ВВЕДЕНИЕ

В наше время развитых электронных технологий высока необходимость в возможности быстро и эффективно оперировать большими, сложно взаимосвязанными объемами данных. Эту возможность предоставляют базы данных. Также к преимуществам баз данных, а именно реляционных баз данных, в которых данные хранятся в виде таблиц, можно отнести компактность, легкую организацию работы с помощью единого языка SQL [1], возможность надежного совместного доступа.

Целью данной курсовой работы является создание информационной системы «Бронирование билетов на соревнования по теннису», которая должна оперировать с, возможно, довольно большим объемом данных об игроках, стадионах, матчах, билетах и пользователях, причем делать это достаточно быстро, а также осуществлять гибкий поиск по данным. Реляционные базы данных, как можно убедиться, предоставляют удобные инструменты для воплощения этой цели в жизнь, при этом при разумных затратах времени и ресурсов со стороны разработчика [2]. Именно поэтому они и были выбраны для использования в проекте.

Данная программа позволяет автоматизировать некоторые операции, совершаемые человеком при бронировании билетов онлайн, облегчить и ускорить этот процесс.

Также при реализации данного программного обеспечения используется объектно-ориентированный подход, главными принципами которого являются инкапсуляция, наследование и полиморфизм. Объектно-ориентированный подход позволяет вести независимую разработку отдельных частей программы, при этом код программы остается легко модифицируемым и наглядным.

1 АНАЛИЗ И КОНЦЕПТУАЛЬНОЕ МОДЕЛИРОВАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ

В данной курсовой работе будет рассмотрена информационная система «Бронирование билетов на соревнования по теннису».

Данная система предназначена для просмотра пользователями информации о теннисном турнире и бронирования билетов на матчи этого турнира. Система позволяет организаторам соревнований вести базу данных игроков, матчей, стадионов и зрителей, которые обращаются к сайту при необходимости бронирования билетов. Администраторы данного сайта смогут добавлять и изменять информацию на данном сайте. При этом обычными пользователями так же может вестись мониторинг базы игроков, матчей и стадионов.

Предметная область «Бронирование билетов на соревнования» на сегодняшний день сталкивается с такими проблемами, как отсутствие возможности бронирования билетов на матчи с наглядным отображением схемы свободных мест стадиона. Также необходима возможность отмены брони пользователем. Разработанная информационная система предназначена для автоматизирования приведенных выше проблем и значительно упростит процесс организации турнира.

Функциональная структура схематически представлена на рисунке 1.1. Данная структура описывает основные взаимодействия, которые происходят в процессе регистрации пользователей и просмотра ими предстоящих матчей турнира. Потенциальный зритель должен иметь возможность зарегистрироваться, предварительно заполнив все необходимые поля должным образом. После этого зарегистрированный пользователь имеет доступ ко всем функциям данной информационной системы. Незарегистрированный пользователь также может просматривать информацию о турнире, но не может бронировать билеты. Администратору данного сайта доступны все те функции, что и зарегистрированному пользователю. Кроме того, он может добавлять, редактировать и удалять содержание данного сайта.

Организация работы данной системы является комплексной областью деятельности. В таком случае функциональными объектами системы являются администраторы, пользователи и сама программа (рис. 1.1).

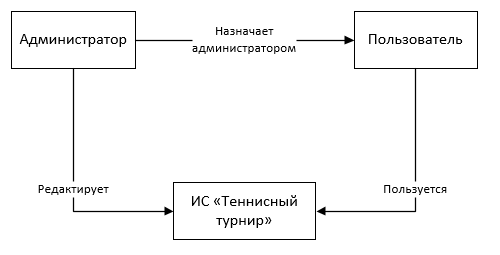


Рисунок 1.1 – Функциональная структура информационной системы «Бронирование билетов на соревнования»

Информационные потребности пользователей данной информационной следующие:

Для незарегистрированных пользователей:

1. регистрация на сайте;
2. вход на сайт;
3. просмотр информации и матчах;
4. просмотр информации о стадионах;
5. просмотр свободных и занятых мест на матч;
6. просмотр информации об игроках, принимающих участие в турнире;

Для зарегистрированных пользователей:

1. все потребности зарегистрированных пользователей;
2. бронирование билетов на определённый матч;
3. отмена брони билетов на определённый матч;
4. обратная связь с администраторами.

Для администраторов:

1. все потребности зарегистрированных пользователей;
2. изменение данных на сайте;
3. добавление данных на сайт;
4. удаление данных с сайта;
5. просмотр профилей зарегистрированных пользователей и забронированных ими билетов;
6. назначение пользователя администратором.

Документооборот предметной области «Бронирование билетов на соревнования по теннису» состоит из:

1. отчёта о всех сыгранных матчах, находящегося в доступе у всех пользователей. Состоит из информации о прошедшем или предстоящем матче.
2. отчёта о брони, показывающего билеты, которые забронировал пользователь, и секретный код;
3. билетов на матч. Состоит из информации об участниках матча, стадионе, дате матча, сектора, ряда и места.

Далее приведено описание объектов данной информационной системы и описание связей между ними.

Обобщенная схема взаимодействия объектов ПО изображается на рисунке 1.2. Рассмотрим ее подробней.

Описание объектов ПО и их атрибутов:

1. пользователь: Id пользователя, фамилия, имя, дата рождения, домашний адрес, контактный номер телефона, email-адрес, роль, логин, зашифрованный пароль.
2. матч: Id матча, результат матча, дата проведения, Id первого участника, Id второго участника, Id стадиона, цена билета;
3. игрок: Id игрока, фамилия, имя, дата рождения, мировой рейтинг, дополнительная информация;
4. стадион: Id стадиона, информация о стадионе, Адрес стадиона;
5. покупка: код, Id пользователя;
6. билет: Id билета, Id матча, Id покупки, место, сектор, ряд.

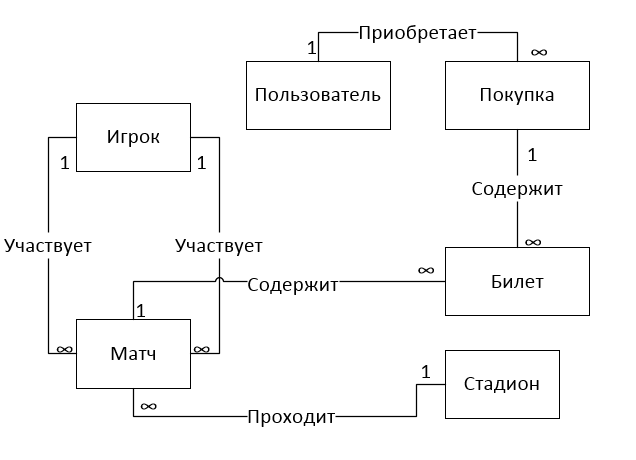


Рисунок 1.2 – Схема взаимосвязи объектов ПО

В данной предметной области присутствуют такие алгоритмические зависимости:

1. составление турнирной рейтинговой таблицы, основанной участниках и их мировых рейтингах;
2. MD5-шифрование паролей для пользователей системы.

В данной предметной области присутствуют такие ограничения целостности для БД:

1. в каждом матче участвуют ровно два игрока;
2. каждый матч проходит ровно на одной арене;
3. каждый билет относится только к одному матчу;
4. каждый билет либо не забронирован, либо принадлежит только одной покупке;
5. каждый билет не может быть купленным, если он забронированный;
6. у каждого пользователя только одна роль;
7. при регистрации пользователя поля Email, Логин и Пароль обязательные;
8. игрок без мирового рейтинга существовать не может;
9. каждая покупка принадлежит ровно одному пользователю.
10. Требования к информационной системе и существующие ограничения:
11. каждый пользователь получает доступ только к функционалу, согласно роли;
12. пользователь может бронировать места только за определённое время до матча;
13. пользователь не может забронировать более десяти билетов.

Для выполнения поставленных задач и целей будет создана информационная система «Бронирование билетов на соревнования по теннису» средствами языка программирования C# в виде веб-приложения на основе технологии ASP.Net. Приложение будет разработано в среде Microsoft Visual Studio 2013 с использованием СУБД Microsoft SQL Server 2012.

2 ПОСТАНОВКА ЗАДАЧИ

Необходимо организовать процесс регистрации пользователя с внесением в базу данных информации о нём, а также с последующим входом в систему. Существует следующая информация о пользователе: фамилия, имя, страна и город проживания, email-адрес. Также пользователь после регистрации имеет свой логин, пароль и роль. Так как целью рассматриваемого проекта является упрощение процесса бронирования билетов на теннисный турнир, отмены бронирования билетов и возможность удобного просмотра информации об этом турнире, разрабатываемая система должна иметь высокую надежность, интуитивно понятный интерфейс и выполнять функции, являющиеся задачами автоматизации в данной системе.

После заполнения регистрационной формы, она проходит проверку системой, и либо сохраняется в базу и создается соответствующий пользователь, либо сообщается об отказе в регистрации.

После регистрации пользователи имеют возможность кроме просмотра информации о турнире ещё и возможность бронирования или отмены бронирования билетов на матчи. Администраторы после входа в систему имеют возможность добавления, редактирования и удаления информации. Во время добавления нового игрока в базу, он попадает в общий рейтинг участников на основе своего мирового рейтинга. При добавлении нового матча в базу данных должны автоматически генерироваться билеты на данный матч. Система должна выполнять следующие задачи:

1. внесение информации о пользователе;
2. подтверждение регистрации пользователем;
3. внесение информации о новом игроке, стадионе, матче;
4. редактирование информации об игроке, стадионе матче;
5. редактирование информации о собственном профиле;
6. удаление информации о матче, стадионе, игроке;
7. составление общей таблицы участников на основе мирового рейтинга игроков;
8. поиск игроков по произвольному признаку;
9. сортировка матчей по дате проведения;
10. фильтрация игроков по мировому рейтингу;
11. получение статистики сыгранных матчей;
12. получение статистики посетителей матчей, исходя из данных, предоставленных пользователем;
13. создание статистик матчей, проведённых в определённый день;
14. возможность обращения с произвольным запросом к базе данных;
15. автоматическое создание отчёта о сыгранных матчах по стандартному шаблону, который содержит фамилию, имя, дату рождения, страну, личную информацию и прочее, а также информацию о его результатах в данном турнире;
16. автоматическое создание билета на матч по стандартному шаблону;
17. шифрование паролей, используемых пользователями сайта;
18. бронирование мест пользователями;
19. отмена бронирования мест пользователями;
20. автоматическая генерация билетов на матч при добавлении нового матча.

3 ПРОЕКТИРОВАНИЕ БАЗЫ ДАННЫХ

3.1 Построение UML-диаграммы

В данной системе присутствуют 3 вида актеров, каждый из которых играет отведенную ему роль. Актер незарегистрированный пользователь имеет возможность внести свои данные в систему для сохранения. Данный элемент USE CASE будет включать валидацию данных. Незарегистрированный пользователь также может просматривать информацию об игроках, стадионах, на которых проходят матчи, результаты матчей и статистику прошедших матчей.

Актер зарегистрированный пользователь имеет возможность просматривать информацию об игроках, результаты матчей, статистику прошедших матчей, бронировать билеты на интересующие его матчи или отменять бронирование мест. Также он может вносить дополнительные данные о себе или изменять персональные данные.

Актер Администратор имеет кроме тех возможностей, который имеет зарегистрированный пользователь, ещё и возможность добавления, изменения и удаления информации об игроках, матчах, аренах. Кроме того, он может назначать других пользователей администраторами. Администратор также может просматривать профили пользователей, зарегистрированных на данном сайте.

На основе этого составим USE CASE диаграмму предложенной системы (рисунок 3.1).

Данная информационная система будет иметь трехуровневую архитектуру. Это позволит улучшить структуру системы и повысить ее безопасность. Учитывая архитектуру системы, можно выделить три основных узла, которые присутствуют в ней: «Клиент», «Сервер», «СУБД».

Узел «Клиент» отвечает за вывод необходимой информации конечному пользователю и так же за предоставление пользователю удобного интерфейса для взаимодействий с системой в целом. В узле «Сервер» будет выполняться вся основная логика по работе системы. И последний узел отвечает за хранение внесенных данных.

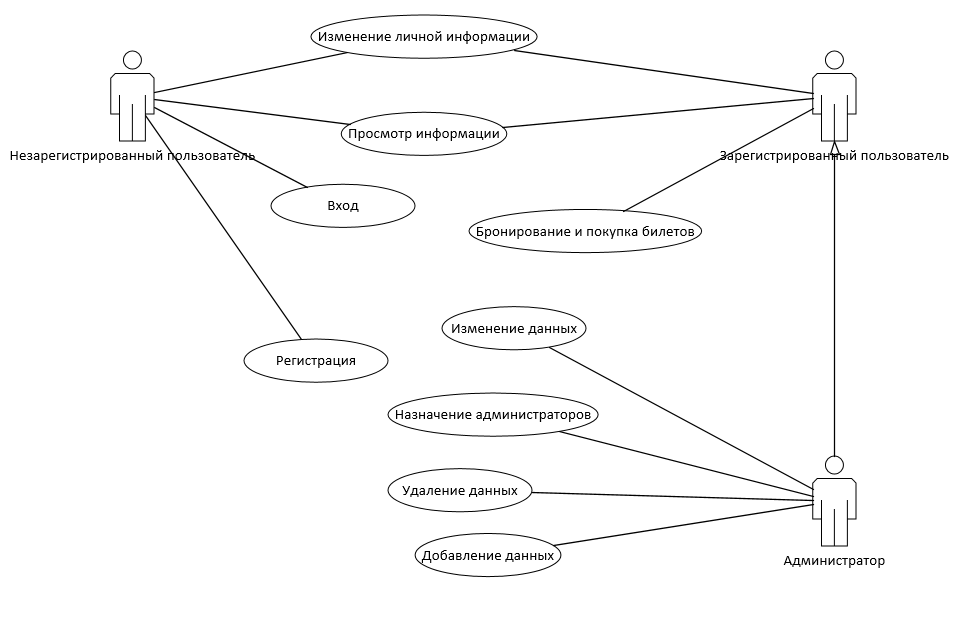


Рисунок 3.1 – USE CASE диаграмма «Бронирование билетов на соревнования»

«Клиент» и «Сервер» связаны между собой напрямую. А «Сервер» и «СУБД» связаны между собой через посредника. В качестве посредника выступает DAO Layer. Он позволяет серверу абстрагироваться от типа СУБД и сохранять и загружать объекты, хранящиеся в базе данных, без информации о том, как эти данные хранятся в базе данных и вообще какая база данных используется. На основе всей информации, приведенной выше можно построить диаграмму развертывания для данной информационной системы (рисунок 3.2).

