

**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ  
УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ  
ГОМЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ  
УНИВЕРСИТЕТ ИМЕНИ П. О. СУХОГО**

Факультет автоматизированных и информационных систем  
Кафедра «Информационные технологии»  
Специальность 1-40 05 01-01 Информационные системы и технологии (в  
проектировании и производстве)

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**  
к курсовому проекту  
по дисциплине «Разработка приложений баз данных для информационных  
систем»

на тему: **«РАЗРАБОТКА WEB-ПРИЛОЖЕНИЯ БАЗ ДАННЫХ  
«РАДИОСТАНЦИЯ»**

Исполнитель: студент гр. ИТП-31  
Расшивалов Н.И.

Руководитель: доцент  
Асенчик О.Д.

Дата проверки: \_\_\_\_\_  
Дата допуска к защите: \_\_\_\_\_  
Дата защиты: \_\_\_\_\_  
Оценка работы: \_\_\_\_\_

Подписи членов комиссии  
по защите курсового проекта: \_\_\_\_\_

Гомель 2021

## СОДЕРЖАНИЕ

Введение.....	4
1 Логическая и физическая структура базы данных программного обеспечения «Радиостанция» .....	5
1.1 Логическая структура базы данных программного обеспечения «Радиостанция» .....	5
1.2 Физическая структура базы данных программного обеспечения «Радиостанция» .....	9
2 Аппаратное и программное обеспечение информационной системы .....	11
2.1 Общие сведения.....	11
2.1 Настройка приложения.....	12
3 Структура программного обеспечения «радиостанция» .....	13
3.1 Архитектура программного обеспечения «Радиостанция».....	13
3.2 Структура контроллеров приложения «Радиостанция» .....	14
3.3 Структура представлений приложения «Радиостанция» .....	15
3.4 Структура моделей и моделей представлений приложения «Радиостанция» .....	17
3.5 Структура системы аутентификации и авторизации пользователей приложения «Радиостанция» .....	18
4 Руководство пользователя.....	20
4.1 Введение. ....	20
4.2 Назначение и условия применения.....	20
4.3 Подготовка к работе. ....	20
4.4 Описание операций.....	20
4.5 Аварийные ситуации .....	28
4.6 Рекомендации по освоению .....	29
5 Руководство программиста .....	30
5.1 Назначение и условия применения. ....	30
5.2 Характеристики приложения.....	30
5.3 Обращение к приложению.....	30
5.4 Входные и выходные данные. ....	30
5.5 Сообщения.....	30
Заключение .....	31
Список использованных источников .....	32
Приложение А Листинг программы .....	33
Приложение Б Структура web-приложения.....	77

## ВВЕДЕНИЕ

На сегодняшний день развитие информационных технологий играет огромную роль в мировом обществе. Объем информации, который увеличивается с каждым днем, вынуждает тратить на обработку данных большое количество временных и иных затрат. По этой причине становится все более необходимым создание приложений, которые позволяют быстро обрабатывать информацию и предоставлять её пользователю в удобном для него виде.

Для эффективной работы любой из организаций важно наличие какой-либо базы или несколько баз данных. Они позволяют хранить большие объемы информации о различных сведениях.

Для рациональной обработки и предоставления информации, которая хранится в базах данных, необходимы информационные системы. Они упрощают обработку больших объемов данных и предоставляют возможность корректно их предоставить.

Наибольшую популярность занимают *web*-приложения. Благодаря их использованию, пользователю нужен только компьютер с браузером и соединение с интернетом. Для того чтобы совершить обновление *web*-приложения, его необходимо обновить только на сервере, и все сразу же смогут работать с новой версией. Так же большинство *web*-приложений являются кроссбраузерными, что позволяет использовать любую операционную систему.

Целью курсового проекта является разработка *web*-приложения баз данных, которое позволит предоставлять и редактировать различную информацию о музыкальных композициях, альбомах, жанрах, трансляциях и т.д. Зарегистрированному пользователю должны предоставляться данные о исполнителях и их композициях, о группах и составе групп, о жанрах и их описанию, о радиотрансляциях. Сотрудники должны иметь возможность редактировать элементы приложения: добавлять, удалять и изменять различную информацию. Администратору необходимо иметь возможность управления штатом сотрудников.

Задачами курсового проекта являются:

- рассмотрение программного обеспечения необходимого для разработки *web*-приложения;
- проектирование базы данных;
- разработка проекта *web*-приложения;
- проектирование пользовательского интерфейса средствами *.NET Core MVC*.

Для разработки был выбран язык программирования *C#*, с использованием технологий *ASP.NET Core* и *Entity Framework Core*. В качестве источника данных была выбрана система управления базами данных (СУБД) *MS SQL Server*.

# 1 ЛОГИЧЕСКАЯ И ФИЗИЧЕСКАЯ СТРУКТУРА БАЗЫ ДАННЫХ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ «РАДИОСТАНЦИЯ»

## 1.1 Логическая структура базы данных программного обеспечения «Радиостанция»

На логическом уровне база данных представлена следующими сущностями:

– сущность *Records* содержит информацию о названии композиции, жанре композиции, исполнителе композиции, названии альбома, рейтинге композиции и длительности. Имеет связь «многие к одному» с сущностью *Performers* и «один к одному» с сущностью *Genres*. Подробная информация о сущности представлена в таблице 1.1.

Таблица 1.1 – Описание атрибутов сущности *Records*

Название атрибута	Тип данных	Примечание
1	2	3
<i>Id</i>	Числовой	Первичный ключ, идентификатор
<i>CompositionName</i>	Текстовый	Название композиции
<i>PerformerId</i>	Числовой	Внешний ключ для связи с сущностью <i>Performers</i>
<i>GenreId</i>	Числовой	Внешний ключ для связи с сущностью <i>Genres</i>
<i>Album</i>	Текстовый	Название альбома
<i>RecordDate</i>	Дата	Дата выхода композиции
<i>Lasting</i>	Длительность	Длительность композиции
<i>Rating</i>	Числовой	Рейтинг композиции

– Сущность *Performers* содержит информацию о имени и фамилии исполнителя, его группе. Имеет связь «многие к одному» с сущностью *Groups*.

Подробная информация о сущности представлена в таблице 1.2.

Таблица 1.2 – Описание атрибутов сущности *Performers*

Название атрибута	Тип данных	Примечание
1	2	3
<i>Id</i>	Числовой	Первичный ключ, идентификатор
<i>Name</i>	Текстовый	Имя исполнителя
<i>Surname</i>	Текстовый	Фамилия исполнителя
<i>GroupId</i>	Числовой	Внешний ключ для связи с сущностью <i>Groups</i> . Может быть пустым если исполнитель не состоит в группе

– Сущность *Groups* содержит описание группы. Имеет связь «один ко многим» с сущностью *Performers*. Информация о сущности представлена в таблице 1.3.

Таблица 1.3 – Описание атрибутов сущности *Groups*

Название атрибута	Тип данных	Примечание
1	2	3
<i>Id</i>	Числовой	Первичный ключ, идентификатор
<i>Description</i>	Текстовый	Описание группы

– Сущность *Employees* содержит информацию о сотрудниках радиостанции и стоит на стороне отношения «многие» с сущностью *AspNetUsers* и на стороне отношения «один» с сущностью *Positions*. Подробная информация о сущности представлена в таблице 1.4.

Таблица 1.4 – Описание атрибутов сущности *Employees*

Название атрибута	Тип данных	Примечание
1	2	3
<i>Id</i>	Числовой	Первичный ключ, идентификатор
<i>WorkTime</i>	Числовой	Время работы сотрудника
<i>Education</i>	Текстовый	Образование сотрудника
<i>PositionId</i>	Числовой	Внешний ключ для связи с сущностью <i>Positions</i>

Продолжение таблицы 1.4

1	2	3
<i>AspNetUserId</i>	Текстовый	Внешний ключ для связи с сущностью <i>AspNetUsers</i>

– Сущность *Positions* содержит информацию о должностях сотрудников радиостанции, стоит на стороне отношения «один» с сущностью *Employees*. Подробная информация о сущности представлена в таблице 1.5.

Таблица 1.5 – Описание атрибутов сущности *Positions*

Название атрибута	Тип данных	Примечание
1	2	3
<i>Id</i>	Числовой	Первичный ключ, идентификатор
<i>Name</i>	Текстовый	Название должности

– Сущность *Genres* содержит информацию о жанрах, стоит на стороне отношения «один» с сущностью *Records*. Подробная информация о сущности представлена в таблице 1.6.

Таблица 1.6 – Описание атрибутов сущности *Genres*

Название атрибута	Тип данных	Примечание
1	2	3
<i>Id</i>	Числовой	Первичный ключ, идентификатор
<i>GenreName</i>	Текстовый	Название жанра
<i>Description</i>	Текстовый	Описание жанра

– Сущность *Broadcasts* содержит информацию о трансляциях на радиостанции. Имеет связь «один к одному» с сущностью *Records*. Подробная информация о сущности представлена в таблице 1.7.

Таблица 1.7 – Описание атрибутов сущности *Broadcasts*

Название атрибута	Тип данных	Примечание
1	2	3
<i>Id</i>	Числовой	Первичный ключ, идентификатор
<i>DateAndTime</i>	Дата и время	Дата и время трансляции

Продолжение таблицы 1.7

1	2	3
<i>EmployeeId</i>	Числовой	Внешний ключ для связи с сущностью <i>Employees</i>
<i>RecordId</i>	Числовой	Внешний ключ для связи с сущностью <i>Records</i>

– Сущность *HomePageImages* содержит информацию о популярных альбомах. Подробная информация о сущности представлена в таблице 1.8.

Таблица 1.8 – Описание атрибутов сущности *HomePageImages*

Название атрибута	Тип данных	Примечание
1	2	3
<i>Id</i>	Числовой	Первичный ключ, идентификатор
<i>SrcImg</i>	Текстовый	Путь к картинке альбома
<i>ImgCaption</i>	Текстовый	Описание альбома

Также имеется ряд встроенных сущностей *ASP.NET Core Identity*, прямое проектирование которых не производилось, за исключением добавления ряда атрибутов для сущности *AspNetUsers*:

*Name* – текстовый тип, имя пользователя;

*Surname* – текстовый тип, фамилия пользователя;

*MiddleName* – текстовый тип, отчество пользователя.

Разбиение данных на сущности соответствует требованиям как минимум третьей нормальной формы [1]:

– все поля являются простыми и содержат только скалярные значения, каждое поле хранит одно единственное значение;

– соответствует второй нормальной форме так как приведена к первой нормальной форме, отсутствуют частичные зависимости. Это означает, что каждый неключевой атрибут (поле) неприводимо зависит от первичного ключа (ключа отношения).

– Соответствует третьей нормальной форме так, как находится во второй нормальной форме, и отсутствуют транзитивные функциональные зависимости неключевых атрибутов от ключевых.

Диаграмма базы данных представлена на рисунке 1.1.

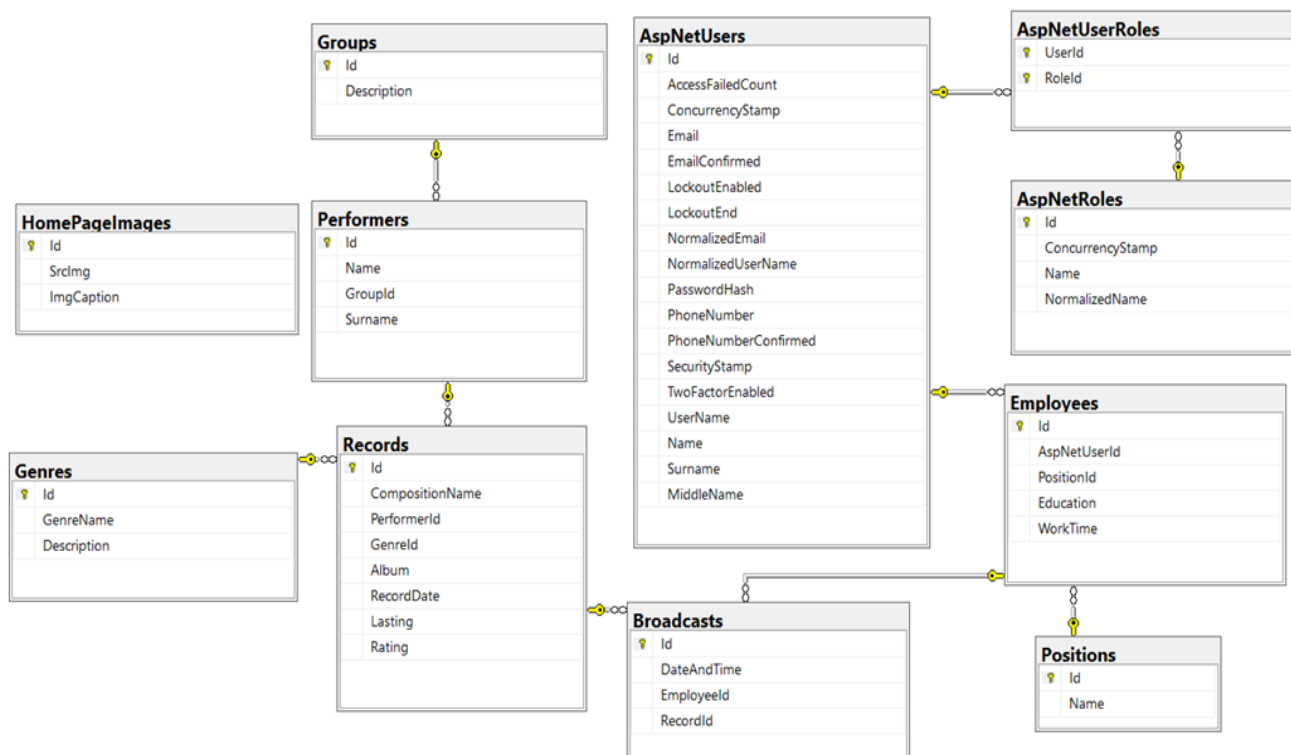


Рисунок 1.1 – Диаграммы базы данных

## 1.2 Физическая структура базы данных программного обеспечения «Радиостанция»

База данных была разработана в СУБД *MS SQL Server* и состоит из двух файлов:

- первичный – содержит сведения, необходимые для запуска базы данных, и ссылки на другие файлы в базе данных, имеет расширение *MDF* и начальный размер 8 МБ с функцией автоматического увеличения до 64 МБ;
- журнал транзакций – содержит информацию для восстановления базы данных, имеет расширение *LDF* и начальный размер 8 МБ с функцией автоматического увеличения до 64 МБ.

Ниже представлена структура таблиц разработанной базы данных.

Структура таблицы *Records*:

- *Id int not null identity(1,1) primary key;*
- *CompositionName varchar(450) not null;*
- *PerformerId int not null;*
- *GenreId int not null;*
- *Album varchar(450) not null;*
- *RecordDate date not null;*
- *Lasting int not null;*
- *Rating decimal(2,1) not null;*



Структура таблицы *Positions*:

- *Id int not null Primary key;*
- *Name varchar(50) not null.*

Структура таблицы *Performers*:

- *Id int not null identity(1,1) primary key;*
- *Name varchar(50) not null;*
- *Surname varchar(50) not null;*
- *GroupId int.*

Структура таблицы *Groups*:

- *Id int not null identity(1,1) primary key;*
- *Description varchar(50).*

Структура таблицы *Groups*:

- *Id int not null identity(1,1) primary key;*
- *GenreName varchar(50) not null;*
- *Description varchar(450).*

Структура таблицы *Employees*:

- *Id int not null identity(1,1) primary key;*
- *AspNetUserId nvarchar(450) not null;*
- *PositionId int not null;*
- *Education nvarchar(450);*
- *WorkTime int.*

Структура таблицы *Broadcasts*:

- *Id int not null identity(1,1) primary key;*
- *DateAndTime Datetime;*
- *EmployeeId int not null;*
- *RecordId int not null.*

Структура таблицы *HomePageItems*:

- *Id int not null Primary key identity;*
- *srcImg nvarchar(450);*
- *ImgCaption nvarchar(450).*

Также при разработке базы данных добавлены ограничения внешнего ключа для обеспечения целостности данных, с запретом на удаление при существовании связанных данных.

## 2 АППАРАТНОЕ И ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ ИНФОРМАЦИОННОЙ СИСТЕМЫ

### 2.1 Общие сведения

Разработанное приложение предназначено для работы с большим потоком данных и использует СУБД *MS SQL Server* [2] для хранения данных, для корректной работы данной СУБД необходимо соблюдение следующих требований:

- рекомендуется минимум 4 ГБ свободного места на диске, требования к месту на диске определяются набором устанавливаемых компонентов *SQL Server* и могут отличаться в зависимости от выбранных компонентов;
- для экспресс-выпуска минимальный объем оперативной памяти: 512 МБ, для всех остальных: 1 ГБ, рекомендованный: 1 ГБ и 4 ГБ соответственно;
- процессор x64 с тактовой частотой не ниже 2 ГГц.

Сервер приложения реализован с помощью технологии *ASP.NET Core* [3, с. 10] и имеет следующие аппаратные требования:

- минимум 4 ГБ свободного места на диске;
- 1 ГБ оперативной памяти (рекомендуется 4 ГБ и выше);
- процессор x86/x64 с тактовой частотой не ниже 2 ГГц.

В качестве клиента может выступать любой браузер, требования могут варьироваться в зависимости от выбранного браузера, в общем случае необходимо соблюдение следующих требований:

- минимум 1 ГБ свободного места на диске;
- 1 ГБ оперативной памяти;
- процессор x86/x64 с тактовой частотой не ниже 1 ГГц.

Требования к программному обеспечению для работы *SQL Server*:

- операционная система *Windows* 8 и выше, поддерживаемые операционные системы для *SQL Server* содержащие встроенное сетевое программное обеспечение необходимое для работы *SQL Server*;
- наличие *.NET Framework* версии 4.1 и выше [3, с. 101].

Для работы сервера приложения необходимо наличие следующего программного обеспечения:

- 32 или 64-разрядные системы, для *Linux*: начиная с *Ubuntu* 16.01, для *macOS* начиная с версии 10.1;
- доступность сетевых протоколов: *TCP/IP*, поддерживающих именованные каналы.

## 2.1 Настройка приложения

Перед началом установки приложения и базы данных на сервер, необходимо убедиться в том, что они удовлетворяют минимальным системным требованиям.

