**СОДЕРЖАНИЕ**

**ВВЕДЕНИЕ** 4

1 Логическая структура базы данных 5

2 Физическая структура базы данных. Аппаратное и программное обеспечение системы 7

2.1 Физическая структура базы данных 7

2.2 Аппаратное и программное обеспечение системы 8

3 Реализация интерфейса пользователя 9

3.1 Описание основной структуры ASP.NET документов 9

4 Руководство пользователя 12

4.1 Общие сведения 12

4.2 Руководство пользователя 12

5 Руководство программиста 17

5.1 Назначения и условия применения программы 17

5.2 Характеристики работы программы 17

**ЗАКЛЮЧЕНИЕ** 18

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ** 19

**ПРИЛОЖЕНИЕ А.** Исходные коды хранимых процедур 20

**ВВЕДЕНИЕ**

В настоящее время происходит автоматизация всех областей человеческой деятельности. Так рутинную работу по учету и обработке различной информации теперь можно возложить на компьютер, который будет справляться с этой задачей в разы быстрее и качественнее человека. Поэтому разработка таких программных комплексов является актуальной задачей в нынешней обстановке. Не лишним будет и разработка такой информационной системы для предприятия покупки и продажи билетов театров. Ведь в таких больших организациях достаточно сложно вести контроль над всеми сотрудниками, а также осуществление продаж и покупки билетов, осуществляемые этими сотрудниками.

Одним из способов разработки такого приложения, является приложение, построенное на актуальной сейчас архитекторе “клиент-сервера”. Принцип данного подхода заключается в том, что для работы с приложением достаточно обыкновенного браузера и доступа в интернет или локальную сеть. А вся обработка данных осуществляется в недоступном для пользователя удаленном сервере.

Для разработки данного приложения выбрана развивающаяся сейчас технология создания веб-приложений веб-форм на основе MVC. Данная технология позволяет достаточно быстро и эффективно создать, и настроить сайт для работы с базами данных. Кроме эффективной работы с базой данных данная технология позволяет спроектировать довольно симпатичный и удобный интерфейс для работы с данными.

В качестве предметной области выбран “Онлайновая система распространения билетов в театре”. К описанию предметной области относится:

* данные о пьесе (Название, автор, жанр, описание);
* заказы (Дата, пользователь, количество билетов, категория, цена, статус заказа);
* пользователи (имя, пароль, роль пользователя, e-mail, телефон).
* даты (номер пьесы, дата).

К дополнительным требованиям отображения данных относятся:

* Полная информация по пьесам;
* Информация о всех сделанных заказах;
* Информация о доступных билетах;
* Многоязычное отображение;
* Персонализация содержимого страницы в зависимости от типа аккаунта;

**1 ЛОГИЧЕСКАЯ СТРУКТУРА БАЗЫ ДАННЫХ**

Изучив предметную область можно выделить 7 основных сущностей: Authors, Genres, Plays, Dates, Orders, Logins. Каждая из них содержит свои атрибуты определённых типов данных. Описания всех сущностей и их атрибутов представлены в таблицах 1.1 – 1.8.

Таблица 1.1 – Описание сущности *Authors*

|  |  |  |
| --- | --- | --- |
| **Название атрибута** | **Тип данных** | **Наличие ключа** |
| Id | Счетчик | Первичный ключ |
| name | Текстовый | – |

Таблица 1.2 – Описание сущности *Genres*

|  |  |  |
| --- | --- | --- |
| **Название атрибута** | **Тип данных** | **Наличие ключа** |
| Id | Счетчик | Первичный ключ |
| name | Текстовый | – |

1.3 – Описание сущности *Plays*

|  |  |  |
| --- | --- | --- |
| **Название атрибута** | **Тип данных** | **Наличие ключа** |
| Id | Счетчик | Первичный ключ |
| name | Текстовый | – |
| authorId | Числовой | Вторичный ключ |
| genreId | Числовой | Вторичный ключ |
| description | Текстовый | – |

Таблица 1.4 – Описание сущности *Dates*

|  |  |  |
| --- | --- | --- |
| **Название атрибута** | **Тип данных** | **Наличие ключа** |
| Id | Счетчик | Первичный ключ |
| playsId | Числовой | Вторичный ключ |
| date | Дата | – |

Таблица 1.5 – Описание сущности *Logins*

|  |  |  |
| --- | --- | --- |
| **Название атрибута** | **Тип данных** | **Наличие ключа** |
| Id | Счетчик | Первичный ключ |
| name | Текстовый | – |
| password | Текстовый | – |
| roleId | Числовой | – |
| email | Текстовый | – |
| phone | Текстовый | – |

Таблица 1.6 – Описание сущности *Orders*

|  |  |  |
| --- | --- | --- |
| **Название атрибута** | **Тип данных** | **Наличие ключа** |
| ID | Счетчик | Первичный ключ |
| dateId | Числовой | Вторичный ключ |
| loginId | Числовой | Вторичный ключ |
| categoryId | Числовой | – |
| quantity | Числовой | – |
| price | Цена | – |
| statusOrderId | Числовой | – |

Между сущностями определены связи типа один ко многим:

* связь между сущностями “*Authors*” (на стороне один) и “*Plays*” (на стороне многие) по полю “*authorId*”;
* связь между сущностями “*Genres*” (на стороне один) и “ *Plays*” (на стороне многие) по полю “*genreId*”;
* связь между сущностями “ *Plays*” (на стороне один) и “*Dates*” (на стороне многие) по полю “*playsId*”;
* связь между сущностями “ *Dates*” (на стороне один) и “*Orders*” (на стороне многие) по полю “*dateId*”;
* связь между сущностями “*Logins*” (на стороне один) и “ *Orders*” (на стороне многие) по полю “loginId”;

Cвязи заданные между сущностями могут быть идентифицирующими или не идентифицирующими. Если задана идентифицирующая связь, то это значит, что каждому экземпляру дочерней сущности должен соответствовать хотя бы один экземпляр родительской.

Все первичные ключи в сущностях спроектированной структуры являются простыми – состоят из одного поля и однозначно идентифицируют запись.

Логическая структура базы данных приведена на рисунке 1.1.

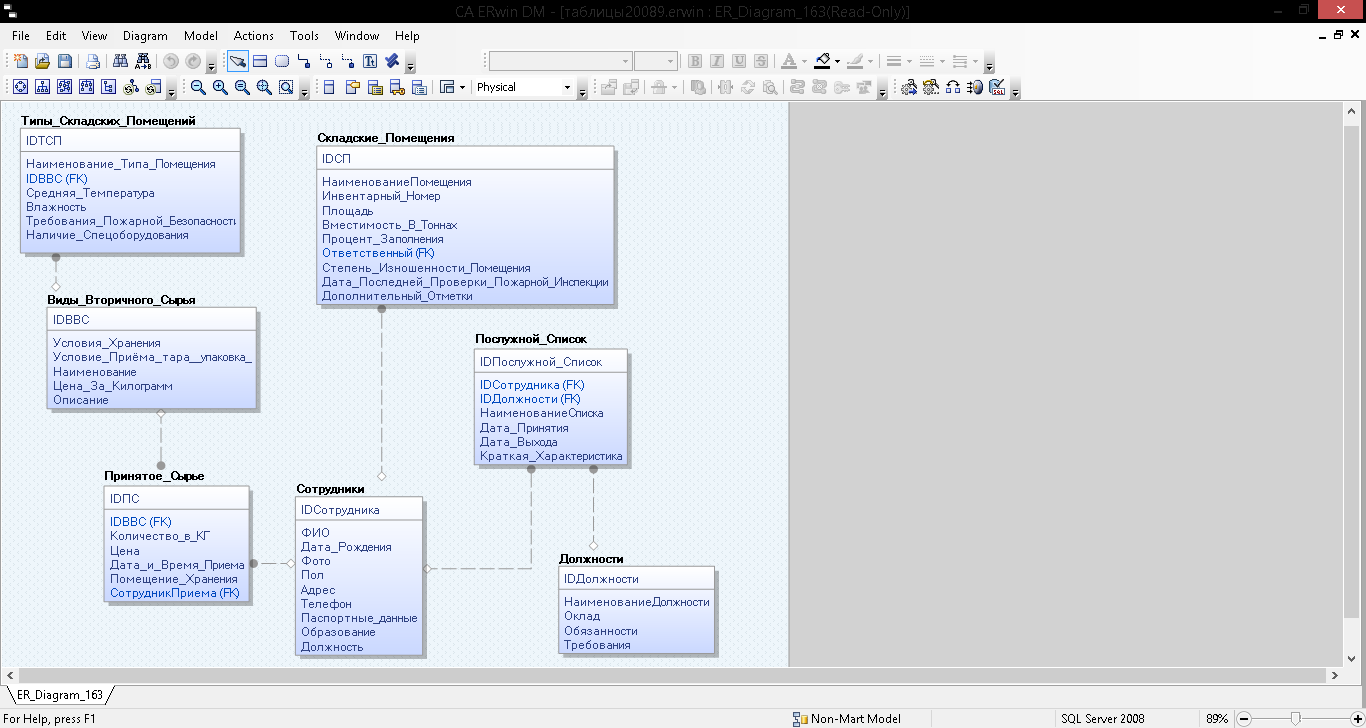


Рисунок 1.1 – Логическая структура базы данных

**2 ФИЗИЧЕСКАЯ СТРУКТУРА БАЗЫ ДАННЫХ. АППАРАТНОЕ И ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ СИСТЕМЫ**

**2.1 Физическая структура базы данных**

По созданной логической структуре была разработана база данных средствами СУБД MS SQL Server. В неё входят 6 таблиц: *Authors*, *Genres*, *Plays*, *Dates*, *Logins*, *Orders*.

