Univerzitet u Beogradu  
Fakultet organizacionih nauka  
Katedra za elektonsko poslovanje

Backbone.JS

Seminarski rad iz Elektronskog poslovanja

Nastavnik: Dr Aleksandra Labus Saradnici: Tijana Ninković  
 Predrag Vulović

Studenti: Minić Nataša 163/16  
 Pavlović Nina 282/16  
 Obradović Darinka 332/16

Beograd, 2019.

# **Sadržaj**

[**Sadržaj** 2](#_Toc298444)

[**Uvod** 3](#_Toc298445)

[**MVC** 4](#_Toc298446)

[Model 5](#_Toc298447)

[Views 7](#_Toc298448)

[Collections 9](#_Toc298449)

[Events 11](#_Toc298450)

[Routers 11](#_Toc298451)

[Sync 12](#_Toc298452)

[**Instalacija Backbone.js** 13](#_Toc298453)

[**Razvoj Web aplikacije** 14](#_Toc298454)

[Home stranica 14](#_Toc298455)

[HTML kod Home stranice 15](#_Toc298456)

[Todos aplikacija 15](#_Toc298457)

[HTML kod Todos aplikacije 16](#_Toc298458)

[Backbone deo Todos aplikacije 17](#_Toc298459)

[Bloglist aplikacija 22](#_Toc298460)

[HTML kod Bloglist aplikacije 23](#_Toc298461)

[Backbone deo Bloglist aplikacije 25](#_Toc298462)

[**Literatura** 30](#_Toc298463)

# **Uvod**

Backbone.js je framework koji omogućava strukturiranje JavaScript koda po konceptu MVC što je skraćeno od “model-view-control”. Kreator ovog framework-a je Jeremy Ashkenas (Backbone.js).

Rad Backbone.js se zasniva na samo jednoj JavaScript bibiloteci koja se zove Underscore.js čime je omogućeno da svi elementi aplikacije budu sinhronizovani, kao, na primer, više klijenata i servera. Takođe, on potencira da se uz pomoć RESTful API obavlja komunikacija sa serverom. RESTful API je programski interfejs koji koristi HTTP zahteve kako bi pronašao neki resurs na serveru i odlučio da li će ga ažurirati, dobiti neke informacije o njemu ili ga izbrisati (Rouse, 2014).

Dizaniran je za kreiranje SPA (“single-page-applications”). SPA je web aplikacija koja komunicira sa korisnikom tako što ne učitava uvek novu stranicu sa servera, već dinamički prepisuje trenutnu stranicu na kojoj se po potrebi, odnosno na osnovu korisničke akcije, kao odgovor učitavaju i dodaju odgovaraući resursi čime se dobija nešto što je slično desktop aplikaciji (Angualar Single Page Applications (SPA): What are the Benefits?, 2018).

Na GitHub-u je dostupan izvorni kod ovog projekta, kao i dokumentacija, uputstvo za bolje razumevanje i primeri nekih aplikacija. Backbone se primenjuje u aplikacijama kao što su: LinkedIn Mobile, Twitter, FourSquare, Pandora, Soundcloud i još mnogim drugim.

Ova biblioteka omogućava organizaciju koda na uredan i efikasan način, što je naročito od velike važnosti za kodove pisane u programskom jeziku JQuery, koji je sam po sebi nepregledan i neuredan (Backbone.js).

Prvo, dat je pregled MVC modela na kome se Backbone.js zasniva, a zatim detaljan opis razvoja Web aplikacije.

# **MVC**

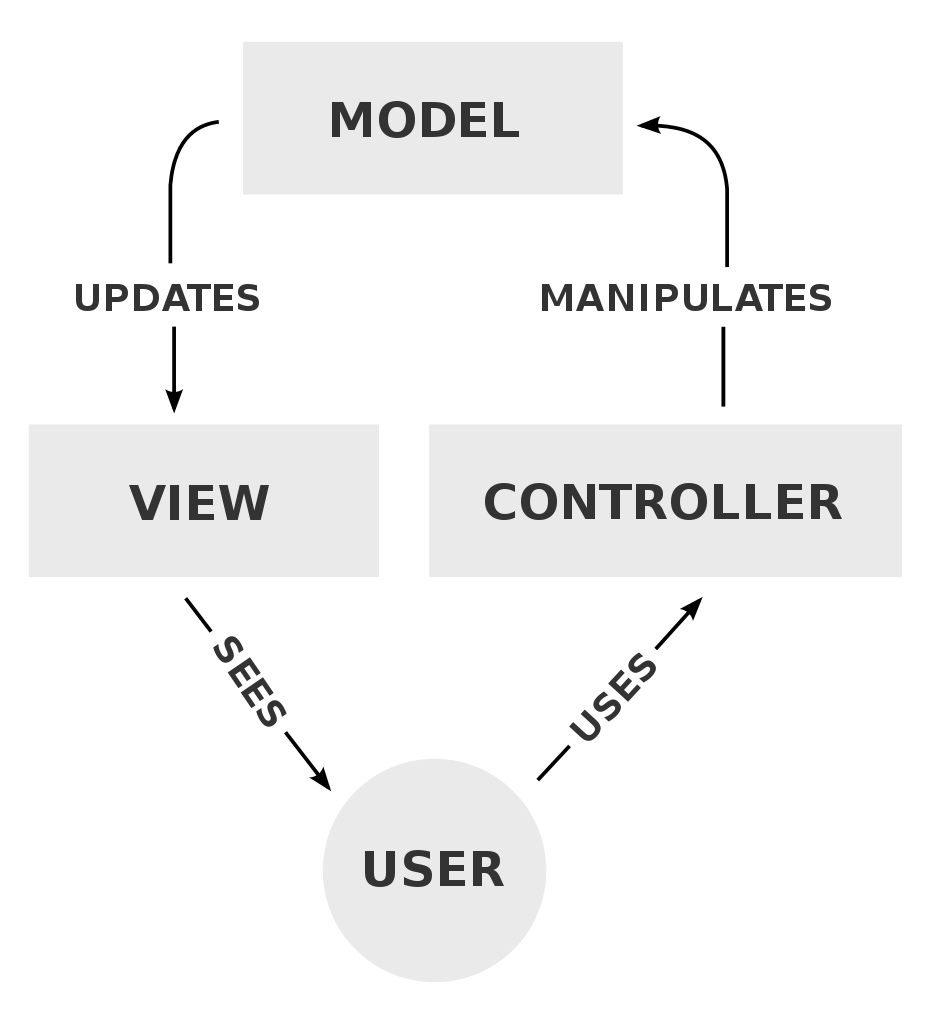
Veliki broj JavaScript okvira omogućava programerima da na lak način organizuju svoj kod koristeći varijacije obrasca poznatog pod imenom MVC ( Model-View-Controller). MVC razdvaja aspekte aplikacije na tri dela: Model, View i Controller.

**Model** je deo koda koji sadrži podatke i upravlja njima.

**View** je HTML prezentacija modela. Sa promenom modela menja se i view deo koda.

**Controller** je posrednik između prikaza podataka i modela. Dozvoljava da se sačuva trenutno stanje JavaScript aplikacije putem URL-a, korišćenjem znaka #. Na primer: <https://twitter.com/#search?q=backbone.js> (Stack Overflow, 2011), (Osmani, 2013).

Na slici 1 prikazana je MVC arhitektura.



Slika 1 - MVC arhitektura

JavaScript MVC okviri ne moraju striktno pratiti gore navedeni obrazac. Pojedina rešenja, uključujući i Backbone.js, su zasnovana na MVC konceptu spajaju funkcionalnosti pojedinih komponenti u druge. Iz tog razloga, definišemo model MV\*(Model-View). Backbone.js se u dva važna aspekta razlikuje od klasičnog MVC modela. Prva razlika je u tome što nema odvojenu controller komponentu (postojala je u prethodnim verzijama). Drugo, umesto tri odvojene komponente, ima šest. To su Model, View, Events, Routers, Sync i Collections (Mandić, 2015).

## Model

Modeli predstavljaju centralni deo bilo koje JavaScript aplikacije koji sadrži interaktivne podatke, kao i veliku logiku koja ga okružuje – konverzacije, validacije, kontrolu pristupa i drugo. Model pruža osnovni skup funkcionalnosti za upravljanje promenama (Backbone.js).

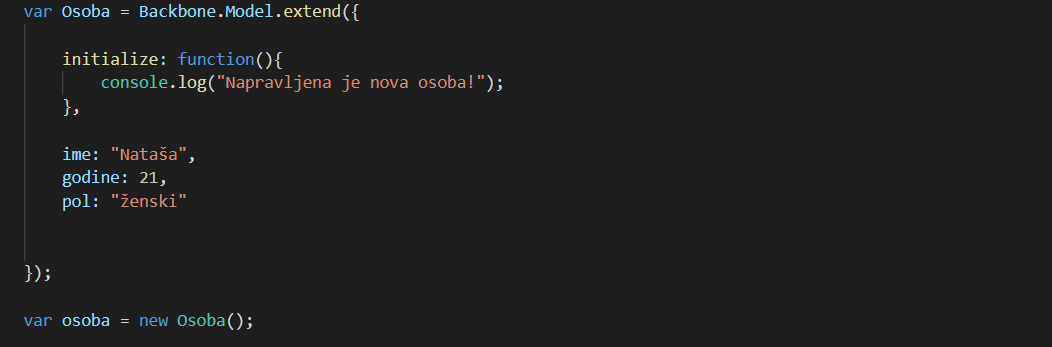
Da bismo kreirali sopstvenu Model klasu, proširujemo *Backbone.Model* i obezbeđujemo instancu *svojstva*, kao i opcionalnu instancu *svojstva* klase da budu priključene direktno na funkciju konstruktora.

*Backbone.Model.extend(properties, [classProperties])*

Extend ispravno postavlja lanac prototipa tako da podklase koje su kreirane sa proširenim opsegom mogu biti dodatno proširene i potklasirane koliko god želimo.

Kada kreiramo instancu Model, možemo preneti početne vrednosti atributa koji će biti postavljeni na modelu. Ako definišemo initialize funkciju, ona će biti pozvana kada se kreira model.

