Univerzitet u Novom Sadu

Tehnički fakultet »Mihajlo Pupin«

Zrenjanin

***SEMINARSKI RAD***

*Predmet: Programski prevodioci*

***Tema:* Kreiranje objektno-orjentisane C# desktop aplikacije uz samostalno kreiranje biblioteke klasa i analiza ispravnosti programskog koda primenom Visual Studio NET okruženja**

**Tehnički pregled vozila**

**(TIP 2)**

Predmetni nastavnik: Prof. dr Ljubica Kazi Student: Boris Ferenčev

Broj indeksa: SI 40/19

Zrenjanin, 2022.godina

Sadržaj

[1. ZADATAK 3](#_Toc48667459)

[2. OPIS POSLOVNOG KONTEKSTA REALIZOVANOG PRIMERA 4](#_Toc48667460)

[3. KRATAK OPIS RAZVOJNOG OKRUŽENJA 5](#_Toc48667461)

[3.1. Microsoft Visual Studio 2019 5](#_Toc48667462)

[3.1.1. Namena alata 5](#_Toc48667463)

[3.1.2. Izgled ekrana alata 5](#_Toc48667464)

[3.1.3. Opis načina korišćenja 6](#_Toc48667465)

[3.2. Microsoft SQL Server Management Studio 18 9](#_Toc48667466)

[3.2.1. Namena alata 9](#_Toc48667467)

[3.2.2. Izgled ekrana alata 9](#_Toc48667468)

[3.2.3. Opis načina korišćenja 10](#_Toc48667469)

[4. Korisničko upustvo razvijene aplikacije 11](#_Toc48667470)

[4.1. Namena aplikacije i osnovne funkcije 11](#_Toc48667471)

[4.2. Ekrani i način korišćenja 1](#_Toc48667472)2

[5. Ključni elementi implementacije 1](#_Toc48667473)5

[5.1. SQL Script 1](#_Toc48667474)5

[5.2. Ključni delovi programskog koda sa objašnjenjem 1](#_Toc48667475)6

[6. Greške programskog koda 1](#_Toc48667476)8

[6.1. Segmenti programskog koda sa greškama i korekcija 1](#_Toc48667477)9

[6.2. Pravila i EBNF prikaz gramatički ispravnog programskog koda](#_Toc48667478) 21

[7. Prikaz primene alata u detektovanju grešaka 2](#_Toc48667479)2

[7.1. Ekranski prikaz izveštaja kompajlera nad neispravnim kodom i nakon popravke 2](#_Toc48667480)2

[7.2. Ekranski prikaz reakcije na run-time greške 2](#_Toc48667481)4

[7.2.1. Prvi primer run-time greške 2](#_Toc48667482)4

[7.2.2. Drugi primer run-time greške 2](#_Toc48667483)5

[8. Zaključak 2](#_Toc48667484)6

[9. Literatura 2](#_Toc48667485)7

[10. Listing 2](#_Toc48667486)8

[10.1. Listing korisničkog interfejsa 2](#_Toc48667487)8

[10.2. Konekcija sa bazom podataka](#_Toc48667488) 30

# 1. ZADATAK

Kreirati desktop aplikaciju C# sa primenom gotove biblioteke SQL DB Utils. Kreirati biblioteku KlasePodataka koja radi sa podacima iz baze podataka i primenjuje SQLDBUtils (rad sa jednom tabelom iz baze podataka). Kreirati u kodu namerne greške 5 tipa – leksičku, sintaksnu, semantičku, run-time i logičku. Pustiti kompajler da detektuje greške i prikazati izveštaj koji daje u vezi grešaka i prikaz ispravnog rada.

# 2. OPIS POSLOVNOG KONTEKSTA REALIZOVANOG PRIMERA

U ovom seminarskom radu opisana je desktop aplikacija za *tehnički pregled vozila*. Aplikacija omogućava unos podataka za radnike, vlasnike, tablice vozila, kao i prikaz već postojećih informacija u bazi podataka. Za svaki tehnički pregled su dati sledeći podaci: *ID broj tehničkog pregleda, ime i prezime radnika koji će uraditi proveru vozila, ime i prezime vlasnika, tablice koje se nalaze na vozilu i datum kada tehnički pregled zakazan.*

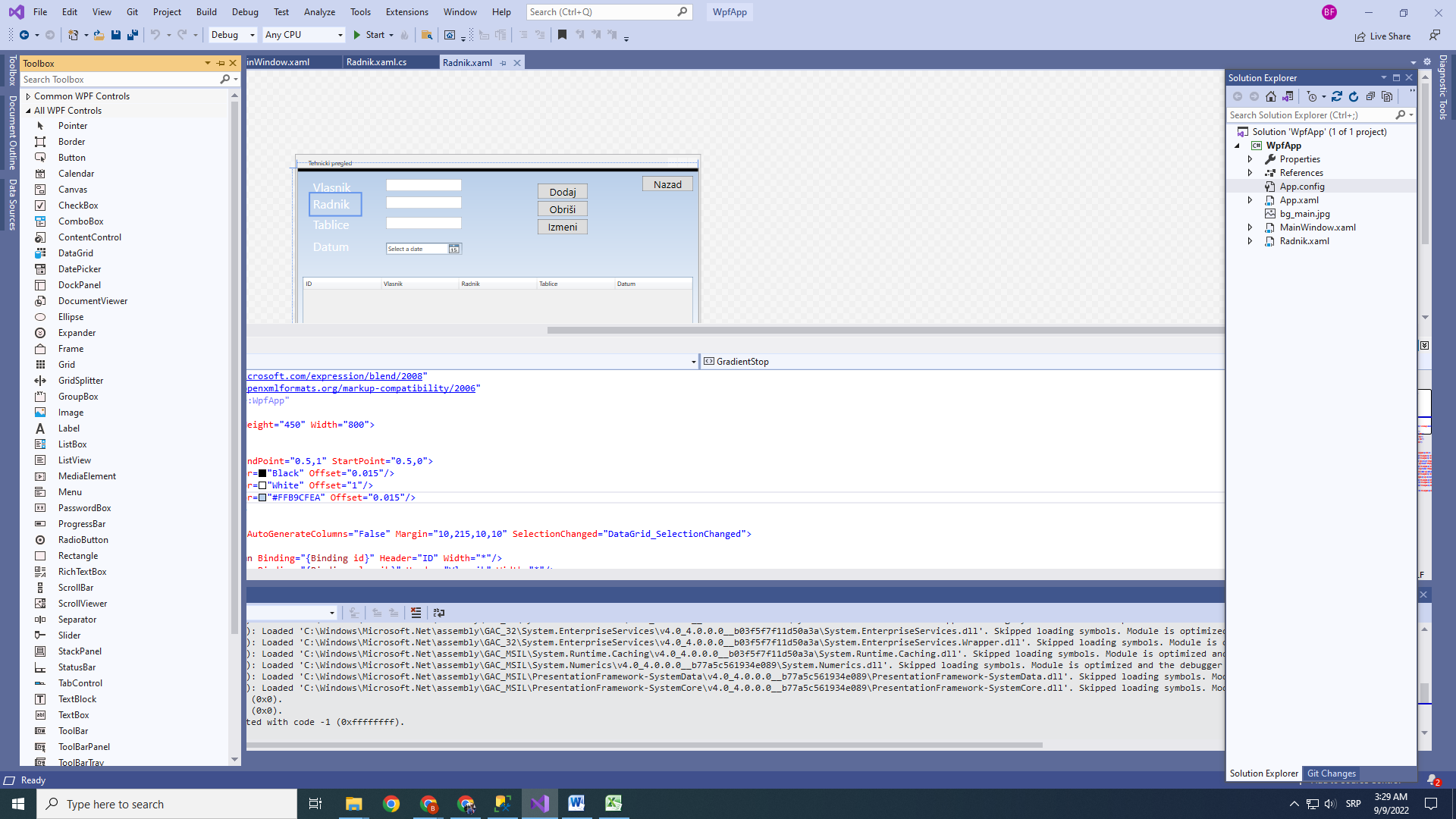
# 3. KRATAK OPIS RAZVOJNOG OKRUŽENJA

## 3.1. Microsoft Visual Studio 2019

### 3.1.1. Namena alata

Microsoft Visual Studio 2019 je Majkrosoftovo integrisano razvojno okruženje (IDE- Integrated development enviroment). Koristi se za razvoj kompjuterskih programa, sajtova, veb sajtova, veb aplikacija kao i mobilnih aplikacija. Visual Studio koristi različite platforme za kreiranje projekata, poput Windows Forms Application i Class Library. Podržava 36 raličitih programskih jezika. Visual Studio uključuje debugger koji radi i kao ispravljač grešaka na nivou izvora i kao ispravljač na nivou mašine.

### 3.1.2. Izgled ekrana alata



*Slika 1. Izgled alata Microsoft Visual Studio 2019 nakon učitavanja projekta desktop aplikacije*

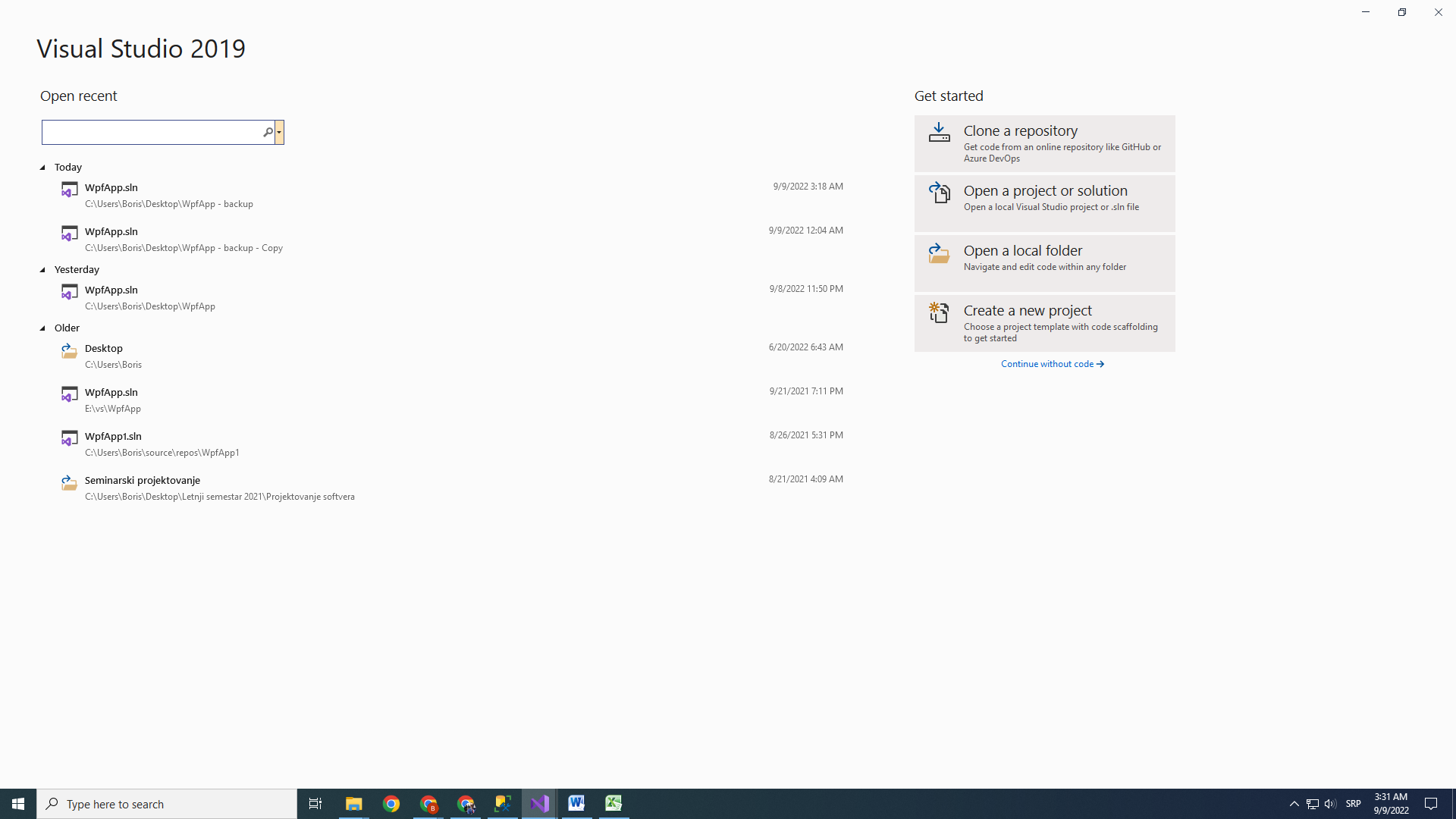
Na slici 1.prikazan je završni izgled aplikacije u razvojnom okruženju Microsoft Visual Studio 2019. Sa leve strane nalazi se *ToolBox*, koji omogućava lakše postavljanje grafičkih kontrola (dugmad, labele, textbox, itd). Sa desne strane se nalazi *Solution* *Explorer*, koji daje pregled fajlova uključenih u ovaj projekat.