Если база данных для приложения отсутствует, воспользоваться проектом базы данных поставляемым вместе с приложением и опубликовать его на необходимом сервере в СУБД *MS SQL Server*, либо использовать заранее подготовленный скрипт базы данных, содержащий частичную инициализацию данных. Если база данных уже существует никаких дополнительных действий не требуется.

После настройки базы данных необходимо получить строку подключения и добавить ее в конфигурационный файл приложения *appsettings.json*, как показано на рисунке 2.1

```
"ConnectionStrings": {  
  "RadiostationDb": "Data Source=\\SQLEXPRESS;Integrated Security=True;Initial Catalog=RadiostationWebDb;Encrypt=False;"  
}
```

Рисунок 2.1 – Добавление строки подключения к базе в конфигурационный файл

После настройки строки подключения, приложение может быть размещено на любом удобном сервисе, поддерживающем *ASP.NET Core*. Например, можно воспользоваться сервисом *Azure* от *Microsoft*, для публикации приложения достаточно иметь подписку на сервис *Azure*.

Открытие домашней страницы приложения после его настройки свидетельствует о его корректной работе.

Дополнительные возможности в приложении отсутствуют.

При возникновении ошибок подключения к базе данных, проверить правильность указанных данных в конфигурационном файле и перезапустить приложение.

### 3 СТРУКТУРА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ «РАДИОСТАНЦИЯ»

#### 3.1 Архитектура программного обеспечения «Радиостанция»

Общая архитектура разработанного приложения организуется при использовании *MVC* паттерна при помощи взаимодействия моделей, контроллеров и представлений и представлена тремя составными частями [4, с. 15]:

- модели – описывают используемые данные на уровне представления;
- представления – отвечают за пользовательский интерфейс, отображая данные представленные в моделях;
- контроллеры – содержат логику обработки запросов пользователя.

Архитектура уровня представления представлена на рисунке 3.1.

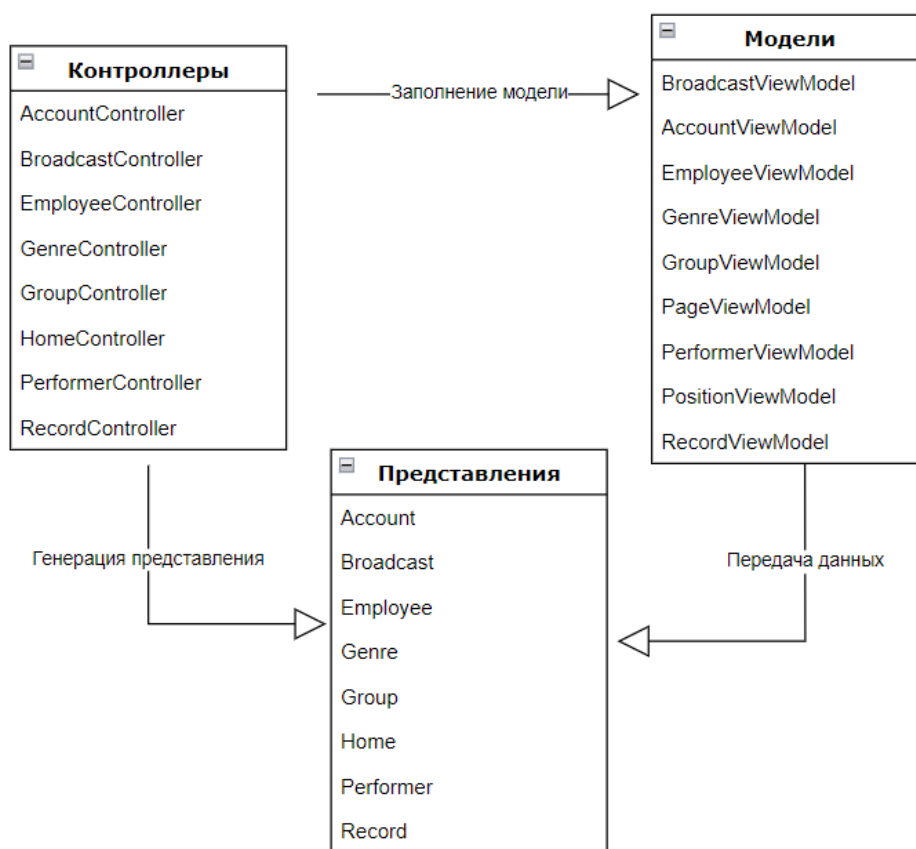


Рисунок 3.1 – Архитектура уровня представления

Общая архитектура приложения заключается в том, что действия пользователя производятся на представлении, после этого представления передают вызов к контроллерам, контроллеры в свою очередь обращаются к изменению модели, а модели сообщают об изменениях в представление.

## 3.2 Структура контроллеров приложения «Радиостанция»

В приложении имеются следующие контроллеры:

- *AccountController* – разработан для предоставления возможности регистрации, авторизации, отображения, удаления, редактирования пользователей, а также для управления ролями пользователей;
- *BroadcastController* – разработан для предоставления возможности отображения, создания, удаления, редактирования радиотрансляций;
- *EmployeeController* – разработан для предоставления возможности отображения сотрудников;
- *HomeController* – разработан для отображения домашней страницы и страницы об ошибке;
- *GenreController* – разработан для предоставления возможности отображения, создания, удаления, редактирования жанров;
- *GroupController* – разработан для предоставления возможности отображения, создания, удаления, редактирования музыкальных групп;
- *PerformerController* – разработан для предоставления возможности отображения, создания, удаления, редактирования музыкальных исполнителей;
- *RecordController* – разработан для предоставления отображения, создания, удаления, редактирования музыкальных композиций.

UML-диаграмма контроллеров представлена на рисунке 3.2.

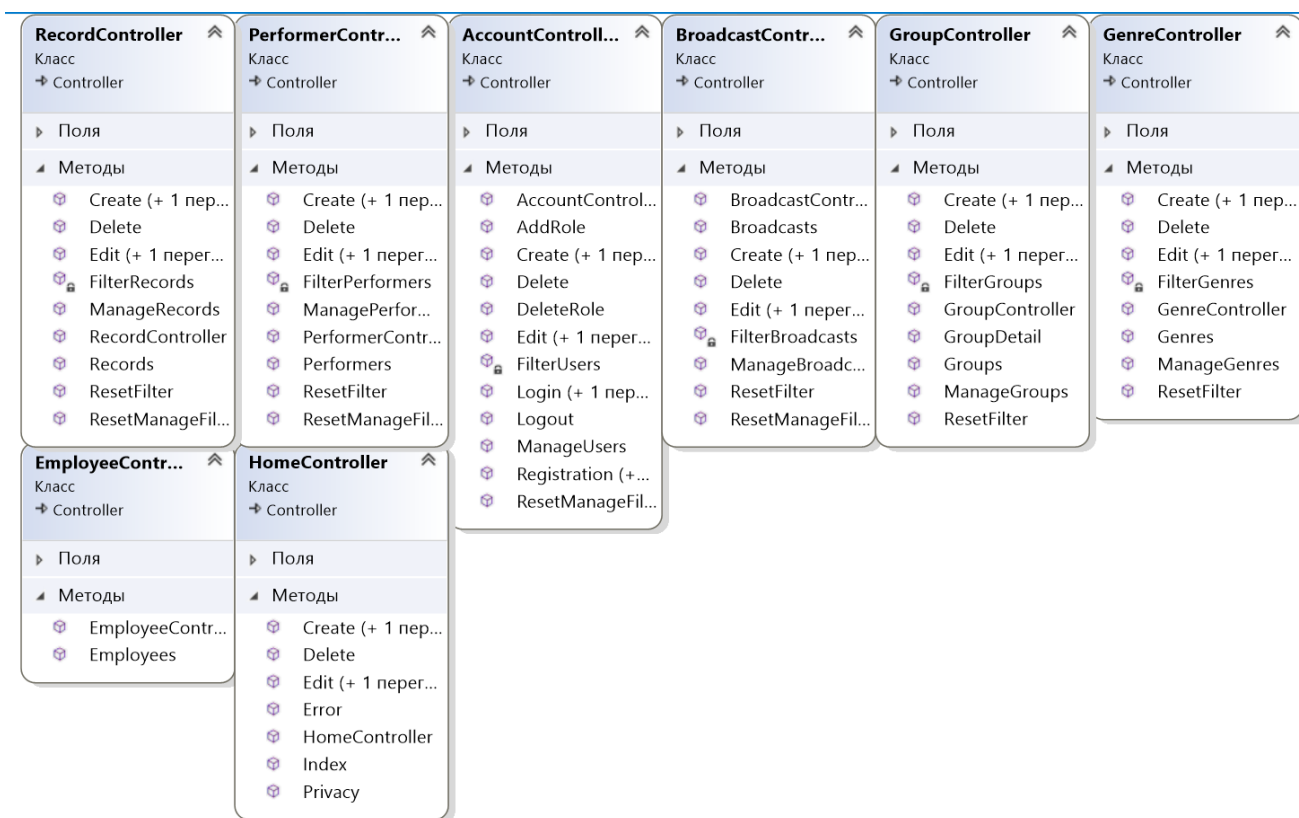


Рисунок 3.2 – UML-диаграмма контроллеров

### 3.3 Структура представлений приложения «Радиостанция»

Для контроллера *AccountController* в папке *Account* присутствуют следующие представления:

- *Login* представляет собой представление, которое отображает страницу для авторизации пользователя;
- *Registration* представляет собой представление, которое отображает страницу для регистрации пользователя;
- *Create* представляет собой представление, которое отображает страницу для создания пользователя администратором приложения;
- *ManageUsers* представляет собой представление, которое отображает страницу для просмотра пользователей приложения, с возможностью редактирования.

Для контроллера *BroadcastController* в папке *Broadcast* присутствуют следующие представления:

- *Create* представляет собой представление, которое отображает страницу для создания радиотрансляции;
- *Edit* представляет собой представление, которое отображает страницу для редактирования информации о радиотрансляции;
- *ManageBroadcasts* представляет собой представление, которое отображает страницу для просмотра и редактирования радиотрансляции;
- *Broadcasts* представляет собой представление, которое отображает страницу для просмотра радиотрансляции.

Для контроллера *EmployeeController* в папке *Employee* присутствуют следующие представления:

- *Employees* представляет собой представление, которое отображает страницу для просмотра сотрудников.

Для контроллера *HomeController* в папке *Home* присутствуют следующие представления:

- *Index* представляет собой представление, которое отображает домашнюю страницу приложения, с информацией о популярных альбомах и их описанию;
- *Error* представляет собой представление, которое отображает страницу ошибки;
- *Edit* представляет собой представление, которое отображает страницу для редактирования информации о популярных альбомах с возможностью смены картинки;
- *Create* представляет собой представление, которое отображает страницу для добавления информации популярных альбомов с возможностью выбора картинки.

Для контроллера *GenreController* в папке *Genre* присутствуют следующие представления:

- *Create* представляет собой представление, которое отображает страницу для создания жанра;
- *Edit* представляет собой представление, которое отображает страницу для редактирования информации о жанре;
- *ManageGenres* представляет собой представление, которое отображает страницу для просмотра и редактирования жанров;
- *Genres* представляет собой представление, которое отображает страницу для просмотра жанров.

Для контроллера *GroupController* в папке *Group* присутствуют следующие представления:

- *Create* представляет собой представление, которое отображает страницу для создания группы;
- *Edit* представляет собой представление, которое отображает страницу для редактирования информации о группе;
- *ManageGroups* представляет собой представление, которое отображает страницу для просмотра и редактирования групп;
- *Groups* представляет собой представление, которое отображает страницу для просмотра групп;
- *GroupDetail* представляет собой представление, которое отображает страницу для просмотра исполнителей групп.

Для контроллера *PerformerController* в папке *Performer* присутствуют следующие представления:

- *Create* представляет собой представление, которое отображает страницу для создания музыкального исполнителя;
- *Edit* представляет собой представление, которое отображает страницу для редактирования информации о музыкальном исполнителе;
- *ManagePerformers* представляет собой представление, которое отображает страницу для просмотра и редактирования музыкальных исполнителей;
- *Performers* представляет собой представление, которое отображает страницу для просмотра музыкальных исполнителей.

Для контроллера *RecordController* в папке *Record* присутствуют следующие представления:

- *Create* представляет собой представление, которое отображает страницу для создания музыкальной композиции;
- *Edit* представляет собой представление, которое отображает страницу для редактирования информации о музыкальной композиции;
- *ManageRecords* представляет собой представление, которое отображает страницу для просмотра и редактирования музыкальных композиций;

– *Records* представляет собой представление, которое отображает страницу для просмотра музыкальных композиций.

Также в папке *Shared* находятся представления, используемые как макет:

– *\_Layout* представляет собой представление, которое содержит макет страницы с навигационным меню сайта и подвалом сайта.

### 3.4 Структура моделей и моделей представлений приложения «Радиостанция»

Приложение представлено рядом моделей, которые проецируют сущности базы данных:

- *Broadcast* представляет собой информацию о радиотрансляции;
- *Record* представляет собой информацию о музыкальной композиции;
- *Employee* представляет собой информацию о сотруднике;
- *Position* представляет собой информацию о должности сотрудника радиостанции;
- *Genre* представляет собой информацию о музыкальном жанре;
- *Group* представляет собой информацию о музыкальной группе;
- *Performer* представляет собой информацию о музыкальном исполнителе.

По той причине, что данные хранимые в базе данных и отображаемые данные имеют разный вид, большинство представлений имеет собственные модели. Модели представлений, имеющиеся в приложении:

- *UserEmployeeEditViewModel* содержит свойства с атрибутами валидации, которые необходимо заполнить при редактировании пользователя если он является сотрудником, а также коллекции класса *SelectListItem* для выбора данных из выпадающего списка;
- *LoginViewModel* содержит свойства с атрибутами валидации, которые необходимо заполнить при авторизации в приложении;
- *RegistrationViewModel* содержит свойства с атрибутами валидации, которые необходимо заполнить при регистрации пользователя, а также коллекции класса *SelectListItem* для выбора данных из выпадающего списка;
- *CreateBroadcastViewModel* содержит свойства с атрибутами валидации, которые необходимо заполнить при добавлении радиотрансляции, а также коллекцию класса *SelectListItem* для выбора данных из выпадающего списка;
- *EditBroadcastViewModel* содержит свойства с атрибутами валидации, которые необходимо заполнить при редактировании радиотрансляции, а также коллекцию класса *SelectListItem* для выбора данных из выпадающего списка;
- *BroadcastViewModel* содержит данные для отображения подробной ин-



формации о радиотрансляциях, с заменой внешних ключей на данные из связанных таблиц, с возможностью постраничного просмотра;

- *RecordViewModel* содержит данные для отображения информации о музыкальных композициях, с заменой внешних ключей на данные из связанных таблиц, с возможностью постраничного просмотра;

- *CreateRecordViewModel* содержит свойства с атрибутами валидации, которые необходимо заполнить при добавлении музыкальной композиции, а также коллекцию класса *SelectListItem* для выбора данных из выпадающего списка;

- *EditRecordViewModel* содержит свойства с атрибутами валидации, которые необходимо заполнить при редактировании музыкальной композиции, а также коллекцию класса *SelectListItem* для выбора данных из выпадающего списка;

- *PerformerViewModel* содержит данные для отображения информации о музыкальных исполнителях, с заменой внешних ключей на данные из связанных таблиц, с возможностью постраничного просмотра;

- *CreatePerformerViewModel* содержит свойства с атрибутами валидации, которые необходимо заполнить при добавлении музыкального исполнителя, а также коллекцию класса *SelectListItem* для выбора данных из выпадающего списка;

- *EditPerformerViewModel* содержит свойства с атрибутами валидации, которые необходимо заполнить при редактировании музыкального исполнителя, а также коллекцию класса *SelectListItem* для выбора данных из выпадающего списка;

- *GroupDetailViewModel* содержит данные для отображения информации о музыкальных группах и их исполнителях, с заменой внешних ключей на данные из связанных таблиц, с возможностью постраничного просмотра;

- *EmployeeViewModel* содержит данные для отображения информации о сотрудниках, с заменой внешних ключей на данные из связанных таблиц, с возможностью постраничного просмотра;

- *PageViewModel* представляет данные для реализации разделения на страницы;

- *PageItemsViewModel* представляет обобщенный класс, который хранит модель страницы и коллекцию элементов заданного типа.

### **3.5 Структура системы аутентификации и авторизации пользователей приложения «Радиостанция»**

В качестве системы авторизации и аутентификации выбрана имеющаяся встроенная система: *ASP.NET Core Identity*. Для реализации данной системы создан класс *ApplicationDbContext*, который наследуется от

*IdentityDbContext<ApplicationUser>*, также создана дополнительная модель *ApplicationUser* для расширения информации которая будет храниться о пользователе.

Для обеспечения удобства работы с данными, в качестве места хранения таблиц необходимых *ASP.NET Core Identity* выбрана основная база данных приложения, для этого в основную базу был добавлен ряд встроенных таблиц и созданы связи между таблицами *ASP.NET Core Identity* и таблицами приложения.

В классе *Startup* настроено сохранение авторизационных данных пользователя в куках браузера, с помощью данной функции в течении некоторого времени пользователь может начать новую сессию без необходимости заново проходить авторизацию.

На некоторые из маршрутов контроллеров добавлен авторизационный атрибут, который позволяет защитить приложение от несанкционированного доступа и выполнить разделение функционала приложения по ролям.

Для удобства работы с ролями добавлен класс *Roles* который содержит константные значения ролей. Роли, имеющиеся в приложении:

- *Admin* представляет пользователя, который имеет полный доступ к приложению и возможностью редактирования полной информации;
- *HR\_Employee(Human Resources)* представляет пользователя, который может производить редактирование персонала радиостанции;
- *User* представляет пользователя, который может видеть подробную информацию о радиостанции;
- *Employee* представляет пользователя, который является сотрудником радиостанции и может просматривать, и редактировать информацию на радиостанции.

Исходный код программного обеспечения представлен в обязательном приложении А.

Структура разработанного *web*-приложения представлена в обязательном приложении Б.

