posameznem uporabniku, kot so:

- Ime in priimek.
- Elektronska pošta (angl. *Email*).
- Geslo.
- ID računa strank, ki jim pripada.
- Vloga (user, admin, master).

Uporabnik glede na pripadajoč račun stranke deduje nastavitve ali omejitve v aplikaciji. Ko se uporabnik avtenticira, se nastavitveni podatki uporabnika primerjajo z "*dataset*" podatki. Na podlagi tega se uporabniku omejijo določene dimenzije in metrike. Prav tako se omejijo ali omogočijo določene funkcionalnosti v aplikaciji.

4.3 Prijava in avtentikacija uporabnika

Za vstop v aplikacijo se mora uporabnik prijaviti preko prijavnega okna (slika 4.5). S prijavo se sproži postopek avtentikacije, ki uporabniku glede na nastavitve omogoči določene pravice in funkcionalnosti.



Slika 4.5: Prijavno okno.

Ob pravilno vpisanih podatkih uporabnik proži zahtevo po prijavi, nakar se v krmilniku "*login*" pokliče funkcija za avtentikacijo iz storitve "*Auth*":

```
Auth.login({
  email: $scope.user.email,
  password: $scope.user.password
})
```

Storitev sproži postopek, kjer preveri uporabnika v podatkovni bazi in mu ustvari sejo. Nato preko strežniške poti "*GET /api/users/me*" pridobimo vse uporabnikove podatke in njegove nastavitve. Omenjena storitev "*Auth*" se v vseh krmilnikih podaja kot argument, s tem pa se kličejo notranje funkcije za avtentikacijo in pridobivanje podatkov uporabnika po potrebi. Uporabnik ostane prijavljen, dokler mu ne poteče seja. Seja je zapisana v piškotku na uporabnikovem računalniku. Če ugotovimo, da je uporabniku seja potekla, ga preusmerimo nazaj na pogled "*/login*". Vključno s prijavnim oknom omogočimo uporabniku tudi okno za ponastavljanje gesla.

Uporabnika po uspešni prijavi preusmerimo na prvi (glede na nastavitve) omogočen pogled.

4.4 Navigacijska vrstica

V vsakem pogledu znotraj aplikacije se nad vsebino naloži navigacijska vrstica. Ta služi za preklapljanje med različnimi pogledi, prikazi "*dataset*" izbirnega seznama in uporabniškega menija (slika 4.6).

Slika 4.6: Navigacijska vrstica.

Navigacijska vrstica se bo pojavljala kot kontrola v vseh pogledih, zato jo definiramo kot ločeno komponento. Zanjo definiramo direktivo s posebnim krmilnikom, predlogo in stilom LESS (programska koda 4.3).

```
angular.module('tvAnalyticsApp')
  .directive('navbar', function() {
    return {
      templateUrl: 'components/navbar/navbar.html',
      restrict: 'E',
      controller: 'NavbarControl'
    };
  });
```

Programska koda 4.3: Direktiva za navigacijsko vrstico.

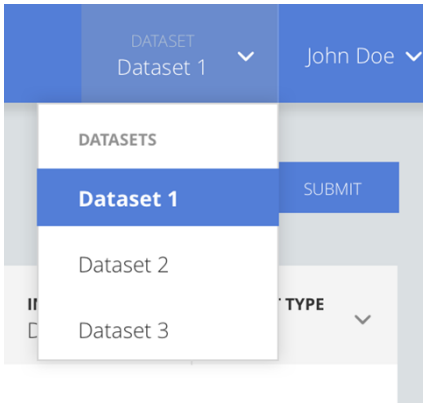
Vstavimo ga kot poseben element HTML v datoteki "*main.html*" (ta stran predstavlja glavno in najbolj zunanjo predlogo, v katero se naložijo vse ostale):

```
<navbar ng-class="{'responsive': isResponsive()}" ></navbar>
```

V navigacijsko vrstico se glede na nastavitve naložijo povezave do pogledov. Vsaka povezava vsebuje naslov URL, ki nas usmeri na ustrezen pogled:

