Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра программного обеспечения информационных технологий

Дисциплина: Базы данных (БД)

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту на тему

Программное средство «Meloman»

БГУИР КП I–40 01 01 01 ПЗ

Студент: гр. 751003 Гринчик В.В.

Проверила: Марина И.М.

Минск, 2020

Содержание

[Введение 3](#_Toc59072596)

[1 Анализ Предметной области 4](#_Toc59072597)

[1.1 Анализ существующих аналогов. 4](#_Toc59072598)

[2 Постановка Задачи 7](#_Toc59072599)

[2.1 Цель 7](#_Toc59072600)

[2.2 Задачи 7](#_Toc59072601)

[3 Разработка программного средства 9](#_Toc59072602)

[3.1 Схемы алгоритмов решения задачи 9](#_Toc59072603)

[3.2 Описание базы данных приложения 13](#_Toc59072604)

[4 Руководство по использованию программного средства 21](#_Toc59072605)

[4.1 Работа с веб-приложением 21](#_Toc59072606)

[Заключение 34](#_Toc59072607)

[Список использованной литературы 35](#_Toc59072608)

[Приложение А 36](#_Toc59072609)

Введение

Не смотря на то что большую часть информации о мире человек поучает от органов зрения, слух я является невероятно важным способом мироощущения. В частности, наибольшее количество информации об обществе человек получает через слух. С древнейших времен одним из наиболее важных для человечества искусств была музыка.

Музыка – это искусство, в котором переживания, чувства и идеи выражаются ритмически и интонационно организованными звуками, а также сами произведения этого искусства. Оно обладает уникальным свойством: с помощью звуков оно доносит до слушателя мысли и чувства композитора, воплощенные в художественных образах.

Конечно, каждый слушатель по-своему воспринимает и понимает музыкальное произведение, составляет собственное мнение о его содержании и средства выразительности в соответствии со своими взглядами, вкусами, опытом. Однако созданные талантливым композитором мелодии, в сочетании с правильно подранными текстами, способны вызывать у людей определенные эмоции будь то ярость или спокойствие, радость или сожаление.

Музыка, которую человек слушает человек может очень многое о нем рассказать, зачастую гораздо больше чем сам человек. Людям со схожими музыкальными вкусами проще общаться, и знакомится. Однако, как правило человек редко заходит дальше картинки и если ему не нравится изображение человека он вряд ли обратит внимание музыку, которую возможный знакомый слушает. Но могут ли люди знакомится основываясь исключительно на музыкальных предпочтениях?

Для решения данного вопроса, было решено разработать веб-приложение, представляющее собой социальную сеть с системой знакомств, основанной на музыкальных предпочтениях пользователей. «Meloman» - это музыкальная социальная сеть, с возможность редактирования и наложения эффектов на музыкальные файлы онлайн.

# Анализ Предметной области

Социальная сеть — онлайн-платформа, которая используется для общения, знакомств, создания социальных отношений между людьми, которые имеют схожие интересы.

Для привлечения внимания к новой социальной сети за частую используются довольно необычные методы. Одним из таких методов является урезание информации, который представляет собой концентрацию на определенной области чувств человека. В частности, ныне чрезвычайно популярная социальная сеть “Instagram” изначально концентрировалась исключительно на визуальной составляющей мировосприятия.

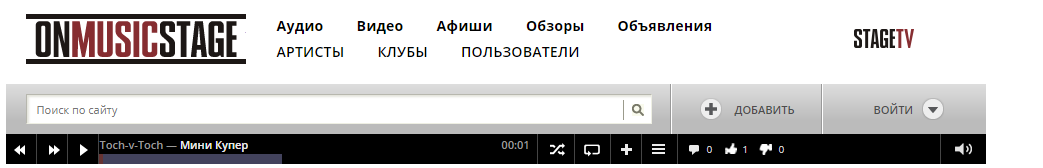
Однако не смотря на существующее множество социальных сетей, идея концентрации на звуковой составляющей мироощущения является свежей и в достаточной степени уникальной.

## Анализ существующих аналогов.

За счет того, что одним из основных преимуществ приложения “Meloman”, является его уникальность, количество его аналогов крайне ограничено. Кроме того, для еще большего выделения приложения среди аналогов, было решено добавить функцию аудио редактора.

Таким образом нахождение полных аналогов приложения стало не возможным. Однако существует множество частичных аналогов приложения. Итак, к ним относятся.

* 1. **.1 Onmusicstage.com/** – наиболее релевантная музыкальная соц. сеть в google.



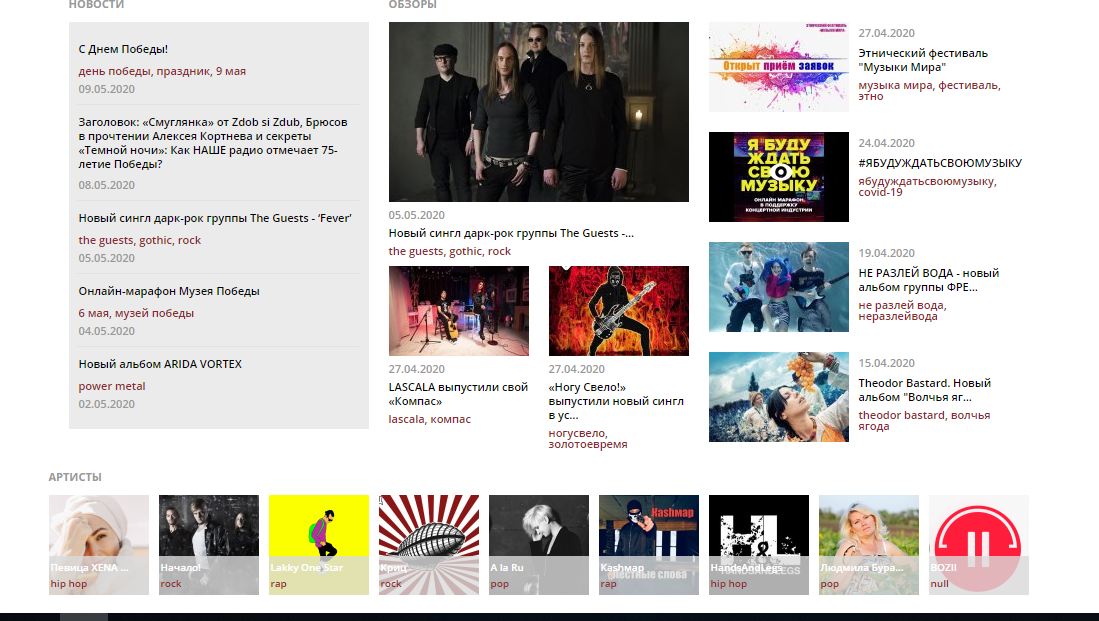


Рисунок 1.1.1– Внешний вид Onmusicstage

**Onmusicstage** – предоставляет возможность добавления и прослушивания музыки, а также поиск музыки и пользователей по параметрам. Кроме того, на сайте могут регистрироватся представители клубов и оставлять афиши.

**Плюсы:**

* Добавление музыки в плейлист;
* Поиск по жанрам;
* Просмотр афиш.

**Минусы:**

* Реклама;
* Невозможно обрабатывать музыку;
* Невозможно скачать музыку.