## 4 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

### 4.1 Введение

Разработанное приложение предназначено для просмотра информации на радиостанции. Приложение предоставляет возможности по добавлению, редактированию и удалению информации о радиостанции, получению информации о музыкальных композициях и группах, музыкальных трансляциях, управлению пользователями и их ролями.

Для работы с программным обеспечением пользователю необходимо иметь начальные сведения и навыки работы с персональным компьютером и любым современным браузером.

### 4.2 Назначение и условия применения

Для работы с приложением пользователю нужен браузер с включенной поддержкой *JavaScript*.

### 4.3 Подготовка к работе

Перед началом работы с приложением пользователю нет необходимости производить какие-либо дополнительные настройки, кроме открытия любого браузера и перехода по адресу сервера на котором расположено приложение.

### 4.4 Описание операций

Операция 1: Авторизация в приложении.

Условия необходимые для выполнения: ввод правильного логина и пароля.

После открытия приложения, открывается домашняя страница.

Не авторизованному пользователю доступен просмотр только домашней страницы и расписание радиотрансляций.

Для того чтобы получить полную информацию, пользователю необходимо войти в аккаунт или зарегистрироваться на сайте нажав на кнопку «*Log in*» (рисунок 4.1).



Рисунок 4.1 – Домашняя страница

После нажатия на «*Log in*», открывается страница входа в аккаунт (рисунок 4.2)

Login

eqweqw

Password

LOG IN

- Invalid name or password

[Registration](#) [Home](#)

Рисунок 4.2 – Страница входа в аккаунт

В приложении предусмотрена проверка корректности введенных данных. При успешной регистрации произойдет перенаправление на домашнюю страницу.

Операция 2: Регистрация в приложении.

Условия необходимые для выполнения: ввод корректных данных.

Если у пользователя еще нет аккаунта, ему необходимо зарегистрироваться, нажав на кнопку «*Registration*». После нажатия на кнопку, произойдет перенаправление на страницу регистрации, где необходимо заполнить соответствующие поля (рисунок 4.3).

Registration

User name

Name

Surname

Middle name

Password

Repeat password

Email

SIGN UP

Рисунок 4.3 – Страница регистрации

При неуспешной регистрации, появятся соответствующие сообщения о некорректно введенных данных. Если же все данные были заполнены корректно, произойдет перенаправление на домашнюю страницу, где в верхнем правом углу будет отображаться логин пользователя (рисунок 4.4).

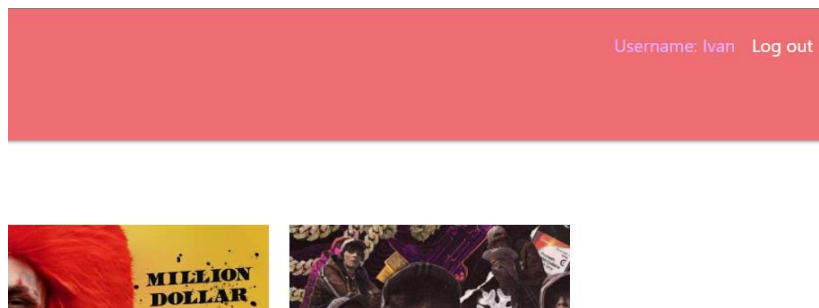


Рисунок 4.4 – Место отображения логина пользователя

Для выхода из аккаунта необходимо нажать кнопку «*Log out*», произойдет выход из аккаунта и перенаправление на домашнюю страницу.

Операция 3: Просмотр информации о музыкальных композициях.

Условия необходимые для выполнения: пройденная авторизация.

После успешной авторизации перейти на вкладку «*Records*» в навигационном меню. После перехода на вкладку, будет отображена страница с музыкальными композициями и возможностью их фильтрации (рисунок 4.5).

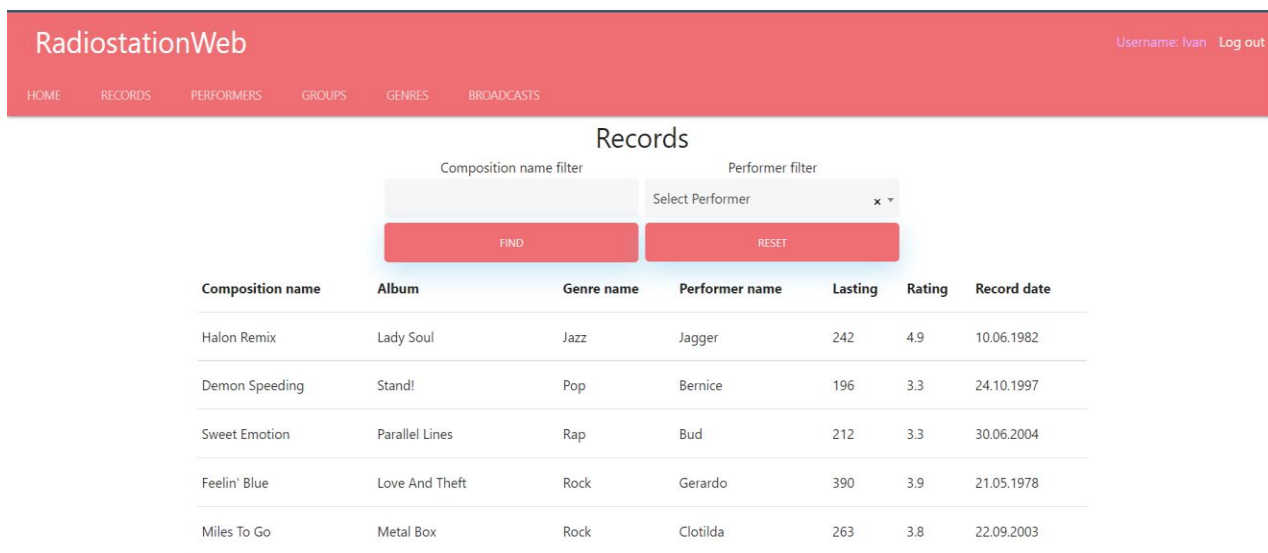


Рисунок 4.5 – Страница с музыкальными композициями

Для ролей «*Admin*» и «*Employee*» появляется возможность редактирования и добавления музыкальных композиций – кнопка «*Manage*». По нажатию на

кнопку происходит перенаправление на страницу редактирования музыкальных композиций (рисунок 4.6).

**RadiostationWeb** Username: Admin Manage users Log out

HOME RECORDS PERFORMERS GROUPS GENRES BROADCASTS EMPLOYEES

### Records management

Name filter Performer filter

Find Performer Select Performer x

FIND RESET

ADD RECORD

Composition name	Album	Genre name	Performer name	Lasting	Rating	Record date	
Halon Remix	Lady Soul	Jazz	Jagger	242	4.9	10.06.1982	EDIT DELETE
Demon Speeding	Stand!	Pop	Bernice	196	3.3	24.10.1997	EDIT DELETE

Рисунок 4.6 – Страница редактирования музыкальных композиций

По нажатию на кнопку «*EDIT*», открывается окно редактирования музыкальной композиции, по нажатию на кнопку «*ADD RECORD*», открывается окно добавления музыкальной композиции.

Операция 4: Просмотр информации о музыкальных исполнителях.

Условия необходимые для выполнения: пройденная авторизация.

После успешной авторизации перейти на вкладку «*Performers*» в навигационном меню. После перехода на вкладку, будет отображена страница с музыкальными исполнителями и возможностью их фильтрации (рисунок 4.7).

**RadiostationWeb** Username: Ivan Log out

HOME RECORDS PERFORMERS GROUPS GENRES BROADCASTS

### Performers

Name filter Surname filter Group filter

Find Surname Select Group x

FIND RESET

Name	Surname	Group name
Marceline	Moore	-
Alta	Allison	Yasy Boys
Pascal	Boyer	Van Halen

Рисунок 4.7 – Страница просмотра музыкальных исполнителей

Для ролей «Admin» и «Employee» появляется возможность редактирования музыкальных исполнителей – кнопка «Manage». По нажатию на кнопку происходит перенаправление на страницу редактирования музыкальных исполнителей (рисунок 4.8).

**RadiostationWeb** Username: Admin Manage users Log out

HOME RECORDS PERFORMERS GROUPS GENRES BROADCASTS EMPLOYEES

### Performers managment

Name filter Surname filter Group filter

ADD PERFORMER FIND RESET

Name	Surname	Group description	
Marceline	Moore	-	EDIT DELETE
Alta	Allison	Yasy Boys	EDIT DELETE

Рисунок 4.8 – Страница редактирования музыкальных исполнителей

По нажатию на кнопку «EDIT», открывается окно редактирования музыкального исполнителя, по нажатию на кнопку «ADD PERFORMER», открывается окно добавления музыкального исполнителя (рисунок 4.9).

### Add performer

Name  
Surname  
Group Description

ADD

Рисунок 4.9 – Окно добавления музыкального исполнителя

После заполнения всех полей и нажатия на кнопку «ADD» музыкальный исполнитель будет успешно добавлен. Если поля будут не заполнены, то высветится соответствующее сообщение.

Операция 5: Просмотр информации о музыкальных группах.

Условия необходимые для выполнения: пройденная авторизация.

После успешной авторизации перейти на вкладку «*Groups*» в навигационном меню. После перехода на вкладку, будет отображена страница с музыкальными группами и возможностью их фильтрации (рисунок 4.10).

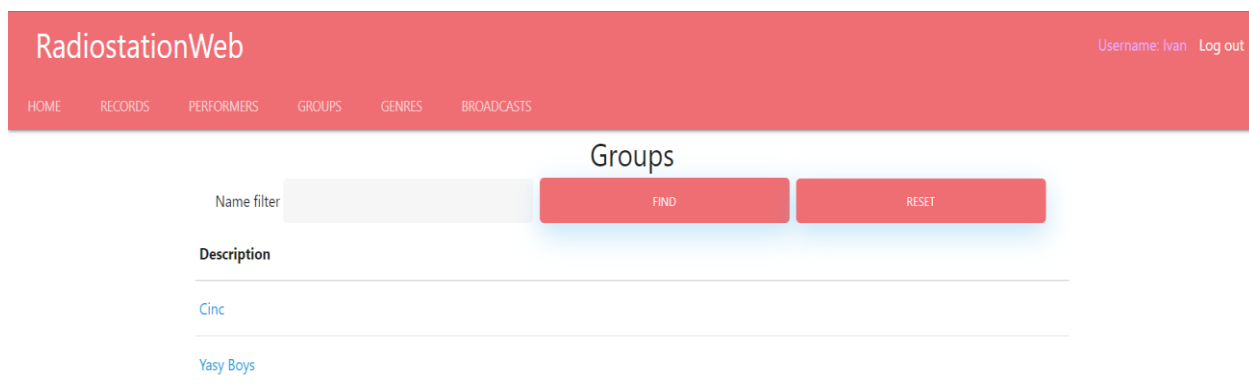


Рисунок 4.10 – Страница просмотра музыкальных групп

При нажатии на описание группы, откроется окно с составом исполнителей данной группы (рисунок 4.11).

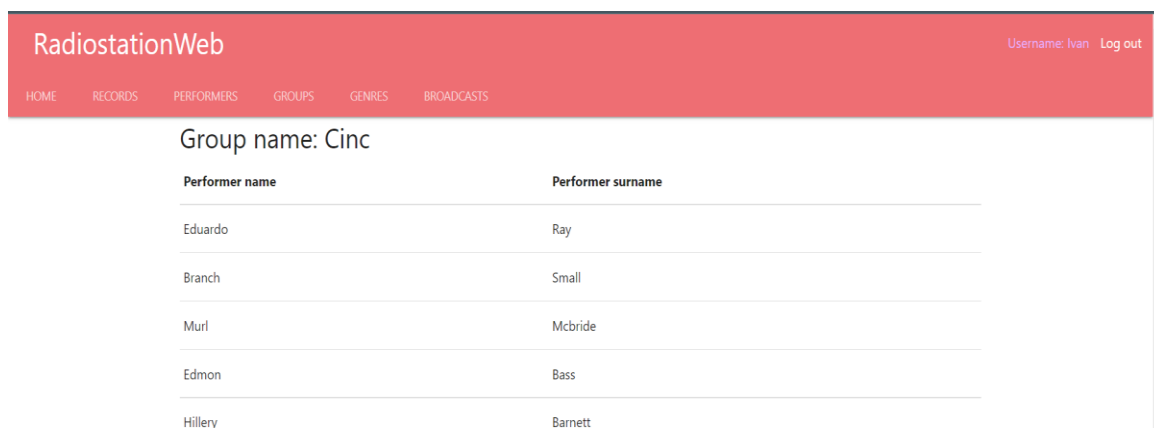


Рисунок 4.11 – Состав музыкальных исполнителей заданной группы

Для ролей «*Admin*» и «*Employee*» появляется возможность редактирования музыкальных групп – кнопка «*Manage*». По нажатию на кнопку происходит перенаправление на страницу редактирования музыкальных групп, где можно добавить группу или изменить существующую.

Операция 6: Просмотр информации о музыкальных жанрах.

Условия необходимые для выполнения: пройденная авторизация.

После успешной авторизации перейти на вкладку «*Genres*» в навигационном меню. После перехода на вкладку, будет отображена страница с музыкальными жанрами и возможностью их фильтрации.



Для ролей «Admin» и «Employee» появляется возможность редактирования музыкальных жанров – кнопка «Manage». По нажатию на кнопку происходит перенаправление на страницу редактирования музыкальных жанров, где можно добавить жанр или изменить соответствующий (рисунок 4.12).

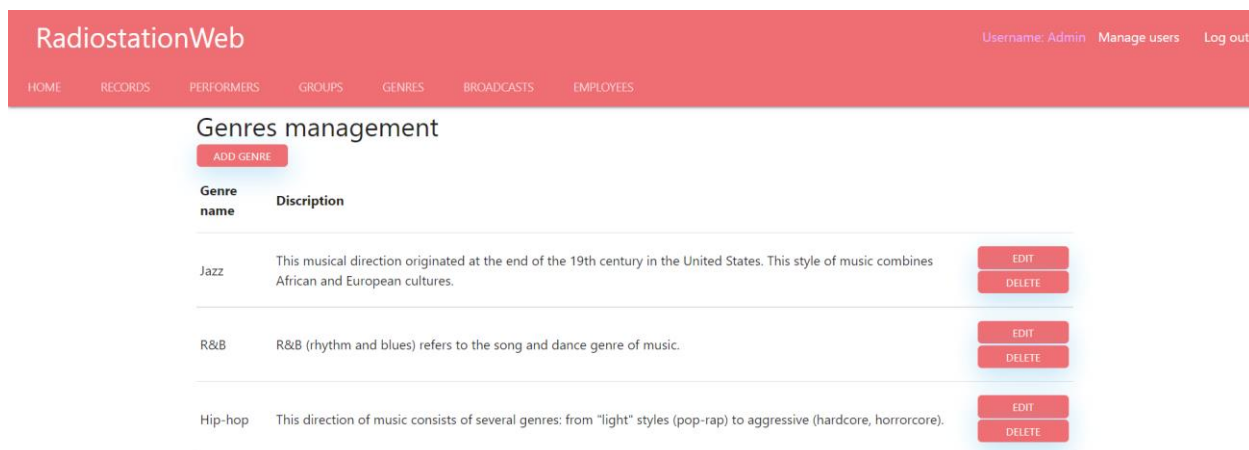


Рисунок 4.12 – Страница редактирования музыкальных жанров

Операция 7: Просмотр информации о радиотрансляциях.

Условия необходимые для выполнения: пройденная авторизация.

После успешной авторизации перейти на вкладку «Broadcasts» в навигационном меню. После перехода на вкладку, будет отображена страница с радиотрансляциями и возможностью их фильтрации и сортировки.

Для ролей «Admin» и «Employee» появляется возможность редактирования радиотрансляций – кнопка «Manage». По нажатию на кнопку происходит перенаправление на страницу редактирования радиотрансляций, где можно добавить трансляцию или изменить соответствующую (рисунок 4.13).

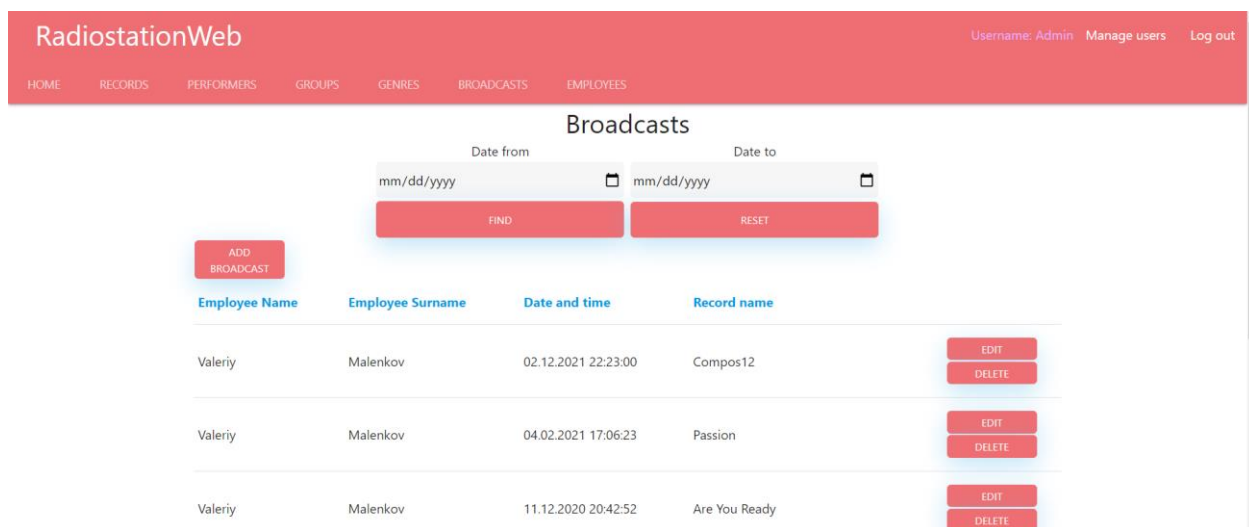


Рисунок 4.13 – Страница редактирования радиотрансляций

По нажатию на кнопку «*EDIT*», открывается окно редактирования радиотрансляции, по нажатию на кнопку «*ADD BROADCAST*», открывается окно добавления радиотрансляции.

Операция 8: Просмотр информации о пользователях.

