Načini pretraživanja podataka u web aplikacijama

Hrvoje, Šoštarić

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Hrvoje Šoštarić

NAČINI PRETRAŽIVANJA PODATAKA U WEB APLIKACIJAMA

DIPLOMSKI RAD

Varaždin, 2021.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Hrvoje Šoštarić

Matični broj: 0016118031

Studij: Informacijsko i programsko inženjerstvo

NAČINI PRETRAŽIVANJA PODATAKA U WEB APLIKACIJAMA

DIPLOMSKI RAD

Mentor/Mentorica:

Izv. prof. dr. sc. Sandro Gerić

Varaždin, rujan 2021.

Hrvoje Šoštarić

Izjava o izvornosti

Izjavljujem da je moj završni/diplomski rad izvorni rezultat mojeg rada te da se u izradi istoga nisam koristio drugim izvorima osim onima koji su u njemu navedeni. Za izradu rada su korištene etički prikladne i prihvatljive metode i tehnike rada.

Autor/Autorica potvrdio/potvrdila prihvaćanjem odredbi u sustavu FOI-radovi

Sažetak

Tema ovog rada je bilo istražiti načine pretraživanja podataka u web aplikacijama i konkretno implementirati ekspertni sustav pomoću velike količine podataka. Na početku istraživanja sam se pozabavio sa algoritmima za pretraživanje nad općim strukturama podataka i pretraživanjem same baze podatka sa SQL-om. Opisan je način rada sa LINQ-om uz navedene primjere kao i osnovne tehnike strojnog učenja. U praktičnome dijelu je implementiran ekspertni sustav za predikciju cijene kuća u obliku web aplikacije. Cjelokupno rješenje se sastoji od poslužiteljske aplikacije implementirane u programskome jeziku C# i klijentske implementirane pomoću programskom okvira Angular, baziranog na TypeScript-u. Osim implementacije funkcionalnosti predikcije cijene kuća, još je implementirano pretraživanje podatka prema cijeni uz sortiranje po stupcima silazno i uzlazno uz straničenje i CRUD operacije.

Ključne riječi: LINQ, strojno učenje, pretraživanje podataka, baza podataka, C#, Angular, web servis.

Sadržaj

Sadržaj iii
1. Uvod 1
2. Načini pretraživanja podataka na različitim platformama 3
2.1. Pretraživanja struktura podataka 3
2.1.1. Slijedno pretraživanje 3
2.1.2. Binarno pretraživanje 4
2.1.3. Pretraživanje hashiranjem 5
2.1.3.1. Prikazivanje rada hash funkcija 6
2.1.4. Pretraživanje u dubinu 7
2.1.5. Pretraživanje u širinu 9
2.2. Pretraživanje baza podataka 11
2.2.1. Pretraživanje strukturalnim upitnim jezikom 11
2.2.2. Pretraživanje baze podataka u programskome jeziku C# 14
3. Jezik integriranog upita (LINQ) 18
3.1. LINQ općenito 18
3.2. Upitna i funkcijska sintaksa 20
4. Strojno učenje 22
4.1. Nadzirano učenje 23
4.1.1. Regresija 24
4.1.1.1. Linearna regresija 25
4.1.2. Klasifikacija 27
4.1.2.1. K-najbližih susjeda 29
4.1.2.2. Stablo odlučivanja 29
4.1.2.3. Klasifikacija „Slučajnom“ šumom 31
4.1.2.4. Klasifikacija „Naivnim“ Bayesom 33
4.2. Nenadzirano učenje 33

4.2.1. Sakupljanje 34
4.2.1.1. Sakupljanje K-sredinom 35
4.2.2. Učenje pravilom udruživanja 37
5. Praktični primjer 39
5.1. Analiza korištenog skupa podataka 39
5.2. Analiza poslužiteljskog dijela aplikacije 40
5.2.1. Izrada komponente ML.NET 42
5.2.2. Implementacija poslužiteljskog dijela aplikacije 46
5.3. Analiza klijentskog dijela aplikacije 51
5.4. Korištenje aplikacije 54
6. Zaključak 60
Popis literature 61
Popis slika 66
Popis tablica 69

1. Uvod

U suvremeno doba mnoge svakodnevne aktivnosti se odvijaju ili su dostupne putem mrežnih platformama. Temeljni sadržaj sadržaja koji su prikazani u obliku mrežnih stranica ili aplikacija su podaci. Sam podatak možemo definirati kao sirov skup simbola koji jednostavno postoje i nemaju nikakav značaj izvan postojanja [1]. Njime ustvari pokušavamo „uhvatiti“ pravu sliku o stvarnom događaju koji moramo znati pročitati i interpretirati [2]. Njegovim interpretiranjem dobivamo informaciju, odnosno obavijest. Informacija ne predstavlja ništa drugo nego podatak sa koji predstavlja određeni značaj te je stvorena analizom odnosa i veza između podataka. Njezin je zadatak da odgovori na pitanja poput “Tko/Što/Gdje/Kada/Koliko/Zašto je” i u odnosu na podatak je točnija, relevantnija i svježija [1][2]. Prikupljanjem informacija dolazimo do sljedećeg pojma koji nam je koristan, a to je znanje. Znanje možemo definirati kao odgovarajuće prikupljanje informacija i podataka tako da njegova namjena bude korisna i da rezultira sa određenom aktivnošću [1] [2]. Uz dodatak ekspertnog mišljenja i vještina, znanje dobiva mogućnost da određuje kako koristiti informaciju [2]. Sljedeći pojam koji nam je bitan kod pristupanju informacijama i podacima jest pojam baza podataka. Definiciju baze podatka možemo objasniti kao kolekciju podataka, ograničenja i operacija koja reprezentira neke aspekte realnog svijeta [3]. Laički, tj. sa korisničke strane baze podataka još možemo definirati i kao skup međusobno povezanih tablica od kojih su danas najpopularnije relacijske baze podataka kod kojih se podaci nalaze u relacijama, odnosno u tablicama [4]. Dva važna pojma koja su direktno povezana na baze podataka jesu model podataka i sustav za upravljanje bazom podatka. Uloga sustava za upravljanje bazom podatka (SUBP) je da oblikuje fizički izgled baze podataka u skladu sa traženom logičkom strukturom, te u ime klijenata obavlja sve operacije nad podacima. Može podržati razne baze, od kojih svaka može imati svoju logičku strukturu, ali moraju imati isti model [4]. Model podataka baze podataka možemo definirati kao skup pravila koja određuju kako sve može izgledati logička struktura baze podataka. On čini osnovu za oblikovanje i implementaciju same baze [3]. Sastoji se od triju komponenata, strukturalne komponente S koja prikazuje u kojemu su obliku prikazani podaci, integritetne komponente UI koju čine ograničenja na dozvoljena stanja strukture i operativne komponente O koja čini operacije nad strukturama [3].

Svi prethodno navedeni pojmovi korisniku sami po sebi nemaju neki preveliki značaj. Da bi korisnik od tih podataka imao koristi potrebno ih je prikazati putem određenih aplikacija. Uspoređujući mrežne aplikacije prije dvadesetak godina kada je bilo specifično da se klijent za odabranu akciju šalje zahtjev mrežnome poslužitelju, dobiva odgovor nakon bijelog ekrana koji je rezultat čekanja odgovora od strane servera, u današnje vrijeme se uz pomoć raznih tehnologija poput AJAX-a (engl. Asynchronous JavaScript And XML) dobiva na raspolaganje

bogate internet aplikacije uz visoku razinu responzivnosti i interakcije pri čemu na jednome ekranu odraditi više događaja tako da mu se podaci prikazuju na ekranu bez da se na zahtjev ekran osvježava [5]. Svakim novim danom korisnikova očekivanja u vezi performansa mrežnih aplikacija rastu iz dana uz dan. Njihova intuitivnost, fleksibilnost i dostupnost u svako doba dana neke su od obveznih karakteristika koje bi svaka mrežna aplikacija morala korisniku pružati. Kako bi se korisniku omogućila dostupnost i brzi pristup do njegovih omiljenih aplikacija sve više aplikacija biva orijentirano na dostupnost putem oblaka (engl. *Cloud*). Također pojavljuju se i neke nove arhitekture poput mikro-servisa, koji programeru olakšava njihov razvoj, testiranje kao, objavljivanje i održavanje. Još neke karakteristike koje nismo spomenuli ranije, a specifične su za današnje mrežne aplikacije jest nezavisnost od platforme na kojoj se izvršavaju. Takve aplikacije još nazivamo i progresivne mrežne aplikacije i mogu se izvršavati i biti dostupne putem raznih mrežnih preglednika, operacijskih sustava kao i na raznim uređajima (računala, tableti, mobilni uređaji i sl.) [6].

2. Načini pretraživanja podataka na različitim platformama

U ovome poglavlju ćemo detaljnije opisati i predočiti načine pretraživanja podataka na različitim platformama kao što su strukture podataka i baza. Kod struktura podataka ćemo započeti sa obradom metoda kao što su slijedno i binarno pretraživanje, a završiti ćemo sa pretraživanjima poput „Pretraživanja u dubinu“ (engl. *Depth-first search*) i „Pretraživanje u širinu“ (engl. *Breadth-first search*). Kod dijela vezanog za baze podataka obradit ćemo osnovna pretraživanja pomoću SQL-a te ćemo se dotaknuti i pretraživanja pomoću programskih okvira koji su nastali kako bi programerima olakšali rad sa bazama i operacijama nad njima.

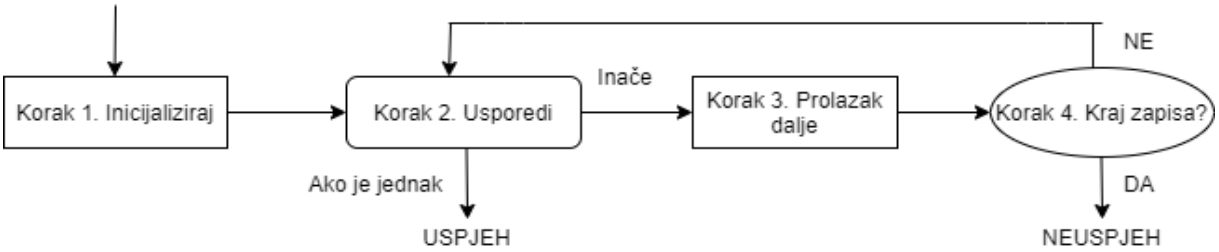
2.1. Pretraživanja struktura podataka

Prema [7, 392. str] pojam pretraživanje možemo definirati kao pokušaj pronalaska određenog skupa podataka nad određenom strukturom sa dva moguća rezultata, a to su uspjeh i neuspjeh. Pod uspjehom podrazumijevamo da je zapis pronađen, dok pod neuspjehom podrazumijevamo da zapis ne postoji te ga je potrebno dodati. Strukture koje možemo pretraživati mogu biti jednostavna polja, liste, rječnici na bazi ključ-vrijednost (engl. *key-value*), grafovi, stabla i sl. Vrste podataka koje možemo pretraživati mogu biti jednostavni poput tekstualnih i numeričkih zapisa, pa mogu varirati i do složenih objekata [8, 393 str.]. Prilikom odabira algoritma kojeg ćemo koristiti moramo imati na umu da je pretraživanje je jedna od vremensko-najzahtjevnijih procedura mnogih aplikacija i zamjena dobre metode za lošu može biti pogubna za rad aplikacije i njezine performanse kao i njegovu prikladnost za korištenu strukturu podataka, specifičnost problema, svojstvo podataka. složenost algoritama i sl. [7][8].

2.1.1. Slijedno pretraživanje

Prvi algoritam, a ujedno i najjednostavniji koji ćemo obraditi jest slijedno pretraživanje. Slijedno pretraživanje kao što se i prema nazivu može zaključiti baziran je na slijednome pretraživanju tako da se krene od početka zadane strukture te se prolazi kroz svaki element strukture te ispituje je li trenutni element jednak traženoj vrijednosti i tako sve dok element/zapis nije pronađen [7][8]. U najgorem slučaju traženi element može biti pozicioniran na kraju strukture te će se pretraživanje izvršiti onoliko puta koliko struktura ima elemenata. Iz toga slijedi da mu je vremenska složenost jednaka $O(N)$.

U nastavku ćemo prikazati grafički prikaz algoritma urađenog prema [7] na kojoj je pojednostavljeni prikaz prethodno opisanih radnji.



Slika 1. Dijagram slijednog pretraživanja (Izrada autora prema [7])

Uz standardnu vrstu slijednog pretraživanja postoje i modificirane verzije poput brzog slijednog pretraživanja (engl. *Quick sequential search*) koji kreće sa pretraživanjem od zadnjeg elementa, slijedno pretraživanje nad sortiranim tablicama koji slijedno pretražuje strukturu sa sortiranim podacima itd. [7].

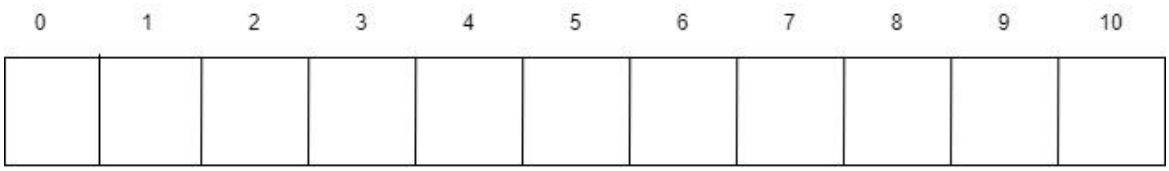
2.1.2. Binarno pretraživanje

U ovome poglavlju ćemo opisati algoritam koji vrši pretraživanje nad isključivo sortiranom strukturom podataka, a to je algoritam pod nazivom binarno pretraživanje. Specifičnost ovog pretraživanja je u tome da se može provoditi samo nad sortiranim strukturama podataka. Brzo se provodi nad zapisima sa velikom i srednjom količinom podataka te održava susjednu podređenost susjednog niza kao i vjerojatnost gdje se ciljana vrijednost sigurno nalazi [7]. Algoritam binarnog pretraživanja je pripadan obitelji algoritama Podijeli pa ovladaj (engl. *Divide-and-conquer*). Algoritam se sprovodi tako uzmemo strukturu sa zapisima nad kojom provodimo traženje elementa sa ključem K. Pozicioniramo vrijednosti desne strane zapisa $r = 0$ te lijeve strane zapisa $l = N$, pri čemu je N broj zapisa u strukturi. Zatim odredimo element sredine m, koji je jednak poziciji $[(l + r)/2]$ te tako dobivamo sortiranu strukturu podijeljenu na dva dijela (*Podijeli*) [7] [8]. Ukoliko je element m jednak traženome elementu završavamo sa traženjem, inače, ovisno o vrijednosti traženog elementa pomičemo traženje na lijevu ili desnu stranu strukture elemenata. Uzmimo za primjer da nam je tražena vrijednost elementa veća od srednjeg elementa što znači da odlazimo sa traženjem na lijevo. Vrijednost elementa l postavljamo na $l = m + 1$ te pronalazimo novi srednji element i ponavljamo postupak rekurzivno sve dok ne pronađemo element ukoliko postoji (*Ovladaj*) [7][8].

Za razliku od slijednog pretraživanja koji ima složenost $O(N)$, algoritam binarnog pretraživanja ima složenost $O(logN)$ te je mnogo prikladniji za odabir ukoliko su elementi zapisani u sortiranome redoslijedu [8]. Algoritam binarnog pretraživanja se može lakše grafički prikazati pomoću binarnog stabla odlučivanja o kojemu će više riječi u poglavlju vezanome za strojno učenje.

2.1.3. Pretraživanje hashiranjem

Sljedeća tehnika pretraživanja koju ćemo opisati jest hashiranje. Pretraživanje hashiranjem jedna je od najboljih i najefikasnijih tehnika pretraživanja čije je vrijeme pretraživanja neovisno o količini podataka u cijeloj tablici. Neki od problema za koje je ovo pretraživanje dobro su konstruiranje indeksa, pohrana objektno orijentiranih baza podataka, brzo pretraživanje velikih organiziranja podataka, autentifikacija i integracija podataka u kriptografiji, pretraživanje dokumenata na internetu i sl. [7]. Podaci su uglavnom sadržani u tablici koju još nazivamo i hash tablicom (Slika 2.) koja se sastoji od M broja utora koji sadrže određenu stavku koju ćemo kasnije tražiti [9].



Slika 2. Prazna hash tablica sa 11 utora

Pozicija zapisa u tablici određena je ključem za svaki zapis, a njegova pozicija je određena pomoću hash funkcije. Ukoliko je H hash funkcija i K je ključ, tada je H(K) se zove hash ključa. Tako hash funkcija H(K) transformira ključ u adresu/hash poziciju zapisa. Ukoliko ključ nije cjelobrojna varijabla tada se on pretvara u cjelobrojnu vrijednost [7]. Neke od hash funkcija koje se koriste za hashiranje jesu:

- Divizijska metoda (engl. *Division method*)
- Kvadratna metoda (engl. *Midsquare method*)
- Metoda mapiranja (engl. *Folding method*)
- Metoda isključivo-ILI (XOR) za znakovne varijable (engl. *Variable string exclusive-or method*)

Divizijska metoda je do svih njih najjednostavnija. Vrijednost $H(K)$ dobiva se pomoću formule:

$$H(K) = K \; mod \; M$$

pri čemu M predstavlja broj raspoloživih mjesta u tablici, a K vrijednosti ključa koji pohranjujemo u tablicu [8].

Kvadratna metoda hashiranja koristi se za veličinu hash tablica koja je potencija od 2. U ovoj metodi ključ se množi sam sa sobom, a adresa se dobiva odabirom broja u sredini kvadratnog rješenja. Hash funkciju $H(K)$ definiramo pomoću jednadžbe:

$$H(K) = L$$

pri čemu je L jednak K^2 [7][8].

U metodi mapiranja ključ dijelimo na jednak broj dijelova (komadića znamenki) te njihova suma daje konačnu hash vrijednost. Hash funkciju H možemo zapisati kao $H(K) = K_1 + K_2 + \cdots + K_n$ [7] [9].

Metoda isključivo-ILI (XOR) za znakovne varijable bazira se na dobivanju hash vrijednosti u rasponu od 0-255, pri čemu su svi bajtovi u skupu znakova zajedno povezani pomoću XOR funkcije. Tijekom rada sa XOR-om, koja je ustvari i bazična metoda za kriptografiju, imamo uključenost nasumične komponente [7].

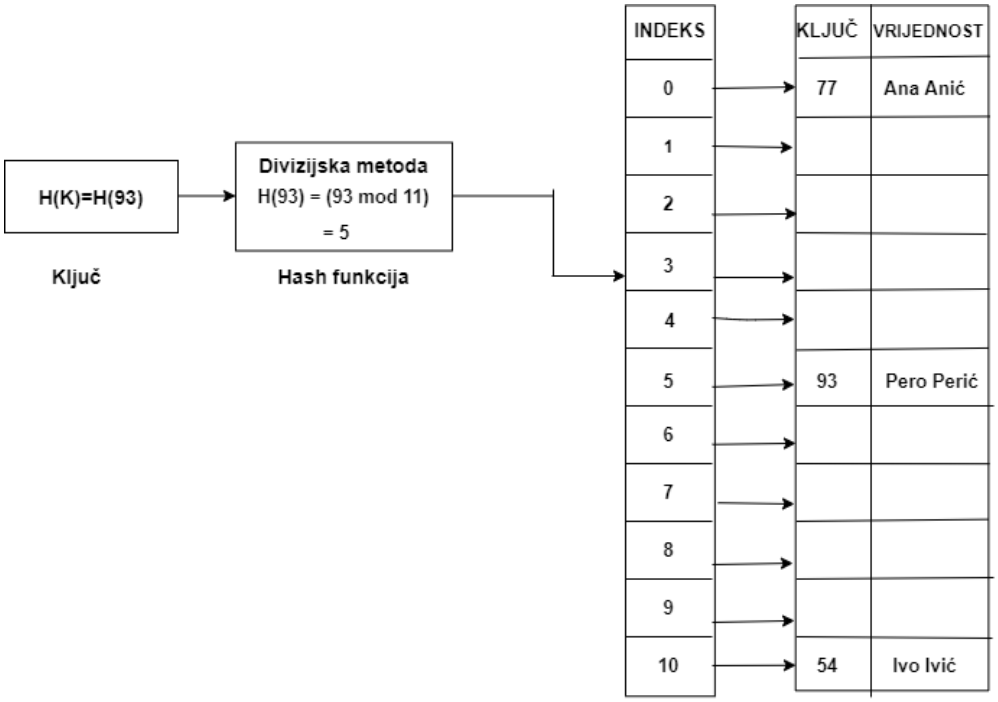
2.1.3.1. Prikazivanje rada hash funkcija

U ovome poglavlju ćemo na primjerima prikazati funkcioniranje hash funkcija opisanih u prethodnom poglavlju. U sljedećoj tablici ćemo prikazati postupke dobivanja hash indeksa za hash funkcije divizijske, kvadratne i metode mapiranja za smještaj elementa sa vrijednosti 93 u hash tablicu veličine 11.

Tablica 1. *Računski prikaz dobivanja indeksa za određene algoritme (Izrada autora)*

Divizijska metoda	**Kvadratna metoda**	**Metoda mapiranja**
$H(93) = (93\ mod\ 11)$ $= 5$	$H(93) = 93^2\ mod\ 11$ $= 8649\ mod\ 11$ $= 64\ mod\ 11$ $= 9$	$H(93) = (9 + 3)\ mod\ 11$ $= 12\ mod\ 11$ $= 1$

U tablici 1 vidimo da svaka hash funkcija daje različitu vrijednost za istu vrijednost ključa. Za razliku od slijednog i binarnog pretraživanja koji su imali složenosti $O(N)$(slijedno) i $O(logN)$ (binarno), pretraživanje u hash tablicama ima složenost $O(1)$ jer se bazira na ključu-vrijednost. Ukoliko stavimo ključeve iz prethodne tablice kao identifikacije studenata postupak pronalaženja vrijednosti je identičan kao na slici 3 pomoću divizijske metode.



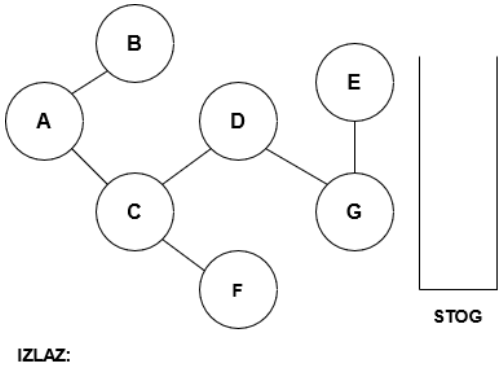
Slika 3. *Pronalazak osobe pomoću hash funkcije (Izrada autora prema [7])*

Na prethodnoj slici vidimo na koji način su pohranjeni zapisi u hash tablicama pomoću najjednostavnije hash funkcije. Međutim, jedan od najvećih nedostataka hash funkcija jest pojava kolizije zbog pojave duplikata vrijednosti hash funkcija. Kako bi se ti problemi riješili koriste se metode koje kako bi riješile te probleme. Neke od tih metoda su Lančane (eng. *Chaining*) i Buckets [8].

2.1.4. Pretraživanje u dubinu

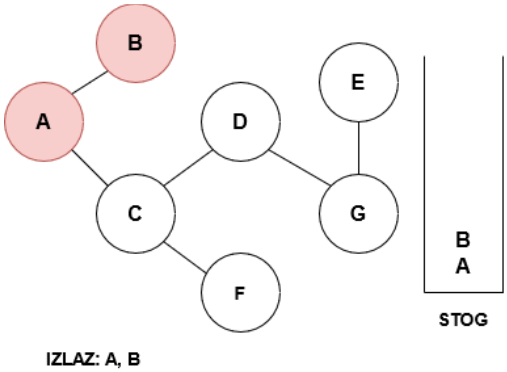
Algoritam pretraživanja u dubinu (engl. *Depth-first search*) rekurzivan je algoritam koji se bazira na obitelji algoritma poznatom pod nazivom pretraživanje s vraćanjem (engl. *Backtracking*). Ova vrsta pretraživanja pogodna je za strukture podataka kao što su grafovi i stabla tako da uključuje obilazak svih njegovih čvorova dok ih ne obiđe sve. Rekurzivno pretraživanje se implementira pomoću stoga. Ideja algoritma je da se krene od korijena stabla/grafa te da se vrijednost stavi u stog. Ukoliko se tijekom obilaska vratimo na čvor koji se već nalazi u stogu maknemo ga iz stoga i to radimo sve dok ne ispraznimo stog [10].

U nastavku ćemo na primjeru prikazati kako taj algoritam u stvari radi. Na slici 4 imamo graf sa 7 vrhova te praznim stogom u koji ćemo stavljati vrhove koje smo obišli.



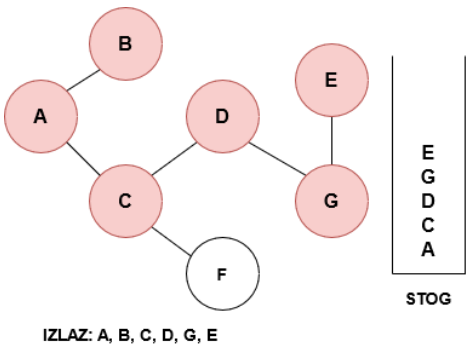
Slika 4. *Inicijalni graf sa praznim stogom*

Krenut ćemo sa vrhom A te ćemo ga dodati na vrh stoga i u izlazni niz. Vidimo da on ima dva susjedna vrha koja nisu posjećena, B i C. Ići ćemo abecednim redoslijedom i dodamo B na vrh stoga i izlazni niz (slika 5).



