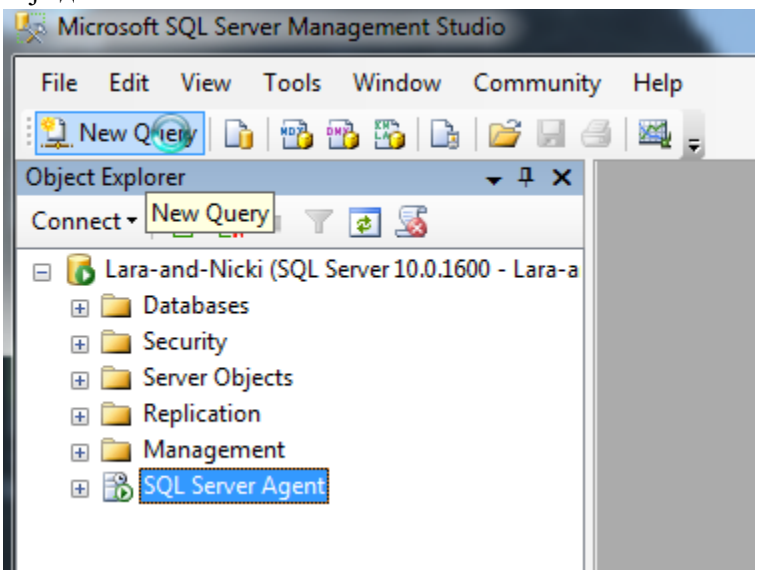
XVIII. ПРИСТУП БАЗАМА ПОДАТАКА

18.1 Упит на бази података уз коришћење отворене конекције

Процес коришћења упита подразумева добијање података из извора података (*data source)* као што је база података. У даљем тексту су дати неопходни кораци за добијање резултата упита са отвореном конекцијом.

1. Креирати конекциони објект *Connection*
2. Креирати командни објект *Command*
3. Креирати *reader* објект *DataReader*
4. Одредити скуп карактера за повезивање (*connection string)* на конекциони објект
5. Одредити који ће конекциони објект да се користи са командним објектом
6. Одредити командни текст (*CommandText*) који ће се извршавати
7. Додати вредности командним параметрима, ако они постоје.
8. Отворити конекцију (*Connection*)
9. Извршити *DataReade*r
10. Прочитати сваки ред из резултујућег скупа података
11. Затворити *Reader*
12. Затворити конекцију (*Connection)*

Да би креирали базу података потребно је покренути *"Microsoft Management Studio"* и отворити "*New Query*", као што је дато сликом 18.1.



Слика 18.1: Отварање новог упита

За креирање нове базе података извршава се следећи скрипт:

CREATE DATABASE FAKULTET

GO

Следећим скриптом се креира нова табела:

CREATE TABLE Studenti

(

StudentID int identity primary key,

Ime varchar(50),

Prezime varchar(50),

Pol varchar(10),

Starost int,

Adresa varchar(50)

)

GO

Затим се уносе подаци коришћењем следећег скрипта:

INSERT INTO Studenti (Ime, Prezime, Pol, Starost, Adresa)

VALUES

('Dragan', 'Popovic', 'Muski', 18, 'Drinciceva 3'),

('Marjan', 'Nenadovic', 'Muski', 19, 'Bulevar AVNOJA 123'),

('Dejan', 'Trkulja', 'Muski', 17, 'Knez Mihajlova 6'),

('Ana', 'Novakovic', 'Zenski', 18, 'Safarikova 8'),

('Milan', 'Jovanovic', 'Muski', 18, 'Hilandarska 2'),

('Jasna', 'Ivosevic', 'Zenski', 19, 'Vojvode Vuka 10'),

('Mila', 'Milosevic', 'Zenski', 19, 'Majke Jevrosime 54'),

('Sanja', 'Preradovic', 'Zenski', 18, 'Cika Ljubina 22'),

('Aleksandra', 'Franges', 'Zenski', 20, 'Njegoseva 14'),

('Jovana', 'Majic', 'Zenski', 18, 'Baba Visnjina 36')

GO

Када је креирана база података и табела, потребно је отворити *Visual Studio. NET* и креирати нови пројекат, а на форму додати *ListBox* контролу и подесити *Name* својство на *studentiListBox*.
Кликнути два пута на форму и написати процедуру за *Form_Load* догађај, при чему треба водити рачуна да је унета референца *using System.Data.SqlClient*.

Код за обраду догађаја *Form_Load* је дат са:

using System.Data;

using System.Drawing;

```
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Data.SqlClient;
namespace PristupBazi
{
public partial class Form1 : Form
{
public Form1()
{
InitializeComponent();
}
private void Form1_Load(object sender, EventArgs e)
{
SqlConnection connection = new SqlConnection();
SqlCommand command = new SqlCommand();
SqlDataReader reader;
connection.ConnectionString = @"Data Source=lara-and-nicki;Initial Catalog=Fakultet;"
+ "Integrated Security=SSPI";
command.Connection = connection;
command.CommandText = "SELECT Ime, Prezime, Starost FROM Studenti";
connection.Open();
reader = command.ExecuteReader();
while (reader.Read())
{
string ime = reader["Ime"].ToString();
string prezime = reader["Prezime"].ToString();
int starost = Convert.ToInt32(reader["Starost"]);              studentiListBox.Items.Add(String.Format("{0}
{1}, {2}",ime,prezime,starost));
}
reader.Close();
connection.Close();
}
```

}

Линија кода *SqlConnection connection = new SqlConnection(),* креира *Connection* објект. Следећа линија кода *SqlCommand command = new SqlCommand(),* креира командни објект, док наредна линија кода *SqlDataReader reader,* декларише *reader* објект. Затим се конекцији додељује одговарајући конекциони стринг преко кода:

connection.ConnectionString = @"Data Source=lara-and-nicki;Initial Catalog=Fakultet;"

+ "Integrated Security=SSPI";.

Овај конекциони стринг користи серверску инстанцу и базу *Fakultet*, као иницијалну базу. У линији кода:

command.Connection = connection;

командном објекту се додељује конекција коју ће користити *SQL* команда додељена *CommandText* стрингу, *Command* објекта преко кога се одређује да желимо да прикажемо све податке из табеле *Studenti*, из колона *Ime, Prezime, Starost.* Затим се отвара конекција коришћењем *DbConnection.Open()* метода. У следећој линији кода креира се *DbDataReader* инстанца коришћењем *DbCommand.ExecuteReader()* метода и додељује се *DbDataReader* објект променљивој која се касније користи. Врши се итерација сваког реда резултујућег скупа који је добијен извршавањем *SELECT* тврђења командног објекта *Command* . Користи се *DbDataReader.Read()* метод да би се добио први ред резултујућег скупа. Ако је доступан најмање један ред, *Read()* ће вратити вредност *true* и петља ће се наставити. Онда се додељују вредности сваке колоне одговарајућим променљивама. Користи се индексер за *DbDataReader* објект који прихвата стринг који представља име колоне. Затим се врши претварање резултата у одговарајуће типове података, као што се врши конвертовање садржаја *reader["Age"]* у *integer* и чува се у *integer* променљивој. Подаци се затим додају *Items* колекцији *ListBox* контроле. После прве петље, *Read*() метод ће се поново извршити и добити нови ред у резултујућем скупу. Ако нема више записа, онда ће *Read()* вратити вредност *false* и изаћи из петље. Када се изађе из петље, затвара се *DataReader* објект а исто тако и конекција. Коришћењем *while* петље у коду, извршава се *Read()* method све док постоји и последњи податак. Сви подаци се додају *ListBox* контроли и приказују кориснику.

Код се може упростити ако се комбинују поједини кораци као што је дато следећим кодом:

```
SqlConnection connection = new SqlConnection(@"Data Source=lara-and-nicki;" +
"Initial Catalog=Fakultet;Integrated Security=SSPI");
SqlCommand command = new SqlCommand(
"SELECT Ime, Prezime, Starost FROM Studenti", connection);
```

Може се користити и *using* тврђење у коду као што је дато са:

```
using (SqlConnection connection = new SqlConnection())
{
SqlCommand command = new SqlCommand();
SqlDataReader reader;
connection.ConnectionString = @"Data Source=lara-and-nicki;Initial Catalog=Fakultet
+ "Integrated Security=SSPI";
command.Connection = connection;
command.CommandText = "SELECT Ime, Prezime, Starost FROM Studenti";
connection.Open();
reader = command.ExecuteReader();
while (reader.Read())
{
string ime = reader["Ime"].ToString();
string prezime= reader["Prezime"].ToString();
int starost = Convert.ToInt32(reader["Starost"]);
studentiListBox.Items.Add(String.Format("{0} {1}, {2}",                    ime, prezime, starost))
}
reader.Close();
}
```

Целокупни процес повезивања се налази између *using* блока. На прву линију кода поставља се декларација и иницијализација *Connection* објекта. Ово сигнализира *using* блоку да одмах уништи конекцију када изађе из блока. Ако извршавање доспе до завршних затворених заграда *using* блока, тада ће конекција бити уништена, и као резултат она ће бити аутоматски затворена, а то је разлог зашто се може изоставити позив метода *DbCommand.Close()*.

