

UNIVERZITET U BEOGRADU
MATEMATIČKI FAKULTET



**RADNI OKVIR EXT JS I NJEGOVA PRIMENA U
RAZVOJU VEB APLIKACIJE ZA ODREĐIVANJA
PUTANJA VOZILA**

Master rad

Aleksandar Ćurković

Mentor: prof. dr Vladimir Filipović
Matematički fakultet, Univerzitet u Beogradu

Članovi komisije: prof. dr Saša Malkov
Matematički fakultet, Univerzitet u Beogradu
doc. dr Aleksandar Kartelj
Matematički fakultet, Univerzitet u Beogradu

Sadržaj

1	Uvod	2
2	Radni okvir Ext JS	3
2.1	Istorijat	3
2.2	Poređenje sa drugim radnim okvirima	6
2.3	Arhitektura	8
2.4	Upravljanje podacima	9
2.4.1	Model pogleda	9
2.4.2	Skladište	10
2.4.3	Model	11
2.4.4	Proksi	13
2.4.5	AJAX	15
2.5	Korisnički interfejs	16
2.5.1	Životni ciklus	16
2.5.2	Kontejneri	17
2.5.3	Poravnanje	18
2.6	Komponente korisničkog interfejsa	25
2.6.1	Grid panel	26
3	Rutiranje	29
3.1	Rutiranje teretnih vozila	30
3.2	Dijkstrin algoritam	32
3.3	Prošireni model grafa u <i>OSRM</i> -u	34
3.4	Hijerarhijske kontrakcije	35
4	Dizajn Ext JS aplikacije za rutiranje	36
4.1	Neki detalji implementacije	40
4.2	OpenLayers	41
5	Zaključak	42

1 Uvod

Softverski razvoj veb aplikacija, odnosno razvoj korisničkih interfejsa za njih, u najvećoj meri je vezan za programski jezik *Javascript*. Zajedno sa *HTML*-om i *CSS*-om, dugo je predstavljao jedini način kreiranja veb sadržaja i to je period u kome su izazovi za programere koji su radili na toj vrsti softvera, vremenom postajali sve veći. Među najveće izazove svakako se ubrajaju nekompatibilnost između različitih veb pretraživača kao i napredak ostvaren na hardverskom planu koji, kao i danas, s jedne strane pruža nove mogućnosti ali sa druge, kreira veće i kompleksnije korisničke zahteve. Navedeni razlozi uslovisu su pojavu Javascript radnih okvira koji su ne samo pružili odgovor na pomenute izazove već i znatno olakšali i ubrzali razvoj veb aplikacija. U ovom radu predstavljen je su, i upoređene sa drugim najznačajnijim konkurentima, glavne karakteristike *Ext JS* radnog okvira i opisani različiti aspekti njegove upotrebe.

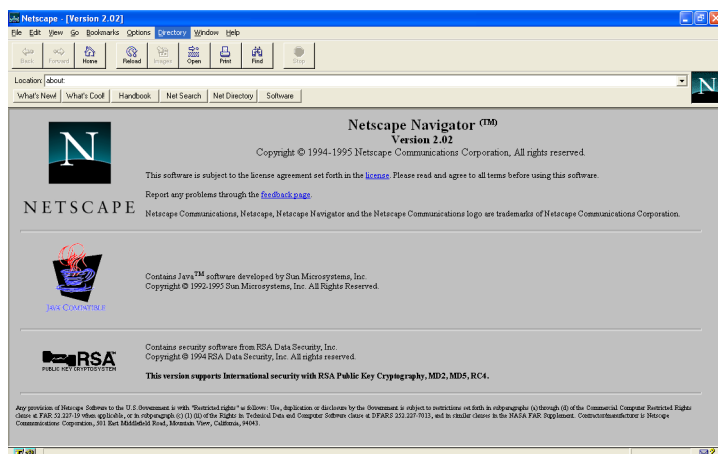
Aplikacija razvijena u okviru ovog master rada, dostupna na adresi <https://github.com/curkovical/masterApp>, svoju primenu bi najpre pronašla u sektoru kompanija koje se bave transportom, ali može biti od koristi svima kojima je potrebno da isplaniraju transport nekog tereta koji se meri u tonama. Glavna specifičnost u odnosu na druge aplikacije za rutiranje je bavljenje aspektom nosivosti mostova. Najčešće, u slučaju kada su u sumnji da težina vozila sa teretom koje prevozi prevazilazi zakonske norme, kompanije su dužne da se obrate nadležnim državnim agencijama koje im, ako je to moguće, izdaju dozvolu i propisuju rutu kojom mogu obaviti transport. U tom smislu, najbitnija namena ove aplikacije je određivanje rute kojom se transport može obaviti bez obraćanja državnim agencijama, uzimajući u obzir propisana zakonska ograničenja koja se odnose na nosivost mostova. Unapređenje koje bi znatno uvećalo šanse za širu realnu primenu aplikacije bi bila mogućnost određivanja da li je, i na koji način, prelazak datog teretnog vozila moguć za svaki most pojedinačno, u zavisnosti od njegovih karakteristika, ali bi to zahtevalo dosta dodatnih znanja iz oblasti građevinskog inženjerstva.

Podaci o mostovima korišćenim u aplikaciji služe za ilustraciju upotrebe, s obzirom da su javno dostupni[19], tako da u tom smislu treba napomenuti da su moguće različite modifikacije u zavisnosti od teritorije na kom se nalaze razmatrani mostovi, odnosno zakonske regulacije koja se odnosi na njihov prelazak.

2 Radni okvir Ext JS

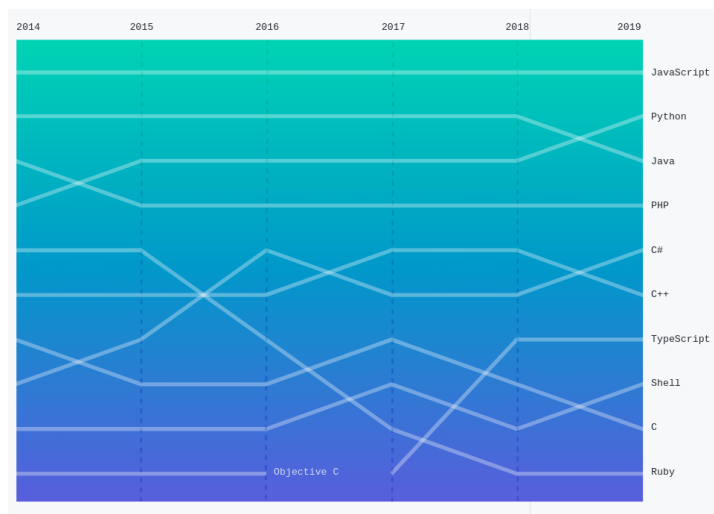
2.1 Istorijat

Sredinom 1995. godine kompanija *Netscape* je imala dominaciju na tržištu veb pretraživača i u cilju osnaživanja te pozicije (glavni konkurent bio je *Microsoft* sa *Internet Explorerom* a sledećih nekoliko godina činiće period poznat pod nazivom *rat pretraživača*) izbacila je prvu verziju veb pretraživača koji je podržavao dinamičke promene u izgledu i funkcionisanju veb stranice. Taj pretraživač je bio *Navigator* a programski jezik koji je te promene omogućio *Livescript*, kasnije preimenovan u *Javascript*.



Slika 1: Netscape Navigator 2.02

Prema rečima Brendana Ajka, kreatora Javascript-a, glavna motivacija za pisanje novog jezika (umesto korišćenja postojećeg) bila je ideja o razlikovanju dva glavna tipa tadašnjih veb programera: onih koji kreiraju komponente i one koji ih koriste. I dok je za prve bila namenjena *Java*, za potonje je bilo potrebno razviti jezik koji bi bio lak za korišćenje i koji bi im omogućio sklapanje elemenata i uređivanja njihove interakcije[28]. Kao što je dobro poznato, Javascript ne samo da je preuzeo dominaciju nad Javom i u prvom segmentu, već je postao jezik bez kog je veb programiranje teško zamislivo, i jedan od najpopularnijih programskih jezika uopšte.



Slika 2: Popularnost programskih jezika po zastupljenosti u GitHub repozitorijumima[13]

Ubrzo nakon pojave Javascript-a, Microsoft je implementirao svoju verziju jezika pod nazivom *JScript*, nakon čega su u kompaniji Netscape odlučili da standardizuju jezik obrativši se organizaciji za standardizaciju *Ecma International*. Prva edicija standarda ECMA-262, koji sadrži specifikaciju jezika *ECMAScript* (tako nazvanog zbog prava koje kompanija *Sun* imala nad rečju *Java*), izašla je u junu 1997. godine, a do trenutka pisanja rada izašlo ih je još 10, poslednja u junu 2020. godine- *ECMAScript 2020*.

Jedna od glavnih novina koju je Javascript doneo je interakcija korisnika sa veb stranicom a da to nužno ne podrazumeva komunikaciju sa serverom. Vremenom, korisnički interfejs ali i veb aplikacije uopšte, su postajale sve kompleksnije, kompatibilnost između različitih pretraživača postajala je sve izraženiji problem što je uz još neke faktore na kraju uslovalo pojavu Javascript biblioteka odnosno radnih okvira. Jedna od prvih takvih biblioteka je *jQuery*, nastala 2006. godine, koja je i danas veoma zastupljena- u trenutku pisanja koristi je 73.6% svih veb stranica[26].

Među glavnim prednostima korišćenja radnih okvira je to što u znatnoj meri olakšavaju izradu aplikacije jer sa sobom nose već gotova i pouzdana rešenja

za probleme koji se često susreću, te su programeri mnogo ranije u prilici da se usredsrede samo na one aspekte koji su specifični za datu aplikaciju. Danas postoji veliki broj Javascript radnih okvira tako da izbor nije nimalo lak ali neke opšte smernice pri izboru postoje. Jedna takva lista kriterijuma može se naći na veb stranici PHP radnog okvira *Symphony*[22]:

- popularnost i brojnost zajednice
- filozofija radnog okvira
- dugoročna održivost
- podrška
- tehnička rešenja
- sigurnost
- dokumentacija
- licenciranje
- dostupnost radnika sa potrebnim znanjem na tržištu
- utisak na osnovu korišćenja

Ono što svakako treba imati u vidu je da ne postoji jedno najbolje rešenje za sve slučajeve već da ono zavisi od okolnosti.

Počeci *Ext JS*-a vezani su za *Yahoo! User Interface (YUI)* biblioteku koju je *Džek Slokum* 2006. godine počeo da menja i nadopunjuje što je već sledeće godine rezultiralo izbacivanjem prve verzije *Ext JS*-a koji se u tom trenutku sastojao od nekih korisničkih komponenti kao što su dugmeta i tabele i načina njihovog raspoređivanja pomoću sistema poravnanja. Naravno, svaka sledeća verzija donosila je nešto novo, od obogaćivanja komponenti korisničkog interfejsa do kontroverznih izmena vezanih za licenciranje a prelazak sa jedne na drugu nije u svim slučajevima bio iste težine. Za izradu aplikacije u okviru ovog master rada korišćena je verzija otvorenog koda (GPLv3) 6.2.0, poslednja verzija u trenutku pisanja je 7.2.0.

