Izrada web aplikacije za prikaz podataka o prometnim nesrećama

Kukor, Marko

Undergraduate thesis / Završni rad

2023

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Zagreb, Faculty of Transport and Traffic Sciences / Sveučilište u Zagrebu, Fakultet prometnih znanosti**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:119:181269

SVEUČILIŠTE U ZAGREBU

FAKULTET PROMETNIH ZNANOSTI

Marko Kukor

Izrada web aplikacije za prikaz podataka o prometnim nesrećama

Završni rad

Zagreb, srpanj 2023.

SVEUČILIŠTE U ZAGREBU

FAKULTET PROMETNIH ZNANOSTI

Završni rad

Izrada web aplikacije za prikaz podataka o prometnim nesrećama

Development of Web Application for Traffic Accidents Report

Mentor: dr. sc. Tomislav Erdelić

Student: Marko Kukor

JMBAG: 0135260870

Zagreb, srpanj 2023.

Sažetak:

Ovaj završni rad rješava problem vizualizacije podataka o prometnim nesrećama. U relacijsku bazu podataka koristeći SQL Server spremljeni su sirovi podaci o prometnim nesrećama na prostoru Republike Hrvatske. Za prikaz podataka izrađena je ASP .NET web aplikacija koja ima interaktivne elemente za rad s podacima. Kako bi se podaci bolje prikazali koristi se prometna digitalna karta (Open Street Map). Web aplikacija je povezana s bazom podataka i po potrebi dohvaćaju se odgovarajući podaci za prikaz u web aplikaciji. Interaktivni elementi aplikacije uključuju filtriranje i grupiranje podataka po određenim atributima. Cilj ovog završnog rada je povezivanje web aplikacije s predputnim informiranjem. Web aplikacija korisnicima daje informacije o lokaciji i učestalosti nastajanja prometnih nesreća. Te informacije služe korisnicima za bolje planiranje ruta.

KLJUČNE RIJEČI: SQL, MVC, web aplikacija, prometne nesreće

Summary:

This final paper solves the problem of visualizing traffic accident data. Raw data on traffic accidents in the Republic of Croatia were stored in the relational database using SQL Server. An ASP .NET web application was created to display the data, which has interactive elements for working with the data. In order to better display the data, a traffic digital map (Open Street Map) is used. The web application is connected to the database and, when necessary, appropriate data is retrieved for display in the web application. Interactive elements of the application include filtering and grouping data by certain attributes. The goal of this final work is to connect the web application with pre-travel information. The web application provides users with information about the location and frequency of traffic accidents. This information serves users for better route planning.

KEY WORDS: SQL, MVC, web application, traffic accidents

SVEUČILIŠTE U ZAGREBU
FAKULTET PROMETNIH ZNANOSTI
ODBOR ZA ZAVRŠNI RAD

Zagreb, 5. svibnja 2023.

Zavod: **Zavod za inteligentne transportne sustave**
Predmet: **Baze podataka**

ZAVRŠNI ZADATAK br. 7065

Pristupnik: **Marko Kukor (0135260870)**
Studij: Inteligentni transportni sustavi i logistika
Smjer: Inteligentni transportni sustavi

Zadatak: **Izrada web aplikacije za prikaz podataka o prometnim nesrećama**

Opis zadatka:

Cilj rada je izraditi interaktivnu web aplikaciju za prikaz prometnih nesreća u Republici Hrvatskoj. Prvotno je podatke o prometnim nesrećama dobivenih u tekstualnom obliku potrebno proučiti te potom dizajnirati relacijsku shemu baze podataka i spremiti podatke u bazu podataka. Potom je potrebno bazu podataka povezati s web aplikacijom. Nastavno na navedeno, potrebno je u web aplikaciju dodati digitalnu prometnu kartu i implementirati dohvat i prikaz podataka o prometnim nesrećama unutar karte. Dodatno, potrebno je dodati interaktivne funkcionalnosti u web aplikaciju s ciljem lakše vizualizacije i obrade podataka, poput: kategorizacije, selekcije određenog područja, prikazom detalja za pojedinu prometnu nesreću i slično. Na kraju je potrebno provesti kratku analizu podataka spremljenih u bazu podataka.

Mentor:

Tomislav Erdelić

dr. sc. Tomislav Erdelić

Predsjednik povjerenstva za završni ispit:

Sadržaj

1. Uvod

Zbog sve veće važnosti prometa i transporta u ljudskim životima dolazi do potrebe za povećanjem njegove učinkovitosti. Tu potrebu rješavaju Inteligentni transportni sustavi (ITS) koji su nadogradnja već postojećeg prometnog i transportnog sustava. Cilj ITS je poboljšati performanse, odvijanje prometa, učinkovitiji transport putnika i robe, poboljšati sigurnost u prometu, udobnost i zaštitu putnika. Glavna funkcionalna područja ITS-a su: informiranje putnika, upravljanje prometom i operacijama, vozila, prijevoz putnika, javni prijevoz, žurne službe, elektronička plaćanja vezana za transport, sigurnost osoba u cestovnom prijevozu, nadzor vremenskih uvjeta i okoliša, upravljanje odzivom na velike nesreće i nacionalna sigurnost i zaštita, (1).

Područje informiranja putnika može se podijeliti na predputno informiranje i putno informiranje. Web aplikacija koja je napravljena u ovom završnom radu spada u predputno informiranja putnika. Predputno informiranje omogućuje korisnicima da iz svog doma ili radnog mjesta dođe do važnih informacija o raspoloženim modovima, vremenu ili cijenama putovanja, (1).

Cilj ovog završnog rada je prikupljene podatke vezane za prometne nesreće vizualizirati na karti kako bi korisnici mogli koristiti te informacije za planiranje svojih putovanja. Web aplikacija može se koristiti kako bi se prikazale prometnice koje imaju veći broj prometnih nesreća što bi koristilo prometnim inženjerima za poboljšanje tih prometnica.

U ovom završnom radu obraditi će se tema vizualizacije podataka prometnih nesreća. Sama će se vizualizacija napraviti uz pomoć web aplikacije. Web aplikacija koristiti će Model-View-Controller (MVC) okruženje koje se sastoji od tri komponente: modela, pogleda i kontrolera. Kako bi se podaci unijeli u bazu podataka koristiti će se program za unos podataka u bazu podatka. Taj program biti će napravljen u Visual Studiu uz pomoć C# programskog jezika.

Rad se sastoji od šest poglavlja, ta poglavlja su:

1. Uvod
2. Podaci i baza podataka
3. Program za upisivanje sirovih podataka u bazu podataka
4. Web aplikacija
5. Analiza dobivenih rezultata
6. Zaključak

U drugom poglavlju opisana je struktura podataka koji će se koristiti u ovom završnom radu. Korištenjem programa yEd Graph Editor-a napraviti će se dijagram entiteta i veza koji će služiti za izradu relacijskog modela baze podataka. U ovom poglavlju navedena se Codd-ova pravila kojom se opisuju relacijske baze podataka. Također su objašnjeni osnovni pojmovi koji se koriste u bazi podataka, neki od pojmova su primarni ključevi, NULL vrijednosti, sekundarni ključ ali i stvaranje same baze podataka.

U trećem poglavlju opisan je program za upisivanje podataka u bazu podataka. Za to je korišten program Visual Studio te je programski kôd napisan u C# programskom jeziku.

U četvrtom poglavlju opisan je MVC model te sam programski kôd za izradu web aplikacije. Opisan je i izgled korisničkog sučelja.

U petom se poglavlju analiziraju dobiveni rezultati ovog rada. Za analizu koriste se upiti napisani u SQL Server. Dobiveni podaci prikazani su pomoću grafova.

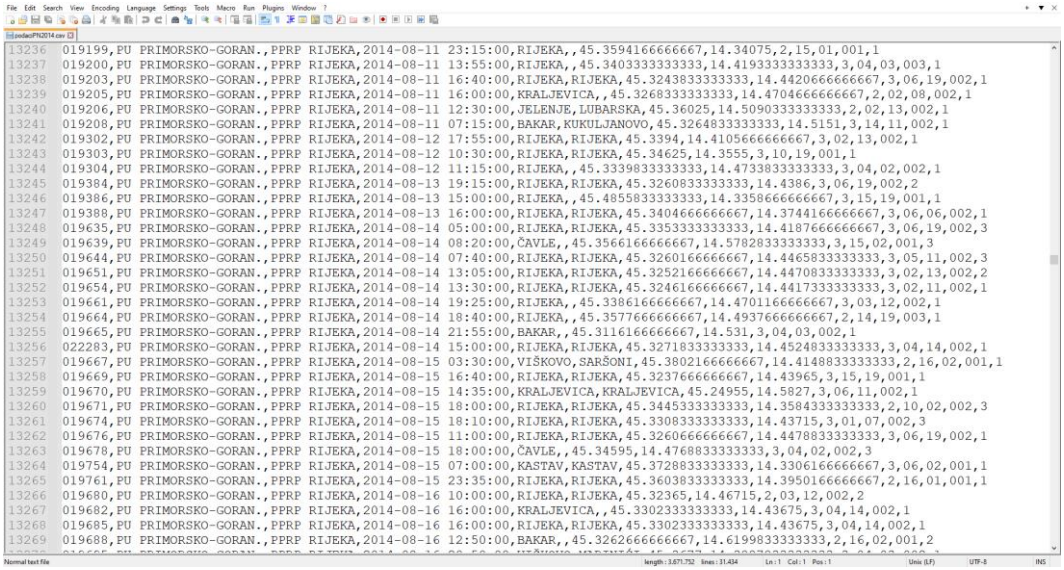
U šestom poglavlju dan je zaključak ovog završnog rada.

2. Podaci i baza podataka

Kako bi se mogla kreirati baza podataka potrebno je znati format dobivenih podataka.

2.1 Struktura podataka

Podaci se nalaze u tekstualnoj datoteci koja je strukturirana u CSV formatu (eng. Coma Separated Values). Tekstualna datoteka sadržava 31 433 redaka te zauzima 3.50 megabajta. Za otvaranje datoteke može se koristiti bilo koji od tekstualnih uređivača, ali ovdje je korišten program Notepad++. Izgled i struktura podataka prikazana je na slici 1.



```
13236 019199,PU PRIMORSKO-GORAN.,PPRP RIJEKA,2014-08-11 23:15:00,RIJEKA,,45.3594166666667,14.34075,2,15,01,001,1
13237 019200,PU PRIMORSKO-GORAN.,PPRP RIJEKA,2014-08-11 13:55:00,RIJEKA,,45.3403333333333,14.4193333333333,3,04,03,003,1
13238 019203,PU PRIMORSKO-GORAN.,PPRP RIJEKA,2014-08-11 16:40:00,RIJEKA,RIJEKA,45.3243833333333,14.4420666666667,3,06,19,002,1
13239 019205,PU PRIMORSKO-GORAN.,PPRP RIJEKA,2014-08-11 16:00:00,KRALJEVICA,,45.3268333333333,14.4704666666667,2,02,08,002,1
13240 019206,PU PRIMORSKO-GORAN.,PPRP RIJEKA,2014-08-11 12:30:00,JELENJE,LUBARSKA,45.36025,14.5090333333333,2,02,13,002,1
13241 019208,PU PRIMORSKO-GORAN.,PPRP RIJEKA,2014-08-11 07:15:00,BAKAR,KUKULJANOVO,45.3264833333333,14.5151,3,14,11,002,1
13242 019302,PU PRIMORSKO-GORAN.,PPRP RIJEKA,2014-08-12 17:55:00,RIJEKA,RIJEKA,45.3394,14.4105666666667,3,02,13,002,1
13243 019303,PU PRIMORSKO-GORAN.,PPRP RIJEKA,2014-08-12 10:30:00,RIJEKA,RIJEKA,45.34625,14.3555,3,10,19,001,1
13244 019304,PU PRIMORSKO-GORAN.,PPRP RIJEKA,2014-08-12 11:15:00,RIJEKA,,45.3339833333333,14.4733833333333,3,04,02,002,1
13245 019384,PU PRIMORSKO-GORAN.,PPRP RIJEKA,2014-08-13 19:15:00,RIJEKA,RIJEKA,45.3260833333333,14.4386,3,06,19,002,2
13246 019386,PU PRIMORSKO-GORAN.,PPRP RIJEKA,2014-08-13 15:00:00,RIJEKA,,45.4855833333333,14.3358666666667,3,15,19,001,1
13247 019388,PU PRIMORSKO-GORAN.,PPRP RIJEKA,2014-08-13 16:00:00,RIJEKA,RIJEKA,45.3404666666667,14.3744166666667,3,06,06,002,1
13248 019635,PU PRIMORSKO-GORAN.,PPRP RIJEKA,2014-08-14 05:00:00,RIJEKA,RIJEKA,45.3353333333333,14.4187666666667,3,06,19,002,3
13249 019639,PU PRIMORSKO-GORAN.,PPRP RIJEKA,2014-08-14 08:20:00,ČAVLE,,45.3566166666667,14.5782833333333,3,15,02,001,3
13250 019644,PU PRIMORSKO-GORAN.,PPRP RIJEKA,2014-08-14 07:40:00,RIJEKA,RIJEKA,45.3260166666667,14.4465833333333,3,05,11,002,3
13251 019651,PU PRIMORSKO-GORAN.,PPRP RIJEKA,2014-08-14 13:05:00,RIJEKA,RIJEKA,45.3252166666667,14.4470833333333,3,02,13,002,2
13252 019654,PU PRIMORSKO-GORAN.,PPRP RIJEKA,2014-08-14 13:30:00,RIJEKA,RIJEKA,45.3246166666667,14.4417333333333,3,02,11,002,1
13253 019661,PU PRIMORSKO-GORAN.,PPRP RIJEKA,2014-08-14 19:25:00,RIJEKA,,45.3386166666667,14.4701166666667,3,03,12,002,1
13254 019664,PU PRIMORSKO-GORAN.,PPRP RIJEKA,2014-08-14 18:40:00,RIJEKA,,45.3577666666667,14.4937666666667,2,14,19,003,1
13255 019665,PU PRIMORSKO-GORAN.,PPRP RIJEKA,2014-08-14 21:55:00,BAKAR,,45.3116166666667,14.531,3,04,03,002,1
13256 022283,PU PRIMORSKO-GORAN.,PPRP RIJEKA,2014-08-14 15:00:00,RIJEKA,RIJEKA,45.3271833333333,14.4524833333333,3,04,14,002,1
13257 019667,PU PRIMORSKO-GORAN.,PPRP RIJEKA,2014-08-15 03:30:00,VIŠKOVO,SARŠONI,45.3802166666667,14.4148833333333,2,16,02,001,1
13258 019669,PU PRIMORSKO-GORAN.,PPRP RIJEKA,2014-08-15 16:40:00,RIJEKA,RIJEKA,45.3237666666667,14.43965,3,15,19,001,1
13259 019670,PU PRIMORSKO-GORAN.,PPRP RIJEKA,2014-08-15 14:35:00,KRALJEVICA,KRALJEVICA,45.24955,14.5827,3,06,11,002,1
13260 019671,PU PRIMORSKO-GORAN.,PPRP RIJEKA,2014-08-15 18:00:00,RIJEKA,RIJEKA,45.3445333333333,14.3584333333333,2,10,02,002,3
13261 019674,PU PRIMORSKO-GORAN.,PPRP RIJEKA,2014-08-15 18:10:00,RIJEKA,RIJEKA,45.3308333333333,14.43715,3,01,07,002,3
13262 019676,PU PRIMORSKO-GORAN.,PPRP RIJEKA,2014-08-15 11:00:00,RIJEKA,RIJEKA,45.3260666666667,14.4478833333333,3,06,19,002,1
13263 019678,PU PRIMORSKO-GORAN.,PPRP RIJEKA,2014-08-15 18:00:00,ČAVLE,,45.34595,14.4768833333333,3,04,02,002,3
13264 019754,PU PRIMORSKO-GORAN.,PPRP RIJEKA,2014-08-15 07:00:00,KASTAV,KASTAV,45.3728833333333,14.3306166666667,3,06,02,001,1
13265 019761,PU PRIMORSKO-GORAN.,PPRP RIJEKA,2014-08-15 23:35:00,RIJEKA,RIJEKA,45.3603833333333,14.3950166666667,2,16,01,001,1
13266 019680,PU PRIMORSKO-GORAN.,PPRP RIJEKA,2014-08-16 10:00:00,RIJEKA,RIJEKA,45.32365,14.46715,2,03,12,002,2
13267 019682,PU PRIMORSKO-GORAN.,PPRP RIJEKA,2014-08-16 16:00:00,KRALJEVICA,,45.3302333333333,14.43675,3,04,14,002,1
13268 019685,PU PRIMORSKO-GORAN.,PPRP RIJEKA,2014-08-16 16:00:00,RIJEKA,RIJEKA,45.3302333333333,14.43675,3,04,14,002,1
13269 019688,PU PRIMORSKO-GORAN.,PPRP RIJEKA,2014-08-16 12:50:00,BAKAR,,45.3262666666667,14.6199833333333,2,16,02,001,2
```

Slika 1. Prikaz podataka uz pomoć Notepad++ programa

Svaki redak sadržava podatke za jednu prometnu nesreću u RH u 2014. godini. Svaki pojedinačni redak sadržava 13 vrijednosti odvojene točka-zarezom (;). Nazivi i opisi vrijednosti prikazane su tablicom 1.

Tablica 1. Opis podataka

Atribut	Opis	Primjer podataka
BrojPN	Poseban broj prometne nesreće	008499
PolUprava	Policijska uprava koja je nadležna za prometnu nesreću	PU ZAGREBAČKA
PolPostaja	Policijska postaja koja je nadležna za prometnu nesreću	PP VELIKA GORICA
DatumPN	Datum i vrijeme kada se dogodila prometna nesreća	2014-04-26 20:30:00
Opcina	Općina u kojoj se dogodila prometna nesreća	V. GORICA
Mjesto	Mjesto u kojoj se dogodila prometna nesreća	V. GORICA
Sirina	Geografska širina prometne nesreće	45.7208333333333
Duljina	Geografska dužina prometne nesreće	16.0548833333333
Posljedica	Posljedice prometne nesreće	3

Vrsta	Vrsta prometne nesreće	04
Okolnosti	Okolnosti prometne nesreće	03
BrojVoz	Broj vozila koja su bila u prometnoj nesreći	002
AtmPrilike	Vremenski uvjeti u trenutku prometne nesreće	1

BrojPN prikazuje broj prometne nesreće koji je jedinstven. *PolUprava* prikazuje policijsku upravu koja je nadležna za prometnu nesreću. Ukupno u RH ima 20 različitih policijskih uprava. *PolPostaja* prikazuje policijsku postaju koja je nadležna za prometnu nesreću, kojih ima 130 . *DatumPN* prikazuje datum i vrijeme trenutka u kojem se dogodila prometna nesreća. Format *DatumPN* je Godina-Mjesec-Dan Sat:Minuta:Sekunda, takav format je standardni format datuma i vremena u Sql Serveru, (2). *Opcina* prikazuje općinu u kojoj se dogodila prometna nesreća, kojih ima 561. *Mjesto* prikazuje mjesto u kojoj se dogodila prometna nesreća, kojih ima 2404. *Širina* i *Duljina* su koordinate prometnih nesreća koje su zapisane u decimalnom zapisu. *Posljedica* označava posljedice prometnih nesreća. *Posljedica* u sebi ima tri vrijednosti: 1 - prometna nesreća s poginulim osobama, 2 - prometna nesreća s ozlijeđenim osobama, 3 - prometna nesreća s materijalnom štetom, (3). *Vrsta* označava vrstu prometne nesreće gdje su vrijednosti od 1 do 19, gdje primjerice vrsta 1 označava sudar vozila u pokretu iz suprotnih smjerova, (4). *Okolnosti* označavaju okolnosti u kojim su se dogodile prometne nesreće gdje su vrijednosti od 1 do 33, gdje primjerice okolnost 2 označava nepropisnu brzinu, (3). *BrojVoz* prikazuje koliko je vozila bilo u prometnoj nesreći. *AtmPrilike* označavaju vremenske uvjete u trenutku prometne nesreće. U sebi sadržavaju sedam vrijednosti: 1 – Vedro, 2 – Oblačno, 3 – Kiša, 4 – Magla, 5 – Snijeg, 6 – Jutarnji mraz, 7 – Ostalo, (4).