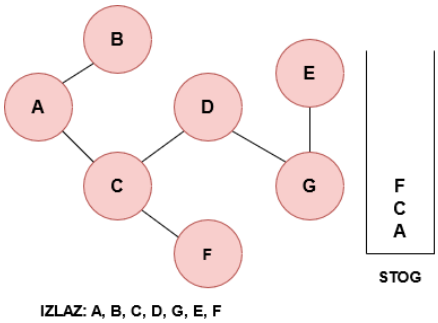
Slika 5. Stanje stoga i niza nakon posjeta vrhu B (Izrada autora)

Kako vidimo da vrh B nema susjednih vrhova koji nisu posjećeni uklonimo ga sa vrha stoga i vratimo se na vrh A i krenemo na sljedeći neposjećeni vrh C te ga dodamo na vrh stoga i vrijednost C dodamo u izlazni niz. Nakon toga pogledamo neposjećene susjedne vrhove (D i F) te posjetimo vrh D te ga dodamo na vrh stoga. Za vrh D vidimo da ima jedan susjedni vrh G te ga dodamo u izlazni niz i istu stvar ponovimo za vrh E. Tada dobivamo situaciju kao na slici 6.



Slika 6. Stanje izlaza i stoga nakon posjeta vrhovima D, G, E (Izrada autora)

Pošto vidimo da vrh E nema neposječenih vrhova, uklonimo ga sa vrha stoga i krenemo provjeravat situaciju za vrh G. Vidimo da je ista situacija kao i sa vrhom E te istu stvar ponovimo i za vrh D. Vratimo se na vrh C za koji vidimo da ima neposjećenog susjeda F te ga dodamo na vrh stoga i u izlazni niz (slika 7).



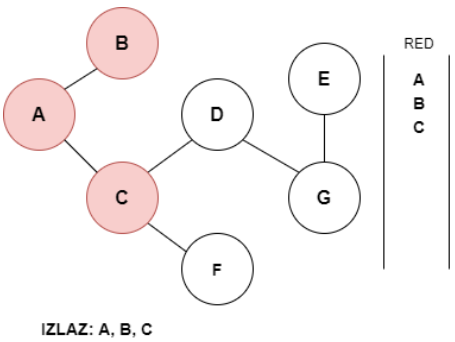
Slika 7. Stanje stoga i izlaznog niza nakon posjeta preostalome vrhu F (Izrada autora)

Nakon posjeta vrha F provjeravamo da nema neposjećenih susjednih vrhova te ga uklanjamo sa stoga. Istu stvar napravimo i sa vrhovima C i A te smo ispraznili stog i posjetili sve vrhove. Vremenska složenost algoritma iznosi O(V+ ε) pri čemu je V broj vrhova, a ε broj bridova grafa [11].

2.1.5.Pretraživanje u širinu

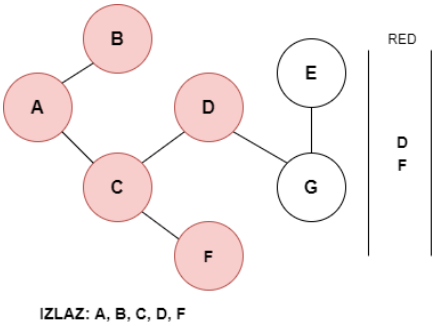
Sljedeći algoritam koji ćemo obraditi također služi za pretraživanje stabala i grafova samo sa drugačijim načinom pretraživanja nego prethodni, a naziva se pretraživanje u širinu (engl. *breadth-first search*). Algoritam pretraživanja u širinu je algoritam pretraživanja u kojemu se iz odabranog vrha (čvora) kreće prema susjednim čvorovima slijedno. Za razliku od prethodnog algoritma koji se implementira pomoću stoga, algoritam pretraživanja u širinu se implementira pomoću reda [11].

U nastavku ćemo na primjeru iz prethodnog poglavlja prikazati kako taj algoritam funkcionira. Započnimo od vrha A. Dodamo ga u izlaznu vrijednost te u red. Zatim posjetimo njegove susjede B i C te ih dodamo u red (slika 8).



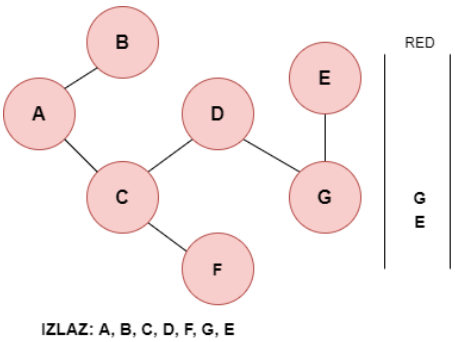
Slika 8. Stanje reda i izlaza nakon posjeta susjeda vrha A (Izrada autora)

Nakon što smo posjetili susjedne vrhove vrha A, uklonimo ga iz reda te idemo provjeravat za vrhove B i C. Pošto vrh B nema susjednih vrhova uklonimo ga iz reda i prebacimo se na vrh C. On ima dva vrha D i F te ih dodamo u izlaz i u red te njega iz reda uklonimo (Slika 9).



Slika 9. Stanje izlaza i reda nakon posjeta vrhova D i F (Izrada autora)

Nakon toga posjetimo susjedni vrh vrha D, a to je G te ga dodamo u red i u izlaz i uklonimo D iz reda. Nakon toga je na redu vrh G i vidimo da ima susjedni vrh E te ga konačno dodamo u red i izlaznu. Konačan rezultat je vidljiv na slici 10.



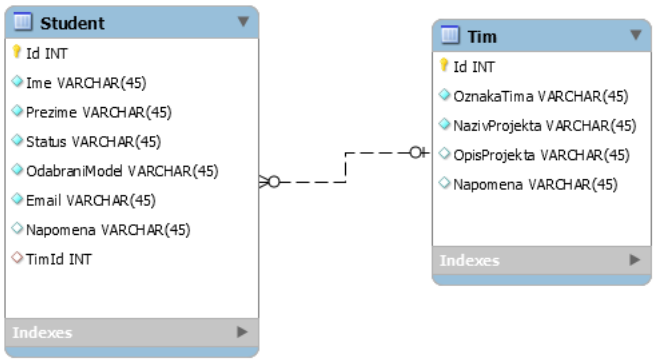
Slika 10. Konačno stanje nakon obilaska svih vrhova

Nakon što smo došli do vrha E i vidimo da on nema susjednih vrhova uklonimo ga iz reda te ga uklonimo iz reda te istu stvar ponovimo i za vrh E. . Vremenska složenost algoritma iznosi O(V+ ε) pri čemu je V broj vrhova, a ε broj bridova grafa isto kao i kod prethodnog grafa.

2.2. Pretraživanje baza podataka

U uvodnome dijelu dotakli smo se osnovnog koncepta baza podataka kao i njezinih osnovnih pojmova i karakteristika. Sljedeće pitanje je na koji način su ti podaci u bazi zapisani i kako oni korisniku bivaju prikazani. Na primjeru relacijskih baza podataka, podaci unutar baze bivaju pohranjeni unutar relacija/tablica. Logičku povezanost tih relacija prilikom kreiranja baze podataka oblikujemo pomoću dijagrama kojega nazivamo ERA.

ERA dijagram (eng. *Entity Relations Attributes*) je vrsta dijagrama pomoću koje prikazujemo i definiramo sve odnose između relacija i definiramo sve korisničke zahtjeve koje bi naša baza podataka trebala sadržavati [12]. Na slici 11 se nalazi primjer jednog jednostavnog ERA dijagrama sa dvije relacije, relacije Student – Tim. .



Slika 11.Primjer ERA dijagram (Izrada autora)

U dijagramu smo definirali vezu između tih dviju relacija (1 na više), tipove atributa, mogu li biti prazni, primarne ključeve i sl. Sljedeći korak nam je napuniti bazu sa podacima te pomoću određenih tehnika dobiti te podatke spremne za obradu/prikaz.

2.2.1. Pretraživanje strukturalnim upitnim jezikom

Strukturalni upitni jezik (eng. *Structured Query Language*), u nastavku SQL, smatra se *lingua franca* u svijetu baza podataka kako se navodi u [4]. Osim za postavljanje upita koristi se i za kreiranje i brisanje objekata u bazi, manipuliranjem podataka i sl. [4]. Prilikom korištenja njegovih naredbi potrebno je poštivati određena pravila sintakse što predstavlja ispravan odabir riječi u SQL naredbi. Osnovna naredba koja omogućuje dohvaćanje i ispis podataka iz baze podataka je SELECT uz dodatak što ustvari želimo i iz koje tablice (FROM *naziv_tablice*), još je potrebno navesti koji uvjeti moraju biti zadovoljeni (WHERE *uvjet_1, uvjet_2,…, uvjet_n*). Ispravan slijed klauzula preko kojih vršimo pretragu podataka uglavnom je sljedećeg oblika : SELECT – FROM – WHERE.

Sljedeći programski isječak [13] prikazuje sintaksu klauzule SELECT za jednostavne upite u sustavu MySQL te ćemo prema tome objasniti njegove najvažnije dijelove.

```
SELECT
    [ALL | DISTINCT | DISTINCTROW ]
    [HIGH_PRIORITY]
    [STRAIGHT_JOIN]
    [SQL_SMALL_RESULT] [SQL_BIG_RESULT] [SQL_BUFFER_RESULT]
    [SQL_NO_CACHE] [SQL_CALC_FOUND_ROWS]
    select_expr [, select_expr] ...
    [into_option]
    [FROM table_references
      [PARTITION partition_list]]
    [WHERE where_condition]
    [GROUP BY {col_name | expr | position}, ... [WITH ROLLUP]]
    [HAVING where_condition]
    [WINDOW window_name AS (window_spec)
        [, window_name AS (window_spec)] ...]
    [ORDER BY {col_name | expr | position}
      [ASC | DESC], ... [WITH ROLLUP]]
    [LIMIT {[offset,] row_count | row_count OFFSET offset}]
```

Kao što smo već prije spomenuli klauzula SELECT je obvezna i označava dohvaćanje te je nećemo pretjerano objašnjavati. Nakon nje slijede opcije koje označavaju jedinstvenost traženih izraza iz relacija (engl. *expressions*) ALL, DISTINCT, DISTINCTROW. Opcija DISTINCT se koristi ukoliko želimo izbjeći ponavljanje iste vrijednosti više puta. Ukoliko ne postoji nikakva opcija podrazumijeva se ALL. Mana opcije DISTINCT je ta da usporava rad zbog traženja duplikata unutar dohvaćenih redova i da ih pritom izbacuje [4].

Klauzulu FROM koristimo zbog specifikacije izvora podataka. Obično to biva tablica, ali i bilo koji drugi izvor podataka koji nam kao rezultat vraća redove podatka [4].

Klauzula WHERE se koristi za filtriranje redova tablice prema određenim uvjetima. Pošto u rijetkim slučajevima želimo baš sve redove iz tablice ovu klauzulu koristimo kako bismo definirali odgovarajuće uvjete kako bismo izbacili neželjene podatke [4]. Jako bitna stavka za koju se koristi klauzula WHERE jest i spajanje više relacija prema vezi PK-FK (primarni – vanjski ključ), ali o tome ćemo više govoriti u nastavku ovoga rada. Klauzula ORDER BY koristimo u slučajevima ukoliko želimo dobiti sortirane podatke. Sortirati možemo prema vrijednostima nekog stupca, nekom izrazu, prema jednom ili više stupaca, a na sam rezultat utječe redoslijed stupaca u klauzuli. Sortiranje se može provoditi na dva načina, a to su uzlazno (ASC) i silazno (DESC). Ukoliko se u pitu ne definira način sortiranja za zadanu vrijednost se uzima uzlazno (ASC) [4]. Klauzule LIMIT i OFFSET koristimo kada želimo ograničiti ili preskočiti određeni broj redova. Klauzulu LIMIT koristimo kada želimo ispisati prva tri retka izvršenog upita prema zadanim uvjetima. Nikad ne vraća više redova od definiranog argumenta, ali može vratiti manje ako sam upit vraća manji broj redova. Klauzula OFFSET omogućava da preskočimo određeni broj redaka u rezultatu dok preostale ispišemo [4].

Klauzule ORDER, LIMIT i OFFSET se u praktičnoj upotrebi na korisničkoj strani koriste za potrebe sortiranja (ORDER BY) i straničenja (LIMIT i OFFSET).

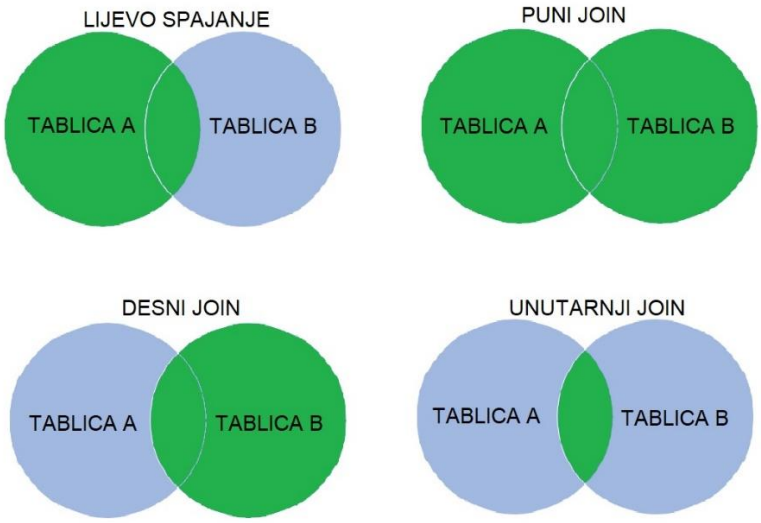
Klauzula GROUP BY SE koristi za dijeljenje podataka u logičke cjeline (grupiranje) nad kojima se obično provode određeni izračuni. Grupiranje se obično provodi prema jednome stupcu te su ti podaci obično i sortirani (korištenjem klauzule ORDER BY). Nakon što kreiramo grupirajuće kolone nad njima se provode različite operacije, a prvenstveno u te operacije svrstavamo agregirajuće funkcije [4].

Posljednja klauzula koja nam preostaje za objasniti jest klauzula HAVING. Uloga ove klauzule jest da se prilikom grupiranja korištenjem klauzule GROUP BY izbace grupe koje ne zadovoljavaju dani uvjet. Možemo zaključiti da se klauzula HAVING koristi ako se dobiveni rezultat želi profiltrirati na temelju rezultata agregirajuće funkcije[4].

Posljednje što nam je preostalo za objasniti jest pretraživanje podataka nad više tablica prema zajedničkim ključevima (veza primarni – vanjski ključ). Ranije smo spomenuli da je moguće izvršiti spajanje dviju tablica pomoću klauzule WHERE, mi ćemo se bazirati na klauzuli JOIN koja je novija. Postoje četiri vrste JOIN-a u SQL-u, a to su:

- **Unutarnji (eng. *Inner*) JOIN:** vraća sve zapise koji se podudaraju obje tablice prema primarnome i vanjskome ključu.
- **Lijevo (eng. Left) JOIN:** vraća sve zapise iz lijeve tablice i rezultate iz desne tablice koji se podudaraju prema primarnome i vanjskome ključu
- **Desni (eng. Right) JOIN:** Vraća sve zapise iz desne tablice te iz lijeve koji se podudaraju prema primarnome i vanjskome ključu.
- **Puni (eng. Full) JOIN:** Vraća sve zapise iz obje tablice koji se podudaraju i koji se ne podudaraju prema primarnome i vanjskome ključu [14]

Na slici 12 je ilustrirani prikaz funkcioniranja spajanja pomoću JOIN klauzule.



Slika 12. Grafički prikaz SQL JOIN-ova (Izrada autora prema [14])

2.2.2. Pretraživanje baze podataka u programskome jeziku C#

Za rad sa bazama podataka u programskome jeziku C# (točnije programskome okviru .NET) koristimo skup klasa kojeg nazivamo ADO.NET. Kao što smo već spomenuli, ADO.NET je skup klasa koji programerima koji rade u programskome jeziku C# nude bogat skup komponenata za stvaranje distribuiranih aplikacija sa svojstvom dijeljenjem podataka (eng. *data - sharing*). Integriran je u programski okvir .NET te pruža pristup do relacijskih, XML i aplikacijskih podatka [15].

ADO.NET odvaja pristup podacima od njegovog upravljanja u odvojene komponente koje se mogu koristiti odvojeno ili se kombinirati zajedno, te uključuju pružatelje podataka za povezivanje na bazu podataka, izvršavanje komandi (upita) i dohvaćanje rezultata. Rezultati se obrađuju direktno, smještajući se u ADO.NET *DataSet-ove* kako bi korisnik mogao kombinirati podatke sa više izvora ili prosljeđujući ih između slojeva arhitekture [16]. Skupovi podataka i pružatelji podataka (eng. *Data Providers*) su dvije glavne komponente ADO.NET-a za pristupanje i manipuliranje podataka u .NET programskome okviru.

Pružatelji podatka u programskome okviru .NET koriste povezani način rada za povezivanje na bazu podataka, izvršavanje naredbi i pribavljanje rezultata. Lagani su za rad i kreiraju minimalni sloj između izvora podataka i programskoga koda, pri tome povećavajući performanse aplikacije bez smanjenja kvalitete njene funkcionalnosti [17]. U tablici 2 su prikazane i opisane bazične klase koje omogućuju prethodno opisane radnje koje pružatelji podataka omogućuju.

Tablica 2. Osnovne klase pružatelja podataka (Izrada autora prema [17] [18, 127. str])

Naziv	**Opis**
Connection	Uspostavlja vezu sa specifičnim izvorom podataka. Nasljeđuje **DbConnection** klasu.
Command	Izvršava naredbu (upit) nad izvorom podataka. Nasljeđuje **DbCommand** klasu.
DataReader	Čita podatke iz izvora podataka redak po redak bez opcije vraćanja na prethodni redak. Nasljeđuje **DbDataReader** klasu.
DataAdapter	Puni DataSet s podacima i brine o ažuriranju tih podataka. Nasljeđuje **DbDataAdapter** klasu.

U nastavku ćemo prikazati elemente iz tablice 2 na programskome primjeru za SQL baze podataka napravljen prema [18].

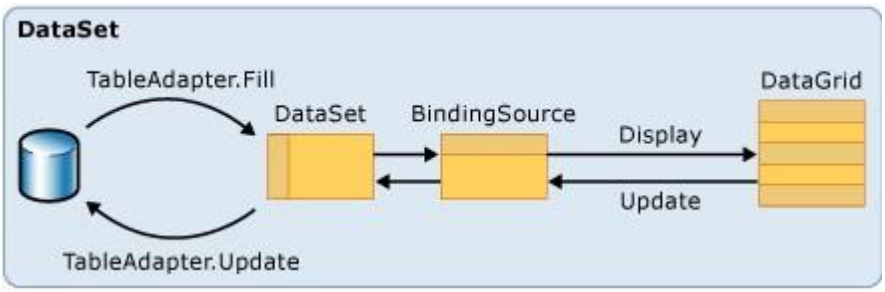
```
public sealed class Baza
    {
        private static Baza instance;
        private string connectionString;
        private SqlConnection connection;

        public static Baza Instance
        {
            get
            {
                if (instance == null)
                {
                    instance = new Baza();
                }
                return instance;
            }
        }
        public string ConnectionString
        {
            get { return connectionString; }
            private set { connectionString = value; }
        }
        public SqlConnection Connection
        {
            get { return connection; }
            private set { connection = value; }
        }

        private Baza()
        {
            ConnectionString = @"Data Source = ..\..\Baza\EvidencijaStudenata.db3";
            Connection = new SqlConnection(ConnectionString);
            Connection.Open();
        }

        public SqlDataReader DohvatiDataReader(string sqlUpit)
        {
            SqlCommand command = new SqlCommand(sqlUpit, Connection);
            return command.ExecuteReader();
        }

        public object DohvatiVrijednost(string sqlUpit)
        {
            SqlCommand command = new SqlCommand(sqlUpit, Connection);
            return command.ExecuteScalar();
        }

        public int IzvrsiUpit(string sqlUpit)
        {
            SqlCommand command = new SqlCommand(sqlUpit, Connection);
            return command.ExecuteNonQuery();
        }
    }
```

Imamo klasu **Baza** koja je implementirana pomoću uzorka dizajna **Singleton** kako bismo tijekom izvođenja aplikacije imali cijelo vrijeme jednu instancu. Bitno je izdvojiti atribut **connectionString** koji sadrži osnovne podatke za spajanje na bazu podatka, atribut **connection** koji je tipa **SqlConnection** i ekvivalentno je **Connection**-u iz tablice 2. Zatim

imamo metode **DohvatiDataReader**, **DohvatiVrijednost** i **IzvrsiUpit** koje kao parametar primaju naredbu (SQL upit) koje će izvršiti i koriste instancu **SqlCommand**-a za izvršavanje upita. Metoda **DohvatiDataReader** vraća vrijednost proslijeđenog upita tipa **SqlDataReader**, dok metoda **DohvatiVrijednost** vraća vrijednost tipa **object** te služi za dohvaćanje skalarnih (MIN, MAX, AVG) vrijednosti proslijeđenog upita. Posljednja metoda koja nam je preostala za opisati jest **IzvrsiUpit**. Ona služi za izvršavanje upita koji u sebi sadrže INSERT, UPDATE, DELETE. Prethodno naveden kod i opisane metode nam služe kao polazište za rad sa pružateljima podataka.

Druga komponenta koju ćemo opisati jesu DataSet-ovi. DataSet objekti predstavljaju nepovezani način rada za rad sa podacima sa ADO.NET. Predstavljaju offline pristup podacima te njeni objekti predstavljaju memorijske reprezentacije dohvaćenih podataka. Mogu biti korišteni sa više i različitih izvora podataka, XML podacima ili za upravljanje lokalnim podacima aplikacije. Predstavljaju kompletni skup podataka uključujući tablice, ograničenja i veze između tablica [18, 148.str][19].

Na slici 13 je prikazana shema ažuriranja i dohvaćanja podataka pomoću DataSet-a.



Slika 13. Rad sa podacima pomoću DataSet-a (Preuzeto sa [20])

Sa lijeve strane imamo bazu podataka koja sadrži podatke, dok je sa desne strane korisnik koji pregledava podatke na DataGrid kontroli. BindingSource DataGridu predstavlja izvor podataka preko koje je povezana sa podacima. Svaku korisnikovu promjenu propagira na sljedeći sloj prema bazi podataka, dok svaku promjenu baze podataka propagira prema korisniku. Podatke koje prikazuje korisniku dobiva iz DataSet kontrole, koji kako smo već i ranije spominjali predstavljaju memorijsku reprezentaciju dohvaćenih podataka. DataSet podaci pune se preko TableAdapter klase koja služi za razmjenu podataka između baze podataka i podatkovnih objekata u aplikaciji. Sadrži metode za čitanje i filtriranje podataka iz baze podataka, kao i zapisivanje podataka u suprotnome smjeru. Kada korisnik potraži podatke, BindingSource potraži podatke iz DataSet-a kojeg je TableAdapter prethodno

napunio. Kada korisnik odluči promijeniti podatke, oni će se putem BindingSource-a mijenjati samo lokalno, točnije u DataSet-u. Podaci se promjene u bazi podataka tek kada okine naredba „Save“[19][20].

3. Jezik integriranog upita (LINQ)

U ovome poglavlju ćemo opisati i detaljnije pojasniti skup tehnologija programskog jezika C# za pretraživanje i dobivanje podataka iz različitih izvora i formata, a to je ***jezik integriranog upita*** (eng. *Language Integrated Query*) u nastavku LINQ.

LINQ možemo opisati kao skup tehnologija integriranih upitnih operatora u programskome jeziku C#. Prednost ove vrste tehnologije nad tradicionalnim pisanjem upita je u tome da su tradicionalni upiti uglavnom pisani u tekstualnom obliku bez mogućnosti provjere tipova podataka tijekom izvršavanja programa ili mogućnosti automatske nadopune (eng. *IntelliSense*). Programeru se olakšava pisanje jedinstvenih upita nad različitim izvorima poput kolekcije objekata, SQL baza podataka, ADO.NET skupova podataka, XML dokumenata i sl.[21].

U nastavku ćemo detaljnije obraditi LINQ te njegove naredbe kao i oblike.

3.1. LINQ općenito

Spomenuli smo da je LINQ skup tehnologija koji nam omogućuju da pomoću upitnih operatora vrši dobavljanje podataka sa različitih izvora podataka kao i formata. Sa strane programera koji pišu upite najznačajniji dio LINQ-a je upitni izraz pisan u deklaracijskoj upitnoj sintaksi (eng. *query syntax*) ili funkcijskoj sintaksi (eng. *method syntax*) koje rasterećuju programera da uči novi upitni jezik za svaku tehnologiju zasebno [21] [22].