Ext JS spada među najcelovitije Javascript radne okvire, u smislu da programerima koji ga koriste nudi sve što je potrebno za razvoj višeplatformskih aplikacija kompatibilnih sa svim popularnijim veb pretraživačima. Veliki broj integrisanih

komponenti korisničkog interfejsa predstavlja jednu od njegovih glavnih prednosti u odnosu na druge JS radne okvire. Treba napomenuti da kompanija koja je vlasnik samog radnog okvira- *Sencha*, nudi dodatne alate od kojih bi trebalo izdvojiti

- *Sencha Architect*, kojim se veb aplikacija može programirati u WYSIWYG režimu i
- *Sencha Themer*, kojim se kreiraju teme, takođe bez pisanja koda

2.2 Poređenje sa drugim radnim okvirima

U okviru razmatranja pitanja poređenja ovog radnog okvira sa drugim, pođimo od toga šta sama kompanija *Sencha*, navodi kao najvažnije prednosti[27]:

- veliki broj integrisanih komponenti korisničkog interfejsa
- ujednačen stil programiranja
- podrška za više platformi
- objektno-orijentisano programiranje odnosno dobra strukturiranost koda
- komercijalna podrška

Sa druge strane, kao najveće zamerke Ext JS radnom okviru mogu se izdvojiti:

- količina podataka odnosno performanse
- licenciranje

Tako je izvršeno poređenje količina podataka potrebnih za pokretanje aplikacije istih funkcionalnosti napisanih u Ext JS i *AngularJS* radnim okvirima. Nađeno je da aplikacija napisana u Ext JS-u koristi tri puta više podataka za pokretanje u odnosu na AngularJS aplikaciju (uz pretpostavku da ništa od sadržaja nije keširano, u suprotnom odnos je još više na strani AngularJS-a)[18], što se naravno, odražava na brzinu učitavanja aplikacije.

Za korišćenje Ext JS-a postoje komercijalne i GNU Opšta javna licenca (verzija 3) (eng. *General Public License(GPL) v3*)[21], što u najkraćem znači da, ukoliko se koristi GPL licenca, kreiran softver mora biti otvorenog koda. Dakle, ako se softver koristi u komercijalne svrhe potrebno je koristiti komercijalnu licencu koja se kupuje za svakog programera ponaosob. U poređenju sa drugim značajnim Javascript radnim okvirima kao što su *Angular*, *React* ili *Vue*,

	Ext JS	Angular	React	Vue
Prva zvanična verzija	2007	2016	2013	2014
Dvosmerno povezivanje podataka	✓	✓	✗	✓
Domenski specifičan jezik	✗	TypeScript	JSX	✗
Integrirano rutiranje	✓	✓	✗	✗
Virtual DOM	✓	✗	✓	✓
Brzina učenja	spora	spora	srednja	brza
Licenca	GPLv3	MIT	MIT	MIT

Tabela 1: Pregled karakteristika radnih okvira

koji su svi dostupni pod MIT licencom, opravdana je sumnja da ovo pitanje značajno utiče i na popularnost Ext JS-a. Pored načina licenciranja razlike između GPL i komercijalnih licenci ogledaju se i u tome što neki alati kao već pomenuti Sencha Themer i Sencha Architect, kao i paketi sa komponentama za vizualizaciju podataka, odnosno njihovo preuzimanje kao fajlova različitih formata (*pdf*, *csv*, itd.), nisu dostupni u okviru GPL licence. Isti je slučaj i sa dva radna okvira koja kompanija Sencha ima u svojoj ponudi a koji objedinjuju Ext JS i *Angular*, odnosno *React*:

- ExtAngular
- ExtReact

U pogledu sintakse ova dva radna okvira, glavna promena u odnosu na Angular i React, ogleda se u proširenju skupa *HTML* odnosno *JSX* oznaka koje se mogu upotrebiti, kao i njima odgovarajućih atributa koji predstavljaju ekvivalent (konfiguracionim) svojstvima klasa u Ext JS-u.

2.3 Arhitektura

MVC (Model-View-Controller) projektni uzorak može se objasniti upravo analogijom sa mapom i teritorijom: mapa je reprezentacija teritorije, ista teritorija može se predstaviti pomoću više mapa. Na isti način, postoji više načina za predstavljanje istih podataka. Takođe, za prezentaciju podataka bi trebalo da bude irelevantno na koji su način podaci dobavljeni. Imajući u vidu te različite aspekte aplikacije ona se ovim projektnim uzorkom deli na 3 dela:

- model čine strukturirani podaci koje aplikacija treba da prikaže
- pogled predstavlja reprezentaciju tih podataka
- kontroler se bavi interakcijom među njima kao i reagovanjem na korisničke zahteve

Ext JS podržava MVC od verzije 4, a podrška za MVVM (eng. *Model-View-ViewModel*) uvedena je od verzije 5. Još jedna novina u toj verziji je kontroler pogleda (eng. *ViewController*), koji za razliku od kontrolera u verziji 4 ne radi na nivou cele aplikacije već na nivou pogleda, tako da je odnos između pogleda i njemu pripadajućeg kontrolera 1-1. Na taj način prevazilazi se jedna od bitnijih mana prethodnog pristupa: u velikim aplikacijama nije redak slučaj da različiti timovi rade na različitim delovima aplikacije koji samostalno mogu raditi ispravno a da do grešaka u radu dolazi prilikom integracije koda. Jedan od uzroka može biti to što kontroler reaguje na događaje pogrešnog pogleda ili menja delove pogleda koje ne bi trebao jer kontroler na nivou aplikacije može da vidi sve njene delove te prilikom selekcije onog što treba izmeniti lako može doći do greške. Pored jednostavnijeg odnosa između pogleda i kontrolera, kontroler pogleda zadržava mogućnost delovanja na komponente koje su niže u hijerarhiji od pogleda kome je taj kontroler namenjen.

Pored kontrolera pogleda, pogled ima pripadajući model pogleda: to je klasa koja sadrži podatke vezane za pogled i jedna od glavnih prednosti koje pruža je povezivanje podataka (eng. *data binding*) čime se smanjuje količina posla o kojoj kontroler treba da vodi računa. Komponente imaju svojstvo *bind* kojim vrednosti svojih drugih svojstava mogu povezati sa vrednostima u modelu pogleda, tako da se vrednost tih svojstava promeni kad god dođe do promene u modelu pogleda. Takođe, i ovde je zadržana hijerarhijska struktura podataka u smislu da sve komponente-deca pogleda mogu da vide podatke koji se nalaze u njegovom modelu pogleda.

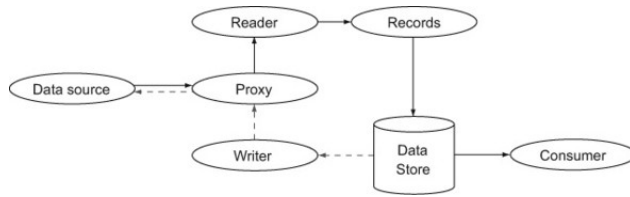
2.4 Upravljanje podacima

2.4.1 Model pogleda

Kao što je već istaknuto, model pogleda (eng. *ViewModel*) je klasa pridružena pogledu kojoj je glavna funkcija da vodi računa o podacima koje pogled (i eventualno drugi pogledi koje on može sadržati) i njegove komponente koriste. Da bi se bliže razjasnilo šta se pod tim podrazumeva i kako se to ostvaruje može se razmotriti primer iz aplikacije:

```
1 Ext.define('masterApp.view.main.MainModel', {
2   extend: 'Ext.app.ViewModel',
3   alias: 'viewmodel.main',
4   data: {
5     name: 'masterApp',
6     startCmb: {
7       data: {
8         name: 'start'
9       }
10    },
11    endCmb: {
12      data: {
13        name: 'end'
14      }
15    },
16    vehicleType: 'single',
17    swapDisabled: true,
18  },
19  stores: {
20    startCmbStore: {
21      type: 'comboStore'
22    },
23    endCmbStore: {
24      type: 'comboStore'
25    }
26  },
27  formulas: {
28    swapDisabled: function(get){
29      if (get('startCmb.value') == '' || get('endCmb.value')
30      == '')
31        return true;
32        return false;
33    }
34  });
```

Listing 1: Model pogleda korišćen u aplikaciji



Slika 3: Prikaz protoka podataka u skladištu[8]

Prva dva svojstva nisu vezana samo za model pogleda već se definišu za svaku klasu i određuju:

- **extend** - klasu koju novokreirana klasa nasleđuje
- **alias** - skraćeni naziv klase, kako bi se ona lakše referencirala u drugim delovima aplikacije

Sledeća dva svojstva karakteristična su za model pogleda i zajedno sa *formulas* čine grupu najbitnijih svojstava ove klase:

- **data** je objekat u koji se dodaju željeni podaci
- **stores** sadrži konfiguracije skladišta koja predstavljaju kolekciju objekata na klijentskoj strani
- **formulas** sadrži funkcije koje vraćaju obrađene podatke iz modela pogleda

2.4.2 Skladište

Skladište (eng. *Store*) ima ulogu upravljanja podacima koje koriste komponente korisničkog interfejsa. Najbitnija svojstva koja određuju skladište su

- model
- proxy

Nad skladištem su definisane metode za dodavanje, brisanje, pretragu itd. Metode *add* i *remove* manipulišu sa lokalno sadržanim podacima dok se pozivom metode *sync* izmene na tom nivou prenose dalje putem konfigurisanog proksija. Svojsvom *autoSync* u definiciji skladišta taj proces se može automatizovati tako da se ta komunikacija dešava posle svake izmene nad podacima u skladištu.

```

1 Ext.define('masterApp.store.comboStore', {
2   extend: 'Ext.data.Store',
3   alias: 'store.comboStore',
4   autoLoad: false,
5   proxy: {
6     type: 'ajax',
7     reader: {
8       type: 'json'
9     }
10  }
11 });

```

Listing 2: Primer skladišta korišćenog u aplikaciji

U okviru konfiguracije moguće je definisati način na koji će podaci biti sortirani ili filtrirani. Npr., uzmimo da skladište sadrži objekte modela *Bridge*, definisanim u poglavlju 3.2.3. Tada bi se sortiranje i filtriranje objekata moglo definisati dodavanjem sledećih svojstava u definiciju skladišta:

```

1 sorters: [{
2   property: 'bridge_name',
3   direction: 'ASC'
4 }, {
5   property: 'rating',
6   direction: 'DESC'
7 }],
8 filters: {
9   property: 'closed',
10  value: 'No'
11 }

```

Listing 3: Sortiranje i filtriranje u skladištu

2.4.3 Model

Modelom se definiše vrsta podataka koju *store* sadrži, to je klasa čije se promenljive definišu u okviru svojstva *fields*, a može imati i svoje metode.

```

1 Ext.define('Bridge', {
2   extend: 'Ext.data.Model',
3   fields: [
4     {name: 'bridge_name', type: 'string'},
5     {name: 'rating', type: 'int'},
6     {name: 'latitude', type: 'number'},
7     {name: 'longitude', type: 'number'},
8     {name: 'yearRated', type: 'int'},
9     {name: 'closed', type: 'string'}

```

```

10     ],
11
12     changeRating: function(int newRating) {
13         this.set('rating', newRating);
14     }
15 });

```

Listing 4: Primer modela

Mogući tipovi promenljivih su:

- auto
- string
- int
- number
- boolean
- date

Moguće je i definisati posebne tipove podataka ako predefinisani nisu dovoljno deskriptivni u datom slučaju.