Условия необходимые для выполнения: наличие роли «*Admin*» или «*HR\_Employee*».

После успешной авторизации перейти на вкладку «*Manage users*» в навигационном меню. После перехода на вкладку, будет отображена страница с пользователями *web*-сайта и возможностью их фильтрации, редактирования, удаления и добавления пользователей (рисунок 4.14).

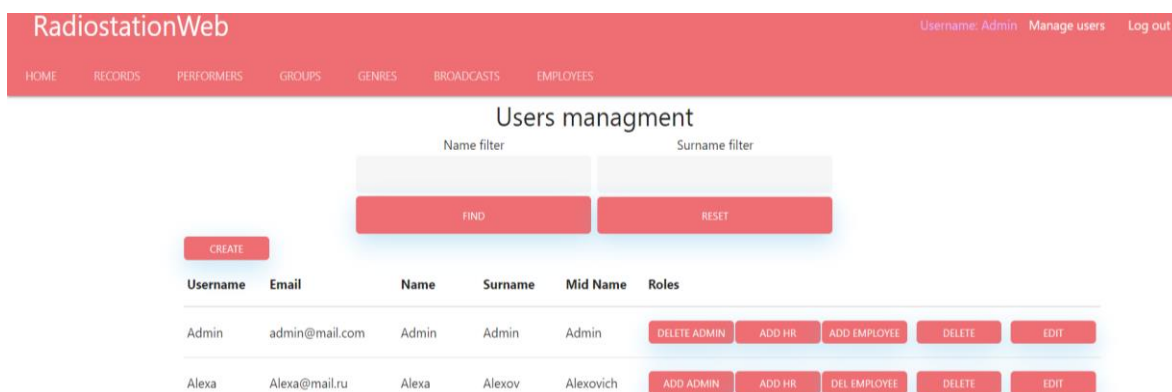


Рисунок 4.14 – Страница управления пользователями

По нажатию на кнопку «*DELETE*», в случае если пользователь не связан с какими-либо данными, он будет успешно удален, иначе удаление не произойдет и будет выведено соответствующее сообщение. По нажатию на кнопку «*EDIT*», откроется окно редактирования соответствующего пользователя. Если пользователь имеет роль «*Employee*», то в окне редактирования появятся дополнительные поля для сотрудника (рисунок 4.15).

Surname  
Alexov

Middle Name  
Alexovich

Position  
Select Position x ▾

Education  
Select Education x ▾

Work time  
WorkTime

EDIT

Рисунок 4.15 – Окно редактирования пользователя с ролью «*Employee*»

По нажатию на кнопки, означающие удаление/добавление роли, у пользователя добавится или удалится роль (рисунок 4.16)

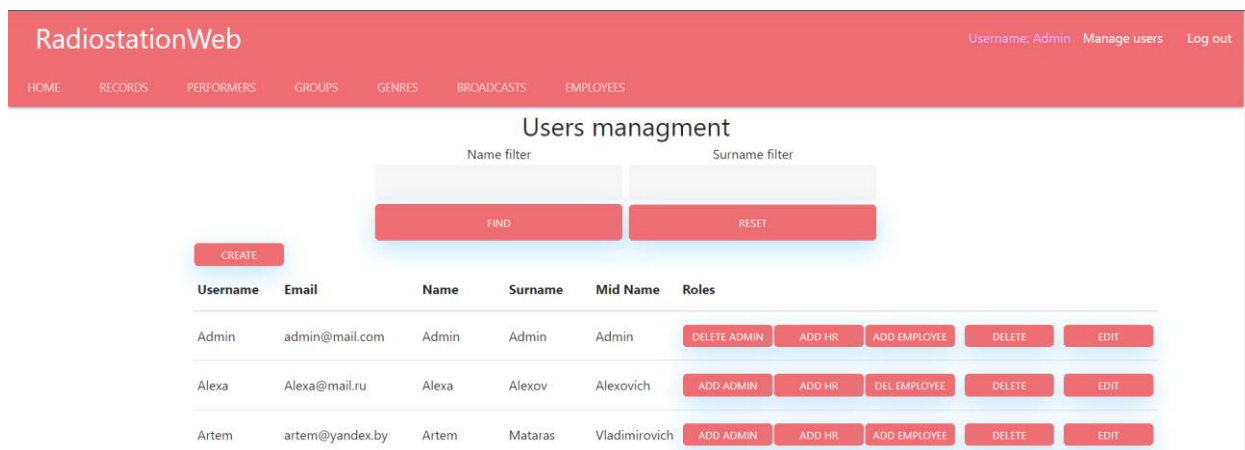


Рисунок 4.16 – Кнопки управления ролями

По нажатию на кнопку «*ADD EMPLOYEE*», пользователю будет добавлена роль «*Employee*» и значение кнопки поменяется на «*DEL EMPLOYEE*».

По нажатию на кнопку «*DEL EMPLOYEE*», у пользователя будет удалена роль «*Employee*» и значение кнопки поменяется на «*ADD EMPLOYEE*».

По нажатию на кнопку «*ADD ADMIN*», пользователю будет добавлена роль «*Admin*» и значение кнопки поменяется на «*DEL ADMIN*».

По нажатию на кнопку «*DEL ADMIN*», у пользователя будет удалена роль «*Admin*» и значение кнопки поменяется на «*ADD ADMIN*».

По нажатию на кнопку «*ADD HR*», пользователю будет добавлена роль «*HR\_Employee*» и значение кнопки поменяется на «*DEL HR*».

По нажатию на кнопку «*DEL HR*», у пользователя будет удалена роль «*HR\_Employee*» и значение кнопки поменяется на «*ADD HR*».

У пользователя может быть сразу несколько ролей.

Главного администратора и его роль удалить нельзя.

## 4.5 Аварийные ситуации

В приложении используется обработка ошибок, поэтому в случае обнаружения сообщения об ошибке необходимо обратиться к должностному лицу, занимающемуся сопровождением программного продукта, продиктовать текст ошибки и, по возможности, назвать условия появления данной ошибки.

В случае аварийного завершения работы веб-браузера или операционной системы все данные могут быть потеряны. Поэтому при работе с данными рекомендуется в течение некоторого промежутка сохранять сделанные изменения.

В случае обнаружения несанкционированного вмешательства в работу

приложения, либо в данные, необходимо, не совершая никаких действий, прекратить работу и в кратчайшие сроки сообщить об этом должностному лицу, занимающемуся сопровождением программного обеспечения.

При обнаружении ошибок необходимо как можно подробнее описать признаки несанкционированного вмешательства и дату обнаружения.

#### **4.6 Рекомендации по освоению**

Приложение имеет интуитивно понятный интерфейс и рассчитано на работу с людьми, не имеющими специального образования в сфере информационных технологий. Поэтому для освоения данного программного продукта достаточно краткой лекции и 15 минутного изучения интерфейса приложения. Перед началом работы с основными данными рекомендуется произвести тренировку на тестовых наборах данных.

## **5 РУКОВОДСТВО ПРОГРАММИСТА**

### **5.1 Назначение и условия применения**

Программное обеспечение предназначено для предоставления возможности осуществлять просмотр и редактирование информации на радиостанции.

Условия необходимые для выполнения программы:

- объем оперативной памяти устройства не менее 1 ГБ;
- процессор с тактовой частотой не ниже 2 ГГц;
- операционная система *Windows 7* и выше;
- наличие *.NET 5* и выше;
- доступ к серверу базы данных СУБД *MS SQL Server*.

### **5.2 Характеристики приложения**

Приложение может использовать любой пользователь, в случае если пользователь авторизуется, набор функционала будет существенно больше.

### **5.3 Обращение к приложению**

Для доступа к приложению необходимо в адресной строке браузера ввести ссылку на сервер, где приложение запущено в данный момент.

Для просмотра и редактирования исходного кода приложения может быть использован любой редактор, поддерживающий открытие *.sln* файлов, например, *Visual Studio*. Для открытия проекта необходимо запустить файл: *Radiostation.sln*.

### **5.4 Входные и выходные данные**

Входными данными программы являются пользовательский ввод и данные полученные из базы данных.

Выходными данными является информация, отображаемая на страницах приложения.

### **5.5 Сообщения.**

В ходе работы с приложением могут появляться сообщения об ошибках, предупреждающие о том, что что-то введено неправильно для исправления таких ошибок достаточно изучить текст ошибки и ввести корректные данные.

## ЗАКЛЮЧЕНИЕ

В ходе выполнения курсового проекта изучена и проанализирована информация в области разработки *web*-приложений средствами языка *C#*, при использовании СУБД *MS SQL Server*. Разработано программное обеспечение для просмотра и редактирования информации на радиостанции.

В ходе разработки приложения решены следующие задачи:

- изучение средств и подходов разработки *web*-приложений;
- проектирование базы данных с соблюдением правил нормализации;
- разработка программного кода, отвечающего заданным условиям;
- проектирование пользовательского интерфейса.

Разработанное приложение имеет ряд преимуществ между аналогами, среди которых кроссплатформенность, легкость доступа (для доступа к приложению необходим любой интернет-браузер), использование ролей пользователей, защита от несанкционированного доступа, интуитивно понятный пользовательский интерфейс.

В коде приложения присутствует обработка ошибок, поэтому неправильные действия пользователя не приведут к экстренному завершению работы приложения. Работоспособность приложения протестирована локально, ошибок не выявлено.

В случае необходимости данное приложение может быть дополнено другими модулями, что позволит упростить его использование и отладку.

При реализации программного обеспечения использованы язык программирования *C#* и его средства *Entity Framework Core* и *ASP.NET Core*, а также СУБД *MS SQL Server* для хранения данных.

Таким образом разработано гибкое и удобное приложение, которое предоставляет весь требуемый функционал и с учетом некоторых доработок.

## Список использованных источников

1. Описание основных принципов нормализации баз данных [Электронный ресурс]. – Электрон. данные. – Режим доступа: <https://docs.microsoft.com/ru/office/troubleshoot/access/database-normalization-description> – Дата доступа: 02.12.2021
2. Система управления базами данных *MS SQL Server* [Электронный ресурс]. – Электрон. данные. – Режим доступа: <https://brainoteka.com/courses/ms-sql-dlya-nachinayushih/chto-takoe-subd> – Дата доступа: 02.12.2021.
3. Троелсен Э., Джепикс Ф., Язык программирования *C# 7.0* и платформы *.NET* и *.NET Core* / Троелсен Э., Джепикс Ф. 8-е изд., перераб. и доп. – Санкт-Петербург: 2018 – 1328 с.
4. Рихтер Дж., *CLR VIA C#*. Программирование на платформе *Microsoft .NET Framework 4.5* на языке *C#* / Рихтер Дж. 4-е изд., перераб. и доп. – Питер: 2019 – 896 с
5. Фримен А., *ASP.NET Core MVC 2* с примерами на *C#* / Фримен А. 7-е изд., перераб. и доп. – Диалектика/Вильямс: 2019 – 1008 с

## ПРИЛОЖЕНИЕ А

(обязательное)

### Листинг программы

#### Листинг *ApplicationDbContext*

```
using Microsoft.AspNetCore.Identity.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore;

namespace RadiostationWeb.Data
{
    public class ApplicationDbContext : IdentityDbContext<ApplicationUser>
    {
        public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options)
            : base(options)
        {
        }
    }
}
```

#### Листинг *ApplicationUser*

```
using Microsoft.AspNetCore.Identity;
namespace RadiostationWeb.Data
{
    public class ApplicationUser : IdentityUser
    {
        [PersonalData]
        public string Name { get; set; }
        [PersonalData]
        public string Surname { get; set; }
        [PersonalData]
        public string MiddleName { get; set; }
    }
}
```

#### Листинг *AuthorizeRolesAttribute*

```
using Microsoft.AspNetCore.Authorization;

namespace RadiostationWeb.Data
{
    public class AuthorizeRolesAttribute : AuthorizeAttribute
    {
        public AuthorizeRolesAttribute(params string[] roles) : base()
        {
            Roles = string.Join(" ", roles);
        }
    }
}
```

#### Листинг *RadiostationWebDbContext*

```
using Microsoft.EntityFrameworkCore;
using RadiostationWeb.Models;
```



```

namespace RadiostationWeb.Data
{
    public partial class RadiostationWebDbContext : DbContext
    {
        public RadiostationWebDbContext()
        {
        }

        public RadiostationWebDbContext(DbContextOptions<RadiostationWebDbContext> options)
            : base(options)
        {
        }

        public virtual DbSet<Broadcast> Broadcasts { get; set; }
        public virtual DbSet<Employee> Employees { get; set; }
        public virtual DbSet<Genre> Genres { get; set; }
        public virtual DbSet<Group> Groups { get; set; }
        public virtual DbSet<Performer> Performers { get; set; }
        public virtual DbSet<Record> Records { get; set; }
        public virtual DbSet<Position> Positions { get; set; }
        public virtual DbSet<HomePageImage> HomePageImages { get; set; }

    }
}

```

### Листинг *SortState*

```

namespace RadiostationWeb.Data
{
    public enum SortState
    {
        NameAsc,
        NameDsc,
        SurnameAsc,
        SurnameDsc,
        RecorNameAsc,
        RecordNameDsc,
        DateAsc,
        DateDsc,
    }
}

```

### Листинг *LoginViewModel*

```

namespace RadiostationWeb.Models
{
    public class LoginViewModel
    {
        public string Username { get; set; } = "";

        public string Password { get; set; } = "";

        public string returnUrl { get; set; } = "/";
    }
}

```

### Листинг *RegistrationViewModel*

```

using System.ComponentModel.DataAnnotations;

namespace RadiostationWeb.Models
{
    public class RegistrationViewModel

```

```

{
    public string Id { get; set; }

    [Required(ErrorMessage = "Username is required")]
    public string Username { get; set; } = "";

    [Required(ErrorMessage = "Password is required")]
    [RegularExpression(@"^(?=.*[a-z])(?=.*[A-Z])(?=.*\d){6,}$", ErrorMessage = "Easy password!")]
    public string Password { get; set; } = "";

    [Required(ErrorMessage = "Required repeat password")]
    [Compare("Password", ErrorMessage = "Repeat must match")]
    public string RepeatPassword { get; set; } = "";

    [Required(ErrorMessage = "Name is required")]
    public string Name { get; set; } = "";

    [Required(ErrorMessage = "Surname is required")]
    public string Surname { get; set; } = "";

    public string MiddleName { get; set; } = "";

    [Required(ErrorMessage = "Email")]
    [EmailAddress(ErrorMessage = "Incorrect email")]
    public string Email { get; set; } = "";
}
}

```

## Листинг *RoleType*

```

namespace RadiostationWeb.Models
{
    public class RoleType
    {
        public const string Admin = "Admin";
        public const string User = "User";
        public const string Employee = "Employee";
        public const string HR_Employee = "HR_Employee";
    }
}

```

## Листинг *UserEmployeeEditViewModel*

```

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

namespace RadiostationWeb.Models
{
    public class UserEmployeeEditViewModel
    {
        public string Id { get; set; }
        public int EmployeeId { get; set; }

        [Required(ErrorMessage = "Username is required")]
        public string UserName { get; set; } = "";

        [Required(ErrorMessage = "Name is required")]
        public string Name { get; set; } = "";

        [Required(ErrorMessage = "Surname is required")]
        public string Surname { get; set; } = "";
    }
}

```

```

public string MiddleName { get; set; } = "";

[Required(ErrorMessage = "Email is required")]
[EmailAddress(ErrorMessage = "Incorrect email")]
public string Email { get; set; } = "";
[Required(ErrorMessage = "Position is required")]
public int? PositionId { get; set; }
public string Education { get; set; }
public bool EmployeeRole { get; set; }
[Range(1, 12, ErrorMessage = "The value must be in the limit from 1 to 12")]
public int? WorkTime { get; set; }
public IEnumerable<SelectListItem> PositionList { get; set; }
public IEnumerable<SelectListItem> EducationList { get; set; }
}
}

```

## Листинг *Broadcast*

```

using System;
using System.ComponentModel.DataAnnotations;

namespace RadiostationWeb.Models
{
    public partial class Broadcast
    {
        public int Id { get; set; }
        [Required]
        public DateTime? DateAndTime { get; set; }
        [Required]
        public int EmployeeId { get; set; }
        [Required]
        public int RecordId { get; set; }
    }
}

```

## Листинг *BroadcastsItemModel*

```

using System.Collections.Generic;
namespace RadiostationWeb.Models
{
    public class BroadcastsItemModel
    {
        public IEnumerable<BroadcastViewModel> Items { get; set; }

        public PageViewModel PageModel { get; set; }

        public SortViewModel SortViewModel { get; set; }
    }
}

```

## Листинг *BroadcastViewModel*

```

using System;
namespace RadiostationWeb.Models
{
    public class BroadcastViewModel
    {
        public int Id { get; set; }
    }
}

```

```

        public DateTime? DateAndTime { get; set; }
        public string EmployeeName { get; set; }
        public string EmployeeSurname { get; set; }
        public string RecordName { get; set; }
    }
}

```

### Листинг *CreateBroadcastViewModel*

```

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

namespace RadiostationWeb.Models
{
    public class CreateBroadcastViewModel
    {
        [Required]
        public int EmployeeId { get; set; }
        [Required]
        public int RecordId { get; set; }
        [Required(ErrorMessage = "Date and time is required")]
        public DateTime? DateAndTime { get; set; }

        public IEnumerable<SelectListItem> EmployeeList { get; set; }
        public IEnumerable<SelectListItem> RecordList { get; set; }
    }
}

```

### Листинг *EditBroadcastViewModel*

```

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

namespace RadiostationWeb.Models
{
    public class EditBroadcastViewModel
    {
        public int Id { get; set; }
        [Required]
        public int EmployeeId { get; set; }
        [Required]
        public int RecordId { get; set; }
        [Required(ErrorMessage = "Data and time is required")]
        public DateTime? DateAndTime { get; set; }
        public IEnumerable<SelectListItem> EmployeeList { get; set; }
        public IEnumerable<SelectListItem> RecordList { get; set; }
    }
}

```

### Листинг *Employee*

```

namespace RadiostationWeb.Models
{
    public partial class Employee
    {
        public int Id { get; set; }
    }
}

```

```

        public string AspNetUserId { get; set; }
        public int? PositionId { get; set; }

        public string Education { get; set; }
        public int? WorkTime { get; set; }

    }
}