U nastavku je dat primer koji pokazuje definisanje modela sa prilagođenom metodom kao i postavljanje atributa. ( Slika 2 )



Slika 2 - Inicijalizacija modela

Napravljena je klasa Osoba koja nasleđuje klasu *Backbone.Model*, pri čijoj

inicijalizaciji se poziva funkcija initialize i ispisuje se u konzolu:

„Napravljena je nova osoba!“ i koja ima dodeljene početne atribute ime,

godine i pol. Dalje, inicijalicuje se objekat osoba, klase Osoba.

Atributi se takođe mogu dodeljivati implicitno, korišćenjem određenih metoda:

**Default** – koristi se za specifikaciju podrazumevanih atributa za model. Kada se kreira instanca modela, svi nespecificirani atributi će biti postavljeni na njihovu podrazumevanu vrednost.

*model.defaults ili model.defaults()*

**Set** – za postavljanje atributa ( jednog ili više ) na modelu.

*model.set(attributes, [options])*

**Get** – za pridobijanje trenutne vrednosti određenog atributa iz modela.

*model.get(attribute)*

**Unset** – uklanja određeni atribut iz modela.

*model.unset(attribute, [options])*

**Clear** – uklanja sve atribute iz modela.

*model.clear([options])*

**Has** – vraća true ukoliko je određeni atribut postavljen na non-null ili non-undefined vrednost. U suprotnom, vraća false.

*model.has(attribute)*

Backbone.js je objektno-orijentisan, pa se mogu koristiti i principi

nasleđivanja. Pri pozivanju određene funkcije koja se javlja u klasi, kao i u

njenoj podklasi, izvršava se samo hijerarhijski niža funkcija (Backbone.js).

## Views

Views ( pogledi ) predstavljaju HTML prezentaciju modela koja se menja sa samom promenom modela. Umesto kopanja u JSON objektu, traženja elementa u DOM-u i ručnog ažuriranja HTML koda, možemo vezati funkciju renderovanja pogleda na promenljivi događaj modela. Podaci modela su ažurirani svuda gde su prikazani u korisničkom interfejsu. Views se ponašaju kao Controllers (kontrolori) u MVC modelu. Njihova najznačajnija karakteristika je ta što poseduju svoj DOM element. Na jednostavan način nam omogućavaju da vršimo izmene nad HTML kodom, kao što su na primer, dodavanje novog reda, menjanje boje fonta, veličine, stila i drugo (Osmani, 2013). Počinjemo sa prikazima kreiranjem prilagođene klase View.

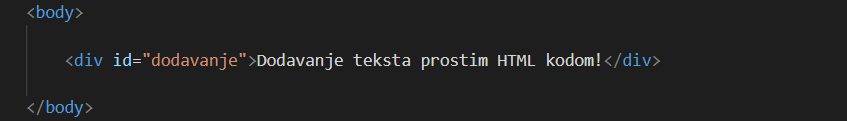
*Backbone.View.extend(properties, [classProperties])*

Postoji nekoliko specijalnih opcija koje će, ukoliko budu prosleđene, biti priključene direktno na prikaz: *model, collection, el, id, className, tagName, attributes i events*. Ukoliko prikaz definiše inicijalizacijsku funkciju, on će biti pozvan kad prvi prikaz bude kreiran. Ukoliko kreiramo prikaz koji se odnosi na element koji je već u DOM-u, unosimo ga opcijom: *new View({el: existingElement})*

**Render** – *view.render()*

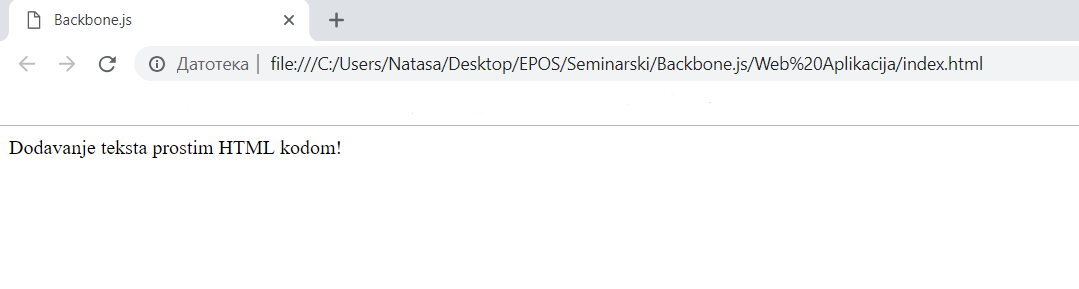
Funkcija Render je ugrađena Podrazumevana implementacija rendera je no-op. Funkcija render se koristi za dodavanje više linija HTML koda u postojećem HTML fajlu (Backbone.js).

U datom primeru predstavljamo način unošenja jednostavnog HTML koda. (Slika 3)



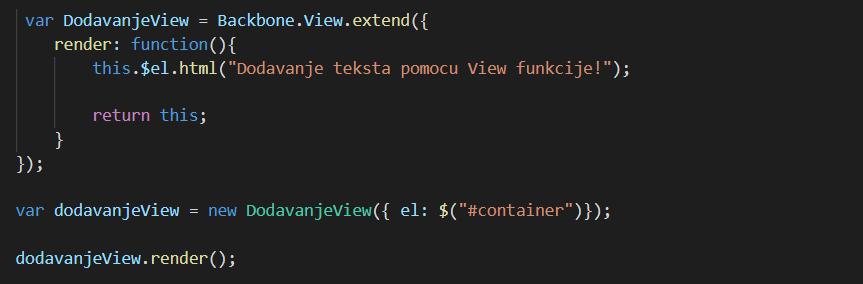
Slika 3 - Jednostavan HTML kod

Na ekranu se ispisuje sledeće: (Slika 4)



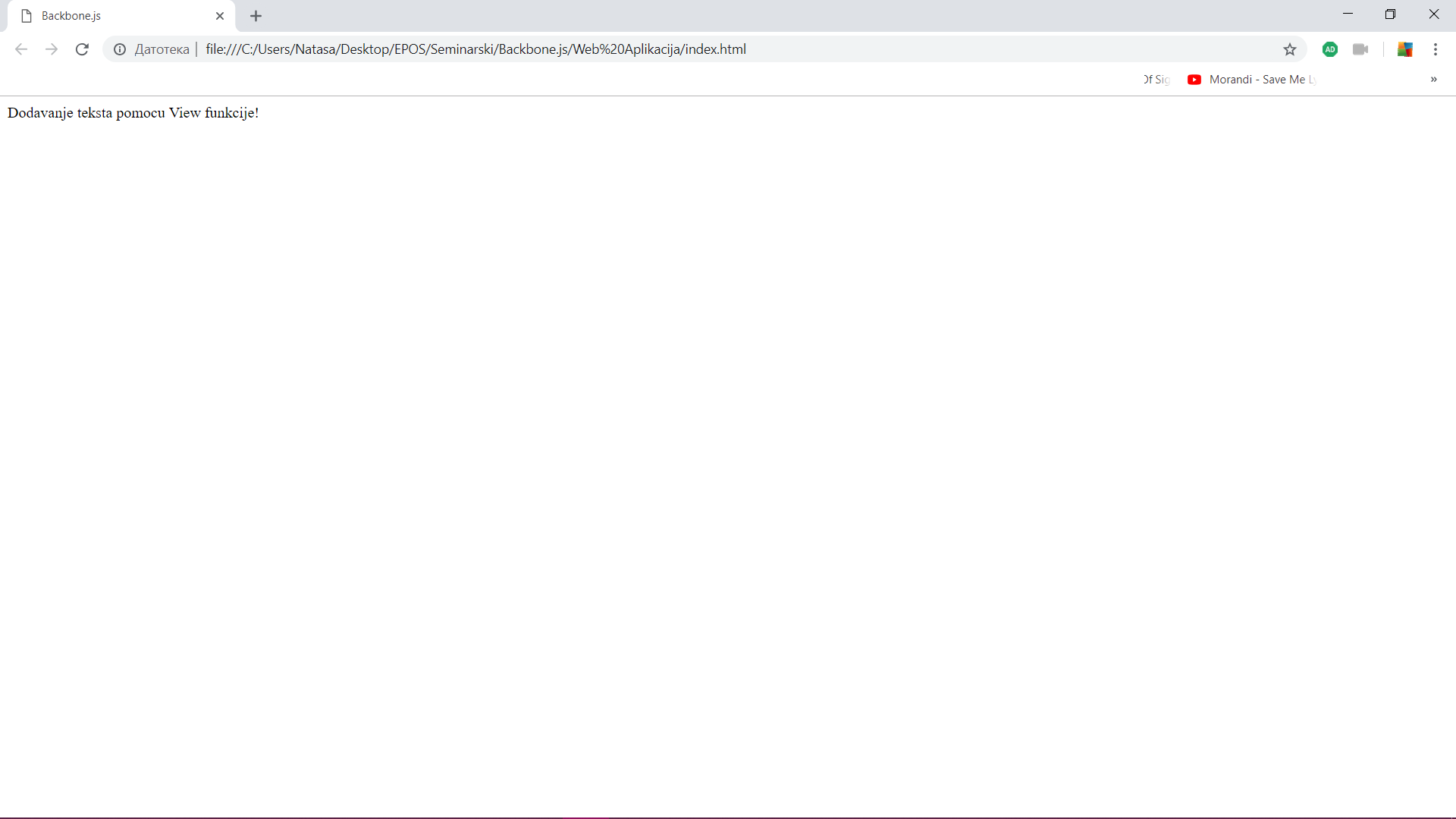
Slika 4 - Izgled HTML fajla nakog unosa koda

HTML dokument takođe možemo da menjamo tako što u diviziju sa određenim identifikatorom, u našem slučaju „container“ dodajemo određeni tekst.