В целом, я считаю, что это неплохой сайт для обмена и постинга музыки, однако отсутствие обработки музыки и монтажа делает сайт не конкурентно способным. Кроме того, в последнее время на сайте перестала работать функция регистрации.

**1.1.2 Bear Audio Tool** – онлайн аудио редактор.

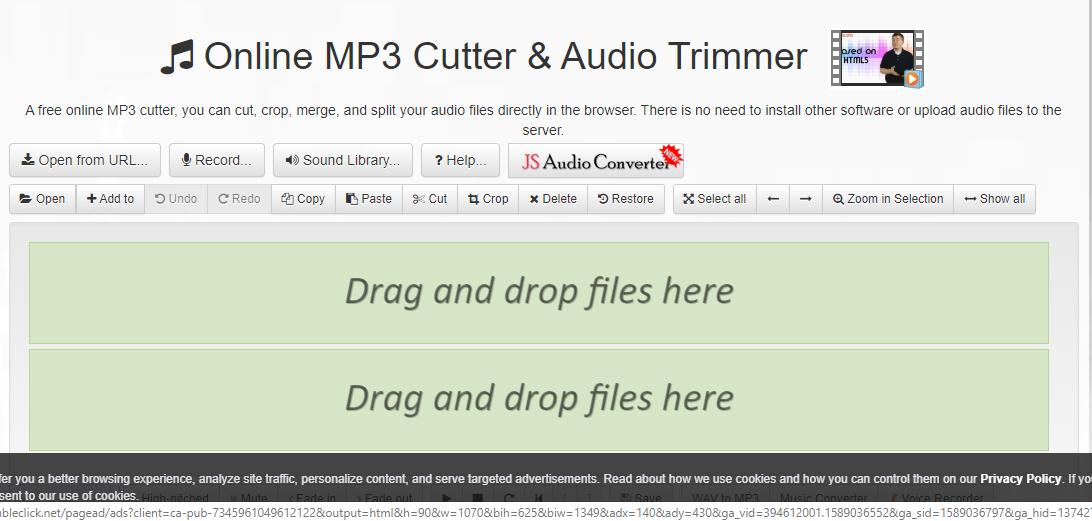


Рисунок 1.1.2 – Внешний вид Bear Audio Tool

**Плюсы:**

* Доступность.
* Возможность монтажа музыки.
* Возможность обработки аудио.

**Минусы:**

* Нет функционала соц. сети.
* Отсутствие тонкой настройки эквалайзера.
* Нельзя сохранить файлы на стороне сервера.

Хороши аудио редактор, с полным набором необходимых функций. Однако может считаться лишь частичным аналогом нашего приложения.

# Постановка Задачи

## Цель

Целью данного курсового проекта является разработка веб-приложения под названием «Meloman» с проработанной базой данных.

В процессе реализации программного средства планируется разработать веб-сайт с большим количеством разнообразных веб-страниц, связанных с музыкой. Особое внимание будет уделено разработке базы данных для приложения: будет произведена нормализация отношений.

## Задачи

В результате анализа аналогов принято решение особое внимание уделить внимание функционалу программного средства, сделать его максимально широким.

Назначение приложения заключается в возможности использования его как

социальной сети, информационного веб-сайта о музыке и как сайта для обработки музыкальных файлов. В связи со всем этим пользователю будут доступны следующие функции:

* просмотр списка звукозаписей с возможностью фильтрации по жанрам, поиска по названию записи;
* просмотр детальной информации о музыкальных группах: название, страна к которой относится группа, описание, дата создания, жанр в котором выступает группа, описание, список участников группы, список альбомов- выпущенных группой, список синглов;
* просмотр детальной информации о музыкальных альбомах: название, описание, дата выпуска, жанр, языки исполнения, лейбл звукозаписи;
* просмотр детальной информации о выпущенных отдельно мелодиях: название, описание, дата выпуска, язык исполнения, просмотры, описание, лейбл звукозаписи;
* просмотр списка музыкальных исполнителей и участников музыкальных групп;
* просмотр детальной информации об исполнителях: имя, фамилия, прозвище, дата рождения, дата смерти, национальность, профессии, языки исполнения, списки групп, альбомов и синглов с его участием;
* просмотр списка пользователей с возможностью поиска по никнейму;
* возможность создавать посты, об определенных плейлистах с уточнением следующей информации: название поста, описание поста, плейлист с возможностью прослушивания треков;
* возможность оставлять комментарии и лайки к постам;
* добавление плейлиста в список избранных;
* создание плейлиста;
* просмотр информации о плейлисте, включающий список треков в плейлисте;
* добавление новых треков;
* просмотр списков добавленных треков и плейлистов;
* возможность добавления новых треков в плейлист;
* возможность поиска групп и исполнителей;
* прослушивание музыки;
* визуализация музыки;
* возможность поиска плейлистов по названию
* регистрация/авторизация профиля;
* возможность редактирования профиля;
* возможность комментировать посты;
* регистрация/авторизация профиля;
* возможность редактирования профиля;
* возможность скачивания музыки;
* просмотр чужих профилей;
* обработка музыки.

# 3 Разработка программного средства

## 3.1 Схемы алгоритмов решения задачи



Рисунок 3.23 – Схема алгоритма получения списка фильмов, отфильтрованных по жанру



Рисунок 3.24 – Схема алгоритма удаления фильма из списка просмотренных



Рисунок 3.25 – Схема алгоритма представления списка всех постов в качестве маркера на карте