Prema [22] vidimo da je specifično da se svi LINQ upiti sastoje od tri jedinstvenih akcija, a to su:

1. Dobivanje izvora podataka
2. Kreiranje upita
3. Izvršavanje upita

U LINQ-u prvi korak je specificiranje izvora podataka. U LINQ upitu, **FROM** klauzula dolazi prva kako bi odredila izvor podataka i raspon varijable. Varijabla raspona je identična slijednoj varijabli u **foreach** petlji osim što se u izrazu upita ne događa stvarna iteracija [23]. Kako bi se konačan rezultat mogao iterirati, izvor podataka mora podržavati klasu **IEnumerable<T>** ili iz njega izvedeno sučelje **IQueryable<T>** koje nazivamo upitnim tipovima. Upitni tipovi ne

zahtijevaju nikakvu posebnu modifikaciju ili pripremu za prikaz kao LINQ izraz. Ukoliko izvor podataka u memoriji nije upitnog tipa, LINQ pružatelj usluga ga može lako pretvoriti da bude. Primjer takve prilagodbe mogu biti upitne tipovi za učitavanje XML datoteka, skupova podatka i sl. [22].

Akcija koja slijedi nakon dobivanja izvora podataka je kreiranje samog upita. Uloga upita je da specificira koje će informacije korisnik dobiti iz ciljanog izvora podataka. Specificira na koji način će podaci biti filtrirani, sortirani, grupirani prije nego li ih vratimo. Upit pohranjujemo u upitnu varijablu i inicijaliziramo je sa upitnim izrazom. Bitna stavka za LINQ je ta da upitna varijabla sama po sebi ne poduzima nikakvu akciju i ne vraća nikakve podatke. Samo pohranjuje informacije potrebne za stvaranje rezultata kada se upit izvrši u nekom kasnijem trenutku [22].

Izvršavanje LINQ-a se može odvijati na dva načina, a to su odgođeno (eng. *deffered*) i neposredno (eng. *Forcing immediate*). Pošto smo spomenuli da se u upitnoj varijabli pohranjuju upitne naredbe, u odgođenom izvršavanju se stvarno izvršavanje upita odvija tijekom njegovog obilaska pomoću programskih petlji (for, foreach, while i sl.). Pošto upitna varijabla ne sadrži rezultate, možemo izvršavati upit kada god želimo, dobivajući pri tom uvijek „svježe“ podatke [22]. Pod neposrednim izvršavanjem podrazumijevamo upite koji pohranjuju rezultate u varijable prije nego ih obilazimo petljama. Primjeri upita koji se neposredno izvršavaju kao rezultat uglavnom vraćaju jedan rezultat pomoću agregirajućih funkcija (Min, Max, Count, First) ili se rezultat upita „kešira“ pomoću metoda **ToList** ili **ToArray** u objekt kolekcije koji se zatim naknadno mogu obilaziti pomoću programskih petlji, dajući pri tome uvijek isti rezultat, kakav je bio u trenutku „keširanja“ [22].

Osim dohvata podataka, još jedna mogućnost LINQ-a je i pretvaranje jednog oblika podataka u drugi. Koristeći LINQ upite moguće je odraditi radnje kao što su kreiranje novog izlaznog polja kombinirajući više ulaznih izvora pri tome koristeći sve atribute iz svih izvora podataka ili kombinirajući po nekoliko atributa iz svakog izvora kao i kreiranje izlaza različitog formata u odnosu na ulazni format (pretvaranje ulaznih podataka dobivenih iz baze u tekstualne ili u XML datoteke) [24].

3.2. Upitna i funkcijska sintaksa

Spomenuli smo već da se LINQ upitni izrazi mogu zapisati na dva načina, tj. dvije sintakse, a to su upitna i funkcijska. Upitna sintaksa slična je samome SQL-u te je sama šablona LINQ sljedećeg oblika [25]:

```
from <varijabla raspona> in <IEnumerable<T> or IQueryable<T> Kolekcija>
<Standardni Upitni Operator> <Lambda izraz> <select ili groupBy operator>
<format rezultata>
```

Upitni izraz započinje sa klauzulom **from** te se nastavlja sa varijablom raspona koja označava jedan zapis (jedan objekt) iz izvora podataka kojeg predstavlja **IEnumerable** ili **IQueryable** kolekcija. Taj dio upita možemo objasniti kao „za svaki objekt unutar kolekcije (*foreach (var zapis in kolekcija)*)“. Nakon definiranja izvora podatka dolaze *Standardni upitni operatori* koji definiraju uvjete pomoću kojih provodimo upitne mogućnosti poput filtriranja, sortiranja, grupiranja i sl. Nakon standardnog upitnog operatora dolaze upitni izrazi koji su uglavnom definirani pomoću lambda izraza. Na kraju dolaze select ili group by klauzula pomoću kojih definiramo objekt ili pojedinačne atribute koje želimo za danju obradu [25].

Funkcijska sintaksa se za razliku od upitne sintakse zapisuju u obliku standardno-upitnih operatora proširenih metoda (eng. *Standard Query Operator Extension Methods*). Proširene metode standardnih upitnih operatora su zapravo statičke metode koje se „nadodaju“ na **IEnumerable** i **IQueryable** izvore podataka i koje zapravo izgledaju poput standardnih upitnih operatora kao što su Where, OrderBy, Select, SelectMany. Proširene metode u većini slučajeva kao argument primaju delegatsku funkciju koja je definirana na lambda izrazu. Kao ulazni parametar delegatska metoda prima objekt tipa T (isti tip kojega je i varijabla u koju se pohranjuje) te vraća vrijednost tipa **boolean** [26]. U nastavku se nalazi isječak programskog koda napisanog funkcijskom sintaksom.

```
List<Student> studenti = izvorPodataka.Where(student =>
student.OdabraniModel.Equals("A")).OrderBy(student => student.Status).ToList();
```

Funkcionalnost prethodnog programskog isječka je da dohvati iz izvora podatka sve studente čiji je odabrani model jednak vrijednosti A, sortiranih prema njihovome statusu te ih pohranjuje u listu. Kao što smo već ranije spomenuli, LINQ pisan funkcijskom sintaksom započinje sa izvorom podataka, što je u našem slučaju **izvorPodataka**. Zatim slijede metode **Where** i **OrderBy** koje predstavljaju Standardne upitne operatore koje kao parametar primaju delegatsku metodu temeljenu na lambda izrazu. Metoda **Where** kao ulazni parametar prima objekt **student** koji je tipa **Student**, te vraća **boolean** izraz koji je ovisan o rezultatu jednakosti modela praćenja studenta sa modelom **A** na temelju kojega filtrira zapise iz izvora podataka. Nakon filtriranja metodom **Where** slijedi metoda **OrderBy** koja služi za sortiranje dobivenih

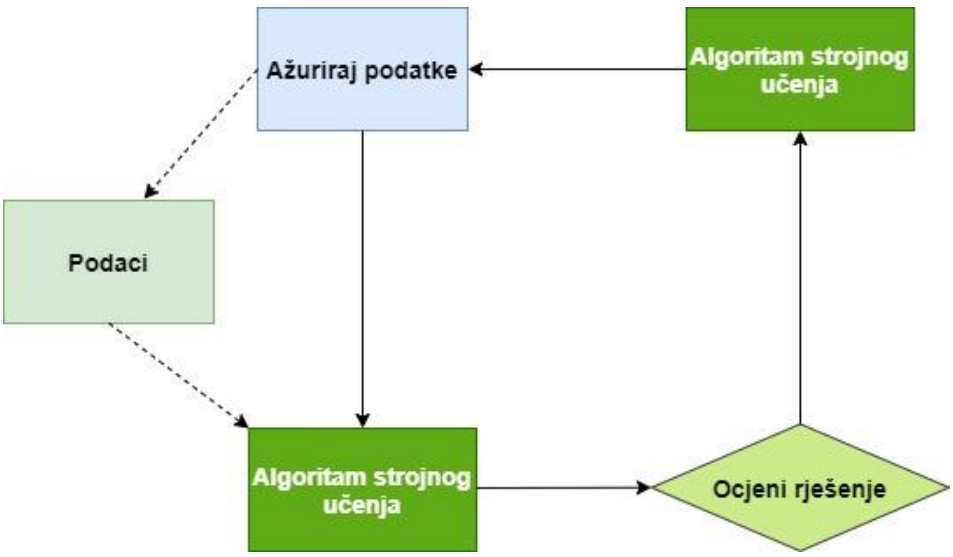
podataka prema odabranom/im atributu/ima. Ona kao parametar prima delegacijsku funkciju koja kao ulazni parametar lambda izraza prima objekt **student** koji je tipa **Student**, te kao izlaznu vrijednost prima atribut prema kojemu vrši sortiranje. Na kraju imamo metodu **ToList** koja ustvari čini ovo izvršavanje upita čini neposrednim tako da dobivene rezultate pohrani u kolekciju Liste, te nam je čini dostupnom za izvršavanje daljnjih obrada nad njome.

4. Strojno učenje

U ovome poglavlju ćemo opisati i definirati što je to zapravo strojno učenje (eng. *machine learning*). Strojno učenje možemo definirati kao znanost (i umjetnost) programiranja računala kako bi računalo moglo učiti iz podataka [27]. Kao primjer ove tvrdnje možemo uzeti primjer iz [27] za prepoznavanje mailova dobrog sadržaja u odnosu na mailove ometajućeg (engl. *spam*) sadržaja temeljenog na strojnome učenju. Zadatak takvog sustava je prepoznavanje i označivanje zastavicom ometajućih mailova, a za učenje aplikacija koristi skup primjera koji se sastoji od email-ova ometajućeg sadržaja koji su označeni za zastavicom i email-ova sa dobrim sadržajem. Taj skup nazivamo *skupom za treniranjem* (engl. *training set*). Svaki pokušaj treniranja možemo nazvati *treniranim uzorkom* (engl. *training sample*). Cilj našeg sustava je označavanje novih mailova sa spam zastavicom na temelju iskustva iz *treniranih podatka* (engl. *training data*) i na temelju dobivenih mjerenja učinaka (engl. *accuray*) omogućuje kvalitetno obavljanje traženih operacija.

Strojno učenje je podskup umjetne inteligencije što omogućuje računalu da samostalno uči iz podataka i događajima (iskustvom) iz prošlosti, poznatim kao *trenirani podaci* (engl. *training data*), algoritme strojnog učenja grade matematički model koji nam pomaže u izradi predikcija ili odluka bez prevelikog programiranja. Povezuje računalnu znanost i statistiku zajedno za izradu predikcijskih modela [28].

Na slici 14 je prikazan dijagram koji prikazuje proces strojnog učenja.



Slika 14. Dijagram procesa strojnog učenja (Izrada autora prema [27])

Element od kojeg polazi proces strojnog učenja jesu podaci. Sustav strojnog učenja uči iz povijesnih podatka gradeći predikcijske modele, te dobivajući nove podatke predviđa za njih izlaz. Kvaliteta predviđenog izlaza ovisi o samoj količini podataka jer ogromna količina podatka pomaže u izgradnji boljeg modela koji daje preciznije i točnije rezultate. Proces strojnog učenja zamjenjuje kompleksnu poslovnu logiku sa generičkim algoritmima kojima se hrpa programskoga koda zamjenjuje sa algoritmom koji gradi logiku sa podacima i na taj način predviđa izlaznu vrijednost [28].

Potreba za strojnim učenjem se povećava svakim danom sve više iz razloga što ima kapacitet odraditi što je prekomplicirano za direktnu implementaciju. Algoritme strojnog učenja možemo trenirati pružajući im veliku količinu podataka dopuštajući im da ih istražuju, kreiraju modele i predviđaju potrebni izlaz automatski. Algoritmi strojnog učenja mogu pomoći ljudima da vide što su naučili te primjenom tehnika strojnog učenja kako bi ga mogli lako razumjeti koristeći vlastite slučajeve [27][28]. Na kraju vrijedi navesti da je strojno učenje pogodno za rješavanje problema kao što su problemi čija postojeća rješenja zahtijevaju već ranije spomenutu zamjenu kompleksne poslovne logike sa jednostavnijim kodom genetskog algoritma koji pruža bolje performanse, kompleksnih problema za koja nema dobrih rješenja koristeći tradicionalni pristup tako da tehnike strojnog učenja uvijek pronađu rješenje, promjenjivog okruženja tako da se prilagodi novim podacima, dobivanje uvida o složenim problemima i velikim količinama podataka, otkrivanje novih uzoraka rovanjem po podacima, izdvajanjem korisnih informacija iz podataka [27].

Za kraj ćemo još navesti da se strojno učenje može klasificirati u tri grupe, a to su:

- **Nadzirano učenje** (engl. *Supervised learning*)
- **Nenadzirano učenje** (engl. *Unsupervised learning*)
- **Pojačano učenje** (engl. *Reinforcement learning*) [28]

U nastavku ćemo se detaljnije baviti sa nadziranim i nenadziranim učenjem.

4.1. Nadzirano učenje

Nadzirano učenje vrsta je strojnog učenja u kojoj rezultat algoritma uvelike ovisi o podacima koje koristi za njegovo treniranje. Instance koje sačinjavaju skup podataka koji se koristi za treniranje nazivamo oznakama (engl. *labels*) [27], što su zapravo podaci koji već imaju vrijednost koja bi trebala predstavljati izlaznu vrijednost. Cilj ovog sustava je da izradi model koristeći označene podatke za razumijevanje skupa podatka, te da uči o svakome podatku i da jednom kada završi sa treniranjem i procesuiranjem, krenemo sa testiranjem modela tako da mu damo uzorak za koji želimo dobiti izlazni rezultat i tako provjerimo da li imamo točno predviđanje ili nemamo [28].

Proces provođenja nadziranog učenja možemo definirati sa sljedećim koracima:

1. Odabrati vrstu skupa podataka za treniranje
2. Sakupiti trenirani skup podataka sa označenim podacima
3. Razdvojiti skup podataka za treniranje u skup podataka za treniranje, skup podatka za testiranje i skup podatka za validaciju
4. Odrediti ulazne elemente skupa podataka za treniranje, koji bi trebao imati dovoljno znanja kako bi model mogao precizno predvidjeti izlazni rezultat
5. Odrediti prigodan algoritam za model
6. Izvršiti algoritam na skupu podataka na treniranome skupu podataka
7. Procijeniti točnost modela pružajući testni skup. Ukoliko model daje točnu izlaznu vrijednost, to znači da je naš model točan [29]

Ova vrsta učenja može se podijeliti u dvije vrste algoritama za rješavanje problema, a to su:

1. **Klasifikacija**
2. **Regresija** [28]

Tehnike koje ulaze u kategorije nadziranog učenja, a o kojima će više biti u nastavku su svakako treniranje neuronskih mreža i stabala odlučivanja. Obje navedene tehnike jako ovise o ulaznim informacijama kako bi dale što kvalitetniji izlaz. Svaka od prethodne dvije kategorije ima algoritme koji su specifični za njegove slučajeve, ali postoje i algoritmi koji se mogu koristiti za obje kategorije [30].

4.1.1. Regresija

Prvi tip nadziranog učenja kojeg ćemo obraditi jest regresija. Regresija je ustvari statistička metoda koje oblikuje vezu između zavisne (ciljnih) i nezavisnih (predikcijskih) varijabli, te kako se vrijednost zavisnih varijabli mijenja u odnosu na vrijednosti jedne ili više nezavisnih varijabli. Najčešći slučajevi u kojima se regresija koristi jesu:

- Predikcija (engl. *Prediction*)
- Vremensko modeliranje (engl. Timeseries)
- Forecasting i sl.[31]

Tipični zadaci kojima se regresija bavi jesu ustvari predviđanja brojčanih vrijednosti (rezultata) kao na primjer cijene automobila prema nezavisnim varijablama kao što su marka, vrsta karoserije, broj vrata, kilometraža i sl. Za treniranje modela sustav će koristiti skup podataka koji sadrže mnogo primjera sa zavisnim i nezavisnim vrijednostima [29].

4.1.1.1. Linearna regresija

Linearna regresija vrsta je regresije koja se koristi za predikcijsku analizu. Koristi se za predikcije kontinuiranih/stvarnih ili numeričkih varijabli kao što su godine, prodajne i kupovne cijene proizvoda i sl. Prikazuje linearnu vezu između zavisne varijable (Y) i jedne ili više nezavisnih varijabli (X) što ustvari znači da utvrđuje kako se vrijednosti ovisnih varijabli mijenjaju prema vrijednostima neovisnih varijabli.[32].

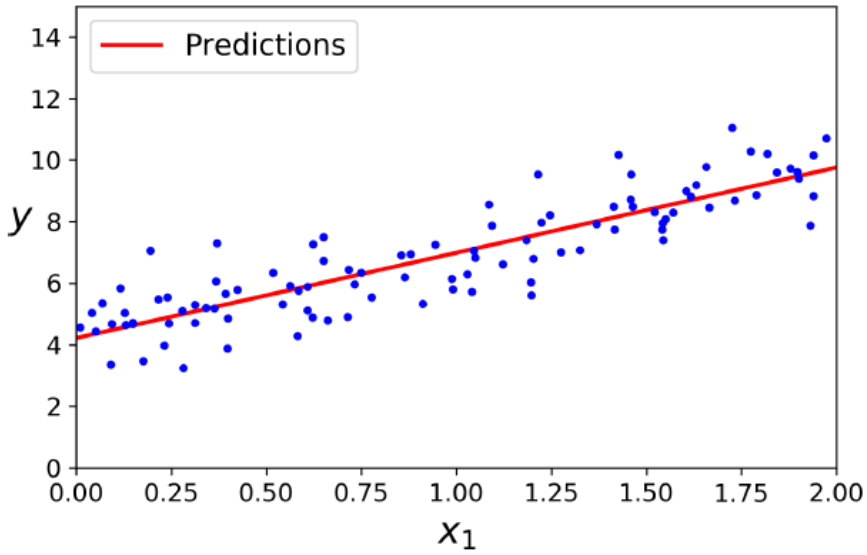
Formula pomoću koje možemo zapisati linearnu regresiju na sljedeći način:

$$\bar{y} = a_0 + a_1 x_1 + a_2 x_2 + \cdots + a_n x_n$$

Pri čemu su :

- $\bar{y}$ - rezultat predikcije
- a_j - koeficijent linearne regresije
- n – broj nezavisnih varijabli
- x_i – nezavisna varijabla [27]

Grafički prethodnu formulu za predikciju linearnom regresije možemo prikazati na sljedeći način (vidi sliku 15)



Slika 15. Predikcija modela linearnom regresijom [27]

Točkice koje su vidljive na grafu su predstavljaju zapravo podatke, dok crvena linija predstavlja liniju regresije, koja prikazuje odnos između zavisnih i nezavisnih varijabli. Postoje dvije vrste linearnih veza koje opisuju vezu između zavisnih i nezavisnih varijabli, a to su pozitivna i negativna. Pozitivna regresijska linija se pojavljuje ukoliko se zavisna varijabla povećava na Y-osi, a nezavisna po X-osi. Sa druge strane negativna regresijska linija se pojavljuje ukoliko se zavisna varijabla smanjuje po Y-osi, dok se nezavisna varijabla povećava po X-osi [32].

Glavni zadatak tijekom rada sa linearnog regresijom je pronaći najbolju liniju razlike koja označava grešku između predviđene vrijednosti i aktualne vrijednosti što se naziva minimizacija. Kako bismo mogli izračunati najbolju liniju linearne regresije za koeficijente a_0 i a_1 koristimo izraz koji se naziva funkcija troškova (engl. *cost function*). Neke od funkcionalnosti funkcije troškova jesu da se koristi za procjenu vrijednosti koeficijenata za pronalazak najbolje prikladne linije, mjeri izvedbu modela linearne regresije, te se koristi za pronalazak točnosti funkcije mapiranja koja preslikava ulaznu na izlaznu varijablu. Ova funkcija mapiranja također je poznata i kao funkcija hipoteze (engl. *hypothesis function*) [32].

Za dobivanje mjerenja kojima dobivamo informaciju u kojoj mjeri model zadovoljava podatke koje koristimo kod linearne regresije jest Korijenska srednja kvadratna pogreška (engl. *Root Mean Square Error* , u nastavku RMSE). RMSE je općenito preferirana mjera izvedbe za zadatke regresije, no u slučajevima gdje imamo više vanjskih ograničenja koristimo *Srednju kvadratnu pogrešku* (engl. *Mean Absolute Error,* u nastavku MSE) [27].

Srednju apsolutnu pogrešku možemo definirati pomoću sljedeće formule [32]:

$$MSE = 1\frac{1}{N}\sum_{i=1}^{n}(y_i - (a_1x_i + a_0))^2$$

Pri čemu su:

- N- ukupan broj uzoraka
- y_i- aktualna vrijednost
- $(a_1x_i + a_0)$- vrijednost predikcije

Uloga MSE-a je izračunavanje prosjeka kvadratne greške između predviđene i aktualne vrijednosti. Razliku između predviđene i aktualne vrijednosti nazivamo *ostatkom* (engl. *residual*), te ovisno o udaljenosti između promatranih točaka od regresijske linije vrijednost ostatka može poprimiti visoku ili nisku vrijednost i takvu vrijednost poprima vrijednost funkcije troška [32].

Kako bismo minimizirali MSE koristit ćemo generički algoritam optimizacije sposobnog za pronalazak optimalnog rješenja za široki raspon problema kojeg nazivamo *stupnjevitim spuštanjem* (engl. *Gradient Descent*). Glavna zamisao stupnjevitog spuštanja je podešavanje parametara u iteracijama kako bi se minimizirala funkcija troškova. To se vrši slučajnim odabirom vrijednosti koeficijenata, a zatim se vrši iterativno ažuriranje vrijednosti koeficjenata kako bi se postigla minimizacija funkcije minimalnih troškova [27] [32].

Dobrobit uklapanja određuje kako se linija regresije uklapa skupu opažanja. Proces pronalaska najboljeg modela iz različitih modela naziva se optimizacijom. Ona se postiže *R-kvadratnom*(engl. *R-squared*) metodom. R-kvadratnom metodom mjeri se snaga odnosa između ovisnih i neovisnih varijabli u rasponu od 0-100%. Njome se određuje kvaliteta regresijske linije, te visoka vrijednost R-kvadratne metode određuje manju razliku između

predviđenih vrijednosti, kao i stvarnih vrijednosti te stoga predstavlja dobar model. Još se naziva i koeficijentom determinacije ili koeficijentom višestrukog određivanja za višestruku regresiju. Izračunava se prema sljedećoj formuli [33]:

$$R - kvadrat = \frac{SS_{regresije}}{SS_{ukupno}}$$

Pri čemu su:

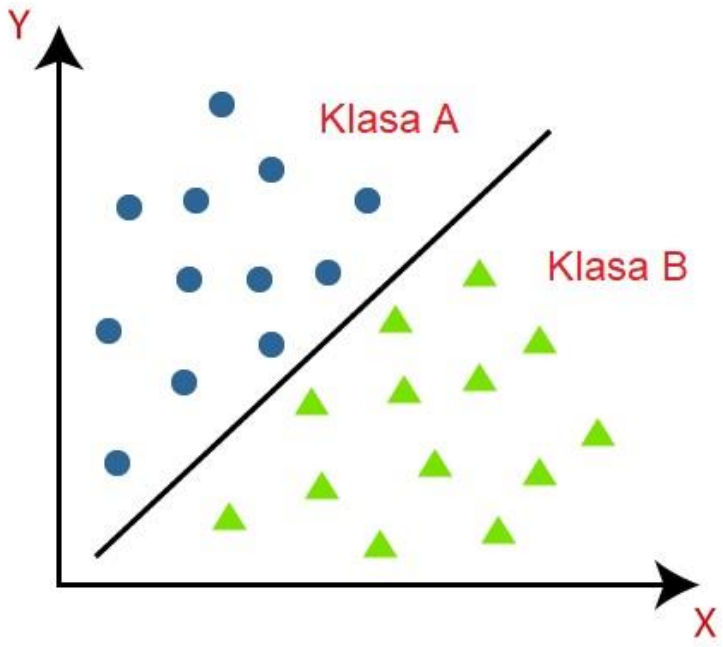
- $SS_{regresije}$ – zbroj svih kvadrata regresije
- SS_{ukupno} – ukupna suma kvadrata

Varijabla $SS_{regresije}$ prikazuje ustvari kako model regresije predstavlja podatke koje koristimo za modeliranje, dok varijabla SS_{ukupno} predstavlja promatrane podatke koji se koriste za modeliranje regresije [33].

Za kraj vrijedi napomenuti da postoje dvije vrste linearne regresije, a to su **jednostavna linearna regresija** (engl. *Simple Linear Regression*) u kojoj se predviđa vrijednost brojčane vrijednosti zavisne varijable pomoću jedne nezavisne varijable, te **višestruka linearna regresija** (engl. *Multiple Linear Regression*) u kojoj se nalazi više od jedne nezavisne varijable za predviđanje vrijednosti brojčane vrijednosti zavisne varijable [32].

4.1.2. Klasifikacija

Drugi tip nadziranog učenja kojeg ćemo obraditi jest klasifikacija. Algoritmi klasifikacije tehnike su nadzornog učenja koje se koriste za identifikaciju novih podataka na temelju skupa treniranih podataka. Učenje se odvija pomoću skupa podataka sa određenim skupinama te ih klasificira u određeni broj klasa i/ili grupa. Za razliku od regresije koja kao rezultat vraća brojčanu vrijednost, izlazna varijabla u klasifikaciji jest kategorija. Glavni cilj algoritama klasifikacije jest identificiranje kategorije za dani skup podataka sa oznakama sa prigodnim vrijednostima ulaznih i ciljanih izlaznih vrijednosti [34]. Na slici 16 se nalazi dijagram koji detaljnije prikazuje klasifikaciju sa dvjema klasama.