В таблицу “*Authors*” входят поля: “*Id*” типа int (первичный ключ), “*name*” типа nvarchar(50).

В таблицу “*Genres*” входят поля: “*Id*” типа int (первичный ключ), “*name*” типа nvarchar(50).

В таблицу “*Plays*” входят поля: “*Id*” типа int (первичный ключ), “name” типа nvarchar(50), “*authorId*” типа int (вторичный ключ), “*genreId*” типа int (вторичный ключ), “*description*” типа nvarchar(max).

В таблицу “Dates” входят поля: “*Id*” типа int (первичный ключ), “*playId*” типа int (вторичный ключ), “*date*” типа date.

В таблицу “*Logins*” входят поля: “*Id*” типа int (первичный ключ), “*name*” типа nvarchar(50), “*password*” типа nvarchar(50), “*roleId*” типа int, “*email*” типа nvarchar(50), “*phone*” типа nvarchar(50).

В таблицу “*Orders*” входят поля: “*Id*” типа int (первичный ключ), “*dateId*” типа int (вторичный ключ), “*loginId*” типа int (вторичный ключ), “*categoryId*” типа int, “*quantity*” типа int, “*price*” типа money, “*statusOrderId*” типа int.

Столбцы всех таблиц с первичными ключами являются автоинкрементными, т.е. значения этих полей генерируются автоматически.

Диаграмма созданной базы данных приведена на рисунке 2.1.

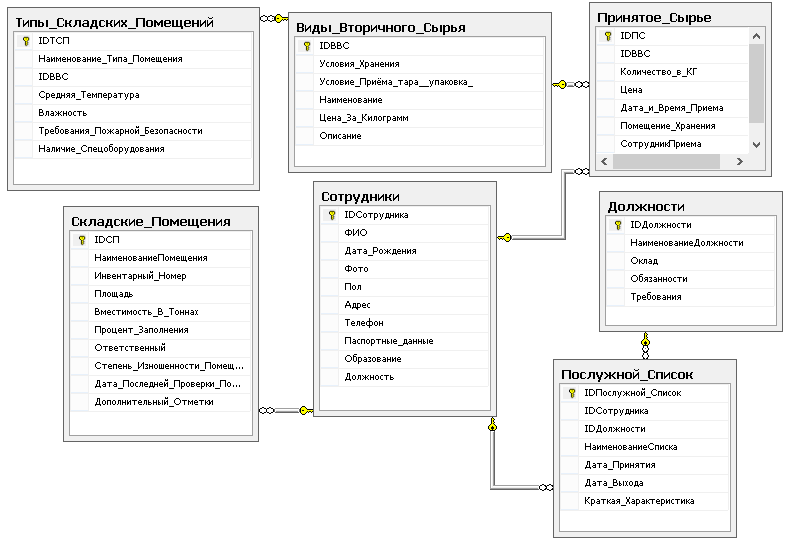


Рисунок 2.1 – Диаграмма базы данных

Данные таблицы соответствуют требованиям нормализации реляционных БД. Таблицы находятся в первой нормальной форме, т.к. все их поля являются простыми (неделимыми), нет повторяющихся полей и каждый столбец таблиц хранит одно единственное значение и не является ни списком, ни множеством значений. Таблицы находятся во второй нормальной форме, т.к. они находятся в первой нормальной форме, имеют простые первичные ключи и каждое не ключевое поле функционально зависит от первичного ключа. Таблицы находится в третьей нормальной форме, т.к. они находятся во второй нормальной форме и все не ключевые поля являются взаимно-независимыми.

**2.2 Аппаратное и программное обеспечение системы**

Разработанная база данных состоит из двух основных файлов:

* PVS.mdf – файл базы данных. В нём хранятся непосредственно данные и структура базы данных, а также специально созданные таблицы, которые хранят данные учетных записей пользователей. Размер данного файла составляет 6 МБ;
* PVS\_log.ldf – журнал транзакций. Размер журнала 10 МБ.

Рост размера базы данных ограничивается лишь свободной памятью на жёстком диске.

К SQL Server 2012 предъявляются следующие аппаратные и программные требования:

* минимум 256 МБ ОЗУ (рекомендуется 1 ГБ или выше);
* 32-разрядные системы: компьютер, оборудованный процессором Intel или совместимым процессором с тактовой частотой 1 ГГц и выше (рекомендуется 2 ГГц и выше, поддерживается только один процессор);
* 64-разрядные системы: процессор с тактовой частотой 1,4 ГГц и выше (рекомендуется 2 ГГц и выше, поддерживается только один процессор);
* 6 ГБ свободного места на диске;
* поддерживаемые операционные системы: Windows Server 2003 Service Pack 2; Windows Server 2008; Windows Vista; Windows Vista Service Pack 1; Windows XP Service Pack 2; Windows XP Service Pack 3 [2].

# 3 РЕАЛИЗАЦИЯ ИНТЕРФЕЙСА ПОЛЬЗОВАТЕЛЯ

## **3.1 Описание основной структуры ASP.NET документов**

Было осуществлено взаимодействие между созданной базой данных и моделями. Используя шаблон DAO были созданы интерфейсы, и классы их реализующие, для работы с СУБД MSSQL Server.

Структура созданных классов для взаимодействия с базой данных отображено на рисунке 3.1.

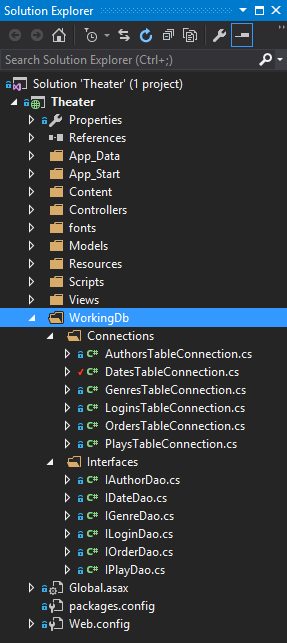


Рисунок 3.1 - Структура приложения

В веб-приложениях, основанных на технологии ASP.NET MVC Framework основные механизмы передачи данных от веб-страниц в функции обработки осуществляются через контроллеры, которые по сути являются главным звеном приложения. Они связывают модель и пользовательский интерфейс.

Контроллеры являются классами .NET, которые содержат логику, требуемую для обработки запроса. Роль контроллера заключается в инкапсуляции логики приложения. Другими словами, контроллеры отвечают за обработку входящих запросов, выполнение операций над моделью предметной области и выбор представлений для визуализации пользователю.

Проектируемое приложение содержит следующий набор контроллеров:

* AccountController - контроллер для связи с таблицей Logins;
* HomeController – контроллер для контроля локализации и вывода начальной страницы;
* PlaysController - контроллер для связи с таблицей Plays;
* CourierController - контроллер для связи с таблицей Orders;
* AdminController - контроллер для связи со всеми таблицами;

Все контроллеры имеют схожую структуру, в которой описаны методы создания, добавления, удаления и обновления данных. В программе содержится ряд дополнительных контроллеров. Их использованию позволяет внедрить в программу дополнительные функции по авторизации и аутентификации.

Хотя работа приложения MVC управляется главным образом контроллерами, но непосредственно пользователю приложение доступно в виде представления, которое и формирует внешний вид приложения. Названия представлений схожи с названиями соответствующих им контроллеров. В ASP.NET MVC 5 представления представляют файлы с расширением cshtml, которые содержат код с интерфейсом пользователя, как правило, на языке html и Razor. Внешний вид представлений основан на их назначении. Основные типы представлений:

* Index - представление используемое для вывода данных таблицы. На каждом представлении такого вила имеется поле для поиска данных по некоторому заданному параметру. Навигация производится с помощью специальных объектов в виде нумерованных списков.
* Edit - представление, используемое для редактирования данных выбранной записи в исходной таблице. Содержит поля ввода и выпадающие списки, благодаря которым пользователь может изменить значения записи.
* Create - представление, аналогичное типу Edit. Используется для создания новой записи в таблице.
* Delete - представление, используемое для удаления. С его помощью осуществляется вывод данных записи и кнопка удаления, при нажатии которой производится удаление данных из таблицы.
* Details - представление, являющееся переходным к редактирования. Вывод запись в виде отдельной таблицы для просмотра всего набора полей.

