Проект

на тема

Web site

Нов български университет Факултет: Бакалавърски факултет

Департамент: Информатика

Програма: Информатика

Курс: CSCB766 Програмиране с ASP.NET

Преподавател: гл. ас. д-р Лъчезар Томов

Александър Тодоров Тодоров, F24020

Специалност: Информатика

Учебна година: 2013/2014

Семестър: Пролетен семестър

Индивидуален протокол: 6371

Съдържание на проекта

1. Въведение
2. Услуги обезпечени от проекта
3. Дизайн и архитектура на проекта
4. Реализация на проекта

Въведение

Приложението представлява уеб сайт , в който могат да се регистрират професионални играчи от League of Legends към определени отбори заедно с треньорите им. Сайта дава пълна информация за имената на играчите, треньорите, играни шампиони, КДА рейтинг, ранк разделен на bronze (B), silver (S), Gold( G) и тн. Имаме възможност за регистрация на нови играчи както и за промяната и изтриването на настящи такива. В таба за детайли може да видим и някои от статистиките на играчите. Имаме разделение и по региони. Сайта би бил интересен за хора, който следят e sport турнирите по League of legends и искат да се запознаят с отделните обори и активните им играчи, както и техните треньори.

Услуги обезпечени от проекта

Сайта цели да запознае потребителя с основна информация относно професионалните отбори и играчи в League of Legends.

Имаме отделни табове за:

-Home

-About

-Players

-Roles

-Coaches

-Regions

Допълните ще беде добавена и възможност за регистрация и логин като администратор или обикновен потребител. Необходимите табове вече са добавени в лейаута на сайта.

След натискане на определения таб трябва да се зарежда база данни с информация за играчите, треньорите и т.н. Базата с данни се реализира чрез натискане на определения таб .

Изисквания към реализацията: проекта е реализиран чрез C#,ASP.NET,JavaScript и MySQL.

За целта е използвано Microsoft Visual Studio Ultimate 2015

Version 12.0.40629.00 Update 5

с включени ASP.NET and Web Tools, ASP.NET Web Frameworks and Tools, SQL Server Data Tools

MVC 5.2.3

Entity Frameworks 6.1.3

Инсталирани са и отделни NuGet Packages необходими за приложението като PagedList.MVC.

Дизайн и архитектура на приложението

Тъй като приложението ще е онлайн базирано, с цел по-лесното експортиране на различни типове хостинг услуги, дизайнът се придържа към най-основните и подържаните от най-много хостинг провайдъри технологии. За обезпечение на данните се използва най-разпространената система за контрол на SQL бази данни – кросплатформената и безплатна оупънсорс MySQL. При проектирането на базата данни трябва да се вземат предвид два основни фактора – бързореакция при обслужване на заявки и ограничение на излишеството на данни.

Първоначално са добавени необходимите табове към лейаута на сайта:

<li>@Html.ActionLink("Home", "Index", "Home")</li>

<li>@Html.ActionLink("About", "About", "Home")</li>

<li>@Html.ActionLink("Players", "Index", "Player")</li>

<li>@Html.ActionLink("Roles", "Index", "Role")</li>

<li>@Html.ActionLink("Coaches", "Index", "Coach")</li>

<li>@Html.ActionLink("Regions", "Index", "Region")</li>

</ul>

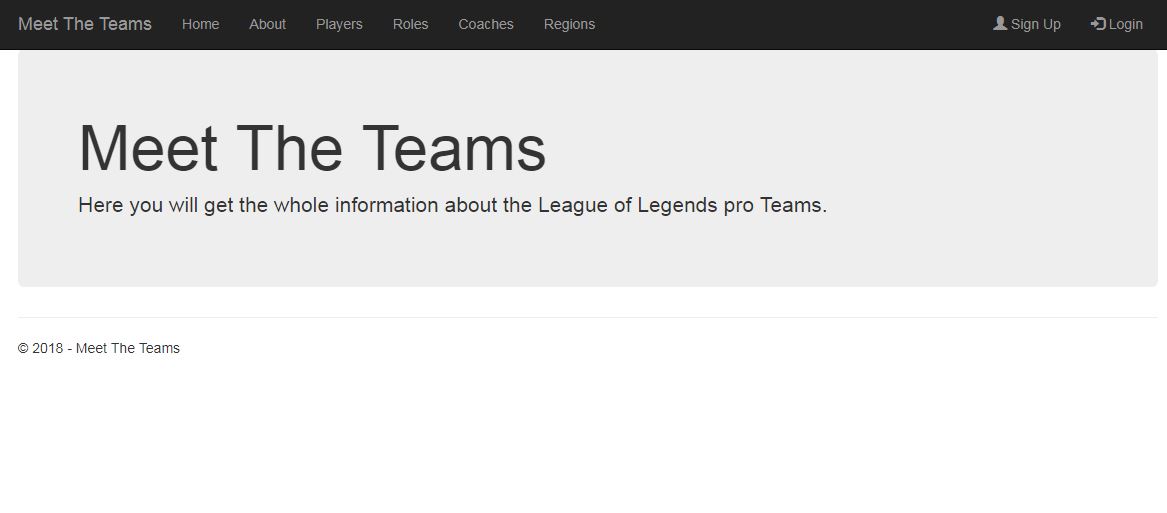
Както споменахме по-горе двата допълнителни таба за регистрация и логин са добавени,но все още са неактини тъй като не са добавени функциите, базата данни за администратор и роли и отделните цонтролери.

<ul class="nav navbar-nav navbar-right">

<li><a href="#"><span class="glyphicon glyphicon-user"></span> Sign Up</a></li>

<li><a href="#"><span class="glyphicon glyphicon-log-in"></span> Login</a></li>

Начален изглед с описаните табове.



Следващата стъпка е създаването на DATA модела, като добавяме отделните класове. Пример Player.cs:

namespace MTT.Models

{

public class Player : Person

{

[DataType(DataType.Date)]

[DisplayFormat(DataFormatString = "{0:yyyy-MM-dd}", ApplyFormatInEditMode = true)]

[Display(Name = "Join Date")]

public DateTime JoinDate { get; set; }

public virtual ICollection<Team> Teams { get; set; }

}

}

За опростяване беше създаден клас Person.cs , към който препращат двата класа Player I Coach.cs като по този начин събираме необходимата за двата класа информация в един.

public abstract class Person

{

public int ID { get; set; }

[Required]

[StringLength(50)]

[Display(Name = "Last Name")]

public string LastName { get; set; }

[Required]

[StringLength(50, ErrorMessage = "First name cannot be longer than 50 characters.")]

[Column("FirstName")]

[Display(Name = "First Name")]

public string FirstMidName { get; set; }

[Display(Name = "Full Name")]

public string FullName

{

get

{

return LastName + ", " + FirstMidName;

}

Имаме отделни класове в Models за Coach, Player, Person, Role, Region и Team. Главният клас, който координира функционалността на Entity Framework за даден модел с данни, е контекстният клас на базата данни. Този клас се създава, като се извлече от класа System.Data.Entity.DbContext. В кода си посочваме кои обекти са включени в модела за данни. Можем също да персонализираме определено поведение на рамката на Entity. В този проект класът се нарича OrganizationContext.

Класа се е създаден в определена за това DAL(Data Access Layer) папка:

public class OrganizationContext : DbContext

{

public DbSet<Role> Roles { get; set; }

public DbSet<Region> Regions { get; set; }

public DbSet<Team> Teams { get; set; }

public DbSet<Coach> Coaches { get; set; }

public DbSet<Player> Players { get; set; }

public DbSet<Person> People { get; set; }

protected override void OnModelCreating(DbModelBuilder modelBuilder)

{

modelBuilder.Conventions.Remove<PluralizingTableNameConvention>();

modelBuilder.Entity<Role>()

.HasMany(c => c.Coaches).WithMany(i => i.Roles)

.Map(t => t.MapLeftKey("RoleID")

.MapRightKey("CoachID")

.ToTable("RoleCoach"));

modelBuilder.Entity<Region>().MapToStoredProcedures();

}

Този код създава свойство DbSet за всеки набор от обекти. В терминологията на Entity Framework набора от обекти обикновено съответства на таблица на базата данни, а предприятието съответства на ред в таблицата.

