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](#_Toc405887875)

[Вступление 3](#_Toc405887876)

[1.Основная часть 4](#_Toc405887877)

[1.1.Бизнес Требования 4](#_Toc405887878)

[1.2.Функциональные требования 4](#_Toc405887879)

[1.3.Выбор архитектуры 7](#_Toc405887880)

[1.4.Выбор инструментов 7](#_Toc405887881)

[1.5. Организация работы 10](#_Toc405887882)

[2. Реализация 13](#_Toc405887883)

[3.Результаты работы 17](#_Toc405887884)

[Выводы 20](#_Toc405887885)

[Список использованных источников 21](#_Toc405887886)

[ПРИЛОЖЕНИЕ А. 22](#_Toc405887887)

**Вступление**

В настоящее время устойчивость информационных систем имеет большое значение в процес се 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 достаточно много. Самымы известными среди них являються SCRUM, CMMI и пр.

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

**1.Основная часть**

## 1.1.Бизнес 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 редактирование клубов, игроков, турниров.

## 1.2.Функциональные требования

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.

Use Case 1. Регистрация.

Базовый сценарий

1.Пользователь входит на сайт.

2.Пользователь регистрируется

3.Система сохраняет его данные в базу.

4.Пользователь автоматически авторизируется.

Альтернативная

2.1.Введенные данных некорректны.

2.1.1.Система возвращает ошибку.

2.1.2.Предложение ввести новые данные.

Post-condition: Пользователь становится админом и может осуществлять CRUD операции с сущностями.

Use Case 2. Создание нового клуба.

Pre-condition: пользователь должен авторизоваться чтобы стать админом.

Базовый сценарий

1.Администратор нажимает кнопку создать новый клуб.

2.Система предоставляет форму для создания.

3.Админ вводит данные, и сохраняет.

4.Система заносит данные в базу.

Use Case 3. Удаление клуба.

Pre-condition: see Use case 2.

Базовый сценарий

1.Админ нажимает кнопку удалить турнир.

2.Система выдает окно, с подтверждением удаления.

3.Система удаляет турнир из базы. Возвращает главную страницу.

Альтернативная

2.1.Админ отменяет предложение удаления.

2.1.1.Система перенаправляет на главную страницу.

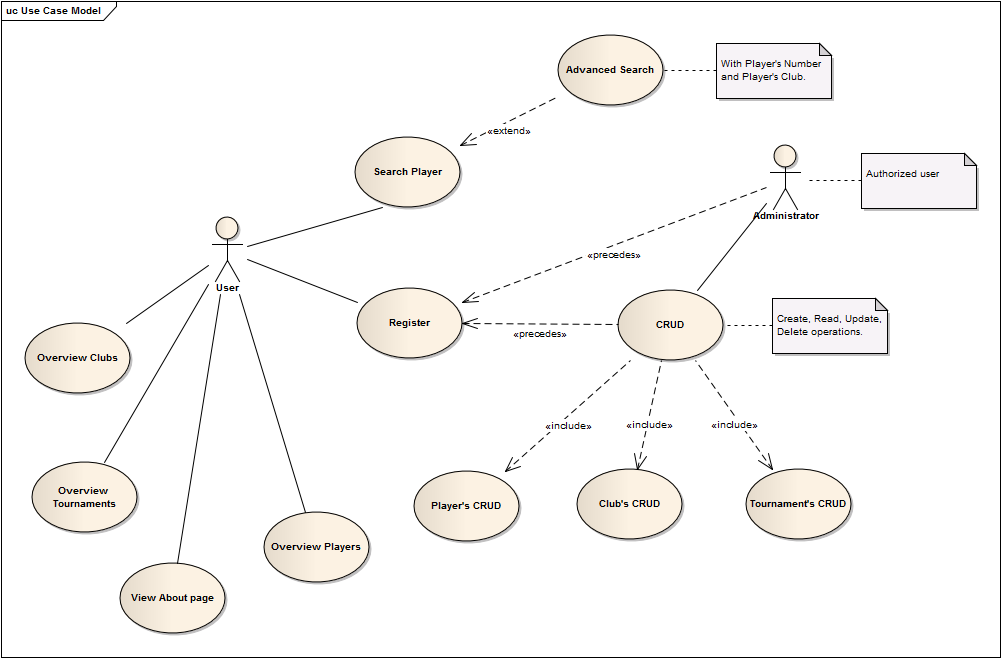


Рис.1 Use-case диаграмма системы Footballmanager.