Preko *Solution Explorer ->References* uključujemo naše biblioteke klasa, *KlasePodataka* i *SQLDBUtils*.

Zatim ispod *Solution Explorer*-a nalazi se prozor *Properties*, gde se podešavaju karakteristike (pozicija, naziv, tekst koji se prikazuje, itd) vezane za grafičke kontrole.

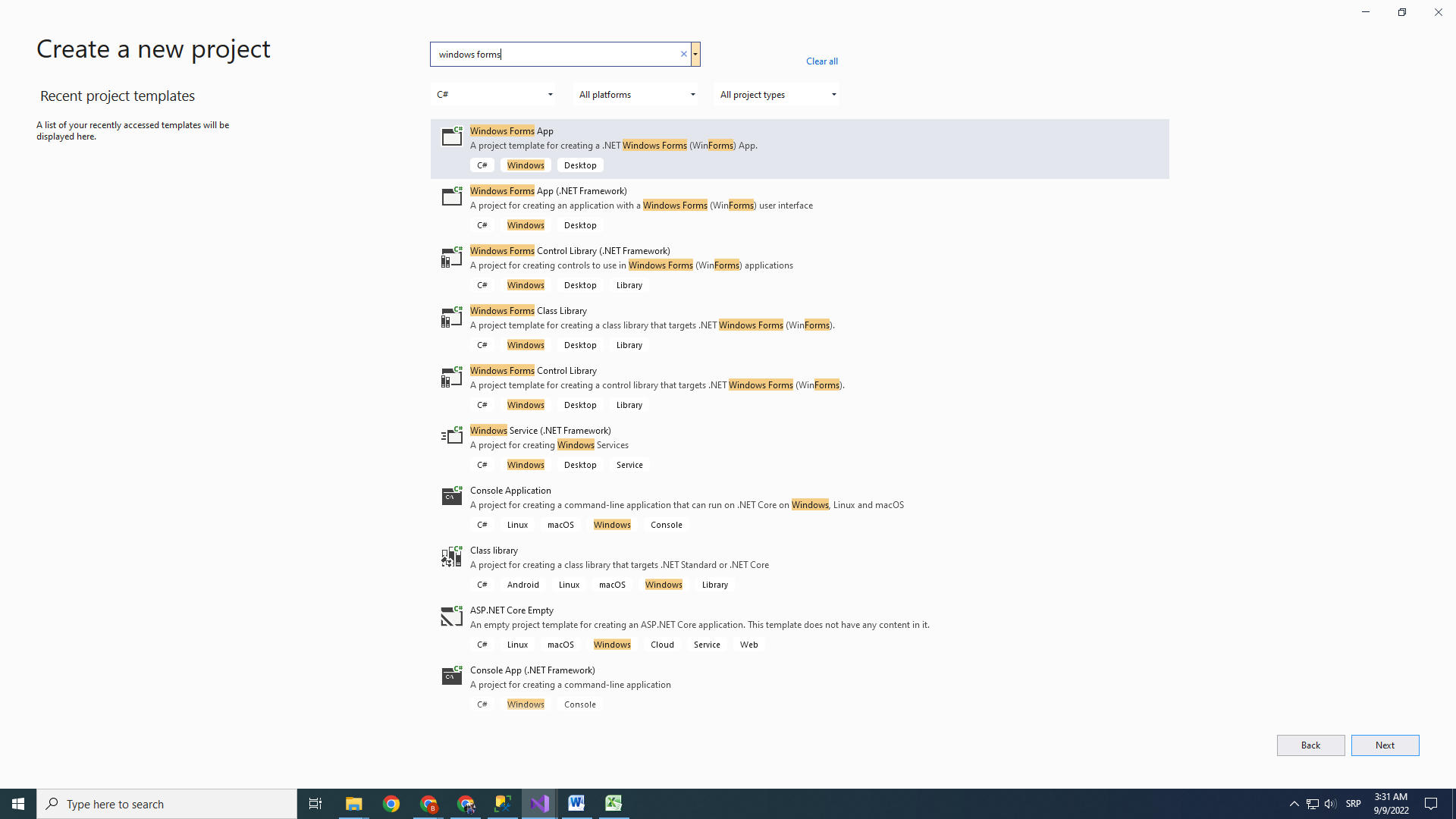
### 3.1.3. Opis načina korišćenja

Kada otvorimo alat u kojem pravimo našu aplikaciju, izgled ekrana je sledeći (Slika 2.):



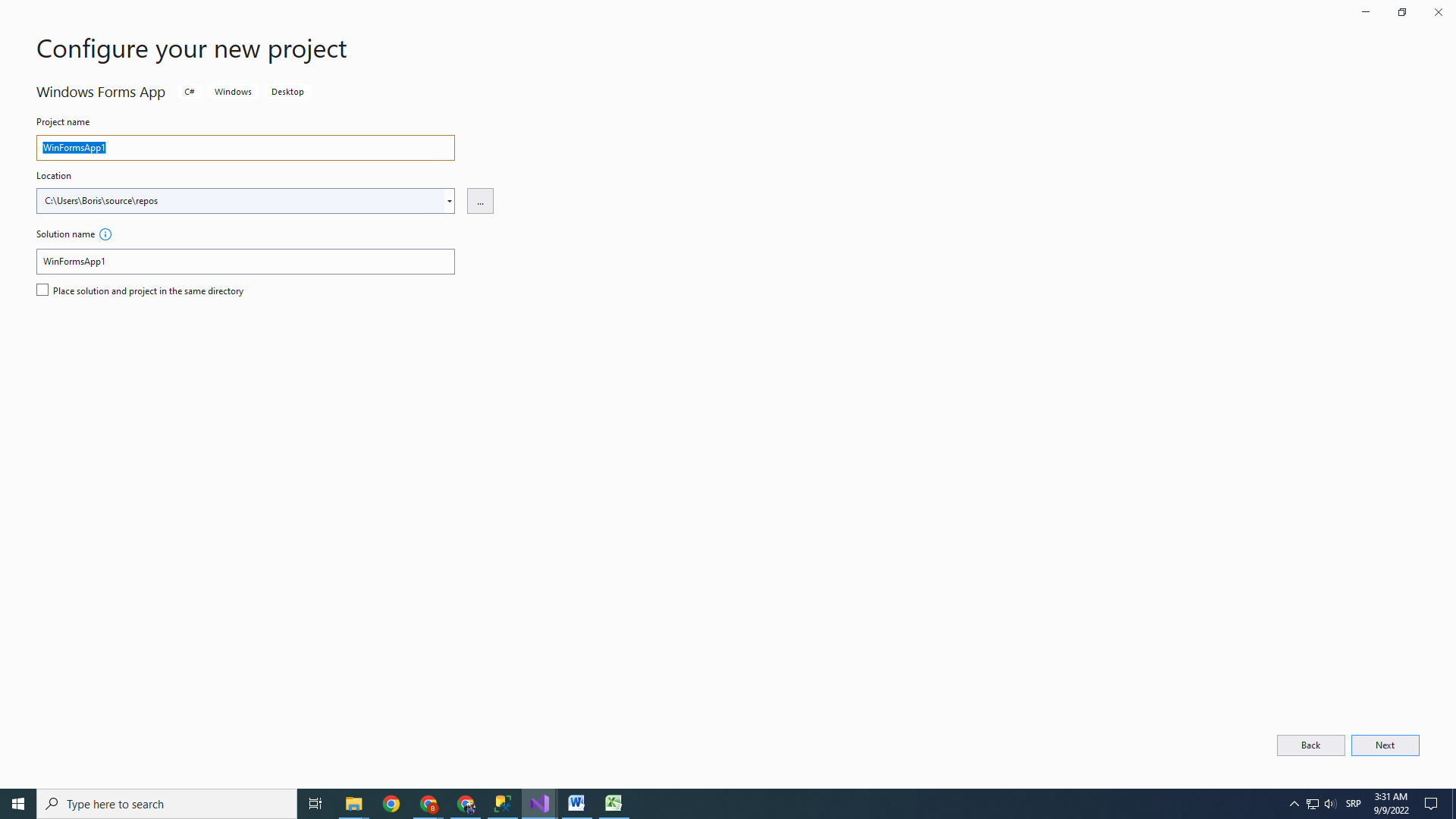
*Slika 2. Kreiranje novog projekta u alatu Microsoft Visual Studio 2019*

Kada se pritisne na kreiranje novog projekta, dobija se sledeći prozor u kojem se pomoću filtera bira programski jezik, platforma, kao i koju vrstu aplikacije želimo da pravimo. U našem slučaju biramo programski jezik C#, platforma je Windows, aplikacija je Desktop. Zatim nam program da koje sve vrste aplikacija možemo praviti, biramo Windows Forms App (.NET Framework), prikazano na slici 3.



*Slika 3. Kreiranje Windows Forms App (.NET Framework)*

Kao poslednji korak pri kreiranju projekta, jeste davanje naziva projektu i njegovu lokaciju gde će biti sačuvan na našem računaru (Slika 4.).

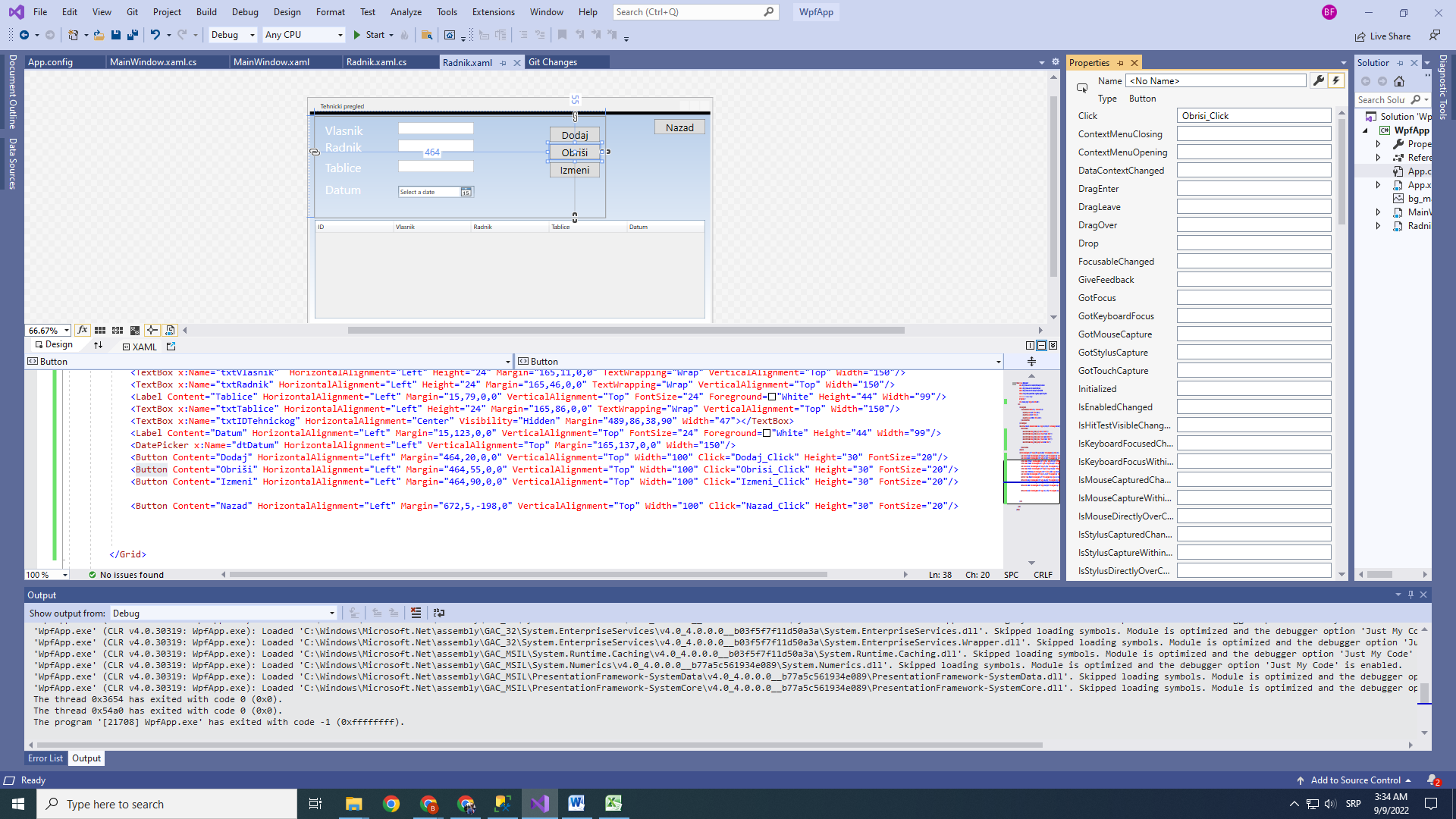


*Slika 4. Dodavanje naziva i lokaciju projekta*

Nakon pritiska na dugme *Create,* dobijamo grafički dizajn korisničkog interfejsa, a zatim u programskom kodu dodajemo određene akcije.