Пример добре праксе је постављање конекционог процеса, унутар блока *try catch finally*, што обезбеђује руковање грешкама базе или изузецима који се могу десити за време извршавања програма, као што је дато следећим кодом:

```
try
{
```

```
connection.Open();
reader = command.ExecuteReader();
while (reader.Read())
{
string ime= reader["Ime"].ToString();
string prezime= reader["Prezime"].ToString();
int starost = Convert.ToInt32(reader["Starost"]);
studentiListBox.Items.Add(String.Format("{0} {1}, {2}",
ime, prezime, starost));
}
}
catch (SqlException e)
{
MessageBox.Show(e.Message);
}
finally
{
reader.Close();
connection.Close();}
```

У *catch* блоку користи се *SqlException* класа што представља *SQL Server* верзију провајдера *DbException*, да би се руковало грешкама базе које се могу појавити. Затим се приказују поруке грешке коришћењем *Message* својства. Код који затвара објекте *DataReader* и *Connection* поставља се унутар *finally* блока.

18.2 Упит на бази података без отварања конекције

Коришћењем *DataAdapter* класе и *DataSet* класе може се вршити упит базе података без држања отворене конекције.Објекат *DataAdapter* извршава команде ка бази података и пуни резултате добијене извршавањем упита, унутар табеле објекта *DataSet*. Затим се може приступити сваком податку коришћењем својства *Rows* објекта *DataTable* навођењем имена колона. Основни кораци за добијање резултата упита на бази података без отварања конекције су:

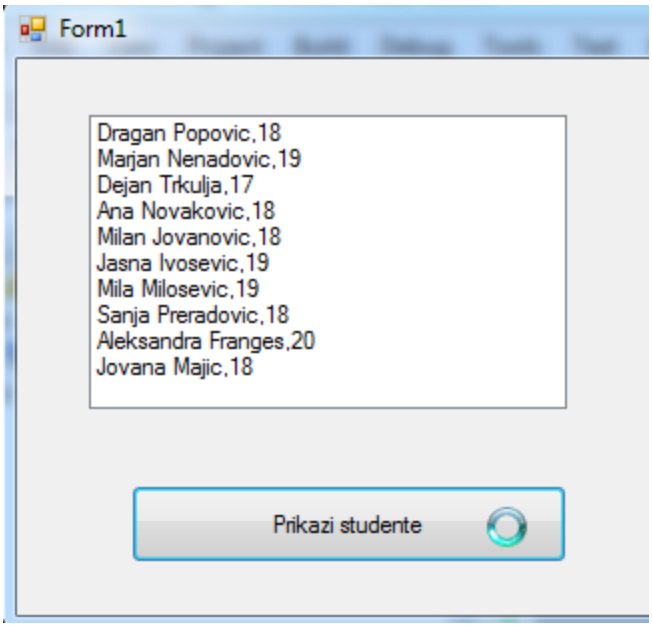
1. Креирање објекта *Connection*;
2. Креирање објекта *Command*;
3. Креирање објекта *DataAdapter*;
4. Креирање објекта *DataSet*;
5. Одредити конекциони стринг за објекат *Connection*;

6. Одредити *Connection* објекат који ће се користити са *Command* објектом;
7. Одредити командни текст *CommandTex*t који ће бити извршен;
8. Додати вредности командним параметрима;
9. Изабрати *Specify SelectCommand* за *DataAdapter*;
10. Попунити *DataSet* коришћењем *DataAdapter*;
11. Приказати резултате.

Креирати нови пројекат, на форму додати дугме и написати следећу процедуру за обраду догађаја *Click*:

```
private void button2_Click(object sender, EventArgs e)
{
SqlConnection connection = new SqlConnection();
SqlCommand command = new SqlCommand();
SqlDataAdapter adapter = new SqlDataAdapter();
DataSet dataset = new DataSet();
connection.ConnectionString = "Integrated Security=true;Initial Catalog=Fakultet;" +
"Data Source=lara-and-nicki";
command.Connection = connection;
command.CommandText = "SELECT Ime, Prezime, Starost FROM Studenti";
adapter.SelectCommand = command;
adapter.Fill(dataset, "Studenti");
foreach (DataRow student in dataset.Tables["Studenti"].Rows)
{
studentiListBox.Items.Add(String.Format("{0} {1},{2}",
student["Ime"], student["Prezime"], student["Starost"]));
}
}
```

Притиском на дугме "*Prikazi studente*", добијају се подаци о студентима:



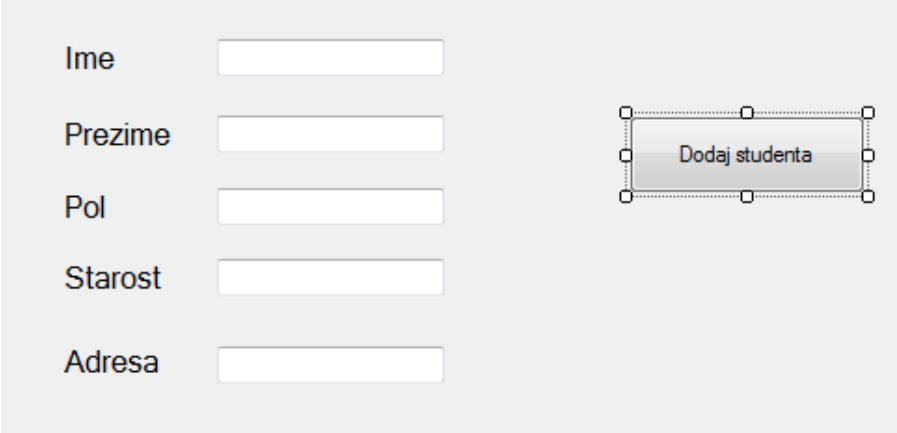
Слика 18.2: Приказ студената

18.3 Уношење података у табелу базе података са отвореном конекцијом и без отварања конекције

Неопходни кораци за уношење података у табелу базе података су:

1. Креирати конекциони објект *Connection*;
2. Креирати командни објект *Command*;
3. Одредити конекциони стринг;
4. Одредити конекцију коју ће командни објект користити;
5. Одредити *INSERT* тврђење за командни текст командног објекта;
6. Додати вредности командним параметрима;
7. Отворити конекцију;
8. Извршити команду;
9. Затворити конекцију.

За демонстрацију уношења података у табелу базе података потребно је отворити нови пројекат и на форму унети одговарајуће контроле, као што је представљено доњом сликом.



Слика 18.3: Изглед форме

Унети следећи код за процедуру за обраду догађаја *Click* за дугме "*Dodaj studenta*" :

```
private void btnDodaj_Click(object sender, EventArgs e)
{
SqlConnection connection = new SqlConnection();
SqlCommand command = new SqlCommand();
connection.ConnectionString = connection.ConnectionString = "Integrated Security=true;Initial Catalog=Fakultet;" +
"Data Source=lara-and-nicki";
command.Connection = connection;
command.CommandText = "INSERT INTO Studenti " +
"(Ime, Prezime, Pol, Starost, Adresa) VALUES " +
"(@Ime, @Prezime, @Pol, @Starost, @Adresa)";
command.Parameters.AddWithValue("@Ime", txtIme.Text);
command.Parameters.AddWithValue("@Prezime", txtPrezime.Text);
command.Parameters.AddWithValue("@Pol", txtPol.Text);
command.Parameters.AddWithValue("@Starost", txtStarost.Text);
command.Parameters.AddWithValue("@Adresa", txtAdresa.Text);
try
{
connection.Open();
int result = command.ExecuteNonQuery();
if (result > 0)
MessageBox.Show("Student je uspesno dodat u tabelu!");
else
MessageBox.Show("Neuspesno dodavanje!");
}
catch (SqlException ex)
{
MessageBox.Show("Pojava greske!");
}
finally
{
connection.Close();
```

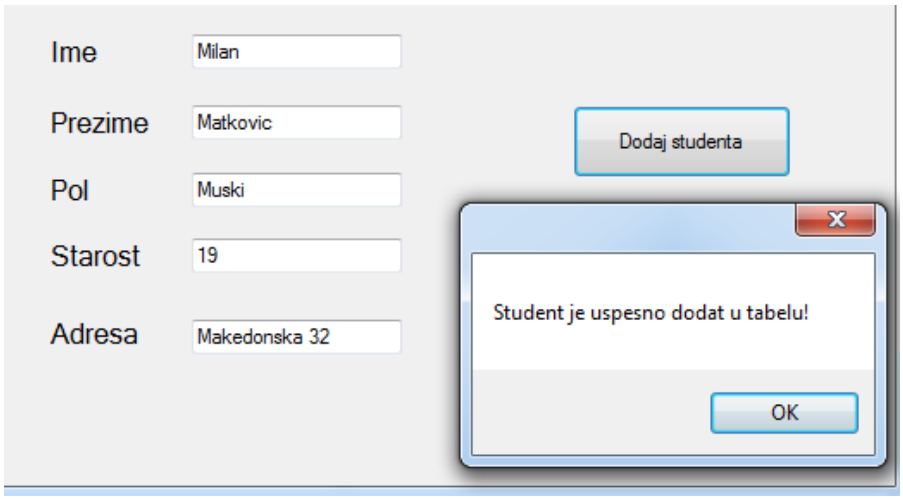
}

}

Декларишемо *Connection* и *Command* објект који су довољни за извршавање задатка уношења података. Исто тако додајемо *INSERT* тврђење у командном тексту (*CommandText*) за дефинисану команду. Команда има неколико параметара који почињу симболом @. У листи параметара није одређен *StudentID* зато што се ради о *identity* пољу при чему ће се аутоматски доделити вредност. Сваком параметру се додељује вредност коришћењем *AddWithValue()* метода *DbCommand Parameters* својства. Овај метод прихвата име командног параметра, као и вредност која ће бити замењена. Параметар *@Ime* замењујемо текстом који је унет у поље под именом *txtIme*. Унутар блока *try*, отвара се конекција и извршава команда коришћењем метода *DbCommand.ExecuteNonQuery()*. Овај метод се користи за извршавање SQL тврђења као што су *INSERT, DELETE, UPDATE,* и *CREATE* који не враћају никакав резултујући скуп података. Овај метод враћа целобројну вредност која симболизира број редова који је захваћен.

