ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ»
(Н И У « Б е л Г У »)

ИНСТИТУТ ИНЖЕНЕРНЫХ ТЕХНОЛОГИЙ И ЕСТЕСТВЕННЫХ НАУК

КАФЕДРА МАТЕМАТИЧЕСКОГО И ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ИНФОРМАЦИОННЫХ СИСТЕМ

АВТОМАТИЗИРОВАННАЯ СИСТЕМА ПОДДЕРЖКИ ОРГАНИЗАЦИОННОЙ ДЕЯТЕЛЬНОСТИ КЛУБА ЕДИНОБОРСТВ

Выпускная квалификационная работа
обучающегося по направлению подготовки 02.03.02 Фундаментальная информатика и информационные технологии
заочной формы обучения, группы 07001360
Селина Андрея Геннадьевича

Научный руководитель
доцент
Румбешт В.В.

БЕЛГОРОД 2017

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

С момента появления человечества, развитие технологий являлось одним из ключевых аспектов, влияющих на изменение образа жизни людей, на то, как они добывают свою пищу, обустраивают быт, постигают знания, взаимодействуют друг с другом и многое другое. На сегодняшний день, научное развитие населения Земли достигло такого уровня, что благодаря постоянно развивающимся технологиям, людям стала доступна система практически мгновенного взаимодействия друг с другом и получения практическим любой информации несколькими щелчками мыши. Другими словами - человечество вступило в «информационную эпоху». Исходя из этого, следует осознать что, каждое дело, которым занимается человек, или организация – требует качественного информационного сопровождения, для получения наилучшего результата в своей отрасли.

Целью данной дипломной работы является информационное сопровождение такой важной части в жизни современного общества, как развитие молодежного спорта. В ходе данной дипломной работы будет разработана автоматизированная система поддержки организационной деятельности клуба единоборств «Атом». Основным задачами данной системы будут являться возможность автоматизированной и удаленной работы со спортсменами, занимающимися в клубе и их родственниками, с управление данных о них, а главное с размещением информации о деятельности клуба во всемирной системе объединённых компьютерных сетей для хранения и передачи информации под названием «Интернет».

В результате выполнения поставленной задачи будет разработано веб-приложение, способное реализовать цели преследуемые клубом единоборств и способное отвечать на множество вызовов, возникающих перед организациями в современную информационную эру.

Для того чтобы достигнуть перечисленных выше задач, следует выделить и указать ряд ключевых компонентов, который включает в себя:

1) анализ деятельности организации, требующей информационного сопровождения;

2) выявление ключевых направлений деятельности организации, требующих автоматизации, либо публикации в сети;

3) обзор и подробный анализ систем, выполняющих схожие задачи;

4) постановка требований к разрабатываемой системе;

5) проектирование информационной системы;

6) реализация спроектированной системы;

7) разработка методики испытаний и ее реализация.

Данная выпускная квалификационная работа состоит из введения, трех глав, заключения списка литературы и приложения.

Введение содержит общие сведения о работе, ее актуальность, цели, задачи и способы их достижения.

Первая глава содержит описание организации, анализ ее деятельности и постановку задач, требующих решения.

Вторая глава посвящена проектированию и реализации информационной системы

В ходе выполнения третьей главы были разработана программа и методика испытаний, а также проведены соответствующие ей испытания.

В заключении подводится итог проведенной работе.

Дипломная работа состоит из 73 страниц, 52 рисунков и приложения включающего 27 страниц.

Глава 1. ПОСТАНОВКА ЗАДАЧИ

1.1 Особенности организации

Клуб единоборств является некоммерческой организацией. Основной деятельностью клуба является пропаганда и развитие физической культуры и спорта среди молодежи, формирование физически здорового и способного защитить себя и окружающих человека. Помимо основной цели клуб так же осуществляет вклад в развитие социальных, образовательных, духовных потребностей граждан и в целом его деятельность направленна на достижение общественных благ. В клубе работает ряд профессиональных специалистов, имеющих большой опыт работы. У каждого тренера могут быть одна или несколько своих групп в зависимости от количества спортсменов и уровня их подготовки. Критериями, по которым спортсменов делят по группам, являются их возраст, уровень мастерства и вид единоборства.

Деятельность клуба единоборств обширна и разнообразна. Клуб регулярно принимает участие во множестве различных массовых мероприятиях в основном спортивно-состязательного и спортивно-развлекательного характера. Наиболее частыми из них являются мероприятия соревновательного толка. К таким мероприятиям можно отнести: соревнования, турниры, кубки, первенства, чемпионаты которые могут быть, как приурочены к памяти какого-либо известного человека, или проводится в честь знаменательной даты или праздника так и быть самостоятельным мероприятием. Основными видами единоборств, которым обучают в клубе: являются дзюдо и самбо. Однако, несмотря на ориентацию группы на изучение определенного единоборства, спортсмены могут принимать участие не только в соревнования по изучаемому единоборству, но и участвовать в соревнованиях по смежным видам единоборств, а так же и

в спортивных мероприятиях иного вида, в случае наличия надлежащей подготовки.

Для тех, кто занимается в клубе, устанавливаются квалификационные требования, в соответствии с которыми осуществляется градация спортсменов по уровню их подготовки, что отражает как, то насколько ученик вкладывается в изучение единоборства, так и то насколько он готов к участию в том или ином мероприятии. Данные требования устанавливаются в соответствии с нормами «Детской Юношеской Спортивной Школы». За выполнение квалификационных требований спортсмены награждаются повышением их разряда. Всего в дзюдо есть 16 квалификационных степеней из них 6 ученических – КЮ и 10. Шесть рангов КЮ это ранги начинающих, и считается, что их обладатели еще не имеют права называться настоящими учениками. Однако за каждый ранг спортсмен получает право носить пояс определенного цвета и ему присуждается соответствующий разряд по единой всероссийской спортивной классификации.

Для того чтобы мотивировать спортсменов на более усердные тренировки и привить им заинтересованность в совершенствовании техники борьбы клуб использует различные способы поощрения наиболее активных и целеустремленных спортсменов. К таким поощрениям относятся награждения почетные грамоты и другие памятные вознаграждения самым результативным спортсменам, переходящие кубки за самое большое количество побед на днях борьбы и тому подобные мероприятия.

Так же для развития навыков летом, во время каникул в учебных заведения, клуб регулярно осуществляет вывоз спортсменов в летние спортивные лагеря и санатории. Там под руководством тренеров занимающиеся совмещают активный отдых и регулярные тренировки, с целью совершенствования навыков владения техникой единоборства и прививания молодому поколению здорового образа жизни.

Еще одним выездным мероприятием для членов клуба является участие в спортивных сборах. Спортивные сборы отличаются тем что в них могут

принимать участие лишь более подготовленные участники клуба, со спортсменами из других клубов и секций. На сборах часто присутствуют заслуженные профессионалы, которые проводят мастер-классы для участвующих спортсменов.

Помимо работы непосредственно с учениками клуб так же проводит агитационные, информационные и организационные мероприятия с их родителями. Клуб заинтересован в проведении родительских собраний. Так же, кроме собраний, клуб нуждается в поддержке дистанционной связи с родителями. Работники клуба проводят с родителями работу по информированию их об успехах или проблемах детей, о том как лучше мотивировать и воздействовать на занимающихся, чтобы спортсмены могли показывать более высокие результаты. Так же клуб вынужден регулярно информировать родителей о кратковременных или долгосрочных изменениях в расписании занятий, о требованиях к их детям, для участия в том или ином мероприятии, о необходимости в приобретении амуниции или личного спортивного инвентаря. Иногда клуб вынужден призывать родителей оказать информационную, юридическую, финансовую, или какую либо иную помощь, для решения проблем с которыми может столкнуться клуб в процессе его деятельности.

Хозяйственная деятельность так же входит в обязанности клуба. Клуб обязан рационально распоряжаться всеми имеющимися средствами, таким образом, чтобы наиболее качественно выполнять поставленные перед ним задачи. В обязанности клуба входят мероприятия по содержанию зала в подобающем состоянии: покраска, штукатурка и прочее. Так же в ведении клуба находится спортивный инвентарь - уход за имеющимся инвентарем, ремонт либо замена инвентаря пришедшего в негодность, а так же его приобретение в случае иных причин.

Таким образом, деятельность клуба имеет широкий круг задач, которые нуждаются в автоматизации и информационном сопровождении.

Исходя из вышеперечисленного, следует перечень аспектов деятельности клуба подлежащих автоматизации:

1) построение отчетности о текущей деятельности клуба. Отчетность о деятельности должна быть представлена в виде блога, который содержит в себе множество статей добавляемых после участия в мероприятиях, или в случае необходимости информационного сопровождения какого – либо другого важного момента жизни клуба. Структура блога должна представлять собой страницу с перечнем имеющихся отчетов о мероприятиях и страницы, которая будет открываться при выборе какого-то конкретного мероприятия, и отображать более подробную информацию о нем;

2) анонсирование грядущих мероприятий и предоставление информации о месте и времени проведения мероприятия, а так же информирование о подробностях. Информация должна группироваться по дням, или в виде списка событий, отсортированных по дате и времени проведения. В случае выбора конкретного мероприятия должна открываться новая страница с подробной информацией о нем. Страница с анонсом выбранного мероприятия должна содержать дату проведения мероприятия, время начала мероприятия. А также иметь возможность указать время окончания мероприятия, требованиями к участникам мероприятия, требования к зрителям мероприятия, электронную копию положения о мероприятии, фото места проведения мероприятия, изображение с картой указывающей маршрут к месту проведения мероприятия и прочую текстовую информацию, или изображения, если это нужно;

3) предоставление информации необходимой для изучения единоборств. К такой информации могут относиться: правила и требования для проведения соревнований, правила ведения схватки участниками соревнований, аттестационные требования для повышения спортивного разряда, информация о технике выполнения приемов стоя и лежа, о технике ведения схватки, о технике выполнения приемов для повышения силовых

качеств в тренажерном зале и на открытом воздухе и другой подобной информации. Данная информация должна быть организована в виде страницы со списком статей, из которого можно будет выбрать интересующую информацию для подробного изучения;

4) возможность хранить, редактировать, и иметь быстрый доступ по средствам сети интернет к данным о спортсменах, занимающихся в клубе, и их родителях, или родственников.

Все перечисленные пункты требуется реализовать таким образом, что непосредственно тренер, или занимающиеся, либо специально нанятое лицо могли добавлять на сайт подобную информацию, не обладая глубокими знаниями в сфере программирования.

1.2 Обзор и анализ существующих систем

В век цифровых технологий и проникновения интернета во все сферы человеческой деятельности невозможно успешно вести какую-либо работу, ориентированную на взаимодействие с людьми, не имея представительства в сети. Несмотря на это на данный момент лишь немногочисленное количество клубов единоборств имеют собственные сайты. Однако все эти сайты в той или иной мере не отвечают тем требованиям, которые необходимо выполнять сайту, чтобы подобающим образом упростить ряд аспектов работы клуба, требующих автоматизации. Чтобы убедиться в этом, можно рассмотреть несколько таких сайтов и оценить то, как они справляются с поставленными задачами.

Система 1. Сайт московского клуба дзюдо «Мастер». Первой рассмотренной системой будет сайт московского клуба дзюдо «Мастер» расположенного по интернет адресу: http://www.master.judo-zao.ru/news.shtml. На рис. 1.1 отображена главная страница сайта. Дизайн страницы выглядит явно устаревшим.

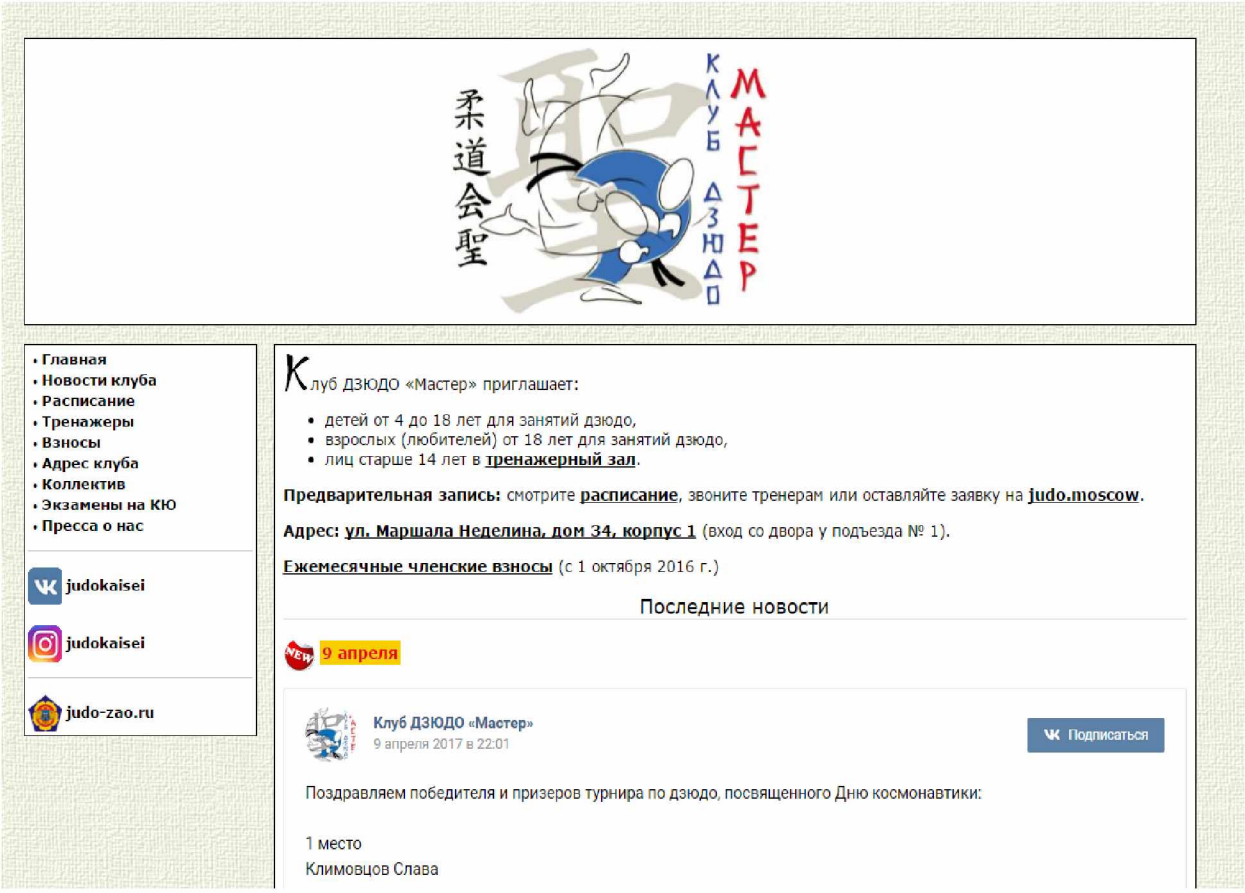

Рис. 1.1. Главная страница

Рассмотрим то, как сайт реализует приведенные выше аспекты требующие автоматизации:

1) построение отчетности о текущей деятельности клуба. На рис. 1.2 изображена страница новости клуба, на которой реализовано предоставление отчетности о деятельности клуба. Новостной раздел представлен в виде блога с перечнем новостей, однако он сделан так, что новость отображается либо полностью, что затрудняет пользователю искать интересующую его информацию, либо информация о новости минимальная и предлагается перейти по ссылке на группу в социальной сети, что так же является минусом, так как многие пользователи не пользуются социальными сетями, а так же потому что работа системы становится зависима от работы стороннего ресурса;

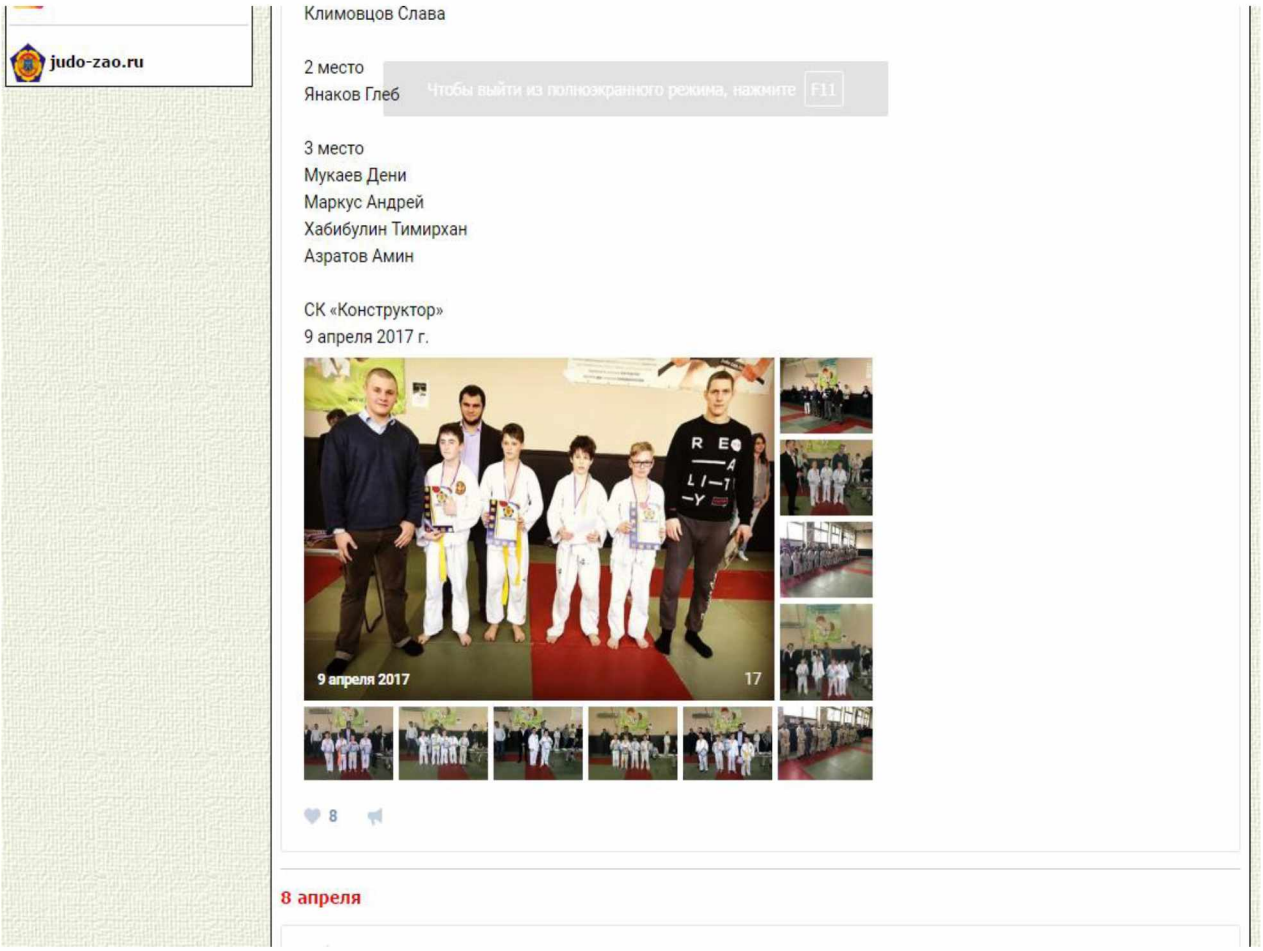

Рис. 1.2. Страница «Новости клуба»

2) какого-либо способа анонсирование грядущих мероприятий и предоставление информации о месте и времени проведения мероприятия, а так же информирование о подробностях на сайте не присутствует. Единственной страницей, выполняющей малую часть необходимого функционала, является страница с расписанием проведения занятий, увидеть которую можно на рис. 1.3. Однако даже тот функционал, который имеется, реализован достаточно неудобно. Для того чтобы посмотреть расписание предлагается скачать pdf файл и посмотреть его за пределами сайта, что является неудобным и не всегда осуществимым, к тому же не каждый пользователь умеет это делать. Использование подобного способа информирования явный минус;

Рис. 1.3. Страница «расписание

3) предоставление информации необходимой для изучения единоборств так же реализовано слабо. Страница сайта, которая частично реализует этот аспект, показана на рис. 1.4 – это страница «Экзамены на КЮ». Перейдя на страницу можно обнаружить крайне мало информации о единоборстве, к тоже же она так же требует от пользователя скачивать архив или pdf файл для последующей работы с ними на компьютере пользователя. К тому же на момент просмотра сайта этот раздел оказался незавершенным.

Рис. 1.4. Страница «Экзамены на КЮ»

Рассмотрев данную систему и оценив все возможности, можно прийти к выводу, что данный сайт слабо организует все аспекты деятельности клуба подлежащего к автоматизации.

Система 2. Сайт клуба дзюдо «Турбостроитель». Следующий рассмотренный сайт располагается по адресу: http://turbojudo.spb.ru/. Как видно из рис. 1.5 дизайн главной страницы сайта выглядит современно.

Рис.1.5. Главная страница сайта

Однако для того чтобы оценить насколько этот сайт отвечает требованиям, которые необходимо выполнять для корректной автоматизации деятельности клуба, следует рассмотреть функционал сайта и оценить то на сколько реализованы требуемые аспекты:

1) построение отчетности о текущей деятельности клуба. Сразу после перехода на сайт перед пользователем на главной странице предстает перечень свежих новостей о работе клуба. Как показано на рис. 1.6 пользователь имеет возможность нажать на кнопку «далее» для того чтобы открыть подробности о событии. Однако после нажатия на кнопку сайт переходит не на страницу с подробностями о выбранной новости, а на страницу с подробностями всех новостей.

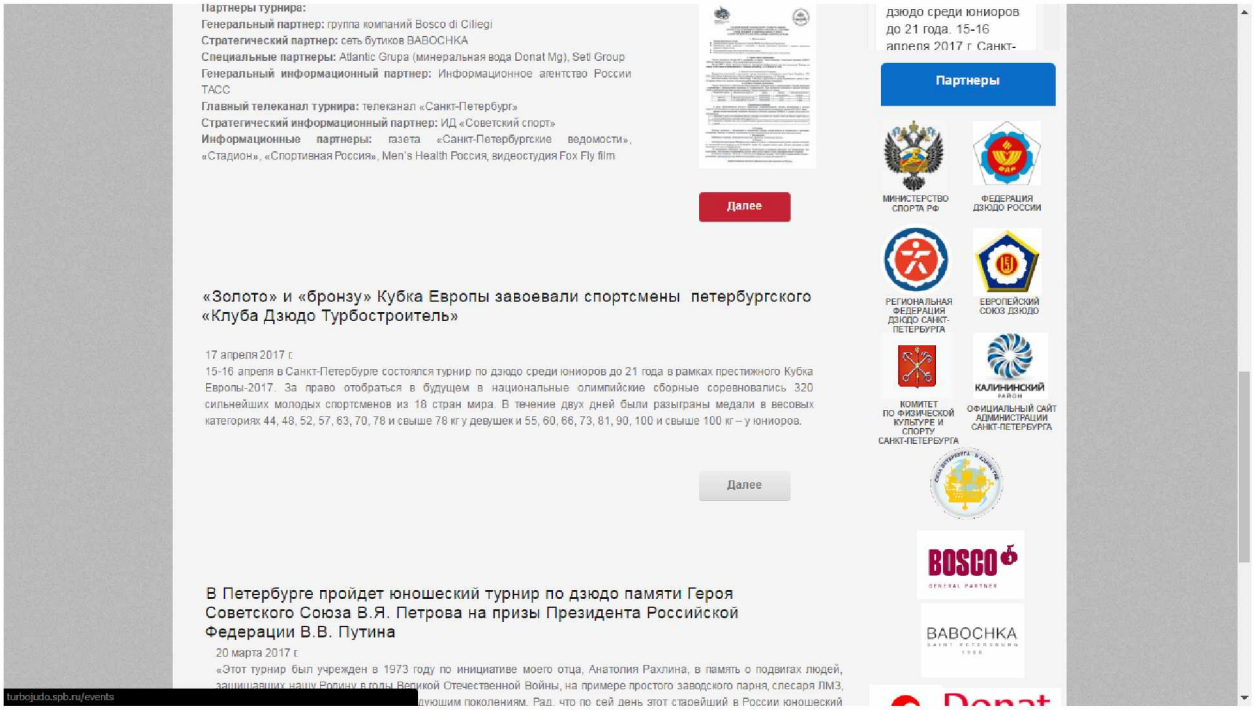

Рис. 1.6. Выбор новости на сайте

Это показано на рис. 1.7. К тому же создатели сайта выбрали плохое сочетание фона и текста, а так же слишком маленький шрифт. В итоге приведенную информацию очень сложно прочесть;

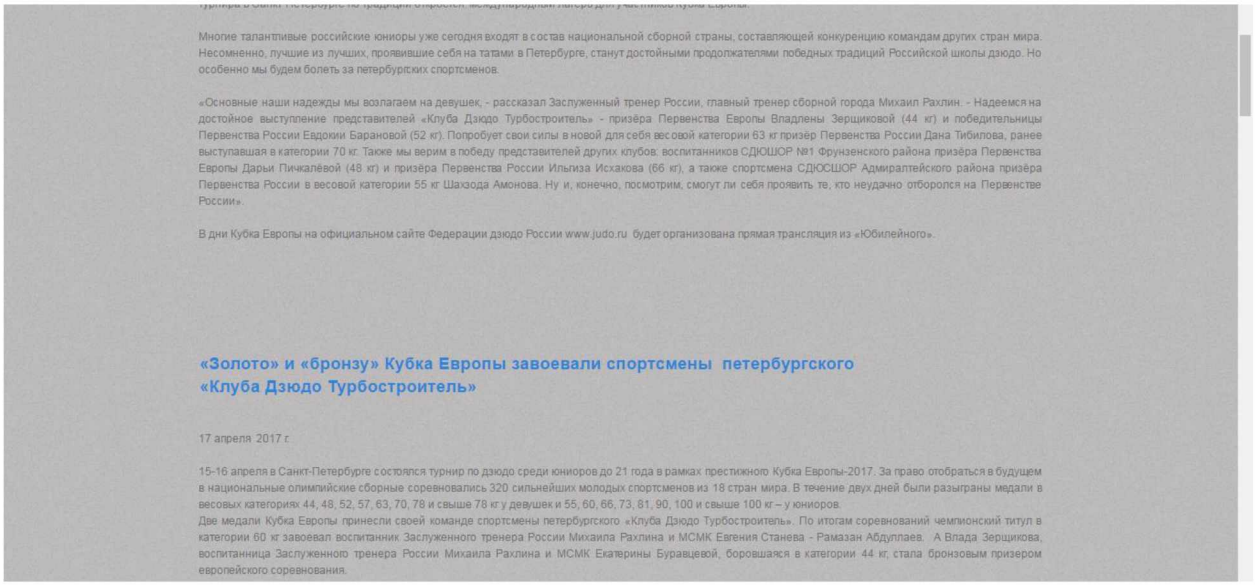

Рис. 1.7. Страница выбранной новости сайта

2) какого-либо способа анонсирования грядущих мероприятий на сайте не реализовано;

3) предоставление информации необходимой для изучения единоборства реализовано слабо. Небольшое количество видеоматериалов

находящихся на странице «видео» рис. 1.8 это единственное что можно отнести к реализации автоматизации этого аспекта деятельности;

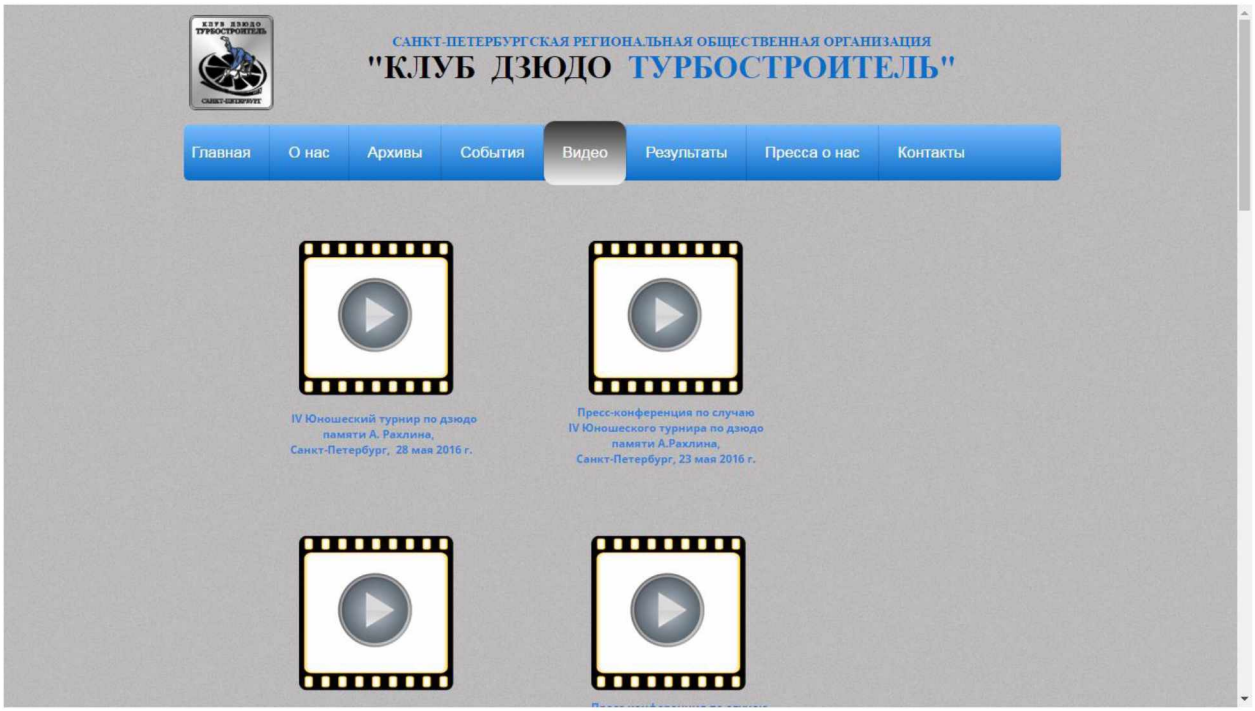

Рис.1.8. Страница «Видео»

Система 3. Сайт клуба самбо «Самбист». Еще одной рассмотренной системой будет сайт Санкт-Петербуржского клуба боевого самбо «Самбист» расположенный по интернет адресу: http://sambopiter.ru/index.html.