Još jedna važna mogućnost u okviru definisanja modela predstavljaju validatori. Njima se nad datim poljima modela mogu definisati kriterijumi koje njihove vrednosti treba da zadovolje. U slučaju gorenavedenog primera to bi se postiglo dodavanjem npr. sledećeg objekta u konfiguraciju modela:

```

1  validators: {
2      bridge_name: 'presence',
3      closed: { type: 'inclusion', list: ['Yes', 'No'] },
4      yearRated: { type: 'range', min: 1900, max: 2020 }
5  }

```

Listing 5: Validacija vrednosti

Na ovaj način može se osigurati da vrednosti instanci datog modela zadovoljavaju zadate kriterijume. Predefinisani su sledeći tipovi validatora:

- oni koji se odnose na vreme u koje spadaju date, time, datetime
- length
- range

- format
- number
- inclusion i exclusion
- presence

Još jedan važan aspekt koji u opisu modela ne bi trebalo izostaviti je mogućnost definisanja odnosa koje dati model ima sa drugim modelima. Neke od prednosti definisanja asocijacija nad modelima su automatsko generisanje metoda za dohvatanje i postavljanje podataka kao i dohvatanje ugnežđenih podataka sa servera u jednom zahtevu[2]. Mogući tipovi asocijacija između modela su:

- one to one
- many to one
- many to many

2.4.4 Proksi

Proksi (eng. *Proxy*) predstavlja način na koji skladište pribavlja podatke i čuva izmene. Proksi može biti definisan i nad korišćenim modelom podataka, dok je u slučaju aplikacije razvijene u okviru master rada definisan u okviru skladišta. Dve glavne podvrste proksija su klijentski i serverski. Klijentski proksiji obuhvataju sledeća tri tipa:

- local storage
- session storage
- memory

Tipovi proksija koji se ubrajaju u serverske su:

- ajax
- jsonP
- rest
- direct

Kao što se može videti u definiciji skladišta u kodu 2, konfigurisan je format podataka koji proksi treba da očekuje. Isto tako se može konfigurisati i format podataka koje proksi treba da šalje serveru. Pored formata mogu se konfigurisati i url i načini formiranja upita koji se šalju serveru dodavanjem sledećih svojstava u konfiguraciju za proksi:

```
1 url: 'https://server.org/search',
2 api: {
3   create : 'https://server.org/new',
4   update  : 'https://server.org/update',
5   destroy : 'https://server.org/delete'
6 },
7 actionMethods:{
8   create: 'POST',
9   read: 'GET',
10  update: 'POST',
11  destroy: 'POST'
12 }
```

Listing 6: Proksi konfiguracija

Na ovaj način definisano je da se npr. pozivom *add* metode nad skladištem pošalje POST zahtev na adresu *https://server.org/new*.

Vrednosti navedene u ovom primeru služe za ilustraciju, u slučaju skladišta u razvijenom programskom kodu deo koji se odnosi na pribavljanje podataka implementiran je pomoću funkcije (u ovom primeru jedini potreban metod je metod za pribavljanje podataka). Pomenuta funkcija implementirana je na sledeći način: nakon što korisnik pritisne taster *enter*, pomoću vrednosti koja je dotad unesena u dato polje, formira se HTTP zahtev ka *Nominatim* servisu, nakon čijeg se odgovora otvara padajući meni sa dobijenim vrednostima:

```
1 locationEntered: function(combo, e){
2   controller = this;
3   // e.HOME, e.END, e.PAGE_UP, e.PAGE_DOWN,
4   // e.TAB, e.ESC, arrow keys: e.LEFT, e.RIGHT, e.UP, e.DOWN
5   if (e.getKey() == e.ENTER) {
6     let rawValue = combo.getRawValue().trim();
7     rawValue = rawValue.replace(/ +/g, '+');
8     var query = 'https://nominatim.openstreetmap.org/search?q='
9     ' + rawValue + '&format=json';
10    var store = combo.getStore();
11    store.getProxy().setUrl(query);
12    store.load({
13      callback: function(){
14        combo.expand();
15      }
16    });
17  }
```



```
15     });
16   }
17 }
```

Listing 7: Reagovanje na događaj i slanje zahteva

2.4.5 AJAX

AJAX (Asynchronous JavaScript And XML) je tehnika za komunikaciju između klijentske i serverske strane u veb aplikacijama. U Ext JS-u postoji singleton klasa, *Ext.Ajax* koja obuhvata komunikaciju sa serverom tog tipa. Zahtevi ka serveru šalju se pomoću *request* metode koja kao argument prima objekat sa različitim svojstvima među kojima u najbitnije spadaju:

- **url** određuje adresu na koju se zahtev šalje
- **method** određuje HTTP metod koji se koristi za slanje zahteva
- **success** je funkcija u kojoj se definiše reakcija na uspešan odgovor servera, tačnije na odgovor koji ima HTTP status 200-299
- **failure** je funkcija u kojoj se definiše reakcija na neuspešan odgovor servera, tačnije na odgovor koji ima bilo koji HTTP status koji označava grešku
- **callback** je funkcija koja se poziva bez obzira na HTTP status odgovora pristiglog od servera

Primer takvog zahteva koji u razvijenoj aplikaciji služi za dobavljanje ruta nad unesenim lokacijama:

```
1 Ext.Ajax.request({
2   url: Ext.String.format('{0}{1}{2}', 'http://localhost:5000/
3   route/v1/driving/', coordsForService),
4   method: 'GET',
5   cors: true,
6   useDefaultXhrHeader: false,
7   disableCaching: false,
8   scope: this,
9   unauthorized: true,
10  params: {
11    alternatives: true,
12    geometries: 'geojson',
13    overview: 'full'
14  },
15  success: function(response, opts) {
```

```

15     var routeData = Ext.JSON.decode(response.responseText);
16     callback(routeData);
17   },
18   failure: function(response, opts) {
19     console.log(response);
20   }
21 });

```

Listing 8: AJAX zahtev korišćen u aplikaciji

2.5 Korisnički interfejs

Korisnički interfejs Ext JS aplikacije čine komponente. Kontejner (eng. *Container*) je posebna vrsta komponenti koja se od ostalih razlikuje po tome što u sebi može sadržati druge komponente[2]. Način na koje će biti raspoređene te sadržane komponente definiše se poravnanjem (eng. *layout-om*).

2.5.1 Životni ciklus

Svaka komponenta korisničkog interfejsa ima svoj životni ciklus. Ključne faze tog ciklusa su:

- inicijalizacija
- renderovanje
- destrukcija

Svaka komponenta nasleđuje klasu *Ext.Component* koja vodi računa o svim procesima koji se odvijaju u okviru tih faza. Cilj faze inicijalizacije je da se komponenta kreira prema zadatoj konfiguraciji[1]. To obuhvata više stvari među kojima su najbitnije: definisanje reakcija na događaje zajedničke za sve komponente (renderovanje, prikaz itd.), dodavanje jedinstvenog broja (ID-a) komponenti tako da je ona dostupna od strane *Ext.ComponentManager* singleton klase, izvršavanje *initComponent* funkcije (koja se može shvatiti kao konstruktor klase, što znači da se u okviru nje može definisati logika koja bi trebalo da se izvrši prilikom svakog instanciranja komponente), inicijalizacije definisanih *plugin*-ova nad komponentom itd.

Početak faze renderovanja je konfigurabilan, odnosno ukoliko u definiciji komponente postoji svojstvo *renderTo*, faza renderovanja počinje odmah nakon faze inicijalizacije. Ako to svojstvo nije dodato u konfiguraciju, nad instancom se može pozvati

render metoda, odnosno renderovanje se može izvršiti onog trenutka kad se vrši i renderovanje komponente u okviru koje se komponenta nalazi. Cilj ove faze je da se komponenta doda u HTML DOM i pored tog još neki koraci koje ova faza obuhvata su: podešavanja vezana za to da li je komponenta plivajuća (eng. *floating*), eventualna *css* podešavanja u vezi toga da li je kursor nad komponentom, inicijalizacija sadržaja komponente, upravljanje događajima vezanim za renderovanje i na kraju, podešavanja vezana za to da li bi komponenta trebala da bude sakrivena ili isključena (eng. *disabled*).

U fazi destrukcije ciljevi su da se komponenta ukloni iz HTML DOM-a, izbrišu funkcije koje reaguju na događaje vezane za nju i oslobodi zauzeta memorija[1]. Kao i prethodne i ova faza vodi računa o događajima odnosno funkcijama vezanim za nju, uklanjaju se reference iz komponente-roditelja (ako ona postoji) odnosno `Ext.ComponentManager` klase.

2.5.2 Kontejneri

Klasa *Ext.container.Container* predstavlja baznu klasu za sve komponente koje u sebi mogu da sadrže druge komponente. Postoji više klasa koje je nasleđuju ali one koje su najčešće u upotrebi su:

- `Ext.panel.Panel`
- `Ext.tab.Panel`
- `Ext.form.Panel`
- `Ext.window`
- `Ext.Viewport`

Svaka od tih komponenti ima svojstvo *items* u okviru kojeg se definišu komponente koje kontejnerska komponenta treba da sadrži. Pre navođenja i prolaženja kroz primere treba napomenuti da svaki objekat u nizu *items* sadrži svojstvo *xtype*. U najkraćem, to je svojstvo koje nam omogućuje da prilikom instanciranja komponenti ne moramo da koristimo puno ime klase već skraćeno, što sem manje dužine koda donosi dodatni benefit: alternativa svojstvu *xtype* je korišćenje *Ext.create* metode koja instancira komponente bez obzira da li su korisniku potrebne, tako da aplikacija bez korišćenja svojstva *xtype* koristi više memorije i potrebno joj više vremena prilikom učitavanja. Na primer, umesto:

```

1 items: [
2   Ext.create('Ext.form.field.Text', {
3     fieldLabel: 'Name'
4   }),
5   Ext.create('Ext.form.field.Text', {
6     fieldLabel: 'Subject'
7   }),
8   Ext.create('Ext.form.field.TextArea', {
9     fieldLabel: 'Text'
10  })
11 ]

```

Listing 9: Instanciranje komponenti bez svojstva xtype

standardizovan i optimalniji način definisanja komponenti je:

```

1 items: [
2   {
3     xtype: 'textfield',
4     fieldLabel: 'Name'
5   },
6   {
7     xtype: 'textfield',
8     fieldLabel: 'Subject'
9   },
10  {
11    xtype: 'textarea',
12    fieldLabel: 'Text'
13  }
14 ]

```

Listing 10: Instanciranje komponenti pomoću svojstva xtype

Prilikom definisanja novih komponenti svakako treba imati u vidu da svojstvo *xtype* ima globalni opseg tako da treba voditi računa o tome da se *alias* novokreirane klase razlikuje od postojećih.

