МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное
образовательное учреждение высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»

Высшая школа электроники и компьютерных наук
Кафедра «Электронные вычислительные машины»

РАБОТА ПРОВЕРЕНА
Рецензент
_____ /О.И.Пригоренко
«____» _____2019 г.

ДОПУСТИТЬ К ЗАЩИТЕ
Заведующий кафедрой ЭВМ
_____/Г.И.Радченко
«____» _____2019 г.

Разработка веб-приложения для организации и проведения соревнований по кёрлингу

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ

Руководитель работы,
к.т.н., доцент каф. ЭВМ
_____/Е.С.Ярош
«____» _____2019 г.

Автор работы
студент группы КЭ – 452
_____ /А.Д.Вдовина
«____» _____2019 г.

Нормконтролёр
_____/С.В.Сяськов
«____» _____2019 г.

Челябинск-2019

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное
образовательное учреждение высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»
Высшая школа электроники и компьютерных наук
Кафедра «Электронные вычислительные машины»

УТВЕРЖДАЮ
Заведующий кафедрой ЭВМ
_____/Г.И.Радченко

«____» _____2019 г.

ЗАДАНИЕ

на выпускную квалификационную работу бакалавра

студенту группы КЭ-452
Вдовиной Анастасии Даниловне
обучающемуся по направлению
09.03.01 «Информатика и вычислительная техника»

1. **Тема работы:** «Разработка веб-приложения для организации и проведения соревнований по кёрлингу» утверждена приказом по университету от «25» апреля 2019 г. № 899.

2. **Срок сдачи студентом законченной работы:** 1 июня 2019 г.

3. **Исходные данные к работе:**

Обеспечить следующий функционал:

1. Возможность создания и редактирования соревнования.
2. Однократный ввод информации о спортсмене/тренере/судье/члене федерации.
3. Показ актуальной информации о соревнованиях, предоставление возможности в режиме онлайн наблюдать за счетом в играх.
4. Показ турнирных таблиц и итоговых протоколов соревнований.

5. Работу зарегистрированного пользователя организовать через личный кабинет.

Среда реализации – по выбору автора.

4. Перечень подлежащих разработке вопросов:

1. Анализ предметной области и существующих решений.
2. Разработка технического задания для разработки веб-приложения для организации и проведения соревнований по кёрлингу.
3. Выбор среды и средств реализации, разработка основных архитектурных решений.
4. Разработка дизайна веб-приложения для организации и проведения соревнований по кёрлингу .
5. Проектирование структуры базы данных.
6. Реализация пользовательской и серверной частей приложения.
7. Тестирование разработанного веб-приложения для организации и проведения соревнований по кёрлингу.

5. Дата выдачи задания: 1 декабря 2018 г.

Руководитель работы _____/*Е.С. Ярош*/

Студент _____/*А.Д. Вдовина* /

КАЛЕНДАРНЫЙ ПЛАН

Этап	Срок сдачи	Подпись руководителя
Введение и обзор литературы	01.03.2019	
Разработка модели, проектирование	01.04.2019	
Реализация системы	31.05.2019	
Тестирование, отладка, эксперименты	8.05.2019	
Компоновка текста работы и сдача на нормоконтроль	28.05.2019	
Подготовка презентации и доклада	1.06.2019	

Руководитель работы _____/*Е.С. Ярош*/

Студент _____/*А.Д. Вдовина*/

АННОТАЦИЯ

Вдовина А. Д. Разработка веб-приложения для организации и проведения соревнований по кёрлингу. – Челябинск: ФГАОУ ВО «ЮУрГУ (НИУ)», ВШЭКН; 2019, 182 с., 65 ил., библиогр. список – 22 наим., 27 прил.

В данной выпускной квалификационной работе выполнена разработка веб-приложения для организации и проведения соревнований по кёрлингу. В ходе работы был произведен обзор существующих аналогов и выявлены их плюсы и минусы. Также было разработано техническое задание, на основе которого реализован основной функционал веб-приложения и произведено тестирование программы на Открытом Чемпионате Челябинской области по кёрлингу 2019 года.

Пояснительная записка состоит из введения, оглавления, основной части из 5 разделов, заключения, библиографического списка и 27 приложений.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Принято считать, что кёрлинг появился в одном из старейших городов Шотландии в начале XVI века, в Стерлинге. На рубеже XX столетия кёрлинг приобрел репутацию престижной игры, а после Второй мировой войны количество стран, в которых развивался кёрлинг, возросло. Это повлекло за собой необходимость проведения регулярных соревнований, в том числе международных. В 1950 году по инициативе федераций ряда стран Европы и Америки была основана Международная федерация кёрлинга, под эгидой которой с 1951 года проводятся чемпионаты Европы, а с 1959 года чемпионаты Мира.

Современный кёрлинг становится все более эстетичным и спортивным, а азарт, присущий этой игре, сделал ее очень популярной во многих странах мира.

В 1991 году была создана Федерация кёрлинга России (ФКР). Так данный вид спорта начал приобретать популярность в нашей стране и развиваться в регионах. В Челябинске кёрлинг появился в 2004 году и начал активно развиваться. Федерация кёрлинга Челябинской области активно занимается проведением и организацией спортивных мероприятий.

АКТУАЛЬНОСТЬ ТЕМЫ

Одно из основных направлений деятельности ФКР – организация различных соревнований, отборов и турниров.

Организация мероприятий сопряжена с работой с документами, включающими в себя заявки на участие спортсменов, ведение протоколов матчей и составление итогового протокола, список участников и судей. Также создается набор документов для каждого участника.

Создание веб-приложения для организации и проведения соревнований по кёрлингу позволит сократить бумажную работу, и, следовательно, уменьшить количество ошибок. Повышается также оперативность, появляется возможность получать быстрый доступ к необходимой информации.

На сегодняшний день ведение документации для проведения соревнований происходит следующим образом: организатор соревнований назначает главного судью и главного секретаря соревнований, которые образуют ГСК соревнований, далее члены ГСК разрабатывают положение о проведении соревнований и отправляют его в регионы. Получив положение, команды отправляют заявки на участие, в которых указаны данные спортсменов. На основе заявок секретарь составляет расписание, распределяет судей и готовит протоколы матчей. В ходе соревнований судьи заполняют протоколы и отдают их секретарю для внесения информации в турнирные таблицы, по окончании соревнований на основе протоколов матчей формируется итоговый протокол.

Такой способ ведения документации является очень трудоемким, а также ограничивает возможности одновременной работы над мероприятием для нескольких человек.

Таким образом, актуальной становится задача разработки веб-приложения, упрощающего создание всех необходимых документов для проводимых мероприятий.

ЦЕЛЬ И ЗАДАЧИ РАБОТЫ

Целью данной работы является разработка веб-приложения для организации и проведения соревнований по кёрлингу. Для достижения поставленной цели необходимо решить следующие задачи:

1. Провести анализ предметной области и аналогичных проектов;
2. Разработать техническое задание.
3. Выбрать среду и средства реализации.
4. Разработать архитектуру приложения.
5. Разработать дизайн приложения.
6. Спроектировать и создать базу данных.
7. Реализовать функционал приложения.
8. Произвести тестирование разработанного программного обеспечения.

1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

1.1. ОБЗОР АНАЛОГОВ

Для создания информативного, удобного для пользователя и конкурентоспособного приложения, посвященного проведению соревнований по кёрлингу, необходимо рассмотреть решения, имеющиеся на рынке, проанализировать их достоинства и недостатки.

Рассмотрим следующие проекты:

1. **SoftPeelR** [8]

На данном сайте у незарегистрированных пользователей есть возможность ознакомиться со списком проведенных соревнований, а зарегистрированные пользователи могут сами создавать соревнования и вести их в режиме онлайн.

Преимущества данного сайта заключаются в том, что любой желающий может зарегистрироваться в системе и создать соревнование или турнир. Также предусмотрено несколько стандартных систем для их проведения (робин-раунд, шенкель, нокаут), поэтому система автоматически создает и публикует рейтинги в режиме онлайн.

Недостатки состоят в том, что нет поддержки сайта на русском языке (есть только английский, французский, немецкий, итальянский, польский языки), нельзя добавить свою систему для подсчета очков (это необходимо для тех случаев, когда в регламенте проведения соревнований/турниров есть особенности), на каждые соревнования необходимо каждый раз добавлять новых игроков, то есть нет возможности выбрать зарегистрированных участников.

2. **WorldCurling** [10]

Сайт использует Всемирная Федерация кёрлинга, соответственно на нем публикуются данные обо всех соревнованиях, проводимые данной организацией. В онлайн режиме ведется счет и статистика игр, также есть трансляции матчей. Недостатком данного веб-приложения является то, что у

пользователей, которые не имеют отношения к Всемирной Федерации кёрлинга, нет возможности воспользоваться всеми преимуществами данного сайта, можно лишь наблюдать за текущими и прошедшими соревнованиями/турнирами, но создать собственное соревнование не получится.

3. **Curling** [9]

До недавнего времени на сайте был лишь календарь с предстоящими российскими и международными соревнованиями на ближайший год, а также публиковались основные результаты соревнований и новости, которые касались кёрлинга в России и в мире. Недавно сайт усовершенствовали, и появилась возможность наблюдать за ходом соревнований и просматривать итоговые результаты, но счет на сайте появляется после окончания игры, турнирные таблица также выкладывают после того, как просчитают все вручную либо с применением дополнительных программ. Таким образом, данный сайт в большей степени является лишь информационной страничкой.

1.2. АНАЛИЗ ОСНОВНЫХ ТЕХНОЛОГИЧЕСКИХ РЕШЕНИЙ

1.2.1. ВЫБОР ФРЕЙМВОРКОВ ДЛЯ ВЕБ-ПРИЛОЖЕНИЯ

Существует множество фреймворков для создания веб-приложений. Наиболее популярными являются Node JS, Angular, React, NET Core.

1. **Node JS** [7]

Фреймворк с открытым кодом, который применяется для создания как сетевых, так и серверных программ. Есть возможность запускать на распределённых устройствах. Имеет событийно-ориентированный подход, который не блокирует ввод или отображение модели. Такой подход повышает эффективность использования системных ресурсов и уменьшает скорость работы самой платформы. Подходит для приложений, работающих в потоковом режиме. Но ограниченный набор стабильных библиотек; низкая стабильность интерфейса для разработки приложений, делают его неудачным выбором для разработки крупных или сложных приложений.

2. **Angular** [7]

MVC-платформа с открытым исходным кодом, которую можно применять для создания как одностраничных приложений, так и приложений для мобильного телефона. Поддерживает установку зависимостей, комплексную обработку и декларативные шаблоны. Программирование на Angular JS обеспечивает высокую эффективность готового приложения, отличную производительность и множество инструментария, но из-за внедрения многовариантных решений фреймворк довольно сложен в изучении.

3. **React** [7]

Считается одной их лучших платформ для разработки больших веб-приложений, которые основаны на задействование данных и их обработке без обновления веб-страницы. Легко используется в сочетании с JS-библиотеками и Angular.js. Фреймворк позволяет быстрее создавать приложения, а также они получаются более масштабируемыми. Недостатком фреймворка является неполное MVC, из-за чего код из React без корректировки невозможно перенести на прочие платформы или библиотеки.

4. **.NET Core**. [12]

Платформа .NET Core поддерживает четыре кроссплатформенных сценария: веб-приложения ASP.NET Core, приложения командной строки, библиотеки и приложения универсальной платформы Windows .

Элементы, которые делают ASP.NET Core правильным выбором для разработки корпоративных приложений:

1. Архитектура MVC

С MVC разработка веб-приложений стала более естественной, и рабочий процесс стал более эффективным. Кроме того, фреймворк ASP.NET Core также помогает в разработке веб-API и веб-приложений, которые можно достаточно точно тестировать, обеспечивая четкое разделение функционала. Таким образом, ASP.NET Core упрощает разработку, компиляцию и

тестирование программы разработчиками в модели, представлении или контроллере.

2. Функциональность Razor Pages

Razor Pages – это новый элемент ASP.NET Core, который делает программные сценарии, основанные на веб-страницах, более производительными. С технической точки зрения Razor Pages – это модель кодирования, основанная на веб-страницах, которая упрощает создание веб-интерфейса. При работе с фреймворком ASP.NET MVC классы контроллера заполнены большим количеством команд. И дело не только в этом. Они также растут по мере добавления новых элементов. С применением Razor Pages каждая веб-страница становится автономной с компонентом View, код также четко налажен.

3. Улучшение совместной работы и кросс-платформенной поддержки

.NET Core - это кросс-платформенный фреймворк, то есть приложения, построенные с использованием этого фреймворка, могут работать в операционных системах Windows, Linux и MacOS. Кроме того, разработчики могут свободно выбирать ОС разработки. Таким образом, разработчики могут работать в Linux, MacOS или Windows и при этом по-прежнему совместно работать над одним проектом. Это возможно благодаря унифицированной документации, которую предлагает среда разработки Visual Studio. Таким образом, фреймворк .NET Core позволяет создавать и запускать веб-приложения в Windows, Linux и Mac OS.

4. Встроенная поддержка внедрения зависимостей (Dependency Injection)

Фреймворк ASP.NET Core обеспечивает встроенную поддержку внедрения зависимостей. Dependency Injection – это, можно сказать, шаблон, который помогает разработчику отделить различные части своих приложений. С помощью внедрения зависимостей, службы (например, контекст базы данных Entity Framework Core) регистрируются во время запуска приложения.

Затем компоненты, которые используют эти службы (например, Razor Pages), обращаются к ним через параметры конструктора. Это означает, что разработчики больше не ограничены в выборе веб-приложений, и они могут использовать новые библиотеки в более событийно-ориентированных приложениях. Таким образом, внедрение зависимости в фреймворк ASP.NET Core улучшает возможность тестирования и расширяемость веб-приложений

Подведя итоги, можно сказать, что .NET Core дает начало новой эре разработки корпоративных веб-приложений. Благодаря постоянному совершенствованию возможностей ASP.NET, жизнь разработчика ASP.NET стала намного проще, а присутствие предприятий в интернете стало более прибыльным, делая ASP.NET Core главным выбором для создания веб-приложений для организаций. Также .NET Core – проект с открытым исходным кодом, поэтому можно наблюдать за его развитием и поддерживать его на GitHub.

ASP.NET Core Razor Pages [13] – это ориентированная на страницы инфраструктура для создания динамических веб-сайтов, управляемых данными, с четким разделением задач. Razor Pages поддерживает межплатформенную разработку и может быть развернут в операционных системах Windows, Unix и MacOS.

Платформа Razor Pages легкая и очень гибкая. Он предоставляет разработчику полный контроль над отображаемым HTML. Фреймворк построен поверх ASP.NET Core MVC и включен по умолчанию, когда MVC включен в приложении .NET Core. Razor Pages – это рекомендуемая среда для кросс-платформенной генерации HTML на стороне сервера в .NET Core.

Также понадобится CSS – формальный язык описания внешнего вида документа, написанного с использованием языка разметки.

1.2.2. ВЫБОР СУБД

Базы данных – это специально разработанное хранилище для различных типов данных. Каждая база данных имеет определённую модель (реляционная, сетевая, документно-ориентированная и др.), которая обеспечивает удобный доступ к данным. Системы управления базами данных (СУБД) – специальные приложения (или библиотеки) для управления базами данных различных размеров и форм.

Критериями [4] для выбора СУБД были: производительность, стабильность, масштабируемость, возможность разработки веб-приложений, качество и полнота документации, поддержка архитектуры клиент-сервер, совместимость работы с .NET Core.

Нередко возникает ситуация, когда модель меняется. Но при этом уже имеется существующая база данных, в которой есть какие-то данные. Чтобы без потерь обновить базу данных, существует такой механизм как миграция. Функция миграции в Entity Framework Core [14] позволяет последовательно применять изменения схемы к базе данных, чтобы синхронизировать ее с моделью данных в приложении без потери существующих данных.

Миграции включают средства командной строки и API-интерфейсы, которые помогают в решении следующих задач:

1. Создание миграции.
2. Обновление базы данных.
3. Настройка кода миграции.
4. Удаление миграции.
5. Отмена миграции.
6. Создание скриптов SQL.
7. Применение миграции во время выполнения.

Entity Framework Core работает со следующими наиболее популярными СУБД: SQL Server, MySQL, PostgreSQL, SQLite.

SQLite [5]

Легко встраиваемая в приложения СУБД. Так как эта система базируется на файлах, она предоставляет довольно широкий набор инструментов для работы с ней по сравнению с сетевыми СУБД. При работе с этой СУБД обращения происходят напрямую к файлам (в этих файлах хранятся данные) вместо портов и сокетов в сетевых СУБД. Именно поэтому SQLite очень быстрая, а также мощная благодаря технологиям обслуживающих библиотек.

Преимущества SQLite:

1. Файловая структура - вся база данных состоит из одного файла, поэтому её очень легко переносить на разные машины.

2. Используемые стандарты – хотя может показаться, что эта СУБД примитивная, но она использует SQL. Некоторые особенности опущены (RIGHT OUTER JOIN или FOR EACH STATEMENT), но основные все-таки поддерживаются.

3. Отлично подходит при разработке и тестировании. SQLite предлагает всё, что необходимо для этих целей, так как состоит всего из одного файла и библиотеки, написанной на языке C.

Недостатки SQLite:

1. Отсутствие системы пользователей – более крупные СУБД имеют системы управления правами доступа пользователей. Обычно применение этой функции не критично, так как эта СУБД используется в небольших приложениях.

2. Отсутствие возможности увеличения производительности – опять, исходя из проектирования, довольно сложно выжать что-то более производительное из этой СУБД.

Таким образом, SQLite стоит использовать, когда важна возможность легкого переноса приложения и не важна масштабируемость, когда необходимо напрямую обращаться к диску, а также есть необходимость использования дополнительных процессов при тестировании. Но если речь идет о многопользовательском приложении, в котором доступ к данным

осуществляется несколькими пользователями одновременно, да к тому же присутствует разграничение прав пользователей либо идет работа с большим объемом данных, то данная СУБД будет неприемлема для работы.

MySQL [5]

Это одна из самых распространенных полноценных серверных СУБД. MySQL очень функциональная, свободно распространяемая СУБД, которая успешно работает с различными сайтами и веб-приложениями. Обучиться использованию этой СУБД довольно просто, так как существует большое количество документации. Несмотря на то, что в ней не реализован весь SQL функционал, MySQL предлагает довольно много инструментов для разработки приложений. Так как это серверная СУБД, приложения для доступа к данным, в отличии от SQLite работают со службами MySQL.

Преимущества MySQL:

1. Простота в работе – установить MySQL довольно просто. Дополнительные приложения, например GUI, позволяет довольно легко работать с БД.
2. Богатый функционал – MySQL поддерживает большинство функционала SQL.
3. Безопасность – большое количество функций, обеспечивающих безопасность, которые поддерживаются по умолчанию.
4. Масштабируемость – MySQL легко работает с большими объемами данных и легко масштабируется.
5. Скорость – упрощение некоторых стандартов позволяет MySQL значительно увеличить производительность.

Недостатки MySQL:

1. Известные ограничения – по задумке в MySQL заложены некоторые ограничения функционала, которые иногда необходимы в особо требовательных приложениях.

2. Проблемы с надежностью – из-за некоторых способов обработки данных MySQL (связи, транзакции, аудиты) иногда уступает другим СУБД по надежности.

3. Медленная разработка – хотя MySQL это технически открытое ПО, существуют жалобы на процесс разработки.

Таким образом, MySQL можно применять, когда важными критериями являются безопасность, работа с распределенными операциями, работа с веб-приложениями, а также масштабируемость. Но если необходима надежная, быстрая, многопоточная СУБД, то от MySQL лучше отказаться.

PostgreSQL [5]

PostgreSQL является самой профессиональной из выше рассмотренных СУБД. Она свободно распространяемая и максимально соответствует стандартам SQL. В PostgreSQL стараются полностью применять ANSI/ISO SQL стандарты своевременно с выходом новых версий.

От других СУБД PostgreSQL отличается поддержкой востребованного объектно-ориентированного и/или реляционного подхода к базам данных. Например, полная поддержка надежных транзакций, то есть. атомарность, последовательность, изоляционность, прочность (Atomicity, Consistency, Isolation, Durability (ACID).) Благодаря мощным технологиям PostgreSQL имеет высокий уровень производительности. Параллельность достигнута не за счет блокировки операций чтения, а благодаря реализации управления многовариантным параллелизмом (MVCC), что также обеспечивает соответствие ACID. PostgreSQL очень легко расширять своими процедурами, которые называются хранимые процедуры. Эти функции упрощают использование постоянно повторяемых операций.

Достоинства PostgreSQL:

1. Открытое программное обеспечение, соответствующее стандарту SQL. PostgreSQL – бесплатное программное обеспечение (ПО) с открытым исходным кодом. Эта СУБД является очень мощной системой.

2. Большое сообщество – существует довольно большое сообщество, в котором можно найдёте ответы на интересующие вопросы.

3. Большое количество дополнений – несмотря на огромное количество встроенных функций, существует очень много дополнений, позволяющих разрабатывать данные для этой СУБД и управлять ими.

4. Расширения – существует возможность расширения функционала за счет сохранения своих процедур.

5. Объектность – PostrgreSQL это не только реляционная СУБД, но также и объектно-ориентированная с поддержкой наследования и много другого.

Недостатки PostgreSQL:

1. Производительность – при простых операциях чтения PostgreSQL может значительно замедлить сервер и быть медленнее своих конкурентов.

2. Популярность – популярностью эта СУБД похвастаться не может, хотя и присутствует довольно большое сообщество.

3. Хостинг – в силу выше перечисленных факторов иногда довольно сложно найти хостинг с поддержкой этой СУБД.

Таким образом, если основными критериями при выборе СУБД будут целостность данных, использование сложных пользовательских процедур и сложных структур данных, то можно выбрать PostgreSQL. Но если скорость, простая настройка и репликации являются так же важными критериями, то лучше выбрать другую СУБД.

MS SQL Server [3]

MS SQL Server – это платформа для решения критически важных задач в масштабе предприятия, обладающая высокой доступностью, повышенной производительностью и безопасностью. Решение представляет собой хорошо масштабируемый, полностью реляционный, быстродействующий сервер, способный обрабатывать большие объемы данных для клиент-серверных приложений. Рекордная производительность MS SQL Server обеспечивается новыми технологиями работы с памятью, что помогает предприятиям

ускорить свой бизнес и реализовать новые сценарии работы. Кроме того, SQL Server позволяет использовать новые гибридные облачные решения и пользоваться новыми преимуществами облачных вычислений. Расширенные функции безопасности, в сочетании с встроенными, удобными для использования инструментами и управляемым доступом к данным, позволяют организации выполнить требования строгих политик соответствия нормам.

Достоинства MS SQL Server:

1. Обеспечивается максимальная безопасность. Все данные защищены от несанкционированного доступа за счет интеграции сетевой безопасности с сервером безопасности. Поскольку безопасность поддерживается на уровне пользователя, пользователи могут иметь ограниченный доступ к данным, тем самым защищая их от модификации или поиска, указав доступ на уровне пользовательских привилегией. Кроме того, с данными, хранящимися на отдельном сервере, сервер работает как шлюз, который ограничивает несанкционированный доступ.

2. Техническое обслуживание SQL Server очень простое и не требует больших знаний. Возможны изменения в структуре данных, а также резервное копирование во время работы сервера без остановки.

3. SQL Server является приложением базы данных при работе на .Net. При выборе Microsoft SQL Sever в качестве базы данных информации для компании, приложение может расширяться и адаптироваться по мере изменения бизнес-климата.

4. Также следуют отметить, что в MS SQL Server 2017 введена поддержка SQL Server в Linux.

Таким образом, Microsoft SQL Sever удовлетворяет всем критериям выбора СУБД для разработки веб – приложения для организации и проведения соревнований по кёрлингу.

Для данной СУБД необходима лицензия. Была выбрана Microsoft SQL Sever Express Edition [11], так как это бесплатная СУБД начального уровня, которая идеально подходит для обучения и создания приложений для

обработки данных на настольных компьютерах и небольших серверах (размером до 10 ГБ).

1.2.3. ВЫБРОР ЯЗЫКОВ ПРОГРАММИРОВАНИЯ

.NET Core [15] позволяет создавать приложения и библиотеки на языках C#, Visual Basic и F#. Эти языки уже интегрированы или могут быть интегрированы в текстовые редакторы и интегрированные среды разработки, такие как Visual Studio, Visual Studio Code, Sublime Text и Vim.

В качестве языка программирования использовался C#, так как он является основным языком платформы .NET.

C# – мощный объектно-ориентированный язык программирования, который предоставляет готовые решения для некоторых задач в программировании. C# является типобезопасным языком программирования, поэтому по умолчанию безопасными считаются только неявные преобразования. Это обеспечивается во время компиляции, а также в некоторых случаях во время выполнения.

Еще одним преимуществом C# является его относительная простота в освоении, что делает его отличным выбором как для начинающих, так и для уже опытных разработчиков.

Razor Pages использует C # для программирования на стороне сервера и простой в освоении шаблонный синтаксис Razor для встраивания C# в разметку HTML для динамической генерации контента для браузеров.

Также будет использоваться CSS – формальный язык описания внешнего вида документа, написанного с использованием языка разметки.

1.3. ВЫВОД

Челябинская областная общественная организация «Федерация кёрлинга Челябинской области» нуждается в разработке приложения для управления данными о мероприятиях (соревнованиях, турнирах) и их участниках.

Посредством веб-интерфейса система должна предоставлять следующие функции:

1. Ведение базы спортсменов, тренеров, судей, членов ФК.
2. Создание и редактирование соревнований.
3. Ведение личного кабинета спортсменов, тренеров, судей, членов ФК.
4. Выделение списка спортсменов, судей из базы данных для каждого мероприятия.
5. Закрепление протокола соревнований за организатором мероприятия.
6. Хранение результатов всех соревнований.
7. Ведение счета онлайн для каждой игры.
8. Возможность поиска и фильтрации среди спортсменов, тренеров, судей, соревнований.

Основные техничекие решения:

Проектирование структуры базы данных осуществляется с помощью СУБД MS SQL Server Express Edition с применением Entity Framework Core для миграций.

Разработка серверной части веб-приложения происходит на языке C# с использование фреймворка ASP.NET Core.

Для внешнего оформления веб-приложения используется C # для программирования на стороне сервера и простой в освоении шаблонный синтаксис Razor для встраивания C# в разметку HTML для динамической генерации контента для браузеров. Также понадобится CSS – формальный

язык описания внешнего вида документа, написанного с использованием языка разметки.

2. ОПРЕДЕЛЕНИЕ ТРЕБОВАНИЙ

2.1. ТРЕБОВАНИЯ К СТРУКТУРЕ И ФУНКЦИОНИРОВАНИЮ СИСТЕМЫ

Система должна поддерживать следующую ролевую модель, а также возможность управления ролями и их правами.

Основные ролевые группы и их состав:

1. Неавторизованный пользователь.

Пользователь должен получать актуальную доступную запрашиваемую информацию в режиме онлайн.

Пользователь может: просматривать списки соревнований и их результаты (в ограниченном объеме: указаны только место, команда и ФИО спортсменов), а также просматривать результаты соревнований (счет) в режиме онлайн, осуществлять взаимодействие с формами регистрации и авторизации.

2. Зарегистрированный пользователь: судья, тренер, спортсмен, член ФК, администратор системы (один человек может иметь сразу несколько ролей, для этого у него должны быть отдельные вкладки).

Зарегистрированный пользователь должен наследовать права и возможности неавторизованного, должен иметь возможность получать персональные уведомления, восстанавливать пароль через электронную почту, редактировать данные профиля. Доступ к различным функциям должен разграничиваться в зависимости от того, какая роль присвоена пользователю.

2.1. Судья должен иметь следующие возможности: просматривать и изменять счет на турах, которые он судит. После того, как тур окончен, судья уже не может вносить изменения. Их может вносить только создатель соревнований. Все соревнования, в которых принимал участие судья, должны

быть сохранены в таблице, так называемой судейской книжке (название соревнований, город, дата начала, дата окончания, роль судьи).

2.2. Тренер должен иметь следующие возможности:

1. Получать полную информацию о спортсменах, которых он тренирует.
2. Осуществлять поиск спортсменов с помощью фильтров.
3. Осуществлять поиск среди тренеров и судей с помощью фильтров.
4. Подавать заявки на участие спортсменов в соревнованиях.

2.3. Спортсмен должен иметь следующие возможности:

1. Осуществлять поиск спортсменов с помощью фильтров
2. Осуществлять поиск среди тренеров и судей с помощью фильтров.

Соревнования, в которых участвовал спортсмен, должны быть сохранены (название соревнований, дата начала, дата окончания, место на соревнованиях).

2.4. Администратор должен иметь доступ ко всем данным, должен иметь возможность их изменять, разрешать редактирование, должен проверять личную информацию, созданную или изменённую пользователями.

2.5. Член ФК имеет права незарегистрированного пользователя и в зависимости от должности должен запрашивать у администратора права на те или иные действия.

2.6. Организатором соревнований должен иметь возможность выступать любой пользователь, и для этих соревнований пользователь будет наделяться следующими правами:

1. Добавлять и редактировать все данные о соревнованиях, в том числе добавлять команды и необходимую информацию в турнирную таблицу

2. Выбирать судей с помощью фильтров.

3. Назначать роли судьям (главный судья, главный секретарь, помощник главного судьи, помощник главного секретаря, судья тура, старший судья тура, хронометрист).

СВОЙСТВА СОРЕВНОВАНИЙ

На рисунке 2.1 показано, как должны проходить соревнования.

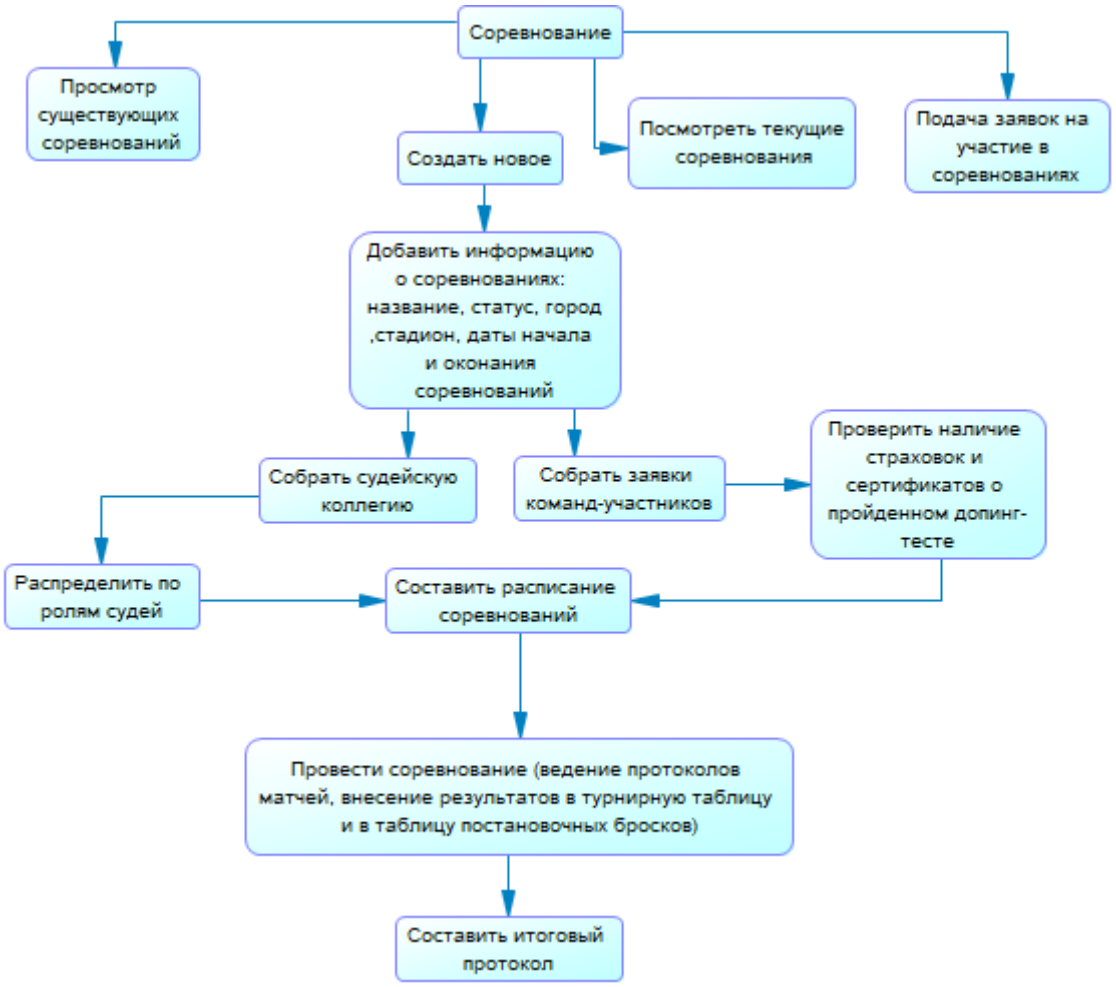


Рисунок 2.1 – Создание соревнований

Соревнование должно содержать следующие свойства:

1. Название – строка.

2. Город - выпадающий список (варианты подбираются по мере набора в текстовом поле) с возможностью добавления нового города.

3. Стадион - выпадающий список (варианты подбираются по мере набора в текстовом поле) с возможностью добавления нового стадиона.

4. Дата начала - строка формата dd.mm.yyyy.

5. Дата окончания - строка формата dd.mm.yyyy.
6. Краткая информация (по усмотрению организатора).
7. Детальная информация (по усмотрению организатора).
8. Список команд.
9. Возможность разделения команд на группы.
10. Вкладка с расписанием игр.
11. Протоколы матчей, турнирные таблицы, итоговый протокол.

Предусмотреть механизмы автоматизированной выгрузки протоколов в структурированном формате (Excel или CSV).

СИСТЕМА УПРАВЛЕНИЯ ПОЛЬЗОВАТЕЛЯМИ

Система управления пользователями должна позволять пользователям регистрироваться в системе (требования к регистрации смотри на рисунке 2.2), редактировать свой профиль, принимать участие в соревнованиях.

Рисунок 2.2 – Поля при регистрации пользователя

Личный кабинет пользователя должен содержать следующие поля, обязательные для заполнения:

1. Фамилия - строковый тип.
2. Имя - строковый тип.
3. Отчество (если есть) - строковый тип.
4. Школа/клуб - строковый тип.
5. Email - строковый тип
6. Пароль и поле для подтверждения - строковый тип (не менее 6 символов)
7. Пол - (Мужской/Женский).
8. Дата рождениям – строковый тип формата dd-mm-yyyy.
9. Телефон - Строковый тип формата +(код)-(номер), несмотря на то, что тип - строковый, допускаются только числовые символы [0-9].
10. Город- выпадающий список (варианты подбираются по мере набора в текстовом поле) с возможностью добавления нового города.
11. Действующий разряд - строковый тип.
12. Тренер.
13. Согласие на обработку персональных данных – чекбокс.

Интерфейс управления участниками мероприятия для организаторов должен позволять:

1. Просматривать список зарегистрированных участников.
2. Осуществлять поиск и фильтрацию среди участников по ФИО, городу, клубу/школе, возрасту.
3. Позволять редактировать информацию об участии пользователей в мероприятии.

Система должна предоставлять возможность поиска по соревнованиям и пользователям.

Критерии поиска и фильтрации для тренеров:

1. Пол.
2. Возраст (от, до, диапазон).
3. Город.
4. Клуб/школа.

Критерии поиска и фильтрации для спортсменов:

1. Пол.
2. Возраст (от, до, диапазон).
3. Город.
4. Клуб/школа.
5. Спортивный разряд.

Критерии поиска и фильтрации для судей:

1. Пол.
2. Возраст (от, до, диапазон).
3. Город.
4. Категория.

Критерии поиска и фильтрации для соревнований:

1. Категория.
2. Город.
3. Ранг.
4. Тип (мужские, женские, смешанные).
5. Год.

ЭЛЕКТРОННЫЕ РАССЫЛКИ

Модуль должен позволять организаторам мероприятий и администратору осуществлять рассылки по базе участников.

Организаторы мероприятий должны иметь доступ только к базе участников подотчетного им мероприятия.

Администраторы должны иметь возможность осуществлять рассылки по всей базе участников.

В модуле должна быть предусмотрена возможность гибкого таргетинга получателей (по мероприятиям, в которых участник принимал участие, по региону, полу). Функционал модуля рассылок должен предусматривать возможность создания ссылок «быстрой авторизации», то есть автоматически по адресу электронной почты идентифицировать и авторизовать пользователя при переходе по специальной ссылке из рассылки.

2.2. ТРЕБОВАНИЯ К НАДЕЖНОСТИ

Для восстановления информации необходимо обеспечить автоматизированное резервное копирование по расписанию, установленному администратором системы. Копии должны храниться на сервере.

Также для обеспечения надежности системы необходимы проверки на ввод неправильных и недостоверных данных. При регистрации нового пользователя достоверность введенных им данных должен проверить администратор. При положительном результате пользователь сможет авторизоваться. Также при смене личных данных пользователь должен отравлять запрос администратору на смену этих данных.

2.3. ТРЕБОВАНИЯ К БЕЗОПАСНОСТИ И ЗАЩИТЕ ИНФОРМАЦИИ ОТ НЕСАНКЦИОНИРОВАННОГО ДОСТУПА

Необходимо разграничить пользователей по категориям: администратор, тренер, спортсмен, судья. Доступ к данным должен осуществляться в соответствии с правилами категорий пользователей.

Все пароли пользователей должны быть зашифрованы любым алгоритмом.

Учетные записи пользователей должны блокироваться после 5 неудачных попыток захода. Исключение делается только для главной учетной записи администратора.

Администратор должен иметь возможность запретить пользователям вход в систему.

Все пароли должны содержать буквы и цифры, минимальная длина пароля должна быть регулируемой администратором.

2.4. ТРЕБОВАНИЯ К ТЕХНИЧЕСКОЙ ЭСТЕТИКЕ

Дизайн веб-приложения должен удовлетворять следующим требованиям по эргономике и технической эстетике:

1. Адекватно отображаться в зависимости от типа подключения пользователя.
2. Корректно отображаться при всех возможных разрешениях и количестве отображаемых цветов монитора.
3. Сохранять идентичность отображения на большинстве современных ОС и браузерах.
4. Обладать системой подсказок в местах, где у пользователя потенциально могут возникнуть затруднения.
5. Обеспечивать отсутствие искажений при распечатке страниц веб-приложения на принтере.

2.5. ТРЕБОВАНИЯ К ПАТЕНТНОЙ ЧИСТОТЕ

Должна быть обеспечена патентная чистота разрабатываемой информационной системы и ее частей согласно действующему законодательству Российской Федерации.

3. ПРОЕКТИРОВАНИЕ СИСТЕМЫ

3.1. АРХИТЕТУРА ПРОГРАММЫ

Как правило компьютеры и программы, входящие в состав информационной системы, не являются равноправными. Некоторые из них владеют ресурсами (файловая система, процессор, принтер, база данных и т.д.), другие имеют возможность обращаться к этим ресурсам. Компьютер (или программа), управляющий ресурсом, называют сервером этого ресурса (файл-сервер, сервер базы данных, вычислительный сервер). Клиент и сервер какого-либо ресурса могут находиться как в рамках одной вычислительной системы, так и на различных компьютерах, связанных сетью.

Основной принцип технологии "клиент-сервер" [14] заключается в разделении функций приложения на три группы:

1. Ввод и отображение данных (взаимодействие с пользователем).
2. Прикладные функции, характерные для данной предметной области.
3. Функции управления ресурсами (файловой системой, базой даных и т.д.).

Поэтому в любом приложении выделяются следующие компоненты:

1. Компонент представления данных, отвечает за пользовательский интерфейс.
2. Прикладной компонент, реализует алгоритм решения конкретной задачи.

3. Компонент управления ресурсом, обеспечивает доступ к необходимым ресурсам.

Архитектура «клиент-сервер» определяет общие принципы организации взаимодействия в сети, где имеются серверы, узлы-поставщики некоторых специфичных функций (сервисов) и клиенты, потребители этих функций.

Виды «клиент-серверной» архитектуры:

1. Двухзвенная (смотри рисунок 3.1)

Двухзвенная архитектура используется в клиент-серверных системах, где сервер отвечает на клиентские запросы напрямую и в полном объеме, при этом используя только собственные ресурсы. Таким образом получается, что сервер не вызывает сторонние сетевые приложения и не обращается к сторонним ресурсам для выполнения какой-либо части запроса.