Листинг котроллеров и представлений основных типов представлен в приложении А.

Оформление представлений может осуществляться с использование Java script, а также CSS файлов, которые подключаются через специальные теги. Среда разработки Visual Studio позволяет подключать уже созданные дополнительные пакеты через менеджер Nuget. В приложении использовались дополнительные пакеты для создания страничного доступа к табличным данным, а также пакеты bootstrap, предоставляющие оформление внешнего вида различных элементов представления.

# 4 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

**4.1 Общие сведения**

Руководство пользователя информационной системы, обеспечивает получение пользователем базовых навыков по настройке и начальной эксплуатации пользовательского программного обеспечения для доступа к функциям, которые обеспечивают работу с базой данных театра.

Разработанная информационная система предназначена для автоматизации продажи билетов, упрощения работы персонала театра, визуализации данных по запросу и внесения изменений в базу театра.

Пользователь системы должен обладать следующей квалификацией:

* пользовательские навыки работы с ЭВМ;
* пользовательские навыки работы с Веб-браузерами, такими как *Mozilla Firefox*, *Opera*, *Microsoft Internet Explorer* и т.п.;

Разработанное программное обеспечение позволяет автоматизировать ежедневно повторяющиеся операции, такие как покупка билетов посетителями театра, продажа билетов работниками театра.

Система не требует инсталляции, все необходимые файлы загружаются на компьютер пользователя автоматически или по запросу.

В состав загружаемых файлов входят динамически генерируемые сервером наборы html-страниц, содержащих текстовый и графический контент в соответствии с запросами пользователя.

**4.2 Руководство пользователя**

Разработанный программный комплект позволяет выполнять следующие основные действия:

* просмотр, удаление, редактирование и добавление данных о авторах предстоящих пьес.
* просмотр, удаление, редактирование и добавление данных о жанрах предстоящих пьес.
* просмотр, удаление, редактирование и добавление данных о пьесах.
* просмотр, удаление, редактирование и добавление данных о заказе на определенную дату.
* просмотр, удаление, редактирование и добавление данных о заказе пользователя.
* добавление данных о пользователях.

При запуске программного комплекса, пользователь попадает на главную страницу сайта, где можно увидеть предстоящие пьесы, а также, меню, для авторизации, регистрации или выбора языка, рисунок 4.1.

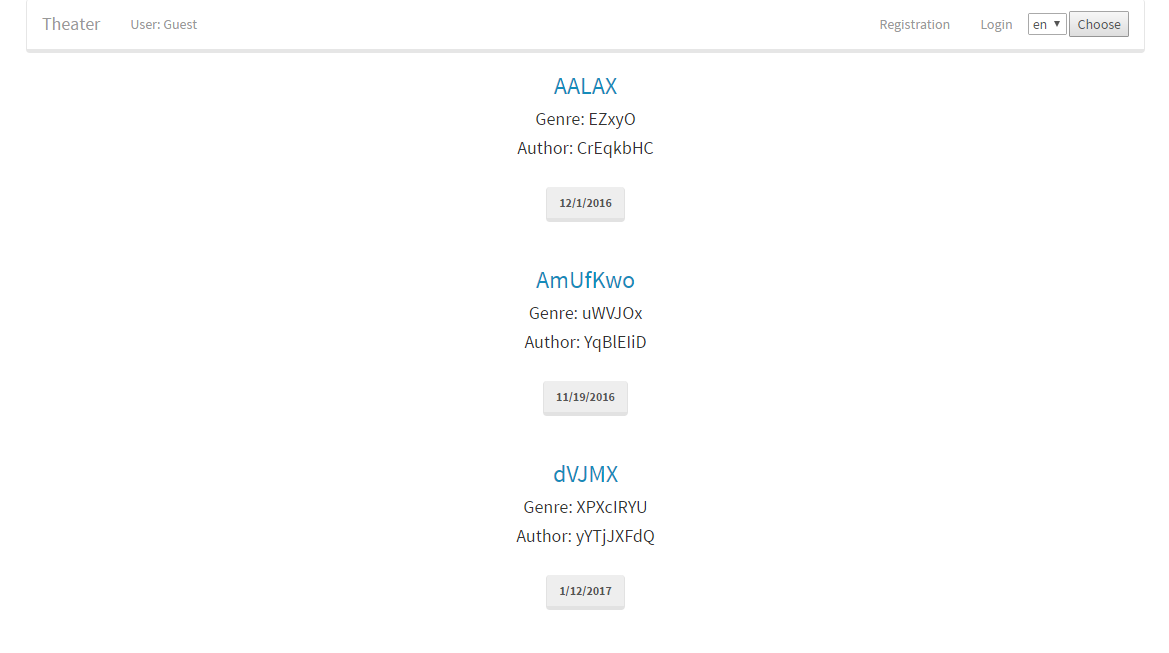


Рисунок 4.1 – Главная страница приложения

Для заказа билетов пользователю нужно авторизоваться или зарегистрироваться, рисунок 4.2.

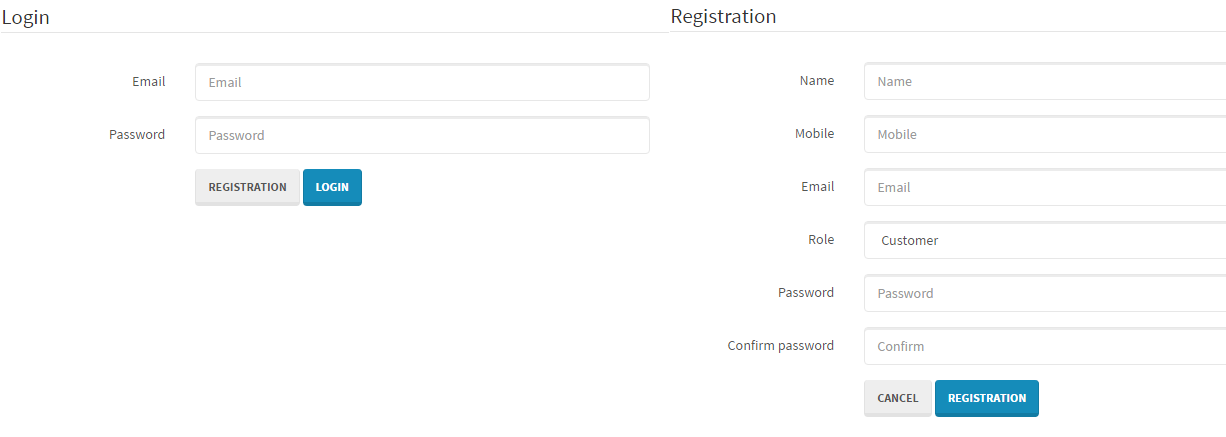


Рисунок 4.2 – Меню регистрации и входа

После входа в систему можно переходить к непосредственной работе с данными. Пользовательское программное обеспечение позволяет заказывать билеты и отменять свои заказы. Если пользователь хочет изменять и выполнять заказы, ему нужно зарегистрироваться или авторизоваться как курьер. Если пользователь хочет изменять все данные, ему нужно авторизоваться под аккаунтом администратора. Для быстрого и удобного перехода между страницами сайта реализовано меню, рисунок 4.3.



Рисунок 4.3 – Меню навигации приложения

Раздел меню “Orders” доступен курьеру и администратору. Он предназначен для работы с данными о заказах.

Раздел меню “Cart” доступен всем авторизованным пользователям. Он предназначен для просмотра своих заказов, и, при желании, их отмене.

Для просмотра, изменения, добавления и удаления всех таблиц нужно под аккаунтом администратора ввести в строку поиска браузера: http://<*SiteName*>/admin. В этом разделе можно изменять все данные.

При выборе любого пункта меню, пользователю предоставляется краткая информация по выбранным данным. Например, выбрав пункт “Plays” в разделе администратора пользователю предоставляется информация о пьесах, рисунок 4.5.