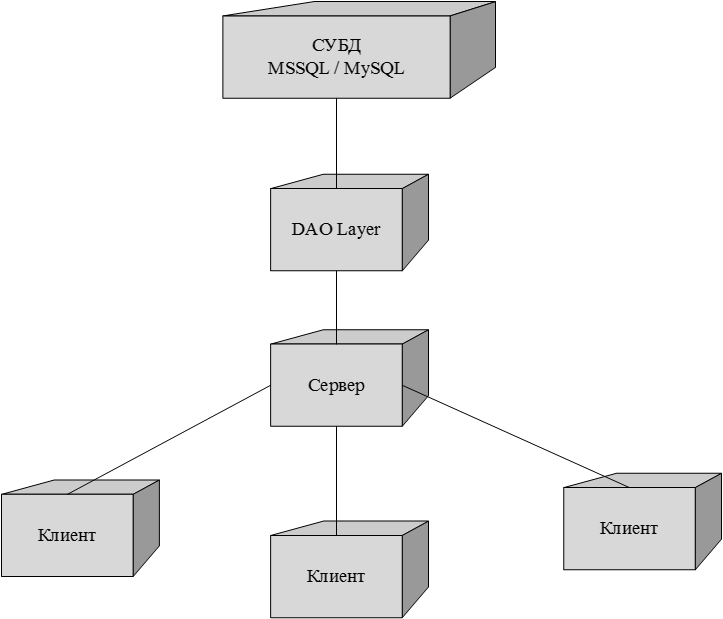


Рисунок 3.2 – Диаграмма развертывания «Системы бронирования билетов на соревнования»

3.2 Построение ER-диаграммы

На основании концептуальной модели предметной области, которая была получена в первом разделе можно получить такие сущности:

1. стержневая сущность «Игрок» с атрибутами фамилия, имя, информация, мировой рейтинг, страна. Первичный ключ – id\_игрока;
2. стержневая сущность «Матч» с атрибутами id\_игрок1, id\_игрок2, id\_стадиона, дата проведения, цена. Первичный ключ – id\_матча;
3. стержневая сущность «Стадион» с атрибутами информация, адрес. Первичный ключ – id\_стадиона;
4. стержневая сущность «Билет» с атрибутами id\_матча, сектор, ряд, место, код покупки, продано. Первичный ключ – id\_билета;
5. стержневая сущность «Покупка» с атрибутом id\_пользователя. Первичный ключ – Код;
6. стержневая сущность «Пользователь» с атрибутами фамилия, имя, логин, пароль, Email, роль, страна, город. Первичный ключ – id\_пользователя.

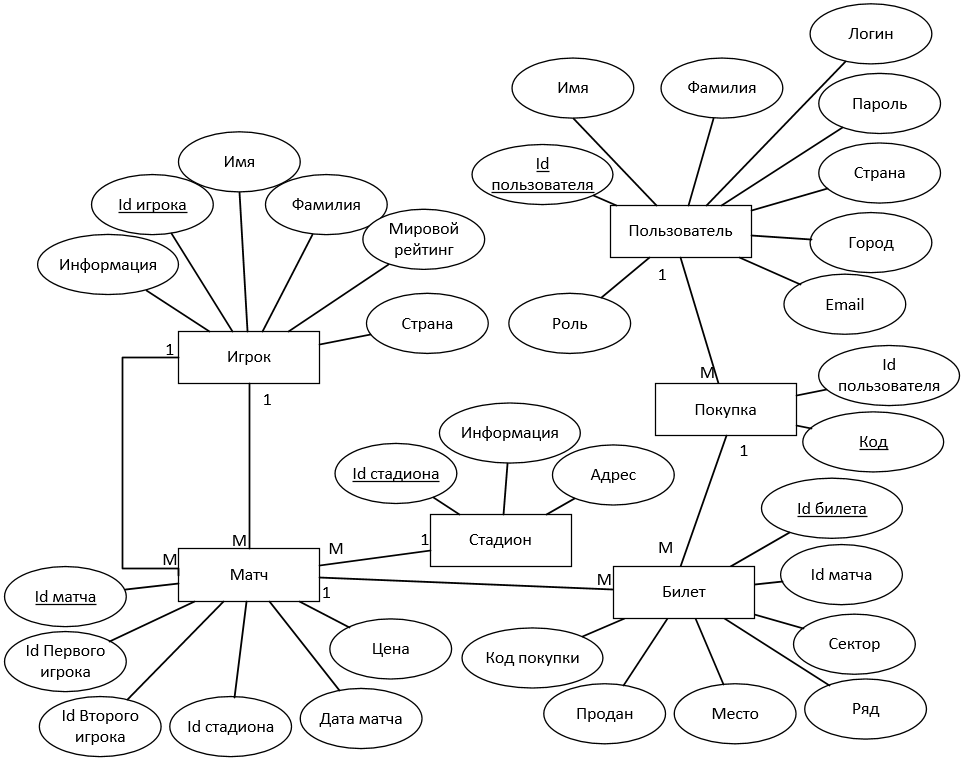


Рисунок 3.3 – ER-диаграмма предметной области «Система бронирования билетов на соревнования по теннису»

3.3 Построение схемы реляционной базы данных в третьей нормальной форме

На основе созданной ER-диаграммы (рис. 3.3) построим схему базы данных.

Данная схема приведена на рисунке 3.4.

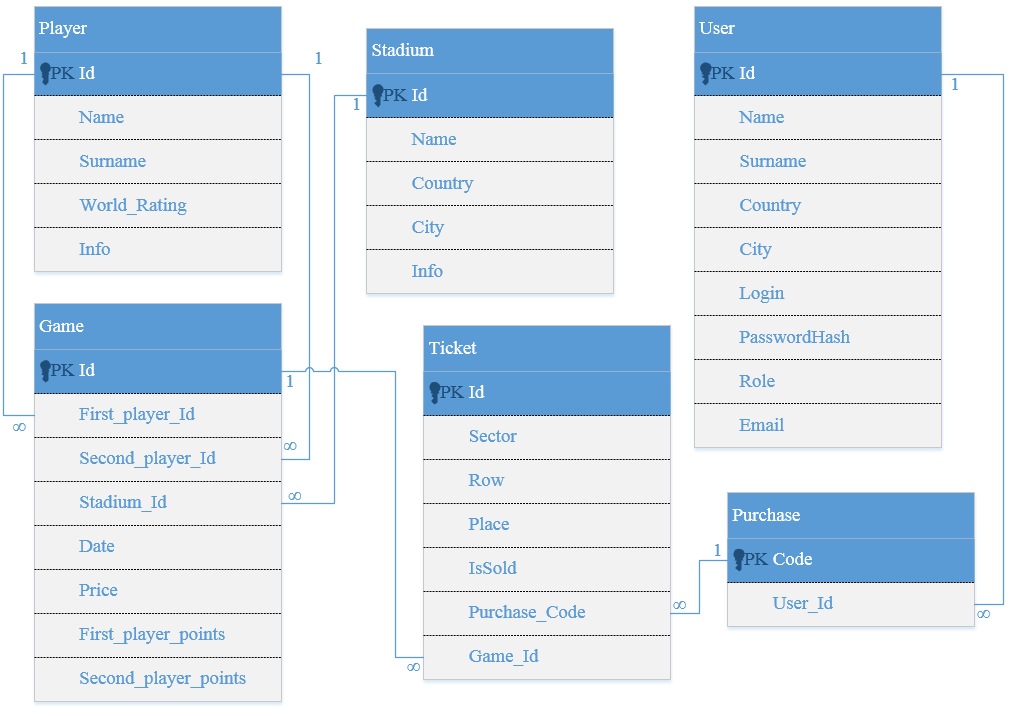


Рисунок 3.4 – Схема базы данных

Докажем, что данная схема базы данных находится в 3-ей нормальной форме (далее – НФ).

По определению, схема базы данных находится в 3-ей НФ, если каждое отношение из этой схемы находится в 3-ей НФ [3].

Рассмотрим отдельно каждое отношение из схемы:

1. Отношение «Player».

В данном отношении все значения атомарны, следовательно, оно находится в первой нормальной форме. Также в нем простой, а не составной ключ, следовательно, все неключевые атрибуты зависят только от Id. По определению, данное отношение находится во второй нормальной форме. Найдем все функциональные зависимости в этом отношении для того, чтобы доказать, что отношение находится в третьей нормальной форме. Они представлены в таблице 3.1.

Таблица 3.1 – Отношение «Player»

|  |
| --- |
| Id\* |
| Name |
| Surname |
| World\_Rating |
| Info |

Как видно по таблице, в данном отношении нет транзитивных зависимостей, следовательно, оно находится в 3-ей НФ.

1. Отношение «Game».

В данном отношении все значения атомарны, следовательно, оно находится в 1-ой НФ. Также в нем простой, а не составной ключ, следовательно, отношение находится во 2-ой НФ. Найдем все функциональные зависимости в этом отношении. Функциональные зависимости показаны в таблице 3.2.

Таблица 3.2 – Отношение «Game»

|  |
| --- |
| Id \* |
| First\_player\_Id |
| Second\_player\_Id |
| Stadium\_Id |
| Date |
| Price |
| First\_player\_points |
| Second\_player\_points |

В отношении нет транзитивных зависимостей, и оно находится в 3-ей НФ. В отношении имеются внешние ключи: атрибуты «First\_player\_Id», «Second\_player\_Id», связанные с атрибутом «Id» отношения «Player», и «Stadium\_Id» связанный с атрибутом «Id» отношения «Stadium».

1. Отношение «Stadium».

В данном отношении все значения атомарны, следовательно, оно находится в первой нормальной форме. Также в нем простой, а не составной ключ, следовательно, все неключевые атрибуты зависят только от Id. По определению, данное отношение находится во второй нормальной форме. Найдем все функциональные зависимости в этом отношении для того, чтобы доказать, что отношение находится в третьей нормальной форме. Функциональные зависимости обозначены в таблице 3.3.

Таблица 3.3 – Отношение «Stadium»

|  |
| --- |
| Id\* |
| Name |
| Info |
| Country |
| City |

В данном отношении нет транзитивных зависимостей, и оно находится в 3-ей НФ.

1. Отношение «Ticket».

В данном отношении все значения атомарны, следовательно, оно находится в первой нормальной форме. Также в нем простой, а не составной ключ, следовательно, все неключевые атрибуты зависят только от Id. По определению, данное отношение находится во второй нормальной форме. Найдем все функциональные зависимости в этом отношении для того, чтобы доказать, что отношение находится в третьей нормальной форме. Функциональные зависимости обозначены в таблице 3.4.

Таблица 3.4 – Отношение «Ticket»

|  |
| --- |
| Id\* |
| Game\_Id |
| Sector |
| Row |
| Place |
| IsSold |
| Purchase\_Code |

Данное отношение находится в 3-ей НФ. В нём имеются внешние ключи. Это атрибуты «Game\_Id», связанный с атрибутом «Id» отношения «Game», и «Purchase\_Code» связанный с атрибутом «Code» отношения «Purchase».

1. Отношение «Purchase».

В данном отношении всего два атрибута, один из них – первичный ключ. Отношение находится в 3-ей НФ. Функциональные зависимости представлены в таблице 3.5.

Таблица 3.5 – Отношение «Purchase»

|  |
| --- |
| Code \* |
| User\_Id |

В данном отношении имеется внешний ключ. Это атрибут «User\_Id», связанный с атрибутом «Id» отношения «User».

1. отношение «User».

В данном отношении все значения атомарны и ключ простой. По определению, оно находится во 2-ой НФ. Функциональные зависимости в данном отношении имеют вид, представленный в таблице 3.6.

Таблица 3.6 – Отношение «User»

|  |
| --- |
| Id\* |
| Name |
| Surname |
| Login |
| PasswordHash |
| Country |
| City |
| Email |
| Role |

Транзитивных зависимостей в отношении нет, значит, оно находится в 3-ей НФ.

Убедившись, что все таблицы находятся в 3-ей НФ, можно утверждать, что и вся схема базы данных находится в 3-ей НФ.

4 ОПИСАНИЕ ПРОГРАММЫ

4.1 Общие сведения

Данный продукт был реализован в виде веб-сайта, с серверной частью на языке C# с помощью фреймворка ASP.NET MVC и клиентской частью на языке разметки HTML5 с использованием JavaScript и CSS3. Используется СУБД MS SQL Server 2012, доступ к которой осуществляется с помощью Entity Framework и технологии Code First. Также используются библиотеки Bootstrap и JQuery.

Итоговая программа занимает 63 мегабайт. Для запуска программы на персональном компьютере потребуется стандартные устройства ввода-вывода (клавиатура, манипулятор типа «мышь»), процессор, монитор, а также наличие Microsoft Visual Studio, желательно самой последней версии.

Никакой дополнительный сервер для установки связи с базой данных, используемой в приложении, не требуется. Все, что необходимо, уже встроено в программный продукт.

4.2 Вызов и загрузка

Приложение поставляется в папке Competitions.

Внутри папки находится главный файл с расширением .sln, а именно − Competitions.sln, обеспечивающий начальный запуск программы. После запуска данного файла в среде Microsoft Visual Studio необходимо нажать кнопку с зелёным треугольником на верхней панели.

Внутри начальной папки также присутствует папка Competitions. В ней присутствует другая папка App\_Data, в которой содержатся файлы, связанные с базой данных − aspnet-Competitions-20151107043436 .mdf и aspnet-Competitions-20151107043436\_log.ldf.

Подключение базы данных происходит автоматически при старте программы.

4.3 Назначение и логическая структура

При реализации программы «Бронирование билетов на соревнования» за основу была взята трехслойная архитектурная модель MVC (Model-View-Controller). В данной модели классы, связанные с логической обработкой данных, классы, связанные со взаимодействием с пользователем, и классы, отвечающие за визуальное представление данных, физически отделяются друг от друга [4]. Также в качестве четвертого слоя были отделены классы, связанные с доступом к внешним данным.

Данная архитектурная модель является удобной и выгодной. Одно из основных ее достоинств состоит в том, что при правильной реализации модификация одного из компонентов оказывает минимальное воздействие на остальные.