```

### Листинг *EmployeeViewModel*

```

namespace RadiostationWeb.Models
{
    public class EmployeeViewModel
    {
        public int Id { get; set; }
        public string Email { get; set; }

        public string Username { get; set; }

        public string AspNetUserId { get; set; }

        public string PositionName { get; set; }

        public string Education { get; set; }
        public int? WorkTime { get; set; }

        public string Name { get; set; }
        public string Surname { get; set; }
        public string MiddleName { get; set; }
    }
}

```

### Листинг *ErrorViewModel*

```

namespace RadiostationWeb.Models
{
    public class ErrorViewModel
    {
        public string RequestId { get; set; }

        public bool ShowRequestId => !string.IsNullOrEmpty(RequestId);
    }
}

```

### Листинг *Genre*

```

using System.ComponentModel.DataAnnotations;

namespace RadiostationWeb.Models
{
    public partial class Genre
    {
        public int Id { get; set; }
        [Required]
        public string GenreName { get; set; }
        [Required]
        public string Description { get; set; }
    }
}

```

```
}
```

### Листинг *Group*

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

```
namespace RadiostationWeb.Models
```

```
{
```

```
    public class Group
```

```
    {
```

```
        public int Id { get; set; }
```

```
        [Required]
```

```
        public string Description { get; set; }
```

```
    }
```

```
}
```

### Листинг *GroupDetailViewModel*

```
namespace RadiostationWeb.Models
```

```
{
```

```
    public class GroupDetailViewModel
```

```
    {
```

```
        public string PerformerName { get; set; }
```

```
        public string PerformerSurname { get; set; }
```

```
        public string GroupName { get; set; }
```

```
    }
```

```
}
```

### Листинг *GroupItemsViewModel*

```
using System.Collections.Generic;
```

```
namespace RadiostationWeb.Models
```

```
{
```

```
    public class GroupItemsViewModel
```

```
    {
```

```
        public IEnumerable<GroupDetailViewModel> GroupsItems { get; set; }
```

```
        public string GroupName { get; set; }
```

```
        public int Id { get; set; }
```

```
    }
```

```
        public PageViewModel PageModel { get; set; }
```

```
    }
```

```
}
```

### Листинг *PageItemsModel*

```
using Microsoft.AspNetCore.Mvc.Rendering;
```

```
using System.Collections.Generic;
```

```
namespace RadiostationWeb.Models
```

```
{
```

```
    public class PageItemsModel<T>
```

```
    {
```

```
        public IEnumerable<T> Items { get; set; }
```

```
    }
```

```
        public PageViewModel PageModel { get; set; }
```

```
    }
```

```
}
```

## Листинг *PageViewModel*

```
using System;
namespace RadiostationWeb.Models
{
    public class PageViewModel
    {
        public int PageNumber { get; private set; }
        public int TotalPages { get; private set; }

        public PageViewModel(int count, int pageNumber, int pageSize)
        {
            PageNumber = pageNumber;
            TotalPages = (int)Math.Ceiling(count / (double)pageSize);
        }

        public bool HasPreviousPage
        {
            get
            {
                return (PageNumber > 1);
            }
        }

        public bool HasNextPage
        {
            get
            {
                return (PageNumber < TotalPages);
            }
        }
    }
}
```

## Листинг *SortViewModel*

```
using RadiostationWeb.Data;
namespace RadiostationWeb.Models
{
    public class SortViewModel
    {
        public SortState NameSort { get; set; }
        public SortState SurnameSort { get; set; }
        public SortState RecordSort { get; set; }
        public SortState DateSort { get; set; }

        public SortState Current { get; set; }
        public bool Up { get; set; }

        public SortViewModel(SortState sortOrder)
        {
            NameSort = SortState.NameAsc;
            SurnameSort = SortState.SurnameAsc;
            RecordSort = SortState.RecorNameAsc;
            DateSort = SortState.DateAsc;
            Up = true;

            if (sortOrder == SortState.NameDsc
                || sortOrder == SortState.SurnameDsc
                || sortOrder == SortState.RecordNameDsc
                || sortOrder == SortState.DateDsc)
```

```

        {
            Up = false;
        }

        Current = sortOrder switch
        {
            SortState.NameDsc => NameSort = SortState.NameAsc,
            SortState.NameAsc => NameSort = SortState.NameDsc,
            SortState.SurnameDsc => SurnameSort = SortState.SurnameAsc,
            SortState.SurnameAsc => SurnameSort = SortState.SurnameDsc,
            SortState.RecordNameDsc => RecordSort = SortState.RecorNameAsc,
            SortState.RecorNameAsc => RecordSort = SortState.RecordNameDsc,
            SortState.DateAsc => DateSort = SortState.DateDsc,
            _ => DateSort = SortState.DateAsc,
        };
    }
}
}

```

### Листинг *CreatePerformerViewModel*

```

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

namespace RadiostationWeb.Models
{
    public class CreatePerformerViewModel
    {
        [Required]
        public string Name { get; set; }
        [Required]
        public string Surname { get; set; }
        public int GroupId { get; set; }
        public IEnumerable<SelectListItem> GroupsList { get; set; }
    }
}

```

### Листинг *EditPerformerViewModel*

```

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

namespace RadiostationWeb.Models
{
    public class EditPerformerViewModel
    {
        public int Id { get; set; }
        [Required]
        public string Name { get; set; }
        [Required]
        public string Surname { get; set; }
        public int? GroupId { get; set; }
        public IEnumerable<SelectListItem> GroupsList { get; set; }
    }
}

```

### Листинг *Performer*

```

using Microsoft.AspNetCore.Mvc;
using System.ComponentModel.DataAnnotations;

```



```

namespace RadiostationWeb.Models
{
    public class Performer
    {

        [HiddenInput(DisplayValue = false)]
        public int Id { get; set; }
        [Required]
        public string Name { get; set; }
        [Required]
        public string Surname { get; set; }
        public int? GroupId { get; set; }

    }
}

```

### Листинг *PerformersItemsModel*

```

using Microsoft.AspNetCore.Mvc.Rendering;
using System.Collections.Generic;
namespace RadiostationWeb.Models
{
    public class PerformersItemsModel
    {
        public IEnumerable<PerformerViewModel> Items { get; set; }

        public PageViewModel PageModel { get; set; }
        public IEnumerable<SelectListItem> SelectGroups { get; set; }
    }
}

```

### Листинг *PerformerViewModel*

```

namespace RadiostationWeb.Models
{
    public class PerformerViewModel
    {
        public int Id { get; set; }
        public string Name { get; set; }
        public string Surname { get; set; }
        public string GroupName { get; set; }

    }
}

```

### Листинг *Position*

```

namespace RadiostationWeb.Models
{
    public class Position
    {
        public int Id { get; set; }
        public string Name { get; set; }
    }
}

```

### Листинг *CreateRecordViewModel*

```

using Microsoft.AspNetCore.Mvc.Rendering;
using System;
using System.Collections.Generic;

```

```

using System.ComponentModel.DataAnnotations;

namespace RadiostationWeb.Models
{
    public class CreateRecordViewModel
    {
        public int Id { get; set; }
        [Required]
        public int PerformerId { get; set; }
        [Required]
        public int GenreId { get; set; }
        [Required]
        public string Album { get; set; }
        [Required]
        public DateTime RecordDate { get; set; }
        [Required]
        [Range(60, 600, ErrorMessage = "The value must be in the limit from 60 to 600")]
        public int Lasting { get; set; }
        [Required]
        [Range(1.0, 5.0, ErrorMessage = "The value must be in the limit from 1.0 to 5.0")]
        public decimal Rating { get; set; }
        [Required]
        public string CompositionName { get; set; }

        public IEnumerable<SelectListItem> PerformersList { get; set; }
        public IEnumerable<SelectListItem> GenresList { get; set; }
    }
}

```

## Листинг *EditRecordViewModel*

```

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

namespace RadiostationWeb.Models
{
    public class EditRecordViewModel
    {
        public int Id { get; set; }
        [Required]
        public int PerformerId { get; set; }
        [Required]
        public int GenreId { get; set; }
        [Required]
        public string Album { get; set; }
        [Required]
        public DateTime RecordDate { get; set; }
        [Required]
        [Range(60, 600, ErrorMessage = "The value must be in the limit from 60 to 600")]
        public int Lasting { get; set; }
        [Required]
        [Range(1.0, 5.0, ErrorMessage = "The value must be in the limit from 1.0 to 5.0")]
        public decimal Rating { get; set; }
        [Required]
        public string CompositionName { get; set; }
        public IEnumerable<SelectListItem> PerformersList { get; set; }
        public IEnumerable<SelectListItem> GenresList { get; set; }
    }
}

```

## Листинг *Record*

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

namespace RadiostationWeb.Models
{
    public partial class Record
    {
        public int Id { get; set; }
        [Required]
        public int PerformerId { get; set; }
        [Required]
        public int GenreId { get; set; }
        [Required]
        public string Album { get; set; }
        [Required]
        public DateTime RecordDate { get; set; }
        [Required]
        public int Lasting { get; set; }
        [Required]
        public decimal Rating { get; set; }
        [Required]
        public string CompositionName { get; set; }
    }
}
```

## Листинг *RecordsItemModel*

```
using Microsoft.AspNetCore.Mvc.Rendering;
using System.Collections.Generic;

namespace RadiostationWeb.Models
{
    public class RecordsItemModel
    {
        public IEnumerable<RecordViewModel> Items { get; set; }

        public PageViewModel PageModel { get; set; }

        public IEnumerable<SelectListItem> SelectPerformers { get; set; }
    }
}
```

## Листинг *RecordViewModel*

```
using System;
namespace RadiostationWeb.Models
{
    public class RecordViewModel
    {
        public int Id { get; set; }
        public int GenreId { get; set; }
        public string PerformerName { get; set; }
        public string GenreName { get; set; }

        public string Album { get; set; }
        public DateTime RecordDate { get; set; }
        public int Lasting { get; set; }
        public decimal Rating { get; set; }
    }
}
```

```

        public string CompositionName { get; set; }
    }
}

```

## Листинг *HomePageImage*

```

using System.ComponentModel.DataAnnotations;

namespace RadiostationWeb.Models
{
    public class HomePageImage
    {
        public int Id { get; set; }
        [DataType(DataType.Upload)]
        [Display(Name = "Upload File")]
        [Required(ErrorMessage = "Please choose file to upload.")]
        public string SrcImg { get; set; }
        public string ImgCaption { get; set; }
    }
}

```

## Листинг *PageLinkTagHelper*

```

using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.Rendering;
using Microsoft.AspNetCore.Mvc.Routing;
using Microsoft.AspNetCore.Mvc.ViewFeatures;
using Microsoft.AspNetCore.Razor.TagHelpers;
using RadiostationWeb.Models;

namespace RadiostationWeb.TagHelpers
{
    public class PageLinkTagHelper : TagHelper
    {
        private IUrlHelperFactory urlHelperFactory;
        public PageLinkTagHelper(IUrlHelperFactory helperFactory)
        {
            urlHelperFactory = helperFactory;
        }
        [ViewContext]
        [HtmlAttributeNotBound]
        public ViewContext ViewContext { get; set; }
        public PageViewModel PageModel { get; set; }
        public object Sorter { get; set; }
        public string PageAction { get; set; }

        public object ModelId { get; set; }

        public override void Process(TagHelperContext context, TagHelperOutput output)
        {
            IUrlHelper urlHelper = urlHelperFactory.GetUrlHelper(ViewContext);
            output.TagName = "div";

            TagBuilder tag = new TagBuilder("ul");
            tag.AddCssClass("pagination");

            TagBuilder currentItem = CreateTag(PageModel.PageNumber, urlHelper);

            if (PageModel.PageNumber > 2)
            {

```

```

        TagBuilder firstItem = CreateTag(1, urlHelper);
        tag.InnerHtml.AppendHtml(firstItem);
    }

    if (PageModel.HasPreviousPage)
    {
        TagBuilder prevItem = CreateTag(PageModel.PageNumber - 1, urlHelper);
        tag.InnerHtml.AppendHtml(prevItem);
    }

    tag.InnerHtml.AppendHtml(currentItem);
    if (PageModel.HasNextPage)
    {
        TagBuilder nextItem = CreateTag(PageModel.PageNumber + 1, urlHelper);
        tag.InnerHtml.AppendHtml(nextItem);
    }
    if (PageModel.PageNumber + 1 < PageModel.TotalPages)
    {
        TagBuilder lastItem = CreateTag(PageModel.TotalPages, urlHelper);
        tag.InnerHtml.AppendHtml(lastItem);
    }
    output.Content.AppendHtml(tag);
}

TagBuilder CreateTag(int pageNumber, IUrlHelper urlHelper)
{
    TagBuilder item = new TagBuilder("li");
    TagBuilder link = new TagBuilder("a");
    if (pageNumber == this.PageModel.PageNumber)
    {
        item.AddCssClass("active");
    }
    else
    {
        link.Attributes["href"] = urlHelper.Action(PageAction, new { page = pageNumber, id = ModelId, sorter = Sorter });
    }
    item.AddCssClass("page-item");
    item.AddCssClass("page-item-custom");
    link.AddCssClass("page-link");
    link.InnerHtml.Append(pageNumber.ToString());
    item.InnerHtml.AppendHtml(link);
    return item;
}
}
}

```

## Листинг *SortHeaderTagHelper*

```

using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.Rendering;
using Microsoft.AspNetCore.Mvc.Routing;
using Microsoft.AspNetCore.Mvc.ViewFeatures;
using Microsoft.AspNetCore.Razor.TagHelpers;
using RadiostationWeb.Data;

namespace RadiostationWeb.TagHelpers
{
    public class SortHeaderTagHelper : TagHelper
    {
        public SortState Property { get; set; }
        public SortState Current { get; set; }
        public string Action { get; set; }
    }
}

```

```

public bool Up { get; set; }

private IUrlHelperFactory urlHelperFactory;
public SortHeaderTagHelper(IUrlHelperFactory helperFactory)
{
    urlHelperFactory = helperFactory;
}
[ViewContext]
[HtmlAttributeNotBound]
public ViewContext ViewContext { get; set; }

public override void Process(TagHelperContext context, TagHelperOutput output)
{
    IUrlHelper urlHelper = urlHelperFactory.GetUrlHelper(ViewContext);
    output.TagName = "a";
    string url = urlHelper.Action(Action, new { sortOrder = Property });
    output.Attributes.SetAttribute("href", url);
    if (Current == Property)
    {
        TagBuilder tag = new TagBuilder("i");
        tag.AddCssClass("glyphicon");

        if (Up == true)
            tag.AddCssClass("glyphicon-chevron-up");
        else
            tag.AddCssClass("glyphicon-chevron-down");

        output.PreContent.AppendHtml(tag);
    }
}
}
}

```

## Листинг *AccountController*

```

using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.Rendering;
using RadiostationWeb.Data;
using RadiostationWeb.Models;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace RadiostationWeb.Controllers
{
    public class AccountController : Controller
    {
        private readonly UserManager<ApplicationUser> _userManager;
        private readonly SignInManager<ApplicationUser> _signInManager;
        private readonly RadiostationWebDbContext _dbContext;
        public AccountController(UserManager<ApplicationUser> userManager,
            SignInManager<ApplicationUser> signInManager,
            RadiostationWebDbContext dbContext)
        {
            _userManager = userManager;
            _signInManager = signInManager;
            _dbContext = dbContext;
        }

        public ActionResult Login(string returnUrl = "/")

```

```

{
    if (User.Identity.IsAuthenticated)
    {
        return Redirect(returnUrl);
    }
    return View(new LoginViewModel
    {
        returnUrl = returnUrl,
    });
}

[HttpPost]
public async Task<ActionResult> Login(LoginViewModel loginModel)
{
    if (ModelState.IsValid)
    {
        var user = await _userManager.FindByNameAsync(loginModel.Username);
        if (user != null)
        {
            await _signInManager.SignOutAsync();
            var signInResult = await _signInManager.PasswordSignInAsync(user, loginModel.Password, false, false);
            if (signInResult.Succeeded)
            {
                return Redirect(loginModel?.ReturnUrl ?? "/");
            }
        }
    }
    ModelState.AddModelError("", "Invalid name or password");
    return View(loginModel);
}

[Authorize]
public async Task<RedirectResult> Logout(string returnUrl = "/")
{
    await _signInManager.SignOutAsync();
    return Redirect(returnUrl);
}

[AuthorizeRoles(RoleType.Admin, RoleType.HR_Employee)]
public ActionResult ManageUsers(string nameFilter, string surnameFilter, int page = 1)
{
    var pageSize = 10;
    var users = FilterUsers(nameFilter, surnameFilter);
    var pageUsers = users.OrderBy(o => o.UserName).Skip((page - 1) * pageSize).Take(pageSize);
    PageViewModel pageViewModel = new PageViewModel(users.Count(), page, pageSize);
    var viewUsers = pageUsers.ToList();
    var pageItemsModel = new PageItemsModel<ApplicationUser> { Items = viewUsers, PageModel = pageViewModel };
    return View(pageItemsModel);
}

private IQueryable<ApplicationUser> FilterUsers(string nameFilter, string surnameFilter)
{
    IQueryable<ApplicationUser> users = _userManager.Users;
    nameFilter = nameFilter ?? HttpContext.Request.Cookies["nameFilter"];
    if (!string.IsNullOrEmpty(nameFilter))
    {
        users = users.Where(e => e.Name.Contains(nameFilter));
        HttpContext.Response.Cookies.Append("nameFilter", nameFilter);
    }
}

```

```

surnameFilter = surnameFilter ?? HttpContext.Request.Cookies["surnameFilter"];
if (!string.IsNullOrEmpty(surnameFilter))
{
    users = users.Where(e => e.Surname.Contains(surnameFilter));
    HttpContext.Response.Cookies.Append("surnameFilter", surnameFilter);
}
return users;
}

public IActionResult ResetManageFilter()
{
    HttpContext.Response.Cookies.Delete("nameFilter");
    HttpContext.Response.Cookies.Delete("surnameFilter");
    return RedirectToAction(nameof(ManageUsers));
}