Рисунок 3.26 – Схема алгоритма регистрации пользователя

## 3.2 Описание базы данных приложения

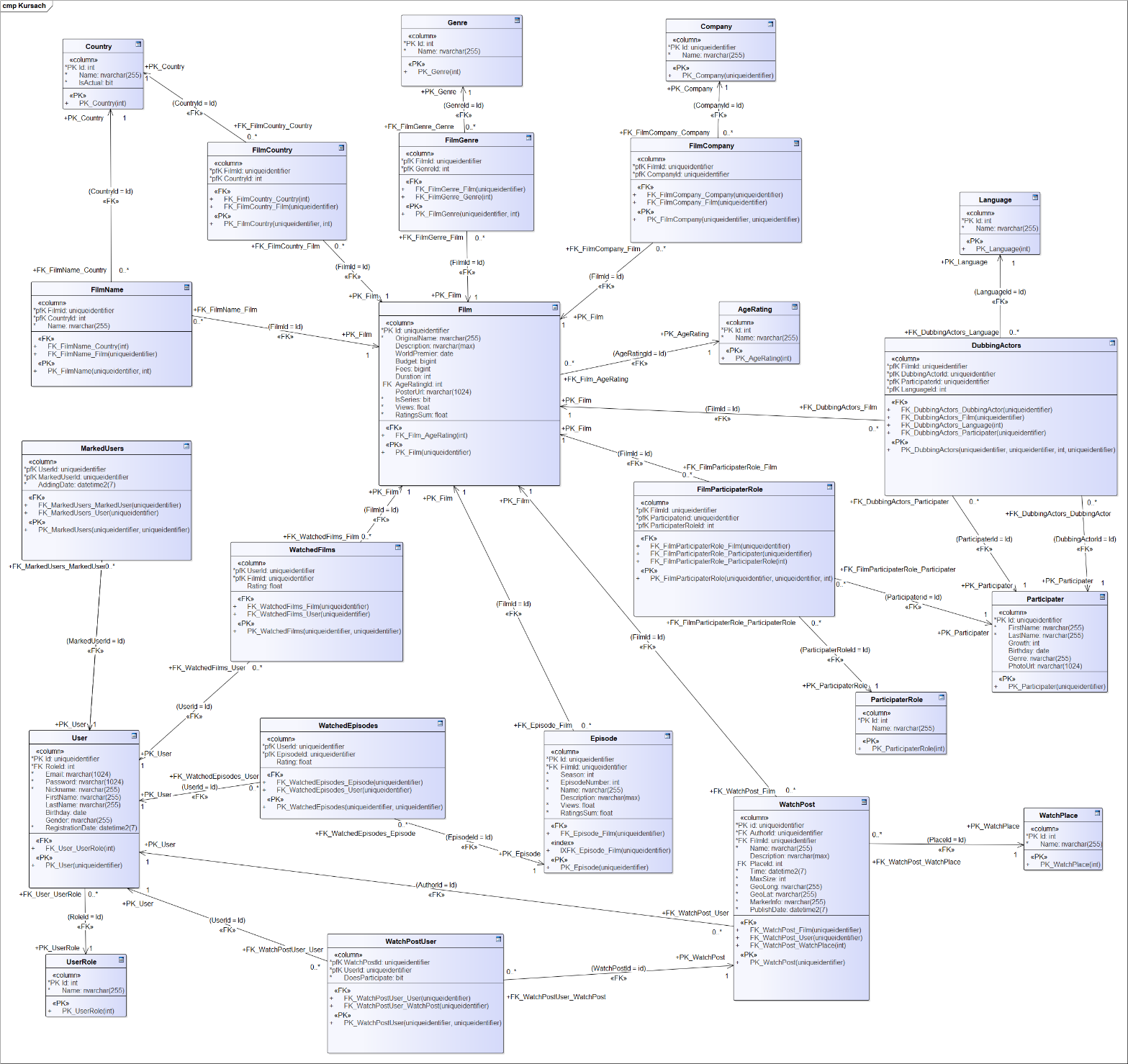


Рисунок 3.27 – Схема базы данных

Составленная база данных состоит из 23 таблиц. Далее мы подробнее рассмотрим каждую из них.

1. *Genre*

Используется для хранения списка существующих жанров, которые могут быть отнесены к фильму.

Связи:

* 1-to-many с таблицей *FilmGenre.*

1. *FilmGenre*

Используется для связи жанров из таблицы *Genre* и фильмов из таблицы *Film*.

Связи:

* many-to-1 с таблицей *Film;*
* many-to-1 с таблицей *Genre.*

1. *Film*

Используется для хранения основной информации о фильме. Является самой важной таблицей, так как имеет множество связей с другими таблицами.

Связи:

* many-to-1 с таблицей *AgeRating;*
* 1-to-many с таблицей *FilmGenre;*
* 1-to-many с таблицей *FilmCountry;*
* 1-to-many с таблицей *FilmName;*
* 1-to-many с таблицей *FilmCompany;*
* 1-to-many с таблицей *FilmParticipaterRole;*
* 1-to-many с таблицей *DubbingActors;*
* 1-to-many с таблицей *WatchedFilms;*
* 1-to-many с таблицей *Episode;*
* 1-to-many с таблицей *WatchPost.*

1. *Country*

Используется для хранения списка существующих и ранее существовавших стран.

Связи:

* 1-to-many с таблицей *FilmCountry;*
* 1-to-many с таблицей *FilmName.*

1. *FilmCountry*

Используется для связи стран, которые участвовали в создании фильма, из таблицы *Country* и самого фильма из таблицы *Film*.

Связи:

* many-to-1 с таблицей *Country;*
* many-to-1 с таблицей *Film.*

1. *FilmName*

Используется для хранения названия фильма из таблицы *Film* в стране из таблицы *Country*.

Связи:

* many-to-1 с таблицей *Country;*
* many-to-1 с таблицей *Film.*

1. *Company*

Используется для хранения списка существующих компаний, которые занимаются созданием фильмов.

Связи:

* 1-to-many с таблицей *FilmCountry;*

1. *FilmCompany*

Используется для хранения связи компании из таблицы *Company* и фильма, в создании которого она принимала участие, из таблицы *Film*.

Связи:

* many-to-1 с таблицей *Company;*
* many-to-1 с таблицей *Film.*

1. *AgeRating*

Используется для хранения списка существующих возрастных рейтингов, один из которых может быть у фильма.

Связи:

* 1-to-many с таблицей *Film.*

1. *Language*

Используется для хранения списка существующих языков, на которых могут говорить актёры дубляжа.

Связи:

* 1-to-many с таблицей *DubbingActors.*

1. *Participater*

Используется для хранения списка существующих языков, на которых могут говорить актёры дубляжа.

Связи:

* 1-to-many с таблицей *DubbingActors (FK ‘DubbingActorId’);*
* 1-to-many с таблицей *DubbingActors (FK ‘ParticipaterId’);*
* 1-to-many с таблицей *FilmParticipaterRole.*

1. *ParticipaterRole*

Используется для хранения списка существующих ролей, которые могут исполнять участники фильма.

Связи:

* 1-to-many с таблицей *FilmParticipaterRole.*

1. *FilmParticipaterRole*

Используется для хранения связи: в каком фильме определённый участник исполнял роль или роли.

Связи:

* many-to-1 с таблицей *Film;*
* many-to-1 с таблицей *Participater;*
* many-to-1 с таблицей *ParticipaterRole.*

1. *DubbingActors*

Используется для хранения связи: в каком фильме определённый участник озвучивал другого участника на определённом языке или языках.

Связи:

* many-to-1 с таблицей *Film;*
* many-to-1 с таблицей *Participater;*
* many-to-1 с таблицей *Participater;*
* many-to-1 с таблицей *Language*.

1. *UserRole*

Используется для хранения списка существующих ролей пользователя, одна из которых обязательна имеется у пользователя.

Связи:

* 1-to-many с таблицей *User.*

1. *User*

Используется для хранения списка зарегистрированных пользователей и информации об их аккаунте.

Связи:

* many-to-1 с таблицей *UseRoler;*
* 1-to-many с таблицей *WatchedFilms;*
* 1-to-many с таблицей *WatchedEpisodes;*
* 1-to-many с таблицей *MarkedUsers (FK UserId);*
* 1-to-many с таблицей *MarkedUsers (FK MarkedUserId);*
* 1-to-many с таблицей *WatchPost;*
* 1-to-many с таблицей *WatchPostUser.*

1. *MarkedUsers*

Используется для хранения пользователей, добавленных в закладки другим пользователем, чтобы легче было найти тех или иных людей.

Связи:

* many-to-1 с таблицей *User (FK UserId);*
* many-to-1 с таблицей *User (FK MarkedUserId).*

1. *WatchedFilms*

Используется для хранения просмотренных пользователем фильмов и оценок, которые он им поставил.

Связи:

* many-to-1 с таблицей *Film;*
* many-to-1 с таблицей *User.*

1. *Episode*

Используется для хранения списка существующих эпизодов сериалов и информации о них.

Связи:

* many-to-1 с таблицей *Film;*
* 1-to-many с таблицей *WatchedEpisodes;*

1. *WatchedFilms*

Используется для хранения просмотренных пользователем эпизодов сериалов и оценок, которые он им поставил.