Тестира се да ли је вредност различита од 1 и тада се избацује одговарајућа порука. Изузетци се хватају унутар блока *catch* при чему се хватају сви изузетци (*SqlException*) који ће бити избачени унутар *try* блока. Уколико се ради о погрешном конекционом стрингу тада се јавља порука о грешци. Блок *finally* садржи код за затварање конекције.

На слици 18.4 је приказано начин уношења студента у базу података.



Слика 18.4: Уношење студента у базу података

Код за уношење података у табелу базе података без отварања конекције је дат са:

```
private void btnDodaj_Click(object sender, EventArgs e)
{
SqlConnection connection = new SqlConnection();
SqlCommand command = new SqlCommand();
SqlDataAdapter adapter = new SqlDataAdapter();
SqlCommandBuilder builder = new SqlCommandBuilder(adapter);
DataSet dataset = new DataSet();
connection.ConnectionString = "Integrated Security=true;Initial Catalog=Fakultet;" "Data Source=lara-and-nicki";
command.Connection = connection;
command.CommandText = "SELECT * FROM Studenti";
adapter.SelectCommand = command;
adapter.Fill(dataset, "Studenti");
DataRow row = dataset.Tables["Studenti"].NewRow();
row["Ime"] = txtIme.Text;
row["Prezime"] = txtPrezime.Text;
row["Pol"] = txtPol.Text;
row["Starost"] = Int32.Parse(txtPol.Text);
row["Adresa"] = txtAdresa.Text;
dataset.Tables["Studenti"].Rows.Add(row);
```

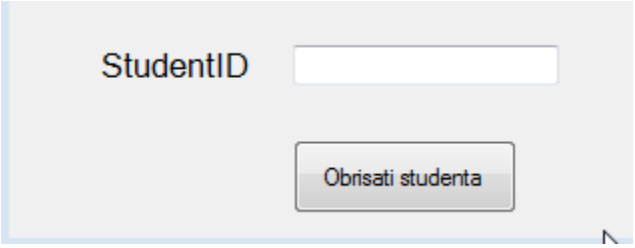
```
try
{
int result = adapter.Update(dataset, "Studenti");
if (result > 0)
MessageBox.Show("Uspesno unesen student!");
else MessageBox.Show("Neuspeh!");
}
catch (SqlException ex)
{
MessageBox.Show(ex.Message);
}
}
```

18.4 Брисање података из табеле базе података са отвореном конекцијом

Брисање података из базе коришћењем отворене конекције, се састоји из следећих корака:

1. Креирати конекциони објекат *Connection*;
2. Креирати командни објекат *Command*;
3. Одредити конекциони стринг;
4. Одредити *DELETE* тврђење у командном тексту *CommandText*;
5. Додати вредности командним параметра;
6. Отворити конекцију;
7. Извршити команду;
8. Затворити конекцију.

На форму додати контроле као на слици:



Слика 18.5: Изглед форме за брисање студента

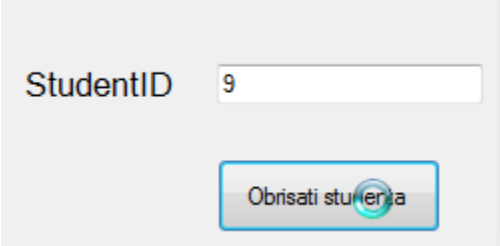
Унети следећи код за процедуру за обраду догађаја *Click* за дугме "*Obrisati studenta*" :

```
private void button4_Click(object sender, EventArgs e)
{
SqlConnection connection = new SqlConnection();
SqlCommand command = new SqlCommand();
connection.ConnectionString = connection.ConnectionString = "Integrated Security=true;Initial Catalog=Fakultet;" +
"Data Source=lara-and-nicki";
command.Connection = connection;
command.CommandText = "DELETE FROM Studenti WHERE StudentID=@StudentID";
command.Parameters.AddWithValue("@StudentID", txtDelete.Text);
try
{
connection.Open();
int result = command.ExecuteNonQuery();
if (result > 0)
MessageBox.Show("Student je obrisan!");
else
MessageBox.Show("Student nije pronadjen");
}
catch (SqlException ex)
{
MessageBox.Show("Desila se greska.");
}
finally
{
```

connection.Close();}

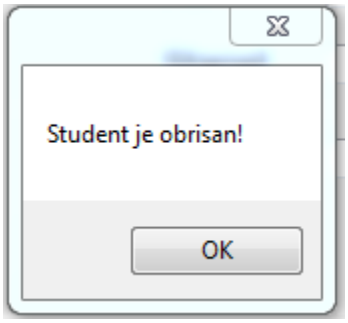
}

Кликнути на дугме "*Obrisati studenta*" и унети редни број студента кога треба обрисати, као што је дато сликом 18.6.



Слика 18.6: Унети редни број студента кога треба обрисати

Након тога се добија обавештење да је студент уклоњен из табеле, као што је представљено доњом сликом.



Слика 18.7: Обавештење да је студент избрисанрисати

18.5 Брисање података из табеле базе података без отварања конекције

Брисање података из базе без отварања конекције, се састоји из следећих корака:

1. Креирати конекциони објекат *Connection*;
2. Креирати командни објекат *Command*;
3. Креирати објекат *DataAdapter*;
4. Креирати објект *CommandBuilder* и повезати га са *DataAdapter*-ом ;
5. Додати вредности командним параметра;
6. Отворити конекцију

18.6 Ажурирање података из табеле базе података са отвореном конекцијом

При ажурирању података користи се метод *DbCommand.ExecuteNonQuery*(), а процес се састоји из следећих корака:

1. Креирати конекциони објекат *Connection*;
2. Креирати командни објекат *Command*;
3. Одредити конекциони стринг;

4. Одредити конекцију коју ће командни објект користити;
5. Одредити *UPDATE* тврђење за командни текст командног објекта;
6. Додати вредности командним параметрима;
7. Отворити конекцију;
8. Извршити команду;
9. Затворити конекцију.

На форму додати контроле као на слици:

Слика 18.8: Изглед форме за ажурирање студента

Унети следећи код за процедуру за обраду догађаја *Click* за дугме "*Update*":

```
private void btnupdate_Click(object sender, EventArgs e)
{
SqlConnection connection = new SqlConnection();
SqlCommand command = new SqlCommand();
connection.ConnectionString = connection.ConnectionString = "Integrated Security=true;Initial Catalog=Fakultet;" +
 "Data Source=lara-and-nicki";  command.Connection = connection;
command.CommandText = "UPDATE Studenti SET Ime=@Ime, " +
 "Prezime=@Prezime, Pol=@Pol, Starost=@Starost, Adresa=@Adresa " +
 "WHERE StudentID=@StudentID";
command.Parameters.Clear();
command.Parameters.AddWithValue("@Ime", txtIme.Text);
command.Parameters.AddWithValue("@Prezime", txtPrezime.Text);
command.Parameters.AddWithValue("@Pol", txtPol.Text);
command.Parameters.AddWithValue("@Starost", txtStarost.Text);
command.Parameters.AddWithValue("@Adresa", txtAdresa.Text);
command.Parameters.AddWithValue("@StudentID", txtStudentID.Text);
```

```
try
{
connection.Open()
int result = command.ExecuteNonQuery();
if (result > 0)
MessageBox.Show("Azuriranje je uspesno uradjeno!");
else
MessageBox.Show("Azuriranje nije uspelo!");
}
catch (SqlException ex)
{
MessageBox.Show("Desila se greska.");
}
finally
{
connection.Close();
}
}
```

UPDATE SQL се користи за ажурирање реда који је одређен јединственим бројем *StudentID* одговарајуће *WHERE* наредбе. Користи се дејство командних параметара на команди. Прво се брише сав садржај *Parameters* својства на команду јер можда садржи параметре претходне операције. Затим се користи метод *DbParameter.AddWithValue()* за додавање вредности одговарајућим командним параметрима у *SQL* командном стрингу. Пошто *SQL UPDATE* команда не враћа никакав запис или резултат, користи се *DbCommand.ExecuteNonQuery()* метод за ажурирање одговарајућег записа. Враћена вредност овог метода представља број редова који је обухваћен.

III. ВАЛИДАЦИЈА ПОДАТАКА

3.1 Увод у валидацију података

Од самог почетка креирања динамичких сајтова неопходно је било увођење валидације. Читав низ грешака које могу настати у апликацији занемаривањем валидације обухвата заобилажење важних поља у која је неопходно унети податке, затим унос података у неодговарајућем формату који се не може користити као што је најчешће случај са e-mail адресом, затим уношење погрешног типа податка, грешке у куцању, изазивање одређене грешке која може проузроковати преузимање података из базе података и друге.