Kada se kreira grafički dizajn korisničkog interfejsa, treba kodirati funkcionalnosti vezane za našu aplikaciju. Te funkcionalnosti možemo dobiti na dva načina:

1. Način – Preko prozora Properties, pritiskom na „munju“, mogu se dodeliti događaji koje želimo za određenu alatku. (npr.za dugme želim događaju „klik“).Slika 5.



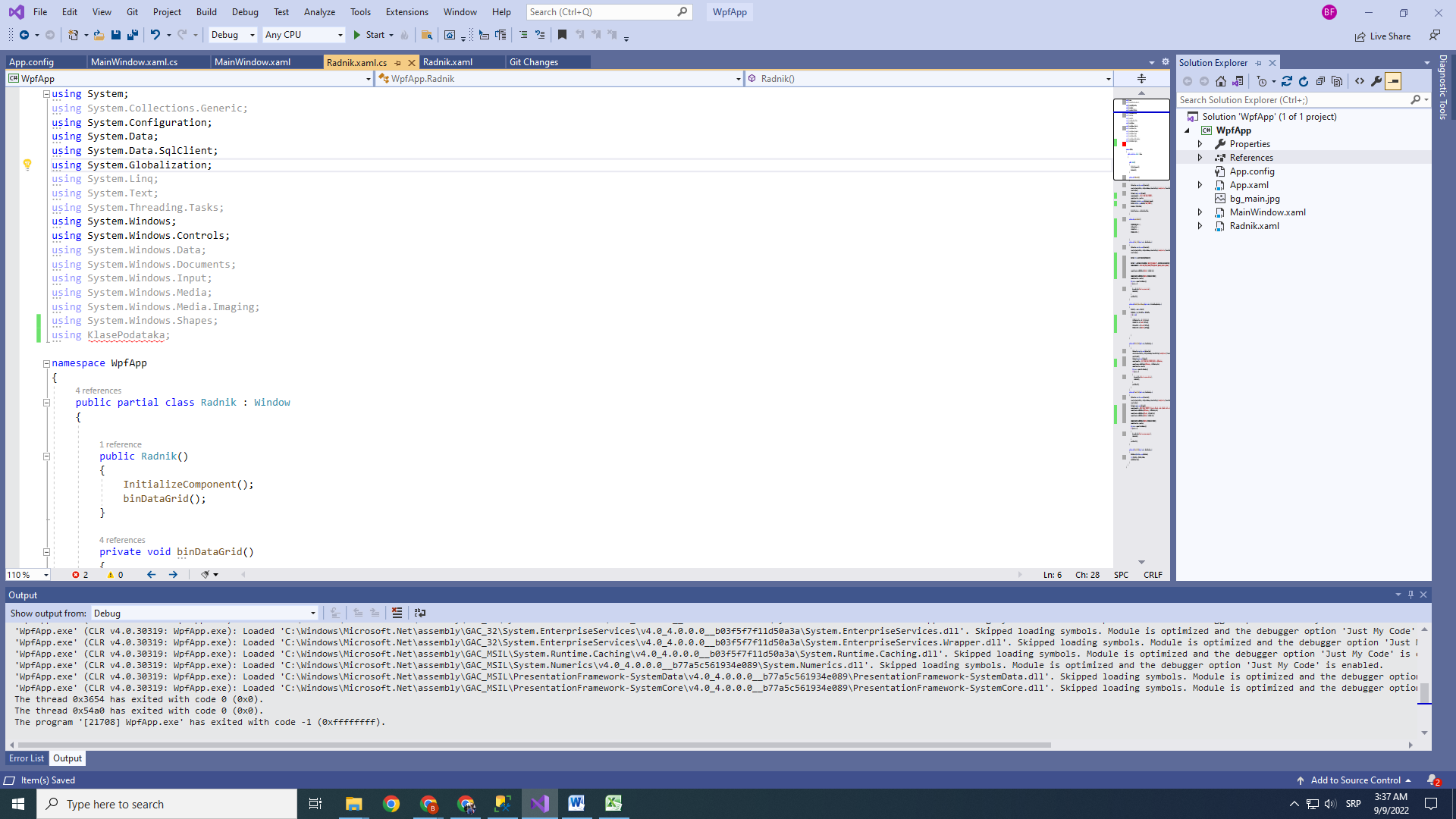
*Slika 5. Opcija za dodelu događaja grafičkom objektu*

1. Način – da se pozicioniramo na određeni alat kojem želimo da dodelimo neki događaj i da kliknemo dva puta na njega. Ovo je kraći put ako želimo da nam događaj bude „klik“.

Pošto naš projekat zahteva biblioteke klasa, da bi se one kreirale biramo poseban projekat *Class Library*. Pri otavranju tog projekta dobijamo klasu, koju sami modifikujemo u odnosu na naše potrebe. Ako želimo još klasa u datom projektu, biramo opciju *Project* -> *Add class...* (ili prečicom preko tastature, *Shift+Alt+C*).

Da bi koristili funkcionalnosti tih klasa, moramo da ih priključimo našoj aplikaciji. Pored standardnih biblioteka koje naša aplikacija automatski koristi, mi biramo i naše biblioteke koje smo napravili. Te biblioteke su *KlasaPodataka* i *SQLDBUtils*.

Dodaju se preko *References*, koji se nalazi u *Solution* *Explorer*-u. Desni klik na *References -> Add references*, ili *Project -> Add references*. Biramo ***.dll*** fajlove. Kada smo uspešno dodali u references, moramo ga napomenuti i u kodu, tačnije u *USING* delu koda, da bi nam program prepoznao biblioteku (slika 6.)



*Slika 6. Dodavanje biblioteke klasa KlasePodataka u aplikaciju*

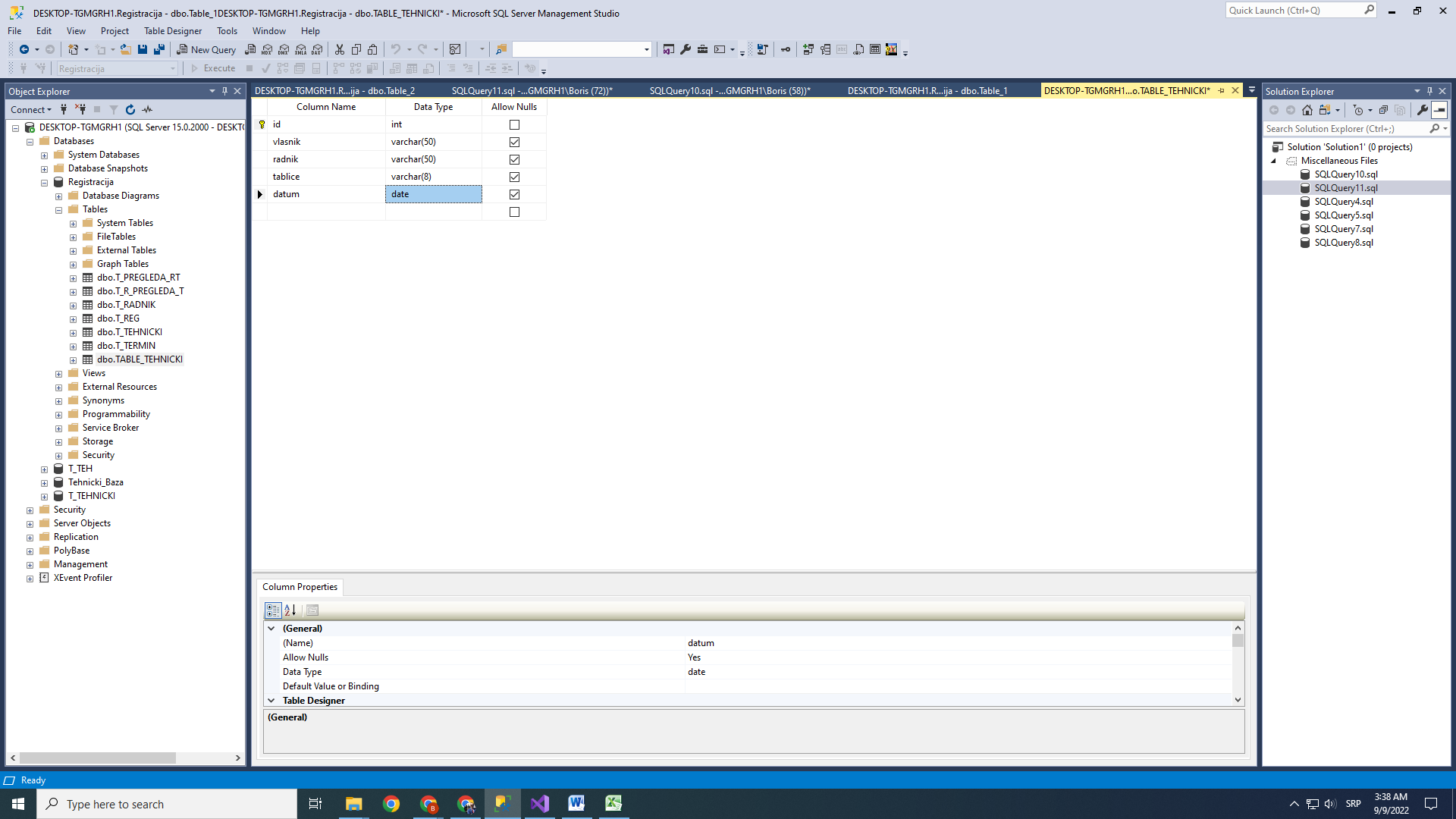
Biblioteku klasa *SQLDBUtils* dodajemo kao referencu (biblioteku) u projekat u kom se nalaza *KlasePodataka*, u njima se nalaze metode potrebne za konekciju sa bazom, čitanje iz baze, ažuriranje baze itd. Dok biblioteku *KlasePodataka* dodajemo kao referencu (biblioteku) u samoj aplikaciji.

## 3.2. Microsoft SQL Server Management Studio 18

### 3.2.1. Namena alata

SQL Server Management Studio (SSMS) je softverska aplikacija koja se koristi za konfigurisanje, upravljanje i administraciju svih komponenti u Microsoft SQL Serveru. Alat uključuje uređivače skripti i grafičke alate koji rade sa objektima i komponentama servera.