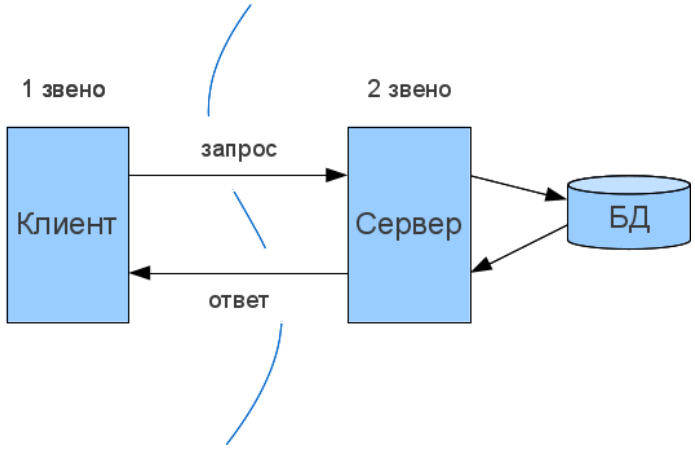


Рисунок 3.1 – Двухзвенная клиент-серверная архитектура

2. Трехзвенная (смотри рисунок 3.2)

Еще одна тенденция в клиент-серверных технологиях связана со все большим использованием распределенных вычислений. Они реализуются на основе модели сервера приложений, где сетевое приложение разделено на две и более частей, каждая из которых может выполняться на отдельном компьютере. Выделенные части приложения взаимодействуют друг с другом, обмениваясь сообщениями в заранее согласованном формате. В этом случае клиент-серверная архитектура становится трехзвенной**.** Как правило, третьим

звеном в трехзвенной архитектуре становится сервер приложений, т.е. компоненты распределяются следующим образом:

а) Представление данных – на стороне клиента.

б) Прикладной компонент – на выделенном сервере приложений.

в) Управление ресурсами – на сервере БД, который и представляет запрашиваемые данные.

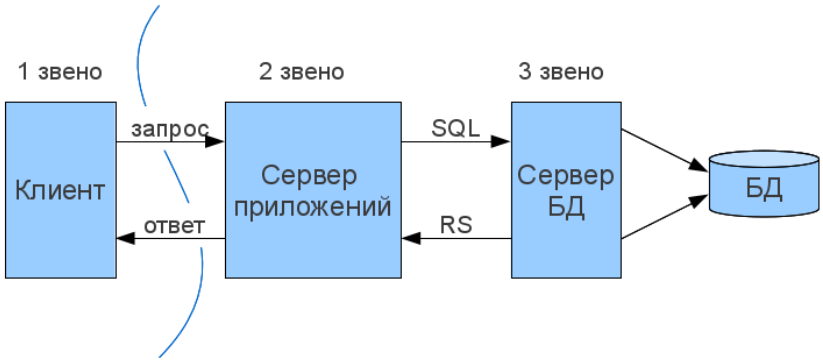


Рисунок 3.2 – трехзвенная архитектура «клиент-сервер»

3. Многозвенная (смотри рисунок 3.3)

Трехзвенная архитектура может быть расширена до многозвенной путем выделения дополнительных серверов, каждый из которых будет представлять собственные сервисы и пользоваться услугами прочих серверов разного уровня.

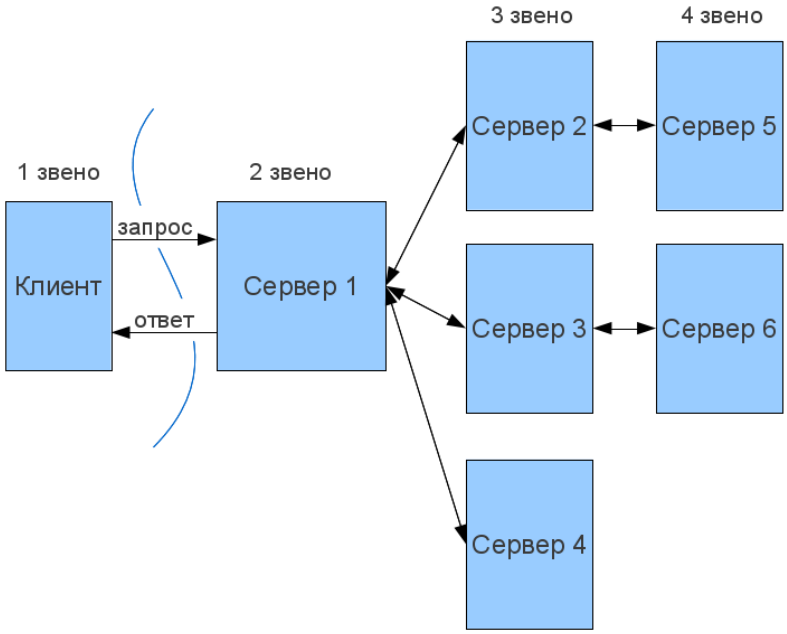


Рисунок 3.3 – трехзвенная архитектура «клиент-сервер»

Для доступа к тем или иным сетевым сервисам используются клиенты, возможности которых характеризуются понятием «толщины». Оно определяет конфигурацию оборудования и программное обеспечение, имеющиеся у клиента.

Виды клиентов:

1. «Тонкий» клиент

Этот термин определяет клиента, вычислительных ресурсов которого достаточно лишь для запуска необходимого сетевого приложения через web-интерфейс. Пользовательский интерфейс такого приложения формируется средствами HTML. Вся прикладная логика выполняется на сервере. Для работы тонкого клиента достаточно лишь обеспечить возможность запуска web-браузера, в окне которого и осуществляются все действия. По этой причине web-браузер часто называют "универсальным клиентом".

2. «Толстый» клиент

«Толстым» клиентом является рабочая станция или персональный компьютер, работающие под управлением собственной дисковой операционной системы и имеющие необходимый набор программного обеспечения. К сетевым серверам «толстые» клиенты обращаются в основном за дополнительными услугами. Также под «толстым» клиентом подразумевается и клиентское сетевое приложение, запущенное под управлением локальной ОС. Такое приложение совмещает компонент представления данных (графический пользовательский интерфейс ОС) и прикладной компонент (вычислительные мощности клиентского компьютера).

3. «Rich» клиент

«Rich» клиент своего рода компромисс между «толстым» и «тонким» клиентом. Как и «тонкий» клиент, «rich»-клиент также представляет графический интерфейс, описываемый уже средствами XML и включающий некоторую функциональность толстых клиентов (например интерфейс drag-and-drop, вкладки, множественные окна, выпадающие меню и т.п.)

Прикладная логика «rich»-клиента также реализована на сервере. Данные отправляются в стандартном формате обмена, на основе того же и интерпретируются клиентом.

Вывод:

Для реализации данного веб-приложения будет удобно использовать трехзвенную архитектуру клиент-сервер, так как в проектируемой системе присутствует сервер БД. Для данной архитектуры будет использован «тонкий» клиент, так как большая часть задач по обработке информации перенесена на сервер приложения и права доступа клиента строго ограничены.

3.2 ВЫБОР АРХИТЕКТУРНОГО ШАБЛОНА ДЛЯ РАЗРАБОТКИ ВЕБ-ПРИЛОЖЕНИЯ

Шаблон проектирования – архитектурная конструкция, предназначенная для проектирования некоторых часто возникающих контекстов. Шаблоны являются наиболее похожими на готовые библиотеки.

Подходящими шаблонами для веб-приложений являются MVC. В контексте разработки веб-приложения возможно использовать классификацию шаблонов проектирования Мартина Фаулера [16]. Согласно ей шаблоны можно разделить по классам:

1. Базовые шаблоны.
2. Шаблоны веб-представления.
3. Шаблоны архитектурных источников данных.
4. Шаблон объектно-реляционной логики.
5. Шаблоны объектно-реляционного структурирования.
6. Шаблоны логики сущности.
7. Шаблоны распределения данных.
8. Шаблоны локальной конкуренции.

Любой из этих классов включает в себя некоторый набор шаблонов, одним из которых и является MVC (Model – View – Control), а также производные:

1. MVP (Model – View – Presenter).
2. MVVM (Model –View – View – Model).
3. HMVC (Hierarchical MVC).
4. PAC (Presentation – Abstraction – Control).

MVC [17] (см. рисунок 3.4) позволяет реализовать бизнес-логику приложения без необходимости затрачивать значительные усилия на программирование. Он разделяет работу веб-приложения на три отдельные функциональные роли: модель данных (model), пользовательский интерфейс (view) и управляющую логику (controller). Таким образом, изменения, вносимые в один из компонентов, оказывают минимально возможное воздействие на другие компоненты. В данном паттерне модель не зависит от представления или управляющей логики, что делает возможным проектирование модели как независимого компонента и, например, создавать несколько представлений для одной модели.

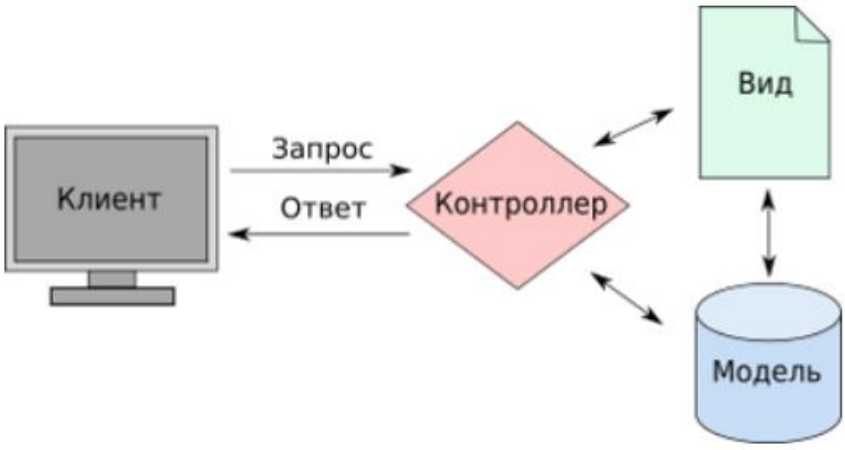


Рисунок 3.4 – концепция паттерна MVC

Паттерн PAC [18] (Presentation–Abstraction–Control) – программный архитектурный паттерн. Это ориентированная на взаимодействие программная архитектура, которая разделяет интерактивную систему на три

типа компонентов, отвечающих за конкретные аспекты функциональности приложения. Компонент абстракции извлекает и обрабатывает данные, компонент представления форматирует визуальное и звуковое представление данных, а компонент управления обрабатывает такие вещи, как поток управления и связь между двумя другими компонентами. В отличие от MVC, PAC используется в качестве иерархической структуры агентов, каждый из которых состоит из триады частей представления, абстракции и контроля (концепция паттерна PAC см. на рисунке 3.5). Агенты (или триады) связываются друг с другом только через контрольную часть каждой триады. Он также отличается от MVC тем, что в каждой триаде он полностью изолирует представление (представление в MVC) и абстракцию (модель в MVC).

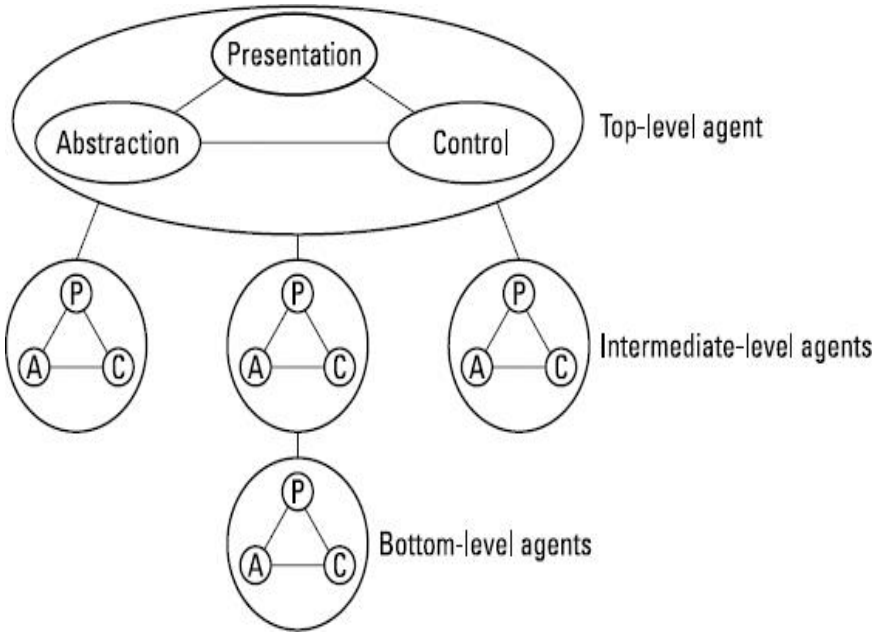


Рисунок 3.5 – концепция паттерна PAC

Паттерн HMVC [19] (Hierarchical Model–View–Controller) – каждая отдельная MVC триада используется в качестве слоя в иерархической структуре. При этом каждая триада в этой иерархии независима от других, и может обратиться к контроллеру другой триады. Такой подход существенно облегчает и ускоряет разработку сложных приложений, облегчает их дальнейшую поддержку и масштабирование, способствует повторному

использованию кода. Некоторые разработчики отмечают, что HMVC, по сути, является переосмыслением более строгого паттерна PAC. Концепция HMVC представлена на рисунке 3.6.

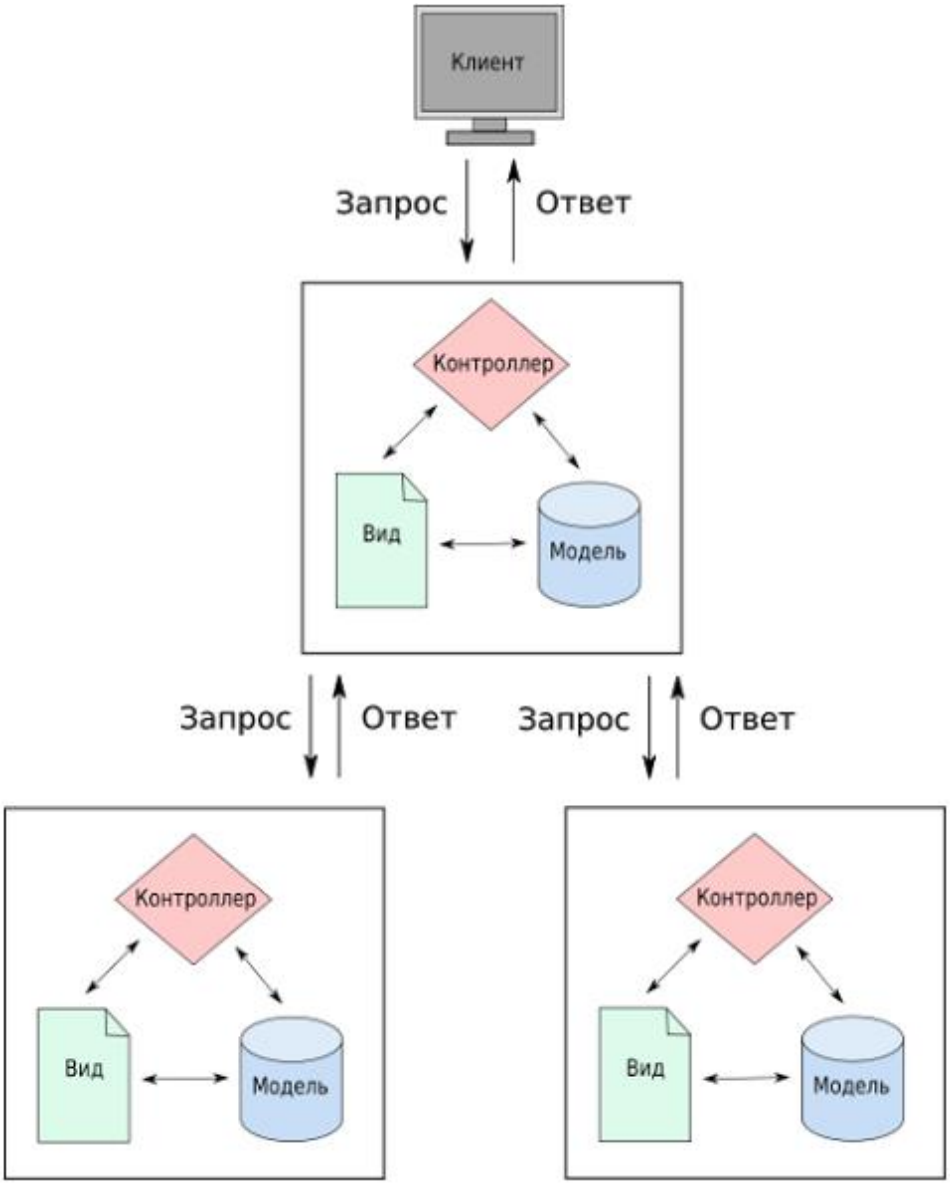


Рисунок 3.6 – концепция HMVC

Паттерн MVP [20] (Model-View-Presenter) – шаблон проектирования пользовательского интерфейса, специально разработанный для легкого автоматического тестирования и разделения ответственности в презентационной логике путем отделения логики от отображения. В MVP Presenter выполняет роль посредника, такую же, как контроллер в MVC. Также Presenter отвечает за управление событиями пользовательского интерфейса,

обработка которых в MVC отводится представлениям (View). Концепция данного паттерна представлена на рисунке 3.7.

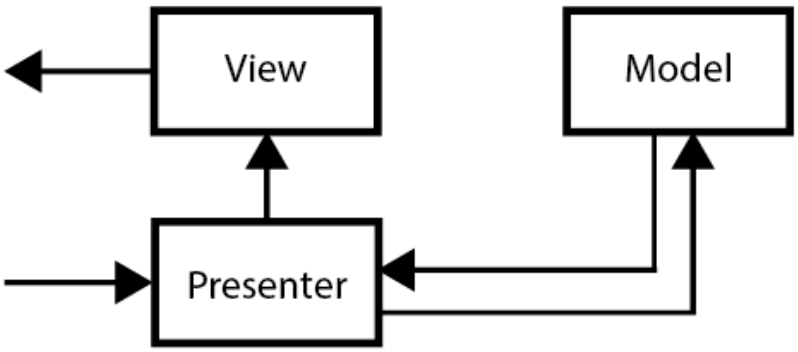


Рисунок 3.7 – концепция MVP

Паттерн MVVM [21] (Model-View-View-Model) – допускает отделение логики приложения от визуальной части (представления). Данный паттерн задает общую архитектуру приложения и имеет более тесную связь между моделью и представлением. Данная концепция реализована в WPF и Silverlight. На рисунке 3.8 приведена концепция MVVM.

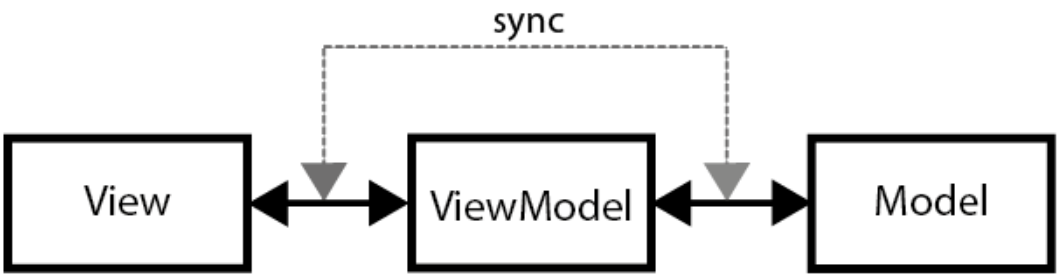


Рисунок 3.8 – концепция MVVM

Для реализации приложения был выбран шаблон MVC, поскольку он обеспечивает простую реализацию бизнес-логики и является типичным решением при разработке веб-приложений. Поэтому рассмотрим подробно компоненты MVC:

1. **Модель** [22]

Она содержит в себе всю логику приложения, хранит и обрабатывает данные, при этом не взаимодействуя с пользователем напрямую.Например,

сохранение информации в БД, проверка правильности введенных в форму данных – это задача Модели, получение этих данных от пользователя или вывод информации на экран или обработка нажатия на кнопку - нет.

Модель не должна никак зависеть и не должна ничего знать о Контроллерах и Видах. Модель – это не один класс или набор однотипных классов. Это основная часть приложения, которая может содержать много разных классов: сервисы, классы для взаимодействия с БД, сущности, валидаторы.

2. **Представление** [22]

Преставление – это код, который отвечает за отображение информации на экране, отрисовку кнопок и других элементов интерфейса. В веб-приложении оно обычно состоит из HTML-шаблонов страниц.

3. **Контроллер** [22]

Контроллер отвечает за выполнение запросов, пришедших от пользователя. В веб-приложении обычно контроллер обращается к модели, чтобы получить или изменить какие-то данные, и в конце вызывает Представление, чтобы отобразить результат выполнения запроса.

Один Контроллер может работать с несколькими Моделями, и наоборот, одна Модель может использоваться в нескольких Контроллерах.

В веб-приложении Контроллеры – это набор однотипных классов, каждому разделу на сайте соответствует свой класс, и в нем делаются методы (их называют "действия", "action").

Весь функционал приложения содержится в Модели. Контроллер и Представление предоставляют лишь возможность пользователю взаимодействовать с моделью и отображать данные из нее.

3.2.1 МОДЕЛЬ MVC В ВЕБ-ПРИЛОЖЕНИИ

В таблице 3. 1 представлены все контроллеры веб-приложения, а также их назначение и связь с представлениями и моделями.

Таблица 3.1 – Контроллеры веб-приложения

Наименование	**Назначение**	**Кол-во action, шт**	**Взаимодействие с другими компонентами**	
			Представления	**Модели**
EndingController	Работа с эндами	8	Create; Delete; Details; Edit; Index.	EndingModel
GameController	Работа с играми	8	Create; Delete; Details; Edit; Index.	GameModel
GroupController	Работа с группами	10	AddTeam; Create; Delete; Details; Edit; Index.	GroupModel
HomeController	Работа с домашней страницей, вкладкой «контакты»	5	About; Contact; Index; Privacy; Error.	
ScoreController	Работа с постановками	8	Create; Delete; Details; Edit; Index.	ScoreModel
TeamController	Работа с командами	8	Create; Delete; Details; Edit; Index.	TeamModel
TourController	Работа с турами	8	Create; Delete; Details; Edit; Index.	TourModel

Продолжение таблицы 3.1

Наименование	**Назначение**	**Кол-во action, шт**	**Взаимодействие с другими компонентами**	
			Представления	**Модели**
Tournament Controller	Работа с соревнованиями	14	_Layout; Create; Delete; Details; Edit; Index; Schedule; Standings; Summary; TeamList; TournamentProgress.	Tournament Model
TeamPlacement Controller	Работа с местами команд на групповом этапе	8	Create; Delete; Details; Edit; Index.	
TeamPlacemen GlobaltController	Работа с местами команд в итоговом протоколе	8	Create; Delete; Details; Edit; Index.	

В таблице 3.2 содержатся все модели и их назначение.

Таблица 3.2 – Модели веб-приложения

Наименование	**Назначение**
CityModel	Города
CoacheModel	Тренеры
EndingModel	Энды
ErrorViewModel	Ошибка представления
GameModel	Игры
GroupAssignmentModel	Принадлежность команды к группе
GroupModel	Группы
JudgeModel	Судьи
ScoreModel	Постановочные броски
TeamAssignmentModel	Принадлежность спортсмена к команде
TeamModel	Команды

Продолжение таблицы 3.2

Наименование	**Назначение**
TeamPlacement	Место в групповом этапе (турнирная таблица)
TeamPlacementGlobal	Итоговое место на соревнованиях
TeamTournamentAssignmentModel	Принадлежность команды к соревнованию
TourModel	Тур
TournamentModel	Соревнование

В таблице 3.3 содержатся представления веб-приложения, их назначение, а также указана принадлежность контроллеру.

Таблица 3.3 – представления веб-приложения

Наименование контроллера, к которому идет привязка	**Наименование представления**	**Назначение представления**
EndingController	Create	Создать
	Delete	Удалить
	Details	Подробное описание
	Edit	Редактировать
	Index	Список эндов
GameController	Create	Создать
	Delete	Удалить
	Details	Подробное описание
	Edit	Редактировать
	Index	Список игр
GroupController	AddTeam	Добавить команду
	Create	Создать
	Delete	Удалить
	Details	Подробное описание
	Edit	Редактировать
	Index	Список групп
HomeController	About	Страница «о веб-сайте»
	Contact	Контакты
	Index	Пустая страница, перенаправление к Index у TournamentController
	Privacy	Конфиденциальность

Продолжение таблицы 3.3

Наименование контроллера, к которому идет привязка	**Наименование представления**	**Назначение представления**
ScoreController	Create	Создать
	Delete	Удалить
	Details	Подробное описание
	Edit	Редактировать
	Index	Список постановочных бросков
Shared (папка, которая хранит общие представления для всех контроллеров)	_Cookie Consent Partial	Доступ к куки браузера
	_Layout	Мастер-страницы
	_LoginPartial	Шаблон для страницы логина
	_ValidationScriptsPartial	Частичное представление, которое подключает скрипты валидации формы.
	_Error	Отображение ошибок
TeamController	Create	Создать
	Delete	Удалить
	Details	Подробное описание
	Edit	Редактировать
	Index	Список команд
TourController	Create	Создать
	Delete	Удалить
	Details	Подробное описание
	Edit	Редактировать
	Index	Список туров
TeamPlacementController	Create	Создать
	Delete	Удалить
	Details	Подробное описание
	Edit	Редактировать
	Index	Список присвоенных мест
TeamPlacementGlobalController	Create	Создать
	Delete	Удалить
	Details	Подробное описание
	Edit	Редактировать
	Index	Список мест в итоговом протоколе

Продолжение таблицы 3.3

Наименование контроллера, к которому идет привязка	**Наименование представления**	**Назначение представления**
Tournament	_Layout	Общие элементы для всех представлений
	Create	Создать
	Delete	Удалить
	Details	Подробное описание
	Edit	Редактировать
	Index	Список соревнований
	Schedule	Расписание игр
	Standings	Таблица со счетом
	Summary	Итоговый протокол
	TeamList	Список всех команд
	TournamentProgress	Список туров с играми
Эти файлы содержат код, который автоматически добавляется ко всем представлениям.	_ViewImports	Устанавливает некоторые общие для всех представлений пространства имен
	_ViewStart	Устанавливает общую мастер-страницу. Код этого файла добавляется в самое начало кода преставлений при их запуске.

Для решения задач, связанных с вводом-выводом (например, для запроса данных из сети или доступа к базе данных), желательно использовать асинхронное программирование [22].

В C# имеется модель асинхронного программирования, реализованная на уровне языка, которая позволяет легко писать асинхронный код, не прибегая к обратным вызовам или библиотекам, которые поддерживают асинхронность.

В основе асинхронного программирования лежат объекты Task и Task<T>, которые моделируют асинхронные операции. Они поддерживаются ключевыми словами async и await. В большинстве случаев модель достаточно проста.

Для кода, ограниченного производительностью ввода-вывода, с помощью await выполняется операция, которая возвращает объект Task или Task<T> внутри метода async.

Именно с помощью ключевого слова await передается управление вызывающему объекту метода, который выполнил await, позволяя, таким образом, пользовательскому интерфейсу или службе отвечать на запросы.

Когда в действии необходимо учитывать известные условия и вводятся несколько путей возврата, то используется интерфейс IactionResult, чтобы возвращать различные типы.

Асинхронные запросы позволяют избежать блокирования потока при выполнении запроса в базе данных. Они помогают предотвратить замораживание пользовательского интерфейса многофункционального клиентского приложения. Асинхронные операции могут также увеличить пропускную способность в веб-приложении, где можно высвободить поток для обработки других запросов во время завершения операции базы данных.

Асинхронный метод Index() предназначен для получения списка всех соревнований. Для данного списка происходит обновление статуса соревнований и получение судей для каждого из них. Исходный код данного метода приведен в листинге 1.

Листинг 1 – исходныйкод метода Index() контроллера TournamentController.

```
// GET: Tournament
        public async Task<IActionResult> Index() // Заглавная страница (список
соревнований)
        {
            // При открытии главной страницы соревнований (список всех соревнований)
получить все данные о существующих соревнованиях
            var list = await _context.Tournaments.ToListAsync();
            foreach (var item in list)
            {
                // Обновляем статус соревнования
                UpdateStatus(item);
                item.Input = new TournamentModel.InputModel();
                // Получаем список судей для каждого соревнования для представления на
странице
                item.Input.JudgeUsers = await DbHelper.GetJudgesForTournament(item);
            }
// возвращаем представление со списком соревнований
            return View(list);
               }
```

Асинхронный метод Details(int? id) позволяет получить подробную информацию о соревнованиях. Для начала происходит проверка идентификатора соревнований. После этого обновляется статус соревнований и происходит получение списка судей. Исходный код метода Details(int? id) приведен в листинге 2.

Листинг 2 – Исходный код метода Details(int? id) контроллера TournamentController.

```
// GET: Tournament/Details
        // Подробнее о соревновании
        public async Task<IActionResult> Details(int? id)
        {
            if (id == null) //проверка идентификатора
            {
                return NotFound(); // возвращается ошибка
            }
            var tournamentModel = await _context.Tournaments
                .FirstOrDefaultAsync(m => m.Id == id);
            if (tournamentModel == null) // проверка идентификатора модели
            {
                return NotFound();// возвращается ошибка
            }

            UpdateStatus(tournamentModel);// обновляется статус соревнований

            tournamentModel.Input = new TournamentModel.InputModel();
            tournamentModel.Input.JudgeUsers = await
DbHelper.GetJudgesForTournament(tournamentModel); // Получаем список судей для каждого
соревнования для представления на странице "Подроднее"
            return View(tournamentModel);//возвращаем представление с информацией о
соревновании
              }
```

Асинхронный метод Create() предназначен для создания нового соревнования. Создается новая модель, в которой указываются даты начала и окончания соревнований, а так же идет получение списка судей для добавления на соревнования и формируется список команд для участия в соревнованиях. Данный метод доступен для администратора и членов ФК. Исходный код метода Create() приведен в листинге 3.

Листинг 3 – Исходный код метода Create() контроллера TournamentController.

```
// GET: Tournament/Create
        [Authorize(Roles = "Admin, Manager")]// Доступно только для ролей "Администратор"
и "Член ФК"
        public async Task<IActionResult> Create()// Создание нового соревнования
        {
            var model = new TournamentModel
            {
                DateStart = DateTime.Now,//дата начала соревнований
                DateEnd = DateTime.Now,//дата окончания соревнований
                Input = new TournamentModel.InputModel
                {
                    Judges = await DbHelper.GetJudges(),//получение списка судей
                    Teams = new List<TeamModel>(),//список команд
                    SelectedIds = new List<SelectListItem>()
                }
            };
            return View(model);//возвращается представление с созданным соревнованием
               }
```

Асинхронный метод Delete(int? id) предназначен для перехода на страницу с удалением соревнований (исходный код приведен в листинге 4). Для начала проверяется идентификатор соревнований. Если все удачно, то происходит переход на страницу с удалением соревнований, в противном случае будет выведена ошибка. Данный метод доступен для администратора и Членов ФК.

Листинг 4 – Исходный код метода Delete(int? id) контроллера TournamentController.

```
// GET: Tournament/Delete
        [Authorize(Roles = "Admin, Manager")]")]// Доступно только для ролей
"Администратор" и "Член ФК"
        public async Task<IActionResult> Delete(int? id) // Переход на страницу с
удалением соревнования
        {
            if (id == null)// проверка идентификатора соревнования
            {
                return NotFound(); //возвращается ошибка
            }

            var tournamentModel = await _context.Tournaments
                .FirstOrDefaultAsync(m => m.Id == id); //получение модели
            if (tournamentModel == null)//проверка на то, что модель существует
            {
                return NotFound();//возвращается ошибка
            }

            return View(tournamentModel);//переход на страницу с удалением
               }
```

Асинхронный метод Create(TournamentModel) который позволяет создавать новые соревнования (исходный код приведен в листинге 5). Сначала

идет проверка на ошибки при заполнении формы для создания соревнований. Объект ModelState сохраняет все значения, которые пользователь ввел для свойств модели, а также все ошибки, связанные с каждым свойством и с моделью в целом. Если в объекте ModelState имеются какие-нибудь ошибки, то свойство ModelState.IsValid возвратит false. Если ошибок нет, то происходит проверка, чтобы начало соревнований было раньше, чем их окончание, после этого происходит обновление статуса соревнований, добавляется список судей и сохраняются все изменения и происходит переход на страницу со списком соревнований. Данный метод доступен только для администратора.

Листинг 5 – Исходный код метода Create(TournamentModel) контроллера TournamentController.

```
// POST: Tournament/Create
        [HttpPost]
        [ValidateAntiForgeryToken]// фильтр предназначен для противодействия подделке межсайтовых запросов, производя верификацию токенов при обращении к методу действия.
        [Authorize(Roles = "Admin")]// Доступно только для роли "Администратор"
        public async Task<IActionResult> Create([Bind("Id,Name,City,Stadium,DateStart,DateEnd,ShortInfo,LongInfo,SelectedIds")] TournamentModel tournamentModel) // Атрибут [Bind] является одним из способов защиты от чрезмерной передачи данных.
        {
            if (ModelState.IsValid)//проверка на ошибки
            {
                _context.Add(tournamentModel); // добавляем в контекст новые соревнования с данными, пришедшими из вьюшки
                if (tournamentModel.DateStart >= tournamentModel.DateEnd)
// Возвращаемся, если дата начала соревнований была выше, чем дата конца
                    return View(tournamentModel);

                UpdateStatus(tournamentModel);//обновляется статус соревнований

                var toAdd = new List<JudgeModel>();

                var allUsers = await _context.Users.ToListAsync();
                var users = allUsers.Where(user => tournamentModel.SelectedIds.Contains(user.Id));
                // Добавляем всех выбранных судей
                foreach (var user in users)
                {
                    var assignment = new JudgeModel();
                    assignment.Judge = user;
                    assignment.Tournament = tournamentModel;
                    toAdd.Add(assignment);
                }
                await _context.SaveChangesAsync();// Сохраняем все изменения
                foreach (var judgeModel in toAdd)
                {
                    _context.Add(judgeModel);
                }
```

Продолжение листинга 5

```
            await _context.SaveChangesAsync();//сохраняем данные
            return RedirectToAction(nameof(Index));//переадресация
        }
        return View(tournamentModel);//возвращаем представление
    }
```

Асинхронный метод Edit(int? id) предназначен для страницы с редактированием соревнований (исходный код приведен в листинге 6). Для начала идет проверка на присутствие идентификатора соревнований, если соревнования найдены, то идет заполнение модели и возвращается страница с предзаполненной формой Метод доступен для администратора и Члена ФК.

Листинг 6 – Исходный код метода Edit(int? id) контроллера TournamentController.

```
// GET: Tournament/Edit
        [Authorize(Roles = "Admin, Manager")] // Доступно только для ролей 
"Администратор" и "Член ФК"
        public async Task<IActionResult> Edit(int? id) // Данные для страницы с 
редактированием соревнования
        {
            if (id == null) //проверка идентификатора соревнований
            {
                return NotFound(); //ошибка
            }

            var tournamentModel = await _context.Tournaments.FindAsync(id);
            if (tournamentModel == null)//проверка модели
            {
                return NotFound();//ошибка
            }

            UpdateStatus(tournamentModel);//обновление статуса соревнований
            return View(tournamentModel);//возвращаем страницу для редактирования 
соревновани
               }
```

Асинхронный метод DeleteConfirmed(int id) позволяет удалить соревнование (исходный код приведен в листинге 7). Происходит проверка на присутствие идентификатора соревнований, если соревнования найдены, то они удаляются из базы данных.

Листинг 7 – Исходный код метода DeleteConfirmed(int id) контроллера TournamentController

```
// POST: Tournament/Delete
        [HttpPost, ActionName("Delete")]
        [ValidateAntiForgeryToken] // фильтр предназначен для противодействия подделке 
межсайтовых запросов, производя верификацию токенов при обращении к методу действия.
```

Продолжение листинга 7

```
        [Authorize(Roles = "Admin, Manager")]
        public async Task<IActionResult> DeleteConfirmed(int id) // Непосредственное
удаление соревнования
        {
            var tournamentModel = await _context.Tournaments.FindAsync(id);
            _context.Tournaments.Remove(tournamentModel);
            await _context.SaveChangesAsync();
            return RedirectToAction(nameof(Index));
        }
```

Асинхронный метод Edit(int id, TournamentModel) позволяет редактировать соревнования. Для начала проверяется существование соревнований с данным идентификатором. Если модель существует, то происходит проверка на корректность введенных данных. Если данные корректны, то происходит обновление модели и отслеживание ошибки при обновлении. Если все прошло успешно, то изменения сохраняются. Исходный код данного метода приведен в листинге 8.

Листинг 8 – Исходный код метода Edit(int id, TournamentModel) контроллера TournamentController.

```
// POST: Tournament/Edit
        [HttpPost]
        [ValidateAntiForgeryToken] // фильтр предназначен для противодействия подделке
межсайтовых запросов, производя верификацию токенов при обращении к методу действия
        [Authorize(Roles = "Admin, Manager")]// Доступно только для ролей "Администратор"
и "Член ФК"
        public async Task<IActionResult> Edit(int id,
[Bind("Id,Name,City,Stadium,DateStart,DateEnd,ShortInfo,LongInfo")] TournamentModel
tournamentModel) // Непосредственное редактирование соревнования (POST-запрос)
        {
            if (id != tournamentModel.Id)//проверка идентификатора соревнований и модели
            {
                return NotFound();//ошибка
            }
            if (ModelState.IsValid)//проверка на ошибки
            {
                Try //фрагмент кода, в котором может быть исключение
                {
                    UpdateStatus(tournamentModel);
                    _context.Update(tournamentModel);
                    await _context.SaveChangesAsync();
                }
                catch (DbUpdateConcurrencyException) // вид исключения
                {
                    if (!TournamentModelExists(tournamentModel.Id))
                    {
                        return NotFound();//ошибка
                    }
                    else
                    {
                        throw; //позволяет заново вызвать исключение
                    }
                }
                return RedirectToAction(nameof(Index));//переадресация
```

Продолжение листинга 8

```
        }
        return View(tournamentModel);//возвращает страницу с соревнованиями
           }
```

Асинхронный метод SetPlacement(int id) позволяет установить место для команды. В первую очередь происходит проверка идентификатора команды, поиск команды по идентификатору, установка места. Исходный код приведен в листинге 9.

Листинг 9 – Исходный код метода SetPlacement(int id) контроллера TournamentController

```
        public async Task<IActionResult> SetPlacement(int id) // Устанавливаем место для
команды
        {
            if (id == 0) //проверка идентификатора
                return NotFound(); //ошибка

            var team = _context.Teams.Find(id);

            if (team == null) //проверка идентификатора команды
                return NotFound();//ошибка

            var teamPlacement = new TeamPlacement();//место команды

            int placement = int.Parse((string)ViewData["Placement"]);//перевод из типа
string в int
            if (placement == 0) //проверка места
                return NotFound(); //ошибка
            teamPlacement.Placement = placement;
            teamPlacement.Team = team;
            _context.Add(teamPlacement); //добавление места
            await _context.SaveChangesAsync();//сохраняем изменения
            return View("Standings");
               }
```

Асинхронный метод Summary(int? id) предназначен для получения данных в итоговом протоколе (исходный код приведен в листинге 10). Происходит поиск соревнований по идентификатору, обновляется статус соревнований, происходит получение команд для данного турнира. Все полученные данные передаются в представление и оно возвращается пользователю.

Листинг 10 – Исходный код метода Summary(int? id) контроллера TournamentController

```
// GET: Tournament/Details/5
        // Данные для итогового протокола
        public async Task<IActionResult> Summary(int? id)
        {
            if (id == null) //проверка идентификатора соревнований
```

Продолжение листинга 10

```
            {
                return NotFound(); //ошибка
            }
            var tournamentModel = await _context.Tournaments
                .FirstOrDefaultAsync(m => m.Id == id);
            if (tournamentModel == null)//проверка идентификатора модели
            {
                return NotFound();//ошибка
            }
            UpdateStatus(tournamentModel);//обновление статуса соревнований

            List<SummaryData> summaryDataList = new List<SummaryData>();
            var teams = await DbHelper.GetTeamsForTournament(tournamentModel);//получение
команд для соревнований
            foreach (var teamModel in teams)
            {
                var summaryData = new SummaryData(_context, teamModel.Id);
                summaryDataList.Add(summaryData);//добавление данных
            }
            ViewData["SummaryData"] = summaryDataList;
            return View(tournamentModel);
               }
```

Асинхронный метод Schedule(int? id) предназначен для получения расписания игр на соревнованиях. Происходит проверка идентификатора соревнований, после происходит получение данных о турах и играх в них, на основании которых формируется само расписание и возвращается пользователю. Исходный код приведен в листинге 11.

