Desktop aplikacija za administraciju privatnog poduzeća izrađena .NET i SQL Server tehnologijama

Pavleković, Stjepan

VISOKO UČILIŠTE ALGEBRA

ZAVRŠNI RAD

Desktop aplikacija za administraciju privatnog poduzeća izrađena .NET i SQL Server tehnologijama

Stjepan Pavleković

Zagreb, veljača 2018.

Student vlastoručno potpisuje završni rad na prvoj stranici ispred predgovora s datumom i oznakom mjesta završetka rada te naznakom:

„Pod punom odgovornošću pisano potvrđujem da je ovo moj autorski rad čiji niti jedan dio nije nastao kopiranjem ili plagiranjem tuđeg sadržaja. Prilikom izrade rada koristio sam tuđe materijale navedene u popisu literature, ali nisam kopirao niti jedan njihov dio, osim citata za koje sam naveo autora i izvor, te ih jasno označio znakovima navodnika. U slučaju da se u bilo kojem trenutku dokaže suprotno, spreman sam snositi sve posljedice uključivo i poništenje javne isprave stečene dijelom i na temelju ovoga rada.“

U Zagrebu, 25.2.2018.

Predgovor

Zahvaljujem svim zaposlenicima Visokog Učilišta Algebra koji su mi kroz ove tri godine omogućili ugodno studiranje te mi pružili mnoštvo znanja za koja sam siguran da će mi u daljnjem životu uvelike pomoći. Posebno bih se htio zahvaliti mentoru Aleksandru Radovanu za to što je prihvatio mentorstvo i puno pridonio sa svim stručnim savjetima i konstruktivnim kritikama.

Isto tako bih se htio zahvaliti i svojoj obitelji koja mi je na kraju krajeva sve ovo i omogućila te mi pružala podršku tijekom cijelog studiranja.

Prilikom uvezivanja rada, Umjesto ove stranice ne zaboravite umetnuti original potvrde o prihvaćanju teme završnog rada kojeg ste preuzeli u studentskoj referadi

Sažetak

S obzirom da su u današnje vrijeme računala napredovala do stadija koji su do ne tako davno bili gotovo nezamislivi, a ista su postala globalno dostupna i veoma pristupačna, ona u kombinaciji s internetom kakav je poznat danas, postala su jedno od temeljnih sredstava u poslovnome svijetu. Svaki oblik poslovanja koje se provodi korištenjem računalnih tehnologija i interneta iza sebe ostavlja velike količine podataka. Kako se u takvim oblicima poslovanja ne rijetko pojavljuje velika količina raznih podataka, kako bi te podatke smisleno mogli koristiti, dolazi do potrebe za nekim oblikom organizacije istih. S obzirom na te potrebe, i velikom potrebom za rješenjem tog problema, danas postoji cijela znanost oko toga, a primarno se fokusira na metode zvane „Skladištenje podataka“. Skladištenje podataka spada u poslovnu inteligenciju, odnosno, ovisno o tome kako su korišteni podaci koje posjedujemo, može se analizirati tržište i ostvariti tržišna prednost. Boljim upravljanjem podacima može se ostvariti prednost nad konkurencijom, ali isto tako mogu se unaprijediti poslovni procesi unutar vlastitog poduzeća. Kako bi upravljanjem podacima u digitalnom obliku bilo moguće unaprijediti neke poslovne procese unutar poduzeća, potrebno je imati softver, odnosno aplikaciju, koja će pomoći vizualizirati te podatke i ostvariti načine upravljanja istima koji su korisniku jednostavni i intuitivni. Ovdje dolazi do pojave raznih alata koji služe kako bi korisnicima koji nemaju naprednih znanja o korištenju takvih sustava omogućili jednostavan način pristupa i upravljanja tim podacima. Upravo u takvim alatima leži tajna učestalog korištenja elektroničkog načina pristupa podacima u današnjim poslovnim sustavima, alatima koji svakome omogućuju rad s podacima.

Ključne riječi: skladištenje podataka, softver, aplikacije.

Summary

Considering the fact that today's computers have evolved to the stages that were previously thought impossible, and those very computers have now become widely available and affordable, combined with the internet as we know it today, they have become one of the core assets in the business world. Every form of business done using computer technologies and the internet produces a large amount and variety of data. Considering the amount and diversity of the data often produced by this approach, in order to utilize it and achieve benefit from it, we come across the need of some way of organization and management of that same data. Nowadays, given the need for the solution of those problems, a lot of research is being focused on the methods of "Data warehousing". Data warehousing falls under the domain of business intelligence, and what it does is help us better understand the data we possess and use it to analyze the market and achieve the advantage over our competition. To have a better data management does not only result with competitive advantages, but it also helps us achieve better performance of the core business processes within our own organization or enterprise. In order to optimize some of our business processes with the help of the digital data we possess, we would need some sort of tool, usually an application, that would allow us to visualize that data and make it simple to manage. This is where we encounter the need for the tools that can display complex sets of data, to a user who does not possess any advanced knowledge in this domain, in a much more user-friendly form, and therefore make it easier to use and manage. The secret of these tools being widely used lies in their ability to display and simplify complex sets of data to an every-day user.

Keywords: Data warehousing, application

Sadržaj

1. Uvod

Ovaj rad baziran je na razvoju programskog rješenja koje ima namjenu olakšavanja nekih osnovnih poslovnih funkcija unutar poduzeća. Fiktivno poduzeće za čije potrebe je izrađena aplikacija nema specifičnog usmjerenja, ali ima osnovne funkcije poput prodaje. S obzirom da se radi o fiktivnom poduzeću, svi podaci u svrhu demonstracije funkcionalnosti aplikacije su izmišljeni i svaka sličnost sa stvarnim podacima u potpunosti je slučajna (uključujući i sam naziv aplikacije).

Kako je krajnji cilj ovog rada izrada programskog rješenja koje bi odgovornim osobama u poduzeću omogućilo jednostavnije upravljanje podacima koje posjeduju, ideja je bila odabrati tehnologije koje imaju mogućnost što jednostavnijeg i jasnijeg prikaza podataka bez potrebe za naprednim znanjima korištenja računala i operacijskog sustava.

Izrađena aplikacija korisniku daje mogućnosti upravljanja zaposlenicima poduzeća (što uključuje izdavanje potvrda o zaposlenju, putnih naloga i slično), evidenciju izdanih računa, unos i evidenciju poslovnih troškova, dodavanje novih korisnika aplikacije te slanje obavijesti korištenjem *Gmail SMTP* (engl. *Simple Mail Transfer Protocol*) protokola. Korištenje *Gmail SMTP* protokola uklanja potrebu za korištenjem *Microsoft Outlook* programa za *email* komunikaciju. Isto tako, jedna od funkcionalnosti aplikacije je uvoz „*Excel*“ datoteka (nastalih korištenjem *Microsoft Excel* alata) bez potrebe za posjedovanjem izvornog alata. Format u kojem su zapisani podaci unutar *Excel* datoteke prilagođen je potrebama ove aplikacije s obzirom na to da je aplikacija namijenjena izmišljenom poduzeću. Prednost ovako izrađenih aplikacija je ta da se mogu prilagoditi ovisno o potrebama, pa se tako u budućnosti programski kôd može prilagoditi nekom drugom formatu podataka.

2. Tehnologije

S obzirom na to da sâm naziv „tehnologija“ u današnje vrijeme obuhvaća vrlo širok spektar pojmova, može se naglasiti kako se u ovom slučaju misli striktno na programsku podršku, preciznije na aplikacijski softver i programski softver.

Aplikacijskim softverom smatraju se programi koji izvršavaju po jednu ili više uskih zadaća. Aplikacijskim softverom smatraju se uglavnom razni programi za podršku, video igre, softver korišten u svrhu obrazovanja, i softver korišten za rad s podacima, odnosno razne baze podataka.

Programskim softverom zvani su računalni programi kojima se koriste osobe koje razvijaju softver, odnosno programeri. Česti primjeri ovakvog softvera su programi za obradu teksta, programi za uređivanje i programi za prevođenje programskog kôda (engl. *compiler* i *transpiler*). [4]

2.1. Tehnologije za izradu *desktop* aplikacije

Računalne aplikacije se u pravilu dijele na dvije glavne cjeline: *Web* aplikacije i *Desktop* aplikacije. Razlika u ova dva načina rada aplikacija leži u tome da su *web* aplikacije namijenjene za rad u internet pretraživaču, te se one rade s namjerom da se pokreću na što većem broju različitih uređaja koji imaju pristup aplikaciji preko interneta. *Web* aplikacije su uglavnom izrađene korištenjem *HTML* (engl. *Hypertext Markup Language*) jezika, *JavaScript* jezika i kombinacijom oba. Funkcionalnost *web* aplikacija bazirana je na komunikaciji klijenta i poslužitelja. *Web* aplikacije uglavnom rade tako da se klijentski program skine na korisnikov uređaj prilikom posjeta aplikaciji na Internet lokaciji, te on tada stupi u komunikaciju s poslužiteljem koji se nalazi na udaljenoj lokaciji.

Dok *web* aplikacije rade na način da im se pristupa korištenjem Internet pretraživača, i spajanjem na iste isključivo putem interneta, *desktop* aplikacije rade na drugačijem principu. *Desktop* aplikacije su napravljene na način da se pokreću samostalno na korisnikovom računalu. Za razliku od *web* aplikacija, ove aplikacije se pokreću direktno unutar operacijskog sustava računala, i bivaju pokrenute sve dok ih se ne isključi.

Desktop aplikacije ne zahtijevaju nužno povezanost s internetom. Neovisnost o internetu ih u pravilu čini puno sigurnijim od *web* aplikacija jer su puno manje izložene prijetnjama. Prednost ovakvih aplikacija u odnosu na *web* aplikacije je i to što imaju puno lakše i jeftinije održavanje pošto ne zahtijevaju poslužitelje koji moraju aplikaciju učiniti dostupnom na zahtjev velikom broju korisnika, već se jednom instalirana aplikacija može na zahtjev pouzdano koristiti na računalu na kojem se nalazi. Jedna od prednosti je i to što ne ovise o brzini i kvaliteti internetske veze, ali isto tako imaju manu što je jedna aplikacija u pravilu dostupna samo jednom korisniku istovremeno.

Ovaj rad se primarno bavi razvojem *desktop* aplikacije. S obzirom na analizu zastupljenosti operacijskih sustava u svijetu, iz tablice ispod vidljivo je kako je *Microsoft Windows* daleko najzastupljeniji. Na temelju tih podataka moguće je zaključiti kako je najsigurnije razvijati *desktop* aplikaciju upravo za taj sustav kako bi se osigurao najveći broj korisnika i najbolja pokrivenost potreba. Tablica u nastavku prikazuje zastupljenost pojedinih operativnih sustava u svijetu u 2016. godini:

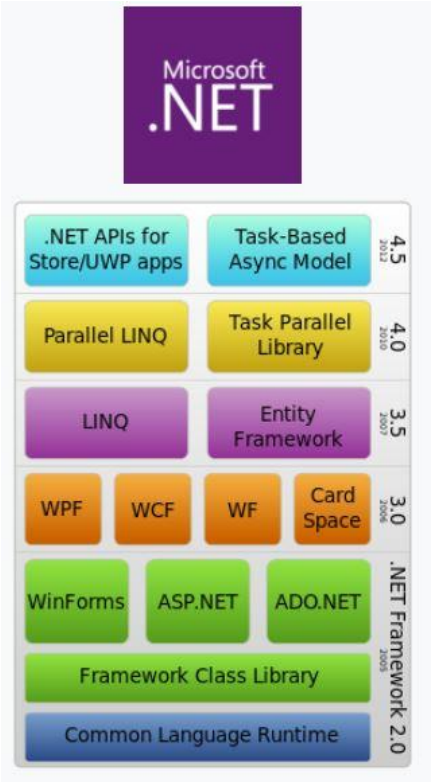
Tablica 1. Zastupljenost operacijskih sustava na računalima [9]

Year	Microsoft Windows: 52.2%					Apple: 26.2%	Linux kernel based: 21.7%				
	10	8/8.1	7	Vista	XP	macOS	Ubuntu	Debian	Mint	Fedora	Other
2016	20.8%	8.4%	22.5%	0.1%	0.4%	26.2%	12.3%	1.9%	1.7%	1.4%	4.4%

Pošto postoji niz tehnologija za izradu aplikacija za *Windows* operacijski sustav, zbog mogućnosti koje pruža, u svrhu izrade ovog projekta odabran je *Windows Forms* pristup.

Windows Forms način je izrade *desktop* aplikacija za *Windows* sustav, koji se bazira na izgradnji takozvanih „formi“. Forma je u biti samo prazno stanje, odnosno bijela prazna podloga (zadano), čije se funkcionalnosti nadodaju korištenjem niza kontrola koje su izrađene u domeni *.NET Framework-a* (okvir koji podržava niz programskih jezika i funkcionalnosti na *Windows* operacijskom sustavu). Pogonski programski jezik u pozadini *Windows Forms* je *C#*.

Ova aplikacije rađena je na sustavu s ciljanom verzijom *.NET Framework* 4.6.1.



Slika 1. .NET Framework struktura [5]

Slika iznad prikazuje komponente dodavane u *.NET* strukturu kroz godine. Vidljivo je kako je 2006. godine dodana *Windows Forms* biblioteka za izradu grafičkog sučelja. Ono što *Windows Forms* pristup čini dobrim odabirom za ovakav projekt je to što su aplikacije bazirane na formama uglavnom veoma pregledne i jednostavne. S obzirom na količinu dostupnih kontrola i komponenti koje pruža *.NET Framework*, dobrom kombinacijom i pametnim korištenjem kontrola, vrlo se brzo i jednostavno može napraviti aplikacija koja ispunjava svakakve funkcionalne zahtjeve. Prednost ovog pristupa je taj što je programeru dostupan niz kontrola koje su vrlo jednostavne za implementaciju, i dolaze s mogućnosti ponovnog korištenja. To znači da se svaka mala kontrola može koristiti bezbroj puta tijekom razvoja aplikacije.