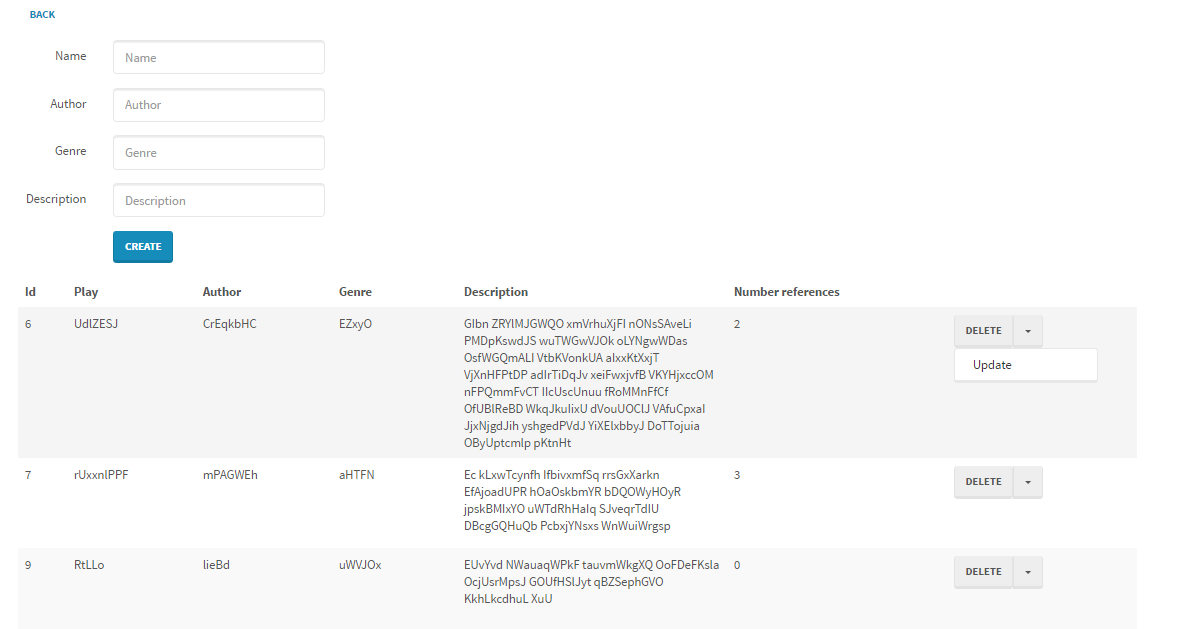


Рисунок 4.5 – Информация о пьесах

Для работы с записями реализовано три функции: удаление, добавление и редактирование записей, рисунок 4.6.

а)  б)  в) 

Рисунок 4.6 – Режимы работы с записями (а – удаление записи,

б – редактирование записи, в – добавление записи)

После нажатия на кнопку удаления, запись безвозвратно будет удалена из базы данных.

После нажатия на кнопку редактирования, пользователю предоставляется возможность отредактировать данные.

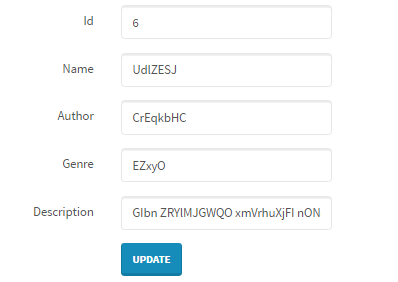


Рисунок 4.7 – Редактирование данных

После нажатия на кнопку добавления, добавляется новая запись в базу данных, рисунок 4.8.

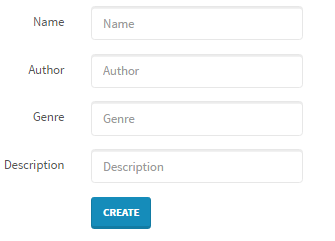


Рисунок 4.8 – Добавление данных

Для более быстрого доступа данным на страницах реализован поиск. Для использования поиска необходимо ввести часть или целиком все слово и нажать на кнопку поиск, рисунок 4.9.

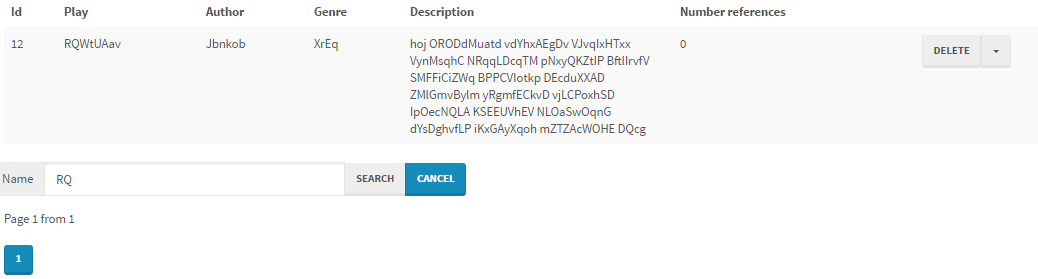


Рисунок 4.9 – Поиск данных

Для начала работы с другими данными, достаточно перейти на другую страницу воспользовавшись меню. Работа с другими данными реализована аналогичным образом.

Для выхода из приложения достаточно закрыть вкладку в браузере, либо браузер целиком.

**5 РУКОВОДСТВО ПРОГРАММИСТА**

**5.1 Назначения и условия применения программы**

Разработанная информационная система позволяет добавлять, редактировать, удалять и отображать данные о заказе билетов театра. Приложение “Информационная система “Онлайновая система распространения билетов в театре” позволит автоматизировать работу подобных организаций.

К компьютеру, на котором будет установлен веб-сервер налагаются следующие условия:

К SQL Server 2012 предъявляются следующие аппаратные и программные требования:

* минимум 256 МБ ОЗУ;
* 32-разрядные системы: компьютер, оборудованный процессором Intel или совместимым процессором с тактовой частотой 1 ГГц и выше;
* 64-разрядные системы: процессор с тактовой частотой 1,4 ГГц и выше;
* поддерживаемые операционные системы: Windows Server 2003 Service Pack 2; Windows Server 2008; Windows Vista; Windows Vista Service Pack 1; Windows XP Service Pack 2; Windows XP Service Pack 3.

К компьютеру, на котором будет выполняться клиентская часть приложения предъявляются следующие требования:

* процессор *Intel Pentium III* 1 ГГц и выше;
* объем оперативного запоминающего устройства 256 Мб;
* видеоадаптер должен поддерживать разрешение 1024х768 точек;
* монитор с возможностью поддерживать разрешение 1024х768 с частотой не менее 60 Гц, отображающий 256 и более цветов;
* устройства ввода информации – клавиатура и манипулятор типа “мышь»;
* 10 Мб свободного места на жёстком диске;
* операционная система *Windows Vista* / *XP /Server 2003* / 7 / 8 /*Server 2008* / *Server 2008 R2* / *Server 2012*;
* *Microsoft.NET Framework* *5*.

Исходный код приложения приведен в приложении А.

**5.2 Характеристики работы программы**

Никаких ограничений и требований на режим работы и запуска программы не налагается. Средства восстановления программы после сбоя не предусмотрены.

Работа начинается после входа на сайт и загрузки главной страницы.

# ЗАКЛЮЧЕНИЕ

В рамках выполнения курсовой работы была спроектирована информационноя система “Онлайновая система распространения билетов в театре”. Проведя анализ предметной области, на его основе разработан основной алгоритм работы разрабатываемого приложения. После чего была разработана база данных, а также веб-приложение для работы с ней. Приложение имеет простой и понятный графический интерфейс, позволяющий пользователю быстро освоиться в системе и начать работу в ней.

Разработанное приложение имеет как общественную так и учебную ценность. С точки зрения общественности, данное приложение может получить хорошее применение в театрах и похожих организациях. Приложение позволит автоматизировать работу организации, а именно продажа и покупка билетов организации. С точки зрения обучения, разработка данного приложения помогла приобрести и укрепить навыки работы с базами данных в среде СУБД MS SQL Server 2012, а также навыки создания и использования ASP.NET MVC приложений.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Горев, А. Эффективная работа с СУБД / А.Горев, Р.Ахаян, С.Макашарипов. – СПб.: Питер, 1997. – 74 с.Эспозито Д. Знакомство с Microsoft ASP.NET 2.0 - Москва; Санкт-Петербург: Русская редакция: Питер, 2006. - 490 с.
2. Stack Overflow на русском [Электронный ресурс]. – Режим доступа: <http://ru.stackoverflow.com/questions/69743> – Дата доступа: 10.12.2015.
3. Википедия - Свободная Энциклопедия [Электронный ресурс]. – Режим доступа: <https://ru.wikipedia.org/wiki/ERwin_Data_Modeler> – Дата доступа: 11.12.2015.
4. Microsoft MSDN [Электронный ресурс]. – Режим доступа: <https://msdn.microsoft.com/ru-ru/library/dd410120(v=vs.100).aspx> – Дата доступа: 11.12.2015.
5. Гаряка А.А. Основы ASP.NET 2.0/ [Электронный ресурс]. – Режим доступа: http://www.intuit.ru/department/se/aspdotnet – Дата доступа: 22.05.2013.