Са ASP.NET технологијом, коначно је добијен стандардни скуп алата за валидацију **web** форме, применом валидационих контрола које су намењене провером улазних података и креирањем корисничких рутина валидације.

ASP.NET омогућава следеће валидационе контроле:

- RequiredFieldValidator
- RangeValidator
- CompareValidator
- RegularExpressionValidator
- CustomValidator
- ValidationSummary.

Једна улазна контрола може бити повезана са сваком валидационом контролом што доводи до вишеструког проверавања улазних података.

Валидација може бити аутоматска при враћању странице на сервер и програмска. Приликом провере валидности података важно је коју вредност има својство командног дугмета **CausesValidation**. У случају када је вредност овог својства **true**, тада је валидација могућа, док у обрнутом случају програмер проверава валидност странице у самој рутини за обраду тог догађаја.

Обично се комбинује серверска и клијентска валидација, као у случају када постоји више контрола на форми у које треба унети одговарајуће податке. Уколико корисник унесе погрешан број у поље за унос података, у које је могуће унети само бројеве у одговарајућем опсегу, а затим без притиска на командно дугме пређе у друго поље притиском на тастер **Tab**, аутоматски ће се генерисати грешка поред контроле која се проверава, јер ASP.NET убацује **JavaScript** функцију која реагује на промену фокуса и без обзира на притисак на командно дугме неће страницу послати серверу.

Валидационе контролне класе наслеђују својства и методе од **BaseValidator** класе. Удоњој табели су дата својства и методе које су везане за све валидационе контроле:

Својства	Опис
ControlToValidate	Дефинише улазну контролу коју валидатор проверава.
Display	Показује како ће грешка бити приказана.

EnableClientScript	Показује да ли ће бити укључена и клијентска валидација.
Enabled	Омогућава или не омогућава валидатор.
ErrorMessage	Дефинише грешку.
Text	Текст који ће се приказати у случају неуспешне валидације.
IsValid	Дефинише да ли је вредност контроле валидна.
SetFocusOnError	Дефинише да ли ће у случају невалидне контроле, фокус бити усмерен на одговарајућу улазну контролу.
ValidationGroup	Логичка група вишеструких валидатора, где ова контрола припада.
Validate()	Овај метод поново врши валидацију контроле и ажурира **IsValid** својство.

Поред аутоматске могућа је и ручна валидација, када програмери узимају у обзир разне факторе, креирају специјализовану поруку грешке, и могу извршити исправност унетих података у сопственом програмском коду када валидационе контроле уопште нису потребне, кликом на командно дугме чије својство **CausesValidation** је постављено на **false** и испитивањем својства **IsValid** и позивом метода **Page.Validate()**, или искључењем својства **EnableScript** за сваку контролу при чему се страница са неисправним подацима шаље серверу након чега програмер сам одлучује о даљим корацима.
Обично се у рутини за обраду догађаја декларише променљива типа *string* у коју се смештају све грешке, као што је дато следећим кодом:

```
string PorukaGreske = "<b>Pronadjene greske</b></br>";

//Prolazak kroz sve validacione kontrole

foreach (BaseValidator grv in this.Validaters)

{

if (!grv.IsValid)

{

PorukaGreske += grv.ErrorMessage + "<br/>";

Textbox Ulaz = (Textbox).this.FindControl(Ulaz.ControlToValidate);

PorukaGreske +="Problem sa vim ulaznim podacima " ;
```

*PorukaGreske +=Ulaz.Text + “
” ;*

}

}

lblGreska.Text = PorukaGreske;

Често се уводи и код за обраду грешака који обезбеђује одговарајућу реакцију на проблеме који се могу појавити. Грешке се пажљиво морају обрадити јер у другом случају страница остаје потуно неразумљива са описом грешке која се јавља приликом уноса података.

3.2 Валидацине контроле

Контрола **RequiredFieldValidator** је веома једноставна, али веома корисна. У следећем примеру од корисника се тражи да унесе име града у контролу **TextBox**, као што је приказано у коду доле:

```
<asp:TextBox runat="server" id="txtImeGrada" />

<asp:RequiredFieldValidator runat="server" id="reqIme" controltovalidate="txtImeGrada" errormessage="Molim Vas unesite ime grada!" />

<br /><br />

<asp:Button runat="server" id="btnPodnesi" text="Ok" />
```

На слици 3.1 је приказано коришћење контроле **RequiredFieldValidator**.

Слика 3.1: Коришћење контроле RequiredFieldValidator

Параметар контроле *RequiredFieldValidator*, *ControlToValidate* треба да буде подешен на ID контроле чији унос је потребно проверити, а посебно прилагођен текст грешке који је написан црвеном бојом је дефинисан у вредности параметра *ErrorMessage.*

Контрола **RangeValidator** се такође користи за осигуравање корисничког улаза који би требало да се нађе у жељеном опсегу, али за разлику од претходне контроле уводе се параметри **MinimumValue** и **MaximumValue**, као што је представљено у коду доле:

```
Unesite broj godina da bi utvrdili da li mozete da igrate pionirskoj konkurenciji:<br />

<asp:TextBox runat="server" id="txtKonkurencija" /><br /><br />

<asp:RangeValidator runat="server" id="rv" controltovalidate="txtKonkurencija"
MinimumValue="10" MaximumValue="14" type="Integer"

errormessage="Morate da unesete broj godina izmedju 10 i 14." /><br />
```

Визуелно бољи ефекат се постиже коришћењем слике која се уноси унутар параметра **Text**, као што је дато са Text = “”.

CompareValidator контрола се користи се користи за упоређивање садржаја **web** контрола са вредношћу као што је минимална количина, а исто тако и за упоређивање садржаја **web** контрола. Провера да ли је корисник пунолетан је дата доњим кодом:

```
Unesite koliko imate godina:<br />

<asp:TextBox runat="server" id="txtGodine" /><br /><br />

<asp:CompareValidator runat="server" id="cmpGodine"
controltovalidate="txtGodine"ValueToCompare="18"

operator="GreaterThanEqual" type="Integer" errormessage="Morate imati najmanje 18 godina da bi mogli da gledate film 50 nijansi sive!" /><br />.
```

Употребом контроле **RegularExpressionValidator** остварује се валидација унетог израза, тако што се врши поређење са задатим шаблоном.

Честе кључне речи у задатим шаблонима су:

\d- одговара јој цифра;

\s- одговара размак;

\w- одговара знак речи;

[abc]- одговара било који од знакова a, b или c;

[^abc]- одговара било који од знакова који нису a, b или c;

W\S\D- одговара знак који није реч, нити је размак, нити је цифра;

(abc)- одговара abc тачно једанпут;

(abc)?- одговара abc једанпут или ниједанпут;

(abc)+- одговара abc произвољан број пута.

Учесталост понављања Опис

*	Нула или више подударања.
+	Једно или више подударања.
?	Нула или једно подударање.
{N}	N подударања.
{N,}	N или више пдударања.
{N,M}	Између N и M подударања.

Синтакса контроле је дата са:

<asp:RegularExpressionValidator ID="string" runat="server" ErrorMessage="string"

ValidationExpression="string" ValidationGroup="string">

</asp:RegularExpressionValidator>

У доњем примеру је представљена валидација унетог поштанског броја. Корисник мора поштански број унети или као петоцифрени број, или у таквој форми, да је дозвољена цртица и додатна четири броја, што је илустровано доле приказаним кодом:

*Unesite postanski broj u odgovarajucoj formi:
*

*<asp:TextBox runat="server" id="txtPosBroj" />

*

<asp:RegularExpressionValidator runat="server" id="rev" controltovalidate="txtPosBroj" ValidationExpression=" \d{5}[-s-]?(\d{4})?"

errormessage="Morate da unesete postanski broj u odgovarajucem formatu." /><br /

Део задате форме у облику *\d{5}[-s-]?(\d{4})?* значи да поштански број може да садржи 5 цифара, или додатне четири цифре, при чему *[-s-]* значи да је допуштена цртица или размак, а затим додатне четири цифре.

Контрола **CustomValidator** омогућава писање рутина како за клијентску тако и за серверску валидацију.

Клијентска валидација се остварује преко својства **ClientValidationFunction**. Рутине за клијентску валидацију су написане у скрипт језицима као што су *JavaScript* или *VBScript*, које претраживач може да разуме.

Рутина за серверску валидацију се позива преко процедуре за обраду догађаја **ServerValidate**. и може бити написана у било ком *.Net* језику, као што је *C#* или *VB.Net*.

Основна синтакса за ову контролу је дата са:

<asp:CustomValidator ID="CustomValidator1" runat="server"

ClientValidationFunction=.cvf_func. ErrorMessage="CustomValidator">

</asp:CustomValidator>.

ValidationSummary контрола не врши никакву валидацију али показује суму свих грешака које се појављују на страници. Ова контрола приказује вредности својства *ErrorMessage* свих валидационих контрола које нису прошле валидацију. Уколико се жели да се грешке прикажу у дијалог прозору тада се користи својство *ShowMessageBox="true"*.

Следећа два својства приказују особине грешке:

- **ShowSummary**: приказује пруку грешке у специфичном формату;
- **ShowMessageBox**: приказује поруку грешке у посебном простору.

