МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

БЕЛОРУССКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

ФАКУЛЬТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И РОБОТОТЕХНИКИ

Кафедра “Системы автоматизированного проектирования”

**Пояснительная записка к курсовой работе**

по курсу “Программирование сетевых приложений”

**«**Созданиеспортивного портала в Asp .Net Mvc**»**

**Исполнитель:** студент гр.10702314

Демух С.Г.

**Руководитель:** Кочуров В.А.

Минск 2016

Оглавление

[Цель 3](#_Toc469485641)

[Обзор предметной области: 3](#_Toc469485642)

[Преимущества ASP.NET MVC: 3](#_Toc469485643)

[База данных 5](#_Toc469485644)

[Порядок разработки проекта 6](#_Toc469485645)

[Создание базы данных 6](#_Toc469485646)

[Доступ к данным 8](#_Toc469485647)

[Страницы веб-приложения 8](#_Toc469485648)

[Главная страница веб-приложения 9](#_Toc469485649)

[Страница бокс: 11](#_Toc469485650)

[Страница Футбол: 12](#_Toc469485651)

[Страница Хоккей: 13](#_Toc469485652)

[Страница добавления статьи: 15](#_Toc469485653)

[Страница удаление статьи: 18](#_Toc469485654)

[Страница авторизация пользователя: 19](#_Toc469485655)

[Страница регистрации пользователя: 21](#_Toc469485656)

[Вывод: 24](#_Toc469485657)

# Цель

Целью работы является создания веб-приложения для просмотра, добавления, удаления спортивных новостей.

# Обзор предметной области:

ASP.NET MVC - это инфраструктура для разработки веб-приложений от Microsoft, которая сочетает в себе эффективность и аккуратность архитектуры "модель-представление-контроллер" (model-view-controller - MVC), новейшие идеи и приемы гибкой разработки, а также все лучшее из существующей платформы ASP.NET. Она представляет собой полномасштабную альтернативу традиционной технологии ASP.NET Web Forms, предоставляя преимущества для всех проектов веб-разработки, кроме самых тривиальных.

Важно различать архитектурный шаблон MVC и инфраструктуру ASP.NET MVC Framework. **Шаблон MVC** далеко не нов (его появление датируется 1978 г. и связано с проектом Smalltalk в Xerox PARC), но в наши дни он завоевал огромную популярность в качестве шаблона для веб-приложений по перечисленным ниже причинам:

* Взаимодействие пользователя с приложением MVC осуществляется в соответствии с естественным циклом: пользователь предпринимает действие, в ответ на которое приложение изменяет свою модель данных и доставляет обновленное представление пользователю. Затем цикл повторяется. Это хорошо укладывается в схему веб-приложений, предоставляемых в виде последовательностей запросов и ответов HTTP.
* Веб-приложения, нуждающиеся в комбинировании нескольких технологий (например, баз данных, HTML-разметки и исполняемого кода), обычно разделяются на ряд слоев или уровней. Полученные в результате шаблоны естественным образом вписываются в концепции MVC.

## Преимущества ASP.NET MVC:

ASP.NET MVC направлен на устранение многих недостатков технологии ASP.NET Web Forms, что, в свою очередь, заставляет разработчиков делать выбор в ее пользу при создании новых веб-приложений на платформе .NET.

1.Близость к протоколу

В то время как ASP.NET Web Forms пытается полностью скрыть не сохраняющую состояние сущность разметки, ASP.NET MVC не пытается ее скрыть. За счет того, что в нем используется MVC паттерн, а также за счет возможности преобразования отдельного HTTP запроса в вызов конкретного метода ASP.NET MVC, позволяет получить навыки разработки, которая более знакома разработчикам, обладающим теоретическими познаниями в создании веб-приложений. Модель также радикально упрощена – уход от сложных событий жизненного цикла страницы, используемых в Web Forms, а также минимальное количество абстрактных конструкций HTTP.

1.Концепция разделения

В то время как ASP.NET Web Forms устойчиво связывает пользовательский интерфейс с его выделенным кодом, ASP.NET MVC поддерживает конструкцию, в которой пользовательский интерфейс (представление) сохраняет свою изолированность от кода, управляющего им (контроллера). При хорошей реализации это означает, что разработчикам легче передвигаться по приложениям, а также такая возможность упрощает процесс сохранения приложения – то, что вы внесете изменения в контроллер, еще не означает, что вам придется изменять пользовательский интерфейс.

1.Тестируемость

С помощью отделения логики приложения от его пользовательского интерфейса ASP.NET MVC упрощает тестирование отдельных компонентов приложения. Классы контроллеров могут быть протестированы без тестирования реального пользовательского интерфейса. В отличие от Web Forms MVC контроллеры не имеют прямой зависимости от имеющего позорную славу, нетестируемого класса HttpContext, а вместо этого полагаются на абстракцию, что упрощает процесс написания автоматизированных тестов.