# 

# ПРИЛОЖЕНИЕ А

(обязательное)

**Листинг программных модулей**

\_Layout.chtml

<!DOCTYPE html>

<html>

<head>

<meta name="viewport" content="width=device-width" />

<link href="~/Content/bootstrap-lumen.css" rel="stylesheet" />

<script type="text/javascript" src="~/Scripts/sorttable.js"></script>

<script type="text/javascript" src="~/Scripts/jquery-1.9.1.js"></script>

<script type="text/javascript" src="~/Scripts/admin.js"></script>

<script type="text/javascript" src="~/Scripts/bootstrap.js"></script>

<link type="text/css" rel="stylesheet" href="~/Content/Site.css" />

@RenderSection("JavaScript", required: false)

</head>

<body>

<div>

<div class="row">

<div class="col-lg-1"></div>

<div class="col-lg-10">

@Html.Partial("\_NavBar")

@RenderBody()

@Html.Partial("\_Footer")

</div>

<div class="col-lg-3"></div>

</div>

</div>

</body>

</html>

Контроллер HomeController

using System;

using System.Collections.Generic;

using System.Data.OleDb;

using System.Data.SqlClient;

using System.Globalization;

using System.Linq;

using System.Threading;

using System.Web;

using System.Web.Mvc;

using Theater.Models.Attribute;

using Theater.Models.Plays;

using Theater.WorkingDb.Connections;

using Theater.WorkingDb.Interfaces;

using PagedList;

namespace Theater.Controllers

{

[Culture]

public class HomeController : Controller

{

private static IDateDao datesDb;

private static IPlayDao playsDb;

private static IAuthorDao authorsDb;

private static IGenreDao genresDb;

private static int pageSize = 5;

public HomeController()

{

authorsDb = AuthorsTableConnection.Instance;

genresDb = GenresTableConnection.Instance;

datesDb = DatesTableConnection.Instance;

playsDb = PlaysTableConnection.Instance;

}

// GET: Home

public ActionResult Index(int page = 1)

{

ViewBag.Authors = authorsDb.GetAllAuthors();

ViewBag.Genres = genresDb.GetAllGenres();

List<DatePlay> sortedDates = datesDb.GetAllDates().OrderBy(x => x.Date).ToList();

ViewBag.Dates = sortedDates;

var sortedPlays = playsDb.GetAllPlays().OrderBy(a => a.Name).ToList()

.Where(play => sortedDates.Where(date=> date.PlayId == play.Id).Count() > 0);

ViewBag.Plays = sortedPlays.ToPagedList(page, pageSize);

return View();

}

[HttpPost]

public ActionResult ChangeLang(string lang)

{

string returnUrl = Request.UrlReferrer.AbsolutePath;

Session["lang"] = lang;

return Redirect(returnUrl);

}

}

}

Контроллер AccountController

using PagedList;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Web.Mvc;

using System.Web.Security;

using Theater.Models;

using Theater.Models.Account;

using Theater.Models.Attribute;

using Theater.Models.AuthorizeAttributes;

using Theater.Models.Plays;

using Theater.WorkingDb.Connections;

using Theater.WorkingDb.Interfaces;

namespace Theater.Controllers

{

[Culture]

public class AccountController : Controller

{

private static ILoginDao loginsDb;

private static IDateDao datesDb;

private static IOrderDao ordersDb;

private static IPlayDao playsDb;

private static int pageSize = 10;

private static int normalyPageSize = 10;

private static int pageSizeForFilter = 1000;

public AccountController()

{

loginsDb = LoginsTableConnection.Instance;

datesDb = DatesTableConnection.Instance;

ordersDb = OrdersTableConnection.Instance;

playsDb = PlaysTableConnection.Instance;

}

// GET: Account

public ActionResult Index()

{

return View();

}

// GET: /Account/Register

[HttpGet]

[NoAuthorize]

public ActionResult Register()

{

//var newUser = new User();

return View();

}

[HttpPost]

[NoAuthorize]

public ActionResult Register(User user, string confirmPassword)

{

if (isTrueUser(user, confirmPassword))

{

loginsDb.AddAccount(user);

CreateAuthCookie(user);

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

}

else

{

return View();

}

}

// GET: /Account/Login

[HttpGet]

[NoAuthorize]

public ActionResult Login()

{

return View();

}

[HttpPost]

[NoAuthorize]

public ActionResult Login(string email, string password)

{

User user = loginsDb.GetUserByEmailAndPassword(email, password);

if (user == null)

{

ModelState.AddModelError("Incorrectrly entered", Resources.Resource.ErrorLoginInput);

return View();

}

else

{

CreateAuthCookie(user);

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

}

}

// GET: /Account/Logout

[Authorize]

public ActionResult Logout()

{

FormsAuthentication.SignOut();

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

}

// GET: /Account/Cart

[HttpGet]

[Authorize]

public ActionResult Cart(int page = 1)

{

ViewBag.Dates = datesDb.GetAllDates();

List<Order> filterOrder = (TempData["OrderList"] as List<Order>);

List<Order> orders = filterOrder ?? ordersDb.GetOrdersByIdLogin(CurrentUserService.GetCurrentUser().Id)

.OrderBy( x=> datesDb.GetDateById(x.DateId).Date).ToList();

ViewBag.Plays = playsDb.GetAllPlays();

int sizeForPage = pageSize;

pageSize = normalyPageSize;

page = truePage(page, (int)Math.Ceiling((double)orders.Count / sizeForPage));

ViewBag.Plays = playsDb.GetAllPlays();

ViewBag.Logins = loginsDb.GetAllLogins();

return View(orders.ToPagedList(page, sizeForPage));

}

/// <summary>

/// Creates cookie of authorize for person

/// </summary>

/// <param name="user">user, who authorize</param>

/// <param name="timeout">time out for cookie</param>

/// <param name="persistent">works cookie for close browser</param>

private void CreateAuthCookie(User user)

{

DateTime timeoutCookie = DateTime.Now.AddMinutes(15);

bool persistentCookie = false;

var ticket = new FormsAuthenticationTicket(1, user.Email, DateTime.Now,

timeoutCookie, persistentCookie, user.Role.ToString());

var encTicket = FormsAuthentication.Encrypt(ticket);

var cookie = new HttpCookie(FormsAuthentication.FormsCookieName, encTicket);

cookie.Expires = timeoutCookie;

Response.Cookies.Add(cookie);

}

[HttpPost]

public ActionResult GetOrders(int? countTickets)

{

try

{

if (countTickets != null)

{

TempData["orderList"] = ordersDb.GetOrderByCountTicketsAndUserId((int)countTickets, CurrentUserService.GetCurrentUser().Id);

pageSize = pageSizeForFilter;

}

return RedirectToAction("Cart", "Account");

}

catch

{

return RedirectToAction("Cart", "Account");

}

}

/// <summary>

/// Checks user on correct

/// </summary>

/// <param name="user">User, who need check</param>

/// <param name="confirmPassword">password for confirm</param>

/// <returns>If user correct return true, else return false</returns>

private bool isTrueUser(User user, string confirmPassword)

{

bool trueUser = true;

if (!ValidationAccount.IsTrueName(user.Name))

{

ModelState.AddModelError("Name", Resources.Resource.ErrorName);

trueUser = false;

}

if (!ValidationAccount.IsTruePhone(user.Phone))

{

ModelState.AddModelError("Phone", Resources.Resource.ErrorPhone);

trueUser = false;

}

if (!ValidationAccount.IsTrueEmail(user.Email))

{

ModelState.AddModelError("Email", Resources.Resource.ErrorEmail);

trueUser = false;

}

if (!ValidationAccount.IsTruePassword(user.Password))

{

ModelState.AddModelError("Password", Resources.Resource.ErrorPassword);

trueUser = false;

}

else

{

if (!ValidationAccount.IsTrueConfirmPassword(user.Password, confirmPassword))

{

ModelState.AddModelError("ConfirmPassword", Resources.Resource.ErrorConfirmPassword);

trueUser = false;

}

}

return trueUser;

}

private int truePage(int page, int pageCount)

{

if (page > pageCount)

{

page = pageCount;

}

if (page < 1)

{

page = 1;

}

return page;

}

}

}

Контроллер CourierController

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Web.Mvc;

using Theater.Models.Attribute;

using Theater.Models.Plays;

using Theater.WorkingDb.Connections;

using Theater.WorkingDb.Interfaces;

using Theater.Models.Account;

using PagedList;

namespace Theater.Controllers

{

[Culture]

[Authorize(Roles = "Сourier, Admin")]

public class CourierController : Controller

{

private static ILoginDao loginsDb;

private static IDateDao datesDb;

private static IOrderDao ordersDb;

private static IPlayDao playsDb;

private static int pageSize = 10;

private static int normalyPageSize = 10;

private static int pageSizeForFilter = 1000;

public CourierController()

{

loginsDb = LoginsTableConnection.Instance;

datesDb = DatesTableConnection.Instance;

ordersDb = OrdersTableConnection.Instance;

playsDb = PlaysTableConnection.Instance;

}

// GET: Courier

public ActionResult Index()

{

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

}

// GET: Courier/Orders

public ActionResult Orders(int page = 1)

{

ViewBag.Dates = datesDb.GetAllDates();

List<Order> filterOrder = (TempData["OrderList"] as List<Order>);

List<Order> orders = (filterOrder ?? ordersDb.GetAllOrders()).Where(order => order.Status != StatusOrder.Completed)

.OrderBy(x => datesDb.GetDateById(x.DateId).Date).ToList();

int sizeForPage = pageSize;

pageSize = normalyPageSize;

page = truePage(page, (int)Math.Ceiling((double)orders.Count / sizeForPage));