- /dataset1/analyzer

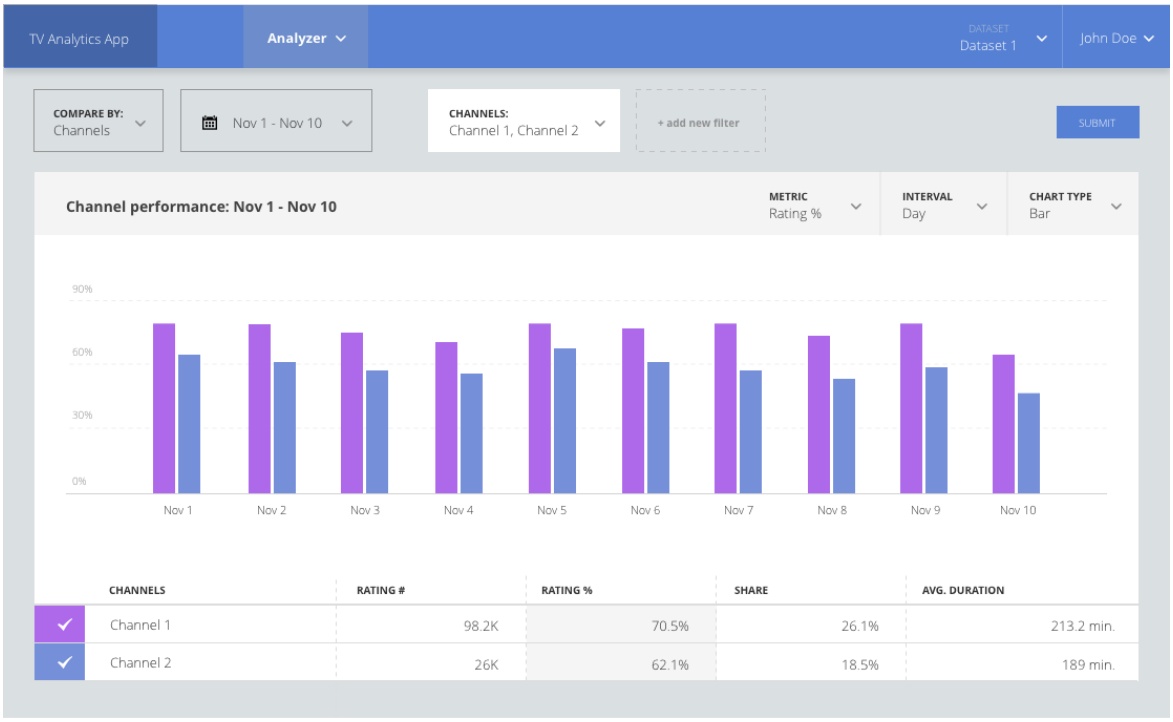
V izbornem seznamu "*dataset*" (slika 4.7) lahko izbiramo med želenimi nabori podatkov, ki temu primerno spremenijo naslov URL, in sicer "*/[dataset_name]/[view]*". Isti način tvorbe naslova URL se uporablja pri klicu podatkov gledanosti, vsebin in podatkov določenega nabora.



Slika 4.7: Izborna vrstica za nabor podatkov.

4.5 Analyzer - orodje za grajenje poizvedb

Analyzer (slika 4.8) je pogled, na katerega je uporabnik preusmerjen po uspešni prijavi. Z uporabo uporabniškega vmesnika omogoča enostavno grajenje poizvedb in predstavitev podatkov gledanosti. S kombinacijo različnih filtrov in možnosti lahko tvorimo kompleksne poizvedbe in jih prikažemo v obliki grafov ter tabel.



Slika 4.8: Orodje Analyzer.

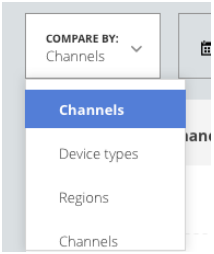
Analyzer je sestavljen iz skupine kontrol, s katerimi sestavljamo poizvedbo in spreminjamo prikaz podatkov v obliki grafa in tabele. Kontrole so izbirni seznami, ki jih ločimo kot posebne komponente, z namenom, da jih uporabimo v več primerih v različnih delih aplikacije. Vsi parametri, izbrani v kontrolah orodja Analyzer, se tvorijo in spreminjajo naslov URL. To omogoča, da lahko specifično poizvedbo shranimo ali delimo kot naslov, kot nam kaže spodaj prikazani primer:

```
https://localhost:3333/dataset1/analyzer?filters={"channel.id":[1,2]}&offset=0&sort={"order":"descending","value":"rating"}&time={"$gte":"2015-11-01T00:00:00+00:00","$lt":"2015-11-10T23:59:00+00:00"}&dimension=channel.id&intervals=["day","total"]&limit=30&displayType=all&metric=rating&interval=day&chartType=column
```

V prvi vrstici orodja Analyzer se nahajajo kontrole, s katerimi zgradimo poizvedbo, ki jo bomo poslali na analitični strežnik. Vsebuje kontrole za izbiro primerjalne ("*Compare*") dimenzije, datumskega obdobja in izbiro enega ali več filtrov.

Primerjalna ("*Compare*") dimenzija

Z izbirnim seznamom za izbiro primerjalne dimenzije (slika 4.9) izbiramo želeno dimenzijo. Vsaka dimenzija vsebuje nabor vrednosti, glede na katere se razdelijo podatki gledanosti. Če npr. izberemo dimenzijo "*Channel*", bomo dobili podatke razdeljene na vse kanale v tej dimenziji (Channel 1, Channel 2 itd.).



Slika 4.9: Izbirni seznam za primerjalne dimenzije.