1) Открыв главную страницу данного сайта, которая показана на рис. 1.9 видно, что данный сайт не предоставляет какой-либо информации о текущей деятельности клуба, например информации о результатах участия в соревнованиях. Однако в нижнем углу сайта есть небольшой участок страницы, в котором транслируется последняя новость клуба, из размещенных новостей в группе в социальной сети. Такой способ предоставления информации не подходит, так как не каждый пользователь сайта умеет пользоваться социальными сетями, поэтому размещение информации в социальных сетях может быть лишь вспомогательным средством информирования.

Рис.1.9. Главная страница сайта

2) Какого либо способа анонсирования мероприятий так же не предусмотрено возможностями сайта.

3) Функцию по предоставлению информации об изучаемом единоборстве выполняется разделом сайта «Фото и видео» изображение которого показано на рис. 1.10.

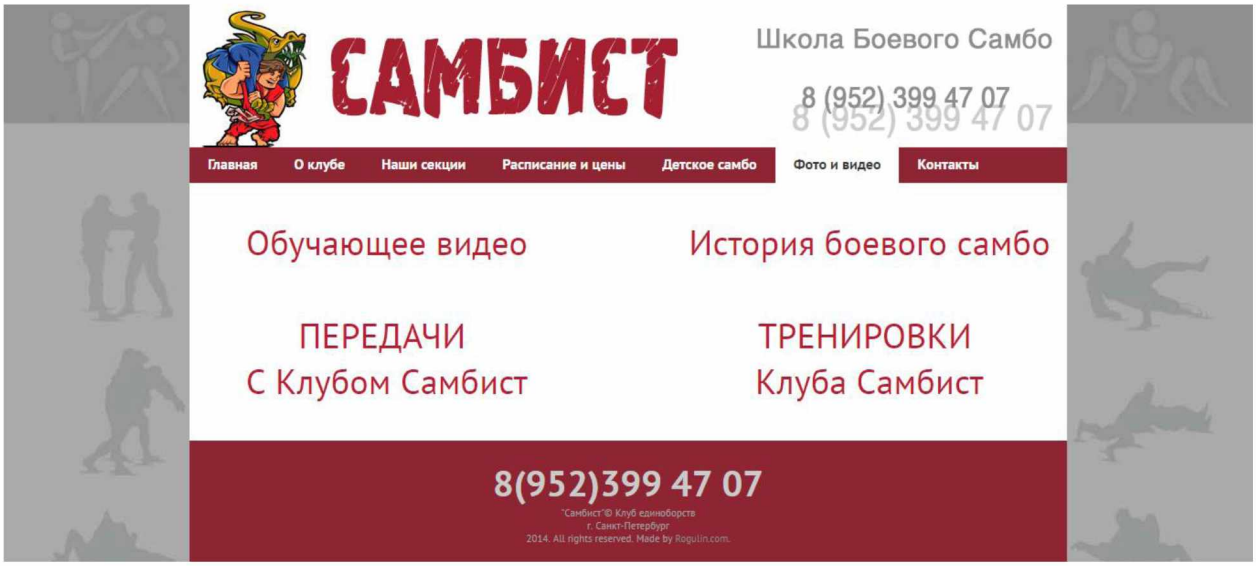

Рис. 1.10. Страница «Фото и видео»

Из четырех ссылок представленных на страницы – три ведут в группу в социальной сети и только четвертая ссылка реализует предоставление информации средствами сайта. Как видно из рис. 1.11 в этом разделе

предоставляется возможность просмотреть два видео, расположенных на стороннем видео хостинге.

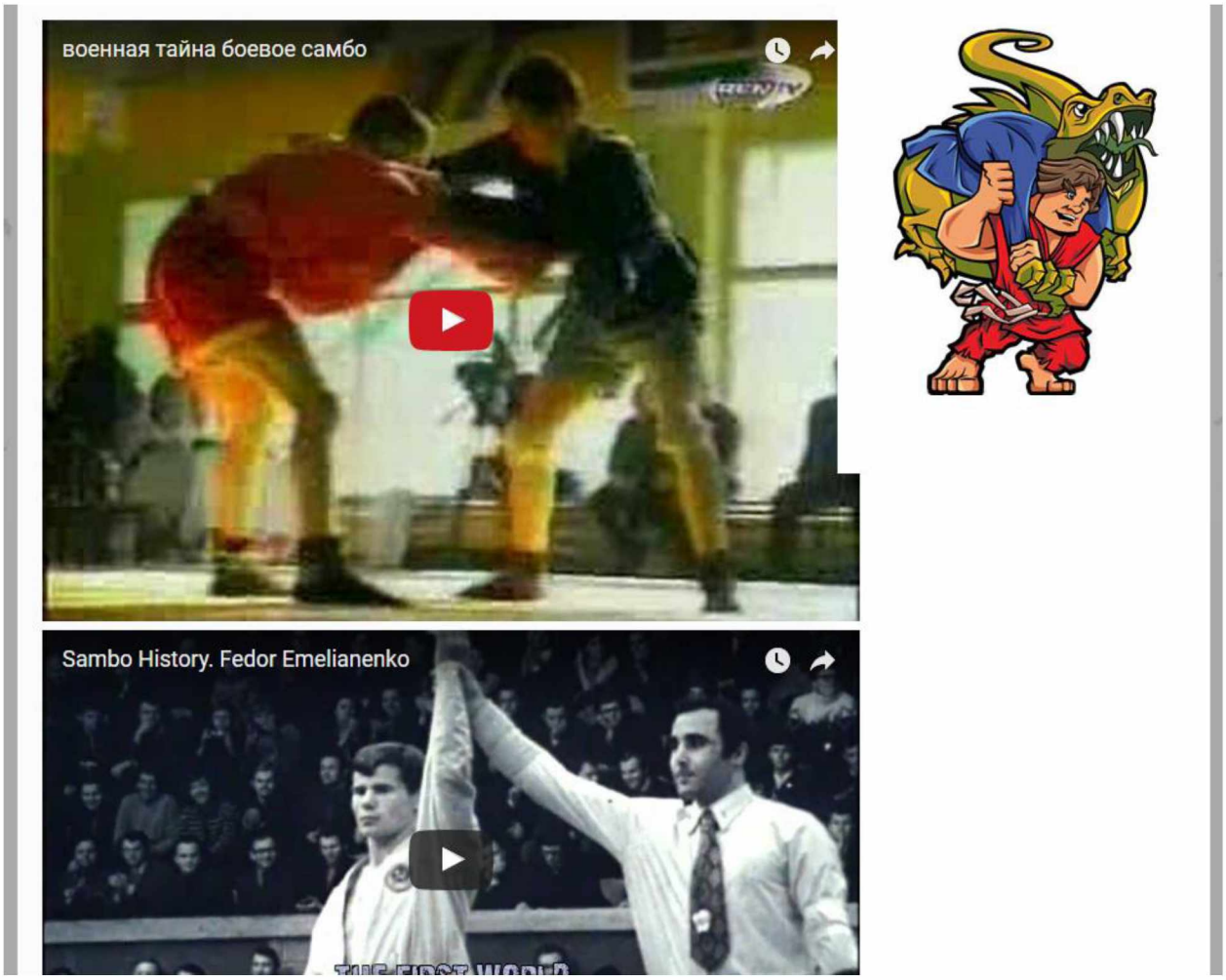

Рис. 1.11. Страница «Фото и видео»

Таким образом, рассмотрев ряд систем, три из которых представлены выше, можно прийти к выводу, что те системы, которые уже существуют – не способный в полной мере удовлетворить требования, представленные к сайту потребностями клуба. Анализ этих систем основывался из трех критериях, указанных как аспекты деятельности клуба, подлежащие автоматизации. Для большей наглядности результаты анализа представлены в таблице 1.1.

Таблица 1.1

Анализ Существующих систем

	Отчет о деятельности	Анонс мероприятий	Образовательные материалы
Сайт «Мастер»	+	-	+/-
Сайт «Турбостроитель»	+	-	+
Сайт «Самбист»	-	-	+

Исходя из анализа, возникает потребность разработать собственный сайт, который будет автоматизировать деятельность клуба, в соответствии с описанными выше аспектами, подлежащими автоматизации.

1.3 Требования к автоматизированной системе

Требования к функциональным характеристикам. Автоматизированная система обязана обеспечивать выполнения перечисленных функций:

- предоставление пользователям сайта отчетности о текущей деятельности клуба. Отчетность должна включать в себя информацию об участии и победах в соревнованиях, о проведении домашних соревнований, об участии в спортивных мероприятиях и любых других важных событиях в жизни клуба;

- предоставление посетителям анонса грядущих событий в жизни клуба;

- предоставление доступа к информации необходимой для постижения и совершенствования техники единоборства, а так же содержащей информацию о правилах проведения и участия в соревнованиях;

- реализовать возможность хранения, редактирования информации о занимающихся и их родственниках, а также удаленного доступа посредствам сети интернет к этой информации;

- защита базы данных от несанкционированного доступа к данным.

Требования к организации входных данных. Входными данными в программе являются данные, которые сторонние пользователи вводят в процессе регистрации – такие как:

- логин;
- пароль.

Так же те, которые обладающий правами администратора пользователь вводит на странице сайта. Входные данные различаются своим предназначением в зависимости он того какой раздел сайта наполняет администратор.

На странице добавления новостей администратор сайта должен иметь возможность ввести следующие данные:

- название новости;
- изображение заголовка;
- содержимое новости;
- изображения для содержимого новости.

На странице добавления анонсов мероприятий администратор сайта должен иметь возможность ввести следующие данные:

- название мероприятия;
- изображение заголовка;
- содержимое анонса;
- изображения для содержимого новости.

На странице добавления обучающей информации администратор сайта должен иметь возможность ввести следующие данные:

- название статьи;
- изображение заголовка;
- содержимое статьи;
- изображения для содержимого статьи.

На странице добавления спортсмена в список администратор сайта должен иметь возможность ввести следующие данные:

- имя;

- отчество;
- фамилия;
- дата рождения;
- вес;
- разряд;
- номер телефона.

На странице добавления родственника в список администратор сайта должен иметь возможность ввести следующие данные:

- имя;
- отчество;
- фамилия;
- номер телефона;
- спортсмен;
- степень родства.

Общие требования к входным данным, вводимым пользователем в программу: данные должны вводиться в специальные поля если это текстовые данные, либо перетягивая мышью на форму если это файлы изображения. Те поля, в которые пользователь вводит информацию вручную, должны проверяться на наличие данных как на стороне клиента, так и после отправки на стороне сервера; данные, вводимые из файла, проверяются в ходе анализа и размещения данных. Загруженные файлы изображений должны размещаться в специальной папке на сервере, текстовые данные должны быть отсортированы и храниться в базе данных.

Требования к организации выходных данных. Вывод данных в программы должен осуществляться путем вывода информации на интернет страницы. Эти страницы отличаются и делятся на несколько групп, по назначению информации и по типу пользователя.

На странице новости должны выводиться данные из таблицы со списком новостей, которые представляют собой список, состоящий из:

- заголовка новости;

- изображения заголовка;

- первой строки текста из содержания.

На странице обучающей информации должны выводиться данные из таблицы со списком обучающих статей, которые представляют собой список, состоящий из:

- заголовка статьи;

- изображения заголовка;

- первой строки текста из содержания.

Также общими являются следующие требования к выводимым данным:

- доступ к таблицам должен зависеть от принадлежности пользователя к определенной группе пользователя с теми или иными правами;

- интернет страницы должны формироваться динамически, в зависимости от данных хранящихся в базе, а так же от действий пользователя уточняющих запрос.

Требования к временным характеристикам. Требования к временным характеристикам зависит от количества запросов и пропускной способности сервера. Информационная система должна выполнять работу, осуществляя как можно меньше запросов к серверу.

Требования к надежности. Надежная работа программы должна обеспечиваться выполнением организационно-технических мероприятий, приведенных ниже:

- организацией бесперебойного питания сервера;

- исполнением всех рекомендаций Министерства труда и социального развития РФ, записанных в Постановлении от 23 июля 1998 г. «Об утверждении межотраслевых типовых норм времени на работы по сервисному обслуживанию ПЭВМ и оргтехники и сопровождению программных средств»;

- исполнением требований ГОСТ 51188-98. Защита информации;

- испытания программных средств на наличие компьютерных вирусов.

Отказы из-за некорректных действий оператора. Отказы программы возможны вследствие некорректных действий оператора (пользователя) при взаимодействии с операционной системой. Во избежание возникновения отказов программы по указанной выше причине следует обеспечить работу конечного пользователя без предоставления ему административных привилегий.

Климатические условия эксплуатации. Климатические эксплуатационные условия, в которых обязаны обеспечиваться заданные характеристики, должны соответствовать требованиям, которые предъявляются к техническим средствам в части условий их эксплуатации.

Требования к численности и квалификации персонала. Для управления работы сайта требуется минимум один человек, который должен наполнять его актуальной информацией. Так же требуется персонал, который обслуживает сервер.

Требования к составу и параметрам технических средств. Для работы автоматизированной системы на стороне клиента требуется персональный компьютер с доступом в интернет. Для работы приложения на стороне сервера требуется компьютер или специализированное оборудование способные выполнять задачи сервера и имеющие выход в интернет

Требования к информационным структурам и методам решения. Пользовательский интерфейс должен быть интуитивно понятным и содержать подсказки. Должен существовать программный доступ к управлению профилями. Добавленные страницы с данными должны отображаться по принципу стека – последняя добавленная должна отображаться первой.

Требования к исходным кодам и языкам программирования. Языком программирования, реализующим исходные коды стороне сервера, является язык программирования C# с использованием ASP.NET MVC Framework. Интегрированной средой разработки системы должна быть среда Microsoft Visual Studio 2013 для Web. Для взаимодействия с СУБД и создания базы

данных должно использоваться объектно-реляционное отображение ADO.NET Entity Framework. Языками программирования на стороне клиента являются HTML и Javascript с использованием библиотек Jquery и AJAX.

Требования к программным средствам, используемым программой. У клиента, для работы с автоматизированной системой должна быть установлена операционная система, поддерживающая взаимодействие с интернетом и работу современного веб браузера. Так же должен быть установлен браузер с поддержкой Javascript и стандарта HTML5.

Требования к защите информации и программ. В Системе должен быть обеспечен надлежащий уровень защиты информации в соответствии с законом о защите персональной информации и программного комплекса в целом от несанкционированного доступа - “ Об информации, информатизации и защите информации” РФ N 24-ФЗ от 20.02.95.

Специальные требования. Программа должна обеспечивать взаимодействие с пользователем (оператором) посредством графического пользовательского интерфейса. Программа должна обеспечивать высокую защиту данных и удобный просмотр необходимой информации.

Глава 2. ПРОЕКТИРОВАНИЕ И ПРОГРАММНАЯ РЕАЛИЗАЦИЯ ИНФОРМАЦИОННОЙ СИСТЕМЫ

2.1 Выбор средств разработки

На данный момент существует множество средств, с помощью которых можно написать сайт, который смог бы автоматизировать информационное сопровождение деятельности клуба единоборств. Однако для реализации серверной части данной системы были выбраны язык программирования C# и платформа ASP.NET MVC framework работа с которыми происходит в среде программирования Microsoft Visual Studio Express 2013 для Web. Эти инструменты выбраны не случайно, дело в том, что в последнее время стал популярным такой шаблон проектирования как «Модель–Представление–Контроллер», который является идеальной основой для разработки сложного веб приложения. ASP.NET MVC framework включает в себя всю базовую инфраструктуру, которая необходима для реализации шаблона, и является практически идентичной для большинства веб приложений использующих данный шаблон. Таким образом, используя данную платформу, можно освободиться от написания от идентичного, для подобных приложений, кода и сосредоточиться на реализации функционала автоматизированной системы. Еще одним аргументом к использованию данного инструмента является то, что весь исходный код данной платформы находится в открытом доступе, а компоненты из которых она состоит связаны между собой с помощью интерфейсов, что позволяет заменить любой компонент собственной реализацией в случае если его функционал не отвечает требованиям поставленной задачи.

2.2 Технология построения веб сайта

Шаблон проектирования «Модель–Представление–Контроллер». «Model–View–Controller» является схемой, которая основана на разделении данных, пользовательского интерфейса и управляющей логики в виде трех независимы составляющих: модель, представление и контроллер, при этом в случае необходимости модифицировать любой из них не придется менять остальные.

Сама концепция MVC не нова, она была описана еще в 1978 сотрудником научно-исследовательского центра «Xerox PARC» Трюгве, который работал над языком программирования «Smalltalk».Только в 1988 году была напечатана конечная версия концепции MVC была в журнале «Technology Object». Позже «Apple», внедрила технологию «WebObjects», реализованную на «Objective-C» и стала популяризировать шаблон и в вебе. На «Java» — шаблон популяризировался после порта «WebObjects». Следующий виток популярности наступил во время развития фреймворков, которые ориентируются на быструю развёртку, на языках «Python» и «Ruby» [1].

В октябре 2007 года компания Майкрософт анонсировали новую платформу для веб-разработки с использованием шаблона MVC, построенную на основе ASP.NET. Инфраструктура ASP.NET MVC Framework реализует шаблон MVC и при этом обеспечивает существенно улучшенное разделение ответственности. На самом деле в ASP.NET MVC внедрен современный вариант MVC, который особенно хорошо подходит для веб-приложений [2].

Особенности архитектурного шаблона MVC. При использовании архитектурного шаблона MVC предполагается, разделение приложения MVC, как минимум, на три части, которые описаны ниже.

- Модели, которые содержат данные, с которыми работают пользователи, или представляют эти данные. Модели могут быть как

обычными моделями представлений, представляющими данные, которые передаются между представлениями и контроллерами, так и моделями предметной области, содержащими бизнес-данные, а так же операции, для преобразования и правила для обработки данных.

- Представления, которые применяются для визуализации определенной части модели в виде пользовательского интерфейса.

- Контроллеры, обрабатывающие запросы, которые к ним поступают - выполняют операции над моделью и подбирают то представление, которое требуется для визуализации пользователю.

Все составляющие архитектуры MVC четко определены и самодостаточны – они реализуют принцип разделения ответственности. Логика, управляющая данными модели, содержится только в ней, логика, которая отображает данные, располагается исключительно в представлении, а тот код, который обрабатывает пользовательский ввод и запросы находится в контроллере. Таким образом, четко разграничивая все части в приложении, сопровождение и расширение в течение срока его существования происходит значительно проще, вне зависимости от его увеличения и усложнения [3].

Модель предметной области. Модель предметной области является ключевой частью MVC приложения. Эта модель называется предметной областью и создается путем определения реальных объектов, правил и операций, которые существуют в данной отрасли или роде деятельности, и должны поддерживать приложение.

После создается программное представление предметной области – модель предметной области. Для ASP.NET MVC Framework набор типов C#, например, таких как классы и структуры, которые все вместе называются типами предметной области – это и есть модель предметной области. Методами, которые определенны в типах предметной области, представляются операции из предметной области. При помощи логики выражаются правила предметной области, заключенной в методах, или путем использования атрибутов C#. Таким образом объект предметной

области создается путем создания экземпляра типа предметной области для представления определенной порции данных. Модели предметной области обычно долговечны и устойчивы. Есть разные способы, которыми это достигается, однако реляционные базы данных остаются более распространенным вариантом [4].

Таким образом, единственное официальное определение бизнес-данных и процессов внутри приложения это модель предметной области. Устойчивая модель предметной области является также официальным определением состояния представления предметной области. Множество проблем, которые возникают во время сопровождения приложения решает подход с применением модели предметной области. В случае возникновения потребности манипулирования данными в модели, либо необходимости добавления нового правила или процесса, то придется изменять только модель предметной области.

Реализация MVC в ASP.NET. В инфраструктуре MVC контроллеры представляют собой классы C#, которые обычно наследуют класс System.Web.Mvc.Controller. Все открытые методы в производном от Controller классе - это методы действий, которые ассоциируется с конфигурируемым URL с помощью системы маршрутизации ASP.NET. После того как по URL, связанному с методом действия, отправляется запрос, в классе контроллера выполняются операторы, чтобы провести некоторую операцию над моделью предметной области и затем выбрать представление для отображения клиенту. Взаимодействия между моделью, представлением и контроллером показаны на рис. 2.1 [5].

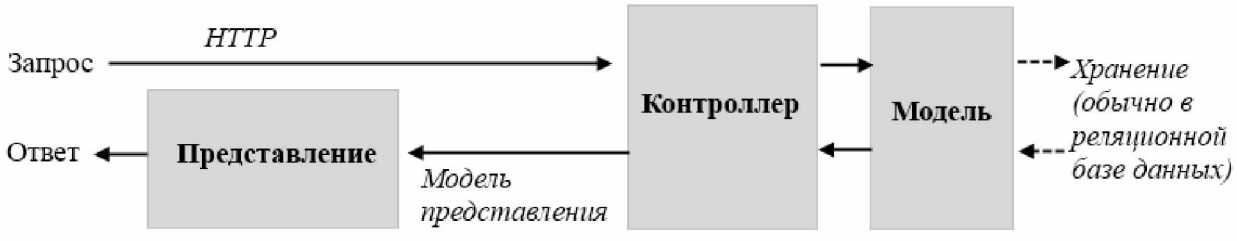

Рис.2.1. Взаимодействие в приложении MVC

В ASP.NET MVC Framework инфраструктуре применяется механизм визуализации Razor – компонент, который отвечает за обработку представления с целью генерации ответа для браузера [6].

Инфраструктура ASP.NET MVC не налагает никаких ограничений на реализацию модели предметной области. Модель можно создать с помощью обычных объектов C# и реализовать постоянство с применением любой базы данных, инфраструктуры объектно-реляционного отображения или другого инструмента работы с данными, поддерживаемого .NET [7].

Проектирование приложения. В связи с тем, что данное приложение, будет разработано с помощью объектно-ориентированного языка программирования имеет смысл начать проектирование приложения с построения UML диаграмм.

Первой спроектированной диаграммой, будет диаграмма вариантов использования. Построение диаграммы даст возможность представить задачи автоматизации и требования к проектируемой системе в виде отношений между актёрами и вариантами использования, что в свою очередь позволяет сформировать функциональные требования к будущей системе и представить ее поведение.

Актёрами в проектируемой системе являются пользователи, использующие ее, которые делятся на три группы – по уровню доступа к ее функционалу. В основную группу входят пользователи не авторизованные в системе и авторизованные пользователи, не имеющие расширенных прав – этот актёр на диаграмме назван как «Посетитель». Посетитель имеет возможность получить информацию, либо зарегистрироваться в системе. Получаемая пользователем информация может быть, в свою очередь, отчетом о прошедших мероприятиях, информацией о технике единоборств, анонсом мероприятий, либо расписанием занятий. Следующим актером в системе является «Модератор». Права модератора расширены по сравнению с правами посетителя вследствие чего он получает возможность редактировать расписание, просматривать и редактировать перечень

спортсменов и их родителей и добавлять новые статьи с информацией. Добавляемыми статьями с информацией могут являться статьи с анонсами мероприятий, статьи о технике единоборств и статьи с отчетами о мероприятиях. Прежде чем выполнить действия доступные модератору пользователь будет направлен на страницу с авторизацией, для подтверждения своих прав. Последним актером является «Администратор». Администратор обладает полным доступом ко всем возможностям взаимодействия с системой, и помимо возможностей модератора, он получает возможность назначать права доступа для пользователей. Для осуществления действий администратора – пользователь так же должен авторизоваться.

Исходя из вышесказанного была построена следующая диаграмма использования, которая изображена на рис. 2.2.