Slika 16. Graf klasifikacije (Izrada autora prema [34])

Na slici 16 imamo dvije klase, A i B. Svaka klasa ima neke elemente koji su im zajednički kao i elemente koji im nisu zajednički. Na algoritmu je da odluči prema ulaznim podacima koja kategorija najbolje odgovara prema ulaznim podacima.

Algoritam koji implementira klasifikaciju na skupu podataka naziva se klasifikator. Prema broju mogućih izlaznih kategorija poznajemo dvije vrste klasifikacija, **binarnu** i **višeklasnu**. Kod binarnih klasifikacija moguća su jednino dva izlazna stanja, dok su kod višeklasnih moguća više od dva [34].

Kod klasifikacije podrazumijevamo dvije vrste algoritama na koji način uče, a to su algoritmi **lijenog** (eng. *lazy*) učenja i algoritmi **željnog** (engl. *eager*) učenja. Kod algoritama lijenog učenja prvo se pohrane podaci o treniranome skupu podataka i pričeka dok ne dobije testni skup podataka. Klasifikacija se provodi na temelju najpovezanijih podataka pohranjenih u treniranome skupu podataka. Zahtjeva manje vremena za treniranje, dok zahtjeva više za predikciju. Najpoznatiji algoritam iz ove skupine je svakako KNN algoritam. Sa druge strane kod algoritma željnog učenja klasifikacija se odvija na temelju treniranog skupa podataka prije nego li dobi testni skup podataka. Zahtjeva manje vremena za treniranje, a više za predikciju. Poznatiji algoritmi iz te skupine jesu Stabla odlučivanja, Naive Bayes i sl. [34].

U nastavku ćemo opisati neke od najpoznatijih predstavnika algoritma klasifikacije.

4.1.2.1. K-najbližih susjeda

Prvi algoritam iz klasifikacije koji ćemo obraditi jest algoritam **K-najbližih susjeda** (eng. *K-Nearest Neighbour*). Jedan je od najjednostavnijih algoritama strojnog učenja baziranog na nadziranome učenju. Pretpostavlja sličnost između novog slučaja/podatka i dostupnih slučajeva te ih svrstava u kategoriju koja je najsličnija dostupnim kategorijama. Pohranjuje sve dostupne podatke i klasificira novu podatkovnu točku na temelju sličnosti. Znači da kad se pojave novi podaci, pomoću KNN algoritma novi podaci se mogu lako klasificirati u odgovarajuću kategoriju. Mogu se koristiti za regresiju i klasifikaciju, ali najčešće za probleme klasifikacije. U fazi treniranja pohranjuje skup podataka te kada dobi novi podatak klasificira ga u kategoriju koja ima najviše sličnosti sa novim podatkom [35].

Postupak pomoću kojeg se provodi algoritam opisan je sljedećim koracima:

1. Odaberemo vrijednost *K* koja označava broj kategorija/susjeda
2. Izračunamo Euklidovu udaljenost K broja susjeda pomoću formule:

$$\sqrt{(X_2 - X_1)^2 + (Y_2 - Y_1)^2}$$

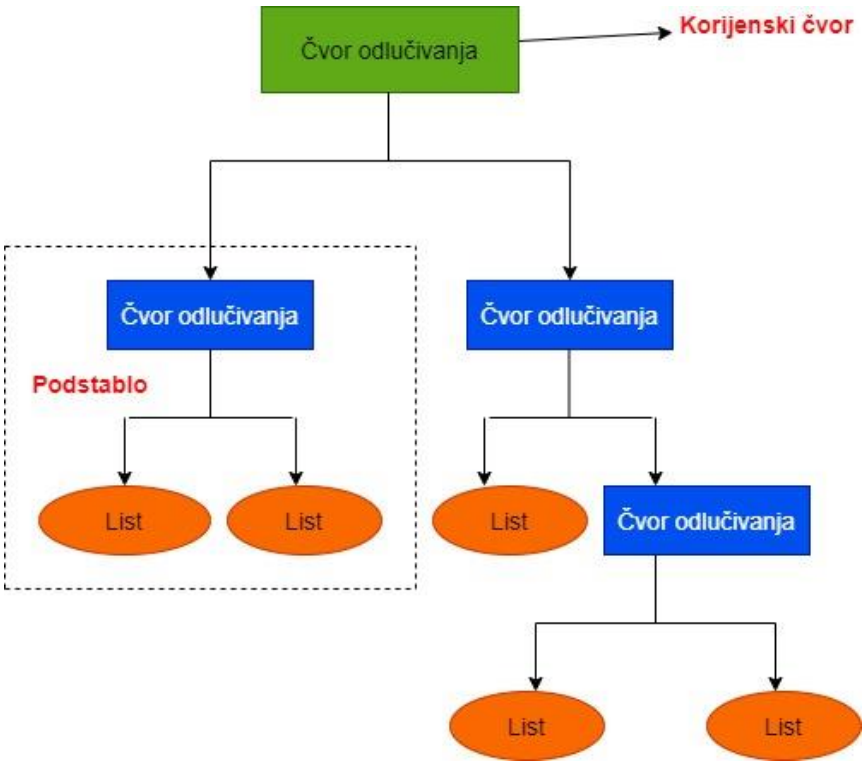
3. Odaberi K najbližih susjeda prema izračunatoj Euklidovoj udaljenosti
4. Među dobivenih k susjeda, prebrojimo broj podatkovnih bodova u svakoj kategoriji
5. Dodijelimo nove podatkovne točke onoj kategoriji za koju je broj susjeda maksimalan
6. Imamo spreman model

Neke od prednosti ovog algoritma svakako je njegova lagana implementacija, kao i veća efikasnost ako je trenirani skup podataka veći, dok su mane ovog algoritma kompleksnost broja K ako je njegova vrijednost veća, te visoki troškovi izračuna zbog udaljenosti između podatkovnih točaka za sve trenirane uzorke [35].

4.1.2.2. Stablo odlučivanja

Algoritam stabla odlučivanja algoritam je strojnog učenja koji se može koristiti za zadatke klasifikacije i regresije. Vrlo je moćan algoritam koji je sposoban da se uklopi u složene skupove podataka [27]. Klasifikator je strukturiran u obliku stabla gdje unutarnji čvorovi predstavljaju značajke skupa podataka, grane predstavljaju pravila odluke, a svaki čvor lista predstavlja rezultat. Čvorovi odluka koriste se za donošenje bilo koje odluke i imaju više grana, dok su listovi izlaz tih odluka i ne sadrže daljnje grane. Odluke ili test provode se na temelju značajki datog skupa podataka. To je grafički prikaz za dobivanje svih mogućih rješenja problema/odluke na temelju zadanih uvjeta. Naziva se stablom odlučivanja jer slično stablu

započinje korijenskim čvorom koji se širi na daljnje grane i gradi strukturu nalik stablu [36]. Na slici 17 je prikaz stabla odlučivanja sa prigodnim čvorovima.



Slika 17. Primjer stabla odlučivanja sa čvorovima

Prilikom gradnje stabla koristimo algoritam *stabla klasifikacije i regresije* (eng. *Classification and Regression Tree algorithm*, u nastavku CART), pri čemu se u stablu odlučivanju čvorovi baziraju na pitanjima sa Da/Ne odgovorima i tako razdvaja stablo na podstabla. Stabla odlučivanja često oponašaju ljudsko razmišljanje kod donošenja odluka, pa su prema tome laka za razumijevanje. Na taj način je logika lako razumljiva jer je prikazana u obliku strukture stabla. Sam algoritam se provodi na način da se za početak stabla odabere korijenski čvor koji sadrži kompletni skup podataka. Za pronalazak najbolje odgovarajućeg atributa skupa podataka koristimo tehniku poznatu kao *mjera odabira atributa* (eng. *Attribute Selection Measure,* u nastavku ASM). Tim mjerenjem lako odabiremo najbolji atribut za čvorove stabla, a dvije popularne tehnike ASM-a, o kojima će više riječi biti u nastavku su Gini indeks (eng. *Gini Index*) i dohvat informacija (eng. *Information Gain*). Nakon odabira najbolje prikladnog atributa, podijelimo korijenski čvor na podskupove koji sadrže moguće vrijednosti za najbolje atribute. Generiramo čvor stabla odlučivanja koji sadrži najbolji atribut te rekurzivno generiramo nova stabla odlučivanja koristeći podskupove skupa podataka prethodno kreiranog. Proces kreiranja sve dok se ne postigne razina u kojoj ne možemo dalje klasificirati čvorove i zadnji čvor ne nazovemo listom [36].

Sada ćemo detaljnije opisati tehnike ASM-a. Prva tehnika dobivanje informacija bavi se mjerenjem promjena u entropiji (metrika koja se koristi za mjerenje nečistoće u danom atributu koja određuje slučajnost podataka) nakon segmentacije skupa podataka baziranog na atributu. Izračunava koliko nam informacija značajke govori o klasi. Prema vrijednosti o dobivenoj informaciji, dijelimo čvor i gradimo stablo. Uvijek se pokuša maksimizirati vrijednost dobivene informacije i čvor/atribut sa najvećom vrijednošću se odvoji prvo [35]. Izračunava se formulom [36]:

$$Dobivena\ informacija = Entropy(S) - [(Prosječna\ težina) * Entropija(svaka\ značajka)]$$

Pri čemu je izraz za entropiju:

$$Entropija(s) = -P_{(da)}log2 \cdot log2 - P_{(ne)}log2 \cdot P_{(ne)}$$

Gdje su:

- S = Ukupan broj primjeraka
- $P_{(da)}$ = Vjerojatnost odgovora da
- $P_{(ne)}$ = Vjerovatnost odgovora ne

Druga tehnika ASM-a jest Ginijev indeks. Ginijev indeks je zapravo mjera čistoće ili nečistoće korištene tijekom stvaranja stabla odlučivanja algoritmom CART. Atribut s niskim Ginijevim indeksom trebao bi biti poželjniji u usporedbi s visokim indeksom. Kreira binarne podjele koje CART algoritam koristi [35]. Izračunava se koristeći sljedećom formulom [27]:

$$G_i = 1 - \sum_{k=1}^{n} p_{i,k}^2$$

Pri čemu je:

- $p_{i,k}$ = omjer klasa k instanci među primjerima treniranim instancama u i-tom čvoru

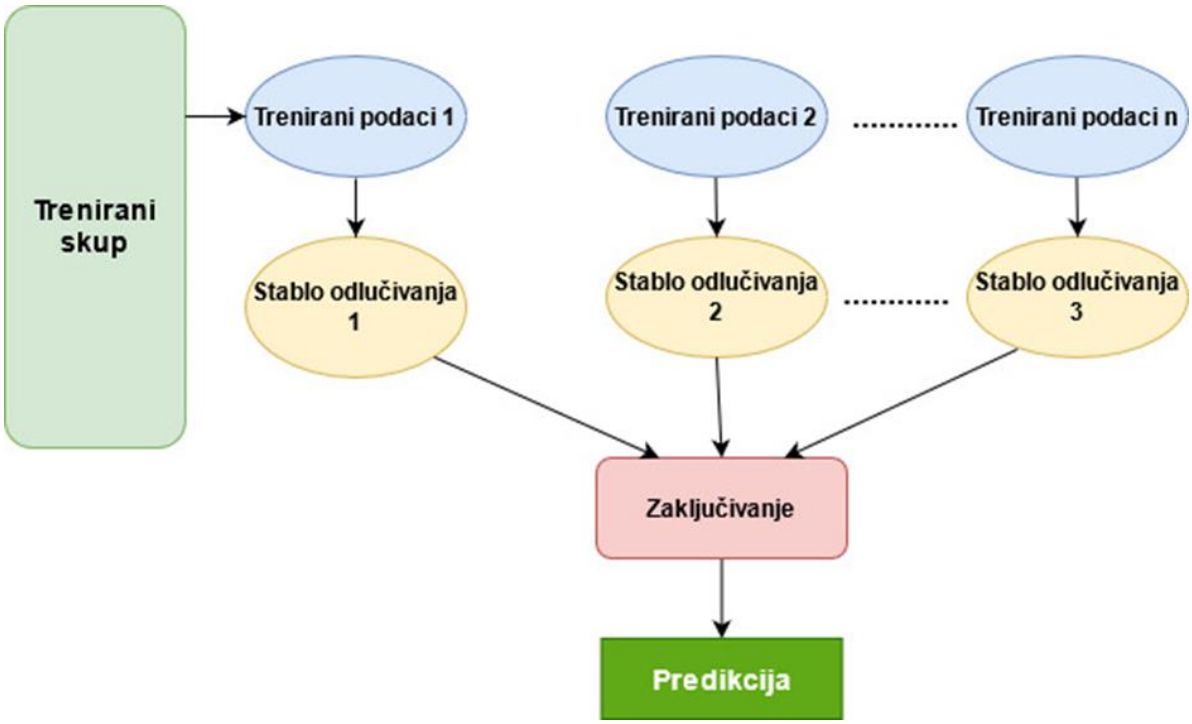
Kako bismo dobili optimalno stablo odlučivanja, potrebno je obrisati nepotrebne čvorove. Taj postupak nazivamo obrezivanjem. Preveliko stablo povećava rizik od redundantnih podataka, dok mala stabla možda neće obuhvatiti važne značajke skupa podataka. Zbog toga obrezivanjem smanjujemo veličinu stabla za učenje bez smanjenja njegove kvalitete [36].

4.1.2.3. Klasifikacija „Slučajnom“ šumom

Algoritam za klasifikaciju poznat pod nazivom „Slučajna“ šuma, popularan je algoritam strojnog učenja koji pripada tehnici nadziranog učenja. Uz već spomenutu klasifikaciju, može se koristiti i za probleme regresije, no više je prikladan za zadatke klasifikacije [37]. Baziran je na konceptu učenja koji se naziva *učenjem skupine* (eng. ***ensemble learning***). Npr. postavimo pitanje skupini od N ljudi te tih N odgovora zaključujemo izlaznu vrijednost koja može biti

kvalitetnija nego da ga postavimo na uzorku od jedne osobe. Kod grupnog učenja agregiramo skupinu predikcija koje su raspoređene po skupinama klasifikatora stabla odlučivanja, pri tome imajući na umu da svako stablo ima različiti, nasumično odabran podskup treniranog skupa podataka. Nakon što svako stablo donese odluku o izlazu za svoj podskup, sve vrijednosti se zbroje i donese se konačna odluka koja je onda konačno rješenje [27].

Na slici 18 je prikazana skica na koji način je klasifikacija Slučajnom šumom raspoređena:



Slika 18. Skica funkcioniranja algoritma Slučajne šume

Pošto se unutar „šume“ može pojaviti veliki broj stabala, postoji mogućnost da će neka stabla predvidjeti točan izlaz, a neka ne. Stoga veći broj stabala u šumi dovodi do veće točnosti modela i njegove pouzdanosti. Kako bi dobili ispravnu vrijednost u varijablama značajki skupa podataka moraju postojati stvarne vrijednosti kako bi klasifikator mogao predvidjeti točnu vrijednost, a ne pretpostavljenu [37].

Algoritam „Slučajne“ šume odvija se prema sljedećim koracima [37] :

1. Odaberemo nasumično K podatkovnih točaka iz treniranog skupa podataka
2. Izgradi stabla odlučivanja povezanim sa odabranim podatkovnim točkama
3. Odaberi broj N za stabla odlučivanja koja želimo izgraditi
4. Ponavljaj korake 1&2
5. Za nove podatkovne točke, pronađi predikciju za svako stablo odlučivanja i dodijeli izabranu kategoriju podskupa ostalim zaključcima i odredi konačan rezultat

Neki od razloga korištenja ovog algoritma jest već spomenuto korištenje kod zadataka regresije i klasifikacije, zahtijevanje manje vremena za treniranje u odnosu na ostale algoritme, predviđa izlaz sa velikom točnošću (brzo treniranje i predikcija), čak i za veliki skup podataka, korištenje za višedimenzionalne probleme, lako se provodi paralelno, ugrađena procjena generacijske pogreške i sl.[38][4].

4.1.2.4. Klasifikacija „Naivnim“ Bayesom

Sljedeći algoritam koji se koristi za klasifikaciju jest „*naivni*“ Bayes (engl. *Näive Bayes*). Naivni Bayes jest algoritam nadziranog učenja koji se koristi za klasifikaciju baziran na temelju Bayesovog teorema (pravila). Bayesov teorem ili Bayesovo pravilo koristi se za utvrđivanje vjerojatnosti hipoteze sa prethodnim znanjem [39]. Ovisi o uvjetnoj vjerojatnosti te je izražen pomoću sljedeće formule:

$$P(A|B) = \frac{P(B|A) \cdot P(A)}{P(B)}$$

Pri čemu su:

- $\mathrm{P(A|B)}$ – Vjerojatnost hipoteze A o promatranome događaju B
- $P(B|A)$ – Vjerojatnost dokaza s obzirom na to da je vjerojatnost hipoteze istinita
- $P(A)$ – Vjerojatnost hipoteze prije promatranja dokaza
- $P(B)$ – Vjerojatnost dokaza

Jedan je od najjednostavnijih i najefikasnijih klasifikacijskih algoritama koji nam pomaže brze modele strojnog učenja koji mogu brzo izraditi predikciju. Koristi se kod binarne i višeklasne klasifikacije u čijem je slučaju jako dobar u usporedbi s ostalim algoritmima [39]. Pokazao se učinkovitim u mnogim praktičnim primjenama kao što su klasifikacija teksta, medicinsku dijagnozu kao i upravljanje performansama sustava [40]. Jedan od glavnih nedostataka jest pretpostavka da su sve značajke neovisne ili nepovezane, pa ne može naučiti odnos između značajki [39].

4.2. Nenadzirano učenje

Nenadzirano učenje vrsta je strojnog učenja, u kojemu za razliku od nadzornog učenja, ne koristimo označene ulazne podatke. Nenadzirano učenje mnogo je teže od nadziranog, jer cilj je da se računalo nauči kako učiniti nešto bez da mu kažemo kako da to radi [30]. Takva vrsta učenja može se usporediti sa razmišljanjem u ljudskome mozgu tijekom učenja novih stvari. Ne može se direktno primijeniti na regresiji ili klasifikaciji jer nemamo odgovarajuće izlazne podatke. Cilj nenadziranog učenja je pronaći temeljnu strukturu skupa podataka, grupirati te podatke prema sličnostima i predstaviti taj skup podataka u prikladnome formatu [41].

Nenadzirano učenje je korisno za pronalaženje korisnih uvida u podacima, te sam rad na neobilježenim i nekategoriziranim podacima ga čini mnogo važnijim u odnosu na nadzirano učenje. Kao što smo već i spomenuli, vrlo je slično ljudskome načinu razmišljanja (učenju), što ga čini bliže umjetnoj inteligenciji, te ga možemo poistovjetiti sa stvarnim svijetom u kojemu nemamo uvijek ulazne podatke s odgovarajućim izlazima [41].

U odnosu na nadzirano učenje, nenadzirano učenje se koristi za složenije zadatke pošto nemamo označene ulazne podatke i lakše ga je izvesti zbog lakšeg dobivanja neoznačenih podataka nego u odnosu na označene, ali je zato mnogo teže jer nemamo označene ulazne podatke i zbog toga konačan rezultat može biti manje točan u odnosu na nadzirano učenje [41]. Dva tipa nenadziranog učenja koja ćemo obraditi u nastavku jesu **skupljanje** (engl. *Clustering*) i **udruživanje** (engl. *Association*).

4.2.1. Sakupljanje

Prvi tip nenadziranog učenja kojeg ćemo opisati jest **sakupljanje** (engl. *clustering*). Sakupljanje ili nakupina (engl. *cluster*) tehnika je strojnog učenja koja grupira neoznačeni skup podataka u različite nakupine, koji se sastoje od sličnih podatkovnih točaka. . Nakon primjene ove tehnike, svaka nakupina (klaster) ili grupa dobiva svoj ID. Sustav strojnog učenja koristi ovaj identifikator za pojednostavljivanje obrade velikih i složenih skupova podataka. Obično se koristi za statističku analizu podataka [42].

Ovo je primjerena tehnika alat za analizu podataka, segmentaciju kupaca, sustave preporuke, tražilice, segmentacija slika, dimenzionalnost, smanjenje i još mnogo toga. Slično je klasifikaciji prema tome što grupira instance u grupe, samo što radi sa neoznačenim skupom podataka [27].

Postoji nekoliko vrsta sakupljanja, a to su [42]:

- **Sakupljanje po particijama** (engl. Partitioning Clustering) : dijeli podatke u ne hijerarhijske skupine. Skup podataka je podijeljen u skupinu od K grupa, pri čemu je K broj predefiniranih grupa. Središte nakupa stvoreno je na takav način da je udaljenost jednog nakupa minimalna u usporedbi s drugim centroidima nakupa.
- **Sakupljanje zasnovano na gustoći** (engl. Density-Based Clustering) : povezuje visoko gusta područja u nakupine , a proizvoljno oblikovane raspodjele nastaju sve dok se gusta područja mogu međusobno povezati.
- **Sakupljanje zasnovano na modelu distribucije** (engl. Distribution Model-Based Clustering): podaci se dijele na temelju vjerojatnosti pripadnosti skupa podataka određenoj distribuciji.
- **Hijerarhijsko sakupljanje** (engl. Hierarchical Clustering): koristi se kao alternativa za sakupljanje po particijama jer nema potrebe za određivanjem broja nakupa koji

će se stvoriti unaprijed. Nakupi su podijeljeni kako bi se stvorila struktura nalik stablu koja se naziva dendrogramom.

- **Nejasno sakupljanje** (engl. Fuzzy Clustering): vrsta nakupljanja u kojoj objekt podataka može pripadati u više grupa/nakupina. Svaki skup podataka ima skup koeficijenata članstva, koji ovise o stupnju pripadnosti određenoj skupini.

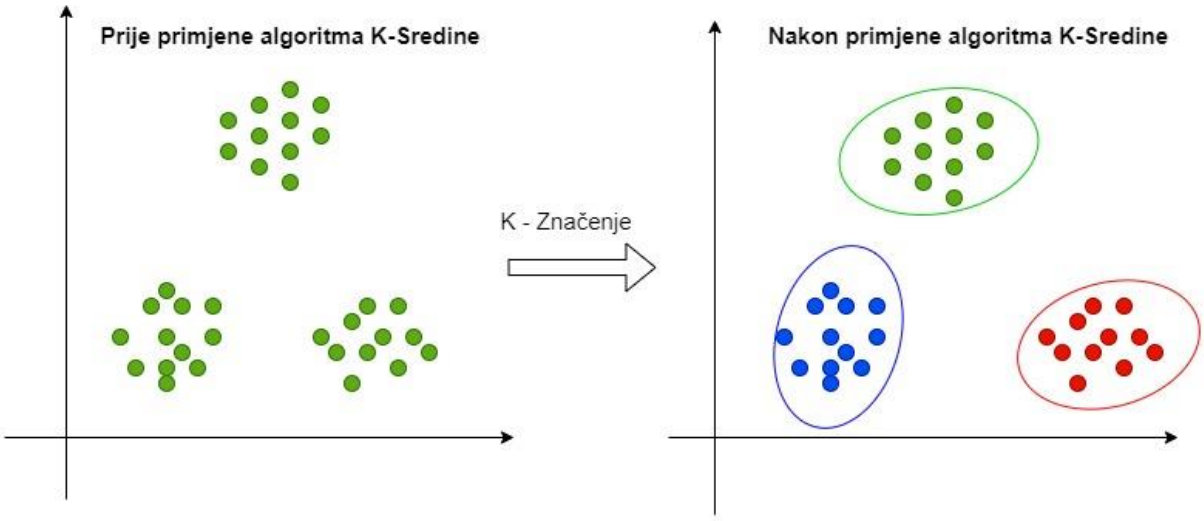
Poznate primjene u kojima se koristi sakupljanje mogu biti identifikacija stanice raka, pri čemu se stanice dijele na kancerogene i ne kancerogene stanice, odvajanju (segmentaciji) gostiju/kupaca, pri čemu se kupci/gosti grupiraju na temelju njihovih kupnji, aktivnosti na mrežnim stranicama u svrsi njihovog razumijevanja (sustavi za preporuke), mehanizmima pretraživanja pri čemu se pomoću tehnike sakupljanja na temelju sličnosti traženog objekta, biologiji za klasifikaciju različitih vrsti biljaka i životinja pomoću tehnike prepoznavanja slike [27][42].

Algoritmi sakupljanja koje ćemo obraditi u nastavku jesu algoritmi **K-sredine** (engl. *K-meaning*) i algoritam **DBSCAN**.

4.2.1.1. Sakupljanje K-sredinom

Algoritam ***K-sredine*** (engl. *K-mean*) je jednostavan iterativni algoritam nenadgledanog učenja koji grupira neobilježeni skup podataka u K različitih nakupina (klastera), a takav način da svaki skup podataka pripada samo jednoj grupi sa kojom ima najviše sličnih svojstva. Grupiranje podatka nam omogućuje na elegantan i prikladan način tako da i sami možemo otkriti kategorije grupa u neobilježenome skupu podataka bez potrebe za prevelikim treniranjem [43].