Концепция паттерна (шаблона) MVC (model - view - controller) предполагает разделение приложения на три компонента:

* *Model (Модель)* – домен, на основе которого строится ваше программное обеспечение. Если бы вы создавали блог, вашими моделями были бы пост и комментарий. Иногда термин "модель" может обозначать конкретную модель представления – отображение домена для конкретной цели демонстрации в пользовательском интерфейсе.
* *View (Представление)* – визуальное отображение модели в определенном контексте. Представление обычно является результирующей разметкой, которую фреймворк передает веб-браузеру, как например, HTML-разметка, представляющая пост блога.
* *Controller (Контроллер)* – координатор, который обеспечивает связь между представлением и моделью. Контроллер отвечает за обработку входных данных, оказывающих влияние на работу модели, и решает, какое действие должно выполняться, к примеру, передача представления или перенаправление на другую страницу. В продолжение примера публикации блога – контроллер может искать самые последние комментарии для публикации (модель) и передавать их в представление для показа.

# База данных

Традиционно используемые для WEB-разработок языки программирования позволяют реализовывать практически любые задачи. Но обрабатывать с их помощью большие объемы данных, имеющих к тому же сложную структуру, достаточно затруднительно. Разработка подобных программ требует все возрастающих затрат труда программистов, в геометрической прогрессии растет объем программного кода и количество ошибок, снижается надежность программного обеспечения.

В такой ситуации на помощь программисту приходят базы данных. Согласно классическому определению, база данных - это упорядоченная совокупность информации, хранящейся в виде множеств, каждое из которых содержит записи унифицированного вида. Системы управления базами данных (СУБД) предоставляют программисту мощнейший инструментарий для создания, обновления и обработки больших объемов информации, имеющей сложную структуру. Выбор СУБД является непростой задачей для разработчиков баз данных.

Для данного проекта в качестве СУБД можно выбрать Microsoft Sql Server 2012. Эта система является довольно распространенной, надёжна и проста в администрировании.

В выпуске Visual Studio 2008 в ADO.NET представлена новая архитектура Entity Framework. **Entity Framework** представляет специальную объектно-ориентированную технологию на базе фреймворка .NET для работы с данными. Если традиционные средства ADO.NET позволяют создавать подключения, команды и прочие объекты для взаимодействия с базами данных, то Entity Framework представляет собой более высокий уровень абстракции, который позволяет абстрагироваться от самой базы данных и работать с данными независимо от типа хранилища. Если на физическом уровне мы оперируем таблицами, индексами, первичными и внешними ключами, но на концептуальном уровне, который нам предлагает Entity Framework, мы уже работает с объектами.

# Порядок разработки проекта

Работа над данным проектом начинается с создания базы данных. Прежде всего, следует определиться с её структурой и создать необходимые таблицы.

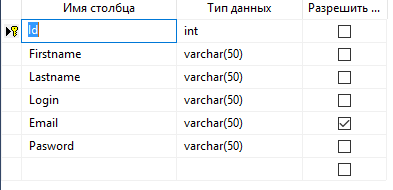
Далее разрабатывается непосредственно web-приложение. Для этого была использована Microsoft Visual Studio 2015.

## Создание базы данных

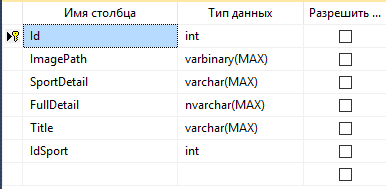
Для создания базы данных я использовал Microsoft Sql Server 2012, и там я создавал базу данных.

База данных представляет собой следующий набор страниц:

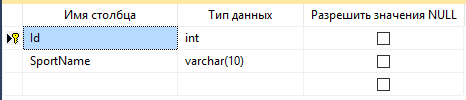
* 1. User – таблица, в которой находится полная информация о пользователях зарегистрированных на нашем сайте;



* 1. ContentTable – таблица, в которой хранятся весь перечень новостей, которые позже будут располагаться на нашем сайте;



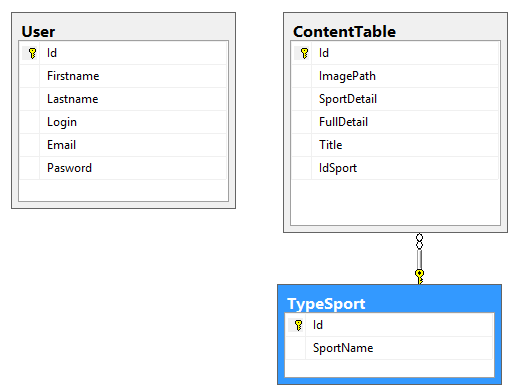
* 1. TypeSport– таблица, в которой хранятся весь перечень спортивных дисциплин;