Листинг 11 – Исходный код метода Schedule(int? id) контроллера TournamentController

```
// GET: Tournament/Details/5
        // Данные для страницы с расписанием игр
        public async Task<IActionResult> Schedule(int? id)
        {
            if (id == null) //проверка идентификатора соревнований
            {
                return NotFound();//ошибка
            }
            var tournamentModel = await _context.Tournaments
                .FirstOrDefaultAsync(m => m.Id == id);
            if (tournamentModel == null)// проверка идентификатора модели
            {
                return NotFound();//ошибка
            }
            Dictionary<TourModel, List<GameModel>> dict = new Dictionary<TourModel,
List<GameModel>>(); // создается коллекция
            var tours = _context.Tours.Where(model => model.Tournament ==
tournamentModel).ToList();
            foreach (var tourModel in tours)//получение туров
            {
                var games = _context.Games.Where(model => model.Tour ==
tourModel).ToList();

                foreach (var gameModel in games)//получение игр
                {
```

Продолжение листинга 11

```
                if (gameModel.FirstTeamId == 0 || gameModel.SecondTeamId == 0)
                    throw new ArgumentNullException("Error, Team IDs are null!");

                gameModel.FirstTeam = _context.Teams.Find(gameModel.FirstTeamId);
                gameModel.SecondTeam = _context.Teams.Find(gameModel.SecondTeamId);

                if (gameModel.FirstTeam == null || gameModel.SecondTeam == null)
                {
                    gameModel.FirstTeam = _context.Teams.SingleOrDefault(model =>
model.Id == gameModel.FirstTeamId);
                    gameModel.SecondTeam = _context.Teams.SingleOrDefault(model =>
model.Id == gameModel.SecondTeamId);
                }
            }
            dict[tourModel] = games;
        }
        ViewData["Data"] = dict;
        return View(tournamentModel);
           }
```

Асинхронный метод TeamList(int? id) предназначен для получения списка команд для определенных соревнований. В первую очередь проверяется идентификатор соревнований, поле этого обновляется статус соревнований, и происходят запросы к базе данных для получения судей и команд для соревнований. После этого данные передаются в представление. Так же происходит получение списка спортсменов и тренеров для каждой команды. Заполненная модель возвращается пользователю. Исходный код приведен в листинге 12.

Листинг 12 – Исходный код метода TeamList(int? id) контроллера TournamentController

```
// Список команд
        // GET: Tournament/TeamList/5
        public async Task<IActionResult> TeamList(int? id)
        {
            if (id == null)//проверка идентификатора
            {
                return NotFound();//ошибка
            }
            var tournamentModel = await _context.Tournaments
                .FirstOrDefaultAsync(m => m.Id == id);
            if (tournamentModel == null)//проверка идентификатора модели
            {
                return NotFound();//ошибка
            }
            UpdateStatus(tournamentModel);//обновляется статус соревнований
            tournamentModel.Input = new TournamentModel.InputModel();
            tournamentModel.SelectedIds = new string[1];
            tournamentModel.Input.JudgeUsers = await
DbHelper.GetJudgesForTournament(tournamentModel); // Передаём представлению  всех судей
            var teams = await DbHelper.GetNonGroupedTeamsForTournament(tournamentModel);
// получаем список всех команд
            tournamentModel.Input.Teams = teams;
```

Продолжение листинга 12

```
            tournamentModel.Input.TeamSportsmans = new Dictionary<TeamModel,
KeyValuePair<List<CurlingUser>, CurlingUser[]>>();
            ViewData["Data"] = await
DbHelper.GetTournamentSportsmanDataForTournament(tournamentModel); // Передача всех
полученных данных представлению
            foreach (var teamModel in teams) //получение спортсменов и тренеров
            {
                var sportsmen = await DbHelper.GetUsersForTeam(teamModel);
                var coaches = new CurlingUser[sportsmen.Count];
                for (int i = 0; i < sportsmen.Count; i++)
                {
                    var coach = await DbHelper.GetCoachForSportsman(sportsmen[i]);
                    coaches[i] = coach;
                }
                var kv = new KeyValuePair<List<CurlingUser>, CurlingUser[]>(sportsmen,
coaches);
                tournamentModel.Input.TeamSportsmans[teamModel] = kv;
            }
            return View(tournamentModel);
               }
```

Асинхронный метод Standings(int? id) предназначен для получения данных в турнирной таблице (исходный код приведен в листинге 13). В первую очередь происходит проверка идентификатора соревнований. Далее обновляется статус соревнований и происходит запрос к базе данных для получения данных. Готовое представление возвращается пользователю.

Листинг 13 – Исходный код метода Standings(int? id) контроллера TournamentController

```
  // GET: Tournament/Details
        // Данные для страницы с турнирной таблицей
        public async Task<IActionResult> Standings(int? id)
        {
            if (id == null)//проверка идентификатора
            {
                return NotFound();//ошибка
            }
            var tournamentModel = await _context.Tournaments
                .FirstOrDefaultAsync(m => m.Id == id);
            if (tournamentModel == null)//идентификатор модели
            {
                return NotFound();//ошибка
            }
            UpdateStatus(tournamentModel);//обновляется статус
            ViewData["Data"] = await
DbHelper.GetTournamentSportsmanDataForTournament(tournamentModel);//получение данных
            return View(tournamentModel);
               }
```

Асинхронный метод TournamentProgress(int? id) предназначен для получения данных на странице «ход турнира». Происходит проверка идентификатора соревнований, обновляется статус соревнований. После этого происходит запрос к базе данных для получения списка туров и списка игр и

эндов для них. Так же здесь происходит заполнение таблиц со счетом. После этого, как все данные получены, пользователю возвращается представление с заполненными на данный момент данными. Исходный код приведен в листинге 14.

Листинг 14 – Исходный код метода TournamentProgress(int? id) контроллера TournamentController

```
// GET: Tournament/Details/5
        // Данные для страницы с ходом турнира
        public async Task<IActionResult> TournamentProgress(int? id)
        {
            if (id == null) //провверка идентификатора
            {
                return NotFound();//ошибка
            }
           var tournamentModel = await _context.Tournaments
                .FirstOrDefaultAsync(m => m.Id == id);
            if (tournamentModel == null)/идентификатор модели
            {
                return NotFound();/ошибка
            }
            UpdateStatus(tournamentModel);//обновление статуса соревнований
            Dictionary<TourModel, List<KeyValuePair<GameModel, EndingModel[]>>>
tourGamesDictionary = new Dictionary<TourModel, List<KeyValuePair<GameModel,
EndingModel[]>>>();// Список всех игр и эндов для этих игр для всех туров

            var tours = await DbHelper.GetToursForTournament(tournamentModel); //
Получаем список всех туров в соревновании
            foreach (var tourModel in tours) //получение туров
                var games = await DbHelper.GetGamesForTour(tourModel); // Для каждого
тура ищем все игры
                var list = new List<KeyValuePair<GameModel, EndingModel[]>>();
                foreach (var gameModel in games)
                {
                    if (gameModel.FirstTeamId == 0 || gameModel.SecondTeamId == 0)
                        throw new ArgumentNullException("Error, Team IDs are null!");
                    // Для каждой игры заполняем данные: играющие команды и список их
эндов
                    gameModel.FirstTeam = _context.Teams.Find(gameModel.FirstTeamId);
                    gameModel.SecondTeam = _context.Teams.Find(gameModel.SecondTeamId);
                    if (gameModel.FirstTeam == null || gameModel.SecondTeam == null)
                    {
                        gameModel.FirstTeam = _context.Teams.SingleOrDefault(model =>
model.Id == gameModel.FirstTeamId);
                        gameModel.SecondTeam = _context.Teams.SingleOrDefault(model =>
model.Id == gameModel.SecondTeamId);
                    }
                    List<EndingModel> endingsForGame = await
DbHelper.GetEndingsForGame(gameModel);получение эндов для игр
                    var kv = new KeyValuePair<GameModel, EndingModel[]>(gameModel,
endingsForGame.ToArray());
                    list.Add(kv);
                }

                tourGamesDictionary[tourModel] = list;
            }
            tournamentModel.Input = new TournamentModel.InputModel();
            tournamentModel.Input.Ending = new EndingModel();
```

Продолжение листинга 14

```
        ViewData["TourGames"] = tourGamesDictionary;
        return View(tournamentModel);
           }
```

Метод UpdateStatus предназначен для обновления статуса соревнований. Статус может быть следующим: «ожидает начала», «началось», «закончилось».

Листинг 15 – Исходный код метода UpdateStatus контроллера TournamentController

```
private void UpdateStatus(TournamentModel tournamentModel)
        {
            if (tournamentModel.DateStart > DateTime.Now)
                tournamentModel.Status = TournamentStatus.Coming;
            else if (tournamentModel.DateEnd < DateTime.Now)
                tournamentModel.Status = TournamentStatus.Ended;
            else
                tournamentModel.Status = TournamentStatus.Started;
              }
```

В листинге А.1 приложения А содержится исходный код контроллера ENDING_CONTROLLER.

В листинге В.1 приложения Б содержится исходный код контроллера GAME_CONTROLLER.

В листинге В.1 приложения В содержится исходный код контроллера GROUP_CONTROLLER.

В листинге Г.1 приложения Г содержится исходный код контроллера HOME_CONTROLLER.

В листинге Д.1 приложения Д содержится исходный код контроллера SCORE_CONTROLLER.

В листинге Е.1 приложения Е содержится исходный код контроллера TEAM_CONTROLLER.

В листинге Ж.1 приложения Ж содержится исходный код контроллера TEAM_PLACEMENT_CONTROLLER.

В листинге К.1 приложения К содержится исходный код контроллера TEAM_PLACEMENT_GLOBAL_CONTROLLER.

В листинге Л.1 приложения Л содержится исходный код контроллера TOUR_CONTROLLER.

В листинге М.1 приложения М содержится исходный код контроллера TOURNAMENT_CONTROLLER.

В листинге Н.1 приложения Н содержится исходный код модели CITY_MODEL.

В листинге П.1 приложения П содержится исходный код модели COACH_MODEL.

В листинге Р.1 приложения Р содержится исходный код модели ENDING_MODEL.

В листинге С.1 приложения С содержится исходный код модели GAME_MODEL.

В листинге Т.1 приложения Т содержится исходный код модели GROUP_ASSIGNMENT_MODEL.

В листинге У.1 приложения У содержится исходный код модели GROUP_MODEL.

В листинге Ф.1 приложения Ф содержится исходный код модели JUDGE_MODEL.

В листинге Х.1 приложения Х содержится исходный код модели SCORE_MODEL.

В листинге Ц.1 приложения Ц содержится исходный код модели TEAM_ASSIGNMENT_MODEL.

В листинге Ш.1 приложения Ш содержится исходный код модели TEAM_MODEL.

В листинге Щ.1 приложения Щ содержится исходный код модели TEAM_PLACEMENT.

В листинге Э.1 приложения Э содержится исходный код модели TEAM_PLACEMENT_GLOBAL.

В листинге Ю.1 приложения Ю содержится исходный код модели TEAM_TOURNAMENT_ASSIGNMENT_MODEL.

В листинге Я.1 приложения Я содержится исходный код модели TOUR_MODEL.

В листинге А.1 приложения А содержится исходный код модели TOURNAMENT_MODEL.

В листинге В.1 приложения В содержится исходный код модели ERROR_VIEW_MODEL.

3.3 ОПИСАНИЕ ДАННЫХ

Схема базы данных приведена на рисунке 3.9.

В таблице 3.4 представлен список таблиц базы данных, а также их назначение.

При описании атрибутов использованы следующие обозначения:

1. * – первичный ключ;
2. ^ – внешний ключ и ссылающийся на указанный атрибут.

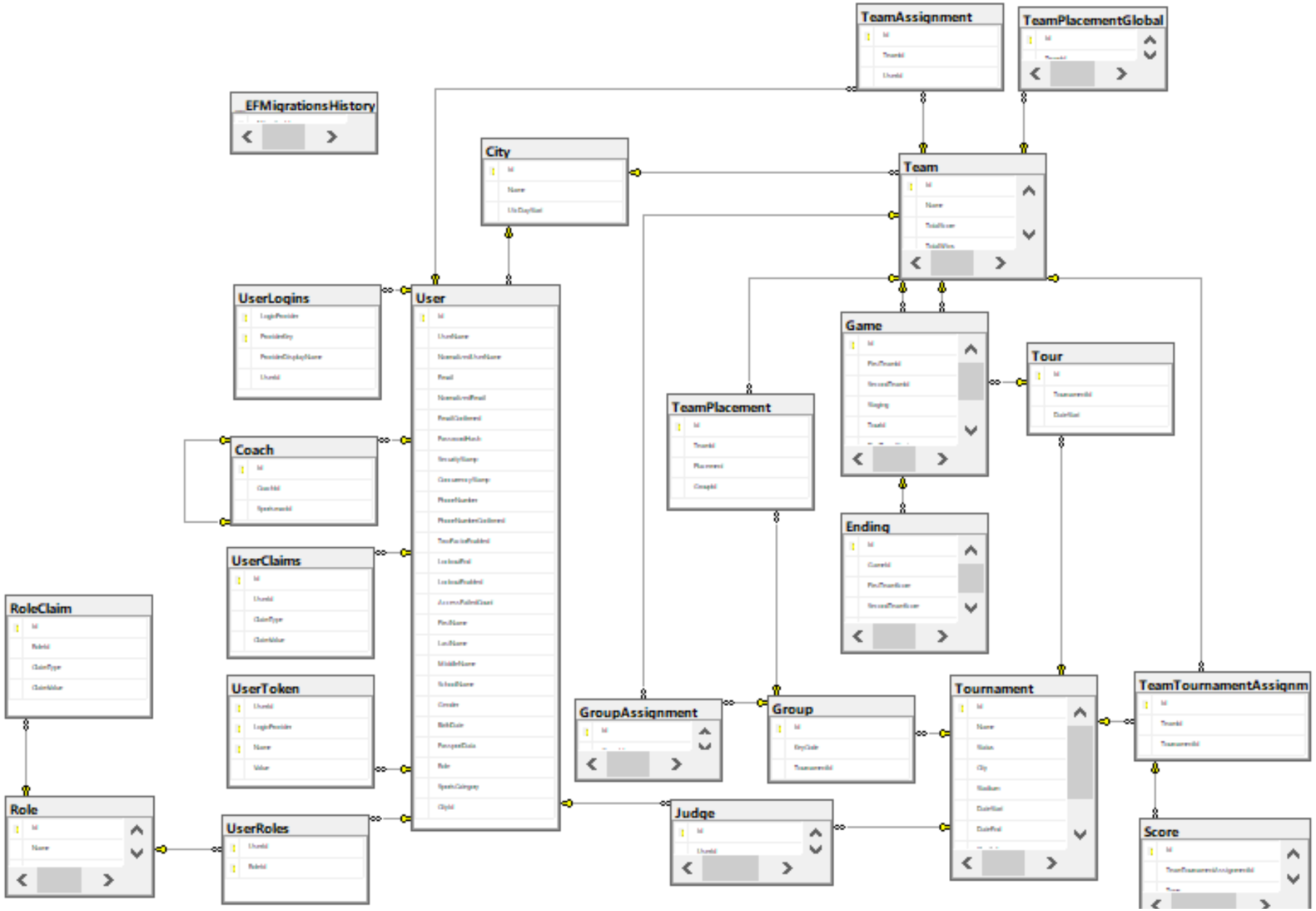


Рисунок 3.9 – Схема БД

В таблице 3.4 содержится список таблиц базы данных.

Таблица 3.4 – Список таблиц базы данных

Название таблицы	**Назначение**
«City»	Список городов
«Coache»	Связи между тренерами и спортсменами
«Ending»	Сведения об эндах
«Game»	Данные об играх
«Group»	Данные о группах
«GroupAssignment»	Связи между группами и командами
« Judge»	Связи между судьями и соревнованиями
«Role»	Список ролей пользователей
«RoleClaim»	Разбивка или разграничение ролей по различным категориям, например, по возрасту или по стране.
«Score»	Данные о постановочных бросках
«Team»	Данные о команде
«TeamAssignment»	Связи между спортсменами и командами
«TeamPlacement»	Данные о местах в группах
«TeamPlacementGlobal»	Данные о местах на соревнованиях
«TeamTournamentAssignments»	Связи между командами и соревнованиями
«Tour»	Данные о турах
«Tournament»	Данные о соревнованиях
«User»	Данные о пользователях
«UserClaims»	Разграничение пользователей по некоторому признаку по аналогии с ролями
«UserLogins»	Данные о подключенных методах авторизации через сервисы, например, через Google+
«UserRoles»	Данные о доступных в системе ролях
«UserToken»	Токены входа пользователя через различные социальные сети.
«_EFMigrationsHistory»	Автоматически генерируемая таблица для отслеживания состояния модели данных, в которой хранятся сериализованные в двоичный формат модели данных

В таблице 3.5 описаны поля таблицы «City».

Таблица 3.5 – Таблица «City»

№	Название	Ключ	Тип	Семантика
1	id	*	int	Уникальный идентификатор
2	Name		nvarchar(max)	Название города
3	UtcDayStart		datetime2	Количество дополнительных часов в часовом поясе, в котором находится город относительно UTC+000

В таблице 3.6 описаны поля таблицы «Coache».

Таблица 3.6 – Таблица «Coache»

№	Название	Ключ	Тип	Семантика
1	id	*	int	Уникальный идентификатор
2	CoachId	^User (id)	nvarchar(max)	Идентификатор тренера
3	SportsmanId	^User (id)	nvarchar(max)	Идентификатор спортсмена

В таблице 3.7 описаны поля таблицы «Ending».

Таблица 3.7 – Таблица «Ending»

№	Название	Ключ	Тип	Семантика
1	id	*	int	Уникальный идентификатор
2	GameId	^Game (id)	int	Идентификатор игры
3	FirstTeamScore		int	Постановочный бросок первой команды
4	SecondTeamScore		int	Постановочный бросок второй команды
5	Position		int	Номер энда (позиция относительно начала игры)
6	IsFinal		bit	Определение последнего энда

В таблице 3.8 описаны поля таблицы «Role».

Таблица 3.8 – Таблица «Role»

№	Название	Ключ	Тип	Семантика
1	id	*	nvarchar(450)	Уникальный идентификатор
2	Name		nvarchar(256)	Название роли
3	NormalizedName		nvarchar(256)	Нормализированное имя (возведенное в верхний регистр)
4	ConcurrencyStamp		nvarchar(max)	Специальный идентификатор, не позволяющий параллельно изменить данные таблицы

В таблице 3.9 описаны поля таблицы «Group».

Таблица 3.9 – Таблица «Group»

№	Название	Ключ	Тип	Семантика
1	id	*	int	Уникальный идентификатор
2	KeyCode		nvarchar(max)	Название группы
3	TournamentId	^Tournament (id)	int	Идентификатор соревнования

В таблице 3.10 описаны поля таблицы «GroupAssignment».

Таблица 3.10 – Таблица «GroupAssignment»

№	Название	Ключ	Тип	Семантика
1	id	*	int	Уникальный идентификатор
2	GroupId	^Group (id)	int	Идентификатор группы
3	TeamId	^Team (id)	int	Идентификатор команды

В таблице 3.11 описаны поля таблицы «Game».

Таблица 3.11 – Таблица «Game»

№	Название	Ключ	Тип	Семантика
1	id	*	int	Уникальный идентификатор
2	FirstTeamId	^Team (id)	int	Идентификатор первой команды
3	SecondTeamId	^Team (id)	int	Идентификатор второй команды
4	Staging		float	Общая постановка
5	TourId	^Tour (id)	int	Идентификатор тура
6	FirstTeamStaging		float	Постановка первой команды
7	SecondTeamStaging		float	Постановка второй команды

В таблице 3.12 описаны поля таблицы « Judge».

Таблица 3.12 – Таблица «Judge»

№	Название	Ключ	Тип	Семантика
1	id	*	int	Уникальный идентификатор
2	UserId	^User (id)	nvarchar(450)	Идентификатор пользователя
3	TournamentId	^Tournament (id)	int	Идентификатор соревнования

В таблице 3.13 описаны поля таблицы «RoleClaim».

Таблица 3.13 – Таблица «RoleClaim»

№	Название	Ключ	Тип	Семантика
1	id	*	int	Уникальный идентификатор
2	RoleId	^Role (id)	nvarchar(450)	Идентификатор роли
3	ClaimType		nvarchar(max)	Тип разграничения (например, по стране)
4	ClaimValue		nvarchar(max)	Значение запроса

В таблице 3.14 описаны поля таблицы «Score».

Таблица 3.14 – Таблица «Score»

№	Название	Ключ	Тип	Семантика
1	id	*	int	Уникальный идентификатор
2	TeamTournament AssignmentId	^TeamTournament Assignment (id)	int	Идентификатор привязки команды и соревнования
3	Type		int	Очки за игры (3-победа, 2-ничья, 1-поражение, 0-неявка)
4	Staging		float	Постановка

В таблице 3.15 описаны поля таблицы «Team».

Таблица 3.15 – Таблица «Team»

№	Название	Ключ	Тип	Семантика
1	id	*	int	Уникальный идентификатор
2	Name		nvarchar(max)	Название команды
3	TotalScore		int	Итоговая постановка
4	TotalWins		int	Количество побед в турнире
5	CityId	^City (id)	int	Идентификатор города

В таблице 3.16 описаны поля таблицы «TeamAssignment».

Таблица 3.16 – Таблица «TeamAssignment»

№	Название	Ключ	Тип	Семантика
1	id	*	int	Уникальный идентификатор
2	TeamId	^Team (id)	int	Идентификатор команды
3	UserId	^User (id)	nvarchar(450)	Идентификатор пользователя

В таблице 3.17 описаны поля таблицы «TeamPlacement».

Таблица 3.17– Таблица «TeamPlacement»

№	Название	Ключ	Тип	Семантика
1	id	*	int	Уникальный идентификатор
2	TeamId	^Team (id)	int	Идентификатор команды
3	Placement		int	Место в группе
4	GroupId	^Group (id)	int	Идентификатор группы

В таблице 3.18 описаны поля таблицы «TeamPlacementGlobal».

Таблица 3.18 – Таблица «TeamPlacementGlobal»

№	Название	Ключ	Тип	Семантика
1	id	*	int	Уникальный идентификатор
2	TeamId	^Team (id)	int	Идентификатор команды
3	Placement		int	Итоговое место команды

В таблице 3.19 описаны поля таблицы «TeamTournamentAssignments».

Таблица 3.19 – Таблица «TeamTournamentAssignments»

№	Название	Ключ	Тип	Семантика
1	Id	*	int	Уникальный идентификатор
2	TeamId	^Team (id)	int	Идентификатор команды
3	TournamentId	^Tournament (id)	int	Идентификатор соревнования

В таблице 3.20 описаны поля таблицы «Tour».

Таблица 3.20 – Таблица «Tour»

№	Название	Ключ	Тип	Семантика
1	id	*	int	Уникальный идентификатор
2	DateStart		datetime2(7)	Начало тура
3	TournamentId	^Tournament (id)	int	Идентификатор соревнования

В таблице 3.21 описаны поля таблицы «Tournament».

Таблица 3.21 – Таблица «Tournament»

№	Название	Ключ	Тип	Семантика
1	id	*	int	Уникальный идентификатор
2	Name		nvarchar(max)	Название соревнования
3	Status		nvarchar(max)	Статус соревнований
4	City		nvarchar(max)	Город
5	Stadium		nvarchar(max)	Место проведения
6	DateStart		datetime2(7)	Начало соревнований
7	DateEnd		datetime2(7)	Окончание соревнований
8	ShortInfo		nvarchar(max)	Краткая информация
9	LongInfo		nvarchar(max)	Подробная информация

В таблице 3.22 описаны поля таблиц «User».

Таблица 3.22 – Таблица «User»

№	Название	Ключ	Тип	Семантика
1	id	*	nvarchar(450)	Уникальный идентификатор
2	UserName		nvarchar(256)	Логин пользователя
3	Normalized UserName		nvarchar(256)	Нормализованный логин пользователя
4	Email		nvarchar(256)	Электронная почта
5	NormalizedEmail		nvarchar(256)	Нормализованная электронная почта
6	EmailConfirmed		bit	Флаг, обозначающий подтверждение почты
7	PasswordHash		nvarchar(max)	Хэш пароля
8	SecurityStamp		nvarchar(max)	Специальный идентификатор для входов пользователей
9	ConcurrencyStamp		nvarchar(max)	Специальный идентификатор, не позволяющий параллельно изменить данные таблицы
10	PhoneNumber		nvarchar(max)	Номер телефона
11	PhoneNumberConfirmed		bit	Флаг обозначающий подтверждение номера телефона

Продолжение таблицы 3.22

№	Название	Ключ	Тип	Семантика
12	TwoFactorEnabled		bit	Флаг, обозначающий, что включена двухфакторная авторизация (например, через Google+)
13	LockoutEnd		datetimeoffset(7)	Дата, до какого времени пользователь будет заблокирован
14	LockoutEnabled		bit	Флаг, обозначающий заблокирован ли пользователь
15	AccessFailedCount		int	Количество неудачных попыток входа
16	FirstName		nvarchar(max)	Имя
17	LastName		nvarchar(max)	Фамилия
18	MiddleName		nvarchar(max)	Отчество
19	SchoolName		nvarchar(max)	Школа/клуб
20	Gender		int	Пол
21	BirthDate		datetime2(7)	Дата рождения
22	PassportData		nvarchar(max)	Паспортные данные
23	Role		nvarchar(max)	Роль пользователя
24	SportsCategory		nvarchar(max)	Спортивный разряд
25	CityId	^City (id)	int	Идентификатор города

В таблице 3.23 описаны поля таблицы «UserClaims».

Таблица 3.23 – Таблица «UserClaims»

№	Название	Ключ	Тип	Семантика
1	id	*	int	Уникальный идентификатор
2	UserId	^User (id)	nvarchar(450)	Идентификатор пользователя
3	ClaimType		nvarchar(max)	Тип разграничения
4	ClaimValue		nvarchar(max)	Значение запроса

В таблице 3.24 описаны поля таблицы «UserLogins».

Таблица 3.24 – Таблица «UserLogins»

№	Название	Ключ	Тип	Семантика
1	LoginProvider	*	nvarchar(450)	Название провайдера/сервиса
2	ProviderKey	*	nvarchar(450)	Ключ доступа к провайдеру
3	ProviderDisplayName		nvarchar(max)	Имя провайдера в системе
4	UserId	^User (id)	nvarchar(450)	Идентификатор пользователя

В таблице 3.25 описаны поля таблицы «UserRoles».

Таблица 3.25 – Таблица «UserRoles»

№	Название	Ключ	Тип	Семантика
1	UserId	* и ^User (id)	nvarchar(450)	Идентификатор пользователя
2	RoleId	* и ^Role (id)	nvarchar(450)	Идентификатор роли

В таблице 3.26 описаны поля таблицы «UserToken».

Таблица 3.26 – Таблица «UserToken»

№	Название	Ключ	Тип	Семантика
1	UserId	* и ^User (id)	nvarchar(450)	Идентификатор пользователя
2	LoginProvider	*	nvarchar(450)	Название провайдера/сервиса
3	Name	*	nvarchar(450)	Название токена
4	Value		nvarchar(max)	Значение токена

В таблице 3.27 описаны поля таблицы «_EFMigrationsHistory».

Таблица 3.2 – Таблица «_EFMigrationsHistory»

№	Название	Ключ	Тип	Семантика
1	MigrationId	*	nvarchar(150)	Идентификатор миграций
2	ProductVersion		nvarchar(32)	Версия Entity Framework Core

В листинге 16 содержится исходный код запроса к базе данных Данный запрос позволяет получить данные для каждой группы, то есть сначала идет получение всех групп на соревнованиях, после этого идет получение всех команд в группе, получение спортсменов для каждой команды, получение тренеров для каждого спортсмена, а так же подсчитывается сумма постановочных бросков для команды.

Листинг 16 – Исходный код запроса к базе данных

```
public static async Task<Dictionary<GroupModel, Dictionary<TeamModel,
KeyValuePair<List<CurlingUser>, CurlingUser[]>>>>
GetTournamentSportsmanDataForTournament(TournamentModel tournament)
        {
            Dictionary<GroupModel, Dictionary<TeamModel, KeyValuePair<List<CurlingUser>,
CurlingUser[]>>> data = new Dictionary<GroupModel, Dictionary<TeamModel,
KeyValuePair<List<CurlingUser>, CurlingUser[]>>>();
            var groups = await Context.Groups.ToListAsync();
            groups = groups.Where(g => g.Tournament == tournament).ToList();
            foreach (var groupModel in groups)//получение групп
            {
                var teamsInGroup = await DbHelper.GetTeamsForGroup(groupModel);
//получение команд для группы
                var dict = new Dictionary<TeamModel, KeyValuePair<List<CurlingUser>,
CurlingUser[]>>();
                foreach (var teamModel in teamsInGroup)//получение спортсменов для
соревнований
                {
                    var sportsmen = await DbHelper.GetUsersForTeam(teamModel);
                    var coaches = new CurlingUser[sportsmen.Count];

                    for (int i = 0; i < sportsmen.Count; i++) //получение тренеров для
спортсменов
                    {
                        var coach = await DbHelper.GetCoachForSportsman(sportsmen[i]);
                        coaches[i] = coach;
                    }
                    var kv = new KeyValuePair<List<CurlingUser>,
CurlingUser[]>(sportsmen, coaches);
                    teamModel.TotalScore = await GetTotalScoreForTeam(teamModel);
//получение посановочных бросков
                    dict[teamModel] = kv;
                }
                data[groupModel] = dict;
            }
            return data;
           }
```

В листинге С.1 приложения С содержится исходный код запросов к базе данных.

4. РЕАЛИЗАЦИЯ

4.1. РЕАЛИЗАЦИЯ ИНТЕРФЕЙСОВ

1. Форма регистрации пользователя

Форма регистрации представлена на рисунке 4.1. Поле «пол» имеет выпадающий список для выбора пола (женский/мужской), поле «Город» имеет выпадающий список городов, в которых развивается кёрлинг, поле «Роль» имеет выпадающий список ролей (спортсмен/тренер/судья/член ФК), поле

«Тренер» имеет выпадающий список зарегистрированных тренеров. Поля, обязательные для заполнения, отмечены *.

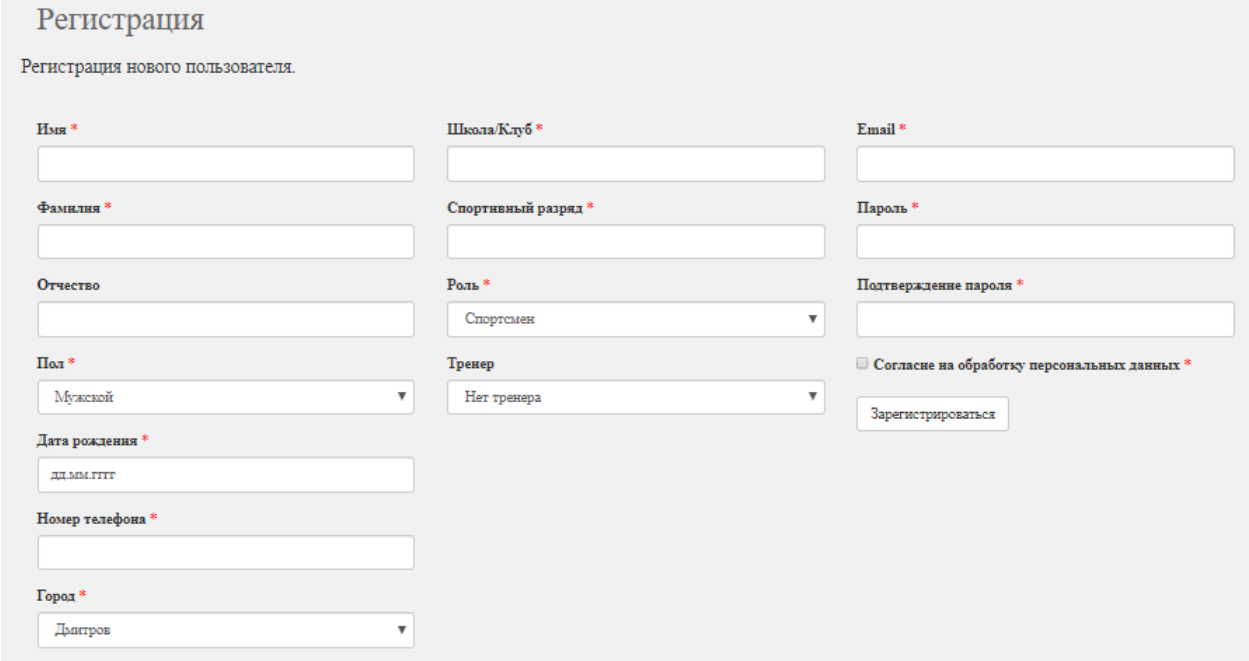


Рисунок 4.1 – Экранная форма регистрации пользователя

2. Форма авторизации пользователя

На рисунке 4.2 изображена форма входа на сайт для зарегистрированных пользователей. В форме ввода отслеживается необходимость заполнения всех полей, проверяется корректность вводимых данных.

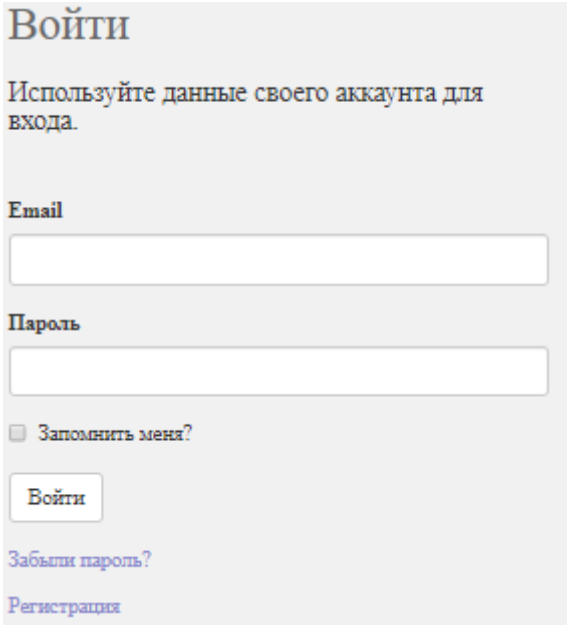


Рисунок 4.2 – Экранная форма входа на сайт

3. Форма редактирования профиля пользователя

На рисунке 4.3 представлен личный кабинет зарегистрированного пользователя.

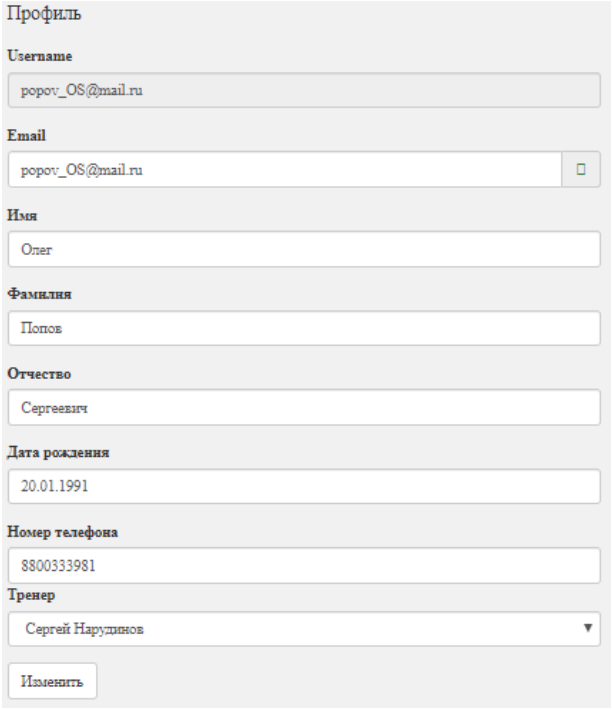
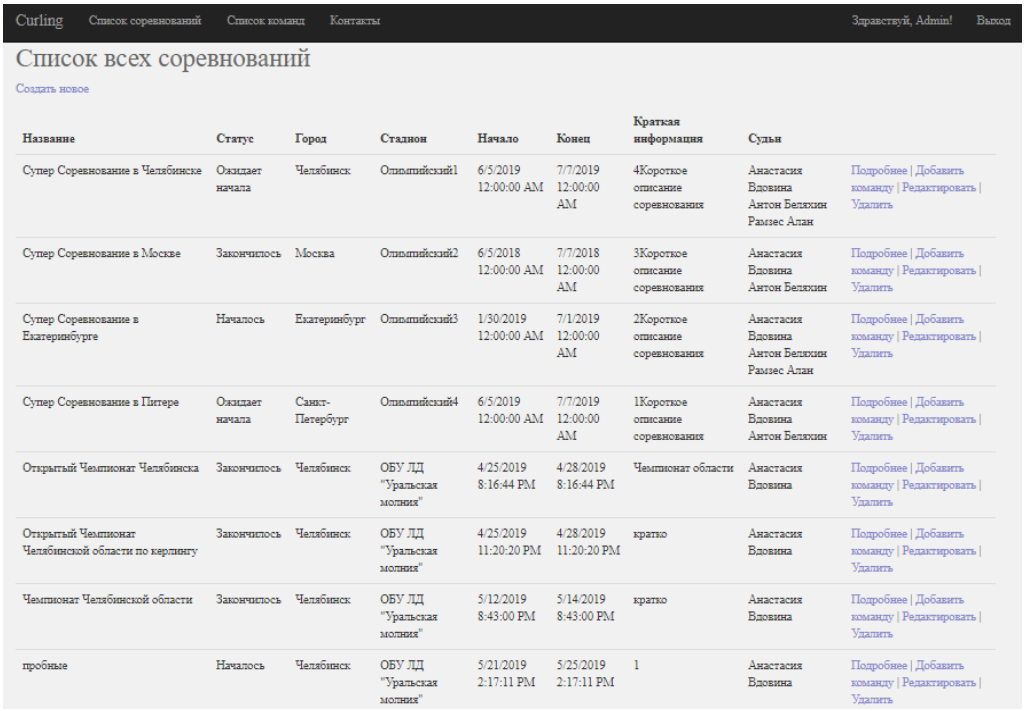


Рисунок 4.3 – Экранная форма личного кабинета пользователя

4. Список соревнований

На рисунке 4.4 изображен список всех добавленных соревнований.



Название	Статус	Город	Стадион	Начало	Конец	Краткая информация	Судьи	
Супер Соревнование в Челябинске	Ожидает начала	Челябинск	Олимпийский1	6/5/2019 12:00:00 AM	7/7/2019 12:00:00 AM	4Короткое описание соревнования	Анастасия Вдовина Антон Беляхин Рамзес Алан	Подробнее | Добавить команду | Редактировать | Удалить
Супер Соревнование в Москве	Закончилось	Москва	Олимпийский2	6/5/2018 12:00:00 AM	7/7/2018 12:00:00 AM	3Короткое описание соревнования	Анастасия Вдовина Антон Беляхин	Подробнее | Добавить команду | Редактировать | Удалить
Супер Соревнование в Екатеринбурге	Началось	Екатеринбург	Олимпийский3	1/30/2019 12:00:00 AM	7/1/2019 12:00:00 AM	2Короткое описание соревнования	Анастасия Вдовина Антон Беляхин Рамзес Алан	Подробнее | Добавить команду | Редактировать | Удалить
Супер Соревнование в Питере	Ожидает начала	Санкт-Петербург	Олимпийский4	6/5/2019 12:00:00 AM	7/7/2019 12:00:00 AM	1Короткое описание соревнования	Анастасия Вдовина Антон Беляхин	Подробнее | Добавить команду | Редактировать | Удалить
Открытый Чемпионат Челябинска	Закончилось	Челябинск	ОБУ ЛД "Уральская молния"	4/25/2019 8:16:44 PM	4/28/2019 8:16:44 PM	Чемпионат области	Анастасия Вдовина	Подробнее | Добавить команду | Редактировать | Удалить
Открытый Чемпионат Челябинской области по керлингу	Закончилось	Челябинск	ОБУ ЛД "Уральская молния"	4/25/2019 11:20:20 PM	4/28/2019 11:20:20 PM	кратко	Анастасия Вдовина	Подробнее | Добавить команду | Редактировать | Удалить
Чемпионат Челябинской области	Закончилось	Челябинск	ОБУ ЛД "Уральская молния"	5/12/2019 8:43:00 PM	5/14/2019 8:43:00 PM	кратко	Анастасия Вдовина	Подробнее | Добавить команду | Редактировать | Удалить
пробные	Началось	Челябинск	ОБУ ЛД "Уральская молния"	5/21/2019 2:17:11 PM	5/25/2019 2:17:11 PM	1	Анастасия Вдовина	Подробнее | Добавить команду | Редактировать | Удалить

Рисунок 4.4 – Экранная форма списка соревнований

5. Форма для создания новых соревнований

На рисунке 4.5 представлена форма для создания новых соревнований. В данной форме должны быть заполнены все поля. В списке судей будут показаны все зарегистрированные судьи.