Bez obzira na veliku raznolikost dostupnih kontrola, ponekad se pojavi potreba za funkcionalnosti koja jednostavno nije pokrivena onime što dolazi s *.NET Framework-om*. To što se može dogoditi da nema dostupnih kontrola s traženom funkcionalnosti u pravilu zapravo nije problem jer *.NET Framework* programeru pruža i mogućnost izrade potpuno novih kontrola koje on može napraviti na način da rade sve što je njemu potrebno.

2.1.1. Visual Studio

Visual Studio je proizvod korporacije *Microsoft*. *Visual Studio* je *IDE* (engl. *Integrated development environment*) koji se koristi za razvoj niza računalnih programa i komponenti. Osim samih računalnih programa, podrška različitih programskih jezika omogućava još i izradu ostalog softvera poput web stranica, web aplikacija, web servisa i mobilnih aplikacija.

Visual Studio ima integrirani uređivač kôda (engl. *Code editor*), koji je zapravo uređivač teksta s naprednim alatima i funkcionalnostima, posebno prilagođen pisanju programskog kôda. Ovaj *code editor* sadrži integriran *IntelliSense*[1] i podršku za refaktoriranje kôda. Najnovija verzija ovog alata trenutno podržava 36 programskih jezika, ali uz korištenje ekstenzija i raznih dodataka može podržati gotovo svaki programski jezik. Ugrađeni jezici uključuju C, C++, Visual Basic, C#, F#, JavaScript, XML, HTML, i druge. [7]

Visual Studio je dostupan u nekoliko različitih izdanja, od kojih je *Community* izdanje besplatno za korištenje.

Popis izdanja:

- Community
- Professional
- Enterprise
- Test Professional
- Express

Razlika u izdanjima je u raznim ekstenzijama, dodacima i podršci za razne tehnologije i programske jezike.

U izradi ovog projekta korišteno je *Enterprise* izdanje iz razloga što uz sve pogodnosti i dodatke *Professional* izdanja sadrži i novi set alata za razvoj softvera, **baza podataka**, kolaboraciju, arhitekture i testiranja.

[1] IntelliSense – Komponenta za automatsku ispunu programskog kôda i preporuke za isti

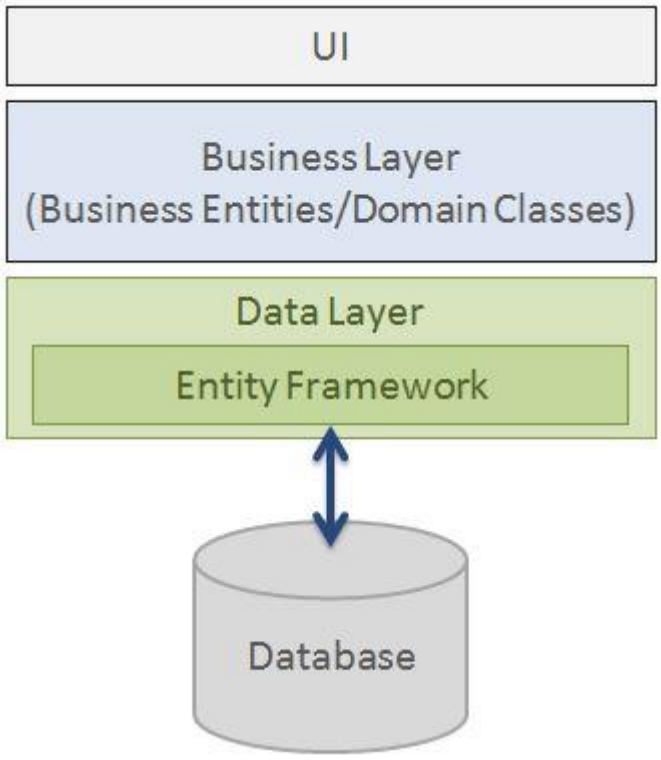
2.1.2. Entity Framework

Prije *.NET 3.5* verzije, developeri su uglavnom pisali *ADO.NET* kôd kako bi spremali ili dohvaćali podatke iz baze podataka. Postupak se sastojao od otvaranja konekcije na bazu, stvaranja odgovarajućeg *DataSet-a* kako bi se dohvatili podaci ili se isti spremili u bazu podataka, pretvaranja podataka iz *DataSet-a* u odgovarajuće objekte *.NET Framework-a* i obratno. Taj postupak je često rezultirao pogreškama.

Kako bi se otklonili problemi, *Microsoft* je izdao tzv. *Entity Framework* kako bi se postigla automatizacija svih aktivnosti povezanih s bazom podataka unutar aplikacije.

Službena definicija glasi:

„Entity Framework je O/RM (engl. Object-relational mapper) koji omogućuje .NET developerima da rade s bazom podataka koristeći .NET objekte. Eliminira potrebu za većinom kôda za pristup podacima koji bi developeri inače trebali pisati.“ [1]



Slika 2. Struktura aplikacije koja koristi *Entity Framework* [1]

Kao što prethodno navedena slika prikazuje, *Entity Framework* najbolje radi kada je stavljen između poslovne logike (engl. *Business Layer*) i baze podataka. Sprema podatke spremljene u svojstvima poslovnih entiteta i isto tako dohvaća podatke iz baze podataka i pretvara ih objekte poslovnih entiteta automatski.

Microsoft je izdao prvu verziju *Entity Framework-a* 2008. godine s *.NET Framework-om 3.5*. Od tada su izdane brojne verzije. Trenutno su aktualne dvije verzije: *EF 6* i *EF Core.* Ispod prikazana tablica opisuje razlike između *Entity Framework* verzija *EF 6* i *EF Core:*

Tablica 2. Razlike između *EF 6* i *EF Core* [1]

EF 6	EF Core
-Prvi put izdano 2008. s .NET Framework 3.5 SP1	-Prvi put izdano u Lipnju 2016. s .NET Core 1.0
-Stabilno i bogato funkcionalnostima	-Novo i u razvoju
-Isključivo za Windows	-Windows, Linux, OSX
-Radi s .NET Framework 3.5+	-Radi s .NET Framework 4.5+ i .NET Core
-Open Source	-Open Source

U izradi ove aplikacije korišten je *Code First from Database* pristup. S ovim pristupom, *Entity Framework* generira *.NET* objekte ovisno o svojstvima entiteta tablica u već postojećoj bazi podataka. Kao što je i navedeno, preduvjet za ovaj pristup je već izrađena baza podataka s postavljenim odnosima između tablica. Korištenjem ovog pristupa nije potrebno pisati procedure unutar baze podataka jer *Entity Framework* radi samostalno sa svojom virtualnom verzijom (modelom) prave baze podataka. Sve što je potrebno jest dodati konekcijski *string* u konfiguraciju aplikacije kako bi *Entity* znao gdje je baza i mogao doći do podataka.

Ispod je prikazan primjer konekcijskog *stringa*:

```
<connectionStrings>
  <add
    name="BloggingContext"
    connectionString="data source=(localdb)\v11.0;initial
catalog=Blogging;integrated
security=True;MultipleActiveResultSets=True;App=EntityFramework"
    providerName="System.Data.SqlClient" />
</connectionStrings>
```
[2]

2.2. Tehnologije za izradu baze podataka

Kao što je spomenuto u prethodnim segmentima ovoga rada, poznato je da se u današnjem poslovnom svijetu pojavljuje jako velika količina podataka. Kako bi se u računalnoj tehnologiji moglo pohranjivati podatke i raditi s istima, potrebno je imati nešto što bi to i omogućilo.

Upravo se kod te potrebe pojavljuju tzv. baze podataka (engl. *Database*). Prema nekoj okvirnoj definiciji, baze podataka su kolekcije informacija koje su organizirane na način da budu jednostavno dostupne i jednostavne za upravljanje. Kod takvih baza podataka, podaci su organizirani u redove i stupce unutar tablica. Podaci u tablicama su indeksirani kako bi se omogućilo brzo traženje željenih podataka. [3]

Računalne baze podataka uglavnom sadrže skupove podatkovnih zapisa ili datoteka, najčešće vezanih uz prodajne transakcije, proizvode, stanja na skladištu…

Upravitelj baze podataka uglavnom omogućuje korisnicima mogućnost pisanja i čitanja podataka, kreiranje izvještaja i analizu. Neke baze podataka nude usklađenost s takozvanim *ACID* (engl. *Atomicity*, *consistency*, *isolation and durability*) svojstvima kako bi garantirale dosljednost i konzistentnost podataka te uspješno izvršavanje transakcija.

Baze podataka su izvorno u 1960-ima radile na hijerarhijskom i mrežnom principu, kroz 1980-e na objektno-orijentiranom principu, a danas su znane kao *SQL* i *NoSQL* baze podataka i *cloud* (hrv. Oblak) baze podataka. [3]

Baza podataka korištena u izradi ove aplikacije spada u vrstu tzv. *Relacijskih baza podataka*. Relacijske baze podataka pojavile su se 1970. godine kao produkt korporacije *IBM*. Relacijske baze podataka načinjene su od seta tablica s podacima koji su smješteni u predefinirane kategorije. Svaka tablica ima barem jednu kategoriju u stupcu a svaki redak ima jednu instancu podatka koji spada u kategoriju definiranu stupcem.

SQL (engl. *Structured Query Language*) je standardno programsko sučelje za relacijske baze podataka. Prednost ovakvih baza podataka je to što je jednostavno dodati novu kategoriju podataka nakon inicijalnog stvaranja baze podataka bez potrebe za modificiranjem svih postojećih aplikacija.

2.2.1. SQL

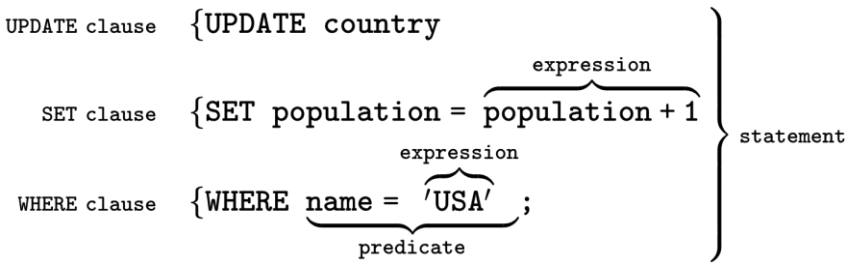
SQL (engl. „*Sequel*“ – *Structured Query Language*) je specifičan jezik korišten u programiranju, a dizajniran je kako bi upravljao podacima pohranjenim u relacijskim bazama podataka. U odnosu na stare pristupe, *SQL* nudi dvije glavne prednosti: prvo, predstavlja koncept pristupa mnoštvu zapisa kroz jednu naredbu; i drugo, eliminira potrebu za definiranjem načina na koji se dolazi do zapisa, s indeksom ili bez indeksa. [8]

SQL je bio jedan od prvih komercijalnih jezika za relacijski model Edgara F. Codda. Bez obzira na to što ne poštuje u potpunosti relacijski model koji je Codd opisao, *SQL* je postao globalno najkorišteniji jezik za baze podataka.

Sintaksa *SQL* jezika podijeljena je u nekoliko elemenata:

- *Klauzule*, koje su sastavne komponente upita
- *Izraze*, koji mogu proizvesti ili skalarne vrijednosti, ili tablice koje se sastoje od stupaca i redaka s podacima.
- *Predikati*, koji specificiraju uvjete koji mogu biti procijenjeni prema „*SQL three-valued*“ logici (*true*, *false*, *unknown*)
- *Upiti*, koji dohvaćaju podatke ovisno o zadanim kriterijima. Jedan od najbitnijih elemenata *SQL*-a.

Na idućoj slici prikazano je kako elementi *SQL-a* čine upit:



Slika 3. Grafikon prikaza elemenata SQL-a koji čine upit [8]

SQL je prihvaćen kao standard od strane Američkog Instituta za Standardizaciju 1986. godine kao *SQL-86* i Internacionalne Organizacije za Standardizaciju 1987. godine. Standard je uglavnom označen s uzorkom: *ISO/IEC 9075*. [8]

2.2.2. Microsoft SQL Server

Microsoft SQL Server je sustav za upravljanje relacijskim bazama podataka kojeg je razvio *Microsoft*. *SQL Server* je softverski produkt s primarnom funkcijom spremanja i dohvaćanja podataka na zahtjev drugih povezanih aplikacija, koje se mogu pokretati ili na lokalnom računalu ili na udaljenoj mrežnoj lokaciji. [6]

Microsoft je izdao nekoliko različitih izdanja *Microsoft SQL Server-a* koje imaju različite grupe ciljanih korisnika i vrste posla koje variraju od malih aplikacija do velikih aplikacija baziranih na Internet komunikaciji s velikim brojem korisnika.

Prvo izdanje *Microsoft SQL Server-a* pojavilo se kao *Microsoft SQL Server 1.0*, 16 bitni server za *OS/2*[2] operacijske sustave 1989. godine, te se i dalje razvija.

Glavna izdanja *Microsoft SQL Server-a* sastoje se od:

- *Enterprise*, izdanje koje uključuje i glavne funkcionalnost i dodatne servise, te niz alata za stvaranje i upravljanje *SQL Server Klastera*. Može upravljati bazama podataka veličine do 524 Petabajta-a[3], adresirati 12 Terabajta[4] memorije, i podržava 640 logičkih procesorskih jezgri.
- *Standard*, uključuje glavne funkcionalnost i zasebne servise. Od *Enterprise* izdanja se razlikuje u tome što podržava manje aktivnih instanci i ne sadržava neke funkcije.
- *Web*, *Low-TCO* (engl. *Low Total cost of ownership*) izdanje koje je namijenjeno za *Web-hosting*.
- *Business Intelligence*, sadrži niz alata za podršku poslovnoj inteligenciji
- *Workgroup*, sadrži glavne funkcionalnosti, ali ne sadrži nikakve dodatne servise. Ne koristi se od izdanja 2012.
- *Express*, je „laka“ i besplatna verzija *SQL Server-a*, koja uključuje glavne funkcionalnosti. Nema ograničenje na broj baza podataka ili korisnika, ali ima ograničenje na korištenje jednog procesora, 1 GB radne memorije i 10 GB memorije za pohranu podataka. [6]

Od ostalih izdanja bitno je spomenuti *Azure*. *Azure SQL Baza Podataka* je verzija *SQL Server-a* koja se temelji na *cloud* principu rada. Jedna je od zastupljenijih opcija po pitanju pohrane podataka u domeni Internet aplikacija.