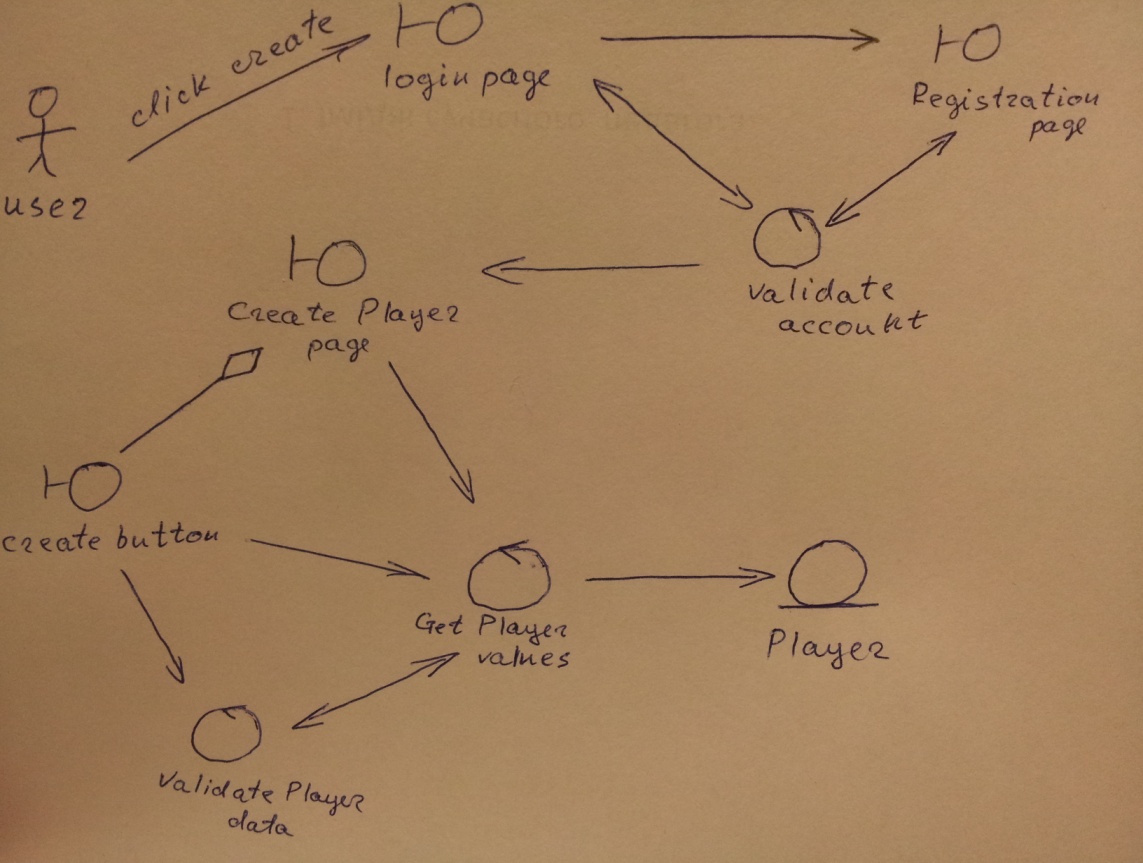


Рис. 2. Robustnessdiagram

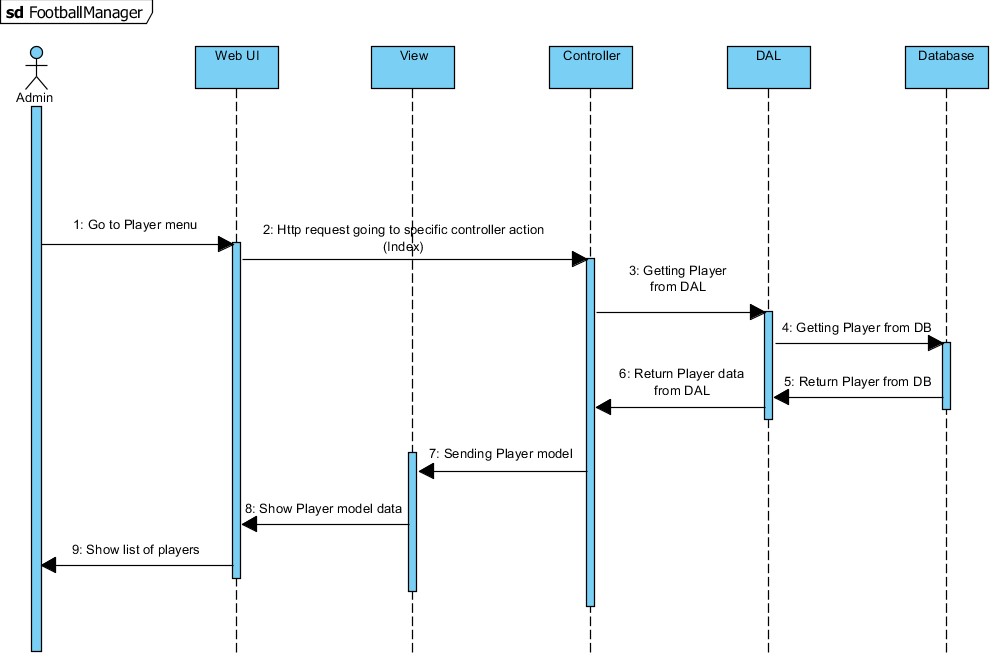


Рис. 3. Диаграмма последовательности

## 1.3.Выбор архитектуры

Для реализации поставленных требований, была выбрана 4 уровневая архитектура с такими уровнями : базы данных, моделей, бизнес логики, и пользовательского интерфейса. Уровни базы данных и модели, будут сожержаться в проекте FootballManager.DAL, а урони бизнесс логики и пользовательского интерфейса в проекте FootballManager.Web

## 1.4.Выбор инструментов

Для решение 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/>
6. Для модульного тестирования MSUnit Framework , library Moq.

При проектирования уровня DAL была предусмотрена возможность использования различных баз данных. Поэтому был примене 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)

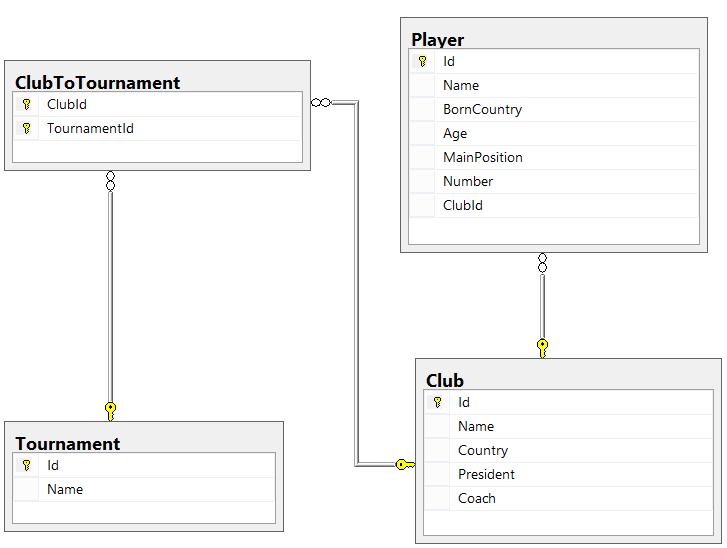


Рис. 2 Схема базы-данных.