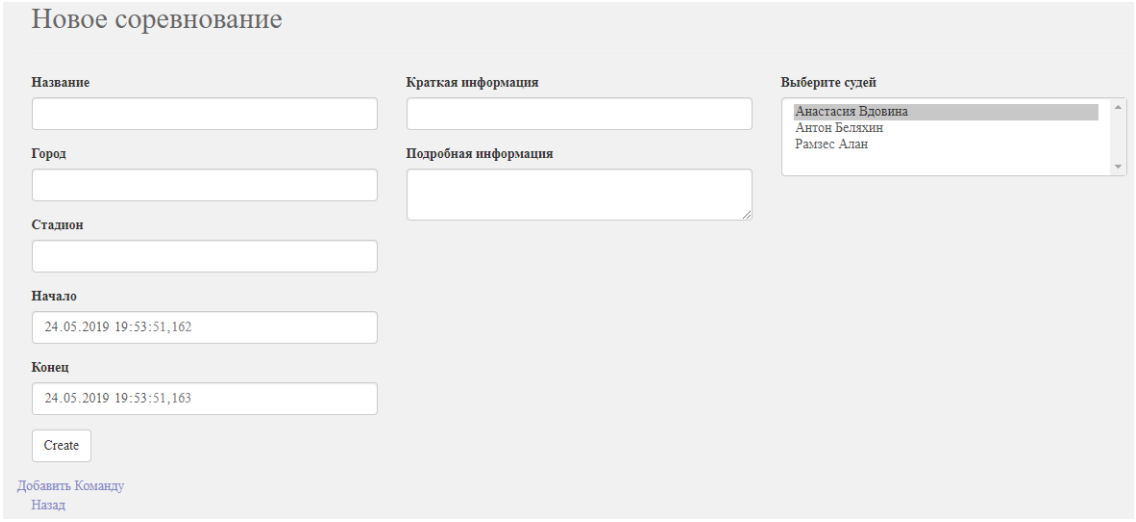


Рисунок 4.5 – Экранная форма создания соревнований

6. Информация о соревнованиях

На рисунке 4.6 представлена общая информация о соревнованиях после того, как они были созданы. Так же видно, что при добавлении соревнования появились следующие вкладки для него: «Общая информация», «Список команд», «Турнирная таблица», «Ход турнира», «Расписание игр», «Итоговый протокол».

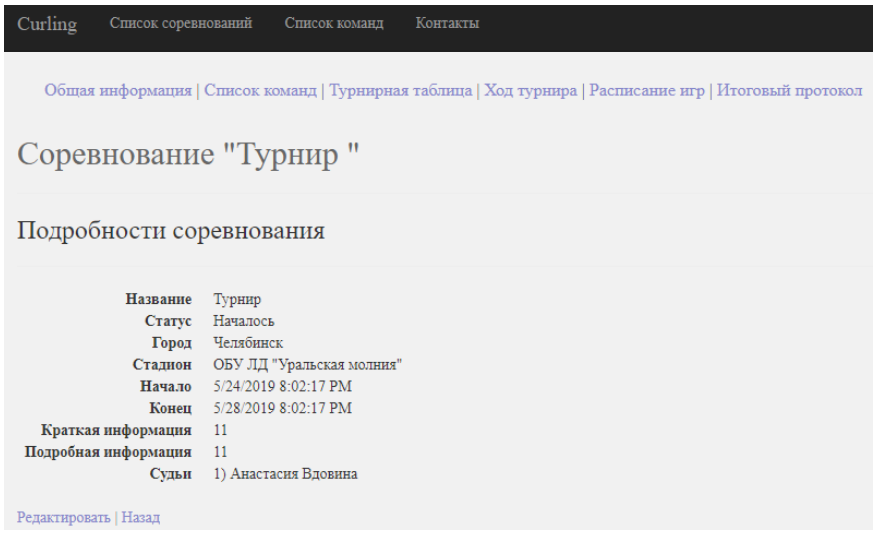


Рисунок 4.6 – Экранная форма общей информации о созданном соревновании

7. Команды на соревнованиях

На рисунке 4.7 показана вкладка «Список команд» до того, как на соревнования были добавлены команды для участия.

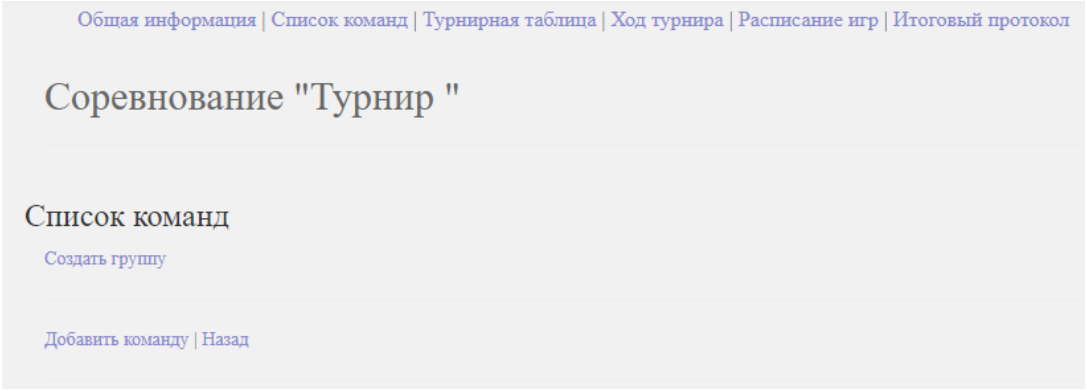


Рисунок 4.7 – Экранная форма вкладки «список команд» до добавления команд

На рисунке 4.8 показа форма для добавления новой команды. При добавлении спортсменов выпадает список зарегистрированных спортсменов.

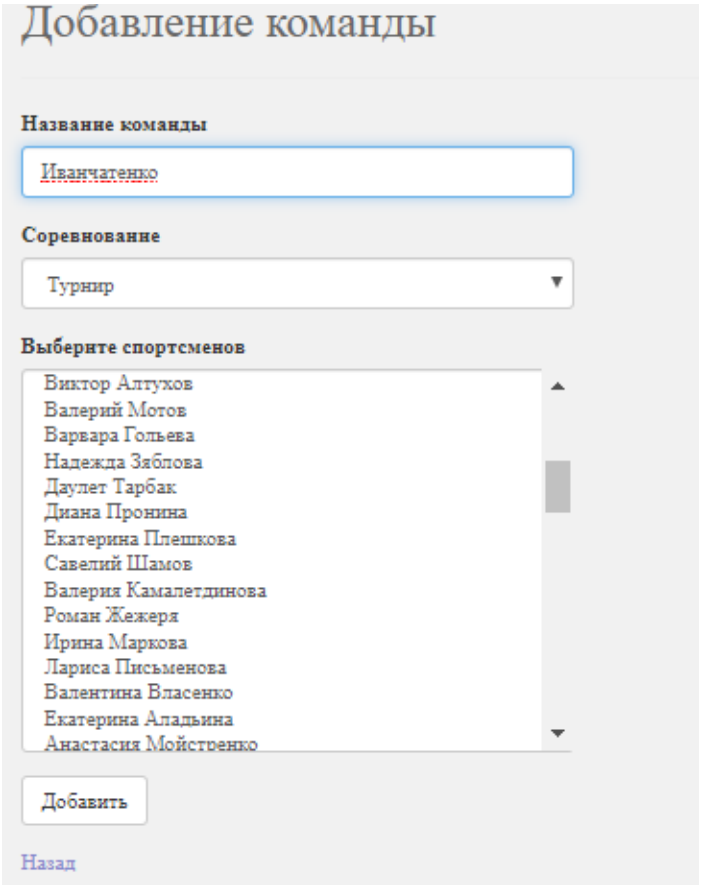


Рисунок 4.8 – Экранная форма добавления новой команды

На рисунке 4.9 показан список команд, которые добавлены на текущие соревнования.

8. Создание групп на соревнованиях

После добавления команд можно разделить их на группы. Добавление групп показано на рисунке 4.10.

9. Турнирные таблицы

После добавления групп во вкладке «Турнирная таблица» автоматически появятся турнирные таблицы для каждой группы (рисунок 4.12).

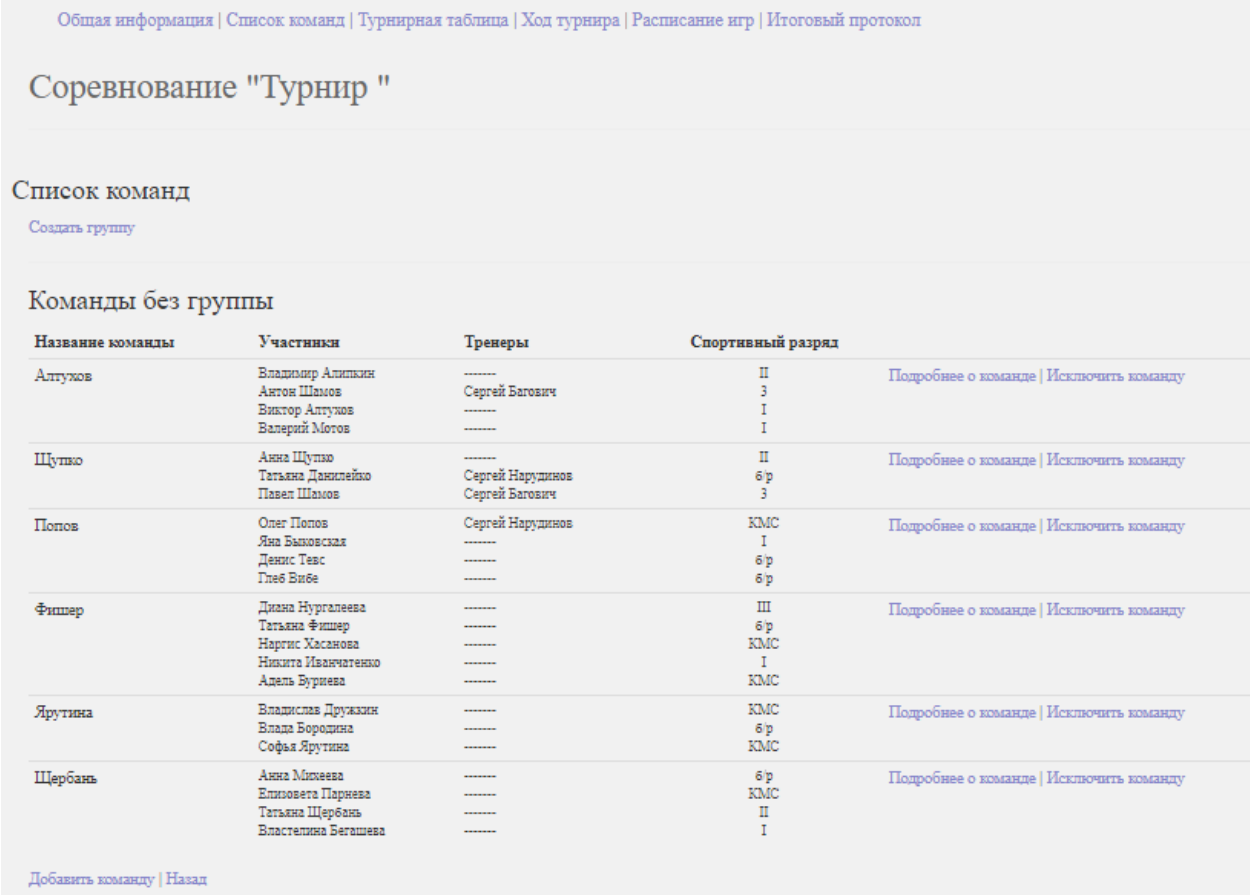


Рисунок 4.9 – Экранная форма списка команд, участвующих на соревнованиях

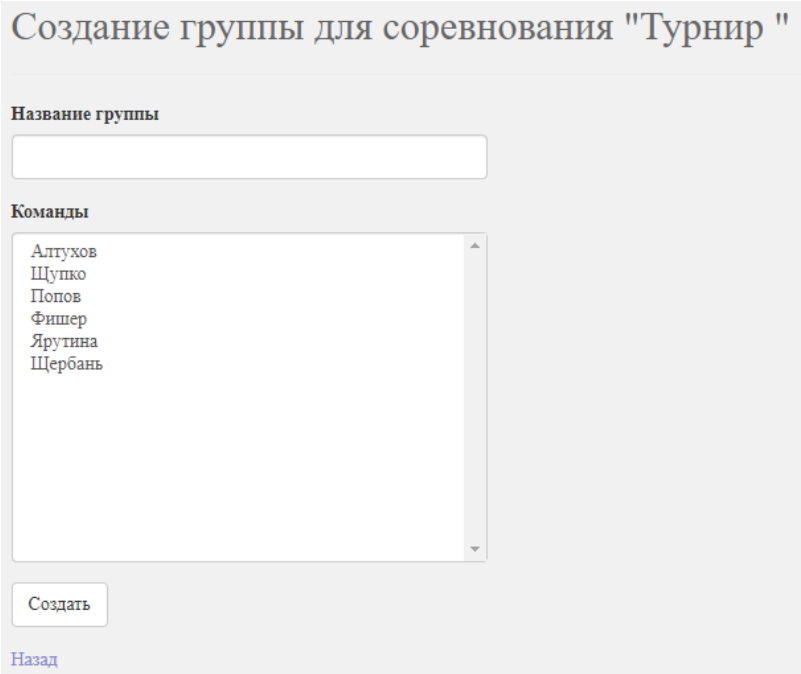


Рисунок 4.10 – Экранная форма создания групп на соревнованиях

На рисунке 4.11 показано, что создана группа, а так же показан список команд оставшихся без группы.

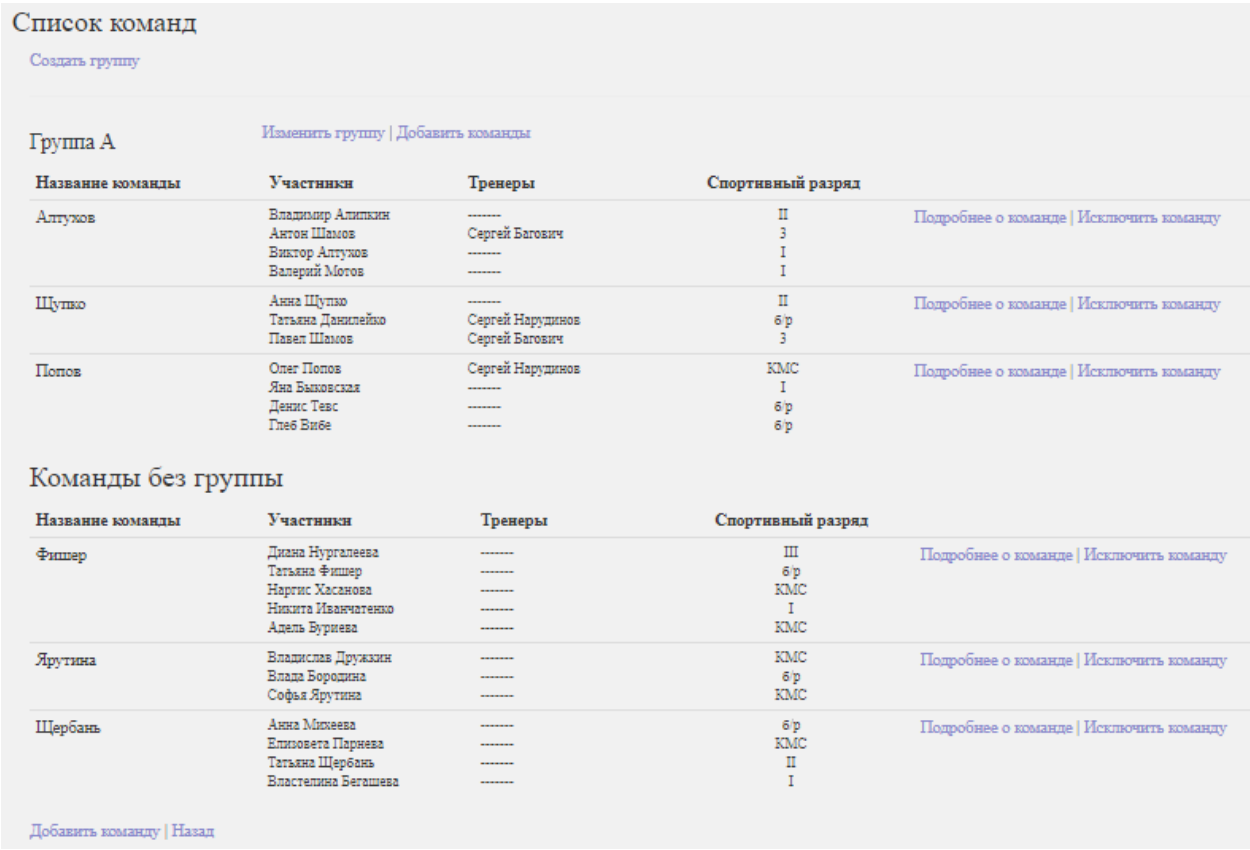


Рисунок 4.11 – Экранная форма группы и команд, которые еще не добавлены в группу

10. Форма для добавления туров

Перейдя на вкладку «Ход турнира» можно создать туры, а в каждом туре – игры. На рисунке 4.13 показано, как создаются туры, на рисунке 4.14 показано то, как выглядит ход турнира, когда тур добавлен.

11. Форма для добавления игр

После того как создан тур, можно добавить игры в этот тур (рисунок 4.15). При добавлении игры есть два поля для команд и два поля для постановки, значения постановки можно добавить позже.

12. Форма для ведения счета

Когда игра добавлена, она появляется в туре и есть возможность изменять постановочные броски и счет после того, как подойдет время начала тура (4.16).

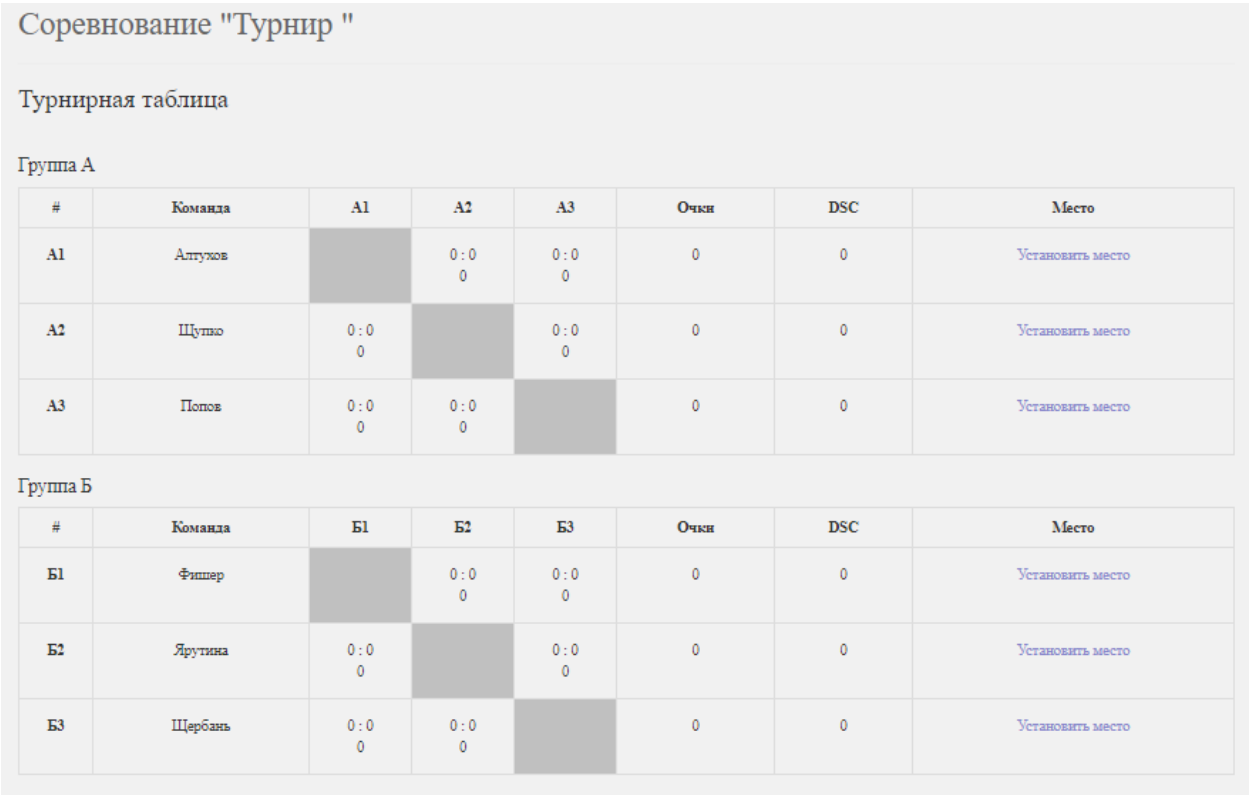


Рисунок 4.12 – Экранная форма турнирной таблицы

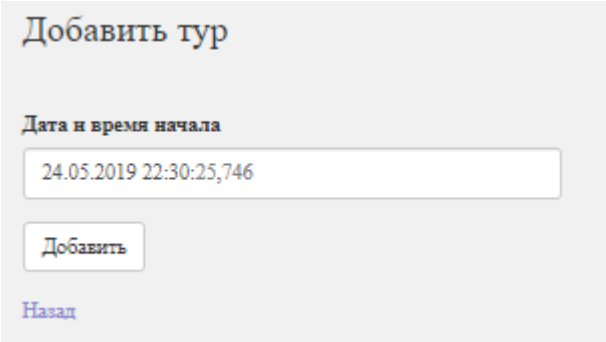


Рисунок 4.13 – Экранная форма создания тура

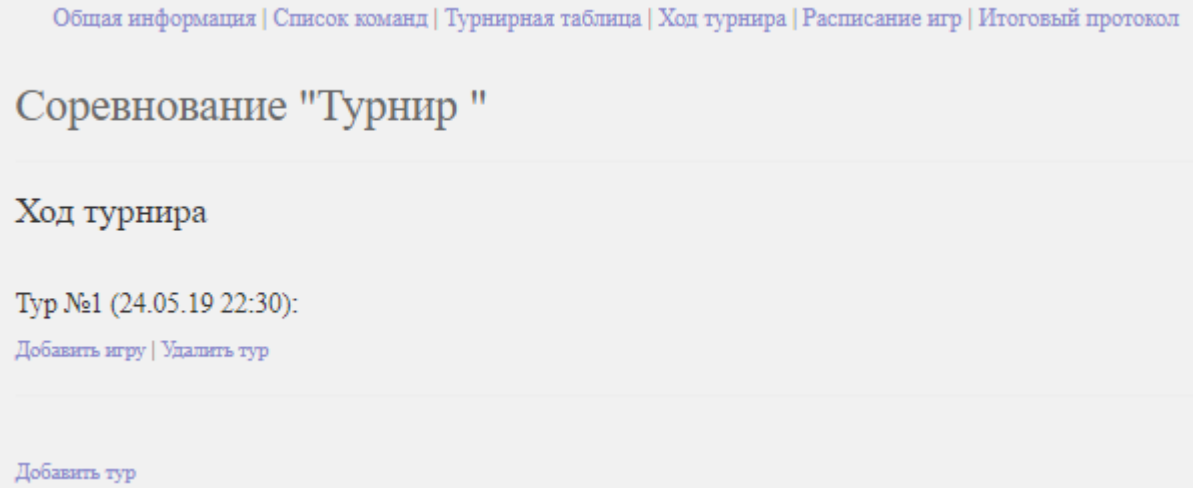


Рисунок 4.14 – Экранная форма хода турнира после добавления тура.

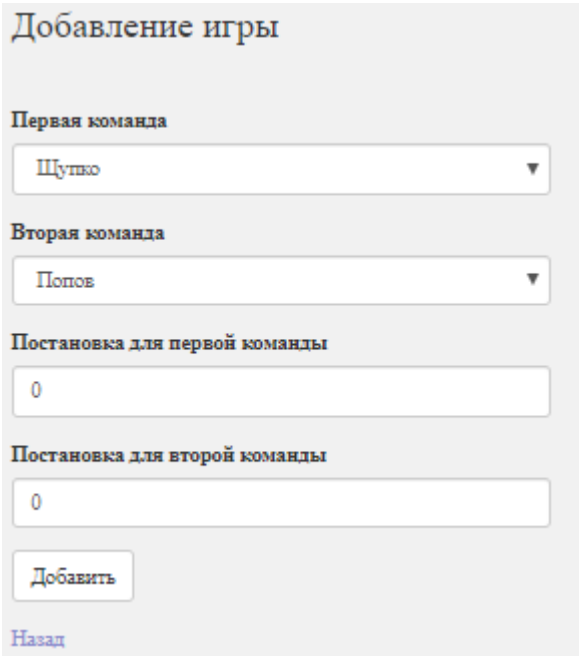


Рисунок 4.15 – Экранная форма добавление игры

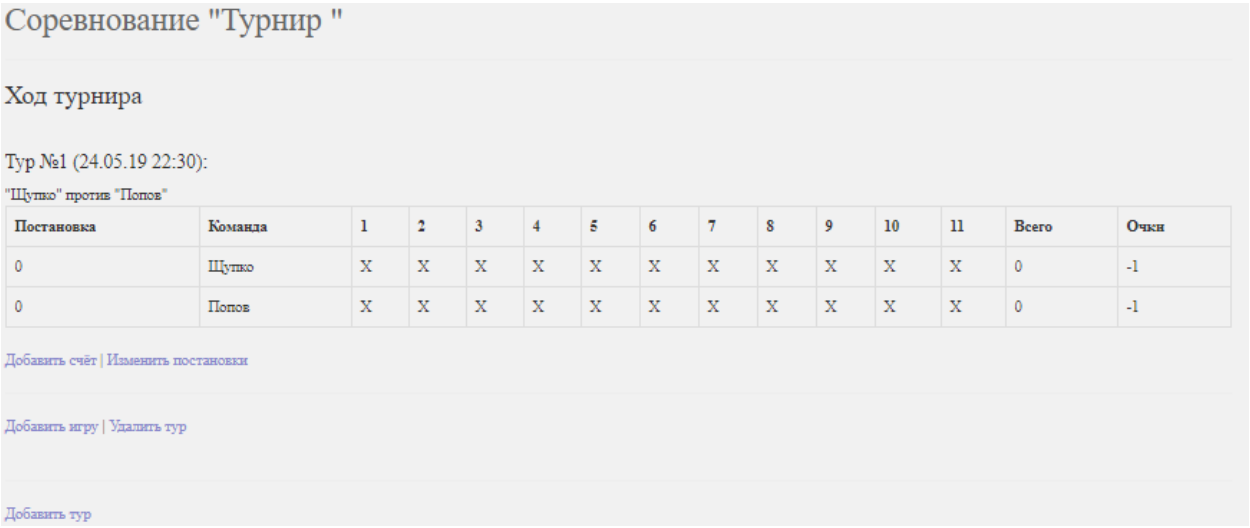


Рисунок 4.16 – Экранная форма вкладки «Ход турнира» после добавления игры.

На рисунке 4.17 показано как выглядят игры при добавлении счета и постановочных бросков. Сумма камней во всех Андах подсчитывается автоматически, так же как и очки (проигравшей команде 1 очко, выигравшей-3, если игра закончилась с равным счетом, то каждая команда получает по 2 очка).

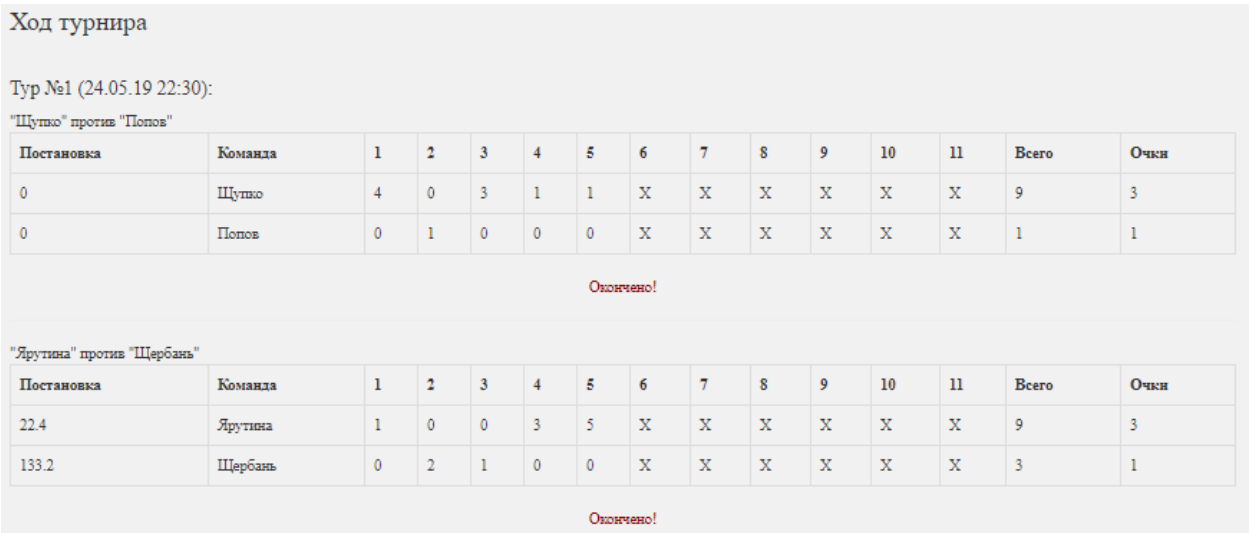


Рисунок 4.17 – Экранная форма ведение счета.

На рисунке 4.18 представлена турнирная таблица, в которую автоматически добавляется счет, а также будут автоматически подсчитаны сумма очков и сумма постановочных бросков (DSC). После того, как в турнирной таблице будут записаны все игры, можно будет установить места.

Турнирная таблица

Группа А

#	Команда	А1	А2	А3	Очки	DSC	Место
А1	Алтухов		0 : 0 0	0 : 0 0	0	0	Установить место
А2	Щупко	0 : 0 0		9 : 1 3	3	0	Установить место
А3	Попов	0 : 0 0	1 : 9 1		1	0	Установить место

Группа Б

#	Команда	Б1	Б2	Б3	Очки	DSC	Место
Б1	Фишер		5 : 3 3	1 : 5 1	4	266.8	Установить место
Б2	Ярутина	3 : 5 1		9 : 3 3	4	24.3	Установить место
Б3	Щербань	5 : 1 3	3 : 9 1		4	180.1	Установить место

Рисунок 4.19 – Экранная форма турнирной таблица

13. Расписание игр

На основании туров и добавленных игр формируется расписание игр (вкладка «Расписание»), расписание представлено на рисунке 4.20.

Соревнование "Турнир "

Расписание

Дата и время		
24.05.2019 22:30	"Щупко" : "Попов"	"Ярутина" : "Щербань"

Рисунок 4.20 – Экранная форма расписания игр

14. Итоговый протокол

Во вкладке «Итоговый протокол» (рисунок 4.21) содержится список команд, их состав, а также сумма набранных очков и постановки, количество побед. Итоговое место устанавливается вручную. После того, как установлено место, происходит сортировка, и команды выстраиваются в нужном порядке.

Итоговый протокол

Место	Название команды	Состав команды	С/Р	Личные тренеры	Очки	Победы	DSC
Установить	Алтухов	Владимир Алипкин Антон Шамов Виктор Алтухов Валерий Мотов	II 3 I I	------- Сергей Багович ------- -------	0	0	0
Установить	Щупко	Анна Щупко Татьяна Данилейко Павел Шамов	II б/р 3	------- Сергей Нарудинов Сергей Багович	3	1	0
Установить	Попов	Олег Попов Яна Быковская Денис Тевс Глеб Вибе	КМС I б/р б/р	Сергей Нарудинов ------- ------- -------	2	0	19
Установить	Фишер	Диана Нургалеева Татьяна Фишер Наргис Хасанова Никита Иванчатенко Адель Буриева	III б/р КМС I КМС	------- ------- ------- ------- -------	7	2	321.8
Установить	Ярутина	Владислав Дружкин Влада Бородина Софья Ярутина	КМС б/р КМС	------- ------- -------	6	2	24.3
Установить	Щербань	Анна Михеева Елизовета Парнева Татьяна Щербань Властелина Бегашева	б/р КМС II I	------- ------- ------- -------	2	0	180.1

Рисунок 4.21 – Экранная форма итогового протокола

5. ТЕСТИРОВАНИЕ

1. Тестирование регистрации авторизации пользователей

Для тестирования работы с помощью веб-приложения был проведен Открытый Чемпионат Челябинской области 2019 года (рисунок 5.1). Для начала все участники соревнования зарегистрировались. После этого были добавлены команды и распределены на группы. А, Б, С и Д (рисунки 5.2 - 5.5). Далее были добавлены туры и игры, на основании которых было сформировано расписание соревнований (рисунок 5.6). Во время тестирования программы постановочные броски еще не подсчитывались, поэтому в окошке для постановок - нули. На рисунках 5.7 – 5.25 показан ход турнира, то есть, изображен счет для каждой игры, подсчитан итоговый счет, а также выставлены очки (1 – проигрыш, 2 – ничья, 3 – победа). После того, как игра закончена, счет и набранные очки автоматически добавляются в турнирную таблицу (рисунки 5.26 – 5.29). На рисунках 5.30 – 5.34 показан Итоговый протокол соревнований.

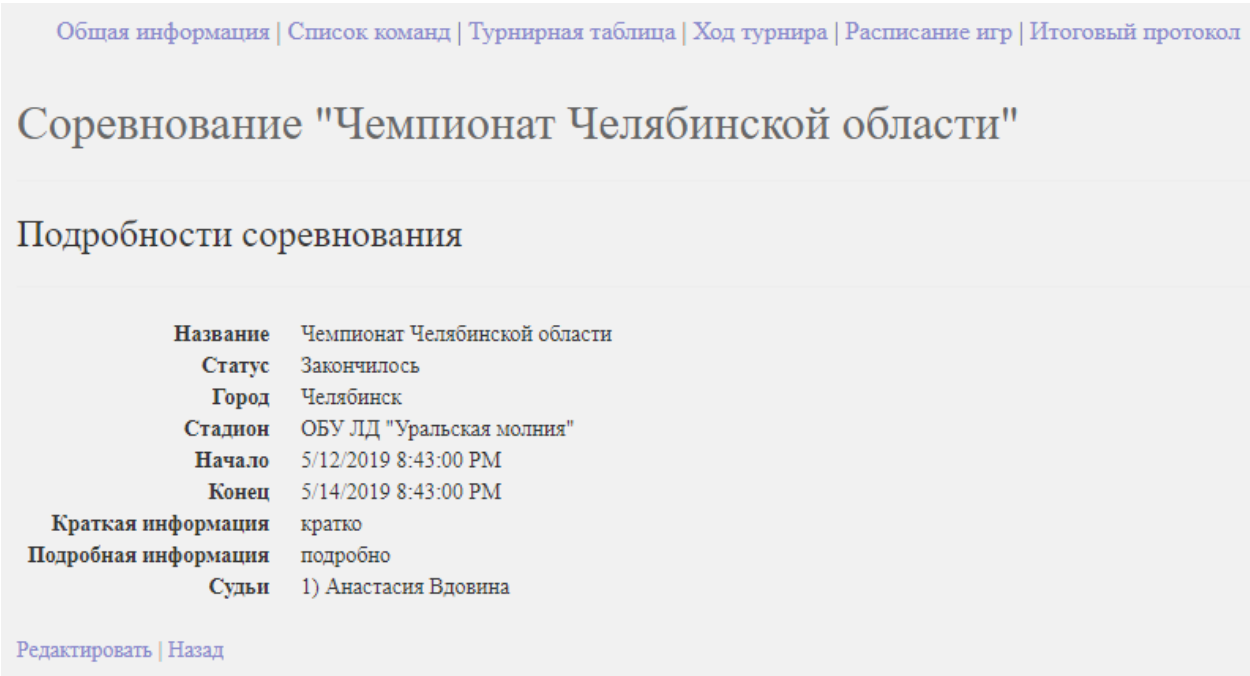


Рисунок 5.1 – Экранная форма сведений о соревновании

Рисунок 5.2 – Экранная форма команд в группе А

Группа Б

Изменить группу | Добавить команды

Название команды	Участники	Тренеры	Спортивный разряд	
Гольева	Варвара Гольева	-------	б/р	Подробнее о команде | Исключить команду
	Елизавета Свиридова	-------	б/р	
	Ольга Пономарева	-------	б/р	
	Дарья Филимонова	-------	б/р	
	Влада Бородина	-------	б/р	
Бартко	Дарья Тугай	-------	КМС	Подробнее о команде | Исключить команду
	Александра Бартко	-------	КМС	
	Динара Байгнина	-------	КМС	
	Анна Шутова	-------	КМС	
	Елизавета Супереко	-------	КМС	
Экспромт	Яна Быковская	-------	I	Подробнее о команде | Исключить команду
	Лариса Письменова	-------	I	
	Ирина Лисовец	-------	I	
	Надежда Кобылянец	-------	I	
Уфа	Анна Морозова	Сергей Багович	III	Подробнее о команде | Исключить команду
	Екатерина Плешкова	-------	III	
	Элина Ханнанова	-------	III	
	Диана Нургалеева	-------	III	
Deaf-girl	Валерия Камалетдинова	-------	КМС	Подробнее о команде | Исключить команду
	Анастасия Мойстренко	-------	КМС	
	Наргис Хасанова	-------	КМС	
	Елизовета Парнева	-------	КМС	

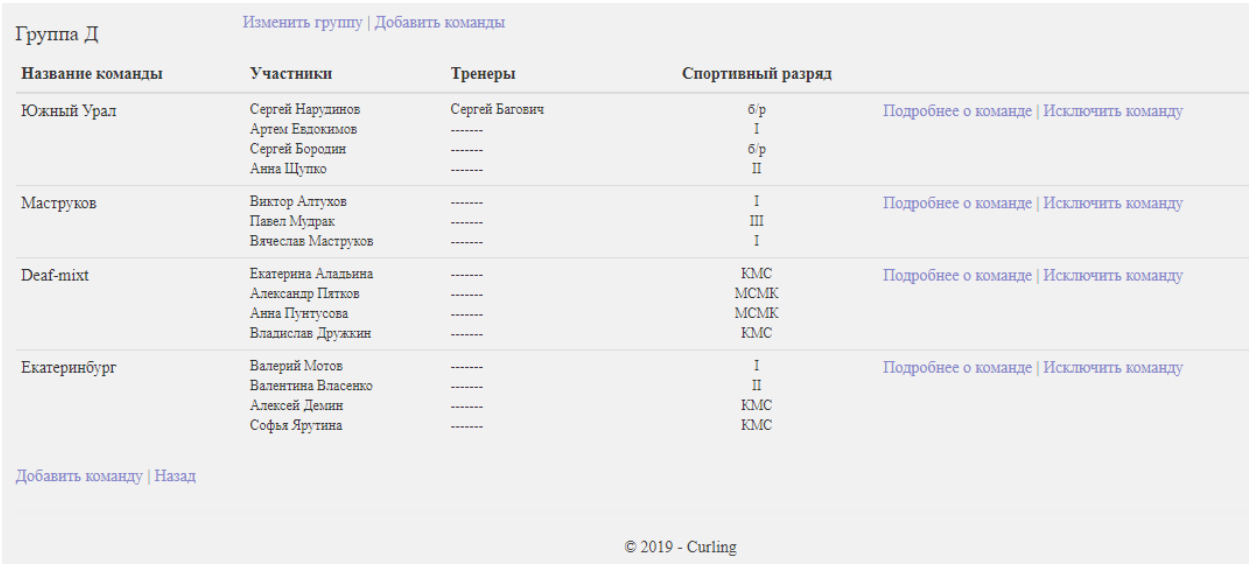
Рисунок 5.3 – Экранная форма команд в группе Б

Группа С

Изменить группу | Добавить команды

Название команды	Участники	Тренеры	Спортивный разряд	
Иванчатенко	Владимир Алипкин	-------	II	Подробнее о команде | Исключить команду
	Дмитрий Соломатин	-------	I	
	Никита Иванчатенко	-------	I	
	Константин Окружной	-------	I	
Самохин	Антон Мишин	Сергей Багович	II	Подробнее о команде | Исключить команду
	Антон Самохин	-------	II	
	Михаил Саморуков	-------	I	
	Никита Москвин	-------	II	
УралИнвестМет	Андрей Двойных	Сергей Багович	б/р	Подробнее о команде | Исключить команду
	Дмитрий Артемов	-------	б/р	
	Денис Тевс	-------	б/р	
	Сергей Ломакин	-------	б/р	
Deaf-man	Роман Кузьмин	-------	КМС	Подробнее о команде | Исключить команду
	Никита Дружкин	-------	I	
	Юрий Макеев	-------	МСМК	
	Олег Дарчиев	-------	МСМК	
Север (Казахстан)	Глеб Вибе	-------	б/р	Подробнее о команде | Исключить команду
	Даулет Тарбак	-------	КМС	
	Роман Жежеря	-------	КМС	
	Денис Зернов	-------	МС	
	Иван Буняшин	-------	КМС	
	Фархат Раев	-------	б/р	