2.5.3 Poravnanje

Poravnanje određuje raspored komponenti u okviru date kontejnerske komponente. Ako on nije definisan, podrazumevana vrednost je *auto* što znači da će komponente biti raspoređene jedna nakon druge. Jedna od glavnih motivacija za uvođenje sistema poravnanja je bila nekompatibilnost CSS implementacija između različitih veb pretraživača, i iako je vremenom taj problem postao manje izražen i dalje je, pogotovo s obzirom da se i dalje koriste starije verzije veb pretraživača, na njega potrebno obratiti pažnju prilikom definisanja poravnanja pomoću

CSS-a. Sistem poravnanja Ext JS-a preuzima brigu o tome a s druge strane omogućava programeru jednostavan i fleksibilan način definisanja rasporeda komponenti. Postoji dosta vrsta poravnanja, ali među najčešće korišćene spadaju:

- absolute
- accordion
- border
- fit
- hbox
- vbox

Apsolutno poravnanje (eng. *absolute layout*) određeno je time što je kontejnerska komponenta shvaćena kao dvodimenzionalni koordinatni sistem u okviru koje se pozicija komponente određuje dodavanjem x i y svojstava. Takođe, to je potklasa *Ext.layout.container.Anchor* klase koja omogućuje da se veličina komponente-dece određuje na osnovu dimenzija komponente-roditelja dodavanjem svojstva *anchor*. U okviru tog svojstva mogu se definisati dve vrednosti od kojih prva predstavlja širinu a druga visinu komponente i mogu biti izražene ili celim brojem i u tom slučaju predstavljaju odstupanje (eng. *offset*), ili procentom kojim se dimenzije komponente određuju u odnosu na dimenzije roditelj komponente.

Sledećim segmentom programskog koda definiše se panel za prijavu korisnika sa apsolutnim poravnanjem, tako da se pozicija i veličina polja za unos i njihovih oznaka i dugmeta za prijavu određuju koristeći pomenuta svojstva, dok je izgled dobijenog panela prikazan na slici 4:

```
1 Ext.create('Ext.form.Panel', {
2   title: 'Absolute Layout',
3   width: 400,
4   height: 200,
5   layout: 'absolute',
6   items: [{
7     x: 10,
8     y: 10,
9     xtype: 'label',
10    text: 'username:'
11  }, {
```

```

12     x: 80,
13     y: 10,
14     xtype: 'textfield',
15     name: 'username',
16     anchor: '90%'
17 }, {
18     x: 10,
19     y: 40,
20     xtype: 'label',
21     text: 'password:'
22 }, {
23     x: 80,
24     y: 40,
25     xtype: 'textfield',
26     name: 'password',
27     anchor: '90%'
28 }, {
29     x: 150,
30     y: 100,
31     xtype: 'button',
32     text: 'Login',
33     anchor: '-150 -20'
34 }],
35     renderTo: Ext.getBody()
36 });

```

Listing 11: Panel sa apsolutnim poravnanjem

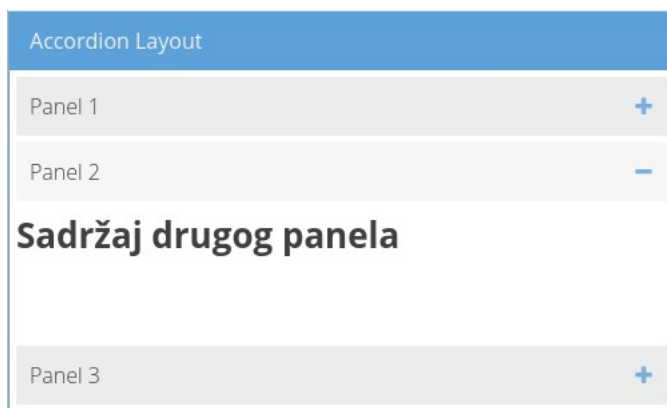
Slika 4: Apsolutno poravnanje

Poravnanje harmonike (eng. *accordion layout*) podrazumeva da su komponente koje data komponenta treba da sadrži klase *Ext.panel.Panel* ili njene potklase, i omogućava da jedna od tih komponenti bude proširena dok je od ostalih komponenti prikazano samo njihovo zaglavlje (eng. *header*). Upotreba datog poravnanja ilustrovana je sledećim primerom, kojim se definiše panel koji u

sebi sadrži tri panela čiji se sadržaj prikazuje nakon klika na njegovo zaglavlje ili + dugme, kao što je prikazano na slici 5.

```
1 Ext.create('Ext.panel.Panel', {
2   title: 'Accordion Layout',
3   width: 500,
4   height: 300,
5   border: true,
6   layout: {
7     type: 'accordion',
8     animate: true
9   },
10  items: [{
11    title: 'Panel 1',
12    html: 'Panel 1'
13  }, {
14    title: 'Panel 2',
15    html: '<h1>Sadržaj drugog panela</h1>'
16  }, {
17    title: 'Panel 3',
18    html: 'Panel 3'
19  }],
20  renderTo: Ext.getBody()
21 });
```

Listing 12: Panel sa poravnanjem harmonike



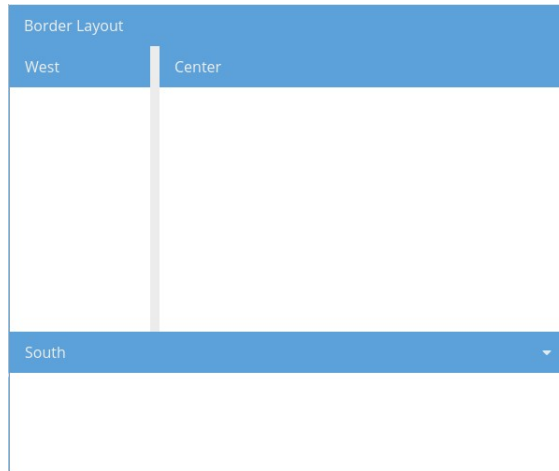
Slika 5: Poravnanje harmonike

Poravnanje sa granicama (eng. *border layout*) deli prostor u 5 regiona: *north*, *west*, *south*, *east* i *center*, od kojih svi sem *center* mogu ostati nedefinisani.

Takođe, moguće je dodati mogućnost promene veličine odnosno prikrivanja regiona od strane korisnika a podrazumevana vrednost svojstva *xtype* sadržanih komponenti je *panel*. Sledeći primer koji služi kao ilustracija prikazuje panel sa poravnanjem sa granicama koji sadrži tri regiona, dobijeni rezultat prikazan je na slici 6.

```
1 Ext.create('Ext.panel.Panel', {
2     width: 600,
3     height: 500,
4     title: 'Border Layout',
5     layout: 'border',
6     border: true,
7     items: [ {
8         xtype: 'panel',
9         title: 'Center',
10        region: 'center',
11        xtype: 'panel',
12        layout: 'fit'
13    }, {
14        title: 'West',
15        region: 'west',
16        width: 150,
17        split: true,
18        layout: 'vbox'
19    }, {
20        title: 'South',
21        region: 'south',
22        height: 100,
23        collapsible: true,
24        titleCollapse: true
25    }],
26    renderTo: Ext.getBody()
27 });
```

Listing 13: Panel sa poravnanjem sa granicama



Slika 6: Poravnanje sa granicama

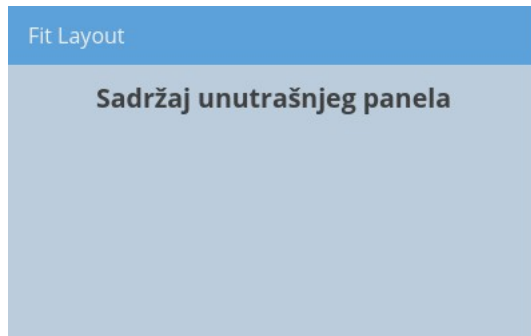
Poravnanje sa uklapanjem (eng. *fit layout*) podrazumeva da kontejner komponenta sadrži samo jednu komponentu (moguće je dodati i više tako da svaka od njih zauzima jednak prostor ali to nije preporučljivo) koja će u okviru nje zauzimati sav mogući prostor. Prilikom promene veličine kontejnerske komponente i dimenzije deteta-komponente će se menjati tako da i dalje zauzima sav raspoloživi prostor. Upotreba je ilustrovana sledećim primerom, u kom unutrašnji panel zauzima sav raspoloživi prostor u okviru spoljašnjeg:

```

1 Ext.create('Ext.panel.Panel', {
2     title: 'Fit Layout',
3     width: 400,
4     height: 250,
5     layout: 'fit',
6     items: {
7         xtype: 'panel',
8         html: '<h2>Sadržaj unutrašnjeg panela</h2>',
9         bodyStyle: {
10            'background': '#bcd',
11            'text-align': 'center'
12        }
13    },
14    renderTo: Ext.getBody()
15 });

```

Listing 14: Panel sa poravnanjem sa uklapanjem



Slika 7: Poravnanje sa uklapanjem

Poravnanje horizontalne i vertikalne kutije (označeno sa *hbox* i *vbox*) omogućava horizontalno, odnosno vertikalno, postavljanje dece-komponenti jedno do drugog. Količina prostora koje će dete komponenta zauzeti određuje se svojstvom *flex*. Na sledećem primeru prikazana su oba poravnanja: spoljašnji panel sa poravnanjem horizontalne kutije sadrži tri panela koji horizontalno redom zauzimaju $\frac{2}{5}$, $\frac{1}{5}$ i $\frac{2}{5}$ raspoloživog prostora, od kojih je nad prvim panelom definisano poravnanje vertikalne kutije tako da dva panela koja on sadrži vertikalno zauzimaju $\frac{3}{4}$ odnosno $\frac{1}{4}$ prostora namenjenog njemu.

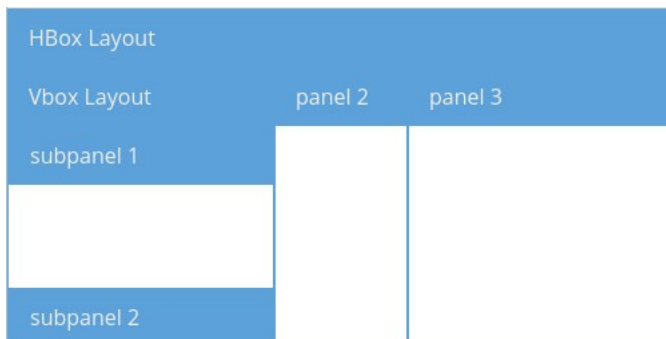
```
1 Ext.create('Ext.Panel', {
2   width: 500,
3   height: 250,
4   title: "HBox Layout",
5   defaults: {
6     border: 'true'
7   },
8   layout: {
9     type: 'hbox',
10    align: 'stretch'
11  },
12  renderTo: Ext.getBody(),
13  items: [{
14    xtype: 'panel',
15    title: 'Vbox Layout',
16    flex: 2,
17    layout: {
18      type: 'vbox',
19      align: 'stretch'
20    },
21    items: [{
22      xtype: 'panel',
```

```

23         title: 'subpanel 1',
24         flex: 3
25     }, {
26         xtype: 'panel',
27         title: 'subpanel 2',
28         flex: 1
29     }
30 ], {
31     xtype: 'panel',
32     title: 'panel 2',
33     flex: 1
34 }, {
35     xtype: 'panel',
36     title: 'panel 3',
37     flex: 2
38 }
39 });

```

Listing 15: Paneli sa poravnanjem horizontalne i vertikalne kutije



Slika 8: Poravnanje horizontalne i vertikalne kutije

2.6 Komponente korisničkog interfejsa

Ext JS nudi veliki broj gotovih komponenti korisničkog interfejsa kao što su *grid* (koji predstavlja tabelu sa dosta dodatnih mogućnosti koje se mogu konfigurirati), različite vrste grafikona, paneli koji treba da prikažu podatke sa drvolikom strukturom (*tree panel*), kalendari itd. Predstavljanje svih njih daleko bi premašilo okvire ovog rada ali će u svrhu ilustracije primene, biti opisana jedna od najčešće korišćenih i razvijanih od početka rada na samom radnom okviru, a to je *grid panel*.

2.6.1 Grid panel

Namena grid panela je tabelarni prikaz podataka, ali ono što izdvaja grid panel u odnosu na druge veb prikaze tog tipa je mogućnost konfigurisanja različitih dodatnih mogućnosti kao što su sortiranje i filtriranje vrednosti u tabeli, izmena vrednosti ćelija tabele, grupisanje redova odnosno zaglavlja kolona, proširivanje željenih redova tabele, određivanje načina prikaza vrednosti u pojedinačnim kolonama, dodavanje dodatnog reda u tabeli koji sadrži vrednost funkcije definisane nad datom kolonom (predefinisane funkcije su nalaženje minimalne i maksimalne vrednosti, računanje sume, proseka i broja redova u datoj koloni) i još mnoge druge.

Imajući u vidu sve te dodatne mogućnosti koje ova komponenta nudi, potrebno je napomenuti svojstva koje je u praktičnoj primeni neophodno definisati a to su *store* i *columns*. Kao što je već istaknuto, *store* određuje način na koji komponenta pribavlja podatke koje komponenta koristi u svom radu, a pomoću svojstva *columns* se određuju kolone koje dati grid treba da sadrži zajedno sa svim njenim dodatnim konfiguracijama. Od vrste podataka koje kolona sadrži zavisi i njen prikaz, tako da postoji nekoliko klasa koje radni okvir nudi za tu namenu od kojih su neke:

- **Ext.grid.column.Action** namenjena je za kolone koje sadrže ikonice i odgovarajuće reagovanje kad se na njih klikne
- **Ext.grid.column.Check** definiše kolonu koja sadrži *checkbox* za svaki red u grid-u
- **Ext.grid.column.Date** je namenjena za kolone koje trebaju da prikažu datum odnosno vreme; pomoću svojstva *format* određuje se željeni format prikaza
- **Ext.grid.column.Template** je klasa kojom se definiše obrazac prikaza datog modela podataka

Sledeći primer prikazuje konfiguraciju grid panela i njemu pripadajućeg skladišta. Dok se u okviru definicije skladišta nalaze same podaci i njihov opis (naziv i tip), definicija grid panela sadrži način prikaza tih podataka, kao i dodatan red koji prikazuje prosečnu vrednost nad vrednostima u koloni *rating*:

```
1 Ext.create('Ext.data.Store', {
2   alias: 'store.bridge',
3   storeId: 'bridgeStore',
4   fields: [
```

```

5     { name: 'name', type: 'string' },
6     { name: 'inspectionDate', type: 'date' },
7     { name: 'loadLimit', type: 'string' },
8     { name: 'closed', type: 'bool' },
9     { name: 'rating', type: 'float' }
10  ],
11  data: [{
12      name: 'Missouri Pacific Railroad',
13      inspectionDate: '2015-06-03',
14      loadLimit: '25-40',
15      closed: false,
16      rating: 6.5
17  }, {
18      name: 'O\Hearne Avenue',
19      inspectionDate: '2014-08-13',
20      loadLimit: '15-25',
21      closed: true,
22      rating: 2.9
23  }, {
24      name: 'Grand Louis Bayou',
25      inspectionDate: '2013-08-07',
26      loadLimit: '20-35',
27      closed: false,
28      rating: 7.1
29  }, {
30      name: 'Crescent City Connection',
31      inspectionDate: '2011-05-13',
32      loadLimit: '25-40',
33      closed: false,
34      rating: 8.4
35  }
36  ]});
37
38  Ext.create('Ext.grid.Panel', {
39      title: 'Bridges',
40      store: Ext.getStore('bridgeStore'),
41      border: true,
42      features: {
43          ftype: 'summary'
44      },
45      columns: [{
46          text: 'Name',
47          dataIndex: 'name',
48          flex: 3
49      }, {
50          text: 'Inspection date',
51          xtype: 'datecolumn',
52          format: 'd.m.Y.',
53          dataIndex: 'inspectionDate',
54          flex: 2

```

```

55     }, {
56         text: 'Load limit',
57         dataIndex: 'loadLimit',
58         flex: 2
59     }, {
60         text: 'Closed',
61         xtype: 'checkcolumn',
62         dataIndex: 'closed',
63         flex: 1
64     }, {
65         dataIndex: 'rating',
66         text: 'Rating',
67         summaryType: 'average',
68         summaryRenderer: function(value, summaryData, dataIndex) {
69             return Ext.String.format('average: {0}', value);
70         },
71         flex: 2
72     }],
73     width: 650,
74     renderTo: Ext.getBody()
75 });

```

Listing 16: Konfiguracija grid panela sa pripadajućim skladištem

Bridges				
Name	Inspection date	Load limit	Closed	Rating
Missouri Pacific Railroad	03.06.2015.	25-40	<input type="checkbox"/>	6.5
O'Hearne Avenue	13.08.2014.	15-25	<input checked="" type="checkbox"/>	2.9
Grand Louis Bayou	07.08.2013.	20-35	<input type="checkbox"/>	7.1
Crescent City Connection	13.05.2011.	25-40	<input type="checkbox"/>	8.4
				average: 6.225

Slika 9: Grid panel

Mogućnosti sortiranja i filtriranja prikaza kolona su podrazumevano dodate i korisniku postaju vidljive nakon prelaska kursorem nad zaglavljem kolone, kao što je prikazano na slici 10. Moguće je i oduzeti tu mogućnost korisniku postavljanjem vrednosti svojstava *sortable* i *hideable* na *false* nad datom kolonom.

Bridges					
Name	Inspection date	Load limit	▼	Closed	Rating
Missouri Pacific Railroad	03.06.2015.	25-40	↑½	Sort Ascending	
O'Hearne Avenue	13.08.2014.	15-25	↓½	Sort Descending	
Grand Louis Bayou	07.08.2013.	20-35	☰	Columns ▶	
Crescent City Connection	13.05.2011.	25-40	☐		8.4
					average

Name
 Inspection date
 Load limit
 Closed
 Rating

Slika 10: Sortiranje i filtriranje prikaza kolona

3 Rutiranje

Značaj transporta, kako ljudi tako i robe, pogotovo u današnjim uslovima kada su veliki delovi sveta jako povezani i kada je npr. preko Interneta moguće naručiti robu sa hiljadama kilometara udaljenog mesta, teško je preceniti. Različiti faktori koje bi algoritam za rutiranje trebao da uzme u obzir pri određivanju rute mogu se kategorisati na sledeći način[11]:

- lične faktore: da li ruta treba da bude najbrža, najkraća, da li treba da sadrži posebne objekte, da li treba da uzme u obzir cenu putarine itd.
- faktore vozila: kategorija vozila, optimalna brzina, potrebe za gorivom itd.
- statičke faktore putanje kao što su: klasifikacija puteva, ograničenja pristupa, brzine ili skretanja, broj traka itd.
- dinamičke faktore putanje kao što su: saobraćajni uslovi, dužina trajanja saobraćajne signalizacije, vremenski uslovi
- kvalitet puta: površinski materijal, kvalitet, širina trake i puta, godine korišćenja itd.

U budućnosti, u aplikacijama za rutiranje, treba očekivati sve veću inkorporaciju[11]

- telemetrije: procesa prikupljanja podataka sa udaljenih mesta. Jednostavno rečeno, radi se o prikupljanju podataka o stanju na putevima
- pogleda s ulice (eng. *street-view*): sve je češća upotreba kamera u vozilima tako da će snimci i slike koje oni prave postajati sve dostupniji za upotrebu
- slika iz vazduha: zahvaljujući dronovima dosta je lakše napraviti snimke predela i to veće rezolucije od satelitskih
- mašinskog učenja: napreci u ovoj oblasti stvaraju mogućnost da se automatizuje proces prenosa podataka sa satelitskih snimaka na mape

3.1 Rutiranje teretnih vozila

Kada je u pitanju rutiranje teretnih vozila, dodatan aspekt koji treba imati u vidu je nosivost mostova koji se nalaze u okviru rute. Mostove, a pogotovo one koji su u sastavu autoputeva i drugih bitnih saobraćajnica koristi veliki broj vozila od kojih su najproblematičnija u smislu opterećenosti i održavanja, ona sa najvećom težinom. U tom pogledu, posebno bitan aspekt je odnos dužine vozila i njegove težine, odnosno što je dužina vozila manja njegova težina je više koncentrovana.

Imajući to u vidu, u Sjedinjenim Američkim Državama su već u prvoj polovini XX veka prepoznali potrebu za zakonskom regulacijom prelaska mostova koja se u početku odnosila na maksimalnu dozvoljenu težinu vozila koje sme preći most, a kasnije prerasla u formulu:



































$$w = 500\left(\frac{ln}{n-1} + 12n + 36\right)$$

gde

- w označava maksimalnu dozvoljenu težinu u funtama na dve ili više uzastopnih osovina
- l označava ukupnu dužinu između razmatranih osovina i
- n njihov broj

I dok ova formula obuhvata mostove u okviru sistema međudržavnih puteva, postoji dosta mostova koji ne moraju zadovoljavati takve kriterijume jer su pre svega namenjeni za korišćenje od strane stanovnika neke manje regije a ne

prevozu velikog tereta. Takođe, veliki broj mostova je davno sagrađen tako da se maksimalna opterećenost koju oni mogu izdržati vremenom izmenila. Poseban tim inženjera, određen od strane uprave pod čijom se nadležnošću most nalazi, periodično obilazi svaki od tih mostova i ispituje stanje u kome se oni nalaze i na osnovu toga određuje da li su oni uopšte bezbedni za upotrebu, odnosno da li je neophodno izvršiti neku popravku mosta a zatim i maksimalne težine vozila koje ti mostovi mogu da podnesu. Ako te težine odstupaju od gorepomenute most-težina formule ili zakonskih normi propisanih na nivou države, ti mostovi bivaју označeni.

Class 1 Motorcycles		Class 7 Four or more axle, single unit		
Class 2 Passenger cars		Class 8 Four or less axle, single trailer		
				
				
				
Class 3 Four tire, single unit		Class 9 5-Axle tractor semitrailer		
				
				
Class 4 Buses		Class 10 Six or more axle, single trailer		
				
		Class 11 Five or less axle, multi trailer		
Class 5 Two axle, six tire, single unit			Class 12 Six axle, multi-trailer	
				
Class 6 Three axle, single unit		Class 13 Seven or more axle, multi-trailer		
				
				
				

Slika 11: Klasifikacija tipova vozila američkog federalnog ministarstva saobraćaja[24]

Postoji više vrsta saobraćajnih znakova tog tipa, i da bi se oni jasnije razumeli treba imati u vidu klasifikaciju tipova vozila propisanih od strane federalnog ministarstva saobraćaja prikazanoj na slici 11. Na toj slici možemo uvideti

razliku između jednočlanih (eng. *single-unit*) i višečlanih tipova teretnih vozila, koja je najbitnija za određivanje mogućnosti prelaska mosta. Naravno, pošto se ograničenje odnosi na nosivost mostova, saobraćajni znakovi koji ih označavaju, bilo da su tekstualni ili simbolički, na sebi nemaju oznake motociklova ili automobila već različitih tipova teretnih vozila.

Primer znaka kojima su označeni mostovi razmatrani u okviru ovog master rada prikazan je na slici 12. Svakako treba imati u vidu da to ne unosi značajne promene na konceptualnom nivou aplikacije već je vezano za konkretne mostove. Podaci korišćeni u ovoj aplikaciji odnose se na američku saveznu državu Luizijanu, njihov glavni značaj je u ilustraciji korišćenja aplikacije.



Slika 12: Saobraćajni znak: nosivost mosta za različite tipove teretnih vozila

3.2 Dijkstrin algoritam

Dijkstrin algoritam predstavlja jedan od osnovnih algoritama za rutiranje a drugi naziv po kom je poznat, na engleskom jeziku, *Single source shortest path*, otkriva njegovu namenu: određivanje najkraće putanje iz jednog čvora grafa ka ostalim čvorovima. Opis ovog algoritma najvećim delom preuzet je iz knjige "Algorithms in a Nutshell"[4].

Da bi se objasnio algoritam, uvodimo strukturu podataka- red sa prioritetom: svakom elementu u ovom redu dodeljen je ceo broj koji predstavlja njegovu

važnost. Operacije na raspolaganju su

- ubaci: dodavanje elementa (sa njemu pridruženim prioritetom)
- getMin: uzimanje elementa sa najmanjim pridruženim brojem (što je taj broj manji, značaj elementa je veći)
- smanjiPrioritet: izmena prioriteta pojedinačnog elementa

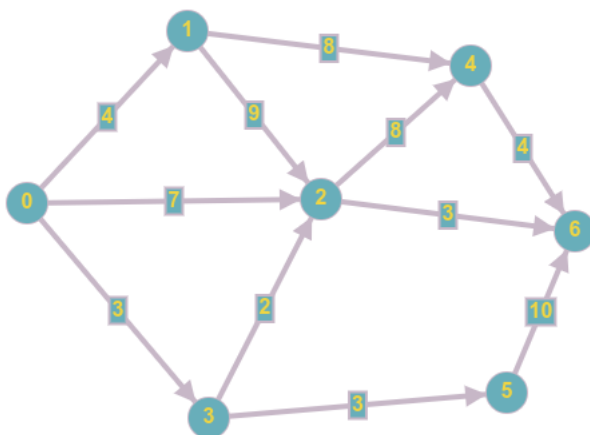
Uz tako definisane metode, prikazan je pseudo-kod algoritma:

Algorithm 1 Dijkstrin algoritam

```
1:  $PQ \leftarrow \emptyset$  ▷ inicijalizacija reda
2: for each  $v \in V$  do
3:    $dist[v] \leftarrow \infty$  ▷ u početku se svi čvorovi smatraju beskonačno
   udaljenim
4:    $pred[v] \leftarrow -1$ 
5: end for
6:  $dist[s] \leftarrow 0$ 
7: for each  $v \in V$  do ▷ popunjavanje reda
8:   INSERT( $v, dist[v]$ ) into  $PQ$ 
9: end for
10: while  $PQ \neq \emptyset$  do
11:    $u \leftarrow$  GETMIN() ▷ uzimanje čvora najbližeg početnom
12:   for each sused  $v$  od  $u$  do
13:      $w \leftarrow$  težina grane ( $u, v$ )
14:      $novaDuzina \leftarrow dist[v] + w$ 
15:     if  $novaDuzina < dist[v]$  then ▷ ako je pronađena kraća putanja
16:       SMANJIPRIORITET( $PQ, v, novaDuzina$ )
17:        $dist[v] \leftarrow novaDuzina$ 
18:        $pred[v] \leftarrow u$ 
19:     end if
20:   end for
21: end while
```

Dakle, Dijkstrin algoritam na ulazu očekuje usmereni težinski graf sa nenegativnim težinama i početni čvor da bi na pohlepan način širio skup čvorova S , koji obuhvata čvorove za koje je poznata najkraća distanca od početnog čvora uzimajući u obzir samo grane koje pripadaju čvorovima u tom skupu. Algoritam u svakoj iteraciji uzima najbliži čvor početnom, koji nije u skupu S i ispituje njegove grane da bi utvrdio da li postoji neki kraći put do ostalih čvorova u

S. Algoritam se izvršava dok se ne prođu svi čvorovi do kojih postoji put iz početnog čvora i nakon izvršavanja imamo dva formirana niza: *dist* označava najkraću udaljenost od početnog do ostalih čvorova u grafu a pomoću *pred* se te putanje mogu rekonstruisati. Kompleksnost algoritma implementovanog na način opisan u pseudokodu, preko reda sa prioritetom, je $O((V+E)\log V)$.



Slika 13: Usmereni težinski graf[5]

3.3 Prošireni model grafa u OSRM-u

Applikacija za rutiranje koristi veb servis otvorenog koda *Open Source Routing Machine (OSRM)*. U svom radu *OSRM* koristi prošireni model grafa. Motivacija za to proširenje može se ilustrovati situacijom prikazanom na slici 14. Koristeći klasičan model grafa i Dijkstrin algoritam, zaobilazna putanja od čvora *a* do *b* (preko *f*, *c*, *d* i *e*) nikad ne bi mogla biti pronađena jer, s obzirom da je putanja od čvora *f* do *b* zabranjena, algoritam bi prošao kroz čvorove *c*, *d* i *e*, stigao do čvora *f* i ustanovio da kraća putanja od čvora *a* do njega već postoji. Obzirom da je zabrana skretanja u određenom pravcu na raskrsnici česta pojava, njeno zanemarivanje bi moglo značajno uticati na rezultate bilo kog servisa (algoritma) za rutiranje u saobraćaju. Kao što je pomenuto, način na koji *OSRM* rešava ovaj problem ogleda se u sledećem: umesto da modeluje raskrsnice i puteve među njima, on modeluje puteve i skretanja među njima[10]. Tačnije, putanje između dve lokacije na mapi podele se u segmente

iteracija	PQ	dist														
0	[0]	<table border="1"> <tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td></tr> <tr><td>0</td><td>∞</td><td>∞</td><td>∞</td><td>∞</td><td>∞</td><td>∞</td></tr> </table>	0	1	2	3	4	5	6	0	∞	∞	∞	∞	∞	∞
0	1	2	3	4	5	6										
0	∞	∞	∞	∞	∞	∞										
1	[1, 2, 3]	<table border="1"> <tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td></tr> <tr><td>0</td><td>4</td><td>7</td><td>3</td><td>∞</td><td>∞</td><td>∞</td></tr> </table>	0	1	2	3	4	5	6	0	4	7	3	∞	∞	∞
0	1	2	3	4	5	6										
0	4	7	3	∞	∞	∞										
2	[1, 2, 5]	<table border="1"> <tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td></tr> <tr><td>0</td><td>4</td><td>5</td><td>3</td><td>∞</td><td>6</td><td>∞</td></tr> </table>	0	1	2	3	4	5	6	0	4	5	3	∞	6	∞
0	1	2	3	4	5	6										
0	4	5	3	∞	6	∞										
3	[2, 5, 4]	<table border="1"> <tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td></tr> <tr><td>0</td><td>4</td><td>5</td><td>3</td><td>12</td><td>6</td><td>∞</td></tr> </table>	0	1	2	3	4	5	6	0	4	5	3	12	6	∞
0	1	2	3	4	5	6										
0	4	5	3	12	6	∞										
4	[5, 6, 4]	<table border="1"> <tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td></tr> <tr><td>0</td><td>4</td><td>5</td><td>3</td><td>12</td><td>6</td><td>10</td></tr> </table>	0	1	2	3	4	5	6	0	4	5	3	12	6	10
0	1	2	3	4	5	6										
0	4	5	3	12	6	10										
5	[6, 4]	<table border="1"> <tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td></tr> <tr><td>0</td><td>4</td><td>5</td><td>3</td><td>12</td><td>6</td><td>10</td></tr> </table>	0	1	2	3	4	5	6	0	4	5	3	12	6	10
0	1	2	3	4	5	6										
0	4	5	3	12	6	10										
6	[4]	<table border="1"> <tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td></tr> <tr><td>0</td><td>4</td><td>5</td><td>3</td><td>12</td><td>6</td><td>10</td></tr> </table>	0	1	2	3	4	5	6	0	4	5	3	12	6	10
0	1	2	3	4	5	6										
0	4	5	3	12	6	10										
7	[]	<table border="1"> <tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td></tr> <tr><td>0</td><td>4</td><td>5</td><td>3</td><td>12</td><td>6</td><td>10</td></tr> </table>	0	1	2	3	4	5	6	0	4	5	3	12	6	10
0	1	2	3	4	5	6										
0	4	5	3	12	6	10										

Tabela 2: Izvršavanje algoritma nad grafom prikazanim na slici 13

i za svaki od njih dodaju se dva čvora- za različite smerove kretanja. Grana između čvorova se dodaje ako su dva segmenta povezana i ako je moguće preći iz jednog segmenta u drugi: težina grane dobija se sabiranjem težine originalne grane i težine skretanja.



Slika 14: Model grafa i zabrana skretanja[10]

3.4 Hijerarhijske kontrakcije

S obzirom na kompleksnost i količinu podataka koju reprezentacija mreže puteva treba da sadrži, *OSRM* koristi tehniku ubrzanja poznatu pod nazivom

hijerarhijska kontrakcija[3]. Ovaj metod ima dve faze:

- fazu preprocesiranja i
- fazu obrade upita

Faza preprocesiranja traje dosta duže od faze upita, njen cilj je da u fazi upita algoritam razmatra što manji broj čvorova kako bi se što brže došlo do rezultata. Može se podeliti u dve celine:

- uređenje čvorova
- kontrakcija

Uređenje čvorova sastoji se u dodeljivanju prioriteta svakom od njih. Primena ove ideje u algoritmu rutiranja i intuitivno deluje ispravno: u razmatranju bilo kakve mreže puteva u realnim okolnostima, uvek se mogu izdvojiti putevi sa višim odnosno nižim značajem. Redosledom određenim dodeljenim prioritetima čvorova, može se otpočeti sa korakom kontrakcije: čvor v se uklanja tako da rezultujući graf i dalje sadrži sve najkraće puteve. To se obezbeđuje time što se u taj rezultujući graf dodaje *grana-prečica* ukoliko je najkraći put između čvorova u i w sadržao uklonjeni čvor.

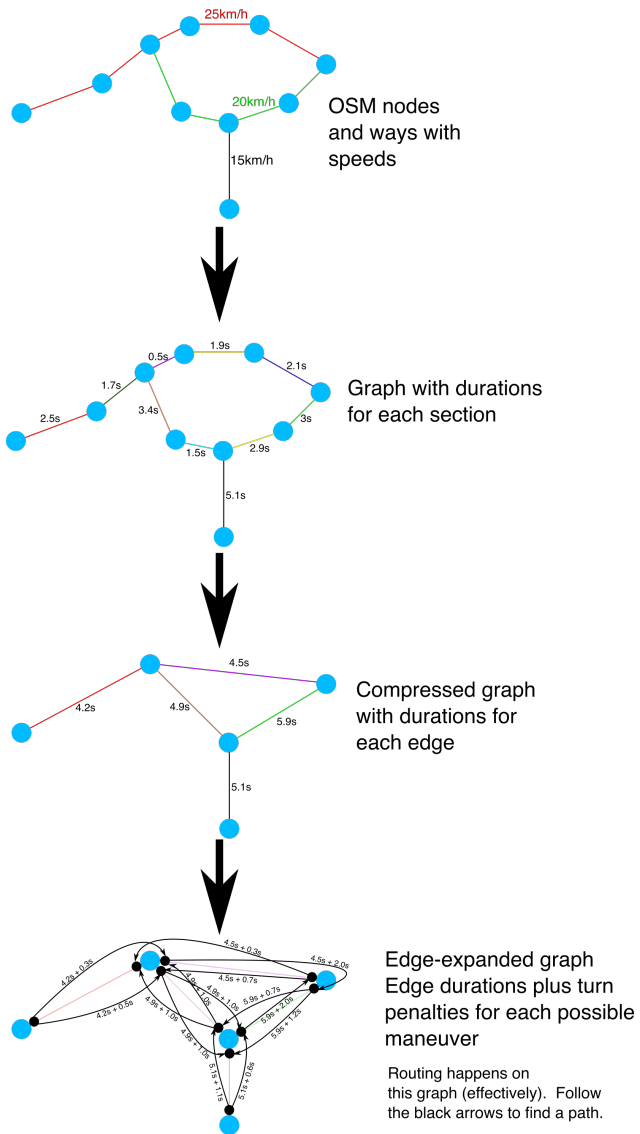
Da bi se odredio prioritet čvora moguće je koristiti različite mere i heuristike. Geisberger et al. u svom radu navode dve ključne mere koje treba uzeti u obzir:

- razlika u granama: razlika između broja novouvedenih grana nakon uklanjanja čvora i broja grana susednih tom čvoru
- uniformnost: umesto kontrahovanja čvorova u nekom manjem regionu grafa, bitno je kontrakcije vršiti u svim delovima grafa

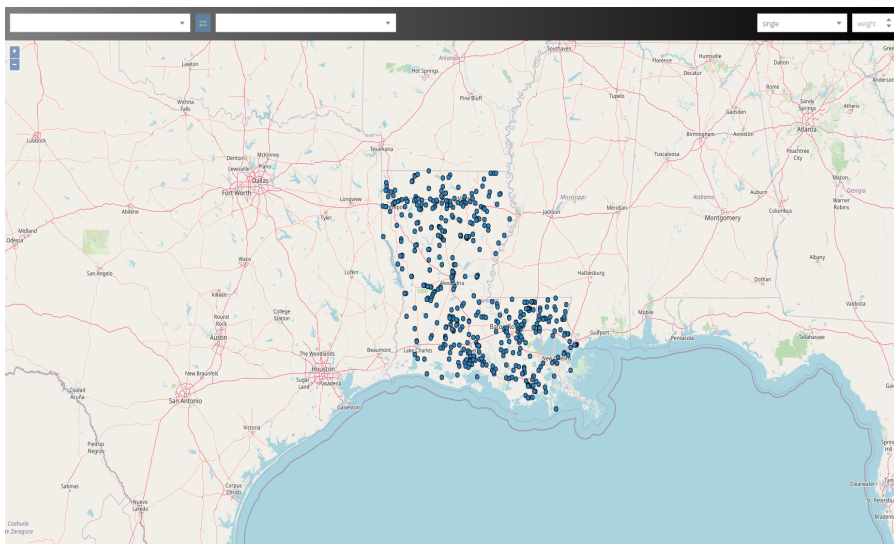
U fazi izvršavanja upita obično se koristi dvosmerni Dijkstrin algoritam, i to u slučaju [3], nad dva podgrafa: jedan koji sadrže samo grane od čvorova niže vrednosti, u prethodno utvrđenom uređenju, ka čvorovima više vrednosti i obrnuto.

4 Dizajn Ext JS aplikacije za rutiranje

Glavna funkcionalnost aplikacije je pronalaženje najbrže rute između dve-tačke uzimajući u obzir sve mostove i nadvožnjake koji se na njoj nalaze. Algoritamski se ta funkcionalnost može podeliti na sledeće faze:



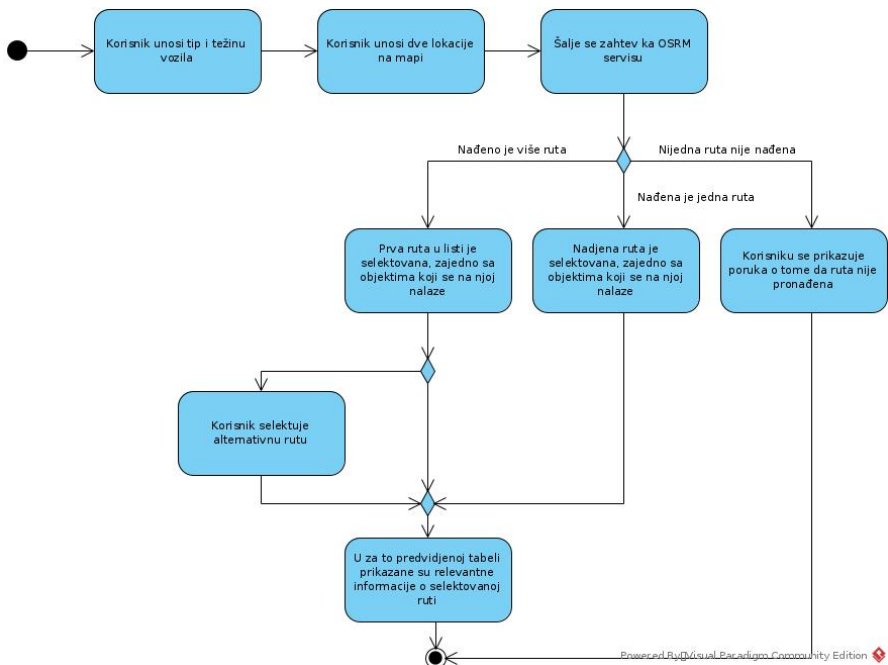
Slika 15: Pojednostavljen primer transformacije grafa pre primene algoritma za rutiranje[6]



Slika 16: Izgled aplikacije nakon učitavanja

- OSRM servisu šalje se zahtev sa koordinatama početne i odredišne tačke
- za svaku nađenu rutu, koristeći biblioteku JSTS, formira se sloj oko rute kojim se ispituje presek sa tačkama koji označavaju mostove (zbog čega je bitno da, u slučaju nadvožnjaka, nema preklapanja)
- za svaku tačku koja je u preseku sa pomenutim slojem ispituju se vrednosti nosivosti

Na slici 17 prikazan je dijagram aktivnosti za očekivani slučaj upotrebe. Kada se aplikacija pokrene, korisniku je prikazana mapa sa ikonicama u obliku kruga koje predstavljaju mostove odnosno nadvožnjake koji sadrže oznake za ograničenje težine vozila, kao što je prikazano na slici 16. Klikom na ikonicu otvara se tabela koja sadrži podatke o datom objektu, kao što su naziv, lokacija, dužina, broj traka, podaci vezani za inspekciju, o tome ko se bavi održavanjem i mnogi drugi. Početna odnosno krajnja tačka željene putanje mogu se uneti ili desnim klikom na mesto na mapi ili tekstualnim unosom u za to predviđena polja u gornjem levom uglu. Ako je jedna tačka već dodata onda se nakon dodavanja druge pokreće funkcija koja vraća moguće rute i najbitnije informacije o njima: dužinu, trajanje i maksimalnu težinu



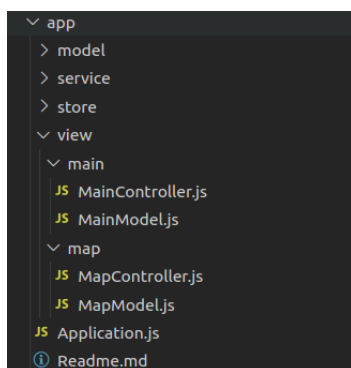
Slika 17: Dijagram aktivnosti[25]

vozila koje tom rutom može da prođe (u slučaju kad je dobijeno više ruta, informacije o željenoj prikazuju se nakon klika na nju). U gornjem desnom uglu korisnik ima mogućnost da izabere tip vozila (podrazumevana vrednost je *single*) kao i da unese ukupnu težinu vozila, tako da budu prikazane samo one rute kojima vozilo tog tipa i te težine može proći. Ukoliko nijedna ruta nije pronađena prikazuje se poruka o tome. Pored pomenutog OSRM servisa[20], aplikacija u svom radu koristi sledeće biblioteke odnosno resurse:

- OpenLayers[14]
- OpenStreetMap[17]
- Nominatim[12]
- Custom Context Menu[15]
- JSTS[9]

4.1 Neki detalji implementacije

Struktura datoteka odnosno direktorijuma u Ext JS aplikacijama bi (po preporuci kompanije Sencha[7]) trebala da bude uniformna, pa tako direktorijum *app* sadrži datoteke vezane za modele, servise, skladišta, kontrolere i modele korišćenih pogleda, kao što se može videti na slici 18. Arhitektura aplikacije je MVVM, te se iz pomenute slike može zaključiti i da aplikacija sadrži dva pogleda: *main* i *map*. Same datoteke u kojima su definisani pogledi se, po pomenutoj preporuci, nalaze na drugoj lokaciji u okviru direktorijuma aplikacije kreirane pomoću alata *Sencha cmd*. To je alat kojim se može kreirati generička aplikacija koja se zatim može modifikovati i dopunjavati na željeni način[23].