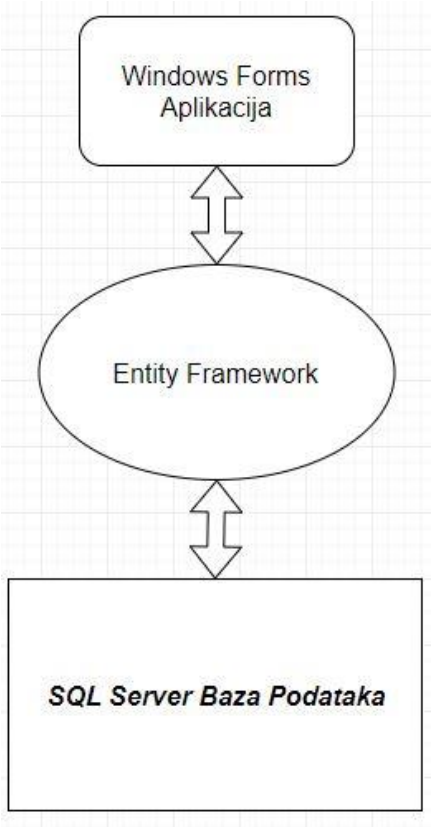
[2] https://en.wikipedia.org/wiki/OS/2
[3] Petabyte - 10^{15} byte
[4] Terabyte - 10^{12} byte

3. Arhitektura

Aplikacija na kojoj se temelji ovaj završni rad sastoji se od tri glavne komponente:

1. *Windows Forms* Aplikacija
2. *Entity Framework*
3. *SQL Server* Baza Podataka

Svaka od navedenih komponenti ima određenu funkciju. *Windows Forms* aplikacija glavna je komponenta za prikaz podataka korisniku i unos novih podataka od istog. Svi podaci koji se ovdje prikazuju prolaze kroz *Entity Framework* komponentu. *Entity Framework* ima ulogu upravljanja podacima. Ova komponenta generira model prema bazi podataka koja je korištena i referencirana. *Entity* radi tako da upravlja podacima unutar modela, što po završetku operacije sprema u stvarnu bazu podataka. Najniža komponenta u arhitekturi je baza podataka. Korisnik ni u kojem trenutku ne radi s bazom direktno, već svu komunikaciju s istom obavlja preko *Entity Framework-a.* Baza podataka sadržava tablice na temelju čijih su stupaca generirani *.NET* objekti unutar aplikacije. Ispod prikazana slika opisuje oblik arhitekture ove aplikacije sastavljene od prethodno navedene i opisane tri glave komponente:



Slika 4. Dijagramski prikaz arhitekture rješenja

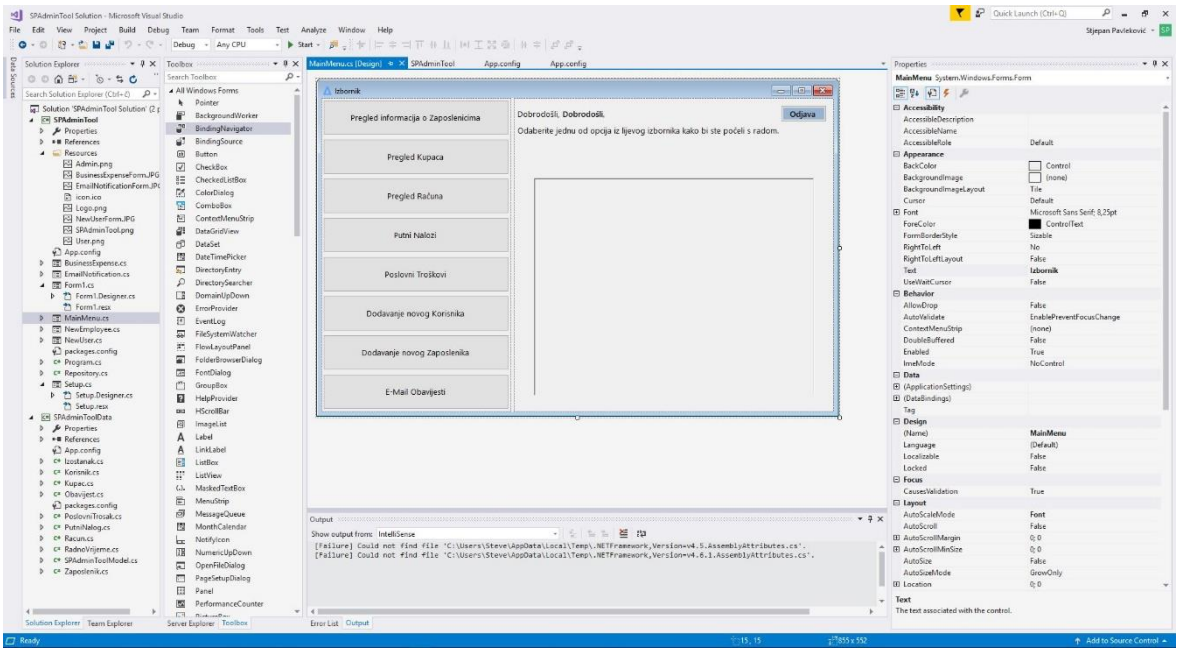
3.1. Windows Forms

Windows Forms aplikacija prvi je od tri glavna elementa ovog rješenja. Ova aplikacija vrši glavnu ulogu za obavljanje interakcije između korisnika i baze podataka. Komponenta se sastoji od ukupno deset glavnih formi, od kojih njih osam ima funkcionalnosti vezane uz dodavanje, prikaz, i ažuriranje podataka unutar baze podataka.

Kod inicijalnog pokretanja aplikacije, umjesto regularne forme za prijavu u sustav, korisniku se prikazuje forma koja ga vodi kroz postavljanje administratorskog računa. Nakon kreiranja računa, moguće se prijaviti u sustav s novokreiranim podacima, te na glavnom izborniku odabrati jednu od funkcionalnih formi:

1. Pregled informacija o Zaposlenicima
2. Pregled Kupaca
3. Pregled Računa
4. Putni Nalozi
5. Poslovni Troškovi
6. Dodavanje novog Korisnika
7. Dodavanje novog Zaposlenika
8. *Email* Obavijesti

Svaka od navedenih formi detaljno je opisana u nadolazećim poglavljima. Slika u nastavku prikazuje izgled sučelja za izradu *Windows Forms* aplikacija korištenjem *Visual Studio* alata:



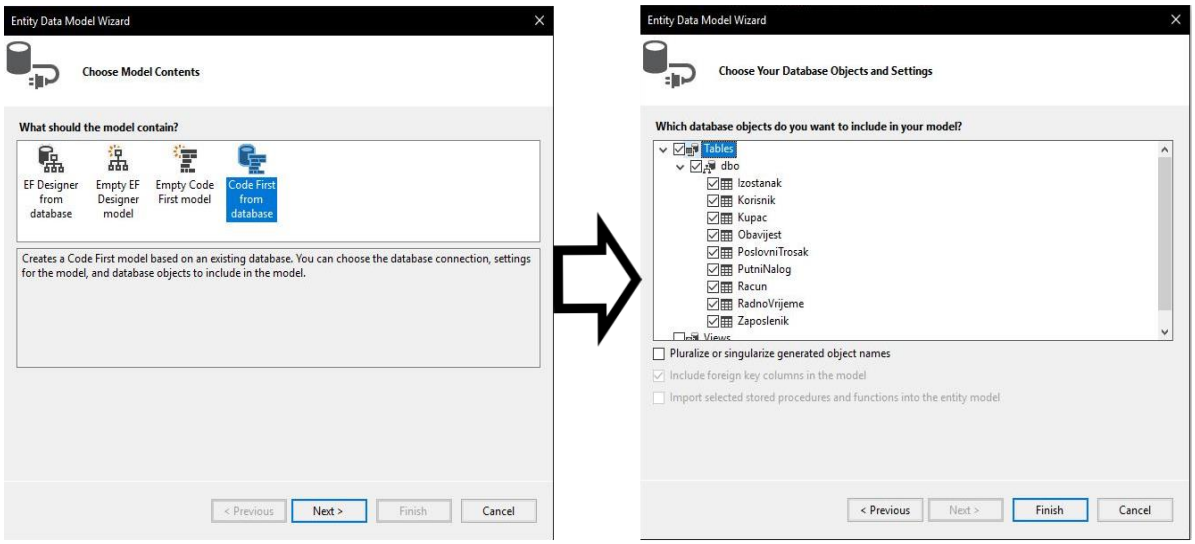
Slika 5. Prikaz sučelja za izradu *Windows Forms* aplikacije

3.2. Entity Framework Sloj

U svrhu bolje organizacije programske strukture, u *Visual Studio* projekt dodan je novi element: *Class Library*[5]. U ovaj *Class Library* dodan je *ADO.NET Entity Table Model*.

Ovaj model temeljna je stavka za izgradnju *Entity Framework* komponente unutar aplikacije. Prilikom dodavanja tog objekta, korisnika se navodi kroz „čarobnjak“ gdje se može odabrati baza podataka na temelju koje on želi kreirati *Entity* model.

Po odabiru tražene baze podataka, potrebno je unijeti sve pristupne informacije (instancu servera na kojem se baza nalazi, te lozinku koja je potrebna za prijavu na tu određenu instancu). Nakon što korisnik unese ispravne podatke, ponuđena mu je lista tablica koje sadrži odabrana baza podataka, i tada korisnik može odabrati koje tablice želi uključiti u *Entity* model. Po završetku odabira, čarobnjak generira *.NET* objekte koji sadrže svojstva koja odgovaraju tablicama iz baze podataka. Ti objekti tada postaju dostupni za korištenje u ostatku aplikacije jednom kada su referencirani. Osim samih objekata, generiran je i spomenuti *model*. Taj model je ključni elementi za rad s podacima, jer upravo on sadrži sve setove podataka s kojima će se dalje raditi. Kao što je prethodno spomenuto, ovaj postupak odnosi se isključivo na *Code first from database* pristup izrade *Entity* modela. Ispod navedena slika prikazuje izgled „čarobnjaka“ za dodavanje *Entity* modela u aplikaciju.



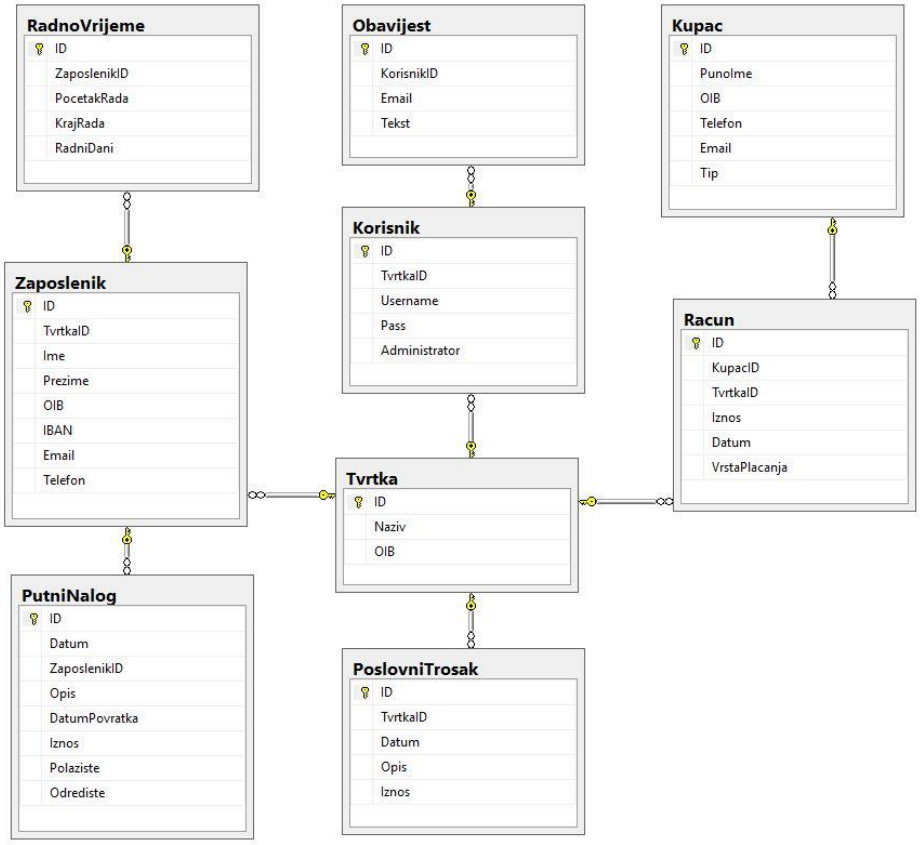
Slika 6. Proces dodavanja *Entity* modela.

[5] Class Library – Temelj .NET Frameworka. Biblioteka u kojoj se nalaze programske klase, sučelja i sl.

3.3. SQL Server baza podataka

Kao što je prethodno navedeno, s obzirom na potrebe, u izradi ove aplikacije korištena je *SQL Server* baza podataka. Struktura u kojoj se pohranjuju podaci u ovakvoj relacijskoj bazi podataka najbolje odgovara iz razloga što se može doći do raznih informacija i znanja na temelju određenih entiteta. Primjerice, odabirom nekog određenog kupca može se doći do popisa računa izdanih tom kupcu.

Trenutna inačica baze podataka nazvana je „SPAdminTool“ u skladu s aplikacijom, te se ona sastoji od devet tablica, od kojih su samo neke povezane s obzirom na funkcionalnost aplikacije. Svaka od tablica kao *primary key*[6] ima polje *ID*, te se ostale tablice međusobno povezuju upravo prema tom polju. Način na koji su određene tablice povezane može se vidjeti na ispod prikazanoj slici. Tablice koje nemaju veza s drugim tablicama nemaju ih iz razloga što trenutni funkcionalni zahtjevi to nisu zahtijevali. Uvođenjem novih zahtjeva to se može mijenjati.



Slika 7. Dijagram baze podataka

[6] Primary key – Poseban stupac koji predstavlja jedinstven zapis unutar relacijske baze podataka.

