Міністерство освіти і науки України

Дніпропетровський національний університет

ім. Олеся Гончара

Факультет фізики, електроніки і комп’ютерних систем

Кафедра електронних обчислювальних машин

**Курсова робота з дисципліни**

**«Інформаційна стійкість інформаційних систем»**

Виконав студент гр. КІ-14м-1

Мага О.Ю.

Перевірив доц.. каф. ЕОМ

Литвинов О.А.

Дніпропетровськ

2014 р.

**Содержание**

[Содержание 2](#_Toc405737323)

[Вступление 3](#_Toc405737324)

[1. Основная часть 4](#_Toc405737325)

[1.1. Требования 4](#_Toc405737326)

[1.2. Выбор архитектуры 5](#_Toc405737327)

[1.3. Выбор инструментов 5](#_Toc405737328)

[1.4. Организация работы 7](#_Toc405737329)

[2. Реализация 10](#_Toc405737330)

[2.1. Создание скрипта 10](#_Toc405737331)

[3. Результаты работы 16](#_Toc405737332)

[Выводы 19](#_Toc405737333)

[Список использованных источников 20](#_Toc405737334)

**Вступление**

В настоящее время устойчивость информационных систем имеет большое значение в процессе жизни и развития любой организации, деятельность короторой так или иначе связана с финансовой деятельностью. Поэтому любой сбой такой ситстемы может стать причиной материального ущерба.

С целью предусмотреть, а также как можно скорее устранить такого рода сбои существуют «лучшие практики» по разработке информационных систем. Таких подходов существует довольно много. Самымы известными среди них являются SCRUM, CMMI и пр.

Данная курсовая работа выполнена с применением практики SCRUM, основных ее этапов и элементов.

**1. Основная часть**

## **1.1. Требования**

Система FootballManager предназначена для администраторов и организаторов футбольных турниров. Администратор может создавать новый турнир, добавлять в него существующие клубы. Система может также использоваться владельцами клубов для добавление новых игроков в свой клуб, осуществлять трансферы.

1. Поддерживаются CRUD – операции для всех типов сущностей, а именно турниры, клубы и игроки.
2. Система должна предоставлять удобный поиск игроков с фильтрами клубов и по номеру игрока.
3. Приложение должно иметь страницу о создателях и их контактные данные.
4. Система должна поддерживать авторизацию для пользователей.

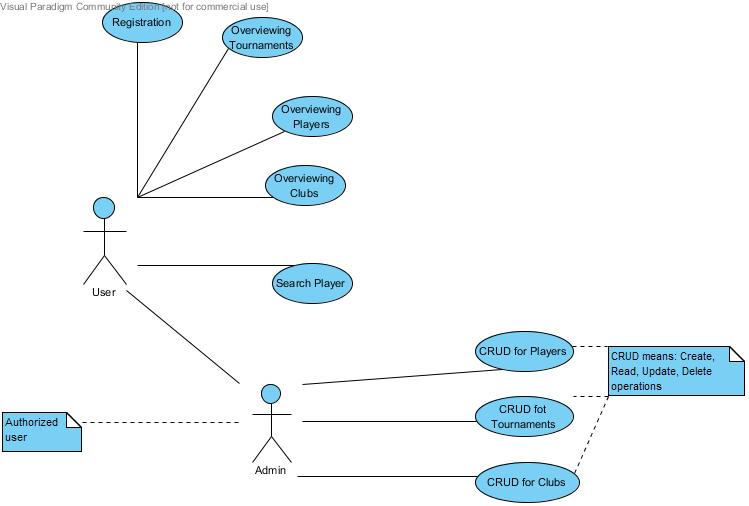


Рис.1 Use-case диаграмма системы Footballmanager.

## **1.2. Выбор архитектуры**

Для реализации поставленных требований, была выбрана двух уровневая архитектура с уровнями Web и DAL.

Уровень Web будет представлять из себя веб-приложение с удобным пользовательским интерфейсом.

Уровень DAL будет обслуживать операции доступа и возврата данных.