Рисунок 5.4 – Экранная форма команд в группе С



Группа Д

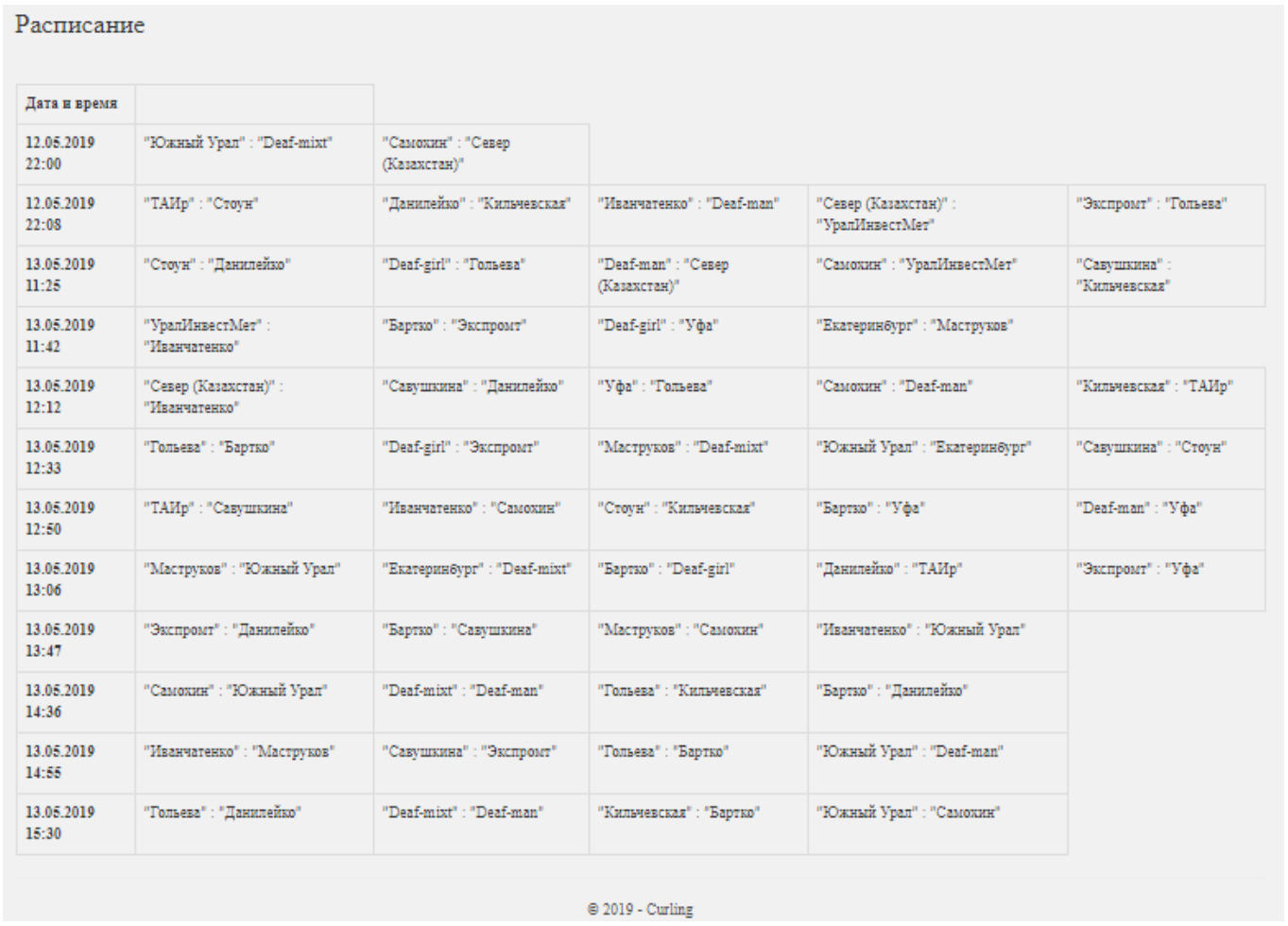
Изменить группу | Добавить команды

Название команды	Участники	Тренеры	Спортивный разряд	
Южный Урал	Сергей Нарудинов	Сергей Багович	б/р	Подробнее о команде | Исключить команду
	Артем Евдокимов	-------	I	
	Сергей Бородин	-------	б/р	
	Анна Щупко	-------	II	
Маструков	Виктор Алтухов	-------	I	Подробнее о команде | Исключить команду
	Павел Мудрак	-------	III	
	Вячеслав Маструков	-------	I	
Deaf-mixt	Екатерина Алальина	-------	КМС	Подробнее о команде | Исключить команду
	Александр Пятков	-------	МСМК	
	Анна Пунтусова	-------	МСМК	
	Владислав Дружкин	-------	КМС	
Екатеринбург	Валерий Мотов	-------	I	Подробнее о команде | Исключить команду
	Валентина Власенко	-------	II	
	Алексей Демин	-------	КМС	
	Софья Ярутина	-------	КМС	

Добавить команду | Назад

© 2019 - Curling

Рисунок 5.5 – Экранная форма команд в группе Д

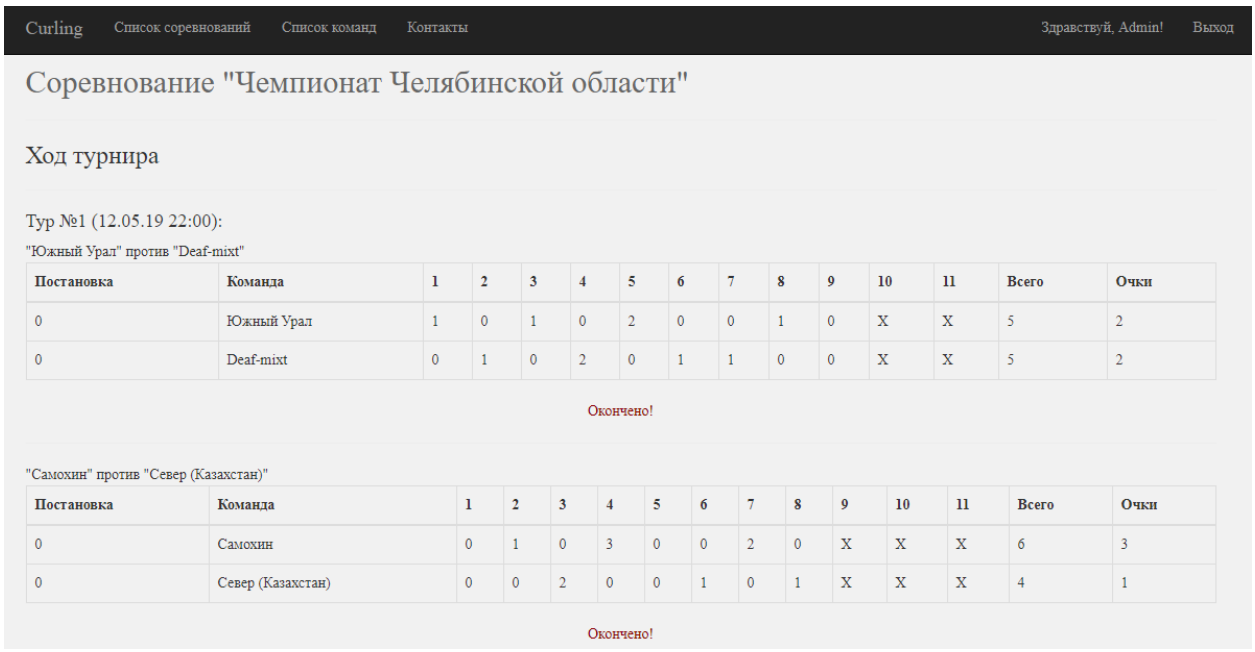


Расписание

Дата и время					
12.05.2019 22:00	"Южный Урал" : "Deaf-mixt"	"Самохин" : "Север (Казахстан)"			
12.05.2019 22:08	"ТАИр" : "Стоун"	"Данилейко" : "Кильчевская"	"Иванчатенко" : "Deaf-man"	"Север (Казахстан)" : "УралИнвестМет"	"Экспромт" : "Гольева"
13.05.2019 11:25	"Стоун" : "Данилейко"	"Deaf-girl" : "Гольева"	"Deaf-man" : "Север (Казахстан)"	"Самохин" : "УралИнвестМет"	"Савушкина" : "Кильчевская"
13.05.2019 11:42	"УралИнвестМет" : "Иванчатенко"	"Бартко" : "Экспромт"	"Deaf-girl" : "Уфа"	"Екатеринбург" : "Маструков"	
13.05.2019 12:12	"Север (Казахстан)" : "Иванчатенко"	"Савушкина" : "Данилейко"	"Уфа" : "Гольева"	"Самохин" : "Deaf-man"	"Кильчевская" : "ТАИр"
13.05.2019 12:33	"Гольева" : "Бартко"	"Deaf-girl" : "Экспромт"	"Маструков" : "Deaf-mixt"	"Южный Урал" : "Екатеринбург"	"Савушкина" : "Стоун"
13.05.2019 12:50	"ТАИр" : "Савушкина"	"Иванчатенко" : "Самохин"	"Стоун" : "Кильчевская"	"Бартко" : "Уфа"	"Deaf-man" : "Уфа"
13.05.2019 13:06	"Маструков" : "Южный Урал"	"Екатеринбург" : "Deaf-mixt"	"Бартко" : "Deaf-girl"	"Данилейко" : "ТАИр"	"Экспромт" : "Уфа"
13.05.2019 13:47	"Экспромт" : "Данилейко"	"Бартко" : "Савушкина"	"Маструков" : "Самохин"	"Иванчатенко" : "Южный Урал"	
13.05.2019 14:36	"Самохин" : "Южный Урал"	"Deaf-mixt" : "Deaf-man"	"Гольева" : "Кильчевская"	"Бартко" : "Данилейко"	
13.05.2019 14:55	"Иванчатенко" : "Маструков"	"Савушкина" : "Экспромт"	"Гольева" : "Бартко"	"Южный Урал" : "Deaf-man"	
13.05.2019 15:30	"Гольева" : "Данилейко"	"Deaf-mixt" : "Deaf-man"	"Кильчевская" : "Бартко"	"Южный Урал" : "Самохин"	

© 2019 - Curling

Рисунок 5.6 – Экранная форма расписания игр



Curling Список соревнований Список команд Контакты Здравствуй, Admin! Выход

Соревнование "Чемпионат Челябинской области"

Ход турнира

Тур №1 (12.05.19 22:00):

"Южный Урал" против "Deaf-mixt"

Постановка	Команда	1	2	3	4	5	6	7	8	9	10	11	Всего	Очки
0	Южный Урал	1	0	1	0	2	0	0	1	0	X	X	5	2
0	Deaf-mixt	0	1	0	2	0	1	1	0	0	X	X	5	2

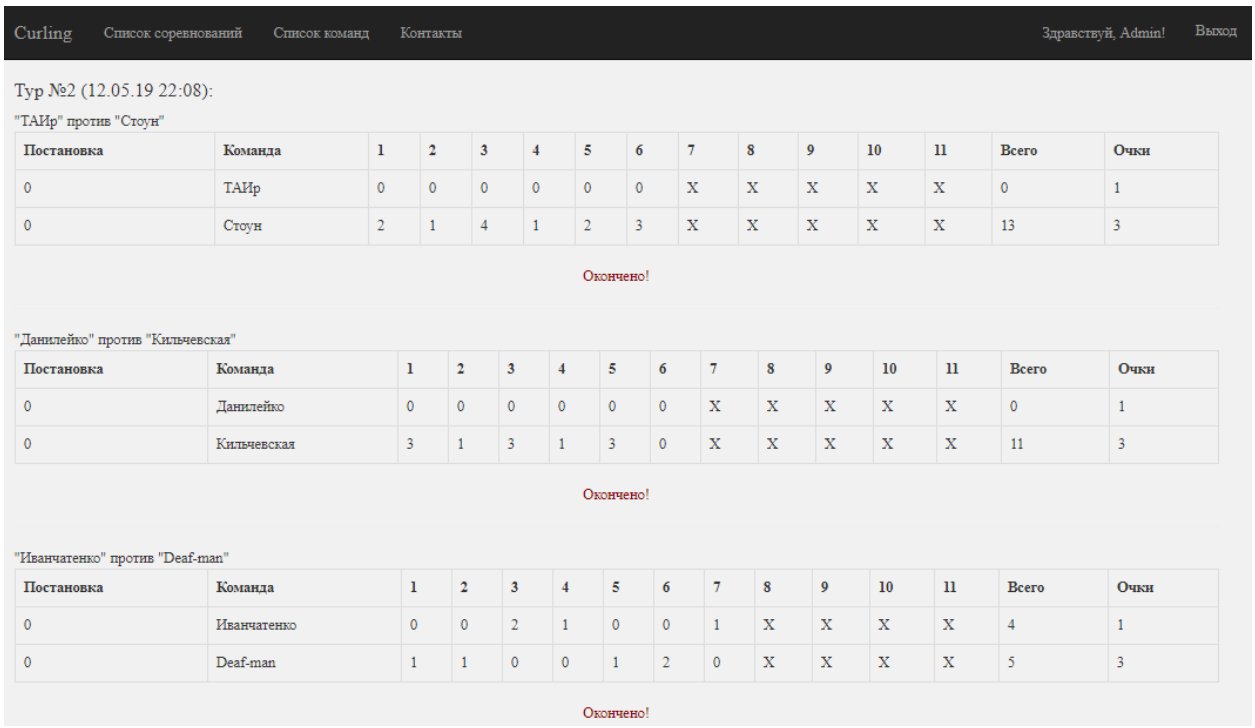
Окончено!

"Самохин" против "Север (Казахстан)"

Постановка	Команда	1	2	3	4	5	6	7	8	9	10	11	Всего	Очки
0	Самохин	0	1	0	3	0	0	2	0	X	X	X	6	3
0	Север (Казахстан)	0	0	2	0	0	1	0	1	X	X	X	4	1

Окончено!

Рисунок 5.7 – Экранная форма хода турнира



Curling Список соревнований Список команд Контакты Здравствуй, Admin! Выход

Тур №2 (12.05.19 22:08):

"ТАИр" против "Стоун"

Постановка	Команда	1	2	3	4	5	6	7	8	9	10	11	Всего	Очки
0	ТАИр	0	0	0	0	0	0	X	X	X	X	X	0	1
0	Стоун	2	1	4	1	2	3	X	X	X	X	X	13	3

Окончено!

"Данилейко" против "Кильчевская"

Постановка	Команда	1	2	3	4	5	6	7	8	9	10	11	Всего	Очки
0	Данилейко	0	0	0	0	0	0	X	X	X	X	X	0	1
0	Кильчевская	3	1	3	1	3	0	X	X	X	X	X	11	3

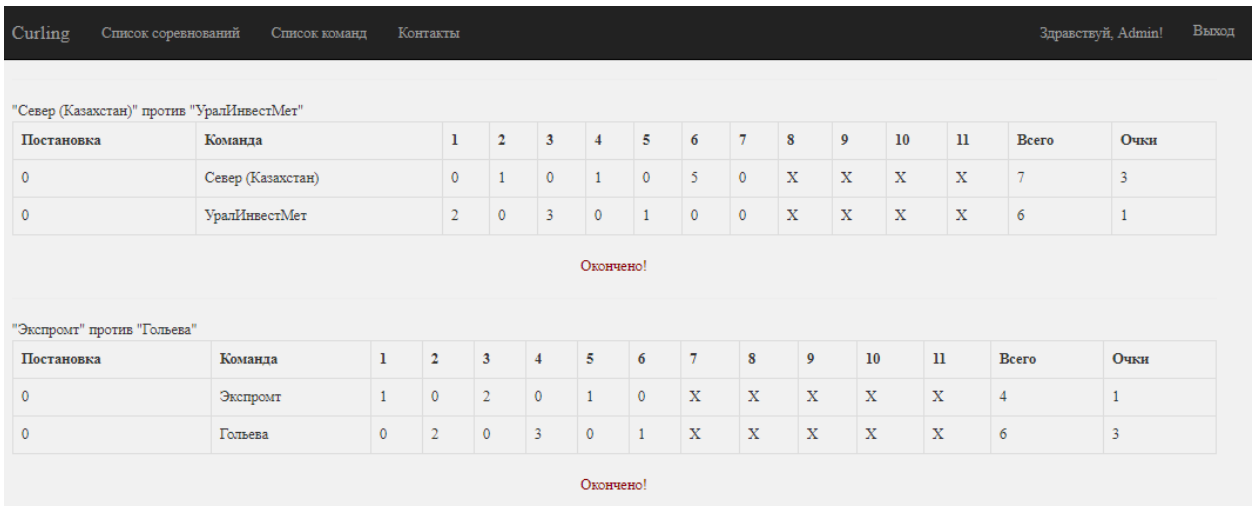
Окончено!

"Иванчатенко" против "Deaf-man"

Постановка	Команда	1	2	3	4	5	6	7	8	9	10	11	Всего	Очки
0	Иванчатенко	0	0	2	1	0	0	1	X	X	X	X	4	1
0	Deaf-man	1	1	0	0	1	2	0	X	X	X	X	5	3

Окончено!

Рисунок 5.8 – Экранная форма хода турнира



Curling Список соревнований Список команд Контакты Здравствуй, Admin! Выход

"Север (Казахстан)" против "УралИнвестМет"

Постановка	Команда	1	2	3	4	5	6	7	8	9	10	11	Всего	Очки
0	Север (Казахстан)	0	1	0	1	0	5	0	X	X	X	X	7	3
0	УралИнвестМет	2	0	3	0	1	0	0	X	X	X	X	6	1

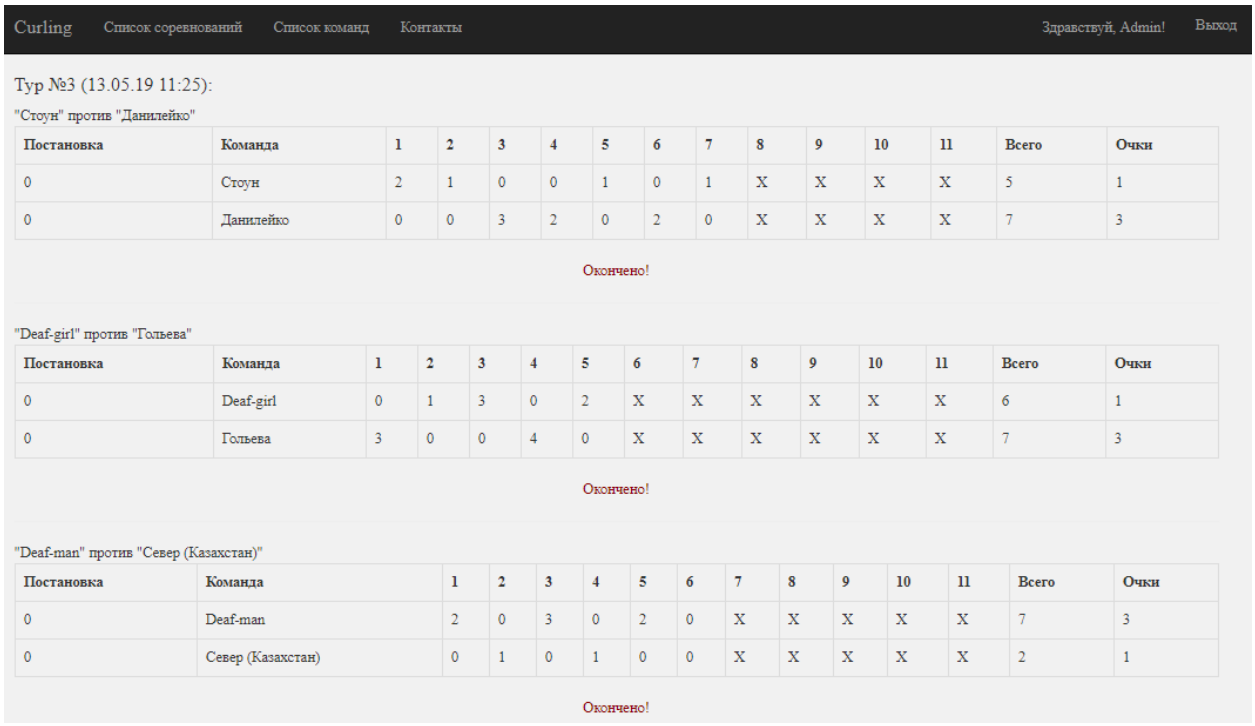
Окончено!

"Экспромт" против "Гольева"

Постановка	Команда	1	2	3	4	5	6	7	8	9	10	11	Всего	Очки
0	Экспромт	1	0	2	0	1	0	X	X	X	X	X	4	1
0	Гольева	0	2	0	3	0	1	X	X	X	X	X	6	3

Окончено!

Рисунок 5.9 – Экранная форма хода турнира



Curling Список соревнований Список команд Контакты Здравствуй, Admin! Выход

Тур №3 (13.05.19 11:25):

"Стоун" против "Данилейко"

Постановка	Команда	1	2	3	4	5	6	7	8	9	10	11	Всего	Очки
0	Стоун	2	1	0	0	1	0	1	X	X	X	X	5	1
0	Данилейко	0	0	3	2	0	2	0	X	X	X	X	7	3

Окончено!

"Deaf-girl" против "Гольева"

Постановка	Команда	1	2	3	4	5	6	7	8	9	10	11	Всего	Очки
0	Deaf-girl	0	1	3	0	2	X	X	X	X	X	X	6	1
0	Гольева	3	0	0	4	0	X	X	X	X	X	X	7	3

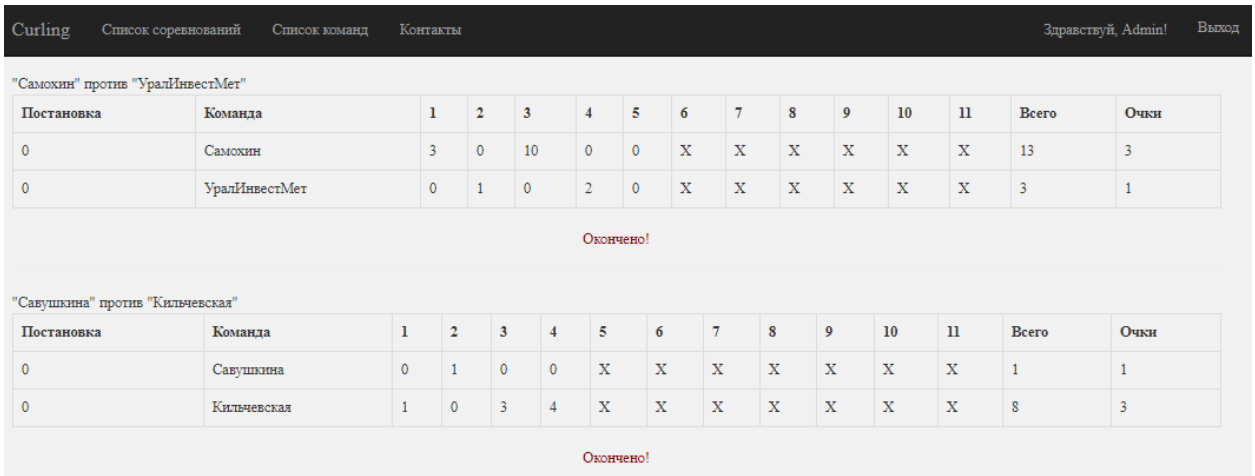
Окончено!

"Deaf-man" против "Север (Казахстан)"

Постановка	Команда	1	2	3	4	5	6	7	8	9	10	11	Всего	Очки
0	Deaf-man	2	0	3	0	2	0	X	X	X	X	X	7	3
0	Север (Казахстан)	0	1	0	1	0	0	X	X	X	X	X	2	1

Окончено!

Рисунок 5.10 – Экранная форма хода турнира



Curling Список соревнований Список команд Контакты Здравствуй, Admin! Выход

"Самохин" против "УралИнвестМет"

Постановка	Команда	1	2	3	4	5	6	7	8	9	10	11	Всего	Очки
0	Самохин	3	0	10	0	0	X	X	X	X	X	X	13	3
0	УралИнвестМет	0	1	0	2	0	X	X	X	X	X	X	3	1

Окончено!

"Савушкина" против "Кильчевская"

Постановка	Команда	1	2	3	4	5	6	7	8	9	10	11	Всего	Очки
0	Савушкина	0	1	0	0	X	X	X	X	X	X	X	1	1
0	Кильчевская	1	0	3	4	X	X	X	X	X	X	X	8	3

Окончено!