Связи:

* many-to-1 с таблицей *Episode;*
* many-to-1 с таблицей *User.*

1. *WatchPlace*

Используется для хранения списка существующих типов мест встречи и просмотра фильма

Связи:

* 1-to-many с таблицей *WatchPost.*

1. *WatchPost*

Используется для хранения списка созданных пользователями постов для просмотра того или иного фильма в определённое время в определённом месте с людьми, которые подавали заявку на участие и которые после были приняты.

Связи:

* many-to-1 с таблицей *WatchPlace;*
* many-to-1 с таблицей *Film;*
* many-to-1 с таблицей *User;*
* 1-to-many с таблицей *WatchPostUser.*

1. *WatchPostUser*

Используется для хранения списка пользователей, которые подали заявку на участие в том или ином мероприятии.

Связи:

* many-to-1 с таблицей *WatchPost;*
* many-to-1 с таблицей *User.*

# 4 Руководство по использованию программного средства

## 4.1 Работа с веб-приложением

После перехода на сайт, отображается главная страница гостя, представленная на рисунке 4.1.1.



Рисунок 4.1.1 – главная страница гостя

В верхней части страницы отображается заголовок рис 4.2.1



Рисунок 4.1.2 – заголовок страницы гостя

При нажатии на логотипа сайта, происходит переход на главную страницу. При нажатии на троеточие, происходит переход на страницу расширенного поиска. Для использования поиска по имени пользователя достаточно, достаточно ввести часть имени и нажать на логотип поиска. При нажатии на ссылку Register происходит переход на страницу регистрации. При нажатии на кнопку Sign In происходит и переход на страницу авторизации.

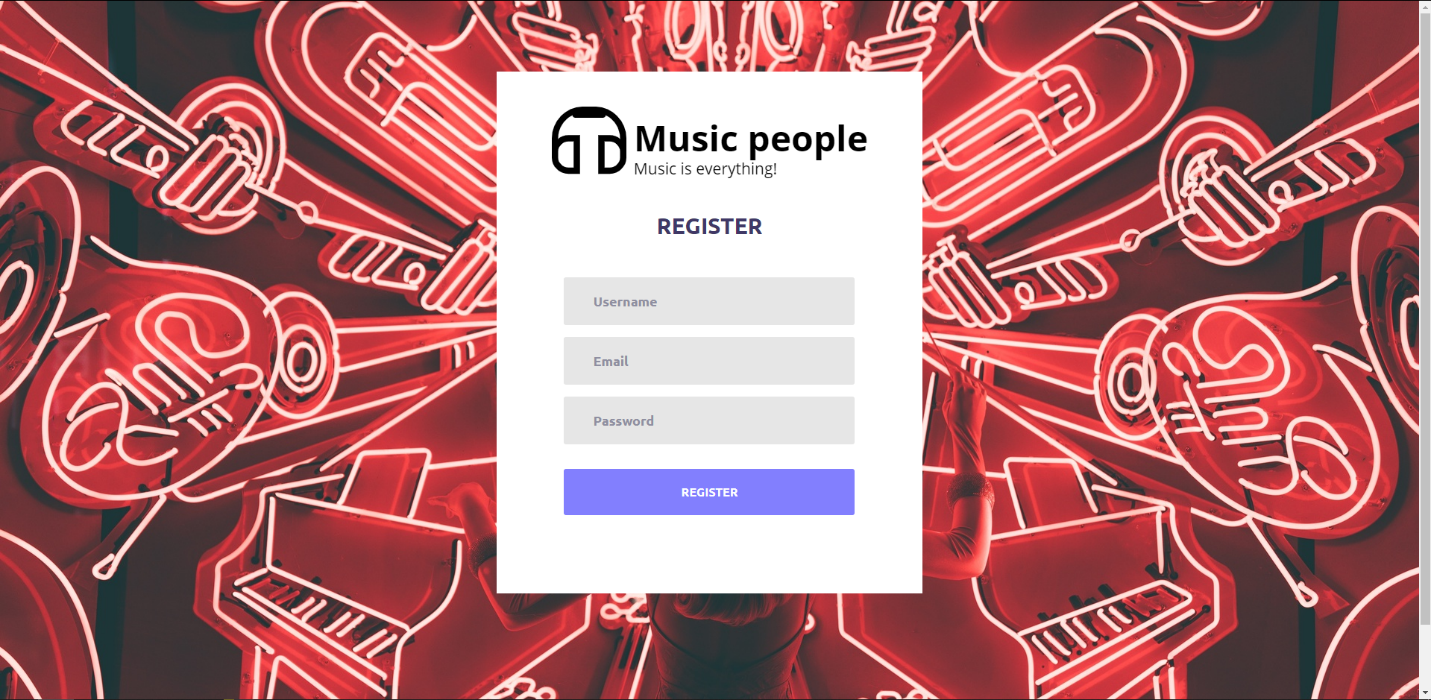


Рисунок 4.1.3 – страница регистрации

После ввода данных, и нажатия на кнопку Register происходит добавление нового пользователя и переход на страницу авторизации.