Исходя из данных построена диаграмма компонентов (рис.3)

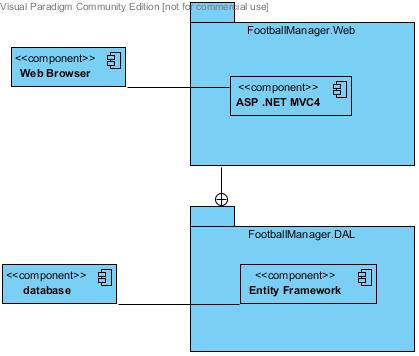


Рис.3 Высокоуровневая Диаграмма компонентов

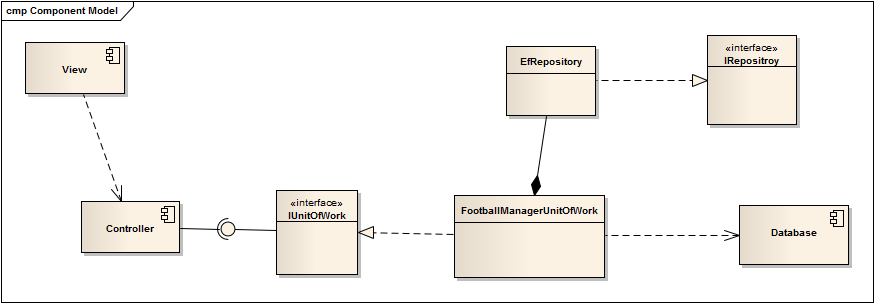


Рис.3.1 Диаграмма компонентов системы.

## 1.5. Организация работы

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

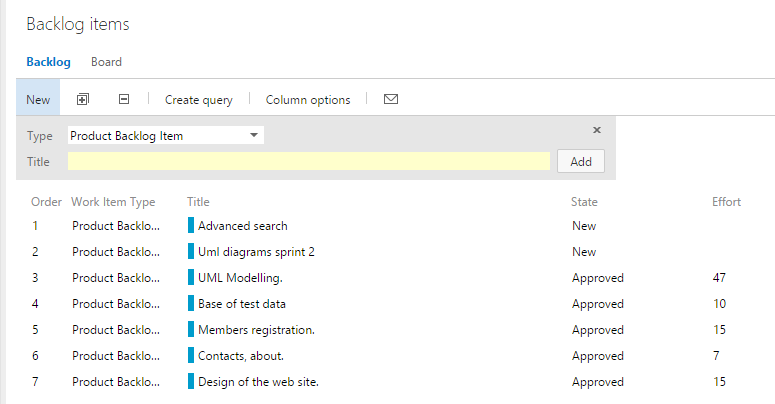


Рис. 4 Product Backlog

Работу планировалось осуществить за 2 спринта по 1 неделе.

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

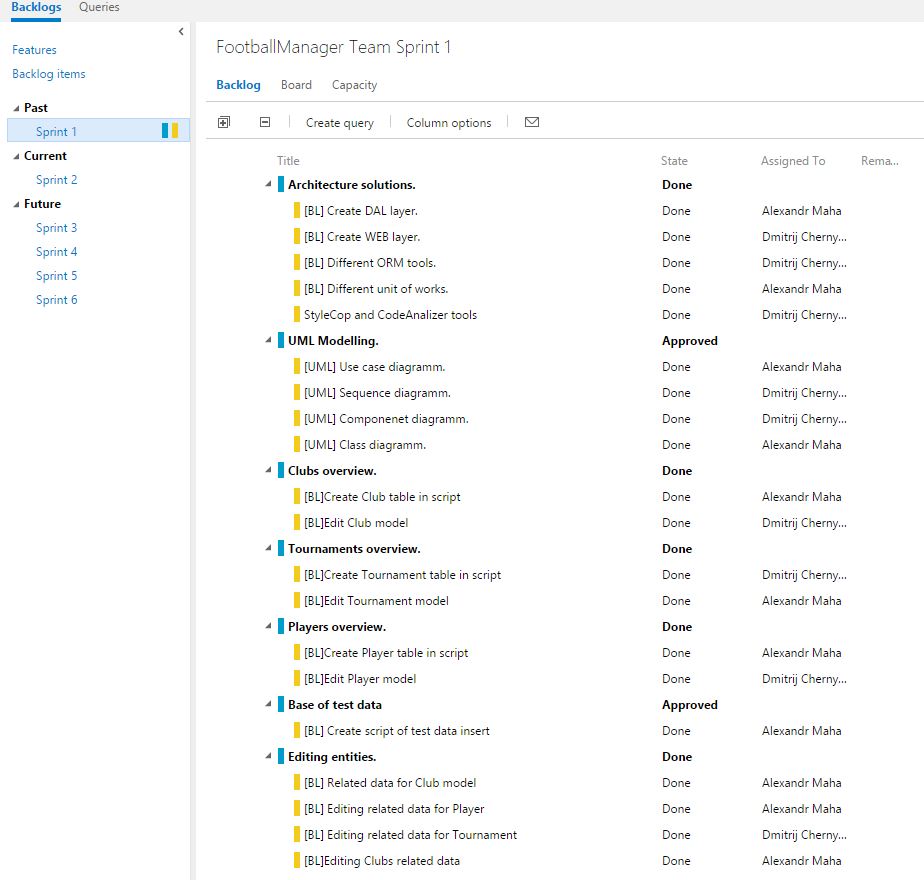


Рис.5 Sprint 1 backlog.