Algoritam je baziran na centroidu gdje je svaka nakupina (klaster) udružen sa centroidom. Glavni cilj ovog algoritma je smanjiti zbroj udaljenosti između podatkovne točke i njihovih odgovarajućih nakupina (klastera). Algoritam se uglavnom izvodi za zadatke da iterativnim postupkom odredi najbolju vrijednost za K središnje točke ili centroide, te da svakoj podatkovnoj točki dodijeli najbliži k-centar. Podatkovne točke koje su blizu određenog k-centra, kreiraju nakupinu. Svaki klaster ima podatkovne točke s nekim zajedničkim karakteristikama i udaljen je od drugih nakupina [43]. Na slici 19 su prikazane podatkovne točke prije i formirane nakupine (klasteri) nakon korištenja algoritma.



Slika 19. Podatkovne točke prije i nakon primjene algoritma K-Sredine (Izrada autora prema [43])

Koraci prema kojima se algoritam K-Sredine provodi opisan je sljedećim koracima [43]:

1. Odredi broj K koji predstavlja broj nakupina (klastera)
2. Odaberi nasumično K točaka ili centroida
3. Svaku podatkovnu točku dodijelite njihovom najbližem centroidu, koji će oblikovati predefinirani K broj nakupina
4. Izračunamo varijancu i postavimo novi centroid svakoj nakupini
5. Ponovite treći korak, što znači da svaku točku dodijelimo novom najbližem težištu svaku skupine
6. Ako se dogodi bilo kakva dodjela, prelazimo na korak 4, inače smo gotovi
7. Model je spreman

Složenost algoritma linearna je s obzirom na broj podatkovnih točaka, broj nakupa (klastera) K te broju dimenzija, ali to vrijedi samo kada podaci imaju strukturu sakupljanja. Ukoliko prethodni slučaj ne vrijedi, složenost se može eksponencijalno povećati sa brojem podatkovnih točaka, ali se to u praksi rijetko događa, te je algoritam jedan od bržih u skupini sakupljanja [27].

Neka od ograničenja (limita) algoritma jest da ga je potrebno pokrenuti nekoliko puta kako bismo dobili optimalno rješenje, uz to da moramo unaprijed odrediti broj nakupa (klastera) što zna biti naporno uz to da se algoritam baš i ne ponaša dobro kada nakupi imaju različitu veličinu, gustoću ili nepravilan oblik [27].

Jako važan zadatak prije samog pokretanja algoritma jest odabrati pravilan broj nakupa (klastera) K. Jedna od najboljih metoda za izračunavanje optimalnog broja nakupa jest *Lakat metoda* (engl. *Elbow method*). Metoda koristi koncept *Unutarnje sume zbroja kvadrata klastera*

(engl. *Within Cluster Sum of Squares* (u nastavku **WCSS**)) što ustvari definira ukupne varijacije unutar nakupa (klastera) [43]. Formula koja se koristi za izračun vrijednosti WCSS (za n klastera) je sljedeća [43]:

$$WCSS = \sum P_{i\ u\ nakupu1} udaljenost(P_i C_1)^2 + \sum P_{i\ u\ nakupu2} udaljenost(P_i C_2)^2 + \cdots + \sum P_{i\ u\ nakupuN} udaljenost(P_i C_N)^2$$

Pri čemu je:

- $\sum P_{i\ u\ nakupu1} udaljenost(P_i C_1)^2$ – zbroj kvadrata udaljenosti između svake podatkovne točke i njezinog centroida unutar prve nakupine i ostalih N nakupina.

Za mjerenje udaljenosti između podatkovnih točaka i centroida možemo se poslužiti metodama kao što su euklidska ili manhattanska. Kako bismo uspješno proveli prethodno spomenutu metodu prvo nam je potrebna vrijednost K elementa te za svakog od njih izračunavamo WCSS vrijednost. Zatim iscrtavamo krivulju izračunatih WCSS vrijednosti i broja nakupa K te za optimalnu vrijednost K uzima točku koja ima oštar nagib (ima položaj lakta) [43].

4.2.2. Učenje pravilom udruživanja

Pravilo **udruživanja** (engl. *association*) tehnika je nenadgledanog učenja koja se temelji na ovisnosti jedne podatkovne jedinice o drugoj , pridružujući ih tako da kombinacija bude čim više isplativija [44]. Cilj ove tehnike jest „*kopanje*“ u velike količine podataka i otkrivanje zanimljivih poveznica između atributa [27]. Objasnimo ovu izjavu na primjeru trgovine sa namirnicama. Prilikom vođenja dnevnika prodaje otkrili smo da ljudi koji kupuju krumpiriće i umak za roštilj također kupuju i odrezak. Nakon što smo otkrili prethodnu činjenicu napravit ćemo novi raspored u trgovini gdje će prethodno spomenute namirnice biti smještene blizu jedna drugoj. Osim u trgovinama pravilo se primjenjuje se i u analizi tržišnih košarica, rudarstvu korištenja weba, kontinuiranoj proizvodnji itd.

Učenje pravila udruživanja možemo podijeliti u tri vrste algoritma [44], a to su:

1. **Apriori**: koristi česte skupove podataka za generiranje pravila, te je dizajniran za rad sa bazama podataka koje sadrže transakcije. Za pretraživanje koristi pretraživanje u širinu (engl. *breadth-first search*) i Hash stabla za učinkovito izračunavanje skupa predmeta. Uglavnom se koristi za analizu tržišne košarice i pomaže u razumijevanju proizvoda koji se mogu kupiti zajedno. Također se može koristiti u zdravstvu za pronalaženje reakcija na lijekove za pacijente

2. **Eclat**: kratica za *Transformaciju klase ekvivalencije* (engl. *Equivalence Class Transformation*), te za pretraživanje koristi algoritam pretraživanja u dubinu (engl. *depth-first search*). Također se koristi za rad sa bazama podataka koje sadrže transakcije i radi brže nego **Apriori** algoritam.
3. **Algoritam F-P rasta**: oznaka je za *Česti uzrok* (engl. ***Frequent Pattern***) i poboljšana je verzija **Apriori** algoritma. Predstavlja bazu podataka u obrazcu strukture stabla poznatom kao česti uzorak ili stablo. Svrha ovog stabla je izvlačenje najčešćih uzoraka.

Pravilo pridruživanje temelji se na konceptu izjava „If - Else“ pri ćemu se izjava „*If*“ naziva **prethodnik** (engl. *antecedent*), a izjava „*Else*“ **posljedica** (engl. *consequent*). Te su vrste odnosa u kojima možemo otkriti povezanost ili povezanost dviju stavki poznate kao **pojedinačna kardinalnost** (engl. *single cardinality*). Temelji se na kreiranju pravila i ako se broj pravila povećava tada i se i sukladno tome povećava i kardinalnost. Za mjerenje ovisnosti između uzoraka imamo nekoliko metrika, a to su [44]:

- **Podrška**
- **Samouvjerenost**
- **Lift**

5. Praktični primjer

U ovome poglavlju će biti prikazana aplikacija koja će prikazati načine na koji se način mogu pretraživati podaci i kako se dodavanjem komponente strojnog učenja sa tim istim podacima može implementirati interesantan ekspertni sustav. U ovome radu je implementiran ekspertni sustav za predviđanje cijene nekretnina (kuća, zgrada) prema određenim parametrima, o kojima će više riječi biti u nastavku. Implementirani sustav se sastoji od dviju aplikacija, prve koja predstavlja poslužiteljski dio i implementirana je u programskome jeziku C# i sastoji se od ASP.NET CORE-a i ML.NET komponente i druge koja predstavlja klijentski dio i implementirana je u programskome okviru Angular. Korišteni skup podataka preuzet je sa *Kaggle-a*, a pohranjuje se u bazu podataka sa pogodnostima programskoga jezika C#-a. Sustav za upravljanje bazom podataka korišten u ovome projektu jest Microsoft SQL Server, dok se u aplikaciji za rad sa bazom podataka koristi *Entity Framework Core*, i to princip *Prvo kôd – nova baza* (eng. *Code First – New Database*). Također uz već spomenute funkcionalnosti, također su i implementirane CRUD operacije za oblikovanje postojećih podataka. Cijelo programsko rješenje zajedno sa klijentskom i poslužiteljskom aplikacijom dostupno je na sljedećoj poveznici: https://github.com/hsostaric/upravljanjeNekretninama .

5.1. Analiza korištenog skupa podataka

Kao što smo već i ranije spomenuli, za potrebe implementacije aplikacije bit će nam potreban nekakav skup podataka sa relevantnim podacima. Na mrežnoj stranici Kaggle-a pronašli smo skup podatka sa prodajom kuća na području King Country-a u SAD-u. Navedeni skup podataka sadrži prodajne cijene kuća King Country-a koje su prodane između svibnja 2014-te i svibnja 2015-te je dostupan na poveznici navedenoj u literaturi [45].

Za potrebu izrade programskog rješenja nisam koristio sve atribute koji su sadržani u skupu podatka već samo one koje sam smatrao najrelevantnijim. Korišteni atributi sa kratkim objašnjenjem koje sam pohranio u bazu podataka su sljedeći:

- **Price** – cijena po kojoj je kuća prodana (izražena u američkim dolarima $).
- **Bedrooms** – broj soba u kući.
- **Bathrooms** – broj kupaonica, pri čemu je moguće da je u decimalnome zapisu pri čemu .5 označava prostoriju sa toaletom bez tuša.
- **Square footage living** – Kvadratna stopa unutarnjeg stambenog prostora unutar interijera (prostora namijenjenog za život).
- **Square footage lot** - Kvadratna stopa kopnenog prostora.

- **Floors** – broj katova.
- **View** – Indeks u rasponu od 1-4 koji označava koliko je bila dobra zainteresiranost za nekretninu.
- **Condition** – Indeks u rasponu od 1-5 koji označava stanje kuće.
- **Year built**– godina u kojoj je kuća izgrađena.
- **Year renovated** – godina zadnjeg renoviranja.
- **Grade** - Indeks u rasponu od 1-13, pri čemu između 1-3 označava da zaostaje u izgradnji i dizajnu, 7 da ima prosječnu razinu dizajna i izgradnje i 11-13 ima visoku kvalitetu izgradnje i dizajna.
- **Square footage above** – kvadratna stopa unutarnjeg stambenog prostora iznad razine tla.
- **Square footage basement** – kvadratna stopa unutarnjeg stambenog prostora ispod razine tla

Za izgradnju svoje aplikacije odabrao sam parametre koji se konkretno vežu za kuću, pri čemu sam izbavio datum kada je kuća prodana, pogled na rivu (vodu), poštanski broj, geografsku dužinu i širinu, kvadraturu unutarnjeg stambenog prostora unutarnjeg stambenog prostora za najbližih 15 susjeda, te kvadratura zemljišnih parcela najbližih 15 susjeda. Na slici 20 se nalaze zapisi pohranjeni u bazi podataka sa prethodno definiranim vrijednostima.

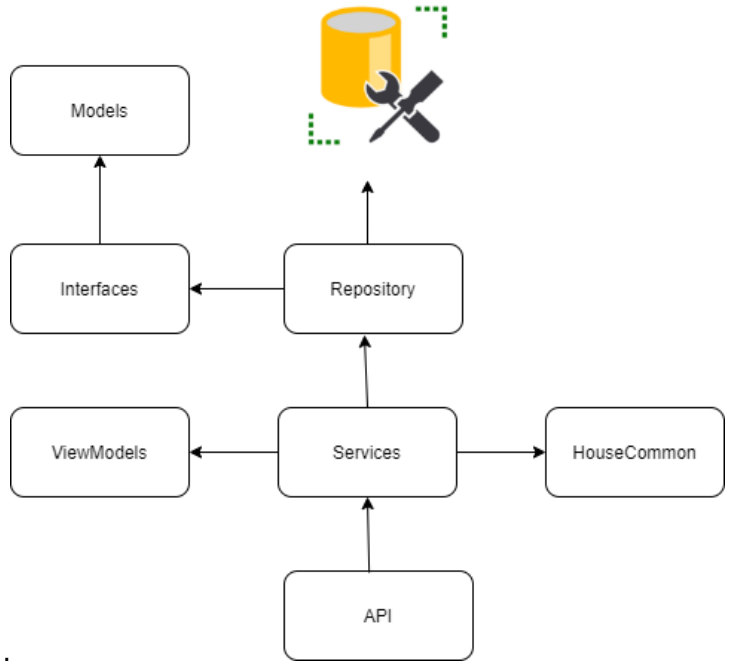
	Id	Price	Bedrooms	Bathrooms	SqftLiving	SqftLot	Floors	View	Condition	YearBuilt	YearRenovated	Grade	SqftAbove	SqftBasement
1	1	221900	3	1	1180	5650	1	0	3	1955	0	7	1180	0
2	2	538000	3	2,25	2570	7242	2	0	3	1951	1991	7	2170	400
3	3	180000	2	1	770	10000	1	0	3	1933	0	6	770	0
4	4	604000	4	3	1960	5000	1	0	5	1965	0	7	1050	910
5	5	510000	3	2	1680	8080	1	0	3	1987	0	8	1680	0
6	6	1225000	4	4,5	5420	101930	1	0	3	2001	0	11	3890	1530
7	7	257500	3	2,25	1715	6819	2	0	3	1995	0	7	1715	0
8	8	291850	3	1,5	1060	9711	1	0	3	1963	0	7	1060	0
9	9	229500	3	1	1780	7470	1	0	3	1960	0	7	1050	730
10	10	323000	3	2,5	1890	6560	2	0	3	2003	0	7	1890	0

Slika 20. Zapisi kuća u bazi podataka sa korištenim atributima (Izrada autora)

5.2. Analiza poslužiteljskog dijela aplikacije

U ovome poglavlju ćemo opisati poslužiteljski dio aplikacije koji je implementiran pomoću programskog jezika C#, koristeći ASP.NET CORE API i komponentu strojnog učenja ML.NET. Programsko rješenje (engl. *Solution*) se sastoji od 10 modula (projekata) pri čemu svaki od njih ima određenu ulogu u ostvarivanju končanog rješenja. Neki od modula bivaju generirani tijekom kreiranja modela za ML.NET, dok sam veliku većinu kreirao samostalno. Kod kreiranja

strukture projekata vodio sam se preporukama sa [46], a na slici 21 je grafički prikaz modula sa međusobnom povezanošću, te ćemo ukratko objasniti ulogu svakog od njih.



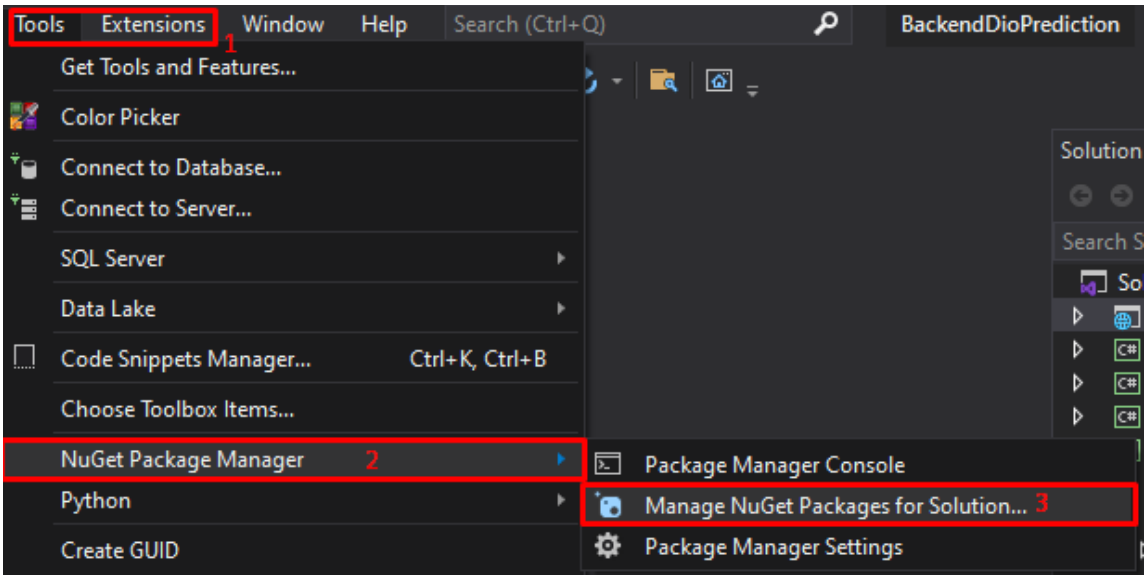
Slika 21. Struktura modula poslužiteljskog dijela aplikacije (Izrada autora prema [46])

Sada ćemo objasniti module sa prethodne slike, a prvi od njih je **Repository**. On sadrži klasu koja komunicira sa bazom podataka i sadrži metode za CRUD operacije. Za rad sa podacima koristi se LINQ. Sljedeći moduli koje ćemo opisati i koje koristi **Repository** su **Interfaces** i **Models**. Modul **Models** sadrži entitetsku klasu, čiji se atributi migracijom preslikavaju na bazu podataka. Također sadrži generirane klase migracija koje terminalnim naredbama rade izmjene nad bazom podataka. Posljednja klasa koja se nalazi unutar **Models**-a jest *Context* klasa. Njena uloga jest da predstavlja sesiju s bazom podatka koja se može koristiti za postavljanje upita i spremanje instanci korisnikovih entiteta u bazu podataka [47]. Sadrži nadjačavanje metode za učitavanje podatka iz datoteke o kojoj će više riječi biti u jednome od sljedećih poglavlja. Modul **Interfaces** koristimo za kreiranje sučelja, koja bivaju implementirana kod repozitorija i servisa. Konkretno se koristi kako bismo implementirali uzorak *nezavisnog ubrizgivanja* (engl. *Dependency injection*) za potrebe lakšeg rada i održavanja koda. Modul **ViewModels** koristimo za pretvorbu podatka koje dobijemo sa klijentske strane u tip podataka sa kojima radimo na poslužiteljskoj strani. U modulu **HouseCommons** se nalaze generirana ulazna i izlazna klasa za potrebe predviđanja vrijednosti cijene kuće. U nastavku ćemo detaljnije opisati dio vezan uz ML.NET i samu implementaciju poslužiteljske aplikacije sa odgovarajućim kodom.

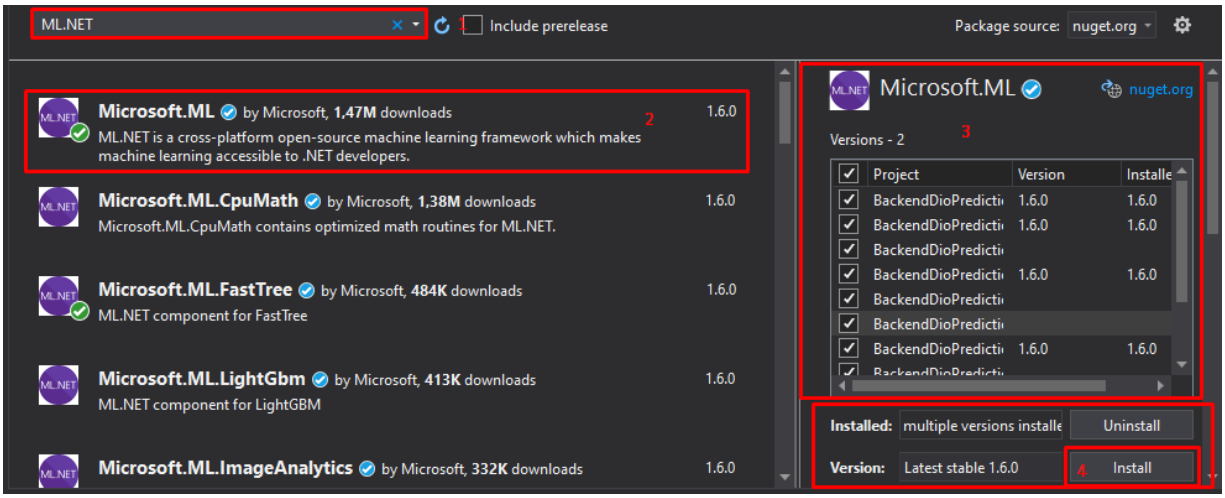
5.2.1. Izrada komponente ML.NET

Biblioteka ML.NET je besplatna platforma, otvorenog koda namijenjena za strojno učenje razvijena za .NET razvojnu platformu. Omogućuje treniranje, izgradnju i isporuku prilagođenih modela pomoću C# i F# za razne ML scenarije. Uključuje značajke poput automatiziranog strojnog učenja (AutoML koja će biti korištena u ovome projektu) i alate poput ML.NET CLI i ML.NET Model Builder, koji čine integraciju strojnog učenja u aplikaciji još lakšom [49].

Kako bismo dodali komponentu ML.NET-a u naš projekt, za početak je potrebno dodati proširenje pomoću NuGet upravitelja. Za dodavanje paketa u naše rješenje potrebno je ići na sljedeće *Tools – NuGet Package manager – Manage NuGet Packages for Solution* (Slika 22), te nakon što nam se otvori prozor za instalaciju ekstenzija u tražilicu upišemo ML.NET te odaberemo *Microsoft.ML* i odaberemo Projekt (rješenje) u koji želimo dodati ML.NET i kliknemo na gumb *Install* (Slika 23).

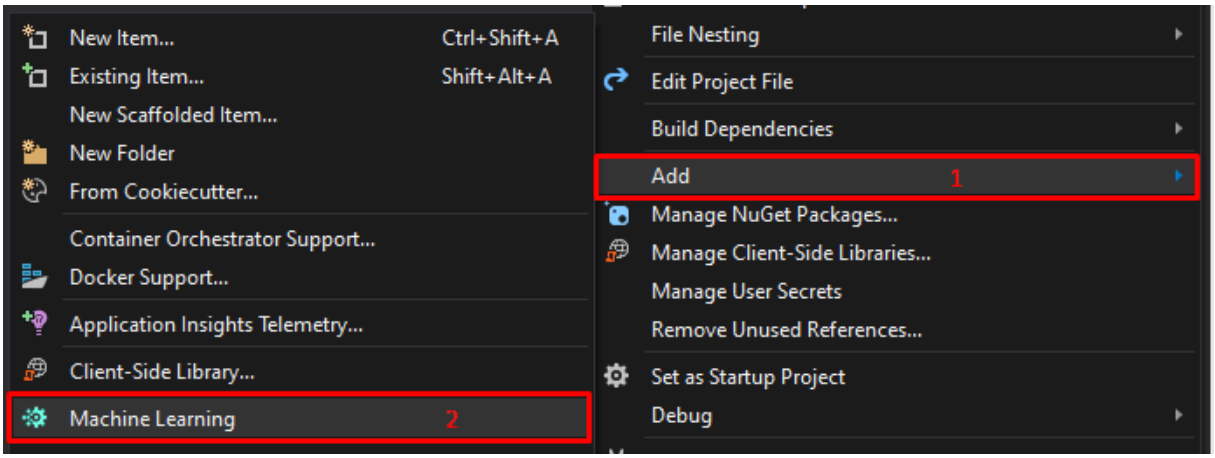


Slika 22. Koraci za otvaranje Solution Manager-a



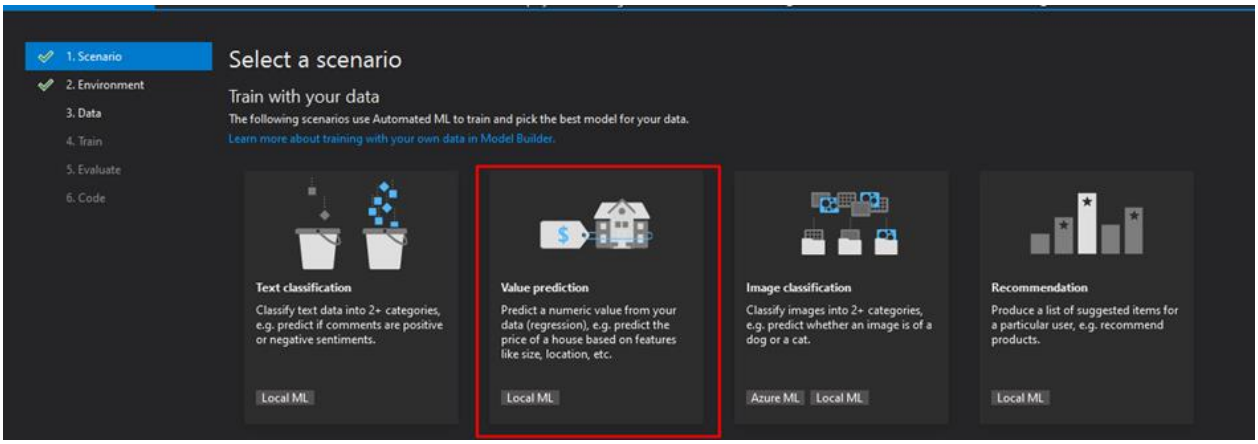
Slika 23. Instaliranje ML.NET-a

Nakon instalacije proširenja potrebno je dodati *Model Builder* unutar projekta. To postižemo desnim klikom miša na jedno od naših rješenja i odabirom opcije *Add – Machine Learning* (Slika 24).