[AuthorizeRoles(RoleType.Admin, RoleType.HR_Employee)]
public async Task<ActionResult> AddRole(string userId, string roleName)
{
    var user = await _userManager.FindByIdAsync(userId);
    if (user != null && roleName != null)
    {
        await _userManager.AddToRoleAsync(user, roleName);
        if (roleName.Equals(RoleType.Employee))
        {
            Employee employee = new Employee { AspNetUserId = userId };
            if (!_dbContext.Employees.ToList().Any(o => o.AspNetUserId == userId))
            {
                _dbContext.Employees.Add(employee);
                _dbContext.SaveChanges();
            }
        }
    }
    return RedirectToAction(nameof(ManageUsers));
}

[AuthorizeRoles(RoleType.Admin, RoleType.HR_Employee)]
public async Task<ActionResult> DeleteRole(string userId, string roleName)
{
    var currentUserId = _userManager.GetUserId(User);
    if (currentUserId != userId || roleName != RoleType.Admin || roleName != RoleType.HR_Employee)
    {
        var user = await _userManager.FindByIdAsync(userId);
        var employee = _dbContext.Employees.FirstOrDefault(o => o.AspNetUserId.Equals(userId));
        if (user != null && roleName != null)
        {
            await _userManager.RemoveFromRoleAsync(user, roleName);
            if (roleName.Equals(RoleType.Employee))
            {
                if (!_dbContext.Broadcasts.ToList().Any(o => o.EmployeeId == employee.Id))
                {
                    _dbContext.Employees.Remove(employee);
                    _dbContext.SaveChanges();
                    return RedirectToAction(nameof(ManageUsers));
                }
            }
        }
    }
    return RedirectToAction(nameof(ManageUsers));
}

[AuthorizeRoles(RoleType.Admin, RoleType.HR_Employee)]

```



```

public async Task<ActionResult> Delete(string userId)
{
    var currentUserId = _userManager.GetUserId(User);
    if (userId != null && userId != currentUserId)
    {
        var user = await _userManager.FindByIdAsync(userId);
        var userRoles = await _userManager.GetRolesAsync(user);
        var emplRole = userRoles.Contains("Employee");
        if (emplRole)
        {
            var employee = _dbContext.Employees.FirstOrDefault(o => o.AspNetUserId.Equals(userId));

            if (!dbContext.Broadcasts.ToList().Any(o => o.EmployeeId == employee.Id))
            {
                await _userManager.DeleteAsync(user);
            }
        }
        else
        {
            await _userManager.DeleteAsync(user);
        }
    }

    return RedirectToAction(nameof(ManageUsers));
}

[AuthorizeRoles(RoleType.Admin, RoleType.HR_Employee)]
public ActionResult Create()
{
    return View(new RegistrationViewModel());
}

[AuthorizeRoles(RoleType.Admin, RoleType.HR_Employee)]
[HttpPost]
public async Task<ActionResult> Create(RegistrationViewModel registrationModel, string userRole = RoleType.User)
{
    if (registrationModel.Username != null && registrationModel.Email != null)
    {
        if (await _userManager.FindByNameAsync(registrationModel.Username) != null)
        {
            ModelState.AddModelError("", "Username already exists");
        }

        if (await _userManager.FindByEmailAsync(registrationModel.Email) != null)
        {
            ModelState.AddModelError("", "Email already exists");
        }
    }

    if (ModelState.IsValid)
    {
        var user = new ApplicationUser
        {
            UserName = registrationModel.Username,
            Email = registrationModel.Email,
            Name = registrationModel.Name,
            Surname = registrationModel.Surname,
            MiddleName = registrationModel.MiddleName
        };
        var creatingResult = await _userManager.CreateAsync(user, registrationModel.Password);
    }
}

```

```

        if (creatingResult.Succeeded)
        {
            await _userManager.AddToRoleAsync(user, userRole);
            if (userRole.Equals(RoleType.Employee))
            {
                _dbContext.Employees.Add(new Employee { AspNetUserId = user.Id, Education="", PositionId = null, Work-
Time= null });
                _dbContext.SaveChanges();
            }
            return RedirectToAction(nameof(ManageUsers));
        }
    }

    return View(registrationModel);
}

[AuthorizeRoles(RoleType.Admin, RoleType.HR_Employee)]
public async Task<ActionResult> Edit(string userId)
{
    var currentUser = await _userManager.FindByIdAsync(userId);
    var userRoles = await _userManager.GetRolesAsync(currentUser);
    var emplRole = userRoles.Contains("Employee");
    var positions = _dbContext.Positions.Select(c => new SelectListItem
    {
        Value = c.Id.ToString(),
        Text = c.Name,
    }).ToList();

    IList<SelectListItem> educations = new List<SelectListItem>
    {
        new SelectListItem{ Text = "higher", Value = "higher"},
        new SelectListItem{ Text = "secondary", Value = "secondary"},
        new SelectListItem{ Text = "without education", Value = "without education"},
    };
    if (currentUser != null)
    {
        var userEditViewModel = new UserEmployeeEditViewModel
        {
            Id = currentUser.Id,
            UserName = currentUser.UserName,
            Email = currentUser.Email,
            Name = currentUser.Name,
            Surname = currentUser.Surname,
            MiddleName = currentUser.MiddleName,
            EmployeeRole = emplRole,
            PositionList = positions,
            EducationList = educations,
        };
        if (userEditViewModel.EmployeeRole)
        {
            var currentEmployee = _dbContext.Employees.FirstOrDefault(o => o.AspNetUserId == currentUser.Id);
            userEditViewModel.PositionId = currentEmployee.PositionId;
            userEditViewModel.Education = currentEmployee.Education;
            userEditViewModel.EmployeeId = currentEmployee.Id;
            userEditViewModel.WorkTime = currentEmployee.WorkTime;
        }

        return View(userEditViewModel);
    }
    else
    {

```

```

        return RedirectToAction("Error", "Home");
    }
}

[AuthorizeRoles(RoleType.Admin, RoleType.HR_Employee)]
[HttpPost]
public async Task<ActionResult> Edit(UserEmployeeEditViewModel user)
{
    var userByUsername = await _userManager.FindByNameAsync(user.UserName);

    if (userByUsername != null && userByUsername.Id != user.Id)
    {
        ModelState.AddModelError("", "Username already exists");
    }

    var userByEmail = await _userManager.FindByEmailAsync(user.Email);
    if (userByEmail != null && userByEmail.Id != user.Id)
    {
        ModelState.AddModelError("", "Email already exists");
    }

    if (ModelState.IsValid)
    {
        ApplicationUser currentUser = await _userManager.FindByIdAsync(user.Id);
        currentUser.UserName = user.UserName;
        currentUser.Email = user.Email;
        currentUser.Name = user.Name;
        currentUser.Surname = user.Surname;
        currentUser.MiddleName = user.MiddleName;
        await _userManager.UpdateAsync(currentUser);
        var userRoles = await _userManager.GetRolesAsync(currentUser);
        var emplRole = userRoles.Contains("Employee");

        if (emplRole)
        {
            var currentEmployee = _dbContext.Employees.FirstOrDefault(o=>o.AspNetUserId.Equals(currentUser.Id));
            currentEmployee.Education = user.Education;
            currentEmployee.PositionId = user.PositionId;
            currentEmployee.WorkTime = user.WorkTime;
            _dbContext.Employees.Update(currentEmployee);
            _dbContext.SaveChanges();
        }

        ViewData["SuccessMessage"] = "Successfully edited";
    }
    return RedirectToAction(nameof(ManageUsers));
}

public ActionResult Registration(string returnUrl = "/")
{
    if (User.Identity.IsAuthenticated)
    {
        return Redirect(returnUrl);
    }
    return View(new RegistrationViewModel());
}

[HttpPost]
public async Task<ActionResult> Registration(RegistrationViewModel registrationModel)
{
    if (await _userManager.FindByNameAsync(registrationModel.Username) != null)
    {
        ModelState.AddModelError("", "Username already exists");
    }
}

```

```

    }

    if (await _userManager.FindByEmailAsync(registrationModel.Email) != null)
    {
        ModelState.AddModelError("", "Email already exists");
    }

    if (ModelState.IsValid)
    {
        var user = new ApplicationUser
        {
            UserName = registrationModel.Username,
            Email = registrationModel.Email,
            Name = registrationModel.Name,
            Surname = registrationModel.Surname,
            MiddleName = registrationModel.MiddleName
        };
        var creatingResult = await _userManager.CreateAsync(user, registrationModel.Password);
        if (creatingResult.Succeeded)
        {
            await _userManager.AddToRoleAsync(user, RoleType.User);
            await _signInManager.PasswordSignInAsync(user, registrationModel.Password, false, false);
            return RedirectToAction("Index", "Home");
        }
    }

    return View(registrationModel);
}
}
}

```

## Листинг *BroadcastController*

```

using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.Rendering;
using RadiostationWeb.Data;
using RadiostationWeb.Models;
using System;
using System.Collections.Generic;
using System.Linq;

namespace RadiostationWeb.Controllers
{
    public class BroadcastController : Controller
    {
        private readonly RadiostationWebDbContext _dbContext;
        private readonly ApplicationDbContext _applicationDbContext;
        public BroadcastController(RadiostationWebDbContext context, ApplicationDbContext applicationDbContext)
        {
            _dbContext = context;
            _applicationDbContext = applicationDbContext;
        }

        public ActionResult Broadcasts(DateTime? start, DateTime? end, SortState sortOrder = SortState.DateAsc, int page = 1)
        {
            var pageSize = 10;
            var broadcasts = FilterBroadcasts(start, end).ToList();
            var pageBroadcasts = broadcasts.OrderByDescending(o => o.Id).Skip((page - 1) * pageSize).Take(pageSize);
            PageViewModel pageViewModel = new PageViewModel(broadcasts.Count(), page, pageSize);
            var viewBroadcasts = from b in pageBroadcasts
                                join e in _dbContext.Employees.ToList() on b.EmployeeId equals e.Id
                                join a in _applicationDbContext.Users.ToList() on e.AspNetUserId equals a.Id
                                join r in _dbContext.Records on b.RecordId equals r.Id

```

```

        select new BroadcastViewModel
        {
            Id = b.Id,
            EmployeeName = a.Name,
            EmployeeSurname = a.Surname,
            DateAndTime = b.DateAndTime,
            RecordName = r.CompositionName,
        };
viewBroadcasts = sortOrder switch
{
    SortState.NameDsc => viewBroadcasts.OrderByDescending(s => s.EmployeeName),
    SortState.NameAsc => viewBroadcasts.OrderBy(s => s.EmployeeName),
    SortState.SurnameAsc => viewBroadcasts.OrderBy(s => s.EmployeeSurname),
    SortState.SurnameDsc => viewBroadcasts.OrderByDescending(s => s.EmployeeSurname),
    SortState.RecorNameAsc => viewBroadcasts.OrderBy(s => s.RecordName),
    SortState.RecordNameDsc => viewBroadcasts.OrderByDescending(s => s.RecordName),
    SortState.DateDsc => viewBroadcasts.OrderByDescending(s => s.DateAndTime),
    _ => viewBroadcasts.OrderByDescending(s => s.DateAndTime),
};
var pageItemsModel = new BroadcastsItemModel
{
    Items = viewBroadcasts,
    PageModel = pageViewModel,
    SortViewModel = new SortViewModel(sortOrder)
};
return View(pageItemsModel);
}

public IActionResult ResetFilter()
{
    HttpContext.Response.Cookies.Delete("broadcastsFrom");
    HttpContext.Response.Cookies.Delete("broadcastsTo");
    return RedirectToAction(nameof(Broadcasts));
}

public IActionResult ResetManageFilter()
{
    HttpContext.Response.Cookies.Delete("broadcastsFrom");
    HttpContext.Response.Cookies.Delete("broadcastsTo");
    return RedirectToAction(nameof(ManageBroadcasts));
}

private IQueryable<Broadcast> FilterBroadcasts(DateTime? start, DateTime? end)
{
    IQueryable<Broadcast> broadcasts = _dbContext.Broadcasts;
    var cookiesStringStart = HttpContext.Request.Cookies["broadcastsFrom"];
    var cookiesStringEnd = HttpContext.Request.Cookies["broadcastsTo"];
    DateTime? cookiesStart = null;
    DateTime? cookiesEnd = null;
    if (end is null && start is null && !string.IsNullOrEmpty(cookiesStringStart) && !string.IsNullOrEmpty(cookiesStringEnd))
    {
        cookiesStart = DateTime.Parse(cookiesStringStart);
        cookiesEnd = DateTime.Parse(cookiesStringEnd);
    }
    start ??= cookiesStart;
    end ??= cookiesEnd;
    if (start != null && end != null)
    {
        broadcasts = broadcasts.Where(e => e.DateAndTime > start && e.DateAndTime < end);
        HttpContext.Response.Cookies.Append("broadcastsFrom", start.ToString());
        HttpContext.Response.Cookies.Append("broadcastsTo", end.ToString());
    }
}

```

```

    }
    return broadcasts;
}

[AuthorizeRoles(RoleType.Admin, RoleType.Employee)]
public ActionResult ManageBroadcasts(DateTime? start, DateTime? end, SortState sortOrder = SortState.DateAsc, int page = 1)
{
    var pageSize = 10;
    var broadcasts = FilterBroadcasts(start, end).ToList();
    var pageBroadcasts = broadcasts.OrderByDescending(o => o.Id).Skip((page - 1) * pageSize).Take(pageSize);
    PageViewModel pageViewModel = new PageViewModel(broadcasts.Count(), page, pageSize);
    var viewBroadcasts = from b in pageBroadcasts
        join e in _dbContext.Employees.ToList() on b.EmployeeId equals e.Id
        join a in _applicationDbContext.Users.ToList() on e.AspNetUserId equals a.Id
        join r in _dbContext.Records on b.RecordId equals r.Id
        select new BroadcastViewModel
        {
            Id = b.Id,
            EmployeeName = a.Name,
            EmployeeSurname = a.Surname,
            DateAndTime = b.DateAndTime,
            RecordName = r.CompositionName,
        };
    viewBroadcasts = sortOrder switch
    {
        SortState.NameDsc => viewBroadcasts.OrderByDescending(s => s.EmployeeName),
        SortState.NameAsc => viewBroadcasts.OrderBy(s => s.EmployeeName),
        SortState.SurnameAsc => viewBroadcasts.OrderBy(s => s.EmployeeSurname),
        SortState.SurnameDsc => viewBroadcasts.OrderByDescending(s => s.EmployeeSurname),
        SortState.RecorNameAsc => viewBroadcasts.OrderBy(s => s.RecordName),
        SortState.RecordNameDsc => viewBroadcasts.OrderByDescending(s => s.RecordName),
        SortState.DateDsc => viewBroadcasts.OrderByDescending(s => s.DateAndTime),
        _ => viewBroadcasts.OrderByDescending(s => s.DateAndTime),
    };
    var pageItemsModel = new BroadcastsItemModel
    {
        Items = viewBroadcasts,
        PageModel = pageViewModel,
        SortViewModel = new SortViewModel(sortOrder)
    };
    return View(pageItemsModel);
}

[AuthorizeRoles(RoleType.Admin, RoleType.Employee)]
public ActionResult Delete(int id)
{
    var broadcast = _dbContext.Broadcasts.Find(id);
    if (broadcast != null)
    {
        try
        {
            _dbContext.Broadcasts.Remove(broadcast);
            _dbContext.SaveChanges();
        }
        catch
        {
            return RedirectToAction("Error", "Home",
                new { message = "Broadcast contain related data and cannot be deleted" });
        }
    }
    else

```

```

    {
        return RedirectToAction("Error", "Home",
            new { message = "Broadcast not found" });
    }

    return RedirectToAction(nameof(ManageBroadcasts));
}

[AuthorizeRoles(RoleType.Admin, RoleType.Employee)]
public ActionResult Create()
{
    var employees = _dbContext.Employees.ToList()
        .Join(_applicationDbContext.Users.ToList(),
            e => e.AspNetUserId, t => t.Id,
            (e, t) => new SelectListItem
            {
                Value = e.Id.ToString(),
                Text = t.Name + " " + t.Surname,
            }).ToList();
    var records = _dbContext.Records
        .Select(c => new SelectListItem { Value = c.Id.ToString(), Text = c.CompositionName }).ToList();

    return View(new CreateBroadcastViewModel { EmployeeList = employees, RecordList = records });
}

[AuthorizeRoles(RoleType.Admin, RoleType.Employee)]
[HttpPost]
public ActionResult Create(CreateBroadcastViewModel broadcast)
{
    if (ModelState.IsValid)
    {
        _dbContext.Broadcasts.Add((new Broadcast
        {
            EmployeeId = broadcast.EmployeeId,
            RecordId = broadcast.RecordId,
            DateAndTime = broadcast.DateAndTime }));
        _dbContext.SaveChanges();
        return RedirectToAction(nameof(ManageBroadcasts));
    }
    var employees = _dbContext.Employees.ToList()
        .Join(_applicationDbContext.Users.ToList(),
            e => e.AspNetUserId, t => t.Id,
            (e, t) => new SelectListItem
            {
                Value = e.Id.ToString(),
                Text = t.Name + " " + t.Surname,
            }).ToList();
    var records = _dbContext.Records
        .Select(c => new SelectListItem { Value = c.Id.ToString(), Text = c.CompositionName }).ToList();

    broadcast.RecordList = records;
    broadcast.EmployeeList = employees;

    return View(broadcast);
}

[AuthorizeRoles(RoleType.Admin, RoleType.Employee)]
public ActionResult Edit(int id)
{
    var broadcast = _dbContext.Broadcasts.Find(id);
    var employees = _dbContext.Employees.ToList()
        .Join(_applicationDbContext.Users.ToList(),

```

```

        e => e.AspNetUserId, t => t.Id,
        (e, t) => new SelectListItem
        {
            Value = e.Id.ToString(),
            Text = t.Name + " " + t.Surname,
            Selected = broadcast.EmployeeId == e.Id,
        })
        .ToList();
var records = _dbContext.Records
    .Select(c => new SelectListItem
    {
        Value = c.Id.ToString(),
        Text = c.CompositionName,
        Selected = broadcast.RecordId == c.Id
    })
    .ToList();
if (broadcast != null)
{
    return View(new EditBroadcastViewModel
    {
        Id=id,EmployeeList = employees,
        RecordList = records,
        DateAndTime = broadcast.DateAndTime,
        EmployeeId=broadcast.EmployeeId,
        RecordId=broadcast.RecordId
    });
}
else
{
    return RedirectToAction("Error", "Home",
        new { message = "broadcast not found" });
}
}