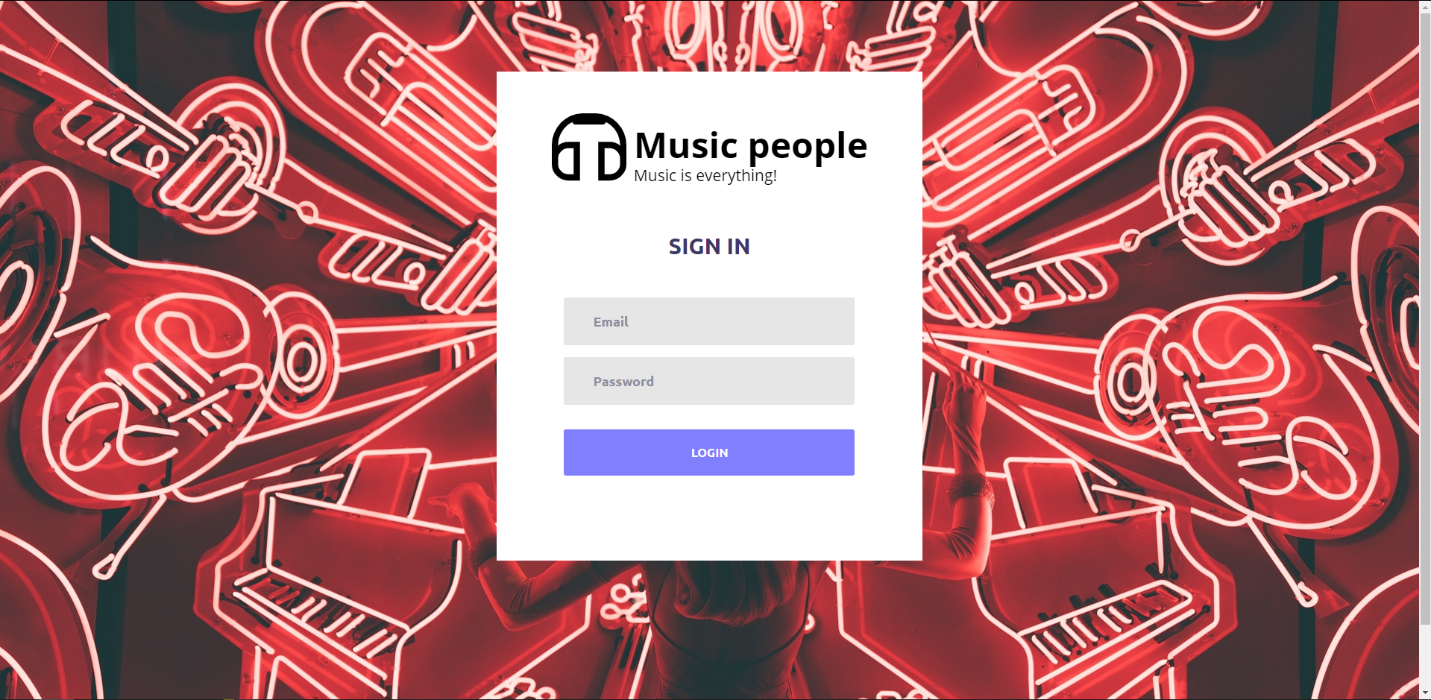


Рисунок 4.1.4 – страница авторизации

После ввода данных, и нажатия на кнопку LOGIN происходит авторизация и переход на страницу пользователя.

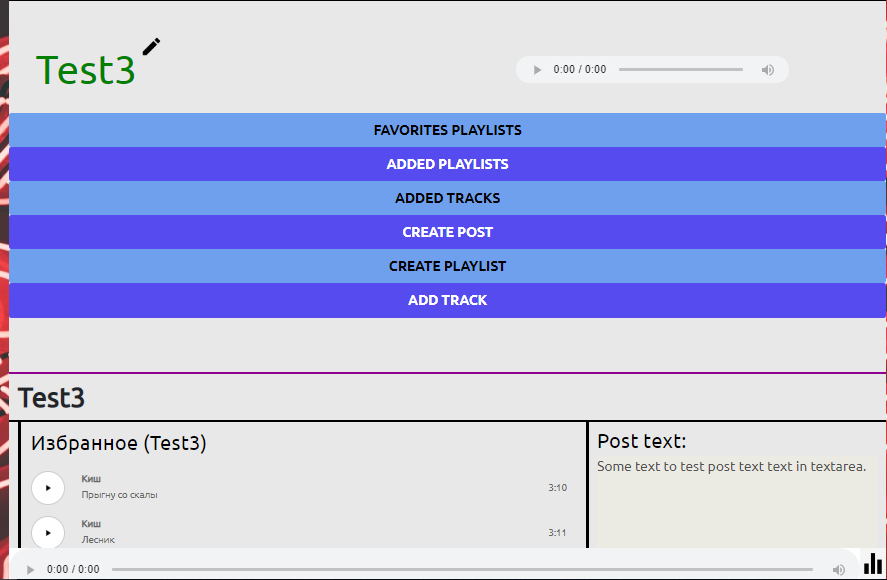


Рисунок 4.1.5 –страница пользователя

В верхней части страницы отображается заголовок рис 4.2.6.

При нажатии на логотип “Карандаша”, происходит переход на страницу редактирования пользователя.

При нажатии на кнопку “FAVOURITES PLAYLISTS”, происходит переход на страницу любимых плейлистов пользователя.

При нажатии на кнопку “ADDED PLAYLISTS”, происходит переход на страницу добавленных плейлистов пользователя.

При нажатии на кнопку “ADDED TRACKS”, происходит переход на страницу добавленных треков пользователя.

При нажатии на кнопку “CREATE POST”, происходит переход на страницу создания поста.

При нажатии на кнопку “CREATE PLAYLIST”, происходит переход на страницу создания плейлиста.

При нажатии на кнопку “ADD TRACK”, происходит переход на страницу добавления трека.



Рисунок 4.1.6 – заголовок страницы

При нажатии на логотипа сайта, происходит переход на главную страницу. При нажатии на троеточие, происходит переход на страницу расширенного поиска. Для использования поиска по имени пользователя достаточно, достаточно ввести часть имени и нажать на логотип поиска. При нажатии на ссылку “My profile” происходит переход на страницу пользователя. При нажатии на кнопку Sign Out происходит выход, и переход на страницу авторизации.

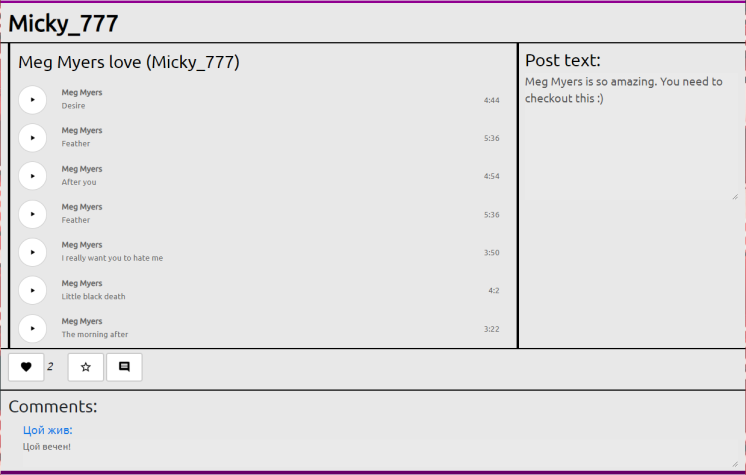


Рисунок 4.1.7 – отображение поста

При нажатии на кнопку с логотипом “Сердца”, к посту добавляется/убирается лайк. При нажатии на кнопку с логотипом “Звезды”, плейлист добавляется к/убирается из избранного. При нажатии на кнопку комментировать, отображается форма добавления комментария. При начатии на кнопку проигрывания, трек запускается в плеере, расположенном в нижней части страницы.