2.2 Baza podataka

Baze podataka su skupovi povezanih podataka koji se zajedno pohranjuju bez nepotrebne ili štetne zalihosti. Za korištenje i upravljanje bazama podataka koriste se sustavi za upravljanje bazama podataka. Sustav za upravljanje bazama podataka DBMS (eng. Database Management System) je programska podrška koja omogućuje upravljanje podacima koji se nalaze u bazama podataka, (2). Sam sustav za upravljanje bazama podataka sastoji se od tri sloja: vanjska razina, logička razina, fizička razina, (2).

Vanjska razina prikazuje na koji način korisnici poimaju podatke koji se nalaze u Bazi podataka. Logička razina određuje međuovisnosti između vanjske i unutarnje razine, također ih mapira pomoću relacija, tablica i ključeva. Fizička razina prikazuje kako sustav za upravljanje bazama podataka percipira podatke, ali i određuje fizičku organizaciju podataka.

Logičku razinu definiraju modeli podataka, koji su formalni sustavi napravljeni od skupova objekata, operacija i pravila cijelosti. Oni opisuju samu strukturu baza podataka uz pomoć tipova podataka, odnosa između podataka i ograničenja.

Postoje više vrsta modela podataka, a to su: hijerarhijski model podataka, mrežni model podataka, objektni model podataka, relacijski model podataka, objektno-relacijski model podataka, dimenzijski model podataka, (2).

2.2.1. Relacijski model

Relacijski model podataka koristi se u relacijskim bazama podataka kojima upravljaju sustavi koji se nazivaju RDBMS (eng. Relation Database Management System). Takvi sustavi koriste SQL (eng. Structured Query Language) jezik za davanje upita i ažuriranje baza podataka, (2).

Najpoznatiju definiciju što čini relacijsku bazu podataka opisuju dvanaest Coddovih pravila. Edgar F. Codd je 1970. godine opisao pravila u svom radu „A Relation Model of Dana for Large Shared Dana Banks“ , (2). Ta pravila glase:

- nulto pravilo – sustav za kojeg tvrdimo da je relacijski mora upravljati bazom podataka samo na relacijski način,

- 1. pravilo – odnosi se na predstavljanje podataka gdje se sve informacije predstavljaju vrijednostima u tablicama,

- 2. pravilo – odnosi se na pravilo pristupa gdje se svakoj zapisanoj vrijednosti može pristupiti pomoću imena ili kombinacije imena tablice, primarnog ključa i atributa,

- 3. pravilo – objašnjava vrijednost NULL vrijednosti koja može zamijeniti bilo koju vrijednost,

- 4. pravilo – odnosi se na relacijski pristup online kataloga baza podataka gdje se na lokalnoj razini baza podataka opisuje na isti način kao i podaci,

- 5. pravilo – pravilo sveobuhvatnog jezika koji opisuje jezik za komunikaciju s bazom podataka koji podržava definiranje podataka i pogleda, manipulaciju podacima i administraciju,

- 6. pravilo – pravilo pogleda objašnjava da svi pogledi koji se u relacijskoj teoriji mogu ažurirati moraju se ažurirati i implementirati u model,

- 7. pravilo – pravilo koje opisuje visok nivo unosa, ažuriranja i brisanje gdje svojstva manipulacije podataka moraju podržavati unos, ažuriranje i brisanje podataka,

- 8. pravilo – pravilo nezavisnosti fizičkih podataka gdje aktivnosti koje radi korisnik prema bazi podataka ne smiju biti ovisne o fizičkom načinu spremanja podataka,

- 9. pravilo – pravilo nezavisnosti logičkih podataka kod kojeg se odnosi između tablica mogu mijenjati tako da ne utječu na aplikacije koje spajaju te tablice,

- 10. pravilo – pravilo nezavisnosti integriteta podataka gdje se sam sustav za upravljanje bazama podataka brine o integritetu podataka,

- 11. pravilo – pravilo distribuirane nezavisnosti kod kojeg aplikacija mora nastaviti operativno raditi kada se uvede distribuirana verzija sustava za upravljanje bazama podataka ili kada se distribuirana verzija centralizira,

- 12. pravilo – pravilo o nenarušavanju integriteta koje opisuje da se integritet podataka ne smije narušiti putevima u bazi podataka koji zaobilaze pravila integriteta i ograničenja, (2).

Danas većina komercijalnih sustava za upravljanjem bazama podataka koriste SQL jezik. Današnje komercijalne implementacije relacijskih modela ne ispunjavaju sva Coddova pravila, ali moraju ispunjavati najmanje dva pravila. Jedno od pravila je da komercijalne relacijske baze podataka prikazuju korisnicima svoje podatke kao relacije, prikaz u tabličnom obliku koji ima svoje retke i stupce. Drugo pravilo glasi da moraju koristiti relacijske operatore za upravljanje podacima, (5).

Elementi relacijskog modela podataka su relacije što su zapravo cijele tablice. Svaka relacija u sebi sadrži stupce i retke. Stupci predstavljaju listu atributa, koji imaju svoju domenu ili tip. Ta domena predstavlja vrstu podatka. Redci u sebi imaju n-torke, koje imaju vrijednosti svih atributa, (2).

Na slici 2. prikazana je tablica StudentPrimjer koja ima tri redaka i četiri stupaca. Stupci predstavljaju atribute koje je potrebno navesti za svakog studenta kao što su ID, Ime, Prezime i DatumUpisa. Dok svaki redak predstavlja jednog studenta što je zapravo instanca sheme. U slučaju prvog retka imamo studenta čije je ime i prezime Ivan Horvat, vrijednost njegovog id-a je 1 a upisao se na datum 1.1.2000. Podaci kao što su 1, Ivan, Horvat, 2000-01-01 zasebno čine elementarni podatak.

	ID	Ime	Prezime	DatumUpisa
1	1	Ivan	Horvat	2000-01-01
2	2	Jura	Juric	2000-04-04
3	3	Ivan	Ivic	2002-03-20

Slika 2. Prikaz tablice StudentPrimjer

Kada neki od elementarnih podataka nedostaje u tablici koristi se vrsta podatka NULL vrijednost. Prema 3. Coddovom pravilu NULL vrijednost zamjenjuje bilo koju vrstu podataka te predstavlja podatak koji nedostaje. NULL vrijednost se koristi kako bi se mogle koristiti nepoznate informacije u relacijskoj algebri. Primjer za tablicu koja ima NULL vrijednost prikazana je na slici 3.

	ID	Ime	Prezime	DatumUpisa	GradID	Email
1	1	Ivan	Horvat	2000-01-01	1	ihorvat@fpz.hr
2	2	Jura	Juric	2000-04-04	2	jura.juric@gmail.com
3	3	Ivan	Ivic	2002-03-20	1	NULL

Slika 3. Prikaz tablice StudentPrimjer

Primarni ključ je neki atribut ili skup atributa koji na jedinstveni način identificiraju svaki redak u tablici. Njega opisuje 2. Coddovo pravilo koje opisuje da se do svakog zapisanog podatka može doći pomoću primarnog ključa. Svaki primarni ključ u tablici mora zadovoljavati tri uvjeta. Ti osnovni uvjeti su: jedinstvenost, minimalnost, pravilo integriteta primarnog ključa.

Jedinstvenost pokazuje da se u tablici ne smiju postojati dva retka sa istim primarnim ključevima. Minimalnost pokazuje da ako se primarni ključ sastoji od više atributa, tada se niti jedna njegova komponenta ne može ukloniti. Uklanjanjem bilo koje njegove komponente dolazi do narušavanja pravila jedinstvenosti. Pravilo integriteta primarnog ključa opisuje da niti jedna komponenta primarnog ključa ne smije imati NULL vrijednost, već mora imati kardinalitet(1,1) što znači da mora imati samo jednu vrijednost. Prva znamenka označava da mora imati napisanu vrijednost, što znači da vrijednost ne smije biti NULL. Druga znamenka označava koliko vrijednosti mogu biti upisane u taj atribut, pošto je primarni ključ (1,1) to znači da mora imati samo jednu zapisanu vrijednost.

Strani ključ je primarni ključ jedne tablice, koja se pojavljuje u drugoj tablici. On predstavlja vezu između tablica u relacijskom modelu. Također on uvijek referencira neki primarni ključ. Tablica StudentPrimjer sadržava strani ključ *GradID* koji se može vidjeti na slici 4., dok se tablica GradPrimjer iz koje dolazi strani ključ može vidjeti na slici 5.

	ID	Ime	Prezime	DatumUpisa	GradID	Email
1	1	Ivan	Horvat	2000-01-01	1	ihorvat@fpz.hr
2	2	Jura	Juric	2000-04-04	2	jura.juric@gmail.com
3	3	Ivan	Ivic	2002-03-20	1	NULL

Slika 4. Prikaz tablice StudentPrimjer

	ID	Naziv
1	1	Zagreb
2	2	Osijek

Slika 5. Prikaz tablice GradPrimjer

Složeni primarni ključ je posebna vrsta ključa, koji u sebi spaja dva stupca. Opisan je kôdom ispod, gdje se za primarni ključ spajaju dva stupca *StudentId* i *GradId*. Stvaranjem složenih primarnih ključeva povećava se jedinstvenost.

```
CREATE TABLE GradStudent (
       StudentId int not null,
       GradId int not null,
       CONSTRAINT PK_GradStudent
              primary key (StudentID, GradId)
)
```

2.2.2 Stvaranje baze podataka

Kako bi se stvorila baza podataka potrebno je u SQL Serveru uz pomoću *New Query* napisati kôd. Stvaranje baze podataka i njena manipulacija biti će objašnjeni na primjeru. Kako bi se kreirala baza podataka s nazivom *BazaPodatakaPrimjer* koristi se naredba `CREATE DATABASE`. Kôd naredbe za kreiranje baze podatka prikazan je ispod.

```
CREATE DATABASE BazaPodatakaPrimjer
```

Kako bi se mogla koristiti kreirana baza potrebno je napisati naredbu `USE`. Nakon izrade i odabira baze podataka potrebno je stvoriti tablicu. U ovom primjeru stvara se tablica s imenom *Student* čiji su atributi *Id*, *Ime*, *Prezime*, *DatumUpisa* i *Email*. Svi atributi imaju kardinalitet (1,1) osim *Emaila* koji ima kardinalitet (0,1). To znači da je svaki atribut potrebno upisati osim *Emaila*. Kôd naredbe za stvaranje tablice prikazan je ispod.

```
CREATE TABLE Student
(
Id int PRIMARY KEY NOT NULL,
Ime nvarchar(50) NOT NULL,
Prezime nvarchar(50) NOT NULL,
DatumUpisa date NOT NULL,
Email nvarchar(50) NULL
)
```

Kako bi se u tablicu stavili podaci potrebno je koristiti naredbu `INSERT INTO`. Ta nam naredba omogućuje da u ovom primjeru stvorimo novog studenta. Kada se koristi naredba `INSERT INTO` potrebno je upisivati vrijednosti redoslijedom kako su postavljeni atributi u tablici. U ovom primjeru dodana su tri studenta. Kôd naredbe za dodavanje studenata u tablicu prikazan je ispod.

```
INSERT INTO Student
(
1,'Ivan','Juric','2014-05-02','ijuric@fpz.hr',
2,'Jura','Ivic','2015-05-02','jivic@fpz.hr',
```

```
3,'Ivan','Ivic','2015-05-02','ivic5@gmail.com'
)
```

Za prikazivanje svih podataka u tablici koristi se naredba `SELECT`. Naredba radi na način da odabere sve studente i njihove podatke u tablici student. Kôd naredbe za odabir svih studenata i njihovih podataka u tablici Student prikazan je ispod.

```
SELECT * FROM Student
```

Kako bi se podaci mogli filtrirati koristi se naredba `WHERE`. Ona omogućuje da se naredbi `SELECT` dodaju dodatni kriteriji koji utječu na odabir podataka. Kôd naredbe za filtriranje podataka u tablici prikazan je ispod.

```
SELECT * FROM Student
WHERE Ime='Ivan'
```

U ovom primjeru odabiru se svi podatci koji za *Ime* imaju vrijednost Ivan. Odgovor na ovaj upit prikazuje sve podatke studenata Ivana Jurica i Ivana Ivica.

Za odabir jedinstvenih vrijednosti koristi se naredba `DISTINCT`. Ona omogućava da se prikažu sve jedinstvene vrijednosti nekog atributa. U ovom primjeru odabiru se sve jedinstvene vrijednosti *Imena*. Kôd naredbe prikazan je ispod.

```
SELECT DISTINCT Ime FROM Student
```

Ovaj upit kao odgovor daje samo dvije jedinstvene vrijednosti *imena*: Ivan, Jura.

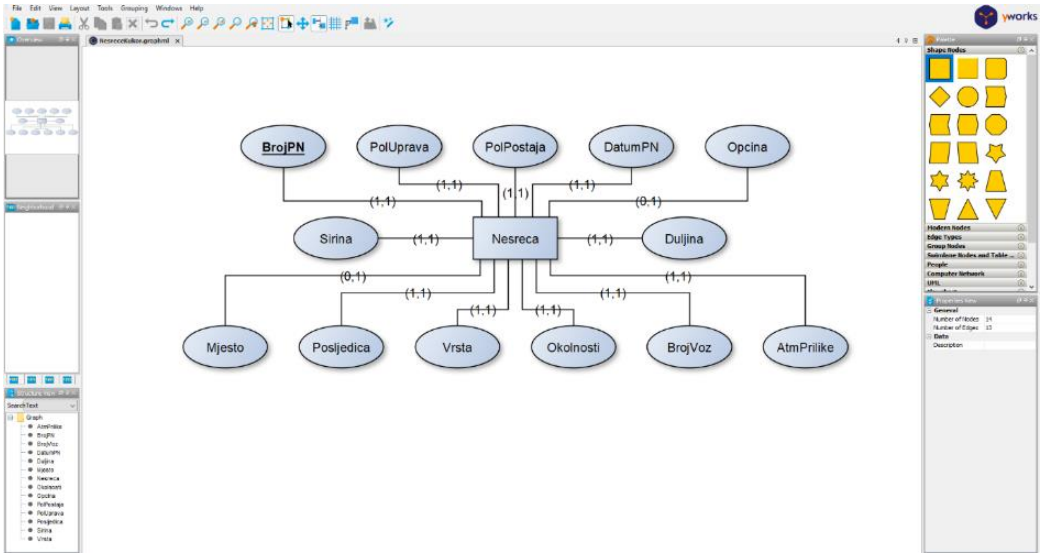
Kako bi se u bazi podataka moglo izbrojati koliko ima studenta koristi se naredba `COUNT`. Kôd ove naredbe prikazan je ispod.

```
SELECT COUNT(*) FROM Student
```

Odgovor prethodnog upita je 3.

2.2.3 Er dijagram

Baze podataka mogu se prikazati pomoću grafikonom. Za to se u ovom radu koristio yEd Graph Editor. Korištenje aplikacije je jednostavno jer radi na principu drag and drop, gdje se odaberu željeni objekti koji se onda povuku na radnu površinu. Shema baze podataka koja se koristi u ovom radu prikazana je uz pomoć dijagrama entiteta i veza koji se može vidjeti na slici 6. ispod.



Slika 6. Prikaz dijagrama entiteta i veza

Oblik pravokutnika predstavlja entitet, u slučaju ovog dijagrama je Nesreca. Nesreca je tablica u bazi podataka. Elipse prikazuju atribute, stupce u tablici. Elipsa s podcrtanim tekstom predstavlja primarni ključ. Brojevi u zagradama prikazuju kardinalitete. Postoje tri tipa kardinaliteta:

- (1,1) – atribut prima samo jednu vrijednost
- (0,1) – atribut prima samo jednu vrijednost, ta vrijednost može poprimiti vrijednost NULL
- (0,n) – atribut prima više vrijednost, ta vrijednost može poprimiti vrijednost NULL.

Za pretvaranje dijagrama entiteta i veza u relacijski model koriste se transformacijska pravila. Transformacijska pravila su skup pravila koja omogućuju razvoj relacijskog modela na temelju modela entiteta i veza. Postoje osam transformacijskih pravila.

1. transformacijsko pravilo - dijagram entiteta i veza preslikava se u relacijsku shemu tako da svaki entitet postane tablica i da svaki atribut postane stupac te tablice. Također je potrebno da se u tablici odabere primarni ključ, (2).

Članstvo pokazuje koliko redak pojedinog entiteta sudjeluje u vezi između entiteta. Članstvo može biti obavezno i neobavezno. Kod obaveznog članstva najmanje jedan redak mora sudjelovati u vezi između entiteta. Kod neobaveznog članstva niti jedan redak ne mora sudjelovati u vezi između entiteta. Pretvaranje veza ovisi o članstvu veze i broju entiteta u vezi. Postoje tri kategorije brojeva entiteta, a to su 1:1, 1:N, N:M. Entiteti se prikazuju kao E_1 i E_2, (2).

2. transformacijsko pravilo - prikazuje vezu 1:1 koja se može ostvariti na tri načina.

1. način – ako je članstvo između E_1 i E_2 obavezno. Entiteti se spajaju u jednu tablicu koja ima njihove atribute te imaju zajednički primarni ključ.

2. način – ako je članstvo obavezno samo za jedan od entiteta. Entitetu koji ima obavezno članstvo dodaje se strani ključ entiteta koji nema obavezno članstvo.

3. način – ako su oba entiteta neobavezna. Tada se radi jedna tablica koja sadržava primarne ključeve iz obje tablice, točnije koristi se složeni primarni ključ.

3. transformacijsko pravilo – prikazuje pretvaranje veze 1:N. Ako su E_1 i E_2 u vezi 1:N, E_1 ne utječe na vezu dok se E_2 proširuje sa stranim ključem koji je primarni ključ E_1. Ako su članstva neobavezna onda se strani ključ u E_2 postavi da može primati NULL vrijednosti.