Slika 18: Modeli i kontroleri definisanih pogleda

Kao što se može zaključiti iz samog naziva, pogled *main* je glavni pogled aplikacije, što znači da se on prikazuje u pretraživaču kada se aplikacija pokrene. On sadrži dve komponente od kojih je prva panel prikazan u gornjem delu aplikacije u kom su definisane komponente korisničkog interfejsa za unos lokacija i detalje o vozilu a druga panel u okviru kog je definisan i pogled *map* (pogled je takođe komponenta). Model pogleda *main* prikazan je u kodu 1 dok kontroler pogleda sadrži funkcije koje reaguju na događaje kao što su unos lokacije (funkcija koja reaguje na tekstualni unos lokacije predstavljena je kodom 7), zamena početne i krajnje tačke itd.

```
1 initializeMap: function() {  
2     var controller = this;
```

```

3   var vm = this.getViewModel();
4   //EPSG:3857
5   var mapView = new ol.View({
6     center: ol.proj.fromLonLat([-92.411, 30.924]),
7     zoom: 7
8   });
9   var map = new ol.Map({
10    interactions: ol.interaction.defaults({
11      shiftDragZoom: false
12    }),
13    layers: controller.createLayers(),
14    view: mapView,
15    controls: [
16      new ol.control.Zoom(),
17      new ol.control.Rotate()
18    ]
19  });
20  map.addControl(this.createContextMenu(map));
21  this.getView().setMap(map);
22  vm.set('map', map);
23  this.addMapInteractions(map);
24 }

```

Listing 17: Inicijalizacija kontrolera pogleda *map*

Kontroler pogleda *map* sadrži funkcije vezane za dodavanje OpenLayers mape i njoj pridruženih slojeva kao i funkciju *getRoute* koja vrši slanje zahteva ka OSRM servisu i obradu odgovora tj. poređenje relevantnih vrednosti za svaku tačku na mapi koja je u preseku sa nekom od dobijenih ruta. Funkcija prikazana u kodu 17 predstavlja inicijalizaciju kontrolera pogleda u okviru koje se instancira OpenLayers mapa koja se sačuva u modelu pogleda nakon čega se poziva funkcija kojom se dodaju korisničke interakcije za selektovanje objekata na mapi i prevlačenje početne i krajnje tačke.

4.2 OpenLayers

OpenLayers je Javascript biblioteka otvorenog koda koja se koristi za dodavanje mapa na veb stranice. Mapa, kao ključna komponenta ove biblioteke, se sastoji od[16]

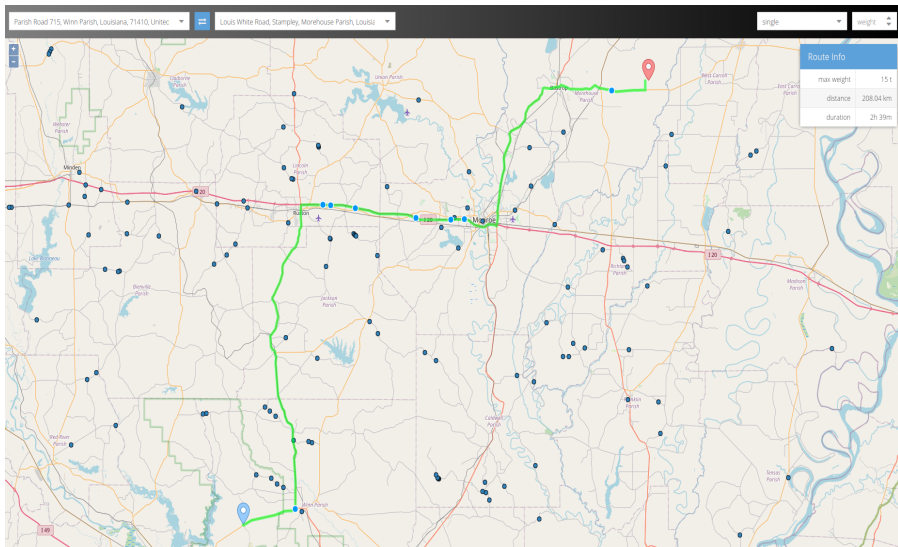
- slojeva
- pogleda
- interakcija
- kontrola

Sloj predstavlja mapu u užem značenju, dakle ekvivalent onome što se podrazumeva pod rečju mapa u svakodnevnom govoru. U ovom radu za te svrhe korišćen je *OpenStreetMap*.

Pogled kontroliše koji se deo mape, i na koji način, vidi.

Interakcije determinišu načine na koje korisnik može da koristi mapu, kao što su prevlačenje, zumiranje itd.

Kontrole predstavljaju komponente korisničkog interfejsa, na primer, dugmeta pomoću kojih se kontroliše zumiranje.



Slika 19: Prikaz rute nakon unosa podataka

5 Zaključak

Izrada veb aplikacija u velikoj većini slučajeva podrazumeva korišćenje nekog Javascript radnog okvira. U ovom radu predstavljene su glavne karakteristike Ext JS radnog okvira i ilustrovan način upotrebe njegovih najčešće korišćenih komponenti. Zahvaljujući tim ilustracijama čitalac takođe može da stekne utisak o doslednom stilu programiranja koji uz dostupnu dokumentaciju može značajno da olakša usvajanje znanja i upotrebu ovog radnog okvira čije mogućnosti ga, ovaj rad je imao za cilj da pokaže, sa svojim prednostima i manama čine nesumnjivo vrednim razmatranja u odnosu na druge radne okvire.

Kao što je već istaknuto, glavnu prednost predstavlja veliki broj integrisanih komponenti korisničkog interfejsa, što može značajno skratiti vreme potrebno za izradu veb aplikacije i što zajedno sa drugim komponentama radnog okvira smanjuje potrebu za korišćenjem eksternih biblioteka. S druge strane, licenciranje radnog okvira je u velikoj meri uticalo na brojnost programerske zajednice, što je svakako važan faktor pri odlučivanju. Utisak je da je ovaj radni okvir posebno pogodan za aplikacije koje treba da prikažu i obrade veliku količinu podataka a za koje nije neophodan preveliki nivo prilagođavanja korisničkog interfejsa. Ext JS aplikacija razvijena tokom rada na ovoj master tezi, se nalazi na adresi <https://github.com/curkovical/masterApp>.

Da bi se shvatio značaj aplikacija za rutiranje, dovoljno je prisetiti se kako je naš transport izgledao bez njih. Naravno, za kompanije koje se time profesionalno bave, pitanja kao što su optimizacija rute, dobavljanje svih informacija vezanih za stanje na putu itd. imaju i mnogo veći značaj. Aplikacija predstavljena u ovom radu bavi se aspektom koji za te kompanije, ali ne samo za njih, može biti od velikog značaja a kome nije poklonjena dovoljna pažnja. Izrada same aplikacije ostavlja dosta prostora za unapređenja od kojih bi se mogla izdvojiti: izrada android aplikacije, mogućnost dodavanja među-tačaka na ruti od početne do krajnje kao i dodavanje korisničke komponente koja bi prikazivala uputstva vezana za samu putanju.

Nažalost, veliki broj komponenti radnog okvira i njihovih mogućnosti (zbog brojnosti i obima) u ovom radu nisu mogle biti dovoljno obrađene, te preporuka onima koji su zainteresovani za više primera da posete <https://examples.sencha.com/extjs/6.2.0/examples/kitchensink/#all>.

Korišćeni izvori

- [1] Armando Gonzalez Carlos A. Mendez Crysfel Villa. *Learning Ext JS, 4th Edition: Create powerful web applications with the new and improved Ext JS 5 library*. Packt Publishing, 2015. ISBN: 978-1-78439-438-7.
- [2] *Ext JS official documentation*. URL: <https://docs.sencha.com/extjs/6.2.0/index.html>.
- [3] Robert Geisberger et al. "Contraction Hierarchies: Faster and Simpler Hierarchical Routing in Road Networks". In: May 2008, pp. 319–333. DOI: 10.1007/978-3-540-68552-4_24.
- [4] Stanley Selkow George T. Heineman Gary Pollice. *Algorithms in a Nutshell*. O'Reilly Media, 2016.
- [5] *Graph Online*. URL: <https://graphonline.ru/en/>.
- [6] *Graph Representation*. URL: <https://github.com/Project-OSRM/osrm-backend/wiki/Graph-representation>.
- [7] *Introduction to Application Architecture*. URL: https://docs.sencha.com/extjs/6.2.0/guides/application_architecture/application_architecture.html.
- [8] Jacob K. Andresen Jesus Garcia Grgur Grisogono. *Ext JS in Action*. Manning Publications, 2014.
- [9] *JSTS*. URL: <https://github.com/bjornharrtell/jsts>.
- [10] Dennis Luxen. *Smart Directions Powered by OSRM's Enhanced Graph Model*. URL: <https://blog.mapbox.com/smart-directions-powered-by-osrms-enhanced-graph-model-3ae226974b2>.
- [11] *Mapping with OpenStreetMap*. URL: <https://labs.mapbox.com/mapping/mapping-for-navigation/introduction/>.
- [12] *Nominatim*. URL: <https://nominatim.org/>.
- [13] octoverse.github.com. 2020. URL: <https://octoverse.github.com/#top-languages> (visited on 04/18/2020).
- [14] *OpenLayers*. URL: <https://openlayers.org/>.
- [15] *OpenLayers Custom Context Menu*. URL: <https://github.com/jonataswalker/ol-contextmenu>.
- [16] *OpenLayers dokumentacija*. URL: <https://openlayers.org/en/latest/doc/> (visited on 05/16/2020).

- [17] *OpenStreetMap*. URL: <https://www.openstreetmap.org/> (visited on 05/17/2020).
- [18] *Performance Comparison Stats*. URL: <http://www.techferry.com/articles/ExtJS-vs-AngularJS.html#performance>.
- [19] *Posted Bridges*. URL: http://gisweb.dotd.la.gov/ArcGIS/rest/services/LADOTDAGO/LA_Posted_On_System_Bridges/FeatureServer/0/query?outFields=*&where=1%3D1 (visited on 07/22/2020).
- [20] *Project OSRM*. URL: <http://project-osrm.org/>.
- [21] *Sencha Licensing*. URL: <https://www.sencha.com/legal/>.
- [22] *Symphony. Ten criteria for choosing the correct framework*. 2020. URL: <https://symfony.com/ten-criteria> (visited on 04/20/2020).
- [23] *Using Sencha Cmd with Ext JS 6*. URL: https://docs.sencha.com/cmd/guides/extjs/cmd_app.html#extjs--cmd_app--generating_your_application.
- [24] *Vehicle types*. URL: https://www.fhwa.dot.gov/policyinformation/tmguidetmg_2013/vehicle-types.cfm.
- [25] *Visual Paradigm*. URL: <https://www.visual-paradigm.com/>.
- [26] *W3Techs*. 2020. URL: <https://w3techs.com/technologies/details/js-jquery> (visited on 04/20/2020).
- [27] *Why Develop with Sencha?* URL: <https://www.sencha.com/comparison/>.
- [28] *Computer World. The A-Z of Programming Languages: JavaScript*. 2008. URL: <https://www.computerworld.com/article/3458282/the-a-z-of-programming-languages-javascript.html> (visited on 04/18/2020).