За да определим връзката добавяме името на низа към конструктора в Web.config:

<connectionStrings>

<add name="OrganizationContext" connectionString="Data Source=(LocalDb)\v12.0;Initial Catalog=MTT2;" providerName="System.Data.SqlClient " />

</connectionStrings>

Рамката на Entity може автоматично да създава (или да пуска и пресъздава) база ни данни, когато се изпълнява приложението. За целта е добавен клас OrganizationInitializer към DAL папката:

public class OrganizationInitializer : System.Data.Entity.DropCreateDatabaseIfModelChanges<OrganizationContext>

{

protected override void Seed(OrganizationContext context)

{

var players = new List<Player>

{

new Player{FirstMidName="Gabriel",LastName="Rau",JoinDate=DateTime.Parse("2018-01-07")},

new Player{FirstMidName="Paul",LastName="Boyer",JoinDate=DateTime.Parse("2016-12-01")},

new Player{FirstMidName="Mads",LastName="Brock-Pedersen",JoinDate=DateTime.Parse("2017-02-06")},

new Player{FirstMidName="Rasmus",LastName="Winther",JoinDate=DateTime.Parse("2016-12-01")},

new Player{FirstMidName="Martin",LastName="Larsson",JoinDate=DateTime.Parse("2015-05-14")},

new Player{FirstMidName="Zdravets",LastName="Iliev Galabov",JoinDate=DateTime.Parse("2017-12-14")},

new Player{FirstMidName="Park",LastName="Kwon-hyuk",JoinDate=DateTime.Parse("2018-01-07")},

new Player{FirstMidName="Lee",LastName="Sang-hyeok",JoinDate=DateTime.Parse("2014-11-23")}

};

players.ForEach(s => context.Players.Add(s));

context.SaveChanges();

var roles = new List<Role>

{

new Role{RoleID=1,Champion="Sion",KDA=9,},

new Role{RoleID=2,Champion="Swain",KDA=4,},

new Role{RoleID=3,Champion="Yasuo",KDA=3,},

new Role{RoleID=4,Champion="Gangplank",KDA=4,},

new Role{RoleID=5,Champion="Zoe",KDA=10,},

.......

Методът SEED приема контекстния обект на база данни като входен параметър и кодът в метода използва този обект за добавяне на нови обекти към базата данни. За всеки тип обект кодът създава колекция от нови обекти, добавя ги в съответната собственост на DbSet и след това записва промените в базата данни.

Необходимо е да извикаме метода SaveChanges след всяка група обекти, както се прави тук, но това ни помага да намерим източника на проблем, ако се получи изключение, докато кодът се записва в базата данни.

След създаването на базата данни преминаваме към създаване на отделните контроллери(Controllers) и изгледи(Views). За целта използваме добавяне на

New Scaffolded Item и MVC 5 Controller with views, using Entity Framework. Указваме на контролерите да използват вече създадения OrganizationContext.

Виждаме, че е създадена променлива от клас, която създава контекст на контекстния обект на базата данни:

private OrganizationContext db = new OrganizationContext();

Методът за действие на индекса получава списък от играчи, зададени от Players, като четат собствеността на Players на контекста на базата данни:

public ViewResult Index

return View(players.ToPagedList)

За да можем да трием, актуализираме и създаваме допълнителна информация към контролерите и изгледите добавяме основни CRUD функции. Сайта позволява детайлен изглед на информацията за играчите, както и възможност да трием, създаваме и актуализираме информацията. Добавяне необходимия ком към Controllers и Views.