4. transformacijsko pravilo – prikazuje pretvaranje veza M:N. Stvara se nova tablica koja se sastoji od primarnih ključeva E_1 i E_2 što čini složeni primarni ključ.

5. transformacijsko pravilo – ako entitet E_1 ima viševrijednosni atribut. Taj se viševrijednosni atribut prikazuje u posebnoj tablici te se gleda veza između nove i stare tablice, (1:1, 1:N, M:N).

6. transformacijsko pravilo – pretvaranje involuirane veze radi se tako da se prepozna da li je 1:1, 1:N, M:N veza.

7. transformacijsko pravilo – pretvara podskup veze tako da svi podskupovi postaju posebne tablice te sadrže strani ključ nadskup tablice.

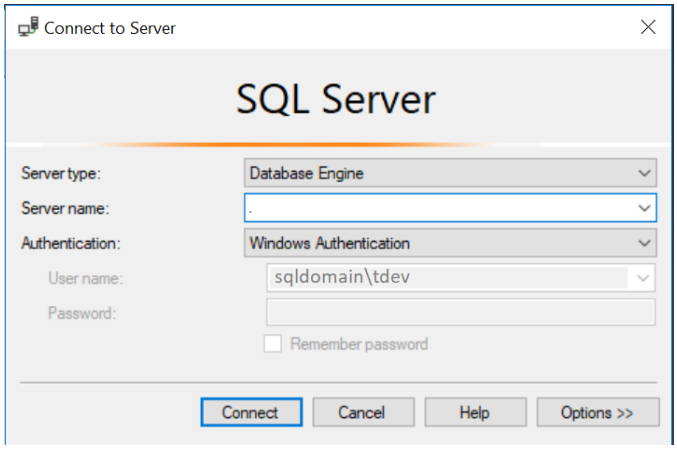
8. transformacijsko pravilo – prikazuje kako se pretvara ternarna veza. Svaki se entitet prikazuje posebnom tablicom. Kako bi se entiteti povezali uvodi se nova tablica koja u sebi sadrži primarne ključeve sva tri entiteta. U novoj tablici kao primarni ključ mogu se koristiti sva tri strana ključa ili se može stvoriti poseban primarni ključ, (2).

2.2.4 SQL Server Management Studio

U ovom radu za stvaranje relacijskog modela i upravljanje bazom podataka koristit će se SQL Server Management Studio verzije v18.12.1. SQL Server Management Studio je softverska aplikacija razvijena od strane Microsofta, koja omogućuje upravljanje, stvaranje i administraciju baza podataka. Aplikacija podržava korištenje grafičkih sučelja ili korištenje programski kôda za rad s bazama podataka. SQL Server Management Server koristi relacijski model podataka te radi sa relacijskim bazama podataka.

Autentifikacija je proces određivanja identiteta nekog subjekta, koji je najčešće fizička osoba. Neki od primjera vrsta autentifikacije su: upisivanje imena i lozinke, korištenje kartice i upisivanja PIN-a, biometrijske metode autentifikacije, (6). SQL Server Management Server koristi dvije vrste autentifikacije: Windows autentifikacija i SQL Server autentifikacija.

Korištenjem Windows autentifikacije SQL Server Management Studio provjerava valjanost imena računa i lozinke pomoću glavnog Windows tokena operativnog sustava. To znači da Windows utvrđuje identitet korisnika, tako da SQL Server Management Studio ne provodi provjeru identiteta i ne traži korisnika da upiše lozinku. Windows autentifikacija koristi Kerberos sigurnosni protokol. Dobra karakteristika ove vrste autentifikacije je mogućnost kreiranja Windows grupa koje se nalaze na razini domene. To omogućuje da SQL Server Management Studio može kreirati prijave za cijelu grupu. Samo upravljanje pristupom na razini domene omogućuje pojednostavljenje administracije računa, (7). Prikaz Windows autentifikacije može se vidjeti na slici 7, (8).

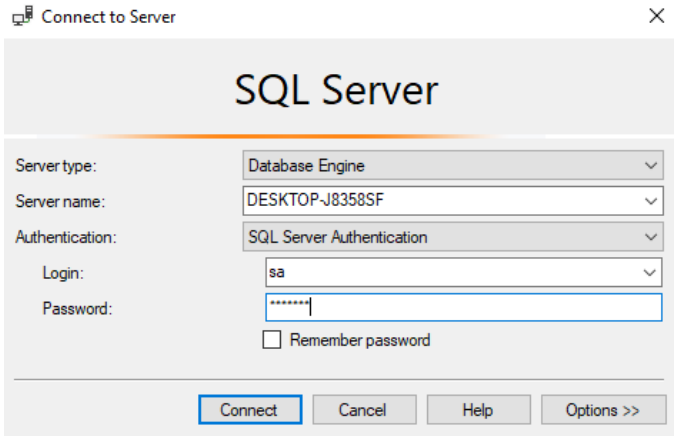


Slika 7. Prikaz Windows autentifikacije, (8)

Druga vrsta autentifikacije je SQL Server autentifikacija. Kod korištenja SQL Server autentifikacije prijava se stvara u aplikaciji SQL Server Management Studio te se ona ne temelji na korisničkom računu Windows-a. Prilikom prijave uvijek je potrebno upisati ime i lozinku koje se kreiralo i pohranilo u SQL Server Management Studiju. Neki nedostaci Server autentifikacije su: Server autentifikacije ne podržava Kerberos sigurnosni protokol, Server autentifikacija daje manje pravila za lozinke nego Windows autentifikacija, potreba za stalnim upisivanjem imena i lozinke prilikom svakog korištenja baze podataka

Neke od prednosti korištenja SQL Server autentifikacije su: SQL Server Management Studio podržava starije aplikacije, podržava okruženja s različitim operativnim sustavima gdje korisnici nisu autentificirani u Windows domeni, podržava web aplikacije gdje korisnici stvaraju svoje identitete, (7).

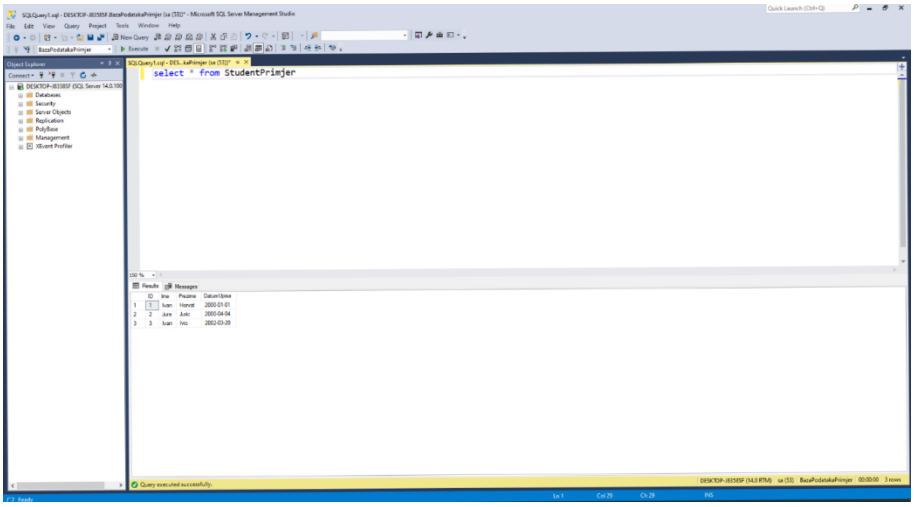
Prikaz SQL Server autentifikacije može se vidjeti na slici 8.



Slika 8. Prikaz SQL Server autentifikacije

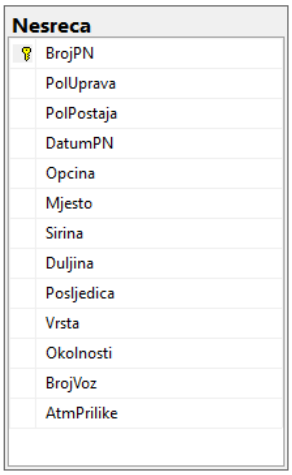
Sučelje programa SQL Studija može se podijeliti na tri dijela. Gornji dio sačinjava alatna traka, koja u sebi ima velik broj korisnih funkcija. U njoj se nalazi funkcija za odabir baze podataka na slici 9. je odabrana baza podataka pod nazivom *BazaPodatakaPrimjer*. Pored okvira za odabir baze podataka nalazi se gumb *Execute* koji omogućuje da se programskim kôdom zapisane funkcije pokrenu i izvrše. Iznad njega nalazi se gumb *New Query* koja ima funkciju stvaranja novog upita. Za stvaranje baza podataka ili za dodavanje podataka u baze podataka mogu se koristiti dva načina. Jedan je grafički, a drugi koristi programski kôd. Lijevi dio sučelja je sastavljen od *Object Explorera*. Preko njega se može

obaviti grafički način stvaranja baza podataka. Također se mogu vidjeti strukture i podaci baza podataka . Desni i veći dio sučelja koristi se za pisanje programskog kôda upita i prikaz rezultata tog upita. Sučelje se može vidjeti na slici 9.



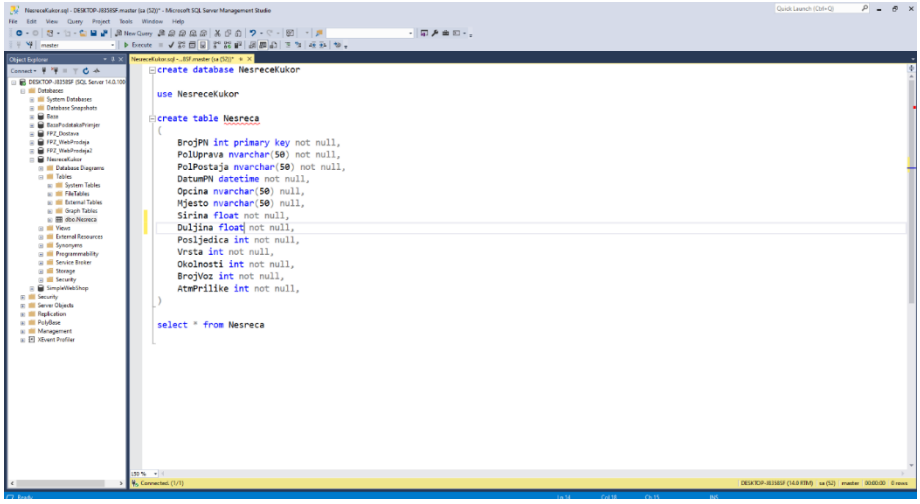
Slika 9. Prikaz sučelja SQL Servera

Na temelju već napravljenog dijagrama entiteta i veza napravljena je relacijska shema baze podataka koja se koristi u ovom radu. Relacijska shema baze podataka prikazana je na slici 10. Za pretvorbu iz dijagrama entiteta i veza u relacijsku shemu korišteno je 1. transformacijsko pravilo.



Slika 10. Prikaz relacijske sheme

Za stvaranje baze podataka u ovom radu koristi se način koji koristi programski kôd, tj. upit. Prva naredba je `CREATE TABLE` koja stvara bazu podataka, baza ima naziv NesreceKukor. Zatim se ili koristeći alatnu traku odabere baza podataka NesreceKukor ili se koristi naredba use NesreceKukor. Nakon izrade baze podataka potrebno je stvoriti tablicu gdje će se podaci spremati, za to se koristi naredba `CREATE TABLE`. Na slici 11. je prikazan kôd upita za stvaranje baze podataka NesreceKukor i tablice pod nazivom Nesreca.



Slika 11. Prikaz upita tablice NesreceKukor

Tablica *Nesreca* ima 13 atributa. Atribut *BrojPN* je broj pa se zato za njega koristi tip podataka za cijele, odnosno prirodne brojeve (int). Primarni ključ ove tablice je *BrojPN*, jer on predstavlja jedinstveni identifikator za prometne nesreće. Za stvaranje primarnog ključa koristi se naredba `PRIMARY KEY` koja još može iza sebe imati druge oznake, npr. `IDENTITY (1,1)`. Prvi broj označava s kojom se vrijednošću označava primarni ključ prvog retka, dok drugi broj označava za koliko se povećava vrijednost primarnog ključa za svaki slijedeći redak. Primjer toga bila bi oznaka (1,1) kod koje prvi redak ima vrijednost primarnog ključa 1 dok se svaki slijedeći povećava za 1. Prilikom korištenja tih oznaka SQL Server sam brine o vrijednostima primarnih ključeva. U slučaju tablice *Nesreca* nije potrebno dodavati oznake jer u podacima već postoji broj prometnih nesreća koji je jedinstven za svaku prometnu nesreću. Pošto je *BrojPN* primarni ključ on ne prima *NULL* vrijednosti pa se zato koristi *NOT NULL*.

Podaci atributa *PolUprava*, *PolPostaja*, *Opcina*, *Mjesto* koriste istu vrstu zapisa, a to je tekst (niz znakova). Za korištenje teksta u SQL Serveru koriste se četiri vrste vrijednosti char, varchar, nchar i nvarchar. Postoji razlika između char, varchar, nchar i nvarchar gdje char i varchar primaju ASCII znakove dok nchar i nvarchar primaju UNICODE znakove. Znakovni tipovi podataka char i nchar koriste se kada označavamo znakovni niz koji ima fiksnu dužinu. Dok se znakovni tipovi podataka varchar i nvarchar koriste kada označujemo niz znakova koji imaju varijabilne dužine, što znači ako se stavi nvarchar(50) tada riječ od 10 znakova iskoristi samo 10 mjesta, a ostala se ne gledaju, čime smanjujemo količinu memorije potrebne za unos tog podatka, (2).

Atribut *DatumPN* označava datum i vrijeme kada se dogodila prometna nesreća, zato ima tip vrijednosti datetime. Takav tip podatka u SQL Serveru ima već definirani način zapisa, a to je gg-mm-dd hh:mm:ss (godina-mjesec-dan sati:minute:sekunde), podaci koji se upisuju u bazu podataka imaju također takav način zapisa, (2).

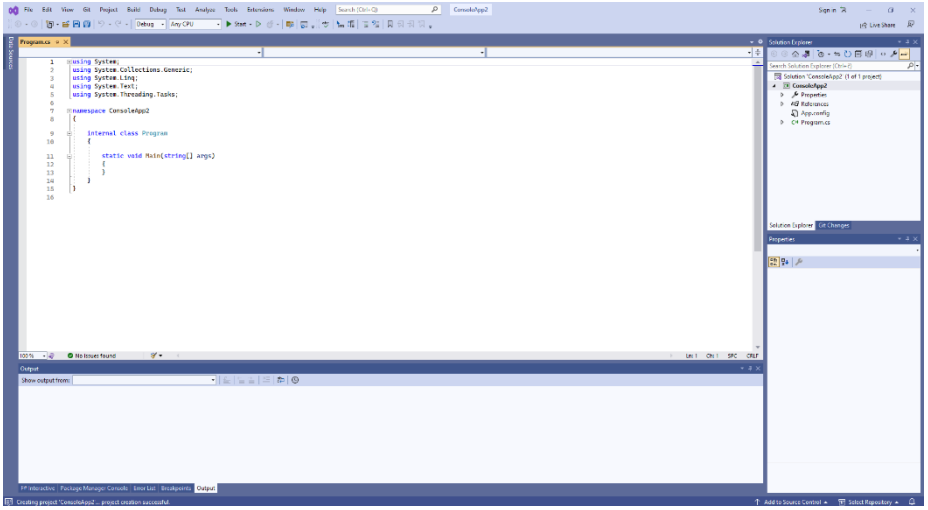
Podaci atributa *Sirina* i *Duljina* su brojevi sa decimalnim zarezom. U SQL Serveru za brojeve s decimalnim zarezom koriste se decimal i float. Decimal se koristi pomoću decimal(m,n) gdje m predstavlja ukupan broj znamenki dok n predstavlja broj znamenki koje se nalaze iza decimalnog zareza. Primjer bi bio decimal(5,2) gdje bi decimalni broj imao najviše 5 znamenki, od kojih su 2 iza decimalnog zapisa. Float isto se tako koristi za zapisivanje decimalnih brojeva. Razlika između decimala i floata je ta što decimal prikazuje egzaktne brojeve dok float prikazuje aproksimirane brojeve. Egzaktni brojevi su oni brojevi koji se zapisuju u računalo točno kakvi jesu, dok se kod aproksimiranih brojeva spremaju približne vrijednosti. U ovoj se bazi podataka koristi tip float jer podaci nemaju isti broj znamenki, (2).

Atributi *Posljedica*, *Vrsta*, *Okolnosti*, *BrojVoz*, *AtmPrilike* koriste tip podatka int jer njihovi podaci imaju vrijednosti egzaktnih prirodnih brojeva, (2).

3. Spremanje podataka u bazu podataka

U ovom radu za unos podataka iz .csv datoteke u bazu podataka koristit će se program napisan u Visual Studiju 2022 verzije 17.6.4. Visual Studio je integrirano razvojno okruženje (eng. Integrated Development Environment, IDE) razvijeno od strane Microsofta. Služi za razvoj grafičkih korisničkih sučelja (eng. Graphical User Interface, GUI), konzola, web aplikacija, mobilnih aplikacija, oblaka i web usluga. Pruža podršku za različite programske jezike, od kojih će se u ovom radu koristiti C# i JavaScript, (9).

Sučelje programa Visual Studia prikazano slikom 12. možemo podijeliti na pet dijelova. U gornjem dijelu nalazi se alatna traka koja ima velik broj korisnih funkcija, također se tamo nalazi gumb za pokretanje programa. Lijevo ispod njega nalazi se dio gdje se piše programski kôd programa. Ispod njega se nalazi prozorčić u kojem se mogu vidjeti outputi tog programa. Desno se nalaze dva dijela *Solution explorer* i prozorčić koji prikazuje svojstva. *Solution explorer* pokazuje sve datoteke koje korisnik koristi za taj program.



Slika 12. Prikaz sučelja Visual Studia

Za program koji upisuje podatke u bazu podataka koristi se C# jezik. C# je programski jezik koji ima opću namjenu visoke razine koji podržava višestruke paradigme, (10).

Prvo je potrebno dodati direktive: `using System.Data.SqlClient` i `using System.IO`. Naredba `using System.Data.SqlClient` omogućuje povezivanje programa s bazom podataka, dok je naredba `using System.IO` potrebna za korištenje klase za čitanje podataka.