Slika 5 - Korišćenje View funkcije

Inicijalizujemo objekat dodavanjeView klase DodavanjeView i prosleđujemo *el: $(„#container“)*, koja služi za referenciranje na identifikator „container“ u HTML kodu u kojem je skripta pozvana. Pokreće se njegova funkcija render i na ekranu se ispisuje rečenica „Dodavanje teksta pomoću View funkcije!“



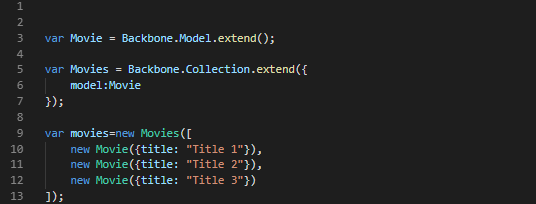
Slika 6 - Izgled HTML fajla nakon unosa koda

## Collections

Kolekcijepredstavljaju grupe modela,koje nastaju nasleđivanjem klase *Backbone.Collection.*

Prilikom kreiranja kolekcije neophodno je definisati svojstvo koje specificira tip modela koji će kolekcija sadržati, zajedno sa zahtevanim atributima.

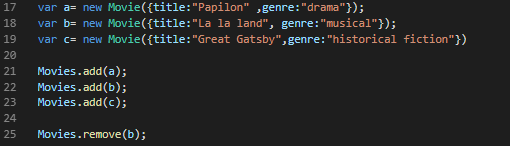
Na slici 7 je prikazan prost primer inicijalizacije kolekcije filmova.



Slika 7 - Incijalizacija kolekcije Movies

U posmatranom primeru, varijabla Movies predstavlja kolekciju modela Movie, dok ‘movies’ označava inicijalizaciju te kolekcije.

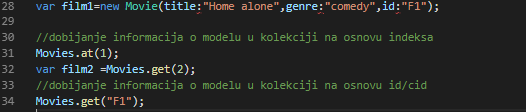
Nakon kreiranja same kolekcije, modeli mogu biti dodati ili izbrisani iz iste korišćenjem metoda *add()* ili *remove().*



Slika 8 - Ubacivanje elemenata posredstvom metode add(), izbacivanje elementa pomoću metode remove()

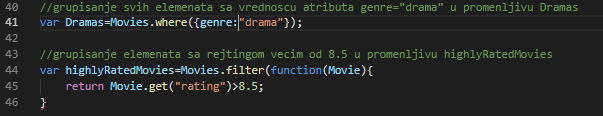
Osim pomenutih funkcija, brojne druge se koriste u upravljanju kolekcijama: *get(), at(), where(), filter(), change(), isEmpty(), size(), sort()* i druge.

U klijent-server aplikacijama, kolekcije sadrže modele pridobijene od servera. Bilo koje razmenjivanje podataka između klijenta i servera zahteva jedinstven nacin identifikacije modela. U Backbone.js, ovo je moguće zahvaljujući atributima *id, cid i idAttribute.*



Slika 9 - Pridobijanje elementa kolekcije na osnovu vrednosti indeksa,id/cid

Pretragu pojedinih elemenata, na osnovu poznatih vrednosti atributa je moguce izvrsiti pomocu metoda *where(), findWhere(), filter()* (Osmani, 2013)*.*

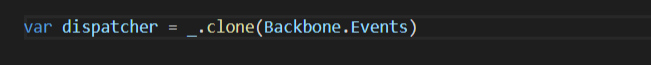


Slika 10 - Pretraživanje onih modela kolekcije, koji odgovaraju filterima pretrage

## Events

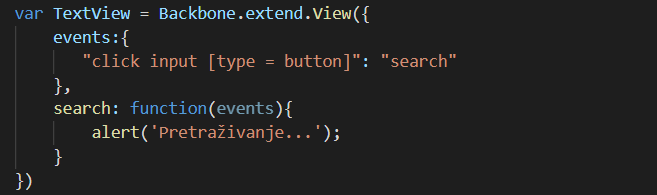
Events predstavlja modul koji se može mešati sa bilo kojim objektom, čime objektu daje mogućnost vezivanja i aktiviranja određenih događaja. Oni ne moraju biti deklarisani pre vezivanja. Neke od metoda koje se mogu koristiti za upravljanje događajima su: *on, off, trigger, one, listenTo, stopListening* i druge. Backbone ima neke Build-in događaje sa utvrđenim argumentima, neki od njih su: "add" (model, collection, options), "sort" (collection, options), "change" (model, options), "destroy" (model, collection, options) (Backbone.js), (Backbone.js Events).

Events se može uporebiti za pravljenje dispečera koji će koordinirati događajima između različitih oblasti aplikacije.



Slika 11 – Dodavanje dispečera

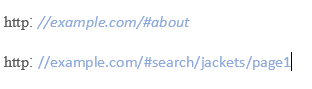
Na narednoj slici prikazana je funkcija *search* koja se pokreće klikom na button.



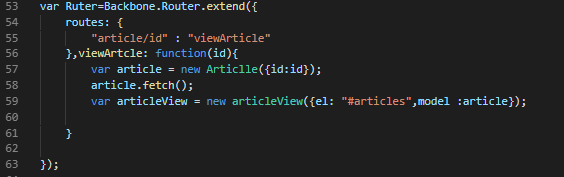
Slika 12 – Primena funkcije search

## Routers

Bilo koji deo vaše aplikacije, za koji želite realizaciju opcija bookmark-ovanja, deljenja i slično, zahteva URL. U Backbone-u, ruteri se koriste za kreiranje navigacije na aplikaciji, preciznije, omogućavaju da povežete URL-ove sa delovima vaše aplikacije. Povezivanje se realizuje uz pomoć hashtag-a(#).



Slika 13 – Primeri ruta



Slika 14 - Primer inicijalizacije rutera

Većina aplikacija realizovanih korišćenjem Backbone-a koriste primarno jedan ruter, ali nekada i dva, jer skoro svo neophodno rutiranje u aplikaciji može da bude organizovano u pojedinačnom ruteru, a da pritom ne postane nezgrapano.

## Sync

Sync je funkcija koja se poziva uvek kada se pokušava pročitati ili sačuvati model na serveru. Sync koristi *Jquery.ajax* da bi napravio RESTful JSON zahtev. Funkcije i metode u paketu ajax omogućavaju učitavanje podataka sa servera bez osveživanja stranice pretraživača (Ajax).

*Backbone.sync* ima potpis metode koji izgleda na sledeći način: *sync(method, model, [options]),* gde argumenti imaju naredna značenja:

**Method** – koristi CRUD metodu što je skraćeno od “create – read – update - delete“,

**Model** – model koji će biti sačuvan ili kolekcija za čuvanje,

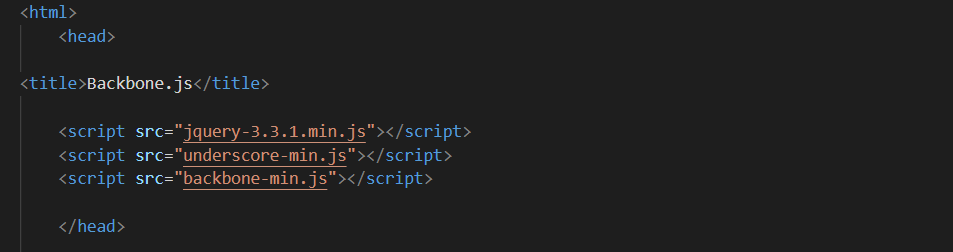
**Options** – povratne informacije o uspehu ili grešci izvršenja metode i drugim JQuery zathevima.

Prema podrazumevanoj implementaciji, kada *Backbone.sync* pošalje zahtev za čuvanje modela, njegovi atributi se prosleđuju serijalizovani kao JSON, i šalju se u telo HTTP-a sa nazivom application/json. Prilikom dobijanja JSON odgovora, atributi koji su promenjeni od strane servera moraju biti ažurirani na klijenstkoj strani. Ako se zahtev uspešno završi, kao povratnu informaciju dobijamo “sync“, u suprotnom “error“.

Funcija *sync* može se globalno zameniti sa *Backbone.sync*, a pored toga se može dodati Backbone kolekciji ili nekom pojedinačnom modelu.

# **Instalacija Backbone.js**

Preuzimanje se vrši putem zvaničnog www.backbonejs.org sajta. Nakon preuzimanja, neophodno je u projekat uključiti i biblioteku Underscore.js, kao i jQuery. Potrebno je referencirati ih u HTML fajlu, sa naglaskom da se referenciranje vrši u body tag-u. Neophodno je da referenca Jquery bude referencirana prva, zadim Underscore.js biblioteka i na kraju Backbone.js biblioteka. Razlog tome je činjenica da su Jquery i Underscore.js osnove za korišćenje Backbone.js biblioteke. Nakon ispunjenja zahteva za referenciranje, prelazi se na razvoj web aplikacije (Underscore.js), (JQuery), (Backbone.js).