Рисунок 5.11 – Экранная форма хода турнира

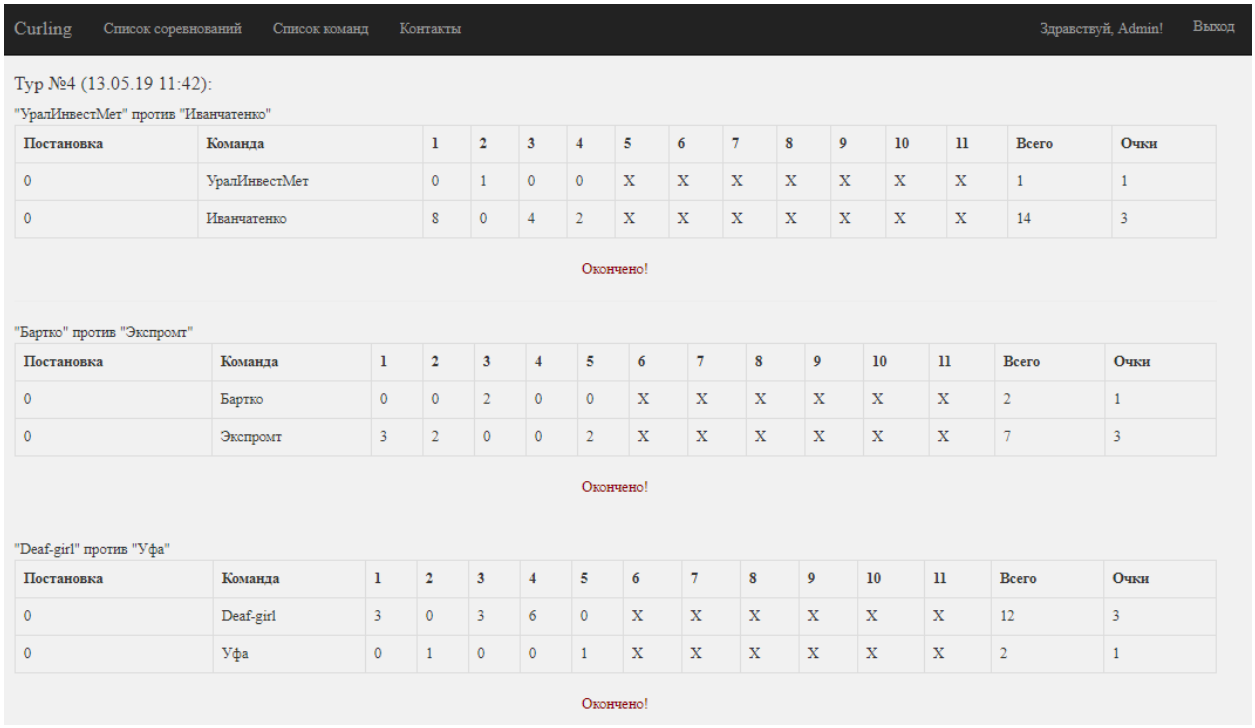


Рисунок 5.12 – Экранная форма хода турнира

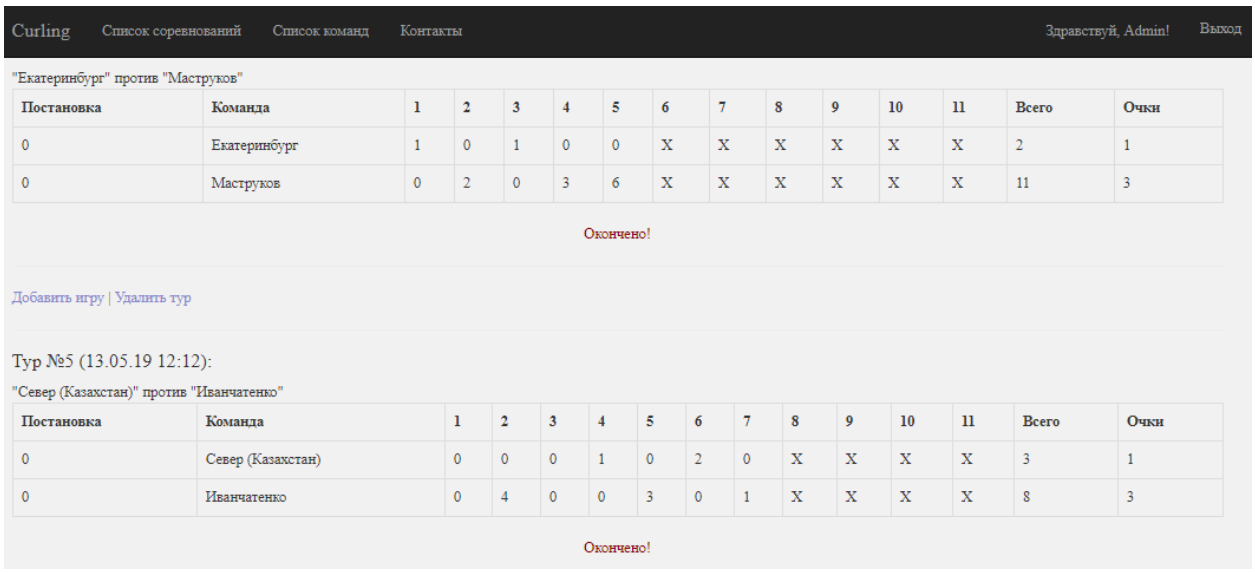


Рисунок 5.13 – Экранная форма хода турнира

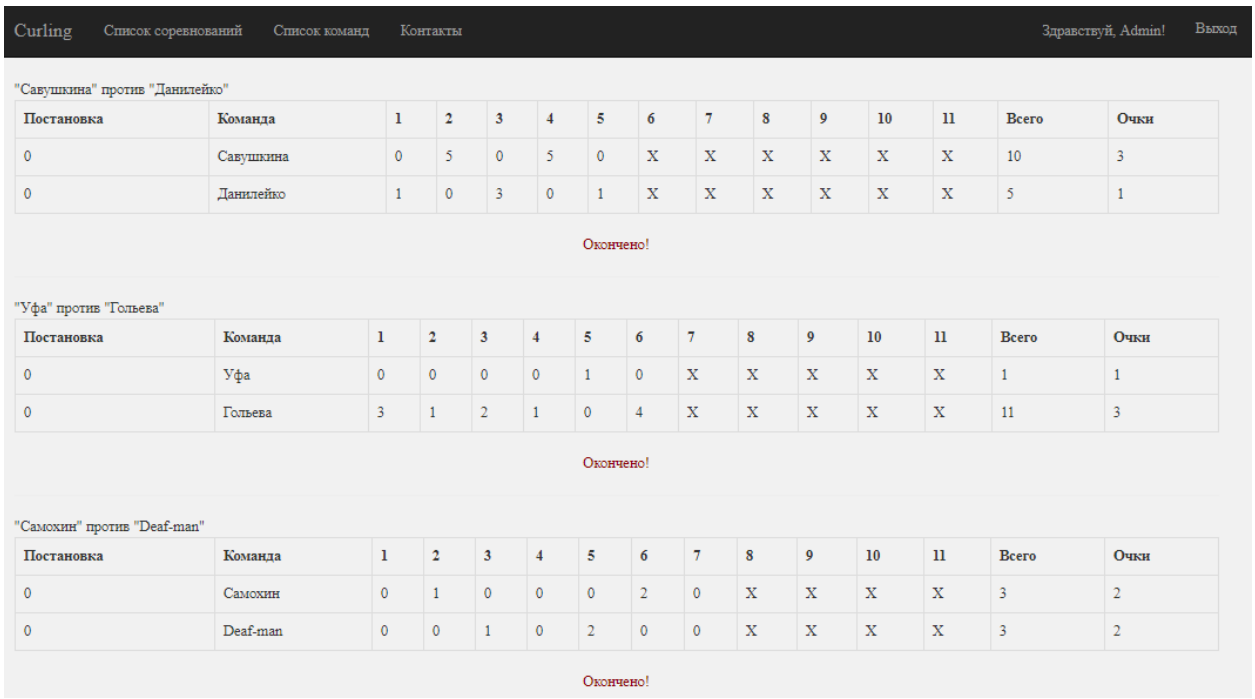


Рисунок 5.14 – Экранная форма хода турнира турнира

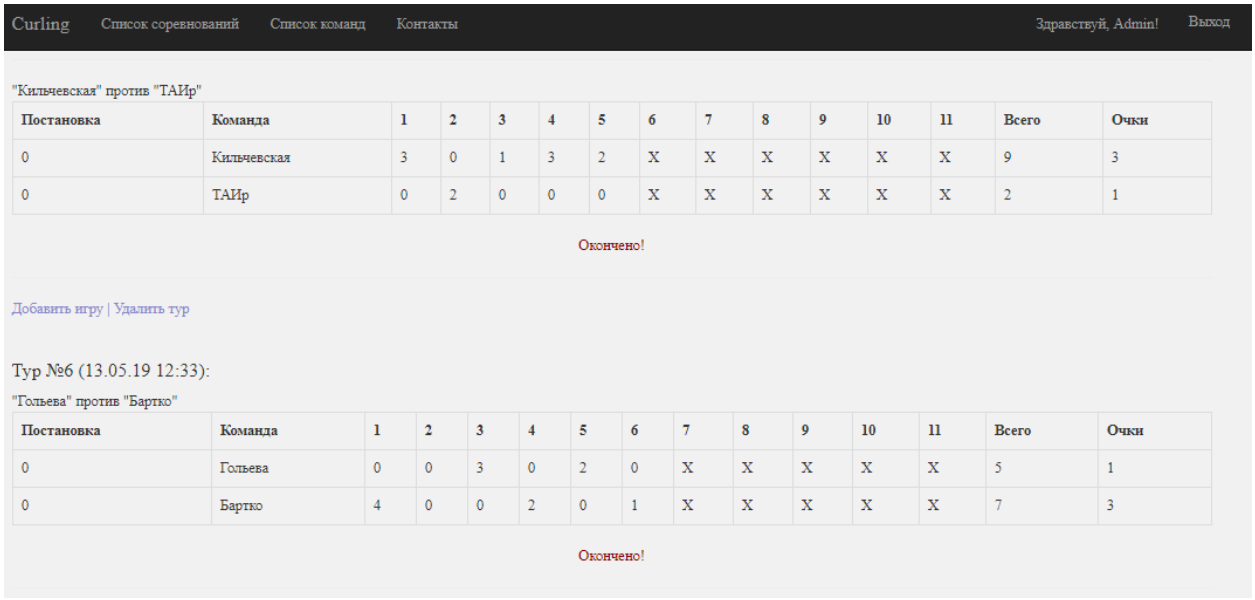


Рисунок 5.15 – Экранная форма хода турнира

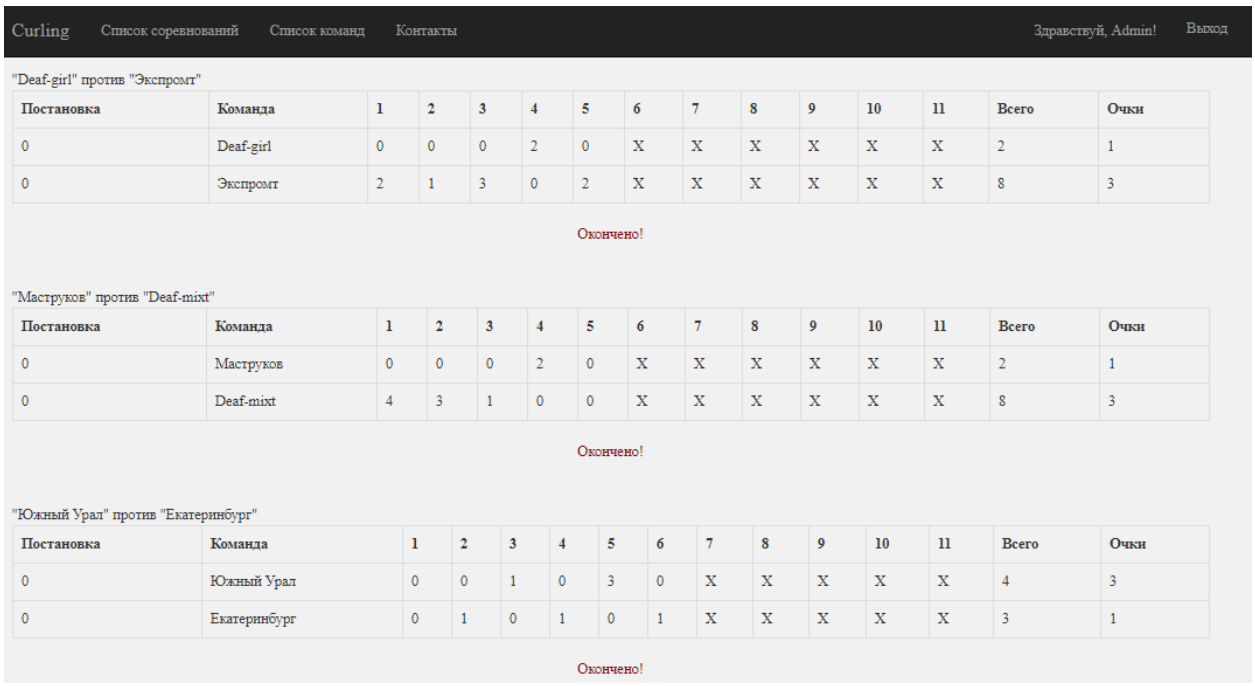


Рисунок 5.16 – Экранная форма хода турнира

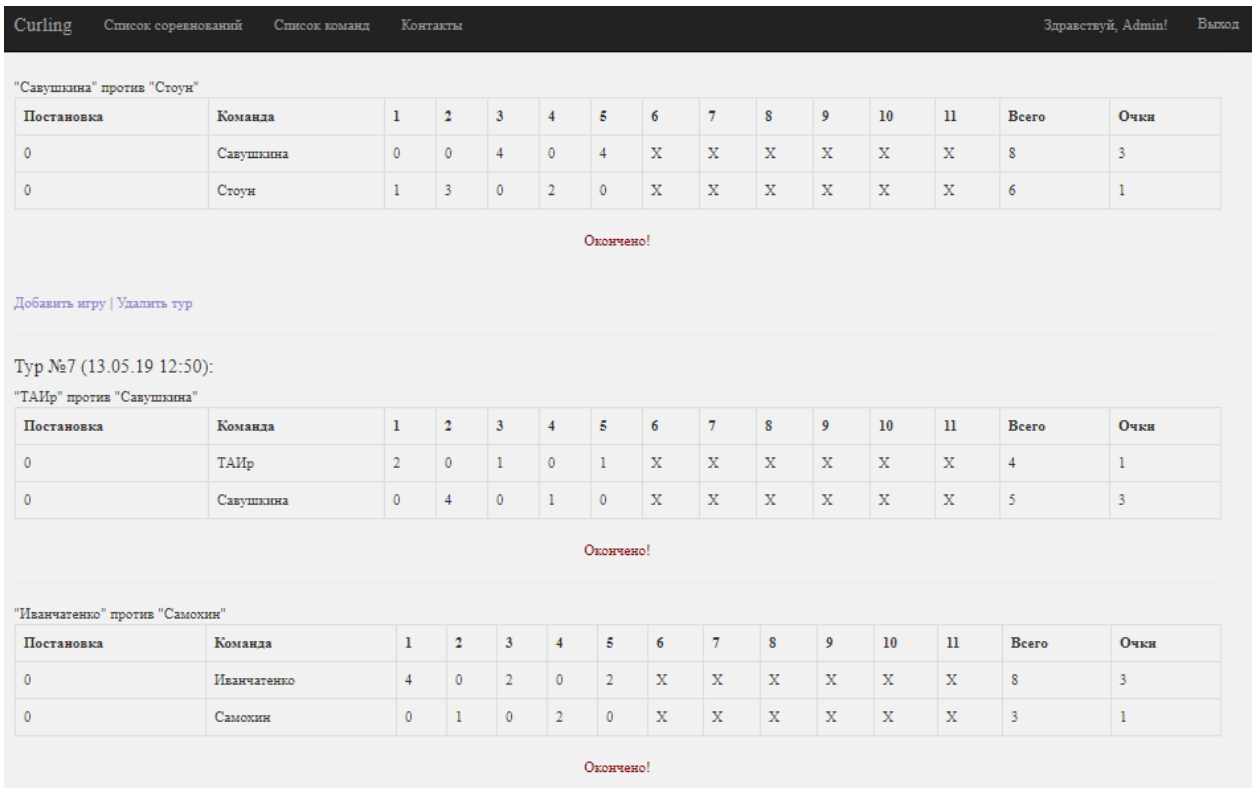


Рисунок 5.17 – Экранная форма хода турнира

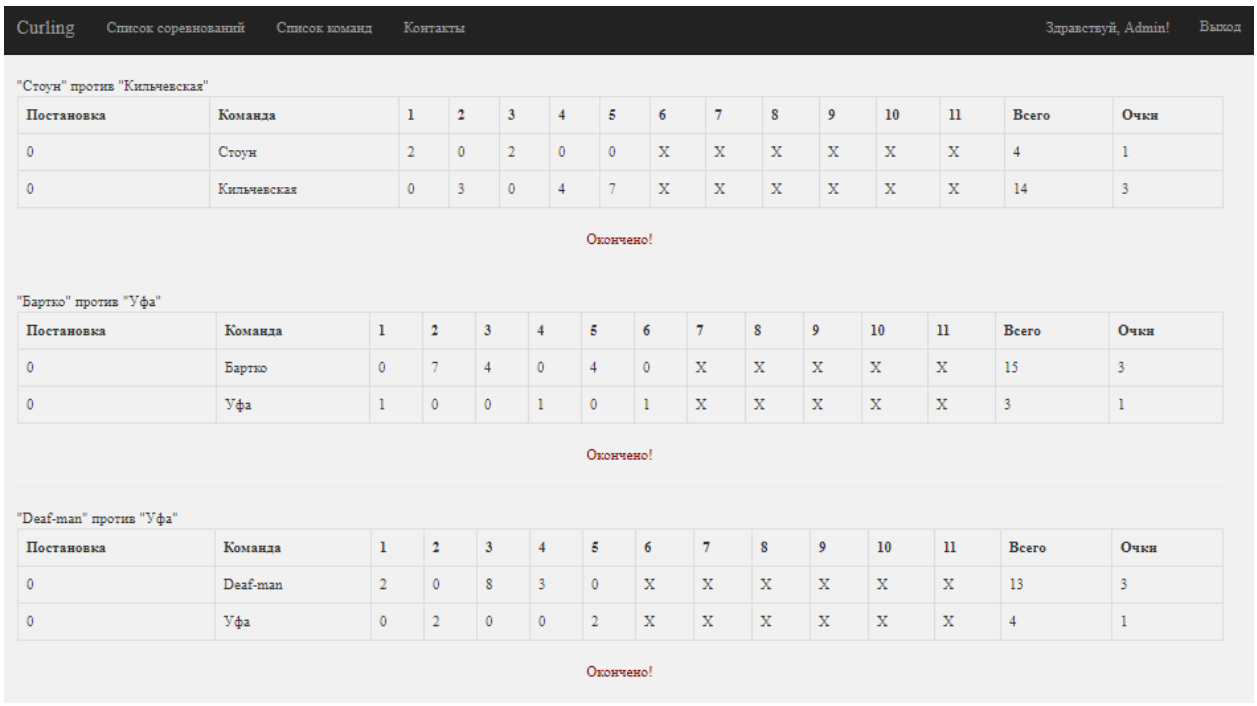


Рисунок 5.18 – Экранная форма хода турнира

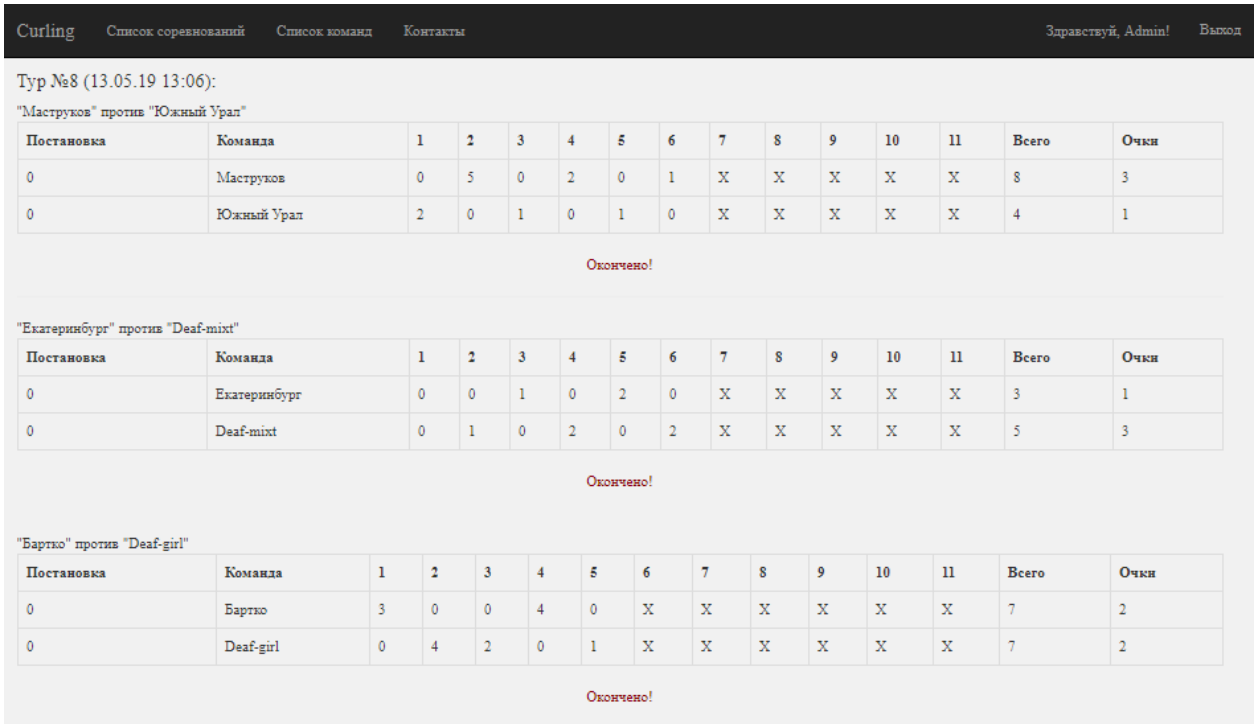


Рисунок 5.19 – Экранная форма хода турнира

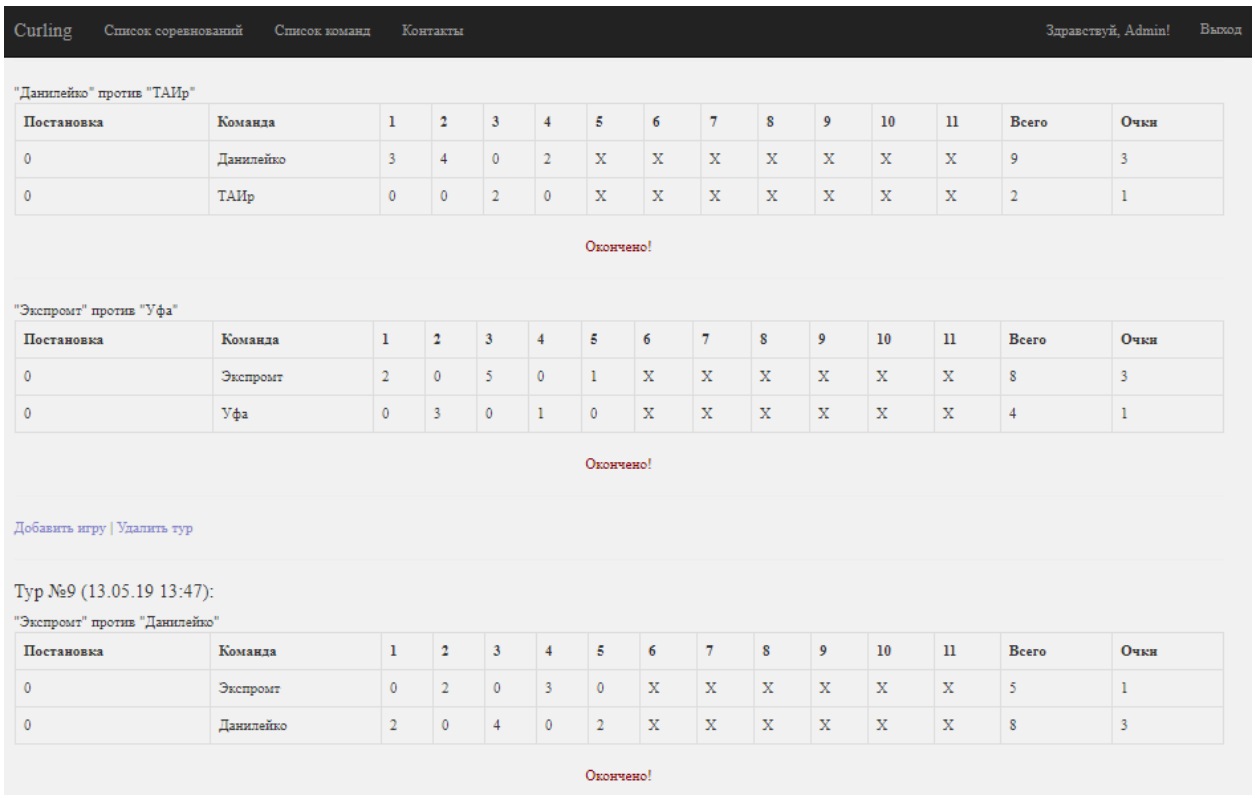


Рисунок 5.20 – Экранная форма хода турнира

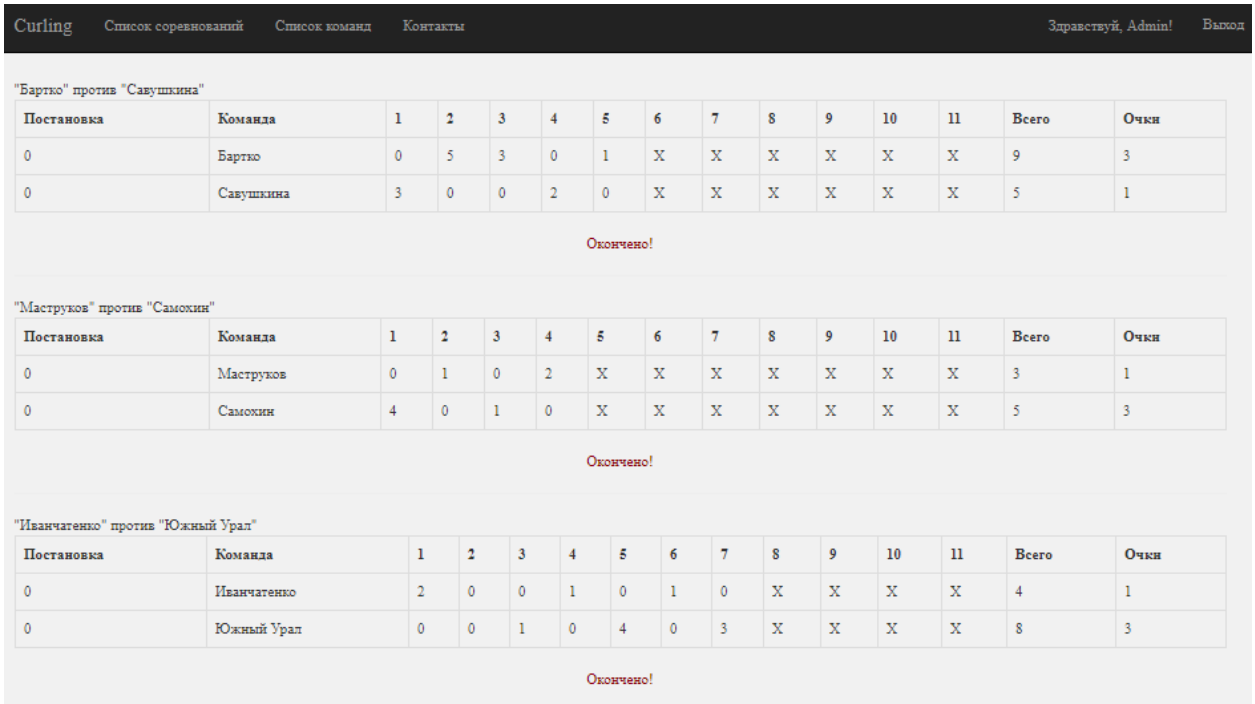


Рисунок 5.21 – Экранная форма хода турнира

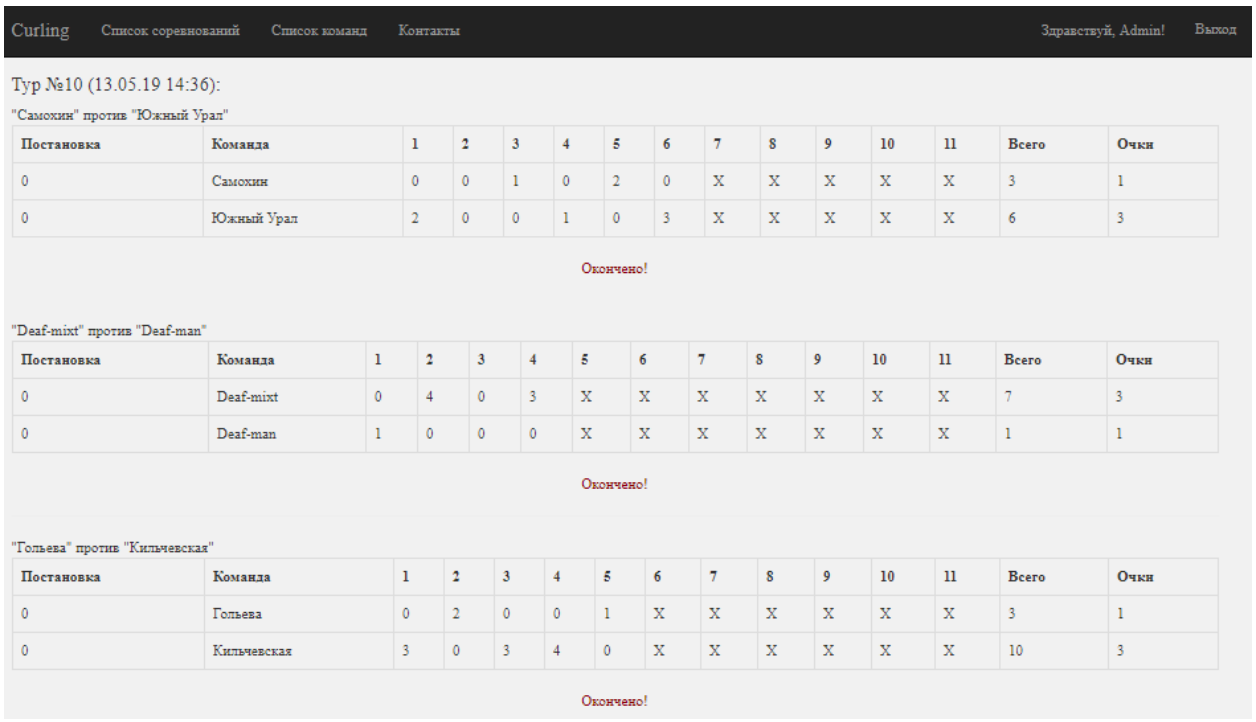


Рисунок 5.22 – Экранная форма хода турнира

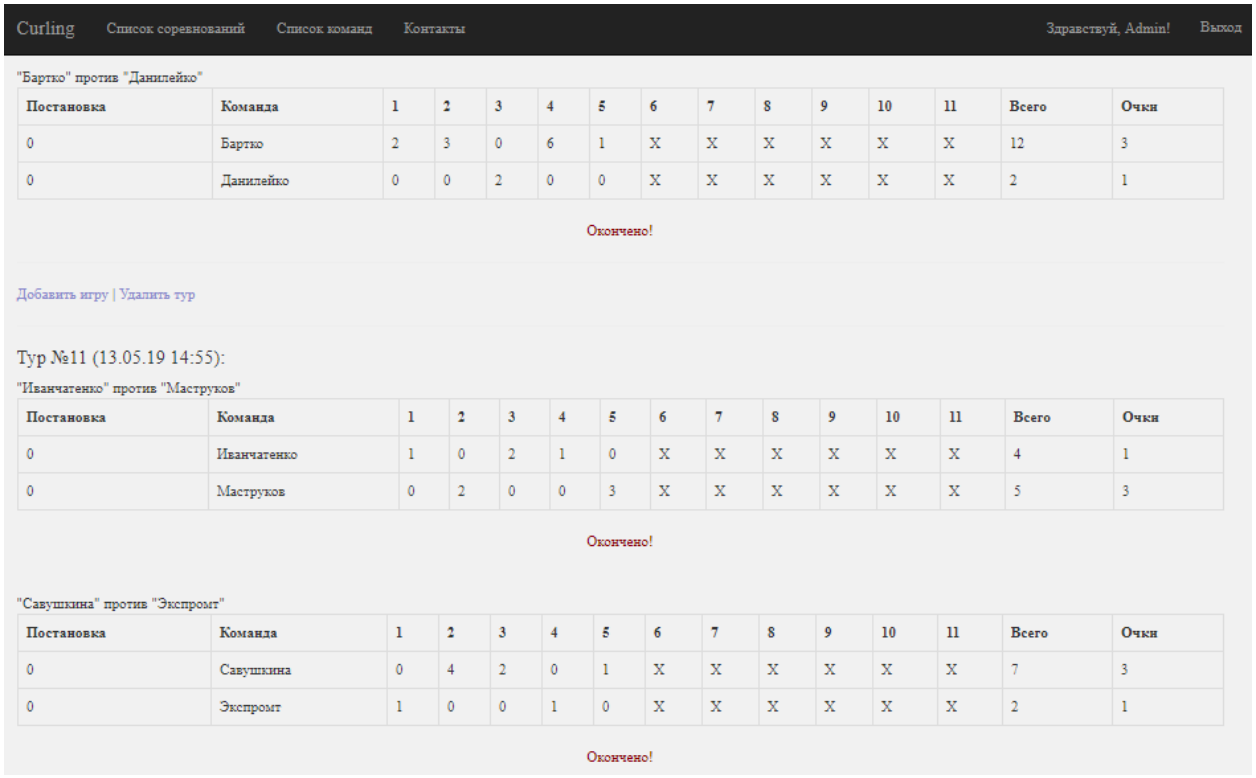
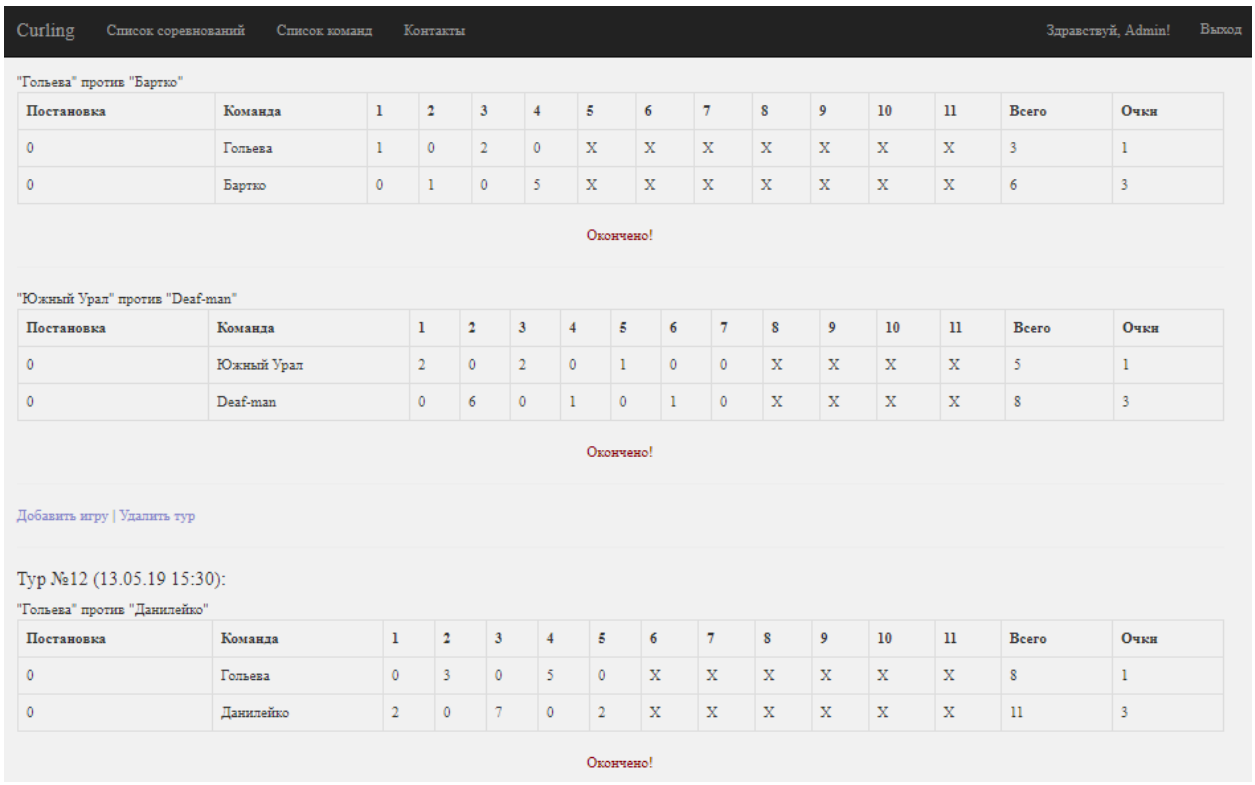


Рисунок 5.23 – Экранная форма хода турнира



Curling Список соревнований Список команд Контакты Здравствуй, Admin! Выход

"Гольева" против "Бартко"

Постановка	Команда	1	2	3	4	5	6	7	8	9	10	11	Всего	Очки
0	Гольева	1	0	2	0	X	X	X	X	X	X	X	3	1
0	Бартко	0	1	0	5	X	X	X	X	X	X	X	6	3

Окончено!

"Южный Урал" против "Deaf-man"

Постановка	Команда	1	2	3	4	5	6	7	8	9	10	11	Всего	Очки
0	Южный Урал	2	0	2	0	1	0	0	X	X	X	X	5	1
0	Deaf-man	0	6	0	1	0	1	0	X	X	X	X	8	3

Окончено!

Добавить игру | Удалить тур

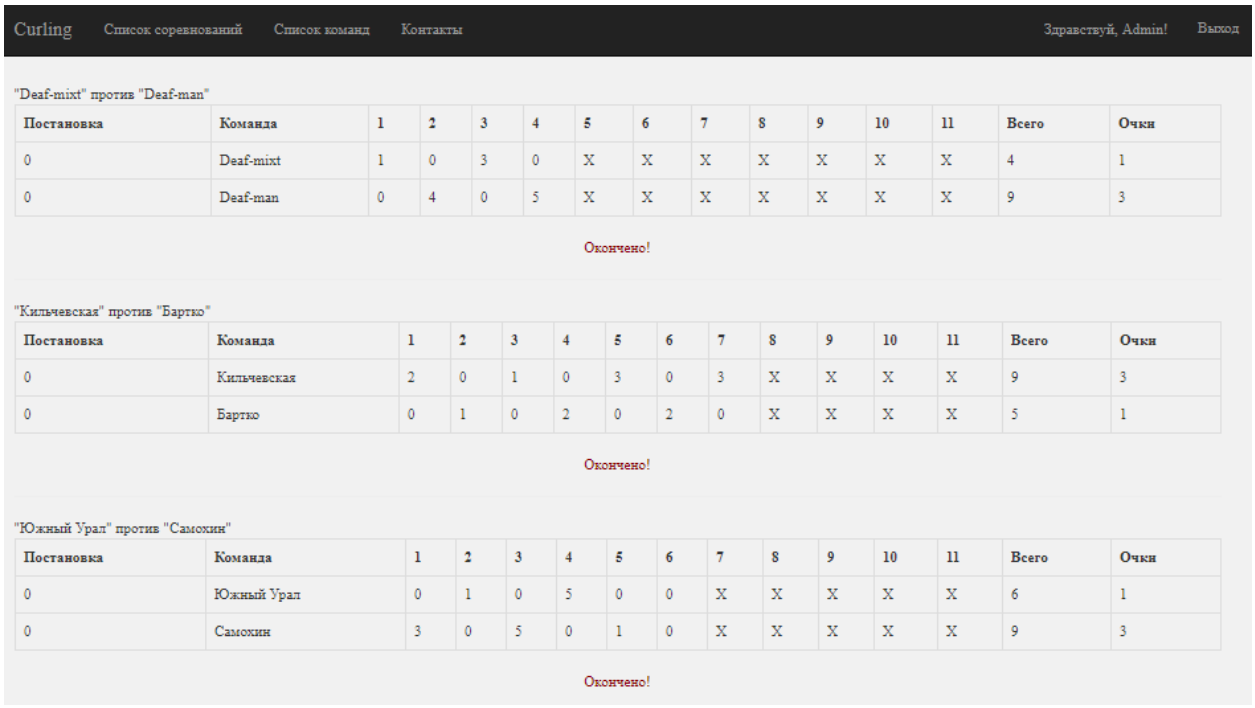
Тур №12 (13.05.19 15:30):

"Гольева" против "Данилейко"

Постановка	Команда	1	2	3	4	5	6	7	8	9	10	11	Всего	Очки
0	Гольева	0	3	0	5	0	X	X	X	X	X	X	8	1
0	Данилейко	2	0	7	0	2	X	X	X	X	X	X	11	3

Окончено!

Рисунок 5.24 – Экранная форма хода турнира



Curling Список соревнований Список команд Контакты Здравствуй, Admin! Выход

"Deaf-mixt" против "Deaf-man"

Постановка	Команда	1	2	3	4	5	6	7	8	9	10	11	Всего	Очки
0	Deaf-mixt	1	0	3	0	X	X	X	X	X	X	X	4	1
0	Deaf-man	0	4	0	5	X	X	X	X	X	X	X	9	3

Окончено!

"Кильчевская" против "Бартко"

Постановка	Команда	1	2	3	4	5	6	7	8	9	10	11	Всего	Очки
0	Кильчевская	2	0	1	0	3	0	3	X	X	X	X	9	3
0	Бартко	0	1	0	2	0	2	0	X	X	X	X	5	1

Окончено!

"Южный Урал" против "Самохин"

Постановка	Команда	1	2	3	4	5	6	7	8	9	10	11	Всего	Очки
0	Южный Урал	0	1	0	5	0	0	X	X	X	X	X	6	1
0	Самохин	3	0	5	0	1	0	X	X	X	X	X	9	3

Окончено!

Рисунок 5.25 – Экранная форма хода турнира

Турнирная таблица

Группа А

#	Команда	А1	А2	А3	А4	А5	Очки	DSC	Место
А1	Данилейко		10 : 5 3	0 : 11 1	5 : 7 1	9 : 2 3	8	0	3 (изменить)
А2	Савушкина	5 : 10 1		1 : 8 1	8 : 6 3	4 : 5 1	6	0	2 (изменить)
А3	Кильчевская	11 : 0 3	8 : 1 3		4 : 14 1	9 : 2 3	10	0	1 (изменить)
А4	Стоун	7 : 5 3	6 : 8 1	14 : 4 3		0 : 13 1	8	0	4 (изменить)
А5	ТАИр	2 : 9 1	5 : 4 3	2 : 9 1	13 : 0 3		8	0	5 (изменить)

Рисунок 5.26 – Экранная форма урнирной таблицы группы А

Группа Б

#	Команда	Б1	Б2	Б3	Б4	Б5	Очки	DSC	Место
Б1	Гольева		8 : 13 1	4 : 6 1	1 : 11 1	6 : 7 1	4	0	1 (изменить)
Б2	Бартко	13 : 8 3		2 : 7 1	15 : 3 3	7 : 7 2	9	0	3 (изменить)
Б3	Экспромт	6 : 4 3	7 : 2 3		8 : 4 3	2 : 8 1	10	0	2 (изменить)
Б4	Уфа	11 : 1 3	3 : 15 1	4 : 8 1		12 : 2 3	8	0	5 (изменить)
Б5	Deaf-girl	7 : 6 3	7 : 7 2	8 : 2 3	2 : 12 1		9	0	4 (изменить)

Рисунок 5.27 – Экранная форма турнирной таблицы группы Б

Группа С

#	Команда	С1	С2	С3	С4	С5	Очки	DSC	Место
С1	Иванчатенко		8 : 3 3	1 : 14 1	4 : 5 1	3 : 8 1	6	0	2 (изменить)
С2	Самохин	3 : 8 1		13 : 3 3	3 : 3 2	6 : 4 3	9	0	3 (изменить)
С3	УралИнвестМет	14 : 1 3	3 : 13 1		0 : 0 0	7 : 6 3	7	0	5 (изменить)
С4	Deaf-man	5 : 4 3	3 : 3 2	0 : 0 0		7 : 2 3	8	0	1 (изменить)
С5	Север (Казахстан)	8 : 3 3	4 : 6 1	6 : 7 1	2 : 7 1		6	0	4 (изменить)

Рисунок 5.28 – Экранная форма турнирной таблицы группы С

Группа Д

#	Команда	Д1	Д2	Д3	Д4	Очки	DSC	Место
Д1	Южный Урал		8 : 4 3	5 : 5 2	4 : 3 3	8	0	3 (изменить)
Д2	Маструков	4 : 8 1		2 : 8 1	2 : 11 1	3	0	2 (изменить)
Д3	Deaf-mixt	5 : 5 2	8 : 2 3		3 : 5 1	6	0	1 (изменить)
Д4	Екатеринбург	3 : 4 1	11 : 2 3	5 : 3 3		7	0	4 (изменить)

Рисунок 5.29 – – Экранная форма турнирной таблицы группы Д

Итоговый протокол

Место	Название команды	Состав команды	С/Р	Личные тренеры	Очки	Победы	DSC
1 (изменить)	Кильчевская	Диана Пронина Адель Буриева Властелина Бегашева Анастасия Кильчевская Владислав Дружкин	I КМС I КМС КМС	------- ------- ------- ------- -------	18	6	0
2 (изменить)	Deaf-man	Роман Кузьмин Никита Дружкин Юрий Макеев Олег Дарчиев	КМС I МСМК МСМК	------- ------- ------- -------	16	4	0
3 (изменить)	Иванчатенко	Владимир Алипкин Дмитрий Соломатин Никита Иванчатенко Константин Окружной	II I I I	------- ------- ------- -------	12	3	0
4 (изменить)	УралИнвестМет	Андрей Двойных Дмитрий Артемов Денис Тевс Сергей Ломакин	б/р б/р б/р б/р	Сергей Багович ------- ------- -------	3	0	0

Рисунок 5.30 – Экранная форма итогового протокола

Установить	Южный Урал	Сергей Нарудинов Артем Евдокимов Сергей Бородин Анна Щупко	б/р I б/р II	Сергей Багович ------- ------- -------	14	3	0
Установить	Самохин	Антон Мишин Антон Самохин Михаил Саморуков Никита Москвин	II II I II	Сергей Багович ------- ------- -------	16	4	0
Установить	Маструков	Виктор Алтухов Павел Мудрак Вячеслав Маструков	I III I	------- ------- -------	11	3	0
Установить	Deaf-mixt	Екатерина Аладьина Александр Пятков Анна Пунтусова Владислав Дружкин	КМС МСМК МСМК КМС	------- ------- ------- -------	14	4	0
Установить	Север (Казахстан)	Глеб Вибе Даулет Тарбак Роман Жежеря Денис Зернов Иван Буняшин Фархат Раев	б/р КМС КМС МС КМС б/р	------- ------- ------- ------- ------- -------	6	1	0

Рисунок 5.31 – Экранная форма итогового протокола

Установить	Екатеринбург	Валерий Мотов Валентина Власенко Алексей Демин Софья Ярутина	I II КМС КМС	------- ------- ------- -------	3	0	0
Установить	Гольева	Варвара Гольева Елизавета Свиридова Ольга Пономарева Дарья Филимонова Влада Бородина	б/р б/р б/р б/р б/р	------- ------- ------- ------- -------	13	3	0
Установить	Данилейко	Татьяна Данилейко Кристина Груздева Влада Дородная Ева Макарова	б/р б/р б/р б/р	Сергей Нарудинов ------- ------- -------	15	4	0
Установить	Савушкина	Вероника Коновалова Анна Щупко Полина Энес Кристина Савушкина Александра Доронько	КМС II II II КМС	------- ------- ------- ------- -------	14	4	0

Рисунок 5.32 – Экранная форма итогового протокола

Установить	Бартко	Дарья Тугай Александра Бартко Динара Байгнина Анна Шутова Елизавета Супереко	КМС КМС КМС КМС КМС	------- ------- ------- ------- -------	19	5	0
Установить	Стоун	Антонина Редреева Надежда Зяблова Любовь Озерова Светлана Коган Татьяна Щербань	б/р I I I II	Сергей Багович ------- ------- ------- -------	6	1	0
Установить	Экспромт	Яна Быковская Лариса Письменова Ирина Лисовец Надежда Кобылянец	I I I I	------- ------- ------- -------	12	3	0
Установить	ТАИр	Ирина Маркова Татьяна Фишер Анна Михеева Ирина Худяк	б/р б/р б/р б/р	------- ------- ------- -------	4	0	0
Установить	Уфа	Анна Морозова Екатерина Плешкова Элина Ханнанова Диана Нургалеева	III III III III	Сергей Багович ------- ------- -------	5	0	0

Рисунок 5.33 – Экранная форма итогового протокола

Установить	Deaf-girl	Валерия Камалетдинова Анастасия Мойстренко Наргис Хасанова Елизовета Парнева	КМС КМС КМС КМС	------- ------- ------- -------	7	1	0

Рисунок 5.34 – Экранная форма итогового протокола

ЗАКЛЮЧЕНИЕ

В ходе дипломного проектирования выполнено следующее:

1. Проведен анализ предметной области и существующих решений;

2. Разработано техническое задание;

3. Выбрана среда и средства реализации, разработаны основные архитектурные решения;

5. Разработана база данных;

6. Реализована пользовательская и серверная части приложения;

7. Протестировано разработанное программное обеспечение.

В веб-приложении были реализованы следующие функции:

1. Регистрация пользователей;

2. Создание соревнований;

3. Закрепление судей на соревнованиях;

4. Добавление команд на соревнования;

5. Формирование групп на турнирах;

6. Добавление туров и игр на соревнования;

7. Ведение счета онлайн и добавление постановочных бросков на игру;

8. Формирование турнирных таблиц и итогового протокола с автоматическим подсчетом постановочных бросков, очков и побед.

9. Места в турнирных таблицах выставляются вручную, в итоговом протоколе места так же выставляются вручную, но по мере добавления мест список упорядочивается по возрастанию итогового места.

В настоящее время веб-приложения готово к эксплуатации, но его необходимо доработать, в частности: доработать ролевую модель, добавить поиск и фильтрацию спортсменов/судей/тренеров/членов ФК/соревнований,.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1 MS SQL Server. – https://navicongroup.ru/platforms/4025/. Дата обращения: 03.04.2019.

2 Критерии для выбора СУБД. – http://citforum.ru/database/articles/criteria/. Дата обращения: 03.04.2019.

3 Сравнение СУБД – http://devacademy.ru/posts/sqlite-vs-mysql-vs-postgresql/. Дата обращения: 03.04.2019.

4 Маркин Е.И. Разработка web-приложения с использованием архитектуры «клиент–сервер» / Е.И. Маркин, К.М. Рябов., Е.А. Артюшина // Международный студенческий научный вестник. – 2016. – № 3-1. – http://www.eduherald.ru/ru/article/view?id=14732. Дата обращения: 13.04.2019.

5 Описание фреймворков для веб-разработки – https://itproger.com/news/178. Дата обращения: 17.04.2019.

6 SoftPeelR – https://www.softpeelr.com/. Дата обращения: 10.03.2019.

7 Сайт Федерации кёрлинга России – http://www.curling.ru/. Дата обращения: 10.03.2019.

8 Сайт Всемирной Федерации кёрлинга – http://www.worldcurling.org/. Дата обращения: 10.03.2019

9 Лицензии MS SQL Server – https://www.microsoft.com/ru-ru/sql-server/sql-server-2017-pricing. Дата обращения: 15.04.2019

10 .NET Core – http://tqm.com.ua/likbez/article/pochemu-net-ru. Дата обращения: 10.03.2019

11 Razor Pages – https://docs.microsoft.com/ru-ru/aspnet/core/razor-pages/?view=aspnetcore-2.2&tabs=visual-studio. Дата обращения: 1.04.2019.

12 EF core – https://docs.microsoft.com/ru-ru/ef/core/managing-schemas/migrations/. Дата обращения: 31.03.2019.

13 Сведения о .NET Core – https://docs.microsoft.com/ru-ru/dotnet/core/about. Дата обращения: 16.03.2019.

14 Архитектура клиент-сервер. – https://redcomrade.ru/materinskie-platy/izuchenie-setevogo-vzaimodeistviya-v-arhitekture-klient-server/. Дата обращения: 1.05.2019.

15 Классификация Мартина Фаулера – http://design-pattern.ru/patterns/. Дата обращения: 19.05.2019.

16 Патерн проектирования MVC – http://design-pattern.ru/patterns/mvc.html. Дата обращения: 19.05.2019.

17 Патерн проектирования PAC – http://ru.knowledgr.com. Дата обращения: 19.05.2019.

18 Патерн проектирования HMVC – https://ru.wikipedia.org/wiki/HMVC. Дата обращения: 19.05.2019.

19 Патерн проектирования MVP – https://ru.wikipedia.org/wiki/Model-View-Presenter. Дата обращения: 19.05.2019.

20 Патерн проектирования MVVM – https://metanit.com/sharp/wpf/22.1.php. Дата обращения: 19.05.2019.

21 Взаимодействие компонентов MVC – https://github.com/codedokode/pasta/blob/master/arch/mvc.md. Дата обращения: 19.05.2019.