Povezivanje s bazom podatka ovisi o vrsti autentifikacije koju koristi baza podataka, razlika je samo u konekcijskom stringu. Ako se koristi Windows autentifikacija konekcijski string glasi:

```
SqlConnection sql = new SqlConnection("Server = DESKTOP-J8358SF; Database=NesreceKukor; Integrated Security=True")
```

Ali ako baza koristi Server autentifikaciju konekcijski string glasi:

```
SqlConnection sql = new SqlConnection("Server= DESKTOP-J8358SF; Database=NesreceKukor1; User Id=sa;Password=Baze123")
```

Kako bi se uspostavila veza s bazom potrebno je koristiti naredbu: `sql.Open()`. Nakon uspostave veze s bazom podataka slijedi dio kôda koji čita tekstualnu datoteku redak po redak te stavlja podatke u listu. Kôd tog čitača prikazan je slikom 13.

```
string path = "Podaci_Nesrece.csv";
StreamReader citanje = new StreamReader(path);

int i = 0;

citanje.ReadLine();
while (!citanje.EndOfStream)
{
    string linija = citanje.ReadLine();
    if (String.IsNullOrEmpty(linija.Trim()))
    {
        break;
    }
    string[] stupac = linija.Split(',');

    int BrojPN;
    bool isNumber = int.TryParse(stupac[0], out BrojPN);
    string PolUprava = stupac[1];
    string PolPostaja = stupac[2];
    string DatumPN = stupac[3];
    string Opcina = stupac[4];
    string Mjesto = stupac[5];
    int Posljedica = Convert.ToInt32(stupac[8]);
    int Vrsta = Convert.ToInt32(stupac[9]);
    int Okolnosti = Convert.ToInt32(stupac[10]);
    int BrojVoz = Convert.ToInt32(stupac[11]);
    int AtmPrilike = Convert.ToInt32(stupac[12]);

    SqlCommand cmd = new SqlCommand("insert into Nesreca123(BrojPN, PolUprava, PolPostaja, DatumPN, Opcina, Mjesto, Sirina, Duljina, Posljedica, Vrsta, Okolnosti, BrojVoz, AtmPrilike) values(" +
        checkNullInt(stupac[0]) + "," +
        checkNullString(stupac[1]) + "," +
        checkNullString(stupac[2]) + "," +
        checkNullString(stupac[3]) + "," +
        checkNullString(stupac[4]) + "," +
        checkNullString(stupac[5]) + "," +
        checkNullString(stupac[6]) + "," +
        checkNullString(stupac[7]) + "," +
        checkNullInt(stupac[8]) + "," +
        checkNullInt(stupac[9]) + "," +
        checkNullInt(stupac[10]) + "," +
        checkNullInt(stupac[11]) + "," +
        checkNullInt(stupac[12]) + ")", sql);

    i++;

    cmd.ExecuteNonQuery();
}
citanje.Close();
```

Slika 13. Kôd čitača podataka

Nakon što se jedan redak teksta stavi u matricu on se iz matrice stavlja u bazu podataka pomoću naredbe `SqlCommand`. Ona u sebi sadržava naredbu koja u SQL jeziku upisuje podatke u bazu podataka, ta naredba je `INSERT INTO`. Naredba radi tako da vrijednosti u svakom stupcu stavi u određene stupce u tablici Nesreca koja se nalazi u bazi podataka NesreceKukor. Kako bi se izvršila `SqlCommand cmd` potrebno je koristiti naredbu `cmd.ExecuteNonQuery()`. Kada se cijela datoteka učita u bazu podataka potrebno je čitač zatvoriti naredbom `citanje.Close()`. Cijeli kôd čitača stavljen je u try-catch blok. On omogućuje da ako se kôd izvrši bez pogreške u konzoli se ispiše '' Uspjesno dodano u bazu podataka. '', a ako dođe do pogreške u konzoli se ispiše '' Nije uspjelo dodavanje u bazu podataka. '' i koja je to pogreška. Neke od pogreška su: nepovezivanje sa bazom podataka i neuspjeh unosa podataka u bazu.

Također se u kôdu koriste dvije metode. Prva je metoda `checkNullString` koja provjerava da li je vrijednost u podacima praznina, ako je onda se u bazu podataka upisuje NULL, a ako ima neku vrijednost onda se upisuje ta vrijednost u obliku stringa. Kôd metode `checkNullString` prikazan je slikom 14.

```
7 references
private static object checkNullString(string value)
{
    if (string.IsNullOrEmpty(value.Trim()))
    {
        return "NULL";
    }
    else
    {
        return "'" + value + "'";
    }
}
```

Slika 14. Kôd metode `checkNullString`

Druga je metoda `checkNullInt` koja provjerava da li je vrijednost u podacima prazna, ako je onda se u bazu podataka upisuje NULL, a ako ima neku vrijednost onda se upisuje ta vrijednost u obliku int. Kôd metode `checkNullInt` prikazan je slikom 15.

```
6 references
private static object checkNullInt(string value)
{
    if (string.IsNullOrEmpty(value))
    {
        return "NULL";
    }
    else
    {
        return Convert.ToInt32(value);
    }
}
```

Slika 15. Kôd metode `checkNullInt`

Za izvršavanje programa koji upisuje podatke u bazu podataka potrebno je 23 sekunde. Nakon što se svi podaci unesu, oni se mogu vidjeti u SQL Serveru pomoću upita:

```
SELECT * FROM Nesreca
```

Prethodni upit daje odgovor da ima 31432 prometnih nesreća, a vrijeme koje je bilo potrebno da se upit izvrši bilo je 1 sekunda. Prikaz svih prometnih nesreća u bazi podataka može se vidjeti na slici 16.

Results | Messages

	BrojPN	PolUprava	PolPostaja	DatumPN	Opcina	Mjesto	Sirina	Duljina	Posljedica	Vrsta	OkolnostiBrojVoz	AtmPrilike
1	4	PU DUBROVACKO-NERET.	PPRP DUBROVNIK	2014-01-01 21:05:00.000	DUBROVNIK	NULL	42,734	17,9378666666667	3	8	2	1	2
2	10	PU VUKOVARSKO-SRIJ.	PPRP VINKOVCI	2014-01-02 00:50:00.000	VINKOVCI	VINKOVCI	45,2830166666667	18,8128166666667	3	8	19	2	1
3	12	PU POŽEŠKO-SLAVONSKA	PP POŽEGA	2014-01-01 06:30:00.000	POŽEGA	POŽEGA	45,3317666666667	17,6722666666667	3	8	2	1	2
4	13	PU VIROVITICKO-PODR.	PPRP VIROVITICA	2014-01-01 00:55:00.000	PITOMACA	PITOMACA	45,95355	17,2327333333333	2	4	3	2	2
5	15	PU ISTARSKA	PP ROVINJ	2014-01-01 00:50:00.000	ROVINJ	ROVINJ	45,0807333333333	13,6416	3	16	2	1	1
6	16	PU VIROVITICKO-PODR.	PPRP VIROVITICA	2014-01-01 05:00:00.000	ŠPIŠ.BUKOVICA	LOZAN	45,8773333333333	17,2888333333333	2	8	20	1	2
7	17	PU MEDIMURSKA	PPRP CAKOVEC	2014-01-01 03:30:00.000	CAKOVEC	CAKOVEC	46,3931	16,4430166666667	2	16	2	1	1
8	18	PU VUKOVARSKO-SRIJ.	PPRP VINKOVCI	2014-01-01 01:15:00.000	BOŠNJACI	NULL	46,2075	18,2038333333333	3	16	2	1	1
9	19	PU OSJECKO-BARANJSKA	PPRP OSIJEK	2014-01-01 17:20:00.000	OSIJEK	OSIJEK	45,5401333333333	18,7236166666667	3	6	11	2	2
10	20	PU OSJECKO-BARANJSKA	PP DONJI MIHOLJAC	2014-01-01 06:40:00.000	POD. MOSLAVINA	NULL	45,75715	18,0377666666667	2	8	2	1	2
11	23	PU ZAGREBACKA	I PPRP ZAGREB	2014-01-01 13:17:00.000	ZAGREB-CENTAR	ZAGREB	45,81045	15,9660166666667	3	6	19	2	1
12	24	PU OSJECKO-BARANJSKA	PP BELI MANASTIR	2014-01-01 14:20:00.000	BILJE	NULL	45,6886333333333	18,8145333333333	3	1	7	2	2
13	25	PU VIROVITICKO-PODR.	PPRP VIROVITICA	2014-01-02 13:10:00.000	VIROVITICA	VIROVITICA	45,8348	17,3844	3	6	11	2	1
14	26	PU ZAGREBACKA	PP VELIKA GORICA	2014-01-02 15:00:00.000	V. GORICA	V. GORICA	45,6953166666667	16,0602333333333	2	16	19	1	2
15	27	PU ZAGREBACKA	PP VELIKA GORICA	2014-01-01 03:00:00.000	V. GORICA	V. GORICA	45,7268666666667	16,0922666666667	3	16	9	1	1
16	28	PU ZAGREBACKA	PP SAMOBOR	2014-01-02 20:30:00.000	SV.NEDJELJA	SV.NEDELJA	45,8034	15,7941166666667	2	8	19	1	1
17	29	PU SPLITSKO-DALMAT.	PP IMOTSKI	2014-01-02 08:00:00.000	PODBABLJE	IVANBEGOVINA	43,42215	17,1492666666667	3	8	31	1	4
18	30	PU KRAPINSKO-ZAGOR.	PP DONJA STUBICA	2014-01-02 12:00:00.000	OROSLAVJE	OROSLAVJE	45,9631833333333	15,9233166666667	3	2	13	2	1
19	31	PU MEDIMURSKA	PPRP CAKOVEC	2014-01-01 07:09:00.000	ŠENKOVEC	NULL	46,4166333333333	16,4033166666667	3	8	20	1	2
20	32	PU KOPRIVNICKO-KRIŽ.	PPRP KOPRIVNICA	2014-01-02 19:40:00.000	SOKOLOVAC	VELIKA MUCNA	46,10755	16,7242833333333	3	8	2	1	2
21	36	PU BRODSKO-POSAVSKA	PPRP SLAVONSKI ...	2014-01-01 00:35:00.000	SL. BROD	SL. BROD	45,15305	18,0213833333333	3	8	2	1	2
22	37	PU BRODSKO-POSAVSKA	PPRP SLAVONSKI ...	2014-01-01 04:55:00.000	SL. BROD	SL. BROD	45,1532	18,0247333333333	3	8	2	1	3
23	38	PU ISTARSKA	PPRP PULA	2014-01-02 11:40:00.000	MEDULIN	NULL	44,82625	13,9210166666667	3	14	19	2	1
24	39	PU ISTARSKA	PPRP PULA	2014-01-03 05:25:00.000	PULA	PULA	44,8726666666667	13,8532833333333	3	6	10	2	3
25	41	PU SPLITSKO-DALMAT.	PP IMOTSKI	2014-01-02 11:35:00.000	IMOTSKI	IMOTSKI	43,4473666666667	17,2241166666667	3	8	2	1	3
26	42	PU DUBROVACKO-NERET.	PP PLOCE	2014-01-02 10:00:00.000	PLOCE	NULL	43,0345166666667	17,5489333333333	2	8	2	1	3
27	43	PU SISACKO-MOSLAV.	PPRP KUTINA	2014-01-02 05:15:00.000	KUTINA	NULL	45,4495666666667	16,7866	2	8	19	1	1

Slika 16. Prikaz svih prometnih nesreća u bazi podataka

4. Web aplikacija

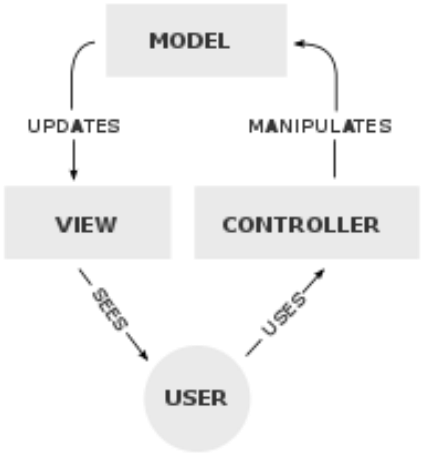
4.1 MVC

Za izradu web aplikacije u ovom radu koristit će se MVC okruženje. MVC je obrazac softverskog dizajna koji se koristi za razvoj korisničkih sučelja. Programsku logiku sačinjavaju tri međusobno povezana elementa: model, pogled i kontroler. Takva je podjela napravljena kako bi se odvojili prikazi informacija i načina na koji se te informacije prezentiraju kod korisnika, (11).

Neke od značajki MVC-a su:

- mogućnost odvajanja poslovne logike, UI logike i ulazne logike,
- kontrola nad HTML-om i URL-ovima što olakšava stvaranje web aplikacije,
- mapiranje URL-a koje omogućuje stvaranje aplikacija koje imaju URL-ove koje je moguće pretraživati,
- podržava Test Driven Development, (12).

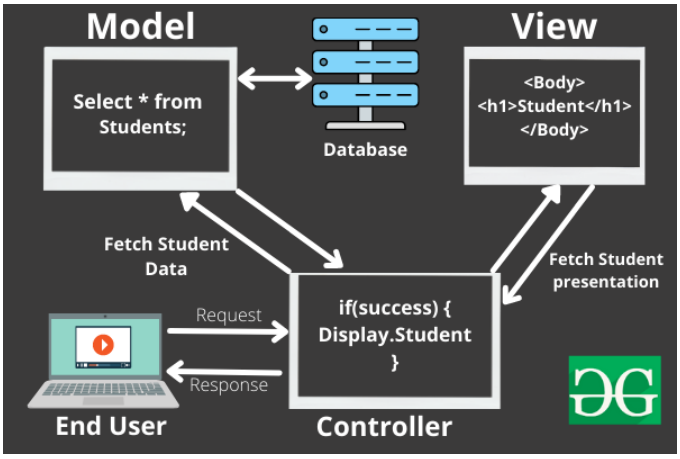
Dijagram dijelova MVC-a prikazani su slikom 17, (11).



Slika 17. Shema MVC, (11)

Arhitektura je temeljna organizacija sustava koja sadržava ključne komponente, njihove odnose i veze prema okolini, te načela njihova dizajniranja i razvoja, (13).

Primjer bi rada MVC bio da krajnji korisnik želi dobiti popis studenata. On šalje zahtjev poslužitelju koji taj zahtjev prosljeđuje kontroleru. Kontroler traži da model koji rukuje studentima vrati popis svih studenata. Model u bazi podataka traži popis svih studenata te vraća popis kontroleru. Nakon što je kontroler primio popis svih studenata, on ga šalje pogledu koji taj popis prezentira krajnjem korisniku. Shema rada ovog primjera prikazana je na slici 18, (12).



Slika 18. Prikaz arhitekture MVC-a na temelju primjera Student, (12)

Prednosti MVC-a su: kôd je jednostavan za održavanje i s lakoćom se može proširiti, svaka se komponenta može testirati zasebno, komponente MVC-a mogu se istovremeno razvijati, smanjuje se složenost aplikacije zbog podijele na tri dijela, sve su klase i objekti neovisni jedni o drugima te to pomaže kod testiranja komponenata. Nedostaci MVC-a su: neprikladnost izrade malih aplikacija, neučinkovitost pristupa podataka u pogledu, povećana složenost podatka, (12).

Komponentu model čini cijela logika podataka koju korisnik koristi. Glavna je zadaća modela prenositi podatke između pogleda i kontrolera. Glavna funkcija koja se koristi u ovom radu je dohvaćanje podataka iz baze podataka. Model radi tako da vrši interakciju s bazom podataka te vraća kontroleru potrebne podatke.

Kotroler je komponenta koja omogućuje povezivanje pogleda i modela. On se ne brine o rukovanju logikom podataka, on samo šalje naredbe modelu što on mora raditi. On određuje poslovnu logiku te dolazne zahtjeve. Njegova je zadaća manipulacija podataka pomoću modela i komunikacija s pogledom kako bi se prikazao završni izlaz.

Pogled je komponenta koja se koristi za cijelu logiku korisničkog sučelja nekakve aplikacije. On stvara korisničko sučelje koje se prikazuje korisniku, (12).

4.2 Web aplikacija

Frontend je dio web aplikacije koji krajnji korisnik direktno koristi. U sebi uključuje izgled web aplikacije kao što su boje i stilovi tekstova, slike, grafikone i tablice, gumbe i navigacijske izbornike. Backend je serverska strana web aplikacije, nju krajnji korisnik ne vidi niti koristi. Zadaća backend-a je pohrana i raspoređivanje podataka kao i osiguravanje dobrog rada aplikacije, (14).

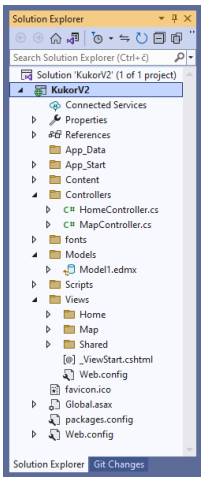
NuGet je službeni upravitelj paketa za .NET. Paketi su u osnovi biblioteke s nekim opisnim metapodacima. NuGet-om je moguće dijeliti kôd, gdje je često takav kôd povezan u "pakete" koji sadrže kompilirani kôd zajedno s ostalim sadržajem potrebnim u projektima koji koriste te pakete, (15).

U ovoj web aplikaciji koriste se slijedeći NuGet paketi:

- bootstrap
- EntityFramework
- jQuery
- Microsoft.AspNet.MVC
- Microsoft.AspNet.Razor
- Newtonsoft.Json

Bootstrap se koristi za razvoj frontend dijela web aplikacije. Entity Framework služi kao objektno-relacijski maper koji povezuje web aplikaciju s bazom podataka. Paket jQuery je JavaScript knjižnica. Microsoft.AspNet.MVC služi za izradu dinamičke web aplikacije. Microsoft.AspNet.Razor služi za brzu i sažetu kombinaciju poslužiteljskih kodava koji sadrži HTML. JavaScript Object Notation (JSON) je standardni format koji se temelji na tekstu za predstavljanje strukturiranih podataka. Temelji se na sintaksi JavaScript objekta te se koristi za prijenos podataka u web aplikaciji, (16). Newtonsoft.Json omogućuje korištenje Json objekata.

Za kreiranje web aplikacije korišten je program Visual Studio 2022 verzije 17.6.4. Kod odabira predloška koristio se ASP.NET Web Application (.NET Framework) koji omogućuje rad u MVC. Sučelje za izradu MVC-a i programa za unos podataka u bazu je identično osim kod *Solution explorera*. Pošto MVC ima tri razdvojene komponente tako ima više mapa u *Solution exploreru*. *Solution explor*er za MVC aplikaciju prikazan je slikom 19.



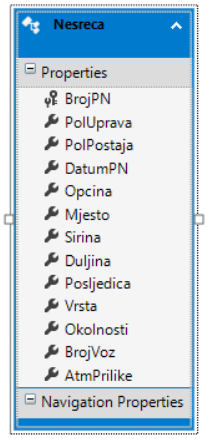
Slika 19. Prikaz
Solution explorera

Bitne mape su Content gdje se nalaze datoteke koje se koriste u pogledu. Mapa Controllers u sebi sadržava datoteke kontrolere. Mapa Model u sebi ima model, te mapa Views u sebi sadrži datoteke svih pogleda. Web aplikacija povezuje se s bazom podataka pomoću Entity Frameworka. Entity Framework omogućava vraćanje podataka u bazu podataka kao objekt. Preko tog modela šalju se upiti u bazu. Kako bi se model mogao povezati potrebno je dodati connection string. Taj connection string ovisi o vrsti autentifikacije koju koristi baza podataka. Connestion string nalazi se u datoteci Web.config. Za dodavanje tog modela i connection stringa koristi se naredba:

```
<add name="NesreceKukorEntities3"
connectionString="metadata=res://*/Models.Model1.csdl|res://*/Models.Model1.ssdl|
res://*/Models.Model1.msl;provider=System.Data.SqlClient;provider connection
string=&quot;data source=DESKTOP-J8358SF;initial catalog=NesreceKukor;user
```

```
id=sa;password=Baze123;MultipleActiveResultSets=True;App=EntityFramework&quot;"
providerName="System.Data.EntityClient" />
```
.