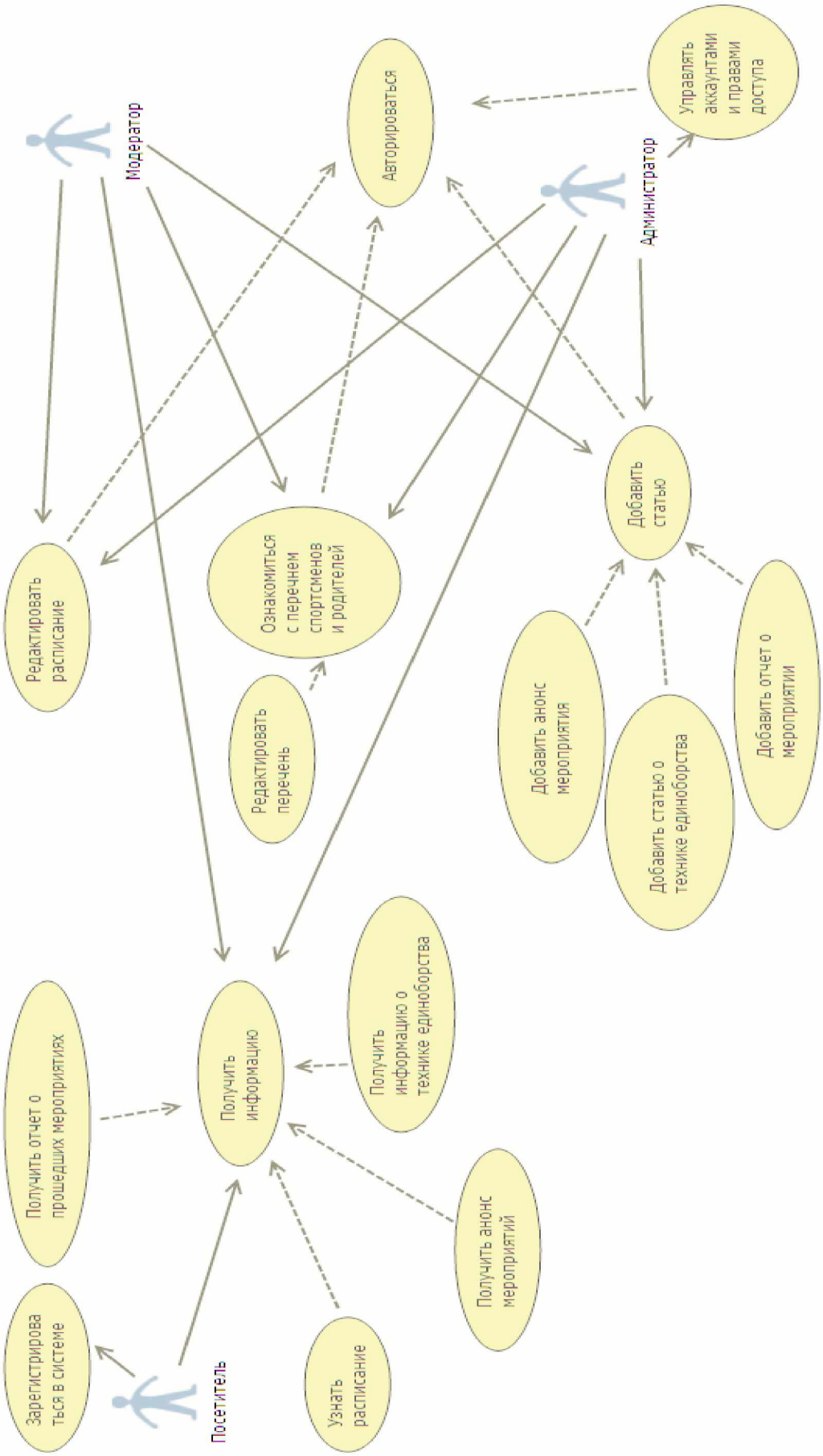

Рис. 2.2. Диаграмма вариантов использования

Следующей спроектированной диаграммой является диаграмма деятельности. С помощью этой диаграммы можно показать способы достижения прецедентов из диаграммы вариантов использования путем разложения деятельности проектируемой системы на её составные части. Под деятельностью понимается спецификация исполняемого поведения в виде координированного последовательного и параллельного выполнения подчинённых элементов — вложенных видов деятельности и отдельных действий, соединённых между собой потоками, которые идут от выходов одного узла к входам другого [8].

Спроектированная диаграмма разделена на три swimlane - визуальные разделенные линии внутри диаграммы, которые группируют действия по ролям: «Гость» или «Пользователь с правами User», пользователь с правами «Moderator» или «Admin» и только пользователь с правами «Admin». Swimlane визуализирует действия, которые выполняются особыми типами ресурсов, ролей или элементами организации, или которые связаны с особым местом. Кроме того, на диаграммах Swim Lane можно указать роли исполнителей работ и таким образом более качественно задокументировать ответственность исполнителе.

На рис. 2.3 представлена диаграмма деятельности, спроектированная для разрабатываемого приложения, которая отображает общий взгляд на действия, которые будет реализовывать приложение. Помимо этого спроектированы еще три диаграммы деятельности, в которых ряд сложных действий разобраны более подробно и представлены в виде поддействий.

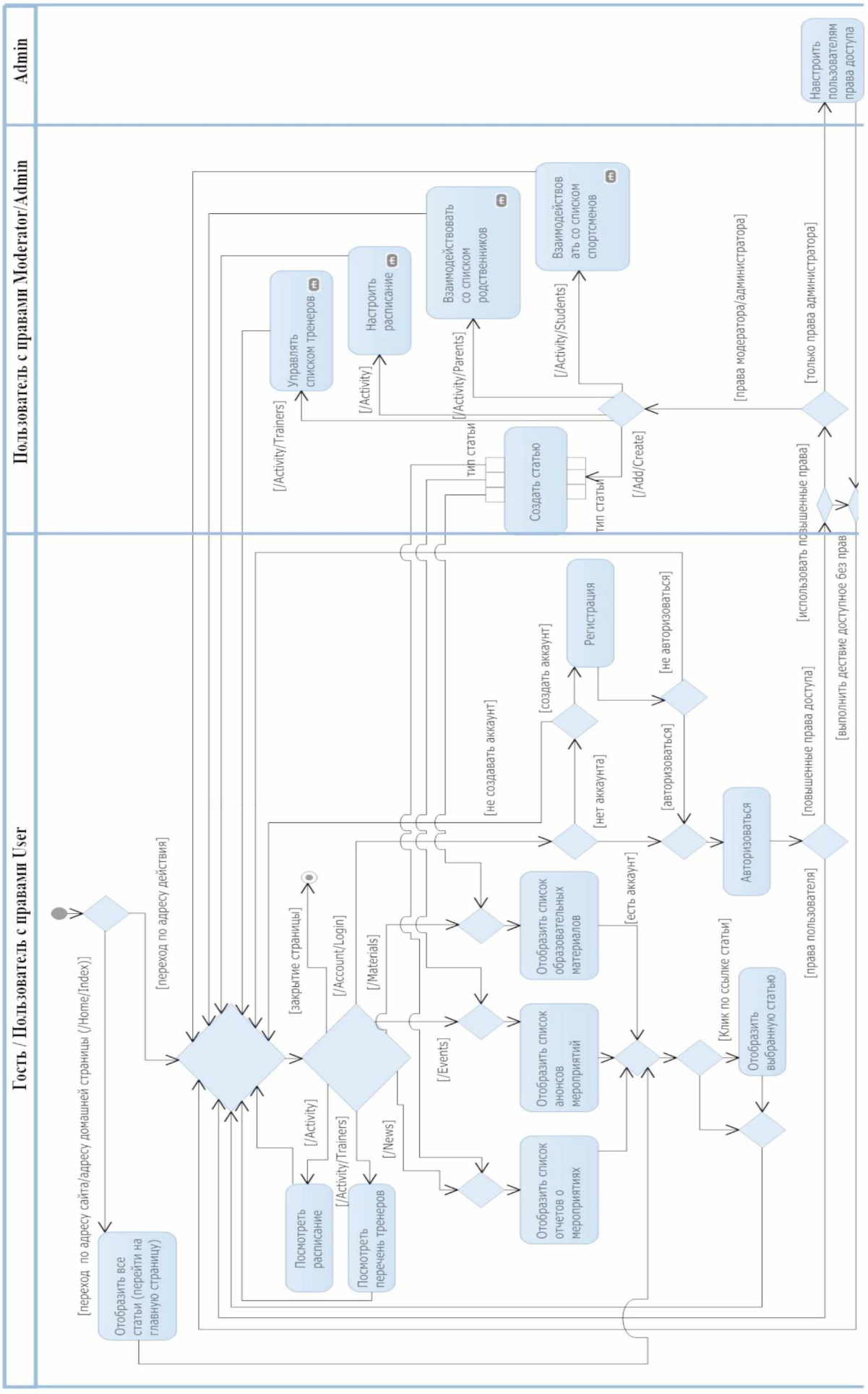

Рис. 2.3. Общая диаграмма активности

На рис. 2.4 более подробно разобрано действие «Управлять списком тренеров».

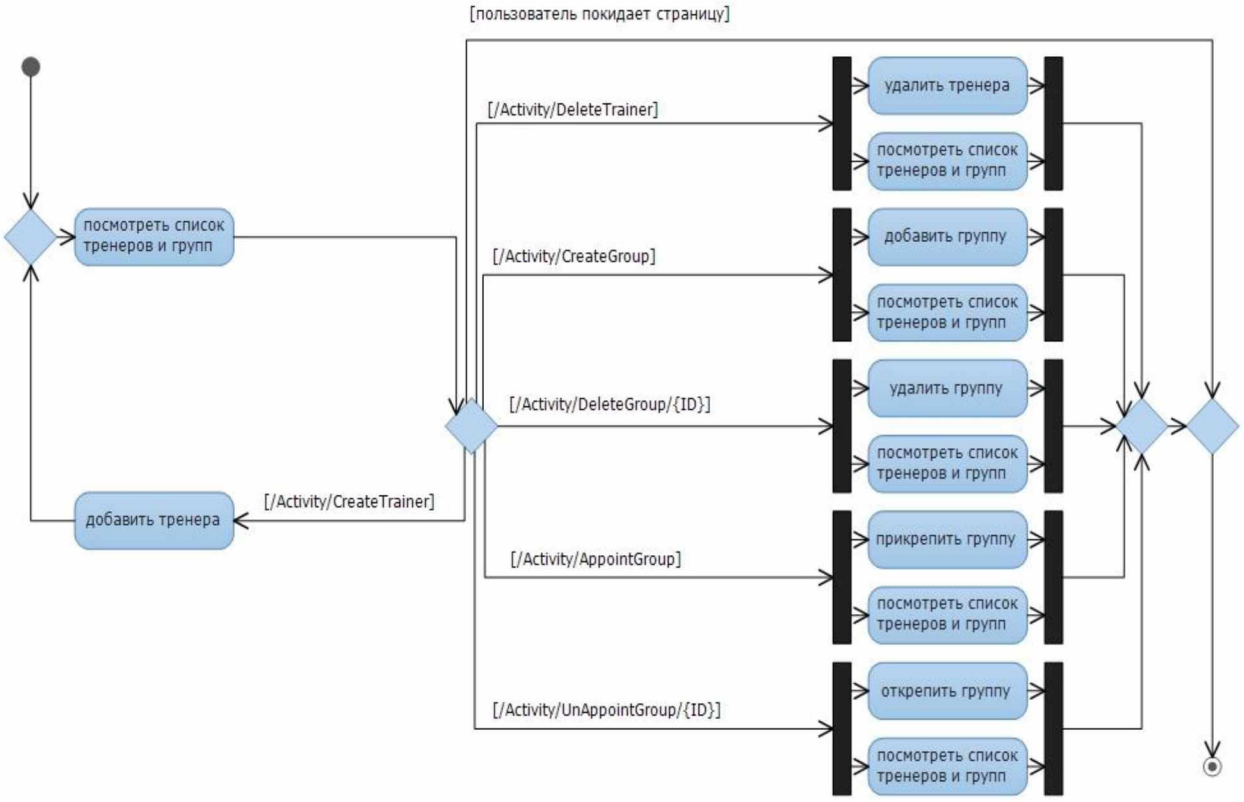

Рис.2.4. Поддиаграмма действия «Управлять списком тренеров»

С помощью рис. 2.5 более подробно разобрано «Настроить расписание».

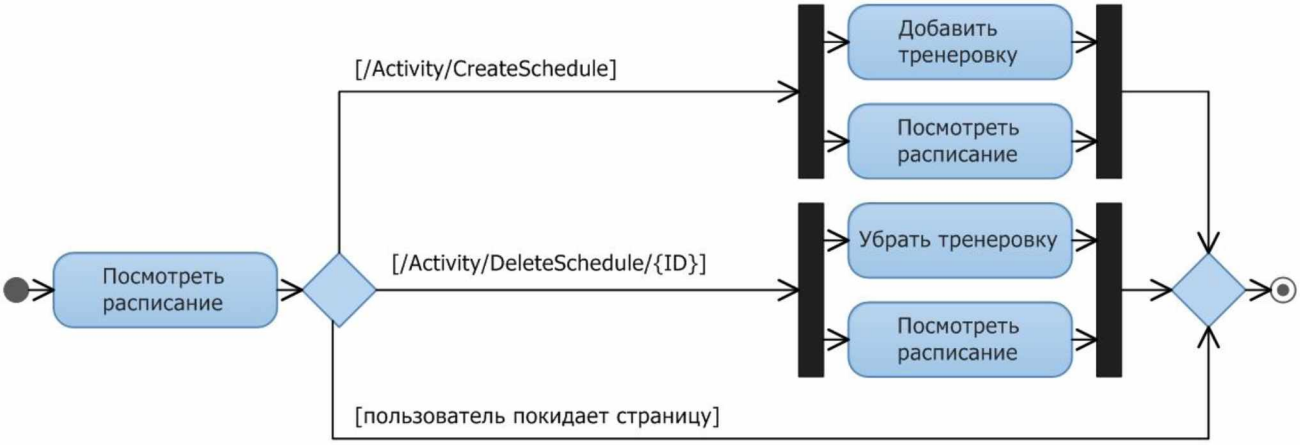

Рис. 2.5. Поддиаграмма действия «Настроить расписание»

На рис. 2.6 более подробно разобрано действие «Взаимодействовать со списком спортсменов».

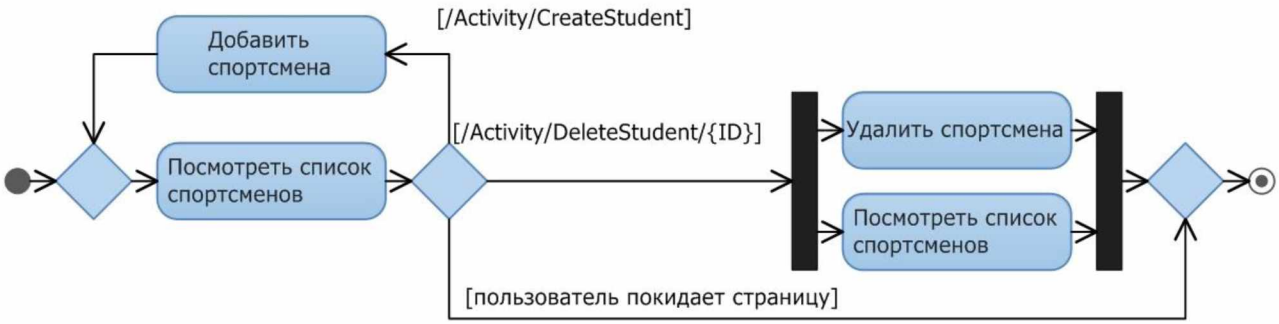

Рис. 2.6. Поддиаграмма действия «Взаимодействовать со списком спортсменов»

С помощью рис. 2.7 более подробно «Взаимодействовать со списком родственников».

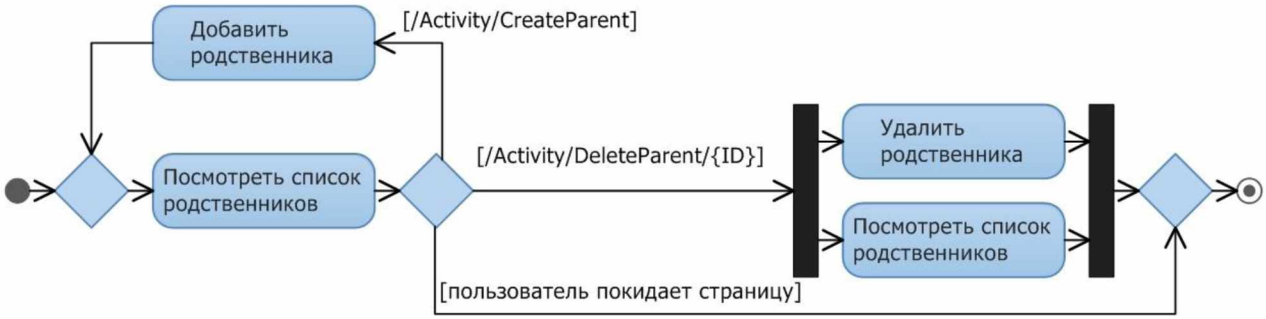

Рис. 2.7. Поддиаграмма действия «Взаимодействовать со списком родственников»

Последняя проектируемая диаграмма – диаграмма классов (см. рис.2.8). С помощью этой диаграммы демонстрируются классы проектируемой системы, их атрибуты, методы и взаимосвязи между ними [9].

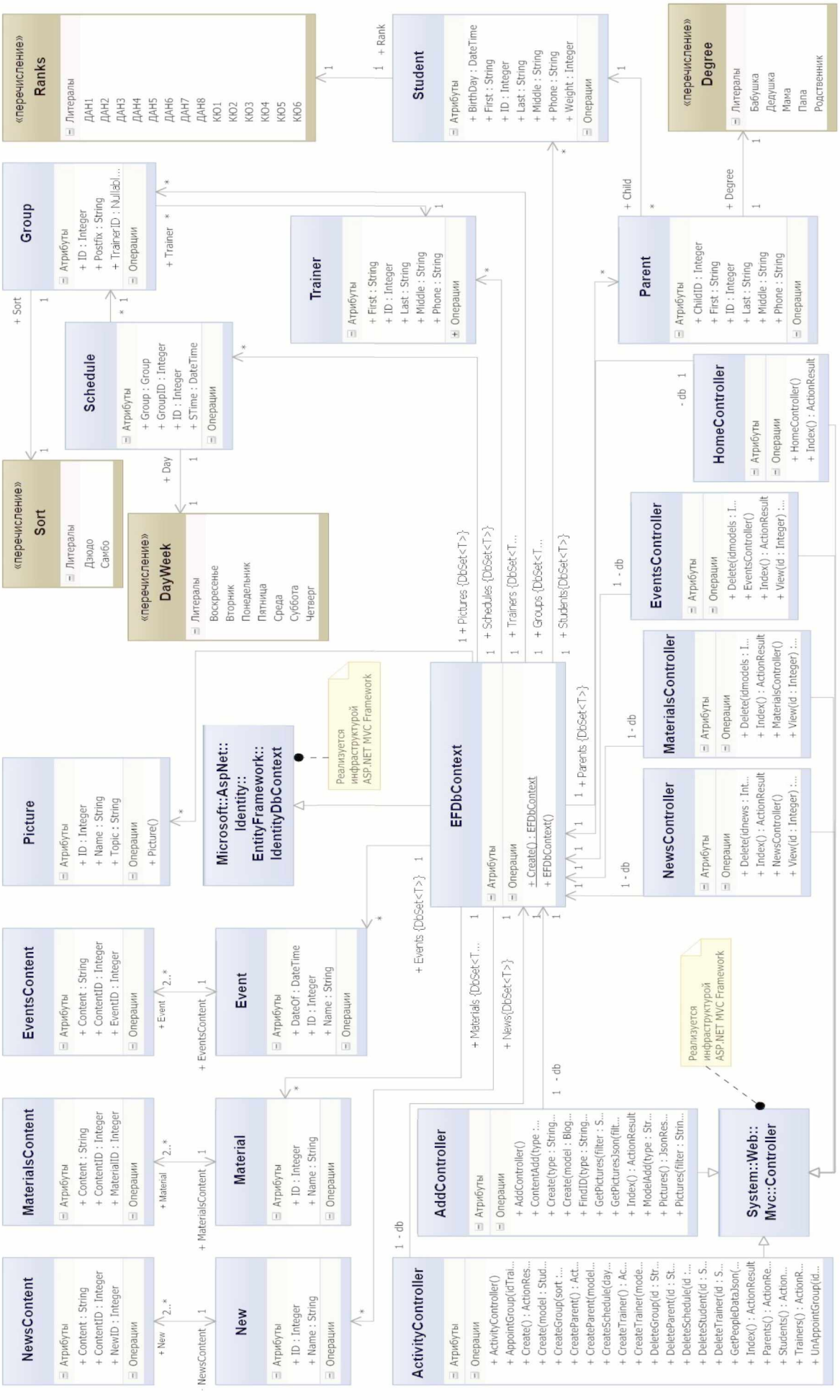

Рис. 2.8. Диаграмма классов

Проектирование базы данных. Основой для любого компьютерного приложения являются данные, которыми оно манипулирует, поэтому проектирование программы стоит начать именно с них. Для того чтобы реализовать требуемый функционал, нужна база данных, которая будет хранить в себе всю необходимую для работы приложения информацию.

Чтобы реализовать базу данных была выбрана инфраструктура объектно реляционного отображения данных, представленная в виде ADO.NET Entity Framework. Эта технология доступа к данным является object-relational mapping решением для .NET Framework от Microsoft. Предоставляет возможность взаимодействия с объектами как посредством LINQ в виде LINQ to Entities, так и с использованием Entity SQL. Таким образом, выбрав Entity Framework можно проектировать базу данных и манипулировать этими данными, используя возможности языка программирования C#. Для создания таблицы с данными необходимо создать класс на языке C#, публичные свойства которого будут отображать поля созданной на основе класса таблицы. Уточняющая информация для Entity Framework – о том, как строить базу данных и механизма визуализации Razor – о том, как визуализировать модель в представлении может указываться в атрибутах свойств. Внешние связи же имитируются путем добавления виртуальных свойств, тип которых задается таким же, каким и тип класса той таблицы, с которой устанавливается связь [10]. Ниже следует перечень классов на основе которых реализуется база данных.

1) Публичный класс «Event» предназначен для создания таблицы, в которой будет храниться список с грядущими событиями. У этого класса есть три свойства:

- целочисленное свойство «id» – это первичный ключ, который базируется на идентификаторе новости;

- свойство «dateof» типа «datetime» предназначено для хранения даты проведения будущего мероприятия;

- строковое свойство «name» хранит в себе название мероприятия.

2) Класс «EventsContent» предназначен для таблицы, она содержит в себе весь контент, который будет выводиться на страницах с анонсами мероприятий. Этот класс включает в себя три свойства, которые обозначают поля таблицы и одно виртуальное свойство, означающее внешний ключ:

- целочисленное свойство «eventid», к которому добавлены три атрибута: «key» и «column» с параметром «order = 1» - которые вместе означают, что свойство является первой частью составного первичного ключа, а также атрибут «foreignkey» с параметром «event» - который говорит, что свойство является так же внешним ключом к таблице «event». это свойство по которому часть содержимого страницы соотносится с тем к какому мероприятию она принадлежит;

- целочисленное свойство «contentid», с двумя атрибутами «key» и «column» с параметром «order = 2», которые означают, что это свойство является второй частью составного первичного ключа. это свойство, которое определяет расположение этой части контента, на странице с мероприятием;

- строковое свойство «content» - свойство содержащее часть контента, который выводится на страницу с мероприятием;

- виртуальное свойство «event» типа «event» указывающее на связь с таблицей «events».

3) Класс «New» - таблица с перечнем отчетов о посещенных мероприятия, или об иных важных для клуба событиях. Он содержит два свойства:

- целочисленное «ID» - первичный ключ;

- строковое «Name» - название новости.

4) Класс «NewsContent» - таблица с наполнением отчетов о мероприятиях. Содержит в себе свойства:

- целочисленное «NewID» с тремя атрибутами «Key», «Column» с параметром «Order = 1» и «ForeignKey». Определяет отчет, которому принадлежит контент;

- целочисленное свойство ContentID, с двумя атрибутами «Key» и «Column» с параметром «Order = 2»;

- строковое свойство «Content»;

- виртуальное свойство «New» типа «New» указывающее на связь с таблицей «News».

5) Класс «Material» предоставляет таблицу, содержащую список статей предназначенных для предоставления информации необходимой для изучения единоборства. Данный включает в себя следующие свойства:

- целочисленное «ID» - первичный ключ;

- строковое «Name» - название новости.

6) Класс «MaterialsContent» это таблица с содержимым для страниц со статьями из таблицы «Materials». Он содержит три свойства:

- целочисленное «MaterialID» с тремя атрибутами «Key», «Column» с параметром «Order = 1» и «ForeignKey». Определяет отчет, которому принадлежит контент;

- целочисленное свойство ContentID, с двумя атрибутами «Key» и «Column» с параметром «Order = 2»;

- строковое свойство «Content»;

- виртуальное свойство «Material» типа «Material» указывающее на связь с таблицей «Materials».

7) Класс «Picture» этот класс предназначен для создания таблицы, содержащей имена изображений загружаемых на сервер при создании новой статьи с новостями, анонсом, или материалами по изучению единоборства и название темы, во время создания которой они были добавлены. Данный класс содержит в себе следующие свойства:

- целочисленное «ID» - первичный ключ, с атрибутами «Key» и «DatabaseGenerated» с параметром «DatabaseGeneratedOption.None» -

которые означаю, что для данного поля функция автоматического инкрементирования отключена;

- строковое «Name» - название изображения;

- строковое «Topic» с название темы, при создании которой было добавлено изображение.

8) Класс «Student» предназначен для создания таблицы содержащей перечень учеников занимающихся в клубе единоборств. Эта таблица включает в себя поля содержащие такие данные, как: полное имя ученика, дата рождения, вес, разряд и номер мобильного телефона. Данный класс включает в себя 8 свойств, определяющих поля таблицы, а так же для поддержки функционала класса создано специальный перечисляемый тип:

- целочисленное свойство «ID» - первичный ключ;

- строковые свойства «First», «Middle» и «Last» для хранения таких данных как имя, отчество и фамилия спортсмена – соответственно;

- свойство «BirthDay» типа «DateTime» для хранения данных о дате рождения;

- целочисленное свойство «Weight» – вес спортсмена;

- свойство «Rank» - перечисление типа «Ranks». Предназначено для хранения данных о разряде спортсмена. Перечисляемый тип «Ranks» включает в себя следующие константы: КЮ6, КЮ5, КЮ4, КЮ3, КЮ2, КЮ1, ДАН1, ДАН2, ДАН3, ДАН4, ДАН5, ДАН6, ДАН7, ДАН8;

- строковое свойство «Phone» для хранения номера телефона спортсмена.