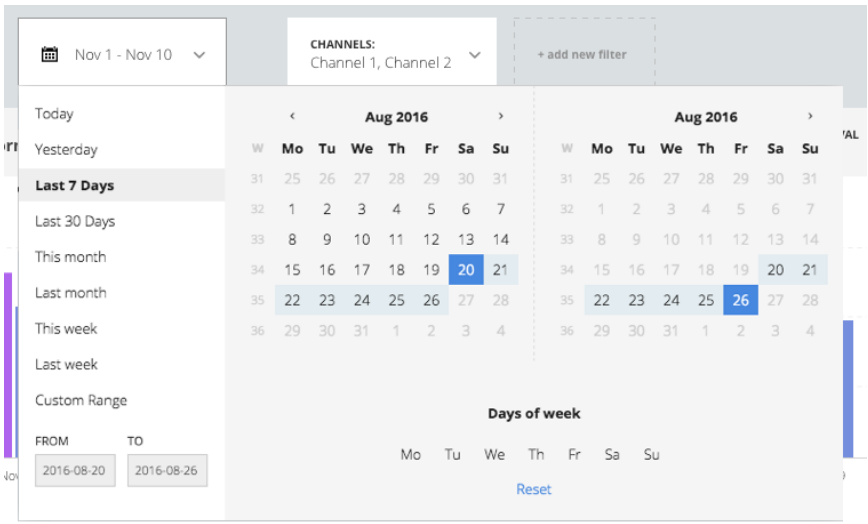
Ta stil izbirnega seznama uporabimo večkrat v aplikaciji in ga s tem razlogom povzamemo v posebni komponenti, ki jo imenujemo "*dropdown*". Zanjo ustvarimo posebno predlogo in direktivo, imenovano "*tvaDropdown*". V direktivi definiramo funkcije za nalaganje začetnih vrednosti, izbiranje vrednosti itd. V glavni predlogi vstavimo to komponento kot poseben element HTML (Programska koda 4.4).

```
<div class="btn-compare">
        <tva-dropdown title="Compare:" options="::compareOptions"
selected-value="query.dimension"></tva-dropdown>
</div>
```

Programska koda 4.4: Primer elementa HTML "*dropdown*" komponente za izbirnik primerjalne dimenzije.

Časovni filter ali datumsko obdobje

S časovnim filtrom (slika 4.10) določimo časovno obdobje (od/do) poizvedbe. Prav tako imamo možnost izbire vnaprej določenih obdobij in izbiro posameznih dni v tednu.

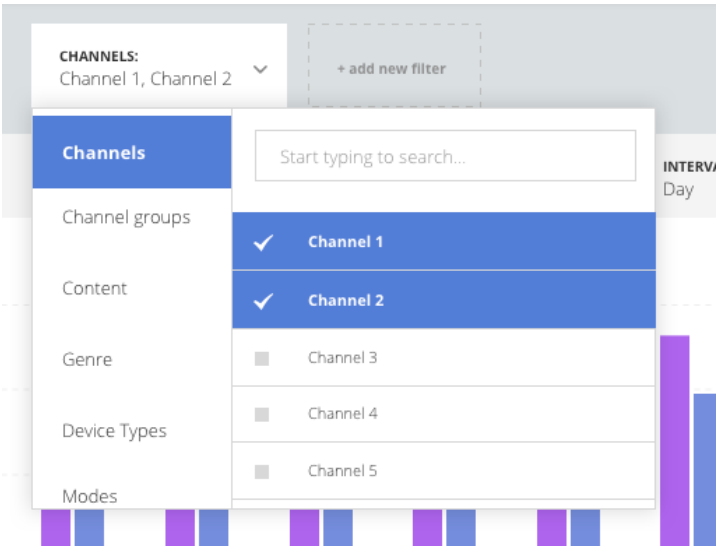


Slika 4.10: Časovni filter.

Časovni filter je posebna komponenta, ki je zgrajena iz več posameznih komponent. Direktiva "*tvaDatePicker*" definira filter in vsebuje funkcije za inicializacijo ter vstavljanje podkomponente "*tvaDaysOfWeek*", ki omogoča izbiro dni v tednu. Za osnovne funkcionalnosti izbire datuma iz koledarčkov se uporabi odprtokodna JavaScript komponenta "*bootstrap-daterangepicker*", ki je razširjena in dodelana za potrebe spletne aplikacije.

Filtri poizvedbe

Z izbirnikom filtrov (slika 4.11) imamo možnost dodajanja filtrov, po katerih se bodo rezultati poizvedbe filtrirali. Filtre lahko dodajamo enega za drugim s pomočjo "*tva-filter*" komponente, ki vsebuje seznam omogočenih dimenzij in njihovih vrednosti. Direktiva "*tvaFilters*" vsebuje funkcije, kjer naložimo potrebne dimenzije in vrednosti, kot tudi iskanje po vrednostih ter sestavljanje podkomponent, potrebnih za delovanje.



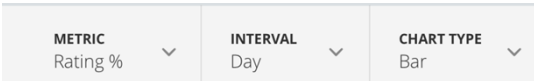
Slika 4.11: Izbirnik filtrov.

Vsak filter se prikaže v filter vrstici. Kadar jih dodajamo, se omejujejo tudi filtri, ki so že uporabljeni. Seznam za izbor, ki se odpira ob dodajanju novega ali urejanju obstoječega filtra, se glede na obravnavan filter primerno napolni z novimi ali že izbranimi vrednostmi.

Vrstica možnosti prikaza grafa in tabele

V tej vrstici (slika 4.12) navedemo naslov, ki opisuje, kakšni podatki v katerem časovnem obdobju so prikazani. Prav tako določimo tri izbirne sezname, ki določajo naslednje komponente:

- Metriko, po kateri se uredi tabela in temu primerno tudi prikaz v grafu.
- Interval, ki določa, po katerem časovnem intervalu bodo podatkovne točke prikazane v grafu.
- Tip grafa - palični in črtni.