4. Forme

U poglavlju koje slijedi, detaljno je opisana svaka forma unutar aplikacije. Opisani su razlozi za odabirom pojedinih kontrola, način na koji se radi s podacima, te način na koji određene forme „komuniciraju“ s ostalim formama.

Svaki *.NET* objekt stvoren prema uzorku iz baze podataka ima pripadajuću formu. Svaka od tih formi sastoji se od niza kontrola koje odgovaraju pojedinim svojstvima objekata za koje je predviđena mogućnost uređivanja ili ručnog unosa prilikom kreiranja navedenog objekta.

Kako bi se eliminirala ili barem minimizirala mogućnost unosa pogrešnih vrijednosti za pojedina svojstva objekata, svaka forma u kojoj je predviđen rad s podacima ima ručno napravljene elemente za provjeru i validaciju unesenih podataka. Greške i neispravnosti u formama korisniku su naglašene korištenjem *MessageBox* objekta umjesto *ErrorProvider* kontrola jer su one na temelju ispitivanja par osoba koje nemaju iskustva s računalnom tehnologijom ispale intuitivnije i jasnije. Slika u nastavku prikazuje primjer *MessageBox* objekta:

Slika 8. Stil prikazivanja pogrešaka kod korištenja aplikacije

Osim generičkih *if/else* blokova za provjeru ispravnosti unesenih podataka, kroz velik dio aplikacije moguće je vidjeti tzv. *try i catch* blokove. Ti blokovi u programskom kôdu služe kako bi osigurali da se aplikacija pokreće nesmetano čak i onda kada se u pokretanju kôda dogodi iznimka (engl. *Exception*) koja nije bila predviđena. U *try* blok uglavi se sav kôd koji se želi pokrenuti, a u *catch* bloku se može napisati zaseban kôd koji se želi odraditi u slučaju da se u glavnom dijelu dogodi neka iznimka. Korištenje ovih blokova spada u tzv. dobru praksu u programiranju iz razloga što osigurava aplikaciju od prekida rada i odvijanja nekih neželjenih ili nepredviđenih akcija i postupaka.

4.1. Forma za prijavu u sustav

S obzirom na to da su glavne funkcionalnosti ove aplikacije vezane uz rad s podacima iz baze podataka, poželjno je u sustav uvesti neku vrstu kontrole pristupa. Uvođenje potrebe za korištenjem korisničkih računa ne samo da osigurava sustav od upada neželjenih stranki, već u aplikaciju uvodi mogućnost praćenja aktivnosti korisnika.

U ovoj aplikaciji, prijava u sustav ima demonstrativnu ulogu kako bi se istaknula potreba za takvom funkcionalnosti. Do te potrebe dolazi kod nekih funkcionalnosti koje mogu uređivati ili brisati osjetljive podatke. Kako nije poželjno da baš svaki korisnik ima jednaku razinu pristupa svim funkcijama i podacima kao i administrator, njegove ovlasti se mogu ograničiti.

U ovom slučaju, svakom korisniku aplikacije prilikom kreiranja korisničkog računa dodijeljen je atribut „Administrator“. Stanje tog atributa određuje koje će se funkcionalnosti korisniku pojaviti jednom kada se prijavi u sustav.

Kako bi se osiguralo da u bazi postoji barem jedan korisnički račun, prilikom inicijalnog pokretanja aplikacije pokrene se upit u bazu koji provjeri ima li korisničkih računa. Te funkcije prikazane su kroz isječke u ispod navedenoj tablici:

Tablica 3. Provjera dostupnih korisničkih računa

Isječak 1. Dohvaćanje korisnika iz baze	Isječak 2. Provjera dostupnih računa

`public void RefreshUsers()`
`{`
`users = Repository.GetUsers();`
`}`
	`private void Form1_Shown(object sender, EventArgs e)`
	`{`
	`if(users.Count < 1)`
	`{`
	`this.Hide();`
	`Setup setup = new Setup(this, korisnik);`
	`setup.Show();`
	`}`
	`}`

Ukoliko lista[7] korisničkih računa sadrži barem jednu stavku, korisniku se pokazuje forma za prijavu. Ako je lista prazna, korisnika se preusmjerava na formu za kreiranje administratorskog računa gdje će kroz niz koraka korisnik moći izraditi isti. Taj korisnički račun tada postaje dostupan za korištenje. Ta forma je opisana u idućem poglavlju.

[7] Lista – Objekt u kojem su pohranjene varijable u specifičnom redoslijedu

Jednom kada je kreiran korisnički račun, i prethodno navedena provjera ustanovi da lista korisničkih računa više nije prazna, korisniku se umjesto forme za kreiranje računa pojavljuje forma za prijavu u sustav.

Forma za prijavu sastoji se od nekoliko kontrola:

- *TextBox* za unos korisničkog imena i lozinke
- *Button* za prijavu

Slika u nastavku prikazuje izgled forme za prijavu u aplikaciju:

Slika 9. Forma za prijavu u sustav

Jednom kada korisnik ispuni obvezna polja za korisničko ime i lozinku, klikom na gumb za prijavu pokreće se programski blok koji provjerava ispravnost unesenih korisničkih podataka. Klikom na gumb, može se ostvariti jedan od dva predviđena ishoda:

1. **Uneseni korisnički podaci su neispravni** – Slučaj se dogodi onda kada funkcija za provjeru korisničkih podataka u bazi podataka ne pronađe korisnički račun kojem odgovaraju uneseni podaci. Korisniku se tada na formi prikaže odgovarajuća poruka te se polja za unos očiste.
   Ista poruka korisniku se prikaže i onda kada polja za ime i lozinku ostavi neispunjena s obzirom da je kôd napisan tako da ne dozvoljava unos tzv. „praznih stringova“.

2. **Uneseni su podaci koji se podudaraju s računom u bazi** – Korisnik je unio ime i lozinku za koje je funkcija ustanovila da odgovaraju jednom od računa u bazi podataka. Po završetku provjere, na razini glavne forme kreira se instanca objekta *Korisnik* kojem se postavljaju vrijednosti učitanog objekta iz baze. Od tada pa do završetka rada (odjave) za trenutno prijavljenog korisnika se koriste ti podaci i za tog korisnika će se voditi evidencija ondje gdje je to potrebno.

Ispod je prikazana funkcija za provjeru ispravnosti unesenih korisničkih podataka, gdje „*temp*“ predstavlja trenutnu instancu objekta *Korisnik* koja je kreirana na razini forme, i kojoj će se dodijeliti uneseni podaci u slučaju da su ispravni i da u bazi podataka postoji kombinacija istih:

```
bool exists = false;
foreach (Korisnik k in users)
{
    if (k.Username == temp.Username && k.Pass == temp.Pass)
    {
        exists = true;
        temp = k;
    }
}
```

Isječak 3. Provjera ispravnosti unesenih korisničkih podataka

Navedeno rješenje može se riješiti na razne načine, ali s obzirom na potrebe aplikacije, ovo rješenje je odabrano kao zadovoljavajuće.

Jednom kada su korisnički podaci validni, i korisnik je prijavljen u sustav, istog se preusmjerava na glavnu formu za rad, *MainMenu*. O navedenoj formi više je opisano u nastavku.

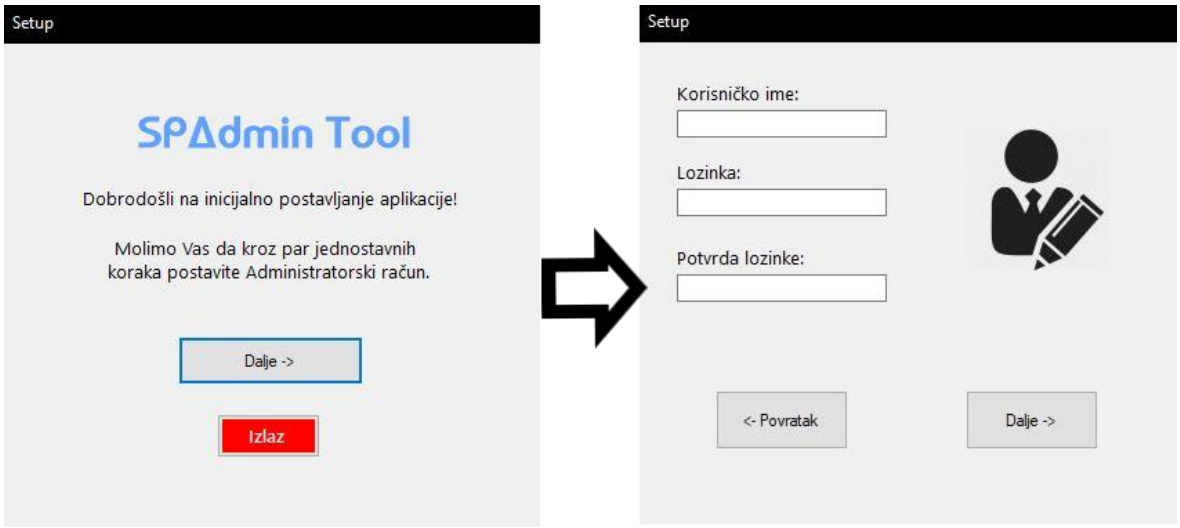
4.2. Forma za kreiranje administratorskog računa

Kao što je i opisano u prethodnom poglavlju, u slučaju da funkcija za provjeru dostupnih računa rezultira praznom listom *Korisnika*, trenutnog korisnika se umjesto na formu za prijavu u sustav preusmjerava na formu za inicijalno postavljanje administratorskog računa.

Ovaj korak zamišljen je s namjerom da vlasnik sustava, odnosno osoba za čije se potrebe izrađuje ova aplikacija, može kreirati račun sa svim ovlastima prilikom inicijalnog pokretanja aplikacije kako bi mu u daljnjem korištenju aplikacije sve funkcionalnosti bile dostupne na korištenje.

Prilikom dolaska na navedenu formu, korisniku je prikazana poruka s uputama i gumbi za daljnje korake i izlaz. Pritiskom na prvi od dva gumba, korisnika se šalje na idući korak na kojem on može odabrati korisničko ime i lozinku za svoj administratorski račun. U tom koraku na formi su prikazane kontrole:

- *TextBox* za korisničko ime, lozinku i potvrdu lozinke
- Gumbi za povratak i kreiranje računa



Slika 10. Koraci za kreiranje inicijalnog računa

Slika iznad prikazuje formu za kreiranje računa, čija polja za unos imena i lozinke dozvoljavaju unos alfanumeričkih i specijalnih znakova, te duljinu znakovnog niza (engl. *String*) duljine 10 i 14 znakova. Nema posebnih razloga za odabrane dozvoljene duljine korisničkog imena i lozinke već je stavljeno iz razloga da odgovaraju nekom globalnom standardu te da se održi čistina i jednostavnost podataka.

Kako bi korisnik mogao nastaviti na idući korak, mora unijeti sve navedene vrijednosti, te je obavezno da se lozinka i potvrda lozinke podudaraju. Ukoliko je korisnik unio lozinke koje se ne podudaraju, ili nije ispunio sva polja, neće mu biti omogućeno napredovanje na idući korak već će biti prisiljen unijeti ispravne vrijednosti. Ako su podaci koje je korisnik unio ispravni i odgovaraju zadanim uvjetima, pokreće se funkcija za kreiranje korisničkog računa (Isječak 4). Ukoliko postoji ispravna konekcija na bazu podataka, i svi su podaci ispravni, korisnički račun trebao bi biti uspješno kreiran, te se korisnika šalje na zadnji korak s kojeg on napokon može pristupiti formi za prijavu u sustav.

```
try
{
    Repository.CreateUser(korisnik);
    lblUser.Text = korisnik.Username;
    lblPass.Text = korisnik.Pass;
    step2.Visible = true;
} catch
{
    MessageBox.Show(e, MessageBoxButtons.OK,  MessageBoxIcon.Error);
}
```

Isječak 4. Funkcija za kreiranje inicijalnog korisničkog računa

Ukoliko se prilikom kreiranja računa dogodila neka pogreška, korisnika se o tome obavještava odgovarajućom porukom, te se postupak terminira.

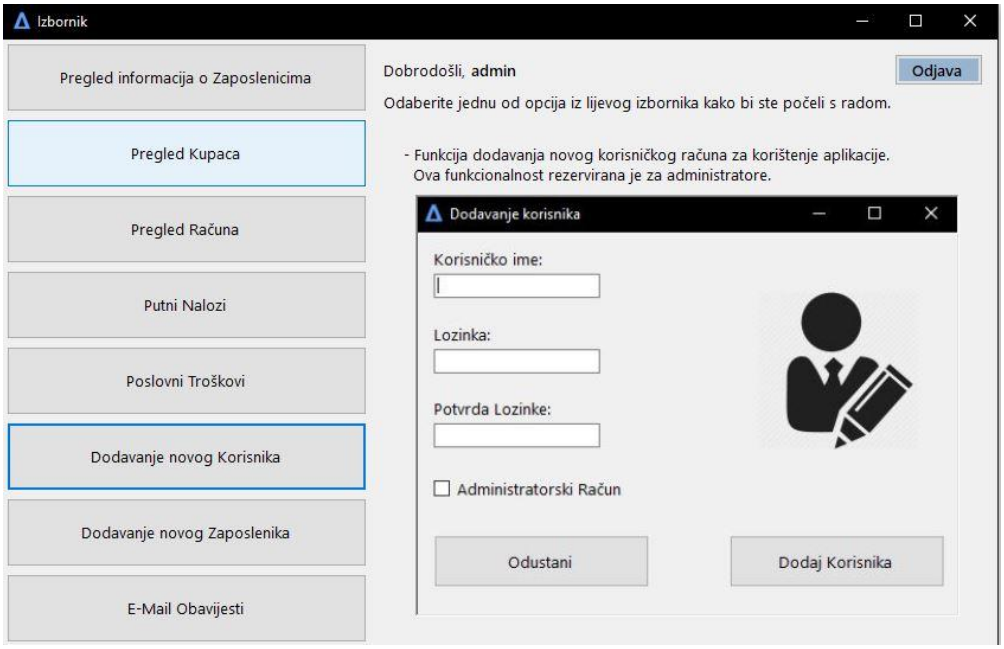
4.3. Glavni izbornik

Jednom kada je korisnik uspješno prijavljen u sustav, njemu se prikazuje glavna forma za izbor ostalih funkcionalnosti. Ova forma sastoji se od niza opcija za prikaz drugih formi, okvira za slike koji prikazuje izgled forme čije informacije korisnik trenutno pregledava, tekstualnog polja za prikaz informacija o formama, te gumba za odjavu iz sustava i povratak na formu za prijavu (Slika 11).