## **1.3. Выбор инструментов**

Для решения поставленой задачи, были выбраны следующие инструменты и технологи:

1. Для разработки была выбрана среда Microsoft Vusial Studio 2012, и язык C# .NET, технология ASP .NET MVC 4.
2. Для хранения даннях выбран сервер MSSQL localdb.
3. Для доступа к данням будет использоваться ORM средство Entity Framework 6.0.
4. Для совместной работы над проектом используется система контроля версий Git.
5. Для таск-трекинга был использован ресурс <https://football-manager.visualstudio.com/>

При проектирования уровня DAL была предусмотрена возможность использования различных баз данных. Поэтому был применен паттерн репозитория для работы с сущностями [<http://msdn.microsoft.com/en-us/library/ff649690.aspx>].

public interface IRepository<T> {

    IEnumerable<T> FindAll();

    IEnumerable<T> FindBy(Expression<Func<T, bool>> predicate);

    T FindById(int id);

    void Add(T newEntity);

    void Remove(T entity);

}

Предполагалось что база данных будет использваться и другими приложениями со своими таблицами, поэтому был использован паттерн

Unit of work [<http://design-pattern.ru/patterns/unit-of-work.html>].

public interface IUnitOfWork {

    IRepository<Tournament> Employees { get; }

    IRepository<Club> Orders { get; }

    IRepository<Player> Customers { get; }

    void Commit();

}

Для инверсии зависимостей был использован IoC – контейнер Ninject.

При создании приложения был принят поход Database first [<http://msdn.microsoft.com/ru-ru/data/jj206878>] При котором классы моделей генерируются из существвующей базы данных(рис. 2)

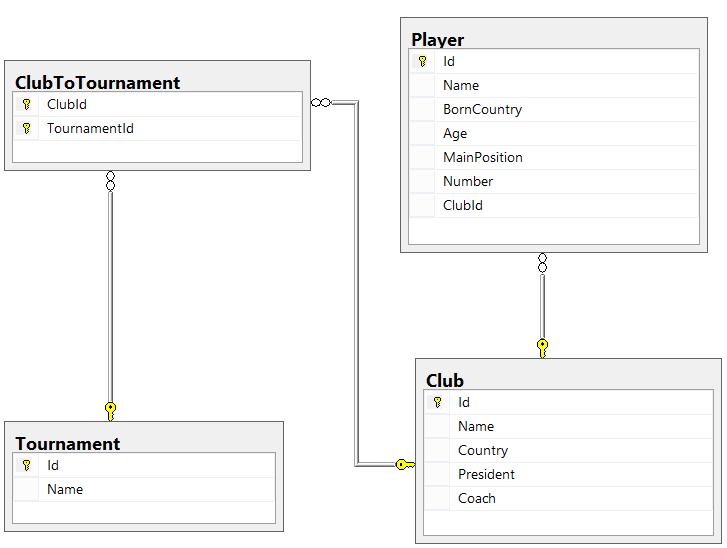


Рис. 2 Схема базы-данных.

Исходя из данных построена диаграмма компонентов (рис.3)

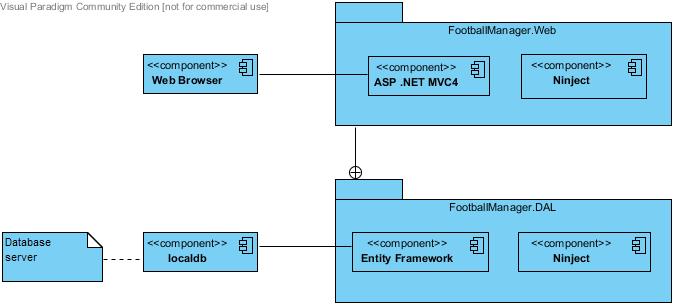


Рис.3 Диаграмма компонентов.