Slika 4.12: Vrstica možnosti prikaza grafa in tabele.

Ti izbirni seznami uporabljajo isto komponento ("*dropdown*") kot že predhodno navedeni izbirni seznam dimenzij.

4.5.1 Graf

Grafični prikaz podatkov je z uporabniškega vidika najbolj pomemben del aplikacije. Z enim pogledom uporabnik dobi sliko, kjer lahko primerja vrednosti in na podlagi tega zbira pomembne informacije ter sprejema odločitve. Graf je definiran kot posebna komponenta "*tva-chart*", za katero deklariramo direktivo "*tvaChart*", ta pa vsebuje vse potrebne funkcije za nastavitve in izrisovanje grafa glede na podatke.

Ko dobimo podatke, jih moramo obdelati in pretvoriti v format, ki se bo prikazoval na grafu, prav tako tudi v tabeli. Na začetku ustvarimo nov element SVG, v katerem se bodo izrisovali vsi elementi grafa, in ga vstavimo v element HTML, ki bo vseboval graf:

```
dimple.newSvg("#chart-container", w, h);
```

Nadalje z "*new dimple.chart(svg, data)*" ustvarimo nov objekt grafa, ki ga podamo tudi v *scope* direktive. Na podlagi izbrane dimenzije in intervala ustvarimo X os z metodo:

```
chart.addCategoryAxis("x", [interval, dimension.name]);
```

Ta metoda bo ustvarila X os, ki bo podatkovne točke razdelila po datumih in jih kategorizirala glede na vrednosti dimenzij, ki se pojavijo v podatkih.

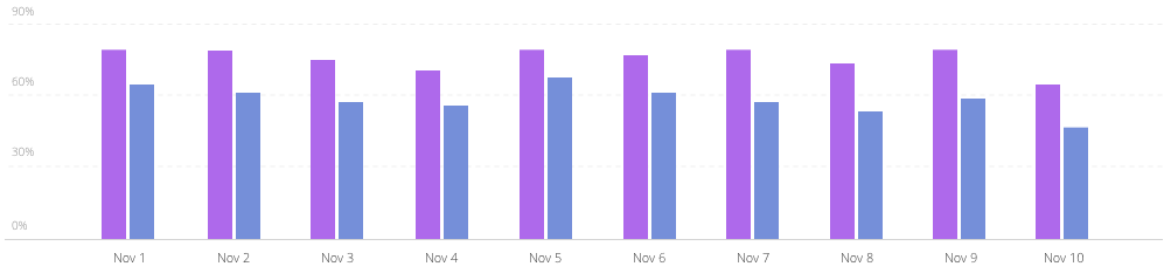
Določimo še Y os, ki predstavlja izbrano metriko. To metriko smo določili v izbornem seznamu za metrike:

```
chart.addMeasureAxis("y", metric);
```

Nazadnje ustvarimo še serije glede na unikatne vrednosti dimenzij v podatkih. Prav tako določimo "*dimple.plot.bar*" ali "*diplme.plot.line*", kjer definiramo, ali se serije prikažejo v palični ali črtni obliki:

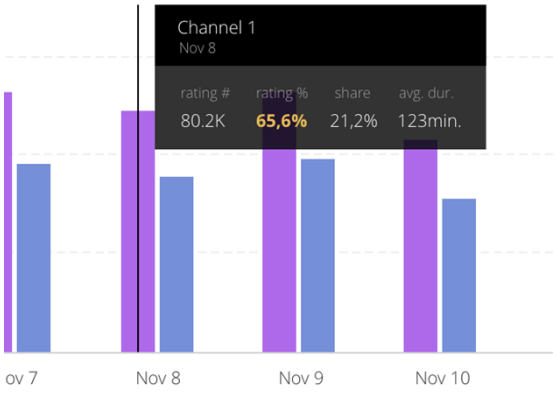
```
chart.addSeries(dimension.name, dimple.plot.bar);
```

V končni fazi izrišemo graf z metodo "*.draw()*". Za primer si poglejmo, kako je videti končni graf (slika 4.13).



Slika 4.13: Graf.

Za pregled podatkov vseh metrik posamičnih podatkovnih točk potrebujemo še t. i. "*tooltip*", ki se prikaže, kadar se s kurzorjem pomaknemo nad določeno podatkovno točko (slika 4.14). Tega zaradi omejitve knjižnice Dimple.js za oblikovanje *tooltipa* po meri, ustvarimo kot posebno komponento "*tva-tooltip*". Z objektom podatkovne točke, ki jo izpostavi knjižnica Dimple.js, določimo koordinate in tam prikažemo svoj *tooltip*.



Slika 4.14: "Tooltip".

4.5.2 Tabela

Tabela (slika 4.15) služi kot dodaten element h grafu, tako da lahko uporabnik glede na barvo razloči vrednost v tabeli. "*Total*" je povprečna vrednost posameznih metrik za celotno serijo vrednosti. Prav tako omogoča razvrščanje s klikom na naslove metrik v glavi tabele. S klikom na potrditveno polje, ki je obarvano s primerno barvo, lahko skrijemo ali prikažemo določeno serijo v grafu.