Ova forma ima dva moguća prikaza koji se mijenjaju ovisno o tome ima li trenutno prijavljeni korisnik status administratora ili ne. Ukoliko se uspostavi da je prijavljeni korisnik administrator, njemu se osim osnovnih kontrola prikazuju još tri dodatne kontrole koje omogućuju pokretanje i prikaz formi za:

- Dodavanje novog korisnika
- Dodavanje novog Zaposlenika
- Slanje *email* obavijesti

Ako trenutni korisnik nije administrator te kontrole neće mu biti prikazane.



Slika 11. Prikaz glavnog izbornika

Instanca objekta Korisnik koja se nalazi na razini ove forme prosljeđuje se u ostale forme u kojima ima potrebe za evidencijom akcija trenutnog korisnika ili forme koje zahtijevaju administratorska prava za određene funkcionalnosti.

Slika koju okvir na formi prikazuje mijenja se ovisno o tome iznad kojeg gumba se trenutno nalazi pokazivač miša. Slika se ne mijenja sve dok se na jednom od gumba ne okine *MouseHover* događaj. Tada se slika okvira postavlja ovisno o tome na kojem se gumbu okinuo taj isti događaj.

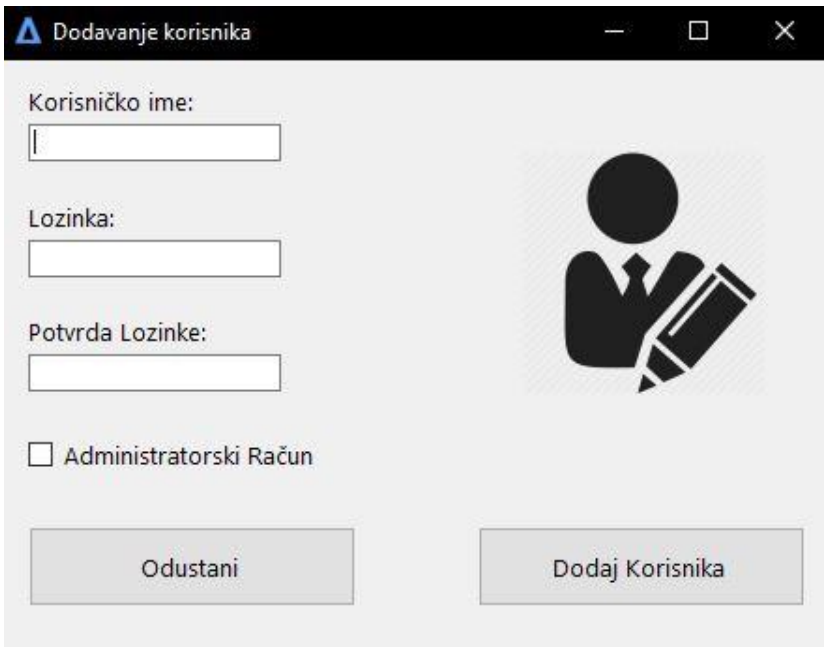
```
private void btnNoviKorisnik_MouseHover(object sender, EventArgs e)
        {
            pbPreview.Image =
Image.FromFile("../..Resources/NewUserForm.JPG");
            lblOpis.Text = "- Funkcija dodavanja novog korisničkog računa za
korištenje aplikacije.\n  Ova funkcionalnost rezervirana je za
administratore.";

        }
```

Isječak 5. Funkcija za postavljanje slike na okidanje *MouseHover* eventa

4.4. Forma za dodavanja Korisnika aplikacije

Kako korištenje ove aplikacije ne bi bilo ograničeno na samo jednu osobu, u sustav je dodana funkcionalnost za kreiranje dodatnih korisničkih računa. Svrha ove funkcionalnosti je ta da poslodavac može određenim osobama dozvoliti pristup podacima i rad s istima. Ovim pristupom poslodavac više nije obvezan aplikacijom upravljati sam, već se time mogu baviti i ostale osobe koje imaju pristup.



Slika 12. Forma za kreiranje novog korisničkog računa

S obzirom na to da svaki korisnički račun mora imati korisničko ime, lozinku i status administratora, ova forma sadrži pripadne kontrole. Osim kontrola za unos korisničkih

podataka, forma sadrži i dva gumba, jedan za povratak i jedan za kreiranje korisnika sa željenim podacima. Kako je odlučeno na razini aplikacije, korisničko ime može biti duljine do deset znakova, a lozinka može biti duljine do četrnaest znakova. Kao ni kod forme za kreiranje inicijalnog administratorskog računa, ni ovdje nema posebnih zahtjeva za kompleksnost lozinke. Jedina potreba je ta da se lozinka poklapa s unosom u polju za potvrdu lozinke. Ova funkcionalnost rezervirana je isključivo za korisnike koji imaju administratorska prava, pa je isto tako na samoj formi moguće odrediti hoće li novi korisnički račun imati administratorska prava ili ne, što se postavlja s obzirom na stanje *CheckBox* kontrole.

Kako se u aplikaciji radi s podacima, da bi se neke određene akcije mogle točno povezati s korisnikom koji ih je izveo, potrebno je da se uneseno korisničko ime za novi račun ne poklapa s niti jednim već postojećim korisničkim računom u bazi. Taj se detalj provjerava pozivom posebne funkcije u trenu kada korisnik pokuša kreirati novi račun.

```
private bool CheckForAvailability(Korisnik korisnik)
{
    List<Korisnik> users = Repository.GetUsers();

    bool exists = false;
    foreach (Korisnik k in users)
    {
        if (k.Username == korisnik.Username)
        {
            exists = true;
        }
    }
    return exists;
}
```

Isječak 6. Provjera dostupnosti unesenog korisničkog imena

Na temelju rezultata koji vrati funkcija, odvija se određena funkcionalnost. Ako je funkcija kao rezultat vratila *true*, korisnika se obavještava kako u bazi podataka već postoji korisnički račun s unesenim korisničkim imenom. Ukoliko je funkcija vratila status *false*, to znači kako u bazi nema korisničkog računa s unesenim korisničkim imenom te se proces dodavanja novog korisničkog računa nastavlja.

Jednom kada je utvrđeno kako su uneseni podaci validni, te se u bazi ne nalazi korisnički račun s unesenim imenom, na temelju unesenih podataka kreira se nova instanca objekta Korisnik, čijim se svojstvima postavljaju vrijednosti koje su prethodno unesene u pojedine i odgovarajuće kontrole. Po uspješnom kreiranju novog objekta, isti se proslijedi u funkciju za dodavanje Korisnika u bazu podataka.

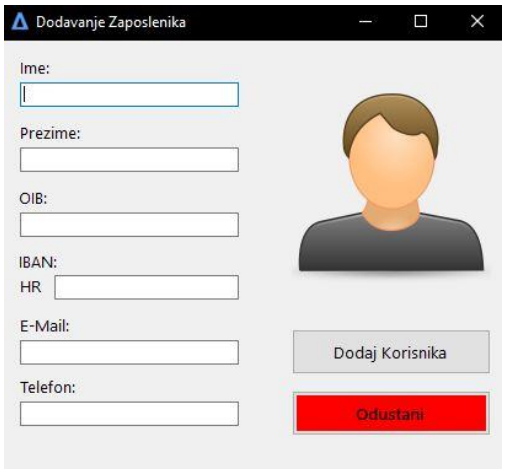
Ukoliko se funkcija uspješno izvrši i ne pojave se problemi prilikom dodavanja novog zapisa u bazu podataka, korisnika se obavještava o uspjehu prikladnom porukom korištenjem *MessageBox* klase te se sva polja za unos vrate na inicijalno prazno stanje. Isto tako, ukoliko je prilikom kreiranja novog korisnika došlo do problema, proces se zaustavlja te se korisnika obavještava o pogrešci te se proces vrati na inicijalno stanje.

Prilikom odabira opcija za odustajanje ili zatvaranje trenutne forme, pokreću se prikladni događaji (engl. *Event*) koji zatvaraju / uništavaju trenutnu instancu forme *NewUser*, te se korisniku po završetku zatvaranja ove forme opet prikazuje forma *MainMenu* u kojoj on može dalje nastaviti s radom.

4.5. Dodavanje novog zaposlenika

Dodavanje novih zaposlenika jedna je od funkcionalnosti rezerviranih isključivo za korisnike s administratorskim ovlastima. Formi se kao i ostalima pristupa klikom na odgovarajući gumb na glavnom izborniku. Jednom kada je forma uspješno pokrenuta, korisnik može nastaviti s radom. Ova forma sastoji se od niza kontrola za unos podataka o zaposleniku. Kontrole za unos su uglavnom tipa *TextBox*, a osim njih forma sadrži još i niz tekstualnih oznaka koje pobliže opisuju svaku kontrolu za unos. Kontrole za unos pokrivaju potrebu za unosom podataka koji odgovaraju svojstvima objekta Zaposlenik, a ona su redom:

- Ime, Prezime, OIB, IBAN, Email, Telefon



Slika 13. Forma za kreiranje novog zaposlenika

Svaki od navedenih podataka obavezan je za unos te dodavanje zaposlenika neće biti moguće sve dok svi potrebni podaci ne budu uneseni i validni.

Osim provjera za unosom potrebnih podataka, ova forma sadrži niz dodatnih provjera koje služe kako bi se osiguralo da je:

- OIB duljine 11 znakova koji su isključivo brojevi
- IBAN duljine 19 znakova koji su isključivo brojevi
- Telefon znakovni niz sastavljen isključivo od brojeva
- Email u skladu s *Regex*[8] uzorkom

Polja za unos koja trebaju dozvoljavati unos isključivo brojeva, to imaju osigurano tako da se događaj koji se okine prilikom pritiska tipke na tipkovnici okine samo onda kada je pritisnuta tipka bila broj. Nadalje, kopiranje i lijepljenje slova i znakova također nije moguće jer je na razini kontrole onemogućeno korištenje sistemskih kratica.

Bez obzira na to što je IBAN duljine 21 znak, polju za unos IBAN broja omogućen je unos niza od 19 znakova. Razlog tome je taj što je za ovu aplikaciju predefinirano to da IBAN počinje znakovima „HR“ iz razloga što je za potrebe ove aplikacije predviđeno zapošljavanje ljudi koji imaju bankovni račun stacioniran isključivo u Republici Hrvatskoj. Tako se unesenom brojevnom nizu od 19 znakova na početak automatski pridodaju znakovi „HR“ kako bi IBAN onda odgovarao propisanom formatu od 21 znaka. S obzirom na svrhu i potrebe ove aplikacije, odlučeno je kako se IBAN i OIB neće moći mijenjati u budućnosti.

U ovoj formi također se i prvi puta pojavljuje potreba za uvođenjem *Regex* provjere. Ta potreba pojavljuje se kod unosa *email* adrese. Potrebno je da unesena adresa odgovara nekom pravilnom formatu iz razloga što aplikacija ima funkcionalnosti slanja obavijesti zaposlenicima korištenjem upravo tih *email* adresa. Ispod je prikazan primjer postavljanja *Regex* uzorka kojem treba odgovarati unesena *email* adresa:

```
System.Text.RegularExpressions.Regex EmailRegex = new
System.Text.RegularExpressions.Regex(@"\w+([-+.]\w+)*@\w+([-.]\w+)*\.\w+([-
.]\w+)*");
                if (EmailRegex.IsMatch(txtEmail.Text))
                {
                    valid = true;

                }
```

Isječak 7. Programski blok za validaciju email adrese korištenjem *Regex-a*

[8] Regex – Niz znakova koji definiraju uzorak kojem neki unos mora odgovarati

4.6. Slanje Email obavijesti

Jedna od funkcionalnosti koje ova aplikacija osigurava je slanje obavijesti korištenjem email-a. Ova funkcionalnost dodana je u aplikaciju iz razloga što bi u suprotnom korisnik trebao imati pristup programu poput *Microsoft Outlook-a,* ili se dodatno koristiti *web* preglednikom kako bi pristupio *online* servisima za slanje poruka.

Bez obzira na to što se *Microsoft Outlook* smatra jednim od najboljih alata za slanje email poruka, on je i dalje zaseban program koji dolazi s *Microsoft Office* paketom. Uz pretpostavku da poslodavac ne želi koristiti taj cijeli paket, ili mu jednostavno nema pristup, jedna od alternativa mu je korištenje *Gmail-a*. *Gmail* je servis besplatan za korištenje za slanje email poruka koji je razvila tvrtka *Google*. Uglavnom se koristi u osobne svrhe, ali *Google* podržava korištenje servisa i u komercijalne svrhe. .*NET* ima podršku za slanje email poruka korištenjem raznih biblioteka, pa je tako u aplikaciju implementirana funkcionalnost za slanje istih korištenjem *Gmail SMTP*[9] protokola.

Kao i ostalim formama, formi za slanje obavijesti pristupa se iz glavnog izbornika jednom kada je korisnik uspješno prijavljen. Ova forma jedna je od tri forme čija je funkcionalnost rezervirana isključivo za korisnike koji imaju administratorske ovlasti. Ukoliko korisnik nije administrator, forma mu na glavnom izborniku neće biti prikazana za odabir. Jednom kada korisnik klikne na gumb za otvaranje forme, glavni izbornik nestaje s ekrana te se korisniku umjesto njega prikazuje forma za slanje obavijesti.