## **1.4. Организация работы**

Согласно Scrum[<http://ru.wikipedia.org/wiki/Scrum>] был составлен Product Backlog.(рис.4)

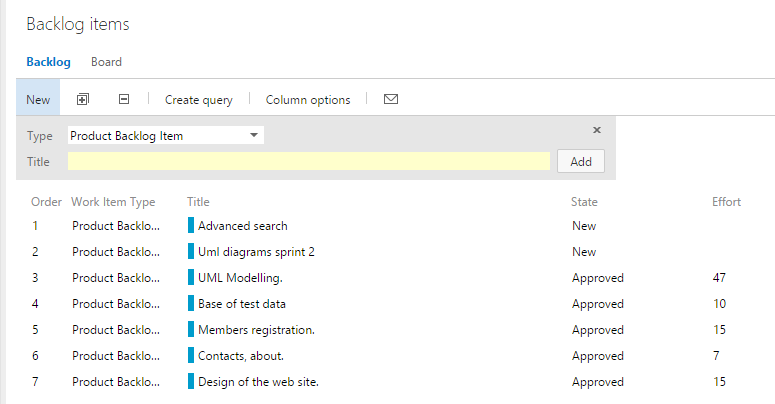


Рис. 4 Product Backlog

Работу планировалось осуществить за 2 спринта по 1 неделе.

На первый спринт были отобраны такие user story (рис .5), на второй (рис.6 и рис. 7)

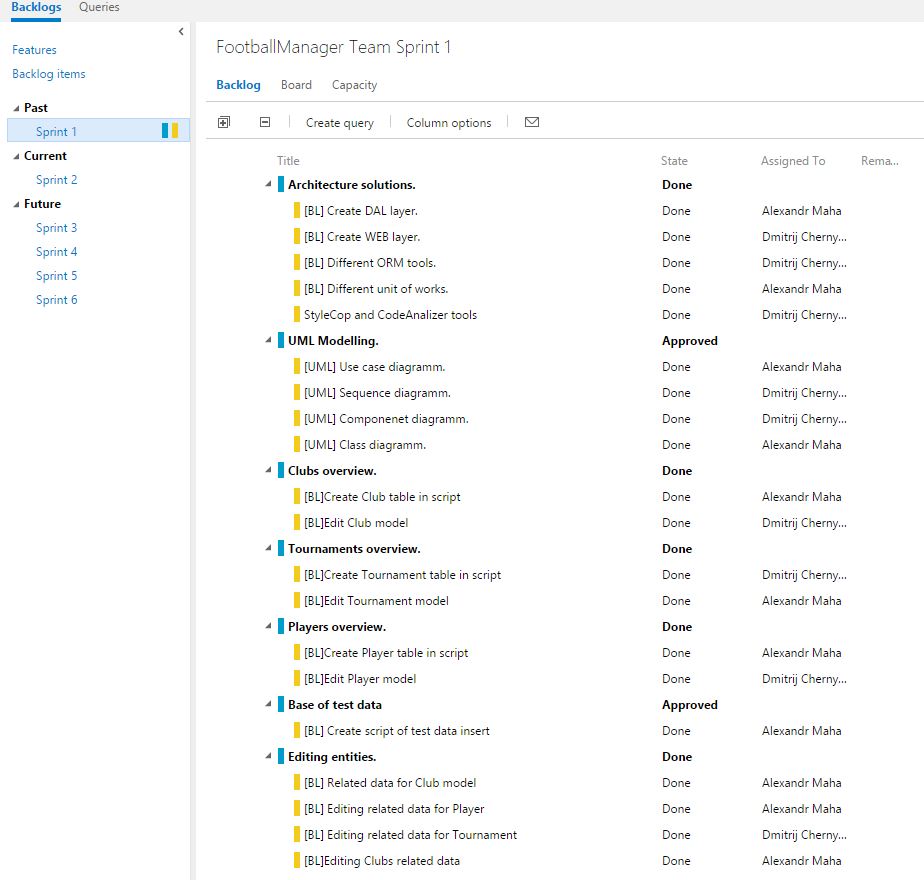


Рис.5 Sprint 1 backlog.