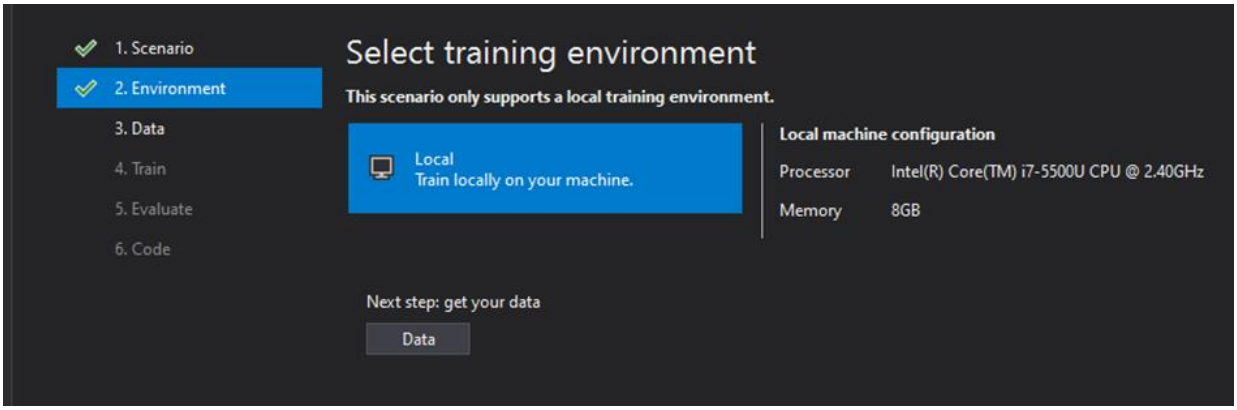
Slika 24. Dodavanje Machine Learning komponente

Nakon što smo odabrali opciju za dodavanje komponente ML.NET otvara nam se opcija za odabir vrste scenarija. Vrste scenarija koji nam se nude vezani su za klasifikaciju tekstualnih podataka, predikcija cijena u što je uključeno predikcija numeričkih vrijednosti iz naših podataka koristeći regresiju, klasifikacija slika u dvije ili više kategorija, te sustav za preporuke. Mi ćemo odabrati scenarij za predikciju cijena (Slika 25).



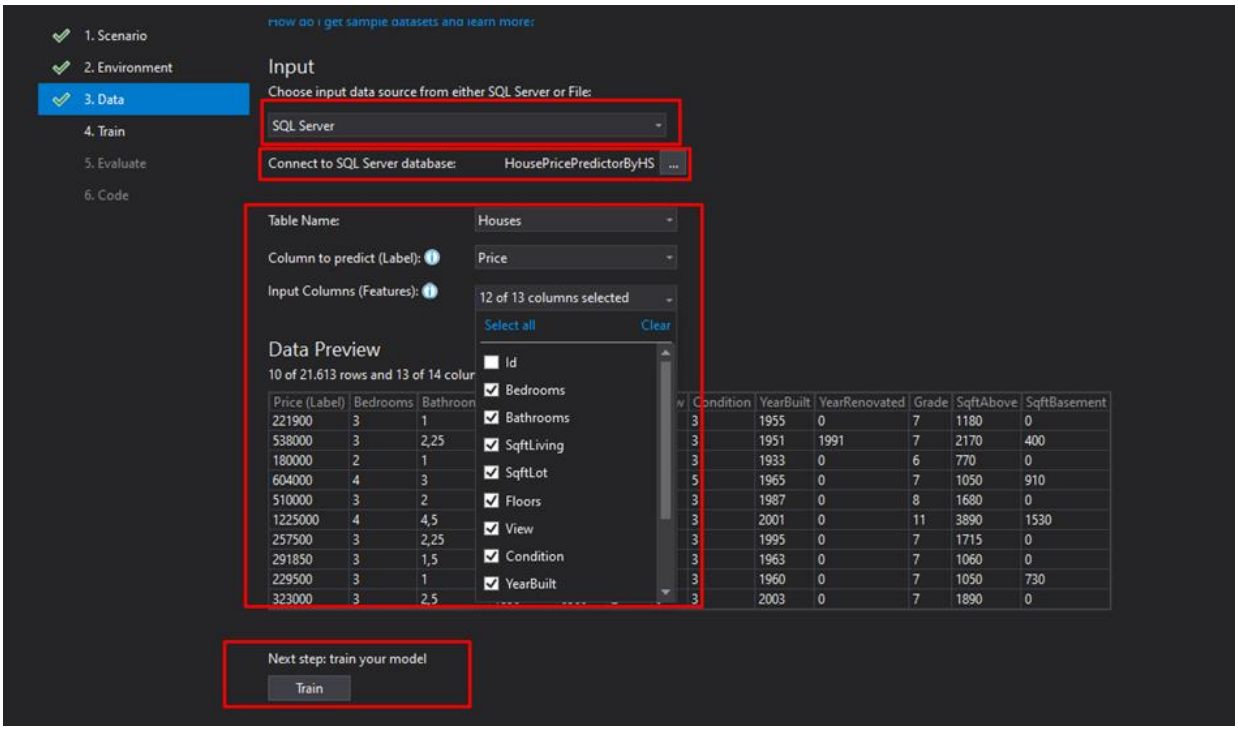
Slika 25. Odabir scenarija

Sljedeći korak koji je potrebno odraditi jest odabir okoline za treniranje. Opcija koja nam se nudi jest da treniramo lokalno, ali u nekim slučajevima je moguće i korištenje tehnologije *oblaka* (engl. *cloud-a*). Treniranje lokalno ovisi o računalnim performansama kao što su kvaliteta procesora i radne memorije i diska, što utječe na kvalitetu i brzinu dobivanja rezultata. U našem slučaju odabiremo ono što nam je jedino i dostupno, a to je lokalno okruženje (Slika 26).

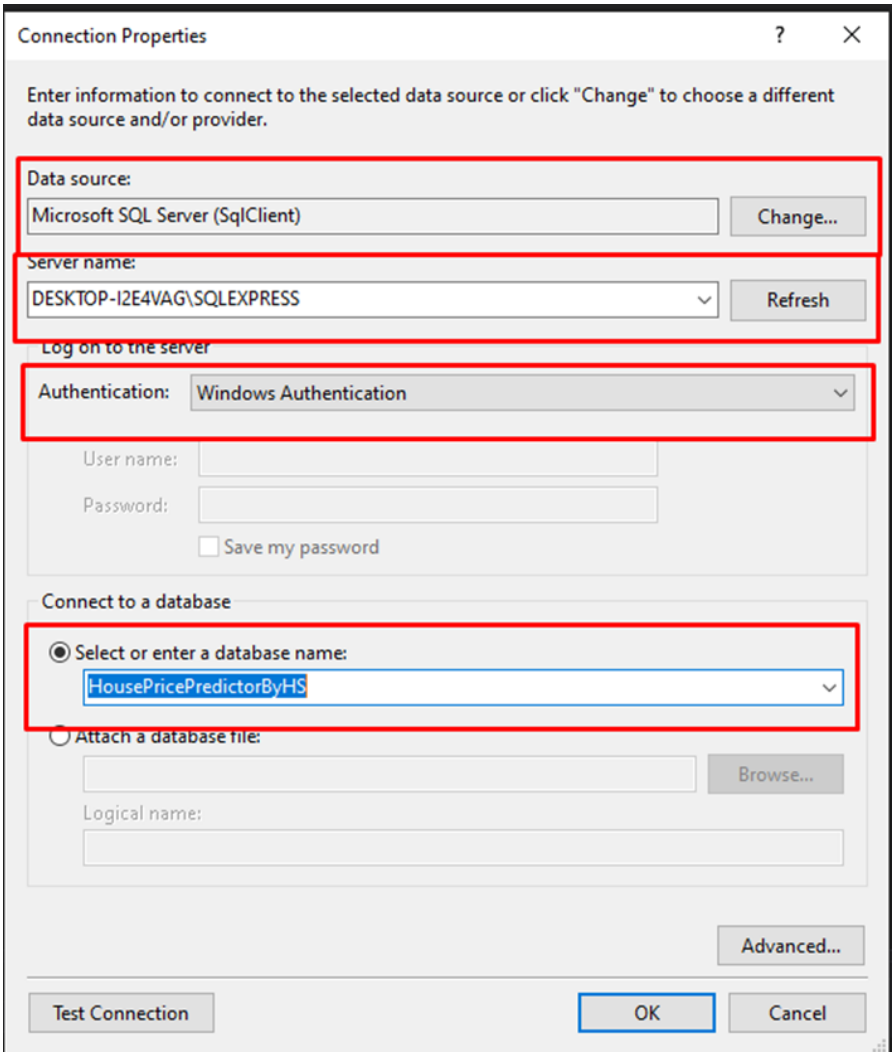


Slika 26. Odabir okoline za treniranje

Nakon odabire okoline za treniranje dolazimo do dijela vezanog za odabir podataka koji ulaze u skup za treniranje. Nude nam se opcije za odabir lokalne datoteke i SQL Server-a. Mi odabiremo opciju za SQL Server-a (Slika 27), nakon čega odabiremo bazu podataka iz koje uzimamo podatke. Kod odabira podataka konfiguracije za bazu podataka kod svojstva konekcije odabiremo izvor podataka što je u našem slučaju *Microsoft SQL Server (SqlClient)*, ime poslužitelja, način autentifikacije na poslužitelja i bazu podataka na koju se spajamo (Vidi sliku 28). Nakon konfiguriranja veze prema bazi podataka odabiremo tablicu iz baze, stupac za predikciju (labelu) i ulazne stupce (obilježja) i odabiremo dugme treniraj (Slika 27).

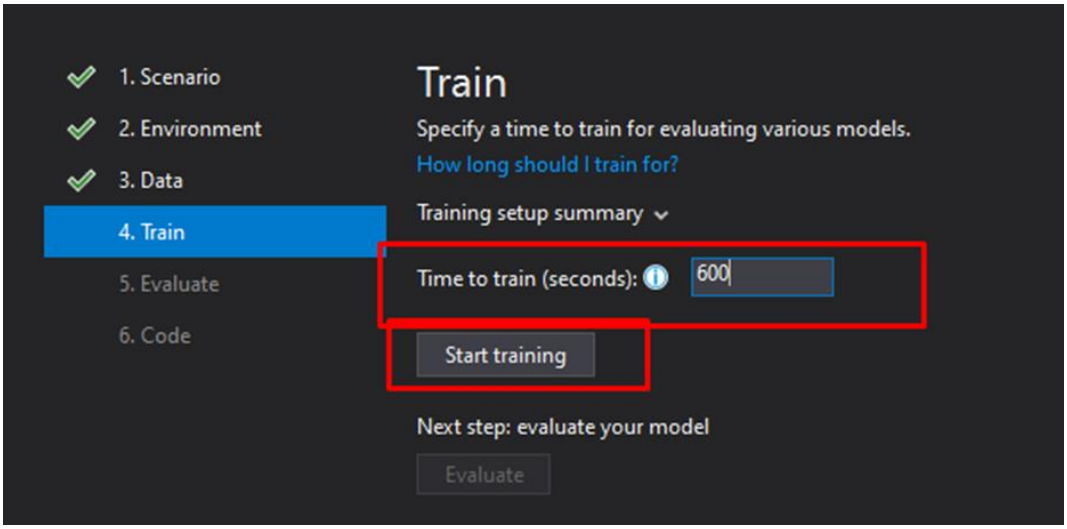


Slika 27. Odabir izvora podataka i ulaznih stupaca



Slika 28. Konfiguriranje veze za bazu podataka

Sljedeće što nam je preostalo za napraviti jest unijeti potrebno vrijeme za treniranje u sekundama (Slika 29). Nakon pritiska na tipku *„Start training“* pričekamo vrijeme koje smo unijeli i nakon toga dobijemo ispis svih algoritama u kombinaciji zajedno sa atributima poput *R-kvadratne* metode, apsolutnog gubitka, kvadratnog gubitka, trajanja, broja iteracija i sl.



Slika 29.Prozor za odabir treniranja podataka.

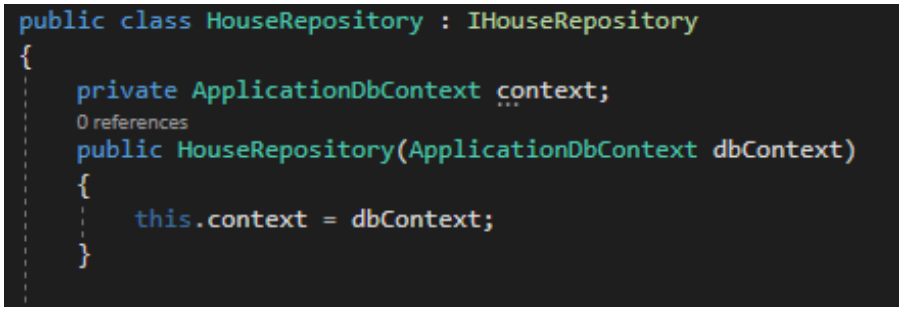
	Trainer	RSquared	Absolute-loss	Squared-loss	RMS-loss	Duration	#Iteration	
1	FastTreeTweedieRegression	0,7051	125584,10	35690157197,93	188918,39	2,0	1	
2	LightGbmRegression	0,7031	123496,81	35937688520,89	189572,38	0,9	2	
3	FastTreeTweedieRegression	0,7012	122635,32	36162718365,06	190164,98	0,8	3	
4	FastTreeTweedieRegression	0,6938	124576,99	37058892803,23	192506,86	4,6	4	
5	LightGbmRegression	0,6925	123910,39	37223450957,15	192933,80	0,8	5	

Slika 30. Konačni rezultati treniranja

Na slici 30 su vidljivi konačni rezultati nakon treniranja modela. Dobili smo da je najbolji model *FastTreeTweedieRegression* sa najboljom kvalitetom R-kvadratne funkcije 0,7051 što je za realne predikcije uveliko prihvatljivo.

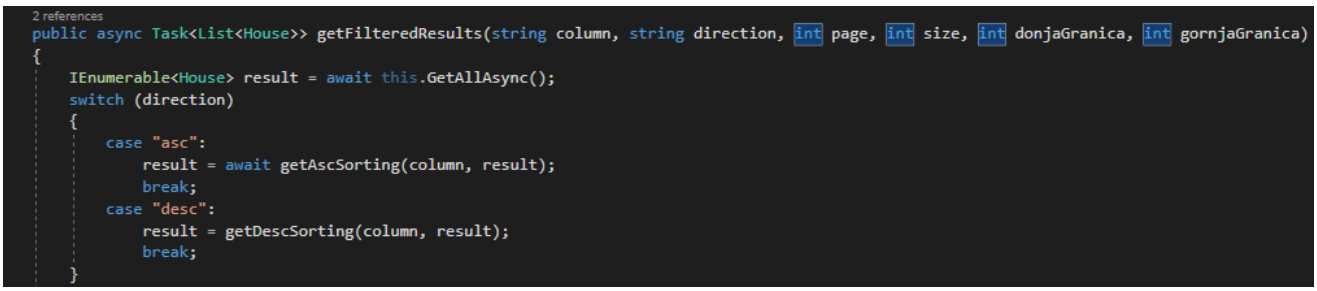
5.2.2. Implementacija poslužiteljskog dijela aplikacije

U ovom poglavlju ćemo detaljnije opisati kako je koji dio aplikacije implementiran na poslužiteljskom dijelu aplikacije. Krenut ćemo od modula koji direktno komunicira sa bazom podataka, a to je **Repository**. Modul **Repository** sadrži klasu **HouseRepository** koja implementira sučelje iz modula **Interfaces** pod nazivom **IHouseRepostory** kako bismo postigli već ranije *nezavisno ubrizgivanje*. Sadrži metode za brisanje zapisa prema identifikatoru, dohvat zapisa prema identifikatoru, dohvat svih zapisa, spremanja zapisa, dohvat filtriranih podataka te dohvat konfiguracijskih podataka kao i dohvat broja zapisa za potrebe straničenja. U samoj klasi deklariramo objekt za dohvat podataka koji nasljeđuje *DbContext* o kojoj smo govorili ranije (Slika 31).



Slika 31. Deklariranje objekta za rad sa podacima pomoću nezavisnog ubrizgivanja (Izrada autora)

Još ćemo iz ove klase detaljnije objasniti metodu za dohvat filtriranih podataka za straničenje, sortiranje i filtriranje prema donjoj i gornjoj granici cijene, **getFilteredResults(** string column, string direction, int page, int size, int donjaGranica, int gornjaGranica**)**. Metoda započinje sa dohvatom svih podataka iz baze podataka, te se ispituje vrijednost atributa *direction* (Slika 32),



Slika 32. Dohvat svih podataka i ispitivanje atributa direction

te ovisno o njenoj vrijednosti se poziva metoda koja sortira podatke prema vrijednosti parametra *column*. Sortiranje se ostvaruje pomoću LINQ naredbe **OrderBy** ili **OrderByDescending**, ovisno o smjeru sortiranja. Nakon odrađenog sortiranja slijedi filtriranje prema vrijednosti cijena kuća. Logika je da se prvo izvrši provjera postoje li vrijednosti gornje i donje granice cijene, te ako postoji kreće se sa provjerom je li možda donja granica veća od gornje i ako je, izvrši zamjenu te pomoću LINQ naredbe **Where** filtriraj vrijednosti. Inače izvršava se zasebna provjera za svaku vrijednost i tako dohvaća vrijednost za tražene rezultate. Na kraju metode se izvršava dohvat određenih broja podatka pomoću LINQ naredbi **Skip** i **Take** koje primaju parametre *page* koji označava broj stranice i parametar *size* koji označava broj zapisa koji će se vratiti korisniku (Slika 33).

```
if (donjaGranica > 0 && gornjaGranica > 0)
{
    if (donjaGranica > gornjaGranica)
    {
        int pom = 0;
        pom = gornjaGranica;
        gornjaGranica = donjaGranica;
    }
    result = result.Where(x => x.Price <= gornjaGranica && x.Price >= donjaGranica);
}
else
{
    if (donjaGranica > 0)
    {
        result = result.Where(x => x.Price >= donjaGranica);
    }
    if (gornjaGranica > 0)
    {
        result = result.Where(x => x.Price <= gornjaGranica);
    }

}

return result.Skip((page - 1) * size).Take(size).ToList();
```

Slika 33. Dio vezan za filtriranje podataka prema vrijednosti cijena (Izrada autora)

Sljedeća klasa koju vrijedi izdvojiti jest **ApplicationDbContext** koja se nalazi u modulu **Models** i koja nasljeđuje **DbContext**. Na slici 34 slijedi slika klase te ćemo ukratko objasniti njene dijelove.

```
public class ApplicationDbContext : DbContext
{
    0 references
    public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options) : base(options)
    {

    }
    5 references
    public DbSet<House> Houses { get; set; }

    0 references
    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        string path = @"D:\foids\ds\3.semestarHS\diplomski_rad\praktičniRad\programskikod\backendDio\BackendDioPrediction.Models\Context\kc_house_data.csv";
        int i = 0;
        modelBuilder.Entity<House>().Property(e => e.Id).UseIdentityColumn(100, 1);
        List<House> items = File.ReadAllLines(path)
            .Skip(1)
            .Select(line => line.Split(","))
            .Select(house => new House
            {
                Id = ++i,
                Price = (int)this.parseToFloat(house[2]),
                Bedrooms = this.parseToInt(house[3]),
                Bathrooms = this.parseToFloat(house[4]),
                SqftLiving = this.parseToInt(house[5]),
                SqftLot = this.parseToInt(house[6]),
                Floors = this.parseToFloat(house[7]),
                View = this.parseToInt(house[9]),
                Condition = this.parseToInt(house[10]),
                Grade = this.parseToInt(house[11]),
                YearBuilt = this.parseToInt(house[14]),
                YearRenovated = this.parseToInt(house[15]),
                SqftAbove = this.parseToInt(house[12]),
                SqftBasement = this.parseToInt(house[13])
            }).ToList();
        modelBuilder.UseIdentityColumns(1, 1).Entity<House>().HasData(items);

    }
```

Slika 34. Klasa zadužena za rad sa bazom podataka (Izrada autora)

Na početku je vidljivo da u konstruktoru „*injektiramo*“ svojstva koja prosljeđuje apstraktnoj klasi. Kako bismo to postigli, u klasi *Startup.cs* u metodi **ConfigureServices** dodajemo

servisima DbContext sa podacima za povezivanje sa bazom podataka što smo definirali u datoteci **appsettings.json**. Sljedeće što definiramo jest skup podataka **DbSet** sa tipom **House** koji koristimo za dohvat operacija za rad sa bazom podataka i preko kojih sa migracijama radimo promjene u bazi. Nadjačavanu metodu **OnModelCreating** koristimo za učitavanje podataka iz .csv datoteke za zapisima kuća koje smo preuzeli sa [45]. Zapise stavljamo u listu te ih pomoću **modelBuilder-a** pripremamo za zapisivanje u bazu podataka pomoću migracija.

Sljedeće modul koji ćemo opisati jest **Services**, točnije klasu **HouseService** u kojoj pozivamo metode iz **Repository** modula te radimo predikciju cijene kuće prema korisnikovome unosu. Klasa **HouseService** implementira sučelje **IHouseService** za potrebe lakšeg *„injektiranja“* u druge klase, te ima definirane atribute tipa **IHouseRepository** sa kojim dohvaća metode iz **Repository** klase, te atribut tipa **PredictEnginePool<HouseData, HouseOutput>** koji koristimo za predikciju vrijednosti kuća, pri čemu **HouseData** sadrži korisnikove ulazne podatke za predikciju, dok **HouseOutput** sadrži vrijednost za izlaz (Slika 35).

```
public class HouseService : IHouseService
{
    private IHouseRepository houseRepository;
    private readonly PredictionEnginePool<HouseData, HouseOutput> predictEnginePool;
    0 references
    public HouseService(IHouseRepository _houseRepository, PredictionEnginePool<HouseData, HouseOutput> _predictEnginePool)
    {
        houseRepository = _houseRepository;
        predictEnginePool = _predictEnginePool;
    }
    2 references
    public float PredictHousePrice(HouseData houseData)
    {
        HouseOutput houseOutput = predictEnginePool.Predict(modelName: "HouseValuePredictModel", example: houseData);
        return houseOutput.Score;
    }
```

Slika 35. Klasa HouseService sa definiranim atributima i metodom za predikciju (Izrada autora)

Za razliku od objekta tipa **PredictionEngine** koji nije siguran od dretvi (engl. *not thread-safe*) kojeg bismo trebali kreirati svugdje u projektu, što prilikom rasta aplikacije postaje neizvodivo za korištenje, koristimo servis **PredictEnginePool** u kombinaciji sa „*nezavisnim ubrizgivanjem*“, čije je korištenje prihvatljivije. Izvor koji koristi za predikciju kao i naziv modela definiramo u metodi **ConfigureServices** u klasi *Startup* zajedno sa registracijom ostalih klasa za potrebe „*nezavisnog ubrizgivanja*“ kao i konfiguracijom za bazu podataka i davanjem dozvola za bilo kakvim zahtjevima sa klijentske aplikacije prema poslužiteljskoj (Slika 36).

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddDbContext<ApplicationDbContext>(option => option.UseSqlServer(Configuration.GetConnectionString("DefaultConnection")));
    services.AddScoped<IHouseRepository, HouseRepository>();
    services.AddScoped<IHouseService, HouseService>();
    services.AddPredictionEnginePool<HouseData, HouseOutput>()
   .FromFile(modelName: "HouseValuePredictModel", filePath: "./MLModel.zip", watchForChanges: true);
    services.AddControllers();

    services.AddCors(c =>
        c.AddPolicy("AllowOrigin", options => options.AllowAnyOrigin().AllowAnyMethod().AllowAnyHeader()));
}
```

Slika 36. Konfiguriranje postavki za baze podataka, servisa i konfiguriranje modela za predikciju (Izrada autora)

Posljednji važniji modul koji nam je preostao za objasniti jest **API**. Taj modul sadrži **Controller** klasu u kojoj se nalaze krajnje točke (engl. *EndPoint*) za komunikaciju prema korisniku i **Response** klasa koja sadrži zastavicu tipa **boolean** i Listu tipa **House** za vraćanje zapisa sa poslužitelja. U njoj se pozivaju metode iz **Services** modula. Bazična putanja kontrolera jest **„api/house“**. Svaka metoda dobiva dodatan dio putanje kao i vrstu HTTP zahtjeva. Na slici 37 je prikazana klasa **HouseController** sa „*injektiranom*“ instancom **IHouseService-a** i metodom za dohvat kuća **GetHouseAsync**. Svaki parametar metode predstavlja dio upitnog parametra (engl. *query parameter*) putanje URL-a korisnikovog zahtjeva.