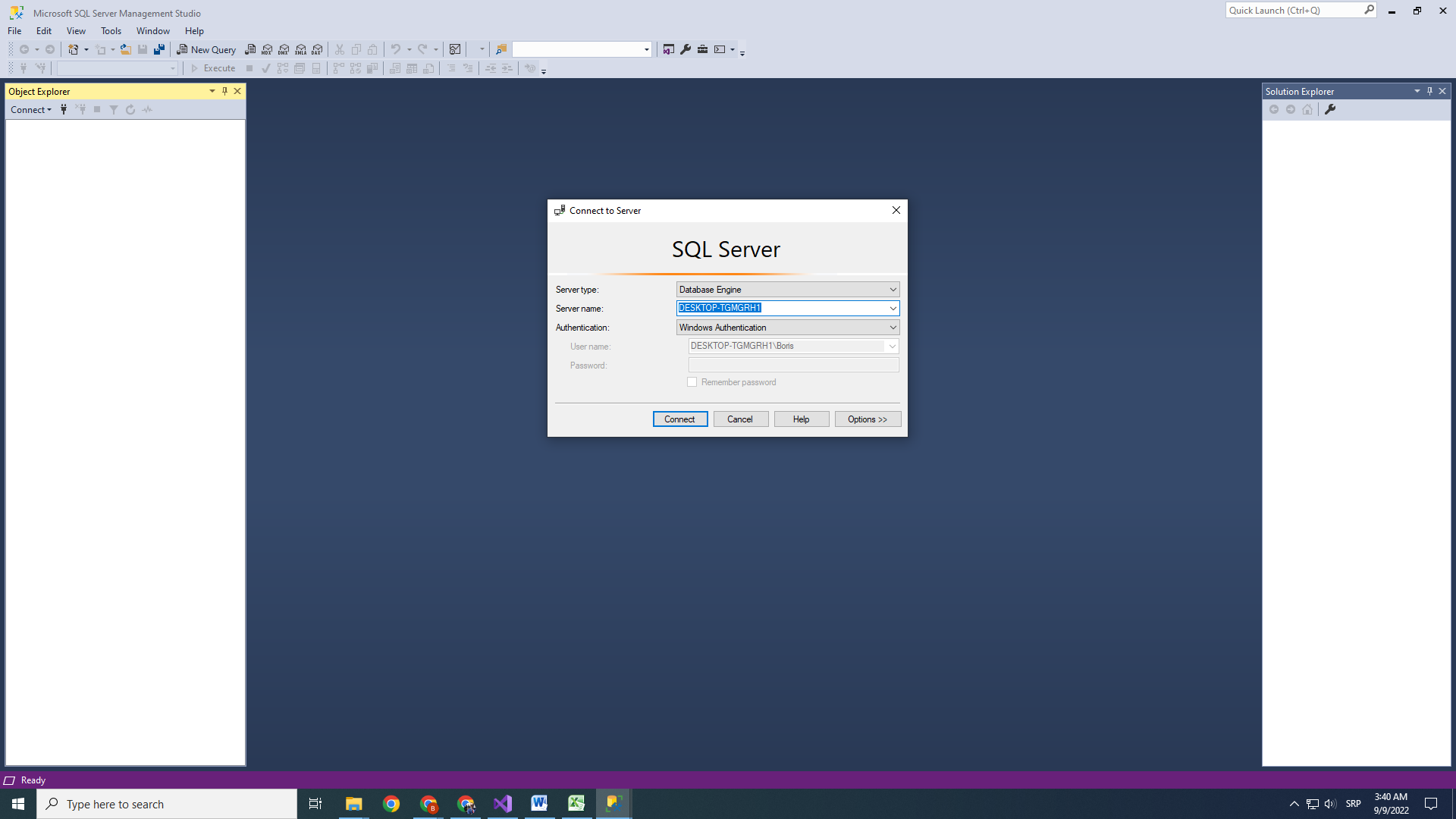
### 3.2.2. Izgled ekrana alata



*Slika 7. Izgled ekrana alata Microsoft SQL Server Management Studio 18*

Na sliki 7. možemo videti izgled tabele *TABLE\_TEHNICKI,* iz baze podataka *Registracija,* na slici se vidi kolone u datoj tabeli u sredini ekrana. Sa leve strane gore tekst predstavlja ime servera koji je potreban za konekciju naše aplikacije sa bazom podataka. Takođe sa leve strane se vidi baza podataka *Registracija* i tabela *TABLE\_TEHNICKI*.

### 3.2.3. Opis načina korišćenja

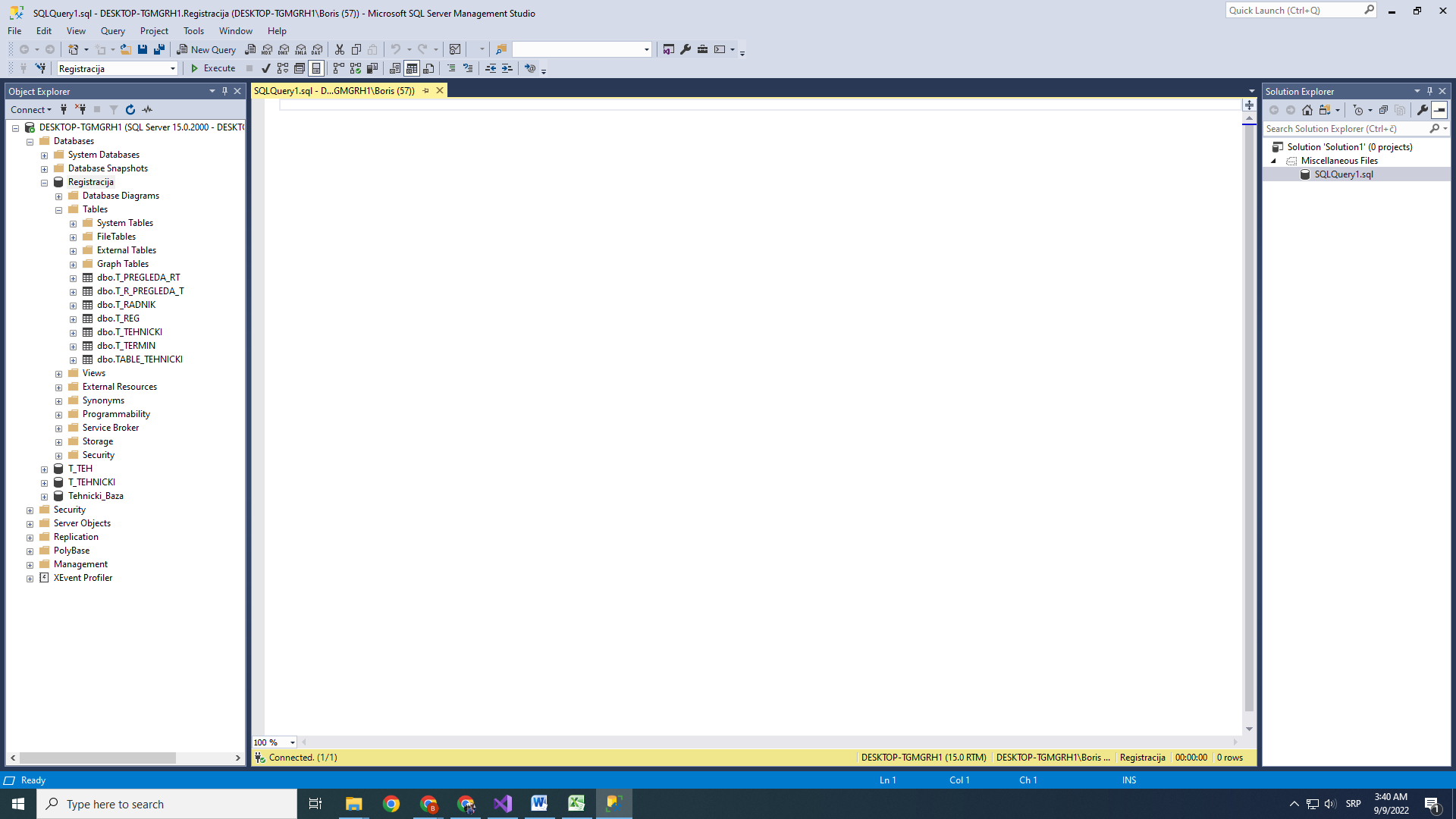


*Slika 8. Početni ekran pokretanja Microsoft SQL Server Management Studio 18*

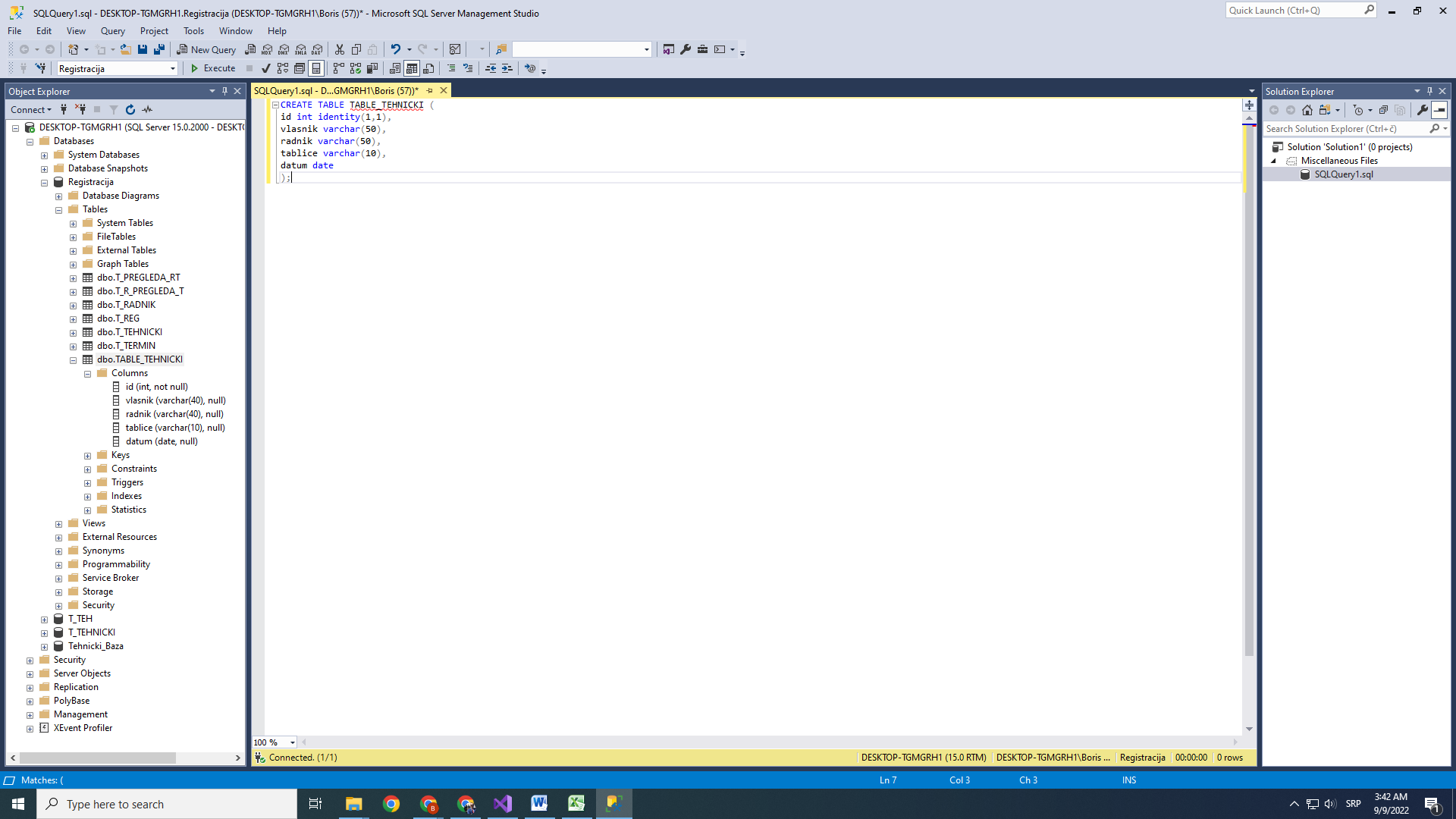
Pri pokretanju Microsoft SQL Server Management Studio-a dobija se prozor prikazan na slici 8. Pre kreiranja baze prvo se moramo konektovati na server, ukoliko je naziv servera dobar, konekcija će biti uspešna.

Zatim, kada se uspešno konektujemo, pozicioniramo se na naš naziv servera i kliknemo na *New Query*, dobićemo prazno polje prikazan na slici 9.

Jedan od načina pravljenja baze podataka je kucanje koda, koristeći sintaksu SQL jezika. Pisani kod pri kreiranju baze *TABLE\_TEHNICKI* dat je na slici 10. Kada je kod napisan pravilno klikne se na *Execute*, i ako takva baza ne postoji, baza podataka će biti uspešno napravljena (kao što se može videti na slici 7.).