Все классы информационной системы на основании их смысла и назначения разбиты на следующие логические модули:

1. модуль доступа к данным, который включает такие функции, как:
2. сохранение изменений в базе данных (содержится в файле ApplicationDbContext.cs);
3. инициализирование базы данных при первом создании базы данных (содержится в файле DbInitializer.cs).
4. модуль управления учетными записями пользователя, который включает такие функции как:
5. регистрация нового пользователя;
6. вход авторизация пользователя в систему и авторизация его;
7. изменение данных из профиля пользователя.

Реализация данных функций по большей части содержится в следующих файлах:

1. IdentityModel.cs, AccountViewModel.cs, ManageViewModel.cs (модели);
2. ManageController.cs, AccountController.cs (слой контроллеров);
3. ChangePassword.chtml, Index.chtml, Details.chtml, Login.chtml, Register.chtml (слой представления).
4. модуль работы с данными о стадионах, играх и игроках, в основе которого лежат такие функции:
5. добавление, редактирование и удаление стадионов;
6. добавление, редактирование и удаление игроков, а также возможность их поиска по имени, фамилии и стране, фильтрация по мировому рейтингу;
7. добавление, редактирование и удаление игр, автоматическое добавление билетов на новый матч, возможность сортировки по дате, возможность выбора всех игр, уже сыгранных игр или предстоящих игр.

Реализация данных функций по большей части содержится в следующих файлах:

1. Game.cs, Player.cs, Stadium.cs (модель);
2. GameController.cs, PlayerController.cs, StadiumController.cs (слой контроллеров);
3. Слой представления представлен в трёх папках Game, Stadium и Player, в каждой из которых присутствуют следующий представления: Create.chtml, Delete.chtml, Edit.chtml, Details.chtml, Index.chtml.
4. модуль, отвечающий за работу с пользователями, который включает такие функции:
5. отображение списка всех пользователей, поиск пользователя по логину, имейлу и стране, фильтрация по роли;
6. назначение пользователя администратором сайта, разжалование пользователя из администраторов сайта;
7. просмотр профиля пользователя и всех его покупок и заказанных билетов.

За реализацию отвечают следующие файлы:

1. IdentityModel.cs (модель);
2. HomeController.cs (слой контроллеров);
3. ShowUsers.chtml, UsersSearch.chtml, Details.chtml, Index.chtml, PromoteToAdminButton.chtml (слой представления).
4. модуль, отвечающий за задачу автоматизации. Активно взаимодействует с другими модулями.

Основная реализация содержится в файлах:

1. Ticket.cs, Purchase.cs (модель);
2. GameController.cs, HomeController.cs (слой контроллеров);
3. Details.chtml, BookTickets.chtml, FindPurchase.chtml, PurchasePartial.chtml (слой представления).
4. модуль формирования отчетов посредством экспорта данных в файлы .chtml, реализующий следующие функции:
5. формирование отчетов по данным о сыгранных матчах;
6. формирование отчета о купленных билетах;
7. формирование билетов на определённый матч.

За реализацию отвечают следующие файлы:

1. GameController.cs, HomeController.cs (слой контроллеров);
2. PrintTickets.cs, ShowPurchase.chtml, ShowReport.cs (слой представления).
3. модуль, отвечающий за возможность выполнения пользователем произвольного SQL-запроса.

За реализацию отвечают следующие файлы:

1. HomeController.cs (слой контроллеров);
2. MyQuery.chtml, MyQuryPartial.chtml, QueryErrorPartial.chtml (слой представления).

4.4 Описание физической модели базы данных

Схема физической модели базы данных приведена на рисунке 4.1

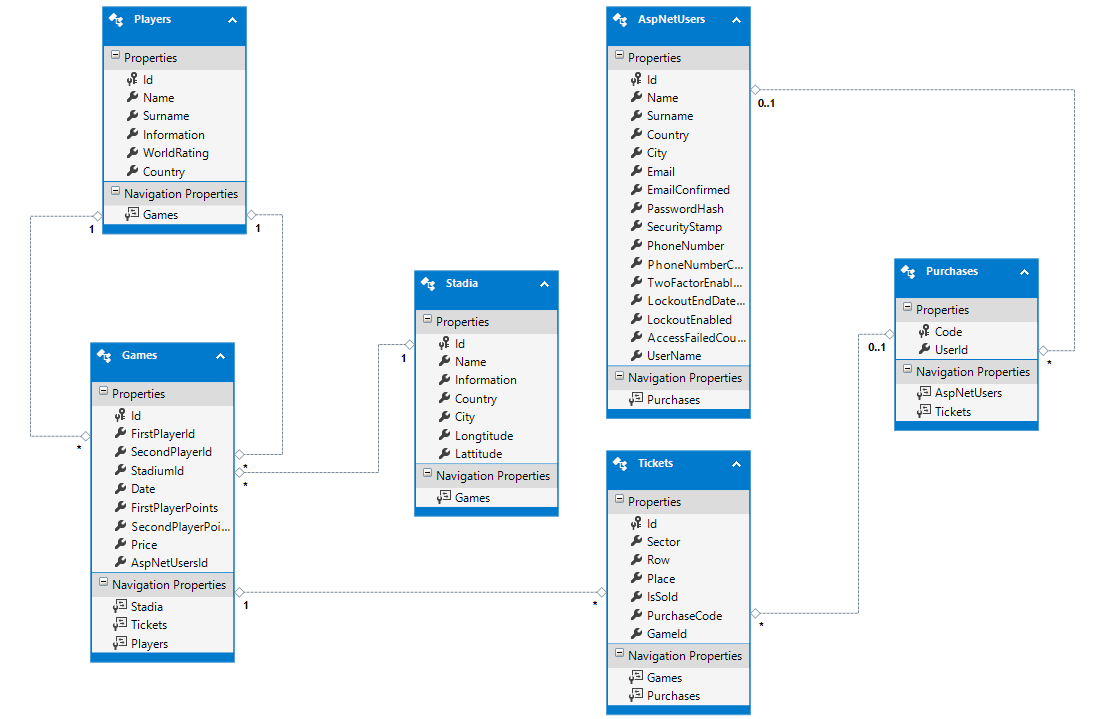


Рисунок 4.1 – Схема физической модели базы данных

Рассмотрим отдельно каждую таблицу из схемы.

Описание таблицы «Players» приведено в таблице 4.1.

Таблица 4.1 – Поля таблицы «Players»

|  |  |  |  |
| --- | --- | --- | --- |
| Поле | Тип данных | Ограничения | Описание |
| Id | INT | PRIMARY KEY, IDENTITY (1, 1) NOT NULL | Идентификатор игрока |
| Name | NVARCHAR (MAX) | NOT NULL | Имя игрока |
| Surname | NVARCHAR (MAX) | NOT NULL | Фамилия пользователя |
| Information | NVARCHAR (MAX) | NULL | Информация об игроке |
| WorldRating | INT | NOT NULL | Мировой рейтинг игрока |
| Country | NVARCHAR (MAX) | NOT NULL | Национальность игрока |

Описание таблицы «Games» приведено в таблице 4.2. Есть внешние ключи [5]: «Games.FirstPlayerId» зависит от «Player.Id», «Games.SecondPlayerId» зависит от «Players.Id», а «Games.StadiumId» зависит от «Stadia.Id».

Таблица 4.2 – Поля таблицы «Games»

|  |  |  |  |
| --- | --- | --- | --- |
| Поле | Тип данных | Ограничения | Описание |
| Id | INT | PRIMARY KEY, NOT NULL, IDENTITY (1, 1) | Идентификатор игры |
| FirstPlayerId | INT | NOT NULL | Идентификатор первого игрока, участвующего в данном матче |
| SecondPlayerId | INT | NOT NULL | Идентификатор второго игрока, участвующего в данном матче |
| StadiumId | INT | NOT NULL | Идентификатор стадиона, на котором проходит матч |
| Date | DATETIME | NOT NULL | Дата и время проведения матча |
| FirstPlayerPoints | INT | NOT NULL | Очки, набранные первым игроком |
| SecondPlayerPoints | INT | NOT NULL | Очки, набранные вторым игроком |
| Price | INT | NOT NULL | Цена билетов на матч |

Описание таблицы «Stadia» приведено в таблице 4.3.

Таблица 4.3 – Поля таблицы «Stadia»

|  |  |  |  |
| --- | --- | --- | --- |
| Поле | Тип данных | Ограничения | Описание |
| Id | INT | PRIMARY KEY, NOT NULL, IDENTITY (1, 1) | Идентификатор стадиона |
| Name | NVARCHAR(MAX) | NOT NULL | Название стадиона |
| Information | NVARCHAR(MAX) | NOT NULL | Дополнительная информация о стадионе |
| Country | NVARCHAR(MAX) | NULL | Страна, в которой находится стадион |
| City | NVARCHAR(MAX) | NULL | Город, в котором находится стадион |
| Longtitude | NVARCHAR(MAX) | NULL | Долгота стадиона на карте |
| Lattitude | NVARCHAR(MAX) | NULL | Широта стадиона на карте |

Описание таблицы «Ticket» приведено в таблице 4.4. Есть внешние ключи: «Ticket.PurchaseCode» зависит от «Purchase.Code», а «Ticket.GameId» зависит от «Game.Id».

Таблица 4.4 – Поля таблицы «Ticket»

|  |  |  |  |
| --- | --- | --- | --- |
| Поле | Тип данных | Ограничения | Описание |
| Id | INT | PRIMARY KEY, NOT NULL, IDENTITY (1, 1) | Идентификатор билета |
| Sector | INT | NOT NULL | Сектор, в котором находится место |
| Row | INT | NOT NULL | Ряд, в котором находится место |
| Place | INT | NOT NULL | Номер сидения места |
| IsSold | BIT | NOT NULL | Данный атрибут показывает, продан ли билет |
| PurchaseCode | NVARCHAR(6) | NULL | Код покупки, к которой принадлежит билет |
| GameId | INT | NOT NULL | Идентификатор игры, к которой относится данный билет |

Описание таблицы «Purchases» приведено в таблице 4.5. Есть внешний ключ «Purchases.UserId», который зависит от «AspNetUsers.Id».

Таблица 4.5 – Поля таблицы «Purchases»

|  |  |  |  |
| --- | --- | --- | --- |
| Поле | Тип данных | Ограничения | Описание |
| Code | NVARCHAR(6) | PRIMARY KEY, NOT NULL | Код покупки |
| UserId | NVARCHAR(128) | NOT NULL | Идентификатор пользователя, которому принадлежит данная покупка |

Описание таблицы «AspNetUsers» приведено в таблице 4.6.

Таблица 4.6 – Поля таблицы «AspNetUsers»

|  |  |  |  |
| --- | --- | --- | --- |
| Поле | Тип данных | Ограничения | Описание |
| Id | NVARCHAR(128) | PRIMARY KEY, NOT NULL | Идентификатор пользователя |
| Name | NVARCHAR(MAX) | NULL | Имя пользователя |
| Country | NVARCHAR(MAX) | NULL | Страна проживания пользователя |
| City | NVARCHAR(MAX) | NULL | Город проживания пользователя |
| Email | NVARCHAR(256) | NULL | E-mail пользователя |
| PasswordHash | NVARCHAR(MAX) | NOT NULL | Захешированый пароль пользователя |
| UserName | NVARCHAR(256) | NOT NULL | Логин пользователя |

4.5 Описание программной реализации

При начальном запуске открывается главное страница сайта (рисунок 4.2). На данном окне мы можем зарегистрироваться или авторизоваться, нажав на кнопку Register или Log in в правой верхней части страницы. Также мы можем просмотреть информацию об игроках, играх и стадионах не зарегистрировавшись.

Заметим, что благодаря панели быстрого доступа вверху страницы возможна быстрая навигация по сайту.

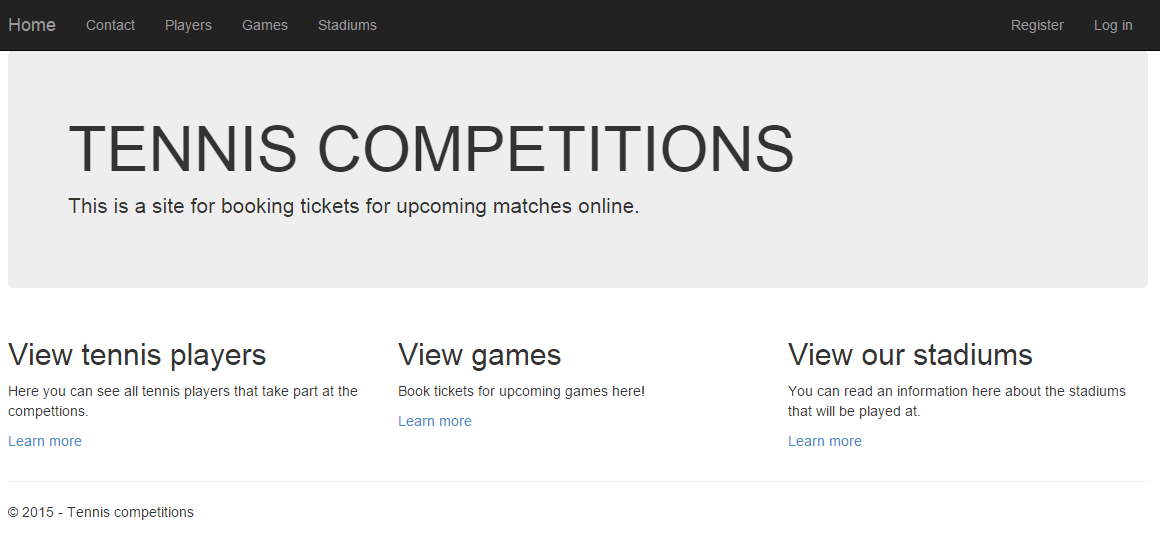


Рисунок 4.2 – Главная страница

При нажатии на кнопку Register перед нами появится форма, представленная на рисунке 4.3.

Заметим, что все обязательные для заполнения поля помечены звёздочкой. Поле City будет заблокировано, пока пользователь не введёт страну проживания. После нажатия кнопки Register данные о новом пользователе сохраняются в базе данных и его переносит на главную страницу. Если данные были введены неправильно, то пользователь будет уведомлён об этом с просьбой ввести корректные данные.