Burndown for sprint 1.

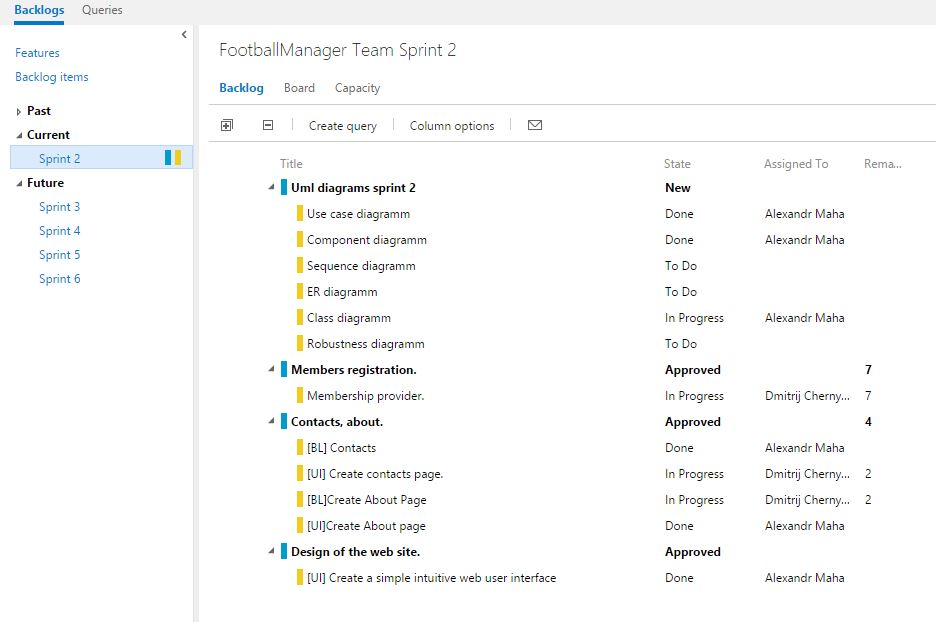
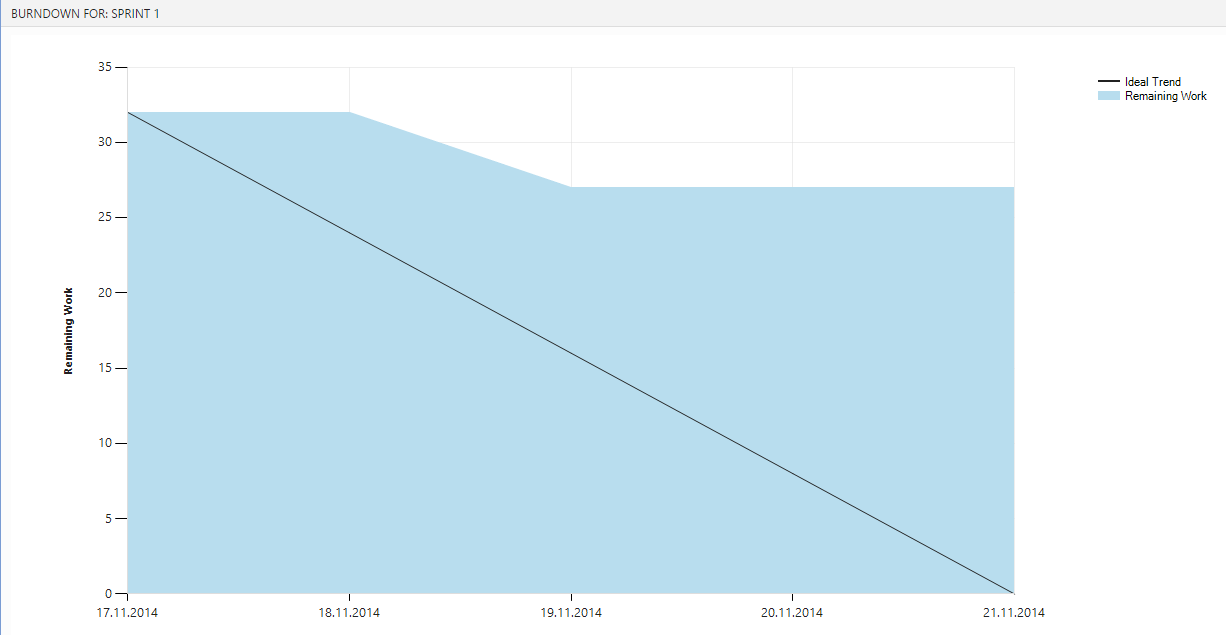
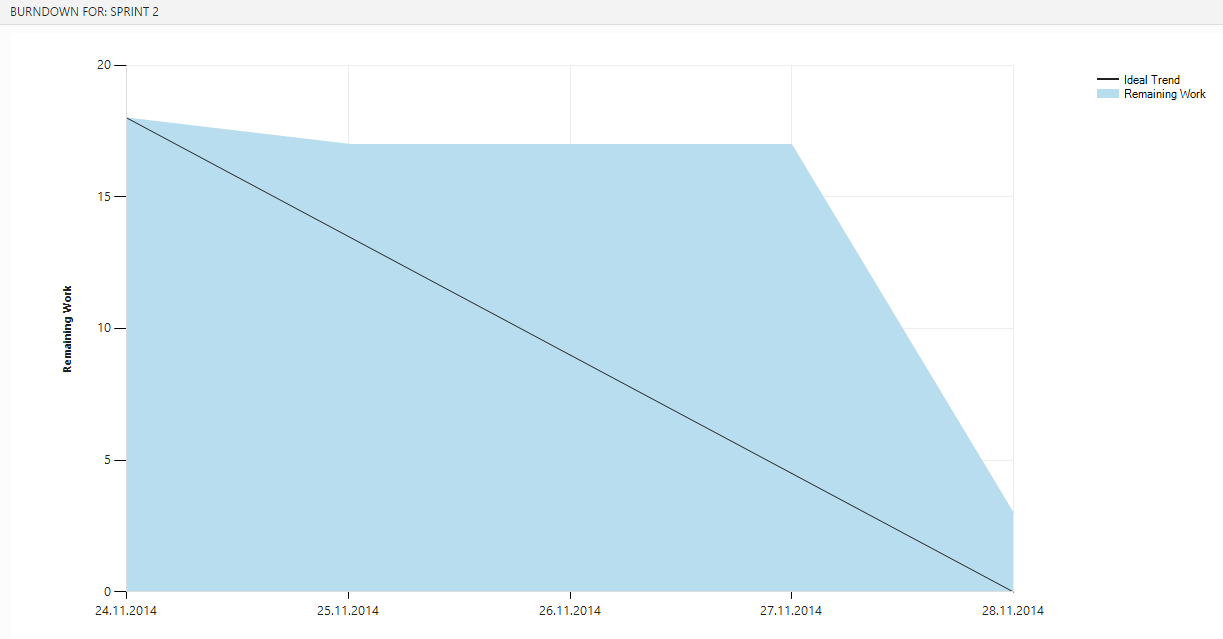