Nakon što se uspješno web aplikacija poveže s bazom podataka u mapi Models stvori se datoteka Model1.edmx čiji je izgled prikazan na slici 20.



Slika 20. Prikaz modela koji je spojen s bazom podataka

Kako bi kontroler mogao raditi sa modelom povezanim sa bazom podataka potrebno je na početku kôda dodati direktive `using`:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;
using Newtonsoft.Json;
using KukorV2.Models;
```

Using System naredba koristi System knjižnicu u kojoj se nalaze klase. Using System.Collections.Generic sadržava sučelja i klase koje definiraju generičke zbirke. Using System.Linq pruža klase i sučelja za rad s Language-Intefrated Query (Linq) koji se koristi za rad s bazom podataka. Using System.Web sadržava klase koje omogućuju stvaranje korisničkih sučelja i web servisa. Using System.Web.MVC u sebi sadrži klase koje podržavaju ASP.NET MVC za stvaranje web aplikacija, (16). Using Newtonsoft.Json omogućuje rad s Json objektima. Using KukorV2.Models omogućuje rad s modelom koji je spojen na bazu podataka.

Da bi se omogućilo da kontroler vraća podatke u web aplikaciju, kako bi ih krajnji korisnik mogao vidjeti, koristi se klasa `WebResponse` koja u sebi sadržava listu *accidents* u koju će se spremati prometne nesreće, varijablu *succ* koja označava da li je nešto uspjelo te ima samo dvije vrijednosti: istina ili laž. Također klasa `WebResponse` ima varijablu *errorDesc* u kojoj se sprema vrsta pogreške ako do iste dođe.

```
public class WebResponse
    {
        public List<Nesreca> accidents;
        public bool succ;
        public string errorDesc;
    }
```

Kontroler koji upravlja web aplikacijom naziva se MapController. U njemu se nalaze metode koje upravljaju svim gumbima koje krajnji korisnik može koristiti. Kako bi se mogli prikazati padajući izbornici u kojima se nalaze varijable kojima filtriramo prometne nesreće koriste se Viewbag-ovi.

ViewBag služi za dinamičko dijeljenje vrijednosti iz kontrolera u pogled. Zbog toga smatra se dinamičkim objektom koji nema unaprijed postavljena svojstva, (17).

Na slici 21. su prikazani svi ViewBag-ovi koji se koriste u ovoj web aplikaciji.

```
0 references
25  public ActionResult Index()
26  {
27      ViewBag.listaPolUprava = dbNesrece.Nesrecas.Select(x=>x.PolUprava).Distinct().ToList();
28
29      ViewBag.listaOpcina = dbNesrece.Nesrecas.Select(x => x.Opcina).Distinct().ToList();
30
31      ViewBag.listaMjesto = dbNesrece.Nesrecas.Select(x => x.Mjesto).Distinct().ToList();
32
33      ViewBag.listaVrsta = dbNesrece.Nesrecas.Select(x => x.Vrsta).Distinct().ToList();
34
35      ViewBag.listaPosljedica = dbNesrece.Nesrecas.Select(x => x.Posljedica).Distinct().ToList();
36
37      ViewBag.listaOkolnosti = dbNesrece.Nesrecas.Select(x => x.Okolnosti).Distinct().ToList();
38
39      ViewBag.listaBrojVoz = dbNesrece.Nesrecas.Select(x => x.BrojVoz).Distinct().ToList();
40
41      ViewBag.listaAtmPrilike = dbNesrece.Nesrecas.Select(x => x.AtmPrilike).Distinct().ToList();
42
43      return View();
44
45  }
```

Slika 21. Prikaz ViewBag-ova

Kôd svih ViewBag-ova kontroler traži od modela koji je povezan s bazom podataka da mu preda sve jedinstvene vrijednosti od atributa te baze podataka u listama.

Pogled web aplikacije nalazi se u mapi Views u datoteci s nazivom Indeks.cshtml. Kôd u Indeks.cshtml sačinjen je od dva dijela. Prvi dio je tijelo koje određuje izgled same web stranice. Drugi dio čini glavni dio kôda koji koristi JavaScript programski jezik. U prvom dijelu nalazi se kôd koji određuje mapu i bočnu traku. Izgled mape i skočnih prozora nalaze se u datoteci siteMap.css koja se poziva u Indeks.cshtml. U Indeksu se nalaze tri div elementa: karta, bočna traka i skočni prozor. Za određivanje stila div elemenata koristi se Cascading Style Sheets (CSS). CSS je stilski jezik koji omogućuje dizajn elemenata web aplikacije, (18). Div element za kartu uređen je tako da je postavljen od desnog ruba 0%, od lijevog ruba 20%, od gornjeg ruba je odmaknut 55px, a do donjeg ruba ide do kraja ekrana. Dok je div element za bočnu traku postavljen od desnog ruba 80%, od lijevog ruba 0%, gornjeg ruba 50px, a do donjeg ruba ide do kraja. Razina preklapanja je 5, dok je ukupna pozicija apsolutna. Na slici 22. prikazani su kôdovi koji određuju izgled mape i bočne trake, dok je na slici 23. prikazan kôd za izgled skočnog prozora.

```
/*Div container za kartu*/
#divMap {
    border: 2px solid #cccccc;
    position: absolute;
    left: 20%;
    right: 0%;
    top: 55px;
    bottom: 0%;
    height: 100%;
    background: #cccccc;
    z-index: 5;
}
/*Div container za opcije u sidebaru*/
#divOptions {
    border: 2px solid #cccccc;
    position: absolute;
    left: 0%;
    right: 80%;
    top: 50px;
    bottom: 0%;
    background: white;
    z-index: 6;
}
```

Slika 22. Kôd izgleda mape i bočne trake

```
/*OSM popup stylings*/
.ol-popup {
    position: absolute;
    background-color: white;
    -webkit-filter: drop-shadow(0 1px 4px rgba(0,0,0,0.2));
    filter: drop-shadow(0 1px 4px rgba(0,0,0,0.2));
    padding: 15px;
    border-radius: 10px;
    border: 1px solid #cccccc;
    bottom: 12px;
    left: -100px;
    min-width: 200px;
}

    .ol-popup:after, .ol-popup:before {
        top: 100%;
        border: solid transparent;
        content: " ";
        height: 0;
        width: 0;
        position: absolute;
        pointer-events: none;
    }

    .ol-popup:after {
        border-top-color: white;
        border-width: 10px;
        left: 100px;
        margin-left: -10px;
    }

    .ol-popup:before {
        border-top-color: #cccccc;
        border-width: 11px;
        left: 100px;
        margin-left: -11px;
    }

.ol-popup-closer {
    text-decoration: none;
    position: absolute;
    top: 2px;
    right: 8px;
}

    .ol-popup-closer:after {
        content: "✖";
    }
```

Slika 23. Kôd izgleda skočnog prozora

Za rad s OpenStreetMapom (OSM) korišteni su Open Layers-i, koji su JavaScript biblioteke za postavljanje karta u web aplikaciju, (19). Open Layer karta smještena je u div element divMap. Početna lokacija postavljena je na koordinate centra grada Zagreba, koje su reprojicirane na koordinatni sustav karte. Razina povećanja prikaza stavljena je na 12. Kôd za kreiranje OSM karte prikazan je ispod.

```
var map = new ol.Map({
        interactions: ol.interaction.defaults({ doubleClickZoom: false }),
        target: 'divMap',
        layers: [
            new ol.layer.Tile({
                source: new ol.source.OSM()
            })
        ],
        view: new ol.View({
            center: ol.proj.fromLonLat([15.981179, 45.805281]),
            zoom: 12
        })
    });
```

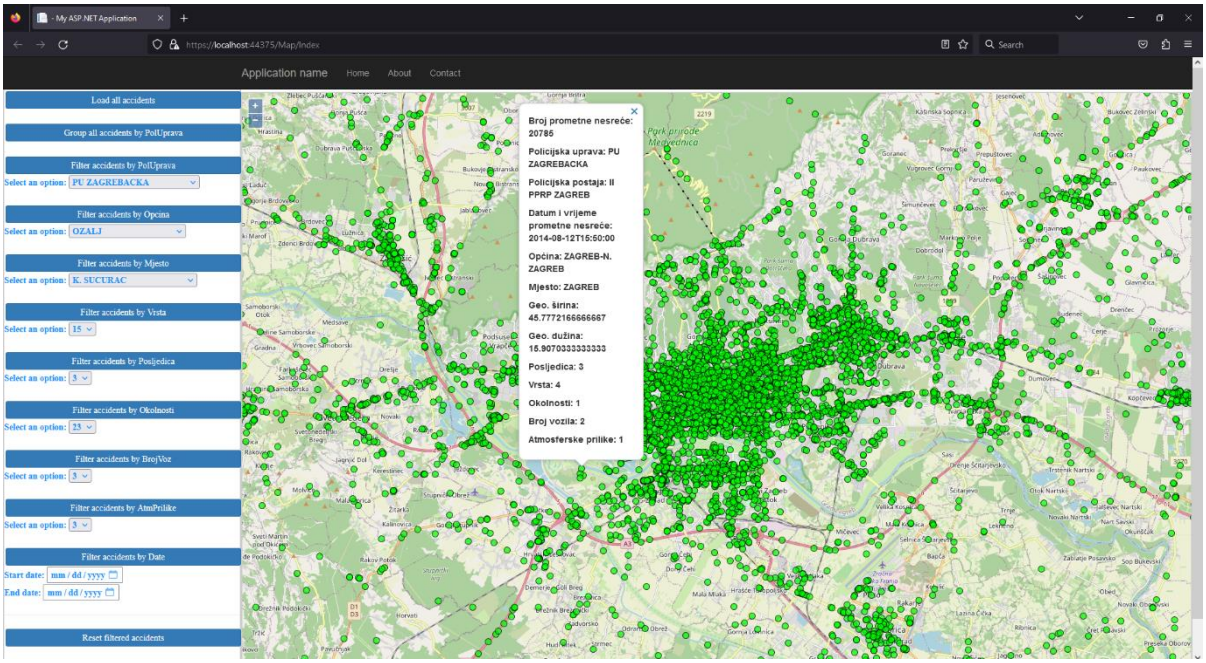
Kako bi se mogao napraviti skočni prozor potrebno je napraviti posebnu varijablu za njega, također je potrebno napraviti dvije funkcije. Prva funkcija otvara skočni prozor kada se klikne na lokaciju prometne nesreće. Dok je druga funkcija zatvaranje skočnog prozora uz pomoć X gumba.

Kako bi se na skočni prozor dodale informacije o prometnim nesrećama koristi se kôd na slici 24.

```
773
774     function processClickOnFeature(event, feature, layer) {
775         var coordinate = event.coordinate;
776         console.log("ispis nesrece");
777         var value = feature.getProperties().value;
778         if (value >= 0) {
779             var plotedObject = plotedObjects[value];
780             var acc = plotedObject.acc;
781             console.log(acc);
782                 content.innerHTML = "";
783             content.innerHTML +=
784                 '<div class="form-inline"><label for="BrojPN" class="labelLeft">' + acc.BrojPN + '</label></div>';
785             content.innerHTML +=
786                 '<div class="form-inline"><label for="PolUporava" class="labelLeft">' + acc.PolUprava + '</label></div>';
787             content.innerHTML +=
788                 '<div class="form-inline"><label for="PolPostaja" class="labelLeft">' + acc.PolPostaja + '</label></div>';
789             content.innerHTML +=
790                 '<div class="form-inline"><label for="DatumPN" class="labelLeft">' + acc.DatumPN + '</label></div>';
791             content.innerHTML +=
792                 '<div class="form-inline"><label for="Opcina" class="labelLeft">' + acc.Opcina + '</label></div>';
793             content.innerHTML +=
794                 '<div class="form-inline"><label for="Mjesto" class="labelLeft">' + acc.Mjesto + '</label></div>';
795             content.innerHTML +=
796                 '<div class="form-inline"><label for="Sirina" class="labelLeft">' + acc.Sirina + '</label></div>';
797             content.innerHTML +=
798                 '<div class="form-inline"><label for="Duljina" class="labelLeft">' + acc.Duljina + '</label></div>';
799             content.innerHTML +=
800                 '<div class="form-inline"><label for="Posljedica" class="labelLeft">' + acc.Posljedica + '</label></div>';
801             content.innerHTML +=
802                 '<div class="form-inline"><label for="Vrsta" class="labelLeft">' + acc.Vrsta + '</label></div>';
803             content.innerHTML +=
804                 '<div class="form-inline"><label for="Okolnosti" class="labelLeft">' + acc.Okolnosti + '</label></div>';
805             content.innerHTML +=
806                 '<div class="form-inline"><label for="BrojVoz" class="labelLeft">' + acc.BrojVoz + '</label></div>';
807             content.innerHTML +=
808                 '<div class="form-inline"><label for="AtmPrilike" class="labelLeft">' + acc.AtmPrilike + '</label></div>';
809
810                     overlay.setPosition(coordinate);
811             }
812             else {
813                 content.innerHTML = "";
814                 contentRoute.innerHTML = "";
815                 overlay.setPosition(undefined);
816                 closer.blur();
817             }
818     }
```

Slika 24. Kôd za prikaz informacija na skočnom prozoru

Izgled skočnog prozora na web aplikaciji kada se klikne prometna nesreća prikazan je slikom 25. Kao primjer uzeta je prometna nesreća čiji je broj 1850. Skočni prozor u sebi sadrži sve informacije o prometnoj nesreći.



Slika 25. Prikaz skočnog prozora

S lijeve strane korisničkog sučelja nalazi se dvanaest funkcija. Te funkcije su: učitaj sve prometne nesreće, vizualno grupiraj sve prometne nesreće, filtriraj prometne nesreće po policijskoj upravi, filtriraj prometne nesreće po općini, filtriraj prometne nesreće po mjestu, filtriraj prometne nesreće po vrsti, filtriraj prometne nesreće po posljedicama, filtriraj prometne nesreće po okolnostima, filtriraj prometne nesreće po atmosferskim prilikama, filtriraj prometne nesreće po datumu i funkcija za brisanje filtera. Kako svi filteri osim filtera prometne nesreće po datumu rade na isti način samo im se mijenja potreban atribut u bazi podataka u završnom radu biti će objašnjen samo filter po policijskim upravama

Gumb *Load all accidents* prikazuje sve prometne nesreće. Njegov izgled napisan je kôdom:

```
<button class="btn btn-primary" type="button" style="width: 100%; margin: 2px;
padding: 3px" + onclick="loadTrafficAccidents()">
        Load all accidents
</button>
```

Funkcija loadTrafficAccidents u sebi ima *ajax* poziv koji zove odgovarajuću metodu kontroleru. Kontroler naređuje modelu koji šalje upit u bazu podataka. Sve prometne nesreće koje se nalaze u bazi podataka stavljaju se u listu te se natrag šalju u pogled preko instance klase *WebResponse*. Pritom se cijela lista serijalizira kao JSON objekt. Kako radi kontroler prikazano je kôdom ispod.

```
[HttpGet]
public ActionResult LoadTrafficData()
{
    WebResponse wres = new WebResponse();
    try
    {

        wres.accidents = dbNesrece.Nesrecas.ToList();
        wres.succ = true;
    }
    catch (Exception ex)
    {
        wres.errorDesc = ex.Message;
        wres.succ = false;
```

```
            }

            return Content(JsonConvert.SerializeObject(wres),
"application/json");
        }
```

Funkcija `loadTrafficAccidents()` poziva se kada korisnik pritisne gumb *Load all accidents*. Ako je funkcija neuspješna otvara se prozor koji pokazuje koja je pogreška, ali ako se funkcija uspješno izvršila poziva se funkcija `successfullyLoadedFunction` koja za svaku nesreću u listi nesreća stvara ikonu te tu ikonu stavlja u sloj (eng. layer). Na kraju taj sloj učitava u kartu. Boja ikone definirana je stilom *ol.styte.Style*, a ovdje je odabran krug popunjen crvenom bojom i crnim rubom. U kôdu se koristi var allAccidents = [] koja je potrebna kasnije za vizualno grupiranje prometnih nesreća. Taj kôd se može vidjeti na slici 26.

```
291 var allAccidents = [];
292 function loadTrafficAccidents() {
293     $.ajax({
294         type: "GET",
295         url: "@Url.Action("loadTrafficData")",
296         contentType: "application/json; charset=utf-8",
297         dataType: "json",
298         success: successfullyLoadedFunction,
299         failure: function (response) {
300             alert(response.error);
301         }
302     });
303 }
304 function successfullyLoadedFunction(responseWeb) {
305     if (responseWeb.succ == false) {
306         alert(responseWeb.error);
307     }
308     else {
309         console.log(VarPolUprava);
310         allAccidents = responseWeb.accidents;
311         for (var i = 0; i < responseWeb.accidents.length; i++) {
312             var acc = responseWeb.accidents[i];
313             var iconFeature = new ol.Feature({
314                 geometry: new ol.geom.Point(ol.proj.fromLonLat([acc.Duljina, acc.Sirina])),
315                 value: plotedObjects.length
316             });
317             vectorSource.addFeature(iconFeature);
318             var icon = { postion: plotedObjects.length, iconFeature: iconFeature, acc: acc };
319             plotedObjects.push(icon);
320         }
321         var layer = new ol.layer.Vector({
322             source: vectorSource,
323             style: new ol.style.Style({
324                 image: new ol.style.Circle({
325                     radius: 5,
326                     fill: new ol.style.Fill({
327                         color: 'rgba(255,0,0,0.7)'
328                     }),
329                     stroke: new ol.style.Stroke({
330                         color: 'rgba(0, 0, 0, 0.8)',
331                         width: 1
332                     })
333                 })
334             })
335         });
336         map.addLayer(layer);
337     }
338 }
```

Slika 26. JavaScript kôd za učitavanje prometnih nesreća

Gumb *Group all accidents by PolUprava* vizualno grupira sve prometne nesreće po policijskim upravama. Njegov izgled napisan je kôdom:

```
<button class="btn btn-primary" type="button" style="width: 100%; margin: 2px;
padding: 3px" + onclick="groupTrafficAccidents()">
        Group all accidents by PolUprava
</button>
```

Segment kontrolera isti je kao i kod učitavanja svih prometnih nesreći. Kontroler treba poslati naredbu modelu da u bazi izabere sve prometne nesreće. Kontroler sve prometne nesreće stavlja u listu *accidents* čija je klasa WebResponse. Ta se lista prosljeđuje natrag u pogled. Izgled kontrolera prikazan je kôdom ispod.