	CHANNELS	RATING #	RATING %	SHARE	AVG. DURATION
✓	Channel 1	98.2K	70.5%	26.1%	213.2 min.
✓	Channel 2	26K	62.1%	18.5%	189 min.

Slika 4.15: Tabela podatkov.

Tabela je definirana kot posebna komponenta "*tva-table*" in v direktivi "*tvaTable*" vsebuje funkcije za urejanje in skrivanje/prikazovanje podatkov v grafu.

5 ANALIZA PRIPRAVE PODATKOV V SPLETNI APLIKACIJI

Kadar uporabljamo veliko količino podatkov, je pomembno, kako hitro lahko prikažemo podatke uporabniku. Hitrost je v glavnem odvisna od zunanjega analitičnega strežnika podjetja TVbeat , ki surove podatke izračuna in poveže v primerno obliko in jih vrne naši spletni aplikaciji. Ko podatke dobi naša spletna aplikacija, je seveda te podatke potrebno še pripraviti v obliko, ki jo zahteva knjižnica za izrisovanje grafov.

Za preizkušanje smo uporabljali podatke gledanosti, ki jih je naša aplikacija, glede na sestavljeno poizvedbo, dobila s klicem na analitični strežnik podjetja TVbeat. Podatki so v obliki zapisa JSON.

Primer enega zapisa podatkov gledanosti:

```
...
{
  "rating": 0.4637094907,
  "ratingPercent": 0.0000045807,
  "share": 0.0055668783,
  "reach": 16,
  "averageDuration": 41.73385416666667,
  "label": "Channel 1",
  "reachPercent": 0.0001580534,
  "time.interval": "2015-11-1T03:00:00Z",
  "avgDurAll": 0.00659617689,
  "channel.id": 1,
  "totalDuration": 667.7416666666667
},
...
```

Vsak posamezen zapis predstavlja eno podatkovno točko (v našem primeru 1 dan) za en kanal. Vsebuje vrednosti metrik, kanal in časovno značko.

V fazi preizkušanja aplikacije smo testirali čas, ki ga je porabila funkcija "*prepareData()*" za pripravo podatkov. Ta funkcija je del spletne aplikacije in pregleduje elemente dobljenih podatkov gledanosti in pretvori vrednosti metrik v končno obliko, prav tako formatira

vrednosti časovnih značk (angl. *Timestamp*) v uporabniku prijazno obliko. Doda tudi lastnosti serij, ki se uporabijo pri izrisovanju grafov.

Čas izvajanja funkcije bomo testirali tako da bomo funkcijo za pripravo podatkov "*prepareData()"* ovili s funkcijami "*console.time()*" in "*console.timeEnd()*":

```
console.time('prepareData');
prepareData();
console.timeEnd('prepareData');
```

V konzoli brskalnika bomo dobili izpisan čas v milisekundah, ki predstavlja trajanje izvajanja funkcije (Primer: "*prepareData: 1.521ms*").

Za preizkus bomo vzeli poizvedbo, ki bo vrnila 10 kanalov na dnevnem intervalu v časovnih obdobjih 1, 7, 14, 30, 60, 90, 180 ali 360 dni. Opazovali bomo, kako se čas priprave podatkov v omenjeni funkciji povečuje z večjo količino podatkov. Količina podatkov je odvisna od števila dni, ki so zajeti v časovnem obdobju poizvedbe. Tako npr. poizvedba, ki ima časovno obdobje 7 dni in 10 kanalov vrne 70 zapisov katere imenujemo podatkovne točke.

Časovno obdobje	**Število kanalov**	**Količina zapisov**
1 dan	10	10
7 dni	10	70
14 dni	10	140
30 dni	10	300
60 dni	10	600
90 dni	10	900
180 dni	10	1800
360 dni	10	3600

Tabela 5.1: Količina zapisov glede na časovno obdobje in število kanalov.

Za primerjavo bomo k času izvajanja funkcije "*prepareData()"* še dodali čas, ki ga porabi analitični strežnik da sestavi podatke preden jih vrne naši aplikaciji.