ViewBag.Plays = playsDb.GetAllPlays();

ViewBag.Logins = loginsDb.GetAllLogins();

return View(orders.ToPagedList(page, sizeForPage));

}

[HttpPost]

public ActionResult GetOrders(int? countTickets)

{

try

{

if (countTickets != null)

{

TempData["orderList"] = ordersDb.GetOrderByCountTickets((int)countTickets);

pageSize = pageSizeForFilter;

}

return RedirectToAction("Orders", "Courier");

}

catch

{

return RedirectToAction("Orders", "Courier");

}

}

[HttpPost]

public ActionResult ChangeStatusOrder(int orderId, int? statusId)

{

if (statusId != null)

{

ordersDb.UpdateOrderStatusById(orderId, (int)statusId);

}

return RedirectToAction("Orders", "Courier");

}

private int truePage(int page, int pageCount)

{

if (page > pageCount)

{

page = pageCount;

}

if (page < 1)

{

page = 1;

}

return page;

}

}

}

Контроллер PlaysController

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Web.Mvc;

using Theater.Models;

using Theater.Models.Attribute;

using Theater.Models.Plays;

using Theater.Models.Theater;

using Theater.WorkingDb;

using Theater.WorkingDb.Connections;

using Theater.WorkingDb.Interfaces;

namespace Theater.Controllers

{

[Culture]

public class PlaysController : Controller

{

private static ILoginDao loginsDb;

private static IDateDao datesDb;

private static IOrderDao ordersDb;

private static IPlayDao playsDb;

private static IAuthorDao authorsDb;

private static IGenreDao genresDb;

public PlaysController()

{

loginsDb = LoginsTableConnection.Instance;

datesDb = DatesTableConnection.Instance;

ordersDb = OrdersTableConnection.Instance;

playsDb = PlaysTableConnection.Instance;

authorsDb = AuthorsTableConnection.Instance;

genresDb = GenresTableConnection.Instance;

}

// GET: Plays

public ActionResult Index()

{

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

}

// GET: Plays/Description

public ActionResult Description(int id)

{

try

{

Play play = playsDb.GetPlayById(id);

ViewBag.Play = play;

ViewBag.Genre = genresDb.GetGenreById(play.GenreId);

ViewBag.Author = authorsDb.GetAuthorById(play.AuthorId);

ViewBag.Dates = datesDb.GetDatesByIdPlay(play.Id).OrderBy(x => x.Date).ToList();

}

catch (NullReferenceException)

{

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

}

return View();

}

//Plays/Order/[id]

[HttpGet]

public ActionResult Order(int id)

{

try

{

getViewBagForOrderPage(id);

}

catch (NullReferenceException)

{

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

}

return View();

}

[HttpPost]

[Authorize]

public ActionResult Order(Order order, int dateId)

{

DatePlay date = datesDb.GetDateById(dateId);

Play play = PlaysTableConnection.Instance.GetPlayById(date.PlayId);

getViewBagForOrderPage(dateId);

if (date.Date < DateTime.Now)

{

ModelState.AddModelError("Error order time!", Resources.Resource.ErrorOrderTime);

return View();

}

if (!isTrueOrder(order, dateId))

{

ModelState.AddModelError("Error order!", Resources.Resource.ErrorOrderNumber);

return View();

}

else

{

try

{

ordersDb.AddOrder(new Order(0,

dateId,

CurrentUserService.GetCurrentUser().Id,

(int)order.Category,

order.Quantity,

TheaterInformation.GetPriceByCategoryId((int)order.Category) \* order.Quantity,

0));

return View("OrderAccepted");

}

catch (InvalidOperationException)

{

ModelState.AddModelError("Error order!", Resources.Resource.ErrorOrder);

return View();

}

}

}

[HttpPost]

[Authorize]

public ActionResult DeleteOrder(int orderId)

{

ordersDb.DeleteOrderById(orderId);

return RedirectToAction("Cart", "Account");

}

//Plays/OrderAccepted

[Authorize]

public ActionResult OrderAccepted()

{

return View();

}

/// <summary>

/// Gets all need information in ViewBag

/// </summary>

/// <param name="id">Date id</param>

private void getViewBagForOrderPage(int id)

{

ViewBag.DateId = id;

DatePlay currentDate = datesDb.GetDateById(id);

Play play = playsDb.GetPlayById(currentDate.PlayId);

ViewBag.Play = play;

ViewBag.Genre = genresDb.GetGenreById(play.GenreId);

ViewBag.Author = authorsDb.GetAuthorById(play.AuthorId);

ViewBag.Dates = datesDb.GetDatesByIdPlay(play.Id).OrderBy(x => x.Date).ToList();

ViewBag.TotalCountBalconySeats = TheaterInformation.TotalCountBalconySeats;

ViewBag.PriceBalconySeats = TheaterInformation.PriceBalcony;

ViewBag.FreeBalconySeats = (TheaterInformation.TotalCountBalconySeats -

ordersDb.GetCountBusySeetsByDateIdAndCategory(id, 0));

ViewBag.TotalCountParterreSeats = TheaterInformation.TotalCountParterreSeats;

ViewBag.PriceParterreSeats = TheaterInformation.PriceParterre;

ViewBag.FreeParterreSeats = (TheaterInformation.TotalCountParterreSeats -

ordersDb.GetCountBusySeetsByDateIdAndCategory(id, 1));

}

/// <summary>

/// Checks order on correct

/// </summary>

/// <param name="order">Order, which need check</param>

/// <param name="orders">databas, where saves orders</param>

/// <param name="dateId">date id, when will be play</param>

/// <returns>If order correct return true, else false</returns>

private bool isTrueOrder(Order order, int dateId)

{

int count1 = TheaterInformation.TotalCountBalconySeats - ordersDb.GetCountBusySeetsByDateIdAndCategory(dateId, 0);

int count2 = TheaterInformation.TotalCountParterreSeats - ordersDb.GetCountBusySeetsByDateIdAndCategory(dateId, 1);

return (((order.Category == Category.Balcony && order.Quantity < count1) ||

(order.Category == Category.Parterre && order.Quantity < count2))

&& order.Quantity > 0);

}

}

}

Контроллер AdminController

using PagedList;

using System;

using System.Collections;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Web.Mvc;

using Theater.Models.Attribute;

using Theater.Models.Plays;

using Theater.WorkingDb.Connections;

using Theater.WorkingDb.Interfaces;

namespace Theater.Controllers

{

[Culture]

[Authorize(Roles = "Admin")]

public class AdminController : Controller

{

private static IAuthorDao authorsDb;

private static IGenreDao genresDb;

private static IPlayDao playsDb;

private static IDateDao datesDb;

private static int pageSize = 10;

private static int normalyPageSize = 10;

private static int pageSizeForFilter = 1000;

private static string messagesAuthorTable;

private static string messagesGenreTable;

private static string messagesPlayTable;

private static string messagesDatePlayTable;

public AdminController()

{

authorsDb = AuthorsTableConnection.Instance;

genresDb = GenresTableConnection.Instance;

playsDb = PlaysTableConnection.Instance;

datesDb = DatesTableConnection.Instance;

}

// GET: Admin

public ActionResult Index()

{

return View("Admin");

}

#region AuthorsCrud

public ActionResult AuthorsTable(int page = 1)

{

ModelState.AddModelError("messages", messagesAuthorTable);

messagesAuthorTable = null;

List<Author> filterAuthors = TempData["AuthorList"] as List<Author>;

List<Author> authors = filterAuthors ?? authorsDb.GetAllAuthors();

List<Play> plays = playsDb.GetAllPlays();

Hashtable idAndCountInPlays = new Hashtable();

authors.ForEach(author => idAndCountInPlays.Add(author.Id, plays.Where(play => play.AuthorId == author.Id).Count()));

ViewBag.idAndCountInPlay = idAndCountInPlays;

int sizeForPage = pageSize;

pageSize = normalyPageSize;

page = truePage(page, (int)Math.Ceiling((double)authors.Count / pageSize));

return View(authors.ToPagedList(page, pageSize));

}

[HttpPost]

public ActionResult AuthorDelete(int id)

{

try

{

if (authorsDb.GetAuthorById(id) != null)

{

authorsDb.DeleteById(id);

messagesAuthorTable = Resources.Resource.MessageDeleted;

}

else

{

messagesAuthorTable = Resources.Resource.ErrorFound;

}

return RedirectToAction("AuthorsTable", "Admin");

}

catch

{

messagesAuthorTable = Resources.Resource.Error;

return RedirectToAction("AuthorsTable", "Admin");

}

}

[HttpPost]

public ActionResult CreateAuthor(Author author)

{

try

{

authorsDb.Create(author);

messagesAuthorTable = Resources.Resource.MessageCreated;

return RedirectToAction("AuthorsTable", "Admin");