Рис.6 Sprint 2 backlog.

Burndown for sprint 2.



# 2. Реализация

3.Так как используется подход Database first. Необходимо написать скрипт создание и заполнения таблицы тестовыми данными.

Листинг 1. Скрипт создания таблицы:

CREATE TABLE dbo.Club (

Id int identity(1, 1) NOT NULL

CONSTRAINT PK\_Club\_Id PRIMARY KEY CLUSTERED,

Name varchar(60) NOT NULL

CONSTRAINT Club\_Name UNIQUE(Name),

Country varchar(30) NOT NULL,

President varchar(60) NOT NULL,

Coach varchar(60) NOT NULL,

);

GO

Листинг 2. Скрипт заполнения тестовыми данными:

SET IDENTITY\_INSERT dbo.Club ON;

INSERT INTO dbo.Club (

[Id], [Name],[Country],[President],[Coach]

) VALUES

(1, 'Read Madrid','Spain','Florentino Peres','Carlo Ancelotti'),

(2, 'Barcelona','Spain','Joseph Bartomeu','Louis Enrique'),

(3,'Chelsea','England','Roman Abramovich','Jose Mourinho'),

(4,'Bayern Munchen','Germany','Karl Hophner', 'Josep Guardiola'),

(5,'Milan AC','Italy','Silvio Berluskoni','Filippo Indzaggi')

GO

SET IDENTITY\_INSERT dbo.Club OFF;

Далее были сгенерированы классы моделей.

Листинг 3. Код класса Club.

[Table("Club")]

public partial class Club {

public Club() {

this.Players = new HashSet<Player>();

this.Tournaments = new HashSet<Tournament>();

}

public int Id { get; set; }

[Required]

[StringLength(60)]

public string Name { get; set; }

[Required]

[StringLength(30)]

public string Country { get; set; }

[Required]

[StringLength(60)]

public string President { get; set; }

[Required]

[StringLength(60)]

public string Coach { get; set; }

public virtual ICollection<Player> Players { get; set; }

public virtual ICollection<Tournament> Tournaments { get; set; }

}

За главную страницу сайта отвечает HomeController и его Index View.

Было решено отображать на ней список всех игроков, и левое меню с клубами.

Листинг 4. Код класса HomeController

public class HomeController : Controller

{

private readonly IFootballManagerUnitOfWork unitOfWork;

public HomeController(IUnitOfWorkFactory unitOfWorkFactory)

{

this.unitOfWork = unitOfWorkFactory.CreateFootballManagerUnitOfWork();

}

public ActionResult Index()

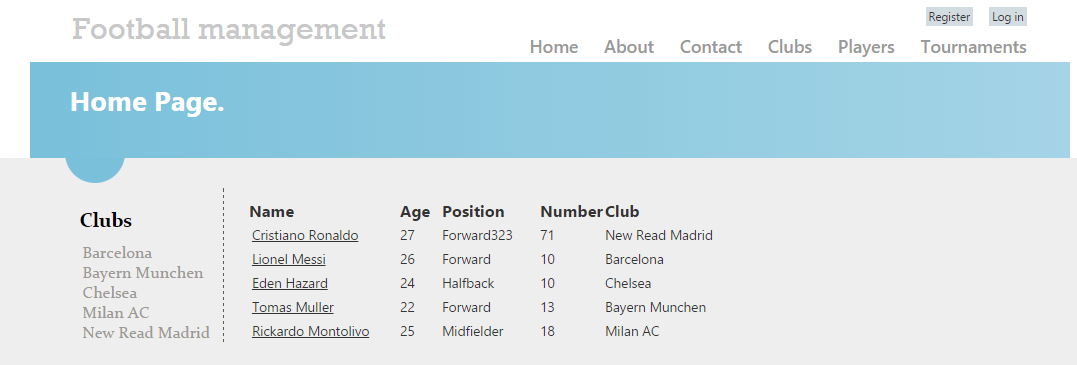
{

var players = unitOfWork.Players.FindAll().Include(p=>p.Club).ToList();

return View(players);

}

Главная страница сайта:



Пример редактирования существующего клуба.