Forma za slanje obavijesti sadrži niz kontrola. Na vrhu forme nalaze se kontrole za odabir načina slanja. Obavijesti se mogu slati na dva načina:

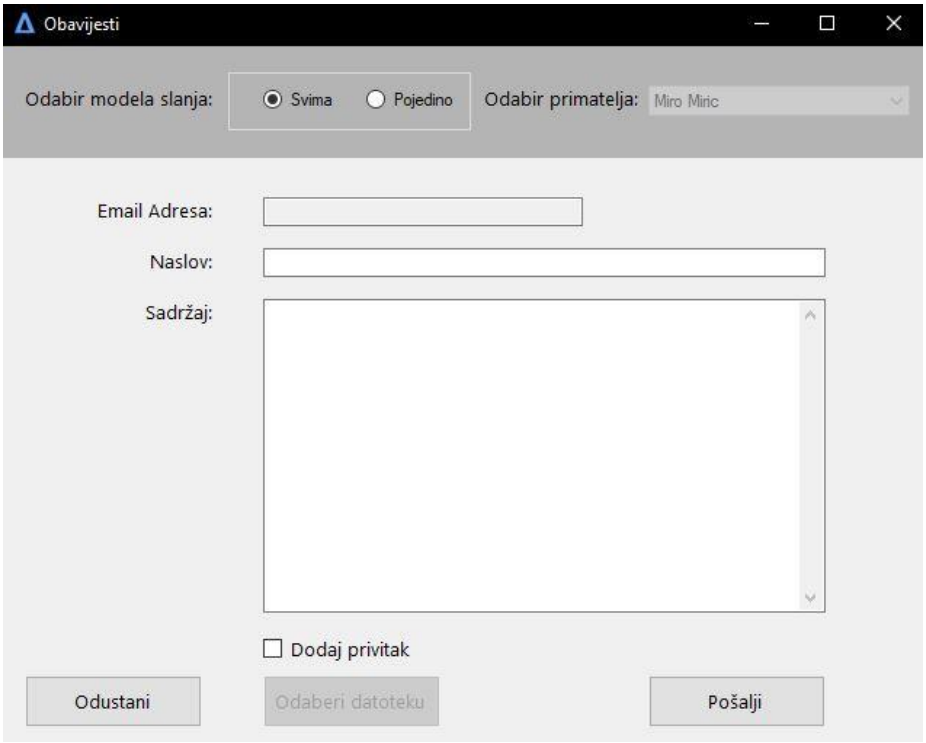
1. Svim zaposlenicima
2. Pojedinom zaposleniku

Ovisno o tome koji je način slanja odabran, omogućuju se ili onemogućuju pojedine kontrole. Ako je korisnik odabrao način slanja pojedinom zaposleniku, omogućuje mu se izbor zaposlenika iz padajućeg izbornika. Jednom kada je zaposlenik odabran iz izbornika, njegova adresa se automatski unosi u polje primatelja. Tada korisniku preostaje upisati naslov poruke te njen sadržaj.

[9] SMTP – Simple mail transfer protocol

Osim samog tekstualnog sadržaja, korisnik u poruku može dodati privitak koji može biti isključivo u *PDF*[10] formatu. Razlog ograničavanja formata na *PDF* je iz razloga što je predviđeno da korisnik kao privitak može dodavati potvrde o zaposlenju i putne naloge koji se u ostalim formama mogu izvesti isključivo u *PDF* formatu. Korisnik odabire hoće li dodati privitak ili ne klikom na odgovarajuću kontrolu na formi.

Slika u nastavku prikazuje izgled forme za slanje email obavijesti:



Slika 14. Forma za slanje email obavijesti

Jednom kada korisnik ispuni sva potrebna polja za unos, klikom na gumb za slanje započinje validaciju unesenih podataka. Ukoliko funkcija za validaciju potvrdi da su ispunjena sva potrebna polja, ona nastavlja proces slanja email poruke. Ukoliko je poruka uspješno sastavljena i poslana, korisnika se o tome obavještava prikladnom porukom u obliku *MessageBox* objekta. Na isti se način korisnika obavještava o eventualnim pogreškama, bilo da se radi o nedostatku potrebnog unosa ili grešci prilikom slanja poruke.

Ukoliko korisnik odustane od slanja poruke, klikom na gumb za odustajanje može odbaciti započetu poruku te se vratiti na glavni izbornik prilikom čega se forma za slanje obavijesti zatvara te se korisniku opet prikazuje forma glavnog izbornika.

[10] PDF - Portable Document Format

Ispod prikazana funkcija (Isječak 8.) opisuje princip slanja email obavijesti unutar aplikacije. S obzirom da je ciljani princip slanja korištenjem *Gmail* adrese, kako bi se mogla provesti autentifikacija korisničkih podataka vezanih uz *Gmail* adresu koja se koristi za slanje obavijesti, ciljani server mora biti „smtp.gmail.com“. Funkcija sastavlja poruku tipa *MailMessage*, u čije se vrijednosti postavljaju podaci iz pripadnih polja za unos.

Jednom kada je poruka sastavljena, ista se šalje na *Gmail* server. Adresa s koje se šalje poruka je napravljena isključivo za potrebe demonstracije ove aplikacije. Kako bi račun bio uspješno autentificiran, a poruka poslana, u *Gmail* postavkama na *web* servisu potrebno je omogućiti pristup navedenom računu iz manje sigurnih aplikacija kako pristup iz ove aplikacije ne bi bio blokiran[11].

```
private void SendEmail()
        {
            try
            {
                MailMessage mail = new MailMessage();
                SmtpClient SmtpServer = new SmtpClient("smtp.gmail.com");
                mail.From = new MailAddress("spatool.demo@gmail.com");
                mail.To.Add(txtAddress.Text);
                mail.Subject = txtSubject.Text;
                mail.Body = txtContent.Text;

                //System.Net.Mail.Attachment attachment;
                //privitak = new System.Net.Mail.Attachment("putanja");
                //mail.Attachments.Add(privitak);

                SmtpServer.Port = 587;
                SmtpServer.Credentials = new
System.Net.NetworkCredential(EMAIL_ADDRESS, EMAIL_PASSWORD);
                SmtpServer.EnableSsl = true;

                SmtpServer.Send(mail);
                MessageBox.Show("Poruka je uspješno poslana!");
            }
            catch
            {
                MessageBox.Show("Neuspjelo slanje poruke.");
            }
        }
```

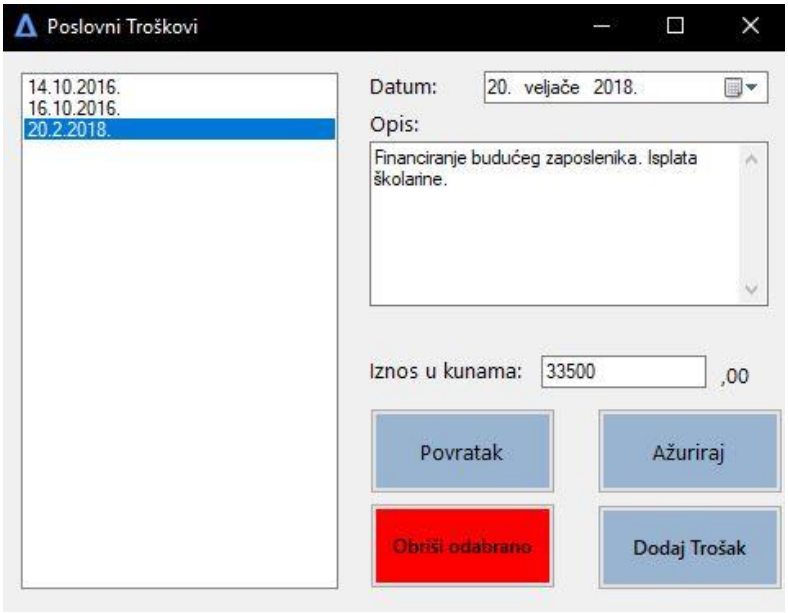
Isječak 8. Primjer funkcije za slanje email obavijesti

[11] https://support.google.com/accounts/answer/6010255?hl=en

4.7. Poslovni troškovi

Smisao ove aplikacije je omogućavanje jednostavnijeg pregleda i uvida u neke poslovne podatke i informacije. S obzirom na to da se radi o nekoj vrsti poduzeća, vrlo je vjerojatno da isto ima neke zapise o poslovnim troškovima. Ova forma ima ulogu prikazati te poslovne troškove u jednostavnom obliku, te olakšati upravljanje s istima. U demonstrativne svrhe, određeno je da svojstva objekta „Poslovni trošak“ budu:

- ID – Tipa *Integer*, jedinstvena oznaka svakog troška
- Datum – *Date*, označava datum na koji je trošak zabilježen
- Opis – *String*, tekstualni opis poslovnog troška
- Iznos – *Integer*, iznos poslovnog troška u kunama



Slika 15. Forma za upravljanje poslovnim troškovima

Kontrole koje sadržava iznad prikazana forma po svojim funkcionalnostima odgovaraju svojstvima objekta „Poslovni trošak“. Jednom kada je forma pokrenuta i prikazana korisniku, ukoliko lista poslovnih troškova nije prazna, u *ListBox* kontroli se selektira prvi po redu trošak, taj trošak je tada kopiran u instancu objekta „Poslovni trošak“ kreiran na razini forme. Kontrole za unos podataka na formi su zatim ispunjene podacima iz svojstava tog objekta.

Korisnik ima na raspolaganju nekoliko funkcionalnosti:

- Ažuriranje odabranog troška
- Brisanje odabranog troška
- Dodavanje novog troška

Ako korisnik želi ažurirati podatke trenutno odabranog troška, potrebno je u kontrolama za unos unijeti ispravne podatke. Polja za unos opisa i iznosa ne mogu ostati prazna, dok je datum nemoguće ostaviti neispunjen. Jednom kada je korisnik unio željene podatke, klikom na gumb za ažuriranje pokreće funkciju koja ažurira trenutno odabrani trošak pod uvjetom da su svi podaci ispravno uneseni. Ako ažuriranje ne uspije, korisnika se o tome obavještava odgovarajućom porukom. U polje za opis dozvoljen je unos do duljine 1000 znakova, dok je u polje za iznos dozvoljen unos do maksimalne vrijednosti *int* tipa podataka s obzirom da očekivani troškovi za malo poduzeće nikad ne bi trebali prelaziti taj iznos[12]. Ukoliko korisnik želi dodati novi zapis u bazu podataka, on može iskoristiti podatke kojima su polja za unos već ispunjena, ili obrisati trenutne podatke i unijeti nove. Jednom kada je korisnik unio željene podatke, novi trošak u bazu unosi pritiskom na gumb za dodavanje troška. Validacija unesenih podataka odvija se po istom principu kao i kod ažuriranja troška. Ako je trenutno prijavljeni korisnik administrator, njemu je omogućen i gumb za brisanje odabranog troška. Korisnik trošak može obrisati tako da odabere trošak iz *ListBox* kontrole, te pritisne gumb za brisanje. Ispod je prikazana funkcija koje se izvodi jednom kad korisnik pritisne gumb za dodavanje novog troška. Prvo se pozivaju funkcije za validaciju podataka, i ukoliko iste potvrde da su uneseni podaci ispravni, svojstva objekta „PoslovniTrosak“ se postavljaju na vrijednosti odgovarajućih kontrola za unos unutar forme. Jednom kada su sva svojstva postavljena, novi trošak šalje se na sloj za rad s podacima gdje se isti dodaje u bazu.

```
if (ValidNumber())
{
    if (txtOpis.Text.Length > 0)
    {
        PoslovniTrosak temp = new PoslovniTrosak();
        temp.Datum = dtpDatum.Value;
        temp.Opis = txtOpis.Text;
        temp.Iznos = int.Parse(txtIznos.Text);

        try
        {
            Repository.AddExpense(temp);
            MessageBox.Show("Trošak je uspješno dodan!", "Uspjeh");
            if (NapuniListu())
            {
                dtpDatum.Value = trosak.Datum;
                txtOpis.Text = trosak.Opis;
                txtIznos.Text = trosak.Iznos.ToString();
            }
            lblError.Text = "";
```

Isječak 9. Funkcija za dodavanje novog poslovnog troška

[12] Maksimalna vrijednost *int* tipa podataka iznosi: 2,147,483,647 u spektru pozitivnih brojeva.

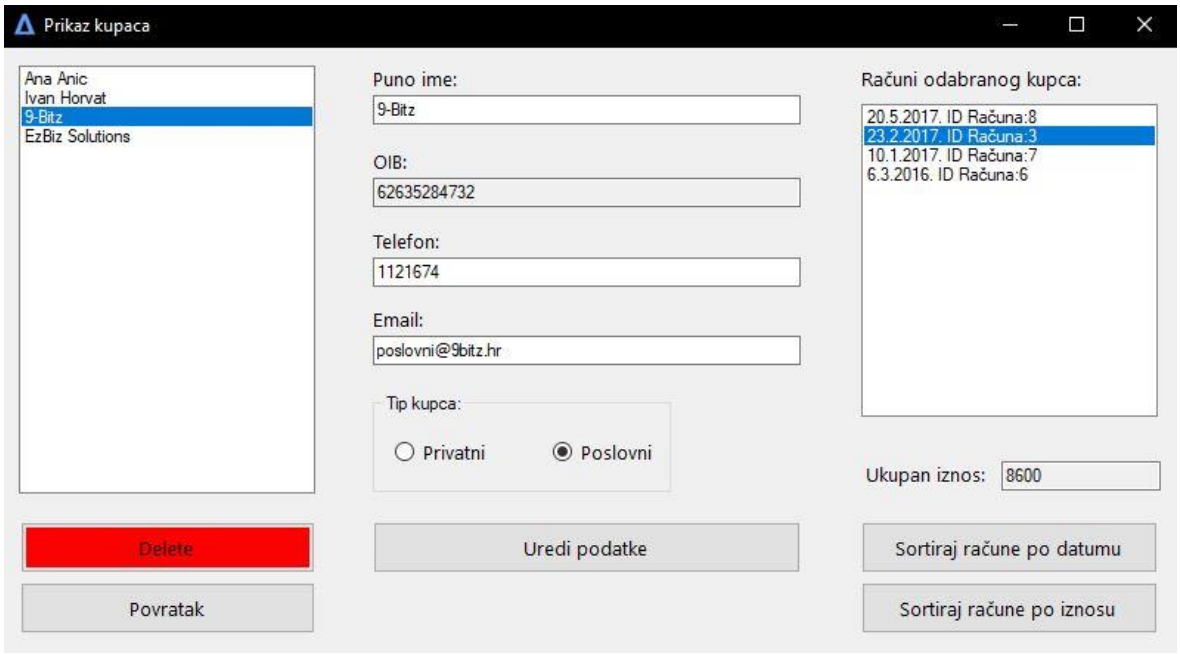
4.8. Forma za prikaz i rad s podacima o kupcima

Za poduzeće za čije je potreba izrađena aplikacija pretpostavlja se da se bavi nekom vrstom prodaje neodređenih dobara ili usluga. S obzirom na to da se poduzeće bavi prodajom, podrazumijeva se da se za prodaje čuvaju podaci o kupcima i računima.