Пример PlayerController.cs :

Details:

public ActionResult Details(int? id)

{

if (id == null)

{

return new HttpStatusCodeResult(HttpStatusCode.BadRequest);

}

Player player = db.Players.Find(id);

if (player == null)

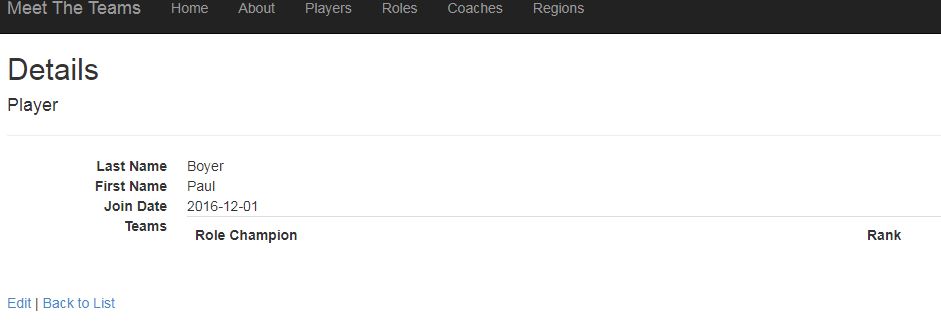
{

return HttpNotFound();

}

return View(player);

}



Create:

public ActionResult Create()

{

return View();

}

[HttpPost]

[ValidateAntiForgeryToken]

public ActionResult Create([Bind(Include = "LastName, FirstMidName, JoinDate")]Player player)

{

try

{

if (ModelState.IsValid)

{

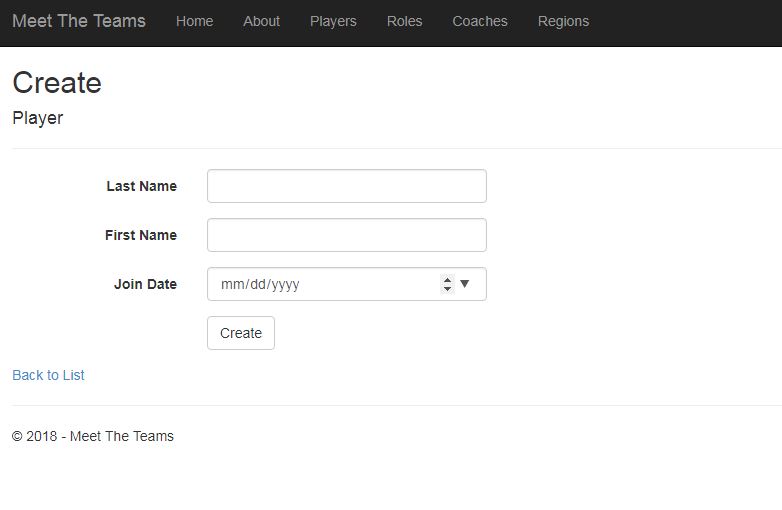
db.Players.Add(player);

db.SaveChanges();

return RedirectToAction("Index");

}

}



Edit:

public ActionResult Edit(int? id)

{

if (id == null)

{

return new HttpStatusCodeResult(HttpStatusCode.BadRequest);

}

Player player = db.Players.Find(id);

if (player == null)

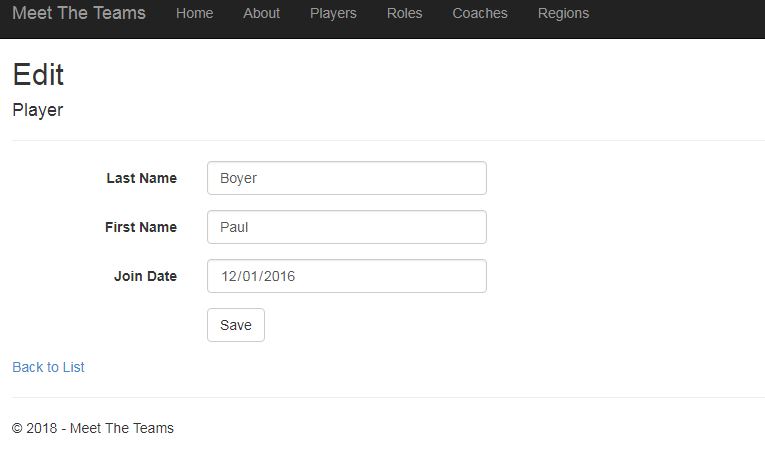
{

return HttpNotFound();