Пользователь нажимает кнопку редактировать, отображается Edit View.

Листинг 1. Код Clubs Edit view.

@model SoftServe.FootballManager.DAL.Models.Club

@{

ViewBag.Title = "Edit";

}

<h2>Edit</h2>

@using (Html.BeginForm()) {

@Html.ValidationSummary(true)

<fieldset>

<legend>Club</legend>

@Html.HiddenFor(model => model.Id)

@Html.Partial("\_CreateOrEdit", Model)

<p>

<input type="submit" value="Save" />

</p>

</fieldset>

}

<div>

@Html.ActionLink("Back to List", "Index")

</div>

И пользвателю приходит \_CreateOrEdit View, с полями для ввода новых значений. Дальше вызывается метод [HttpPost]Edit в контроллере TournamentsController, после чего обновляются данные.

Листинг 2. Метод Edit контроллера Clubs.

// POST: /Clubs/Edit/5

[HttpPost]

public ActionResult Edit(int id, string[] selectedTournaments)

{

Club club = unitOfWork.Clubs.FindWhere(x => x.Id == id).Single();

if (TryUpdateModel(club, "",

new string[] { "Name", "Clubs" }))

{

try

{

UpdateClubTournaments(selectedTournaments, club);

unitOfWork.Clubs.Update(club);

unitOfWork.Commit();

return RedirectToAction("Index");

}

catch (DataException)

{

ModelState.AddModelError("", "Unable to save changes. Try again, and if the problem persists see your system administrator.");

}

}

PopulateAssignedTournaments(club);

return View(club);

}

И происходит переадресация на страницу Index, где пользователь может видеть клуб с новыми данными которыми он ввел. Контроллер обращается к unitOfwork тот в свою очередь к репозиторию Tournaments, где вызывается метод Update

Листинг 3. Метод Update класса EfRepository.

public void Update(T updatedEntity)

{

this.objectSet.Attach(updatedEntity);

this.objectSet.Context.ObjectStateManager.ChangeObjectState(updatedEntity, EntityState.Modified);

}

Модульный тест на метод Update.

Листинг 4. Код Unit теста.

[TestMethod]

public void Update\_NewDataForExistingTournament\_TournamentDataChanged()

{

var repositoryMock = new Mock<IRepository<Tournament>>();

Tournament oldTournament = new Tournament { Id = 1, Name = "Tournament 1" };

Tournament expectedTournament = new Tournament { Id = 2, Name = "Tournament" };

List<Tournament> tournaments = new List<Tournament>();

repositoryMock.Setup(r => r.Update(It.IsAny<Tournament>()))

.Callback((Tournament tournament) =>

{

oldTournament.Id = tournament.Id;

oldTournament.Name = tournament.Name;

});

IRepository<Tournament> tournamentRepository = repositoryMock.Object;

tournamentRepository.Update(expectedTournament);

Assert.AreEqual(oldTournament.Id, expectedTournament.Id);

}

# 3.Результаты работы

После авторизации, пользователь становится админом, заходит на страницу клубов, по адресу <http://localhost:50457/Clubs>. (рис. 7)

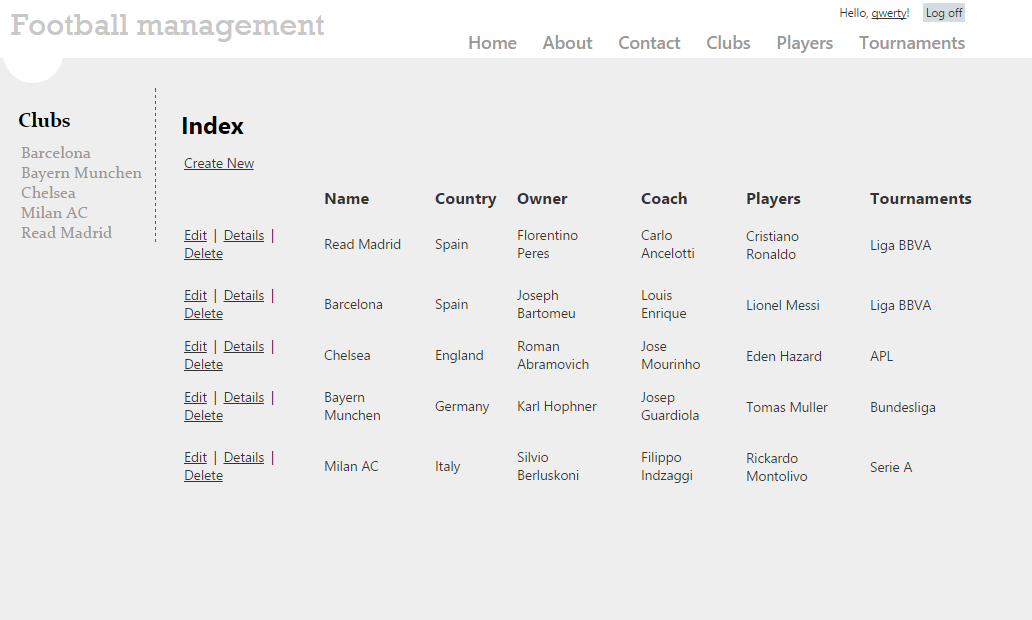


Рис.7 Главная страница клубов.