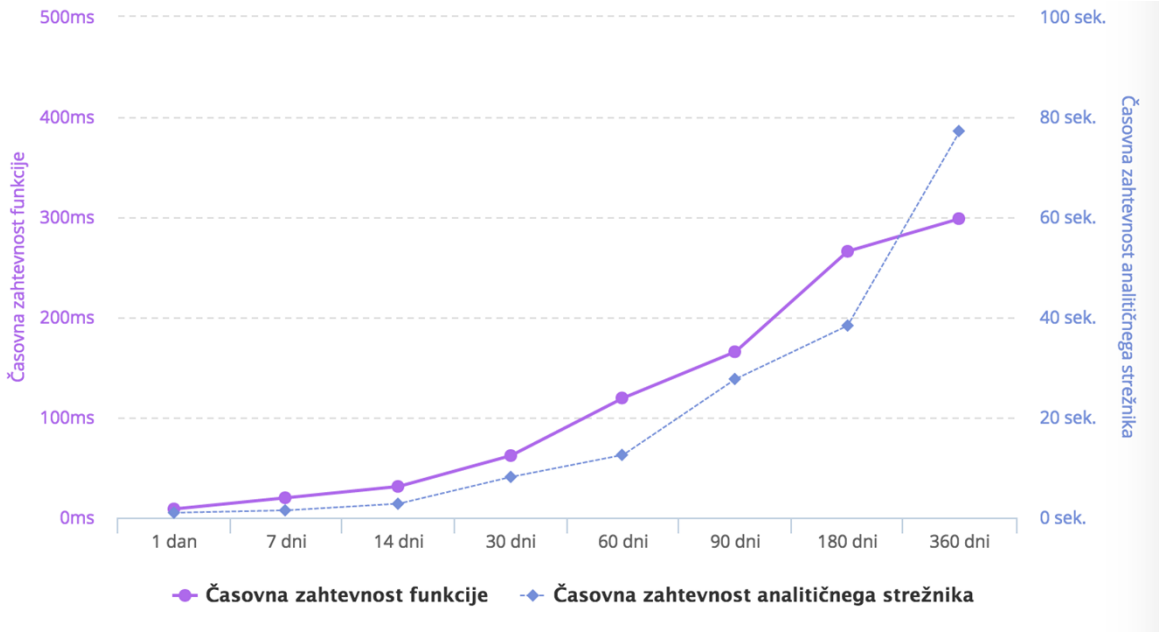
Ta nam v odzivu, poleg podatkovnih točk, še vrne čas izvajanja (v sekundah) sestave podatkov:

```
"queryExecutionTime": 23.2,
```

Računalnik, na katerem je bil opravljen preizkus, je MacBook Pro (Retina 13-inch, Late 2013) z naslednjimi specifikacijami:

- Procesor: 2,8 GHz Intel Core i7
- Pomnilnik: 8 GB 1600Mhz DDR3
- Operacijski sistem: OS X, verzija 10.11.5

Na sliki 5.1 so razvidni rezultati testiranja časovne zahtevnosti funkcije "*prepareData()*" za pripravo podatkov in časovne zahtevnosti sestave podatkov analitičnega strežnika. Obe časovni zahtevnosti sta prikazani vsaka na svoji Y osi (označeni z različno barvo).



Slika 5.1: Časovna zahtevnost funkcije za obdelavo podatkov in analitičnega strežnika.

Ugotovili smo, da se čas izvajanja funkcije "*prepareData()*" povečuje enakomerno s številom dni, zajetih v datumskem obdobju. Za primer vzemimo obdobje 7 dni, kjer je časovna zahtevnost v povprečju 7-8 ms in 14 dni, kjer je povprečje 13-15 ms. Vidimo, da se časovna zahtevnost podvoji, tako kot se podvoji časovno obdobje. Podobno se zgodi za časovna obdobja 30, 60 90 itd. Prav tako opazimo podobne rezultate časovne

zahtevnosti izvajanja poizvedbe na analitičnem strežniku. Na osnovi teh podatkov lahko zaključimo, da pri bistveno večjih količinah rezultatov funkcija v povprečju deluje z enako hitrostjo in se delovanje ne poslabša. Ker je funkcija odvisna od hitrosti brskalnika oz. hitrosti JavaScripta, je ta rezultat pričakovan. Povečanje časovne zahtevnosti izvajanja bi lahko pričakovali šele v primerih, ko bi bila količina podatkov tako velika, da brskalnik glede na svoje interne omejitve ne bi zmogel le-teh hraniti v pomnilniku. Do tega v sklopu same aplikacije ne more priti, saj je le-ta optimizirana in omejena.

6 SKLEP

Skozi spoznavanje sodobnih tehnologij merjenja televizijske gledanosti smo ugotavljali, kako se lahko prikaz le-teh podatkov uporabnikom predstavi na sodoben način, kot to počnejo mnoga analitična orodja za merjenje uporabe spletnih strani. V diplomskem delu smo preučevali JavaScript ogrodje AngularJS, ki je zelo popularno ogrodje za hiter in učinkovit razvoj spletnih aplikacij, kot tudi ogrodje D3.js, ki ponuja knjižnico Dimple.js za fleksibilno ter enostavno prikazovanje podatkov v obliki grafov. Pri preučevanju omenjenih tehnologij smo spoznali še vrsto ostalih tehnologij (Node.js, Express.js, MongoDB itd.), ki so potrebne za celostno rešitev, same po sebi pa ponujajo zelo modularen način razvoja. Navedene ugotovitve vsekakor pomagajo pri nadaljnjem razvoju, ki se bo zaradi novih funkcionalnosti in kompleksnosti aplikacij bistveno povečal. Skozi potek diplomskega dela smo ugotovili, da je kljub že vnaprej pripravljeni strukturi, ki jo ponuja ogrodje "*MEAN stack*", za potrebe uporabe te aplikacije potrebno dobro premisliti in si prilagoditi njeno samo strukturo. Ker že sama začetna struktura vsebuje veliko število različnih ogrodij, vsaka od teh pa narekuje svojo metodologijo razvoja, smo se morali dodobra spoznati s tehnologijami, da bi se aplikacije lahko lotili na pravi način.