9) Класс «Parent» отвечает за создание таблицы хранящей сведения о родителях учеников. Этот класс включает в себя следующие свойства:

- целочисленное свойство «ID» - первичный ключ;

- строковые свойства «First», «Middle» и «Last» для хранения таких данных как имя, отчество и фамилия родителя – соответственно;

- строковое свойство «Phone» для хранения номера телефона родителя;

- целочисленное свойство «ChildID» которое является внешним ключом для таблицы «Students»;

- свойство «Degree» - перечисление типа «Degree». Предназначено для хранения данных о степени родства. Перечисляемый тип «Degree» включает в себя следующие константы: Папа, Мама, Дедушка, Бабушка, Родственник;

- виртуальное свойство «Child» типа «Student» – связь с таблицей «Students».

10) В таблице, созданной на основе класс «Trainer», хранится информация о тренерах работающих в клубе единоборства. Класс содержит следующие свойства:

- целочисленное свойство «ID» - первичный ключ;

- строковые свойства «First», «Middle» и «Last» для хранения таких данных как имя, отчество и фамилия родителя – соответственно;

- строковое свойство «Phone» для хранения номера телефона тренера.

11) Класс «Group» предоставляет таблицу, в которой будет отображен перечень имеющихся в клубе единоборств групп. Класс имеет следующие свойства:

- целочисленное свойство «ID»;

- свойство «Sort» типа перечисление «Sort». Свойство предназначено для указания вида единоборства изучаемого в группе. Перечисляемый тип «Sort» включает в себя следующие константы: Дзюдо, Самбо;

- строковое «Postfix» для указания названия группы;

- целочисленное свойство «TrainerID» с возможностью присвоения значения типа null. Является внешним ключом к таблице Trainers. Возможность указания значения null нужна для создания

группы до присвоения к ней тренера, либо для открепления тренера от группы;

- виртуальное свойство «Trainer» типа «Trainer» – связь с таблицей «Trainers».

12) Класс «Schedule» имитирует таблицу в которой будет храниться расписание занятий. Таблица хранит в себе информацию о дне недели, времени начала тренировки и группе к которой эта тренировка относится. Этот класс включает в себя следующие свойства:

- целочисленное свойство «ID»;

- свойство «Day» типа перечисление «DayWeek». Свойство предназначено для указания вида единоборства изучаемого в группе. Перечисляемый тип «DayWeek» включает в себя следующие константы: Понедельник, Вторник, Среда, Четверг, Пятница, Суббота, Воскресенье;

- свойство «STime» типа «DateTime»;

- целочисленное свойство «GroupID» является внешним ключом к таблице «Groups»;

- виртуальное свойство «GroupID» означает связь с таблицей «Groups».

Такие таблицы как: AspNetRoles, AspNetUserClaims, AspNetUserLogins, AspNetUserRoles, AspNetUsers – являются базой для работы систему авторизации разработанной Microsoft под названием ASP.NET Identity. Поэтому эти таблицы идентичны для любого приложения использующего эту систему и создаются без участия разработчика.

Еще одной таблицей является таблица «__MigrationHistory». Эта таблица хранит в себе все изменения, внесенные в структуру базы данных после первоначального создания базы. Эта таблица также создается автоматически и является идентичной для всех проектов использующих Entity Framework.

Так как для написания приложения используется большой объем кода, то нет возможности прокомментировать весь код целиком. Однако следует описать те разработки, которые являются ключевыми и представляют особенный интерес для рассмотрения. Как и было заявлено в требованиях, приложение реализует предоставление информации о прошедших мероприятиях, о грядущих мероприятиях и информацию для изучения единоборства в виде блога. Для того, чтобы сделать добавление статей в такую систему без навыков программирования, а так же чтобы данные введенные создателем данных статей могли храниться в базе данных и после извлекаться и интерпретироваться в нужном виде - был разработан специальный инструмент. Этим инструментом является расширяющий метод Interpred.

Суть этого метода заключается в следующем. Метод принимает строки с текстом и именами изображений из базы данных. Далее строки анализируются и в случае если в строке содержится имя изображения – создается слайдер типа «карусель» с использованием bootstrap классов. Далее метод анализирует следующую строку и в случае если там изображение – добавляет в слайдер; если нет - закрывает его. Если элементом коллекции оказывается строка текста, то она добавляется заключенная в тег <p>.

Метод Interpred принимает в параметрах статический класс, содержащий вспомогательные методы, для быстрой генерации шаблонов HTML разметки и коллекцию строк.

Выглядит это следующим образом: «public static MvcHtmlString Interpred(this HtmlHelper html, List<string> calls){».

Подразумевается, что принимаемые строки будут строками с данными, из которых нужно сгенерировать разметку страницы. Далее следует инициализация переменных, используемых для создания индикаторов в карусели:

Листинг 2.1. Фрагмент кода метода Interpred

```
TagBuilder tag = new TagBuilder("div");
TagBuilder tagIndOl = new TagBuilder("ol");
TagBuilder tagContDiv=new TagBuilder("div");
int idCarousel=0;
int imageRow = 0;
```

Класс TagBuilder - содержит классы и свойства для создания HTML-элементов. В коде выше создаются три экземпляра класса TagBuilder:

Листинг 2.2. Фрагмент кода метода Interpred

```
tag
tagIndOl
tagContDiv
```

И две целочисленных переменных:

```
idCarousel
imageRow
```

Далее следует код цикл, который перебирает все элементы коллекции строк переданной в параметре и ищет в них текст «<img», который означает, что в этой строке указано изображение:

Листинг 2.3. Фрагмент кода метода Interpred

```
foreach(string call in calls)
{
   if (call.Contains("<img"))
   {
```

В случае если это первое изображение в слайдере, его идентификатор увеличивается: «if (imageRow++ == 0) idCarousel++;».

Далее создается экземпляр класса TagBuilder для создания маркированного списка. Список станет основой для создания индикатора выбранного изображения за счет класса «active». Этот класс присваивается первому изображению в слайдере за счет тернарного оператора

«imageRow>1?"":"active"» - который проверяет, что индекс изображения в слайдере не больше единицы. А так же список дает возможность выбрать нужное изображение за счет атрибута «data-slide-to» которому присваивается значение «imageRow-1». Получение нужного функционала, на основе этих атрибутов обеспечивает Bootstrap фреймворк, за счет собственной таблицы стилей и библиотеки javascript.

Листинг 2.4. Фрагмент кода метода Interpred

```
TagBuilder tagIndLi = new TagBuilder("li");
tagIndLi.MergeAttributes(new Dictionary<string, string>
   {
      {"class",imageRow>1?"":"active"},
      {"target-data","#myCarousel"+idCarousel},
      {"data-slide-to",Convert.ToString(imageRow-1)}
   });
```

Следующий фрагмент кода создает элемент HTML разметки «img» присваивая ему адрес расположения изображения на сервере добавляя имя изображения путем отделения его от остального содержимого строки из коллекции.

Листинг 2.5. Фрагмент кода метода Interpred

```
TagBuilder tagImg = new TagBuilder("img");
string[] split = call.Split(new string[] { "=" }, StringSplitOptions.None);
tagImg.MergeAttributes(new Dictionary<string, string>
{
   {"src", "/Content/img/" + (split[1].Replace(">",""))},
   {"class", "center-block img-responsive"},
   {"height","400"}
});
```

Этот фрагмент кода создает контейнер и помещает в него переменную хранящую элемент с изображением.

Листинг 2.6. Фрагмент кода метода Interpred

```
TagBuilder tagItemDiv = new TagBuilder("div");
tagItemDiv.AddCssClass((imageRow>1?"":"active") + " item");
tagItemDiv.InnerHtml += (tagImg.ToString());
```

Далее, в случае если это первое изображение в слайдере, создается общий контейнер в который помещаются все контейнеры с изображениями слайдера.

Листинг 2.7. Фрагмент кода метода Interpred

```
if (imageRow == 1)
{   tagContDiv = new TagBuilder("div");
   tagContDiv.AddCssClass("carousel-inner");}
tagContDiv.InnerHtml += tagItemDiv.ToString();
```

В следующем фрагменте кода, в случае если это первое изображение в слайдере, создается список в который помещаются все элементы списка отвечающие за индикаторы выбранного изображения.

Листинг 2.8. Фрагмент кода метода Interpred

```
   if (imageRow == 1)
   {
      tagIndOl = new TagBuilder("ol");
      tagIndOl.AddCssClass("carousel-indicators");
   }
   tagIndOl.InnerHtml += tagIndLi.ToString();}
```

Следующий код добавляет кнопки на слайдере, позволяющие поочередно перелистывать изображения, если создан слайдер.

Листинг 2.9. Фрагмент кода метода Interpred

```
else if(imageRow>0)
{
```

Листинг 2.9. Продолжение

```
TagBuilder tagLeftSpan = new TagBuilder("span");
tagLeftSpan.AddCssClass("glyphicon glyphicon-chevron-left");
TagBuilder tagLeft = new TagBuilder("a");
tagLeft.MergeAttributes(new Dictionary<string, string>
    {
        {"class","carousel-control left"},
        {"href","#myCarousel"+idCarousel},
        {"data-slide","prev"}
    });
tagLeft.InnerHtml += tagLeftSpan.ToString();
TagBuilder tagRightSpan = new TagBuilder("span");
tagRightSpan.AddCssClass("glyphicon glyphicon-chevron-right");
TagBuilder tagRight = new TagBuilder("a");
tagRight.MergeAttributes(new Dictionary<string, string>
    {
        {"class","carousel-control right"},
        {"href","#myCarousel"+idCarousel},
        {"data-slide","next"}
    });
tagRight.InnerHtml += tagRightSpan.ToString();
```

В следующем коде создается общий контейнер для слайдера, в параметрах которого указываются инструкции для фреймворка о том, как должен функционировать слайдер. Атрибут «data-ride» со значением «carousel» - говорит о том, что слайдер типа «карусель» будет автоматически перелистывать сообщения. Атрибут "data-interval" задает промежуток времени ожидание, перед очередной сменой изображения. Атрибут «data-pause» со значением «hover» означает, что автоматическая смена изображения отключается в случае, если курсор мыши наведен на слайдер.

Листинг 2.10. Фрагмент кода метода Interpred

```
TagBuilder tagCarousel = new TagBuilder("div");
tagCarousel.MergeAttributes(new Dictionary<string, string>
   {
      {"id","myCarousel"+idCarousel},
      {"class", "carousel slide"},
      {"data-interval", "3000"},
      {"data-ride", "carousel"},
      {"data-pause", "hover"}
   });
tagCarousel.InnerHtml += tagIndOl.ToString() + tagContDiv.ToString() + tagLeft.ToString() + tagRight.ToString();
      tag.InnerHtml += tagCarousel.ToString();
      imageRow = 0;}
```

Этот участок кода существует для работы с контентом не содержащим изображение. Он просто добавляет такой контент в тег : «if (!call.Contains("<img")){tag.InnerHtml += " " + call + " ";}}».

Оканчивается метод тем, что возвращает экземпляр класса, которому ASP.NET MVC «доверяет» генерирование разметки. Этот объект содержит всю созданную выше разметку страницы: «return new MvcHtmlString(tag.ToString());».

ГЛАВА 3. ИСПЫТАНИЯ

3.1 Программа и методика испытаний

Объектом испытаний является ранее спроектированная и разработанная «Автоматизированная система поддержки организационной деятельности клуба единоборств». По сфере применения данная информационная система является информационно-справочной системой.

Целью испытаний информационной системы являются проверка ее работоспособности, а так же ее соответствие указанным в первой главе требованиям к автоматизированной системе.

Полный перечень требований к автоматизированной системе перечислен в третьем разделе первой главы, однако ключевыми пунктами, требующими установки соответствия, являются следующие:

- возможность предоставить посетителям отчетности о деятельности клуба;

- возможность информировать посетителей о грядущих событиях;

- предоставление информации о единоборстве: технике, правилам и другой подобной информации;

- возможность добавления новых информационных статей, не владея навыками программирования;

- возможность мониторинга и редактирования данных о спортсменах и их родственниках;

- возможность регистрации и авторизации в системе;

- разграничение доступа к возможностям сайта, путем разделения ролей авторизованных пользователей.

Исходя из приведенных выше требований - был разработан следующий алгоритм для проведения испытаний автоматизированной системы.

1. Тестирование возможностей системы для неавторизованного пользователя.

1.1. Запустить информационную систему.

1.2. Нажать на ссылку «Новости» расположенной на основном горизонтальном меню в верхней части страницы.

1.3. Убедиться в наличии списка новостей.

1.4. Нажать на ссылку любой из перечисленных новостей.

1.5. Убедиться в работе страницы с подробностями о новости.

1.6. Нажать на ссылку «Обучающие материалы» расположенной на основном горизонтальном меню в верхней части страницы.

1.7. Убедиться в наличии списка обучающих материалов.

1.8. Нажать на ссылку любой из перечисленных статей с обучающими материалами.

1.9. Убедиться в работе страницы с подробностями статьи.

1.10. Нажать на ссылку «Анонс мероприятий» расположенной на основном горизонтальном меню в верхней части страницы.

1.11. Убедиться в наличии списка статей с анонсами.

1.12. Нажать на ссылку любой из перечисленных статей.

1.13. Убедиться в работе страницы с подробностями статьи.

1.14. Нажать на ссылку «Деятельность клуба» расположенной на основном горизонтальном меню в верхней части страницы.

1.15. Убедить в работе страницы с расписанием занятий.

1.16. Нажать на ссылку «Тренеры».

1.17. Убедиться в работе страницы с перечнем тренеров.

1.18. Попробовать перейти по ссылке на контроллер доступный только для авторизованного пользователя с правами модератора или администратора, добавив в адресной строке к адресу сайта «/Roles».

1.19. Нажать на ссылку «Войти» расположенной на основном горизонтальном меню в верхней части страницы.

1.20. Нажать на ссылку «Регистрация» расположенную под формой для авторизации.

1.21. Заполнить форму регистрации корректными данными: ввести логин, пароль, повторно ввести пароль в поле с подтверждение пароля.

1.22. Нажать на кнопку «Зарегистрироваться».

1.23. Повторить пункты 1.20 – 1.22 для создания второй учетной записи, которая впоследствии будет удалена.

1.24. Убедиться в том, что сайт перенаправляет пользователя на страницу авторизации.

2. Тестирование возможностей системы для авторизованного пользователя с правами «Администратор».

2.1. Авторизоваться на сайте с помощью заранее известной учетной записи, имеющей права доступа администратора, заполнив поля формы на странице авторизации: логин, пароль.

2.2. Повторить действия из пункта 1.18.

2.3. Убедиться в том, что теперь пользовать имеет доступ к странице настройки доступа.

2.4. Найти в списке учетных записей запись, созданную в пункте 1.23.

2.5. Для созданной записи выбрать в выпадающем списке права доступа «Модератор» и нажать на кнопку применить.

2.6. Нажать на кнопку в виде красного креста возле появившейся надписи «Модератор» у созданной учетной записи, для удаления прав доступа.

2.7. Нажать на кнопку «Удалить», для удаления учетной записи.

2.8. Найти учетную запись, созданную при выполнении пунктов 1.20 – 1.22.

2.9. Для найденной записи выбрать в выпадающем списке права доступа «Модератор» и нажать на кнопку применить.

2.10. Нажать на ссылку «Выйти» расположенной на основном горизонтальном меню в верхней части страницы.

3. Тестирование возможностей системы для авторизованного пользователя с правами «Модератор».

3.1. Авторизоваться на сайте с помощью, созданной во время выполнения пунктов 1.20 – 1.22 учетной записи, заполнив поля формы на странице авторизации: логин, пароль.

3.2. Нажать на ссылку «Новости» расположенной на основном горизонтальном меню в верхней части страницы.

3.3. Нажать на появившуюся после авторизации с правами доступа кнопку создать для добавления статьи.

3.4. Перетянуть тестовые изображения в поле для добавления изображений.

3.5. Заполнить поля форма, в соответствии с требованиями указанными серым текстом посредством тега «placeholder».

3.6. Выделить три произвольных участка текста, не содержащие кода изображения и нажать на кнопки «жирный», «курсивный», «цитирование» для каждого участка текста по одной кнопке соответственно.

3.7. Нажать на кнопку «Создать» для создания статьи.

3.8. После перенаправления на страницу со списком статей, убедиться в том, что новая статья добавлена.

3.9. Нажать на ссылку добавленной статьи.

3.10. Убедиться в том, что добавленная статья отображается корректно.

3.11. Нажать на ссылку «Анонс мероприятий» расположенной на основном горизонтальном меню в верхней части страницы.

3.12. Повторить выполнение пунктов 3.3 – 3.10.

3.13. Нажать на ссылку «Обучающие материалы» расположенной на основном горизонтальном меню в верхней части страницы.

3.14. Повторить выполнение пунктов 3.3 – 3.10.

3.15. Нажать на ссылку «Деятельность клуба» расположенной на основном горизонтальном меню в верхней части страницы.

3.16. Заполнить, появившуюся после авторизации с правами доступа, форму, расположенную под расписание: день недели, время, название группы и нажать на кнопку добавить, чтобы дополнить расписание.

3.17. Нажать на кнопку в виде красного креста возле появившейся новой записи в расписании, для ее удаления.

3.18. Нажать на ссылку «Тренеры».

3.19. Нажать на появившуюся после авторизации с правами доступа кнопку «Добавить тренера».

3.20. В открывшейся странице заполнить форму для добавления тренера: Имя, Фамилия, Отчество, Номер телефона и нажать кнопку «Добавить».

3.21. После перенаправления на страницу со списком тренеров, убедиться в том, что новый тренер добавлен.

3.22. Заполнить, появившуюся после авторизации с правами доступа, форму, расположенную под списком тренеров, для добавления новой спортивной группы, указать: вид единоборства, постфикс и нажать на кнопку добавить.

3.23. Убедиться в том, что добавленная группа появилась в списке групп под формой.

3.24. Для добавленного в пунктах 3.19-3.21 тренера в выпадающем списке выбрать группу, добавленную в пункте 3.22 и нажать на кнопку «Добавить», чтобы привязать группу к тренеру.

3.25. Нажать на кнопку в виде красного креста возле появившейся у тренера привязанной группы, для того, чтобы отвязать ее.

3.26. Нажать на кнопку «Удалить», напротив добавленного тренера, чтобы удалить его.

3.27. Нажать на кнопку в виде красного креста возле добавленной в пункте 3.22 группы, которая расположена под формой в списке с группами, чтобы удалить ее.

3.28. Нажать на появившуюся после авторизации с правами доступа ссылку «Ученики».

3.29. Убедить в работе страницы со списком спортсменов.

3.30. Нажать на кнопку «Добавить спортсмена».

3.31. В открывшейся странице заполнить форму для добавления спортсмена: Имя, Фамилия, Отчество, Номер телефона и нажать кнопку «Добавить».

3.32. После перенаправления на страницу со списком спортсменов, убедиться в том, что новый спортсмен добавлен.

3.33. Нажать на кнопку «Удалить», напротив добавленного спортсмена, чтобы удалить его.

3.34. Нажать на появившуюся после авторизации с правами доступа ссылку «Родственники».

3.35. Убедить в работе страницы со списком родственников.

3.36. Нажать на кнопку «Добавить родственника».

3.37. В открывшейся странице заполнить форму для добавления родственника: Имя, Фамилия, Отчество, Номер телефона и нажать кнопку «Добавить».

3.38. После перенаправления на страницу со списком родственников, убедиться в том, что новый родственник добавлен.

3.39. Нажать на кнопку «Удалить», напротив добавленного родственника, чтобы удалить его.

3.2 Испытание информационной системы

Начать проведение испытаний следует с тестирования возможностей системы для неавторизованного пользователя.

Первым делом запускаем информационную системы, для этого в адресной строке веб браузера вводим адрес страницы. После того как главная страница сайта загрузилась, мы видим список включающий в себя все новости, анонсы и обучающие статьи сайта. Работа главной страницы изображена на рис. 3.1.

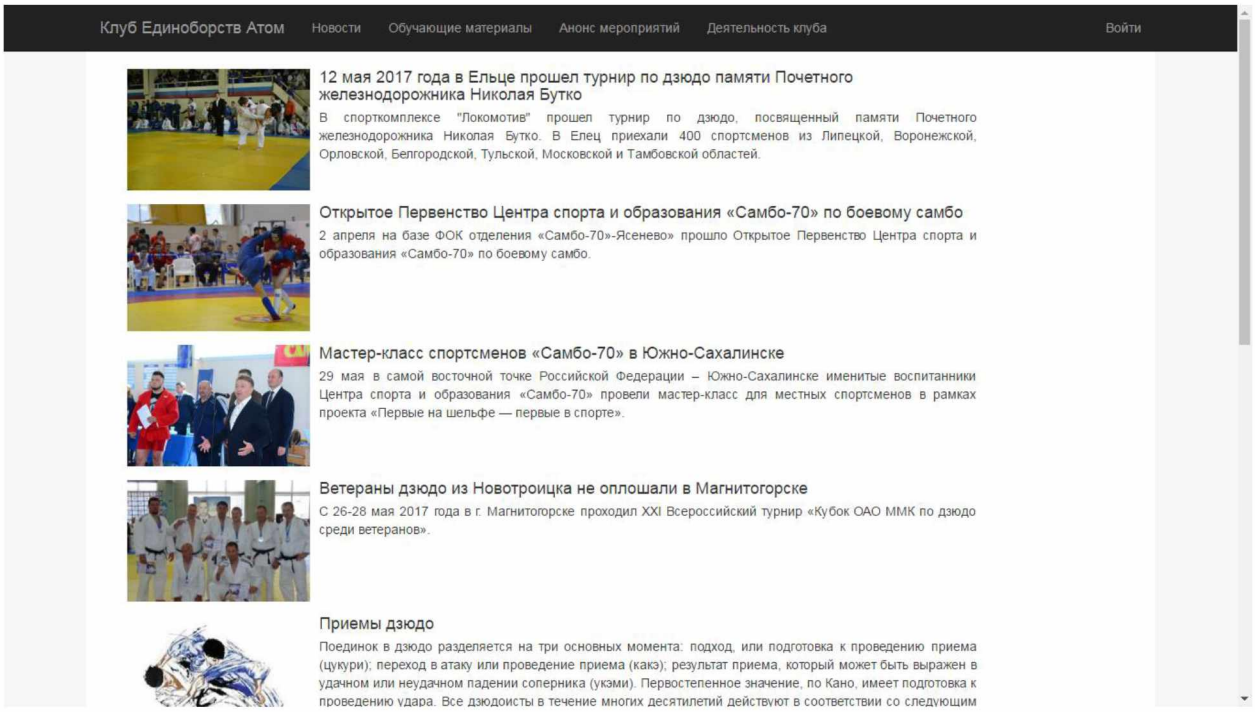

Рис. 3.1. Главная страница

Для того чтобы проверить как сайт реализует возможность неавторизованного пользователя ознакомиться отчетностью о текущей деятельности клуба переходим по ссылке «новости» расположенной в меню сайта. Перейдя по ссылке, мы попадаем в раздел, который отображает список новостей, проиллюстрированный рис. 3.2.

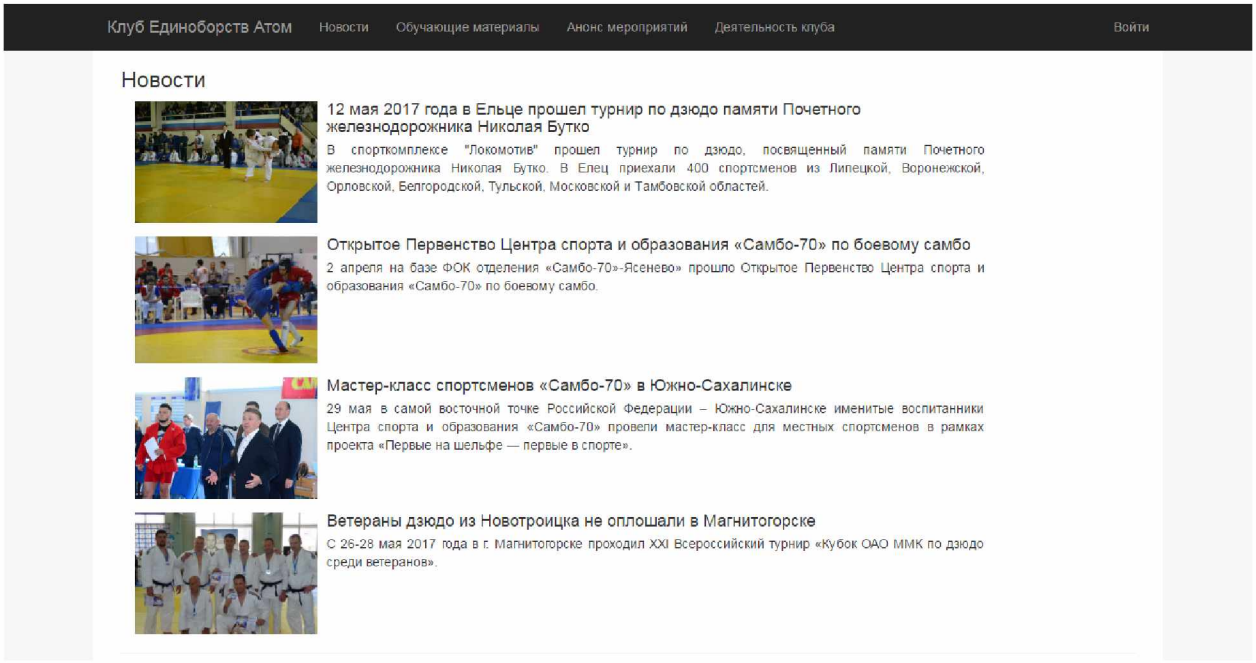

Рис. 3.2. Страница «Новости»

Выполняя следующий шаг в тестировании отчетности, выберем новость «Ветераны дзюдо из Новотроицка не оплошали в Магнитогорске», для рассмотрения подробной информации. Выполняем этот шаг, кликнув по изображению выбранной новости. Результат выполнения этого действия проиллюстрирован на рис. 3.3.

Рис. 3.3. Страница с выбранной новостью

Далее убедимся в том, что система предоставляет пользователям возможность ознакомиться со списком статей, содержащих обучающие материалы. Для этого переходим по ссылке «Обучающие материалы». С результатами этого действия можно ознакомиться на рис. 3.4.