<asp:ValidationSummary ID="ValidationSummary1" runat="server"

DisplayMode = "BulletList" ShowSummary = "true" HeaderText="Errors:" />

3.3 Пример примене серверских контрола

Примена серверских контрола је демонстрирана у доле приказаном примеру, што је представљено доњим кодом.

Листинг 3.1

<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="SummaryValidacija.aspx.cs" Inherits="Validacija.SummaryValidacija" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" >

<head runat="server">

<title>Untitled Page</title>

</head>

<body>

<form id="form1" runat="server">

* *

<asp:Label ID="lblmsg"

Text="IZBOR IZBORNIH PREDMETA STUDENATA TRECE GODINE"

runat="server" style="font-weight: 700" />

*

*

Korisnicko ime:

<asp:TextBox ID = "txtUserName" runat ="server"></asp:TextBox>

<asp:RequiredFieldValidator ID ="RuserName" runat ="server" ControlToValidate="txtUserName" ErrorMessage="Morate uneti korisnicko ime"></asp:RequiredFieldValidator>

*

*

Lozinka:

<asp:TextBox ID = "txtPassword" runat ="server"></asp:TextBox>

<asp:RequiredFieldValidator ID ="RPassword" runat ="server" ControlToValidate="txtPassword" ErrorMessage="Morate uneti lozinku"></asp:RequiredFieldValidator>

*

*

Ponovite lozinku:

<asp:TextBox ID = "txtRPassword" runat ="server"></asp:TextBox>

<asp:CompareValidator ID ="RRpassword" runat ="server" ControlToCompare="txtPassword" ControlToValidate="txtRPassword" ErrorMessage="Vasa lozinka se ne podudara"></asp:CompareValidator>

*

*

<asp:DropDownList ID="IzIspit" runat="server">

<asp:ListItem>Izaberite izborni predmet</asp:ListItem>

<asp:ListItem>Fleksibilni proizvodni sistemi</asp:ListItem>

<asp:ListItem>Informacioni sistemi 2</asp:ListItem>

<asp:ListItem>industrijski racunarski sistemi</asp:ListItem>

<asp:ListItem>kompjutersko projektovanje 2</asp:ListItem>

</asp:DropDownList>

<asp:RequiredFieldValidator ID="RIzIspit"

runat="server" ControlToValidate ="IzIspit"

ErrorMessage="Niste izabrali izborni predmet"

InitialValue="Izaberite izborni predmet">

```
</asp:RequiredFieldValidator>
<br /><br />
Unesite broj godina:
<asp:TextBox ID="txtGodine" runat="server"></asp:TextBox>
<asp:RangeValidator ID="Rvgodine" runat="server" ControlToValidate="txtGodine"
ErrorMessage="Unesite broj godina (21 - 23)" MaximumValue="23" MinimumValue="21"
Type="Integer">
</asp:RangeValidator>
 <br /><br />
Unesite e-mail adresu:
<asp:TextBox ID="txtEmail" runat="server"></asp:TextBox>
<asp:RegularExpressionValidator ID="remail" runat="server"
ControlToValidate="txtEmail" ErrorMessage="Unesite Vas e-mail u pravilnoj formi"
ValidationExpression="\w+([-+.']\w+)*@\w+([-.]\w+)*\.\w+([-.]\w+)*">
</asp:RegularExpressionValidator>
 <br /><br />
 <asp:ValidationSummary ID="ValidationSummary1" runat="server"
DisplayMode ="BulletList" ShowMessageBox="true" />
<br /><br />
<asp:Button ID="btnsubmit" runat="server" Text="Submit" onclick="btnsubmit_Click" />
<br /><br />
<asp:Label ID="lblPoruka" runat="server" ></asp:Label>
</div>
</form>
</body>
</html>
protected void btnsubmit_Click(object sender, EventArgs e)
{
 if (Page.IsValid)
{
lblPoruka.Text = "Vasa forma je validna";
}
else
```

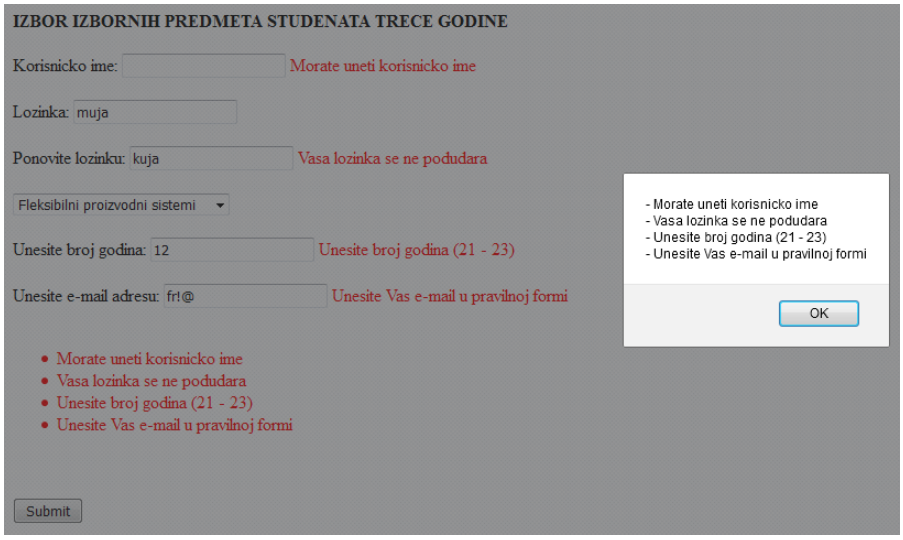
{

lblPoruka.Text = "Popunite sva neophodna polja";

}

}

На слици 3.1 је демонстриран приказ свих грешки које се јављају у програму.



Слика 3.2: Приказ свих грешки у дијалог прозору

У случају сложенијих страница користе се засебни панели на које се постављају одвојене групе контрола, и тада се валидација може одвијати засебно.

Уколико на форму поставимо командно дугме које није везано ни за један панел, тада се кликом на дугме врши валидација свих контрола које се налазе на форми.

ТЕСТ 3

1. Навести све валидацине серверске кнтроле и описати њихову функцију?
2. Описати серверску валидацију и образложити који услов треба да буде задовољен?
3. Како се извршава клијентска валидација?
4. Који услови су неопходни да би се извршила ручна валидација?
5. Дати сопствени пример за контролу *RangeValidator*.
6. Чему служи контрола ValidationSummary на који начин се грешке могу приказати у дијалог прозору?
7. Која контрола служи за упоређивање садржаја две контроле при чему ствара поруку о грешци у случају када нема поклапања садржаја?
8. Приказати сопствени пример при чему је потребно користити све серверске валидационе контроле.

IV. СЛОЖЕНЕ КОНТРОЛЕ

4.1 Calendar контрола

Контрола **Calendar** служи за избор датума, обично се на **web** форму уносе и додатна командна дугмад, листе за избор и друге контроле како би се кориснику омогућила жељена промена.

Важна својства ове контроле су представљене у наредној табели.

Својства	**Опис**
Caption	Даје или поставља наслов за **caption** контролу.
CaptionAlign	Врши поравнавање наслова.
CellPadding	Представља простор у пикселима између ивице њене ћелије и њеног садржаја.
CellSpacing	Успоставља размак између ћелија.
DayHeaderStyle	Даје особине стила за секцију која приказује дане у недељи.
DayNameFormat	Поставља формат за дане у недељи.
DayStyle	Поставља особине стила за дане које су представљени у приказаном месецу.
FirstDayOfWeek	Поставља дан у недељи који ће бити приказан у првој колони.
NextMonthText	Поставља текст за навигациону контролу за следећи месец. Текућа вредност је >.
NextPrevFormat	Поставља формат за навигациону контролу за следећи или претходни месец.
OtherMonthDayStyle	Даје особине стила за дане на Calendar контроли који се не налазе у приказаном месецу.
PrevMonthText	Поставља текст за навигациону контролу за преходни месец од приказаног. Текућа вредност је <.
SelectedDate	Даје или поставља изабрани датум.
SelectedDates	Даје колекцију **DateTime** објеката који представљају изабране датуме.

SelectedDayStyle	Даје својства стила за изабране датуме.
SelectionMode	Успоставља изборни мод који одређује да ли ће корисник изабрати појединачни дан, недељу или читав месец.
SelectMonthText	Подешава текст за изабрани елемент месеца у изабраној колони.
SelectorStyle	Подешава својства стила за изабрану колону недеље или месеца.
SelectWeekText	Подешава текст који ће бити приказан за елемент који означава избор недеље у изабраној колони.
ShowDayHeader	Подешава вредност која показује да ли приказано заглавље за дане у недељи.
ShowGridLines	Подешава вредност која показује да ли ће бити приказана мрежа.
ShowNextPrevMonth	Подешава вредност која показује да ли ће елементи за навигацију претходног или следећег месеца бити приказани у избору наслова.
ShowTitle	Подешава вредност која показује да ли ће наслов бити приказан.
TitleFormat	Подешава формат за избор наслова.
Titlestyle	Подешава својства стила текста у заглављу.
TodayDayStyle	Даје својства стила за данашњи датум на Calendar контроли.
TodaysDate	Даје или подешава вредност за данашњи датум.
UseAccessibleHeader	Подешава вредност која показује да ли треба рендеровати табелу заглавља <th> HTML елемент за заглавље дана уместо табеле података <td> HTML елемената.
VisibleDate	Подешава датум који одређује месец који се приказује.
WeekendDayStyle	Подешава стил за викенд датуме на контроли Calendar.