Slika 15 - Referenciranje biblioteka

# **Razvoj Web aplikacije**

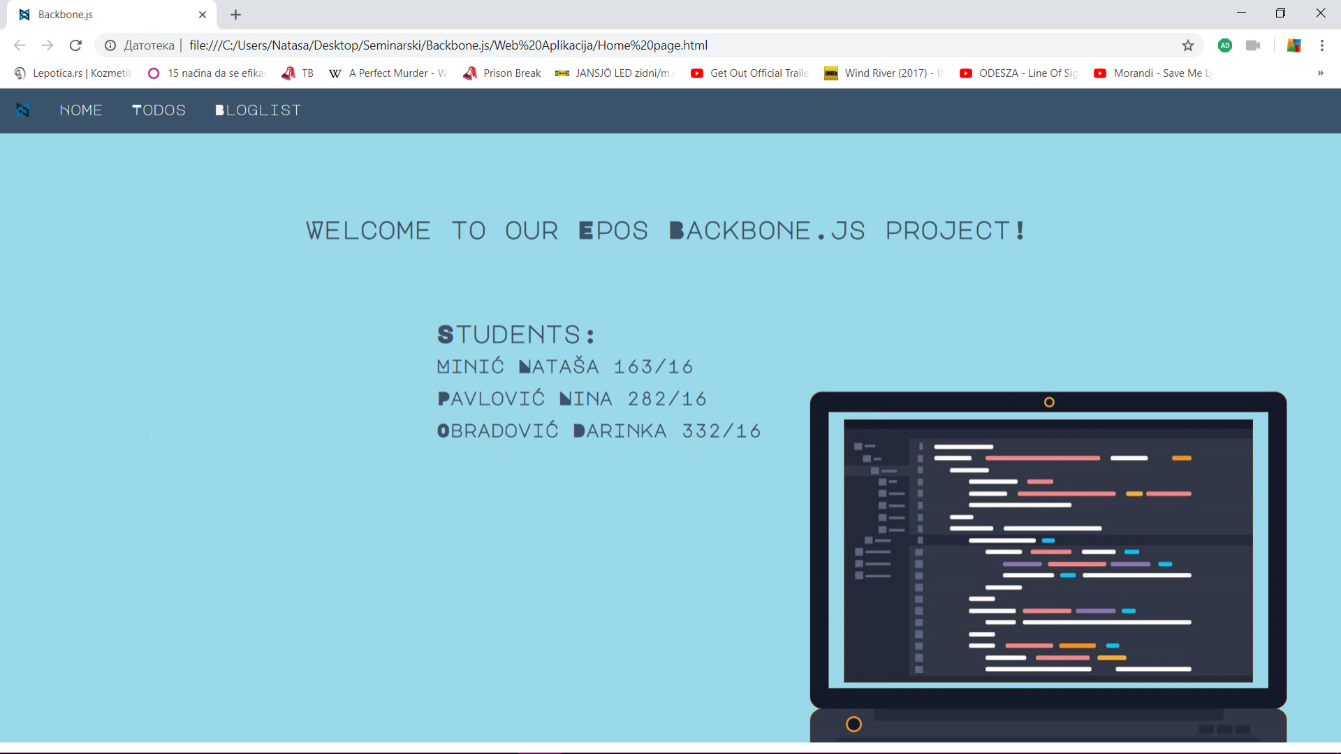
Našu Web aplikaciju podelile smo na tri celine: početnu ( home ) stranicu, aplikaciju Todos i aplikaciju Bloglist. U njenom kreiranju koristile smo HTML, CSS, Backbone, Bootstrape i Underscore biblioteke. Na svakoj stranici nalazi se meni pomoću koga je mogući prelazak sa jedne stranice na drugu.



Slika 16 - Navbar koji se nalazi na vrhu svake stanice

## Home stranica

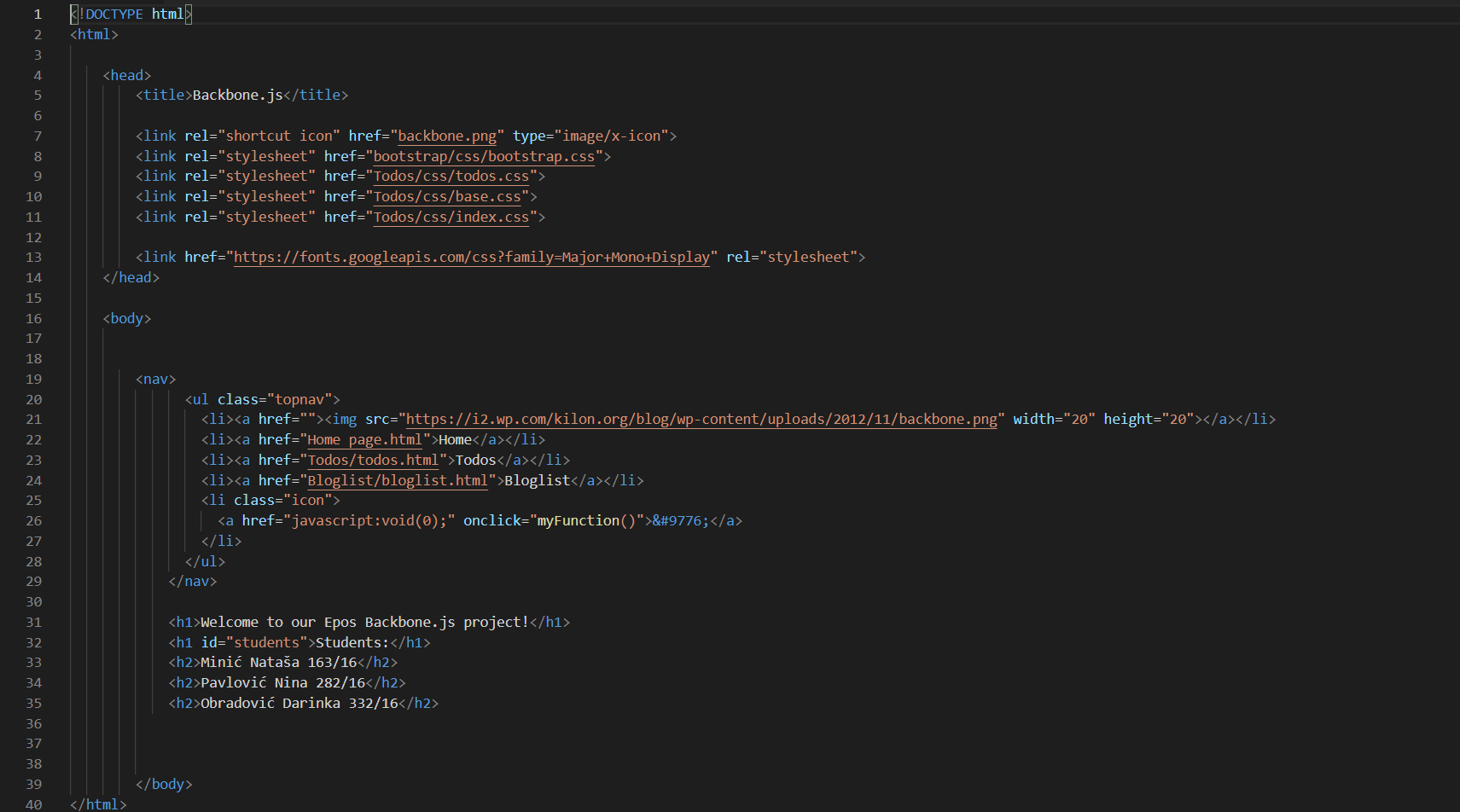
Home stranica je osnovni i najjednostavniji deo naše Web aplikacije. Predstavlja mesto koje okuplja sve celine aplikacije, a osim toga, nema drugu funkcionalnost. U meniju na vrhu strane se nalaze putanje do aplikacija Todos i Bloglist, kao i logo.



Slika 17 - Početna Home stranica

## HTML kod Home stranice

HTML kod stranice je relativno kratak iz razloga što ne sadrži neke specifične funkcionalnosti, kao ostale stranice. HTML kod je dat na slici u nastavku.



Slika 18 - HTML kod Home stranice

## Todos aplikacija

Ova celina Web aplikacije ima za svrhu podsećanje korisnika aplikacije o njegovim obavezama. Omogućava korisniku da unosi obaveze, čekira ih da su gotove, filtrira gotove, kao i da ih briše. Prvi korak predstavlja upisivanje obaveze u textbox.



Slika 19 - Textbox za unos obaveza

Nakon upisa obaveze, ona se unosi i aplikacija izgleda ovako.



Slika 20 - Aplikacija nakon unošenja obaveza

U aplikaciji sa leve strane postoje checkbox-ovi kojima obeležavamo ispunjenje određene obaveze, gde se one nakon obeležavanja filtriraju po kategorijama: All, Active i Completed. Prelaskom miša preko obaveze se sa desne strane pojavljuje znak x čijim pritiskom brišemo određenu obavezu.

## HTML kod Todos aplikacije

HTML kod Todos aplikacije je složeniji od Home aplikacije. Sadrži meni, koji je isti u svim stranicama, „todo sekciju“ koja je sačinjena od polja za unos teksta. Na polje za unos podešen je autofocus. Sekcija takođe sadrži dugme u obliku strelice čijim pritiskom selektujemo sve obaveze u listi obaveza, kao i dva template script bloka. Njihovo postojanje i funkcionalnosti biće objašnjeni u Backbone delu aplikacije.

## Backbone deo Todos aplikacije

Prvo se bavimo objašnjavanjem fajla *app-view.js.*

**

Slika 21 - Funcija initialize

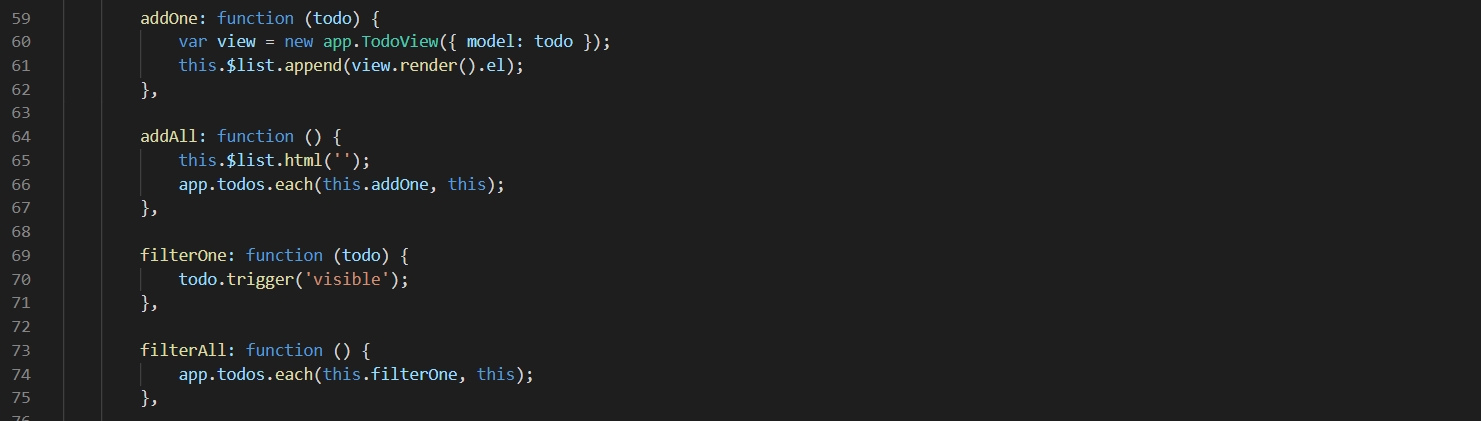
Uzimamo vrednosti iz HTML dokumenta u kome smo referencirali skriptu, koja sadrže klase *toogle-all, footer, todo-list i main* i dodeljujemo ih određenim varijablama. Zatim, postavljamo evente listenTo koji, ukoliko dođe do promena nad *app.todos* pozivaju odgovarajuće funkcije, koje su definisane:

**addOne** – Za $list smo uzeli da je <ul> ekement HTML dokumenta, pa je ova funkcija zaslužna za ubacivanje varijabli view u nenumerisanu listu.