Pri preučevanju ogrodja D3.js in knjižnic Dimple.js smo raziskovali možnost uporabe le-teh za enostavno, učinkovito in hitro izrisovanje podatkov. Kljub temu da sta knjižnici zelo fleksibilni in hitri, pa bi za potrebe obravnavane spletne aplikacije bila zadostna uporaba bolj enostavnih in bolj celovitih knjižnic za grafe, kot je recimo knjižnica "*Highcharts*". Čeprav se prednosti ogrodja D3.js in knjižnice Dimple.js kažejo v večji fleksibilnosti pri ustvarjanju posebnih vizualizacij podatkov, smo v sklopu aplikacije spoznali, da smo ob njej dokaj omejeni v početju, in sicer ob dejstvu, v kakšni obliki želimo podatke prikazati. Zatorej se komercialne knjižnice, podprte s strani podjetij, izkažejo kot varnejša izbira, saj tako nismo primorani vsega vzdrževati in razvijati sami.

V praktičnem segmentu pričujočega diplomskega dela smo obravnavali osnovne funkcionalnosti spletne aplikacije, ki omogoča prikaz podatkov gledanosti. Ta uporabniku ponuja fleksibilno orodje za vpogled v podatke. Z abstrakcijo posameznih delov aplikacije v komponente smo lahko le-te uporabili večkrat in si s tem zagotovili lažji nadaljnji razvoj in večjo doslednost v aplikaciji. Spoznali smo tudi osnovno strukturo in vrsto podatkov, ki so potrebni za analizo televizijske gledanosti.

Ogrodje AngularJS zajema tehnologijo, ki je doživela zelo veliko podporo s strani razvijalcev in je dokazano uporabna za razvoj tako majhnih in enostavnih kot tudi velikih in kompleksnih spletnih aplikacij. Ogrodje D3.js in knjižnica Dimple.js še vedno služita kot kakovostno izhodišče za posebne rešitve, kjer sta za bolj specifične rešitve potrebna hitrost in fleksibilnost. Komercialne knjižnice pa so z napredkom hitrosti in učinkovitosti novejših brskalnikov postale zelo uporabne za enostavne vizualizacije.

VIRI IN LITERATURA

[1] Lerner, A. *Ng-book – The Complete Book on AngularJS*, str. 8

[2] Sotelo, C. A conceptual introduction to AngularJS. Dostopno na: http://paislee.io/a-conceptual-introduction-to-angularjs/ [10. 8. 2016].

[3] Databinding. Dostopno na: https://docs.angularjs.org/guide/databinding [10. 8. 2016].

[4] Lerner, A. *Ng-book – The Complete Book on AngularJS*, str. 18

[5] Mutero, F. MEAN - One language to rule them all. Dostopno na: https://lgitsmart.com/2015/02/25/mean-one-language-to-rule-them-all/ [11. 8. 2016].

[6] Trivedi, J. Databinding in AngularJS. Dostopno na: http://www.c-sharpcorner.com/UploadFile/ff2f08/data-binding-in-angularjs/ [11. 8. 2016].

[7] Fisher, T. AngularJS Scopes: An introduction. Dostopno na: http://blog.carbonfive.com/2014/02/11/angularjs-scopes-an-introduction/ [11. 8. 2016]

[8] Lerner, A. *Ng-book – The Complete Book on AngularJS*, str. 25 – 30

[9] Expression. Dostopno na: https://docs.angularjs.org/guide/expression [11. 8. 2016].

[10] Lerner, A. *Ng-book – The Complete Book on AngularJS*, str. 37

[11] Directive. Dostopno na: https://docs.angularjs.org/api/ng/directive [12. 8. 2016].

[12] AngularJS Services. Dostopno na: http://www.tutorialspoint.com/angularjs/angularjs_services.htm [12. 8. 2016].

[13] "MEAN stack". Dostopno na: http://learn.mean.io/ [12. 8. 2016].

[14] Node.js. Dostopno na: https://en.wikipedia.org/wiki/Node.js [10. 8. 2016].

[15] Hahn, E. Understanding Express. Dostopno na: https://evanhahn.com/understanding-express/ [10. 8. 2016].

[16] Jenkov, J. AngularJS Routes. Dostopno na: http://tutorials.jenkov.com/angularjs/routes.html [16. 8. 2016].

[17] MongoDB. Dostopno na: https://en.wikipedia.org/wiki/MongoDB [12. 8. 2016].

[18] Data-Driven Documents, Dostopno na: https://d3js.org/ [16. 8. 2016].

[19] Why build with D3.js. Dostopno na: https://www.dashingd3js.com/why-build-with-d3js [16. 8. 2016].

[20] Television history - The first 75 Years. Dostopno na: http://www.tvhistory.tv/First%20Day%20of%20TV.htm [17. 8. 2016].

[21] Intelsat I razvoj. Dostopno na: https://en.wikipedia.org/wiki/Intelsat_I razvoj [10. 8. 2016].

[22] Television. Dostopno na: https://en.wikipedia.org/wiki/Television#Satellite_television [17. 8. 2016].