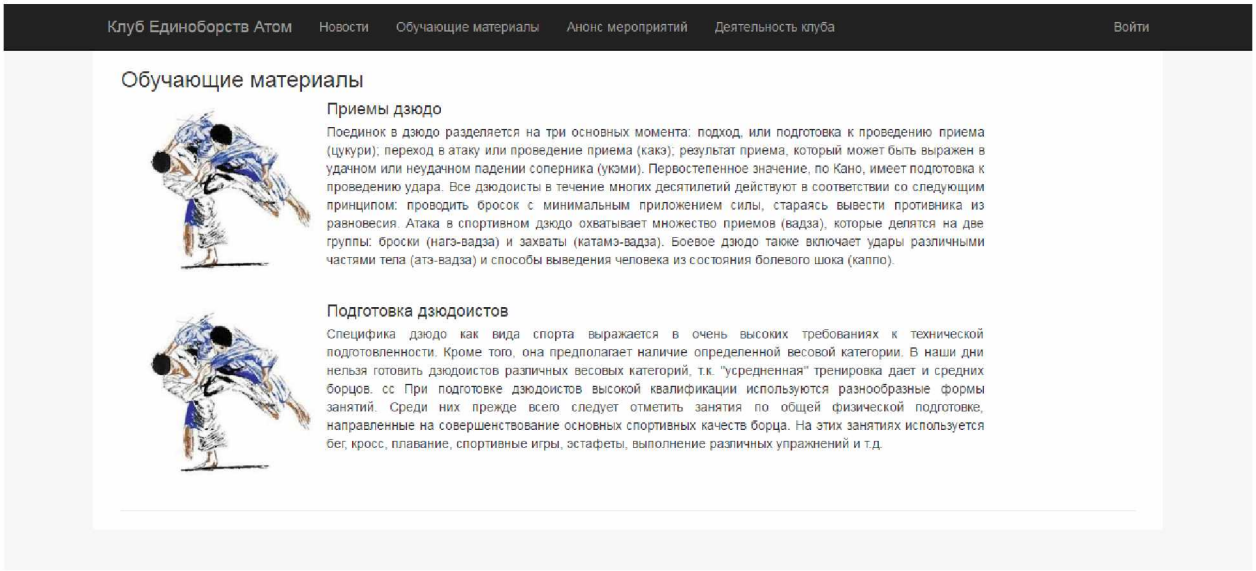

Рис.3.4. Страница «Обучающие материалы»

Результаты запуска статьи отображены на рис. 3.5.

Рис. 3.5. Страница с обучающей статьей

Следующим шаг - проверка реализации возможности анонсирования мероприятий. Для этого переходим по ссылке «Анонс мероприятий».

Перейдя по ссылке, мы попадаем в раздел, который содержит перечень анонсируемых мероприятий, проиллюстрированный рис. 3.6.

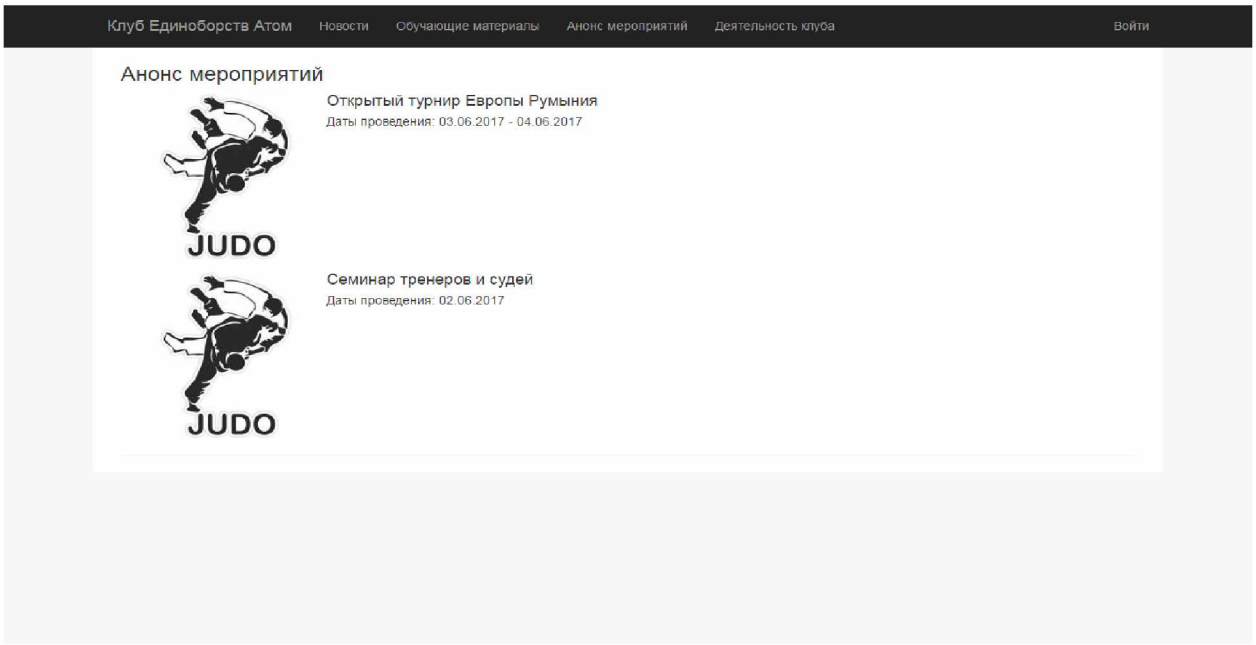

Рис. 3.6. Страница «Анонс мероприятий»

Результаты запуска статьи отображены на рис. 3.7.

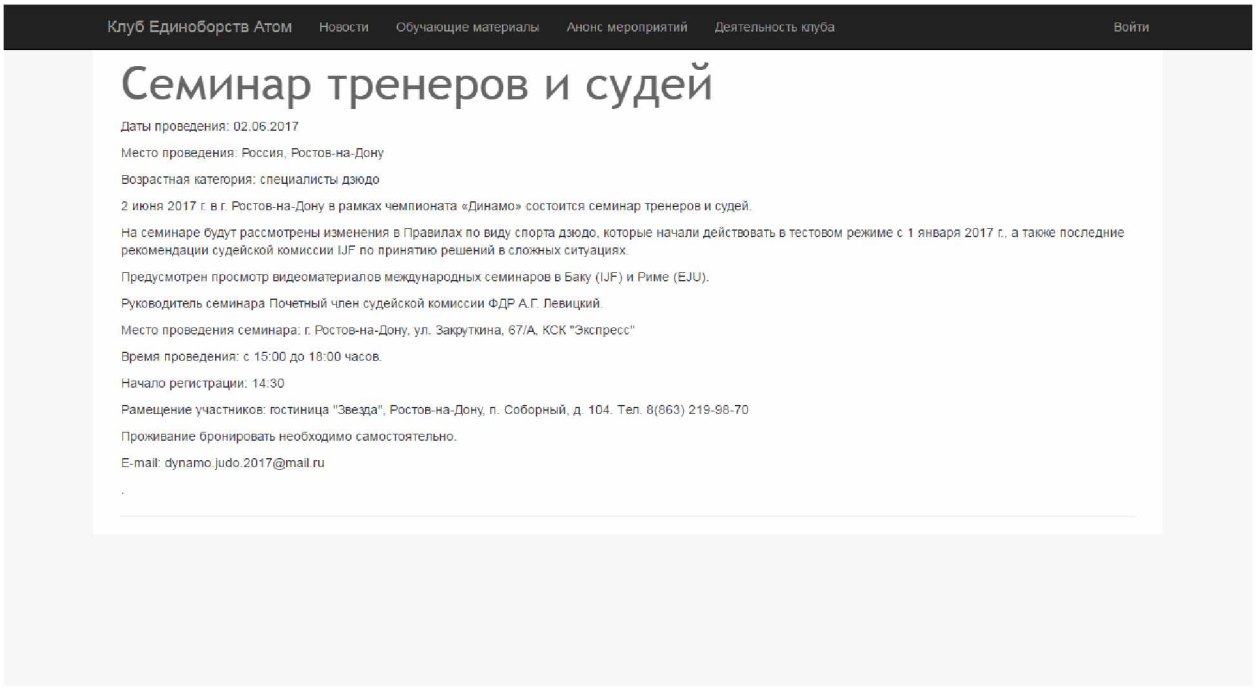

Рис. 3.7. Страница с Анонсом мероприятий

Далее переходим по ссылке «Деятельность клуба». На этой странице есть две ссылки, одна из них ведет к странице с расписанием, которое отображено на рис. 3.8.

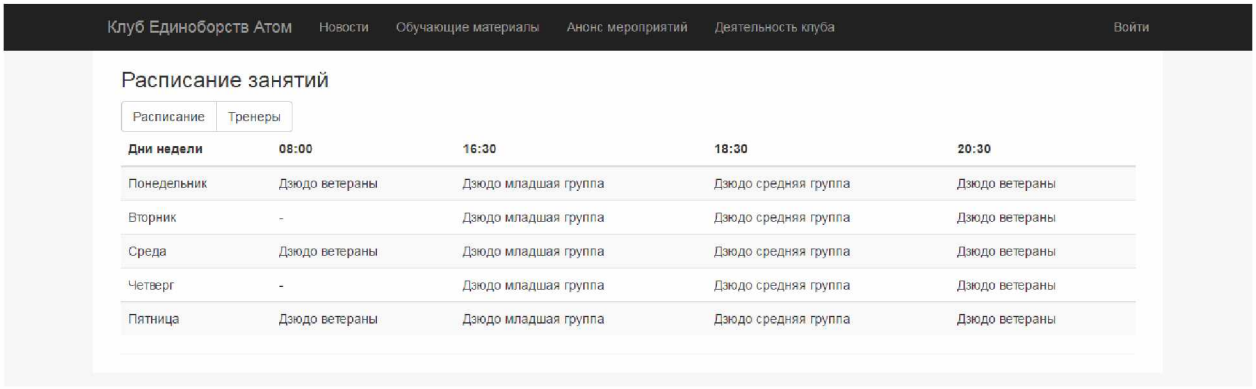

Рис.3.8. Расписание занятий

Вторая ссылка ведет на страницу с перечнем тренеров – рис. 3.9.

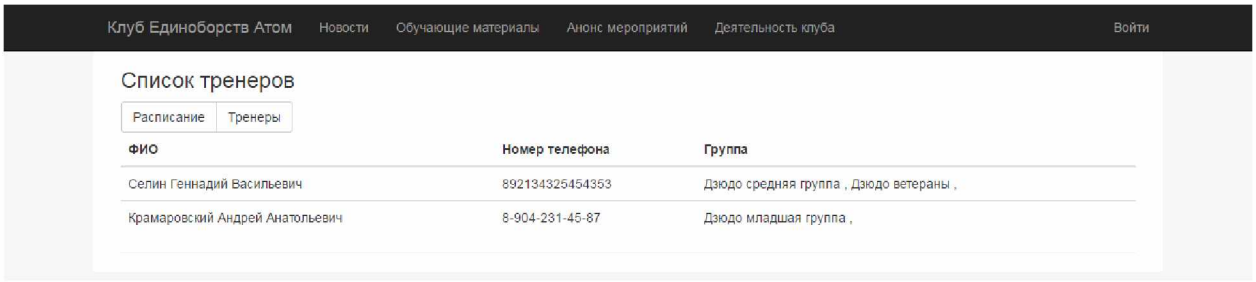

Рис. 3.9. Страница «Список тренеров»

Прежде, чем протестировать возможности предоставляемые пользователям с правами доступа «Модератор», необходимо создать учетную запись. Для этого переходим на страницу авторизации – рис. 3.10.

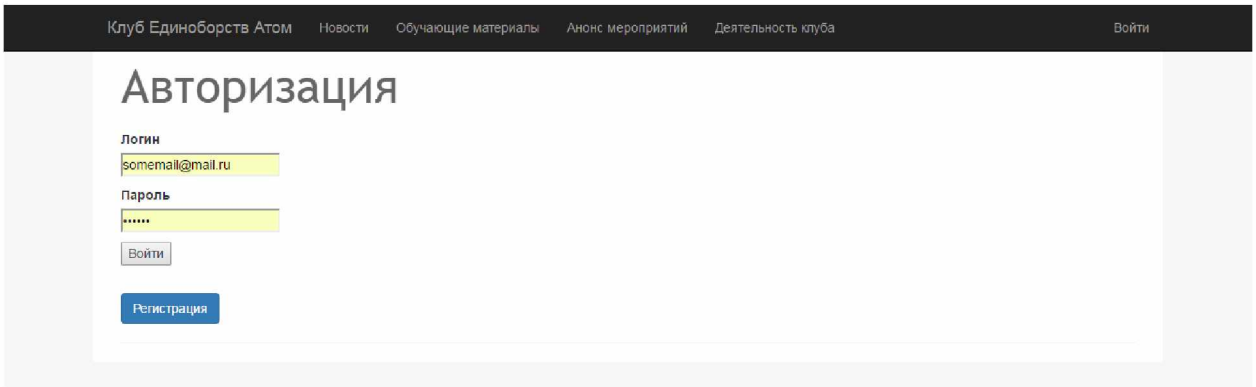

Рис. 3.10. Страница для авторизации

После этого переходим на форму регистрации, отображенную на рис. 3.11.

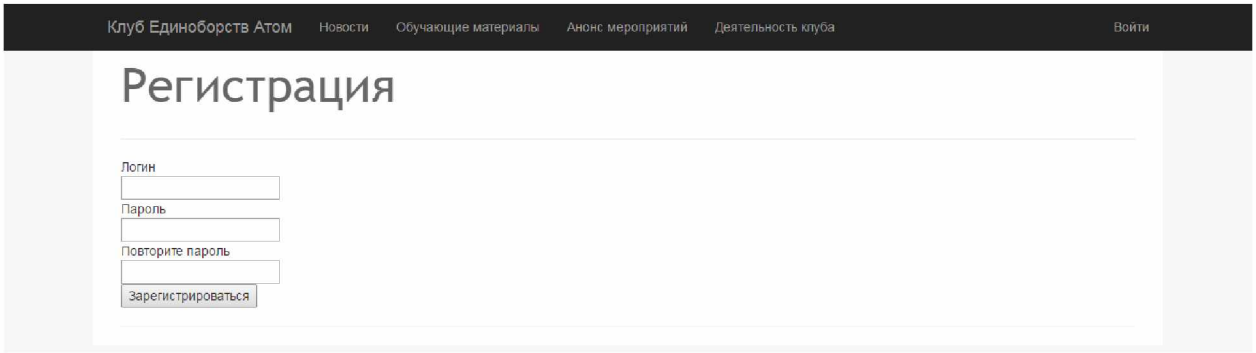

Рис. 3.11. Страница для регистрации

Далее заполняем поля форма. В поле логин вводим название создаваемой учетной записи – «Test», в поле пароль – «12345test» - рис. 3.12.

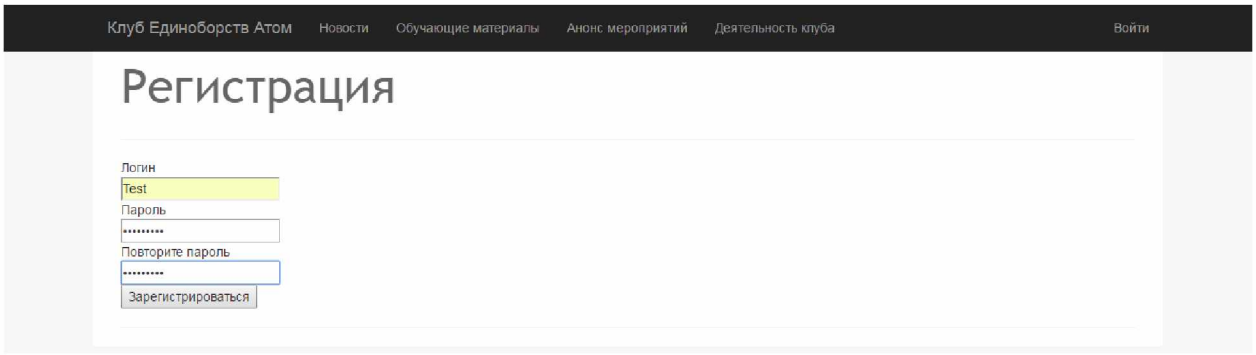

Рис. 3.12. Регистрация новой учетной записи

Создаем вторую учетную запись, как показано на рис. 3.13.

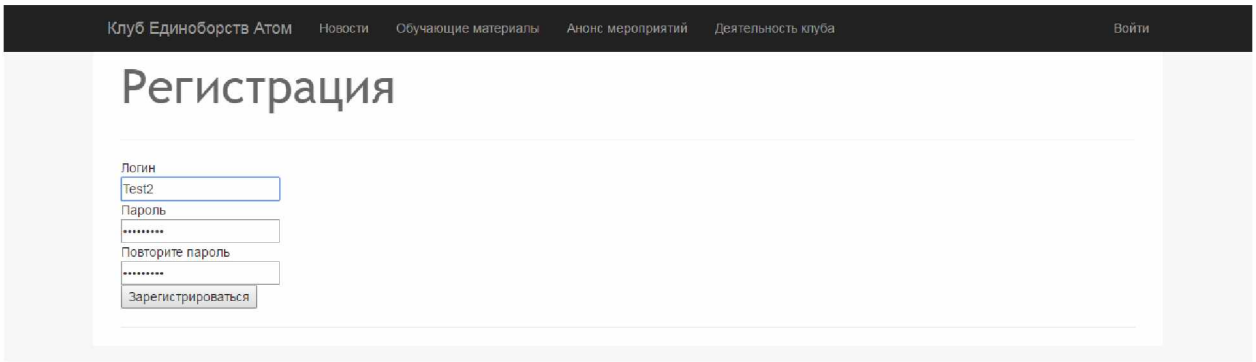

Рис. 3.13. Регистрация второй учетной записи

Теперь нам нужно авторизоваться с помощью заранее добавленной учетной записи с правами администратора, что изображено на рис. 3.14. Таким образом мы протестируем возможности работы с учетными записями, для пользователя с правами «Администратор», а также присвоим ранее созданной учетной записи права модератора.

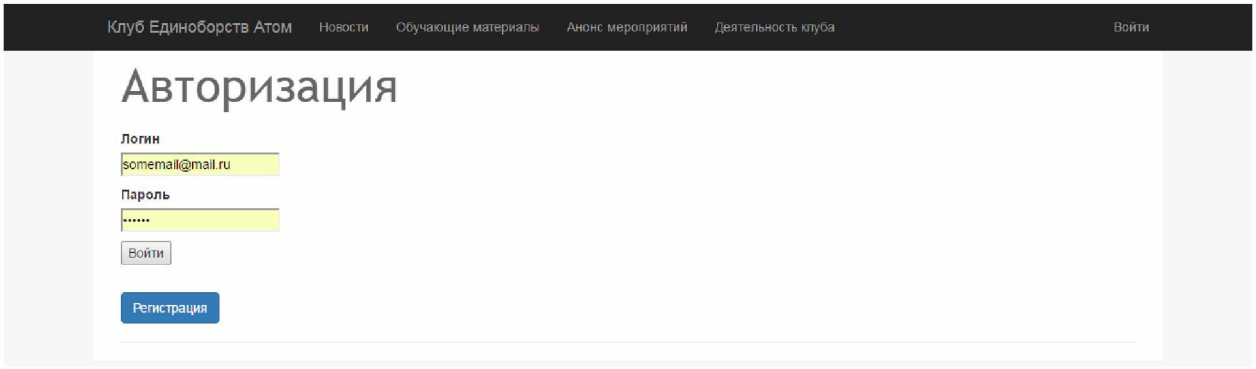

Рис. 3.14. Авторизация с правами администратора

После успешной авторизации переходим по появившейся ссылке «Аккаунты» и попадаем на страницу настройки доступа – рис.3.15.

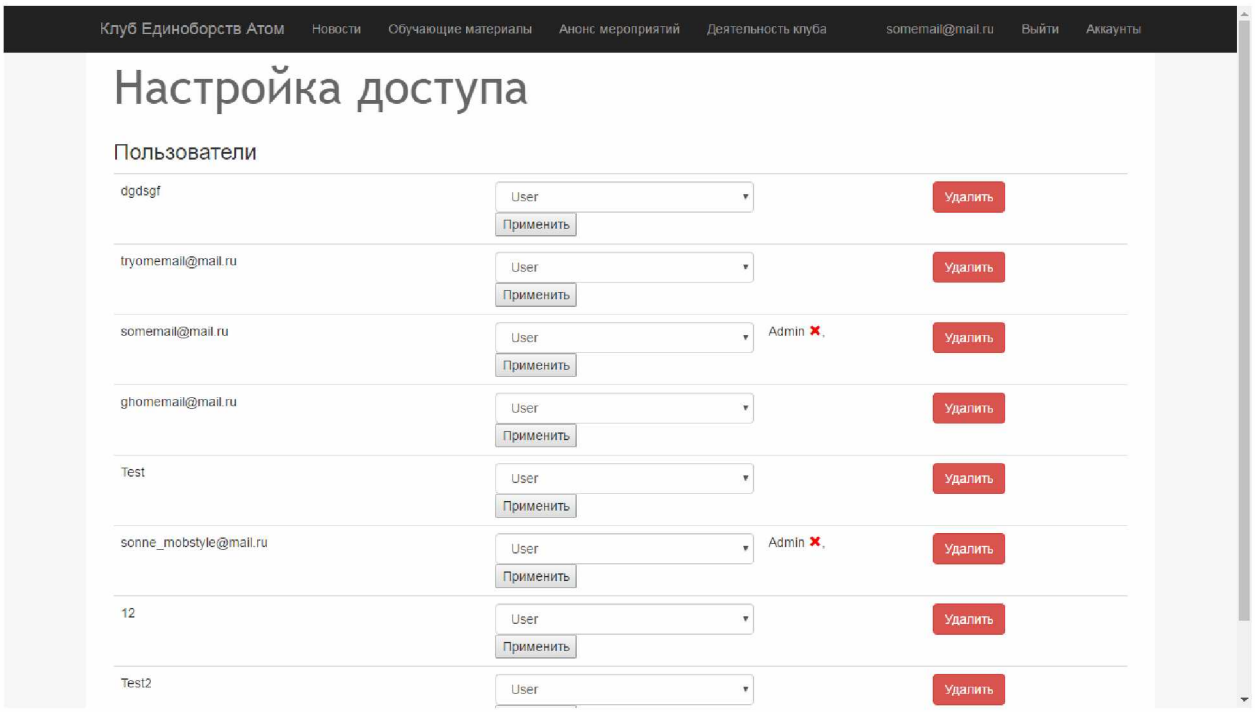

Рис. 3.15. Страница «Настройка доступа»

Чтобы проверить работоспособность страницы выбираем в выпадающем меню для аккаунта «Test2» права доступа модератор и жмем применить для назначения прав. Далее жмем на «крестик» возле появившейся надписи, для откреплення прав доступа от учетной записи. После этого жмем на кнопку удалить напротив аккаунта, для его удаления. После этого добавляет права «Moderator» для учетной записи «Test», чтобы протестировать возможность работы с системой пользователя с правами

модератора. Результат выполнения манипуляций с аккаунтами отображен на рис.3.16.

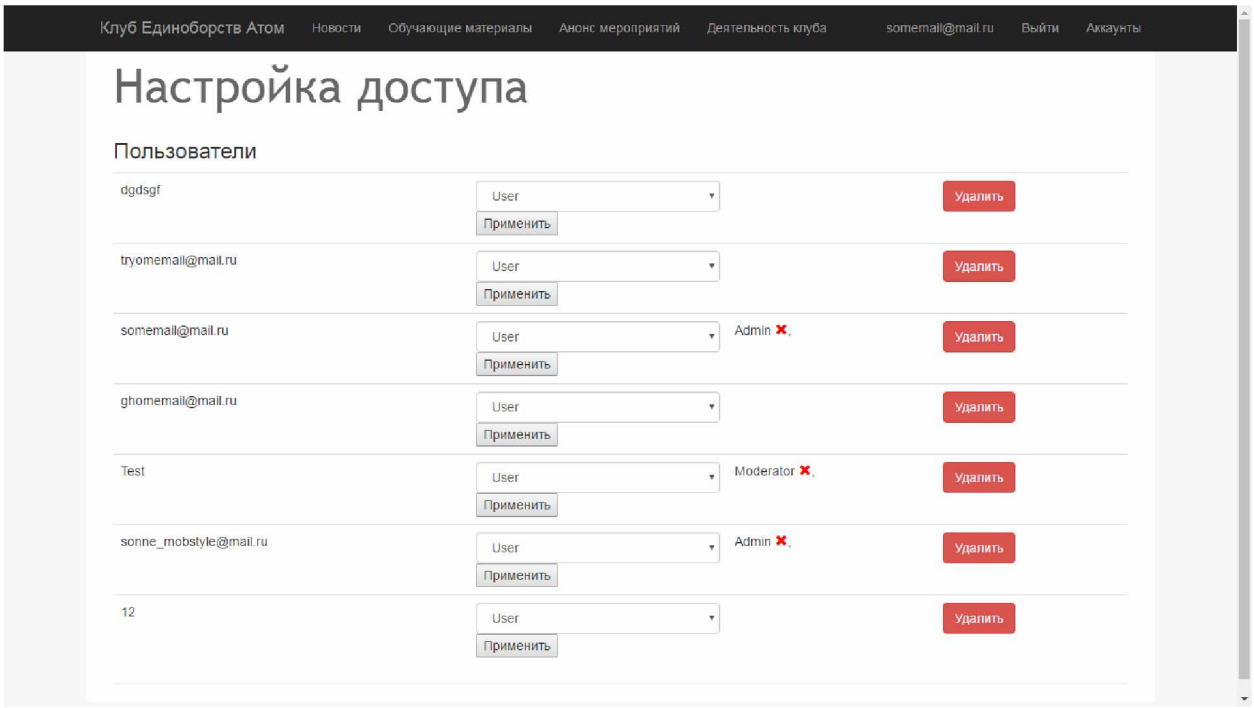

Рис. 3.16. Управление учетными записями

Далее возвращаемся на страницу «Новости» и видим, что появилась новая кнопка «Создать», предназначенная для создания новой кнопки. Ознакомить с иллюстрацией данного действия можно на рис. 3.17.