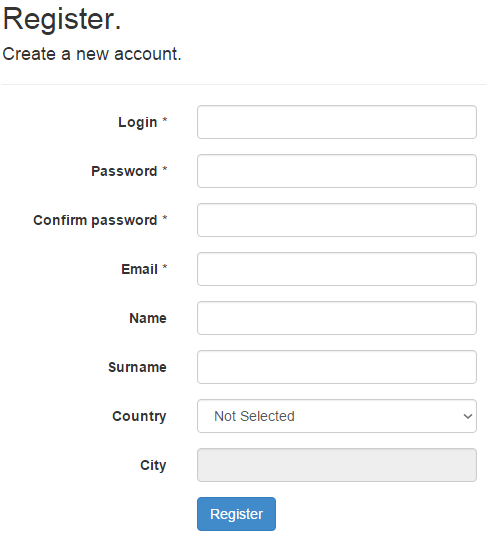


Рисунок 4.3 – Форма регистрации

Далее рассмотрим форму авторизации (рисунок 4.4).

Для того, чтобы авторизироваться, пользователю необходимо ввести логин и пароль после чего его перенесёт на главную страницу. Также на этой форме есть ссылка на страницу регистрации. При нажатии на неё пользователь перейдёт на страницу, изображённую на рисунке 4.3. Если поставить галочку в пункте Remember me, то браузер запомнит логин и пароль, введённые пользователем, поэтому при последующей авторизации эти поля будут уже автоматически заполнены.

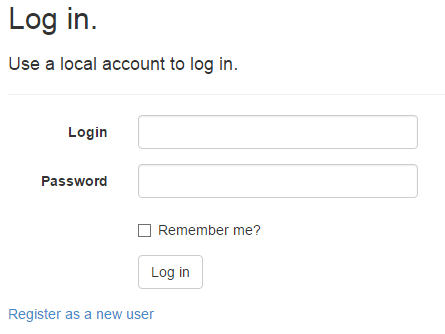


Рисунок 4.4 – Форма авторизации

Ели пользователь является администратором сайта, то панель быстрого доступа выглядит так, как показано на рисунке 4.5:



Рисунок 4.5 – Панель быстрого доступа администратора

На данной панели мы имеем доступ к следующим функциям: переход на главную страницу (кнопка Home), связь с разработчиком (кнопка Contact), работа с игроками, участвующими в соревнованиях (кнопка Players), работа с играми, которые проходят в рамках данных соревнований (кнопка Games), работа со стадионами, на которых проходят игры для соревнований (кнопка Stadiums), работа с пользователями данного сайта (кнопка Users), возможность написать свой собственный запрос к базе данных (кнопка MyQuery), поиск покупки, произведённой пользователем (кнопка Find purchase). При нажатии на кнопку Hello пользователь перейдёт к настройкам собственного профиля, а при нажатии на кнопку Log off он выйдет из системы. Рассмотрим каждую из этих функций в отдельности.

При нажатии на кнопку Players перед нами появится страница, представленная на рисунке 4.6. Здесь выводится информация о всех игроках, участвующих в турнире, а именно: имя, фамилия, мировой рейтинг и страна.

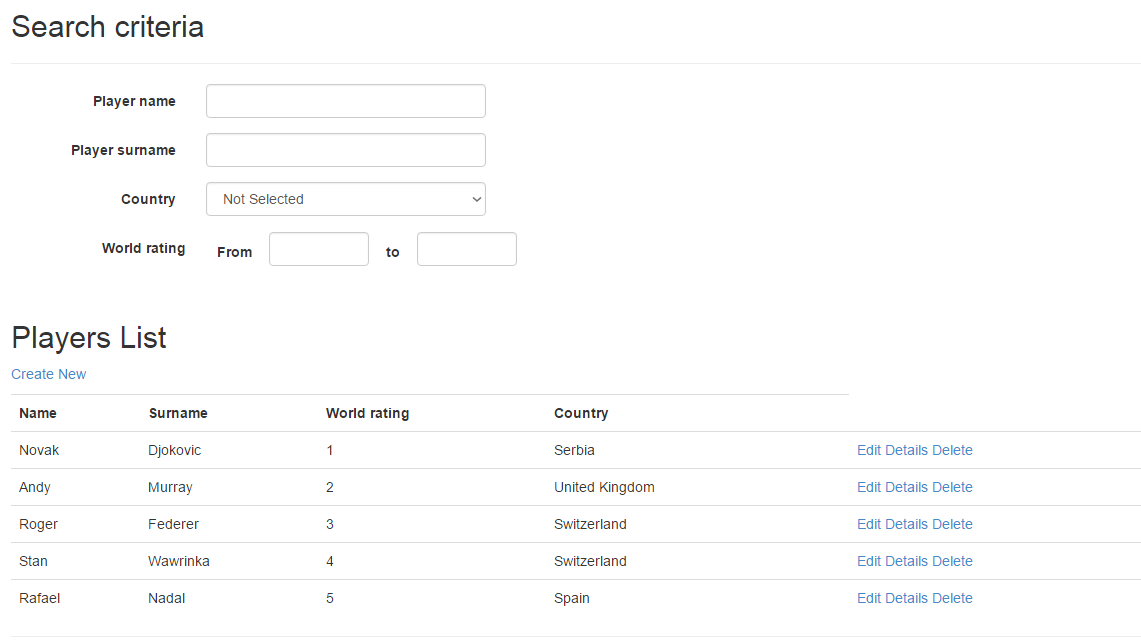


Рисунок 4.6 – Список всех игроков

На данной странице мы можем производить динамический поиск игроков по имени, фамилии и стране. Кроме того, доступна фильтрация по мировому рейтингу. Если все поля оставить незаполненными, то будут выведены все игроки, участвующие в соревнованиях. Мы можем добавить нового игрока, перейдя по ссылке Create new. В результате перед нами появится форма для заполнения информации о новом игроке (рисунок 4.7). Для создания нового игрока пользователь должен ввести его имя, фамилию, мировой рейтинг и страну. Вводить информацию о новом игроке необязательно.

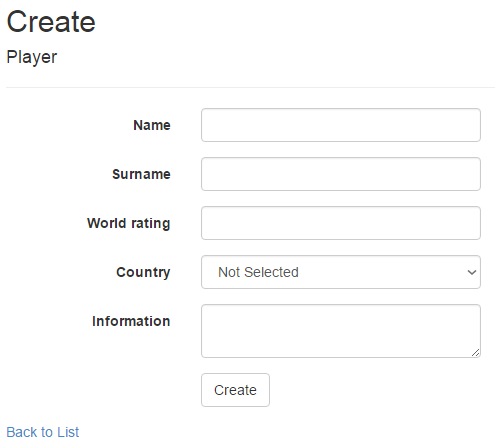


Рисунок 4.7 – Форма добавления нового игрока

В случае некорректного ввода данных пользователь будет уведомлён об этом (рисунок 4.8). Проверка на правильность заполнения полей есть во всех формах добавления и изменения информации данного веб-сайта. Стоит заметить, что уже введённая информация в полях не удаляется, а запоминается.

После нажатия кнопки Create игрок будет добавлен в базу данных. При нажатии ссылки Back to list пользователь просто вернётся к списку всех игроков, причём данные, введённые в форме добавления игрока не сохранятся.

При просмотре списка игроков пользователь также может просматривать информацию о каждом игроке отдельно, нажав на ссылку Details рядом с ним. В результате перед пользователем появится страница, представленная на рисунке 4.9.

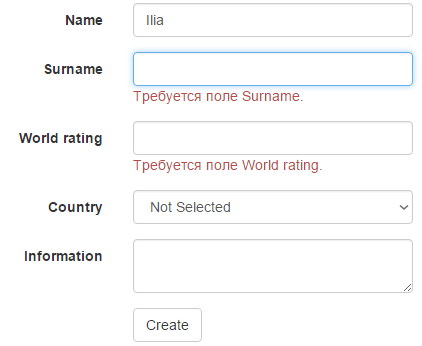


Рисунок 4.8 – Уведомление об ошибках

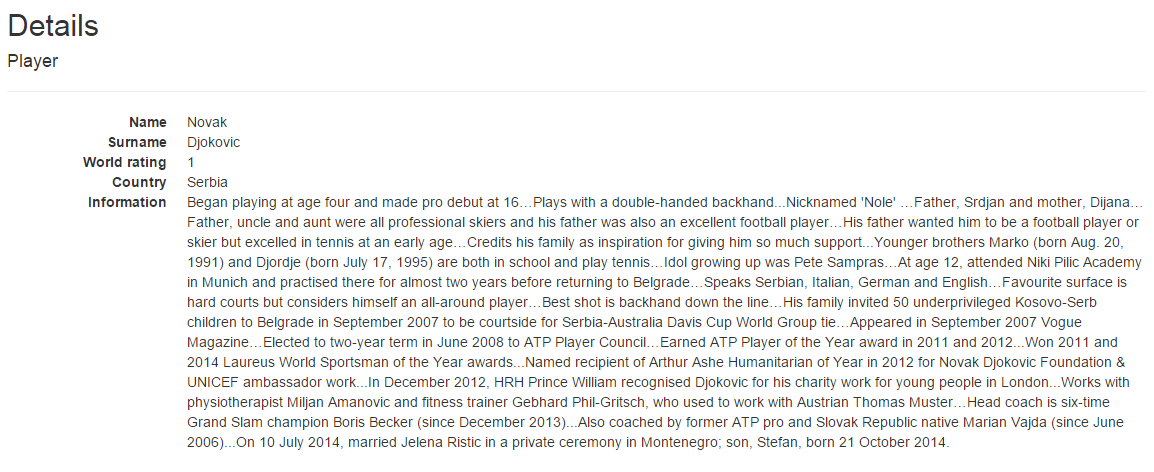


Рисунок 4.9 – Информация об игроке

При просмотре списка игроков администратор также может удалить игрока, перейдя по ссылке Delete или изменить информацию о нём, перейдя по ссылке Edit. В результате перед пользователем появится форма, представленная на рисунке 4.10.

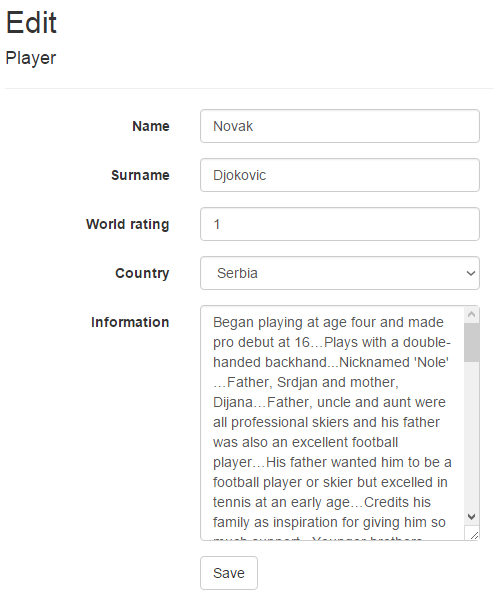


Рисунок 4.10 – Изменение информации об игроке

Работа со стадионами абсолютна аналогична работе с игроками. На рисунке 4.11 представлена страница со списком всех существующих стадионов.

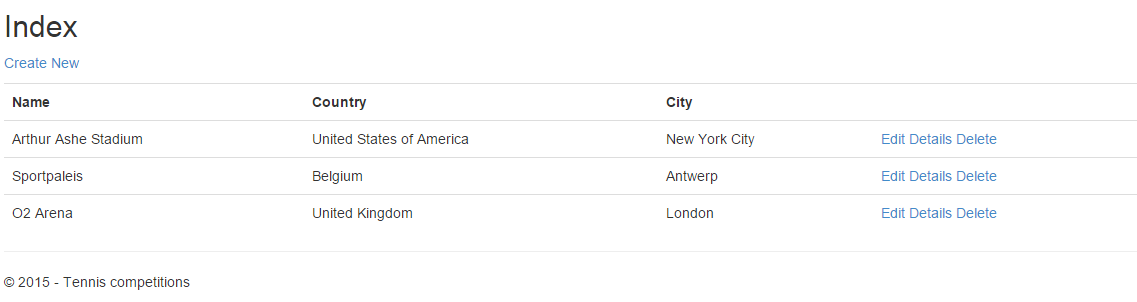


Рисунок 4.11 – Список всех стадионов

Далее представлена форма добавления нового стадиона (рисунок 4.12). Заметим, что здесь необходимо указать месторасположение данного стадиона на карте, которое потом будет сохранено в виде полей Longtitude и Lattitude.

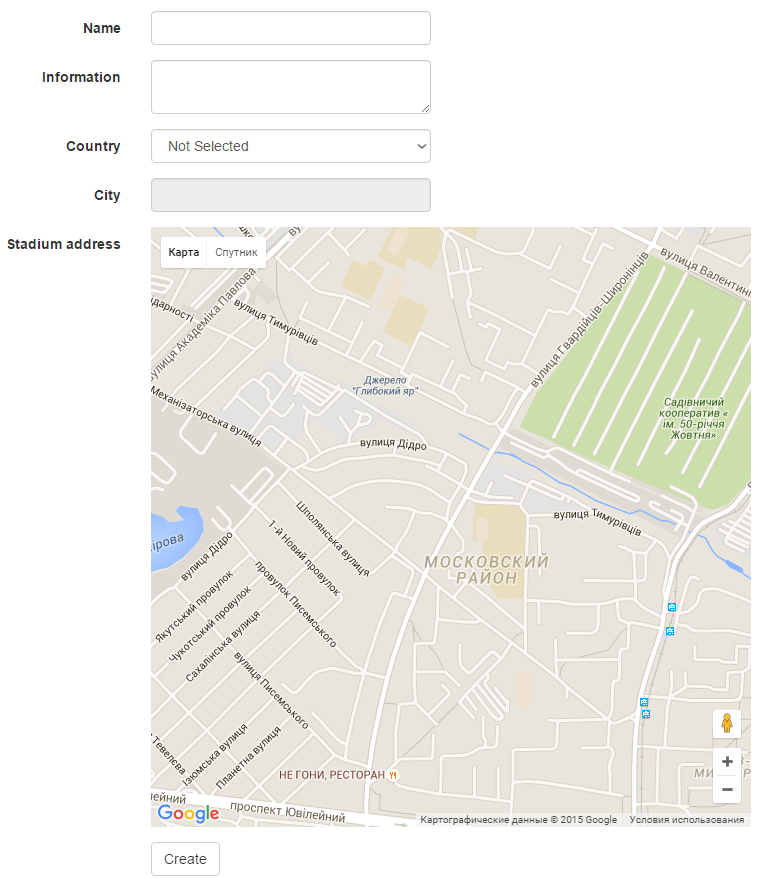


Рисунок 4.12 – Форма добавления нового стадиона

На рисунке 4.13 представлена страница, на которой пользователь может просматривать информацию о стадионе. Здесь он может посмотреть название стадиона, его описание, город и страну, в которой он находится. Кроме того, пользователю предлагается посмотреть месторасположение данного стадиона на Google-карте.