22 Асинхронное программирование. – https://docs.microsoft.com/ru-ru/dotnet/csharp/async. Дата обращения 22.05.20019

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД КОНТРОЛЛЕРА ENDING_CONTROLLER

Листинг А.1 – Исходный код «EndingController.cs»

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.Rendering;
using Microsoft.EntityFrameworkCore;
using Curling.Models;

namespace Curling.Controllers
{
    public class EndingController : Controller
    {
        private readonly CurlingContext _context;
        public EndingController(CurlingContext context)
        {
            _context = context;
        }
        public async Task<IActionResult> Index()
        {
            return View(await _context.Endings.ToListAsync());
        }
        public async Task<IActionResult> Details(int? id)
        {
            if (id == null)
            {
                return NotFound();
            }
            var endingModel = await _context.Endings
                .FirstOrDefaultAsync(m => m.Id == id);
            if (endingModel == null)
            {
                return NotFound();
```

Продолжение листинга А.1

```
            }
            return View(endingModel);
        }
        public IActionResult Create(int? id)
        {
            if (id == null)
                return NotFound();
            var game = _context.Games.Find(id);
            if (game == null)
                return NotFound();
            var allEndingsForGame = _context.Endings.Where(model => model.Game ==
game);
            var endingModel = new EndingModel();
            endingModel.GameId = game.Id;
            endingModel.Game = game;
            endingModel.Position = allEndingsForGame.Count() + 1;
            ViewBag.returnUrl = Request.Headers["Referer"].ToString();
            return View(endingModel);
        }
        [HttpPost]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult>
Create([Bind("FirstTeamScore,SecondTeamScore,GameId,Position,IsFinal")] EndingModel
endingModel, string returnUrl)
        {
            endingModel.Id = 0;
            var gameModel = _context.Games.Find(endingModel.GameId);
            endingModel.Game = gameModel;
            if (endingModel.Position >= 11)
                endingModel.IsFinal = true;
            _context.Add(endingModel);
            await _context.SaveChangesAsync();
            return Redirect(returnUrl); //RedirectToAction(nameof(Index));
        }
        public async Task<IActionResult> Edit(int? id)
        {
```

Продолжение листинга А.1

```
            if (id == null)
            {
                return NotFound();
            }
            var endingModel = await _context.Endings.FindAsync(id);
            if (endingModel == null)
            {
                return NotFound();
            }
            return View(endingModel);
        }
        [HttpPost]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> Edit(int id,
[Bind("Id,FirstTeamScore,SecondTeamScore,Position,IsFinal")] EndingModel endingModel)
        {
            if (id != endingModel.Id)
            {
                return NotFound();
            }
            if (ModelState.IsValid)
            {
                try
                {
                    _context.Update(endingModel);
                    await _context.SaveChangesAsync();
                }
                catch (DbUpdateConcurrencyException)
                {
                    if (!EndingModelExists(endingModel.Id))
                    {
                        return NotFound();
                    }
                    else
                    {
```

Продолжение листинга А.1

```
                    throw;
                }
            }
            return RedirectToAction(nameof(Index));
        }
        return View(endingModel);
    }
    public async Task<IActionResult> Delete(int? id)
    {
        if (id == null)
        {
            return NotFound();
        }
        var endingModel = await _context.Endings
            .FirstOrDefaultAsync(m => m.Id == id);
        if (endingModel == null)
        {
            return NotFound();
        }
        return View(endingModel);
    }
    [HttpPost, ActionName("Delete")]
    [ValidateAntiForgeryToken]
    public async Task<IActionResult> DeleteConfirmed(int id)
    {
        var endingModel = await _context.Endings.FindAsync(id);
        _context.Endings.Remove(endingModel);
        await _context.SaveChangesAsync();
        return RedirectToAction(nameof(Index));
    }
    private bool EndingModelExists(int id)
    {
        return _context.Endings.Any(e => e.Id == id);
    }
}
```

ПРИЛОЖЕНИЕ Б
ИСХОДНЫЙ КОД КОНТРОЛЛЕРА GAME_CONTROLLER

Листинг Б.1 – Исходный код «GameController.cs»

```
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Curling.Data;
using Curling.Models;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.Rendering;
using Microsoft.EntityFrameworkCore;

namespace Curling.Controllers
{
    public class GameController : Controller
    {
        private readonly CurlingContext _context;
        public GameController(CurlingContext context)
        {
            _context = context;
            DbHelper.Context = context;
        }
        public async Task<IActionResult> Index()
        {
            return View(await _context.Games.ToListAsync());
        }
        public async Task<IActionResult> Details(int? id)
        {
            if (id == null)
            {
                return NotFound();
            }
            var gameModel = await _context.Games
                .FirstOrDefaultAsync(m => m.Id == id);
            if (gameModel == null)
            {
                return NotFound();
```

Продолжение листинга Б.1

```
            }
            return View(gameModel);
        }
        public async Task<IActionResult> Create(int? id)
        {
            if (id == null)
                return NotFound();
            var tour = _context.Tours.Find(id.Value);
            if (tour == null)
                return NotFound();
            var gameModel = new GameModel();
            gameModel.Staging = 0;
            gameModel.Tour = tour;
            gameModel.TourId = tour.Id;
            List<SelectListItem> teamList1 = new List<SelectListItem>();
            List<SelectListItem> teamList2 = new List<SelectListItem>();
            var tournament = _context.Tournaments.Find(tour.TournamentId);
            if (tournament == null)
                return NotFound();
            var suitableTeams = await DbHelper.GetTeamsForTournament(tournament);
            foreach (var team in suitableTeams)
            {
                teamList1.Add(new SelectListItem(team.Name, team.Id.ToString()));
                teamList2.Add(new SelectListItem(team.Name, team.Id.ToString()));
            }
            ViewData["TeamList1"] = teamList1;
            ViewData["TeamList2"] = teamList2;
            return View(gameModel);
        }
        [HttpPost]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult>
Create([Bind("FirstTeamId,SecondTeamId,Staging,FirstTeamStaging,SecondTeamStaging,TourId"
)] GameModel gameModel)
        {
            var firstTeam = _context.Teams.Find(gameModel.FirstTeamId);
```

Продолжение листинга Б.1

```
            var secondTeam = _context.Teams.Find(gameModel.SecondTeamId);
            var tour = _context.Tours.Find(gameModel.TourId);
            if (gameModel.FirstTeamId == gameModel.SecondTeamId)
                return NotFound();
            if (firstTeam == null || secondTeam == null || tour == null)
                return View(gameModel);
            gameModel.FirstTeam = firstTeam;
            gameModel.SecondTeam = secondTeam;
            gameModel.Tour = tour;
            _context.Add(gameModel);
            await _context.SaveChangesAsync();
            return RedirectToAction(nameof(Index));
        }
        public async Task<IActionResult> Edit(int? id)
        {
            if (id == null)
            {
                return NotFound();
            }
            var gameModel = await _context.Games.FindAsync(id);
            if (gameModel == null)
            {
                return NotFound();
            }
            return View(gameModel);
        }
        [HttpPost]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> Edit(int id,
[Bind("Id,FirstTeamId,SecondTeamId,Staging,FirstTeamStaging,SecondTeamStaging,TourId")]
GameModel gameModel)
        {
            if (id != gameModel.Id)
            {
                return NotFound();
            }
```

Продолжение листинга Б.1

```
            try
            {
                _context.Update(gameModel);
                await _context.SaveChangesAsync();
            }
            catch (DbUpdateConcurrencyException)
            {
                if (!GameModelExists(gameModel.Id))
                {
                    return NotFound();
                }
                else
                {
                    throw;
                }
            }
            return RedirectToAction(nameof(Index));

            return View(gameModel);
        }
        public async Task<IActionResult> Delete(int? id)
        {
            if (id == null)
            {
                return NotFound();
            }
            var gameModel = await _context.Games
                .FirstOrDefaultAsync(m => m.Id == id);
            if (gameModel == null)
            {
                return NotFound();
            }
            return View(gameModel);
        }
```

Продолжение листинга Б.1

```
        [HttpPost, ActionName("Delete")]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> DeleteConfirmed(int id)
        {
            var gameModel = await _context.Games.FindAsync(id);
            _context.Games.Remove(gameModel);
            await _context.SaveChangesAsync();
            return RedirectToAction(nameof(Index));
        }
        private bool GameModelExists(int id)
        {
            return _context.Games.Any(e => e.Id == id);
        }
    }
}
```

ПРИЛОЖЕНИЕ В
ИСХОДНЫЙ КОД КОНТРОЛЛЕРА GROUP_CONTROLLER

Листинг В.1 – Исходный код «GroupController.cs»

```
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Curling.Data;
using Curling.Models;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.Rendering;
using Microsoft.EntityFrameworkCore;

namespace Curling.Controllers
{
    public class GroupController : Controller
    {
        private readonly CurlingContext _context;
        public GroupController(CurlingContext context)
        {
            _context = context;
            DbHelper.Context = context;
        }
        public async Task<IActionResult> Index()
        {
            return View(await _context.Groups.ToListAsync());
        }
        public async Task<IActionResult> Details(int? id)
        {
            if (id == null)
            {
                return NotFound();
            }
            var groupModel = await _context.Groups
                .FirstOrDefaultAsync(m => m.Id == id);
            if (groupModel == null)
            {
                return NotFound();
```

Продолжение листинга В.1

```
            }
            return View(groupModel);
        }
        public async Task<IActionResult> Create(int? id)
        {
            if (id == null)
                return NotFound();
            var tournament = await _context.Tournaments.FindAsync(id.Value);
            if (tournament == null)
                return NotFound();
            var allTeams = await
DbHelper.GetNonGroupedTeamsForTournament(tournament);
            var group = new GroupModel();
            group.Tournament = tournament;
            group.TournamentId = tournament.Id;
            group.Input = new GroupModel.InputModel();
            group.Input.Teams = new List<SelectListItem>();
            group.Input.TournamentName = tournament.Name;
            foreach (var team in allTeams)
            {
                group.Input.Teams.Add(new SelectListItem(team.Name,
team.Id.ToString()));
            }
            ViewBag.returnUrl = Request.Headers["Referer"].ToString();
            return View(group);
        }
        [HttpPost]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult>
Create([Bind("Id,KeyCode,Tournament,TournamentId,SelectedIds")]
            GroupModel groupModel, string returnUrl)
        {
            groupModel.Tournament = await
_context.Tournaments.FindAsync(groupModel.TournamentId);
            if (!groupModel.SelectedIds.Any() || groupModel.Tournament == null)
                return View(groupModel);
            groupModel.Id = 0;
```

Продолжение листинга В.1

```
            _context.Add(groupModel);
            await _context.SaveChangesAsync();
            foreach (var id in groupModel.SelectedIds)
            {
                GroupAssignmentModel gam = new GroupAssignmentModel();
                gam.Id = 0;
                gam.Group = groupModel;
                gam.Team = _context.Teams.Find(int.Parse(id));
                _context.Add(gam);
            }
            await _context.SaveChangesAsync();

            return Redirect(returnUrl); //RedirectToAction(nameof(Index));
        }
        public async Task<IActionResult> Edit(int? id)
        {
            // TODO: Edit Groups
            if (id == null)
            {
                return NotFound();
            }
            var groupModel = await _context.Groups.FindAsync(id);
            if (groupModel == null)
            {
                return NotFound();
            }
            return View(groupModel);
        }
        public async Task<IActionResult> AddTeam(int? id)
        {
            if (id == null)
                return NotFound();
            var groupModel = await _context.Groups.FindAsync(id);
            if (groupModel == null)
            {
```

Продолжение листинга В.1

```
                    return NotFound();
                }
                var tournament = _context.Tournaments.Find(groupModel.TournamentId);


                var teams = await
DbHelper.GetNonGroupedTeamsForTournament(tournament);
                List<SelectListItem> items = new List<SelectListItem>();
                foreach (var teamModel in teams)
                {
                    items.Add(new SelectListItem(teamModel.Name,
teamModel.Id.ToString()));
                }
                ViewData["Teams"] = items;
                ViewBag.returnUrl = Request.Headers["Referer"].ToString();
                return View(groupModel);
            }
            [HttpPost]
            [ValidateAntiForgeryToken]
            public async Task<IActionResult> AddTeam(int id,
[Bind("Id,SelectedIds,TournamentId")] GroupModel groupModel, string returnUrl)
            {
                if (id != groupModel.Id)
                {
                    return NotFound();
                }
                if (groupModel.SelectedIds == null)
                    return NotFound();
                var grpModel = _context.Groups.Find(id);
                foreach (var ids in groupModel.SelectedIds)
                {
                    var grpAssignment = new GroupAssignmentModel();
                    grpAssignment.Id = 0;
                    grpAssignment.Group = grpModel;
                    grpAssignment.Team = _context.Teams.Find(int.Parse(ids));
                    _context.Add(grpAssignment);
                }
```

Продолжение листинга В.1

```
            await _context.SaveChangesAsync();
            return Redirect(returnUrl);
        }
        [HttpPost]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> Edit(int id, [Bind("Id,KeyCode")]
GroupModel groupModel)
        {
            if (id != groupModel.Id)
            {
                return NotFound();
            }
            if (ModelState.IsValid)
            {
                try
                {
                    _context.Update(groupModel);
                    await _context.SaveChangesAsync();
                }
                catch (DbUpdateConcurrencyException)
                {
                    if (!GroupModelExists(groupModel.Id))
                    {
                        return NotFound();
                    }
                    else
                    {
                        throw;
                    }
                }
                return RedirectToAction(nameof(Index));
            }
            return View(groupModel);
        }
        public async Task<IActionResult> Delete(int? id)
```

Продолжение листинга В.1

```
        {
            if (id == null)
            {
                return NotFound();
            }
            var groupModel = await _context.Groups
                .FirstOrDefaultAsync(m => m.Id == id);
            if (groupModel == null)
            {
                return NotFound();
            }
            return View(groupModel);
        }
        [HttpPost, ActionName("Delete")]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> DeleteConfirmed(int id)
        {
            var groupModel = await _context.Groups.FindAsync(id);
            _context.Groups.Remove(groupModel);
            await _context.SaveChangesAsync();
            return RedirectToAction(nameof(Index));
        }
        private bool GroupModelExists(int id)
        {
            return _context.Groups.Any(e => e.Id == id);
        }
    }
}
```

ПРИЛОЖЕНИЕ Г
ИСХОДНЫЙ КОД КОНТРОЛЛЕРА HOME_CONTROLLER

Листинг Г.1 – Исходный код « HomeController.cs»

```
namespace Curling.Controllers
{
    public class HomeController : Controller
    {
        public IActionResult Index()
        {
            return Redirect("/");
        }
        public IActionResult About()
        {
            ViewData["Message"] = "Описание сайта.";
            return View();
        }
        public IActionResult Contact()
        {
            ViewData["Message"] = "Контакты.";
            return View();
        }
        public IActionResult Privacy()
        {
            return View();
        }
        [ResponseCache(Duration = 0, Location = ResponseCacheLocation.None,
NoStore = true)]
        public IActionResult Error()
        {
            return View(new ErrorViewModel { RequestId = Activity.Current?.Id ??
HttpContext.TraceIdentifier });
        }
    }
}
```

ПРИЛОЖЕНИЕ Д
ИСХОДНЫЙ КОД КОНТРОЛЛЕРА SCORE_CONTROLLER

Листинг Д.1 – Исходный код «ScoreController.cs»

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Curling.Models;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.Rendering;
using Microsoft.EntityFrameworkCore;

namespace Curling.Controllers
{
    public class ScoreController : Controller
    {
        private readonly CurlingContext _context;
        public ScoreController(CurlingContext context)
        {
            _context = context;
        }
        public async Task<IActionResult> Index()
        {
            return View(await _context.Scores.ToListAsync());
        }
        public async Task<IActionResult> Details(int? id)
        {
            if (id == null)
            {
                return NotFound();
            }
            var scoreModel = await _context.Scores
                .FirstOrDefaultAsync(m => m.Id == id);
            if (scoreModel == null)
            {
                return NotFound();
            }
```

Продолжение листинга Д.1

```
            return View(scoreModel);
        }
        public async Task<IActionResult> Create(int? id)
        {
            if (id == null)
                return NotFound();
            var allTeamAssignments = await
_context.TeamTournamentAssignments.ToListAsync();
            var teamAssignment = allTeamAssignments.FirstOrDefault(t => t.Id ==
id);
            if (teamAssignment == null)
                return NotFound();
            var teams = await _context.Teams.ToListAsync();
            var team = teams.FirstOrDefault(t => t == teamAssignment.Team);
            if (team == null)
                return NotFound();
            var tours = await _context.Tournaments.ToListAsync();
            var tour = tours.FirstOrDefault(t => t == teamAssignment.Tournament);
            if (tour == null)
                return NotFound();
            var model = new ScoreModel();
            model.TeamTournamentAssignmentId = teamAssignment.Id;
            model.TeamModel = team;
            model.Tournament = tour;
            model.TeamTournamentAssignment = teamAssignment;
            model.TypeItems = new List<SelectListItem>()
            {
                new SelectListItem("Победа", "Win", true),
                new SelectListItem("Поражение", "Loss", false),
                new SelectListItem("Ничья", "Draw", false),
                new SelectListItem("Неявка", "None", false)
            };
            return View(model);
        }
        [HttpPost]
        [ValidateAntiForgeryToken]
```

Продолжение листинга Д.1

```
            public async Task<IActionResult>
Create([Bind("Id,Type,Staging,TeamTournamentAssignmentId")] ScoreModel scoreModel)
            {
                var tta =
_context.TeamTournamentAssignments.Find(scoreModel.TeamTournamentAssignmentId);
                if (tta == null)
                    return View(scoreModel);
                scoreModel.TeamTournamentAssignment = tta;
                _context.Add(scoreModel);
                await _context.SaveChangesAsync();
                return RedirectToAction(nameof(Index));
            }
            public async Task<IActionResult> Edit(int? id)
            {
                if (id == null)
                {
                    return NotFound();
                }

                var scoreModel = await _context.Scores.FindAsync(id);
                if (scoreModel == null)
                {
                    return NotFound();
                }
                return View(scoreModel);
            }
            [HttpPost]
            [ValidateAntiForgeryToken]
            public async Task<IActionResult> Edit(int id, [Bind("Id,Type,Staging")]
ScoreModel scoreModel)
            {
                if (id != scoreModel.Id)
                {
                    return NotFound();
                }
                if (ModelState.IsValid)
```

Продолжение листинга Д.1

```
        {
            try
            {
                _context.Update(scoreModel);
                await _context.SaveChangesAsync();
            }
            catch (DbUpdateConcurrencyException)
            {
                if (!ScoreModelExists(scoreModel.Id))
                {
                    return NotFound();
                }
                else
                {
                    throw;
                }
            }
            return RedirectToAction(nameof(Index));
        }
        return View(scoreModel);
    }
    public async Task<IActionResult> Delete(int? id)
    {
        if (id == null)
        {
            return NotFound();
        }
        var scoreModel = await _context.Scores
            .FirstOrDefaultAsync(m => m.Id == id);
        if (scoreModel == null)
        {
            return NotFound();
        }
        return View(scoreModel);
    }
```

Продолжение листинга Д.1

```
        [HttpPost, ActionName("Delete")]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> DeleteConfirmed(int id)
        {
            var scoreModel = await _context.Scores.FindAsync(id);
            _context.Scores.Remove(scoreModel);
            await _context.SaveChangesAsync();
            return RedirectToAction(nameof(Index));
        }
        private bool ScoreModelExists(int id)
        {
            return _context.Scores.Any(e => e.Id == id);
        }
    }
}
```

ПРИЛОЖЕНИЕ Е
ИСХОДНЫЙ КОД КОНТРОЛЛЕРА TEAM_CONTROLLER

Листинг Е.1 – Исходный код «TeamController.cs»

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Curling.Areas.Identity.Data;
using Curling.Data;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.Rendering;
using Microsoft.EntityFrameworkCore;
using Curling.Models;

namespace Curling.Controllers
{
    public class TeamController : Controller
    {
        private readonly CurlingContext _context;
        public TeamController(CurlingContext context)
        {
            _context = context;
            DbHelper.Context = _context;
        }
        public async Task<IActionResult> Index()
        {
            var teams = await _context.Teams.ToListAsync();
            foreach (var teamModel in teams)
            {
                teamModel.Input = new TeamModel.InputModel();
                var sportsmen = await DbHelper.GetUsersForTeam(teamModel);
                var coaches = new CurlingUser[sportsmen.Count];
                for (int i = 0; i < sportsmen.Count; i++)
                {
                    var coach = await DbHelper.GetCoachForSportsman(sportsmen[i]);

                    coaches[i] = coach;
```

Продолжение листинга Е.1

```
                    }
                    var kv = new KeyValuePair<List<CurlingUser>,
CurlingUser[]>(sportsmen, coaches);
                    teamModel.Input.TeamSportsmans = kv;
                    teamModel.Input.TournamentParticipation = await
DbHelper.GetTournamentsForTeam(teamModel);
                }
                return View(teams);
            }
            public async Task<IActionResult> Details(int? id)
            {
                if (id == null)
                {
                    return NotFound();
                }
                var teamModel = await _context.Teams
                    .FirstOrDefaultAsync(m => m.Id == id);
                if (teamModel == null)
                {
                    return NotFound();
                }

                teamModel.Input = new TeamModel.InputModel();
                var sportsmen = await DbHelper.GetUsersForTeam(teamModel);
                var coaches = new CurlingUser[sportsmen.Count];
                for (int i = 0; i < sportsmen.Count; i++)
                {
                    var coach = await DbHelper.GetCoachForSportsman(sportsmen[i]);
                    coaches[i] = coach;
                }
                var kv = new KeyValuePair<List<CurlingUser>, CurlingUser[]>(sportsmen,
coaches);
                teamModel.Input.TeamSportsmans = kv;
                teamModel.Input.TournamentParticipation = await
DbHelper.GetTournamentsForTeam(teamModel);

                return View(teamModel);
```

Продолжение листинга Е.1

```
        }
        private async Task<List<TournamentModel>> GetTournaments()
        {
            return await _context.Tournaments.ToListAsync();
        }
        private async Task<List<CurlingUser>> GetSportsmans()
        {
            var users = await _context.Users.ToListAsync();
            return users.Where(user => user.Role == "Sportsman").ToList();
        }
        private async Task<TournamentModel> GetTournamentById(int tournamentId)
        {
            return await _context.Tournaments.FindAsync(tournamentId);
        }
        public async Task<IActionResult> Create(int? id)
        {
            var model = new TeamModel();
            var tns = await GetTournaments();
            model.Input = new TeamModel.InputModel();
            model.Input.Tournaments = new List<SelectListItem>();
            var sms = await GetSportsmans();
            model.Input.Sportsmans = new List<SelectListItem>();
            model.SelectedIds = new string[1];
            foreach (var user in sms)
            {
                var item = new SelectListItem($"{user.FirstName} {user.LastName}",
user.Id);
                model.Input.Sportsmans.Add(item);
            }
            foreach (var t in tns)
            {
                var sil = new SelectListItem(
                    t.Name,
                    t.Id.ToString(),
                    id.HasValue && t.Id == id.Value,
```

Продолжение листинга Е.1

```
                        id.HasValue && t.Id != id.Value);
                    model.Input.Tournaments.Add(sil);
                }
                ViewBag.returnUrl = Request.Headers["Referer"].ToString();
                return View(model);
            }
            [HttpPost]
            [ValidateAntiForgeryToken]
            public async Task<IActionResult>
Create([Bind("Id,Name,TotalScore,TotalWins,TournamentId,SelectedIds")] TeamModel
teamModel, string returnUrl)
            {
                if (ModelState.IsValid)
                {
                    var allTeams = await _context.Teams.ToListAsync();
                    teamModel.Id = 0;
                    _context.Add(teamModel);
                    await _context.SaveChangesAsync();
                    var addedTeam = _context.Teams.ToList().FirstOrDefault(tm =>
tm.Name == teamModel.Name);
                    TeamTournamentAssignmentModel assignment = new
TeamTournamentAssignmentModel();
                    assignment.Id = 0;
                    assignment.Team = addedTeam;
                    assignment.Tournament = await
GetTournamentById(teamModel.TournamentId);
                    _context.Add(assignment);
                    await _context.SaveChangesAsync();
                    foreach (var id in teamModel.SelectedIds)
                    {
                        var user = _context.Users.Find(id);
                        if (user == null)
                            return NotFound();
                        var nn = new TeamAssignmentModel();
                        nn.Sportsman = user;
                        nn.Team = teamModel;
                        _context.Add(nn);
```

Продолжение листинга Е.1

```
                }
                await _context.SaveChangesAsync();
                return Redirect(returnUrl);
            }
            return View(teamModel);
        }
        public async Task<IActionResult> Edit(int? id)
        {
            if (id == null)
            {
                return NotFound();
            }
            var teamModel = await _context.Teams.FindAsync(id);
            if (teamModel == null)
            {
                return NotFound();
            }
            var tns = await GetTournaments();
            teamModel.Input = new TeamModel.InputModel();
            teamModel.Input.Tournaments = new List<SelectListItem>();
            var sms = await GetSportsmans();
            teamModel.Input.Sportsmans = new List<SelectListItem>();
            teamModel.SelectedIds = new string[1];
            foreach (var user in sms)
            {
                var item = new SelectListItem($"{user.FirstName} {user.LastName}",
user.Id, true);
                teamModel.Input.Sportsmans.Add(item);
            }
            foreach (var t in tns)
            {
                var sil = new SelectListItem(
                    t.Name,
                    t.Id.ToString(),
                    id.HasValue && t.Id == id.Value,
```

Продолжение листинга Е.1

```
                        id.HasValue && t.Id != id.Value);
                    teamModel.Input.Tournaments.Add(sil);
                }
                return View(teamModel);
            }
            [HttpPost]
            [ValidateAntiForgeryToken]
            public async Task<IActionResult> Edit(int id,
[Bind("Id,Name,TotalScore,TotalWins")] TeamModel teamModel)
            {
                if (id != teamModel.Id)
                {
                    return NotFound();
                }
                if (ModelState.IsValid)
                {
                    try
                    {
                        _context.Update(teamModel);
                        await _context.SaveChangesAsync();
                        var addedTeam = _context.Teams.ToList().FirstOrDefault(tm =>
tm.Name == teamModel.Name);
                    }
                    catch (DbUpdateConcurrencyException)
                    {
                        if (!TeamModelExists(teamModel.Id))
                        {
                            return NotFound();
                        }
                        else
                        {
                            throw;
                        }
                    }
                    return RedirectToAction(nameof(Index));
                }
```

Продолжение листинга Е.1

```
            return View(teamModel);
        }
        public async Task<IActionResult> Delete(int? id)
        {
            if (id == null)
            {
                return NotFound();
            }
            var teamModel = await _context.Teams
                .FirstOrDefaultAsync(m => m.Id == id);
            if (teamModel == null)
            {
                return NotFound();
            }
            teamModel.Input = new TeamModel.InputModel();
            teamModel.Input.SportsmanList = await
DbHelper.GetSportsmenForTeam(teamModel);
            return View(teamModel);
        }
        [HttpPost, ActionName("Delete")]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> DeleteConfirmed(int id)
        {
            var teamModel = await _context.Teams.FindAsync(id);
            _context.Teams.Remove(teamModel);
            await _context.SaveChangesAsync();
            return RedirectToAction(nameof(Index));
        }
        private bool TeamModelExists(int id)
        {
            return _context.Teams.Any(e => e.Id == id);
        }
    }
}
```

ПРИЛОЖЕНИЕ Ж
ИСХОДНЫЙ КОД КОНТРОЛЛЕРА TEAM_PLACEMENT_CONTROLLER

Листинг Ж.1 – Исходный код «TeamPlacementController.cs»

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.Rendering;
using Microsoft.EntityFrameworkCore;
using Curling.Models;

namespace Curling
{
    public class TeamPlacementController : Controller
    {
        private readonly CurlingContext _context;
        public TeamPlacementController(CurlingContext context)
        {
            _context = context;
        }
        public async Task<IActionResult> Index()
        {
            var curlingContext = _context.TeamPlacements.Include(t => t.Team);
            return View(await curlingContext.ToListAsync());
        }
        public async Task<IActionResult> Details(int? id)
        {
            if (id == null)
            {
                return NotFound();
            }
            var teamPlacement = await _context.TeamPlacements
                .Include(t => t.Team)
                .FirstOrDefaultAsync(m => m.Id == id);
            if (teamPlacement == null)
```

Продолжение листинга Ж.1

```
                {
                    return NotFound();
                }
                return View(teamPlacement);
            }
            public IActionResult Create(int? id, int? groupId)
            {
                if (!id.HasValue || !groupId.HasValue)
                    return NotFound();
                var groupAssignments = _context.GroupAssignments.Where(model =>
model.Group.Id == groupId).ToList(); // Or GroupId
                var teams = _context.Teams.Where(model =>
groupAssignments.Any(assignmentModel => assignmentModel.Team == model)).ToList();
                ViewData["TeamId"] = new SelectList(teams, "Id", "Name", id);
                ViewData["GroupId"] = new SelectList(_context.Groups, "Id", "KeyCode",
groupId);
                return View();
            }
            [HttpPost]
            [ValidateAntiForgeryToken]
            public async Task<IActionResult> Create([Bind("TeamId,GroupId,Placement")]
TeamPlacement teamPlacement)
            {
                if (ModelState.IsValid)
                {
                    _context.Add(teamPlacement);
                    await _context.SaveChangesAsync();
                    return RedirectToAction(nameof(Index));
                }
                ViewData["TeamId"] = new SelectList(_context.Teams, "Id", "Name",
teamPlacement.TeamId);
                return View(teamPlacement);
            }
            public async Task<IActionResult> Edit(int? id, int? groupId)
            {
                if (id == null || groupId == null)
                {
```

Продолжение листинга Ж.1

```
                return NotFound();
            }
            var teamPlacement = await _context.TeamPlacements.FindAsync(id);
            var group = await _context.Groups.FindAsync(groupId);
            if (teamPlacement == null || group == null)
            {
                return NotFound();
            }
            ViewData["TeamId"] = new SelectList(_context.Teams, "Id", "Name",
teamPlacement.TeamId);
            ViewData["GroupId"] = new SelectList(_context.Groups, "Id", "KeyCode",
groupId);
            return View(teamPlacement);
        }
        [HttpPost]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> Edit(int id,
[Bind("Id,TeamId,GroupId,Placement")] TeamPlacement teamPlacement)
        {
            if (id != teamPlacement.Id)
            {
                return NotFound();
            }
            if (ModelState.IsValid)
            {
                try
                {
                    _context.Update(teamPlacement);
                    await _context.SaveChangesAsync();
                }
                catch (DbUpdateConcurrencyException)
                {
                    if (!TeamPlacementExists(teamPlacement.Id))
                    {
                        return NotFound();
                    }
```

Продолжение листинга Ж.1

```
                    else
                    {
                        throw;
                    }
                }
                return RedirectToAction(nameof(Index));
            }
            ViewData["TeamId"] = new SelectList(_context.Teams, "Id", "Name",
teamPlacement.TeamId);
            return View(teamPlacement);
        }
        public async Task<IActionResult> Delete(int? id)
        {
            if (id == null)
            {
                return NotFound();
            }
            var teamPlacement = await _context.TeamPlacements
                .Include(t => t.Team)
                .FirstOrDefaultAsync(m => m.Id == id);
            if (teamPlacement == null)
            {
                return NotFound();
            }
            return View(teamPlacement);
        }
        [HttpPost, ActionName("Delete")]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> DeleteConfirmed(int id)
        {
            var teamPlacement = await _context.TeamPlacements.FindAsync(id);
            _context.TeamPlacements.Remove(teamPlacement);
            await _context.SaveChangesAsync();
            return RedirectToAction(nameof(Index));
        }
```

Продолжение листинга Ж.1

```
        private bool TeamPlacementExists(int id)
        {
            return _context.TeamPlacements.Any(e => e.Id == id);
        }
    }
}
```

ПРИЛОЖЕНИЕ К
ИСХОДНЫЙ КОД КОНТРОЛЛЕРА TEAM_PLACEMENT_GLOBAL_CONTROLLER

Листинг К.1 – Исходный код «TeamPlacementGlobalController.cs»

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.Rendering;
using Microsoft.EntityFrameworkCore;
using Curling.Models;

namespace Curling
{
    public class TeamPlacementGlobalController : Controller
    {
        private readonly CurlingContext _context;
        public TeamPlacementGlobalController(CurlingContext context)
        {
            _context = context;
        }
        public async Task<IActionResult> Index()
        {
            var curlingContext = _context.TeamPlacementGlobals.Include(t =>
t.Team);
            return View(await curlingContext.ToListAsync());
        }
        public async Task<IActionResult> Details(int? id)
        {
            if (id == null)
            {
                return NotFound();
            }
            var teamPlacementGlobal = await _context.TeamPlacementGlobals
                .Include(t => t.Team)
                .FirstOrDefaultAsync(m => m.Id == id);
```

Продолжение листинга К.1

```
            if (teamPlacementGlobal == null)
            {
                return NotFound();
            }
            return View(teamPlacementGlobal);
        }
        public IActionResult Create(int? id)
        {
            ViewData["TeamId"] = new SelectList(_context.Teams, "Id", "Name", id);
            return View();
        }
        [HttpPost]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> Create([Bind("TeamId,Placement")]
TeamPlacementGlobal teamPlacementGlobal)
        {
            if (ModelState.IsValid)
            {
                _context.Add(teamPlacementGlobal);
                await _context.SaveChangesAsync();
                return RedirectToAction(nameof(Index));
            }
            ViewData["TeamId"] = new SelectList(_context.Teams, "Id", "Name",
teamPlacementGlobal.TeamId);
            return View(teamPlacementGlobal);
        }
        public async Task<IActionResult> Edit(int? id)
        {
            if (id == null)
            {
                return NotFound();
            }
            var teamPlacementGlobal = await
_context.TeamPlacementGlobals.FindAsync(id);
            if (teamPlacementGlobal == null)
            {
```

Продолжение листинга К.1

```
                    return NotFound();
                }
                ViewData["TeamId"] = new SelectList(_context.Teams, "Id", "Name", id);
                return View(teamPlacementGlobal);
            }
            [HttpPost]
            [ValidateAntiForgeryToken]
            public async Task<IActionResult> Edit(int id,
[Bind("Id,TeamId,Placement")] TeamPlacementGlobal teamPlacementGlobal)
            {
                if (id != teamPlacementGlobal.Id)
                {
                    return NotFound();
                }
                if (ModelState.IsValid)
                {
                    try
                    {
                        _context.Update(teamPlacementGlobal);
                        await _context.SaveChangesAsync();
                    }
                    catch (DbUpdateConcurrencyException)
                    {
                        if (!TeamPlacementGlobalExists(teamPlacementGlobal.Id))
                        {
                            return NotFound();
                        }
                        else
                        {
                            throw;
                        }
                    }
                    return RedirectToAction(nameof(Index));
                }
                ViewData["TeamId"] = new SelectList(_context.Teams, "Id", "Name",
teamPlacementGlobal.TeamId);
```

Продолжение листинга К.1

```
            return View(teamPlacementGlobal);
        }
        public async Task<IActionResult> Delete(int? id)
        {
            if (id == null)
            {
                return NotFound();
            }
            var teamPlacementGlobal = await _context.TeamPlacementGlobals
                .Include(t => t.Team)
                .FirstOrDefaultAsync(m => m.Id == id);
            if (teamPlacementGlobal == null)
            {
                return NotFound();
            }
            return View(teamPlacementGlobal);
        }
        [HttpPost, ActionName("Delete")]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> DeleteConfirmed(int id)
        {
            var teamPlacementGlobal = await
_context.TeamPlacementGlobals.FindAsync(id);
            _context.TeamPlacementGlobals.Remove(teamPlacementGlobal);
            await _context.SaveChangesAsync();
            return RedirectToAction(nameof(Index));
        }
        private bool TeamPlacementGlobalExists(int id)
        {
            return _context.TeamPlacementGlobals.Any(e => e.Id == id);
        }
    }
}
```

ПРИЛОЖЕНИЕ Л
ИСХОДНЫЙ КОД КОНТРОЛЛЕРА TOUR_CONTROLLER

Листинг Л.1 – Исходный код «TourController.cs»

```
using System;
using System.Linq;
using System.Threading.Tasks;
using Curling.Models;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;

namespace Curling.Controllers
{
    public class TourController : Controller
    {
        private readonly CurlingContext _context;
        public TourController(CurlingContext context)
        {
            _context = context;
        }
        public async Task<IActionResult> Index()
        {
            return View(await _context.Tours.ToListAsync());
        }
        public async Task<IActionResult> Details(int? id)
        {
            if (id == null)
            {
                return NotFound();
            }
            var tourModel = await _context.Tours
                .FirstOrDefaultAsync(m => m.Id == id);
            if (tourModel == null)
            {
                return NotFound();
            }
            return View(tourModel);
        }
```

Продолжение листинга Л.1

```
        public async Task<IActionResult> Create(int? id)
        {
            if (id == null)
                return NotFound();
            var tournament = await _context.Tournaments.FindAsync(id);
            if (tournament == null)
                return NotFound();
            ViewData["TournamentId"] = id.Value;
            ViewData["Tournament"] = tournament;
            var tm = new TourModel();
            tm.DateStart = DateTime.Now;
            tm.Tournament = tournament;
            tm.TournamentId = tournament.Id;
            return View(tm);
        }
        [HttpPost]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult>
Create([Bind("Id,DateStart,TournamentId")] TourModel tourModel)
        {
            tourModel.Id = 0;
            var tournament = _context.Tournaments.Find(tourModel.TournamentId);
            tourModel.Tournament = tournament;
            _context.Add(tourModel);
            await _context.SaveChangesAsync();
            return RedirectToAction(nameof(Index));
            return View(tourModel);
        }
        public async Task<IActionResult> Edit(int? id)
        {
            if (id == null)
            {
                return NotFound();
            }
```

Продолжение листинга Л.1

```
            var tourModel = await _context.Tours.FindAsync(id);
            if (tourModel == null)
            {
                return NotFound();
            }
            return View(tourModel);
        }
        [HttpPost]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> Edit(int id, [Bind("Id,DateStart")]
TourModel tourModel)
        {
            if (id != tourModel.Id)
            {
                return NotFound();
            }
            if (ModelState.IsValid)
            {
                try
                {
                    _context.Update(tourModel);
                    await _context.SaveChangesAsync();
                }
                catch (DbUpdateConcurrencyException)
                {
                    if (!TourModelExists(tourModel.Id))
                    {
                        return NotFound();
                    }
                    else
                    {
                        throw;
                    }
                }
                return RedirectToAction(nameof(Index));
```

Продолжение листинга Л.1

```
            }
            return View(tourModel);
        }
        public async Task<IActionResult> Delete(int? id)
        {
            if (id == null)
            {
                return NotFound();
            }
            var tourModel = await _context.Tours
                .FirstOrDefaultAsync(m => m.Id == id);
            if (tourModel == null)
            {
                return NotFound();
            }
            return View(tourModel);
        }
        [HttpPost, ActionName("Delete")]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> DeleteConfirmed(int id)
        {
            var tourModel = await _context.Tours.FindAsync(id);
            _context.Tours.Remove(tourModel);
            await _context.SaveChangesAsync();
            return RedirectToAction(nameof(Index));
        }
        private bool TourModelExists(int id)
        {
            return _context.Tours.Any(e => e.Id == id);
        }
    }
}
```

ПРИЛОЖЕНИЕ М
ИСХОДНЫЙ КОД КОНТРОЛЛЕРА TOURNAMENT_CONTROLLER

Листинг М.1 – Исходный код «TournamentController.cs»

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Linq.Expressions;
using System.Threading.Tasks;
using Curling.Areas.Identity.Data;
using Curling.Data;
using Curling.Models;
using Curling.Models.Helpers;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Hosting.Internal;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.Rendering;
using Microsoft.EntityFrameworkCore;

namespace Curling.Controllers
{
    public class TournamentController : Controller
    {
        private readonly CurlingContext _context;
        public TournamentController(CurlingContext context)
        {
            _context = context;
            DbHelper.Context = _context;
        }
        public async Task<IActionResult> Index()
        {
            var list = await _context.Tournaments.ToListAsync();
            foreach (var item in list)
            {
                UpdateStatus(item);
                item.Input = new TournamentModel.InputModel();
                item.Input.JudgeUsers = await
DbHelper.GetJudgesForTournament(item);
```

Продолжение листинга М.1

```
            }
            return View(list);
        }
        public async Task<IActionResult> Details(int? id)
        {
            if (id == null)
            {
                return NotFound();
            }
            var tournamentModel = await _context.Tournaments
                .FirstOrDefaultAsync(m => m.Id == id);
            if (tournamentModel == null)
            {
                return NotFound();
            }
            UpdateStatus(tournamentModel);
            tournamentModel.Input = new TournamentModel.InputModel();
            tournamentModel.Input.JudgeUsers = await
DbHelper.GetJudgesForTournament(tournamentModel);
            return View(tournamentModel);
        }
        [Authorize(Roles = "Admin, Manager")]
        public async Task<IActionResult> Create()
        {
            var model = new TournamentModel
            {
                DateStart = DateTime.Now,
                DateEnd = DateTime.Now,
                Input = new TournamentModel.InputModel
                {
                    Judges = await DbHelper.GetJudges(),
                    Teams = new List<TeamModel>(),
                    SelectedIds = new List<SelectListItem>()
                }
            };
```

Продолжение листинга М.1

```
            return View(model);
        }
        [HttpPost]
        [ValidateAntiForgeryToken]
        [Authorize(Roles = "Admin")]
        public async Task<IActionResult>
Create([Bind("Id,Name,City,Stadium,DateStart,DateEnd,ShortInfo,LongInfo,SelectedIds")]
TournamentModel tournamentModel)
        {
            if (ModelState.IsValid)
            {
                _context.Add(tournamentModel);
                if (tournamentModel.DateStart >= tournamentModel.DateEnd)
                    return View(tournamentModel);
                UpdateStatus(tournamentModel);
                var toAdd = new List<JudgeModel>();
                var allUsers = await _context.Users.ToListAsync();
                var users = allUsers.Where(user =>
tournamentModel.SelectedIds.Contains(user.Id));
                foreach (var user in users)
                {
                    var assignment = new JudgeModel();
                    assignment.Judge = user;
                    assignment.Tournament = tournamentModel;
                    toAdd.Add(assignment);
                }
                await _context.SaveChangesAsync();
                foreach (var judgeModel in toAdd)
                {
                    _context.Add(judgeModel);
                }
                await _context.SaveChangesAsync();
                return RedirectToAction(nameof(Index));
            }
            return View(tournamentModel);
        }
```

Продолжение листинга М.1

```
        [Authorize(Roles = "Admin, Manager")]
        public async Task<IActionResult> Edit(int? id)
        {
            if (id == null)
            {
                return NotFound();
            }
            var tournamentModel = await _context.Tournaments.FindAsync(id);
            if (tournamentModel == null)
            {
                return NotFound();
            }
            UpdateStatus(tournamentModel);
            return View(tournamentModel);
        }
        [HttpPost]
        [ValidateAntiForgeryToken]
        [Authorize(Roles = "Admin, Manager")]
        public async Task<IActionResult> Edit(int id,
[Bind("Id,Name,City,Stadium,DateStart,DateEnd,ShortInfo,LongInfo")] TournamentModel
tournamentModel)
        {
            if (id != tournamentModel.Id)
            {
                return NotFound();
            }
            if (ModelState.IsValid)
            {
                try
                {
                    UpdateStatus(tournamentModel);
                    _context.Update(tournamentModel);
                    await _context.SaveChangesAsync();
                }
                catch (DbUpdateConcurrencyException)
                {
```

Продолжение листинга М.1

```
                if (!TournamentModelExists(tournamentModel.Id))
                {
                    return NotFound();
                }
                else
                {
                    throw;
                }
            }
            return RedirectToAction(nameof(Index));
        }
        return View(tournamentModel);
    }
    [Authorize(Roles = "Admin, Manager")]
    public async Task<IActionResult> Delete(int? id)
    {
        if (id == null)
        {
            return NotFound();
        }
        var tournamentModel = await _context.Tournaments
            .FirstOrDefaultAsync(m => m.Id == id);
        if (tournamentModel == null)
        {
            return NotFound();
        }
        return View(tournamentModel);
    }
    [HttpPost, ActionName("Delete")]
    [ValidateAntiForgeryToken]
    [Authorize(Roles = "Admin, Manager")]
    public async Task<IActionResult> DeleteConfirmed(int id)
    {
        var tournamentModel = await _context.Tournaments.FindAsync(id);
        _context.Tournaments.Remove(tournamentModel);
```