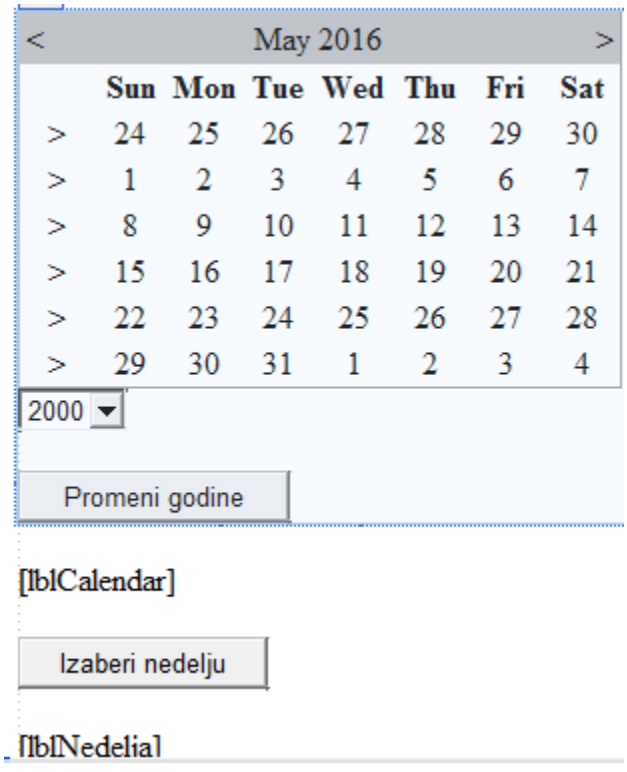
У доњем примеру потребно је на **web** форму поставити контролу **Calendar**, две **label** и једну **DropDown** контролу, као што је представљено доњом сликом.

Код **Default.aspx** стране је дат доњим листингом.

Листинг 4.1

```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="Default.aspx.cs"
Inherits="SadrzaneKontrole._Default" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" >

<head runat="server">

<title>Untitled Page</title>

</head>

<body>

 <form id="form1" runat="server">

<div>

<asp:Calendar ID="MyCalendar" runat="server" ondayrender="MyCalendar_DayRender"
SelectionMode="DayWeek"  ShowDayHeader="True
onselectionchanged="MyCalendar_SelectionChanged"></asp:Calendar>

<asp:DropDownList ID="Godina" runat="server">

<asp:ListItem>2000</asp:ListItem>

 <asp:ListItem>2001</asp:ListItem>

<asp:ListItem>2002</asp:ListItem>

<asp:ListItem>2003</asp:ListItem>

<asp:ListItem>2004</asp:ListItem>

<asp:ListItem>2005</asp:ListItem>

<asp:ListItem>2006</asp:ListItem>

<asp:ListItem>2007</asp:ListItem>

 <asp:ListItem>2008</asp:ListItem>

<asp:ListItem>2009</asp:ListItem>

<asp:ListItem>2010</asp:ListItem>

<asp:ListItem>2011</asp:ListItem>

<asp:ListItem>2012</asp:ListItem>

<asp:ListItem>2013</asp:ListItem>

<asp:ListItem>2014</asp:ListItem>

<asp:ListItem>2015</asp:ListItem>

 <asp:ListItem>2016</asp:ListItem>

</asp:DropDownList>

 <br />
```

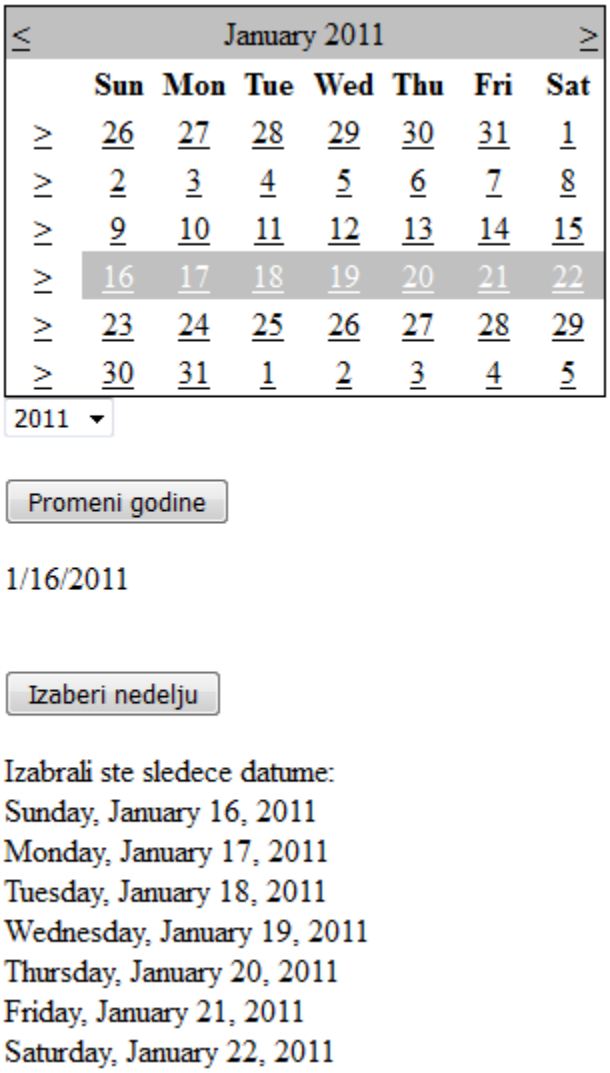
```
<br />
<asp:Button ID="OdabirGodine" Text="Promeni godinu" runat="server"
 onclick="OdabirGodine_Click" />
</div>
 <p>
<asp:Label ID="lblCalendar" runat ="server"></asp:Label>
 </p>
<asp:Button ID = "IzabratiNedelju" Text = "Izaberi nedelju" runat ="server"
onclick="IzabratiNedelju_Click" />
<p>
<asp:Label ID ="lblNedelja" runat ="server"></asp:Label>
</p>
 </form>
</body>
</html>
```



Слика 4.1: Приказ дизајн погледа

Корисник прво из понуђене листе бира одговарајућу годину, а затим кликом на дугне **“Promeni godinu”**, мења у календару годину. Избором неког датума на измењеној години добија се порука о селектованом датуму.

Затим треба изабрати **Calendar** контролу, и подесити својство **SelectionMode** на **DayWeek**. Изабрати неку недељу у календару, а затим кликнути на дугме **“Izaberi nedelju”**, да би се приказали изабрани датуми, као што је дато на следећој слици.



Слика 4.2: Избор датума

Листинг 4.2

```
protected void MyCalendar_DayRender(object sender, DayRenderEventArgs e)
{
//Pronalazi 9 Septembar u bilo kojoj godini i formatira ga
if (e.Day.Date.Day == 9 && e.Day.Date.Month == 9)
{
e.Cell.BackColor = System.Drawing.Color.Yellow;
Label lbl = new Label();
```

```
lbl.Text = "<br />Moj rodjendan! ";
e.Cell.Controls.Add(lbl);
}
}
protected void MyCalendar_SelectionChanged(object sender, EventArgs e)
{
DateTime dtOdabrano = MyCalendar.SelectedDate;
lblCalendar.Text = MyCalendar.SelectedDate.ToShortDateString();
}
protected void OdabirGodine_Click(object sender, EventArgs e)
{
String strGodina = Godina.SelectedItem.Text;
int nGodina = Int32.Parse(strGodina);
MyCalendar.VisibleDate = new DateTime(nGodina, MyCalendar.VisibleDate.Month,
MyCalendar.VisibleDate.Day);
}
protected void IzabratiNedelju_Click(object sender, EventArgs e)
{
lblNedelja.Text = "Izabrali ste sledece datume: <br />";
foreach (DateTime dt in MyCalendar.SelectedDates)
{
lblNedelja.Text += dt.ToLongDateString() + "<br />";
}
}
```

Три најважнија догађаја ове контроле су представљена у доњој табели.

Догађај	**Опис**
SelectionChanged	Овај догађај се ствара када се означи дан, недеља или читав месец.
DayRender	Догађај се подиже када се свака ћелија са подацима ове контроле рендерује.

VisibleMonthChanged Подиже се када корисник мења месец.

Догађај **DayRender** омогућава помоћу својства **e.Cell** избор датума и подешавање ћелије у којој је датум селектован.

У листингу 4.2 приказан је догађај **DayRender** и селекција датума у свакој изабраној години, што је приказано следећим кодом:

```
//Pronalazi 9 Septembar u bilo kojoj godini i formatira ga
if (e.Day.Date.Day == 9 && e.Day.Date.Month == 9)
{
e.Cell.BackColor = System.Drawing.Color.Yellow;
Label lbl = new Label();
lbl.Text = "<br />Moj rodjendan! ";
e.Cell.Controls.Add(lbl);
}
```

Исто тако коришћењем овог догађаја је могуће ограничити избор појединих датума као што је приказано кодом:

```
protected void MyCalendar_DayRender(object sender, DayRenderEventArgs e)
{
//Ogranicenje u slucaju kada mesec nije tekuci i godina manja od 2016
if (e.Day.IsOtherMonth && e.Day.Date.Year< 2016)
{
e.Day.IsSelectable = false;
}
}
```

4.2 Multiview контрола

Ова контрла омогућава креирање више различитих приказа на **web** страници при чему је само један поглед активан.