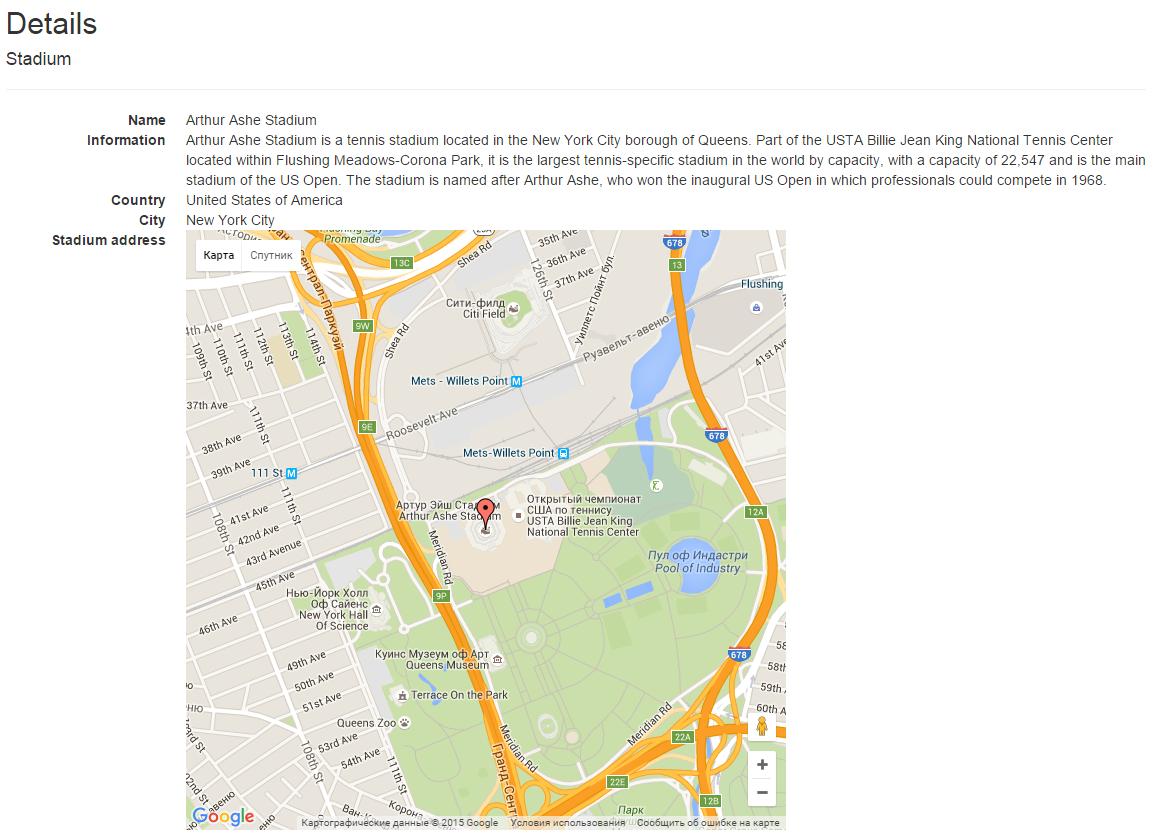


Рисунок 4.13 – Просмотр информации о стадионе

На рисунке 4.14 представлена страница, на которой администратор может редактировать информацию о стадионе. Стоить отметить, что он не может удалить маркер, поставленный на карте, а может только переместить его. Если пользователь поставит страну стадиона в состояние Not selected, то поле автоматически очистится и заблокируется до того момента, пока пользователь не выберет страну, в которой находится стадион.

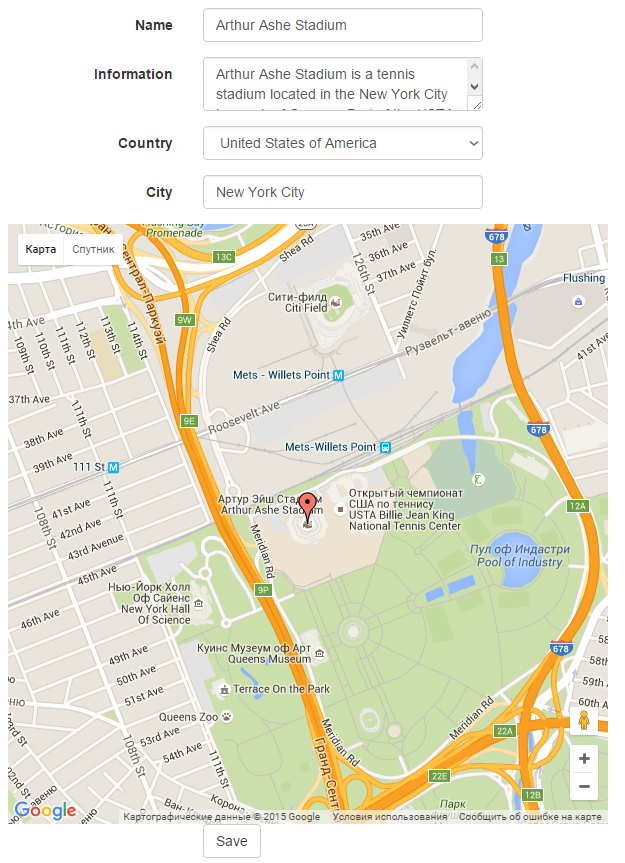


Рисунок 4.14 – Редактирование информации о стадионе

При нажатии на панели быстрого доступа кнопки Games пользователь попадает на страницу со списком всех игр данных соревнований (рисунок 4.15).

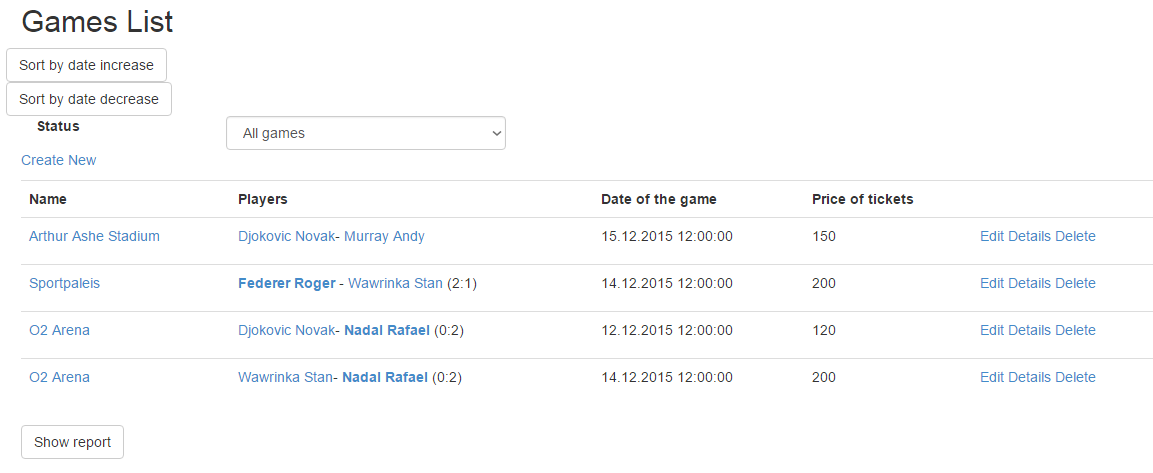


Рисунок 4.15 – Список всех игр

Обратим внимание на сам список игр. Если игра уже сыграна, то справа от фамилий указан счёт, с которым закончилась встреча, а игрок-победитель выделен жирным. При нажатии на фамилию игрока мы перейдём на страницу с его описанием. На данной странице мы можем отсортировать все игры по возрастанию или убыванию даты (кнопки Sort by date increase и Sort by date decrease соответственно). Кроме того, здесь есть встроенные запросы, которые могут показать все игры, только сыгранные игры или предстоящие игры. Данные запросы на языке SQL представляются в следующем виде:

а) вывод всех игр:

SELECT Name, FirstPlayerId, SecondPlayerId, FirstPlayerPoints, SecondPlayerPoints, Date, Price

FROM Players;

б) вывод сыгранных игр:

SELECT Name, FirstPlayerId, SecondPlayerId, FirstPlayerPoints, SecondPlayerPoints, Date, Price

FROM Players

WHERE NOT FirstPlayerPoints = SecondPlayerPoints;

в) вывод предстоящих игр:

SELECT Name, FirstPlayerId, SecondPlayerId, FirstPlayerPoints, SecondPlayerPoints, Date, Price

FROM Players

WHERE FirstPlayerPoints = SecondPlayerPoints;

При нажатии кнопки ShowReport, нам будет показана статистика всех уже сыгранных игр (рисунок 4.16).

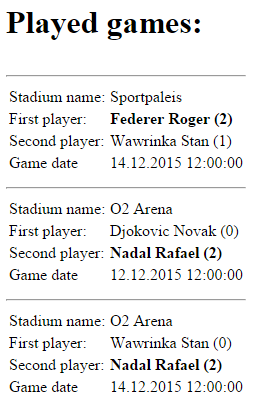


Рисунок 4.16 – Отчёт о сыгранных играх

Пользователь может просматривать информацию о каждой игре в форме, представленной на рисунке 4.17. Заметим, что игроки, участвующие в игре, являются гиперссылками, при нажатии на которые пользователь перейдёт на страницу просмотра этих игроков. Аналогично является гиперссылкой и текст, являющийся названием стадиона.

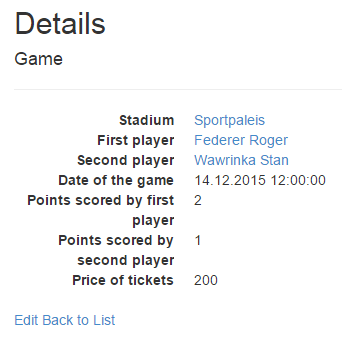


Рисунок 4.17 – Просмотр информации о сыгранном матче

Если игра не сыграна, то пользователь может просматривать свободные места на этот матч и бронировать их (рисунок 4.18).

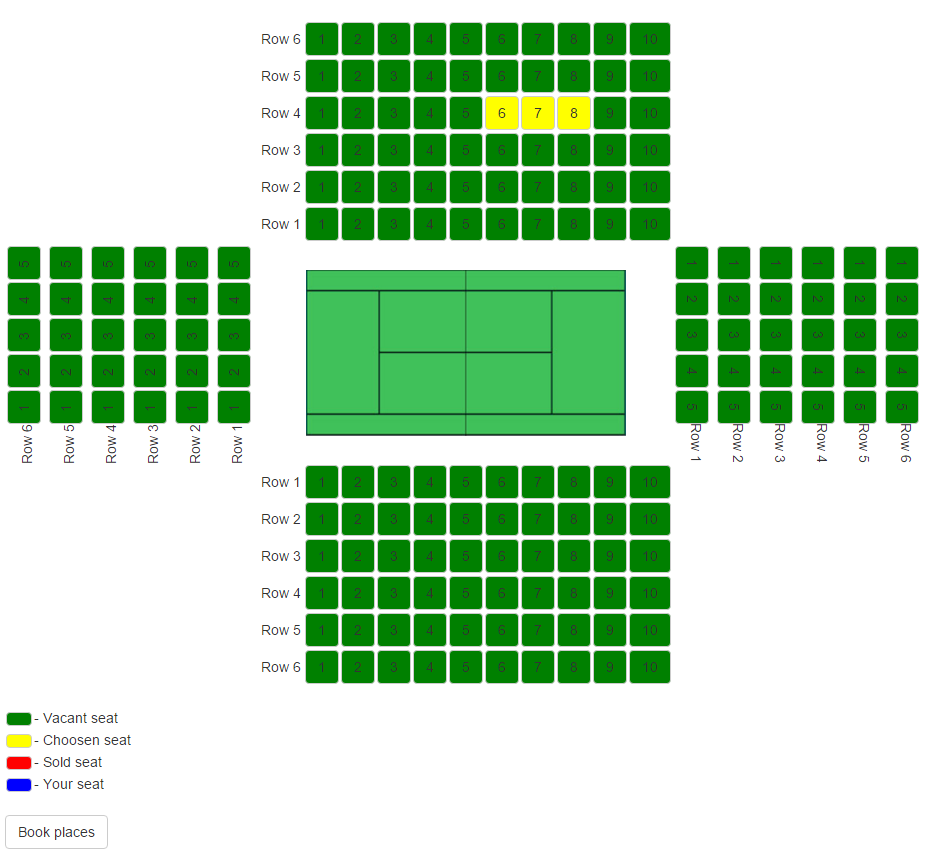


Рисунок 4.18 – Бронирование билетов онлайн

На рисунке 4.19 представлена форма добавления новой игры. Заметим, что в случае выбора одинаковых игроков, пользователю придёт уведомление об ошибке. Кроме того, пользователь не сможет ввести итоговый счёт матча, если дата матч ещё не прошёл.

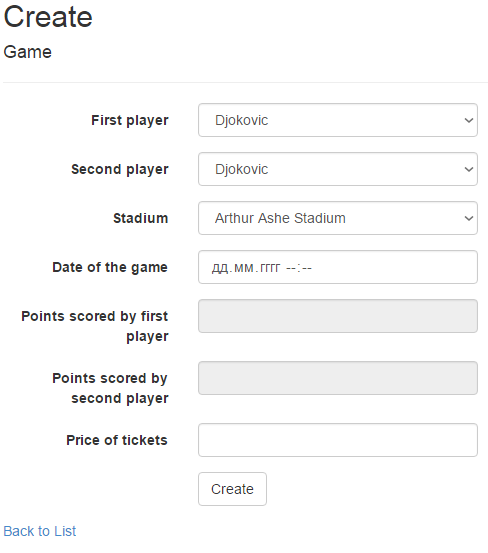


Рисунок 4.19 – Форма добавления новой игры

После добавления новой игры автоматически генерируются билеты на данную игру.

На рисунке 4.20 показана форма редактирования информации об игре.