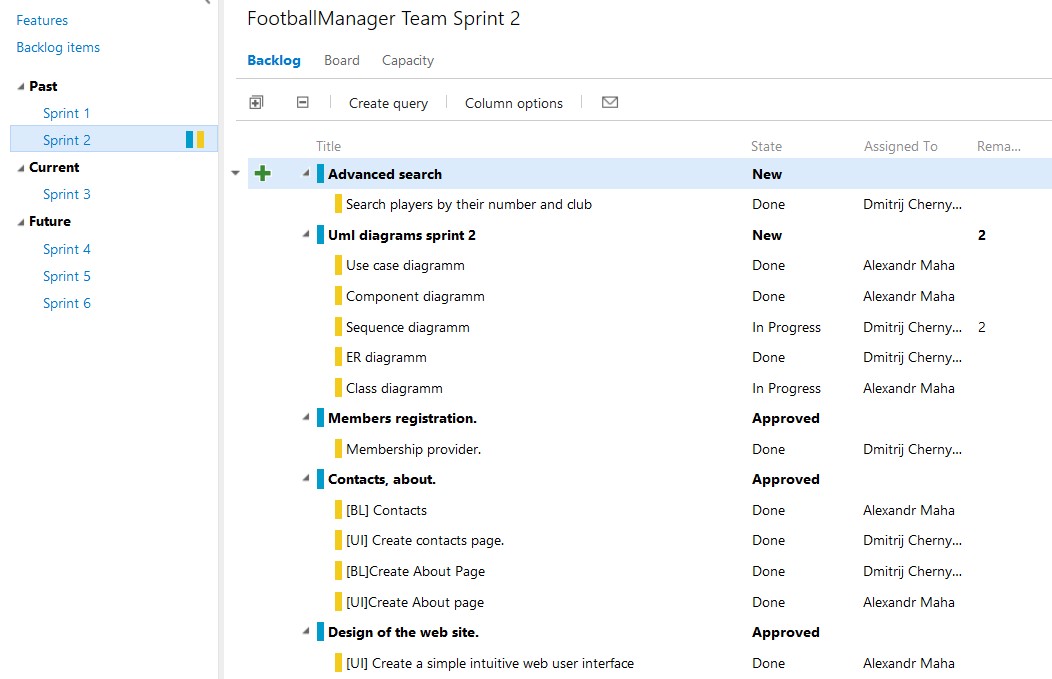


Рис.6 Sprint 2 backlog.