Синтакса **MultiView** контроле је:

```
<asp:MultView ID= "MultiView1" runat= "server">
</asp:MultiView>
```

Синтакса **View** контроле је:

```
<asp:View ID= "View1" runat= "server">
</asp:View>
```

Уколико се **View** контрола самостално користи изазваће грешку. Она се увек користи са **Multiview** контролом на следећи начин:

```
<asp:MultView ID= "MultiView1" runat= "server">
  <asp:View ID= "View1" runat= "server"> </asp:View>
</asp:MultiView>
```

Контрола **Multiview** има следећа важна својства:

Својства	**Опис**
Views	Колекција View контрола унутар **MultiView** контроле.
ActiveViewIndex	Нулти индекс означава активни поглед. Ако ниједан поглед није активан онда је вредност индекса -1.

Важне методе ове контроле су:

Методе	**Опис**
SetActiveview	Подешава активни поглед
GetActiveview	Приказује активни поглед.

Битни догађаји су:

Догађај	**Опис**
ActiveViewChanged	Активира се при промени погледа
Activate	Активира се код активног погледа
Deactivate	Активира се код неактивног погледа.

У следећем примеру је приказана примена **Multiview** контроле, најпре у дизајн погледу, а затим је приказан и одговарајући код.

Слика 4.3: Дизајн поглед

<asp:DropDownList ID="DropDownList1"
OnSelectedIndexChanged="DropDownList1_SelectedIndexChae" runat="server"
AutoPostBack="True">

<asp:ListItem Value="0">Pogled 1</asp:ListItem>

```
<asp:ListItem Value="1">Pogled 2</asp:ListItem>
<asp:ListItem Value="2">Pogled 3</asp:ListItem>
</asp:DropDownList>
<hr />
<br />
<hr />
<asp:MultiView ID="MultiView1" runat="server" ActiveViewIndex="0">
<asp:View ID="View1" runat="server">
Sada se prikazuje pogled #1<br />
<asp:TextBox ID="TextBox1"runat="server">
</asp:TextBox><strong> </strong>
<br />
<asp:CheckBox ID="CheckBox" Text="Polje za potvrdu" runat ="server" />
<br />
<asp:Button ID="Button1" runat="server" Text="Button" /></asp:View>
<asp:View ID="View2" runat="server">
Sada se prikazuje pogled #2<br />
<asp:HyperLink ID="HyperLink1" runat="server" NavigateUrl="http://www.google.com">Veza za
Google web site</asp:HyperLink>
<br />
<asp:Button ID="Button2" runat="server" Text="Button" />
<br />
<asp:HyperLink ID="HyperLink2" runat="server" NavigateUrl="http://www.microsoft.com">Veza za
Microsoft web site</asp:HyperLink>
</asp:View>
<asp:View ID="View3" runat="server">Sada se prikazuje pogled #3<br />
<asp:Calendar ID="Calendar1"runat="server">
</asp:Calendar>
</asp:View>
</asp:MultiView>
protected void DropDownList1_SelectedIndexChanged(object sender, EventArgs e)
{
MultiView1.ActiveViewIndex = DropDownList1.SelectedIndex;
```

}

Својство *MultiView.ActiveViewIndex* је текуће подешено на -1, а уколико је подешено на 0, биће активан први приказ.

ТЕСТ 4

1. Која је улога **Calendar** контроле?

2. Приказати важна својства ове контроле.

3. Представити догађаје **Calendar** контроле.

4. На који начин се ограничава избор одговарајућих датума, дати пример?

5. Дати сопствени пример за контролу **MultiView**.

6. Навести својства, методе и догађаје ове контроле.

V. НАПРЕДНЕ ASP.NET КОНТРОЛЕ

5.1 Panel контрола

Panel контрола служи као контејнер за остале контроле на страни. Она контролише појављивање и видљивост контрола које садржи, и омогућава стварање контрола програмски.

Основна синтакса ове контроле је:

<asp:Panel ID= "Panel1" runat = "server">

</asp:Panel>

Panel контрола је изведена из **WebControl** класе. Ова контрола поред великог броја наслеђених својстава има и нека сопствена као што је дато у доњој табели:

Својства	**Опис**
BackImageUrl	URL слике позадине панела.
DefaultButton	Подешава идентификатор текућег дугмета које се налази унутар Panel контроле.
Direction	Смер текста на панелу.
GroupingText	Обезбеђује груписање текста.
HorizontalAlign	Хоризонтално поравнавање садржаја панела.
ScrollBars	Обезбеђује видивост клизача на панелу.
Wrap	Обезбеђује прекид текста.

У доњем коду је приказана примена **Panel** контроле.

<asp:Panel ID="KonPanel" runat="server" BorderColor="#990000" BorderStyle="Solid" Borderstyle="width:1px" Height="150px" ScrollBars="Auto" style="width:60%" BackColor="#CCCCFF" Font-Names="Courier" HorizontalAlign="Center">

*
*

*
*

</asp:Panel>

<asp:Button ID="Dugme" Text="Pritisni me" OnClick="DugmePritisnuto" runat="server" />

*
*

<asp:DropDownList ID="PoljePadajuceListe" OnSelectedIndexChanged="MojaPadajucaIzabrana" runat="server" AutoPostBack="true">

<asp:ListItem>Stavka 1</asp:ListItem>

<asp:ListItem>Stavka 2</asp:ListItem>

<asp:ListItem>Stavka 3</asp:ListItem>

</asp:DropDownList>

*
*

<asp:CheckBoxList ID="PoljePotvrda" runat="server" OnSelectedIndexChanged="MojaListaSaPotvrdama" AutoPostBack="true">

<asp:ListItem>Potvrda 1</asp:ListItem>

<asp:ListItem>Potvrda 2</asp:ListItem>

<asp:ListItem>Potvrda 3</asp:ListItem>

</asp:CheckBoxList>

*
*

<asp:Label ID="lblizlaz" runat="server"></asp:Label>

5.2 Ajax контрола

AJAX контрола је изведена од **Asynchronous JavaScript and XML**. Ова платформа убрзава време одзива, на тај начин што AJAX сервер контрола додаје скрипт на страну који се затим извршава и процесира од стране претраживача.

However like other ASP.NET server controls, these AJAX server controls also can have methods and event handlers associated with them, which are processed on the server side.

Контролни алат у **Visual Studio IDE** садржи групу контрола која се назива '**AJAX Extensions**'.

Слика 5.1: Група AJAX контрола

Контрола **ScriptManager** је најважнија контрола и мора бити присутна на страници да би остале контроле могле да раде.

Основна синтакса ове контроле је:

<asp:ScriptManager ID="ScriptManager1" runat="server">

</asp:ScriptManager>

Уколико корисник укључи *'Ajax Enabled site'* или дода *'AJAX Web Form'* са **'Add Item'** дијалог прозора, web форма аутоматски садржи контролу **Script Manager**. Контрола **ScriptManager** води рачуна о скрипту са клијентске стране за све контроле са серверске стране.

Контрола **UpdatePanel** представља контролу која је изведена из класе **Control**. Она делује као контејнер за децу контроле које се налазе унутар ње, и нема свој сопствени интерфејс.

Када контрола унутар ње покрене **post back**, контрола **UpdatePanel** интервенише да покрене post асинхроно и ажурира само део стране.

Уколико је дугме контрола унутар **update** панела и ако се кликне на њу, биће захваћене контроле само унутар панела **update**, док контроле на другој страни панела неће бити захваћене. Ово се зове парцијални *post back* или асинхрони *post back*.

Код је приказан у доњем листингу.

Листинг 5.1

AjaxControl.aspx

```
<form id="form1" runat="server">
 <div>
<asp:ScriptManager ID="ScriptManager1" runat="server" />
</div>

<asp:UpdatePanel ID="UpdatePanel1" runat="server">
<ContentTemplate>
 <asp:Button ID="btnpartial" runat="server" onclick="btnpartial_Click" Text="Parcijalni PostBack"/>
<br />
<br />
<asp:Label ID="lblpartial" runat="server"></asp:Label>
</ContentTemplate>
</asp:UpdatePanel>
<p> </p>
<p>Izvan Update Panela</p>
<p>
<asp:Button ID="btntotal" runat="server" onclick="btntotal_Click" Text="Total PostBack" />
</p>
<asp:Label ID="lbltotal" runat="server"></asp:Label>
</form>
```

AjaxControl.aspx.cs

protected void btnpartial_Click(object sender, EventArgs e)

{

string time = DateTime.Now.ToLongTimeString();

lblpartial.Text = "Showing time from panel" + time;

lbltotal.Text = "Showing time from outside" + time;

}

protected void btntotal_Click(object sender, EventArgs e)

{

string time = DateTime.Now.ToLongTimeString();

lblpartial.Text = "Showing time from panel" + time;

lbltotal.Text = "Showing time from outside" + time;}

Када се кликне на дугме **post back**, ажурира се време у обе **label** контроле, али ако се кликне на дугме за парцијални **post back**, само се ажурира лабела унутар **update** панела.

5.3 AdRotator контрола

Контрола **AdRotator** произвољно бира банер из листе која је дата у спољној *XML* датотеци, која се зове рекламна датотека.