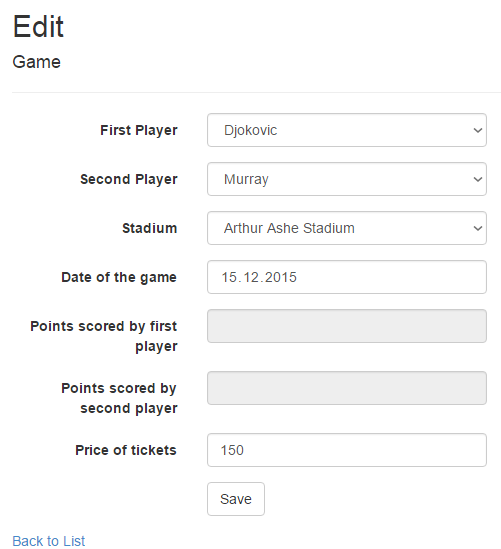


Рисунок 4.20 – Форма изменения информации об игре

Все последующие функции данной программы доступны только администратору сайта. При нажатии на кнопку Users в панели быстрого доступа программы перед нами появляется список всех пользователей данной сайта (рисунок 4.21). На данной странице есть динамически й поиск по логину, имейлу и стране проживания пользователя. Если мы хотим показать всех пользователей, то необходимо оставить все поля незаполненными. Кроме того, есть возможность показать всех пользователей, только администраторов или только не администраторов. Каждого пользователя, который является администратором, можно разжаловать из администраторов. И наоборот, каждого обычно пользователя можно сделать администратором.

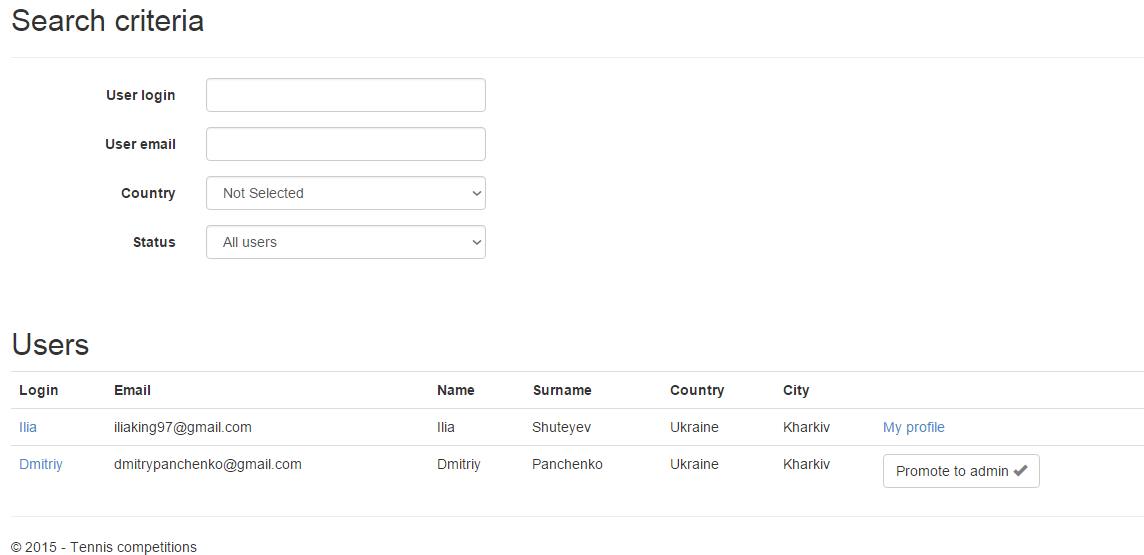


Рисунок 4.21 – Список пользователей

Также можно посмотреть информацию о пользователе в виде, представленном на рисунке 4.22.

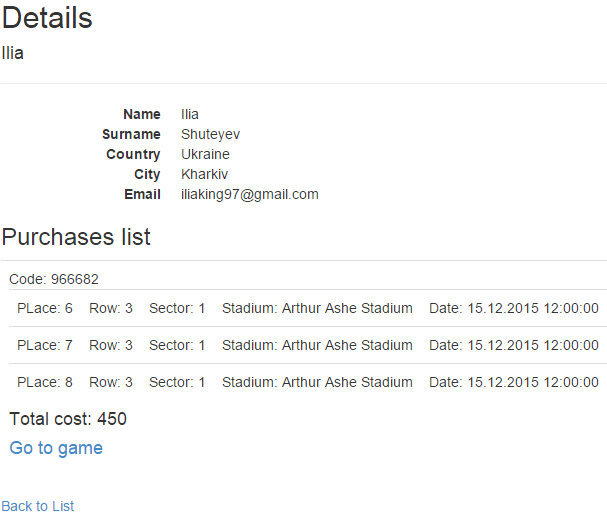


Рисунок 4.22 – Информация о пользователе

При нажатии кнопки MyQuery на панели быстрого доступа пользователь переносится на страницу, на которой он может ввести свой произвольный запрос к БД (рисунок 4.23).

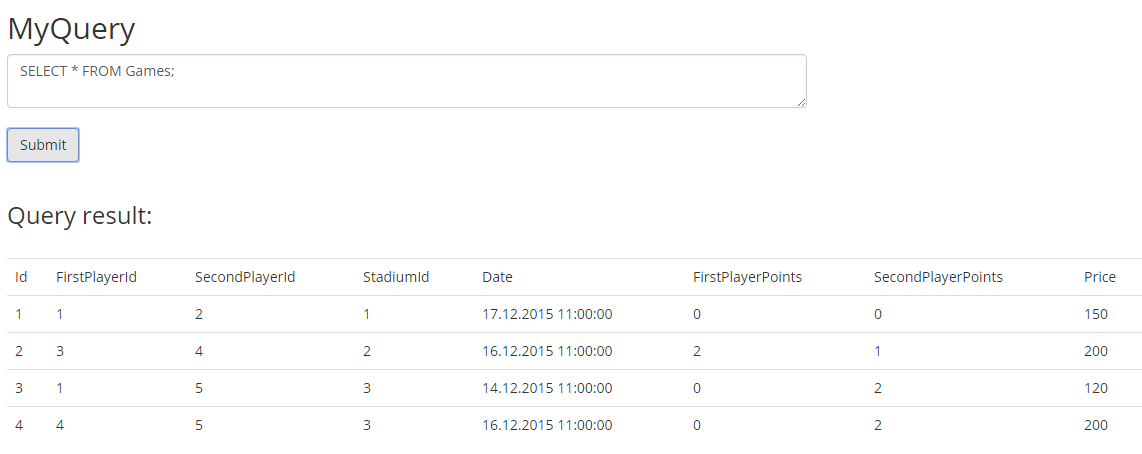


Рисунок 4.23 – Произвольный запрос

Последней функцией является просмотр своего профиля и редактирование информации о себе (рисунок 4.24). На данной странице пользователь также может изменить свой пароль и посмотреть все свои покупки.

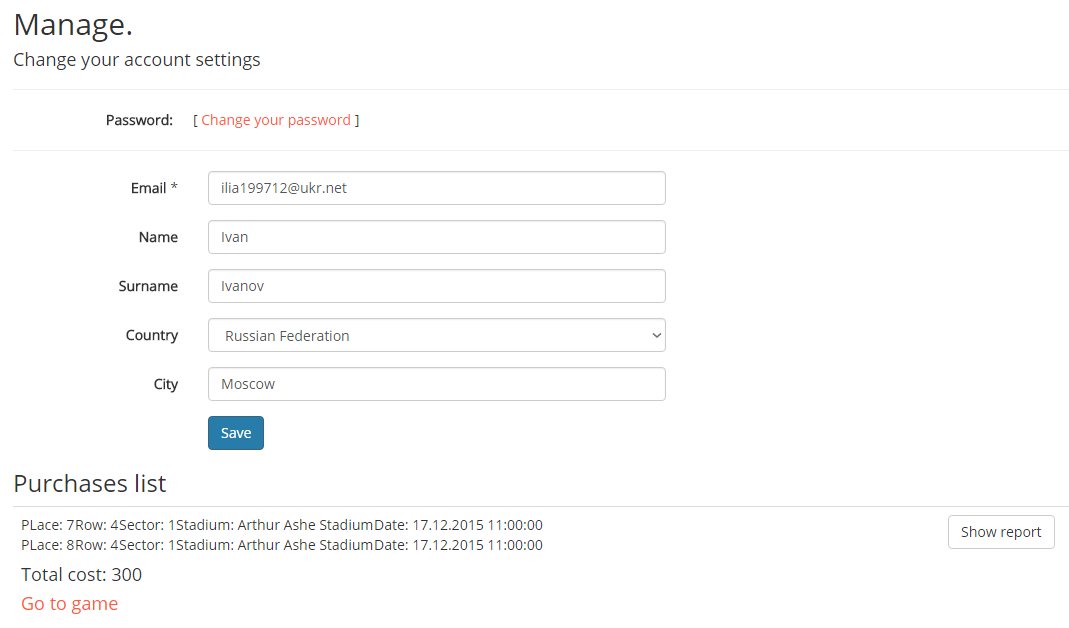


Рисунок 4.24 – Редактирование профиля

4.6 Описание задачи автоматизации

Чтобы помочь пользователю удобно выбрать места на интересующую его игру заранее, была введена задача автоматизации бронирования билетов на матч. Рассмотрим данную задачу более подробно.

На форме просмотра информации об игре пользователь может посмотреть схему стадиона, увидеть занятые, свободные и забронированные им билеты (рисунок 4.18). При нажатии на свободное место, выделенное зелёным, оно станет жёлтым, что означает, что пользователь выбрал данное место. При нажатии на красное, уже занятое место ничего не произойдёт. После того, как пользователь выберет все интересующие его места, он должен нажать кнопку Book places. В результате пользователя перебросит на страницу, представленную на рисунке 4.25. На данной странице пользователя просят окончательно подтвердить бронирование и напоминается вся информация о местах, которые он заказал, а именно: стадион, дата матча, сектор, ряд и место для каждого билета в отдельности. Также написано, что после подтверждения на имейл пользователя будет выслан секретный код, который он должен будет предъявить на кассе, когда придёт забрать забронированные билеты.

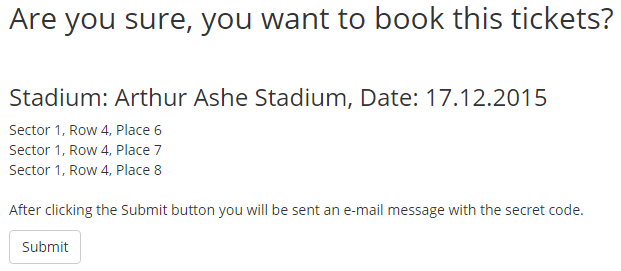


Рисунок 4.25 – Подтверждение бронирования билетов

После нажатия кнопки Sumbit пользователь переходит на страницу просмотра информации о матче. Заметим, что теперь заказанные билеты выделены синим (рисунок 4.26).

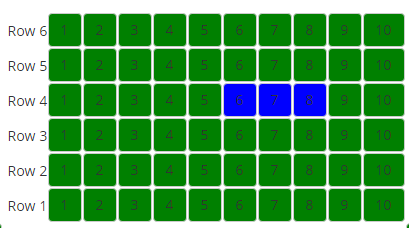


Рисунок 4.26 – Просмотр забронированных билетов

При нажатии на забронированное место перед пользователем появится окно с просьбой подтвердить отмену бронирования (рисунок 4.27).

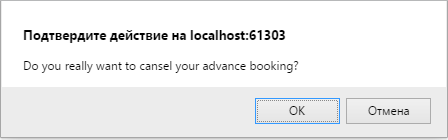


Рисунок 4.27 – Подтверждение отмены бронирования

В результате место становится зелёным и его опять можно бронировать (рисунок 4.28).

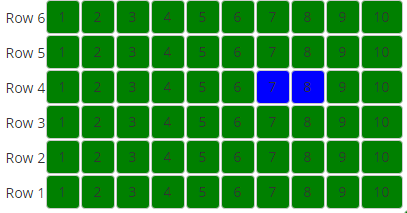


Рисунок 4.28 – Просмотр забронированных билетов

Как уже отмечалось ранее, все свои забронированные билеты пользователь может посмотреть в своём профиле. После бронирования билетов пользователь должен зайти на почту и прочитать письмо, в котором указаны места и секретный код (рисунок 4.29).

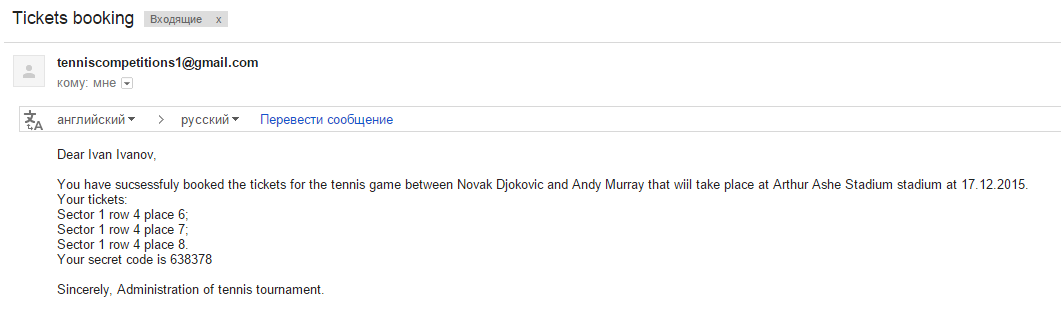


Рисунок 4.29 – Уведомление об успешном бронировании

В данном письме указывается секретный код, который пользователь должен предъявить на кассе. В свою очередь, кассир, при выдаче забронированных билетов должен зайти на сайт как администратор и по коду, предъявленному заказчиком выдать ему билеты. Для этого он должен в панели быстрого доступа зайти на вкладку Find purchase, в поле Secret code ввести код и нажать кнопку Search. Результат показан на рисунке 4.30.

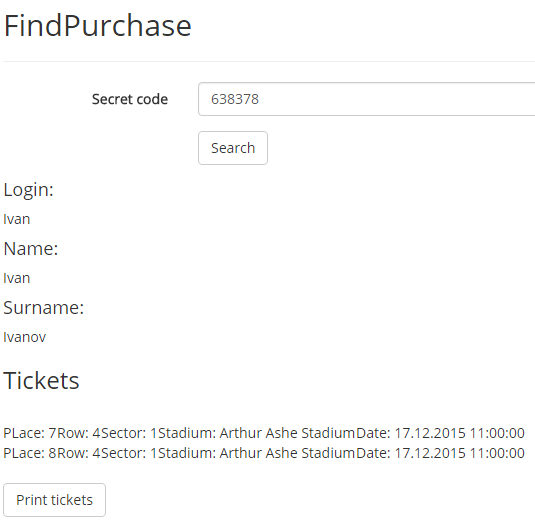


Рисунок 4.30 – Поиск покупки пользователя

В результате появятся все билеты данной заказа, имя, фамилия и логин пользователя, совершившего заказ. При нажатии кнопки Print tickets кассир сможет распечатать билеты (рисунок 4.31).