После нажатия кнопки Edit.(рис.8)

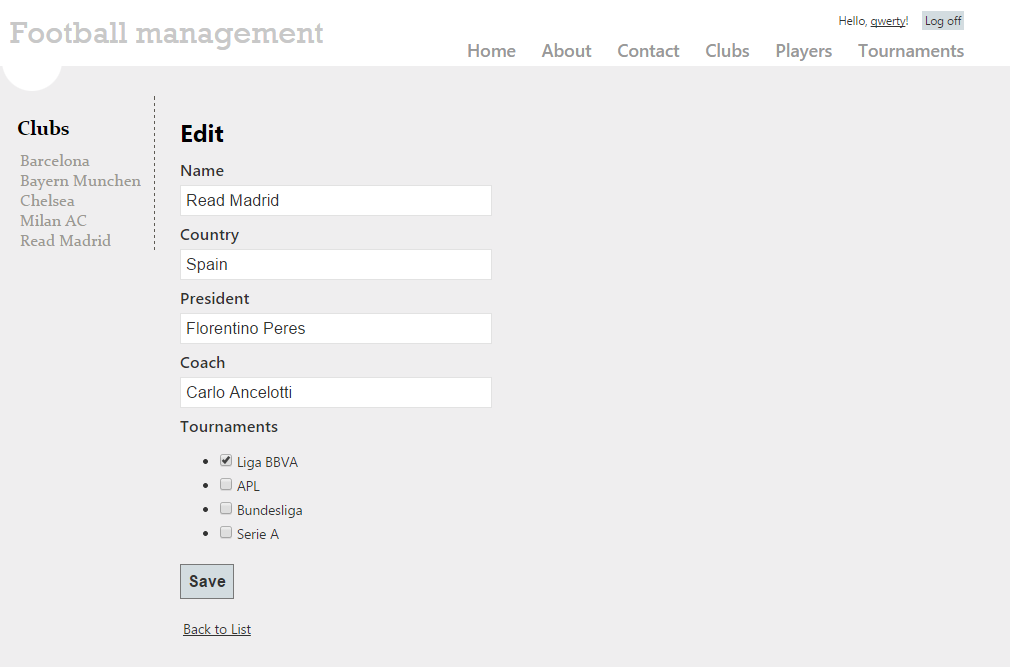


Рис.8 Страница редактирования клуба.

После введения новых значений(рис.9)

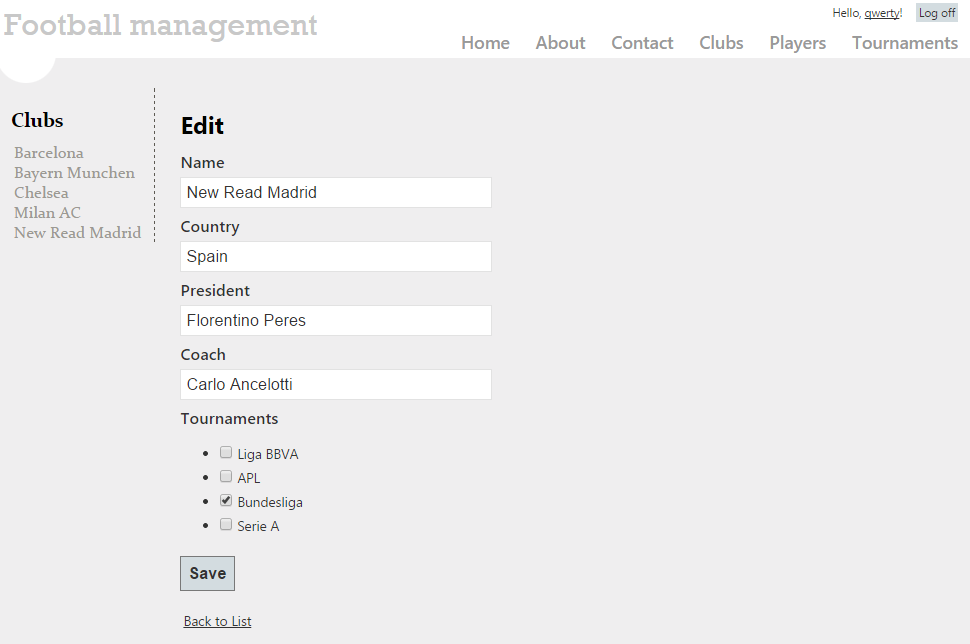


Рис.9 Новые значения для клуба.

После перенаправления на главную страницу клуба, видим изменившуюся информацию по клубу (рис.10)

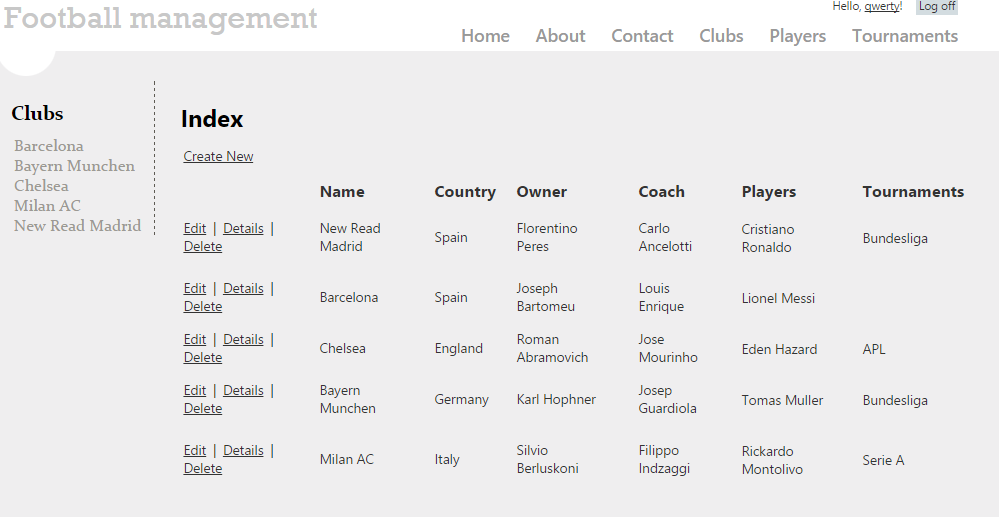


Рис.10 Клуб после редактирования.