Рисунок 4.1.8 – отображение формы добавления комментария

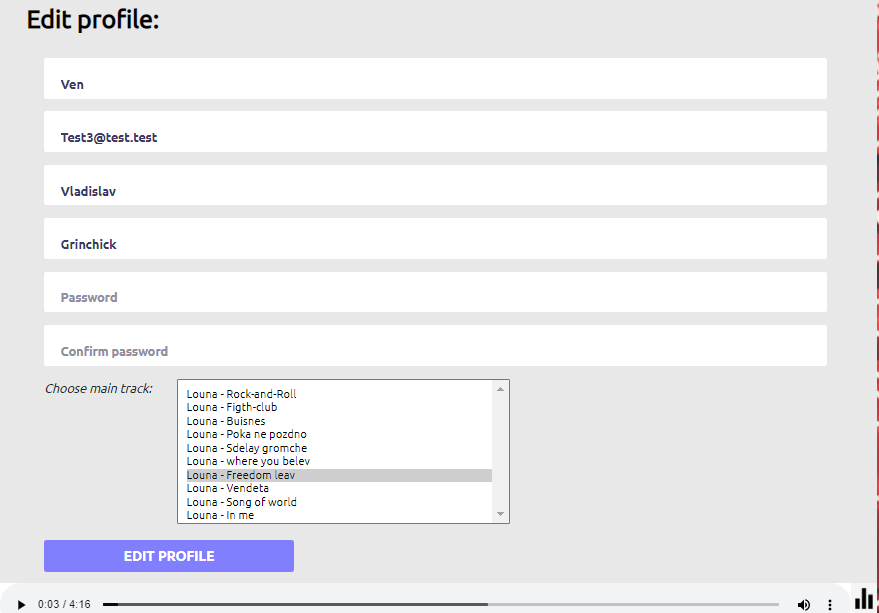


Рисунок 4.1.9 – страница изменения информации о пользователе

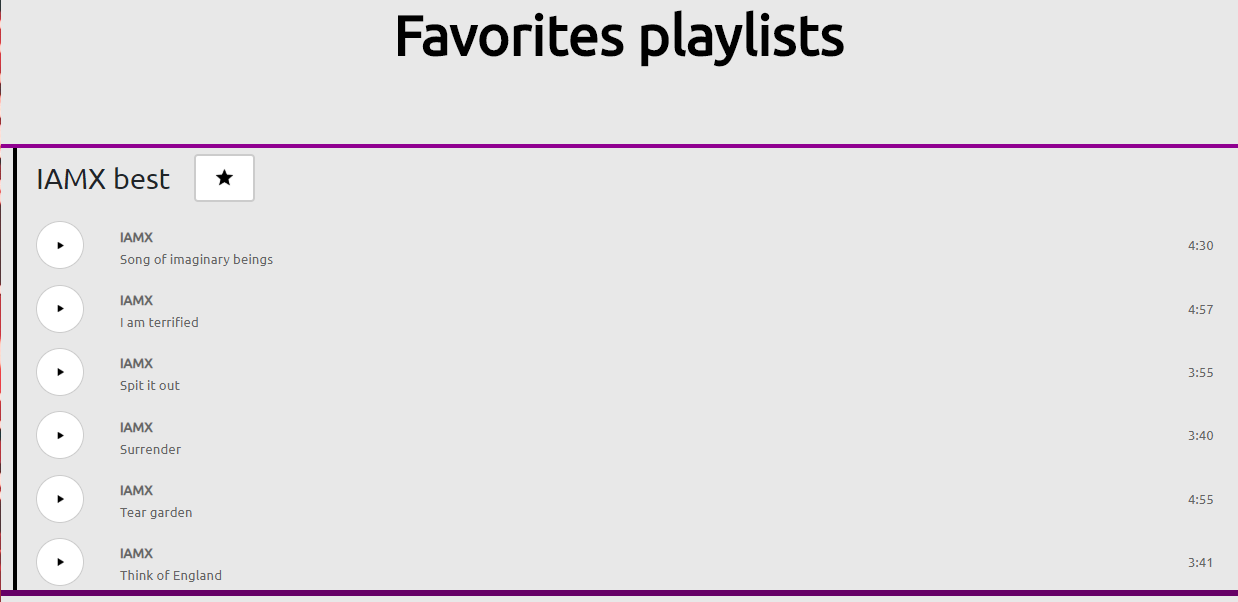


Рисунок 4.1.10 – Страница избранных плейлистов

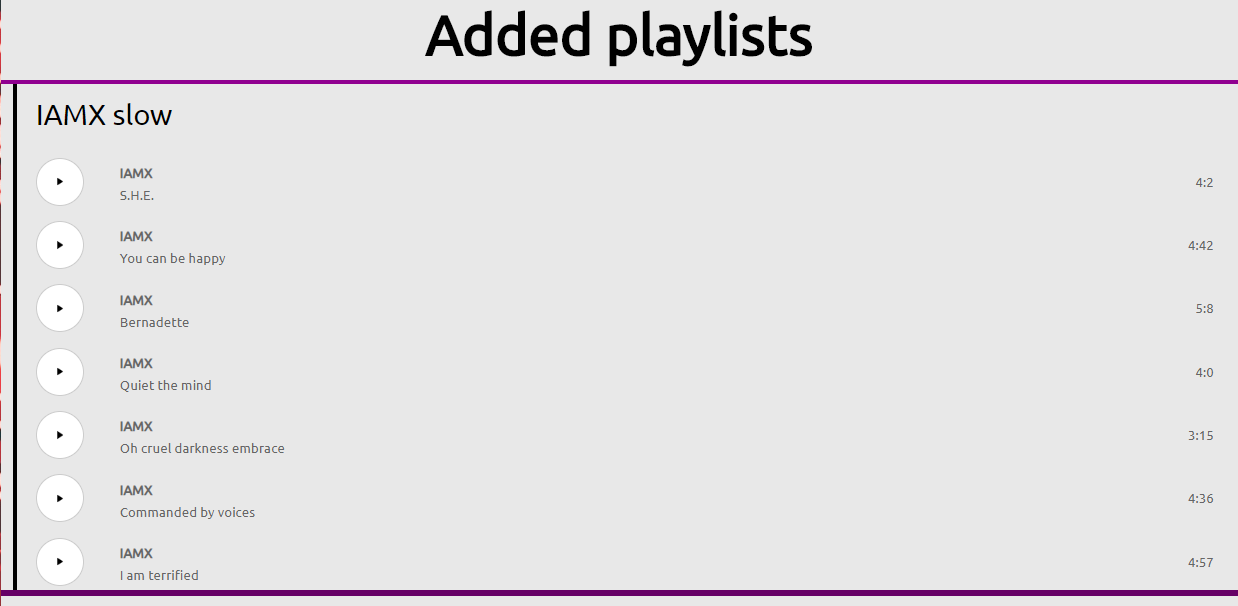


Рисунок 4.1.11 – Страница добавленных плейлистов