*Slika 9. Postupak pravljenja baze podataka*



*Slika 10. Kod za pravljenje baze podataka*

# 4. Korisničko upustvo razvijene aplikacije

## 4.1. Namena aplikacije i osnovne funkcije

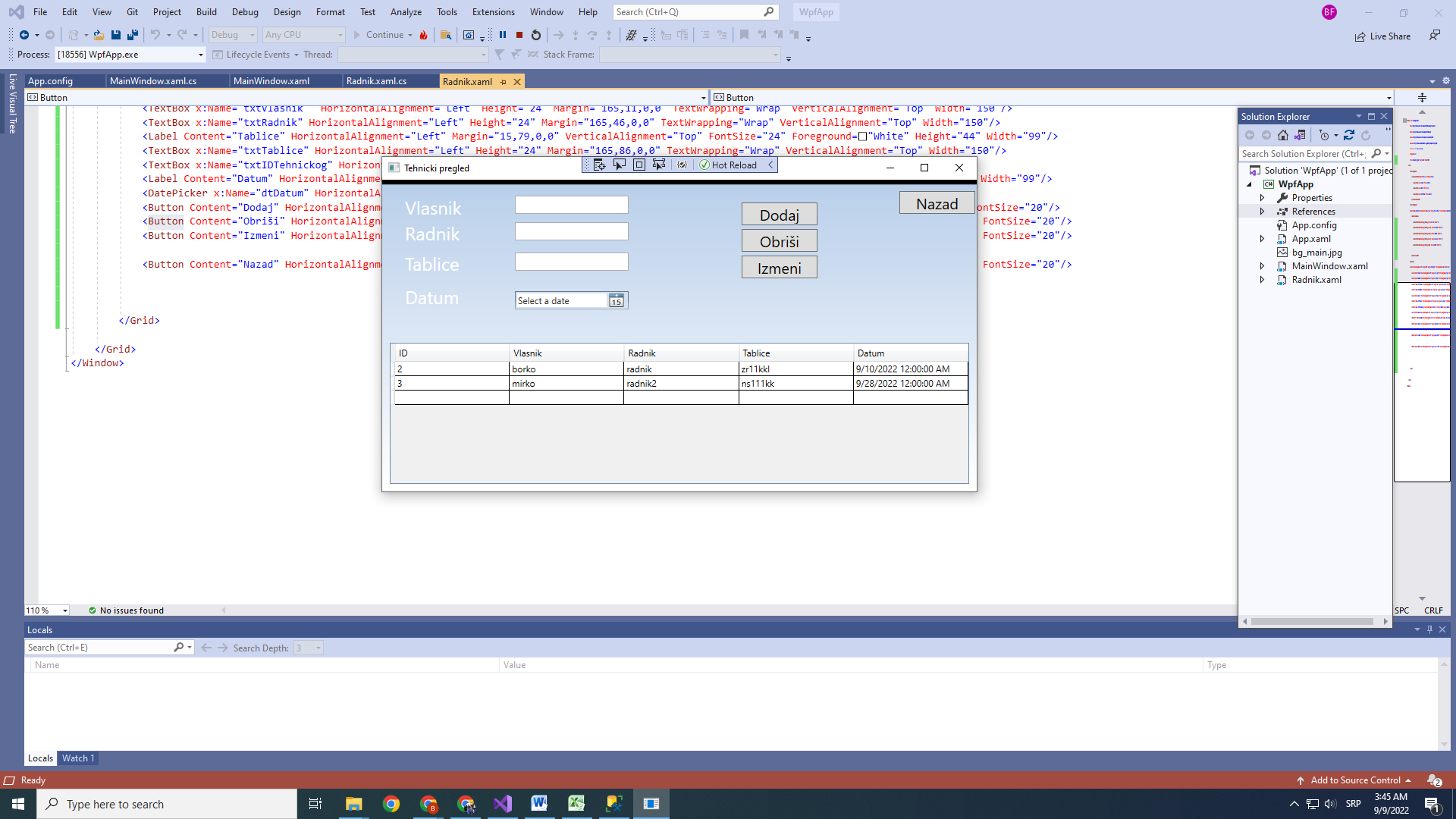
Osnovna namena desktop aplikacije je evidencija tehničkog pregleda.

Osnovne funkcije:

* Tabelarni prikaz svih podataka u bazi
* Unos novih termina tehničkog pregleda
* Odustajanje od unosa

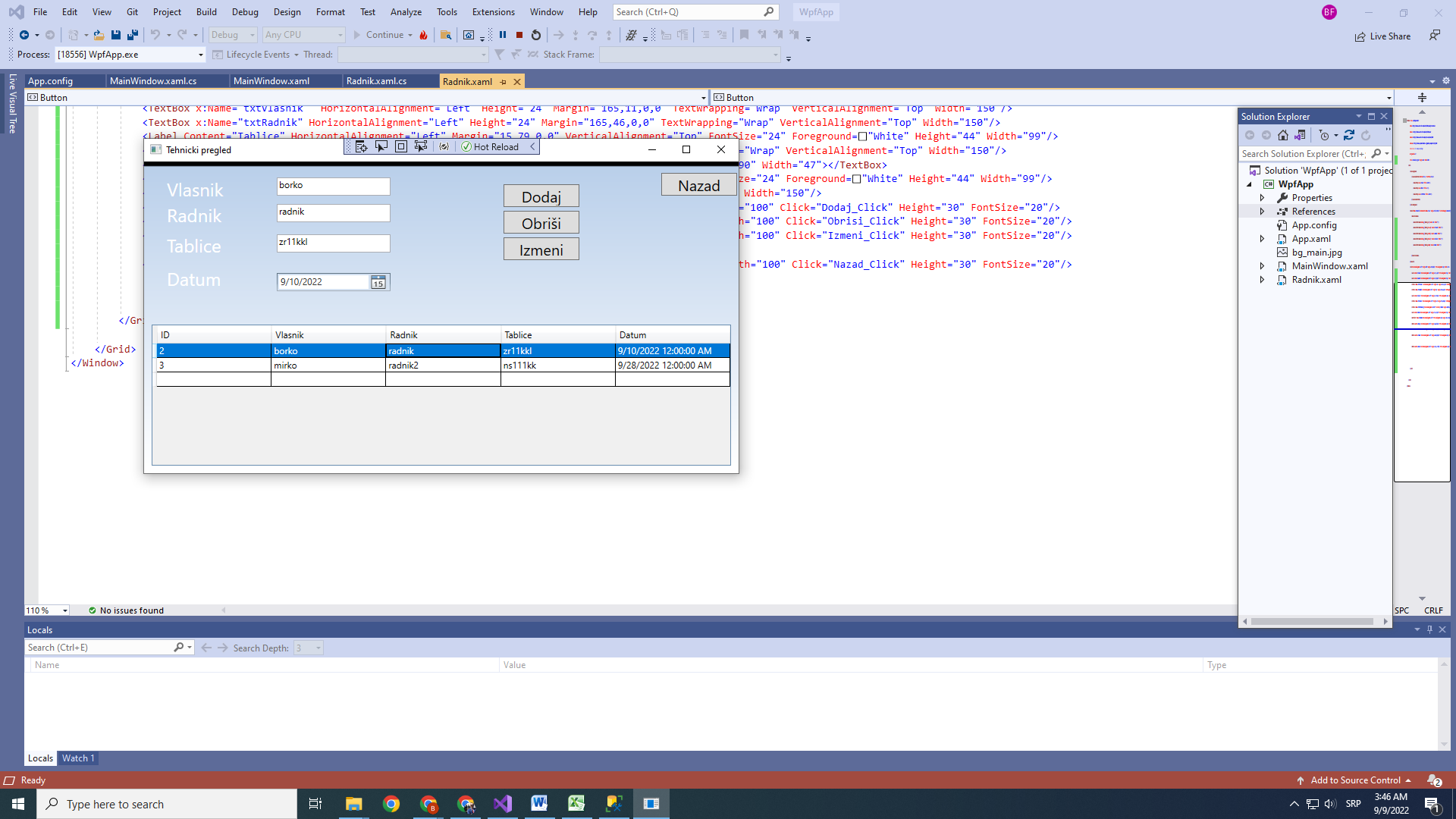
## 4.2. Ekrani i način korišćenja

Kada se pokrene aplikacija dobija se izgled sličan kao u dizajneru, ali sa učitanim podacima u tabelu. Kada pokrenemo program, dobijamo ekran prikazan na slici 11.



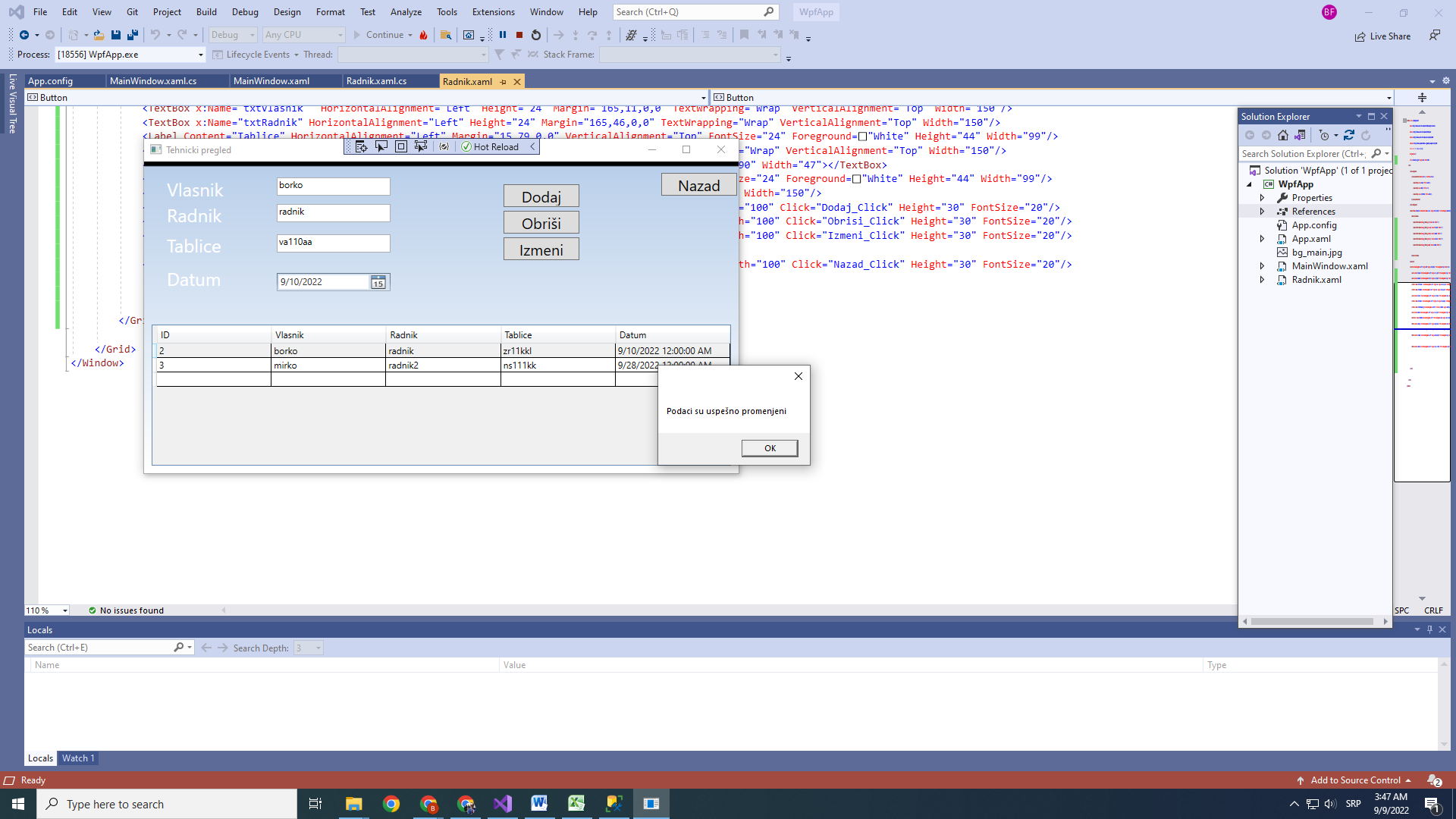
*Slika 11. Prikaz podataka nakon pokretanja*

Zatim kada popunimo sve podatke koji se traže, dobijamo sledeći ekran, prikazan na slici 12.



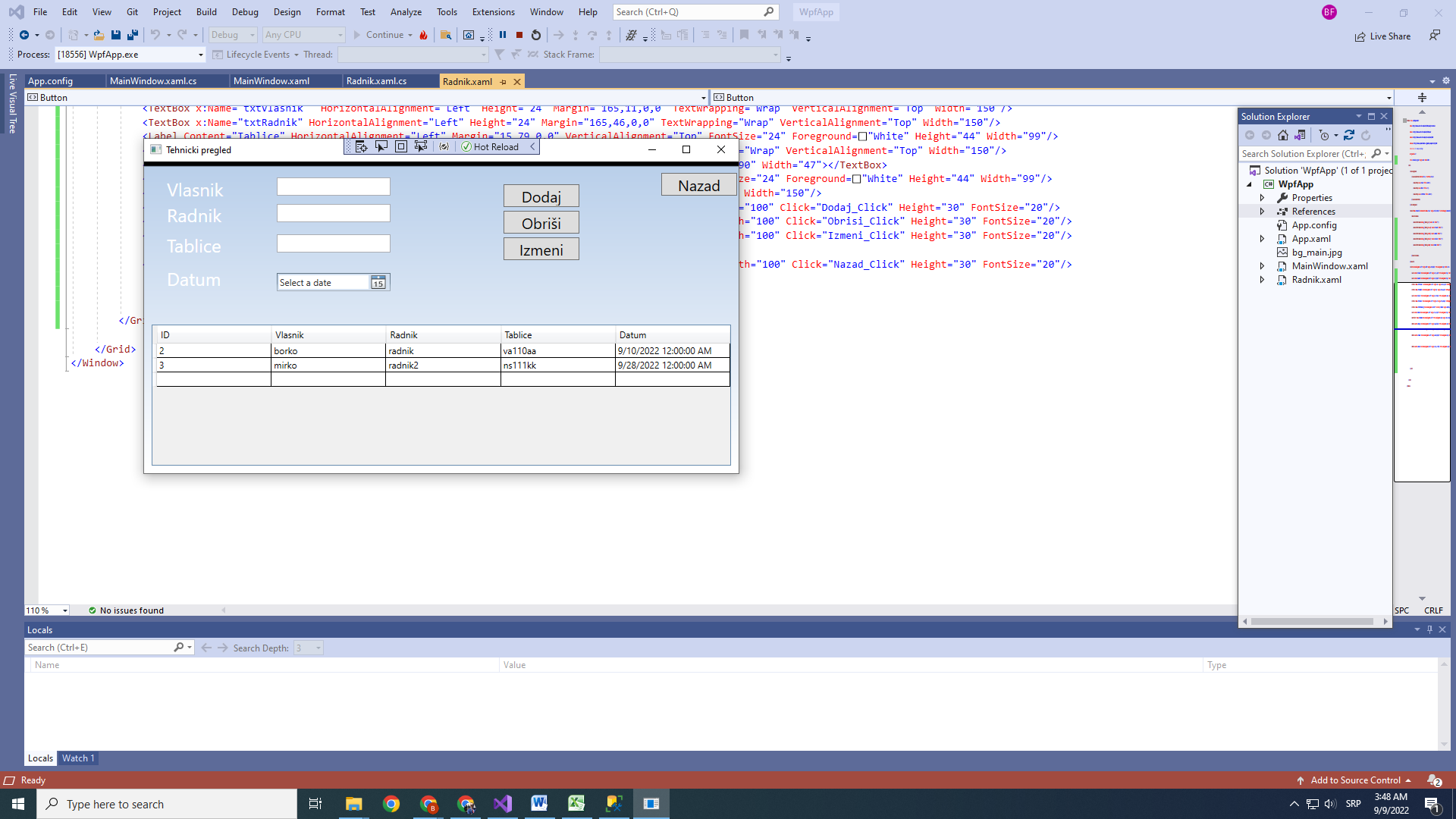
*Slika 12. Ekran nakon popunjenih podataka*