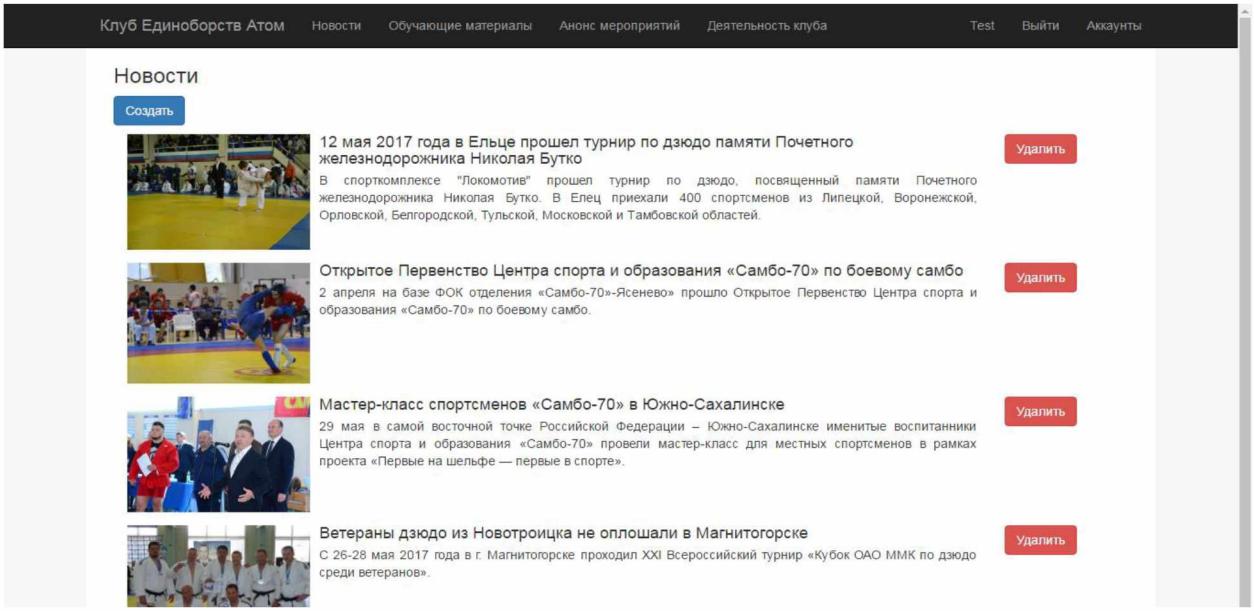

Рис. 3.17. Страница «Новости» с правами модератора

Жмем на кнопку «Создать», после чего попадаем на страницу «Добавление статьи». Чтобы создать статью заполняем поля «Название», «Изображение иконки», «Содержание». Добавляем изображения путем «перетягивания» из директории на диске в область для изображений на сайте, после чего указываем код изображения в необходимых местах. Иллюстрация действий – рис. 3.18.

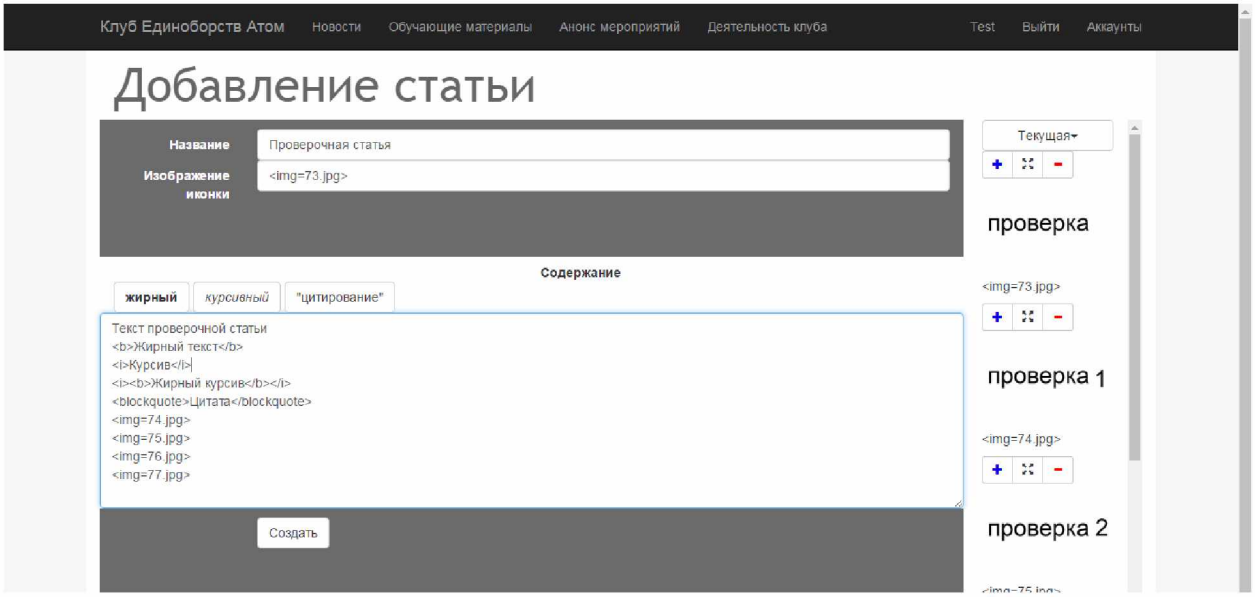

Рис. 3.18. Добавление статьи

После того, как мы создали статью, сайт перенаправляет нас на страницу с перечнем статей, где новая статья возглавляет список – рис. 3.19.

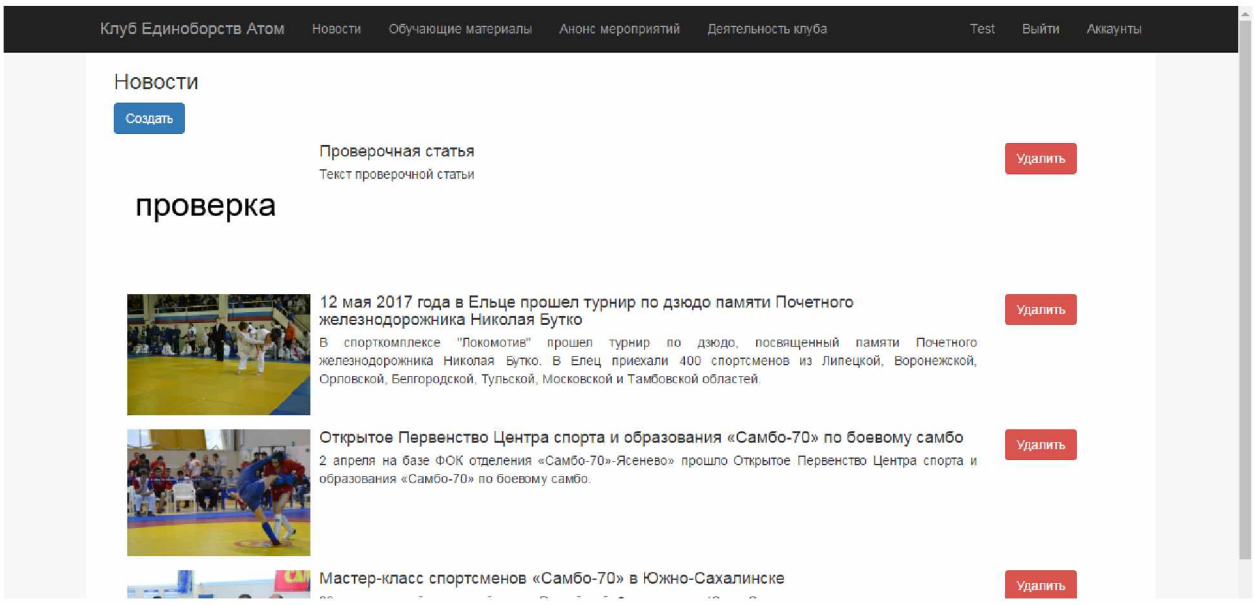

Рис. 3.19. Результат добавления статьи

Жмем на ссылку статьи, чтобы убедить в том, что все отображается как нужно. Результат показан на рис. 3.20.

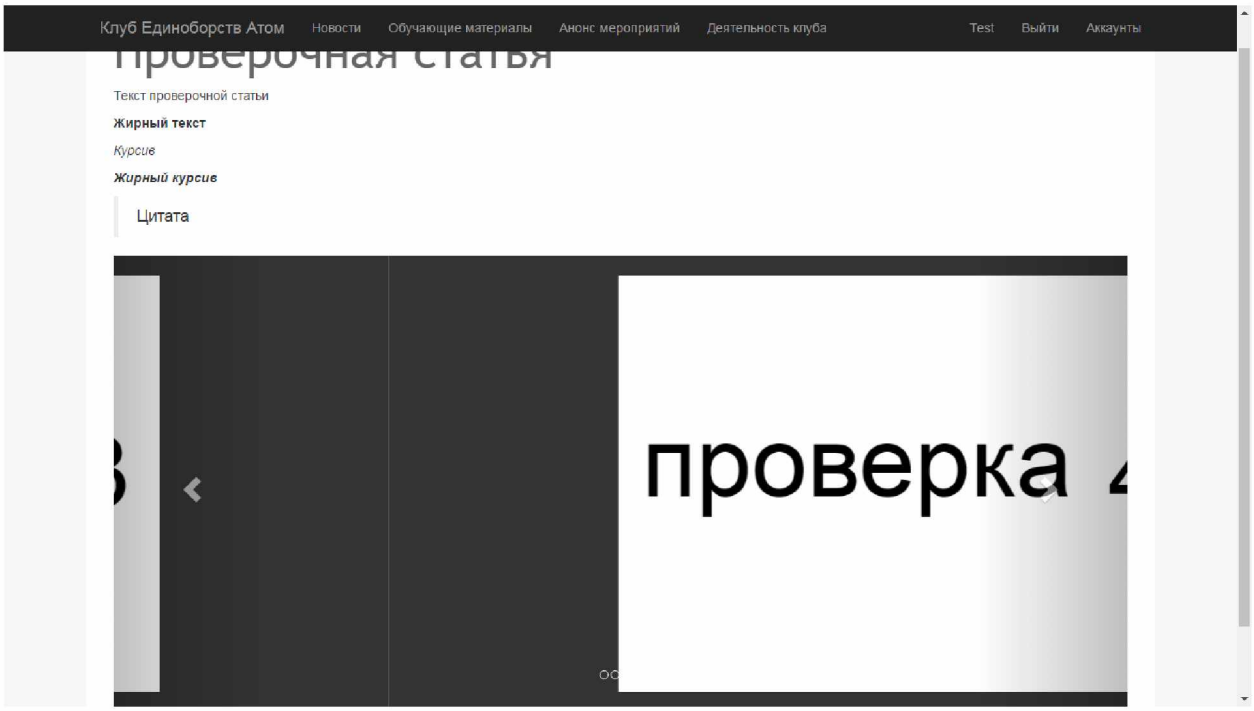

Рис. 3.20. Страница добавленной статьи

Результат добавления статьи в раздел «Обучающие мероприятия » показан на рис. 3.21.

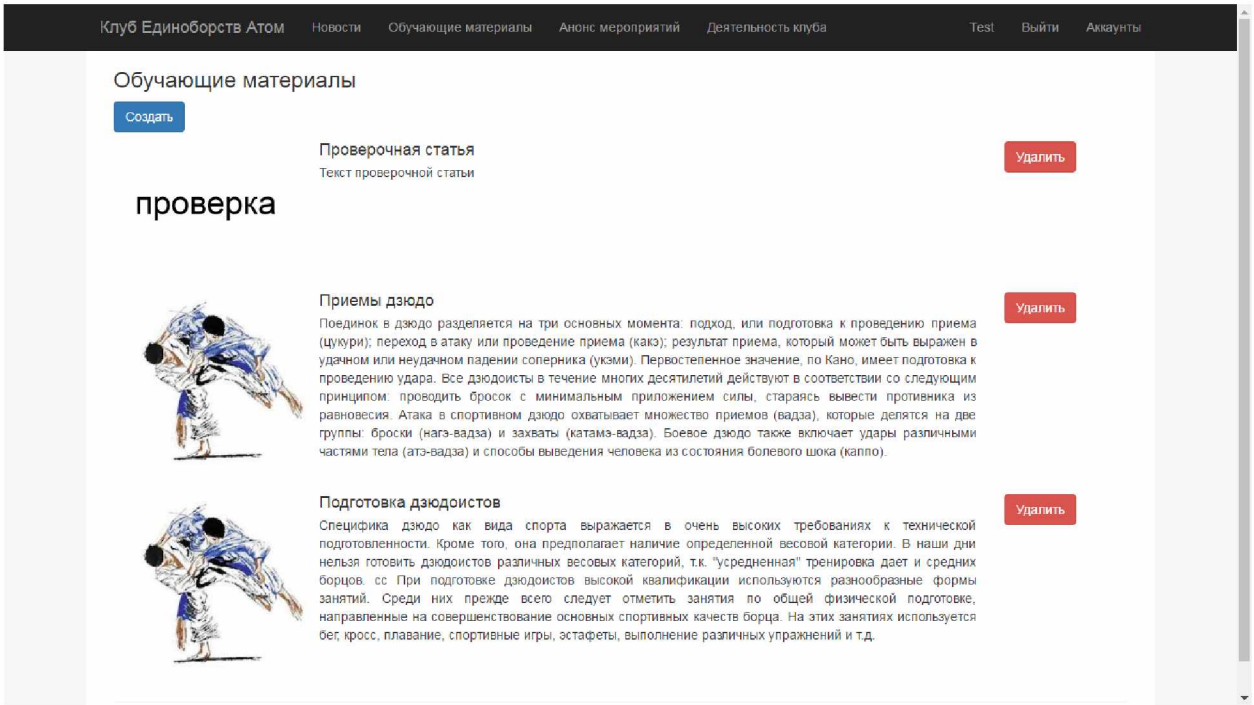

Рис. 3.21. Добавление статьи в раздел «Обучающие материалы»

Результат добавления статьи в раздел «Анонс мероприятий » показан на рис. 3.22.

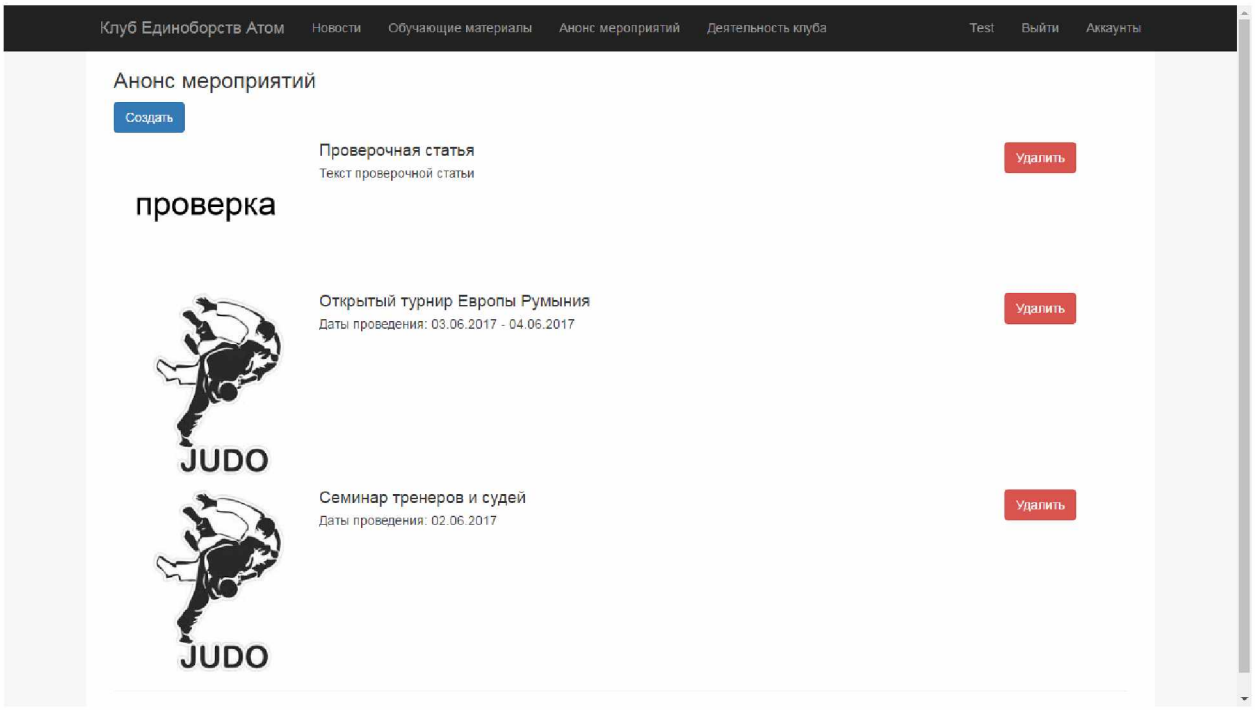

Рис. 3.22. Добавление статьи в раздел «Анонс мероприятий»

Теперь переходим к тестированию возможности модерирования раздела, «Деятельность клуба». После перехода на страницу с расписанием на рис. 3.23 видим, что теперь расписание доступно для редактирования.

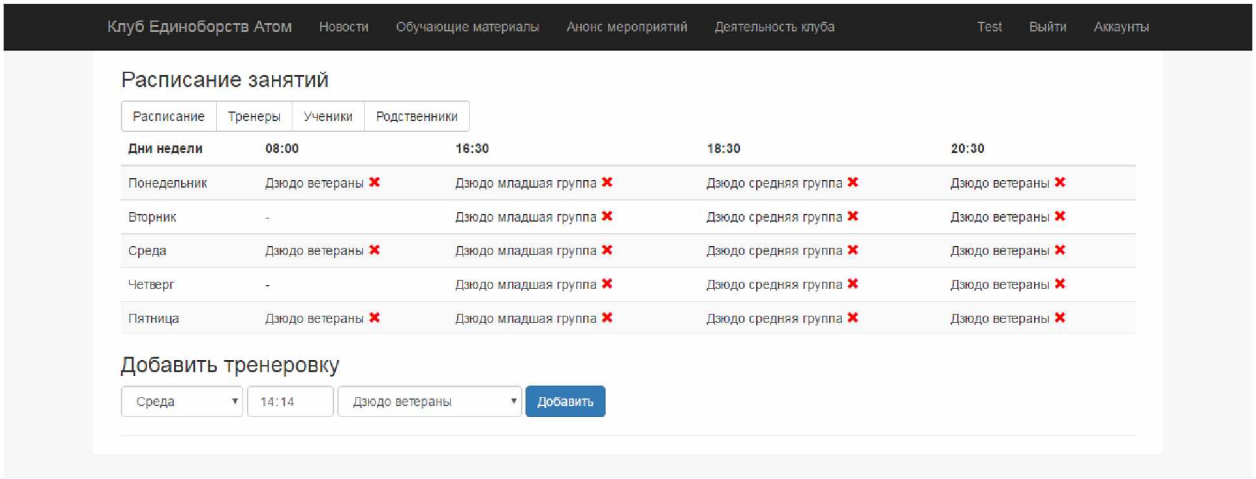

Рис. 3.23. Страница «Расписание занятий» с правами модератора

Для того чтобы протестировать возможность добавления тренировки, вводим в форму «день недели», «время» и выбираем нужную группу, после чего жмем добавить. Результат добавления отображен на рис. 3.24.

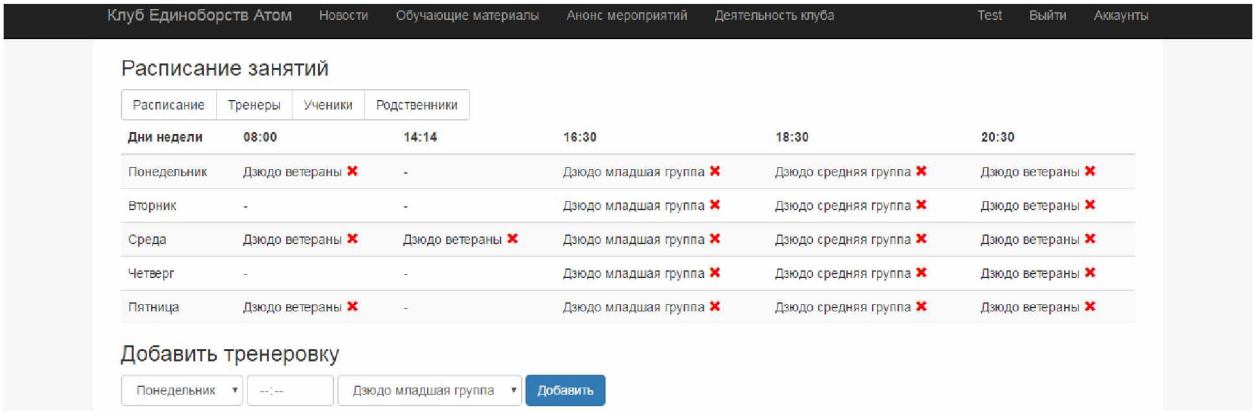

Рис. 3.24. Добавление пункта в расписание

Следующим шагом проверим возможность добавить в систему еще одного тренера. Для этого переходим на страницу «Тренеры» и жмем на появившуюся кнопку «Добавить тренера». После этого заполняем форму данными, как показано на рис. 3.25 и жмем на кнопку «Добавить».

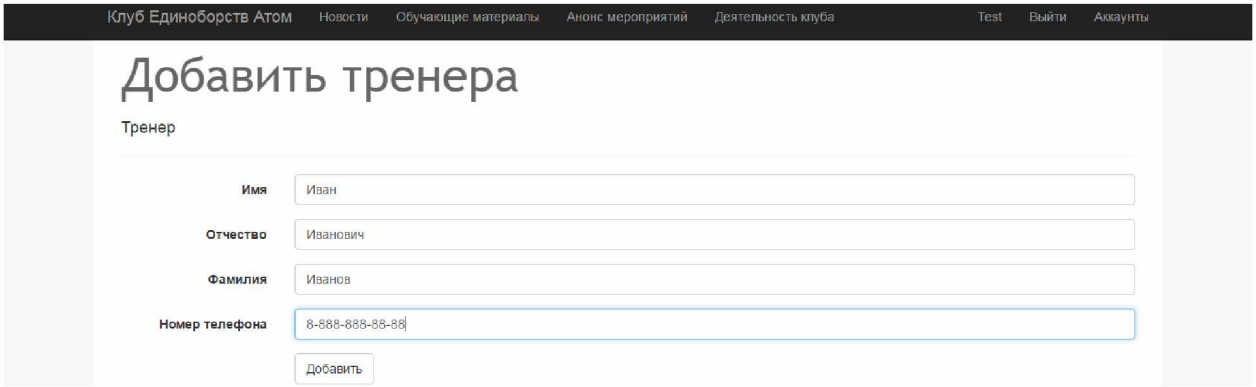

Рис. 3.25. Добавление тренера

Далее проверяем возможность добавить новую группу. Для этого нужно заполняем форму под списком и жмем на кнопку «Добавить», как показано на рис. 3.26.

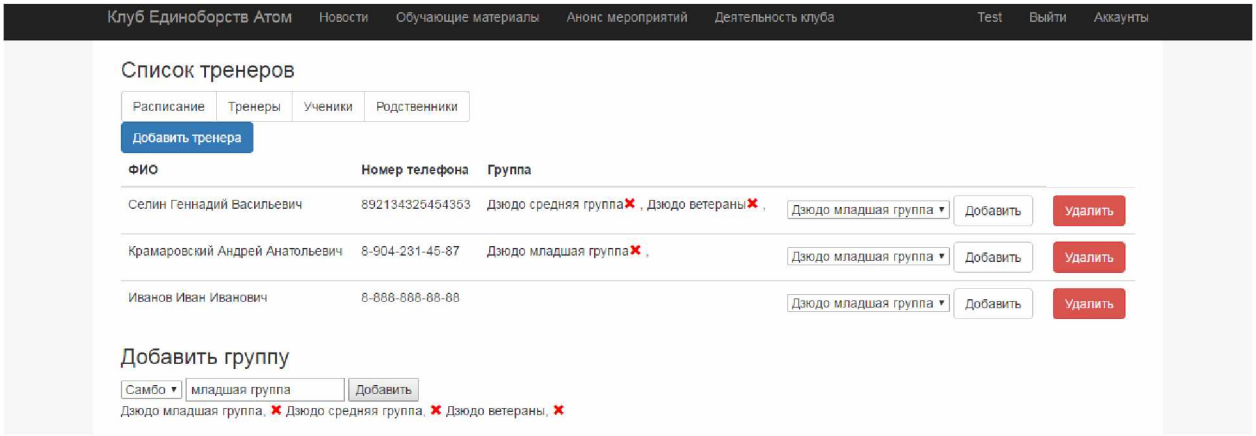

Рис. 3.26. Добавление группы

Далее в выпадающем списке возле добавленного тренера выбираем созданную группу и жмем на кнопку «Добавить», как показано на рис. 3.27.

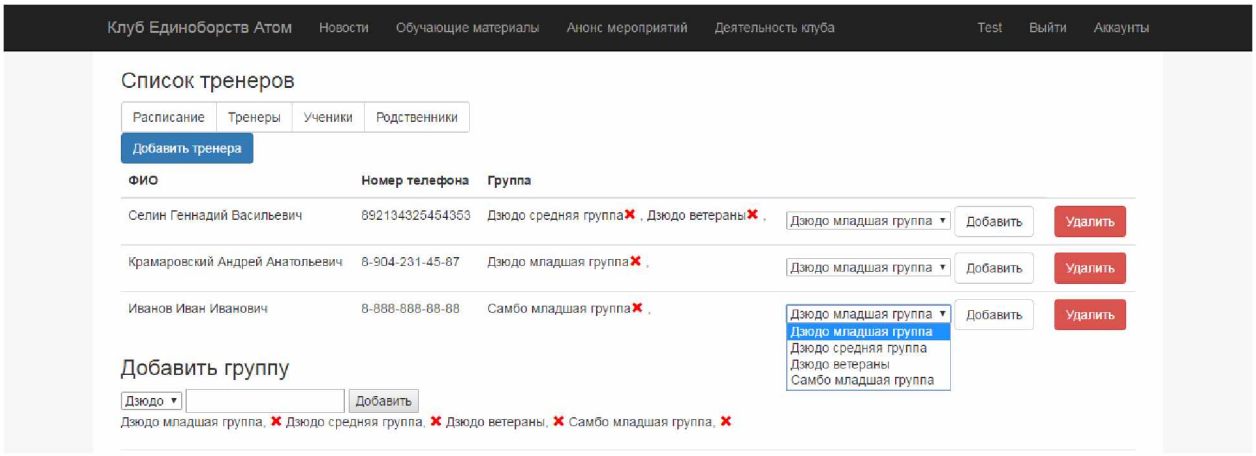

Рис. 3.27. Прикрепление группы к тренеру

Для пользователя с правами доступа модератора имеется возможность просматривать и пополнять списки спортсменов и их родителей. Для просмотра списка спортсменов нужно нажать на ссылку «ученики». Результат отображен на рис. 3.28.