**addAll** - Prva linija funkcije briše postojeću listu, a zatim se na svakoj sledećoj liniji prolaskom kroz kolekciju, korišćenjem funkcije *each*, svaki element kolekcije *app.todos,* funkcijom *addOne* ubacuje u nenumerisanu listu. Svaki put kada klijent upiše neku stavku u Todos aplikaciju, odvija se ova funkcija.

**filterOne** – Koristi se za prikazivanje određenog elementa

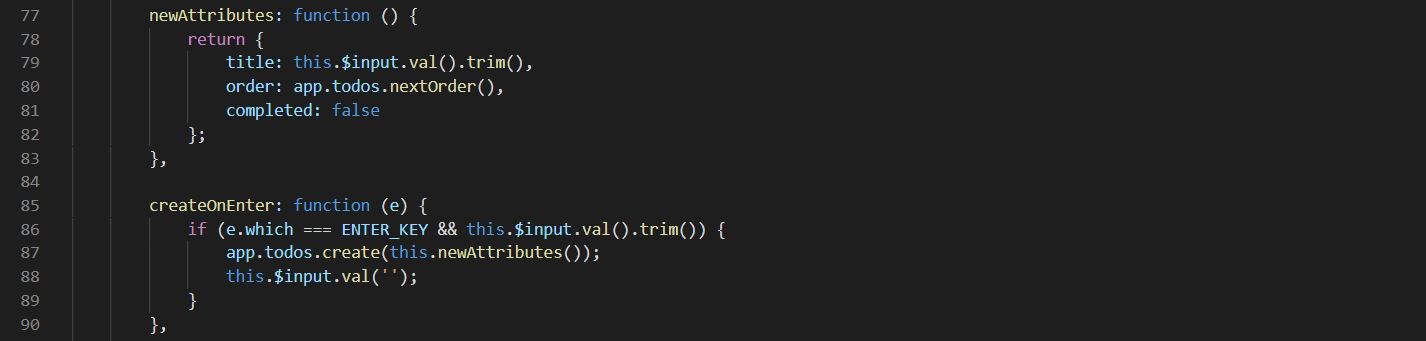
**filterAll** – Koristi se za prikaz određenih stavki prilikom prolaska kroz kolekciju korišćenjem funkcije *each*. Ukoliko želimo da nam aplikacija pokazuje, na primer, samo završene stavke, koristimo ovu funkciju.



Slika 22 - Funkcije

**newAttributes** – Ova funkcija vraća objekat koji sadrži tekst iz polja za unos, kao i *completed* atribut, čija je vrednost na početku *false* i postavlja za *order* određenu vrednost koja se nalazi u bazi.

**createOnEnter** – Pritiskom na enter, ova funkcija poziva funkciju *newAttributes* i objekat koji se vraća dodeljuje *app.todos* kolekciji. Zatim se briše vrednost polja za unos teksta.

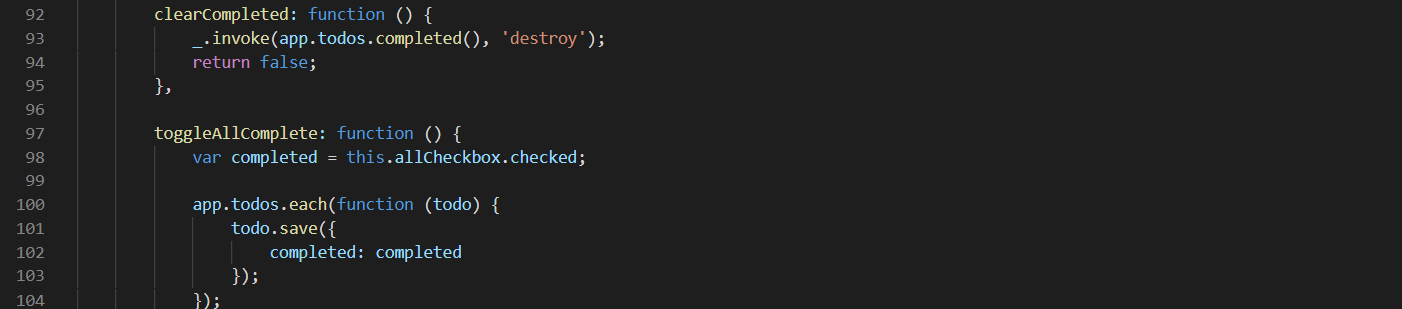


Slika 23 - Funkcije newAttributes i createOnEnter

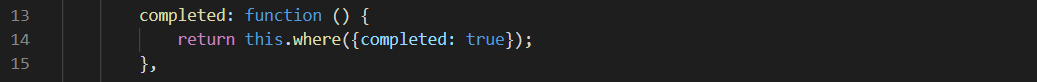
**clearCompleted** – Koristi funkciju *.invoke()* iz biblioteke Underscore.js. Koristi se za brisanje svih elemenata kolekcije koju vrati *app.todos.completed.* Njena aktivacija se vrši klikom na „Clear completed“ koje se nalazi u donjem desnom uglu aplikacije.

\* *app.todos.completed* je funkcija koja je definisana u skriptnom fajlu todos.js

**toggleAllCompleted** – Ovom funkcijom se za svaki element kolekcije app.todos postavlja vrednost njegovog atributa na „completed“. Za posledicu ima čekiranje svake stavke na glavnoj Todos listi i izvršava se klikom na strelicu na glavnoj stranici.



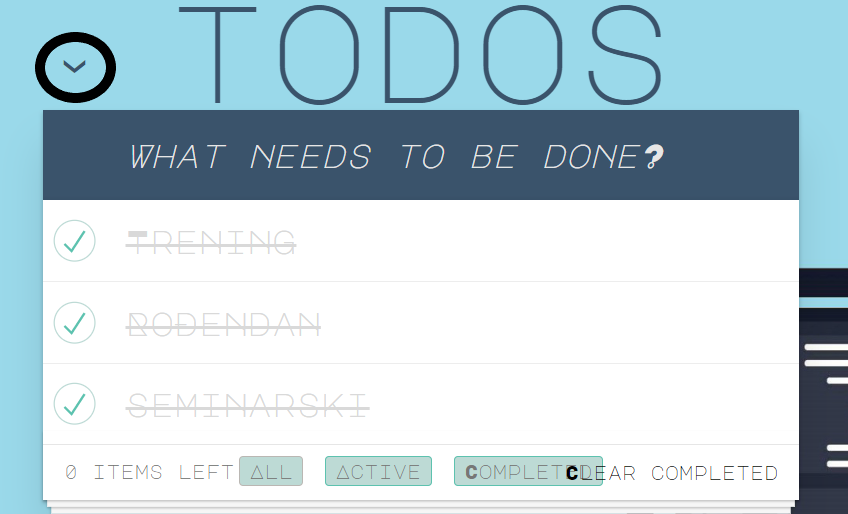
Slika 24 - Funcije clearCompleted i toggleAllComplete



Slika 25 - Funkcija completed, korišćena u prethodnoj slici

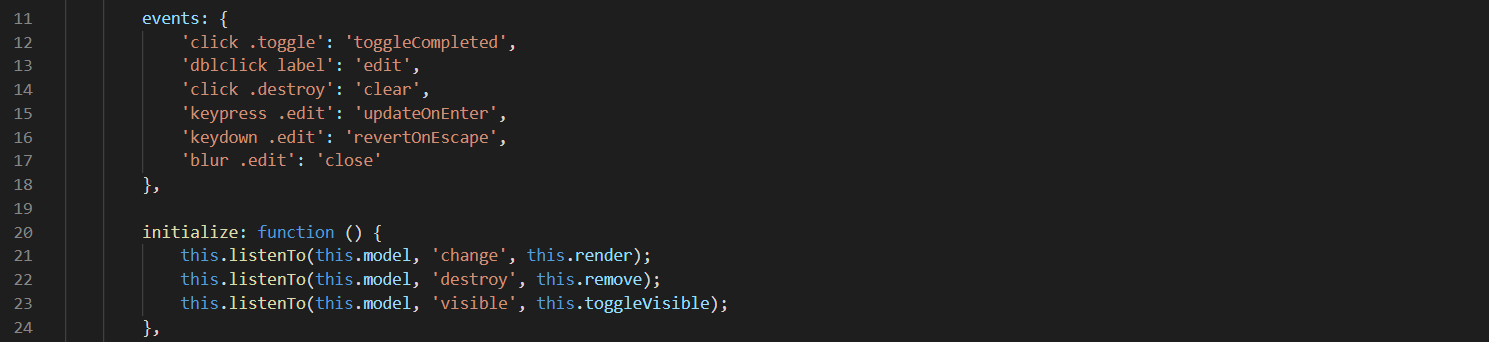


Slika 26 - Label koji aktivira funkciju clearCompleted



Slika 27 - Dugme koje aktivira funkciju toggleAllComplete

Dalje sledi objašnjavanje fajla *todo-views.js*.



Slika 28 - Događaji glavnog pregleda

Prilikom inicijalizacije ovog fajla, postavljaju se tri *listenTo* eventa: Prvi čeka promenu sadržaja, drugi čeka brisanje elemenata iz kolekcije, dok treći čeka promenu vidljivosti. Kada dođe do ispunjavanja određene promene, pozivaju se određene funkcije.

Osim osluškivača, postavljeno je i šest događaja, koji u skladu sa tokom programa pozivaju određene funkcije. U nastavku sledi opis određenih funkcija:

**edit** – Koristi se da bismo stavki koju menjamo dodali klasu „*editing*“, što je od koristi prilikom lakšeg upravljanja istom, u zavisnosti od njenog postojanja.