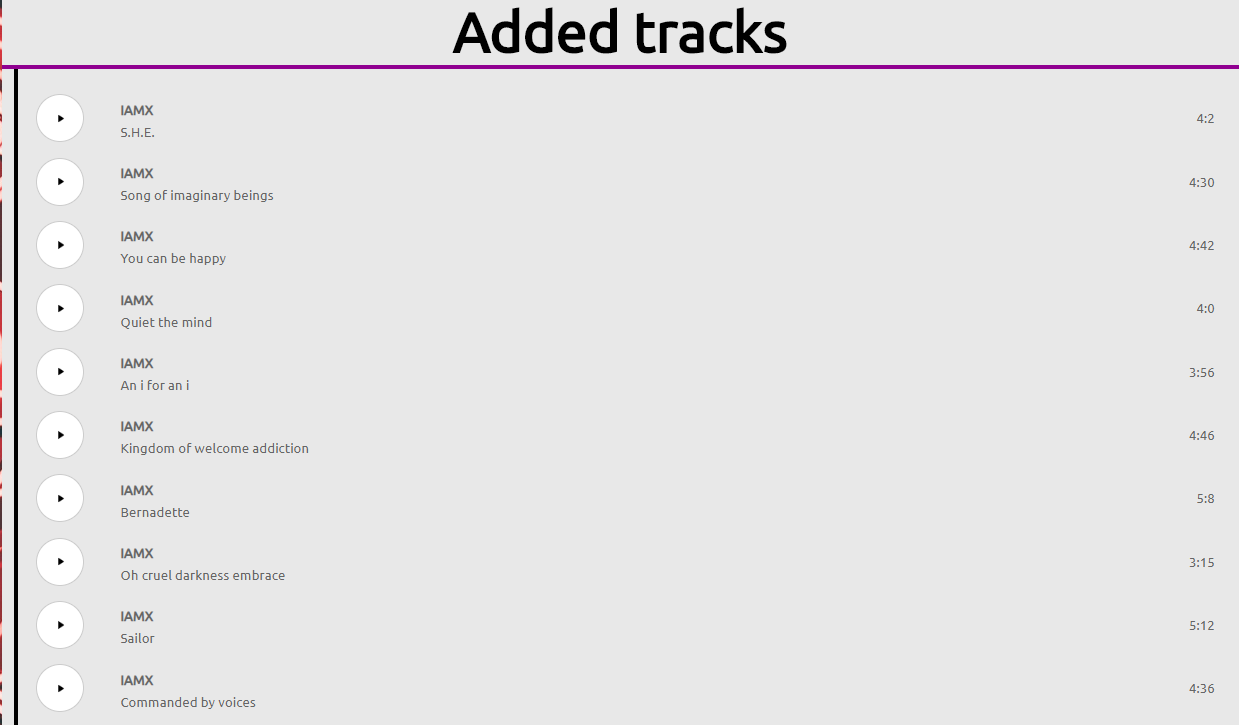


Рисунок 4.1.12 – Страница добавленных треков

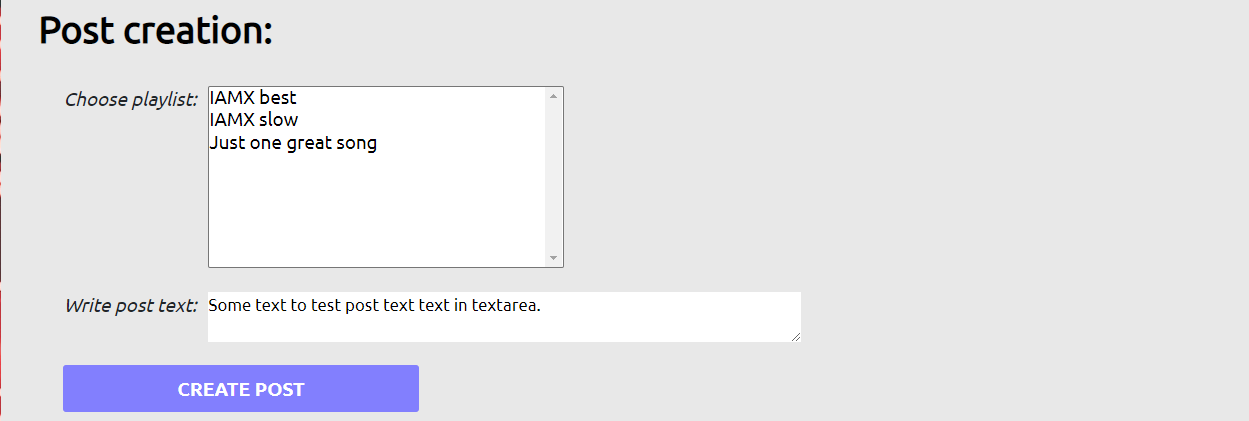


Рисунок 4.1.13 – Страница создания поста

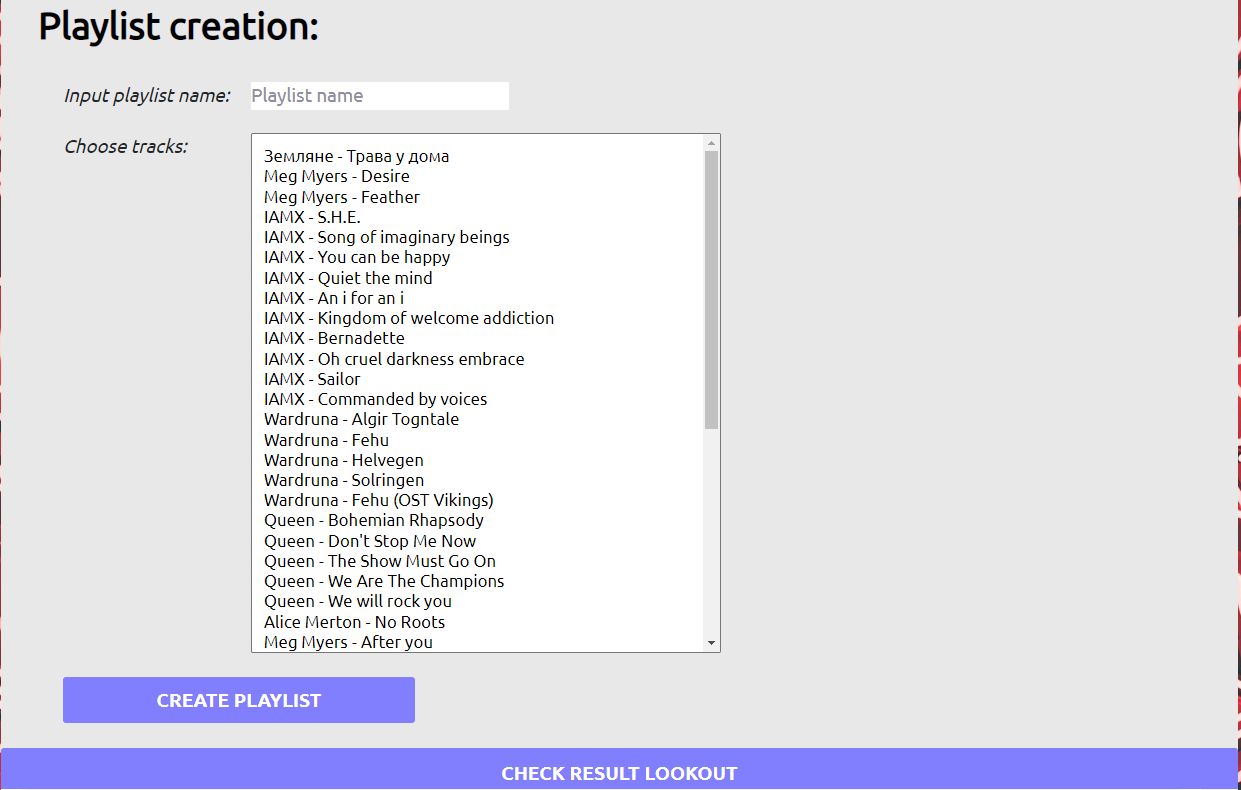


Рисунок 4.1.14 – Страница создания плейлиста

При нажатии на кнопку “CHECK RESUL LOOKOUT” отображается плейлист, без добавления на сервер.