Между таблицами устанавливаются следующие связи:

* таблица ContentTable связывается с таблицей TypeSport по полям IdSport–Id в отношении "один ко многим";

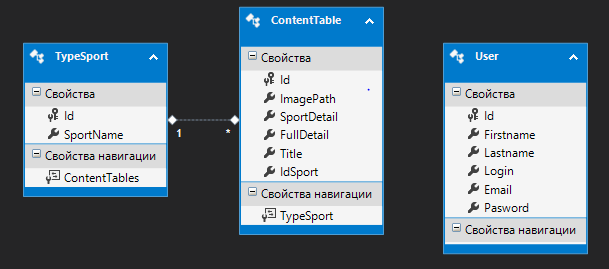
Диаграмма базы данных:



## Доступ к данным

После создания таблиц создается модель классов базы данных с помощью дополнения Entity Framework. Пользуясь моделью можно взаимодействовать с самой базой данных, абстрагируясь от SQL запросов.

Для создания модели данных нужно добавить папку Models и нажав на нее правой кнопкой мыши выбрать пункт Добавить -> Создать элемент. В диалоговом окне выбрать Модель ADO.NET EMD и назвать ее EFModel. Далее создать подключение к базе данных. Далее выбрать версию Entity Framework. Далее выбрать таблицы базы данных.



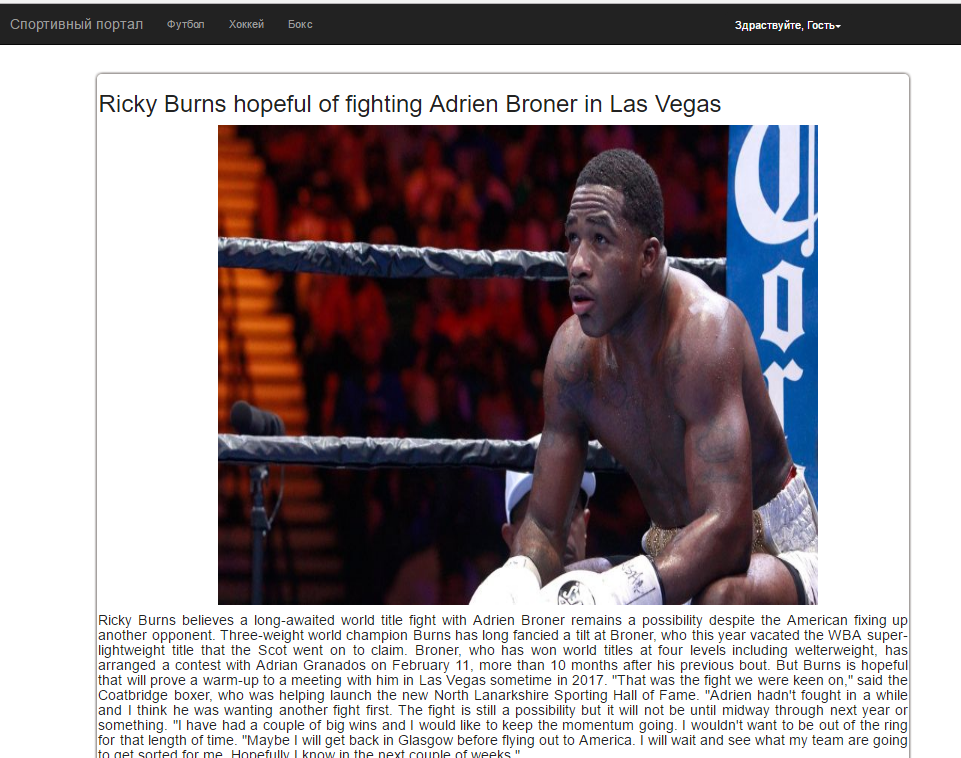
# Страницы веб-приложения

При создании страниц был использован подход блочной вёрстки. «Блочная вёрстка» или вёрстка с помощью слоёв заключается в конструктивном использовании тегов <div> и стилей.

При блочной вёрстке, конечно же, используются таблицы, но только в тех случаях, когда они нужны, например, для наглядного отображения чисел и других табличных данных. Вариант, когда от таблиц предлагается отказаться вообще, является нецелесообразным и, более того, вредным.

## Главная страница веб-приложения

Страница приложения, на которой показан список новостей. Все новости мы берем из базы данных.



При загрузке первоначальной страницы мы не авторизированный пользователь, и видим все новости которые берутся с базы данных. Для того чтобы авторизироваться, или зарегистрироваться нужно зайти на необходиимые страницы.

Тело главной страницы выглядит следующим образом:

@model IEnumerable<ClassLibraryEF.ContentTable>

@{

ViewBag.Title = "Index";

}

<link href="@Url.Content("~/Content/Content.css")" rel="stylesheet" type="text/css" />

<div class="col-md-11 col-md-offset-1">

<div class="inner-content">

@foreach (var item in Model)

{

<h2>@Html.DisplayFor(modelItem => item.Title)</h2>

@Html.Raw("<img style='width:900px; height:600px;' src=\"data:image/jpeg;base64,"

+ Convert.ToBase64String(item.ImagePath) + "\" />")

<br />

<h4>@Html.DisplayFor(modelItem => item.FullDetail)

@if (Session["Role"] != null && Session["Role"].ToString() == "admin")

{

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

}

</h4>

<br />

}

</div>

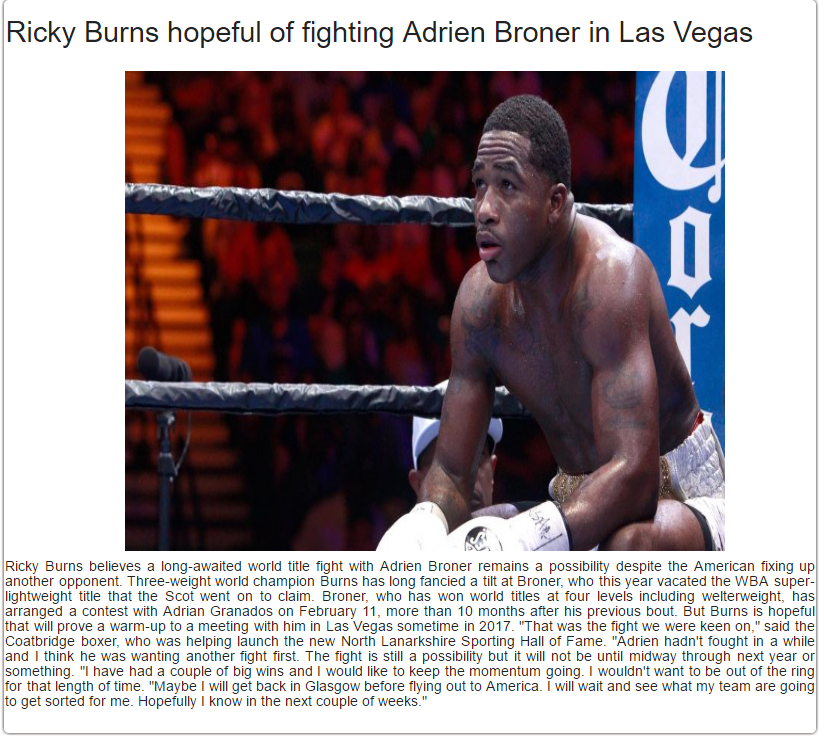
</div>

Также добавим в наш проект таблицу стилей CSS, чтобы украсить оформления web-приложения. Для этого создадим папку Content и в этой папке создадим документ Content.css.

Так же сделаны страницы отдельно для каждого спортивной игры, которые динамичны:

На странице **бокс** показывается все новости связанные с боксом, на странице **футбол** показывается все новости связанные с футболом, на странице **хоккей** показывается все новости связанные с хоккеем.

## Страница бокс:



Тело страницы бокс выглядит следующим образом:

@model IEnumerable<ClassLibraryEF.ContentTable>

@{

ViewBag.Title = "Box";

}

<link href="@Url.Content("~/Content/Content.css")" rel="stylesheet" type="text/css" />

<div class="col-md-11 col-md-offset-1">

<div class="inner-content">

@foreach (var item in Model)

{

<h1>@Html.DisplayFor(modelItem => item.Title)</h1>

<br />

@Html.Raw("<img style='width:900px; height:600px;' src=\"data:image/jpeg;base64,"

+ Convert.ToBase64String(item.ImagePath) + "\" />")

<br />

<h4>@Html.DisplayFor(modelItem => item.FullDetail)</h4>

<br />

}

</div>

</div>

## Страница Футбол:



Тело страницы футбол выглядит следующим образом:

@model IEnumerable<ClassLibraryEF.ContentTable>

@{

ViewBag.Title = "Box";

}

<link href="@Url.Content("~/Content/Content.css")" rel="stylesheet" type="text/css" />

<div class="col-md-11 col-md-offset-1">

<div class="inner-content">

@foreach (var item in Model)

{

<h1>@Html.DisplayFor(modelItem => item.Title)</h1>

<br />

@Html.Raw("<img style='width:900px; height:600px;' src=\"data:image/jpeg;base64,"

+ Convert.ToBase64String(item.ImagePath) + "\" />")

<br />

<h4>@Html.DisplayFor(modelItem => item.FullDetail)</h4>

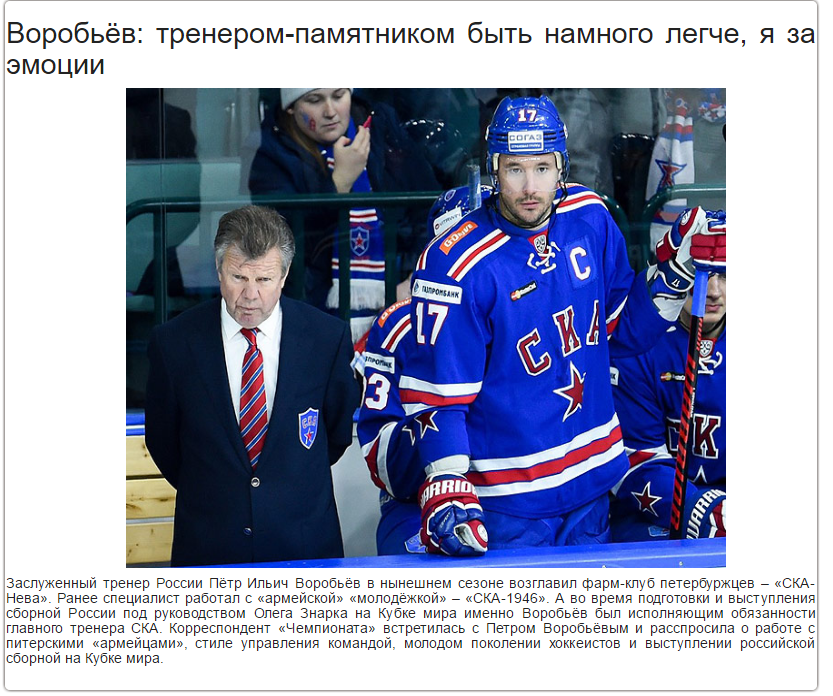
<br />

}

</div>

</div>

## Страница Хоккей:



Тело страницы Хоккей выглядит следующим образом:

@model IEnumerable<ClassLibraryEF.ContentTable>

@{

ViewBag.Title = "Box";

}

<link href="@Url.Content("~/Content/Content.css")" rel="stylesheet" type="text/css" />

<div class="col-md-11 col-md-offset-1">

<div class="inner-content">

@foreach (var item in Model)

{

<h1>@Html.DisplayFor(modelItem => item.Title)</h1>

<br />

@Html.Raw("<img style='width:900px; height:600px;' src=\"data:image/jpeg;base64,"

+ Convert.ToBase64String(item.ImagePath) + "\" />")

<br />

<h4>@Html.DisplayFor(modelItem => item.FullDetail)</h4>

<br />

}

</div>

</div>

Так же мы создаем контроллере методы, в котором будем обрабатывать все наши события связанные с нашими новостями:

public class ContentController : Controller

{

NewsDBProvider dbprovider = new NewsDBProvider();

ContetBuilder cbuilder = new ContetBuilder();

public ActionResult Index()

{

return View(dbprovider.Index());

}

В методе Index мы выводим все наши новости на главную страницу Index.cshtml

public ActionResult Hockey()

{

return View(dbprovider.Hockey());

}

В методе Hockey мы выводим все наши новости на главную страницу Hockey.cshtml

public ActionResult Football()

{

return View(dbprovider.Football());

}

В методе Football мы выводим все наши новости на главную страницу Football.cshtml

public ActionResult Box()

{

return View(dbprovider.Box());

}

В методе Box мы выводим все наши новости на главную страницу Box.cshtml

public ActionResult Create()

{

ViewBag.IdSport = new SelectList(db.TypeSports, "Id", "SportName");

return View();

}

[HttpPost]

public ActionResult Create(ContentViewModel contentTable, HttpPostedFileBase uploadImage)

{

var content = dbprovider.Add(cbuilder.CBuilder(contentTable), uploadImage);

if (content != null)

{

return Redirect("/Content/Index");

}

ModelState.AddModelError("", "login and password is incorect");

return View(contentTable);

}

В методе Create мы добавляем новость.

public ActionResult Delete(int? id)

{

if (id == null)

{

return new HttpStatusCodeResult(HttpStatusCode.BadRequest);

}

return View(dbprovider.GetDelete(id));

}

[HttpPost, ActionName("Delete")]

[ValidateAntiForgeryToken]

public ActionResult DeleteConfirmed(int id)

{

dbprovider.DeleteArticle(id);

return RedirectToAction("Index");

}

}

}

В методе Delete мы удаляем новость.

public class ContentViewModel

{

[Required(ErrorMessage = "Введите тип спорта")]

public byte[] ImagePath { get; set; }

[Required(ErrorMessage = "Введите детали о вашей статье")]

public string SportDetail { get; set; }

[Required(ErrorMessage = "Введите вашу статью")]

public string FullDetail { get; set; }

[Required(ErrorMessage = "Введите заглавие статьи")]

public string Title { get; set; }

public int IdSport { get; set; }

public TypeSportViewModel TypeSport { get; set; }

}

public ContentTable CBuilder(ContentViewModel Content)

{

return new ContentTable

{

ImagePath = Content.ImagePath,

FullDetail = Content.FullDetail,

SportDetail = Content.SportDetail,

Title = Content.Title,

IdSport = Content.IdSport,

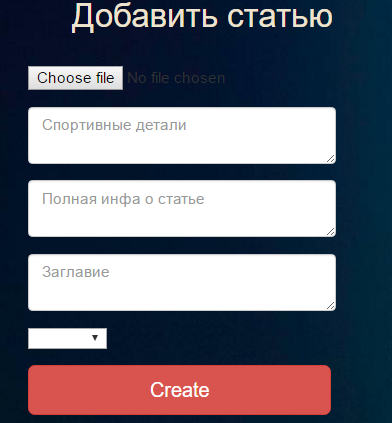
};

}

## Страница добавления статьи:

На данной странице у нас будет происходить добавления статьи для дальнейшего отображения на главной странице. Всю информацию о статье мы будем хранить в базе данных.

Страница добавления статьи выглядит следующим образом:



Тело страницы добавления статьи:

@model Kursach.Models.ContentViewModel

@{

ViewBag.Title = "Добавить статью";

}

<link href="@Url.Content("~/style.css")" rel="stylesheet" type="text/css" />

@using (Html.BeginForm("Create", "Content", FormMethod.Post, new { enctype = "multipart/form-data" }))

{

@Html.AntiForgeryToken()

<div class="form-horizontal">

@Html.ValidationSummary(true)

<div class="form-group">

<div class="col-md-8 col-md-offset-4">

<h2>@ViewBag.Title</h2>

<br />

<input type="file" name="uploadImage" />

</div>

</div>

<div class="form-group">

<div class="col-md-8 col-md-offset-4">

@Html.TextAreaFor(model => model.SportDetail, new { @class = "form-control", @placeholder = "Спортивные детали" })

@Html.ValidationMessageFor(model => model.SportDetail)

</div>

</div>

<div class="form-group">

<div class="col-md-8 col-md-offset-4">

@Html.TextAreaFor(model => model.FullDetail, new { @class = "form-control", @placeholder = "Полная инфа о статье" })

@Html.ValidationMessageFor(model => model.FullDetail)

</div>

</div>

<div class="form-group">

<div class="col-md-8 col-md-offset-4">

@Html.TextAreaFor(model => model.Title, new { @class = "form-control", @placeholder = "Заглавие" })

@Html.ValidationMessageFor(model => model.Title)

</div>

</div>

<div class="form-group">

<div class="col-md-8 col-md-offset-4">

@Html.DropDownList("IdSport", String.Empty)

@Html.ValidationMessageFor(model => model.IdSport)

@Html.ValidationSummary(true)

</div>

</div>

<div class="form-group">

<div class="col-md-offset-2 col-md-10">

<input type="submit" value="Create" class="btn btn-danger btn-lg" style="padding-right: 110px; padding-left: 110px; margin-left:195px;" />

</div>

</div>

</div>

}

На этой странице есть поля для добавления картинки, введения заглавия статьи, полной информации о статье, выбора к какому виду спортивных игр она относится. Все эти данные посылаются в контроллер. В методе Create нашего контроллера вызывается метод Add объекта dbprovider и в этом методе добавляется наши введенные данные в базу данных.

NewsDBProvider dbprovider = new NewsDBProvider();

ContetBuilder cbuilder = new ContetBuilder();

public ActionResult Create(ContentViewModel contentTable, HttpPostedFileBase uploadImage)

{

var content = dbprovider.Add(cbuilder.CBuilder(contentTable), uploadImage);

if (content != null)

{

return Redirect("/Content/Index");

}

ModelState.AddModelError("", "login and password is incorect");

return View(contentTable);

}

public ContentTable Add(ContentTable contentTable, HttpPostedFileBase uploadImage)

{

if (uploadImage != null)

{

byte[] imageData = null;

using (var binaryReader = new BinaryReader(uploadImage.InputStream))

{

imageData = binaryReader.ReadBytes(uploadImage.ContentLength);

}

contentTable.ImagePath = imageData;

db.ContentTables.Add(contentTable);

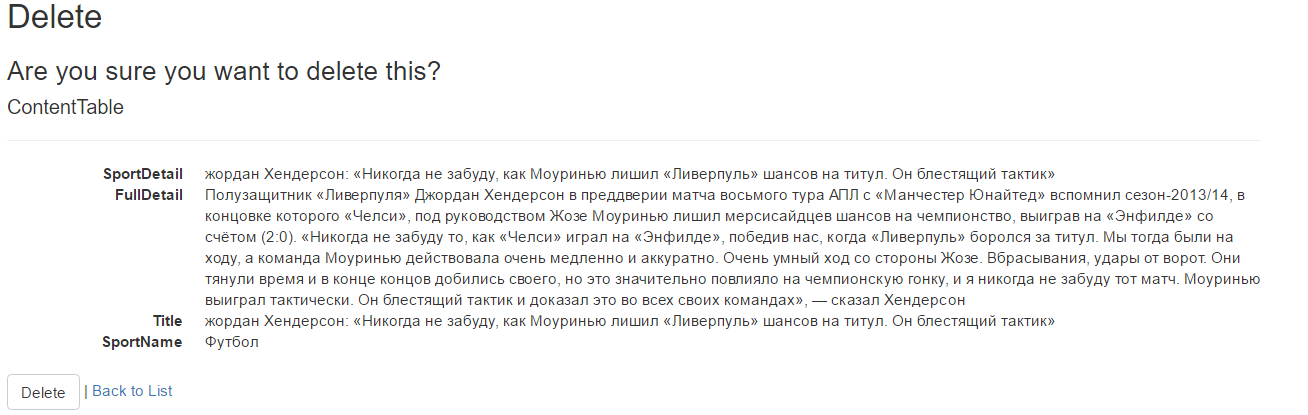
db.SaveChanges();

}

return contentTable;

}

## Страница удаление статьи:



Тело страницы добавления статьи:

@model ClassLibraryEF.ContentTable

@{

ViewBag.Title = "Delete";

}

<h2>Delete</h2>

<h3>Are you sure you want to delete this?</h3>

<div>

<h4>ContentTable</h4>

<hr />

<dl class="dl-horizontal">

<dt>

@Html.DisplayNameFor(model => model.SportDetail)

</dt>

<dd>

@Html.DisplayFor(model => model.SportDetail)

</dd>

<dt>

@Html.DisplayNameFor(model => model.FullDetail)

</dt>

<dd>

@Html.DisplayFor(model => model.FullDetail)

</dd>

<dt>

@Html.DisplayNameFor(model => model.Title)

</dt>

<dd>

@Html.DisplayFor(model => model.Title)

</dd>

<dt>

@Html.DisplayNameFor(model => model.TypeSport.SportName)

</dt>

<dd>

@Html.DisplayFor(model => model.TypeSport.SportName)

</dd>

</dl>

@using (Html.BeginForm()) {

@Html.AntiForgeryToken()

<div class="form-actions no-color">

<input type="submit" value="Delete" class="btn btn-default" /> |

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

</div>

}

</div>

На этой странице мы удаляем статью. В нашем контроллере ContentController есть метод DeleteConfirmed, который вызывает метод DeleteArticle. В этот метод передаём Id нашей статьи, и в этом методе удаляем эту статью, и сохраняем изменения в базу данных.

[HttpPost, ActionName("Delete")]

[ValidateAntiForgeryToken]

public ActionResult DeleteConfirmed(int id)

{

dbprovider.DeleteArticle(id);

return RedirectToAction("Index");

}

public void DeleteArticle(int id)

{

ContentTable contentTable = db.ContentTables.Find(id);

db.ContentTables.Remove(contentTable);

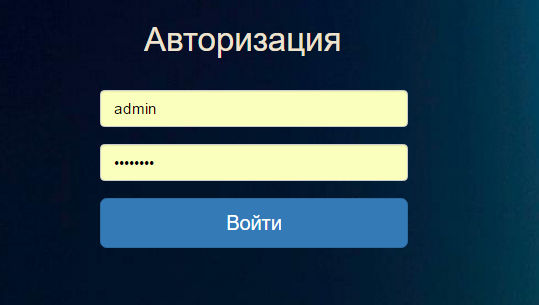
db.SaveChanges();

}

## Страница авторизация пользователя:

На данной странице у нас будет происходить авторизация уже зарегистрированного, на нашем сайте пользователя. Введенную нами информацию мы будем сверять со значениями из базы данных.

Тело страницы авторизации:



@model Kursach.Models.LoginViewModel

@{

ViewBag.Title = "Авторизация";

Session["Role"] = null;

}

<link href="@Url.Content("~/style.css")" rel="stylesheet" type="text/css" />

@using (Html.BeginForm())

{

@Html.AntiForgeryToken()

<div class="form-horizontal">

<div class="form-group">

<div class="col-md-offset-4 col-md-8">

<h2>@ViewBag.Title</h2>

<br />

</div>

<div class="col-md-offset-4 col-md-8">

@Html.TextBoxFor(model => model.Login, new { @class = "form-control", @placeholder = "Логин" })

@Html.ValidationMessageFor(model => model.Login)

</div>

</div>

<div class="form-group">

<div class="col-md-offset-4 col-md-8">

@Html.PasswordFor(model => model.Pasword, new { @class = "form-control", @placeholder = "Пороль" })

@Html.ValidationMessageFor(model => model.Pasword)

@Html.ValidationSummary(true)

</div>

</div>

<div class="form-group">

<div class="col-md-offset-4 col-md-8">

<input type="submit" value="Войти" class="btn btn-primary btn-block btn-lg" />

</div>

</div>

</div>

}

В Методе Login нашего контроллера AccountController метод проверяет пользователя в базе данных. Если пользователь с таким логином и паролем существует, то происходит вход в аккаунт.

UserRepository userRepository = new UserRepository();

UserBuilder userBuilder = new UserBuilder();

[HttpPost]

public ActionResult Login(LoginViewModel loginViewModel)

{

var user = userRepository.Find(loginViewModel.Login, loginViewModel.Pasword);

if (user != null) {

Session["Id"] = user.Id.ToString();

Session["Name"] = user.Firstname.ToString();

Session["Role"] = user.Login.ToString();

return Redirect("/Content/Index");

}

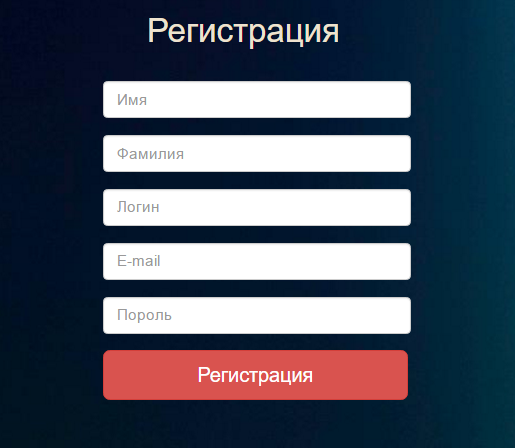
return View();

}

## Страница регистрации пользователя:

На данной странице у нас будет происходить регистрация пользователя для дальнейшего посещения нашего сайта. Всю информацию о пользователе мы будем хранить в базе данных для последующего входа на сайт.

Страница регистрации выглядит следующим образом:



Тело страницы регистрация:

@model Kursach.Models.UserViewModel

@{

ViewBag.Title = "Регистрация";

}

<link href="@Url.Content("~/style.css")" rel="stylesheet" type="text/css" />

@using (Html.BeginForm())

{

@Html.AntiForgeryToken()

<div class="form-horizontal">

@Html.ValidationSummary(true)

<div class="form-group">

<div class="col-md-offset-4 col-md-8">

<h2>@ViewBag.Title</h2>

<br />

@Html.TextBoxFor(model => model.Firstname, new { @class = "form-control", @placeholder = "Имя" })

@Html.ValidationMessageFor(model => model.Firstname)

</div>

</div>

<div class="form-group">

<div class="col-md-offset-4 col-md-8">

@Html.TextBoxFor(model => model.Lastname, new { @class = "form-control", @placeholder = "Фамилия" })

@Html.ValidationMessageFor(model => model.Lastname)

</div>

</div>

<div class="form-group">

<div class="col-md-offset-4 col-md-8">

@Html.TextBoxFor(model => model.Login, new { @class = "form-control", @placeholder = "Логин" })

@Html.ValidationMessageFor(model => model.Login)

</div>

</div>

<div class="form-group">

<div class="col-md-offset-4 col-md-8">

@Html.TextBoxFor(model => model.Email, new { @class = "form-control", @placeholder = "E-mail" })

@Html.ValidationMessageFor(model => model.Email)

</div>

</div>

<div class="form-group">

<div class="col-md-offset-4 col-md-8">

@Html.PasswordFor(model => model.Pasword, new { @class = "form-control", @placeholder = "Пороль" })

@Html.ValidationMessageFor(model => model.Pasword)

</div>

</div>

<div class="form-group">

<div class="col-md-offset-4 col-md-8">

<input type="submit" value="Регистрация" class="btn btn-danger btn-lg" style="padding-right: 85px; padding-left: 85px;" />

</div>

</div>

</div>

}

На этой странице есть поля для введения Имени, фамилии, логина, E-Mail и пароля. Все эти данные посылаются в метод Create контроллера ContentController. Контроллер записывает данные в таблицу Users и сохраняет.

UserRepository userRepository = new UserRepository();

UserBuilder userBuilder = new UserBuilder();

[HttpPost]

public ActionResult Create(UserViewModel userViewModel)

{

if (ModelState.IsValid)

{

var user = userRepository.Add(userBuilder.Build(userViewModel));

if (user != null) {

Session["Id"] = user.Id.ToString();

Session["Name"] = user.Firstname.ToString();

Session["Role"] = user.Login.ToString();

return RedirectToAction("Index", "Content");

}

return View(userViewModel);

}

# Вывод:

В ходе данного курсового проекта было разработано веб приложение, реализующие следующие задачи:

* Авторизация пользователя
* Добавление нового пользователя.
* Добавления статьи
* Удаления статьи

В результате разработки проекта были достигнуты поставленные цели и получен ожидаемый результат.