Kada se posle popunjavanja klikne na opciju Izmeni, dobija se poruka o snimanju podataka, i ti podaci se ubacuju u tabelu, slika 13.



*Slika 13. Ekran nakon pritiska na opciju Izmeni*

Izgled ekrana nakon poruke o snimanju, slika 14. Kao što se vidi na slici, podaci su uspešno dodati u bazu i prikazani u tabeli.



*Slika 14. Ekran nakon uspešno snimljenih podataka i njihov prikaz u tabeli*

# 5. Ključni elementi implementacije

## 5.1. SQL Script

CREATE DATABASE Registracija

GO

USE DATABASE Registracija

GO

CREATE TABLE TABLE\_TEHNICKI (

id int identity(1,1),

vlasnik varchar(50),

radnik varchar(50),

tablice varchar(10),

datum date

)

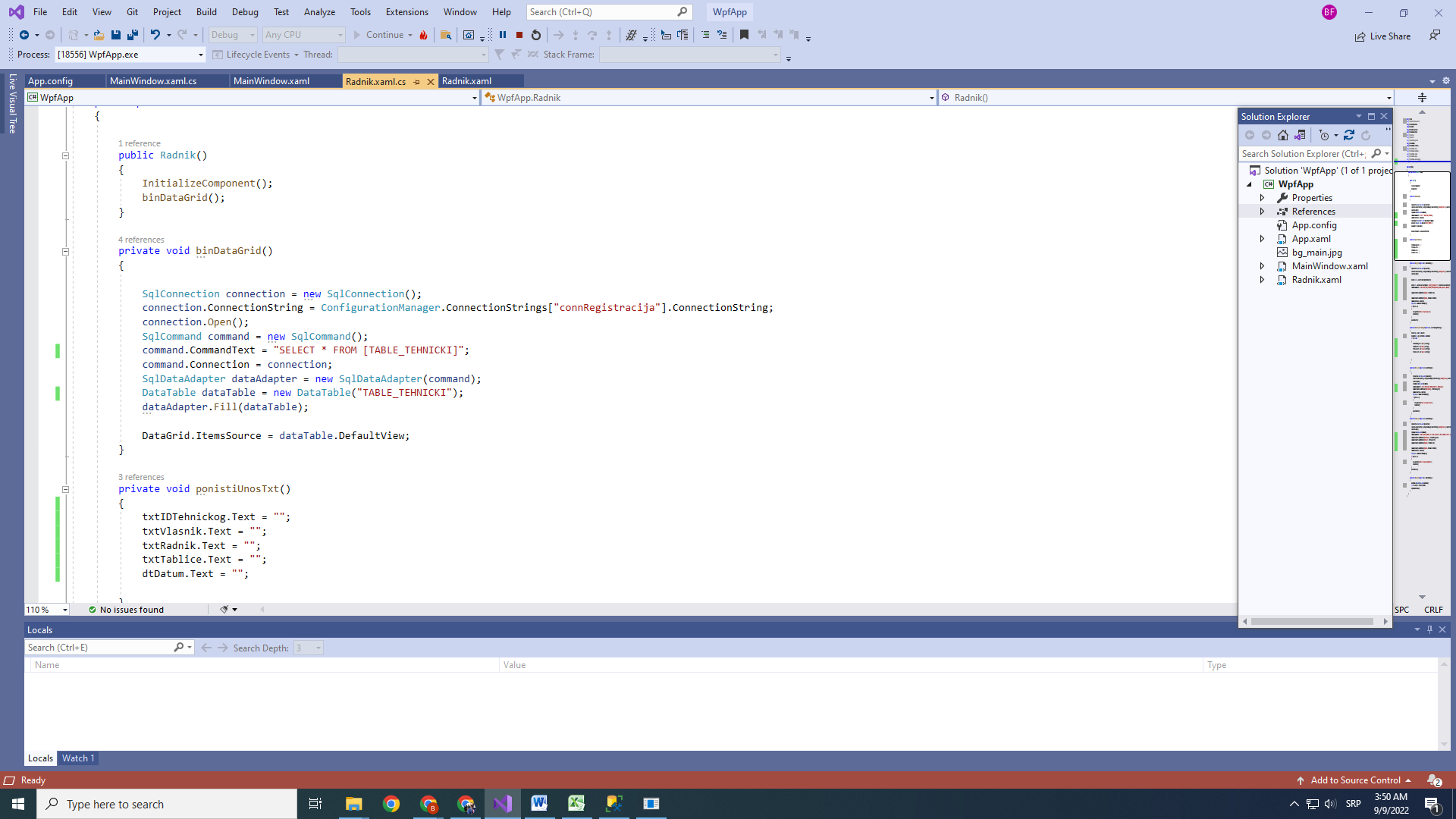
GO

## 5.2. Ključni delovi programskog koda sa objašnjenjem

**Uključivanje biblioteke klasa:**

using KlasePodataka;

**Početni događaj koji se automatski izvršava na početku** (slika 16.).



*Slika 16. Pokretanje događaja*

Kao što je dato na slici 16. napravljeni su objekti datih klasa, i ovo se prvo izvršava, kada se pokrene kod.

**Preuzimanje podataka sa korisničkog interfejsa:**

**

*Slika 17. Preuzimanje podataka unetih preko tastature*

**Snimanje podataka:**

**

*Slika 18. Snimanje unetih podataka*

**Tabelarni prikaz:**

**

*Slika 19. Kod za tabelarni prikaz podataka*

# 

# 6. Greške programskog koda

Postoje tri glavne greške koje kompajler može detektovati, a to su:

1. Sintaksa greška
2. Leksička greška
3. Sematička greška

Postoji i takozvana run-time greška, koju kompajler ne može da detektuje, ali ona može da se uvidi pri pokretanju koda, obično je to pogrešna putanja do baze, nepostojeća tabela, ili kolona u tabeli i slično.

## 6.1. Segmenti programskog koda sa greškama i korekcija

***SINTAKSNE GREŠKE***

|  |  |  |
| --- | --- | --- |
| *NEISPRAVNO* | *ISPRAVNO* | *OBJAŠNJENJE GREŠKE* |
|  |  | Nedostaje ; |
|  |  | Kada se poziva metoda, mora imati ’()’. |

***LEKSIČKE GREŠKE***

|  |  |  |
| --- | --- | --- |
| *NEISPRAVNO* | *ISPRAVNO* | *OBJAŠNJENJE GREŠKE* |
|  |  | Nije dobro napisan tip podatka |
|  |  | Varijabla *txtRadnici* ne postoji, greška pri kucanju. |

***SEMATIČKE GREŠKE***

|  |  |  |
| --- | --- | --- |
| *NEISPRAVNO* | *ISPRAVNO* | *OBJAŠNJENJE GREŠKE* |
|  |  | Promenljiva *txtRadnik* je tipa string, ne može da prima tip int. |
|  |  | U pitanju je korišćenje promenljive koja nije inicijalizovana. |

***RUN-TIME GREŠKE***

Da bi videli run-time greške, treba uneti pogrešne podatke unutar koda korisničkog interfejsa. Kompajler se neće buniti, jer je sve dobro napisano i nema gore navedenih grešaka, ali kod će “pući” kada pokušamo da unesemo pogrešan put do baze podataka ili kada pokušamo da upišemo u tabelu koja se ne nalazi u datoj bazi podataka. Na sledećim primerima prikazan je ispravan i neispravan kod.

NEISPRAVNO



ISPRAVNO



U prograsmkom kodu biblioteke klasa, *KlasaPodataka*, u klasi *clsEvidencijaDB* postavljena je fiksna putanja za konekciju sa bazom. Promenljiva *objSqlTabela* predstavlja objekat koji se instancira pomoću konstruktora koji prima dva parametara, konekciju ka bazi podataka, kao prvi parametar, i kao drugi naziv tabele u datoj bazi.

Da bismo napravili run-time grešku, daćemo naziv tabele pogrešan.

NEISPRAVNO



ISPRAVNO



## 6.2. Pravila i EBNF prikaz gramatički ispravnog programskog koda

PRAVILO: Evidencija uvek ima ; na kraju.

EBNF:

<naredba>::=<telo\_naredbe>”;”

PRAVILO: Procedura ima zagrade.

EBNF:

<procedura>::=<modifikator\_pristupa><tip<naziv\_procedure>”(“[<parameter>]”)”

PRAVILO: String promenljiva uvek dobija string vrednost.

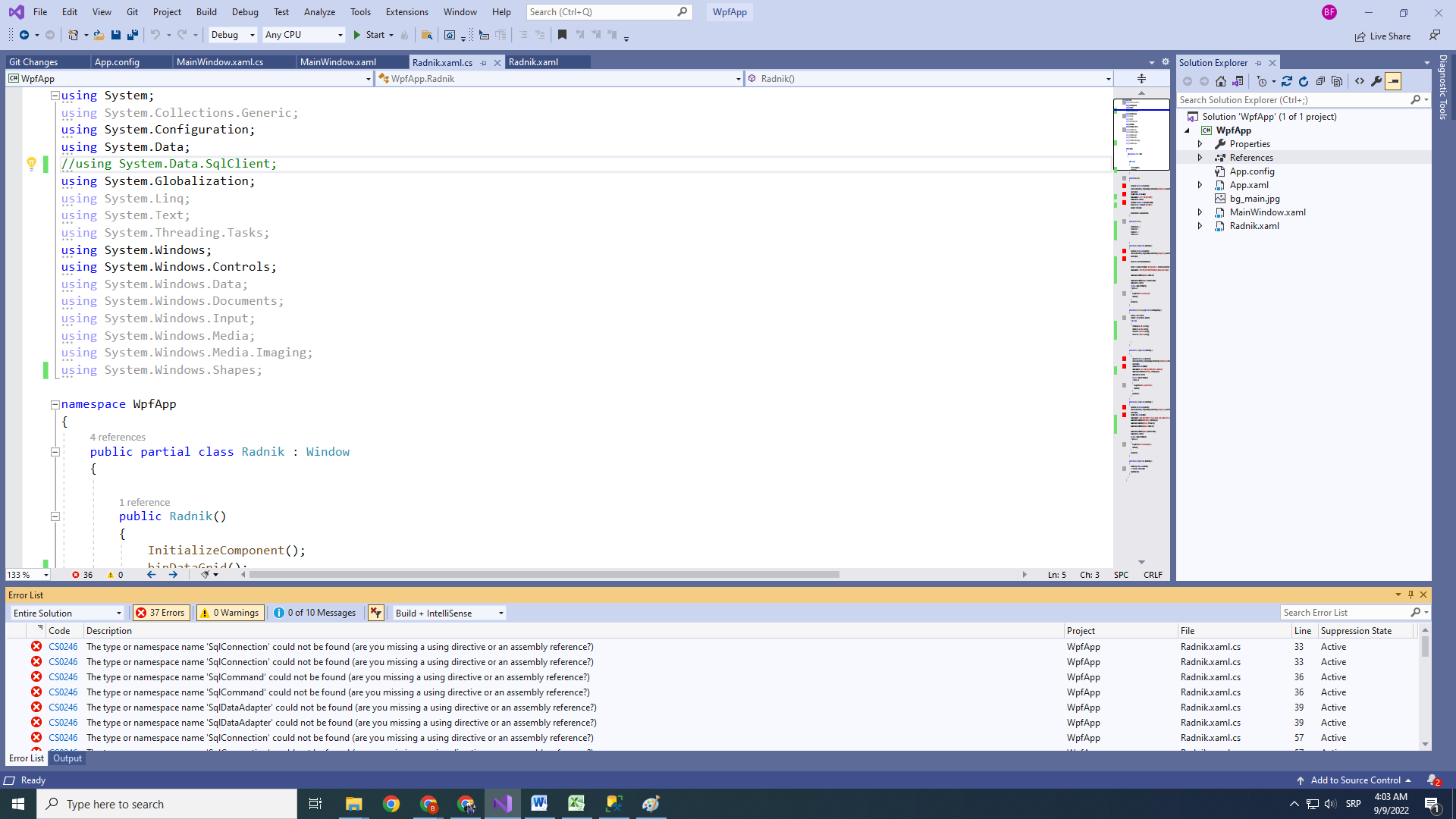
EBNF:

<string\_promenljiva>::=” “ ”<vrednost\_promenljive>” “ ”

# 7. Prikaz primene alata u detektovanju grešaka

## 7.1. Ekranski prikaz izveštaja kompajlera nad neispravnim kodom i nakon popravke

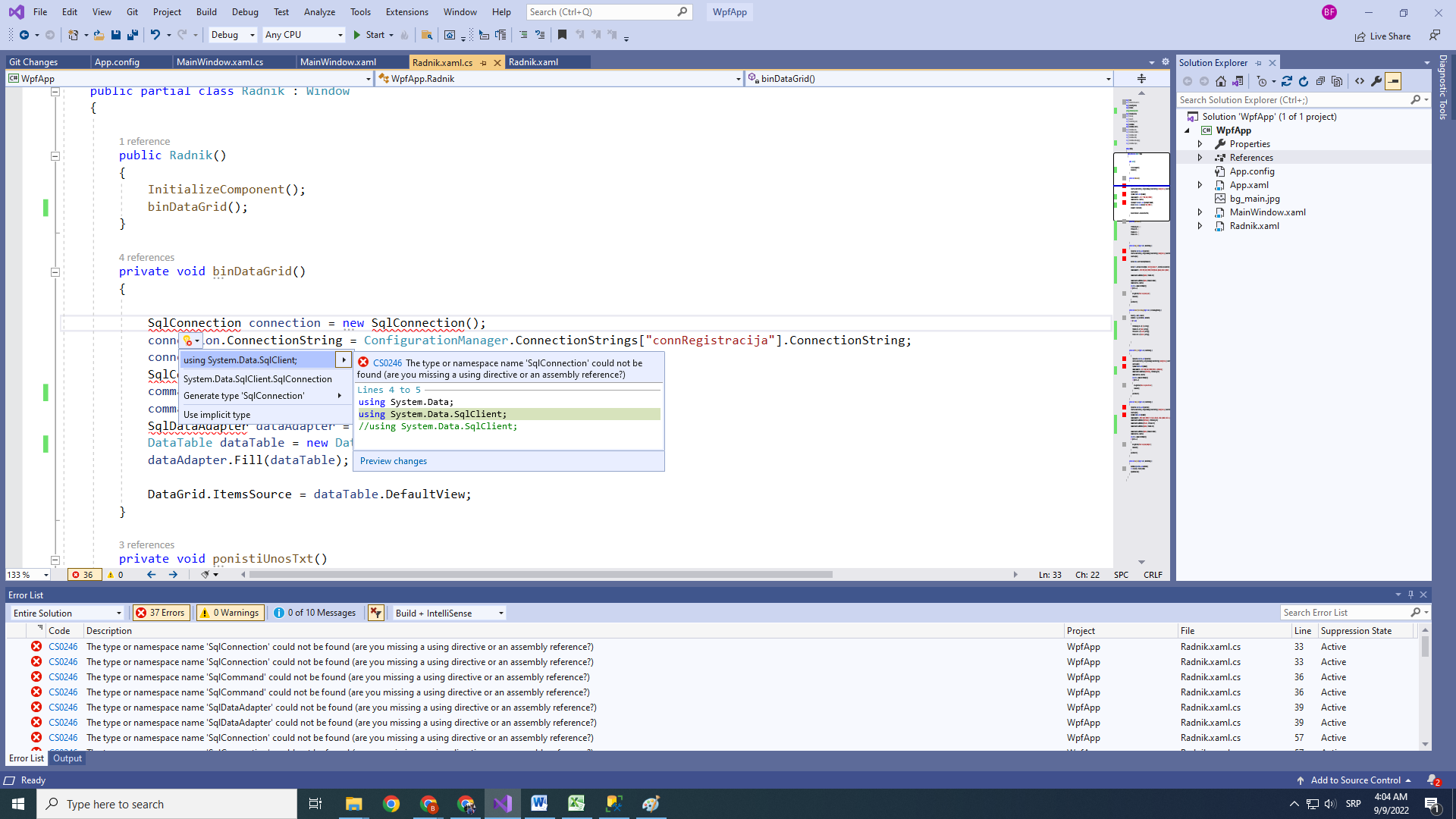
Prilikom pokretanja Build opcije iz Menu-a (*Build -> Build Solution*), dobijamo izveštaj kompajlera u okviru *Error Liste,* kao što je prikazano na slici 22.



*Slika 22. Error lista pri pokretanju Build opcije*

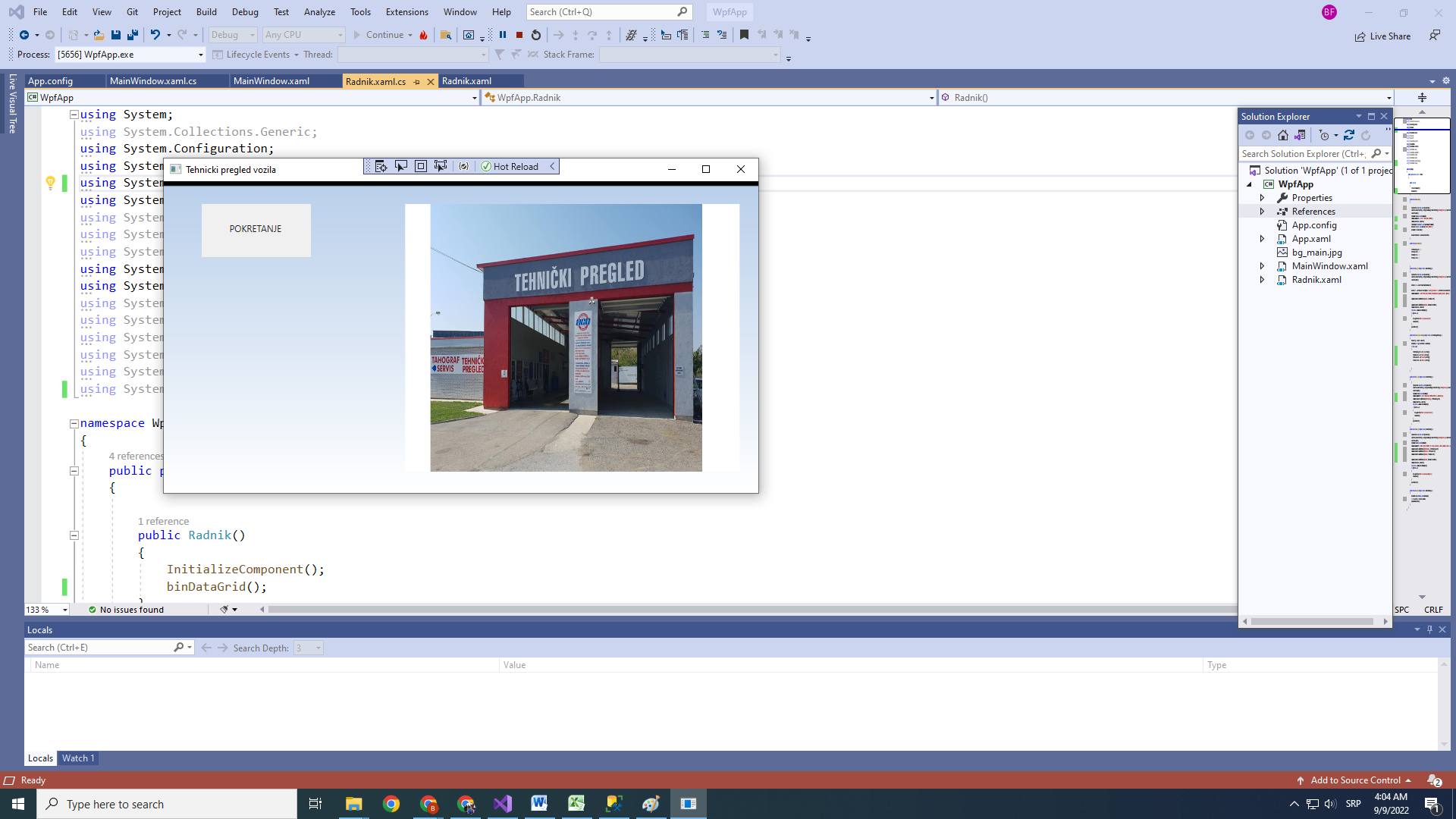
Error lista pored detaljnog opisa greške daje nam tačno mesto gde se greška pojavila, znači tačan naziv projekta, fajl u kome se javila i liniju koda. Time nam olakšava njeno traženje, a samim tim i njeno otklanjanje.

Postoji i još jedan način za pronalazak greške u kodu, a to je, da prilikom greške, kod će biti podvučen drugom bojom kako bi bio vidljiv da se tu nalazi greška. I ako s pređe mišem preko te greške, sam program nam nudi potencijalne ispravke. Pored toga, dobijamo detaljan opis greške (slika 23.).



*Slika 23. Drugi način za detektovanje greške u kodu*

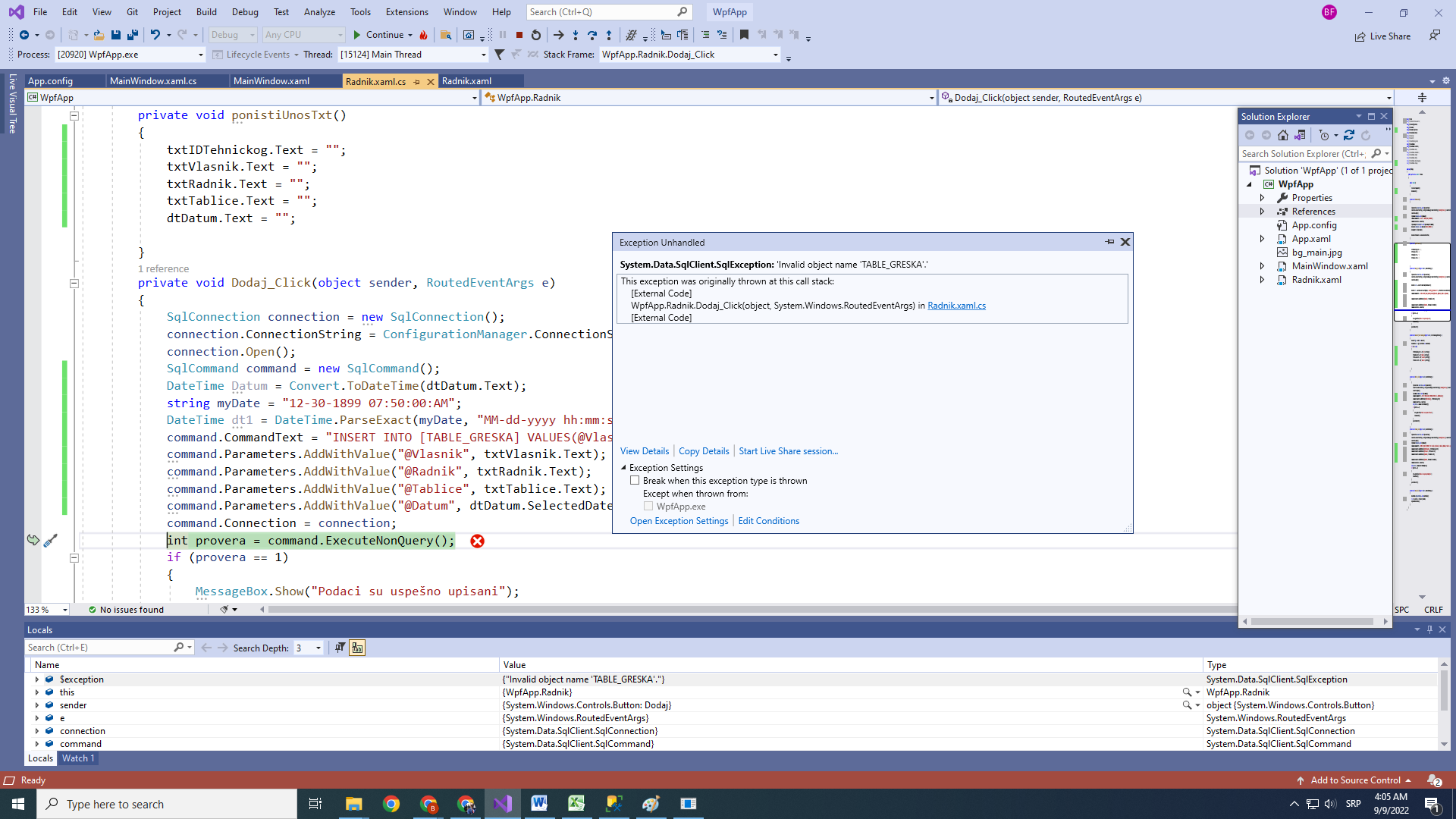
Nakon popravki grešaka, dobijamo Izveštaj: Error = 0, Build succeeded.