Основна синтакса контроле **AdRotator** је:

<asp:AdRotator runat = "server" AdvertisementFile = "adfile.xml" Target = "_blank"

при чему одредиште *_blank* значи да ће линк бити отворен у новом прозору без оквира, *_parent* да ће линк бити отворен у родитељском прозору текућег оквира, *_self* да ће бити отворен у текућем оквиру, а *_top* да ће линк бити отворен у горњем оквиру текућег прозора и да ће заузети читаву страну.

У следећем примеру на форму је додата контрола **AdRotator**, креирана је спољна *XML* датотека, тако да се са сваким освежавањем странице приказује друга слика, као и хиперлинк који води до одговарајуће **web** странице.

Листинг 5.2

AdRotator.aspx

<asp:AdRotator ID="AdRotator1" runat="server" AdvertisementFile ="~/ads.xml" target ="_blank" onadcreated="AdRotator1_AdCreated" />

* *

<asp:HyperLink ID="lnlBanner"runat="server"></asp:HyperLink>

ads.xml

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

<Advertisements>

<Ad>

<ImageUrl>TeniskaOprema.jpg</ImageUrl>
<NavigateUrl>http://www.colouredsand.com.au</NavigateUr>

<AlternateText>

Teniska oprema

</AlternateText>

<Impressions>20</Impressions>

<Keyword>Teniska oprema</Keyword>

</Ad>

<Ad>

<ImageUrl>TvrdiTereni.jpg</ImageUrl>

<NavigateUrl>http://www.vidanta.com/experience-vidanta/tennis</NavigateUrl>

<AlternateText>Tvrdi tereni</AlternateText>

<Impressions>20</Impressions>

<Keyword>Tvrdi tereni</Keyword></Ad>

<Ad>

<ImageUrl>TeniskeLopte.jpg</ImageUrl>

<NavigateUrl>http://www.rockcreektennis.com</NavigateUr><AlternateText>Teniske lopte</AlternateText>

<Impressions>20</Impressions>

<Keyword>Teniske lopte</Keyword>

</Ad>

</Advertisements>

AdRotator.aspx.cs

```
protected void AdRotator1_AdCreated(object sender, AdCreatedEventArgs e)
{
//Sinhronizacija kontrole
lnlBanner.NavigateUrl = e.NavigateUrl;
lnlBanner.Text = "Kliknite na ovaj link za više informacija: ";
lnlBanner.Text += e.AlternateText;
}
```

У својство *AdvertisementFile* смешта се одговарајућа датотека са огласима, као и врста прозора који ће се отворити. Одредиште (**Target**), може бити оквир али могу бити и специјална одредишта као што је *_blank* када ће се линк отворити у специјалном прозору без оквира, *_self* када се линк

отвара у текућем оквиру, *_parent* када се линк отвара у родитељском прозору без текућег оквира, *_top* при чему се линк отвара у горњем оквиру текућег прозора.

Подешавањем својства *KeywordFilter* бирају се банери који су дефинисани кључном речи.

Својства појединачних елемената <*Ad*> подешавају учестаност приказа, подешавају слике и линкове, а најчешће коришћена су:

NavigateUrl- представља линк на који корисник прелази када кликне на њега;

AlternateText- ако слика не може да се прикаже тада се приказује текст, али је та особина углавном везана за старије претраживаче, док се у новијим претраживачима користи као текст који објашњава појам такозвани *ToolTip*;

Keyword- користи се за филтрирање и као идентификатор групе огласа, и у зависности од ње се укључује одговарајућа група огласа;

Impressions- одређује колико пута ће оглас бити приказан, виши број дефинише и већу учестаност приказивања огласа;

ImageUrl- одређује која ће слика бити приказана при чему се може користити релативни линк или апсолутна интернет адреса.

Својства класе *AdRotator* су дата у табели 5.1.

Својство	**Опис**
AdvertisementFile	Путања за датотеку **advertisement**.
AlternateTextFeild	Име поља где је обезбеђен алтернативни текст. Текућа вредност је **AlternateText**.
DataMember	Име специфичне листе података за коју се врши везивање када се не користи датотека **advertisement**.
DataSource	Одређује одакле ће бити преузети подаци.
DataSourceID	Id контроле одакле се преузимају подаци.
Font	Одређује својства фонта одговарајуће контроле за рекламу.
ImageUrlField	Име поља где се обезбеђује URL за слику. Текућа вредност је **ImageUrl**.
KeywordFilter	Служи за приказ кључних речи.
NavigateUrlField	Име поља где је URL за навигацију обезбеђен. Текућа вредност је **NavigateUrl**.

Target	Прозор претраживача која приказује садржај странице која је повезана.
UniqueID	Обезбеђује јединствени, хијерархијски уређен идентификатор за AdRotator контролу.

Табела 5.1

У табели 5.2 дати су важни догађаји AdRotator класе:

Догађаји	**Опис**
AdCreated	Подиже се једном приликом креирања кнтроле али пре него што се рендерује страница.
DataBinding	Извршава се када се серверска контроле везује за извор података.
DataBound	Извршава се након што се серверска контрола веже за извор података.
Disposed	Извршава се када серверска контрола напусти меморију, што представља и њену последњу фазу у животном циклусу када се тражи ASP.NET страна.
Init	Извршава се када се иницијализује серверска контрола што представља први корак у животном циклусу.
Load	Извршава се када се серверска контрола учита у објект Page.
PreRender	Извршава се када се објект **Control** учита али пре рендеровања.
Unload	Извршава се када се серверска контрола уклони из меморије.

Табела 5.2

5.4 Различити начини приказивања страница

Најчешћи случај код већине *web site*- ова представља пребацивање са једне на другу страницу, што посебно долази у обзир приликом креирања *web site*- ова електронске трговине када се уноси страна са подацима о купцу, затим страница са потрошачком корпом, страница са завршним плаћањем и слично.

Некада се користи и решење странице која је мултифункционална и која остварује више задатака. У овом случају је пожељно креирати динамичку страницу која обезбеђује више начина приказа и тада је најбоље користити контроле типа *Panel*.

Приликом извршавања апликације ако је својство *Visible* неке контроле постављено на *false*, та контрола неће бити видљива али ће бити и даље видљива у Visual Studi-у у режиму дизајна.

Уређење панела може бити подешено на следећи начин:

<asp: Panel Id= "panel1" runat = "server">... </asp:Panel>

<asp: Panel Id= "panel2" Visible = "False" runat = "server">... </asp:Panel>

<asp: Panel Id= "panel3" Visible = "False" runat = "server">... </asp:Panel>

<asp: Panel Id= "panel4" Visible = "False" runat = "server">... </asp:Panel>

<asp: Panel Id= "panel5" Visible = "False" runat = "server">... </asp:Panel>

<asp: Panel Id= "panel6" Visible = "False" runat = "server">... </asp:Panel>

Када желимо да укључимо појединo навигационо дугме, као и да активирамо или деактивирамо поједине панеле, може се користити следећи код:

```
if (Panel1.Visible)
{
//Korak 2
Panel1.Visible = false;
Panel2.Visible = true;
Panel3.Visible = false;
Panel4.Visible = false;
}
{
//Korak 3
else if (Panel2.Visible)
Panel1.Visible = false;
Panel2.Visible = false;
Panel3.Visible = true;
Panel4.Visible = false;
// Moguce koriscenje  natpisa ili funkcije dugmeta
}
{
//Korak 4
else if (Panel3.Visible)
Panel1.Visible = false;
```

Panel2.Visible = false;

Panel3.Visible = false;

Panel4.Visible = true;

// Moguce koriscenje natpisa ili funkcije dugmeta

}

5.5 Wizard контрола

Ова контрола представља напреднију и усавршенију верзију контроле **Multiview**, која осим ппедстављања једног приказа из групе садржи и разне друге могућности као што су употреба навигационих дугмади, текст са линковима ка одговарајућим линковима у друге.

Коришћење чаробњака је везано уобичајено за одговарајући задатак, тако што се прелази са једног корака на други или се враћа на претходни корак, али постоји могућност игнорисања појединих корака, што је повезано са подацима које корисник уноси. Подешавањем својства **WizardDisplaySideBar**, односно давањем вредности *false* овом својству може се искључити приказ појединих линкова, али исто тако се може контрилосати и обављање навигације корак по корак.

Појединачни кораци се краирају коришћењем специјалних тагова, као што је дато следећим кодом:

<asp:Wizard id= Wizard1" runat="server" ...>

<WizardSteps>

<asp: WizardStep runat="server" Title="Step 1">

<asp: WizardStep runat="server" Title="Step 2">

<asp: WizardStep>

<WizardSteps>

<asp: Wizard>

Најважнија својства контроле *WizardStep* су:

Title- представља опис поједоних корака, и овај назив се користи у одвојеном тексту за линк;

AllowReturn- обезбеђује да се крисник може вратити на претходни корак ако је вредност **true**;

StepType- одређује која ће врста навигационих дугмади бити приказана, подразувана вредност је **Auto**, тако да ће први корак бити **Start**, а последњи **Finish**, док сви остали су **Step**.

ТЕСТ 5

1. Који је задатак **Panel** контроле?

2. Приказати синтаксу за **Ajax** контролу.

3. Представити примером коришћење **AdRotator** контроле.

4. Како се остварује различити начин приказивања странице?

5. Навести и објаснити основна својства и догађаје **AdRotator** контроле .

6. Чему служи **Wizard** контрола и приказати синтаксу за креирање појединих корака.