[23] Kent, R. *Measuring media audiences*. London: Routledge, 1994 (str. 1).

[24] Suhadolnik, D. *Razvoj metod merjenja gledanosti televizije*: Diplomsko delo. Ljubljana: Ekonomska fakulteta Univerze v Ljubljani, 2002.

[25] Kent, R. *Measuring media audiences*. London: Routledge, 1994 (str.7-8).

Univerza v Mariboru

Fakulteta za elektrotehniko, računalništvo in informatiko

FERI

IZJAVA O AVTORSTVU

Spodaj podpisani/-a Alen Karahasanović

z vpisno številko 93593253

sem avtor/-ica diplomskega dela z naslovom:

SPLETNA APLIKACIJA ZA ANALIZO TELEVIZIJSKE GLEDANOSTI Z UPORABO OGRODIJ ANGULARJS IN D3.JS

(naslov diplomskega dela)

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal/-a samostojno pod mentorstvom (naziv, ime in priimek)

doc. dr. Borko Bošković

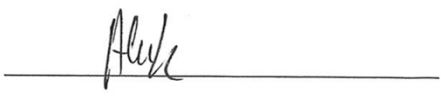
in somentorstvom (naziv, ime in priimek)

doc. dr. Boštjan Šumak

- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela.
- soglašam z javno objavo elektronske oblike diplomskega dela v DKUM.

V Mariboru, dne 29.8.2016

Podpis avtorja/-ice:



Fakulteta za elektrotehniko, računalništvo in informatiko

IZJAVA O USTREZNOSTI ZAKLJUČNEGA DELA

Podpisani mentor :

doc. dr. Borko Bošković

(ime in priimek mentorja)

in somentor (eden ali več, če obstajata):

doc. dr. Boštjan Šumak

(ime in priimek somentorja)

Izjavljam (-va), da je študent

Ime in priimek: Alen Karahasanović

Vpisna številka: 93593253

Na programu: Visokošolski strokovni študijski program računalništvo in informatika

izdelal zaključno delo z naslovom:

SPLETNA APLIKACIJA ZA ANALIZO TELEVIZIJSKE GLEDANOSTI Z UPORABO OGRODIJ ANGULARJS IN D3.JS

(naslov zaključnega dela v slovenskem in angleškem jeziku)

WEB APPLICATION FOR THE ANALYSIS OF TELEVISION VIEWING USING ANGULARJS AND D3.JS FRAMEWORKS

v skladu z odobreno temo zaključnega dela, Navodilih o pripravi zaključnih del in mojimi (najinimi oziroma našimi) navodili.

Preveril (-a, -i) in pregledal (-a, -i) sem (sva, smo) poročilo o plagiatorstvu.

Datum in kraj: 29. 8. 2016, Maribor Podpis mentorja:

Datum in kraj: 29.8.2016, MARIBOR Podpis somentorja (če obstaja):

Priloga:

– Poročilo o preverjanju podobnosti z drugimi deli.

Univerza v Mariboru

Fakulteta za elektrotehniko, računalništvo in informatiko

FERI

IZJAVA O ISTOVETNOSTI TISKANE IN ELEKTRONSKE VERZIJE ZAKLJUČNEGA DELA IN OBJAVI OSEBNIH PODATKOV DIPLOMANTOV

Ime in priimek avtorja-ice: Alen Karahasanović

Vpisna številka: 93593253

Študijski program: Visokošolski strokovni študijski program računalništvo in informatika

Naslov zaključnega dela: SPLETNA APLIKACIJA ZA ANALIZO TELEVIZIJSKE GLEDANOSTI Z UPORABO OGRODIJ ANGULARJS IN D3.JS

Mentor: doc. dr. Borko Bošković

Somentor: doc. dr. Boštjan Šumak

Podpisani-a Alen Karahasanović izjavljam, da sem za potrebe arhiviranja oddal elektronsko verzijo zaključnega dela v Digitalno knjižnico Univerze v Mariboru. Zaključno delo sem izdelal-a sam-a ob pomoči mentorja. V skladu s 1. odstavkom 21. člena Zakona o avtorskih in sorodnih pravicah dovoljujem, da se zgoraj navedeno zaključno delo objavi na portalu Digitalne knjižnice Univerze v Mariboru.

Tiskana verzija zaključnega dela je istovetna z elektronsko verzijo elektronski verziji, ki sem jo oddal za objavo v Digitalno knjižnico Univerze v Mariboru.

Zaključno delo zaradi zagotavljanja konkurenčne prednosti, varstva industrijske lastnine ali tajnosti podatkov naročnika: _____ ne sme biti javno dostopno do _____ (datum odloga javne objave ne sme biti daljši kot 3 leta od zagovora dela).

Podpisani izjavljam, da dovoljujem objavo osebnih podatkov, vezanih na zaključek študija (ime, priimek, leto in kraj rojstva, datum zaključka študija, naslov zaključnega dela), na spletnih straneh in v publikacijah UM.

Datum in kraj: 29.8.2016, MARIBOR Podpis avtorja-ice: Alen

Podpis mentorja: _____
(samo v primeru, če delo ne sme biti javno dostopno)

Podpis odgovorne osebe naročnika in žig: