Факултет инжењерских наука Универзитета у Крагујевцу

Назив студијског програма: Машинско инжењерство

Ниво студија: Основне академске студије

Модул: Информатика у инжењерству

Предмет: Софтверски инжењеринг

Број индекса: 119/2015

Никола Р. Петровић

Софтвер који рачуна оптимални авионски лет између два задата места у свету

завршни рад

Комисија за преглед и одбрану:

1. **Проф. Др Ненад Филиповић - ментор**
2. _____
3. _____

Датум одбране:_____

Оцена:_____

У оквиру овог завршног рада кандидат треба да направити софтвер са комплетном документацијом (упуство за кориснике, упуство за програмере, УМЛ дијаграме итд) који рачуна оптимални авионски лет између два задата места у свету. Унети реалну мапу света, и ставити само веће градове. Америка, Европа, Азија су довољне.

Крагујевац, датум

Ментор:

Проф. Др Ненад Филиповић

_____

Садржај

Summary

Software for finding the optimal path between the two cities on the map was created in the programming language javascript. Software contains the largest cities that are located in Europe, Asia, South and North America. Airports that are located in these cities are connected in graph. Dijkstra algorithm is used to find the most optimal route between the starting and final airport and that optimal route is shown on map.

Keywords: **map**, **javascript**, **optimal flight**, **airports**, **Dijkstra**

Резиме

Софтвер за проналажење оптималне путање између два задата града на карти је израђен у програмском језику „javascript“. Унети су највећи градови који се налазе у Европи, Азији, јужној и северној Америци. Аеородроми који се налазе у овим градовима су повезани у граф. Над графом је употрбљен Дајкстрин алгоритам који проналази најоптималнију путању између почетног и одредишног аеродрома и приказује је на карти.

Кључне речи: **карта**, **javascript**, **оптимални лет**, **аеродром**, **Дајкстра**

Увод

Рад садржи опис софтвера за налажење оптималне путање између два задата града.

Софтвер је писан у програмском језику „javascript“. Овај језик је изабран из разлога што има најбољу подршку за овакав тип софтвера. За „javascript“ језик постоји мноштво библиотека које олакшавају израду различитих пројеката. Веома је битно нагласити да је „javascript“ програмски језик најзаступљенији на интернету и такође један од тренутно најпопларнијих програмских језика на свету, што је највише утицало на избор језика у коме ће овај софтвер бити написан.

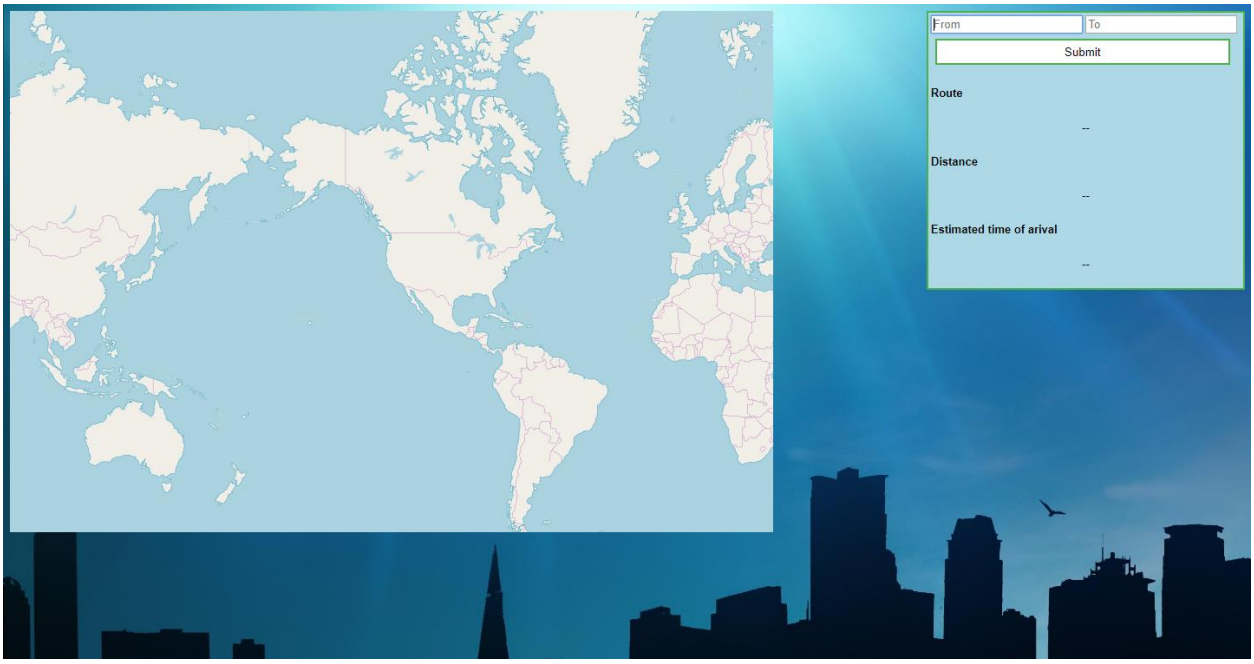
Софтвер је направљен у виду веб странице. Кôд није писан у објектно оријентисаном стилу програмирања, а разлог тога је једноставност самог пројеката.

За израду овог софтвера је коршћена библиотека „modestmaps“ [1], која знатно олакшава коришћење карте света.За одређивање оптималне путање је употребљен Дајкстрин алгоритам.

Кôд је одвојен по фајловима који су објашњени појединачно, као и све функције које кôд садржи. На крају рада се налази прилог А у коме се налази објашњење Дајкстриног алгоритма.

Упутство за кориснике

Софтвер за проналажење оптималне путање лета авиона је веома једноставно користити. Потребно је покренути фајл „index.html“ из фолдера „FlightPath“, нако чега ће се у вашем интернет претраживачу приказати страница са слике 1.



Слика 1 – *Приказ веб странице*

Са леве стране на слици 1 је приказана карта света која је потпуно интерактивна тј. Могуће је увећавати, умањивати и померати по жељи корисника.

Са десне стране, у горњем десном углу се виде два поља „From“ и „To“. Прво поље прима унос полазног аеродрома, док друго прима одредишни аеродром. Поља за унос имају могућност да довршавају речи које корисник уноси, што олакшава претрагу аеродрома.

Пример софтвера у раду је приказан на слици 2 где се види унет полазни аеродиром из Москве (Русија), а одредишни Валенсија (Шпанија).

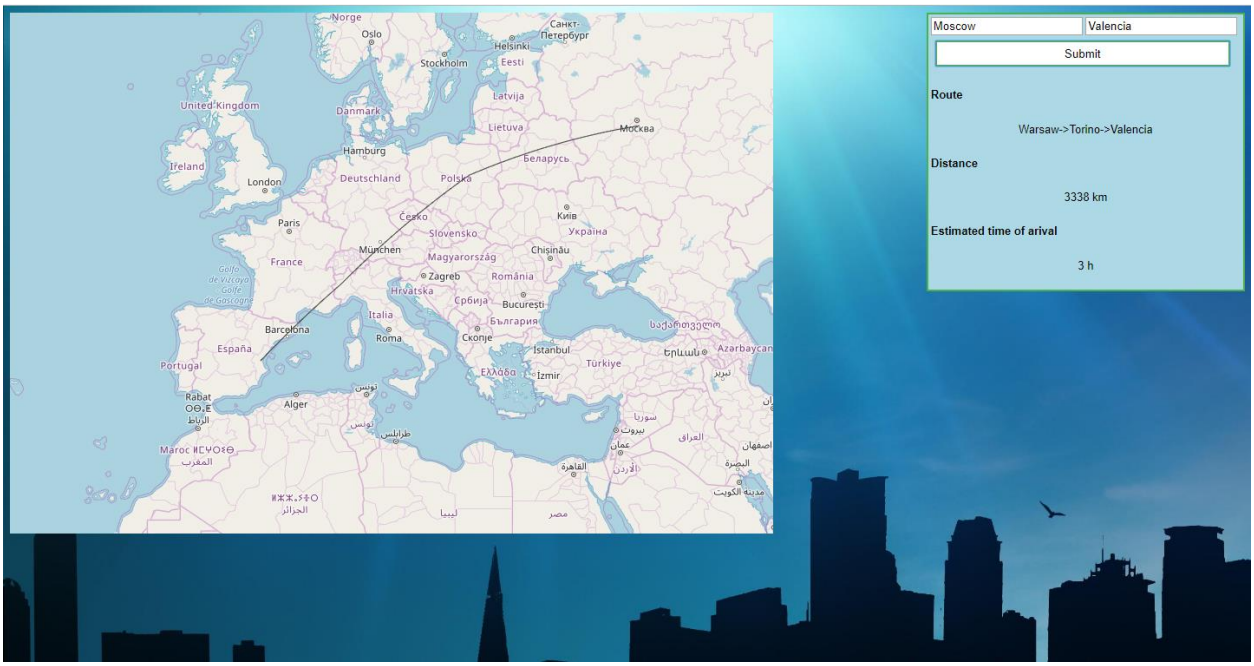
На слици 2 је приказана путања коју би авион требало да пређе од првог до другог аеродрома преко свих аеродрома који се налазе између та два.

Напомиње се да уколико би сви аеродроми који постоје били унети, (што у овом раду није био случај због смањења комплексности пројекта) путања би се исцртавала оптималније од ове приказане на слици 2.

На слици 2 је испод поља и дугмета за потврду уноса приказана конкретна путања по којој авион путује до своје дестинације. У овом случају авион полази из Вашингтона, прелази преко Торина (Италија) и слеће у Валенсију.

Испод путање је приказна укупно растојање од полазног аеордрома до одредишног преко свих посредних аеродрома. Пример у овом случају је од Вашингтона до Валенсије, преко Торина која износи 3338 km.

На крају у пољу „Estimated time of arival“ је информација о времену које је потребно да авион пређе према задатој путањи. На слици се види да је од Вашингтона до Валенсије, преко Торина потребно три сата лета. (Уколико је брзина лета авиона константна и инзноси 1000 km/h).



Слика 2 – *Пример путање лета авиона од Вашингтона до Валенсије*

Могуће је одабрати било које одредиште и полазни аеродром из наведене листе градова:

"Moscow", "Kiev", "Lyon", "Valencia", "Stockholm", "Oslo", "Berlin", "Helsinki", "Warsaw", "Torino", "London", "Belgrade", "Beijing", "Delhi", "Karaganda", "Tehran", "Jakarta", "Islamabad", "Istanbul", "Tokyo", "Washington", "Brasilia", "Mexico City", "Bogota", "Buenos Aires", "Ottawa", "Lima", "Caracas", "Havana"

Опис кода (Програмерско упутство)

Фолдер „FlightPath“ се састоји од фолдера „autocomplite“ и фајлова „app.js“ „Airports.js“, „index.html“, „Locations.js“, и „style.css“.

Фајл „index.html“

Фајл „index.html“ садржи кôд који повезује све остале поменуте фајлове. Такође је у кôду овог фајла укључен и приказ текста који се види у интернет претраживачу.

На слици 3 је приказан први део „index.html“ фајла, прецизније „head“ заглавље.

```
1  <!DOCTYPE html>
2  <html>
3
4
5  <head>
6    <meta charset=utf-8 />
7    <title>Optimal Flight Path</title>
8
9    <link rel="stylesheet" href="autocomplite/awesomplete.css" />
10   <script src="autocomplite/awesomplete.js" async></script>
11   <link rel='stylesheet' href='style.css'>
12 </head>
```

Слика 3 – *„head“ заглавље „index.html“ фајла*

На линији бр. 7 у ознакама „title“ је постављен назив картице у интернет претраживачу.

На линијама бр. 8 и 9 су учитани фајлови из фолдера „autocomplite“ који се користе за аутоматско довршавање назива градова приликом претраге.

На линији бр. 11 је учитан фајл „style.css“, који саджи дизајн саме странице.

Na слици 4 је приказан део кôда који је задужен за постављање поља за унос градова на веб страници (линије бр. 18 и 19). Док је на линији бр. 21 постављено дугме „Submit“ које када се укључи активира методу „mapInitialization()“, која се налази у фајлу „app.js“.

```
17     <div>
18       <input type="search" placeholder="From" autocomplete="off" oninput="" id="start" name="" value="">
19       <input type="search" placeholder="To" autocomplete="off" oninput="" id="end" name="" value="">
20       <br>
21       <button id="button" onclick="mapInitialization()">Submit</button>
22     </div>
```

Слика 4 – *Дугме и поља у фајлу „index.html“*

На слици 5 је приказан кôд у коме је текст који помаже информисању корисника о лету тј. путањи лета, растојању и времену трајања лета.

```
24      <div id="info">
25        <h4>Route</h4>
26
27        <p id="t1">--</p>
28
29        <h4>Distance</h4>
30
31        <p id="t2">--</p>
32
33        <h4>Estimated time of arival</h4>
34
35        <p id="t3">--</p>
36      </div>
```

Слика 5 – *Помоћ приказивању информација о лету.*

Слика 6 приказује учитане фајлове, а то су:

На линији бр. 41 је учитана библитека „modestmaps“ која се користи за приказивање карте на веб страници,

на линији бр. 42 је учитан фајл „Locations.js“ који садржи називе градова ради аутоматског довршавања имена истих,

на линији бр. 43 је учитан фајл „Airports.js“ који садржи податке о аеродромима,

на линији бр. 44 је учитан фајл „app.js“ који садржи javascript кôд.

```
41    <script src='https://rawgithub.com/stamen/modestmaps-js/master/modestmaps.min.js'></script>
42    <script src="Locations.js"></script>
43    <script src='Airports.js'></script>
44    <script src='app.js'></script>
```

Слика 6 – *Учитани фајлови*

Фајл „Airports.js“

Фајл „Airports.js“ садржи информације о аеродромима. На слици 7 се виде прва два аеродрома из фајла „Airports.js“ и информације о њима.

Као пример ће се узети аеродром „Moscow-Domodedovo“ који се налази у Москви који је први приказан на слици 7.

Информације које су битне из приказаног фајла су географска ширина и географска дужина, које су дефинисане под ознакама „Latitude“ и „Longitude“. Ова два податка се користе касније у кôду за одређивање позиције аеродрома на карти.

Називи градова у којима се налазе аеродроми се користе за претрагу. У овом случају је тај податак под ознаком „City“.

Остале информације о аеродрому које су приказана се могу користи, али у овом пројекту није било потребе за њима.

Аеродроми из фајла су дефинисани променљивом „d“.

```
var d = [
{
   "ID": 1,
   "Name": "Moscow-Domodedovo",
   "City": "Moscow",
   "Country": "Russia",
   "IATA/FAA": "BKA",
   "ICAO": "UUDD",
   "Latitude": 55.4,
   "Longitude": 37.9,
   "Altitude": 427,
   "Timezone": 4,
   "DST": "N",
   "CityPopulation": 13100000,
   "AIRPORTS with IATA and ICAO codes": "",
   "http://openflights.org/data.html": "",
   "": ""
 },
 {
   "ID": 62,
   "Name": "Gostomel Antonov",
   "City": "Kiev",
   "Country": "Ukraine",
   "IATA/FAA": "GML",
   "ICAO": "UKKM",
   "Latitude": 50.603611,
   "Longitude": 30.191944,
   "Altitude": 517,
   "Timezone": 2,
   "DST": "E",
   "CityPopulation": 2900000,
   "AIRPORTS with IATA and ICAO codes": "",
   "http://openflights.org/data.html": "",
   "": ""
 },
```

Слика 7 - *Информације о аеродромима „Moscow-Domodedovo“ и „Gostomel Antonov“*

Фајл „Locations.js“

Фајл „Locations.js“ садржи називе градова који су извучени из фајла „Airports.js“.

Називи градова из овог фајла се користе за аутоматску претрагу аеродрома и декларисани су за даљу употребу као променљива „locationsAutocomplete“.

На слици 8 је приказан фајл „Locations.js“ у коме су називи смештени као низ података.

```
var locationsAutocomplete =

["Moscow",
 "Kiev",
 "Lyon",
 "Valencia",
 "Stockholm",
 "Oslo",
 "Berlin",
 "Helsinki",
 "Warsaw",
 "Torino",
 "London",
 "Belgrade",
 "Beijing",
 "Delhi",
 "Karaganda",
 "Tehran",
 "Jakarta",
 "Islamabad",
 "Istanbul",
 "Tokyo",
 "Washington",
 "Brasilia",
 "MexicoCity",
 "Bogota",
 "BuenosAires",
 "Ottawa",
 "Lima",
 "Caracas",
 "Havana"];
```

Слика 8 – *Називи градова у коме се налазе аеродроми*

Фајл „style.css“

Фајл „style.css“ се користи за стилизовање података који су приказани на веб страници, а који су дефинисани у фајлу „index.html“.

На слици 9 су приказана прва два циљна елемента.

Први циљни елемент је „body“ у коме дефинише општи изглед текста који је дефинисан под ознаком истог имена у „html“ фајлу. Под њим су дефинисани:изглед фонта и његова величина, боја позадине која је светло плаве боје, слика „city-background.jpg“ која је учитана и на крају дефинисана тако да попуњава целину веб странице.

Под циљним елементо „map“ је дифинисана позиција мапе која је у центру веб странице и величина мапе, која је у овом случају висине 600 пиксела а дужине 880 пиксела.

```css
body {
  font: 13px/22px 'Helvetica Neue', Helvetica, sans;
  background: lightcyan;

  background-image: url(city-background.jpg);
  background-size: cover
}
#map {
  float: center;
  height: 600px;
  width: 880px;
}
```

Слика 9 – *Ознаке „body“ и „map“ у фајлу „style.css“*

На слици 10 под ознаком „button“ и је дефинисан изглед дугмета за потврду уноса, док је под ознаком „sidebar“ дефинисан изглед дела на веб страници на коме су постављена поља, дугме и информације о путањи авиона.

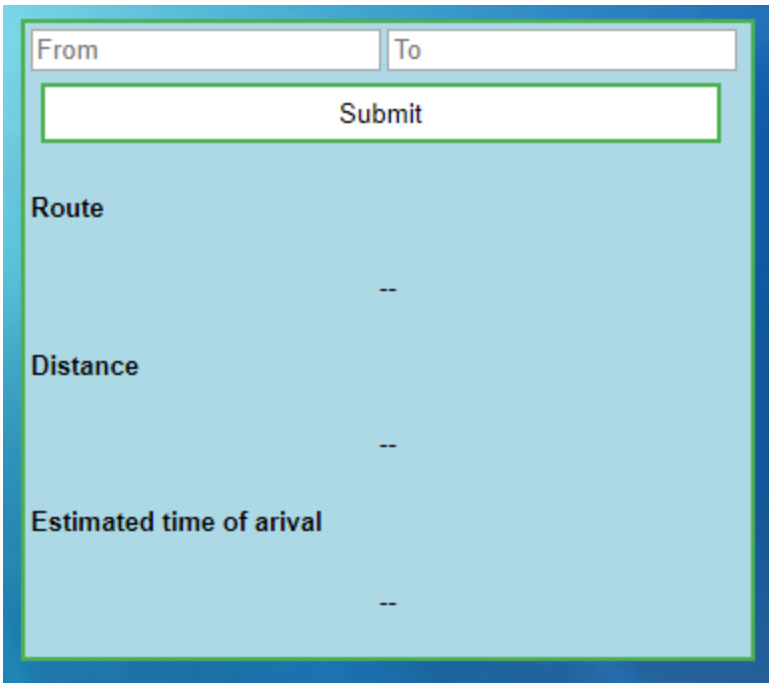
Дугме има маргину вредности 5 пиксела. Висина је 30 пиксела, док је његова дужина 340 пиксела. Дугме је беле боје, а оквир црне боје.

Део веб странице који је приказна на слици 11 ,под ознаком „sidebar“ је светло плаве боје , док је његов оквир црне боје и постављен је са десне стране веб странице.

```css
#button{
  margin: 5px;
  cursor: pointer;
  height:30px;
  width:340px;
  background-color: white;
  color: black;
  border: 2px solid #4CAF50; /* Green */
}

#sidebar{
  padding: 3px;
  background-color: lightblue;
  color: black;
  border: 2px solid #4CAF50; /* Green */
  float: right;
}
```

Слика 10 – *Ознаке „button“ и „sidebar“ у фајлу „style.css“*



Слика 11 – *Поља, дугмад и информације о путањи које се налазе под ознаком „sidebar“ у фајлу „style.css“*

Последњи део фајла „style.css“ приказан на слици 12 дефинише изглед текста који се приказује на веб страници и представља информације о путањани лета авиона, трајање лета и растојање.

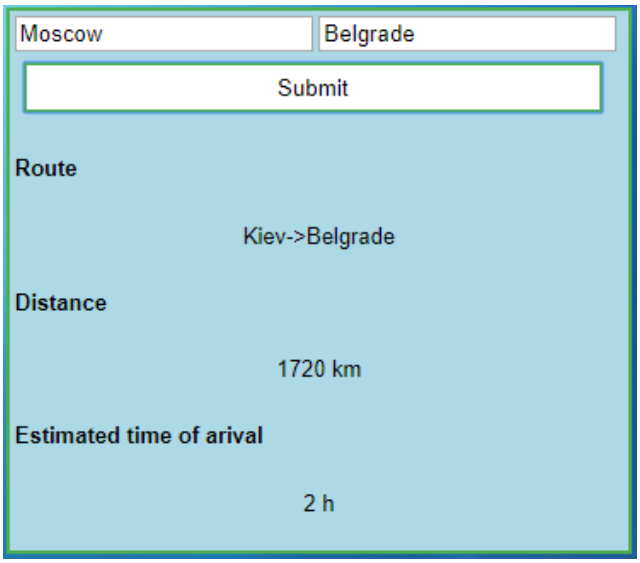
Ознака „t1“ се односи на информацију које ће бити приказана испод текста „Route“. Информација ће стојати у центрз квадрата за приказ информација и уколико је њена дужина већа од ширине квадрата, прећиће у нови ред.

Ознаке „t1“ и „t2“ су дефинисане да стоје у центру квадрата за приказ информација.

Пример дизајна приказа информација може се видети на слици 13.

```
#t1{
  text-align: center;
  white-space: pre-wrap;
}
#t2{
  text-align: center;
}
#t3{
  text-align: center;
}
```

Слика 12 – *Стилизавње информација о лету*



Слика 13 – *Приказ информација о одређеном лету*

Фајл „app.js“

Фајл „app.js“ садржи javascript кôд који контролише све остале претходно поменуте фајлове и означава суштину овог рада.

Овај фајл ће бити објашњен према редоследу извршавања кôда. Такође ће бити показан граф путања авиона између градова, као и Дајкстрин алгоритам који бира најоптималнију путању из понуђеног графа.

Приликом покретања фајла „app.js“ тј. пре свих акција на веб страници које корисник изврши. Покреће су две функције: **AutocompliteLoad()** и функција **loadMap()**.

Функција **AutocompliteLoad()** је приказана на слици 14.

```
301 function AutocompliteLoad() {
302     var start = document.getElementById("start");
303     var end = document.getElementById("end");
304 
305     new Awesomplete(start, {
306         list: locationsAutocomplete
307     });
308     new Awesomplete(end, {
309         list: locationsAutocomplete
310     });
311 
312 }
```

Слика 14 – *Функција за аутоматско довршавање имена градова*

За потребе аутоматског довршавања имена градова коришћена је библотека „Awesomplete“ [1].

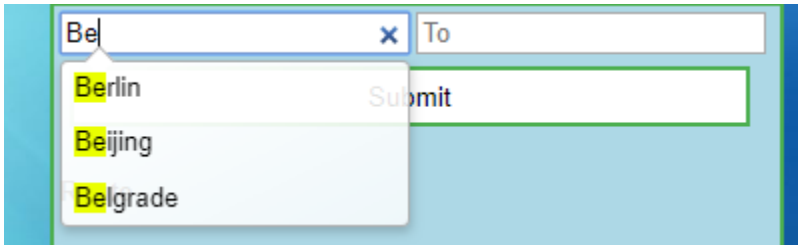
Функција **AutocompliteLoad()** имплементира библиотеку која је учитана у фајлу „index.html“, што је приказано на слици 3 под линијама бр. 9 и 10.

Променљиве „start“ и „end“ приказане у функцији **AutocompliteLoad()**, садрже вредности које корисник уноси у прво или друго поље на веб страници.

Под објектом који је креиран на линији бр. 305 су декларисани, назив поља за унос (у овом случају поље „start“) и низ података који ће се користити за довршавање речи, а који је приказан на слици 8.

На линији бр. 308 је креиран исти објекат који има исту функционалност као и претходни, осим што важи за друго поље под ознаком „end“.

На слици 15 је приказано поље на коме се види функција **AutocompliteLoad()** у раду.



Слика 15 – *Аутоматско довршавање имена градова*

Функција **loadMap()** приказана на слици 16 и користи се за приказ карте на веб страници.

```
175 function loadMap(){
176     var provider = new com.modestmaps.TemplatedLayer('http://tile.openstreetmap.org/{Z}/{X}/{Y}.png');
177     var map = new com.modestmaps.Map('map', provider);
178     map.setCenterZoom(new MM.Location(37.811530, -122.2666097), 2);
179     return map
180 }
```

Слика 16 – *Функција за приказ карте на веб страници*

За потребе овог пројекта је коришћена библиотека „modestmaps“ која пружа услуге контролисања и приказивања карти света у програмском језику javascript. [2]

Функција за приказ карте прво смешта провајдера карте у променљиву „provider“. Провајдер карте је http://tile.openstreetmap.org/{Z}/{X}/{Y}.png. Након доделе провајдера се креира карта и смешта у променљиву „map“.

На линији бр. 178 је изабрана прозвољна локација на карти из разлога активације карте која се иначе не би догодила, па карта не би била приказана на веб страници без акције корисника. Други параметар на линији бр. 178 представља величину увећања карте приказане на веб страници (у овом случају је увећање, вредности 2).

Карта се овако подешена враћа функцији за даљу употребу.

Након прве две активиране функције (**AutocompliteLoad()** и **loadMap()**), се у променљивој „graph“ декларише граф у којем са налазе сви градови који су укључени у претрагу најоптималније путање.

Променљива „graph“ је приказана на слици 17.

Ради лакшег објашњавања графа, узет је пример аеродрома у Москви који је први декларисан на слици 17. Аеродром у Москви је повезан у овом случају са градовима: Пекингом, Кијевом, Варшавом и Хелсиникијем. Метода **getDistance** има вредност растојања између два задата града у километрима.

Овако постављен граф је потпуно произвољно изабран. Даље се сваки аеродром повезује са следећим призвољно изабраним.

Оваква структура података која је приказана на слици 17, се може превести у графички облик.

Графички приказан граф се може видети за:

аеродроме у Азији на слици 18,

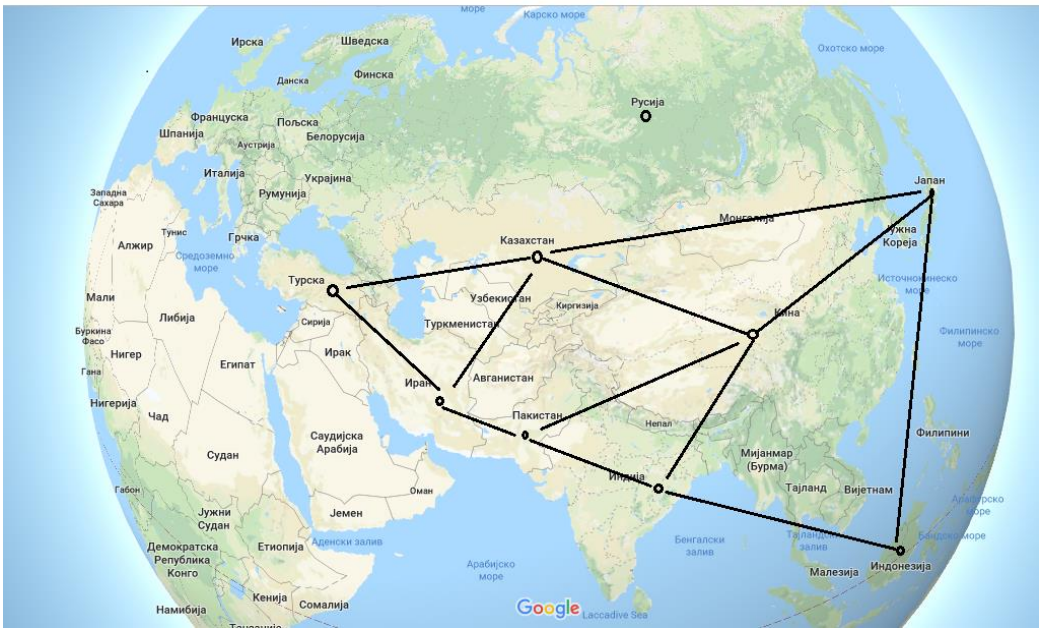
аеродроме у Европи на слици 19,

аеродроме у јужној и северној Америци на слици 20 и

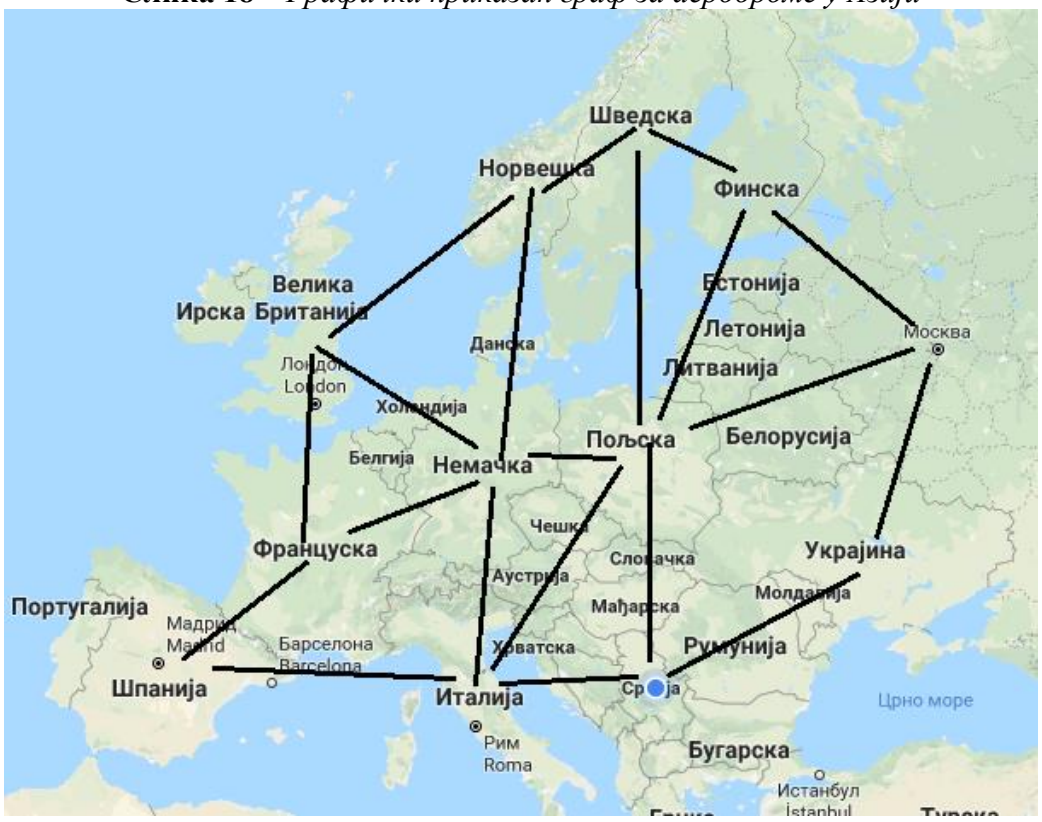
сва три међусобно повезана на слици 21.

```
graph = {
    Moscow: {
        Beijing: getDistance("Moscow", "Beijing"),
        Kiev:  getDistance("Moscow", "Kiev"),
        Warsaw:  getDistance("Moscow", "Warsaw"),
        Helsinki: getDistance("Moscow", "Helsinki") //Finland
    },
    Kiev: {
        Istanbul: getDistance("Kiev", "Istanbul"),
        Beijing: getDistance("Kiev", "Beijing"),
        Belgrade: getDistance("Kiev", "Belgrade"),
        Moscow: getDistance("Kiev", "Moscow")
    },
```

Слика 17 – *Структура података графа аеродрома*



Слика 18 – *Графички приказан граф за аеродроме у Азији*



Слика 19 – *Графички приказ повезаних аеродрома у Европи*

Слика 20 – *Графички приказ повезаних аеродрома у јужној и северној Америци*

Слика 21 – *Графички приказ међусобно повезаних аеродорма на четири континента*

Функција **getDistance,** која је позвана у променљивој „graph“ се види на слици 22

```
182 function getDistance(location1,location2){
183     var i = search(location1);
184     var j = search(location2);
185     return calcDistance(d[i].Latitude, d[i].Longitude, d[j].Latitude, d[j].Longitude);
186 }
```

Слика 22 – *Функција* ***getDistance***

Функција **getDistance** прима два параметра „location1“ и „location2“. На линији бр. 183 позива функцију „search“ приказану на слици 23 .

Функција „search“ прима један параметар „location“ којим се представља град који се пртражује.

Претрага се врши у фајлу „Airports.js“, коју предствља променљива „d“. Уколико је претрага извшена успешно, враћа се индекс на ком месту се налази у низу град који је тражен.

```
189 function search(location){
190     for (var i = 0; i < d.length; i++) {
191         if(d[i].City === location){
192             return i;
193         }
194     }
195 }
```

Функција 23 – *Функција за претрагу градова*

Након успешне претраге се у функцији **getDistance** на линијама бр. 183 и 184, додељују вредности индекса променљивама „i“ и „j“, на коме се налазе тражени градови.

Последња ставка на линији бр 185 у функцији **getDistance** представља метода **calcDistance** која је приказана на слици 24 и која служи за израчунавање растојања између два тражена града.

```
197 function calcDistance(lat1, lon1, lat2, lon2){
198     var l1 = new MM.Location(lat1, lon1);
199     var l2 = new MM.Location(lat2, lon2);
200     return MM.Location.distance(l1, l2, 6371);
201 }
```

Слика 24 – *Функција за израчунавање растојања између два задата града*

Функција **calcDistance** прима параметре географске дужине и ширине за оба града. На линијама бр. 198 и 199 се локације првог и другог града смештају у променљиве „l1“ и „l2“. Функцији се на линији бр. 200, враћа вредност растојања између два града у километрима. Растојање израчунава функција **distnace(location1, location2, r)** која се

налази у библиотеци „modestmaps“. (параметар „r“ је средњи полупречник земље који износи 6371km).

Вредност растојања између две локације се на крају враћа у функцију **getDistance** и користи се у променљивој „graph“.

Овим се завршавају све операције учитавања веб странице.

Веб страница је сада у „standby“ моду док год корисник не изабере два града између којих ће се лет авиона извшити.

Извшавање кôда се наставља када корисник одабере оба града и кликне на дугме „Submit“, при чему се позива метода **main()** приказана на слици 25.

```
384 function main() {
385
386     optimalRoute = [];
387     dijsktraLoad();
388     var map = loadMap();
389
390     var canvas = document.createElement('canvas');
391     canvas.style.position = 'absolute';
392     canvas.style.left = '0';
393     canvas.style.top = '0';
394     canvas.width = map.dimensions.x;
395     canvas.height = map.dimensions.y;
396     map.parent.appendChild(canvas);
397
398     var arcCo = GenerateArcCoordiantes();
399     arcDraw(arcCo, map, canvas);
400     displayHtml();
401 }
```

Слика 25 – *Метода која се позива након клика на „Submit“ дугме*

На линији бр. 386 се уверевамо да је низ „optimalRoute“ испражњен приликом сваког новог избора градова од стране корисника. Након тога на линији бр. 387 се позива функција **dijkstraLoad** приказана на слици 26 .

```
314 function dijsktraLoad() {
315     var start_city = document.getElementById("start").value;
316     var end_city = document.getElementById("end").value;
317
318     //dijstra is initilazied to find best route based on givn graph
319     var path = dijkstra(graph, start_city, end_city);
320     //varibale used to display route in html
321     route = path["path"];
322
323     //pronalazimo svaku pojedinu lokaciju aerodroma koju je dajkstrin algoritam nasao.
324     for (var i = 0; i < path["path"].length; i++) {
325         searchAirports(path["path"][i]);
326     }
327 }
```

Слика 26 – *Функција за проналажење оптималне путање*

На линији бр. 315 и 316 се преузимају градови у облику текста, које је корисник изабрао.

На линији бр. 319 се позива функција **dijkstra(graph, start_city, end_city),** која је Дајкстрин алгоритам приказан на сликама 27а и 27б. Први параметар Дајкстриног алгоритма је граф, други и трећи параметар представљају почетни и одредишни град у коме се налази аеродром.

```
238 const dijkstra = (graph, startNodeName, endNodeName) => {
239     // track the lowest cost to reach each node
240     let costs = {};
241     costs[endNodeName] = "Infinity";
242     costs = Object.assign(costs, graph[startNodeName]);
243 
244     // track paths
245     const parents = {endNodeName: null};
246     for (let child in graph[startNodeName]) {
247         parents[child] = startNodeName;
248     }
249     // track nodes that have already been processed
250     const processed = [];
251 
252     let node = lowestCostNode(costs, processed);
253 
254     while (node) {
255         let cost = costs[node];
256         let children = graph[node];
257         for (let n in children) {
258             if (String(n) !== String(startNodeName)) {
259                 let newCost = cost + children[n];
260                 if (!costs[n] || costs[n] > newCost) {
261                     costs[n] = newCost;
262                     parents[n] = node;
263                 }
264             }
265         }
266         processed.push(node);
267         node = lowestCostNode(costs, processed);
268     }
```

Слика 27а – *Први део функције* ***dijkstra***

```
270     let optimalPath = [endNodeName];
271     let parent = parents[endNodeName];
272     while (parent) {
273         optimalPath.push(parent);
274         parent = parents[parent];
275     }
276     optimalPath.reverse();
277 
278     const results = {
279         distance: costs[endNodeName],
280         path: optimalPath
281     };
282     return results;
283 };
```

Слика 27б – *Други део функције* ***dijkstra***

На линији бр. 240 је дефинисан празан низ који ће садржати све „цене“. На линијама бр. 241 је дефинисано да последњи чвор има вредност „бесконачно“, док су на линији бр. 242 преузете „цене“ првог чвора и копиране у низ „цена“.

На линији бр. 245 је дефинисан објекат у коме се смештају „родитељи“ чворова. На линијама од бр. 246 до 248, је додељен „родитељ“ сваком „детету “почетног чвора.

На линији бр. 252 је дефинисан низ којим се прате обрађени чворови.

На линији бр. 252 се позива функција „lowestCostNode(costs, processed)“ која је приказана на слици 28.

```
225 const lowestCostNode = (costs, processed) => {
226     return Object.keys(costs).reduce((lowest, node) => {
227 
228         if (lowest === null || costs[node] < costs[lowest]) {
229             if (!processed.includes(node)) {
230                 lowest = node;
231             }
232         }
233         return lowest;
234     }, null);
235 };
```

Слика 28 – *Функција за проналажење следећег чвора са најмањом „ценом“*

Функција „lowestCostNode“ преузима вредност „цене“ доласка до чворова и листу обрађених чворова и враћа следећи чвор до кога је најмања „цена“ доласка.

Након тога се на линији бр. 254 у функцији „dijkstra“, активира while петља у којој се констатно тражи следећи чвор са најнижом „ценом“.

Објашњење процеса у while петљи је следеће:

1. Добавља се „цена“ тренутог чвора
2. Добављају се сва „даца“ тренутног чвора
3. Пролази се кроз свако „дете“ чвора и рачуна се цена доласка до тог чвора („детета“) . Уколико се пронађе јефтинији чвор или је само тај један доступан онда се он додаје објекту „costs“ .
4. Такође се додаје тренутни чвор објекту „parents“
5. Када се чвор обради додаје се низу „processed“ да би се осигурали да се не обрађује исти чвор поново.
6. Тражимо нови „најефтинији“ чвор који није обрађен и процес се понавља.

По завшетку while петље, се у другом делу функције која је приказана на слици 27б, декларише „optimalPath“ променљива и додељује се родитељ последњем чвору.

Потом се покреће нова while петља у којој се додељују родитељи у оптималну путању.

На линији бр. 276 се вредностима које се налазе у „optimalPath“ променљивој, обрће редослед.

Након тога се на линији бр. 278. декларише објекат „results“ који се враћа функцији као резултат који садржи најоптималнију путању којом би се лет авиона обавио између полазног и одредишног аеродрома.

Овим се завршава обрада функције „dijkstra“ и извршавање се наставља у функцији **dijkstraLoad** приказаној на слици 26, где се пронађени аеродроми који су налазе на оптималној путањи смештају у променњиву „route“.

За сваки од аеродрома који су у оптималној путањи се налази географска ширина и дужина и поставља у променљиву „optimalRoute“ .

Извршавање кôда се наставља у функцији **main()** приказаној на слици 25 , где се преузима карта и поставља у променљиву „map“ и такође се креира ткзв. „canvas” на коме се карта налази на веб страници. На линији бр. 396 се поставља карта на „canvas“ .

Потом се позива функција **GenerateArcCoordiantes()** која је приказана на слици 29

```
323 function GenerateArcCoordiantes() {
324     var locations = [];
325     for (var i = 0; i < optimalRoute.length - 1; i++) {
326 
327         var firstCoordinate = new com.modestmaps.Location(optimalRoute[i][0], optimalRoute[i][1]);
328         var secondCoordinate = new com.modestmaps.Location(optimalRoute[i + 1][0], optimalRoute[i + 1][1]);
329 
330         for (var j = 0; j <= 100; j++) {
331             var f = j / 100;
332             locations.push(com.modestmaps.Location.interpolate(firstCoordinate, secondCoordinate, f));
333         }
334     }
335     return locations;
336 }
```

Слика 29 – *Функција за генерисање координата по којима се црта лук*

Функција **GenerateArcCoordiantes()** у променљиву „firstCoordinate” смешта координату првог аеродрома (листа аеродрома коју је Дајкстрин алгоритам изабрао), а у променљиву „secondCoordinate“ кооридинату другог аеродрома. Потом се у петљи узима вредност дељења сваког броја од 0 до 100 са 100. (0/100, 1/100, 2/100..) и сваки резултат дељења се користи за интерполацију вредности координата првог и другог аеродрома. На крају се све вредности смештају у низ и враћају функцији за поновно коришћење.

Извршавање кôда се наставља на линији бр. 399 у фунцкији **main()**, где се позива функција **arcDraw(arcCo, map, canvas)** која је приказана на слици 30.

Функција **arcDraw** преузима координате које је функција **GenerateArcCoordiantes** генерисала, карту и „canvas“. На линији бр. 341 се фокусира путања по којој авион лети и праказује на веб страници. Потом се декларише функција **redraw()**.

Објашњење функције **redraw()** :

1. Брише се „canvas“.
2. Декларише линија за исцртавање путање.
3. Почетак исцртавања путање.
4. Преузима се прва координата и исцртава на карти.
5. У петљи се исцртава свака следећа координата.
6. Завршава се исцртавање путање.

Последњи део методе од бр. 358 до 365 има функцију цртања нових координата и брисања старих уколико било која акција изврши над картом (транслација, увећање или умањење).

```
339 function arcDraw(arcCoordinates, map, canvas) {
340
341     map.setExtent(arcCoordinates);
342     var ctx = canvas.getContext('2d');
343     function redraw() {
344         ctx.clearRect(0, 0, canvas.width, canvas.height);
345
346         ctx.strokeStyle = '#404040';
347         ctx.beginPath();
348         var p = map.locationPoint(arcCoordinates[0]);
349         ctx.moveTo(p.x, p.y);
350         for (var i = 1; i < arcCoordinates.length; i++) {
351             p = map.locationPoint(arcCoordinates[i]);
352             ctx.lineTo(p.x, p.y);
353         }
354
355         ctx.stroke();
356     }
357
358     map.addCallback('drawn', redraw);
359     map.addCallback('resized', function () {
360         canvas.width = map.dimensions.x;
361         canvas.height = map.dimensions.y;
362         redraw();
363     });
364
365     redraw();
366
367 }
```

Слика 30 – *Функција за исцртавање путање авиона*

Последњи део фајла „app.js“ и уједно последњи део функције **main()** је фунцкија **displayHtml()** приказана на слици 31, која уређује информације о времену лета и растојању између аеродорма.

Функција приказана на слици 31 преузима све аеродроме на релацији лета и форматира их тако да уколико има више од 4 аеродрома, текст буде преломљен. Функција такође преузима и исписује вредност укупног растојања између почетне и одредишне дестинације и време потребно за лет (брзина авиона 1000km/h). Ове информације су приказане у квадрату са информацијама (слика 13).

```
202 function displayHtml(){
203     var string = "";
204     var fullDistance = 0;
205     for (var i = 0; i < route.length-1; i++) {
206         if(i == 4){
207             string += "\n";
208         }
209         string += route[i+1] + "->";
210 
211         fullDistance += getDistance(route[i],route[i+1]);
212     }
213     string = string.slice(0, -2);
214     document.getElementById('t1').textContent = string;
215     document.getElementById('t2').textContent = Math.round(fullDistance) + " km";
216     var averageSpeedOfPlane = 1000;
217     document.getElementById('t3').textContent = Math.round(fullDistance/averageSpeedOfPlane) + " h";
218 
219 }
```

Слика 31 – *Функција за приказивање времена и растојања лета*

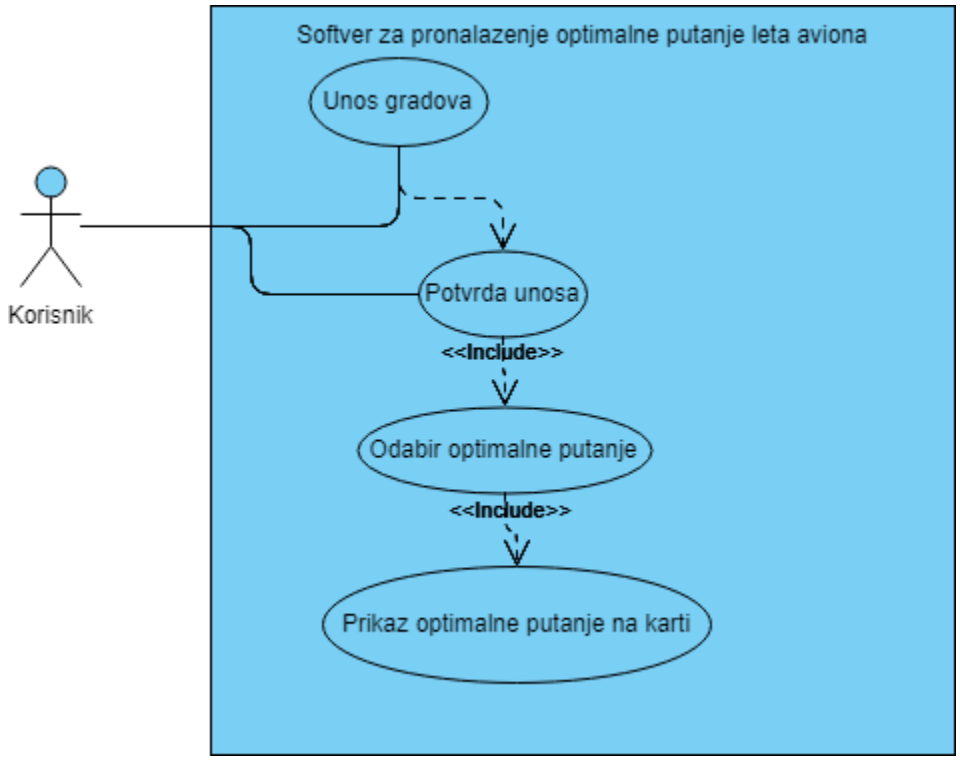
„UML“ дијаграми

Представљени су „use case“ дијаграм и дијаграм активности.

Остали дијаграми су изостављени јер кôд веб странице не садржи методе објектно оријентисаног програмирања.

„Use case“ дијаграм

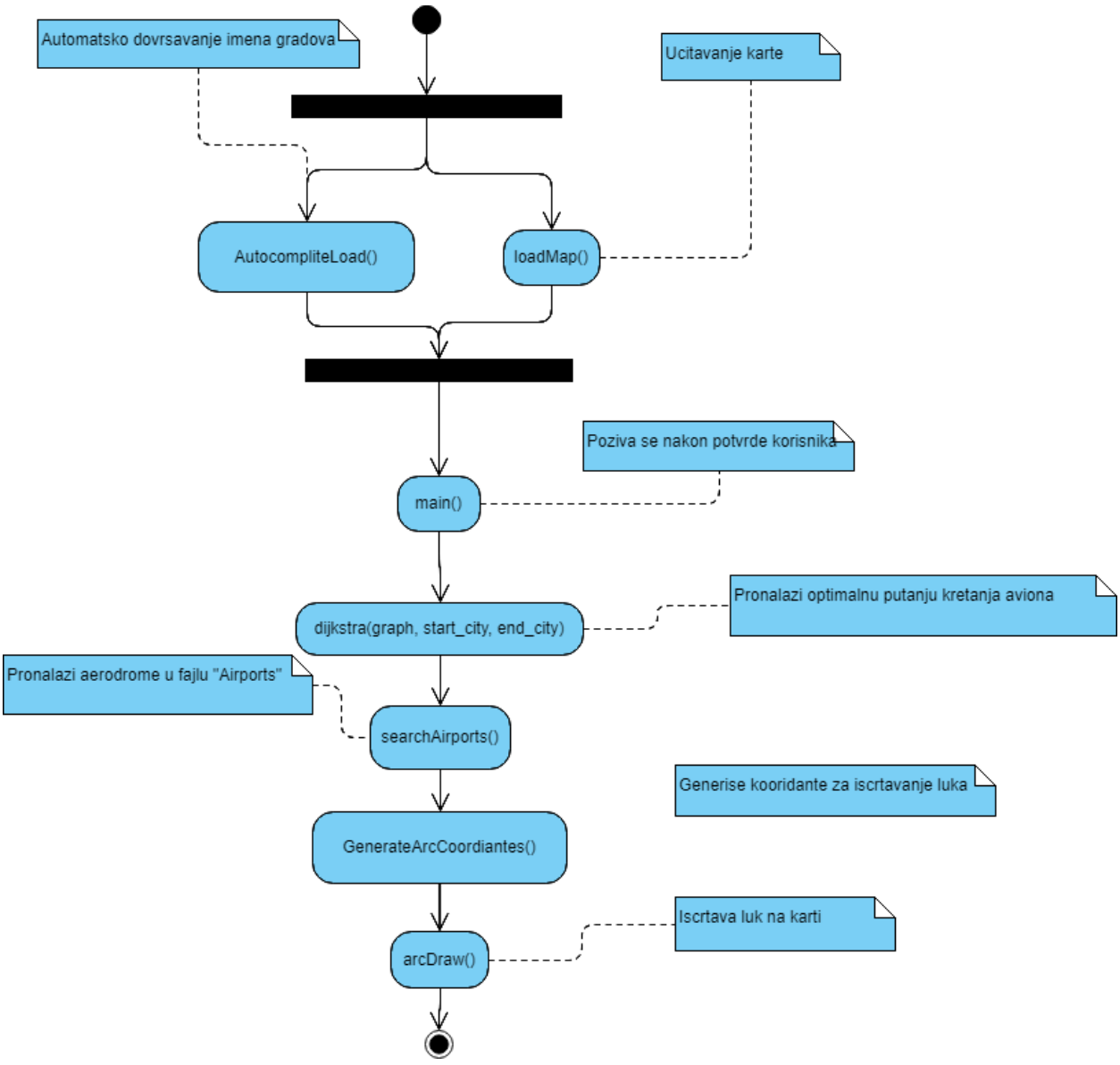
На слици 32 је приказан једноставан „use case“ дијаграм који представља интеракцију корисника са веб страницом, као и ток најбитнијих операција веб странице.



Слика 32 – „*Use case“ дијаграм*

Дијаграм активности

На слици 33 је приказана дијграм активности на коме се виде најважније функције које се налазе у кôду и редослед њиховог извршавања.

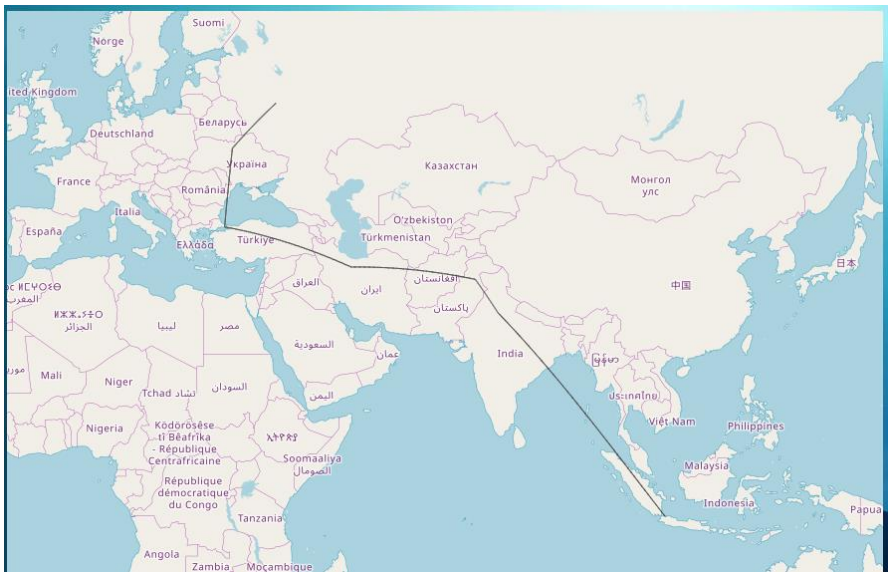


Слика 33 – *Дијаграм активности*

Закључак

Софтвер приказан у раду је солидан пример технологија које су у времену писања овог рада најзаступљеније.

Софтвер је могуће побољшати тиме што би се додали сви аеродроми света према чему се не би догађале нелогичности, као што је приказано на слици 34, са које се види да авион лети путањом која очито није оптимална. Ово би се решило додавањем већег броја аеродорма из којег би алгоритам могао да изабере оптималнију путању од приказане.



Слика 34 – *Путања која није потпуно оптимална*

Дајкстрин алгоритам има временску сложеност $O(V^2)$ где је V, број чворова графа. Према наведеном ово није најбољи алгоритам који је могуће употребити на овом проблему.

Проблем који је описан у раду не захтева бољи алгоритам за претраживање графа, али уколико се број градова и аеродрома повећа време извршавања би било доста захтевније па је најбоље решење употрбити A* алгоритам који има временску сложеност O(V).

Даљи развој софтвера приказаног у раду захтева да наведене измене буду имплементиране и да се уједно цео пројекат преведе у објектно оријентисани модел, чиме би се даљи развој знатно олакшао.

Веома је битно напоменути да подакте из фајлова „Locations.js“ и „Airports.js“ је могуће поставити у базу података по избору и тиме олакшати приступ подацима о аеродромима и уједно заштити податке од неауторизованог приступа.

Дизајн веб странице је могуће унапредити, али тренутни изглед не утуче на функционалност софтвера.

Прилог А

Дајкстрин алгоритам

Услови:

- Сва растојања морају бити позитивна
- Чворови графа морају бити повезани
- Може се применити на усмерене и неусмерене графове

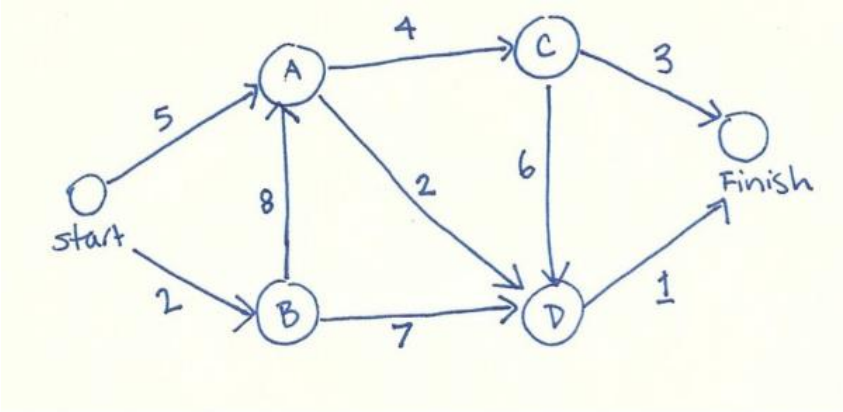
Алгоритам:

Именујмо чвор од кога почињемо, почетним чвором. Кажимо да је дужина чвора Y, дужина од почетног чвора до чвора Y. Дајкстрин алгоритам ће доделити неке иницијалне дужине и оптимизоваће се, корак по корак.

1. Доделити сваоком чвору дужину доласка. Првом чвору доделити вредност, нула. Свим осталим чворовима доделити вредност „бесконачно“.
2. Означити све чворове као непосећене. Поставити почетни чвор као тренутни. Креирати листу непосећених чворова који ће садржати све чворове.
3. За тренутни чвор размотрити све његове непосећене „комшије“ и израчунати дужину до сваког „комшије“.
4. Када се дужине до сваког „комшије“ израчунају, тренутни чвор се означи као посећен и уклони из листе непосећених чворова. Овај чвор неће бити поново провераван.
5. Узети следећи чвор до кога је растојање најкраће и поставити га као нови тренутни чвор. Поновити од корака 3 .
6. Уколико су сви чворови посећени алгоритам се завршава.

Пример:

На слици 35 је приказан граф, где су чворови: start, A, B, C, D, finish.



Слика 35 – *Пример графа*

Налазимо најкраће растојање чвора start до чвора finish.

За почетак је потребно направити листу (табела 1) свих чворова, свих родитеља и свих „цена“ доласка до чворова.

Табела 1 – *Табела којом се прати путања*

Родитељ	Чвор	Цена
/	start	0
	A	∞
	B	∞
	C	∞
	D	∞
	finish	∞

Потом одабрати почетни чвор „start“, израчунати растојања до сваког „комшије“ овог чвора и попунити табелу.

Табела 2 сада садржи следеће податке:

Табела 2 – *Табела којом се прати путања*

Родитељ	Чвор	Цена
/	start	0
start	A	5
start	B	2
	C	∞
	D	∞
	finish	∞

Овим се означава чвор „start“ као посећен и неће се поново обрађивати.

Бирамо из табеле чвор који је **непосећен** и са **најмањом ценом** доласка до њега. Ово је чвор „B“.

Чвор „B“ постаје нови тренутни чвор. Сада се табела попуњава у односу на њега и његове комшије, па се табела ажурира тако што се од саберу растојања од чвора „start“ до сваког комшије чвора „B“. Приметићемо да од чвора „start“ до „A“, преко „B“ растојање је 2+8 = 10, што је веће од тренутног растојања које 5. Ово растојање неће бити ажурирано.

Ажурирани подаци су приказани у табели 3.

Табела 3 – *Табела којом се прати путања*

Родитељ	Чвор	Цена
/	start	0
start	A	5
start	B	2
	C	∞
B	D	9
	finish	∞

Означавмо чвор „B“ као посећен.

Бирамо из табеле чвор који је **непосећен** и са **најмањом ценом** доласка до њега. Ово је чвор „A“.

Понављамо поступак.

Ажурирамо табелу са новим вредностима која се рачунају преко чвора „A“. Проверавамо све „цене“ доласка до комшија чвора „A“. Ажурирани подаци су приказани у табели 4.

Примећујемо да се вредност доласка до чвора „D“, самњила и тренутно преко чвора „A“ износи 7. Па у табели постављамо да је краће растојање преко чвора „A“, уместо преко „B“. Такође долазак до чвора „C“ је сада вредности 9.

Табела 4 – *Табела којом се прати путања*

Родитељ	Чвор	Цена
/	start	0
start	A	5
start	B	2
A	C	9
A	D	7
	finish	∞

Означавамо чвор „A“ као посећен чвор.

Кораци се понављају док се сви чворови не обаред ,па се тако долази до коначног облика табеле :

Табела 5 – *Коначни облик табеле*

Родитељ	Чвор	Цена
/	start	0
start	A	5
start	B	2
A	C	9
A	D	7
D	finish	10

Подаци у у табели показују да је „цена“ доласка до чвора „finish“ : 10 . Док се путања налази уочавањем у табели повезаних чворова, кретањем од последње врсте.

Последња врста садржи чвор „finish“, чији је родитељ „D“.

Врста бр. 5 садржи чвор „D“, чији је родитељ „A“.

Врста бр. 2 садржи чвор „A“, чији је родитељ „start“.

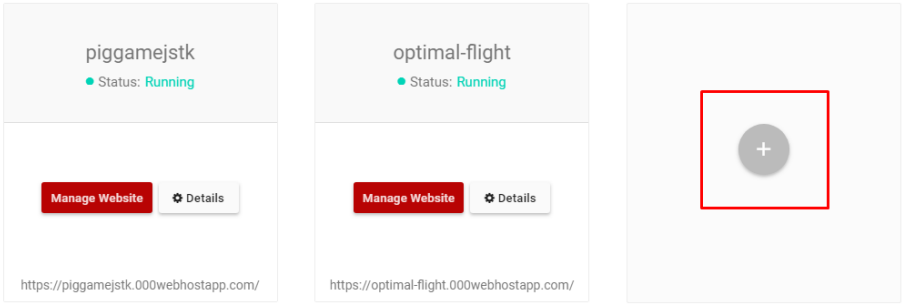
Према наведеном, оптимална путања доласка од чвора „start“ до „finish“ је: **start, A, D, finish**.

Прилог Б

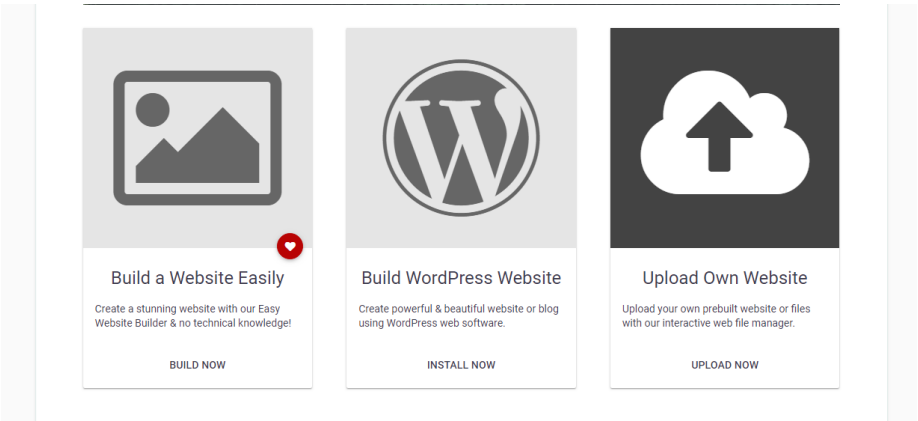
Постављање веб сајта на интернет (hosting)

Потребно је посетити неки од сајтова који пружају бесплатан веб „hosting“. У овом случају је одабран: https://www.000webhost.com.

Потребно је посетити овај веб сајт и регистривати се. Након регистрације потребно је одабрати опцију приказану на слици 36 која је уоквирена црвеним квадратом. Потом одабрати опцију „Uload Own Website“ приказану на слици 37 .

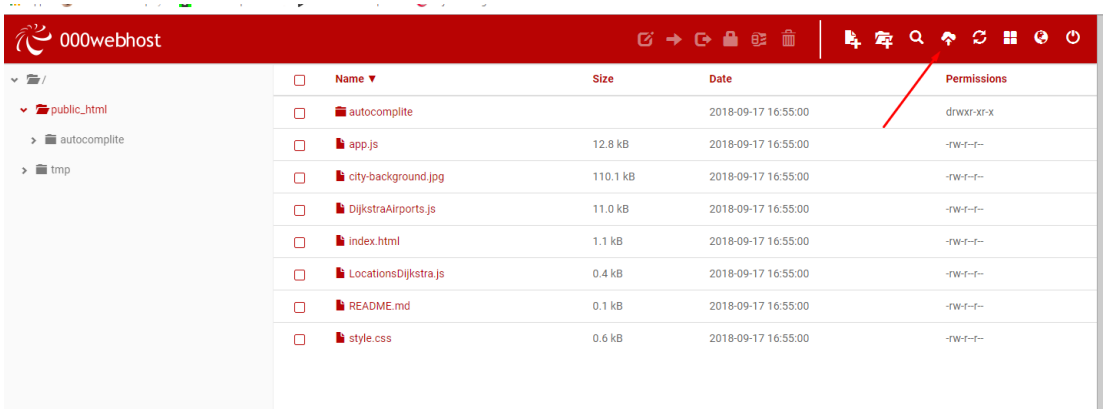


Слика 36 – *Постављање веб сајта на „hosting“*



Слика 37 – *Опција за отпремање фајлова*

Следећи корак је отпремање фајлова, кликом на иконицу на коју указује стрелица са слике 38 . Отпремање је врло једноставно и своди се на одабир фајлова веб странице који је чине.



Слика 38 – *Отпремљени фајлови веб странице*

Уколико су кораци који су наведени исправно извршени, сада је могуће приступити веб страници на линку: https://optimal-flight.000webhostapp.com .

Литература

[1] http://modestmaps.com , август, 2018.

[2] Aditya Y. Bhargava **grokking algorithms,** Manning Publications Co. 20, Baldwin Road Shelter Island

[3] https://developer.mozilla.org/bm/docs/MDN/ , август, 2018

[4] https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-greedy-algo-7/, август, 2018

[5] Martin Fowler, Kendall Scott , **UML Distilled Second Edition**, Addison Wesley Second Edition August 18, 1999