*Slika 24. Izgled ekrana nakon ispravljanja greški*

## 7.2. Ekranski prikaz reakcije na run-time greške

### 7.2.1. Prvi primer run-time greške



*Slika 25. Prikaz prozora za prvu run-time grešku*

Greška se nalazi u SQL upitu. Baza podataka koja je navedena ne postoji.

NEISPRAVNO

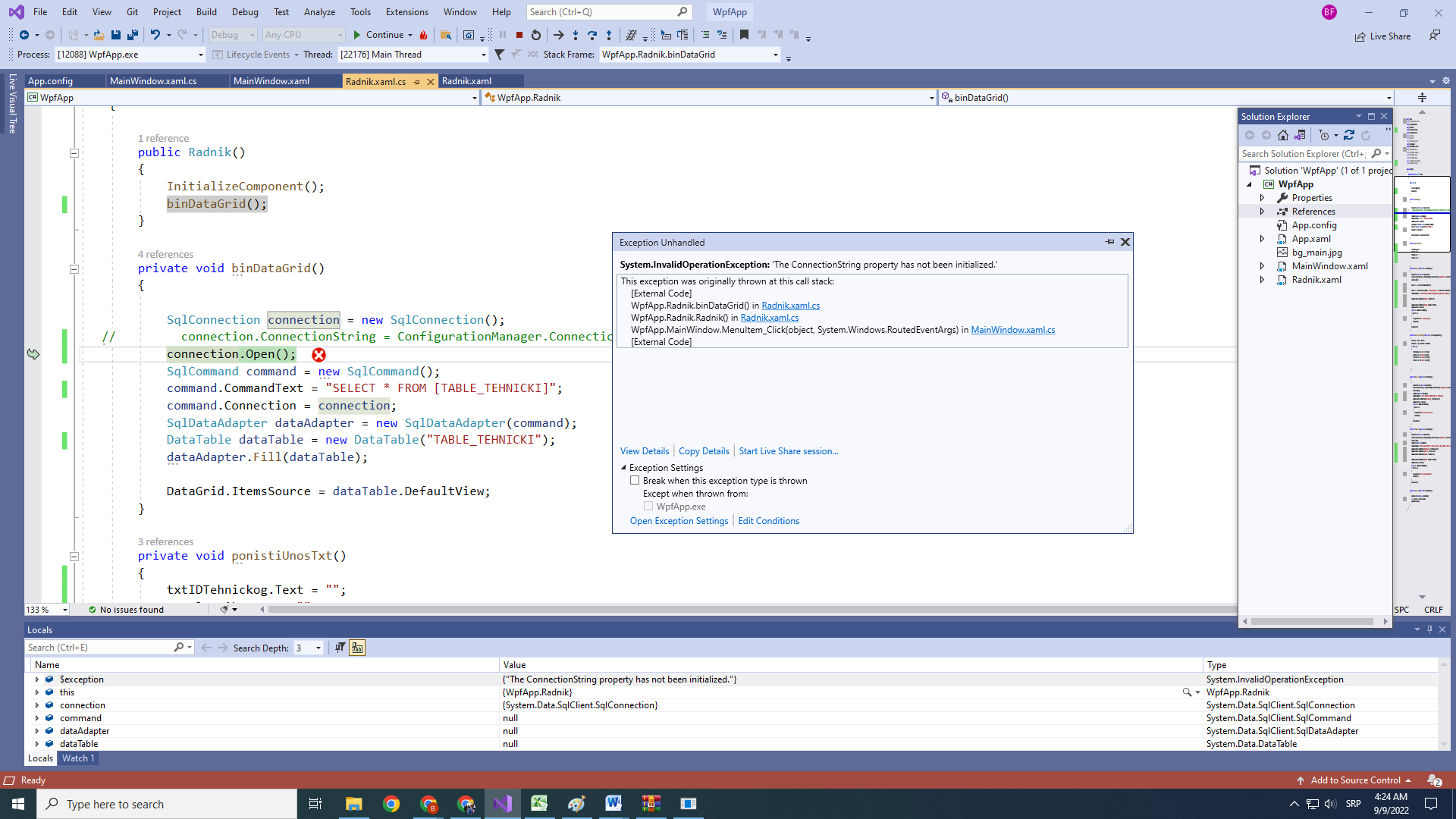


ISPRAVNO



### 

### 7.2.2. Drugi primer run-time greške



*Slika 26. Prikaz ekrana posle druge run-time greške*

Druga run-time greška predstavlja, *System.InvalidOperationException,* koja znači da pokušavamo da pristupimo bazi koja nije online (servis SQL Servera je u ovom trenutku ugašen).

# 

# 8. Zaključak

Cilj ovog seminarskog rada je da predstavi izradu aplikacije u Microsoft Visual Studio 2019 .NET framework, kao i povezivanje aplikacije sa bazom podataka, a alat korišćen za izradu baze je Microsoft SQL Server. Poseban akcenat ovog seminarskog rada je na objektnom- orijentisanom programiranju.

Rad je rađen iz predmeta Programski prevodioci, i kao takav sadrži greške koje nam prijavljuje kompajler (prevodilac), pored grešaka, u radu je prikazano i njihovo otklanjanje. Na kraju, aplikacija je funkcionalna i bez grešaka.

Postoji mogućnost dorade same aplikacije, kao na primer, dodavanje opcije za registraciju vozila. Još jedna od mogućnosti jeste i omogućavanje da se pre prikazivanja glavnog prozora, omogući prijavljivanje korisnika, zavisno od toga da li je radnik na tehničkom pregledu mušterija koji želi da zakaže tehnički pregled.

Ovo su samo neki od načina poboljšanja date aplikacije.

# 9. Literatura

[1] <http://www.tfzr.uns.ac.rs/Content/files/0/99_PRAKTIKUM_sa13poglavlja.pdf>

[2]<http://www.tfzr.uns.ac.rs/Content/files/0/PripremniMaterijalZaKolokvijumDOPUNJENO3.pdf>

[3] <https://bs.wikipedia.org/wiki/Microsoft_Visual_Studio>

[4] <https://en.wikipedia.org/wiki/SQL_Server_Management_Studio>

# 10. Listing

## 10.1. Listing korisničkog interfejsa

using System;

using System.Collections.Generic;

using System.Configuration;

using System.Data;

using System.Data.SqlClient;

using System.Globalization;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Shapes;

using KlasePodataka;

namespace WpfApp

{

public partial class Radnik : Window

{

public Radnik()

{

InitializeComponent();

binDataGrid();

}

private void binDataGrid()

{

SqlConnection connection = new SqlConnection();

connection.ConnectionString = ConfigurationManager.ConnectionStrings["connRegistracija"].ConnectionString;

connection.Open();

SqlCommand command = new SqlCommand();

command.CommandText = "SELECT \* FROM [TABLE\_TEHNICKI]";

command.Connection = connection;

SqlDataAdapter dataAdapter = new SqlDataAdapter(command);

DataTable dataTable = new DataTable("TABLE\_TEHNICKI");

dataAdapter.Fill(dataTable);

DataGrid.ItemsSource = dataTable.DefaultView;

}

private void ponistiUnosTxt()

{

txtIDTehnickog.Text = "";

txtVlasnik.Text = "";

txtRadnik.Text = "";

txtTablice.Text = "";

dtDatum.Text = "";

}

private void Dodaj\_Click(object sender, RoutedEventArgs e)

{

SqlConnection connection = new SqlConnection();

connection.ConnectionString = ConfigurationManager.ConnectionStrings["connRegistracija"].ConnectionString;

connection.Open();

SqlCommand command = new SqlCommand();

DateTime Datum = Convert.ToDateTime(dtDatum.Text);

string myDate = "12-30-1899 07:50:00:AM";

DateTime dt1 = DateTime.ParseExact(myDate, "MM-dd-yyyy hh:mm:ss:tt", CultureInfo.InvariantCulture);

command.CommandText = "INSERT INTO [TABLE\_TEHNICKI] VALUES(@Vlasnik, @Radnik, @Tablice, @Datum)";

command.Parameters.AddWithValue("@Vlasnik", txtVlasnik.Text);

command.Parameters.AddWithValue("@Radnik", txtRadnik.Text);

command.Parameters.AddWithValue("@Tablice", txtTablice.Text);

command.Parameters.AddWithValue("@Datum", dtDatum.SelectedDate);

command.Connection = connection;

int provera = command.ExecuteNonQuery();

if (provera == 1)

{

MessageBox.Show("Podaci su uspešno upisani");

binDataGrid();

}

ponistiUnosTxt();

}

private void DataGrid\_SelectionChanged(object sender, SelectionChangedEventArgs e)

{

DataGrid dg = sender as DataGrid;

DataRowView dr = dg.SelectedItem as DataRowView;

if (dr != null)

{

txtIDTehnickog.Text = dr["id"].ToString();

txtRadnik.Text = dr["radnik"].ToString();

txtVlasnik.Text = dr["vlasnik"].ToString();

txtTablice.Text = dr["tablice"].ToString();

dtDatum.Text = dr["datum"].ToString();

}

}

private void Obrisi\_Click(object sender, RoutedEventArgs e)

{

{

SqlConnection connection = new SqlConnection();

connection.ConnectionString = ConfigurationManager.ConnectionStrings["connRegistracija"].ConnectionString;

connection.Open();

SqlCommand command = new SqlCommand();

command.CommandText = "DELETE FROM [TABLE\_TEHNICKI] WHERE id = @IDTehnickog";

command.Parameters.AddWithValue("@IDTehnickog", txtIDTehnickog.Text);

command.Connection = connection;

int provera = command.ExecuteNonQuery();

if (provera == 1)

{

MessageBox.Show("Podaci su uspešno obrisani");

binDataGrid();

}

ponistiUnosTxt();

}

}

private void Izmeni\_Click(object sender, RoutedEventArgs e)

{

SqlConnection connection = new SqlConnection();

connection.ConnectionString = ConfigurationManager.ConnectionStrings["connRegistracija"].ConnectionString;

connection.Open();

SqlCommand command = new SqlCommand();

command.CommandText = "UPDATE [TABLE\_TEHNICKI] SET vlasnik = @Vlasnik, radnik = @Radnik, tablice = @Tablice, datum = @Datum WHERE id = @IDTehnickog";

command.Parameters.AddWithValue("@IDTehnickog", txtIDTehnickog.Text);

command.Parameters.AddWithValue("@Vlasnik", txtVlasnik.Text);

command.Parameters.AddWithValue("@Radnik", txtRadnik.Text);

command.Parameters.AddWithValue("@Tablice", txtTablice.Text);

command.Parameters.AddWithValue("@Datum", dtDatum.SelectedDate);

command.Connection = connection;

int provera = command.ExecuteNonQuery();

if (provera == 1)

{

MessageBox.Show("Podaci su uspešno promenjeni");

binDataGrid();

}

ponistiUnosTxt();

}

private void Nazad\_Click(object sender, RoutedEventArgs e)

{

MainWindow objMainWindow = new MainWindow();

this.Visibility = Visibility.Hidden;

objMainWindow.Show();

}

}

}

## 10.2. Konekcija sa bazom podataka

<?xml version="1.0" encoding="utf-8" ?>

<configuration>

<startup>

<supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.5" />

</startup>

<connectionStrings>

<add name="connRegistracija" connectionString="Data Source=DESKTOP-TGMGRH1;Initial Catalog=Registracija;Integrated Security=True"/>

</connectionStrings>

</configuration>