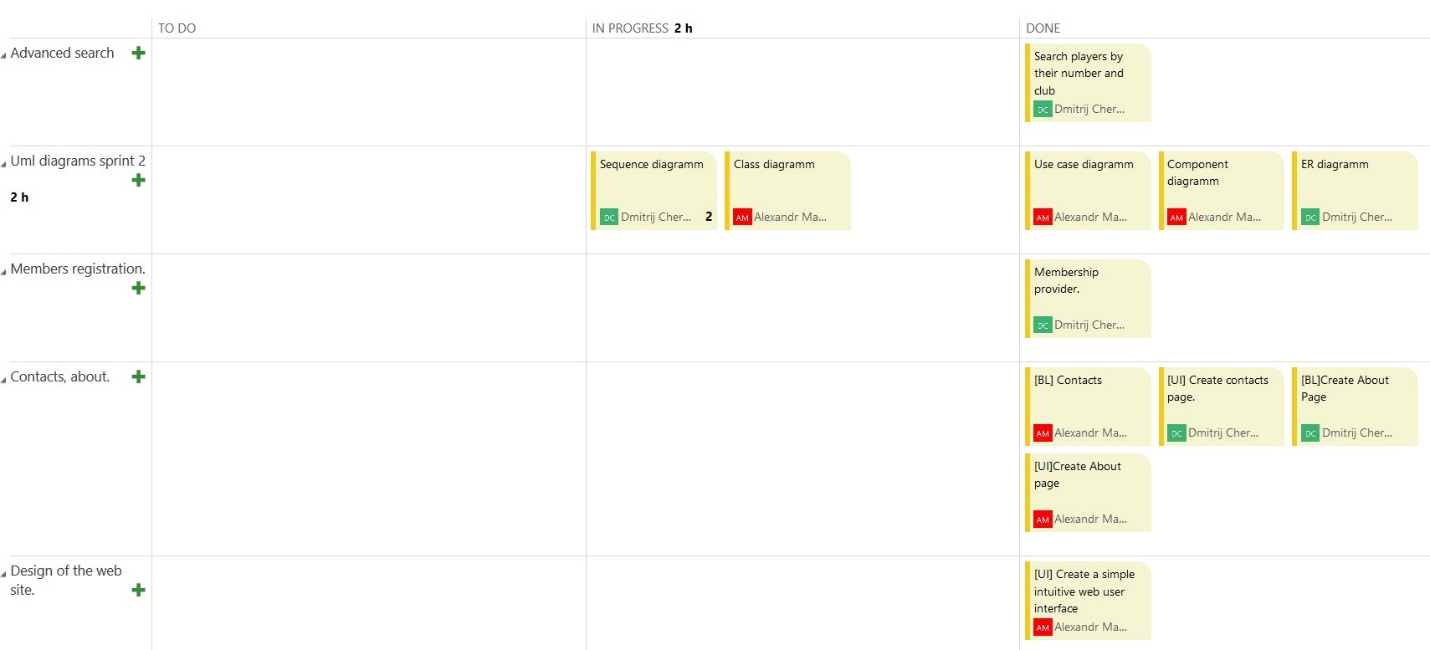


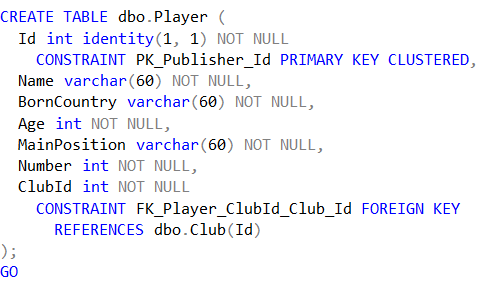
Рис. 7 Sprint 2 board

# 2. Реализация

Рассмотрим на примере сущности Player процесс создания, отображения, а также, работу с Игроками нашего приложения.

## **2.1. Создание скрипта**

Так как в качестве подхода по организации создания моделей была выбрана схема Database first, то первым создается таблица базы данных. Для этого был написан следующий скрипт

,

где задаются поля будущей модели и ограничения в виде внешнего ключа с сущностью Club.

В этом же скрипте задаются значения для заполнения созданной таблицы:



Из скрипта выше (для создания таблицы) получаем модель Player

public partial class Player

{

/// <summary>

/// Gets or sets Player id.

/// </summary>

public int Id { get; set; }

/// <summary>

/// Gets or sets Name.

/// </summary>

[Required]

[StringLength(60)]

public string Name { get; set; }

/// <summary>

/// Gets or sets born country.

/// </summary>

[Required]

[StringLength(60)]

public string BornCountry { get; set; }

/// <summary>

/// Gets or sets age.

/// </summary>

public int Age { get; set; }

/// <summary>

/// Gets or sets main position.

/// </summary>

[Required]

[StringLength(60)]

public string MainPosition { get; set; }

/// <summary>

/// Gets or sets number.

/// </summary>

public int Number { get; set; }

/// <summary>

/// Gets or sets club id.

/// </summary>

public int ClubId { get; set; }

/// <summary>

/// Gets or sets navigation property club.

/// </summary>

public virtual Club Club { get; set; }

}

Далее, используем эту модель в соответствующем контроллере (Player Controller).

[Authorize]

public class PlayersController : Controller

{

/// <summary>

/// unit of work.