Slika 29 - Funkcija edit

**close** – Funkcija koja uzima vrednost iz polja za unos. Koristimo ovu funkciju u situaciji kada smo završili sa izmenom već unete stavke. Ukoliko polje te određene stavke ne sadrži ranije spomenutu klasu „*editing*“, izlazimo iz funkcije i ne dešava se ništa. U suprotnom, uzimamo novu vrednost iz polja za unos i menjamo vrednost atributa „*title*“ tog modela novom vrednošću iz polja za unos. Nakon izmene, brišemo klasu „*editing*“.



Slika 30 - Funkcije close

**revertOnEscape** – Funkcija koja proverava da li je pritisnuti taster jednak dugmetu *escape*, tako što proverava vrednosti iz ASCII tabele. Ako je odgovor potvrdan, funkcija pretpostavlja da je korisnik namerno izašao iz režima izmene, pa vraća stavku na stanje pre ulaska u taj režim. Funkcioniše tako što iz stavke briše klasu „*editing*“ i kao njenu vrednost postavlja vrednost „*title“* pre izmene.



Slika 31 - Funkcija revertOnEscape

**clear** – Njena funkcionalnost je brisanje objekta onog modela koji ga je pozvao. Pokreće se klikom na x koje se nalazi sa desne strane stavke.

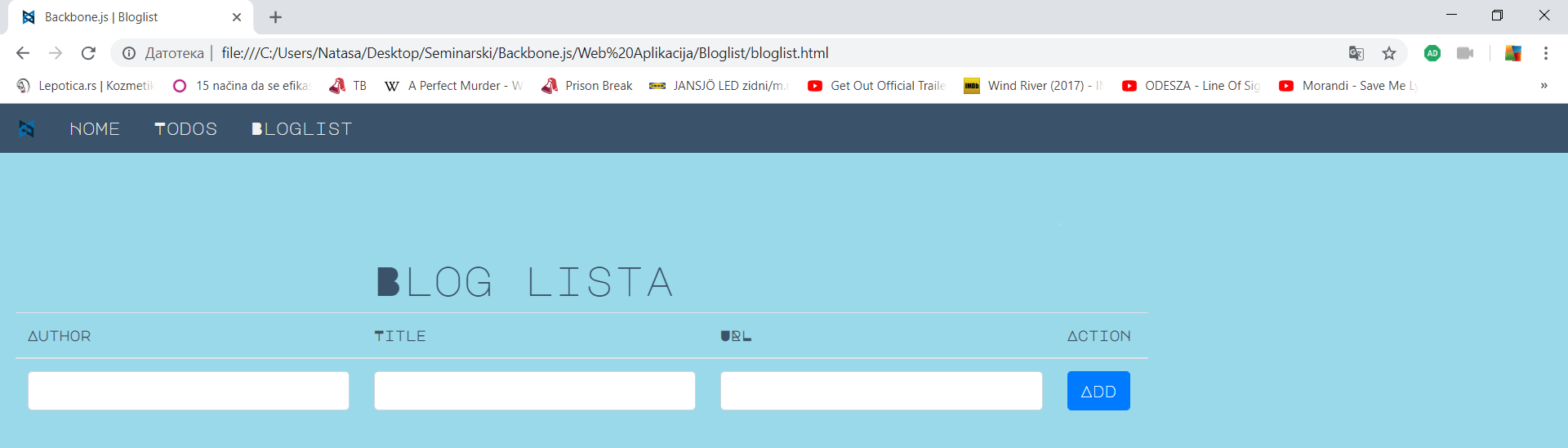


Slika 32 - Funkcija clear

## Bloglist aplikacija

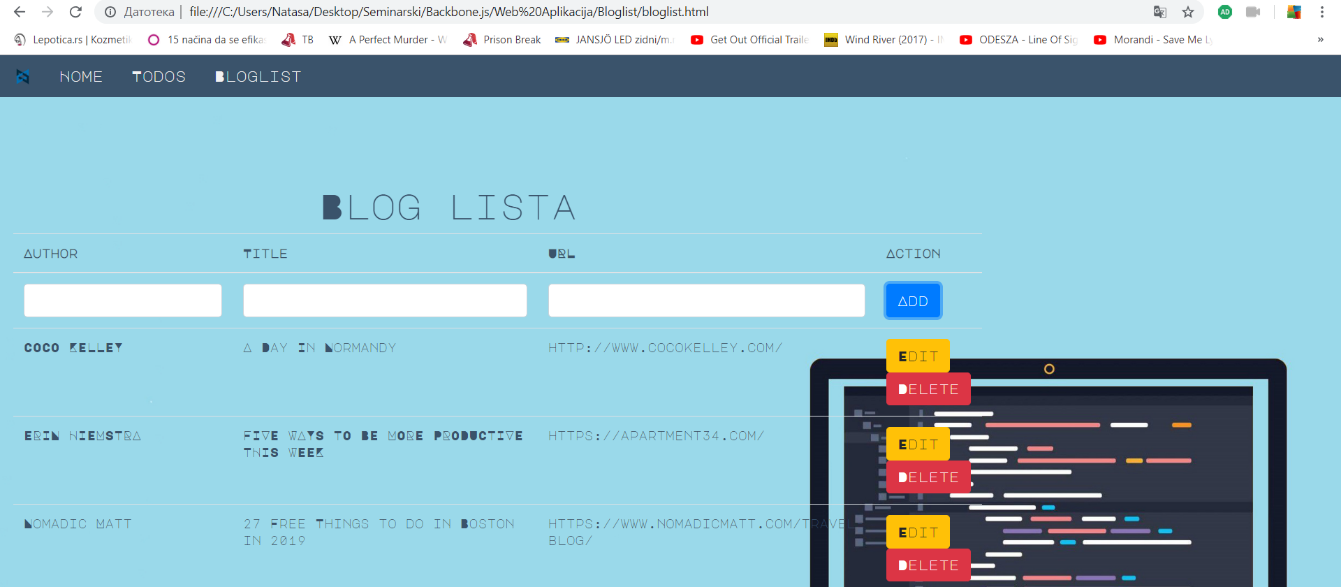
Bloglist je relativno jednostavna aplikacija koja se koristi za grupisanje blogova u odgovarajući spisak blogova. Funkcionalnosti kojima raspolaže obuhvataju: ubacivanje, menjanje (editovanje) i brisanje blogova sa stranice.

Kada se otvori stranica ona izgleda ovako (slika 33).



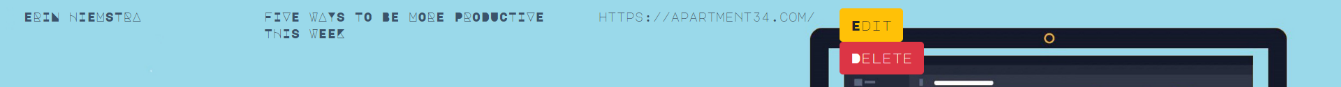
Slika 33 - Početno stanje stranice

Ukoliko želimo da dodamo novi blog, to vršimo pritiskom na dugme Add nakon popunjavanja polja Author Title i URL. Ponavljanjem tog postupka za još nekoliko blogova, formiran je spisak i naša stranica izgleda ovako (slika 34).

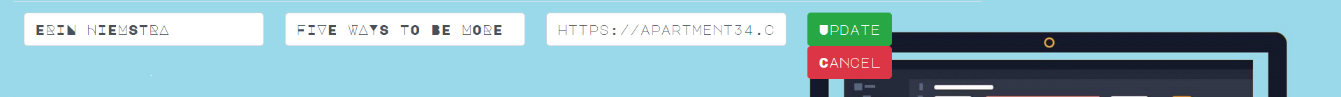


Slika 34 - Stanje aplikacije nakon dodatih nekoliko blogova

Pritisak dugmeta Edit za bilo koji odabran blog na spisku rezultuje privremenim nestajanjem dva dugmeta Edit i Delete, i prikazivanjem dva nova dugmeta Update i Cancel. Na primer klikom na dugme Edit za blog “Five ways to be more productive” dešava se sledeće:



Slika 35 - Pre pritiska na dugme edit



Slika 36 - Posle pritiska na dugme edit

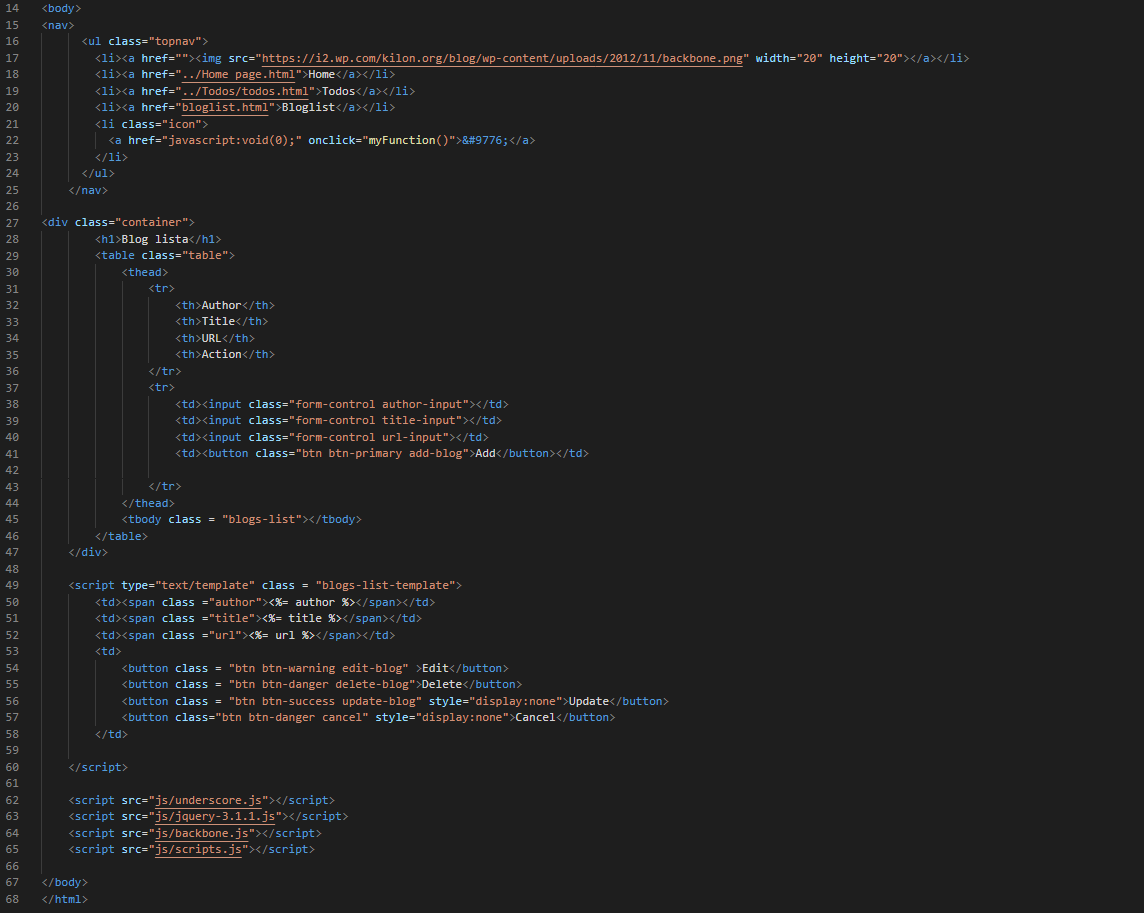
Uočavamo da se prikazuju polja za unos teksta, koja nam omogućavaju izmenu sadržaja prethodno unetih podataka. Nakon izvršenih promena i potvrđivanjem istih klikom na dugme Update spisak se ažurira sa novim podacima. Ukoliko želimo da izbrišemo određeni blog sa spiska, to radimo pritiskom dugmeta Delete. Uzmimo opet za primer blog “27 free things to do in Boston in 2019”, pritiskom dugmeta Delete, dati blog se briše i spisak se ažurira.