# 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

# ПРИЛОЖЕНИЕ А.

namespace SoftServe.FootballManager.DAL

{

using Ninject.Modules;

using SoftServe.FootballManager.DAL.Contracts;

using SoftServe.FootballManager.DAL.Services;

/// <summary>

/// DataAccessModule class.

/// </summary>

public class NinjectDataAccessModule : NinjectModule

{

/// <summary>

/// Method on load.

/// </summary>

public override void Load()

{

this.Bind<IUnitOfWorkFactory>().To<EFUnitOfWorkFactory>();

}

}

}

Код

namespace SoftServe.FootballManager.DAL.Services

{

using System;

using System.Data.Entity;

using System.Data.Entity.Core.Objects;

using System.Linq;

using System.Linq.Expressions;

using SoftServe.FootballManager.DAL.Contracts;

/// <summary>

/// Entity Framework implementation of the IRepository contract.

/// </summary>

/// <typeparam name="T">The type from the store.</typeparam>

internal class EFRepository<T> : IRepository<T>

where T : class

{

/// <summary>

/// Context object.

/// </summary>

private ObjectSet<T> objectSet;

/// <summary>

/// Initializes a new instance of the <see cref="EFRepository{T}"/> class.

/// </summary>

/// <param name="context">

/// Data store context.

/// </param>>

public EFRepository(ObjectContext context)

{

this.objectSet = context.CreateObjectSet<T>();

}

/// <summary>

/// Gets all the data from the context.

/// </summary>

/// <returns>All elements from the context.</returns>

public IQueryable<T> FindAll()

{

return this.objectSet;

}

/// <summary>

/// Gets specified elements from the context.

/// </summary>

/// <param name="predicate">Condition to find elements.</param>

/// <returns>Specified elements from context.</returns>

public IQueryable<T> FindWhere(Expression<Func<T, bool>> predicate)

{

return this.objectSet.Where(predicate);

}

/// <summary>

/// Adds specified element to the context.

/// </summary>

/// <param name="newEntity">The element to add.</param>

public void Add(T newEntity)

{

this.objectSet.AddObject(newEntity);

}

/// <summary>

/// Removes specified element from the context.

/// </summary>

/// <param name="entity">The element to remove.</param>

public void Remove(T entity)

{

this.objectSet.DeleteObject(entity);

}

/// <summary>

/// Updates entity.

/// </summary>

/// <param name="updatedEntity">Entity to update.</param>

public void Update(T updatedEntity)

{

this.objectSet.Attach(updatedEntity);

this.objectSet.Context.ObjectStateManager.ChangeObjectState(updatedEntity, EntityState.Modified);

}

}

}

Код:

namespace SoftServe.FootballManager.DAL.Services

{

using System.Data.Entity.Core.Objects;

using System.Data.Entity.Infrastructure;

using SoftServe.FootballManager.DAL.Contracts;

using SoftServe.FootballManager.DAL.Models;

/// <summary>

/// Describe methods to work with the store.

/// </summary>

public class EFFootballManagerUnitOfWork : IFootballManagerUnitOfWork

{

/// <summary>

/// Context of the data source.

/// </summary>

private readonly ObjectContext context;

/// <summary>

/// Player EF repository.

/// </summary>

private EFRepository<Club> clubs;

/// <summary>

/// Player EF repository.

/// </summary>

private EFRepository<Player> players;

/// <summary>

/// Role EF repository.

/// </summary>

private EFRepository<Tournament> tournaments;

/// <summary>

/// Initializes a new instance of the <see cref="EFFootballManagerUnitOfWork"/> class.

/// </summary>

public EFFootballManagerUnitOfWork()

{

this.context = ((IObjectContextAdapter)new FootballManagerContext()).ObjectContext;

this.context.ContextOptions.LazyLoadingEnabled = true;

}

/// <summary>

/// Gets players repository.

/// </summary>

/// <value>

/// The players.

/// </value>

public IRepository<Club> Clubs

{

get

{

if (this.clubs == null)

{

this.clubs = new EFRepository<Club>(this.context);

}

return this.clubs;

}

}

/// <summary>

/// Gets teamPlayers repository.

/// </summary>

/// <value>

/// The team players.

/// </value>

public IRepository<Player> Players

{

get

{

if (this.players == null)

{

this.players = new EFRepository<Player>(this.context);

}

return this.players;

}

}

/// <summary>

/// Gets teamPlayers repository.

/// </summary>

/// <value>

/// The team players.

/// </value>

public IRepository<Tournament> Tournaments

{

get

{

if (this.tournaments == null)

{

this.tournaments = new EFRepository<Tournament>(this.context);

}

return this.tournaments;

}

}

/// <summary>

/// Commits all the changes.

/// </summary>

public void Commit()

{

this.context.SaveChanges();

}

/// <summary>

/// IDisposable.Dispose method implementation.

/// </summary>

public void Dispose()

{

this.context.Dispose();

}

}

}