[AuthorizeRoles(RoleType.Admin, RoleType.Employee)]
[HttpPost]
public ActionResult Edit(EditBroadcastViewModel broadcast)
{
    if (ModelState.IsValid)
    {
        _dbContext.Broadcasts.Update(new Broadcast
        {
            Id = broadcast.Id,
            EmployeeId = broadcast.EmployeeId,
            RecordId = broadcast.RecordId,
            DateAndTime = broadcast.DateAndTime
        });
        if (_dbContext.SaveChanges() != 0)
        {
            ViewData["SuccessMessage"] = "Information has been successfully edited";
            return RedirectToAction(nameof(ManageBroadcasts));
        }
    }
    var employees = _dbContext.Employees.ToList()
        .Join(_applicationDbContext.Users.ToList(),
            e => e.AspNetUserId, t => t.Id,
            (e, t) => new SelectListItem
            {
                Value = e.Id.ToString(),
                Text = t.Name + " " + t.Surname,
            }).ToList();
    var records = _dbContext.Records

```



```

        .Select(c => new SelectListItem { Value = c.Id.ToString(), Text = c.CompositionName }).ToList();

        broadcast.RecordList = records;
        broadcast.EmployeeList = employees;
        return View(broadcast);
    }
}
}

```

## Листинг *EmployeeController*

```

using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using RadiostationWeb.Data;
using RadiostationWeb.Models;
using System.Linq;

namespace RadiostationWeb.Controllers
{
    public class EmployeeController : Controller
    {
        private readonly RadiostationWebDbContext _dbContext;
        private readonly ApplicationDbContext _applicationDbContext;

        public EmployeeController(RadiostationWebDbContext context, ApplicationDbContext applicationDbContext)
        {
            _dbContext = context;
            _applicationDbContext = applicationDbContext;
        }

        [Authorize]
        public ActionResult Employees(int page = 1)
        {
            var pageSize = 10;
            var employees = _dbContext.Employees.ToList();
            var pageEmployees = employees.OrderBy(o => o.Id).Skip((page - 1) * pageSize).Take(pageSize);
            PageViewModel pageViewModel = new PageViewModel(employees.Count(), page, pageSize);
            var positions = _dbContext.Positions.ToList();
            var viewEmployees = from e in employees
                               join a in _applicationDbContext.Users.ToList() on e.AspNetUserId equals a.Id
                               select new EmployeeViewModel
                               {
                                   Id = e.Id,
                                   AspNetUserId = e.AspNetUserId,
                                   Name = a.Name,
                                   Surname = a.Surname,
                                   MiddleName = a.MiddleName,
                                   Username = a.UserName,
                                   Email = a.Email,
                                   Education = e.Education,
                                   PositionName = positions.FirstOrDefault(g => g.Id == e.PositionId)?.Name,
                                   WorkTime = e.WorkTime,
                               };
            var pageItemsModel = new PageItemsModel<EmployeeViewModel> { Items = viewEmployees, PageModel = pageView-
Model };
            return View(pageItemsModel);
        }
    }
}

```

## Листинг *GenreController*

```
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using RadiostationWeb.Data;
using RadiostationWeb.Models;
using System.Linq;

namespace RadiostationWeb.Controllers
{
    public class GenreController : Controller
    {
        private readonly RadiostationWebDbContext _dbContext;
        public GenreController(RadiostationWebDbContext context)
        {
            _dbContext = context;
        }

        [Authorize]
        public ActionResult Genres(string genreNameFilter, int page = 1)
        {
            var pageSize = 10;
            var genres = FilterGenres(genreNameFilter);
            var pageGenres = genres.OrderByDescending(o => o.Id).Skip((page - 1) * pageSize).Take(pageSize);
            PageViewModel pageViewModel = new PageViewModel(genres.Count(), page, pageSize);
            var viewGenres = pageGenres.ToList();

            var pageItemsModel = new PageItemsModel<Genre> { Items = viewGenres, PageModel = pageViewModel };
            return View(pageItemsModel);
        }

        public IActionResult ResetFilter()
        {
            HttpContext.Response.Cookies.Delete("genreNameFilter");
            return RedirectToAction(nameof(Genres));
        }

        private IQueryable<Genre> FilterGenres(string genreNameFilter)
        {
            IQueryable<Genre> genres = _dbContext.Genres;
            genreNameFilter = genreNameFilter ?? HttpContext.Request.Cookies["genreNameFilter"];
            if (!string.IsNullOrEmpty(genreNameFilter))
            {
                genres = genres.Where(e => e.Description.Contains(genreNameFilter));
                HttpContext.Response.Cookies.Append("genreNameFilter", genreNameFilter);
            }
            return genres;
        }

        [AuthorizeRoles(RoleType.Admin, RoleType.Employee)]
        public ActionResult ManageGenres(int page = 1)
        {
            IQueryable<Genre> genres = _dbContext.Genres;
            var pageSize = 10;
            var pageGenres = genres.OrderByDescending(o => o.Id).Skip((page - 1) * pageSize).Take(pageSize);
            PageViewModel pageViewModel = new PageViewModel(genres.Count(), page, pageSize);
            var viewGenres = pageGenres.ToList();
            var pageItemsModel = new PageItemsModel<Genre> { Items = viewGenres, PageModel = pageViewModel };
            return View(pageItemsModel);
        }
    }
}
```

```
[AuthorizeRoles(RoleType.Admin, RoleType.Employee)]
public ActionResult Delete(int id)
{
    var genre = _dbContext.Genres.Find(id);
    if (genre != null)
    {
        try
        {
            _dbContext.Genres.Remove(genre);
            _dbContext.SaveChanges();
        }
        catch
        {
            return RedirectToAction("Error", "Home",
                new { message = "Genre contain related data and cannot be deleted" });
        }
    }
    else
    {
        return RedirectToAction("Error", "Home",
            new { message = "Genre not found" });
    }

    return RedirectToAction(nameof(ManageGenres));
}
```

```
[AuthorizeRoles(RoleType.Admin, RoleType.Employee)]
public ActionResult Create()
{
    return View(new Genre());
}
```

```
[AuthorizeRoles(RoleType.Admin, RoleType.Employee)]
[HttpPost]
public ActionResult Create(Genre genre)
{
    if (ModelState.IsValid)
    {
        _dbContext.Genres.Add(genre);
        _dbContext.SaveChanges();
        RedirectToAction(nameof(ManageGenres));
    }

    return View(genre);
}
```

```
[AuthorizeRoles(RoleType.Admin, RoleType.Employee)]
public ActionResult Edit(int id)
{
    var genre = _dbContext.Genres.Find(id);
    if (genre != null)
    {
        return View(genre);
    }
    else
    {
        return RedirectToAction("Error", "Home",
            new { message = "Genre not found" });
    }
}
```

```
[AuthorizeRoles(RoleType.Admin, RoleType.Employee)]
[HttpPost]
```

```

public ActionResult Edit(Genre genre)
{
    if (ModelState.IsValid)
    {
        _dbContext.Genres.Update(genre);
        if (_dbContext.SaveChanges() != 0)
        {
            ViewData["SuccessMessage"] = "Information has been successfully edited";
            return RedirectToAction(nameof(ManageGenres));
        }
    }

    return View(genre);
}
}
}

```

## Листинг *GroupController*

```

using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using RadiostationWeb.Data;
using RadiostationWeb.Models;
using System.Linq;

namespace RadiostationWeb.Controllers
{
    public class GroupController : Controller
    {
        private readonly RadiostationWebDbContext _dbContext;
        public GroupController(RadiostationWebDbContext context)
        {
            _dbContext = context;
        }

        [Authorize]
        public ActionResult Groups(string groupFilter, int page = 1)
        {
            var pageSize = 10;
            var groups = FilterGroups(groupFilter);
            var pageGroups = groups.OrderByDescending(o => o.Id).Skip((page - 1) * pageSize).Take(pageSize);
            PageViewModel pageViewModel = new PageViewModel(groups.Count(), page, pageSize);
            var viewGroups = pageGroups.ToList();

            var pageItemsModel = new PageItemsModel<Group> { Items = viewGroups, PageModel = pageViewModel };
            return View(pageItemsModel);
        }

        [Authorize]
        public ActionResult GroupDetail(int id, int page = 1)
        {
            var pageSize = 10;
            var groupPage = _dbContext.Groups.FirstOrDefault(o => o.Id.Equals(id));

            var groupPerformerDetail = _dbContext.Performers.ToList().Where(o => o.GroupId.Equals(id));
            var pageDetail = groupPerformerDetail.OrderByDescending(o => o.Id).Skip((page - 1) * pageSize).Take(pageSize);
            PageViewModel pageViewModel = new PageViewModel(groupPerformerDetail.Count(), page, pageSize);

            var viewDetails = from p in pageDetail
                             select new GroupDetailViewModel
                             {
                                 PerformerName = p.Name,

```

```

        PerformerSurname = p.Surname
    };
    return View(new GroupItemsViewModel {
        GroupsItems = viewDetails,
        GroupName = groupPage.Description,
        PageModel = pageViewModel,
        Id = groupPage.Id
    });
}

public IActionResult ResetFilter()
{
    HttpContext.Response.Cookies.Delete("groupFilter");
    return RedirectToAction(nameof(Groups));
}

private IQueryable<Group> FilterGroups(string groupFilter)
{
    IQueryable<Group> groups = _dbContext.Groups;
    groupFilter = groupFilter ?? HttpContext.Request.Cookies["groupFilter"];
    if (!string.IsNullOrEmpty(groupFilter))
    {
        groups = groups.Where(e => e.Description.Contains(groupFilter));
        HttpContext.Response.Cookies.Append("groupFilter", groupFilter);
    }
    return groups;
}

[AuthorizeRoles(RoleType.Admin, RoleType.Employee)]
public ActionResult ManageGroups(int page = 1)
{
    IQueryable<Group> groups = _dbContext.Groups;
    var pageSize = 10;
    var pageGroups = groups.OrderByDescending(o => o.Id).Skip((page - 1) * pageSize).Take(pageSize);
    PageViewModel pageViewModel = new PageViewModel(groups.Count(), page, pageSize);
    var viewGroups = pageGroups.ToList();
    var pageItemsModel = new PageItemsModel<Group> { Items = viewGroups, PageModel = pageViewModel };
    return View(pageItemsModel);
}

[AuthorizeRoles(RoleType.Admin, RoleType.Employee)]
public ActionResult Delete(int id)
{
    var group = _dbContext.Groups.Find(id);
    if (group != null)
    {
        try
        {
            _dbContext.Groups.Remove(group);
            _dbContext.SaveChanges();
        }
        catch
        {
            return RedirectToAction("Error", "Home",
                new { message = "Group contain related data and cannot be deleted" });
        }
    }
    else
    {
        return RedirectToAction("Error", "Home",
            new { message = "Group not found" });
    }
}

```

```

        return RedirectToAction(nameof(ManageGroups));
    }

    [AuthorizeRoles(RoleType.Admin, RoleType.Employee)]
    public ActionResult Create()
    {
        return View(new Group());
    }

    [AuthorizeRoles(RoleType.Admin, RoleType.Employee)]
    [HttpPost]
    public ActionResult Create(Group group)
    {
        if (ModelState.IsValid)
        {
            _dbContext.Groups.Add(group);
            _dbContext.SaveChanges();
            return RedirectToAction(nameof(ManageGroups));
        }

        return View(group);
    }

    [AuthorizeRoles(RoleType.Admin, RoleType.Employee)]
    public ActionResult Edit(int id)
    {
        var group = _dbContext.Groups.Find(id);
        if (group != null)
        {
            return View(group);
        }
        else
        {
            return RedirectToAction("Error", "Home",
                new { message = "Group not found" });
        }
    }

    [AuthorizeRoles(RoleType.Admin, RoleType.Employee)]
    [HttpPost]
    public ActionResult Edit(Group group)
    {
        if (ModelState.IsValid)
        {
            _dbContext.Groups.Update(group);
            if (_dbContext.SaveChanges() != 0)
            {
                ViewData["SuccessMessage"] = "Information has been successfully edited";
                return RedirectToAction(nameof(ManageGroups));
            }
        }

        return View(group);
    }
}
}

```

### Листинг *HomeController*

```

using System.Diagnostics;
using System.IO;

```

```

using System.Threading.Tasks;
using Microsoft.AspNetCore.Hosting;
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;
using Microsoft.Extensions.Logging;
using RadiostationWeb.Data;
using RadiostationWeb.Models;

namespace RadiostationWeb.Controllers
{
    public class HomeController : Controller
    {
        private readonly ILogger<HomeController> _logger;
        private readonly RadiostationWebDbContext _dbContext;
        private readonly IWebHostEnvironment _environment;
        public HomeController(ILogger<HomeController> logger, RadiostationWebDbContext dbContext, IWebHostEnvironment environment)
        {
            _logger = logger;
            _dbContext = dbContext;
            _environment = environment;
        }

        public IActionResult Index()
        {
            return View(_dbContext.HomePageImages);
        }

        [AuthorizeRoles(RoleType.Admin, RoleType.Employee)]
        public ActionResult Create()
        {
            return View(new HomePageImage());
        }

        [AuthorizeRoles(RoleType.Admin, RoleType.Employee)]
        [HttpPost]
        public async Task<IActionResult> Create(IFormFile uploadedFile, HomePageImage image)
        {
            if (uploadedFile != null)
            {
                if (ModelState.IsValid)
                {
                    string path = "/img/" + uploadedFile.FileName;

                    using (var fileStream = new FileStream(_environment.WebRootPath + path, FileMode.Create))
                    {
                        await uploadedFile.CopyToAsync(fileStream);
                    }
                    HomePageImage file = new HomePageImage { SrcImg = path, ImgCaption = image.ImgCaption };
                    _dbContext.HomePageImages.Add(file);
                    _dbContext.SaveChanges();
                }
            }

            return RedirectToAction(nameof(Index));
        }

        [AuthorizeRoles(RoleType.Admin, RoleType.Employee)]
        public ActionResult Edit(int id)
        {
            var img = _dbContext.HomePageImages.Find(id);
            if (img != null)

```

```

    {
        return View(img);
    }
    else
    {
        return RedirectToAction("Error", "Home",
            new { message = "img not found" });
    }
}

[AuthorizeRoles(RoleType.Admin, RoleType.Employee)]
[HttpPost]
public async Task<ActionResult> Edit(HomePageImage homePageImage, IFormFile uploadedFile)
{
    if (homePageImage.Id <= 0)
    {
        return RedirectToAction("Error", "Home", new { message = "Image not found" });
    }

    if (uploadedFile != null && uploadedFile.Length > 0)
    {
        var img = _dbContext.HomePageImages.Find(homePageImage.Id);
        if (img != null)
        {
            if (ModelState.IsValid)
            {
                var imageName = $"{img.Id}{Path.GetExtension(uploadedFile.FileName)}";
                var imagePath = Path.Combine(_environment.WebRootPath, "img", imageName);
                using (Stream fileStream = new FileStream(imagePath, FileMode.Create))
                {
                    await uploadedFile.CopyToAsync(fileStream);
                }

                img.SrcImg = $"/img/{imageName}";
                img.ImgCaption = homePageImage.ImgCaption;

                _dbContext.SaveChanges();
            }
        }
        else
        {
            return RedirectToAction("Error", "Home", new { message = "Image not found" });
        }
    }

    return RedirectToAction(nameof(Index));
}

[AuthorizeRoles(RoleType.Admin, RoleType.Employee)]
public ActionResult Delete(int id)
{
    var img = _dbContext.HomePageImages.Find(id);
    if (img != null)
    {
        try
        {
            _dbContext.HomePageImages.Remove(img);
            _dbContext.SaveChanges();
        }
        catch
        {
            return RedirectToAction("Error", "Home",
                new { message = "img contain related data and cannot be deleted" });
        }
    }
}

```



```

    }
}
else
{
    return RedirectToAction("Error", "Home",
        new { message = "img not found" });
}

return RedirectToAction(nameof(Index));
}

public IActionResult Privacy()
{
    return View();
}

[ResponseCache(Duration = 0, Location = ResponseCacheLocation.None, NoStore = true)]
public IActionResult Error()
{
    return View(new ErrorViewModel { RequestId = Activity.Current?.Id ?? HttpContext.TraceIdentifier });
}
}
}

```

## Листинг *PerformerController*

```

using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.Rendering;
using RadiostationWeb.Data;
using RadiostationWeb.Models;
using System.Linq;

namespace RadiostationWeb.Controllers
{
    public class PerformerController : Controller
    {
        private readonly RadiostationWebDbContext _dbContext;
        public PerformerController(RadiostationWebDbContext context)
        {
            _dbContext = context;
        }
        [Authorize]
        public ActionResult Performers(string nameFilter, int? groupFilter, string surnameFilter, int page = 1)
        {
            var pageSize = 10;
            var performers = FilterPerformers(nameFilter, groupFilter, surnameFilter).ToList();
            var pagePerformers = performers.OrderByDescending(o => o.Id).Skip((page - 1) * pageSize).Take(pageSize);
            PageViewModel pageViewModel = new PageViewModel(performers.Count(), page, pageSize);
            var groups = _dbContext.Groups.ToList();

            var viewPerformers = from b in pagePerformers
                                select new PerformerViewModel
                                {
                                    Id = b.Id,
                                    Name = b.Name,
                                    Surname = b.Surname,
                                    GroupName = groups.FirstOrDefault(g => g.Id == b.GroupId)?.Description,
                                };

            var groupsList = _dbContext.Groups.Select(c => new SelectListItem { Value = c.Id.ToString(), Text = c.Descrip-
tion }).ToList();

```