Slika 37 - Ažurirana lista bez izbrisanog autora Normadic Matt-a

## HTML kod Bloglist aplikacije

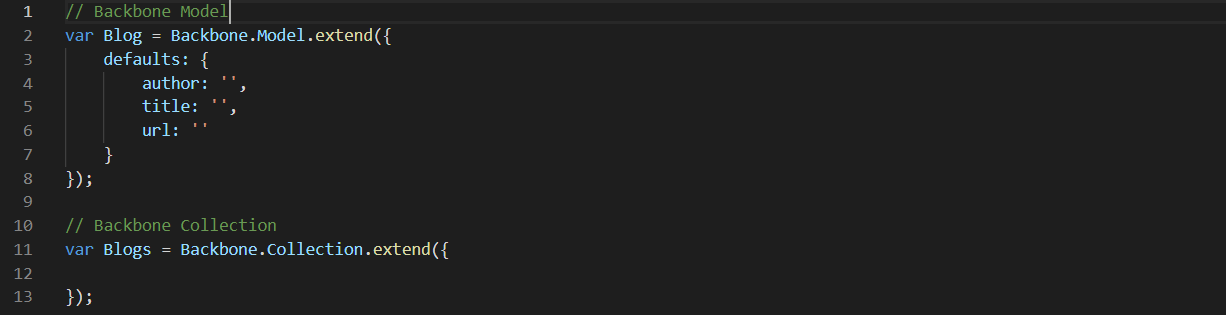
Struktura samog sajta je jednostavna kao što se može primetiti. Sadržaj ove stranice čini samo jedna tabela (table), koja korišćenjem svojstava Backbone-a dinamički dodaje nove redove.Jedina razlika koja postoji u odnosu na klasične HTML stranice, jeste prisustvo takozvanog template script bloka. U datom primeru, naš template block sadrži 3 span elementa(Author, Title, URL), kao i 4 dugmeta (Edit, Delete, Update, Cancel) od kojih su na početku dva definisana sa vrednošću atributa display:none. Pritiskom na dugme Edit ta vrednost se menja (implementirano pomoću Backbone dela aplikacije). Na dnu body elementa se nalazi referenca prema već opisanim bibliotekama, underscore.js, jquery.js, backbone.js, ujedno i prema logičkom delu naše aplikacije, script.js, koji ćemo detaljnije opisati u sledećem poglavlju.



Slika 38-HTML body element aplikacije

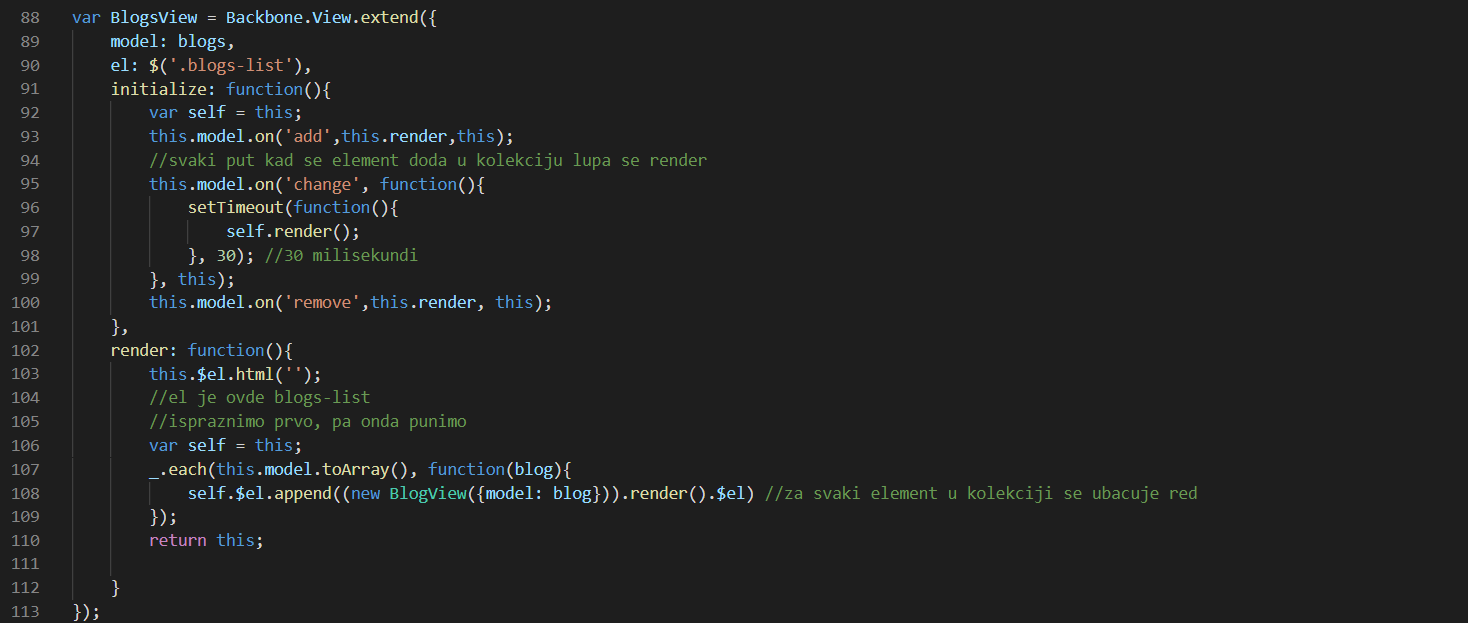
## Backbone deo Bloglist aplikacije

U ovom delu, osvrćemo se na logiku ove aplikacije, odnosno na ono što omogućava našoj aplikaciji da dinamički radi. Celokupna funkcionalnost aplikacije je implementirana korišćenjem pojedinačnih komponenti i funkcionalnosti prethodno opisanih u poglavlju o MVC (Model-View-Controller) konceptu. Prvenstveno je neophodno da napravimo fajl script.js, koji smo i referencirali u našem HTML fajlu. Nakon napravljenog fajla, možemo da počnemo sa implementacijom logike aplikacije. Počinjemo sa pravljenjem modela Blog, kojem se kao početne vrednosti za atribute author, title i url dodeljuju prazni stringovi. Zatim pravimo kolekciju blogova, pod nazivom Blogs, koja predstavlja kolekciju prethodno napravljenog modela Blog. Nakon toga inicijalizovaćemo kolekciju i dodeliti je varijabli blogs.