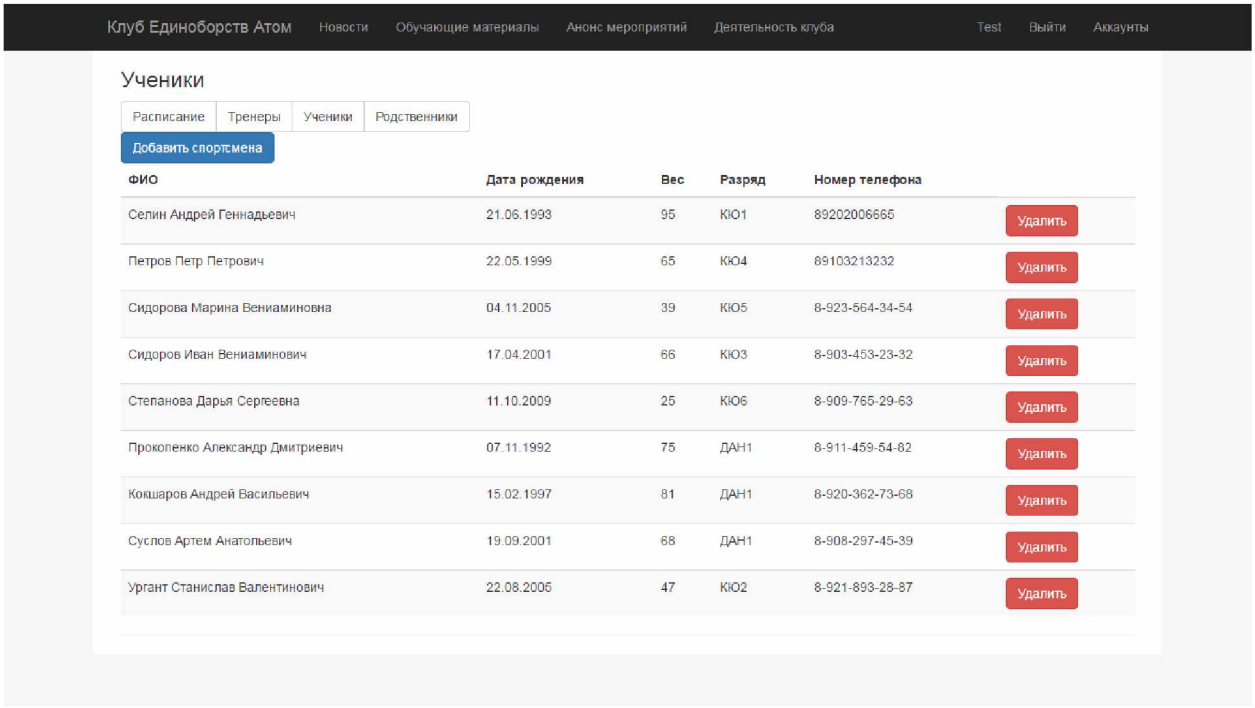

Рис. 3.28. Страница «Ученики»

Проверяем возможность добавления спортсмена, нажав на кнопку «Добавить спортсмена», заполнив форму как показано на рис. 3.29 и нажав на кнопку «Добавить».

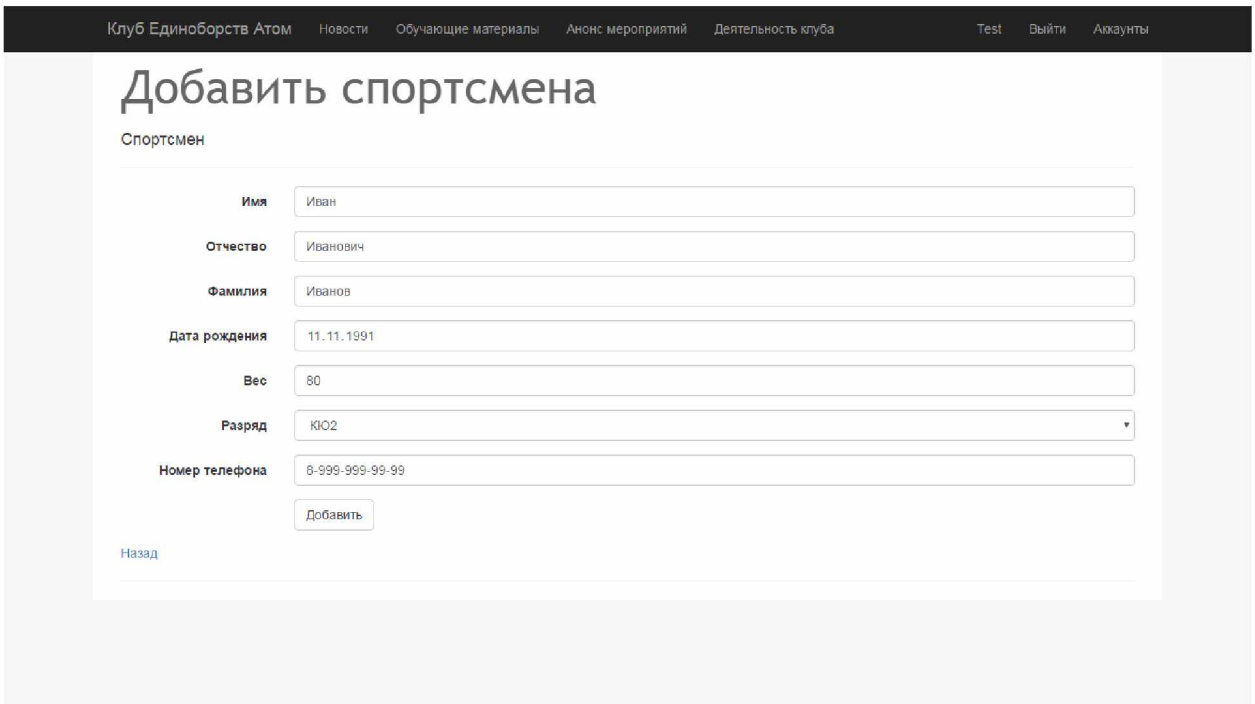

Рис. 3.29. Добавление спортсмена

Результат добавления спортсмена показан на рис. 3.30.

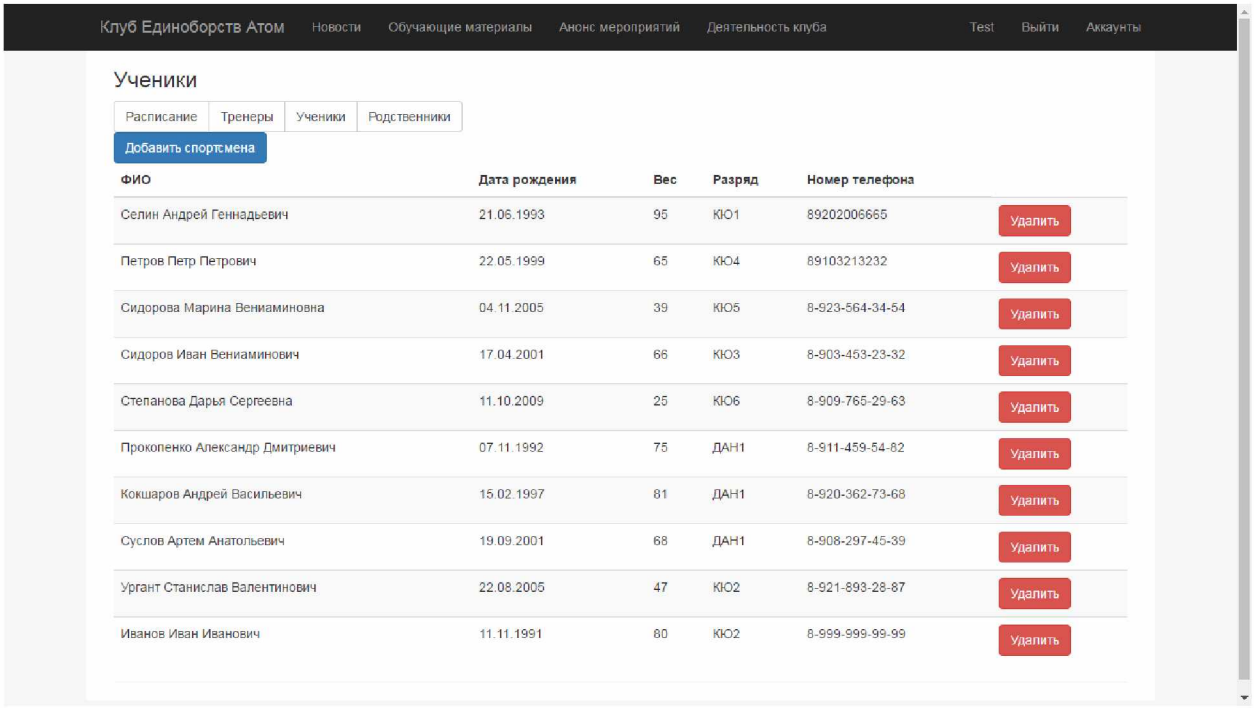

Рис. 3.30. Результат добавления спортсмена

Для просмотра и управления списком родственников переходим на страницу «Родственники», как показано на рис. 3.31.

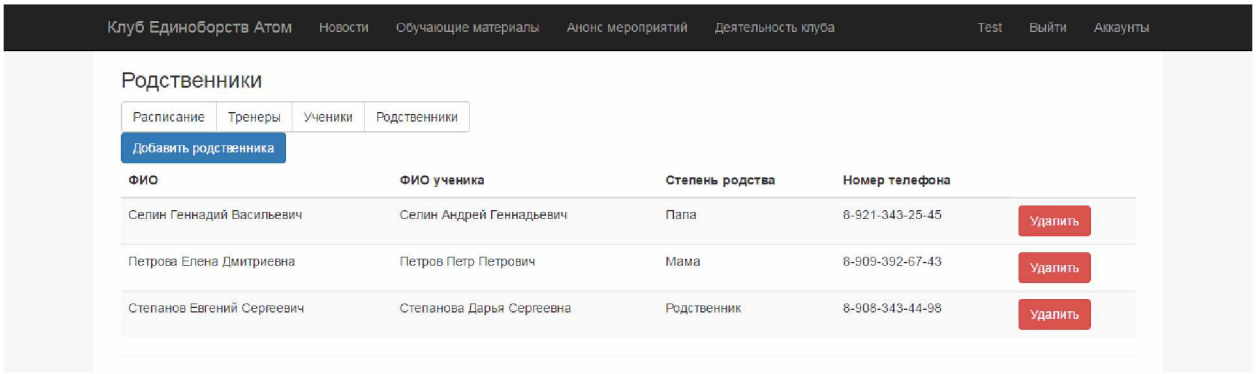

Рис. 3.31. Страница «Родственники»

Проверяем возможность добавления родственника, нажав на кнопку «Добавить родственника».

Заполняем появившуюся форму, в выпадающем списке «Спортсмен» выбираем нужного ученика, как показано на рис. 3.32 и жмем «Добавить».

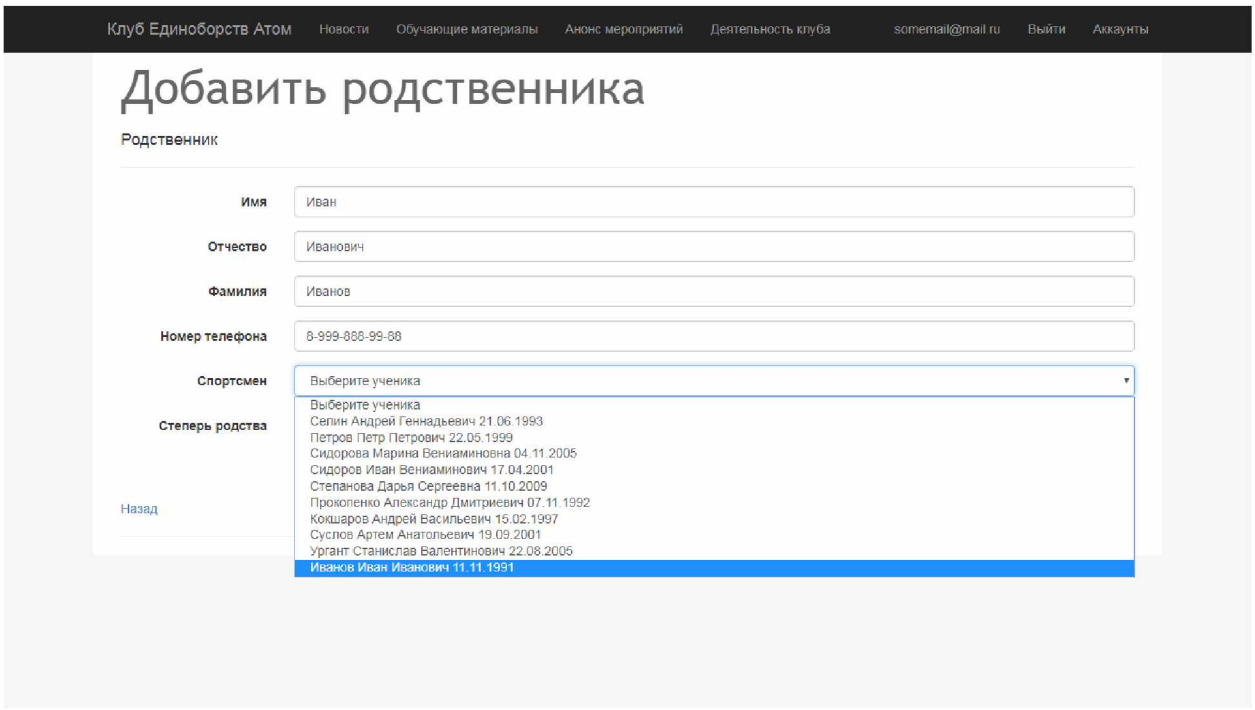

Рис. 3.32. Добавление родственника

Результат добавления родственника показан на рис. 3.33.

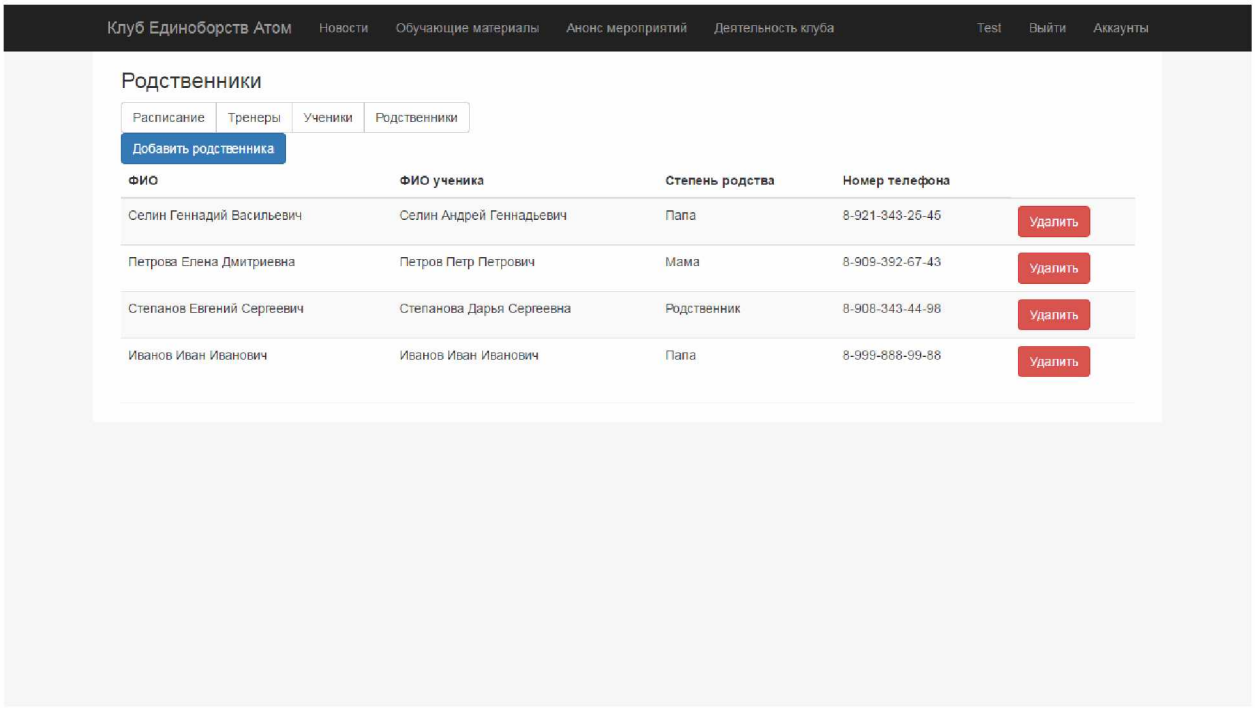

Рис.3.33. Результат добавления родственника

По результатам проведения испытаний разработанной информационной системы было установлено, что система полностью работоспособна и готова к введению в эксплуатацию. Функциональные

возможности разработанной системы полностью удовлетворяют по всем пунктам требований к системе, поставленным в начале ее проектирования.

ЗАКЛЮЧЕНИЕ

В результате выполнения дипломной работы была спроектирована и разработана автоматизированная система поддержки организационной деятельности клуба единоборств «Атом».

Данная информационно-справочная система реализована в форме веб-приложения, которое выполняет информационное сопровождение деятельности клуба единоборств и позволяет любому пользователю сети интернет получить исчерпывающую информацию о текущей деятельности клуба, о тех мероприятиях, в которых клуб планирует принимать участие, а так же получить доступ к обучающим материалам, способствующим более тщательному постижению изучаемого единоборства. Помимо этого система позволяет вести организационную работу, позволяя сторонним посетителям ознакомиться с расписанием занятий, а сотрудникам клуба получить быстрый доступ к данным об учениках, занимающихся в клубе и их родственниках.

После анализа деятельности клуба, изучения уже имеющихся систем решающих схожие задачи, были сформулированы требования, на основе которых спроектирована и разработана автоматизированная система. По результатам тестирования системы, следует сделать вывод – разработанное веб приложение полностью удовлетворяет всем требования, предъявленным к системе не этапе постановки задачи.

СПИСОК ИСПОЛЬЗОВАННЫХ ЛИТЕРАТУРЫ

1. Шаблон проектирования «Модель Представление Контроллер» // Википедия: свободная энциклопедия - 2012 [Электронный ресурс]. – URL: https://ru.wikipedia.org/wiki/Model-View-Controller (дата обращения: 07.10.2016).

2. Преимущества ASP.NET MVC // Программирование на языке C#, платформа .NET Framework. - 2011 [Электронный ресурс]. – URL: https://professorweb.ru/my/ASP_NET/mvc/level1/1_2.php (дата обращения: 25.10.2016).

3. Архитектурный паттерн MVC // Около программирования - тьюториалы и статьи по веб-программированию. - 2011 [Электронный ресурс]. – URL: http://artanovy.com/2011/03/arhitekturnyj-pattern-mvc/ (дата обращения: 21.10.2016).

4. Ильичев С. MVC: что это такое и какое отношение имеет к пользовательскому интерфейсу // Типичный программист – 2015 [Электронный ресурс]. – URL: https://tproger.ru/articles/mvc/ (дата обращения: 03.11.2016).

5. Фримен А. ASP.NET MVC 5 с примерами на C# для профессионалов, 5-е издание: Перевод с английского - М.: ООО “И.Д. Вильямс”, 2015. – 736 с.

6. Магдануров Г.И., Юнев В.А. ASP.NET MVC Framework - СПб.: БХВ-Петербург, 2010. - 320 с.

7. Сандерсон С. ASP.NET MVC Framework с примерами на C# для профессионалов английского - М.: ООО “И.Д. Вильямс”, 2010. – 560 с.

8. Лекция 5: Диаграмма активностей: крупным планом // Национальный Открытый Университет «ИНТУИТ» - 2003 [Электронный ресурс]. – URL: http://www.intuit.ru/studies/courses/1007/229/lecture/5958 (дата обращения: 20.12.2016).

9. Леоненков А. Самоучитель UML - СПб.: БХВ-Петербург, 2007. – 576 с. (дата обращения: 22.12.2016).

10. Общие сведения о платформе Entity Framework // Microsoft Developer Network – 2001 [Электронный ресурс]. – URL: https://msdn.microsoft.com/ru-ru/library/bb399567(v=vs.110).aspx (дата обращения: 15.01.2017).

ПРИЛОЖЕНИЕ

EFDbContext.cs

```
public class EFDbContext : IdentityDbContext<ApplicationUser>
{
 public EFDbContext() : base("EFDbContext") { }
 public static EFDbContext Create()
 {
    return new EFDbContext();
 }
 public DbSet<New> News { get; set; }
 public DbSet<NewsContent> NewsContents { get; set; }
 public DbSet<Material> Materials { get; set; }
 public DbSet<MaterialsContent> MaterialsContents { get; set; }
 public DbSet<Event> Events { get; set; }
 public DbSet<EventsContent> EventsContents { get; set; }
 public DbSet<Student> Students { get; set; }
 public DbSet<Parent> Parents { get; set; }
 public DbSet<Trainer> Trainers { get; set; }
 public DbSet<Group> Groups { get; set; }
 public DbSet<Schedule> Schedules { get; set; }
 public DbSet<Picture> Pictures { get; set; }
}
```

Event.cs

```
public class Event
{
   public int ID{get;set;}
   public DateTime DateOf { get; set; }
   public string Name { get; set; }
}
```

EventContent.cs

```
public class EventsContent
{
   [Key, Column(Order = 1)]
   [ForeignKey("Event")]
   public int EventID { get; set; }
   [Key, Column(Order = 2)]
   public int ContentID { get; set; }
   public string Content { get; set; }
   public virtual Event Event { get; set; }
}
```

Group.cs

```
public enum Sort
```

```
{
    Дзюдо,
    Самбо
}
public class Group
{
    public int ID { get; set; }
    public Sort Sort { get; set; }
    public string Postfix { get; set; }
    public int? TrainerID { get; set; }
    public Trainer Trainer { get; set; }
}
```

LoginModel.cs

```
public class LoginModel
{
    [Required]
    public string Email { get; set; }
    [Required]
    [DataType(DataType.Password)]
    public string Password { get; set; }
}
```

Material.cs

```
public class Material
{
    public int ID { get; set; }
    public string Name { get; set; }
}
```

MaterialContent.cs

```
public class MaterialsContent
{
    [Key, Column(Order = 1)]
    [ForeignKey("Material")]
    public int MaterialID { get; set; }
    [Key, Column(Order = 2)]
    public int ContentID { get; set; }
    public string Content { get; set; }
    public virtual Material Material { get; set; }
}
```

New.cs

```
public class New
{
    public int ID { get; set; }
    public string Name { get; set; }
```

```
}
```

NewsContent.cs

```
public class NewsContent
{
   [Key, Column(Order=1)]
   [ForeignKey("New")]
   public int NewID { get; set; }
   [Key, Column(Order=2)]
   public int ContentID { get; set; }
   public string Content { get; set; }
   public virtual New New {get; set;}

}
```

Parent.cs

```
        public enum Degree
{
   Папа,
   Мама,
   Дедушка,
   Бабушка,
   Родственник
}
public class Parent
{
   public int ID { get; set; }
   [Display(Name = "Имя")]
   public string First { get; set; }
   [Display(Name = "Отчество")]
   public string Middle { get; set; }
   [Display(Name = "Фамилия")]
   public string Last { get; set; }
   [Display(Name = "Номер телефона")]
   public string Phone { get; set; }
   [Display(Name = "Спортсмен")]
   public int ChildID { get; set; }
   [Display(Name = "Степерь родства")]
   public Degree Degree { get; set; }
   public virtual Student Child { get; set; }
}
public class Picture
{
   [Key, DatabaseGenerated(DatabaseGeneratedOption.None)]
   public int ID { get; set; }
   public string Name { get; set; }
   public string Topic { get; set; }
```

```
}
public class RegisterModel
{
    [Required]
    public string Email { get; set; }
    [Required]
    [DataType(DataType.Password)]
    public string Password { get; set; }
    [Required]
    [Compare("Password", ErrorMessage = "Пароли не совпадают")]
    [DataType(DataType.Password)]
    public string PasswordConfirm { get; set; }
}
public class EditRoleModel
{
    public string Id { get; set; }
    public string Name { get; set; }
    public string Description { get; set; }
}
public class CreateRoleModel
{
    public string Name { get; set; }
    public string Description { get; set; }
}
public enum DayWeek{
    Понедельник,
    Вторник,
    Среда,
    Четверг,
    Пятница,
    Суббота,
    Воскресенье
}
public class Schedule
{
    public int ID { get; set; }
    public DayWeek Day { get; set; }
    public DateTime STime { get; set; }
    public int GroupID { get; set; }
    public virtual Group Group { get;set; }
}
public enum Ranks
{
    КЮ6,
    КЮ5,
```

```
    КЮ4,
    КЮ3,
    КЮ2,
    КЮ1,
    ДАН1,
    ДАН2,
    ДАН3,
    ДАН4,
    ДАН5,
    ДАН6,
    ДАН7,
    ДАН8
}
public class Student
{
    public int ID { get; set; }
    [Display(Name = "Имя")]
    public string First { get; set; }
    [Display(Name = "Отчество")]
    public string Middle { get; set; }
    [Display(Name = "Фамилия")]
    public string Last { get; set;}
    [Display(Name = "Дата рождения")]
    [DataType(DataType.Date)]
    public DateTime BirthDay { get; set; }
    [Display(Name = "Вес")]
    public int Weight{get;set;}
    [Display(Name = "Разряд")]
    public Ranks Rank{get;set;}
    [Display(Name = "Номер телефона")]
    public string Phone { get; set; }
}
public class Trainer
{
    public int ID { get; set; }
    [Display(Name="Имя")]
    public string First { get; set; }
    [Display(Name = "Отчество")]
    public string Middle { get; set; }
    [Display(Name = "Фамилия")]
    public string Last { get; set; }
    [Display(Name = "Номер телефона")]
    public string Phone { get; set; }
```

```
}
```

BlogModel.cs

```
public class BlogModel
{
  public int ID { get; set; }
  [Display(Name = "Название")]
  public string Name { get; set; }
  [Display(Name = "Изображение иконки")]
  public string HeadImg { get; set; }
  [Display(Name = "Содержание")]
  public List<string> Contents { get; set; }
  [HiddenInput(DisplayValue=false)]
  public string Type { get; set; }
 public BlogModel(Material _model, List<MaterialsContent> _modelContent)
 {
     ID = _model.ID;
     Name = _model.Name;

     List<MaterialsContent> modelContent = _modelContent;
     Contents = new List<string>();

     foreach (MaterialsContent c in modelContent)
     {
        if (c.ContentID != 1) Contents.Add(c.Content);
        else HeadImg = Convert.ToString(c.Content);
     }

     Type = "Material";
 }
 public BlogModel(New _model, List<NewsContent> _modelContent)
 {
     ID = _model.ID;
     Name = _model.Name;

     List<NewsContent> modelContent = _modelContent;
     Contents = new List<string>();

     foreach (NewsContent c in modelContent)
     {
        if (c.ContentID != 1) Contents.Add(c.Content);
        else HeadImg = Convert.ToString(c.Content);
     }

     Type = "New";
 }
```

```
 public BlogModel(Event _model, List<EventsContent> _modelContent)
 {
    ID = _model.ID;
    Name = _model.Name;

    List<EventsContent> modelContent = _modelContent;
    Contents = new List<string>();

    foreach (EventsContent c in modelContent)
    {
       if (c.ContentID != 1) Contents.Add(c.Content);
       else HeadImg = Convert.ToString(c.Content);
    }

    Type = "Event";
 }
 public BlogModel() { }
}
```