```
[HttpGet]
        public ActionResult GroupTrafficData()
        {
            WebResponse wres = new WebResponse();
            try
            {

                wres.accidents = dbNesrece.Nesrecas.ToList();
                wres.succ = true;
            }
            catch (Exception ex)
            {
                wres.errorDesc = ex.Message;
                wres.succ = false;
            }

            return Content(JsonConvert.SerializeObject(wres),
"application/json");
        }
```

Glavni dio javascript kôda prikazan na slici 27. Prije nego što korisnik pritisne gumb *Group all accidents by PolUprava*, mora pritisnuti gumb *Load all accidents* kako bi se u varijabli *allAccidents* upisale sve prometne nesreće. Zbog toga se poziva funkcija `eraseAccidents()` koja briše prethodne prometne nesreće. Uzimaju se vrijednosti koje se nalaze u ViewBag-u za policijske uprave, te za svaku jedinstvenu upravu spremaju u listu te im se stvara poseban sloj. Kako bi se prometne nesreće za svaku policijsku upravu razlikovale za svaki se sloj koristi druga boja. Boja se dobiva pomoću funkcije koja stvara slučajnu boju svaki put kada se pozove. Kôd te funkcije može se vidjeti na slici 28. Na kraju taj sloj učitava u kartu.

```
876  function groupTrafficAccidents() {
877      eraseAccidents();
878      if (allAccidents == undefined) {
879          alert("Need to load all accidents first!");
880      }
881      else {
882          for (var index in VarPolUprava) {
883              var poUprava = VarPolUprava[index];
884              console.log(poUprava);
885              vectorSource = new ol.source.Vector({});
886              for (var i = 0; i < allAccidents.length; i++) {
887                  if (allAccidents[i].PolUprava == poUprava) {
888                      var acc = allAccidents[i];
889                      var iconFeature = new ol.Feature({
890                          geometry: new ol.geom.Point(ol.proj.fromLonLat([acc.Duljina, acc.Sirina])),
891                          value: plotedObjects.length
892                      });
893
894                      vectorSource.addFeature(iconFeature);
895                      var icon = { postion: plotedObjects.length, iconFeature: iconFeature, acc: acc };
896                      plotedObjects.push(icon);
897                  }
898              }
899              var color = random_rgba();
900              var layer = new ol.layer.Vector({
901                  source: vectorSource,
902                  style: new ol.style.Style({
903                      image: new ol.style.Circle({
904                          radius: 5,
905                          fill: new ol.style.Fill({
906
907                              color: color,
908
909
910                          }),
911                          stroke: new ol.style.Stroke({
912                              color: 'rgba(0, 0, 0, 0.8)',
913                              width: 1
914                          })
915                      })
916                  })
917              });
918              plotedVectorSources.push(vectorSource);
919              map.addLayer(layer);
920          }
921      }
922  }
```

Slika 27. JavaScript kôd za grupiranje prometnih nesreća

```
924
925 function random_rgba() {
926     var o = Math.round, r = Math.random, s = 255;
927     return 'rgba(' + o(r() * s) + ',' + o(r() * s) + ',' + o(r() * s) + ',' + r().toFixed(1) + ')';
928 }
929
```

Slika 28. Kôd za stvaranje slučajne boje

Gumb *Reset filtered accidents* briše prethodni sloj kako bi se moglo napraviti slijedeće filtriranje. Izgled ovog gumba napisan je kôdom:

```
<button class="btn btn-primary" type="button" style="width: 100%; margin: 2px;
padding: 3px" + onclick="filterTrafficAccidentsReset()">
        Reset filtered accidents
</button>
```

Prilikom pritiska na gumb *Reset filtered accidents* poziva se funkcija `eraseAccidents()`. Ta funkcija određuje da li se na karti nalazi više ili jedan sloj te sve slojeve pobrisati kako bi se moglo koristiti novi filter. Kôd funkcije `eraseAccidents()` prikazan je ispod.

```
function eraseAccidents() {
        if (plotedVectorSources != undefined && plotedVectorSources.length > 0) {
            for (var i in plotedVectorSources) {
                plotedVectorSources[i].clear();
            }
        }
        else {
            vectorSource.clear();
        }
        plotedObjects=[];
    }
```

Gumb *Filter accidents by PolUprava* filtrira prometne nesreće po odabranim policijskim upravama. Njegov izgled napisan je kôdom:

```
<button class="btn btn-primary" type="button" style="width: 100%; margin: 2px;
padding: 3px" + onclick="filterTrafficAccidents()">
        Filter accidents by PolUprava
</button>
    <br>
    <label for="dropdown">Select an option:</label>
<select id="dropdownPOUP">
        @foreach (var item in ViewBag.listaPolUprava)
        {
            <option value="@item">@item</option>
        }
</select>
```

U div dijelu gumba *Filter accidents by PolUprava* nalazi se petlja koja za svaku vrijednost u `ViewBag.listaPolUprava` kreira novo polje u padajućem izborniku. U *ViewBagu* nalaze se sve jedinstvene policijske uprave. Kako bi se padajući izbornik mogao odabrati potrebno je napraviti posebnu varijablu u kojoj se sprema odabrana policijska uprava. Ta varijabla se popuni s odgovarajućom vrijednošću kada se dogodi događaj *change* nad padajućom listom. Događaj *change* prepoznaje kada korisnik promjeni vrijednost, odnosno pritisne jednu od opcija padajućeg izbornika te ju šalje natrag u pogled gdje postaje varijabla `selectedOptionPOUP`. Kôd ispod prikazuje stvaranje padajućeg izbornika.

```
var dropdown = document.getElementById("dropdownPOUP");
    var selectedOptionPOUP = "";

    dropdown.addEventListener("change", function () {
        selectedOptionPOUP = dropdown.options[dropdown.selectedIndex].value;
```

```
    });
```

Korisnik prvo mora odabrati željenu policijsku upravu koristeći padajući izbornik, zatim pritiska gumb *Filter accidents by PolUprava*. Pogled vraća u kontroler odabranu policijsku upravu, koju on šalje modelu. Model stvara upit u bazi podataka te sve prometne nesreće vraća u kontroler koji ih stavlja u listu. Dio kôda koji se nalazi u kontroleru glasi:

```
[HttpGet]
        public ActionResult FilterTrafficAccidentsController(string
selectedOptionPOUP)
        {
            WebResponse wres = new WebResponse();
            try
            {
                wres.accidents = dbNesrece.Nesrecas.Where(x => x.PolUprava ==
selectedOptionPOUP).ToList();
                wres.succ = true;
            }
            catch (Exception ex)
            {
                wres.errorDesc = ex.Message;
                wres.succ = false;
            }

            return Content(JsonConvert.SerializeObject(wres),
"application/json");
        }
```

Lista prometnih nesreća koje se nalaze u odabranoj policijskoj upravi šalje se natrag u pogled gdje se svakoj prometnoj nesreći daje ikona koja se nalazi u sloju te se na kraju sloj učitava u kartu. Taj kôd je prikazan na slici 29.

```
293     function loadTrafficAccidents() {
294         $.ajax({
295             type: "GET",
296             url: "@Url.Action("loadTrafficData")",
297             contentType: "application/json; charset=utf-8",
298             dataType: "json",
299             success: successfullyLoadedFunction,
300             failure: function (response) {
301                 alert(response.error);
302             }
303         });
304
305     }
306
307
308     function successfullyLoadedFunction(responseWeb) {
309         if (responseWeb.succ == false) {
310             alert(responseWeb.error);
311         }
312         else {
313             console.log(VarPolUprava);
314             allAccidents = responseWeb.accidents;
315             for (var i = 0; i < responseWeb.accidents.length; i++) {
316                 var acc = responseWeb.accidents[i];
317                 var iconFeature = new ol.Feature({
318                     geometry: new ol.geom.Point(ol.proj.fromLonLat([acc.Duljina, acc.Sirina])),
319                     value: plotedObjects.length
320                 });
321                 vectorSource.addFeature(iconFeature);
322                 var icon = { postion: plotedObjects.length, iconFeature: iconFeature, acc: acc };
323                 plotedObjects.push(icon);
324             }
325             var layer = new ol.layer.Vector({
326                 source: vectorSource,
327                 style: new ol.style.Style({
328                     image: new ol.style.Circle({
329                         radius: 5,
330                         fill: new ol.style.Fill({
331                             color: 'rgba(255,0,0,0.7)'
332                         }),
333                         stroke: new ol.style.Stroke({
334                             color: 'rgba(0, 0, 0, 0.8)',
335                             width: 1
336                         })
337                     })
338                 })
339             });
340             map.addLayer(layer);
341         }
342     }
```

Slika 29. JavaScript kôd za filtriranje prometnih nesreća po policijskim upravama

Funkcija gumba *Filter accidents by DatumPN* je odabir i prikaz prometnih nesreća čiji se datum nalazi u intervalu. Taj interval određuje korisnik. Izgled gumba i izbornika za odabir granica intervala napisan je kôdom:

```
<button class="btn btn-primary" type="button" style="width: 100%; margin: 2px;
padding: 3px" + onclick="filterTrafficAccidentsDate()">

        Filter accidents by Date
</button>
<label for="startDate"> Start date:</label>
<input type="date" id="startDate"><br>
<label for="endDate"> End date:</label>
<input type="date" id="endDate"><br>
```

U web aplikaciji osim upisivanja željenih granica datuma također postoji opcija pritiska na kućicu koja otvara cijeli kalendar te pomoću kojeg korisnik može odrediti datum. U kôdu se nalazi događaj *change* koji kada korisnik odabere datume stavlja tu vrijednost u varijable selectedStartDate i selectedEndDate. Kôd za stvaranje tih varijabli prikazan je ispod:

```
var sDate = document.getElementById("startDate");

    sDate.addEventListener("change", function () {
        selectedStartDate = sDate.value;
    });

    var eDate = document.getElementById("endDate");

    eDate.addEventListener("change", function () {
        selectedEndDate = eDate.value;
    });
```

Nakon stvaranja varijabli za granice intervala, ajax poziv te varijable predaje u dio kontrolera koji upravlja filtriranjem prometnih nesreća po datumima. Kontroler predaje te varijable granica modelu koji šalje upit u bazu podataka. Te granice nisu uključujuće pa se sve prometne nesreće koje se nalaze unutar željenih granica, a nisu uključene, stavljaju u listu te se natrag šalju u pogled. Kako radi kontroler prikazano je kôdom ispod.

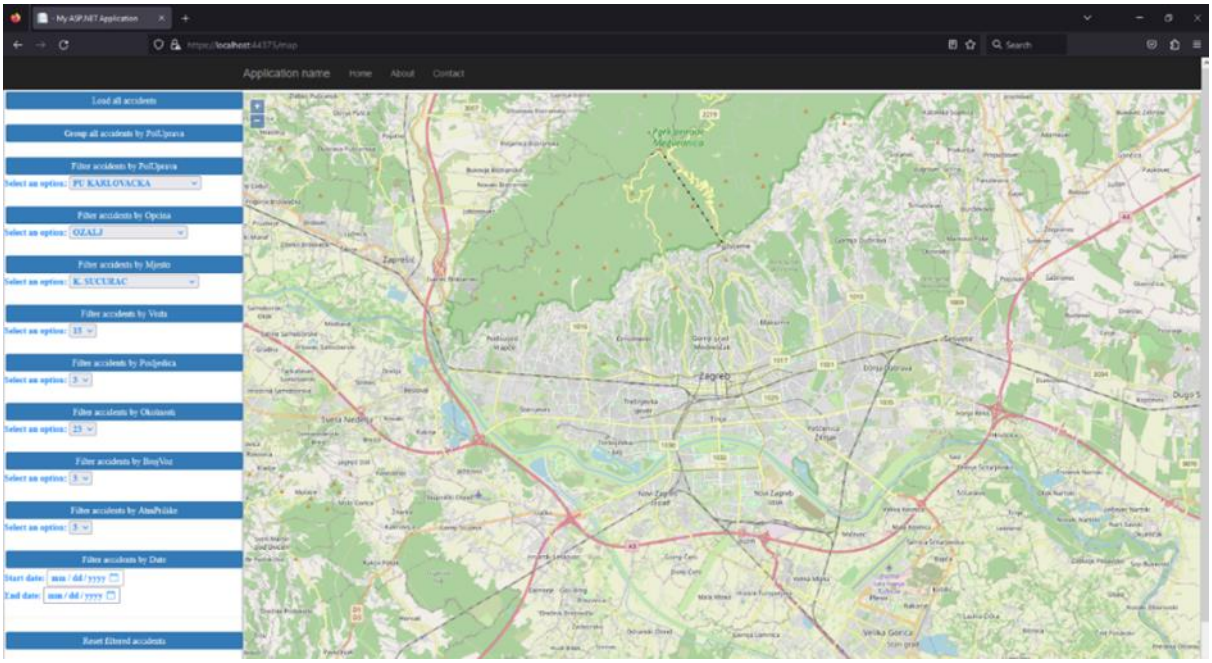
```
[HttpGet]
        public ActionResult FilterTrafficAccidentsDate(DateTime
selectedStartDate, DateTime selectedEndDate)
        {
            WebResponse wres = new WebResponse();
            try
            {
                wres.accidents = dbNesrece.Nesrecas.Where(x => x.DatumPN >
selectedStartDate && x.DatumPN < selectedEndDate).ToList();
                wres.succ = true;
            }
            catch (Exception ex)
            {
                wres.errorDesc = ex.Message;
                wres.succ = false;
            }

            return Content(JsonConvert.SerializeObject(wres),
"application/json");
        }
```

Nakon što se lista sa filtriranim prometnim nesrećama vratila u pogled, stvaraju im se ikone koje se stavljaju u sloj. Na kraju sloj se stavlja na kartu. Izgled kôda koji za sve nesreće stvara ikone te ih stavlja u sloj, a sloj u kartu prikazan je slikom 30.