S obzirom na zamišljen princip rada ovog poduzeća, pretpostavljeno je da kupci mogu biti privatni ili poslovni. Za svakog kupca, bilo poslovnog ili privatnog, u objektu se čuvaju slijedeća svojstva:

- Puno ime – Može biti kombinacija imena i prezimena privatnog kupca ili naziv poslovnog
- OIB – Bez obzira na vrstu kupca, OIB se pohranjuje u bazu
- Telefon – Broj telefona se pohranjuje za svakog od kupaca kao opcija kontakta
- Email – email adresa se pohranjuje kao dodatna opcija za kontakt
- Tip – Status koji određuje je li kupac poslovni ili privatni

Forma za rad s kupcima sadrži prikladna polja za unos i kontrole za sva od navedenih svojstava. Polje za OIB će biti tipa *Read-Only*[13], odnosno neće mu biti omogućeno uređivanje. Ova forma prikazana je na slici ispod:



Slika 16. Forma za upravljanje kupcima

[13] Read-Only – Svojstvo koje kada je omogućeno ne dozvoljava uređivanje unutar pripadne kontrole

Nakon što korisnik klikne gumb za prikaz ove forme, glavni izbornik se sakriva s ekrana te se korisniku prikazuje forma za rad s kupcima. Forma koja se prikazuje sastoji se od niza kontrola, od kojih glave kontrole prikazuju popis kupaca koji se nalaze u bazi podataka te pripadne račune trenutno odabranog kupca.

Jednom kada se odabere određeni kupac iz *ListBox* kontrole, taj se kupac kopira u instancu objekta *Kupac* na razini forme. Podaci tog trenutnog kupca se prikazuju u pripadnim kontrolama i poljima za unos, a u *ListBox* kontroli na desnoj strani prikazuje se popis svih računa izdanih tom kupcu te njihov ukupan iznos u kunama.

Korisniku je na raspolaganju nekoliko funkcionalnosti ovisno o tome ima li trenutno prijavljeni korisnik administratorske ovlasti ili ne. S obzirom da je za kupce predviđeno da u bazu bivaju dodavani iz nekog vanjskog sustava, forma nema mogućnost dodavanja novih kupaca. Korisnik može ažurirati podatke kupaca, te ih brisati ukoliko je on administrator. Validacija unesenih podataka izvršava se pokretanjem ispod prikazane funkcije jednom kada korisnik klikne na gumb za ažuriranje. Na razini funkcije kreirana je varijabla „*valid*“ koja je tipa *bool*. Funkcija prolazi kroz sva polja za unos i provjerava jesu li unesene vrijednosti ispravne. Ukoliko su sve unesene vrijednosti ispravne, vrijednost varijable *valid postavlja se na true*, pri čemu funkcija potvrđuje da su uneseni podaci ispravni. U suprotnom, funkcija vrati rezultat *false* te se ažuriranje prekida i korisnika se obavještava odgovarajućom porukom. Funkcija za validaciju prikazana je u nastavku:

```
private bool ValidateData()
        {
            bool valid = false;

            if (txtIme.Text != "" && txtOIB.Text.Length == 11
                && txtEmail.Text != "" && txtTelefon.Text != "")
            {
                System.Text.RegularExpressions.Regex EmailRegex = new
System.Text.RegularExpressions.Regex(@"\w+([-+.]\w+)*@\w+([-.]\w+)*\.\w+([-
.]\w+)*");
                if (EmailRegex.IsMatch(txtEmail.Text))
                {
                    valid = true;
                }
            }
            return valid;

        }
```

Isječak 10. Funkcija za validaciju unesenih podataka o kupcu

Ukoliko korisnik odluči obrisati trenutnog kupca iz baze podataka, istovremeno se iz baze brišu i svi računi koji pripadaju tom istom kupcu. Upravo je iz tog razloga brisanje kupaca rezervirano isključivo za administratore.

4.9. Forma za upravljanje zaposlenicima

Upravljanje zaposlenicima jedna je od temeljnih funkcionalnosti ove aplikacije. Cijeli smisao izrade ovakve aplikacije zapravo je taj da poslodavac na njemu lak i intuitivan način može lako pristupiti bitnim podacima vezanim uz svoje zaposlenike i s istima upravljati.

Ova forma dostupna je za korištenje svim korisnicima aplikacije, ali isto tako ima i neke funkcionalnosti rezervirane za korisnike s administratorskim računima.

Korisnik formi pristupa klikom na odgovarajući gumb na glavnom izborniku, nakon čega mu se prikazuje ova forma. Jednom kada je forma prikazana, ukoliko ima zaposlenika u bazi podataka, oni su prikazani u *ListBox* kontroli s lijeve strane. Po odabiru određenog zaposlenika, sva polja za unos se automatski ispunjavaju njegovim podacima. Podaci OIB i IBAN se smatraju unikatnim te se za iste pretpostavlja da se neće mijenjati, stoga su polja za unos istih onemogućena, odnosno postavljeni su na *Read-Only*. Radni dani pojedinog zaposlenika u bazi podataka su zapisani u specifičnom formatu. Upravo je iz tog razloga napravljena funkcija za ispunu polja za prikaz i odabir radnih dana zaposlenika:

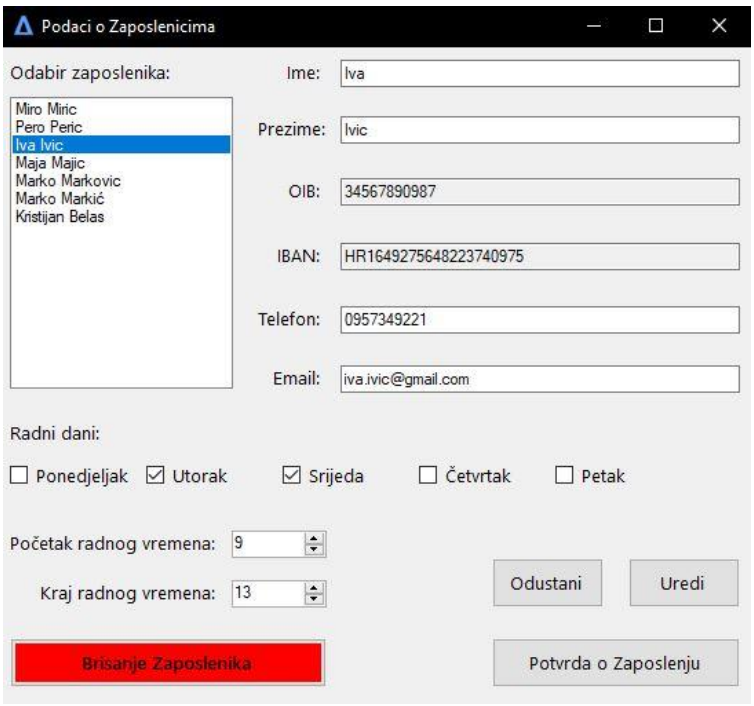
```
String[] dani = rv.RadniDani.Split(';');
for(int i = 0; i < dani.Length; i++)
{
    if (dani[i] == "Ponedjeljak")
    {
        cbPon.Checked = true;
    }
    if (dani[i] == "Utorak")
    {
        cbUto.Checked = true;
    }
    if (dani[i] == "Srijeda")
    {
        cbSri.Checked = true;
    }
    if (dani[i] == "Cetvrtak")
    {
        cbCet.Checked = true;
    }
    if (dani[i] == "Petak")
    {
        cbPet.Checked = true;
    }
```

Isječak 11. Programski blok za ispunu polja vezanih uz radne dane

Prikazana funkcija prolazi kroz sve vrijednosti unutar polja „dani“ i ona označava određene *CheckBox* kontrole ovisno o tome koji su dani navedeni kao radni za odabranog zaposlenika. Ukoliko zaposlenik nema radnih dana, neće biti označenih kontrola.

Radni sati svakog zaposlenika čuvaju se u obliku broja koji može biti u rasponu od 0 do 24. Za svakog zaposlenika čuva se podatak o početku i kraju radnog vremena. S obzirom na tip ovih podataka, u svrhu jednostavnosti odabrana je *NumberPicker* kontrola pomoću koje se vrlo jednostavno i intuitivno može mijenjati vrijednost broja.

Kada je zaposlenik inicijalno kreiran, za njega nisu stvoreni podaci o radnom vremenu, već se ti podaci mogu i moraju prvi puta postaviti upravo korištenjem ove forme. Jednom kada je zaposlenik odabran i polja su ispunjena njegovim podacima, vidljivo je ima li on podataka o radnom vremenu ili ne. Ukoliko zaposlenik nema pripadnih podataka o radnom vremenu, polja za unos radnih dana i radnih sati neće biti označena, i biti će postavljena na broj 0.



Slika 17. Forma za uređivanje podataka o zaposlenicima

Na slici iznad prikazana forma prema sličnom principu kao i kod ostalih formi omogućuje brisanje i uređivanje odabranog zaposlenika. Kod ove forme posebno je to što korisnik može za odabranog zaposlenika ispisati potvrdu o zaposlenju u PDF formatu. Klikom na gumb za potvrdu, ista se generira na temelju podataka odabranog zaposlenika te se ona pohranjuje na predefiniranu lokaciju na disku.

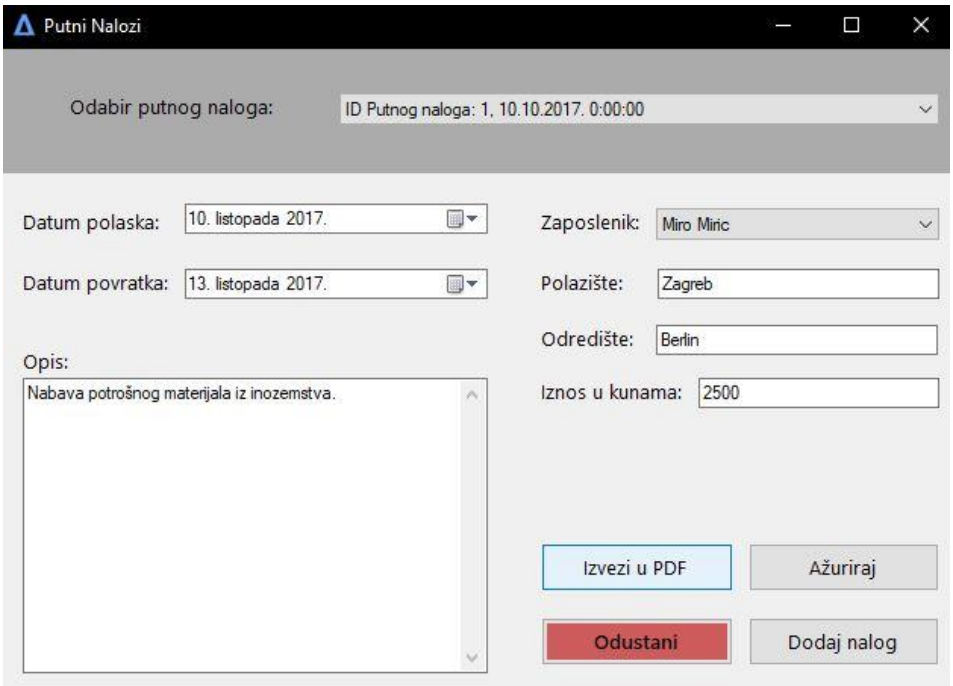
4.10. Forma za pregled putnih naloga

Pretpostavljeno je kako u ovom poduzeću povremeno dolazi do potrebe za poslovnim putovanjima i da se u bazi pohranjuju zapisi o istima. Putni nalozi izdaju se jednom od zaposlenika dostupnih u bazi podataka.

Ukoliko korisnik aplikacije želi pregledati putne naloge koji se trenutno nalaze u bazi, to radi tako da pristupi formi za pregled putnih naloga pritiskom na odgovarajući gumb s glavnog izbornika. Jednom kada pritisne taj gumb, korisniku se sakriva glavni izbornik i prikazuje mu se forma za pregled putnih naloga.

Ova forma sastoji se od niza kontrola za unos i odabir podataka. Prilikom inicijalnog prikazivanja forme, poziva se funkcija koja iz baze podataka dohvaća zapise o putnim nalozima. Ukoliko dohvaćena lista nije prazna, *ComboBox* kontrola za prikaz se napuni podacima iz liste putnih naloga. Ako je lista naloga prazna, korisnika se obavještava prikladnom porukom te se od njega traži unos novog naloga. Jednom kada je *ComboBox* kontrola ispunjena podacima, okida se *event* koji za trenutno odabranu stavku ispunjava sva polja za unos prikladnim podacima iz objekta.