}

catch

{

messagesAuthorTable = Resources.Resource.Error;

return RedirectToAction("AuthorsTable", "Admin");

}

}

[HttpPost]

public ActionResult AuthorUpdate(Author author)

{

try

{

authorsDb.Update(author);

messagesAuthorTable = Resources.Resource.MessageUpdated;

return RedirectToAction("AuthorsTable", "Admin");

}

catch

{

messagesAuthorTable = Resources.Resource.Error;

return RedirectToAction("AuthorsTable", "Admin");

}

}

[HttpPost]

public ActionResult GetAuthorsByName(string name)

{

try

{

TempData["AuthorList"] = authorsDb.GetAuthorsByName(name);

pageSize = pageSizeForFilter;

return RedirectToAction("AuthorsTable", "Admin");

}

catch

{

return RedirectToAction("AuthorsTable", "Admin");

}

}

#endregion

#region GenresCrud

public ActionResult GenresTable(int page = 1)

{

ModelState.AddModelError("messages", messagesGenreTable);

messagesGenreTable = null;

List<Genre> filterGenres = TempData["GenreList"] as List<Genre>;

List<Genre> genres = filterGenres ?? genresDb.GetAllGenres();

List<Play> plays = playsDb.GetAllPlays();

Hashtable idAndCountInPlays = new Hashtable();

genres.ForEach(genre => idAndCountInPlays.Add(genre.Id, plays.Where(play => play.GenreId == genre.Id).Count()));

ViewBag.idAndCountInPlay = idAndCountInPlays;

int sizeForPage = pageSize;

pageSize = normalyPageSize;

page = truePage(page, (int)Math.Ceiling((double)genres.Count / pageSize));

return View(genres.ToPagedList(page, pageSize));

}

[HttpPost]

public ActionResult GenreDelete(int id)

{

try

{

if (genresDb.GetGenreById(id) != null)

{

genresDb.DeleteById(id);

messagesGenreTable = Resources.Resource.MessageDeleted;

}

else

{

messagesGenreTable = Resources.Resource.ErrorFound;

}

return RedirectToAction("GenresTable", "Admin");

}

catch

{

messagesGenreTable = Resources.Resource.Error;

return RedirectToAction("GenresTable", "Admin");

}

}

[HttpPost]

public ActionResult CreateGenre(Genre genre)

{

try

{

genresDb.Create(genre);

messagesGenreTable = Resources.Resource.MessageCreated;

return RedirectToAction("GenresTable", "Admin");

}

catch

{

messagesGenreTable = Resources.Resource.Error;

return RedirectToAction("GenresTable", "Admin");

}

}

[HttpPost]

public ActionResult GenreUpdate(Genre genre)

{

try

{

genresDb.Update(genre);

messagesGenreTable = Resources.Resource.MessageUpdated;

return RedirectToAction("GenresTable", "Admin");

}

catch

{

messagesGenreTable = Resources.Resource.Error;

return RedirectToAction("GenresTable", "Admin");

}

}

[HttpPost]

public ActionResult GetGenresByName(string name)

{

try

{

TempData["GenreList"] = genresDb.GetGenresByName(name);

pageSize = pageSizeForFilter;

return RedirectToAction("GenresTable", "Admin");

}

catch

{

return RedirectToAction("GenresTable", "Admin");

}

}

#endregion

#region PlaysCrud

public ActionResult PlaysTable(int page = 1)

{

ModelState.AddModelError("messages", messagesPlayTable);

messagesPlayTable = null;

List<Play> filterPlays = TempData["PlayList"] as List<Play>;

List<Play> plays = filterPlays ?? playsDb.GetAllPlays();

Hashtable idAndCountInPlays = new Hashtable();

plays.ForEach(play => idAndCountInPlays.Add(play.Id, datesDb.GetDatesByIdPlay(play.Id).Count()));

ViewBag.idAndCountInPlay = idAndCountInPlays;

int sizeForPage = pageSize;

pageSize = normalyPageSize;

page = truePage(page, (int)Math.Ceiling((double)plays.Count / pageSize));

ViewBag.Authors = authorsDb.GetAllAuthors();

ViewBag.Genres = genresDb.GetAllGenres();

return View(plays.ToPagedList(page, pageSize));

}

[HttpPost]

public ActionResult PlayDelete(int id)

{

try

{

if (playsDb.GetPlayById(id) != null)

{

playsDb.DeleteById(id);

messagesPlayTable = Resources.Resource.MessageDeleted;

}

else

{

messagesPlayTable = Resources.Resource.ErrorFound;

}

return RedirectToAction("PlaysTable", "Admin");

}

catch

{

messagesPlayTable = Resources.Resource.Error;

return RedirectToAction("PlaysTable", "Admin");

}

}

[HttpPost]

public ActionResult CreatePlay(Play play, string authorName, string genreName)

{

try

{

play.AuthorId = authorsDb.GetAuthorsByName(authorName).First().Id;

play.GenreId = genresDb.GetGenresByName(genreName).First().Id;

playsDb.Create(play);

messagesPlayTable = Resources.Resource.MessageCreated;

return RedirectToAction("PlaysTable", "Admin");

}

catch (Exception ex)

{

messagesPlayTable = Resources.Resource.Error;

return RedirectToAction("PlaysTable", "Admin");

}

}

[HttpPost]

public ActionResult PlayUpdate(Play play, string authorName, string genreName)

{

try

{

play.AuthorId = authorsDb.GetAuthorsByName(authorName).First().Id;

play.GenreId = genresDb.GetGenresByName(genreName).First().Id;

playsDb.Update(play);

messagesPlayTable = Resources.Resource.MessageUpdated;

return RedirectToAction("PlaysTable", "Admin");

}

catch

{

messagesPlayTable = Resources.Resource.Error;

return RedirectToAction("PlaysTable", "Admin");

}

}

[HttpPost]

public ActionResult GetPlaysByName(string name)

{

try

{

TempData["PlayList"] = playsDb.GetPlaysByName(name);

pageSize = pageSizeForFilter;

return RedirectToAction("PlaysTable", "Admin");

}

catch

{

return RedirectToAction("PlaysTable", "Admin");

}

}

#endregion

#region DatePlaysCrud

public ActionResult DatePlaysTable(int page = 1)

{

ModelState.AddModelError("messages", messagesDatePlayTable);

messagesDatePlayTable = null;

List<DatePlay> filterDate = TempData["Date"] as List<DatePlay>;

List<DatePlay> dates = filterDate ?? datesDb.GetAllDates();

int sizeForPage = pageSize;

pageSize = normalyPageSize;

page = truePage(page, (int)Math.Ceiling((double)dates.Count / pageSize));

ViewBag.Plays = playsDb.GetAllPlays();

return View(dates.ToPagedList(page, pageSize));

}

[HttpPost]

public ActionResult DatePlayDelete(int id)

{

try

{

if (datesDb.GetDateById(id) != null)

{

datesDb.DeleteById(id);

messagesDatePlayTable = Resources.Resource.MessageDeleted;

}

else

{

messagesDatePlayTable = Resources.Resource.ErrorFound;

}

return RedirectToAction("DatePlaysTable", "Admin");

}

catch

{

messagesDatePlayTable = Resources.Resource.Error;

return RedirectToAction("DatePlaysTable", "Admin");

}

}

[HttpPost]

public ActionResult DatePlayCreate(DatePlay date, string dateCheck, string playName)

{

try

{

DateTime newDate;

if (DateTime.TryParse(dateCheck, out newDate) &&

datesDb.GetAllDates().Where(datePlay => datePlay.Date == newDate).Count() == 0)

{

date.PlayId = playsDb.GetPlaysByName(playName).First().Id;

date.Date = newDate;

datesDb.Create(date);

messagesDatePlayTable = Resources.Resource.MessageCreated;

return RedirectToAction("DatePlaysTable", "Admin");

}

else

{

messagesDatePlayTable = Resources.Resource.Error;

return RedirectToAction("DatePlaysTable", "Admin");

}

}

catch (Exception ex)

{

messagesDatePlayTable = Resources.Resource.Error;

return RedirectToAction("DatePlaysTable", "Admin");

}

}

[HttpPost]

public ActionResult DatePlayUpdate(DatePlay date, string dateCheck, string playName)

{

try

{

DateTime newDate;

if (DateTime.TryParse(dateCheck, out newDate) &&

(datesDb.GetAllDates().Where(datePlay => datePlay.Date == newDate).Count() == 0 ||

datesDb.GetAllDates().First(datePlay => datePlay.Date == newDate).Id == date.Id))

{

date.PlayId = playsDb.GetPlaysByName(playName).First().Id;

date.Date = newDate;

datesDb.Update(date);

messagesDatePlayTable = Resources.Resource.MessageUpdated;

return RedirectToAction("DatePlaysTable", "Admin");

}

else

{

messagesDatePlayTable = Resources.Resource.Error;

return RedirectToAction("DatePlaysTable", "Admin");

}

}

catch (Exception e)