```
824  function filterTrafficAccidentsDate() {
825      console.log(selectedStartDate);
826      console.log(selectedEndDate);
827      $.ajax({
828          type: "Get",
829          url: "@Url.Action("FilterTrafficAccidentsDate")",
830          contentType: "application/json; charset=utf-8",
831          data: { selectedStartDate: selectedStartDate, selectedEndDate: selectedEndDate },
832          dataType: "json",
833          success: successfullyLoadedFunctionDate,
834          failure: function (response) {
835              alert(response.error);
836          }
837      });
838  }
839  function successfullyLoadedFunctionDate(responseWeb) {
840      console.log(responseWeb);
841      if (responseWeb.succ == false) {
842          alert(responseWeb.error);
843      }
844      else {
845          for (var i = 0; i < responseWeb.accidents.length; i++) {
846              var acc = responseWeb.accidents[i];
847              var iconFeature = new ol.Feature({
848                  geometry: new ol.geom.Point(ol.proj.fromLonLat([acc.Duljina, acc.Sirina])),
849                  value: plotedObjects.length
850              });
851
852              vectorSource.addFeature(iconFeature);
853              var icon = { postion: plotedObjects.length, iconFeature: iconFeature, acc: acc };
854              plotedObjects.push(icon);
855          }
856          var layer = new ol.layer.Vector({
857              source: vectorSource,
858              style: new ol.style.Style({
859                  image: new ol.style.Circle({
860                      radius: 5,
861                      fill: new ol.style.Fill({
862                          color: 'rgba(100, 255, 50, 0.8)'
863                      }),
864                      stroke: new ol.style.Stroke({
865                          color: 'rgba(0, 0, 0, 0.8)',
866                          width: 1
867                      })
868                  })
869              })
870          });
871          map.addLayer(layer);
872      }
873  }
```

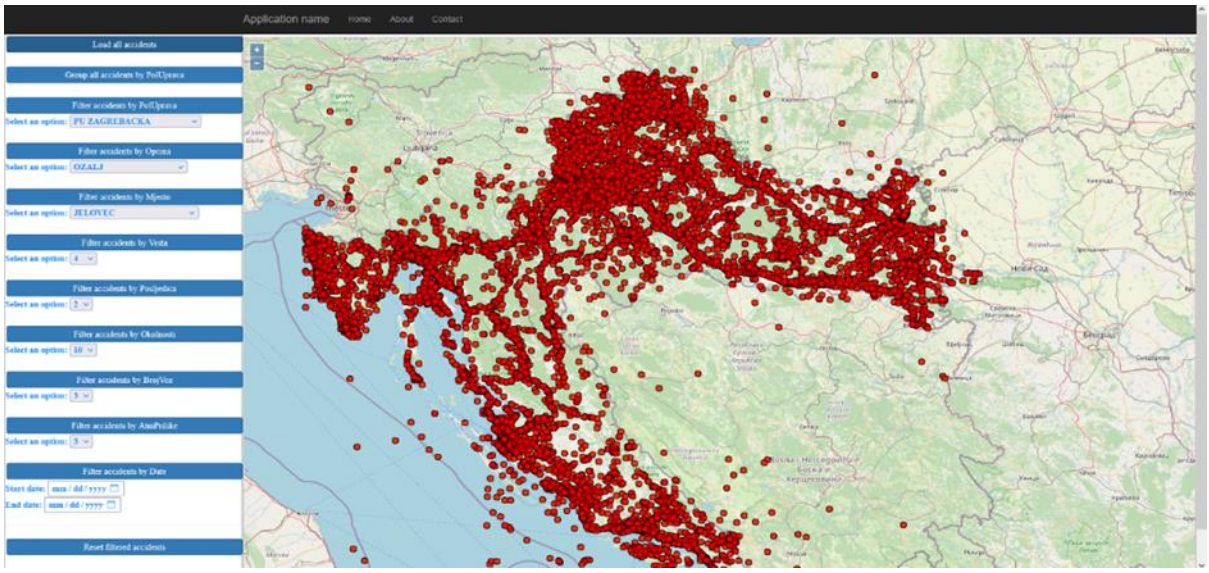
Slika 30. JavaScript kôd za filtriranje prometnih nesreća po datumu

Web aplikacija pokreče se u Firefox-u. Ona omogućuje vizualizaciju prometnih nesreća krajnjem korisniku. Lijeva strana aplikacije sadržava bočnu traku koja ima funkcije za bolju vizualizaciju prometnih nesreća, dok se desna strana sastoji od geografske karte. Karta ima mogućnost zumiranja te je početna točka postavljena na Zagreb. Aplikacija ima mogućnost prikazivanja svih nesreća te njihovu grupaciju po policijskim upravama. Korisničko sučelje aplikacije prikazana je slikom 31.



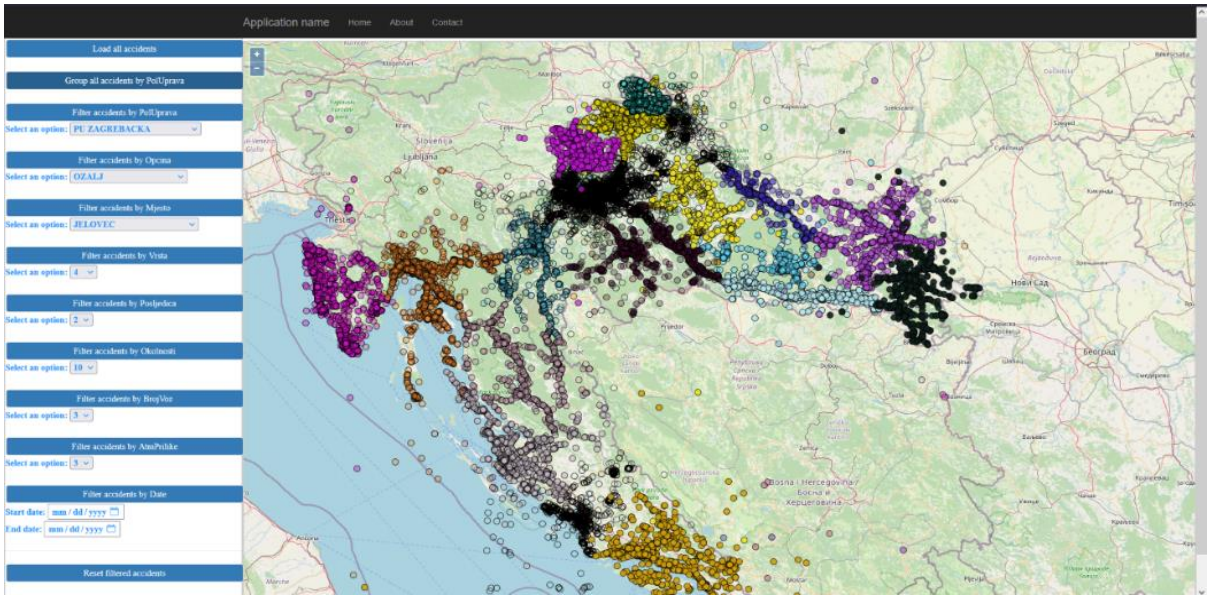
Slika 31. Izgled korisničkog sučelja bez ikakvih filtera

Izgled korisničkog sučelja kada se pritisne gumb *Load all accidents* prikazan je slikom 32.



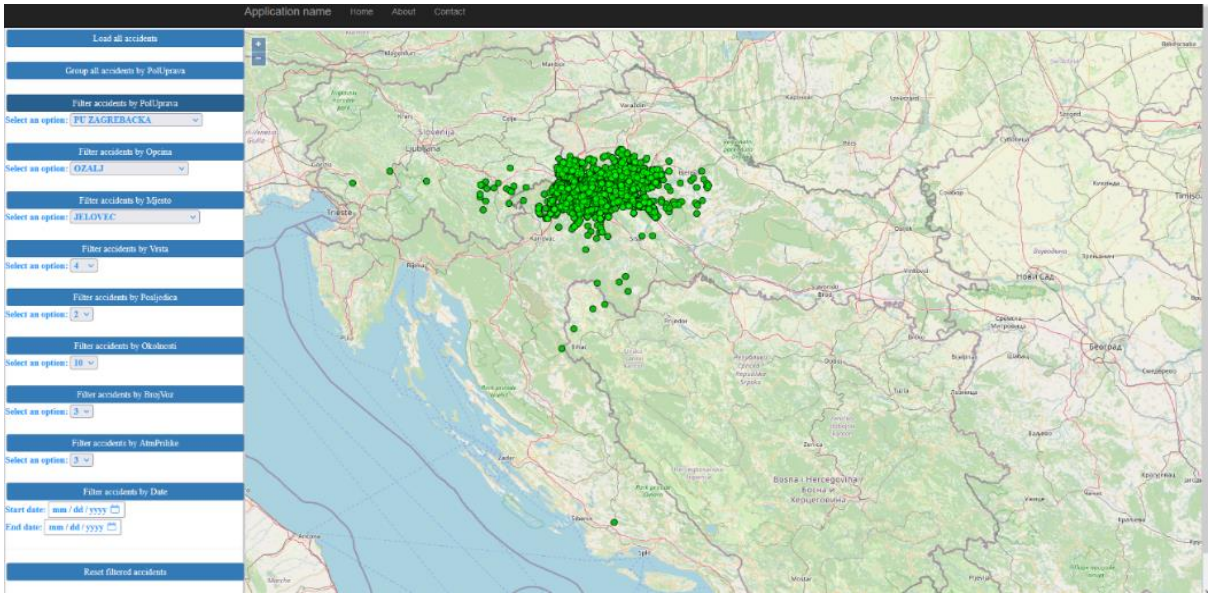
Slika 32. Izgled korisničkog sučelja kada se pritisne gumb *Load all accidents*

Izgled korisničkog sučelja kada se pritisne gumb *Group all accidents by PolUrava* prikazan je slikom 33.



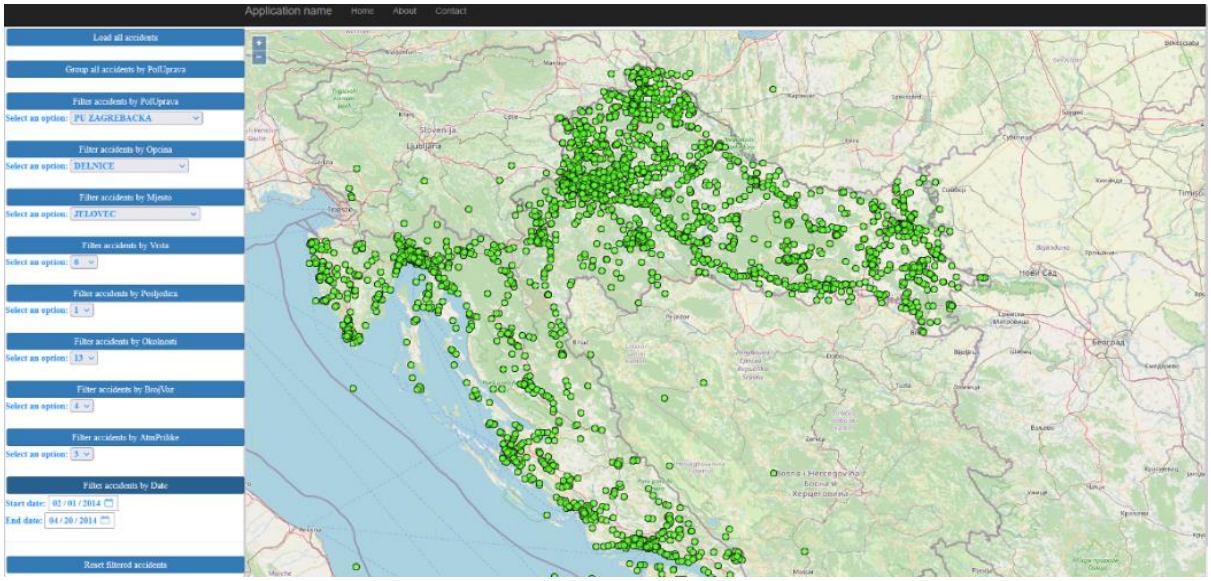
Slika 33. Izgled korisničkog sučelja kada se pritisne gumb *Group all accidents by PolUprava*

Izgled korisničkog sučelja kada se pritisne gumb *Filter accidents by PolUprava* gdje je odabrana zagrebačka policijska uprava prikazan je slikom 34.



Slika 34. Izgled korisničkog sučelja kada se pritisne gumb *Filter accidents by PolUprava*

Izgled korisničkog sučelja kada se pritisne gumb *Filter accidents by DatumPN* gdje je interval od 1. veljače 2014. do 20. travnja 2014. prikazan je slikom 35.



Slika 35. Izgled korisničkog sučelja kada se pritisne gumb *Filter accidents by DatumPN*

5. Analiza dobivenih rezultata

U ovom poglavlju završnog rada analizirati će se podaci koristeći SQL Server. Kako bi se podaci analizirali prvo je potrebno izračunati ukupan broj prometnih nesreća. Upit za izračunavanje broja svih prometnih nesreća glasi:

```
SELECT COUNT(*) FROM Nesreca.
```

Ukupan broj prometnih nesreća glasi 31432.

5.1 Analiza po policijskim upravama

Kako bi izračunali postotke prometnih nesreća za svaku policijsku upravu, prvo je potrebno izvući koje su sve jedinstvene policijske uprave, upit za to glasi:

```
SELECT DISTINCT PolUprava FROM Nesreca.
```

Na slici 36. je prikazan odgovor na prethodni upit.

	PolUprava
1	PU KARLOVACKA
2	PU LICKO-SENJSKA
3	PU ŠIBENSKO-KNINSKA
4	PU VARAŽDINSKA
5	PU ISTARSKA
6	PU DUBROVACKO-NERET.
7	PU VIROVITICKO-PODR.
8	PU BRODSKO-POSAVSKA
9	PU POŽEŠKO-SLAVONSKA
10	PU SPLITSKO-DALMAT.
11	PU MEĐIMURSKA
12	PU OSJECKO-BARANJSKA
13	PU PRIMORSKO-GORAN.
14	PU KOPRIVNICKO-KRIŽ.
15	PU BJELOVARSKO-BIL.
16	PU ZADARSKA
17	PU ZAGREBACKA
18	PU KRAPINSKO-ZAGOR.
19	PU SISACKO-MOSLAV.
20	PU VUKOVARSKO-SRIJ.

Slika 36. Sve jedinstvene policijske uprave

Primjer upita za dobivanje broja prometnih nesreća za svaku policijsku upravu glasi:

```
SELECT  DISTINCT COUNT(PolUprava), PolUprava FROM Nesreca
GROUP BY PolUprava
```

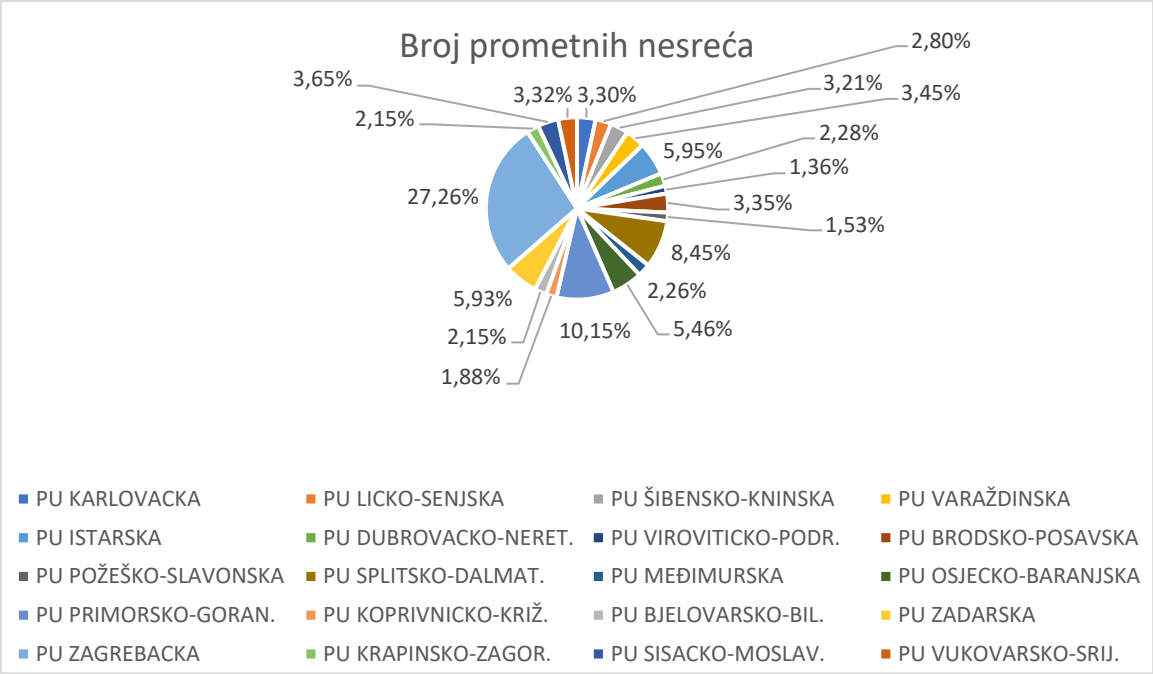
Dobiveni brojevi prometnih nesreća u policijskim upravama prikazani su u tablici 2.

Tablica 2. Broj nesreća za sve policijske uprave

Policijska uprava	Broj prometnih nesreća
PU KARLOVACKA	1038
PU LICKO-SENJSKA	883
PU ŠIBENSKO-KNINSKA	1011
PU VARAŽDINSKA	1085
PU ISTARSKA	1871

PU DUBROVACKO-NERET.	719
PU VIROVITICKO-PODR.	430
PU BRODSKO-POSAVSKA	1054
PU POŽEŠKO-SLAVONSKA	483
PU SPLITSKO-DALMAT.	2659
PU MEĐIMURSKA	713
PU OSJECKO-BARANJSKA	1718
PU PRIMORSKO-GORAN.	3191
PU KOPRIVNICKO-KRIŽ.	593
PU BJELOVARSKO-BIL.	677
PU ZADARSKA	1866
PU ZAGREBACKA	8570
PU KRAPINSKO-ZAGOR.	678
PU SISACKO-MOSLAV.	1148
PU VUKOVARSKO-SRIJ.	1045

Na slici 37. prikazan je grafikon broja prometnih nesreća u postocima.



Slika 37. Graf količina prometnih nesreća po policijskim upravama

Najveći broj prometnih nesreća ima u PU Zagrebacka (8570 ili 27,26%) dok najmanje prometnih nesreća ima u PU Viroviticko-podr. (430 ili 1,53%). Samo PU Zagrebacka, PU Primorsko-goran., PU Splitsko-dalmat., PU Istarska pokrivaju skoro 50% ukupnog broja prometnih nesreća.

5.2 Analiza po posljedicama

Kako bi izračunali postotak prometnih nesreća za svaku vrstu posljedice, prvo trebamo izvući koje su sve jedinstvene posljedice, upit za to glasi:

```
SELECT DISTINCT Posljedica FROM Nesreca.
```

Odgovor tog upita je da postoje tri tipa posljedice: 1, 2, 3. Primjer upita za dobivanje broja prometnih nesreća za sve posljedice glasi:

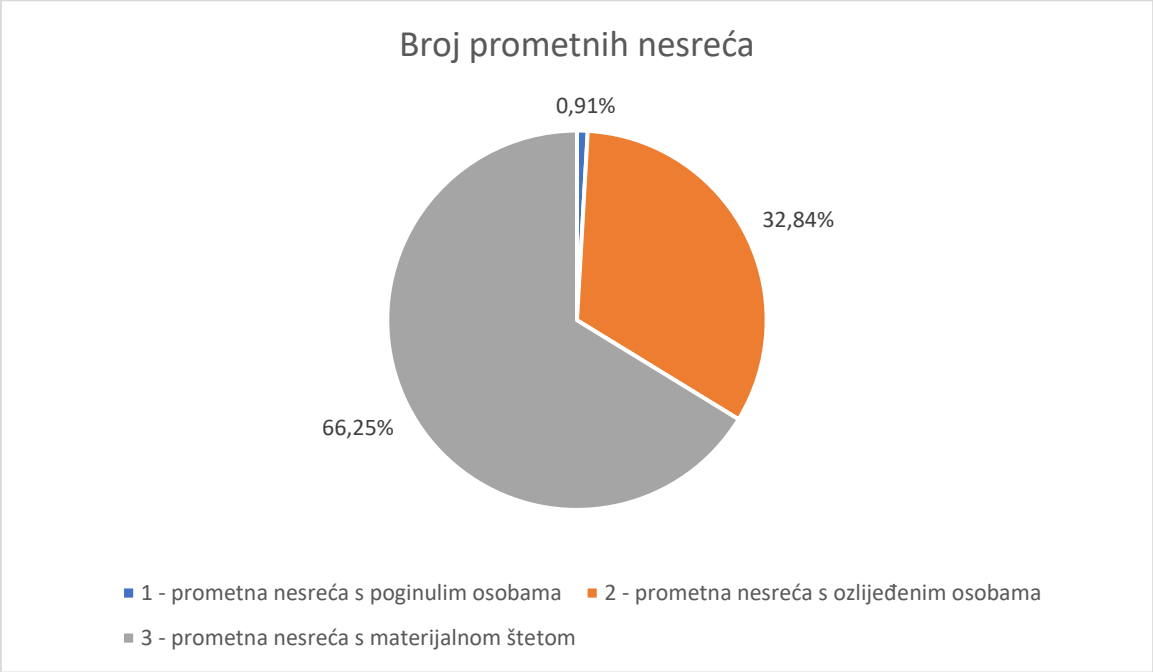
```
SELECT DISTINCT COUNT(Posljedica), Posljedica FROM Nesreca
GROUP BY Posljedica
```

Dobiveni brojevi prometnih nesreća za sve posljedice prikazani su u tablici 3.

Tablica 3. Broj nesreća za sve posljedice

Posljedica	Broj prometnih nesreća
1 - prometna nesreća s poginulim osobama	284
2 - prometna nesreća s ozlijeđenim osobama	10323
3 - prometna nesreća s materijalnom štetom	20825

Na slici 38. prikazan je grafikon broja prometnih nesreća u postocima.



Slika 38. Graf količina prometnih nesreća po posljedicama

Prema vrsti posljedica najviše je bilo prometnih nesreća s materijalnom štetom (20825 ili 66,25%), njih slijede prometne nesreće s ozlijeđenim osobama (10323 ili 32,84%), dok najmanje ima prometnih nesreća s poginulim osobama (284 ili 0,91%).

5.3 Analiza po vrstama

Kako bi izračunali postotak prometnih nesreća za svaku vrstu prometne nesreće, potrebno je izvući koje su sve jedinstvene vrste, upit za to glasi:

```
SELECT DISTINCT Vrsta FROM Nesreca.
```

Na slici 39. je prikazan odgovor na prethodni upit.

	Vrsta
1	15
2	9
3	3
4	12
5	6
6	1
7	18
8	10
9	4
10	19
11	5
12	16
13	2
14	17
15	11
16	14
17	8

Slika 39. Sve jedinstvene vrste prometnih nesreća

Primjer upita za dobivanje broja prometnih nesreća za svaku jedinstvenu vrstu glasi:

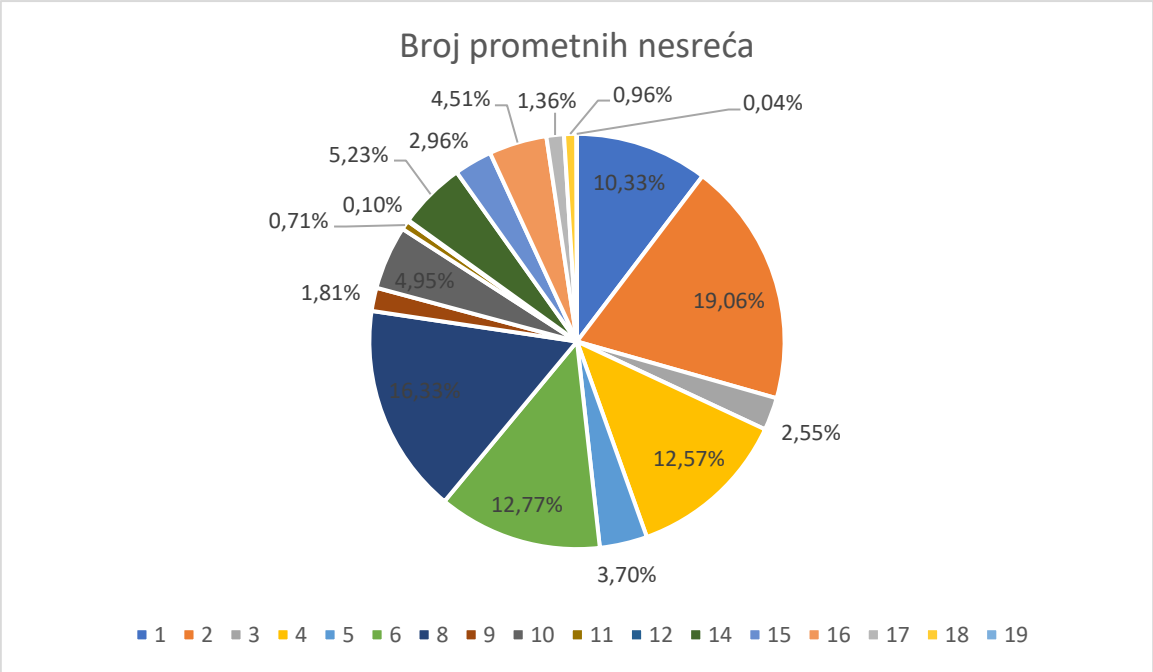
```
SELECT DISTINCT COUNT(Vrsta), Vrsta FROM Nesreca
GROUP BY Vrsta
```

Dobiveni brojevi prometnih nesreća za svaku vrstu prikazani su u tablici 4.

Tablica 4. Broj nesreća za sve vrste

Vrsta	Broj prometnih nesreća
1 – sudar iz bočnog smjera	3250
2 – bočni sudar	5991
3 – usporedna vožnja	802
4 – vožnja u slijedu	3954
5 – vožnja unatrag	1163
6 – udar vozila u parkirano vozilo	4014
8 – slijetanje vozila s ceste	5134
9 – nalet na bicikl	569
10 – nalet na pješaka	1557
11 – nalet na motocikl ili moped	224
12 – sudar sa željezničkim vozilom	32
14 – ostalo	1647
15 – udar vozila u objekt na cesti	931
16 – udar vozila u objekt kraj ceste	1419
17 – nalet na domaću životinju	428
18 – nalet na divlju životinju	304
19 – nalet na pticu	13

Na slici 40. prikazan je grafikon broja prometnih nesreća u postocima.



Slika 40. Graf količina prometnih nesreća po vrsti

Prema vrsti prometne nesreće najviše ima bočnih sudara (5991 ili 19,06%), njih slijede slijetanja vozila s ceste (5134 ili 16,33%). Vrste prometnih nesreća s najmanjim brojem su naleti na pticu (13 ili 0,04%) i sudari s željezničkim vozilom (32 ili 0,10%).

5.4 Analiza po broju vozila

Kako bi izračunali postotak prometnih nesreća za svaki broj vozila, potrebno je izvući sve jedinstvene brojeve vozila, upit za to glasi:

```
SELECT DISTINCT BrojVoz FROM Nesreca.
```

Na slici 41. je prikazan odgovor na prethodni upit.

	BrojVoz
1	3
2	6
3	7
4	1
5	4
6	5
7	2
8	8

Slika 41. Svi jedinstveni brojevi vozila

Primjer upita za dobivanje broja prometnih nesreća za sve brojeve vozila glasi:

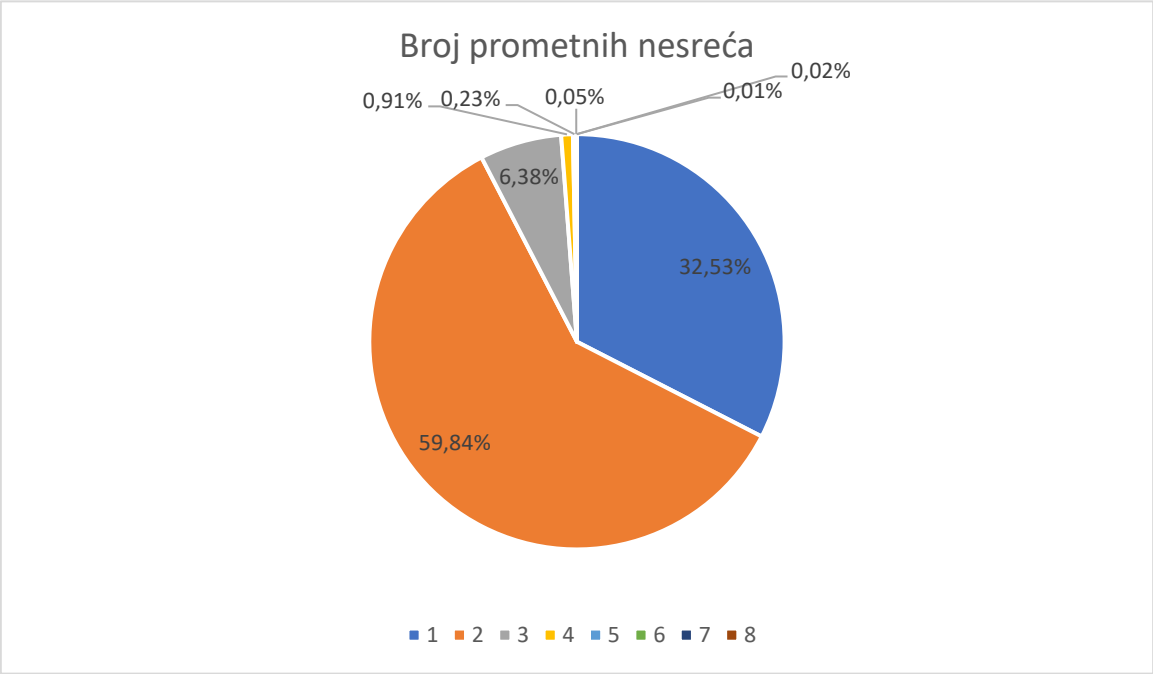
```
SELECT DISTINCT COUNT(BrojVoz), BrojVoz FROM Nesreca
GROUP BY BrojVoz
```

Dobiveni brojevi prometnih nesreća za sve brojeve vozila prikazani su u tablici 5.

Tablica 5. Broj nesreća za sve brojeve vozila

Broj vozila	Broj prometnih nesreća
1	10227
2	18811
3	2006
4	289
5	73
6	18
7	6
8	2

Na slici 42. prikazan je grafikon broja prometnih nesreća u postocima.



Slika 42. Graf količina prometnih nesreća po broju vozila

U najviše prometnih nesreća sudjelovala su dva vozila (18811 ili 59,84%) i jedno vozilo (10227 ili 32,53%). Najmanje je bilo prometnih nesreća koje su imale osam (2 ili 0,01%) i sedam (6 ili 0,02%) vozila. Prometne nesreće s dva i jednim vozilom skoro pokrivaju 93% svih prometnih nesreća. Najčešći uzroci zbog kojih se takve nesreće događaju su nepažnja vozača, neodgovorno ponašanje, konzumacija alkohola i opojnih sredstava, umor i nepoštivanje prometnih propisa i pravila.

5.5 Analiza po atmosferskim prilikama

Kako bi izračunali postotak prometnih nesreća za svaku atmosfersku priliku, potrebno je izvući sve jedinstvene atmosferske prilike, upit za to glasi:

```
SELECT DISTINCT AtmPrilike FROM Nesreca.
```

Na slici 43. je prikazan odgovor na prethodni upit.

	AtmPrilike
1	3
2	6
3	7
4	1
5	4
6	5
7	2

Slika 43. Sve jedinstvene atmosferske prilike

Primjer upita za dobivanje broja prometnih nesreća za svaku atmosfersku priliku glasi:

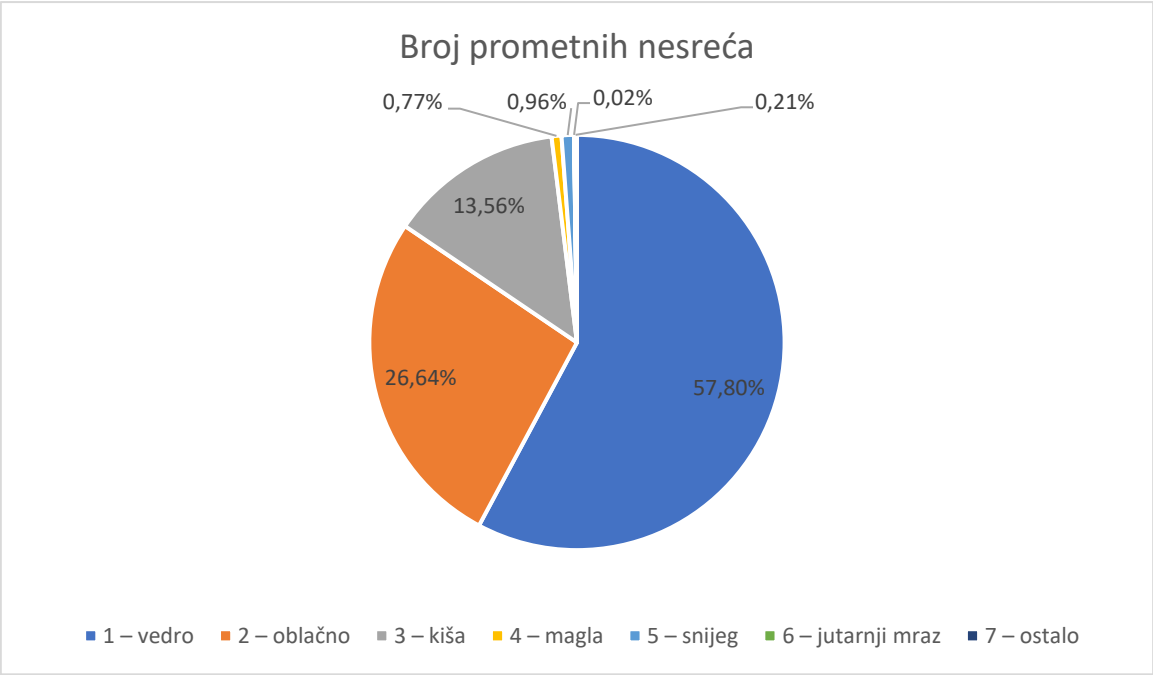
```
SELECT DISTINCT COUNT(AtmPrilike), AtmPrilike FROM Nesreca
GROUP BY AtmPrilike
```

Dobiveni brojevi prometnih nesreća za svaku atmosfersku priliku prikazani su u tablici 6.

Tablica 6. Broj nesreća za sve atmosferske prilike

Atmosferske prilike	Broj prometnih nesreća
1 – vedro	18169
2 – oblačno	8376
3 – kiša	4265
4 – magla	245
5 – snijeg	303
6 – jutarnji mraz	7
7 – ostalo	67

Na slici 44. prikazan je grafikon broja prometnih nesreća u postocima.



Slika 44. Graf količina prometnih nesreća po atmosferskim prilikama

Najviše se prometnih nesreća dogodilo kada je bilo vedro (18169 ili 57,80%), dok ih slijede prometne nesreće koje su se dogodile kada je bilo oblačno (8376 ili 26,64%). Najmanje prometnih nesreća se dogodilo kada je bio jutarnji mraz (7 ili0,02%). Prometne nesreće koje su se dogodile kada je bilo vedro i oblačno pokrivaju skoro 85% svih prometnih nesreća.

5.6 Analiza po datumu

Kod analize prometnih nesreća vezano s datumom koristit će se dvanaest kalendarskih mjeseci. Primjer upita za dobivanje broja prometnih nesreća u jednom kalendarskom mjesecu glasi:

```
SELECT COUNT(*) FROM Nesreca
WHERE (DatumPN >= '2014-09-01 00:00:00.000' AND DatumPN <= '2014-09-30 23:59:59.999')
```

Dobiveni brojevi prometnih nesreća za određene kalendarske mjesece prikazani su u tablici 7.

Tablica 7. Broj nesreća za sve kalendarske mjesece

Kalendarski mjesec	Broj prometnih nesreća
Siječanj	2095
Veljača	2019
Ožujak	2271
Travanj	2509
Svibanj	2757
Lipanj	2823
Srpanj	3108
Kolovoz	3273
Rujan	2803
Listopad	2726
Studeni	2373
Prosinac	2675

Na slici 45. prikazan je grafikon broja prometnih nesreća u postocima.

Slika 45. Graf količine prometnih nesreća po kalendarskim mjesecima

Najviše prometnih nesreća dogodilo se u kolovozu (3273 ili 10,41%) i u srpnju (3108 ili 9,88%). Najmanje se prometnih nesreća dogodilo u veljači (2019 6,42%) i siječnju (2095 ili 6,66%). Veći broj prometnih nesreća u ljetnim mjesecima događa se zbog većeg broja vozila na prometnicama zbog turističke sezone.

6. Zaključak

Zbog povećane potražnje za prometom i transportom dolazi do povećanog rizika nastajanja prometnih nesreća. U ovom završnom radu vizualno se prikazuju sve prometne nesreće koje su se dogodile u 2014. godini na području Hrvatske.

Ovaj rad u sebi ima dva programa. Svrha prvog programa je unos podataka o prometnim nesrećama u bazu podataka. Drugi program te podatke o prometnim nesrećama iz baze podataka vizualizira na karti koja se nalazi na web-u. Drugi program rađen je u MVC modelu koji se sastoji od tri dijela: model, pogled i kontroler. Model se povezuje s bazom podataka i služi za slanje upita bazi. Pogled određuje izgled same web stranice i šalje dobivene informacije korisniku. Kontroler ima funkciju komunikacije između modela i pogleda. Web aplikacija u sebi ima funkcije filtriranja po određenim atributima kao što su policijske uprave, općine, datumi, vrste prometnih nesreća, broj vozila i drugi. Na kraju su napravljene analize svih prometnih nesreća. Analize uključuju broj prometnih nesreća za svaku policijsku upravu, broj prometnih nesreća po vrsti prometne nesreće, broj prometnih nesreća po količini vozila koja su bila u tim prometnim nesrećama, broj prometnih nesreća za svaki kalendarski mjesec i drugi.

Ova aplikacija zamišljena je za predputno informiranje korisnika, da oni mogu vidjeti koje prometnice su više rizične te kako bi bolje mogli izabrati rutu. Ova aplikacija se može dodatno proširiti kako bi se poboljšala učinkovitost i da daje važnije podatke. Moguće je spojiti sve filtere kako bi se mogla obaviti bolja filtracija prometnih nesreća.

Literatura

1. Bošnjak I. Inteligentni transportni sustavi - ITS 1 Zagreb: Fakultet prometnih znanosti; 2006.

2. Carić T, Erdelić T. Odobrene prezentacije iz Baza podataka. [Online].; 2023. [cited 2023. Srpanj.] Available from: https://moodle.srce.hr/2022-2023/mod/resource/view.php?id=2682774.

3. Protrka M. Identifikacija i analiza opasnih mjesta na cestovnoj prometnoj mreži grada imotskog. 2018..

4. Ćosić M. Kontekstualna analiza prometnih nesreća pješaka i biciklista u urbanim sredinama. 2017..

5. Wikipedia. Relation database. [Online].; 2023. [cited 2023. Srpanj.] Available from: https://en.wikipedia.org/wiki/Relational_database.

6. Škorput P. Odobrene prezentacije iz Računalne sigurnosti. [Online].; 2023. [cited 2023. Srpanj.] Available from: https://moodle.srce.hr/2022-2023/course/view.php?id=140512.

7. Microsoft. Choose an Authentication Mode. [Online].; 2023. [cited 2023. Srpanj.] Available from: https://learn.microsoft.com/en-us/sql/relational-databases/security/choose-an-authentication-mode?view=sql-server-ver16.

8. Sqlgeekpro. Windows Authentication. [Online].; 2023. [cited 2023. Srpanj.] Available from: https://sqlgeekspro.com/windows_auth_diff_user/.

9. Aggarwal A. Introduction to Visual Studio. [Online].; 2022. [cited 2023. Srpanj.] Available from: https://www.geeksforgeeks.org/introduction-to-visual-studio/.

10. Wikipedia. C Sharp (programming language). [Online].; 2023. [cited 2023. Srpanj.] Available from: https://en.wikipedia.org/wiki/C_Sharp_(programming_language).

11. Wikipedia. Model-view-controller. [Online].; 2023. [cited 2023. Srpanj.] Available from: https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller.

12. Geekforgeeks. MVC Framework Introduction. [Online].; 2023. [cited 2023. Srpanj.] Available from: https://www.geeksforgeeks.org/mvc-framework-introduction/.

13. Vujić M. Odobrene prezentacije iz Arhitekture inteligentnih transportnih sustava. [Online].; 2023. [cited 2023. Srpanj.] Available from: https://moodle.srce.hr/2022-2023/course/view.php?id=140243.

14. Geekforgeeks. Frontend vs Backend. [Online].; 2023. [cited 2023. Srpanj.] Available from: https://www.geeksforgeeks.org/frontend-vs-backend/.

15. Geekforgeeks. What is NuGet. [Online].; 2020. [cited 2023. Srpanj.] Available from: https://www.geeksforgeeks.org/what-is-nuget/.

16. Mozilla. Working with JSON. [Online].; 2023. [cited 2023. Srpanj.] Available from: https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/JSON.

17. Stackify. ViewBag 101. [Online].; 2023. [cited 2023. Srpanj.] Available from: https://stackify.com/viewbag/.

18. Wikipedia. Cascading Styles Sheets. [Online].; 2023. [cited 2023. Srpanj.] Available from: https://en.wikipedia.org/wiki/CSS.

19. OpenStreetMap. Open Layers. [Online].; 2022. [cited 2023. Srpanj.] Available from: https://wiki.openstreetmap.org/wiki/OpenLayers.

Popis slika

Popis tablica

Sveučilište u Zagrebu
Fakultet prometnih znanosti
Vukelićeva 4, 10000 Zagreb

IZJAVA O AKADEMSKOJ ČESTITOSTI I SUGLASNOSTI

Izjavljujem i svojim potpisom potvrđujem da je _____ završni rad _____
(*vrsta rada*)
isključivo rezultat mojega vlastitog rada koji se temelji na mojim istraživanjima i oslanja se na objavljenu literaturu, a što pokazuju upotrijebljene bilješke i bibliografija. Izjavljujem da nijedan dio rada nije napisan na nedopušten način, odnosno da je prepisan iz necitiranog rada te da nijedan dio rada ne krši bilo čija autorska prava. Izjavljujem, također, da nijedan dio rada nije iskorišten za bilo koji drugi rad u bilo kojoj drugoj visokoškolskoj, znanstvenoj ili obrazovnoj ustanovi.

Svojim potpisom potvrđujem i dajem suglasnost za javnu objavu završnog/diplomskog rada pod naslovom Izrada web aplikacije za prikaz podataka o prometnim nesrećama, u Nacionalni repozitorij završnih i diplomskih radova ZIR.

Student/ica:

U Zagrebu, 17.07.2023.

Marko Kukor
(ime i prezime, *potpis*)