/// </summary>

private readonly IFootballManagerUnitOfWork unitOfWork;

public PlayersController(IUnitOfWorkFactory unitOfWorkFactory)

{

this.unitOfWork = unitOfWorkFactory.CreateFootballManagerUnitOfWork();

}

//

// GET: /Players/

[AllowAnonymous]

public ViewResult Index(string club, string playerNumber)

{

IEnumerable<Player> players;

int parsedNumber;

bool isNumber = int.TryParse(playerNumber, out parsedNumber);

if (string.IsNullOrEmpty(club) && string.IsNullOrEmpty(playerNumber))

{

players = unitOfWork.Players.FindAll();

}

else if (string.IsNullOrEmpty(playerNumber) && !isNumber)

{

players = unitOfWork.Players.FindWhere(p => p.Club.Name == club);

}

else if (!string.IsNullOrEmpty(club) && !string.IsNullOrEmpty(playerNumber))

{

players = unitOfWork.Players.FindWhere(p => p.Club.Name == club)

.Where(p => p.Number == parsedNumber);

}

else

{

players = unitOfWork.Players.FindWhere(p => p.Number == parsedNumber);

}

ViewBag.Clubs = new SelectList(unitOfWork.Clubs.FindAll(), "Name", "Name");

return View(players);

}

//

// GET: /Players/Details/5

[AllowAnonymous]

public ViewResult Details(int id)

{

Player player = unitOfWork.Players.FindWhere(x => x.Id == id).Single();

return View(player);

}

//

// GET: /Players/Create

public ActionResult Create()

{

ViewBag.PossibleClubs = unitOfWork.Clubs.FindAll();

return View();

}

//

// POST: /Players/Create

[HttpPost]

public ActionResult Create(Player player)

{

if (ModelState.IsValid)

{

unitOfWork.Players.Add(player);

unitOfWork.Commit();

return RedirectToAction("Index");

}

ViewBag.PossibleClubs = unitOfWork.Clubs.FindAll();

return View(player);

}

//

// GET: /Players/Edit/5

public ActionResult Edit(int id)

{

Player player = unitOfWork.Players.FindWhere(x => x.Id == id).Single();

ViewBag.PossibleClubs = unitOfWork.Clubs.FindAll();

return View(player);

}

//

// POST: /Players/Edit/5

[HttpPost]

public ActionResult Edit(Player player)

{

if (ModelState.IsValid)

{

unitOfWork.Players.Update(player);

unitOfWork.Commit();

return RedirectToAction("Index");

}

ViewBag.PossibleClubs = unitOfWork.Clubs.FindAll();

return View(player);

}

//

// GET: /Players/Delete/5

public ActionResult Delete(int id)

{

Player player = unitOfWork.Players.FindWhere(x => x.Id == id).Single();

return View(player);

}

//

// POST: /Players/Delete/5

[HttpPost, ActionName("Delete")]

public ActionResult DeleteConfirmed(int id)

{

Player player = unitOfWork.Players.FindWhere(x => x.Id == id).Single();

unitOfWork.Players.Remove(player);

unitOfWork.Commit();

return RedirectToAction("Index");

}

[AllowAnonymous]

public ActionResult Browse(string club)

{

// Retrieve Club and its Associated Players from database

var clubModel = unitOfWork.Clubs.FindAll().Include("Players").Single(c => c.Name == club);

return View(clubModel);

}

[AllowAnonymous]

[ChildActionOnly]

public ActionResult ClubsMenu()

{

var clubs = unitOfWork.Clubs.FindAll().ToList();

clubs = clubs.OrderBy(c => c.Name).ToList();

return PartialView(clubs);

}

protected override void Dispose(bool disposing)

{

if (disposing) {

unitOfWork.Dispose();

}

base.Dispose(disposing);

}

}

На примере конкретного действия (action) покажем представление (view) метода Index

@model IEnumerable<SoftServe.FootballManager.DAL.Models.Player>

@{

ViewBag.Title = "Players";

}

<h2>Players</h2>

<p>

@using (Html.BeginForm("Index", "Players", FormMethod.Get))

{

<p>

Club: @Html.DropDownList("Club", ViewBag.Clubs as SelectList, string.Empty)

Player number: @Html.TextBox("playerNumber")

<br />

<input type="submit" value="Search" class="btn-success"/>

</p>

}

</p>

<p>

@Html.ActionLink("Create New", "Create")