}

return View(player);

}



Delete:

public ActionResult Delete(int? id, bool? saveChangesError = false)

{

if (id == null)

{

return new HttpStatusCodeResult(HttpStatusCode.BadRequest);

}

if (saveChangesError.GetValueOrDefault())

{

ViewBag.ErrorMessage = "Delete failed. Try again, and if the problem persists see your system administrator.";

}

Player player = db.Players.Find(id);

if (player == null)

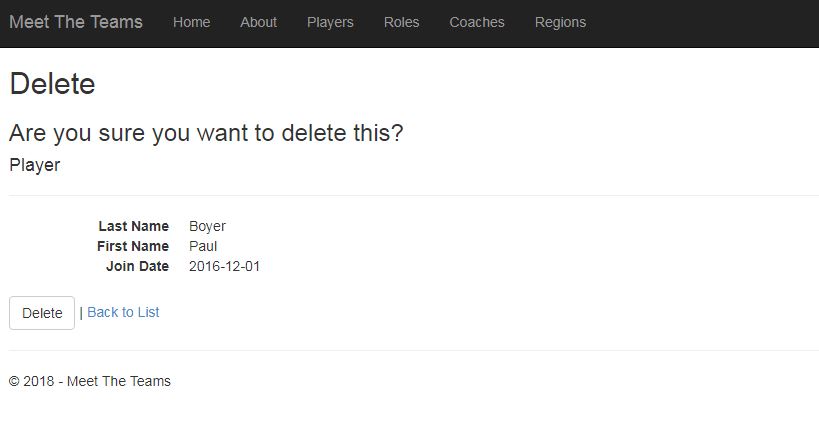
{

return HttpNotFound();

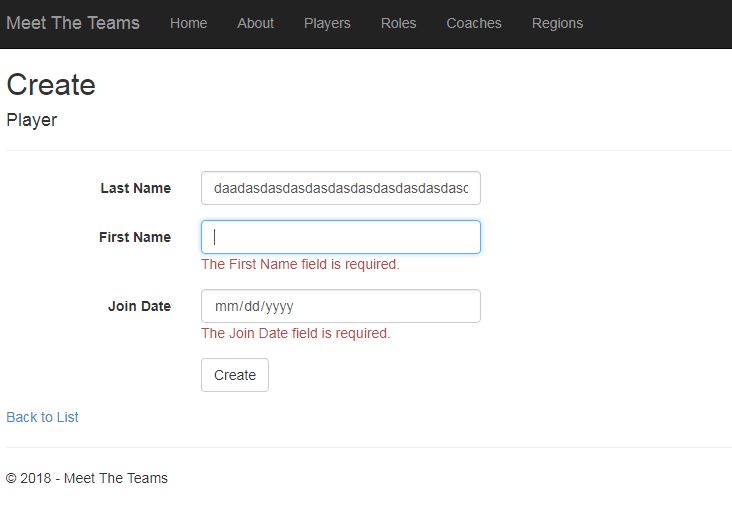
}

return View(player);

}



Добавена е и валидация от сървърна и клиентска страна.



Тъй като беше поставено като изискване са добавени и Async Actions към RegionController:

public async Task<ActionResult> Index()

{

var regions = db.Regions.Include(r => r.Administrator);

return View(await regions.ToListAsync());

}

// GET: Region/Details/5

public async Task<ActionResult> Details(int? id)

{

if (id == null)

{

return new HttpStatusCodeResult(HttpStatusCode.BadRequest);

}

Region region = await db.Regions.FindAsync(id);

if (region == null)

{

return HttpNotFound();

}

return View(region);

}

Целия проект е качен на GitHub:

<https://github.com/CarpeNoctemX/MTT>