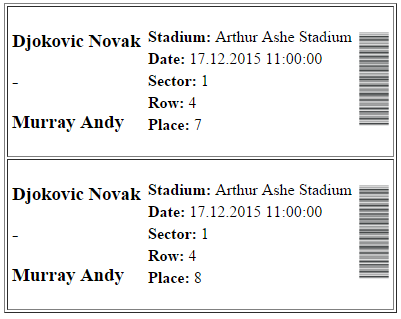


Рисунок 4.31 – Билеты на матч

Далее приведён фрагмент HTML и JavaScript кода, отвечающих за отображение схемы стадиона перед пользователем:

@if (DateTime.Now <= Model.Date)

{

<div>

<h4>Places</h4>

@{

List<Ticket> firstSectorTickets = Model.Tickets.Where(t => t.Sector == 1).ToList();

List<Ticket> secondSectorTickets = Model.Tickets.Where(t => t.Sector == 2).ToList();

List<Ticket> thirdSectorTickets = Model.Tickets.Where(t => t.Sector == 3).ToList();

List<Ticket> forthSectorTickets = Model.Tickets.Where(t => t.Sector == 4).ToList();

}

<br>

@if (User.Identity.IsAuthenticated)

{

@Html.ValidationSummary(true, "", new { @class = "text-danger" })

<br>

<div id="seats">

<table>

<tr>

<td></td>

<td>

<table>

@for (int i = 5; i >= 0; i--)

{

List<Ticket> rowTickets = firstSectorTickets.Where(t => t.Row == i).ToList();

<tr><td><h5>Row @(i + 1) </h5></td>

<td>

@for (int j = 0; j < 10; j++)

{

<td>

@{

Ticket ticket = rowTickets.First(rt => rt.Place == j);

}

<div id="bookTicket @ticket.Id">

@if (ticket.Purchase != null && ticket.Purchase.UserId == currentUser.Id)

{

<input type="hidden" name="blue" />

<div class="btn btn-default" style="background-color:blue" onclick="BlueButtonClick(@ticket.Id)">@(j + 1)</div>

}

else if (ticket.IsSold || ticket.Purchase != null)

{

<div class="btn btn-default" style="background-color:red">@(j + 1)</div>

}

else

{

<div id="button @ticket.Id" class="btn btn-default" style="background-color:green" onclick="buttonClick(@ticket.Id)">@(j + 1)</div>

}

</div>

</td>

}

</tr>

}

</table>

</td>

<td></td>

</tr>

<tr><td>

<table>

@for (int i = 4; i >= 0; i--)

{

List<Ticket> rowTickets = secondSectorTickets.Where(t => t.Place == i).ToList();

<tr>

@for (int j = 0; j < 6; j++)

{

<td>

@{

Ticket ticket = rowTickets.First(rt => rt.Row == j);

}

@if (ticket.Purchase != null && ticket.Purchase.UserId == currentUser.Id)

{

<input type="hidden" name="blue" />

<div class="btn btn-default" style="background-color:blue; transform:rotate(-90deg)" onclick="BlueButtonClick(@ticket.Id)">@(i + 1)</div>

}

else if (ticket.IsSold || ticket.Purchase != null)

{

<div class="btn btn-default" style="background-color:red; transform:rotate(-90deg)">@(i + 1)</div>

}

else

{

<div id="button @ticket.Id" class="btn btn-default" style="background-color:green; transform:rotate(-90deg)" onclick="buttonClick(@ticket.Id)">@(i + 1)</div>

}

</td>

}

</tr>

}

<tr><td><h5 style="transform:rotate(-90deg)">

Row 6

</h5></td>

<td><h5 style="transform:rotate(-90deg)">

Row 5

</h5>

</td>

<td><h5 style="transform:rotate(-90deg)">

Row 4

</h5></td>

<td>

<h5 style="transform:rotate(-90deg)">

Row 3

</h5>

</td>

<td>

<h5 style="transform:rotate(-90deg)">

Row 2

</h5>

</td>

<td>

<h5 style="transform:rotate(-90deg)">

Row 1

</h5>

</td>

</tr>

</table>

</td>

<td align="center">

<img src="~/Pictures/cort.png" width="320" height="166" />

</td>

<td>

<table>

@for (int i = 0; i < 5; i++)

{

List<Ticket> rowTickets = thirdSectorTickets.Where(t => t.Place == i).ToList();

<tr>

@for (int j = 0; j < 6; j++)

{

<td>

@{

Ticket ticket = rowTickets.First(rt => rt.Row == j);

}

@if (ticket.Purchase != null && ticket.Purchase.UserId == currentUser.Id)

{

<input type="hidden" name="blue" />

<div class="btn btn-default" style="background-color:blue; transform:rotate(90deg)" onclick="BlueButtonClick(@ticket.Id)">@(i + 1)</div>

}

else if (ticket.IsSold || ticket.Purchase != null)

{

<div class="btn btn-default" style="background-color:red; transform:rotate(90deg)">@(i + 1)</div>

}

else

{

<div id="button @ticket.Id" class="btn btn-default" style="background-color:green; transform:rotate(90deg)" onclick="buttonClick(@ticket.Id)">@(i + 1)</div>

}

</td>

}

</tr>

}

<tr>

<td>

<h5 style="transform:rotate(90deg)">

Row 1

</h5>

</td>

<td>

<h5 style="transform:rotate(90deg)">

Row 2

</h5>

</td>

<td>

<h5 style="transform:rotate(90deg)">

Row 3

</h5>

</td>

<td>

<h5 style="transform:rotate(90deg)">

Row 4

</h5>

</td>

<td>

<h5 style="transform:rotate(90deg)">

Row 5

</h5>

</td>

<td>

<h5 style="transform:rotate(90deg)">

Row 6

</h5>

</td>

</tr>

</table>

</td>

</tr>

<tr>

<td></td>

<td>

<table>

@for (int i = 0; i < 6; i++)

{

List<Ticket> rowTickets = forthSectorTickets.Where(t => t.Row == i).ToList();

<tr>

<td>

<h5>

Row @(i + 1)

</h5>

</td>

<td>

@for (int j = 0; j < 10; j++)

{

<td>

@{

Ticket ticket = rowTickets.First(rt => rt.Place == j);

}

@if (ticket.Purchase != null && ticket.Purchase.UserId == currentUser.Id)

{

<input type="hidden" name="blue" />

<div class="btn btn-default" style="background-color:blue" onclick="BlueButtonClick(@ticket.Id)">@(j + 1)</div>

}

else if (ticket.IsSold || ticket.Purchase != null)

{

<div class="btn btn-default" style="background-color:red">@(j + 1)</div>

}

else

{

<div id="button @ticket.Id" class="btn btn-default" style="background-color:green" onclick="buttonClick(@ticket.Id)">@(j + 1)</div>

}

</td>

}

</tr>

}

</table>

</td>

<td></td>

</tr>

</table>

<br />

<table>

<tr>

<td><div class="btn btn-default" style="background-color:green"></div></td>

<td> - Vacant seat</td>

</tr>

<tr>

<td><div class="btn btn-default" style="background-color:yellow"></div></td>

<td> - Choosen seat</td>

</tr>

<tr>

<td><div class="btn btn-default" style="background-color:red"></div></td>

<td> - Sold seat</td>

</tr>

<tr>

<td><div class="btn btn-default" style="background-color:blue"></div></td>

<td> - Your seat</td>

</tr>

</table>

<br />

</div>

using (Html.BeginForm("BookTickets", "Game", FormMethod.Get))

{

<input type="hidden" id="boughtTickets" name="boughtTickets" value="" />

<button name="submit" id="submit" type="submit" class="btn btn-default">Book places</button>

}

}

else

{

<h3>To book the tickets, please, @Html.ActionLink("Login", "Login", "Account") or @Html.ActionLink("Register", "Register", "Account")</h3>

}

</div>

}

@using (Html.BeginForm("UnBookTicket", "Game", FormMethod.Get))

{

<input type="hidden" id="blueButton" name="blueButton" />

<input style="visibility:hidden" id="sub" name="submit" type="submit">

}

@section scripts{

<script src="~/Scripts/jquery.unobtrusive-ajax.min.js" type="text/javascript"></script>

}

<script>

var boughtTickets = [];

var boughtTicketCount = document.getElementsByName("blue").length;

function buttonClick(x) {

if (document.getElementById("button " + x).style.backgroundColor == "green") {

if (boughtTicketCount < 10) {

document.getElementById("button " + x).style.backgroundColor = "yellow";

color = "yellow";

boughtTickets[boughtTickets.length] = x;

var s = "";

for (var i = 0; i < boughtTickets.length; i++) {

s += boughtTickets[i] + " ";

}

document.getElementById("boughtTickets").value = s;

boughtTicketCount++;

}

else

{

alert("You have alredy booked 10 tickets.")

}

}

else if (document.getElementById("button " + x).style.backgroundColor == "yellow") {

document.getElementById("button " + x).style.backgroundColor = "green";

color = "green";

boughtTickets[boughtTickets.indexOf(x)] = "";

var s = "";

for (var i = 0; i < boughtTickets.length; i++) {

s += boughtTickets[i] + " ";

}

document.getElementById("boughtTickets").value = s;

boughtTicketCount--;

}

}

function BlueButtonClick(x)

{

if(confirm("Do you really want to cansel your advance booking?"))

{

boughtTicketCount--;

document.getElementById("blueButton").value = x;

document.getElementById("sub").click();

}

}

</script>

ВЫВОДЫ

В курсовом проекте были реализованы основные функции работы информационной системы бронирования билетов на соревнования по теннису. Были изучены основы работы с базами данных, получен опыт разработки и проектирования баз данных, а также разработки программных продуктов с использованием баз данных, а также изучен язык запросов SQL.

Несомненным преимуществом веб-сайта является его кроссплатформенность, а также то, что внедренная база данных не требует наличия у пользователя отдельного сервера.

К недостаткам программы можно отнести долгое время запуска и, возможно, некоторые огрехи в визуальном оформлении пользовательского интерфейса.

Все запланированные поставленные цели в постановке задачи выполнены. Реализована форма для выполнения произвольных запросов. Так же были разработаны методы и формы добавления, редактирования и удаления данных о стадионах, игроках, матчах, билетах и покупках. Был реализован поиск по разным критериям, фильтрация, сортировка данных. Реализованы задача автоматизации и ведение документации отделения, фиксированные запросы.

Разработанная информационная система предоставляет возможность удобно и безопасно хранить данные о соревнованиях по теннису, упрощает ведение документооборота турнира, а также позволяет легко манипулировать каждой сущностью, спроектированной в БД.

ПЕРЕЧЕНЬ ССЫЛОК

1. Грабер М. Введение в SQL [Текст] : пер. с англ. / М. Грабер. – К.: Лори, 1992. – 382 с.
2. Дейт К.Дж. Введение в системы баз данных [Текст] : пер. с англ. / К.Дж. Дейт. – М. : Издательский дом «Вильямс», 2005. – 1328 с.
3. Ульман Дж. Основы реляционных баз данных [Текст] : пер. с англ. / Дж. Ульман. – К.: Лори, 2006. – 376 с.
4. Троелсен, Э. C# и платформа .NET. Библиотека программиста  
   [Текст] / Э. Троелсен – СПб.: Питер, 2013. – 1310с.
5. Грофф Дж. Р.SQL. Полное руководство [Текст] / Грофф Дж. Р., Пол Н. Вайнберг.: пер. с англ. – СПб.: BHV, 2001. – 816 с.

Приложение А

Фрагменты кода приложения

using Competitions.Models;

using System;

using System.Collections.Generic;

using System.Data;

using System.Data.SqlClient;

using System.Linq;

using System.Web;

using System.Web.Mvc;

namespace Competitions.Controllers

{

public class HomeController : Controller

{

ApplicationDbContext db = new ApplicationDbContext();

public ActionResult Index()

{

return View();

}

public ActionResult About()

{

ViewBag.Message = "Your application description page.";

return View();

}

public ActionResult Contact()

{

ViewBag.Message = "Your contact page.";

return View();

}

[HttpGet]

public ActionResult MyQuery()

{

return View();

}

[HttpPost]

public ActionResult MyQueryPart(string text)

{

try

{

SqlConnection sqlconn = new SqlConnection(@"Data Source=(LocalDB)\v11.0;AttachDbFilename=E:\БД\Курсач\Сайт\Competitions\Competitions\App\_Data\aspnet-Competitions-20151107043436.mdf;Integrated Security=True");

sqlconn.Open();

SqlDataAdapter oda = new SqlDataAdapter(text, sqlconn);

DataTable dt = new DataTable();

oda.Fill(dt);

List<string> columnNames = new List<string>();

foreach (DataColumn dc in dt.Columns)

{

columnNames.Add(dc.ColumnName);

}

List<List<string>> cellData = new List<List<string>>();

foreach (DataRow dr in dt.Rows)

{

List<string> rowElements = new List<string>();

foreach (Object cell in dr.ItemArray)

{

rowElements.Add(cell.ToString());

}

cellData.Add(rowElements);

}

sqlconn.Close();

ViewBag.ColumnNames = columnNames;

ViewBag.CellData = cellData;

return PartialView("MyQuryPartial");

}

catch

{

return PartialView("QueryErrorPartial");

}

}

[HttpGet]

public ActionResult ShowUsers()

{

ViewBag.AdminRoleId = db.Roles.First(r => r.Name == "Admin").Id;

ViewBag.Countries = db.Countries.ToList();

return View(db.Users.ToList());

}

[HttpPost]

public ActionResult PromoteToAdmin(string userId)

{

ApplicationUser user = db.Users.First(u => u.Id == userId);

string adminRoleId = db.Roles.First(r => r.Name == "Admin").Id;

user.Roles.Add(new Microsoft.AspNet.Identity.EntityFramework.IdentityUserRole() { RoleId = adminRoleId, UserId = user.Id });

db.SaveChanges();

ViewData["AdminRoleId"] = adminRoleId;

return PartialView("PromoteToAdminButton", user);