</p>

<table>

<tr>

<th></th>

<th>

Name

</th>

<th>

Country

</th>

<th>

Age

</th>

<th>

Position

</th>

<th>

Number

</th>

<th>

Club

</th>

</tr>

@foreach (var item in Model) {

<tr>

<td>

@Html.ActionLink("Edit", "Edit", new { id=item.Id}) |

@Html.ActionLink("Details", "Details", new { id=item.Id }) |

@Html.ActionLink("Delete", "Delete", new { id=item.Id })

</td>

<td>

@item.Name

</td>

<td>

@item.BornCountry

</td>

<td>

@item.Age

</td>

<td>

@item.MainPosition

</td>

<td>

@item.Number

</td>

<td>

@item.Club.Name

</td>

</tr>

}

</table>

Именно данное представление будет передано пользователю в Web UI (в браузере).

# 3. Результаты работы

В качестве результатов проделанной работы приведем примеры страниц, которые видит пользователь в браузере.

Главная страница сайта (рис. 8)

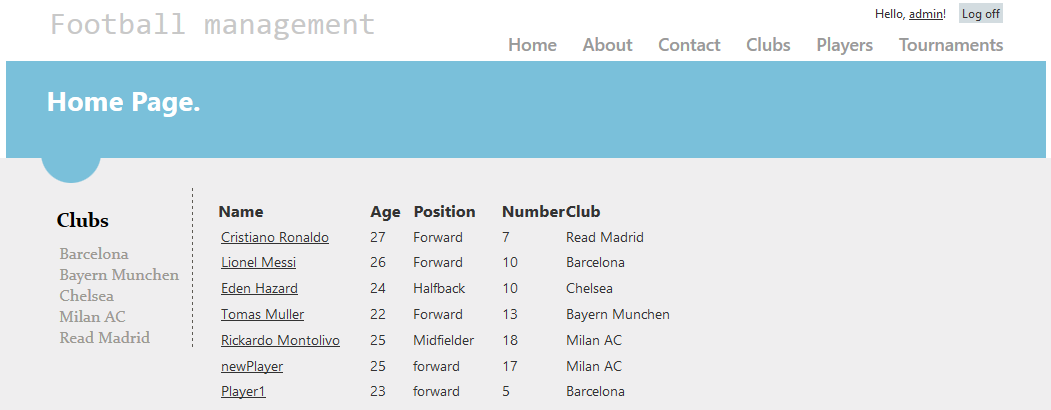


Рис. 8. Главная страница сайта

Меню Players (рис. 9). На этой странице пользователю доступен поиск с фильтром: поиск игрока с заданным номером по определенному клубу

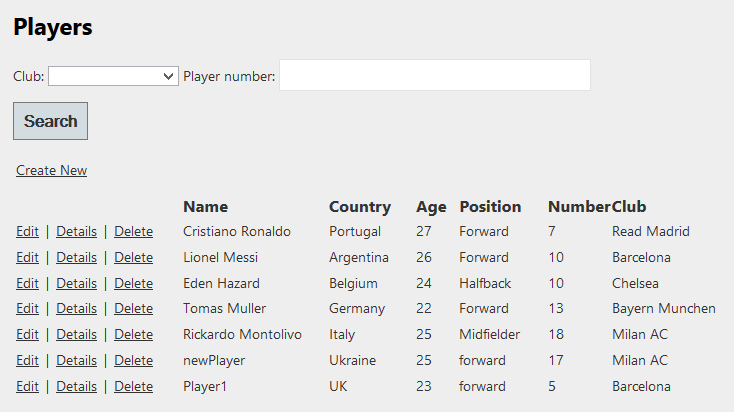


Рис. 9. Меню Players

Далее приведено использование поиска по номеру игроков без учета их клуба (рис. 10).