```

        var pageItemsModel = new PerformersItemsModel { Items = viewPerformers, PageModel = pageViewModel, Select-
Groups = groupsList };
        return View(pageItemsModel);
    }

    public IActionResult ResetFilter()
    {
        HttpContext.Response.Cookies.Delete("nameFilter");
        HttpContext.Response.Cookies.Delete("groupFilter");
        HttpContext.Response.Cookies.Delete("surnameFilter");
        return RedirectToAction(nameof(Performers));
    }

    public IActionResult ResetManageFilter()
    {
        HttpContext.Response.Cookies.Delete("nameFilter");
        HttpContext.Response.Cookies.Delete("groupFilter");
        HttpContext.Response.Cookies.Delete("surnameFilter");
        return RedirectToAction(nameof(ManagePerformers));
    }

    private IQueryable<Performer> FilterPerformers(string nameFilter, int? groupFilter, string surnameFilter)
    {
        IQueryable<Performer> performers = _dbContext.Performers;
        nameFilter = nameFilter ?? HttpContext.Request.Cookies["nameFilter"];
        if (!string.IsNullOrEmpty(nameFilter))
        {
            performers = performers.Where(e => e.Name.Contains(nameFilter));
            HttpContext.Response.Cookies.Append("nameFilter", nameFilter);
        }
        int cookieGroupFilter;
        int.TryParse(HttpContext.Request.Cookies["groupFilter"], out cookieGroupFilter);
        groupFilter = groupFilter ?? cookieGroupFilter;
        if (groupFilter != 0)
        {
            performers = performers.Where(e => e.GroupId == groupFilter);
            HttpContext.Response.Cookies.Append("groupFilter", groupFilter.ToString());
        }

        surnameFilter = surnameFilter ?? HttpContext.Request.Cookies["surnameFilter"];
        if (!string.IsNullOrEmpty(surnameFilter))
        {
            performers = performers.Where(e => e.Surname.Contains(surnameFilter));
            HttpContext.Response.Cookies.Append("surnameFilter", surnameFilter);
        }

        return performers;
    }

    [AuthorizeRoles(RoleType.Admin, RoleType.Employee)]
    public ActionResult ManagePerformers(string nameFilter, int? groupFilter, string surnameFilter, int page = 1)
    {
        var performers = FilterPerformers(nameFilter, groupFilter, surnameFilter).ToList();
        var pageSize = 10;
        var pagePerformers = performers.ToList().OrderByDescending(o => o.Id).Skip((page - 1) * pageSize).Take(pageSize);
        PageViewModel pageViewModel = new PageViewModel(pagePerformers.Count(), page, pageSize);
        var groups = _dbContext.Groups.ToList();
        var viewPerformers = from b in pagePerformers
                             select new PerformerViewModel
                             {
                                 Id = b.Id,

```

```

        Name = b.Name,
        Surname = b.Surname,
        GroupName = groups.FirstOrDefault(g => g.Id == b.GroupId)?.Description,
    };
    var groupsList = _dbContext.Groups.Select(c => new SelectListItem { Value = c.Id.ToString(), Text = c.Description }).ToList();
    var pageItemsModel = new PerformersItemsModel { Items = viewPerformers, PageModel = pageViewModel, SelectGroups = groupsList };
    return View(pageItemsModel);
}

[AuthorizeRoles(RoleType.Admin, RoleType.Employee)]
public ActionResult Delete(int id)
{
    var performer = _dbContext.Performers.Find(id);
    if (performer != null)
    {
        try
        {
            _dbContext.Performers.Remove(performer);
            _dbContext.SaveChanges();
        }
        catch
        {
            return RedirectToAction("Error", "Home",
                new { message = "Performer contain related data and cannot be deleted" });
        }
    }
    else
    {
        return RedirectToAction("Error", "Home",
            new { message = "Performer not found" });
    }

    return RedirectToAction(nameof(ManagePerformers));
}

[AuthorizeRoles(RoleType.Admin, RoleType.Employee)]
public ActionResult Create()
{
    var groups = _dbContext.Groups.Select(c => new SelectListItem { Value = c.Id.ToString(), Text = c.Description }).ToList();
    return View(new CreatePerformerViewModel { GroupsList = groups });
}

[AuthorizeRoles(RoleType.Admin, RoleType.Employee)]
[HttpPost]
public ActionResult Create(CreatePerformerViewModel performer)
{
    if (ModelState.IsValid)
    {
        if (performer.GroupId == 0)
        {
            _dbContext.Performers.Add(new Performer
            {
                Name = performer.Name,
                Surname = performer.Surname,
            });
        }
        else
        {
            _dbContext.Performers.Add(new Performer
            {

```

```

        Name = performer.Name,
        Surname = performer.Surname,
        GroupId = performer.GroupId
    });
}

_dbContext.SaveChanges();
return RedirectToAction(nameof(ManagePerformers));
}
var groups = _dbContext.Groups.Select(c => new SelectListItem { Value = c.Id.ToString(), Text = c.Description }).ToList();
performer.GroupsList = groups;
return View(performer);
}

[AuthorizeRoles(RoleType.Admin, RoleType.Employee)]
public ActionResult Edit(int id)
{
    var performer = _dbContext.Performers.Find(id);
    var groups = _dbContext.Groups.Select(c => new SelectListItem
    {
        Value = c.Id.ToString(),
        Text = c.Description,
        Selected = performer.GroupId == c.Id,
    }).ToList();
    if (performer != null)
    {
        return View(new EditPerformerViewModel
        {
            Id = id,
            GroupsList = groups,
            Surname = performer.Surname,
            Name = performer.Name,
            GroupId = performer.GroupId,
        });
    }
    else
    {
        return RedirectToAction("Error", "Home",
            new { message = "Performer not found" });
    }
}

[AuthorizeRoles(RoleType.Admin, RoleType.Employee)]
[HttpPost]
public ActionResult Edit(EditPerformerViewModel performer)
{
    if (ModelState.IsValid)
    {
        _dbContext.Performers.Update(new Performer
        {
            Id = performer.Id,
            GroupId = performer.GroupId,
            Name = performer.Name,
            Surname = performer.Surname
        });
        if (_dbContext.SaveChanges() != 0)
        {
            ViewData["SuccessMessage"] = "Information has been successfully edited";
            return RedirectToAction(nameof(ManagePerformers));
        }
    }
    var groups = _dbContext.Groups.Select(c => new SelectListItem { Value = c.Id.ToString(), Text = c.Description }).ToList();

```

```

        performer.GroupsList = groups;
        return View(performer);
    }
}
}

```

## Листинг *RecordController*

```

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

namespace RadiostationWeb.Controllers
{
    public class RecordController : Controller
    {
        private readonly RadiostationWebDbContext _dbContext;
        public RecordController(RadiostationWebDbContext context)
        {
            _dbContext = context;
        }
        [Authorize]
        public async Task<ActionResult> Records(string nameFilter, int? performerFilter, int page = 1)
        {
            var pageSize = 10;
            var records = FilterRecords(nameFilter, performerFilter);
            var performers = await _dbContext.Performers.Select(c => new SelectListItem { Value = c.Id.ToString(), Text = c.Name + " " + c.Surname }).ToListAsync();
            var pageRecords = records.OrderByDescending(o => o.Id).Skip((page - 1) * pageSize).Take(pageSize);
            PageViewModel pageViewModel = new PageViewModel(records.Count(), page, pageSize);
            var viewRecords = pageRecords.ToList().Join(_dbContext.Performers.ToList(),
                e => e.PerformerId, t => t.Id,
                (e, t) => new RecordViewModel
                {
                    Id = e.Id,
                    PerformerName = t.Name,
                    GenreId = e.GenreId,
                    Album = e.Album,
                    RecordDate = (e.RecordDate),
                    Lasting = e.Lasting,
                    Rating = e.Rating,
                    CompositionName = e.CompositionName
                }).Join(_dbContext.Genres.ToList(), e => e.GenreId, t => t.Id,
                (e, t) => new RecordViewModel
                {
                    Id = e.Id,
                    PerformerName = e.PerformerName,
                    GenreName = t.GenreName,
                    Album = e.Album,
                    RecordDate = e.RecordDate,
                    Lasting = e.Lasting,
                    Rating = e.Rating,
                    CompositionName = e.CompositionName
                }
            );
        }
    }
}

```

```

    );
    var pageItemsModel = new RecordsItemModel { Items = viewRecords, PageModel = pageViewModel, SelectPerform-
ers = performers };
    return View(pageItemsModel);
}

public IActionResult ResetFilter()
{
    HttpContext.Response.Cookies.Delete("nameFilter");
    HttpContext.Response.Cookies.Delete("performerFilter");
    return RedirectToAction(nameof(Records));
}

public IActionResult ResetManageFilter()
{
    HttpContext.Response.Cookies.Delete("nameFilter");
    HttpContext.Response.Cookies.Delete("performerFilter");
    return RedirectToAction(nameof(ManageRecords));
}

private IQueryable<Record> FilterRecords(string nameFilter, int? performerFilter)
{
    IQueryable<Record> records = _dbContext.Records;
    nameFilter = nameFilter ?? HttpContext.Request.Cookies["nameFilter"];
    if (!string.IsNullOrEmpty(nameFilter))
    {
        records = records.Where(e => e.CompositionName.Contains(nameFilter));
        HttpContext.Response.Cookies.Append("nameFilter", nameFilter);
    }
    int cookiePerformerFilter;
    int.TryParse(HttpContext.Request.Cookies["performerFilter"], out cookiePerformerFilter);
    performerFilter = performerFilter ?? cookiePerformerFilter;
    if (performerFilter != 0)
    {
        records = records.Where(e => e.PerformerId == performerFilter);
        HttpContext.Response.Cookies.Append("performerFilter", performerFilter.ToString());
    }
    return records;
}

[AuthorizeRoles(RoleType.Admin, RoleType.Employee)]
public ActionResult ManageRecords(string nameFilter, int? performerFilter, int page = 1)
{
    var pageSize = 10;
    var records = FilterRecords(nameFilter, performerFilter);
    var performers = _dbContext.Performers.Select(c => new SelectList-
Item { Value = c.Id.ToString(), Text = c.Name + " " + c.Surname })
    .ToList();
    var pageRecords = records.ToList().OrderByDescending(o => o.Id).Skip((page - 1) * pageSize).Take(pageSize);
    PageViewModel pageViewModel = new PageViewModel(records.Count(), page, pageSize);
    var viewRecords = pageRecords.ToList().Join(_dbContext.Performers.ToList(),
    e => e.PerformerId, t => t.Id,
    (e, t) => new RecordViewModel
    {
        Id = e.Id,
        PerformerName = t.Name,
        GenreId = e.GenreId,
        Album = e.Album,
        RecordDate = (e.RecordDate),
        Lasting = e.Lasting,
        Rating = e.Rating,
    }
    );
}

```

```

        CompositionName = e.CompositionName
    }).Join(_dbContext.Genres.ToList(), e => e.GenreId, t => t.Id,
    (e, t) => new RecordViewModel
    {
        Id = e.Id,
        PerformerName = e.PerformerName,
        GenreName = t.GenreName,
        Album = e.Album,
        RecordDate = e.RecordDate,
        Lasting = e.Lasting,
        Rating = e.Rating,
        CompositionName = e.CompositionName
    }
    );
    var pageItemsModel = new RecordsItemModel { Items = viewRecords, PageModel = pageViewModel, SelectPerform-
ers = performers };
    return View(pageItemsModel);
}

[AuthorizeRoles(RoleType.Admin, RoleType.Employee)]
public ActionResult Delete(int id)
{
    var record = _dbContext.Records.Find(id);
    if (record != null)
    {
        try
        {
            _dbContext.Records.Remove(record);
            _dbContext.SaveChanges();
        }
        catch
        {
            return RedirectToAction("Error", "Home",
            new { message = "Record contain related data and cannot be deleted" });
        }
    }
    else
    {
        return RedirectToAction("Error", "Home",
        new { message = "Record not found" });
    }
}

return RedirectToAction(nameof(ManageRecords));
}

[AuthorizeRoles(RoleType.Admin, RoleType.Employee)]
public ActionResult Create()
{
    var performers = _dbContext.Performers.Select(c => new SelectList-
tItem { Value = c.Id.ToString(), Text = c.Name+" "+c.Surname })
    .ToList();

    var genres = _dbContext.Genres.Select(c => new SelectListItem { Value = c.Id.ToString(), Text = c.GenreName })
    .ToList();
    return View(new CreateRecordViewModel { PerformersList = performers, GenresList = genres });
}

[AuthorizeRoles(RoleType.Admin, RoleType.Employee)]
[HttpPost]
public ActionResult Create(CreateRecordViewModel record)
{
    if (ModelState.IsValid)
    {

```

```

        _dbContext.Records.Add(new Record
        {
            CompositionName = record.CompositionName,
            Album = record.Album,
            GenreId = record.GenreId,
            Lasting = record.Lasting,
            Rating = record.Rating,
            RecordDate = record.RecordDate,
            PerformerId = record.PerformerId
        });
        _dbContext.SaveChanges();
        return RedirectToAction(nameof(ManageRecords));
    }
    var performers = _dbContext.Performers.Select(c => new SelectList-
tItem { Value = c.Id.ToString(), Text = c.Name + " " + c.Surname })
    .ToList();

    var genres = _dbContext.Genres.Select(c => new SelectListItem { Value = c.Id.ToString(), Text = c.GenreName })
    .ToList();

    record.GenresList = genres;
    record.PerformersList = performers;

    return View(record);
}

[AuthorizeRoles(RoleType.Admin, RoleType.Employee)]
public ActionResult Edit(int id)
{
    var record = _dbContext.Records.Find(id);
    var performers = _dbContext.Performers.Select(c => new SelectListItem
    {
        Value = c.Id.ToString(),
        Text = c.Name + " " + c.Surname,
        Selected = record.PerformerId == c.Id
    }).ToList();

    var genres = _dbContext.Genres.Select(c => new SelectListItem
    {
        Value = c.Id.ToString(),
        Text = c.GenreName,
        Selected = record.GenreId == c.Id
    }).ToList();
    if (record != null)
    {
        return View(new EditRecordViewModel
        {
            Id = id,
            PerformersList = performers,
            GenresList = genres,
            Album = record.Album,
            GenreId = record.GenreId,
            PerformerId = record.PerformerId,
            RecordDate = record.RecordDate,
            Lasting = record.Lasting,
            Rating = record.Rating,
            CompositionName = record.CompositionName
        });
    }
    else
    {
        return RedirectToAction("Error", "Home",
            new { message = "Record not found" });
    }
}

```



```

    }
}

[AuthorizeRoles(RoleType.Admin, RoleType.Employee)]
[HttpPost]
public async Task<ActionResult> Edit(EditRecordViewModel record)
{
    if (ModelState.IsValid)
    {
        _dbContext.Records.Update(new Record
        {
            Id = record.Id,
            Album = record.Album,
            GenreId = record.GenreId,
            PerformerId = record.PerformerId,
            RecordDate = record.RecordDate,
            Lasting = record.Lasting,
            Rating = record.Rating,
            CompositionName = record.CompositionName
        });
        if (_dbContext.SaveChanges() != 0)
        {
            ViewData["SuccessMessage"] = "Information has been successfully edited";
            return RedirectToAction(nameof(ManageRecords));
        }
    }
}

var performers = await _dbContext.Performers.Select(c => new SelectList-
Item { Value = c.Id.ToString(), Text = c.Name + " " + c.Surname })
.ToListAsync();

var genres = await _dbContext.Genres.Select(c => new SelectListItem { Value = c.Id.ToString(), Text = c.GenreName })
.ToListAsync();

record.GenresList = genres;
record.PerformersList = performers;

return View(record);
}
}
}

```

## Листинг *Startup*

```

using Microsoft.AspNetCore.Authentication.Cookies;
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Localization;
using Microsoft.EntityFrameworkCore;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
using RadiostationWeb.Data;
using System.Globalization;

namespace RadiostationWeb
{
    public class Startup
    {
        public Startup(IConfiguration configuration)
        {

```

```

    Configuration = configuration;
}

public IConfiguration Configuration { get; }

// This method gets called by the runtime. Use this method to add services to the container.
public void ConfigureServices(IServiceCollection services)
{
    services.AddControllersWithViews();
    services.AddAuthentication(CookieAuthenticationDefaults.AuthenticationScheme)
        .AddCookie(options =>
        {
            options.LoginPath = new PathString("/Account/Login");
        });
    services.AddDbContext<ApplicationDbContext>(options =>
        options.UseSqlServer(Configuration["ConnectionStrings:RadiostationDb"]));
    services.AddIdentity<ApplicationUser, IdentityRole>(options => {
        options.Password.RequireDigit = true;
        options.Password.RequiredLength = 6;
        options.Password.RequireNonAlphanumeric = false;
        options.Password.RequireUppercase = true;
        options.Password.RequireLowercase = true;
    })
        .AddEntityFrameworkStores<ApplicationDbContext>();
    services.AddDbContext<RadiostationWebDbContext>(options =>
        options.UseSqlServer(Configuration["ConnectionStrings:RadiostationDb"]));
    services.AddDistributedMemoryCache();
    services.AddSession();
}

// This method gets called by the runtime. Use this method to configure the HTTP request pipeline.
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
    var appDefaultCulture = new CultureInfo("ru-RU")
    {
        NumberFormat =
        {
            NumberDecimalSeparator = ".",
        },
    };

    var supportedCultures = new[] { appDefaultCulture };

    app.UseRequestLocalization(new RequestLocalizationOptions
    {
        DefaultRequestCulture = new RequestCulture(appDefaultCulture,
            SupportedCultures = supportedCultures,
            SupportedUICultures = supportedCultures
        });
    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
    }
    else
    {
        app.UseExceptionHandler("/Home/Error");
    }

    app.UseSession();

    app.UseHttpsRedirection();
    app.UseStaticFiles();
}

```

```

app.UseRouting();
app.UseAuthentication();
app.UseAuthorization();

app.UseEndpoints(endpoints =>
{
    endpoints.MapControllerRoute(
        name: "default",
        pattern: "{controller=Home}/{action=Index}/{id?}");
    });
}
}
}

```

## Листинг *Program*

```

using Microsoft.AspNetCore.Hosting;
using Microsoft.Extensions.Hosting;

namespace RadiostationWeb
{
    public class Program
    {
        public static void Main(string[] args)
        {
            CreateHostBuilder(args).Build().Run();
        }

        public static IHostBuilder CreateHostBuilder(string[] args) =>
            Host.CreateDefaultBuilder(args)
                .ConfigureWebHostDefaults(webBuilder =>
                {
                    webBuilder.UseStartup<Startup>();
                });
    }
}

```

**ПРИЛОЖЕНИЕ Б**  
(обязательное)  
**Структура *web*-приложения**