4.2 Особливості конструювання програмних модулів

4.2.1 Особливості роботи з інтегрованим середовищем розробки

Середовище розробки – Visual Studio.

Microsoft Visual Studio — серія продуктів фірми Майкрософт, які включають інтегроване середовище розробки програмного забезпечення та низку інших інструментальних засобів. Ці продукти дозволяють розробляти як консольні програми, так і програми з графічним інтерфейсом, в тому числі з підтримкою технології Windows Forms, а також веб-сайти, веб-застосунки, веб-служби як в рідному, так і в керованому кодах для всіх платформ, що підтримуються Microsoft Windows, Windows Mobile, Windows Phone, Windows CE, .NET Framework, .NET Compact Framework та Microsoft Silverlight.

1. Visual Studio включає один або декілька з наступних компонентів:
2. Visual Basic .NET, а до його появи — Visual Basic
3. Visual C++
4. Visual C#
5. Visual F# (входить до складу Visual Studio 2010);
6. Visual Studio Debugger

4.2.2 Особливості створення програмної структури з урахуванням

спеціалізованого Фреймворку.

Головний фреймворк розробки є ASP.NET Core.

Платформа ASP.NET Core представляє технологію від компанії Microsoft, призначену для створення різного роду веб-додатків: від невеликих веб-сайтів в великих веб-порталів і веб-сервісів.

ASP.NET Core може працювати поверх крос-платформної середовища .NET Core, котра в свою чергу може бути розгорнута на основних популярних операційних системах: Windows, Mac OS, Linux. І таким чином, за допомогою ASP.NET Core є можливість створювати крос-платформні додатки. І хоча Windows як середовище для розробки і розгортання програми досі превалює, але тепер вже ми не обмежені тільки цією операційною системою. Програми можливо запускати веб-додатки не тільки на ОС Windows, але і на Linux і Mac OS. А для розгортання веб-додатки можна використовувати традиційний IIS, або крос-платформний веб-сервер Kestrel.

4.2.3 Особливості створення програмних класів

Окрім базових класів сутностей, створюється програмний клас ApplicationContext, що унаслідується від DbContext, клас із пакету для роботи із базою даних PostgreSQL, який визначає контекст даних, який використовується для взаємодії з базою даних. Задаються поля, що використовують DbSet, що представляє набір сутностей, які зберігаються в базі даних.

public class User

{

public string login { get; set; }

public string password { get; set; }

public string e\_mail { get; set; }

}

public class Administrator : User

{

public int amount\_of\_commits { get; set; }

}

public class Event

{

public string title { get; set; }

public string beginning\_time { get; set; }

public bool state { get; set; }

}

public class Place

{

public string name { get; set; }

public string description { get; set; }

public int placement { get; set; }

}

public class Information

{

public string description { get; set; }

public int assesment { get; set; }

public string[] preferences = new string[10] { get; set; }

}

namespace PostgresApp

{

public class ApplicationContext : DbContext

{

public DbSet<User> users { get; set; }

public DbSet<Event> events { get; set; }

public DbSet<Information> hobbies { get; set; }

public ApplicationContext()

{

Database.EnsureCreated();

}

protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)

{

optionsBuilder.UseNpgsql("Host=localhost;Port=5123;Database=UPigeon;Username=postgres;Password=password");

}

}

}

namespace FilterApp.Models

{

public class UserListViewModel

{

public IEnumerable<User> users { get; set; }

public SelectList hobbies { get; set; }

public string name { get; set; }

public string description { get; set; }

}

}

4.2.4 Особливості розробки алгоритмів методів програмних класів або

процедур/функцій

Особливістю методів розробки алгоритмів на платформі ASP.NET Core слугує можливість конфігурації командами безпосередньо з бази даних. Як наслідок, завдяки цьому йде менше витрачання часу на виконання операцій, адже взаємодія йде напряму:

public partial class adminlogin : System.Web.UI.Page

{

string strcon = ConfigurationManager.ConnectionStrings["con"].ConnectionString;

protected void Page\_Load(object sender, EventArgs e)

}

// login button click event

protected void Button1\_Click(object sender, EventArgs e)

{

try

{

SqlConnection con = new SqlConnection(strcon);

if (con.State == ConnectionState.Closed)

{

con.Open();

}

SqlCommand cmd = new SqlCommand("select \* from admin\_table where username='" + TextBox1.Text.Trim() + "' AND password='" + TextBox2.Text.Trim() + "'", con);

SqlDataReader dr = cmd.ExecuteReader();

if (dr.HasRows)

{

while (dr.Read())

{

Response.Write("<script>alert('Successful login');</script>");

Session["username"] = dr.GetValue(0).ToString();

Session["fullname"] = dr.GetValue(2).ToString();

Session["role"] = "admin";

//Session["status"] = dr.GetValue(10).ToString();

}

Response.Redirect("homepage.aspx");

}

else

{

Response.Write("<script>alert('Invalid credentials');</script>");

}

}

catch (Exception ex)

{

Response.Write("<script>alert('" + ex.Message + "');</script>");

}

}

}

}

public UserByHobbies Index(int hobby, string name)

{

IQueryable<User> users = db.Users.Include(p => p.hobby);

if (hobby != null && hobby != 0)

{

users = users.Where(p => p.id\_hobby == attributes);

}

if (!String.IsNullOrEmpty(title))

{

users = users.Where(p => p.Name.Contains(name));

}

List<Hobby> hobbies = db.Hobby.ToList();

// устанавливаем начальный элемент, который позволит выбрать всех

companies.Insert(0, new Hobby { Name = "Все", Id = 0 });

UsersListViewModel viewModel = new UsersListViewModel

{

Users = users.ToList(),

Companies = new SelectList(companies, "id\_hobby", "title"),

Title = title

};

return View(viewModel);

}

}

}