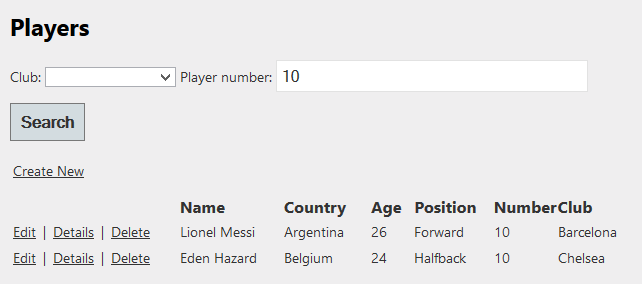


Рис. 10. Поиск игроков по номеру без учета его клуба

Поиск с учетом клуба (рис. 11)

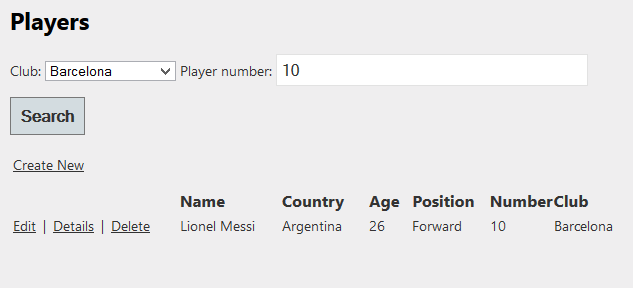


Рис. 11. Поиск игроков по номеру с учетом клуба

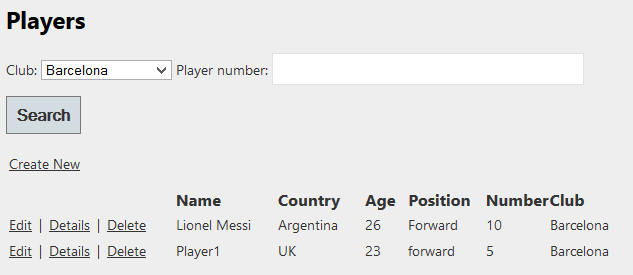
Поиск без учета номера (рис. 12)

Рис. 12. Поиск игрока по клубу без учета его номера

Форма создания игрока (рис. 13)

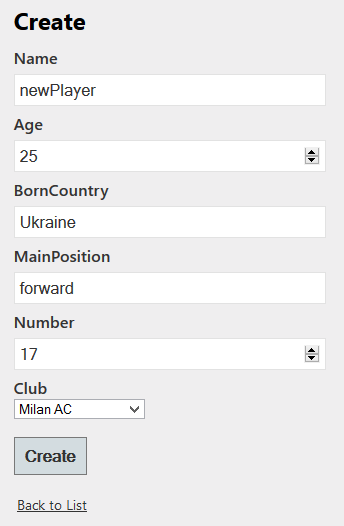


Рис. 13. Создание игрока

Немаловажным моментом является то, что создание, редактирование да и вообще все CRUD операции может выполнять только зарегистрированный пользователь, который является администратором.

# Выводы

В ходе выполнения данной курсовой работы создано полноценное веб-приложение по технологии командной разработки SCRUM. Данная техника позволила участвовать одновременно нескольким людям над выполнением проекта. Была проделана работа по планированию, созданы требования, которые потом разбились на конкретные задачи (с ограничениями по времени) и были назначены конкретным людям. В процессе выполнения был закреплен навык работы в команде.

В практической части курсовой работы представлен, на примерах, процесс разработки и построения приложения. В пункте реализация показан процесс взаимодействия пользователя с системой: поиск с использованием фильтров, создание нового игрока и его редактирование.

# Список использованных источников

1. <http://msdn.microsoft.com/en-us/library/ff649690.aspx> - The Repository Pattern

2. <http://design-pattern.ru/patterns/unit-of-work.html>- Unit of Work (Единица работы)

3. <http://msdn.microsoft.com/ru-ru/data/jj206878> - Data access and storage.

4. Adam Freeman - Pro ASP.NET MVC 5 (Expert's Voice in ASP.Net) Paperback – December 20, 2013

5. [Brian Driscoll](http://www.amazon.com/Brian-Driscoll/e/B00J1SF8ZY/ref=dp_byline_cont_book_1), [Nitin Gupta](http://www.amazon.com/s/ref=dp_byline_sr_book_2?ie=UTF8&field-author=Nitin+Gupta&search-alias=books&text=Nitin+Gupta&sort=relevancerank) - Entity Framework 6 Recipes Paperback – October 29, 2013