Продолжение листинга М.1

```
            await _context.SaveChangesAsync();
            return RedirectToAction(nameof(Index));
        }
        public async Task<IActionResult> TeamList(int? id)
        {
            if (id == null)
            {
                return NotFound();
            }
            var tournamentModel = await _context.Tournaments
                .FirstOrDefaultAsync(m => m.Id == id);
            if (tournamentModel == null)
            {
                return NotFound();
            }
            UpdateStatus(tournamentModel);
            tournamentModel.Input = new TournamentModel.InputModel();
            tournamentModel.SelectedIds = new string[1];
            tournamentModel.Input.JudgeUsers = await
DbHelper.GetJudgesForTournament(tournamentModel);
            var teams = await
DbHelper.GetNonGroupedTeamsForTournament(tournamentModel);
            tournamentModel.Input.Teams = teams;
            tournamentModel.Input.TeamSportsmans = new Dictionary<TeamModel,
KeyValuePair<List<CurlingUser>, CurlingUser[]>>();
            ViewData["Data"] = await
DbHelper.GetTournamentSportsmanDataForTournament(tournamentModel);
            foreach (var teamModel in teams)
            {
                var sportsmen = await DbHelper.GetUsersForTeam(teamModel);
                var coaches = new CurlingUser[sportsmen.Count];
                for (int i = 0; i < sportsmen.Count; i++)
                {
                    var coach = await DbHelper.GetCoachForSportsman(sportsmen[i]);
                    coaches[i] = coach;
                }
```

Продолжение листинга М.1

```
                    var kv = new KeyValuePair<List<CurlingUser>,
CurlingUser[]>(sportsmen, coaches);
                    tournamentModel.Input.TeamSportsmans[teamModel] = kv;
                }
                return View(tournamentModel);
            }
            public async Task<IActionResult> Standings(int? id)
            {
                if (id == null)
                {
                    return NotFound();
                }
                var tournamentModel = await _context.Tournaments
                    .FirstOrDefaultAsync(m => m.Id == id);
                if (tournamentModel == null)
                {
                    return NotFound();
                }
                UpdateStatus(tournamentModel);
                ViewData["Data"] = await
DbHelper.GetTournamentSportsmanDataForTournament(tournamentModel);
                return View(tournamentModel);
            }
            public async Task<IActionResult> TournamentProgress(int? id)
            {
                if (id == null)
                {
                    return NotFound();
                }
                var tournamentModel = await _context.Tournaments
                    .FirstOrDefaultAsync(m => m.Id == id);
                if (tournamentModel == null)
                {
                    return NotFound();
                }
                UpdateStatus(tournamentModel);
```

Продолжение листинга М.1

```
                    Dictionary<TourModel, List<KeyValuePair<GameModel, EndingModel[]>>>
tourGamesDictionary =
                        new Dictionary<TourModel, List<KeyValuePair<GameModel,
EndingModel[]>>>();
                    var tours = await DbHelper.GetToursForTournament(tournamentModel);
                    foreach (var tourModel in tours)
                    {
                        var games = await DbHelper.GetGamesForTour(tourModel);
                        var list = new List<KeyValuePair<GameModel, EndingModel[]>>();
                        foreach (var gameModel in games)
                        {
                            if (gameModel.FirstTeamId == 0 || gameModel.SecondTeamId == 0)
                                throw new ArgumentNullException("Error, Team IDs are
null!");
                            gameModel.FirstTeam =
_context.Teams.Find(gameModel.FirstTeamId);
                            gameModel.SecondTeam =
_context.Teams.Find(gameModel.SecondTeamId);
                            if (gameModel.FirstTeam == null || gameModel.SecondTeam ==
null)
                            {
                                gameModel.FirstTeam = _context.Teams.SingleOrDefault(model
=> model.Id == gameModel.FirstTeamId);
                                gameModel.SecondTeam =
_context.Teams.SingleOrDefault(model => model.Id == gameModel.SecondTeamId);
                            }
                            List<EndingModel> endingsForGame = await
DbHelper.GetEndingsForGame(gameModel);
                            var kv = new KeyValuePair<GameModel, EndingModel[]>(gameModel,
endingsForGame.ToArray());
                            list.Add(kv);
                        }

                        tourGamesDictionary[tourModel] = list;
                    }
                    tournamentModel.Input = new TournamentModel.InputModel();
                    tournamentModel.Input.Ending = new EndingModel();
                    ViewData["TourGames"] = tourGamesDictionary;
                    return View(tournamentModel);
```

Продолжение листинга М.1

```
        }
        public async Task<IActionResult> Schedule(int? id)
        {
            if (id == null)
            {
                return NotFound();
            }
            var tournamentModel = await _context.Tournaments
                .FirstOrDefaultAsync(m => m.Id == id);
            if (tournamentModel == null)
            {
                return NotFound();
            }
            Dictionary<TourModel, List<GameModel>> dict = new
Dictionary<TourModel, List<GameModel>>();
            var tours = _context.Tours.Where(model => model.Tournament ==
tournamentModel).ToList();
            foreach (var tourModel in tours)
            {
                var games = _context.Games.Where(model => model.Tour ==
tourModel).ToList();
                foreach (var gameModel in games)
                {
                    if (gameModel.FirstTeamId == 0 || gameModel.SecondTeamId == 0)
                        throw new ArgumentNullException("Error, Team IDs are
null!");
                    gameModel.FirstTeam =
_context.Teams.Find(gameModel.FirstTeamId);
                    gameModel.SecondTeam =
_context.Teams.Find(gameModel.SecondTeamId);
                    if (gameModel.FirstTeam == null || gameModel.SecondTeam ==
null)
                    {
                        gameModel.FirstTeam = _context.Teams.SingleOrDefault(model
=> model.Id == gameModel.FirstTeamId);
                        gameModel.SecondTeam =
_context.Teams.SingleOrDefault(model => model.Id == gameModel.SecondTeamId);
                    }
                }
```

Продолжение листинга М.1

```
                dict[tourModel] = games;
            }
            ViewData["Data"] = dict;
            return View(tournamentModel);
        }
        public async Task<IActionResult> Summary(int? id)
        {
            if (id == null)
            {
                return NotFound();
            }
            var tournamentModel = await _context.Tournaments
                .FirstOrDefaultAsync(m => m.Id == id);
            if (tournamentModel == null)
            {
                return NotFound();
            }
            UpdateStatus(tournamentModel);
            List<SummaryData> summaryDataList = new List<SummaryData>();
            var teams = await DbHelper.GetTeamsForTournament(tournamentModel);
            foreach (var teamModel in teams)
            {
                var summaryData = new SummaryData(_context, teamModel.Id);
                summaryDataList.Add(summaryData);
            }
            ViewData["SummaryData"] = summaryDataList;
            return View(tournamentModel);
        }
        public async Task<IActionResult> SetPlacement(int id)
        {
            if (id == 0)
                return NotFound();
            var team = _context.Teams.Find(id);
            if (team == null)
                return NotFound();
```

Продолжение листинга М.1

```
            var teamPlacement = new TeamPlacement();
            int placement = int.Parse((string)ViewData["Placement"]);
            if (placement == 0)
                return NotFound();
            teamPlacement.Placement = placement;
            teamPlacement.Team = team;
            _context.Add(teamPlacement);
            await _context.SaveChangesAsync();
            return View("Standings");
        }
        private bool TournamentModelExists(int id)
        {
            return _context.Tournaments.Any(e => e.Id == id);
        }
        private void UpdateStatus(TournamentModel tournamentModel)
        {
            if (tournamentModel.DateStart > DateTime.Now)
                tournamentModel.Status = TournamentStatus.Coming;
            else if (tournamentModel.DateEnd < DateTime.Now)
                tournamentModel.Status = TournamentStatus.Ended;
            else
                tournamentModel.Status = TournamentStatus.Started;
        }
    }
}
```

ПРИЛОЖЕНИЕ Н
ИСХОДНЫЙ КОД МОДЕЛИ CITY_MODEL

Листинг Н.1 – Исходный код «CityModel.cs»

```
using System;
using System.ComponentModel.DataAnnotations;

namespace Curling.Models
{
    public class CityModel
    {
        [Key]
        public int Id { get; set; }
        [Required(ErrorMessage = "{0} - необходимое поле")]
        [Display(Name = "Название")]
        public string Name { get; set; }
        [Display(Name = "Временная зона")]
        public DateTime UtcDayStart { get; set; }
    }
}
```

ПРИЛОЖЕНИЕ П
ИСХОДНЫЙ КОД МОДЕЛИ COACH_MODEL

Листинг П.1 – Исходный код «CoachModel.cs»

```
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;
using Curling.Areas.Identity.Data;

namespace Curling.Models
{
    public class CoachModel
    {
        [Key, Column("Id", Order = 1)]
        [DatabaseGeneratedAttribute(DatabaseGeneratedOption.Identity)]
        public int Id { get; set; }
        [Required(ErrorMessage = "{0} - необходимое поле")]
        [ForeignKey("CoachId")]
        public CurlingUser Coach { get; set; }
        public string CoachId { get; set; }
        [ForeignKey("SportsmanId")]
        public CurlingUser Sportsman { get; set; }
        public string SportsmanId { get; set; }
    }
}
```

ПРИЛОЖЕНИЕ Р
ИСХОДНЫЙ КОД МОДЕЛИ ENDING_MODEL

Листинг Р.1 – Исходный код «EndingModel.cs»

```
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc.ModelBinding;

namespace Curling.Models
{
    public class EndingModel
    {
        [Key]
        public int Id { get; set; }
        [ForeignKey("GameId")]
        [Required(ErrorMessage = "{0} - необходимое поле")]
        public GameModel Game { get; set; }
        public int GameId { get; set; }
        [Required(ErrorMessage = "{0} - необходимое поле")]
        [Display(Name = "Кол-во камней первой команды")]
        public int FirstTeamScore { get; set; }
        [Required(ErrorMessage = "{0} - необходимое поле")]
        [Display(Name = "Кол-во камней второй команды")]
        public int SecondTeamScore { get; set; }
        [Required(ErrorMessage = "{0} - необходимое поле")]
        public int Position { get; set; }
        [Required(ErrorMessage = "{0} - необходимое поле")]
        [Display(Name = "Закончить")]
        public bool IsFinal { get; set; }

    }
}
```

ПРИЛОЖЕНИЕ С
ИСХОДНЫЙ КОД МОДЕЛИ GAME_MODEL

Листинг С.1 – Исходный код «GameModel.cs»

```
using System;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace Curling.Models
{
    public class GameModel
    {
        [Key]
        public int Id { get; set; }
        [ForeignKey("FirstTeamId")]
        public TeamModel FirstTeam { get; set; }
        [Display(Name = "Первая команда")]
        public int? FirstTeamId { get; set; }
        [ForeignKey("SecondTeamId")]
        public TeamModel SecondTeam { get; set; }
        [Display(Name = "Вторая команда")]
        public int? SecondTeamId { get; set; }
        [Display(Name = "Постановка")]
        [Required(ErrorMessage = "{0} - необходимое поле")]
        public double Staging { get; set; }
        [Display(Name = "Постановка для первой команды")]
        public double FirstTeamStaging { get; set; }
        [Display(Name = "Постановка для второй команды")]
        public double SecondTeamStaging { get; set; }
        [Required(ErrorMessage = "{0} - необходимое поле")]
        [ForeignKey("TourId")]
        public TourModel Tour { get; set; }
        [NotMapped]
        public int TourId { get; set; }
    }
}
```

ПРИЛОЖЕНИЕ Т
ИСХОДНЫЙ КОД МОДЕЛИ GROUP_ASSIGNMENT_MODEL

Листинг Т.1 – Исходный код «GroupAssignmentModel.cs»

```
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace Curling.Models
{
    public class GroupAssignmentModel
    {
        [Key]
        public int Id { get; set; }
        [Required(ErrorMessage = "{0} - необходимое поле")]
        [ForeignKey("GroupId")]
        public GroupModel Group { get; set; }
        public int GroupId { get; set; }
        [Required(ErrorMessage = "{0} - необходимое поле")]
        [ForeignKey("TeamId")]
        public TeamModel Team { get; set; }
        public int TeamId { get; set; }
    }
}
```

ПРИЛОЖЕНИЕ У
ИСХОДНЫЙ КОД МОДЕЛИ GROUP_MODEL

Листинг У.1 – Исходный код «GroupModel.cs»

```
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;
using Microsoft.AspNetCore.Mvc.Rendering;

namespace Curling.Models
{    public class GroupModel
    {        [Key]
        public int Id { get; set; }
        [Required(ErrorMessage = "{0} - необходимое поле")]
        [Display(Name = "Название группы")]
        public string KeyCode { get; set; }
        [Required(ErrorMessage = "{0} - необходимое поле")]
        [ForeignKey("TournamentId")]
        public TournamentModel Tournament { get; set; }
        [Display(Name = "Соревнование")]
        public int TournamentId { get; set; }
        [NotMapped]
        [Display(Name = "Команда")]
        public int TeamId { get; set; }
        [NotMapped]
        public InputModel Input { get; set; }
        [NotMapped]
        [Display(Name = "Выберите команды")]
        public string[] SelectedIds { get; set; }
        public class InputModel
        {
            public List<SelectListItem> SelectListTeam { get; set; }
            public string TournamentName { get; set; } = "Unknown";
            [Display(Name = "Команды")]
            public List<SelectListItem> Teams { get; set; }
        }
    }
}
```

ПРИЛОЖЕНИЕ Ф
ИСХОДНЫЙ КОД МОДЕЛИ JUDGE_MODEL

Листинг Ф.1 – Исходный код «JudgeModel.cs»

```
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;
using Curling.Areas.Identity.Data;

namespace Curling.Models
{
    public class JudgeModel
    {
        [Key, Column("Id", Order = 1)]
        [DatabaseGeneratedAttribute(DatabaseGeneratedOption.Identity)]
        public int Id { get; set; }
        [Required(ErrorMessage = "{0} - необходимое поле")]
        [ForeignKey("UserId")]
        public CurlingUser Judge { get; set; }
        public string UserId { get; set; }
        [Required(ErrorMessage = "{0} - необходимое поле")]
        [ForeignKey("TournamentId")]
        public TournamentModel Tournament { get; set; }
        public int TournamentId { get; set; }
    }
}
```

ПРИЛОЖЕНИЕ X
ИСХОДНЫЙ КОД МОДЕЛИ SCORE_MODEL

Листинг X.1 – Исходный код «ScoreModel.cs»

```
namespace Curling.Models
{
    public class ScoreModel
    {
        [Key, Column("Id", Order = 1)]
        [DatabaseGeneratedAttribute(DatabaseGeneratedOption.Identity)]
        public int Id { get; set; }
        [Required]
        [ForeignKey("TeamTournamentAssignmentId")]
        public TeamTournamentAssignmentModel TeamTournamentAssignment { get; set;
}
        public int TeamTournamentAssignmentId { get; set; }
        [NotMapped]
        [Display(Name = "Команда")]
        public TeamModel TeamModel { get; set; }
        [NotMapped]
        [Display(Name = "Соревнование")]
        public TournamentModel Tournament { get; set; }
        [Display(Name = "Количество очков")]
        [NotMapped]
        public int Count => (int) Type;
        [Required(ErrorMessage = "{0} - необходимое поле")]
        [Display(Name = "Тип")]
        public ScoreType Type { get; set; }
        [Required(ErrorMessage = "{0} - необходимое поле")]
        [Display(Name = "Постановка")]
        public double Staging { get; set; }
        [NotMapped]
        public List<SelectListItem> TypeItems { get; set; }
    }
}
```

ПРИЛОЖЕНИЕ Ц
ИСХОДНЫЙ КОД МОДЕЛИ TEAM_ASSIGNMENT_MODEL

Листинг Ц.1 – Исходный код «TeamAssignmentModel.cs»

```
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;
using Curling.Areas.Identity.Data;

namespace Curling.Models
{
    public class TeamAssignmentModel
    {
        [Key, Column("Id", Order = 1)]
        [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
        public int Id { get; set; }
        [Required(ErrorMessage = "{0} - необходимое поле")]
        [ForeignKey("TeamId")]
        public TeamModel Team { get; set; }
        public int TeamId { get; set; }
        [Required(ErrorMessage = "{0} - необходимое поле")]
        [ForeignKey("UserId")]
        public CurlingUser Sportsman { get; set; }
        public string UserId { get; set; }
    }
}
```

ПРИЛОЖЕНИЕ Ш
ИСХОДНЫЙ КОД МОДЕЛИ TEAM_MODEL

Листинг Ш.1 – Исходный код «TeamModel.cs.cs»

```
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;
using Curling.Areas.Identity.Data;
using Microsoft.AspNetCore.Mvc.Rendering;

namespace Curling.Models
{
    public class TeamModel
    {
        [Key, Column("Id", Order = 1)]
        [DatabaseGeneratedAttribute(DatabaseGeneratedOption.Identity)]
        public int Id { get; set; }
        [Required(ErrorMessage = "{0} - необходимое поле")]
        [Display(Name = "Название команды")]
        public string Name { get; set; }
        [Display(Name = "Очки")]
        public int TotalScore { get; set; }
        [Display(Name = "Количество побед")]
        public int TotalWins { get; set; }
        [ForeignKey("CityId")]
        public CityModel City { get; set; }
        public int? CityId { get; set; }
        [NotMapped]
        public InputModel Input { get; set; }
        [NotMapped]
        [Display(Name = "Соревнование")]
        public int TournamentId { get; set; }
        [NotMapped]
        public string[] SelectedIds { get; set; }
        [NotMapped]
        public int Placement { get; set; }
        public class InputModel
        {
```

Продолжение листинга Ш.1

```
                [NotMapped]
                public List<SelectListItem> Tournaments { get; set; }
                [NotMapped]
                [Display(Name = "Выберите спортсменов")]
                public List<SelectListItem> Sportsmans { get; set; }
                [NotMapped]
                public List<CurlingUser> SportsmanList { get; set; }
                [NotMapped]
                [Display(Name = "Выберите спортсменов")]
                public string[] SelectedSportsmans { get; set; }
                [NotMapped]
                public KeyValuePair<List<CurlingUser>, CurlingUser[]> TeamSportsmans {
get; set; }
                public List<TournamentModel> TournamentParticipation { get; set; }
            }
        }
    }
```

ПРИЛОЖЕНИЕ Щ
ИСХОДНЫЙ КОД МОДЕЛИ TEAM_PLACEMENT

Листинг Щ.1 – Исходный код «TeamPlacement.cs»

```
using System;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace Curling.Models
{
    public class TeamPlacement
    {
        [Key]
        public int Id { get; set; }
        [ForeignKey("TeamId")]
        public TeamModel Team { get; set; }
        [Required(ErrorMessage = "{0} - необходимое поле")]
        [Display(Name = "Команда")]
        public int TeamId { get; set; }
        [Required(ErrorMessage = "{0} - необходимое поле")]
        [Display(Name = "Место")]
        public int Placement { get; set; }
        [ForeignKey("GroupId")]
        public GroupModel Group { get; set; }
        [Required]
        [Display(Name = "Группа")]
        public int GroupId { get; set; }
    }
}
```

ПРИЛОЖЕНИЕ Э
ИСХОДНЫЙ КОД МОДЕЛИ TEAM_PLACEMENT_GLOBAL

Листинг Э.1 – Исходный код «TeamPlacementGlobal.cs»

```
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace Curling.Models
{
    public class TeamPlacementGlobal
    {
        [Key]
        public int Id { get; set; }

        [ForeignKey("TeamId")]
        public TeamModel Team { get; set; }
        [Required(ErrorMessage = "{0} - необходимое поле")]
        public int TeamId { get; set; }
        [Required(ErrorMessage = "{0} - необходимое поле")]
        [Display(Name = "Место")]
        public int Placement { get; set; }
    }
}
```

ПРИЛОЖЕНИЕ Ю
ИСХОДНЫЙ КОД МОДЕЛИ TEAM_TOURNAMENT_ASSIGNMENT_MODEL

Листинг Ю.1 – Исходный код «TeamTournamentAssignmentModel.cs»

```
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace Curling.Models
{
    public class TeamTournamentAssignmentModel
    {
        [Key, Column("Id", Order = 1)]
        [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
        public int Id { get; set; }
        [Required(ErrorMessage = "{0} - необходимое поле")]
        [ForeignKey("TeamId")]
        public TeamModel Team { get; set; }
        public int TeamId { get; set; }
        [Required(ErrorMessage = "{0} - необходимое поле")]
        [ForeignKey("TournamentId")]
        public TournamentModel Tournament { get; set; }
        public int TournamentId { get; set; }
    }
}
```

ПРИЛОЖЕНИЕ Я
ИСХОДНЫЙ КОД МОДЕЛИ TOUR_MODEL

Листинг Я.1 – Исходный код «TourModel.cs»

```
using System;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace Curling.Models
{
    public class TourModel
    {
        [Key]
        public int Id { get; set; }
        [Required(ErrorMessage = "{0} - необходимое поле")]
        [ForeignKey("TournamentId")]
        public TournamentModel Tournament { get; set; }
        public int TournamentId { get; set; }
        [Required(ErrorMessage = "{0} - необходимое поле")]
        [Display(Name = "Дата и время начала")]
        public DateTime DateStart { get; set; }
    }
}
```

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД МОДЕЛИ TOURNAMENT_MODEL

Листинг А.1 – Исходный код «TournamentModel.cs»

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;
using Curling.Areas.Identity.Data;
using Curling.Areas.Identity.Pages.Account;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.RazorPages;
using Microsoft.AspNetCore.Mvc.Rendering;

namespace Curling.Models
{
    public class TournamentModel
    {
        [Key, Column("Id", Order = 1)]
        [DatabaseGeneratedAttribute(DatabaseGeneratedOption.Identity)]
        public int Id { get; set; }
        [Required(ErrorMessage = "{0} - необходимое поле")]
        [DisplayName("Название")]
        public string Name { get; set; }
        [DisplayName("Статус")]
        public TournamentStatus Status { get; set; }
        [Required(ErrorMessage = "{0} - необходимое поле")]
        [DisplayName("Город")]
        public string City { get; set; }
        [Required(ErrorMessage = "{0} - необходимое поле")]
        [DisplayName("Стадион")]
        public string Stadium { get; set; }
        [Required(ErrorMessage = "{0} - необходимое поле")]
        [DisplayName("Начало")]
        [DataType(DataType.DateTime)]
        public DateTime DateStart { get; set; }
        [Required(ErrorMessage = "{0} - необходимое поле")]
```

Продолжение листинга А.1

```
            [DisplayName("Конец")]
            [DataType(DataType.DateTime)]
            public DateTime DateEnd { get; set; }
            [Required(ErrorMessage = "{0} - необходимое поле")]
            [DisplayName("Краткая информация")]
            [DataType(DataType.MultilineText)]
            public string ShortInfo { get; set; }
            [Required(ErrorMessage = "{0} - необходимое поле")]
            [DisplayName("Подробная информация")]
            [DataType(DataType.MultilineText)]
            public string LongInfo { get; set; }
            [NotMapped]
            public InputModel Input { get; set; }
            [NotMapped]
            public string[] SelectedIds { get; set; }
            public class InputModel
            {
                public IEnumerable<SelectListItem> SelectedIds { get; set; }
                public List<TeamModel> Teams { get; set; }
                public List<CurlingUser> JudgeUsers { get; set; }
                [Display(Name = "Судьи")]
                public List<SelectListItem> Judges { get; set; }
                public List<CurlingUser> SportsmanUsers { get; set; }
                public Dictionary<TeamModel, KeyValuePair<List<CurlingUser>,
CurlingUser[]>> TeamSportsmans { get; set; }
                public List<CurlingUser> Coaches { get; set; }
                public bool AddingNewTeam { get; set; } = true;
                public EndingModel Ending { get; set; }
            }
        }
    }
```

ПРИЛОЖЕНИЕ В
ИСХОДНЫЙ КОД МОДЕЛИ ERROR_VIEW_MODEL

Листинг В.1 – Исходный код «ErrorViewModel»

```
using System;


namespace Curling.Models
{
    public class ErrorViewModel
    {
        public string RequestId { get; set; }
        public bool ShowRequestId => !string.IsNullOrEmpty(RequestId);
    }
}
```

ПРИЛОЖЕНИЕ С
ИСХОДНЫЙ КОД ДЛЯ ЗАПРОСОВ К БАЗЕ ДАННЫХ

Листинг С.1 – Исходный код запросов к базе данных

```
System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using Curling.Areas.Identity.Data;

using Curling.Models;

using Microsoft.AspNetCore.Mvc.Rendering;

using Microsoft.EntityFrameworkCore;


namespace Curling.Data

{

    public static class DbHelper

    {

        public static CurlingContext Context { get; set; } //для применения
миграций и запросов (работа с бд)

        public static async Task<List<TeamModel>>
GetTeamsForTournament(TournamentModel tournament) //запрос для получение всех команд
турнира

        {

            var teamTournamentAssignmentsCache = await
Context.TeamTournamentAssignments.ToListAsync();

            var teamsCache = await Context.Teams.ToListAsync();

            var result = from tta in teamTournamentAssignmentsCache

                join team in teamsCache on tta.Team.Id equals team.Id

                where tta.Tournament == tournament

                select team;

            return result.ToList();

        }

        public static async Task<List<TeamModel>>
GetNonGroupedTeamsForTournament(TournamentModel tournament)//получение команд без группы

        {

            var teamsForTournament = await GetTeamsForTournament(tournament);

            var groupAssignments = await Context.GroupAssignments.ToListAsync();

            var teams = await Context.Teams.ToListAsync();

            var groups = await Context.Groups.ToListAsync();

            var assignedTeamsQuery = from grp in groupAssignments

                join team in teams on grp.Team equals team
```

Продолжение листинга С.1

```
                    join g in groups on grp.Group equals g
                    where g.Tournament == tournament
                    select team;
                foreach (var teamModel in assignedTeamsQuery.ToList())
                {
                    teamsForTournament.Remove(teamModel);
                }
                return teamsForTournament;
            }
            public static async Task<List<TeamModel>> GetTeamsForGroup(GroupModel
group) // получение команд, которые привязаны к группе
            {
                var groupAssignments = await Context.GroupAssignments.ToListAsync();
                var teams = await Context.Teams.ToListAsync();
                var result = from groupAssignment in groupAssignments
                    join team in teams on groupAssignment.Team equals team
                    where groupAssignment.Group == @group
                    select team;
                return result.ToList();
            }
            public static async Task<List<CurlingUser>> GetUsersForTeam(TeamModel tm)
//получение спортсменов в команде
            {
                var teamAssignmentsCache = await
Context.TeamAssignments.ToListAsync();
                var usersCache = await Context.Users.ToListAsync();
                var result = from teamAssign in teamAssignmentsCache
                    join user in usersCache on teamAssign.Sportsman equals user
                    where user.Role == "Sportsman" && teamAssign.Team == tm
                    select user;
                return result.ToList();
            }
            public static async Task<List<CurlingUser>>
GetJudgesForTournament(TournamentModel tournament) //получение судей соревнований
            {
                var usersCache = await Context.Users.ToListAsync();
                var judgeCache = await Context.Judges.ToListAsync();
```

Продолжение листинга С.1

```
                var result = from user in usersCache
                    join judge in judgeCache on user.Id equals judge.Judge.Id
                    where user.Role == "Judge" && judge.Tournament == tournament
                    select user;
                return result.ToList();
            }
            public static async Task<List<TourModel>>
GetToursForTournament(TournamentModel tournament) //получение списка всех туров
            {
                var tours = await Context.Tours.ToListAsync();

                var result = from t in tours
                    where t.Tournament == tournament
                    select t;
                return result.ToList();
            }
            public static async Task<List<GameModel>> GetGamesForTour(TourModel tour)
//получение игр в туре
            {
                var games = await Context.Games.ToListAsync();
                var result = from g in games
                             where g.Tour == tour
                             select g;
                return result.ToList();
            }

            public static async Task<Dictionary<GroupModel, Dictionary<TeamModel,
KeyValuePair<List<CurlingUser>, CurlingUser[]>>>// получение данных для таблица со
счетом
                GetTournamentSportsmanDataForTournament(TournamentModel tournament)
            {
                Dictionary<GroupModel, Dictionary<TeamModel,
KeyValuePair<List<CurlingUser>, CurlingUser[]>>> data =
                    new Dictionary<GroupModel, Dictionary<TeamModel,
KeyValuePair<List<CurlingUser>, CurlingUser[]>>>();
                var groups = await Context.Groups.ToListAsync();
                groups = groups.Where(g => g.Tournament == tournament).ToList();
                foreach (var groupModel in groups)
```

Продолжение листинга С.1

```
                {
                    var teamsInGroup = await DbHelper.GetTeamsForGroup(groupModel);
                    var dict = new Dictionary<TeamModel,
KeyValuePair<List<CurlingUser>, CurlingUser[]>>();
                    foreach (var teamModel in teamsInGroup)
                    {
                        var sportsmen = await DbHelper.GetUsersForTeam(teamModel);
                        var coaches = new CurlingUser[sportsmen.Count];
                        for (int i = 0; i < sportsmen.Count; i++)
                        {
                            var coach = await
DbHelper.GetCoachForSportsman(sportsmen[i]);
                            coaches[i] = coach;
                        }
                        var kv = new KeyValuePair<List<CurlingUser>,
CurlingUser[]>(sportsmen, coaches);
                        teamModel.TotalScore = await GetTotalScoreForTeam(teamModel);
                        dict[teamModel] = kv;
                    }
                    data[groupModel] = dict;
                }
                return data;
            }
            public static async Task<int> GetTotalScoreForTeam(TeamModel team)
//получение общего счет для команды
            {
                var teamN = 0;
                var games = Context.Games.AsEnumerable().Where(model =>
                {
                    if (model.FirstTeam == team)
                    {
                        teamN = 0;
                        return true;
                    }
                    teamN = 1;
                    return true;
                }).ToList();
```

Продолжение листинга С.1

```
                var total = 0;
                foreach (var gameModel in games)
                {
                    var endings = await Context.Endings.Where(model => model.Game ==
gameModel).ToListAsync();
                    foreach (var endingModel in endings)
                    {
                        if (teamN == 0)
                            total += endingModel.FirstTeamScore;
                        else
                            total += endingModel.SecondTeamScore;
                    }
                }
                return total;
            }
            public class StandingsData //вспмогательные данные для счета
            {
                public int TotalFirst { get; set; } //всего очков
                public int TotalSecond { get; set; }
                public int FinalFirst { get; set; } //общий счет в игре
                public int FinalSecond { get; set; }
                public double TotalDscFirst { get; set; }
                public double TotalDscSecond { get; set; }
            }
            public static StandingsData GetTotalScoreForTeams(TeamModel team1,
TeamModel team2, GroupModel group) //получение постановки для команды
            {
                List<GameModel> games;
                if (group != null)
                {
                    var groupAssigns = Context.GroupAssignments.Where(g => g.Group ==
group).ToList();
                    games = Context.Games.Where(model =>
                        groupAssigns.Any(assignmentModel => assignmentModel.Team ==
team1) &&
                        groupAssigns.Any(assignmentModel => assignmentModel.Team ==
team2) &&
```

Продолжение листинга С.1

```
                        ((model.FirstTeam == team1 && model.SecondTeam == team2) ||
                         (model.SecondTeam == team1 && model.FirstTeam ==
team2))).ToList();
                }
                else
                {
                    games = Context.Games.Where(model =>
                        ((model.FirstTeam == team1 && model.SecondTeam == team2) ||
                         (model.SecondTeam == team1 && model.FirstTeam ==
team2))).ToList();
                }
                var totalFirst = 0;
                var totalSecond = 0;
                var finalFirst = 0;
                var finalSecond = 0;
                var totalDscFirst = 0.0;
                var totalDscSecond = 0.0;
                foreach (var gameModel in games)
                {
                    var endings = Context.Endings.Where(model => model.Game ==
gameModel).ToList();
                    totalDscFirst += gameModel.FirstTeamStaging;
                    totalDscSecond += gameModel.SecondTeamStaging;
                    foreach (var endingModel in endings)
                    {
                        totalFirst += endingModel.FirstTeamScore;
                        totalSecond += endingModel.SecondTeamScore;
                    }
                }
                if (games.Any())
                {
                    if (totalFirst > totalSecond)
                    {
                        finalFirst = (int) ScoreType.Win;
                        finalSecond = (int) ScoreType.Loss;
                    }
```

Продолжение листинга С.1

```
                else if (totalFirst < totalSecond)
                {
                    finalFirst = (int) ScoreType.Loss;
                    finalSecond = (int) ScoreType.Win;
                }
                else
                {
                    finalFirst = (int) ScoreType.Draw;
                    finalSecond = (int) ScoreType.Draw;
                }
            }
            return new StandingsData
            {
                TotalFirst = totalFirst,
                TotalSecond = totalSecond,
                FinalFirst = finalFirst,
                FinalSecond = finalSecond,
                TotalDscFirst = totalDscFirst,
                TotalDscSecond = totalDscSecond
            };
        }
        public class StandingScore //вспомогательные данные
        {
            public int TotalScore { get; set; }
            public int TotalWins { get; set; }
            public double Dsc { get; set; }
        }
        public static StandingScore GetTotalScoreForTeamAndGroup(TeamModel team,
GroupModel group)
            //получение счета для команды, которая в определенной группе
        {
            var ga = Context.GroupAssignments.FirstOrDefault(g => g.Group ==
@group);

            if (ga == null) return new StandingScore();
```

Продолжение листинга С.1

```
                var gamesOfTeam1 = Context.Games.Where(game =>
                    (ga.Team == game.FirstTeam || ga.Team == game.SecondTeam) &&
                    game.FirstTeam == team
                    && game.Tour.Tournament == group.Tournament)
                    .ToList();
                var gamesOfTeam2 = Context.Games.Where(game =>
                    (ga.Team == game.FirstTeam || ga.Team == game.SecondTeam) &&
                    game.SecondTeam == team
                    && game.Tour.Tournament == group.Tournament)
                    .ToList();
                var total = 0;
                var totalWins = 0;
                var dsc = 0.0;
                foreach (var gameModel in gamesOfTeam1)
                {
                    var totalCl = GetTotalScoreForTeams(gameModel.FirstTeam,
gameModel.SecondTeam, group);
                    total += totalCl.FinalFirst;
                    if (totalCl.FinalFirst == (int) ScoreType.Win)
                        totalWins++;
                    dsc += gameModel.FirstTeamStaging;
                }
                foreach (var gameModel in gamesOfTeam2)
                {
                    var totalCl = GetTotalScoreForTeams(gameModel.FirstTeam,
gameModel.SecondTeam, group);
                    total += totalCl.FinalSecond;
                    if (totalCl.FinalSecond == (int)ScoreType.Win)
                        totalWins++;
                    dsc += gameModel.SecondTeamStaging;
                }
                return new StandingScore
                {
                    Dsc = dsc,
                    TotalScore = total,
                    TotalWins = totalWins
```

Продолжение листинга С.1

```
                };
            }
            public static StandingScore GetTotalScoreForTeam2(TeamModel team)
//получение счета для команды
            {
                var gamesOfTeam1 = Context.Games.Where(game => game.FirstTeam ==
team).ToList();
                var gamesOfTeam2 = Context.Games.Where(game => game.SecondTeam ==
team).ToList();
                var total = 0;
                var dsc = 0.0;
                var totalWins = 0;
                foreach (var gameModel in gamesOfTeam1)
                {
                    var totalCl = GetTotalScoreForTeams(gameModel.FirstTeam,
gameModel.SecondTeam, null);
                    total += totalCl.FinalFirst;
                    dsc += gameModel.FirstTeamStaging;
                    if (totalCl.FinalFirst == 3)
                        totalWins++;
                }
                foreach (var gameModel in gamesOfTeam2)
                {
                    var totalCl = GetTotalScoreForTeams(gameModel.FirstTeam,
gameModel.SecondTeam, null);
                    total += totalCl.FinalSecond;
                    dsc += gameModel.SecondTeamStaging;
                    if (totalCl.FinalSecond == 3)
                        totalWins++;
                }
                return new StandingScore
                {
                    Dsc = dsc,
                    TotalScore = total,
                    TotalWins = totalWins
                };
            }
```

Продолжение листинга С.1

```
            public static StandingScore GetTotalScoreForTeam3(TeamModel team,
GroupModel group)
            {
                var gas = Context.GroupAssignments.Where(model => model.Group ==
group).ToList();
                var gamesOfTeam1 = Context.Games.Where(game => game.FirstTeam == team)
                    .ToList();
                var gamesOfTeam2 = Context.Games.Where(game => game.SecondTeam ==
team)
                    .ToList();
                var games1 = gamesOfTeam1.FindAll(model => gas.Any(mdl => mdl.Team ==
model.FirstTeam));
                var games2 = gamesOfTeam2.FindAll(model => gas.Any(mdl => mdl.Team ==
model.SecondTeam));
                var total = 0;
                var dsc = 0.0;
                var totalWins = 0;
                foreach (var gameModel in games1)
                {
                    var totalCl = GetTotalScoreForTeams(gameModel.FirstTeam,
gameModel.SecondTeam, group);
                    total += totalCl.FinalFirst;
                    if (gas.Any(model => model.Team == gameModel.FirstTeam) &&
gas.Any(model => model.Team == gameModel.SecondTeam))
                        dsc += gameModel.FirstTeamStaging;
                    if (totalCl.FinalFirst == 3)
                        totalWins++;
                }
                foreach (var gameModel in games2)
                {
                    var totalCl = GetTotalScoreForTeams(gameModel.FirstTeam,
gameModel.SecondTeam, group);
                    total += totalCl.FinalSecond;
                    if (gas.Any(model => model.Team == gameModel.FirstTeam) &&
gas.Any(model => model.Team == gameModel.SecondTeam))
                        dsc += gameModel.SecondTeamStaging;
                    if (totalCl.FinalSecond == 3)
                        totalWins++;
                }
```

Продолжение листинга С.1

```
                return new StandingScore
                {
                    Dsc = dsc,
                    TotalScore = total,
                    TotalWins = totalWins
                };
            }
            public static async Task<List<SelectListItem>> GetJudges() //получение
всех судей
            {
                var usersCache = await Context.Users.ToListAsync();
                var list = usersCache.Where(user => user.Role == "Judge");
                List<SelectListItem> userList = new List<SelectListItem>();
                foreach (var user in list)
                    userList.Add(new SelectListItem(user.FirstName + " " +
user.LastName, user.Id));
                return userList;
            }
            public static async Task<CurlingUser> GetCoachForSportsman(CurlingUser
sportsman)//поиск тренера для спортсмена
            {
                var usersCache = await Context.Users.ToListAsync();
                var coachCache = await Context.Coaches.ToListAsync();
                var coach = Context.Coaches.FirstOrDefault(c => c.Sportsman ==
sportsman);
                if (coach == null)
                    return null;
                var co = Context.Users.FirstOrDefault(c => c == coach.Coach);
                return co;
            }
            public static async Task<List<CurlingUser>> GetSportsmenForTeam(TeamModel
team)//получить список спортсменов в команде
            {
                var users = await Context.Users.ToListAsync();
                var teamAssignments = await Context.TeamAssignments.ToListAsync();

                var result = from teamAssign in teamAssignments
```

Продолжение листинга С.1

```
                    join u in users on teamAssign.Sportsman equals u
                    where teamAssign.Team == team
                    select u;
                return result.ToList();
            }
            public static async Task<List<TournamentModel>>
GetTournamentsForTeam(TeamModel team) //список всех соревнований, в которых участвует
команда
            {
                var tours = await Context.Tournaments.ToListAsync();
                var teamTourAssign = await
Context.TeamTournamentAssignments.ToListAsync();
                var result = from teamAssign in teamTourAssign
                    join tour in tours on teamAssign.Tournament equals tour
                    where teamAssign.Team == team
                    select tour;
                return result.ToList();
            }
            public static async Task<bool> IsUserInTeam(CurlingUser user, TeamModel
team)//проверка: находтся ли спортсмен в команде
            {
                var teamAssignments = await Context.TeamAssignments.ToListAsync();
                var result = from teamAssign in teamAssignments
                    where teamAssign.Sportsman == user && teamAssign.Team == team
                    select teamAssign;

                return result.Any();
            }

            public static async Task<List<TeamModel>> GetAllTeamsInGroup(GroupModel
group)//получение списка всех команд из группы
            {
                var groupAssignments = await Context.GroupAssignments.ToListAsync();
                var teams = await Context.Teams.ToListAsync();
                var result = from groupAssign in groupAssignments
                    join team in teams on groupAssign.Team equals team
                    where groupAssign.Group == @group
```

Продолжение листинга С.1

```
                    select team;

                return result.ToList();

            }

            public static async Task<List<GroupModel>>
GetAllGroupsInTournament(TournamentModel tournament) //получение списка всех группы на
соревновании

            {

                var groups = await Context.Groups.ToListAsync();

                var result = from grp in groups

                    where grp.Tournament == tournament

                    select grp;

                return result.ToList();

            }

            public static async Task<List<EndingModel>> GetEndingsForGame(GameModel
gameModel) //список всех эндов для игры

            {

                var endings = await Context.Endings.ToListAsync();

                var result = from end in endings

                    where end.Game == gameModel

                    select end;

                return result.ToList();

            }

            public static TeamPlacement GetPlacementForTeamAndGroup(TeamModel team,
GroupModel group)//получить текущее место для команды в группе

            {

                var result = Context.TeamPlacements.FirstOrDefault(placement =>
placement.Team == team && placement.Group == group);

                return result;

            }

        }

    }
```