```
[Route("api/house")]
[ApiController]
1 reference
public class HouseController : ControllerBase
{

    private IHouseService houseService;
    0 references
    public HouseController(IHouseService _houseService)
    {
        houseService = _houseService;
    }

    [HttpGet]
    0 references
    public async Task<IActionResult> GetHousesAsync(string column, string direction, int page, int size, int donjaGranica = 0, int gornjaGranica = 0)
    {
        List<House> houses = await houseService.getFilteredResults(column, direction, page, size, donjaGranica, gornjaGranica);
        Response response = new Response();
        if (houses.Count > 0 && houses != null)
        {
            response.Success = true;
            response.Data.AddRange(houses);
            return Ok(response);
        }
        else
        {
            response.Success = false;
            response.Data = new List<House>();
            return BadRequest(response);
        }

    }
```

Slika 37. Dio koda klase HouseController-a (Izrada autora)

5.3. Analiza klijentskog dijela aplikacije

U ovome poglavlju ćemo proći najbitnije dijelove klijentskog dijela aplikacije. Kao što smo spomenuli u jednome od ranijih poglavlja, klijentski dio aplikacije implementiran je pomoću programskog okvira Angular verzije 12. Projekt se sastoji od 4 komponente koje se sastoje od .ts, .html i .scss datoteka, servisa za povezivanje sa poslužiteljskom stranom, konfiguracijske klase unutar koje se nalazi početna putanja prema poslužitelju i entitetske klase za rad sa objektima kuća. Za početak ćemo obraditi klasu za komunikaciju sa poslužiteljem pod nazivom **simplerest.service.ts**.

Sama klasa sadrži anotaciju **@Injectable** koja omogućuje da bude definirana u konstruktoru komponente u kojoj će biti pozivana. U konstruktoru klase kao parametri se pozivaju atributi za dohvat konfiguracije (**config** : **AppConfig**) i klasu koja je zadužena za komunikaciju za poslužiteljem (**httpClient** : **HttpClient**). Pomoću konfiguracije se dohvaća temeljni link za poslužitelja, te se pohranjuje u varijablu **baseUri**. Klasa sadrži metode za CRUD operacije, slanje podatka za predikciju, dohvat filtriranih podatka i sl. Na slici 38 su vidljive metode za slanje zahtjeva za predikciju cijene kuća kao i metoda za dohvat filtriranih podataka. Metoda **postPredictionValue** kao parametar prima objekt koji sadrži vrijednosti atributa koji se koriste za predikciju (slika 40) i preko objekta **httpClient**-a post metodom šalje JSON zapis atributa potrebnih za predikciju na putanju koja je za to predviđena.

```
@Injectable({
  providedIn: 'root'
})
export class SimpleRestService {
  httpHeader = {
    headers: new HttpHeaders({
      'Content-Type': 'application/json'
    })
  }
  constructor(private httpClient: HttpClient, private config: AppConfig) { }
  private baseUri: string = this.config.setting[PropertiesKeys.PATH_API];

  public postPredictionalue(houseValues: HouseElements): Observable<any> {
    return this.httpClient.post<any>(this.baseUri + '/prediction', JSON.stringify(houseValues), this.httpHeader);
  }

  public getHousesFromDatabase(column:string, direction:string, page:number, size:number, donjaGranica:string, gornjaGranica:string) : Observable<House[]>{
    let params : HttpParams = new HttpParams();
    params = params.append('column', column);
    params = params.append('direction', direction);
    params = params.append('page', +page);
    params =params.append('size', +size);
    if(donjaGranica != ""){
      if(+donjaGranica >=0){
        params = params.append("donjaGranica", +donjaGranica);
      }
    }
    if(gornjaGranica != ""){
      if(+donjaGranica >= 0){
        params = params.append("gornjaGranica", +gornjaGranica);
      }
    }

    return this.httpClient.get<House[]>(this.baseUri, {responseType: 'json', params}).pipe(
      map((response : any) => {
          return <House[]>response.data;
      })
    );
  }
```

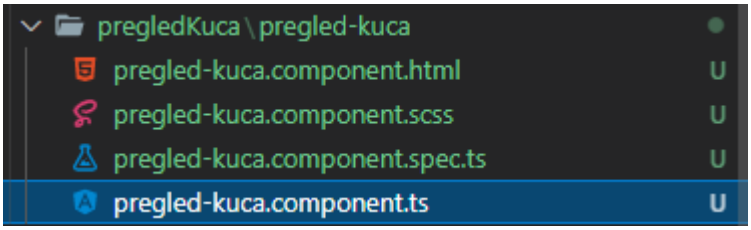
Slika 38. Dio klase namijenjene za komunikaciju sa poslužiteljem (Izrada autora)

```
export interface HouseElements{
  bedrooms: number;
  bathrooms: string;
  sqftLiving: number;
  sqftLot: number;
  floors: string;
  view: number;
  condition: number;
  yearBuilt: number;
  yearRenovated: number;
  grade: number;
  sqftAbove: number;
  sqftBasement: number;
}
```

Slika 39. Entitetska klasa zahtjeva za predikciju (Izrada autora)

Metoda **getHousesFromDatabase** kao parametre prima atribute **column** koji predstavlja naziv stupca po kojemu se vrši sortiranje, **direction** koji označava smjer sortiranja (uzlazni, silazni), **page** koji označava broj stranice za koji se dobivaju podaci, **size** koji označava broj zapisa po stranici, te **donju** i **gornju** stranicu cijena po kojoj se vrše filtriranja. Prethodno definirani atributi se dodaju objektu **params** koji je tipa **HttpParams** i zatim se šalju na poslužitelja i pomoću njih se vrše pripadajuća pretraživanja i vraćaju se pripadajući podaci.

Sljedeće što ćemo prikazati jest primjer jedne komponente i to za pregled kuća. Kao što smo i ranije spomenuli, komponenta se sastoji od 3 datoteke koji čine jednu cjelovitost (Slika 40).



Slika 40. Datoteke komponente pregled-kuca (Izrada autora)

Započet ćemo sa datotekom u kojoj je smještena poslovna logika, a to je **pregled-kuca.component.ts.** Na početku datoteke definiramo sve potrebne atribute koji će nam biti potrebni kako bi nam prikaz podataka bio funkcionalan, poput polja u koje će se kuće pohranjivati, ukupan broj zapisa, broj stranica, atribute za sortiranje, pretragu i straničenje iz **Angular Material** komponenti (Slika 41). Pošto klasa implementira događaj **OnInit**, u metodi **ngOnInit** dohvaća broj zapisa sa servera kao i podatke za početni prikaz prema inicijalnim parametrima.

```
export class PregledKucaComponent implements OnInit {

  public houses : House[] = [];
  public totalCounts : number = 0;
  public pageEvent !: PageEvent;
  public sortEvent !: Sort;
  public itemsPerPage : number = 25;
  public page : number = 1;
  public loading : boolean = false;
  public sortingColumn : string ="id";
  public sortingDirection : string ="asc";
  public donjaCijena : string = "";
  public gornjaCijena : string = "";
  displayedColumns : string [] = [ 'id', 'price', 'bedrooms', 'bathrooms', 'sqftLot', 'condition', 'grade', 'yearBuilt', 'detalji', 'brisanje', 'edit'];
  constructor(private smtp : SimpleRestService) { }

  ngOnInit(): void {
    this.loading=true;
    this.dohvatiBrojZapisa(this.gornjaCijena, this.donjaCijena);
    this.dohvatiKuceSaServera(this.sortingColumn, this.sortingDirection, this.page, this.itemsPerPage, this.donjaCijena, this.gornjaCijena);
  }
```

Slika 41. Klasa PregledKucaCompoent sa definiranim atributima i metodom ngOnInit (Izrada autora)

Korištenjem **Angular Material** komponenti za radnje poput straničenja i sortiranja uvelike nam olakšava posao jer preko njenih atributa možemo manipulirati sa određenim elementima kao što su stupac za sortiranje, smjer sortiranja, broj stranice i broja zapisa po stranici. Kod metoda za straničenje i sortiranje nalazi se na slici 42.

```
public onPaginateChange(event: PageEvent) : void{
  this.page = event.pageIndex;
  this.page += 1;
  this.itemsPerPage = event.pageSize;
  this.dohvatiKuceSaServera(this.sortingColumn, this.sortingDirection, this.page, this.itemsPerPage, this.donjaCijena, this.gornjaCijena);
}

public onSortChange(event : Sort){
  if(!event.active && event.direction === ""){
    this.sortingColumn = "id";
    this.sortingDirection = "asc";
  }
  else{
    this.sortingColumn = event.active;
    this.sortingDirection = event.direction;
}
  this.dohvatiKuceSaServera(this.sortingColumn, this.sortingDirection, this.page, this.itemsPerPage,this.donjaCijena, this.gornjaCijena );
}
```

Slika 42. Metode za straničenje i sortiranje zapisa (Izrada autora)

Kao što je vidljivo na slici 42, promjenom obilježja svake akcije (stupca za sortiranje, smjera sortiranja, broja stranice) poziva se metoda koja šalje zahtjev za dohvatom podataka prema poslužitelju sa trenutno definiranim parametrima. Osim prethodno definiranih operacija, još je moguće odabrati dodavanje novog zapisa, izvršavanje pretrage prema cijeni, pregled detalja zapisa, brisanje zapisa kao i uređivanje. Sljedeća stavka koju ćemo pobliže predočiti jest korištenje komponenti **Angular Material-a**. Na slici 43 imamo prikaz implementacije tablice, zajedno sa definiranim elementima za sortiranje, nazivima za stupce i izvor podataka. U .html dijelu komponente navodimo sve oznake koje je potrebno navesti kako bismo dobili funkcionalnu i korisničko prihvatljivu aplikaciju, dok je na slici 44 dio vezan za straničenje.

```
<mat-table  matSort matSortActive="id" matSortDirection="asc" (matSortChange)="sortEvent =$event; onSortChange($event)" [dataSource]="houses">
    <mat-header-row *matHeaderRowDef = "displayedColumns"></mat-header-row>
    <mat-row *matRowDef ="let row; columns:displayedColumns"></mat-row>
    <ng-container matColumnDef="id">
      <mat-header-cell *matHeaderCellDef mat-sort-header>Id</mat-header-cell>
      <mat-cell *matCellDef="let row">{{row.id}}</mat-cell>
    </ng-container>

    <ng-container matColumnDef="price">
      <mat-header-cell *matHeaderCellDef mat-sort-header>Cijena</mat-header-cell>
      <mat-cell *matCellDef="let row">{{row.price}}</mat-cell>
    </ng-container>
```

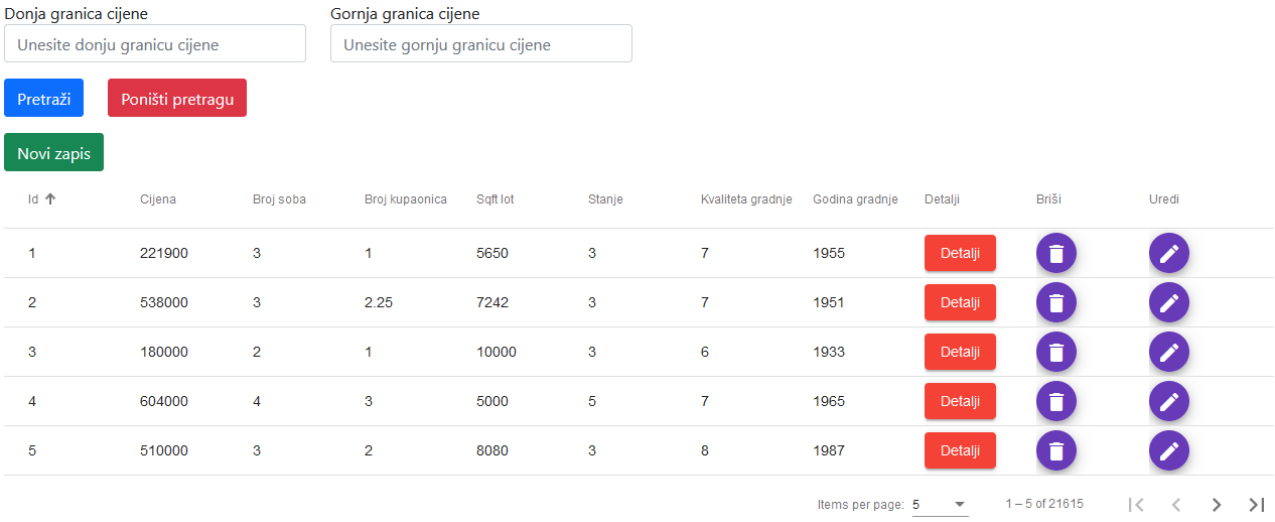
Slika 43. Definiranje tablice i elemenata za sortiranje (Izrada autora)

```
<mat-paginator
[length]="totalCounts"
[pageSize] = "itemsPerPage"
[pageSizeOptions]="[1, 5, 25, 50, 100]"
(page)="pageEvent=$event; onPaginateChange($event)"
showFirstLastButtons>
</mat-paginator>
```

Slika 44. Definiranje straničenja (Izrada autora)

5.4. Korištenje aplikacije

Zadnji dio vezan za praktični dio koji nam je preostao za objasniti jest korištenje same aplikacije. Pošto se cjelokupno rješenje sastoji od dvije aplikacije (poslužiteljske i klijentske), potrebno je pokrenuti obje aplikacije u isto vrijeme kako bi sustav funkcionirao. Nakon pokretanja obje aplikacije otvara nam se ekran za pregled osnovnih informacija o kućama iz baze podataka (Slika 45).



Slika 45. Početni ekran aplikacije (Izrada autora)

Na slici 45 je prikazan početni ekran aplikacije. Zapisi iz baze podataka su sortirani silazno prema stupcu ID, te se prikazuju osnovni podaci kao što su Cijena, Broj soba, Stanje,

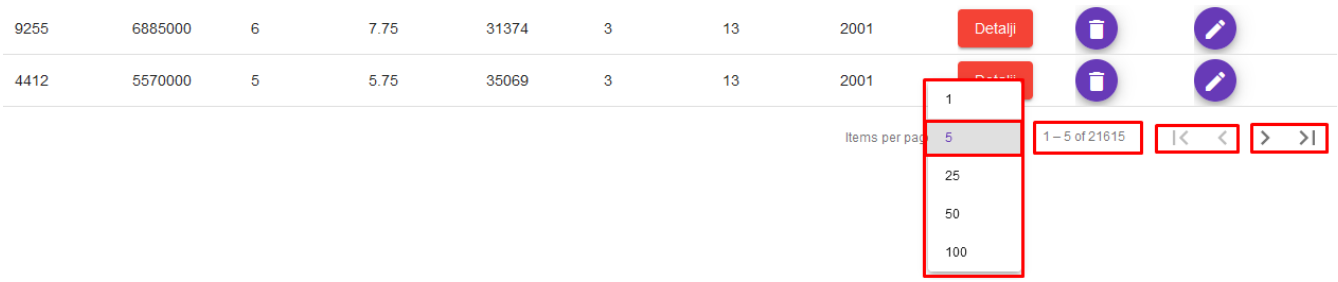
Kvaliteta gradnje, Broj kupaonica, Godina gradnje, Sqftlot kao i poveznice na detaljniji prikaz zapisa, gumb za brisanje i uređivanje zapisa. Kako bismo izvršili sortiranje prema određenom stupcu potrebno je kliknuti na naziv stupca i ovisno o smjeru sortiranja, imamo prikaz takve strelice kraj naziva (Slika 46).

Id	Cijena ↑	Broj soba	Broj kupaonica
1150	75000	1	0
15294	78000	2	1
466	80000	1	0.75
16199	81000	2	1
8275	82000	3	1

Id	Cijena ↓	Broj soba
21614	43433343	2
7253	7700000	6
3915	7062500	5
9255	6885000	6
4412	5570000	5

Slika 46. Sortiranje podataka prema atributu Cijena ulazno (lijevo) i silazno (desno) (Izrada autora)

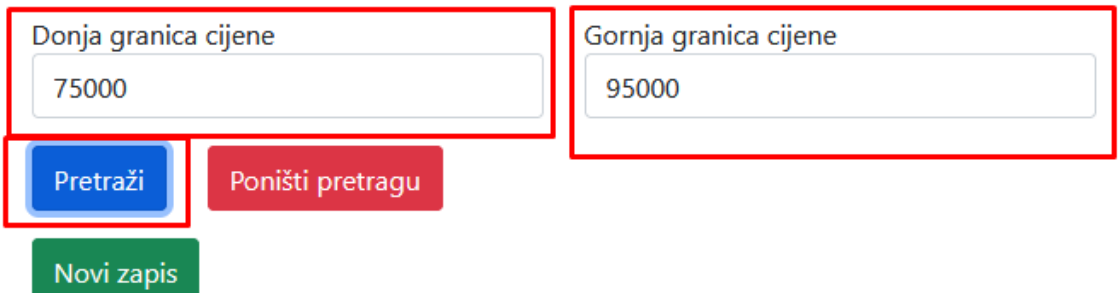
Za rad sa straničenjem pogledajmo sliku 46.



Slika 47. Dio vezan za rad sa straničenjem (Izrada autora)

U padajućem izborniku možemo vidjeti opcije za broj zapisa po stranici, dok se sa desne strane ispisuje broj trenutno ispisanih zapisa od ukupnog. Nakon toga slijede dugmad za povratak na prvu stranicu (|<) i povratak na prethodnu (<) nakon čega dolazi znak za prelazak na sljedeću stranicu (>) i prelazak na posljednju (>|).

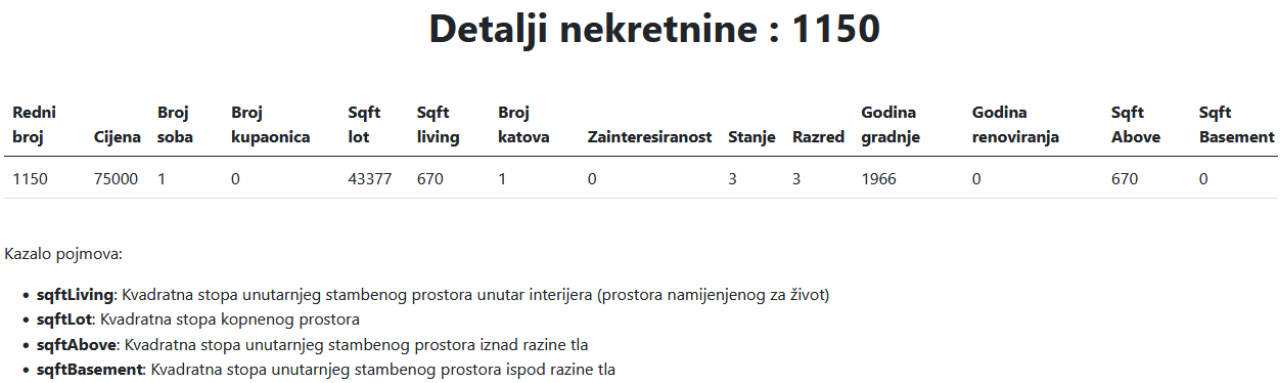
Za pretraživanje podataka prema cijeni unesemo donju i gornju granicu cijene i pritiskom na gumb za pretragu (Slika 48) dobivamo iz baze podataka podatke čija je cijena u rasponu tih cijena (Slika 49).



Slika 48. Forma za pretraživanje (Izrada autora)

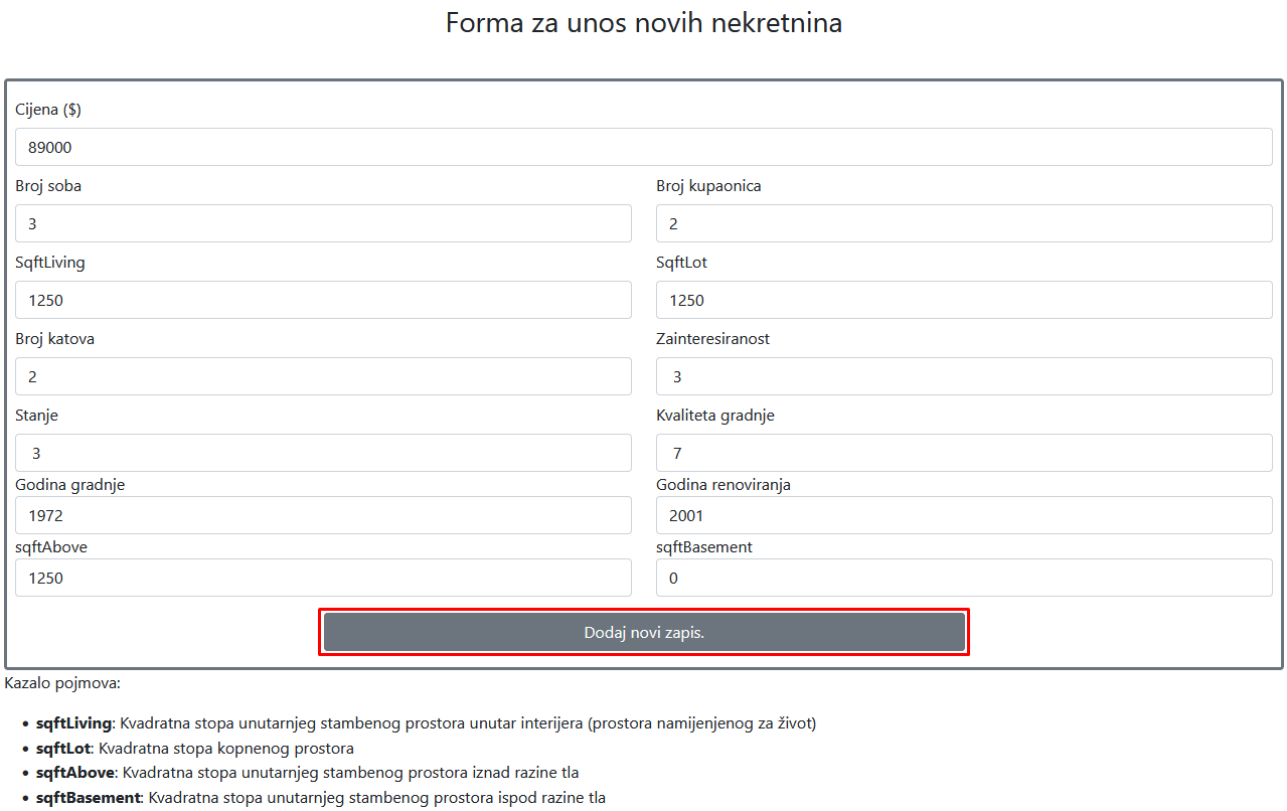
Slika 49. Rezultati pretrage po cijenama (Izrada autora)

Sljedeća funkcionalnost koju ćemo prikazati jest prikaz detalja određenog zapisa. Pritiskom na gumb „*Detalji*“ otvara nam se novi ekran sa tabličnim prikazom odabranog zapisa na kojemu se prikazuju svi atributi kuće koji su dostupni u bazi podataka.



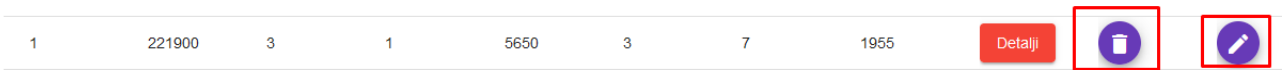
Slika 50. Prikaz detalja kuće/nekretnine (Izrada autora)

Klikom na dugme „*Novi zapis*“, koji je vidljiv preusmjerava nas se na ekran na kojoj se nalazi forma za unos nove nekretnine/kuće (Slika 51).



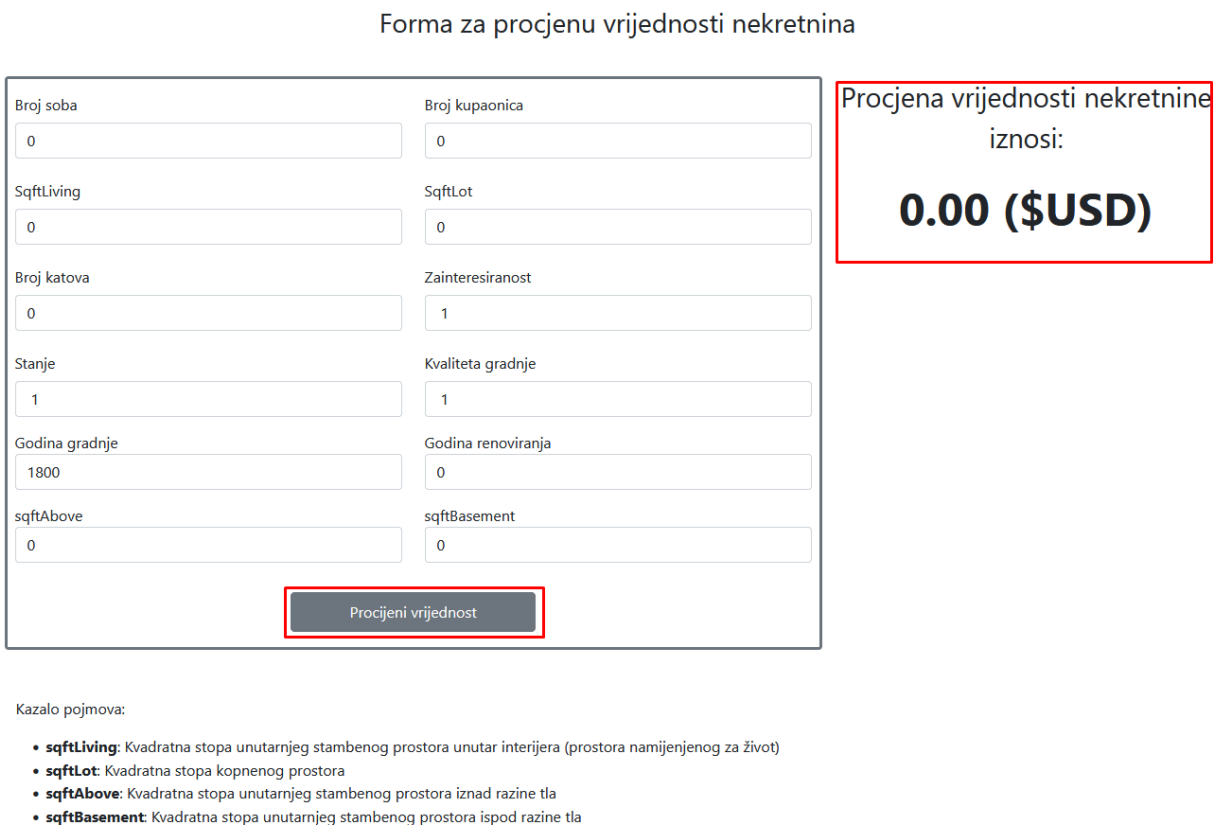
Slika 51. Izgled forme za unos nove kuće (Izrada autora)

Popunjavanjem svih atributa kliknemo na gumb „*Dodaj novi zapis*“, čime se šalje novi zahtjev na poslužitelja i novi zapis se pohranjuje u bazu podataka. Ista forma se koristi i za ažuriranje podataka kuće koja se otvara pritiskom na gumb koji sadrži ikonu olovke (Slika 52). Zadnja aktivnost koju još nismo naveli jest brisanje zapisa, a to izvršimo tako da kliknemo na gumb koji ima ikonu kante za smeće (Slika 52).



Slika 52. Prikaz ikona za ažuriranje i brisanje zapisa (Izrada autora)

Posljednja funkcionalnost ove aplikacije, k tome najvažnija, jest predikcija cijene kuće prema ulaznim obilježjima/parametrima. Izgled ekrana za procjenu cijene nekretnine vidljiva je na slici 52.

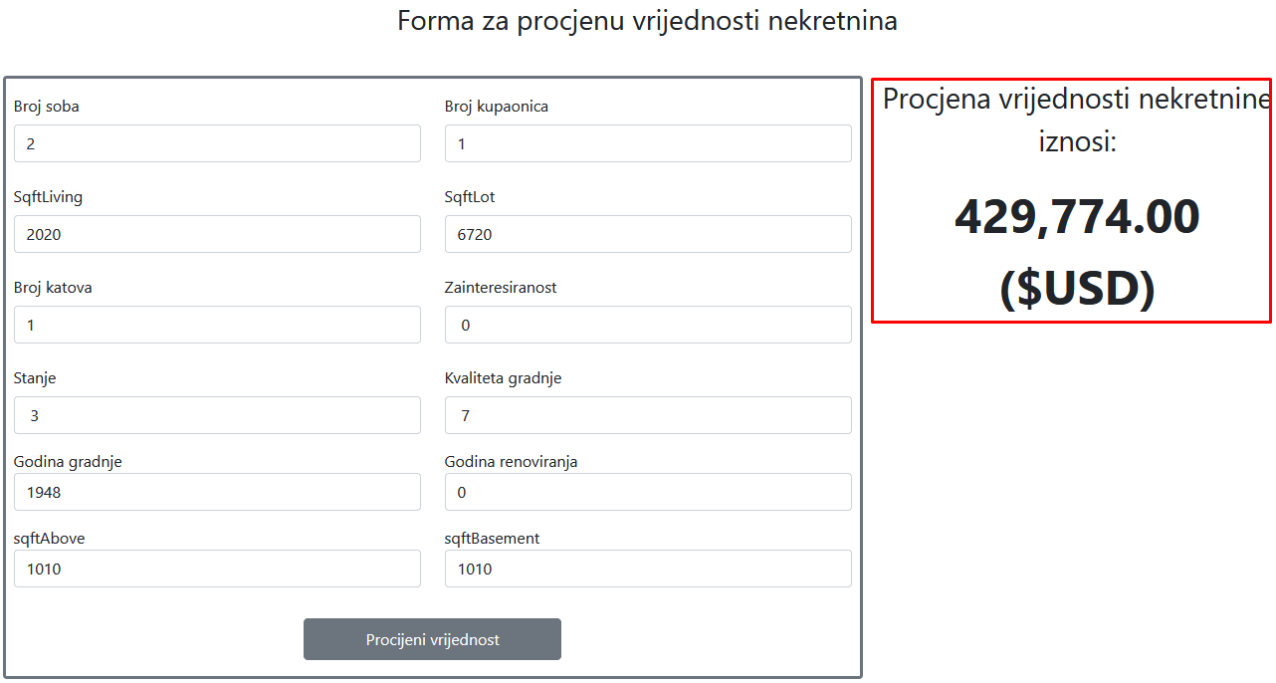


Slika 53. Ekran za procjenu cijene (Izrada autora)

Na slici 53 je vidljivo da se sa lijeve strane nalazi forma za unos obilježja kuće prema kojima se vrši predikcija, dok se sa desne strane nalazi odgovor poslužitelja sa predviđenom cijenom. Za testiranje predikcije koristit ćemo jedan redak iz baze podataka kako bismo mogli usporediti. Unijet ćemo sljedeće podatke:

- **Broj soba** : 2
- **Broj kupaonica** : 1
- **SqftLiving**: 2020
- **SqftLot**: 6720
- **Broj katova** : 1
- **Zainteresiranost** : 0
- **Stanje**: 3
- **Godina gradnje** : 1948
- **Godina renoviranja** : 0
- **Stanje gradnje** : 7
- **SqftAbove**: 1010
- **SqftBasement**: 1010

¸ I dobivamo procjenu iznosa od $429,774.00 (Slika 54), dok je cijena po kojoj je kuća prodana jednaka $355,000.00.



Slika 54. Predviđena cijena kuće za ulazne podatke (Izrada autora)

Kao što vidimo, razlika između stvarne i predviđene cijene iznosi $74, 774. Po mome mišljenju ova razlika u cijeni je prihvatljiva pošto nismo u obzir uzeli atribute kao što su datum prodaje, poštanski broj, geografska širina i duljina, kvadratura unutarnjeg stambenog prostora za najbližih 15 susjeda i kvadrature zemljišnih parcela najbližih 15 susjeda.

6. Zaključak

Tema ovog rada bilo je istražiti načine pretraživanja podataka u web aplikacijama i primijeniti neku od metoda strojnih učenja na velikome skupu podataka. Početak istraživanja ovog rada sam započeo sa osnovnim algoritmima pretraživanja poput jednostavnog slijednog pretraživanja, pa do malo kompleksnijih poput pretraživanja u dubinu i širinu nad osnovnim strukturama podataka. Nakon vrsta pretraživanja nad strukturama podataka, obradio sam pretraživanja nad bazom podataka pomoću SQL-a i konkretno pomoću programskog jezika C#. U zasebnome poglavlju sam opisao i naveo konkretne primjere *Jezika integriranog upita* (LINQ-a) pomoću kojih se mogu obrađivati i dohvaćati podaci. LINQ sam kasnije koristio i za pretraživanje u praktičnome dijelu rada. Za sam kraj u teorijskome dijelu rada sam obradio osnovne metode i koncepte strojnog učenja i njihove potkategorije čime sam proširio svoje trenutno znanje u tom području.

U praktičnome dijelu sam napravio aplikaciju sa dvije glavne funkcionalnosti, a to je predikcija cijene kuća i pretraživanje podataka iz baze podataka preko cijene sa pripadajućim sortiranjem, straničenjem i sl. Programsko rješenje je podijeljeno na dvije aplikacije, klijentsku koja je implementirana pomoću programskog okvira *Angular*-a i poslužiteljske implementirane kao ASP.NET Core (C#). Na Kaggle-u je pronađen skup podataka sa preko 21 000 zapisa koji su pomoću poslužiteljske aplikacije zapisane u bazu podataka. Pomoću podataka iz baze podataka i komponente za strojno učenje ML.NET-a kreiran je model za predikciju cijena kuća. Rad sa bazom podataka je riješen pomoću *Entity Framework Core*-a, principa *Prvo kôd – nova baza* (eng. *Code First – New Database*), dok se dohvat podataka i njihova obrada odvija pomoću ranije već spomenutog LINQ-a. Poslužiteljski dio aplikacije je podijeljen u više biblioteka klasa (engl. *Class library-a*) od kojih svaki od njih ima određenu odgovornost. Pomoću klijentske aplikacije dohvaćamo podatke sa poslužitelja i prikazujemo ih korisniku, te mu omogućujemo izvršavanje svih funkcionalnosti koje su implementirane na prihvatljiv način. Korištenjem komponente Angular Material sam si olakšao prikaz podataka tablično, kao i sortiranje po stupcima i straničenje.

Pisanjem ovog rada i implementacijom programskog rješenja unaprijedio sam svoje vještine korištenih tehnologija, kao i područja vezanog za strojno učenje. Implementacijom programskog rješenja korištenjem pogodnosti tehnologija, dokazao sam da se kompleksne funkcionalnosti ne moraju implementirati u cijelosti od strane programera, već da se većina posla odradi korištenjem spomenutih pogodnosti i mogućnosti.

Popis literature

[1] G. Bellinger, D. Castro i A.Mills „Data, information, knowledge, and wisdom“., (Castro & Mills, 2017). [Na internetu] Dostupno na: https://d1wqtxts1xzle7.cloudfront.net/64110062/Data,%20Information,%20Knowledge,%20&%20Wisdom.pdf?1596707195=&response-content-disposition=inline%3B+filename%3DData_Information_Knowledge_and_Wisdom.pdf&Expires=1616525702&Signature=PxFVaFMxVFeACc8Dyc60cgf0Og3AuUHkBshNXwboiO3lQDZicSmNWNbH8sw~bC6NySZTROxgzCXE83s18XnmFVfQtas-iMLmLQxm7X3jsZJWkLS7RLw359cWsK1rWZwr23DJp-AGjcttnCl-NfkwepDJqJwfLuEpSFuk4tU5Op5G5ndV73KtarfnDPk0AJWK~pbJRwue5RY53pW9yFIutZBsZkMbFYwhobUgZr5J-iNMiifHaTur3-CbUoEGp88s5YWaWOIrq-c1VVJEQL2tYGQDbI-KROcp2zT1YghsUalAgPhqsiBAvQBPcIG-bNRblIPLR5UiP4yVF6HELzBZoA__&Key-Pair-Id=APKAJLOHF5GGSLRBV4ZA [Pristupano: 23.03.2021]

[2] V. Čerić i M. Varga: „Informacijska tehnologija u poslovanju“, Zagreb, Element, 2004.

[3] Maleković M, Rabuzin K (2016) Uvod u baze podataka . Varaždin: Mini-Print-Logo d.o.o

[4] Rabuzin K (2011) Uvod u SQL. Čakovec: Zrinski d.d.

[5] L. D. Paulson, *Building rich web applications with Ajax*. Iz časopisa: IEEE Computer sve. 38, izd. 10 , str. 14-17. 2005. godina, [Na internetu]. Dostupno: https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=1516047 [Pristupano: 29.03.2021]

[6] Zartis Team, „3 Undeniable Characteristics of Modern Web Apps.“, [Blog post] 2020.[Na internetu], Dostupno: https://www.zartis.com/3-undeniable-characteristics-of-modern-web-apps/ [Pristupano: 29.03.2021]

[7] D. E. Knuth, „The Art of Computer Programming “, *„Sorting and Searching“*, sve. 3, izd. 2, str. 392-559, svi. 1998. godina [Na internetu]. Dostupno: https://doc.lagout.org/science/0_Computer%20Science/2_Algorithms/The%20Art%20of%20Computer%20Programming%20%28vol.%203_%20Sorting%20and%20Searching%29%20%282nd%20ed.%29%20%5BKnuth%201998-05-04%5D.pdf [Pristupano: 17.04.2021.]

[8] K. K. Pandey, N. Pradhan, „A comparison and selection on basic type of searching algorithm in data structure“, „*International Journal of Computer Science and Mobile Computing*“, sve. 3, izd. 7, str. 751 – 758, srp. 2014. godina [Na internetu]. Dostupno: https://d1wqtxts1xzle7.cloudfront.net/34306711/V3I7201499a53.pdf?1406543246=&response-content-disposition=inline%3B+filename%3DA_Comparison_and_Selection_on_Basic_Type.pdf&Ex

pires=1618678987&Signature=AXMpiRlDagCGqvMS2w-ELboTHJuATkVkEWStkuIAZOS-aEAbdTTaHvuzK1G8HPGYI9if5SQUP7sdAkc25kSeJf0vVGMhYVpdrJHZafKC-eqTKVfmOluBLnCVVH2odW7pqYIewq3Z-ZvJyvk1yUfAdT9QwkQ~VQ1cqZ6QEmj-PeKW~c2MXA022wuShJZ~FYxseG9Wtt5i8FIrmkt53adTvGSQmppXv06SNkUUYhojDX~3ZbiDnj9T6teZuLQmoLl7nR4tg-BOZovgpjZBOzXt4Wc~0oxnwocthz25trm-niATWAC1CBz2OWIfljALt6EtWMX0B0i1l7j0f9l6cD~kNA__&Key-Pair-Id=APKAJLOHF5GGSLRBV4ZA [Pristupano: 17.04.2021.]

[9] B.Miller, D. Ranum i L. College, „Sorting and Searching“ u *Problem Solving with Algorithms and Data Structures using Python*, [na Internetu]. Dostupno na: https://runestone.academy/runestone/books/published/pythonds/SortSearch/Hashing.html [Pristupano: 01.05.2021.]

[10] Depth First Search (DFS),[na Internetu] Dostupno na: https://www.hackerearth.com/practice/algorithms/graphs/depth-first-search/tutorial/, [Pristupano: 04.05.2021]

[11] Breadth First Search (BFS), [na Internetu] Dostupno na: https://www.hackerearth.com/practice/algorithms/graphs/breadth-first-search/tutorial/ [Pristupano: 05.05.2021.]

[12] Rabuzin, K. (2014) *SQL – Napredne teme*, Fakultet organizacije i informatike, Varaždin.

[13] „SELECT Statement“ [na Internetu] Dostupno na: https://dev.mysql.com/doc/refman/8.0/en/select.html [Pristupano: 09.05.2021]

[14] „SQL Joins“ [na Internetu] Dostupno na: https://www.w3schools.com/sql/sql_join.asp [Pristupano: 10.05.2021.]

[15] „ADO.NET“ [na Internetu] Dostupno na: https://docs.microsoft.com/en-us/dotnet/framework/data/adonet/ [Pristupano: 16.05.2021]

[16] „ADO.NET Overview “ [na Internetu] Dostupno na: https://docs.microsoft.com/en-us/dotnet/framework/data/adonet/ado-ne t-overview [Pristupano : 16:05.2021]

[17] „.NET Framework Data Providers“ [na Internetu] Dostupno na: https://docs.microsoft.com/en-us/dotnet/framework/data/adonet/data-providers [Pristupano: 16.05.2021]

[18] M. Mijač, I. Švogor i B. Tomaš, „*Odabrana poglavlja programskog inženjerstva*“. Sveučilište u Zagrebu, Fakultet organizacije i informatike, Varaždin 2016.

[19] „ADO.NET DataSets“ [na Internetu] Dostupno na: https://docs.microsoft.com/en-us/dotnet/framework/data/adonet/ado-net-datasets [Pristupano: 18.05.2021]

[20] „Resultsets versus Datasets (Devices)“ [na Internetu] Dostupno na: https://docs.microsoft.com/en-us/previous-versions/visualstudio/visual-studio-2008/ms180730(v=vs.90)?redirectedfrom=MSDN [Pristupano: 18.05.2021]

[21] „Language Integrated Query (LINQ) (C#)“ [na Internetu] Dostupno na : https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/linq/ [Pristupano: 22.05.2021]

[22] „Introduction to LINQ Queries (C#)“ [na Internetu] Dostupno na: https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/linq/introduction-to-linq-queries [Pristupano: 24.05.2021.]

[23] „Basic LINQ Query Operations (C#)“ [na Internetu] Dostupno na: https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/linq/basic-linq-query-operations [Pristupano: 24.05.2021.]

[24] „Data Transformations with LINQ (C#)“ [na Internetu] Dostupno na: https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/linq/data-transformations-with-linq [Pristupano 25.05.2021.]

[25] „LINQ Query Syntax“ [na Internetu] Dostupno na: https://www.tutorialsteacher.com/linq/linq-query-syntax [Pristupano 26.05.2021.]

[26] „Query Syntax and Method Syntax in LINQ (C#)“ [na Internetu] Dostupno na: https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/linq/query-syntax-and-method-syntax-in-linq

[27] A. Géron, „*Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow.*“, O’Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472, 2. izdanje, lip. 2019. [Na internetu] Dostupno na: https://www.knowledgeisle.com/wp-content/uploads/2019/12/2-Aur%C3%A9lien-G%C3%A9ron-Hands-On-Machine-Learning-with-Scikit-Learn-Keras-and-Tensorflow_-Concepts-Tools-and-Techniques-to-Build-Intelligent-Systems-O%E2%80%99Reilly-Media-2019.pdf [Pristupano: 09.06.2021]

[28] Machine Learning Tutorial [na Internetu] Dostupno na: https://www.javatpoint.com/machine-learning [Pristupano 13.06.2021.]

[29] Supervised Machine Learning [na Internetu] Dostupno na: https://www.javatpoint.com/supervised-machine-learning [Pristupano 14.06.2021.]

[30] T.O. Ayodele , „*New advances in machine learning*“, Types of machine learning algorithms, izdanje 3, str 19-48, 2010 god. [Na internetu] Dostupno na:

https://pdfs.semanticscholar.org/c4ae/802491724aee021f31f02327b9671cead3dc.pdf [Pristupano: 14.06.2021.]

[31] Regression Analysis in Machine learning, [na Internetu] Dostupno na: https://www.javatpoint.com/regression-analysis-in-machine-learning [Pristupano 15.06.2021.]

[32] Linear Regression in Machine Learning, [na Internetu] Dostupno na: https://www.javatpoint.com/linear-regression-in-machine-learning [Pristupano 15.06.2021.]

[33] R-Squared, „*What is R-Squared ?* "[na Internetu] Dostupno na: https://corporatefinanceinstitute.com/resources/knowledge/other/r-squared/ [Pristupano: 22.06.2021.]

[34] Classification Algorithm in Machine Learning [na Internetu] Dostupno na: https://www.javatpoint.com/classification-algorithm-in-machine-learning [Pristupano: 24.06.2021.]

[35] K-Nearest Neighbor(KNN) Algorithm for Machine Learning [na Internetu] Dostupno na: https://www.javatpoint.com/k-nearest-neighbor-algorithm-for-machine-learning [Pristupano: 26.06.2021.]

[36] Decision Tree Classification Algorithm [na Internetu] Dostupno na: https://www.javatpoint.com/machine-learning-decision-tree-classification-algorithm [Pristupano 27.06.2021.]

[37] Random Forest Algorithm [na Internetu] Dostupno na: https://www.javatpoint.com/machine-learning-random-forest-algorithm [Pristupano: 03.07.2021.]

[38] A.Cutler, D.Richard Cutler i J.R. Stevens, „Random forests", In Ensemble machine learning, str. 157-175, siječanj 2011, Springer, Boston, MA [na Internetu] Dostupno na: https://www.researchgate.net/file.PostFileLoader.html?id=5940c7b040485459f416c743&assetKey=AS:505020374319105@1497417648285 [Pristupano: 04.07.2021.]

[39] Naïve Bayes Classifier Algorithm [na Internetu] Dostupno na: https://www.javatpoint.com/machine-learning-naive-bayes-classifier [Pristupano: 01.07.2021.]

[40] I. Rish, „An empirical study of the naive Bayes classifier," IJCAI 2001 workshop on empirical methods in artificial intelligence, sve. 3, izd. 22, str. 41-46. kol. 2001. [Na internetu]. Dostupno na: https://www.cc.gatech.edu/~isbell/reading/papers/Rish.pdf [Pristupano: 01.07.2021.]

[41] Unsupervised Machine Learning [na Internetu] Dostupno na: https://www.javatpoint.com/unsupervised-machine-learning [Pristupano: 04.07.2021.]

[42] Clustering in Machine Learning [na Internetu] Dostupno na: https://www.javatpoint.com/clustering-in-machine-learning [Pristupano: 05.07.2021.]

[43] K-Means Clustering Algorithm [na Internetu] Dostupno na: https://www.javatpoint.com/k-means-clustering-algorithm-in-machine-learning [Pristupano: 06.07.2021.]

[44] Association Rule Learning [na Internetu] Dostupno na: https://www.javatpoint.com/association-rule-learning [Pristupano: 10.07.2021.]

[45] House Sales in King County, USA, Kaggle [na Internetu], Dostupno na: https://www.kaggle.com/harlfoxem/housesalesprediction [Pristupano 23.08.2021.]

[46] Angular 5 and ASP.NET Core , Toptal [na Internetu], Dostupno na: https://www.toptal.com/angular/angular-5-asp-net-core [Pristupano: 24.08.2021.]

[47] Entity Framework Core: DbContext [na Internetu], Dostupno na : https://www.entityframeworktutorial.net/efcore/entity-framework-core-dbcontext.aspx [Pristupano: 24.08.2021.]

[48] What is ML.NET? [na Internetu], Dostupno na: https://dotnet.microsoft.com/learn/ml-dotnet/what-is-mldotnet [Pristupano: 25.08.2021.]

Popis slika

Slika 1. Dijagram slijednog pretraživanja (Izrada autora prema [7]) 4
Slika 2. Prazna hash tablica sa 11 utora 5
Slika 3. *Pronalazak osobe pomoću hash funkcije (Izrada autora prema [7])* 7
Slika 4. *Inicijalni graf sa praznim stogom* 8
Slika 5. Stanje stoga i niza nakon posjeta vrhu B (Izrada autora) 8
Slika 6. Stanje izlaza i stoga nakon posjeta vrhovima D, G, E (Izrada autora) 8
Slika 7. Stanje stoga i izlaznog niza nakon posjeta preostalome vrhu F (Izrada autora) 9
Slika 8. Stanje reda i izlaza nakon posjeta susjeda vrha A (Izrada autora) 9
Slika 9. Stanje izlaza i reda nakon posjeta vrhova D i F (Izrada autora) 10
Slika 10. Konačno stanje nakon obilaska svih vrhova 10
Slika 11.Primjer ERA dijagram (Izrada autora) 11
Slika 12. Grafički prikaz SQL JOIN-ova (Izrada autora prema [14]) 14
Slika 13. Rad sa podacima pomoću DataSet-a (Preuzeto sa [20]) 17
Slika 14. Dijagram procesa strojnog učenja (Izrada autora prema [27]) 22
Slika 15. Predikcija modela linearnom regresijom [27] 25
Slika 16. Graf klasifikacije (Izrada autora prema [34]) 28
Slika 17. Primjer stabla odlučivanja sa čvorovima 30
Slika 18. Skica funkcioniranja algoritma Slučajne šume 32
Slika 19. Podatkovne točke prije i nakon primjene algoritma K-Sredine (Izrada autora prema [43]) 36
Slika 20. Zapisi kuća u bazi podataka sa korištenim atributima (Izrada autora) 40
Slika 21. Struktura modula poslužiteljskog dijela aplikacije (Izrada autora prema [46]) 41
Slika 22. Koraci za otvaranje Solution Manager-a 42
Slika 23. Instaliranje ML.NET-a 42
Slika 24. Dodavanje Machine Learning komponente 43
Slika 25. Odabir scenarija 43
Slika 26. Odabir okoline za treniranje 44

Slika 27. Odabir izvora podataka i ulaznih stupaca 44
Slika 28. Konfiguriranje veze za bazu podataka 45
Slika 29.Prozor za odabir treniranja podataka 46
Slika 30. Konačni rezultati treniranja 46
Slika 31. Deklariranje objekta za rad sa podacima pomoću nezavisnog ubrizgivanja (Izrada autora) 47
Slika 32. Dohvat svih podataka i ispitivanje atributa direction 47
Slika 33. Dio vezan za filtriranje podataka prema vrijednosti cijena (Izrada autora) 48
Slika 34. Klasa zadužena za rad sa bazom podataka (Izrada autora) 48
Slika 35. Klasa HouseService sa definiranim atributima i metodom za predikciju (Izrada autora) 49
Slika 36. Konfiguriranje postavki za baze podataka, servisa i konfiguriranje modela za predikciju (Izrada autora) 50
Slika 37. Dio koda klase HouseController-a (Izrada autora) 50
Slika 38. Dio klase namijenjene za komunikaciju sa poslužiteljem (Izrada autora) 51
Slika 39. Entitetska klasa zahtjeva za predikciju (Izrada autora) 52
Slika 40. Datoteke komponente pregled-kuca (Izrada autora) 52
Slika 41. Klasa PregledKucaCompoent sa definiranim atributima i metodom ngOnInit (Izrada autora) 53
Slika 42. Metode za straničenje i sortiranje zapisa (Izrada autora) 53
Slika 43. Definiranje tablice i elemenata za sortiranje (Izrada autora) 54
Slika 44. Definiranje straničenja (Izrada autora) 54
Slika 45. Početni ekran aplikacije (Izrada autora) 54
Slika 46. Sortiranje podataka prema atributu Cijena ulazno (lijevo) i silazno (desno) (Izrada autora) 55
Slika 47. Dio vezan za rad sa straničenjem (Izrada autora) 55
Slika 48. Forma za pretraživanje (Izrada autora) 56
Slika 49. Rezultati pretrage po cijenama (Izrada autora) 56
Slika 50. Prikaz detalja kuće/nekretnine (Izrada autora) 56
Slika 51. Izgled forme za unos nove kuće (Izrada autora) 57

Slika 52. Prikaz ikona za ažuriranje i brisanje zapisa (Izrada autora) 57
Slika 53. Ekran za procjenu cijene (Izrada autora)..... 58
Slika 54. Predviđena cijena kuće za ulazne podatke (Izrada autora) 59

Popis tablica

Tablica 1. *Računski prikaz dobivanja indeksa za određene algoritme (Izrada autora)* 6

Tablica 2. Osnovne klase pružatelja podataka (Izrada autora prema [17] [18, 127. str])..... 15