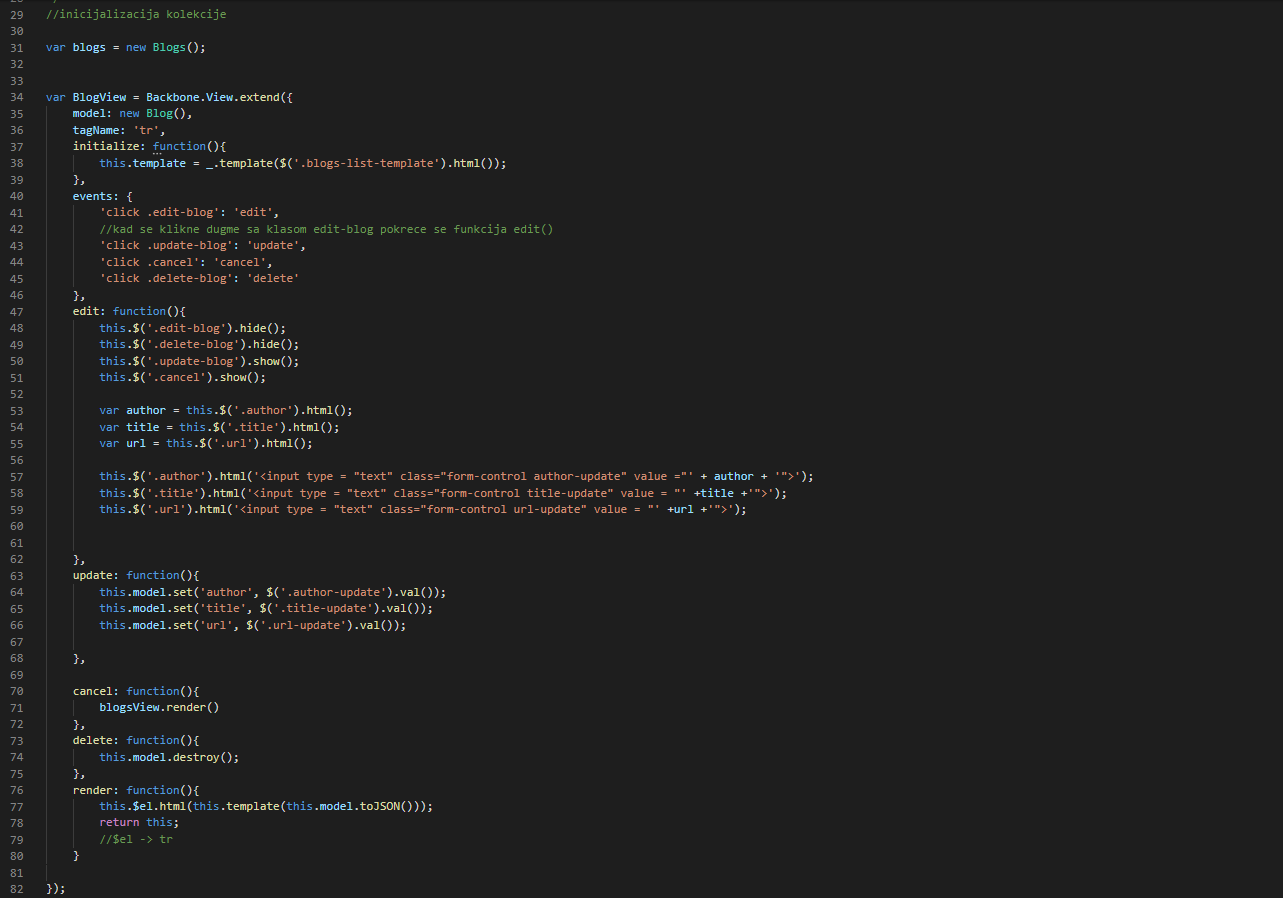
Slika 39-Početak naše aplikacije

Realizacijom ovih koraka imamo praznu kolekciju blogs, koju ćemo kasnije kako korisnik bude unosio svoje blogove popunjavati. Sledeće što preduzimamo je pravljenje dva glavna pogleda (View), od kojih će jedan imati zadatak prikazivanja kolekcije u vidu redova u našoj tabeli (spisku), dok će drugi view biti zadužen za prikaz svih elemenata tabele. Krenimo od drugog pogleda.



Slika 40-Pogled zadužen za prikaz tabele

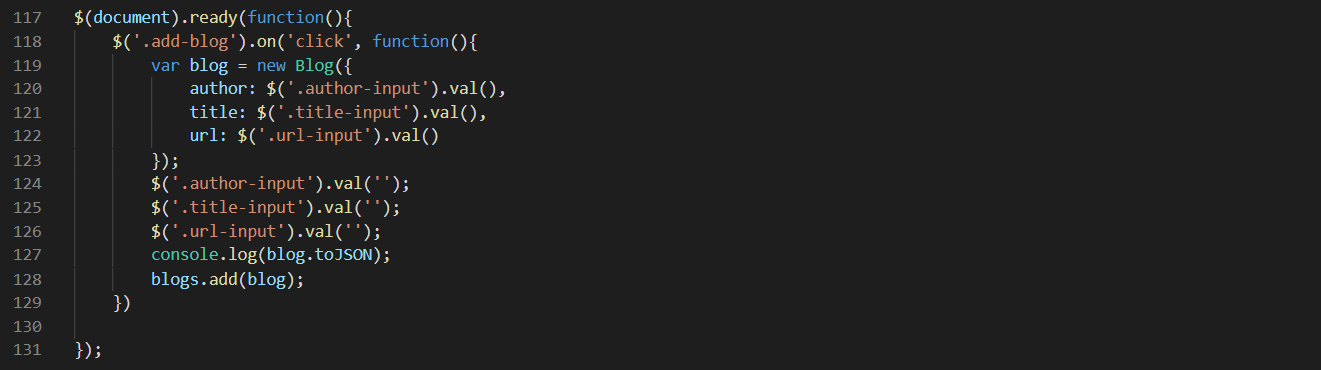
Elementi nad kojima ćemo vršiti izmene su elementi koji u dokumentu u kojem je naša skripta pozvana sadrže klasu .blogs-list. Iz sadržaja body dela naše aplikacije možemo da primetimo da je to element klase tbody i da je on primarno prazan. Pogled zadužen za prikaz tabele se sastoji od dve funkcije, a to su funkcije initialize i render. Prva funkcionalnost koju naša render funkcija obavlja, jeste pražnjenje tabele. Nakon pražnjenja sadržaja tabele, render funkcija pristupa ažuriranju iste sa novom kolekcijom blogova, tako što kroz celu kolekciju prolazi sa underscore funkcijom \_.each. Each funkcija radi po principu koji je sličan principu rada for petlje, u Java programskom jeziku. Za svaki element prosleđene kolekcija, funkcija each prvo prolazi kroz isti, a zatim se za nju poziva prosleđena funkcija. Prosleđena funckija formira pogled Backview, odnosno novi red tabele i ubacuje ga u element sa klasom blogs-list. Uzimajući u obzir da se dati postupak ponavlja za svaki element kolekcije, na taj način se pravi celokupna tabela. Initialize funkcija se pokreće pri instanciranju pogleda Blogsview, kao što je prethodno pomenuto. Obuhvata tri poziva on() Backbone funkcije. To je funkcija koja prima 3 parametra, od kojih prvi parametar predstavlja promenu/događaj (event), drugi parametar je funkcija koja se u slučaju realizacije tog događaja poziva, dok je treći parametar kontekst. U našem primeru, prvi poziv funkcije on se dešava u slučaju dodavanja modela u kolekciju (add), drugi poziv se realizuje ukoliko dođe do promene u kolekciji (change) i treći ujedno i poslednji poziv funkcije se dešava u slučaju brisanja modela iz kolekcije (remove). U svakom od data tri slučaja se poziva funckija render. Odnosno, svaka individualna promena kolekciji se odražava na celu tabelu. Na samom kraju se inicijalizuje naš pogled Blogsview.



Slika 41-Prvi pogled naše Backbone funkcije

Sada možemo da se vratimo na prvi pogled, koji je zadužen za prikazivanje kolekcije u formi redova u tabeli. Za razliku od prvog pomenutog pogleda ovaj pogled je dosta veći, međutim ne i nužno komplikovaniji. U ovom slučaju imamo definisana četiri event-a. Svi oni predstavljaju događaje čija je realizacija prouzrokovana nekim drugim događajem. Ukoliko se klikne (click) na element sa klasom .edit-blog (u ovom primeru je to dugme Edit), vrši se poziv funkcije edit. Edit funkcija započinje svoje izvršavanje tako sakrivanjem dva dugmeta Hide i Edit, a ujedno prikazuje dva dugmeta Cancel i Update, koja su inicijalno postavljena da se ne vide u HTML fajlu aplikacije. Zatim smešta podatke iz elemenata sa klasama .title, .author, .url u varijable title, author, url. Dalje, od tih pravi textfield-ove od tih klasa i u te iste textfield-ove stavlja naše definisane tri promenljive, nakon toga taj textfield označava sa klasama author-update, title-update, url-update. Funkcija delete briše naš model iz kolekcije. Dok funkcija update iz polja sa klasama koje imaju ekstenziju -update, uzima njihove vrednosti uz pomoć funkcije val a potom ih smešta u model u kojem se nalazimo. Svaka od ovih promena se odražava na pravu tabelu preko funkcije initialize, koja usled bilo kakve promene( change, add, delete) poziva render funkciju.

Približavamo se kraju same aplikacije. Korišćenjem jQuery funkcije ready, omogućeno je da se poslednji deo koda pozove tek onda kada se u potpunosti učita cela HTML stranica. Po njenom učitavanja i pritisku element-a sa klasom .add-blog (u datom primeru dugme Add), uzimaju se vrednosti iz input field-ova i stavljamo ih u našu tabelu, nakon čega praznimo text field-ove, kako bi oni bili spremni za sledeći unos podataka. Nakon toga, pristupamo ubacivanju blog-a u kolekciju blogs.



Slika 42-Kraj aplikacije

**Zaključak**

Backbone je više biblioteka nego framework koja pruža mogućnost pravljenja bogatijih web aplikacija. Podrazumeva korišćenje komponenti i njegovih predefinisanih metodi i svojstava.

Može se koristiti kao osnova za pravljenje sopstvenog framework-a, pa su tako primeri framwork-a koji su zasnovani na Backbone-u Aura, Geppetto, Marionette, Thorax, Chaplin.

Njegovi glavni konkurenti su Angular i Ember. Backbone ima manju zajednicu jer je nastao nešto kasnije u odnosu na svoje konkurente, a njegova prednost je veličina za preuzimanje, koja je kompresovana i minimizirana, i iznosi 6,4K.

Pogodan je za pravljenje prilagodljivih i brzih aplikacija, ima praktični radni okvir, jer ne koristi puno šablona, pa samim tim programeru daje slobodu. Kreator ovog framework-a, Jeremy Ashkenas je rekao: “Ako pišete puno šablona u Backbone-u, onda ga ne koristite kako treba“.

# **Literatura**

Preuzeto sa Backbone.js: http://backbonejs.org/

Preuzeto sa Underscore.js: https://underscorejs.org/

Preuzeto sa JQuery: https://jquery.com/

*Ajax*. Preuzeto sa JQuery: https://api.jquery.com/category/ajax/

*Angualar Single Page Applications (SPA): What are the Benefits?* (2018, Jun 18). Preuzeto sa Angular University: https://blog.angular-university.io/why-a-single-page-application-what-are-the-benefits-what-is-a-spa/

*Backbone.js Events*. Preuzeto sa Tutorials Point: https://www.tutorialspoint.com/backbonejs/backbonejs\_events.htm

Mandić, N. (2015, Jun 26). *Uvod u Backbone.js*. Preuzeto sa Pop Web Design: https://www.popwebdesign.net/popart\_blog/2015/06/uvod-backbone-js/

Osmani, A. (2013). *Developing Backbone.js Applications.* O'Reilly Media.

Rouse, M. (2014, Septembar). *RESTful API*. Preuzeto sa Search Microservices: https://searchmicroservices.techtarget.com/definition/RESTful-API

*Stack Overflow*. (2011, Mart 24). Preuzeto sa What Is The Purpose of Backbone.js?: https://stackoverflow.com/questions/5418369/what-is-the-purpose-of-backbone-js