Osim dodavanja novih naloga, korisnik na raspolaganju ima mogućnost ažuriranja i izvoz postojećih naloga u PDF format. Ispod je prikazana forma za upravljanje putnim nalozima:



Slika 18. Forma za pregled i upravljanje putnim nalozima

```
private void cbNalozi_SelectedIndexChanged(object sender, EventArgs e)
        {
            if(nalozi.Count > 0)
            {
                nalog = (PutniNalog)cbNalozi.SelectedItem;
                dtpPolazak.Value = nalog.Datum;
                dtpPovratak.Value = nalog.DatumPovratka;
                txtOpis.Text = nalog.Opis;
                txtPolazak.Text = nalog.Polaziste;
                txtOdrediste.Text = nalog.Odrediste;
                txtIznos.Text = nalog.Iznos + "";
                if (zaposlenici.Count > 0)
                {
                    cbZaposlenici.SelectedValue = nalog.ZaposlenikID;
                }
                txtPolazak.Text = nalog.Polaziste;
                txtOdrediste.Text = nalog.Odrediste;
                txtIznos.Text = nalog.Iznos + "";
            }

        }
```

Isječak 12. Funkcija za ispunu polja putnih naloga

Iznad je prikazana funkcija koja se izvršava na promjenu odabranog naloga iz *ComboBox* kontrole. Ukoliko funkcija zaključi da lista „nalozi“ sadrži barem jednu stavku, izvršava se blok kôda koji ispunjava sva polja za unos odgovarajućim podacima. Za ažuriranje je potrebno je ispuniti sva prikazana polja za unos s obzirom na to da su sva navedena kao obavezna u bazi podataka. Jednom kada korisnik pritisne tipku za ažuriranje naloga, funkcija provjerava ispravnost unesenih podataka, te ako je rezultat funkcije pozitivan, pokreće se funkcija iz repozitorija za uređivanje naloga u bazi.

```
public static void UpdateBusinessTrip(int id, PutniNalog nalog)
        {
            PutniNalog temp = model.PutniNalog.Find(id);
            temp.Datum = nalog.Datum;
            temp.Opis = nalog.Opis;
            temp.DatumPovratka = nalog.DatumPovratka;
            temp.ZaposlenikID = nalog.ZaposlenikID;
            temp.Polaziste = nalog.Polaziste;
            temp.Odrediste = nalog.Odrediste;
            temp.Iznos = nalog.Iznos;
            model.SaveChanges();

        }
```

Isječak 13. Funkcija za ažuriranje odabranog putnog naloga

Ova funkcija kao parametre prima *id* i *PutniNalog*. Prvi parametar služi kako bi *Entity* u modelu baze pronašao točan putni nalog koji korisnik želi ažurirati. Taj nalog se postavlja u instancu objekta „PutniNalog“ na razini funkcije, a u njegova svojstva se postavljaju

vrijednosti svojstava iz instance putnog naloga koji je proslijeđen u funkciju kao drugi parametar. Jednom kada je nalog uspješno uređen, spremaju se promjene.

Osim ažuriranja, korisnik može trenutno prikazani putni nalog izvesti u PDF formatu. U demonstrativne svrhe, i s obzirom na potrebe aplikacije, PDF datoteka koju je korisnik kreirao spremljena je u korijensku lokaciju aplikacije. Važno je napomenuti kako se u slučaju da je korisnik uređivao podatke u poljima za unos, a nije ažurirao iste, u PDF unose stare vrijednosti postavljene u trenutnoj instanci objekta „PutniNalog“ na razini forme. Ispod prikazana funkcija opisuje proces mapiranja vrijednosti trenutno odabranog putnog naloga u pripadna polja unutar ručno izrađenog PDF predloška:

```
            private void IspuniPDF()
             {
                 string template = pdfTemplatePutanja;
                 string ispis = ispisPdfPutanja;

                 PdfReader pdfReader = new PdfReader(template);
                 PdfStamper pdfStamper = new PdfStamper(pdfReader, new
FileStream(ispis, FileMode.Create));
                 AcroFields pdfFormFields = pdfStamper.AcroFields;

                 pdfFormFields.SetField("PolazakField", nalog.Polazak);
                 pdfFormFields.SetField("PovratakField", nalog.DatumPovratka);
                 pdfFormFields.SetField("ZaposlenikField", zaposlenik.Ime +
zaposlenik.Prezime);
                 pdfFormFields.SetField("OpisField", nalog.Opis);
                 pdfFormFields.SetField("PolazisteField", nalog.Polaziste);
                 pdfFormFields.SetField("OdredisteField", nalog.Odrediste);
                 pdfFormFields.SetField("IznosField", nalog.Iznos);

                 pdfStamper.FormFlattening = true;

                 pdfStamper.Close();

             }
```

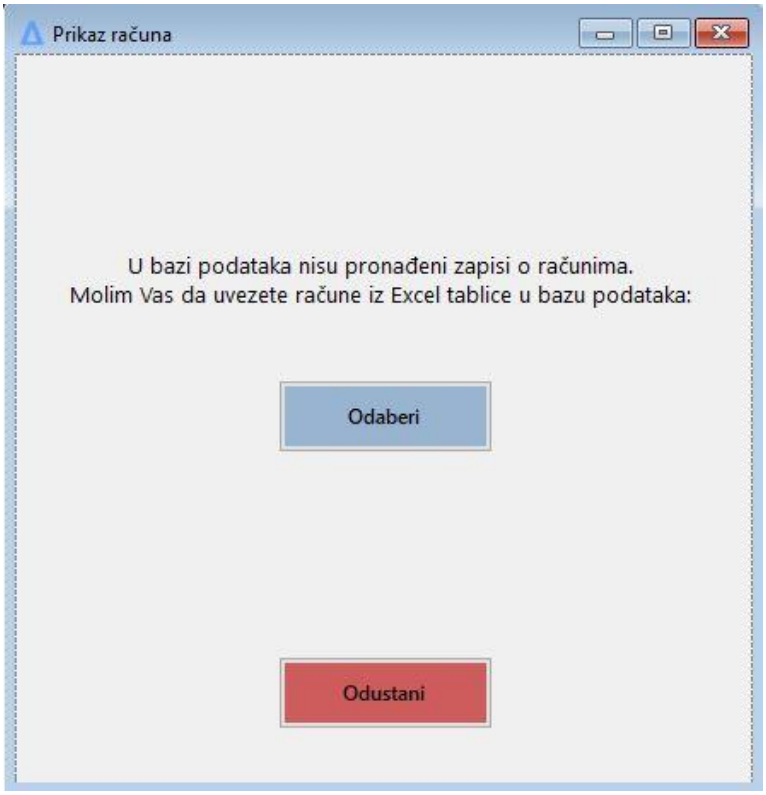
Isječak 14. Funkcija za mapiranje vrijednosti odabranog putnog naloga

4.11. Forma za prikaz računa

Svako poduzeće koje posluje kroz prodaju dobara ili usluga bi trebalo voditi evidenciju izdanih računa. Kroz funkcionalnosti ove aplikacije demonstriran je jedan od načina na koje te račune korisnik može jednostavno pregledavati i uređivati bez nepotrebnih kompleksnosti i nepreglednosti. Format računa s kojim je predviđen rad unutar aplikacije ostavljen je na jednostavnoj razini. Za svaki račun čuvaju se podaci o iznosu, datumu, obliku plaćanja i kupcu. Nije implementirana mogućnost dodavanja pojedinog računa ručno s obzirom na to

da je predviđeno da baza podataka u budućnosti bude povezana s vanjskim sustavom zaduženim za obavljanje prodaje i generiranja računa. S obzirom na to, prilikom prvog kreiranja instance forme za prikaz računa potrebno je učitati račune iz *Excel* datoteke za koju se pretpostavlja da je poslodavac koristio do sada za čuvanje zapisa o računima.

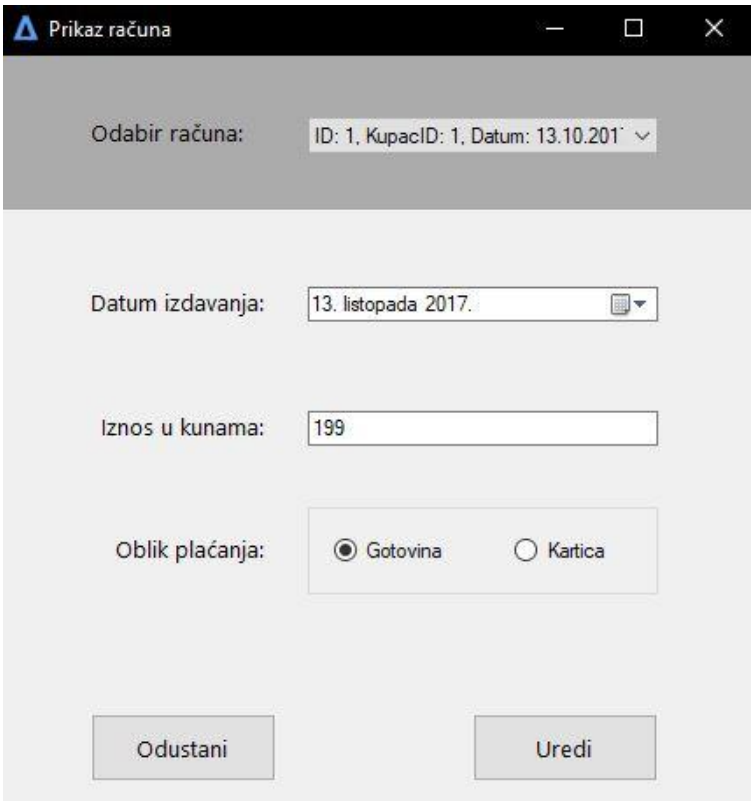
Kao što je prethodno navedeno, format zapisa unutar *Excel* datoteke prilagođen je demonstrativnim potrebama ovog rada. Jednom kada korisnik po prvi put pristupi ovoj formi iz glavnog izbornika, a tablica računa u bazi je prazna, njemu je umjesto regularnog prikaza prikazana forma na kojoj ga se upućuje na uvoz podataka o računima iz *Excel* datoteke:



Slika 19. Izgled forme u slučaju da nema podataka o računima

Korisnik odabire datoteku kroz „čarobnjak“, i ako je datoteka uspješno parsirana i podaci su učitani u bazu, njemu se prikazuje regularna forma za rad s računima.

Jednom kad se u bazi nalaze zapisi o računima, korisniku se prikazuje regularna forma za prikaz računa koja sadrži nekoliko kontrola za unos i *ComboBox* kontrolu za odabir računa. Izgled navedene forme moguće je vidjeti na slici u nastavku.



Slika 20. Regularni prikaz forme za prikaz računa

Prilikom pokretanja forme, provjerava se dostupnost računa. Ukoliko se pokaže da u bazi ima računa, pokreće se funkcija koja napuni *ComboBox* kontrolu svim dostupnim računima. Jednom kad je lista napunjena, trenutno odabrana stavka se kopira u instancu objekta „Racun“ na razini forme, te se podacima iz njegovih svojstava napune i postave polja za unos.

Kako je unos računa moguć samo preko učitavanja iz datoteke u slučaju da je lista računa prazna, korisnik nema mogućnost dodavanja novih računa ručno već mu je na raspolaganju samo funkcionalnost ažuriranja pojedinog računa. Ukoliko korisnik želi ažurirati podatke o računu, to može napraviti pritiskom odgovarajućeg gumba jednom kada su uneseni podaci validni.

Isto tako, korisnik u svakom trenutku može odustati od rada s računima i vratiti se na glavni izbornik klikom na gumb za odustajanje ili izlaskom iz forme na alternativne načine.

Zaključak

Bez obzira na to što je ovo relativno jednostavan primjer neke aplikacije koja bi imala ulogu upravljanja osnovnim podacima i informacijama koje posjeduje neko poduzeće, vidljivo je kako se podaci krajnjem korisniku mogu prezentirati na puno jednostavniji i pregledniji način korištenjem neke vrste grafičkog sučelja nego što bi to bilo prikazano kroz regularne tablične prikaze.

Glavna prednost kod ovako izrađenih aplikacija nalazi se u tome što se one mogu prilagoditi svakakvim zahtjevima korisnika i raznim formatima podataka. U ovakav tip aplikacije vrlo je jednostavno implementirati i funkcionalnosti za prikaz raznih proizvoda, usluga, rezervacija i sličnih potreba koje se uglavnom pojavljuju u poslovnom svijetu manjih poduzeća. U radu je prikazan primjer rada samo s nekim od funkcionalnostima s kojima se u stvarnom svijetu susreću manja poduzeća. S obzirom na usmjerenje i potrebe poduzeća potrebe za funkcionalnostima se mogu drastično razlikovati, ali gotovo svako poduzeće dijeli neke zajedničke funkcije koje su u ovoj aplikaciji implementirane.

Bitno je naglasiti kako se korištenje ovakve aplikacije ne bi preporučalo većim poduzećima s obzirom da se koristi jedna lokalna baza podataka i jedna instanca aplikacije. Ova aplikacija namijenjena je isključivo manjem poduzeću gdje bi pristup istoj imao poslodavac i još nekolicina autoriziranih zaposlenika. Ovdje je prednost ta da nema potrebe za plaćanjem održavanja servera i ostalih usluga vezanih uz rad preko interneta. Veća poduzeća koja rad obavljaju na više lokacija bi svakako trebala koristiti rješenja prilagođena radu preko interneta, što uglavnom rade web aplikacije.

Može se zaključiti da je postignut cilj izrade ove aplikacije s obzirom na to da se svi navedeni tipovi podataka korisniku sada prikazuju na njemu lakše razumljiv i pregledniji način.

Popis programskih isječaka

Popis slika

Popis tablica

Reference

[1] EntityFrameworkTutorial.net. (2018). *What is Entity Framework?* Dohvaćeno iz http://www.entityframeworktutorial.net/what-is-entityframework.aspx

[2] Microsoft. (23. Listopad 2016). *Entity Framework Code First to an existing Database*. Dohvaćeno iz https://msdn.microsoft.com/en-us/library/jj200620(v=vs.113).aspx

[3] Rouse, M. (Veljača 2017). *database(DB)*. Dohvaćeno iz TechTarget: http://searchsqlserver.techtarget.com/definition/database

[4] Wikipedia. (30. Studeni 2017). *Software*. Dohvaćeno iz https://en.wikipedia.org/wiki/Software

[5] Wikipedia. (6. Veljača 2018). *.NET Framework*. Dohvaćeno iz https://en.wikipedia.org/wiki/.NET_Framework

[6] Wikipedia. (1. Veljača 2018). *Microsoft SQL Server*. Dohvaćeno iz https://en.wikipedia.org/wiki/Microsoft_SQL_Server

[7] Wikipedia. (4. Veljača 2018). *Microsoft Visual Studio*. Dohvaćeno iz https://en.wikipedia.org/wiki/Microsoft_Visual_Studio

[8] Wikipedia. (11. Veljača 2018). *SQL*. Dohvaćeno iz https://en.wikipedia.org/wiki/SQL

[9] Wikipedia. (2. Veljača 2018). *Usage share of operating systems*. Dohvaćeno iz https://en.wikipedia.org/wiki/Usage_share_of_operating_systems#Desktop_and_laptop_computers