ParentsPageModel.cs

```
        public class ParentCreateModel
{
 [Display(Name = "Имя")]
 public string First { get; set; }
 [Display(Name = "Отчество")]
 public string Middle { get; set; }
 [Display(Name = "Фамилия")]
 public string Last { get; set; }
  [Display(Name = "Номер телефона")]
 public string Phone { get; set; }
 [Display(Name = "Степерь родства")]
 public Degree Degree { get; set; }
 [Display(Name = "Спортсмен")]
 public int ChildID { get; set; }
 public List<Student> Students { get; set; }
 public ParentCreateModel(List<Student> students)
 {
    Students = students;
 }
 public ParentCreateModel() { }
}
 public class ParentPageModel
 {
 public int ID { get;set; }
 public string First { get; set; }
 public string Middle { get; set; }
```

```
  public string Last { get; set; }
  public string Phone { get; set; }
  public Degree Degree { get; set; }
  public Student Student { get; set; }
  public ParentPageModel(Parent parent, Student student)
  {
     ID = parent.ID;
     First = parent.First;
     Middle = parent.Middle;
     Last = parent.Last;
     Phone = parent.Phone;
     Degree = parent.Degree;
     Student = student;
  }
}
public class ParentsPageModel
{
  public List<ParentPageModel> Parents { get; set; }

 public ParentsPageModel(List<Parent> parents, List<Student> students)
  {
     Parents = new List<ParentPageModel>();
    foreach(Parent p in parents)
     {
        Parents.Add(new ParentPageModel(p,students.Find(s=>s.ID==p.ChildID)));
     }
  }
}
```

ScheduleWithGroup.cs

```
public class ScheduleWithGroup
{
  public List<Group> Groups { get; set; }
  public List<Schedule> Schedules { get; set; }
  public ScheduleWithGroup(List<Group> groups, List<Schedule> schedules)
  {
     Groups = groups;
     Schedules = schedules;
  }
}
```

TrainersWithGroup.cs

```
public class TrainersWithGroups
{
  public List<Trainer> Trainers { get; set; }
  public List<Group> Groups { get; set; }
  public TrainersWithGroups()  { }
```

```
}
```

UsersWithRoles.cs

```
        public class UsersWithRoles
{
 public List<ApplicationUser> Users { get; set; }
 public List<IdentityRole> Roles { get; set; }
 public string Role { get; set; }
 public string UserID { get; set; }
 public UsersWithRoles(EFDbContext context)
 {
    Users = context.Users.ToList();
    Roles = context.Roles.ToList();
 }
 public UsersWithRoles() { }
}
```

RouteConfig.cs

```
public class RouteConfig
{
 public static void RegisterRoutes(RouteCollection routes)
 {
    routes.IgnoreRoute("{resource}.axd/{*pathInfo}");
    routes.MapRoute(
       name: "Default",
       url: "{controller}/{action}/{id}",
       defaults: new { action = "Index", id = UrlParameter.Optional } );
   }
}
```

AccountController.cs

```
public class AccountController : Controller
{
   // GET: Account
   private ApplicationUserManager UserManager
   {
      get
      {
                  return
        HttpContext.GetOwinContext().GetUserManager<ApplicationUserManager>();
      }
}
public ActionResult Register()
{
   return View();
}
[HttpPost]
public async Task<ActionResult> Register(RegisterModel model)
```

```
{
   if(ModelState.IsValid)
   {
                  ApplicationUser user = new ApplicationUser { UserName = model.Email,
      Email = model.Email };
                  IdentityResult result = await UserManager.CreateAsync(user,
      model.Password);
      if(result.Succeeded)
      {
         return RedirectToAction("Login", "Account");
      }
      else
      {
         foreach(string error in result.Errors)
         {
            ModelState.AddModelError("", error);
         }
      }
   }
   return View(model);
}

private IAuthenticationManager AuthenticationManager
{
   get
   {
      return HttpContext.GetOwinContext().Authentication;
   }
}
public ActionResult Login(string returnUrl)
{
   ViewBag.returnUrl = returnUrl;
   return View();
}
[HttpPost]
[ValidateAntiForgeryToken]
public async Task<ActionResult> Login(LoginModel model, string returnUrl)
{
   if(ModelState.IsValid)
   {
                  ApplicationUser user = await UserManager.FindAsync(model.Email,
      model.Password);
      if(user==null)
      {
         ModelState.AddModelError("", "Неверный логин или пароль.");
```

```
        }
        else
        {
                        ClaimsIdentity claim = await UserManager.CreateIdentityAsync(user,
          DefaultAuthenticationTypes.ApplicationCookie);
          AuthenticationManager.SignOut();
          AuthenticationManager.SignIn(new AuthenticationProperties
            {
              IsPersistent = true
            },
            claim);
          if (String.IsNullOrEmpty(returnUrl))
            return RedirectToAction("Index", "Home");
          return Redirect(returnUrl);
        }
    }
    ViewBag.returnUrl = returnUrl;
    return View(model);
}
public ActionResult Logout()
{
    AuthenticationManager.SignOut();
    return RedirectToAction("Login");
}
[Authorize(Roles="Admin")]
[HttpPost]
public async Task<ActionResult> Delete(string id)
{
    ApplicationUser user = await UserManager.FindByIdAsync(id);

    if (user != null)
    {
        IdentityResult result = await UserManager.DeleteAsync(user);
        if (result.Succeeded)
        {
            return RedirectToAction("Index","Roles");
        }
        else
        {
            return RedirectToAction("Index", "Roles");
        }
    }
    else
    {
        return RedirectToAction("Index", "Roles");
```

```
    }
}
}
```

ActivityController.cs

```
        public class ActivityController : Controller
        {

// GET: Activity
private EFDbContext db = new EFDbContext();

public ActionResult Index()
{
   return View(new ScheduleWithGroup(db.Groups.ToList(),db.Schedules.ToList()));
}
[HttpPost]
public ActionResult CreateSchedule(string dayweek, string stime, string groupid)
{
   db.Schedules.Add(new Schedule
   {
      Day = (DayWeek)Enum.Parse(typeof(DayWeek),dayweek),
      STime = DateTime.Parse(stime),
      GroupID=Convert.ToInt32(groupid)
   });
   db.SaveChanges();
   return RedirectToAction("Index");
}
public ActionResult DeleteSchedule(string id)
{
   db.Schedules.Remove(db.Schedules.Find(Convert.ToInt32(id)));
   db.SaveChanges();
   return RedirectToAction("Index");
}
public ActionResult Create()
{
   return View();
}
[HttpPost]
public ActionResult Create(Student model)
{
   db.Students.Add(model);
   db.SaveChanges();
   return RedirectToAction("Index");
}
public ActionResult Students()
{
```

```
    return View(db.Students.ToList());
}
public ActionResult DeleteStudent(string id)
{
    db.Students.Remove(db.Students.Find(Convert.ToInt32(id)));
    db.SaveChanges();
    return RedirectToAction("Students");
}
public ActionResult Parents()
{
    return View(new ParentsPageModel(db.Parents.ToList(),db.Students.ToList()));
}
public ActionResult DeleteParent(string id)
{
    db.Parents.Remove(db.Parents.Find(Convert.ToInt32(id)));
    db.SaveChanges();
    return RedirectToAction("Parents");
}
public ActionResult Trainers()
{
                return View(new TrainersWithGroups{ Trainers = db.Trainers.ToList(), Groups =
        db.Groups.ToList() });
}
public ActionResult CreateTrainer()
{
    return View(new Trainer());
}
[HttpPost]
public ActionResult CreateTrainer(Trainer model)
{
    db.Trainers.Add(model);
    db.SaveChanges();
    return RedirectToAction("Trainers");
}
[HttpPost]
public ActionResult DeleteTrainer(string id)
{
    try
        {
            db.Trainers.Remove(db.Trainers.Find(Convert.ToInt32(id)));
            db.SaveChanges();
        }
    catch (System.Data.Entity.Infrastructure.DbUpdateException) { }
    return RedirectToAction("Trainers");
}
```

```
[HttpPost]
public ActionResult CreateGroup(string sort, string postfix)
{
                db.Groups.Add(new Group { Postfix = postfix, Sort =
        (Sort)Enum.Parse(typeof(Sort), sort), TrainerID=null });
   db.SaveChanges();
   return RedirectToAction("Trainers");
}
[HttpPost]
public ActionResult AppointGroup(string idTrainer, string idGroup)
{
                db.Groups.Find(Convert.ToInt32(idGroup)).TrainerID =
        Convert.ToInt32(idTrainer);
   db.SaveChanges();
   return RedirectToAction("Trainers");
}
public ActionResult UnAppointGroup(string id)
{
   db.Groups.Find(Convert.ToInt32(id)).TrainerID = null;
   db.SaveChanges();
   return RedirectToAction("Trainers");
}
public ActionResult DeleteGroup(string id)
{
   db.Groups.Remove(db.Groups.Find(Convert.ToInt32(id)));
   db.SaveChanges();
   return RedirectToAction("Trainers");
}

public ActionResult CreateParent()
{
   return View(new ParentCreateModel(db.Students.ToList()));
}
[HttpPost]
public ActionResult CreateParent(ParentCreateModel model)
{
   db.Parents.Add(new Parent()
   {
      First = model.First,
      Middle = model.Middle,
      Last = model.Last,
      Phone = model.Phone,
      Degree = model.Degree,
      ChildID = model.ChildID
   });
```

```
    db.SaveChanges();
    return RedirectToAction("Parents");
}
public JsonResult GetPeopleDataJson(string category)
{
    switch (category)
    {
        case "students":
        var data = db.Students.AsEnumerable().Select(
            s => new
            {
                First = s.First,
                Middle = s.Middle,
                Last = s.Last,
                Phone = s.Phone,
                Rank = Enum.GetName(typeof(Ranks), s.Rank),
                BirthDay = s.BirthDay.Date.ToString("dd.MM.yyyy"),
                Weight = s.Weight
            });

        return Json(data, JsonRequestBehavior.AllowGet);
        case "parents":
        var data2 = db.Parents.AsEnumerable().Select(
            s => new
            {
                First = s.First,
                Middle = s.Middle,
                Last = s.Last,
                Phone = s.Phone,
                Degree = Enum.GetName(typeof(Degree), s.Degree),
                Child = s.Child.Last,
            });

        return Json(data2, JsonRequestBehavior.AllowGet);
        default:
          data = db.Students.AsEnumerable().Select(
            s => new
            {
                First = s.First,
                Middle = s.Middle,
                Last = s.Last,
                Phone = s.Phone,
                Rank = Enum.GetName(typeof(Ranks), s.Rank).ToString(),
                BirthDay = s.BirthDay.Date.ToString("dd.MM.yyyy"),
                Weight = s.Weight
```

```
            });

        return Json(data, JsonRequestBehavior.AllowGet);
    }

}
}
```

AddController.cs

```
public class AddController : Controller
{
private EFDbContext db = new EFDbContext();
// GET: Add
public ActionResult Index()
{
    return View();
}
public ActionResult Create(string type)
{
    return View(new BlogModel(){Type=type});
}
[HttpPost]
public ActionResult Create(BlogModel model)
{
    if (TryUpdateModel<BlogModel>(model))
    {
        string name = model.Name.Replace("\r", "").Replace("\n", "");
        ModelAdd(model.Type, name);
        db.SaveChanges();

        string[] s;

                    s = model.Contents.FirstOrDefault().Split(new string[] { "\n" },
          StringSplitOptions.RemoveEmptyEntries);
        int id= FindID(model.Type,name);
        int i = 1;

        ContentAdd(model.Type, id, model.HeadImg, i++);
        foreach (string c in s)
        {
            ContentAdd(model.Type, id, c, i++);
        }
        ContentAdd(model.Type, id, ".", i);

        db.SaveChanges();
    }
```

```
    return View(model);
}
public PicturesModel Pictures(string filter)
{
                List<Picture> pictures = filter == " " ? db.Pictures.ToList() : db.Pictures.Where(x
        => x.Topic == filter).ToList();
    PicturesModel model;
    try
    {
                    model = new PicturesModel(pictures, db.Pictures.Select(x =>
        x.Topic).Distinct().ToList(), filter, db.Pictures.OrderByDescending(x =>
        x.ID).FirstOrDefault().ID);
    }
    catch (NullReferenceException)
    {
        model = new PicturesModel(pictures, filter);
    }
    return model;
}
public PartialViewResult GetPictures(string filter = "none")
{
    return PartialView(Pictures(filter));
}
public JsonResult GetPicturesJson(string filter = "none")
{
    return Json(Pictures(filter), JsonRequestBehavior.AllowGet);
}
[HttpPost]
public JsonResult Pictures()
{
    int i;
    try
    {
        i = db.Pictures.OrderByDescending(x => x.ID).FirstOrDefault().ID;
    }
    catch (NullReferenceException)
    {
        i = 0;
    }
    foreach (string file in Request.Files)
    {
        db.Pictures.Add(new Picture()
        {
            ID = ++i,
            Name = "<img=" + i + ".jpg>",
```

```
            Topic = Request.Form["topic"]
        });
        var upload = Request.Files[file];
        if (upload != null)
        {
            // получаем имя файла
            string fileName = System.IO.Path.GetFileName(upload.FileName);
            // сохраняем файл в папку Files в проекте
            upload.SaveAs(Server.MapPath("~/Content/img/" + i + ".jpg"));
            db.SaveChanges();
        }
    }
    return Json("файл загружен");
}
public int FindID(string type, string name)
{
    switch (type)
    {
        case "New":
                        return db.News.Where(nc => nc.Name == name).OrderByDescending(nc =>
            nc.ID).FirstOrDefault().ID;
        case "Material":
                        return db.Materials.Where(nc => nc.Name == name).OrderByDescending(nc
            => nc.ID).FirstOrDefault().ID;
        case "Event":
                        return db.Events.Where(nc => nc.Name == name).OrderByDescending(nc
            => nc.ID).FirstOrDefault().ID;
        default: return 0;
    }
}
public void ModelAdd(string type, string name)
{
    switch (type)
    {
        case "New":
            db.News.Add(new New() { Name = name });
            break;

        case "Material":
            db.Materials.Add(new Material() { Name = name });
            break;

        case "Event":
                        db.Events.Add(new Event() { Name = name, DateOf=new
            DateTime(1900,01,01) });
```

```
            break;
        }
    }
    public void ContentAdd(string type, int id, string content, int contentId)
    {
        switch (type)
        {
            case "New":
                db.NewsContents.Add(new NewsContent()
                {
                    NewID = id,
                    Content = content.Replace("\r", "").Replace("\n", ""),
                    ContentID = contentId
                });
                break;

            case "Material":
                db.MaterialsContents.Add(new MaterialsContent()
                {
                    MaterialID = id,
                    Content = content.Replace("\r", "").Replace("\n", ""),
                    ContentID = contentId
                });
                break;

            case "Event":
                db.EventsContents.Add(new EventsContent()
                {
                    EventID = id,
                    Content = content.Replace("\r", "").Replace("\n", ""),
                    ContentID = contentId
                });
                break;
        }
    }
}
```

EventsController

```
public class EventsController : Controller
{
// GET: Events
private EFDbContext db = new EFDbContext();

public ActionResult Index()
{
    List<BlogModel> models = new List<BlogModel>();
```

```
    List<EventsContent> content;
    foreach (Event n in db.Events.ToList())
    {
        content = db.EventsContents.Where(x => x.EventID == n.ID).ToList();
        models.Add(new BlogModel(n, content));
    }
    List<string> str = db.Pictures.Select(x => x.Name).ToList();
    List<Picture> str11 = db.Pictures.ToList();
    List<string> str12 = str11.Select(x => x.Name).ToList();
    List<string> str1 = db.EventsContents.Select(x => x.Content).ToList();
                    List<string> str2 = db.EventsContents.Where(x =>
            x.Content.Contains("img")).Select(x => x.Content).ToList();

    return View(models);
}
public ActionResult View(int id)
{
                    BlogModel Events = new BlogModel(db.Events.FirstOrDefault(n => n.ID == id),
            db.EventsContents.Where(c => c.EventID == id).ToList());
    return View(Events);
}

public ActionResult Delete(int idmodels)
{
    string topic = db.Events.Find(idmodels).Name;
                    List<int> pictures = db.Pictures.Where(x => x.Topic == topic).Select(x =>
            x.ID).ToList();
    db.Pictures.RemoveRange(db.Pictures.Where(x => x.Topic == topic));
                    foreach (int p in pictures)
            System.IO.File.Delete(Server.MapPath("~/Content/img/" + p + ".jpg"));
                    db.EventsContents.RemoveRange(db.EventsContents.Where(x => x.EventID ==
            idmodels));
    db.Events.Remove(db.Events.Find(idmodels));
    db.SaveChanges();
    return RedirectToAction("Index");
}
}
```

HomeController.cs

```
public class HomeController : Controller
{
// GET: Home
private EFDbContext db = new EFDbContext();

public ActionResult Index()
{
```

```
    List<BlogModel> models = new List<BlogModel>();
    List<MaterialsContent> contentM;
    List<NewsContent> contentN;
    List<EventsContent> contentE;
    foreach (Material n in db.Materials.ToList())
    {
                    contentM = db.MaterialsContents.Where(x => x.MaterialID == n.ID).ToList();
        models.Add(new BlogModel(n, contentM));
    }
    foreach (New n in db.News.ToList())
    {
        contentN = db.NewsContents.Where(x => x.NewID == n.ID).ToList();
        models.Add(new BlogModel(n, contentN));
    }
    foreach (Event n in db.Events.ToList())
    {
        contentE = db.EventsContents.Where(x => x.EventID == n.ID).ToList();
        models.Add(new BlogModel(n, contentE));
    }
    return View(models);
}
}
```

MaterialsController.cs

```
public class MaterialsController : Controller
{
// GET: Materials
private EFDbContext db = new EFDbContext();

public ActionResult Index()
{
    List<BlogModel> models = new List<BlogModel>();
    List<MaterialsContent> content;
    foreach (Material n in db.Materials.ToList())
    {
        content = db.MaterialsContents.Where(x => x.MaterialID == n.ID).ToList();
        models.Add(new BlogModel(n, content));
    }
    return View(models);
}
public ActionResult View(int id)
{
                BlogModel Materials = new BlogModel(db.Materials.FirstOrDefault(n => n.ID
        == id), db.MaterialsContents.Where(c => c.MaterialID == id).ToList());
    return View(Materials);
}
```

```
public ActionResult Delete(int idmodels)
{
   string topic = db.Materials.Find(idmodels).Name;
                List<int> pictures = db.Pictures.Where(x => x.Topic == topic).Select(x =>
        x.ID).ToList();
   db.Pictures.RemoveRange(db.Pictures.Where(x => x.Topic == topic));
                foreach (int p in pictures)
        System.IO.File.Delete(Server.MapPath("~/Content/img/" + p + ".jpg"));
              db.MaterialsContents.RemoveRange(db.MaterialsContents.Where(x =>
        x.MaterialID == idmodels));
   db.Materials.Remove(db.Materials.Find(idmodels));
   db.SaveChanges();
   return RedirectToAction("Index");
}
}
```

NewsController.cs

```
public class NewsController : Controller
{
private EFDbContext db = new EFDbContext();
// GET: News
public ActionResult Index()
{
   List<BlogModel> news = new List<BlogModel>();
   List<NewsContent> content;
   foreach(New n in db.News.ToList())
   {
      content = db.NewsContents.Where(x => x.NewID == n.ID).ToList();
      news.Add(new BlogModel(n, content));
   }
   List<string> str = db.Pictures.Select(x => x.Name).ToList();
   List<Picture> str11 = db.Pictures.ToList();
   List<string> str12 = str11.Select(x => x.Name).ToList();
   List<string> str1 = db.NewsContents.Select(x => x.Content).ToList();
                List<string>
        str2=db.NewsContents.Where(x=>x.Content.Contains("img")).Select(x=>x.Content).ToL
        ist();

   return View(news);
}
public ActionResult View(int id)
{
                BlogModel news = new
        BlogModel(db.News.FirstOrDefault(n=>n.ID==id),db.NewsContents.Where(c=>c.NewI
        D==id).ToList());
```

```
    return View(news);
}

public ActionResult Delete(int idnews)
{
    string topic = db.News.Find(idnews).Name;
                List<int> pictures = db.Pictures.Where(x => x.Topic == topic).Select(x =>
        x.ID).ToList();
    db.Pictures.RemoveRange(db.Pictures.Where(x => x.Topic == topic));

                foreach (int p in pictures)
        System.IO.File.Delete(Server.MapPath("~/Content/img/" + p + ".jpg"));

    db.NewsContents.RemoveRange(db.NewsContents.Where(x => x.NewID == idnews));
    db.News.Remove(db.News.Find(idnews));
    db.SaveChanges();
    return RedirectToAction("Index");
}
}
```

RolesController.cs

```
[Authorize(Roles="Admin")]
public class RolesController : Controller
{
// GET: Roles
private ApplicationRoleManager RoleManager
{
    get
    {
                    return
        HttpContext.GetOwinContext().GetUserManager<ApplicationRoleManager>();
    }
}
[HttpPost]
public ActionResult Index(UsersWithRoles model, string id)
{
                var userManager = new ApplicationUserManager(new
        UserStore<ApplicationUser>(new EFDbContext()));
    userManager.AddToRole(id, model.Role);

    return View(new UsersWithRoles(new EFDbContext()));
}
public ActionResult Untie(string role, string id)
{
                var userManager = new ApplicationUserManager(new
        UserStore<ApplicationUser>(new EFDbContext()));
```

```
    userManager.RemoveFromRole(id,role);

    return RedirectToAction("Index");
}
public ActionResult Index()
{
    return View(new UsersWithRoles(new EFDbContext()));
}

// GET: Roles/Create
public ActionResult Create()
{
    return View();
}

// POST: Roles/Create
[HttpPost]
public async Task<ActionResult> Create(CreateRoleModel model)
{
    if(ModelState.IsValid)
    {
        IdentityResult result = await RoleManager.CreateAsync(new ApplicationRole
            {
                Name = model.Name,
                Description = model.Description
            });
        if(result.Succeeded)
        {
            return RedirectToAction("Index");
        }
        else
        {
            ModelState.AddModelError("", "Что-то пошло не так");
        }
    }
    return View(model);
}

// GET: Roles/Edit/5
public async Task<ActionResult> Edit(string id)
{
    ApplicationRole role = await RoleManager.FindByIdAsync(id);
    if (role != null)
    {
```

```
                return View(new EditRoleModel { Id=role.Id, Name=role.Name,
        Description=role.Description});
    }
    return RedirectToAction("Index");
}

// POST: Roles/Edit/5
[HttpPost]
public async Task<ActionResult> Edit(EditRoleModel model)
{
    if (ModelState.IsValid)
    {
        ApplicationRole role = await RoleManager.FindByIdAsync(model.Id);
        if (role != null)
        {
            role.Description = model.Description;
            role.Name = model.Name;
            IdentityResult result = await RoleManager.UpdateAsync(role);
            if(result.Succeeded)
            {
                return RedirectToAction("Index");
            }
            else
            {
                ModelState.AddModelError("", "Что-то пошло не так");
            }
        }
    }
    return View(model);
}

// GET: Roles/Delete/5
public async Task<ActionResult> Delete(string id)
{
    ApplicationRole role = await RoleManager.FindByIdAsync(id);
    if(role!=null)
    {
        IdentityResult result = await RoleManager.DeleteAsync(role);
    }
    return RedirectToAction("Index");
}
}
```

InterpredNews.cs

```
public static class InterpredNews
{
```

```
public static bool Check(string call)
{
if (call.Contains("<"))return true;
else return false;
}
public static MvcHtmlString Interpred(this HtmlHelper html, List<string> calls)
{
//-----------------indicator
TagBuilder tag = new TagBuilder("div");
TagBuilder tagIndOl = new TagBuilder("ol");
TagBuilder tagContDiv=new TagBuilder("div");

int idCarousel=0;
int imageRow = 0;
foreach(string call in calls)
{
if (call.Contains("<img"))
{
if (imageRow++ == 0) idCarousel++;
TagBuilder tagIndLi = new TagBuilder("li");
tagIndLi.MergeAttributes(new Dictionary<string, string>
{
{"class",imageRow>1?"":"active"},
{"target-data","#myCarousel"+idCarousel},
{"data-slide-to",Convert.ToString(imageRow-1)}
});

//-------------------------------------------------content
TagBuilder tagImg = new TagBuilder("img");
string[] split = call.Split(new string[] { "=" }, StringSplitOptions.None);
tagImg.MergeAttributes(new Dictionary<string, string>
{
{"src", "/Content/img/" + (split[1].Replace(">",""))},
{"class", "center-block img-responsive"},
{"height","400"}
});
TagBuilder tagItemDiv = new TagBuilder("div");
tagItemDiv.AddCssClass((imageRow>1?"":"active") + " item");
tagItemDiv.InnerHtml += (tagImg.ToString());
if (imageRow == 1)
{
tagContDiv = new TagBuilder("div");
tagContDiv.AddCssClass("carousel-inner");
}
tagContDiv.InnerHtml += tagItemDiv.ToString();
```

```
//-------
if (imageRow == 1)
{
tagIndOl = new TagBuilder("ol");
tagIndOl.AddCssClass("carousel-indicators");
}
tagIndOl.InnerHtml += tagIndLi.ToString();
}
//-----------------------------------------------buttons
else if(imageRow>0)
{
TagBuilder tagLeftSpan = new TagBuilder("span");
tagLeftSpan.AddCssClass("glyphicon glyphicon-chevron-left");
TagBuilder tagLeft = new TagBuilder("a");
tagLeft.MergeAttributes(new Dictionary<string, string>
{
{"class","carousel-control left"},
{"href","#myCarousel"+idCarousel},
{"data-slide","prev"}
});
tagLeft.InnerHtml += tagLeftSpan.ToString();
TagBuilder tagRightSpan = new TagBuilder("span");
tagRightSpan.AddCssClass("glyphicon glyphicon-chevron-right");
TagBuilder tagRight = new TagBuilder("a");
tagRight.MergeAttributes(new Dictionary<string, string>
{
{"class","carousel-control right"},
{"href","#myCarousel"+idCarousel},
{"data-slide","next"}
});
tagRight.InnerHtml += tagRightSpan.ToString();
//-------------------------container
TagBuilder tagCarousel = new TagBuilder("div");
tagCarousel.MergeAttributes(new Dictionary<string, string>
{
{"id","myCarousel"+idCarousel},
{"class", "carousel slide"},
{"data-interval", "3000"},
{"data-ride", "carousel"},
{"data-pause", "hover"}
});
tagCarousel.InnerHtml += tagIndOl.ToString() + tagContDiv.ToString() + tagLeft.ToString() +
tagRight.ToString();
tag.InnerHtml += tagCarousel.ToString();
imageRow = 0;
```

```
}
if (!call.Contains("<img"))
{
tag.InnerHtml += "<p>" + call + "</p>";
}
}
return new MvcHtmlString(tag.ToString());
}
public static string Interpred(string call)
{
if (call.Contains("<img="))
{
string[] split = call.Split(new string[] { "<img=" }, StringSplitOptions.None);
return split[1].Replace(">","");
}
else return call;
}
}
}
```