}

[HttpPost]

public ActionResult DemoteFromAdmin(string userId)

{

ApplicationUser user = db.Users.First(u => u.Id == userId);

string adminRoleId = db.Roles.First(r => r.Name == "Admin").Id;

user.Roles.Remove(user.Roles.First(r => r.RoleId == adminRoleId && r.UserId == user.Id));

db.SaveChanges();

ViewData["AdminRoleId"] = adminRoleId;

return PartialView("PromoteToAdminButton", user);

}

[HttpPost]

public ActionResult Search(string Login, string email, string Country, string Status)

{

if (Country == "Not Selected" || Country == null)

{

Country = "";

}

if(Status == null)

{

Status = "AllUsers";

}

List<ApplicationUser> Users = db.Users.Where(user => user.UserName.Contains(Login) &&

user.Email.Contains(email) &&

user.Country.Contains(Country)).ToList();

switch (Status)

{

case "Admins":

Users = Users.Where(user => user.Roles.Count > 0).ToList();

break;

case "NotAdmins":

Users = Users.Where(user => user.Roles.Count == 0).ToList();

break;

default:

break;

}

ViewBag.AdminRoleId = db.Roles.First(r => r.Name == "Admin").Id;

return PartialView("UsersSearch", Users);

}

[HttpPost]

public ActionResult ShowPurchase(string purchaseCode)

{

return View(db.Purchases.First(purchase => purchase.Code == purchaseCode));

}

public ActionResult FindPurchase()

{

return View();

}

[HttpPost]

public ActionResult PurchaseDetails(string code)

{

Purchase purchase = null;

if(db.Purchases.Count(p => p.Code == code) > 0)

{

purchase = db.Purchases.First(p => p.Code == code);

}

return PartialView("PurchasePartial", purchase);

}

public ActionResult PrintTickets(string purchaseCode)

{

Purchase purchase = db.Purchases.First(p => p.Code == purchaseCode);

int firstPlayerId = purchase.Tickets.ElementAt(0).Game.FirstPlayerId;

int secondPlayerId = purchase.Tickets.ElementAt(0).Game.SecondPlayerId;

ViewBag.FirstPlayer = db.Players.First(p => p.Id == firstPlayerId);

ViewBag.SecondPlayer = db.Players.First(p => p.Id == secondPlayerId);

return View(purchase);

}

}

}

using System;

using System.Collections.Generic;

using System.Data;

using System.Data.Entity;

using System.Linq;

using System.Net;

using System.Web;

using System.Web.Mvc;

using Competitions.Models;

using System.Net.Mail;

namespace Competitions.Controllers

{

public class GameController : Controller

{

private ApplicationDbContext db = new ApplicationDbContext();

// GET: Game

public ActionResult Index()

{

var games = db.Games.Include(g => g.Stadium);

ViewBag.Stadiums = db.Stadiums.ToList();

ViewBag.Players = db.Players.ToList();

return View(games.ToList());

}

// GET: Game/Details/5

public ActionResult Details(int? id)

{

if (id == null)

{

return new HttpStatusCodeResult(HttpStatusCode.BadRequest);

}

Game game = db.Games.Find(id);

if (game == null)

{

return HttpNotFound();

}

ViewBag.FirstPlayer = db.Players.First(p => p.Id == game.FirstPlayerId);

ViewBag.SecondPlayer = db.Players.First(p => p.Id == game.SecondPlayerId);

ViewBag.Stadium = db.Stadiums.First(s => s.Id == game.StadiumId);

ViewBag.CurrentUser = db.Users.FirstOrDefault(u => u.UserName == User.Identity.Name);

return View(game);

}

// GET: Game/Create

public ActionResult Create()

{

ViewBag.StadiumId = new SelectList(db.Stadiums, "Id", "Name");

ViewBag.FirstPlayerId = new SelectList(db.Players, "Id", "Surname");

ViewBag.SecondPlayerId = new SelectList(db.Players, "Id", "Surname");

return View();

}

// POST: Game/Create

// To protect from overposting attacks, please enable the specific properties you want to bind to, for

// more details see http://go.microsoft.com/fwlink/?LinkId=317598.

[HttpPost]

[ValidateAntiForgeryToken]

public ActionResult Create([Bind(Include = "Id,FirstPlayerId,SecondPlayerId,StadiumId,Date,FirstPlayerPoints,SecondPlayerPoints,Price")] Game game)

{

if (ModelState.IsValid)

{

if (game.FirstPlayerId == game.SecondPlayerId)

{

ModelState.AddModelError("", "First player and second player must be different!");

ViewBag.FirstPlayerId = new SelectList(db.Players, "Id", "Surname", game.FirstPlayerId);

ViewBag.SecondPlayerId = new SelectList(db.Players, "Id", "Surname", game.SecondPlayerId);

ViewBag.StadiumId = new SelectList(db.Stadiums, "Id", "Name", game.StadiumId);

return View(game);

}

game.GenerateTickets();

db.Games.Add(game);

db.SaveChanges();

return RedirectToAction("Index");

}

ViewBag.FirstPlayerId = new SelectList(db.Players, "Id", "Surname", game.FirstPlayerId);

ViewBag.SecondPlayerId = new SelectList(db.Players, "Id", "Surname", game.SecondPlayerId);

ViewBag.StadiumId = new SelectList(db.Stadiums, "Id", "Name", game.StadiumId);

return View(game);

}

// GET: Game/Edit/5

public ActionResult Edit(int? id)

{

if (id == null)

{

return new HttpStatusCodeResult(HttpStatusCode.BadRequest);

}

Game game = db.Games.Find(id);

if (game == null)

{

return HttpNotFound();

}

ViewBag.FirstPlayerId = new SelectList(db.Players, "Id", "Surname", game.FirstPlayerId);

ViewBag.SecondPlayerId = new SelectList(db.Players, "Id", "Surname", game.SecondPlayerId);

ViewBag.StadiumId = new SelectList(db.Stadiums, "Id", "Name", game.StadiumId);

return View(game);

}

// POST: Game/Edit/5

// To protect from overposting attacks, please enable the specific properties you want to bind to, for

// more details see http://go.microsoft.com/fwlink/?LinkId=317598.

[HttpPost]

[ValidateAntiForgeryToken]

public ActionResult Edit([Bind(Include = "Id,FirstPlayerId,SecondPlayerId,StadiumId,Date,FirstPlayerPoints,SecondPlayerPoints,Price")] Game game)

{

if (ModelState.IsValid)

{

if (game.FirstPlayerId == game.SecondPlayerId)

{

ModelState.AddModelError("", "First player and second player must be different!");

ViewBag.FirstPlayerId = new SelectList(db.Players, "Id", "Surname", game.FirstPlayerId);

ViewBag.SecondPlayerId = new SelectList(db.Players, "Id", "Surname", game.SecondPlayerId);

ViewBag.StadiumId = new SelectList(db.Stadiums, "Id", "Name", game.StadiumId);

return View(game);

}

db.Entry(game).State = EntityState.Modified;

db.SaveChanges();

return RedirectToAction("Index");

}

ViewBag.FirstPlayerId = new SelectList(db.Players, "Id", "Surname", game.FirstPlayerId);

ViewBag.SecondPlayerId = new SelectList(db.Players, "Id", "Surname", game.SecondPlayerId);

ViewBag.StadiumId = new SelectList(db.Stadiums, "Id", "Name", game.StadiumId);

return View(game);

}

// GET: Game/Delete/5

public ActionResult Delete(int? id)

{

if (id == null)

{

return new HttpStatusCodeResult(HttpStatusCode.BadRequest);

}

Game game = db.Games.Find(id);

if (game == null)

{

return HttpNotFound();

}

ViewBag.FirstPlayer = db.Players.First(p => p.Id == game.FirstPlayerId);

ViewBag.SecondPlayer = db.Players.First(p => p.Id == game.SecondPlayerId);

ViewBag.Stadium = db.Stadiums.First(s => s.Id == game.StadiumId);

ViewBag.CurrentUser = db.Users.FirstOrDefault(u => u.UserName == User.Identity.Name);

return View(game);

}

// POST: Game/Delete/5

[HttpPost, ActionName("Delete")]

[ValidateAntiForgeryToken]

public ActionResult DeleteConfirmed(int id)

{

Game game = db.Games.Find(id);

db.Games.Remove(game);

db.SaveChanges();

return RedirectToAction("Index");

}

protected override void Dispose(bool disposing)

{

if (disposing)

{

db.Dispose();

}

base.Dispose(disposing);

}

[HttpGet]

public ActionResult UnBookTicket(int blueButton)

{

string purchaseCode = db.Tickets.First(t => t.Id == blueButton).PurchaseCode;

db.Tickets.First(t => t.Id == blueButton).PurchaseCode = null;

db.SaveChanges();

if (db.Tickets.Count(t => t.PurchaseCode == purchaseCode) == 0)

db.Purchases.Remove(db.Purchases.First(purchase => purchase.Code == purchaseCode));

db.SaveChanges();

Ticket ticket = db.Tickets.First(t => t.Id == blueButton);

Game game = ticket.Game;

return RedirectToAction("Details", new { id = game.Id });

}

[HttpGet]

public ActionResult BookTickets(string boughtTickets)

{

string[] ticketId = boughtTickets.Split(new char[] { ' ' }, StringSplitOptions.RemoveEmptyEntries);

ApplicationUser currentUser = db.Users.First(u => u.UserName == User.Identity.Name);

List<Ticket> bookedTickets = new List<Ticket>();

for (int i = 0; i < ticketId.Length; i++)

{

int id = Convert.ToInt32(ticketId[i]);

Ticket ticket = db.Tickets.First(t => t.Id == id);

bookedTickets.Add(ticket);

}

return View(bookedTickets);

}

[HttpPost]

public ActionResult BookTickets(int[] Id)

{

ApplicationUser currentUser = db.Users.First(u => u.UserName == User.Identity.Name);

Purchase purchase = new Purchase() { Code = GenerateSecetCode(), UserId = currentUser.Id };

List<Ticket> boughtTickets = new List<Ticket>();

for (int i = 0; i < Id.Length; i++)

{

int id = Id[i];

Ticket boughtTicket = db.Tickets.First(t => t.Id == id);

boughtTickets.Add(boughtTicket);

boughtTicket.PurchaseCode = purchase.Code;

}

db.Purchases.Add(purchase);

db.SaveChanges();

SendMessage(purchase);

int idd = Id[0];

int gameId = db.Tickets.First(t => t.Id == idd).GameId;

return RedirectToAction("Details", new { id = gameId });

}

/// <summary>

/// Метод, который отправляет письмо по электронной почте.

/// </summary>

/// <param name="boughtTickets">Заказанные билеты.</param>

private void SendMessage(Purchase purchase)

{

try

{

List<Ticket> boughtTickets = purchase.Tickets;

ApplicationUser currentUser = db.Users.First(u => u.UserName == User.Identity.Name);

Ticket ticket1 = boughtTickets[0];

Player player1 = db.Players.First(p => p.Id == ticket1.Game.FirstPlayerId);

Player player2 = db.Players.First(p => p.Id == ticket1.Game.SecondPlayerId);

MailMessage mail = new MailMessage();

mail.From = new MailAddress("tenniscompetitions1@gmail.com");

mail.To.Add(new MailAddress(currentUser.Email));

mail.Subject = "Tickets booking";

if (currentUser.Name == null || currentUser.Surname == null)

{

mail.Body = "Dear " + currentUser.UserName + ", \n\n";

}

else

{

mail.Body = "Dear " + currentUser.Name + " " + currentUser.Surname + ", \n\n";

}

mail.Body += "You have sucsessfuly booked the tickets for the tennis game between " +

player1.Name + " " + player1.Surname +

" and " + player2.Name + " " + player2.Surname +

" that wiil take place at " + boughtTickets[0].Game.Stadium.Name +

" stadium at " + ticket1.Game.Date.ToShortDateString() + ".\nYour tickets:\n";

for (int i = 0; i < boughtTickets.Count; i++)

{

mail.Body += "Sector " + boughtTickets[i].Sector + " row " + (boughtTickets[i].Row + 1) + " place " + (boughtTickets[i].Place + 1);

if (i != boughtTickets.Count - 1)

mail.Body += ";\n";

else

mail.Body += ".\n";

}

mail.Body += "Your secret code is " + purchase.Code + "\n\n";

mail.Body += "Sincerely, Administration of tennis tournament.";

SmtpClient client = new SmtpClient();

client.Host = "smtp.gmail.com";

client.Port = 587;

client.EnableSsl = true;

client.Credentials = new NetworkCredential("tenniscompetitions1", "NBAilia1997");

client.DeliveryMethod = SmtpDeliveryMethod.Network;

client.Send(mail);

}

catch { }

}

/// <summary>

/// Метод для генерации секретного кода пользователя при заказе билетов.

/// </summary>

/// <returns>Секретный код.</returns>

private string GenerateSecetCode()

{

Random r = new Random();

string s = "";

for (int i = 0; i < 6; i++)

{

s += r.Next(10);

}

return s;

}

[HttpPost]

public ActionResult SortIncrease(string status1)

{

List<Game> games = db.Games.ToList();

switch(status1)

{

case "PlayedGames":

games = games.Where(game => game.Date < DateTime.Now).ToList();

break;

case "UpcomingGames":

games = games.Where(game => game.Date > DateTime.Now).ToList();

break;

default:

break;

}

games = games.OrderBy(game => game.Date).ToList();

ViewBag.Stadiums = db.Stadiums.ToList();

ViewBag.Players = db.Players.ToList();

return PartialView("OrderedGamesPartial", games);

}

[HttpPost]

public ActionResult SortDecrease(string status2)

{

List<Game> games = db.Games.ToList();

switch (status2)

{

case "PlayedGames":

games = games.Where(game => game.Date < DateTime.Now).ToList();

break;

case "UpcomingGames":

games = games.Where(game => game.Date > DateTime.Now).ToList();

break;

default:

break;

}

games = games.OrderBy(game => game.Date).Reverse().ToList();

ViewBag.Stadiums = db.Stadiums.ToList();

ViewBag.Players = db.Players.ToList();

return PartialView("OrderedGamesPartial", games);

}

[HttpPost]

public ActionResult ShowReport()

{

ViewBag.Stadiums = db.Stadiums.ToList();

ViewBag.Players = db.Players.ToList();

return View(db.Games.Where(game => game.FirstPlayerPoints != game.SecondPlayerPoints).ToList());

}

}

}