{

messagesDatePlayTable = Resources.Resource.Error;

return RedirectToAction("DatePlaysTable", "Admin");

}

}

[HttpPost]

public ActionResult GetDatesByPlayName(string playName)

{

try

{

TempData["Date"] = datesDb.GetDatesByPlayIds(playsDb.GetPlaysByName(playName).Select(a => a.Id).ToList());

pageSize = pageSizeForFilter;

return RedirectToAction("DatePlaysTable", "Admin");

}

catch

{

return RedirectToAction("DatePlaysTable", "Admin");

}

}

#endregion

private int truePage(int page, int pageCount)

{

if (page > pageCount)

{

page = pageCount;

}

if (page < 1)

{

page = 1;

}

return page;

}

}

}

Представление контроллера HomeController (Index.cshtml)

@using Theater.Models.Plays;

@using PagedList;

@using PagedList.Mvc;

@{

List<DatePlay> dates = (List<DatePlay>)ViewBag.Dates;

List<Genre> genres = (List<Genre>)ViewBag.Genres;

List<Author> authors = (List<Author>)ViewBag.Authors;

IPagedList<Play> plays = ViewBag.Plays;

Layout = "~/Views/Shared/\_Layout.cshtml";

}

<!DOCTYPE html>

<html>

<head>

<title>Theater</title>

</head>

<body>

@foreach (var play in plays)

{

var playDates = dates.Where(x => x.PlayId == play.Id).ToList();

if (playDates != null && playDates.Count > 0)

{

<div class="text-center">

<h3>@Html.ActionLink(play.Name, "Description", "Plays", new { id = play.Id }, new { @class = "" })</h3>

<h4>

@Resources.Resource.Genre:

@foreach (var genre in genres)

{

if (genre.Id == play.GenreId)

{

@genre.Name

break;

}

}

</h4>

<h4>

@Resources.Resource.Author:

@foreach (var author in authors)

{

if (author.Id == play.AuthorId)

{

@author.Name

break;

}

}

</h4>

<ul class="pagination">

@foreach (var date in playDates)

{

if (date.PlayId == play.Id)

{

<li>

@Html.ActionLink(date.Date.ToShortDateString(), "Order", "Plays", new { id = date.Id }, new { @class = "" })

</li>

}

}

</ul>

</div>

}

}

@Resources.Resource.Page @(plays.PageCount < plays.PageNumber ? 0 : plays.PageNumber) @Resources.Resource.of @plays.PageCount

@Html.PagedListPager(plays, page => Url.Action("Index", new { page }))

</body>

</html>

Представление таблицы Plays (PlaysTable.cshtml)

@using PagedList;

@using PagedList.Mvc;

@using Theater.Models.Plays;

@using System.Collections;

@model IPagedList<Play>

@{

ViewBag.Title = "PlaysTable";

Layout = "~/Views/Shared/\_Layout.cshtml";

List<Author> authors = ViewBag.Authors;

List<Genre> genres = ViewBag.Genres;

Hashtable idAndCount = ViewBag.idAndCountInPlay;

}

<title>Plays</title>

@Html.ActionLink(Resources.Resource.Back, "Index", "Admin", null, new { @class = "btn btn-link" })

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

<div class="row">

<div class="col-lg-6">

@using (Html.BeginForm("CreatePlay", "Admin", FormMethod.Post, new { @class = "form-horizontal" }))

{

<form class="form-horizontal">

<fieldset>

<div class="form-group">

<label for="Name" class="col-lg-2 control-label">@Resources.Resource.NameField</label>

<div class="col-lg-5">

<input name="Name" type="text" class="form-control" id="inputName" placeholder=@Resources.Resource.NameField>

</div>

</div>

<div class="form-group">

<label for="Author" class="col-lg-2 control-label">@Resources.Resource.Author</label>

<div class="col-lg-5">

<input name="authorName" type="text" class="form-control" id="input-author" placeholder="@Resources.Resource.Author">

</div>

</div>

<div class="form-group">

<label for="Genre" class="col-lg-2 control-label">@Resources.Resource.Genre</label>

<div class="col-lg-5">

<input name="genreName" type="text" class="form-control" id="input-genre" placeholder="@Resources.Resource.Genre">

</div>

</div>

<div class="form-group">

<label for="Description" class="col-lg-2 control-label">@Resources.Resource.Description</label>

<div class="col-lg-5">

<input name="Description" type="text" class="form-control" id="input-description" placeholder=@Resources.Resource.Description>

</div>

</div>

<div class="form-group">

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

<button type="submit" class="btn btn-primary">@Resources.Resource.Create</button>

</div>

</div>

</fieldset>

</form>

}

</div>

<div class="col-lg-6 hide" id="update-form">

@using (Html.BeginForm("PlayUpdate", "Admin", FormMethod.Post, new { @class = "form-horizontal" }))

{

<form class="form-horizontal">

<fieldset>

<div class="form-group">

<label for="Id" class="col-lg-2 control-label">@Resources.Resource.Id</label>

<div class="col-lg-5">

<input name="Id" type="number" class="form-control" id="id-update" placeholder="@Resources.Resource.Id" />

</div>

</div>

<div class="form-group">

<label for="Name" class="col-lg-2 control-label">@Resources.Resource.NameField</label>

<div class="col-lg-5">

<input name="Name" type="text" class="form-control" id="name-update" placeholder=@Resources.Resource.NameField>

</div>

</div>

<div class="form-group">

<label for="Author" class="col-lg-2 control-label">@Resources.Resource.Author</label>

<div class="col-lg-5">

<input name="authorName" type="text" class="form-control" id="author-update" placeholder="@Resources.Resource.Author">

</div>

</div>

<div class="form-group">

<label for="Genre" class="col-lg-2 control-label">@Resources.Resource.Genre</label>

<div class="col-lg-5">

<input name="genreName" type="text" class="form-control" id="genre-update" placeholder="@Resources.Resource.Genre">

</div>

</div>

<div class="form-group">

<label for="Description" class="col-lg-2 control-label">@Resources.Resource.Description</label>

<div class="col-lg-5">

<input name="Description" type="text" class="form-control" id="description-update" placeholder=@Resources.Resource.Description>

</div>

</div>

<div class="form-group">

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

<button type="submit" class="btn btn-primary">@Resources.Resource.Update</button>

</div>

</div>

</fieldset>

</form>

}

</div>

</div>

<table class="table table-striped table-hover sortable">

<thead>

<tr>

<th>@Resources.Resource.Id</th>

<th>@Resources.Resource.Play</th>

<th>@Resources.Resource.Author</th>

<th>@Resources.Resource.Genre</th>

<th>@Resources.Resource.Description</th>

<th>@Resources.Resource.CountRef</th>

</tr>

</thead>

<tbody>

@foreach (var play in Model)

{

<tr id="play\_\_@play.Id">

<td class="id" id="id\_\_@play.Id">@play.Id</td>

<td class="name" id="name\_\_@play.Id">@play.Name</td>

<td class="author" id="author\_\_@play.Id">@authors.Find(author =>author.Id == play.AuthorId).Name</td>

<td class="genre" id="genre\_\_@play.Id">@genres.Find(genre => genre.Id == play.GenreId).Name</td>

<td width="300px" height="100px" class="description" id="description\_\_@play.Id">@play.Description</td>

<td class="count-ref" id="count-ref\_\_@play.Id">@idAndCount[play.Id]</td>

<td class="buttons-action">

<div class="btn-group">

@using (Html.BeginForm("PlayDelete", "Admin", new { id = play.Id }, FormMethod.Post, new { @class = "form-delete" }))

{

<button type="submit" class="btn btn-default">@Resources.Resource.Delete</button>

}

<a href="#" class="btn btn-default dropdown-toggle" data-toggle="dropdown" aria-expanded="false"><span class="caret"></span></a>

<ul class="dropdown-menu">

<li><a href="#" id="update-play" onclick="FillUpdateFormPlay(@play.Id)">@Resources.Resource.Update</a></li>

</ul>

</div>

</td>

</tr>

}

</tbody>

</table>

<div class="row">

<div class="col-lg-5">

@using (Html.BeginForm("GetPlaysByName", "Admin", FormMethod.Post, new { @class = "form-horizontal" }))

{

<form class="form-horizontal">

<fieldset>

<div class="form-group">

<div class="input-group">

<span class="input-group-addon">@Resources.Resource.NameField</span>

<input name="name" type="text" class="form-control">

<span class="input-group-btn">

<button class="btn btn-default" type="submit">@Resources.Resource.Search</button>

<a href="/Admin/PlaysTable" class="btn btn-primary">@Resources.Resource.Cancel</a>

</span>

</div>

</div>

</fieldset>

</form>

}

</div>

</div>

@Resources.Resource.Page @(Model.PageCount < Model.PageNumber ? Model.PageCount : 1) @Resources.Resource.of @Model.PageCount

@Html.PagedListPager(Model, page => Url.Action("PlaysTable", "Admin", new { page }))