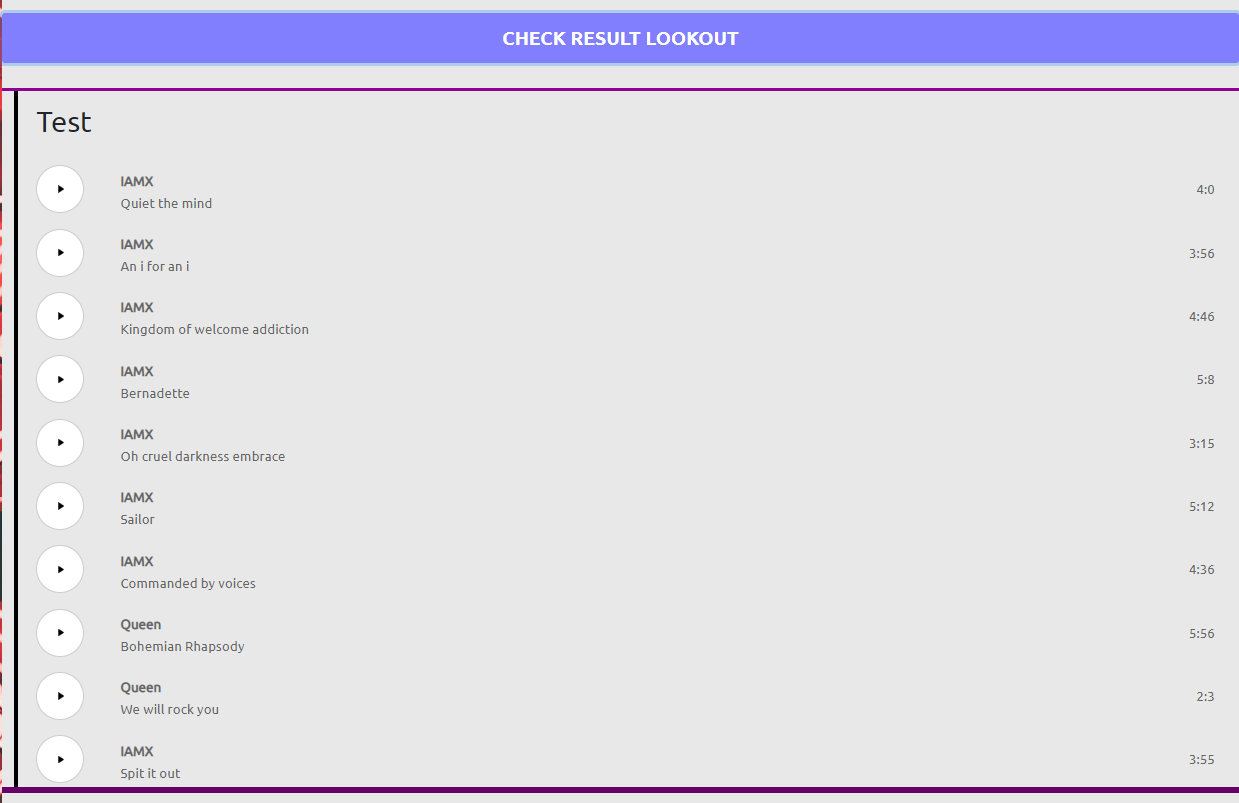


Рисунок 4.1.15 – Превью плейлиста

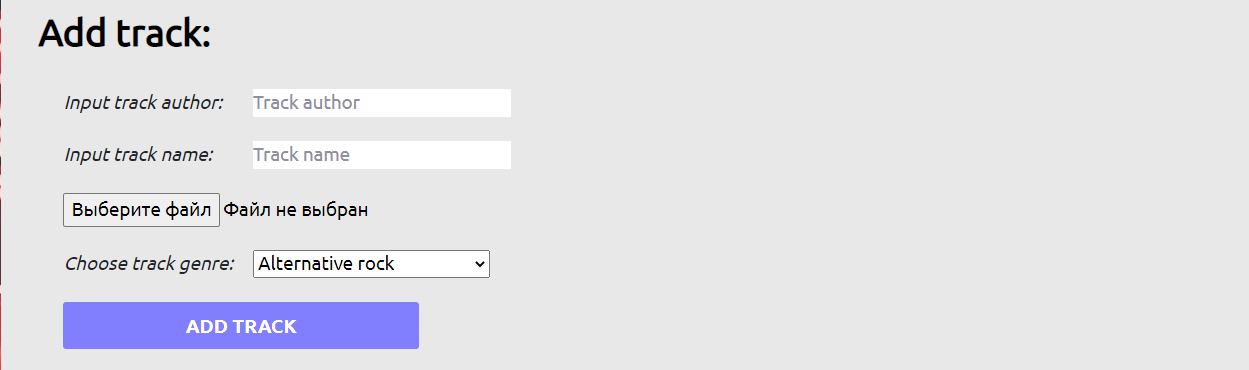


Рисунок 4.1.16 – Страница добавления трека

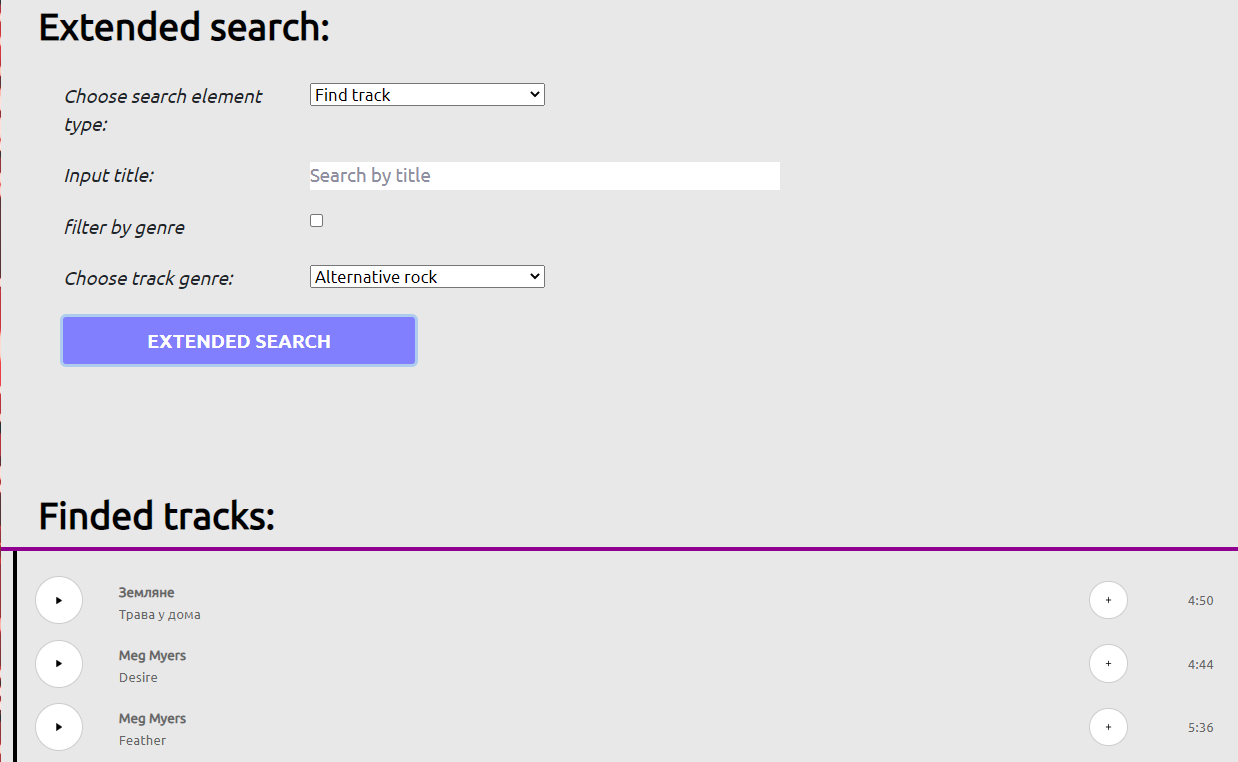


Рисунок 4.1.17 – Страница расширенного поиска

Для выполнения поиска необходимо выбрать тип искомого объекта и установить параметры поиска.

При н нажатии на кнопку с логотипом “+”, отображается боковая панель, добавления трека в плейлист.

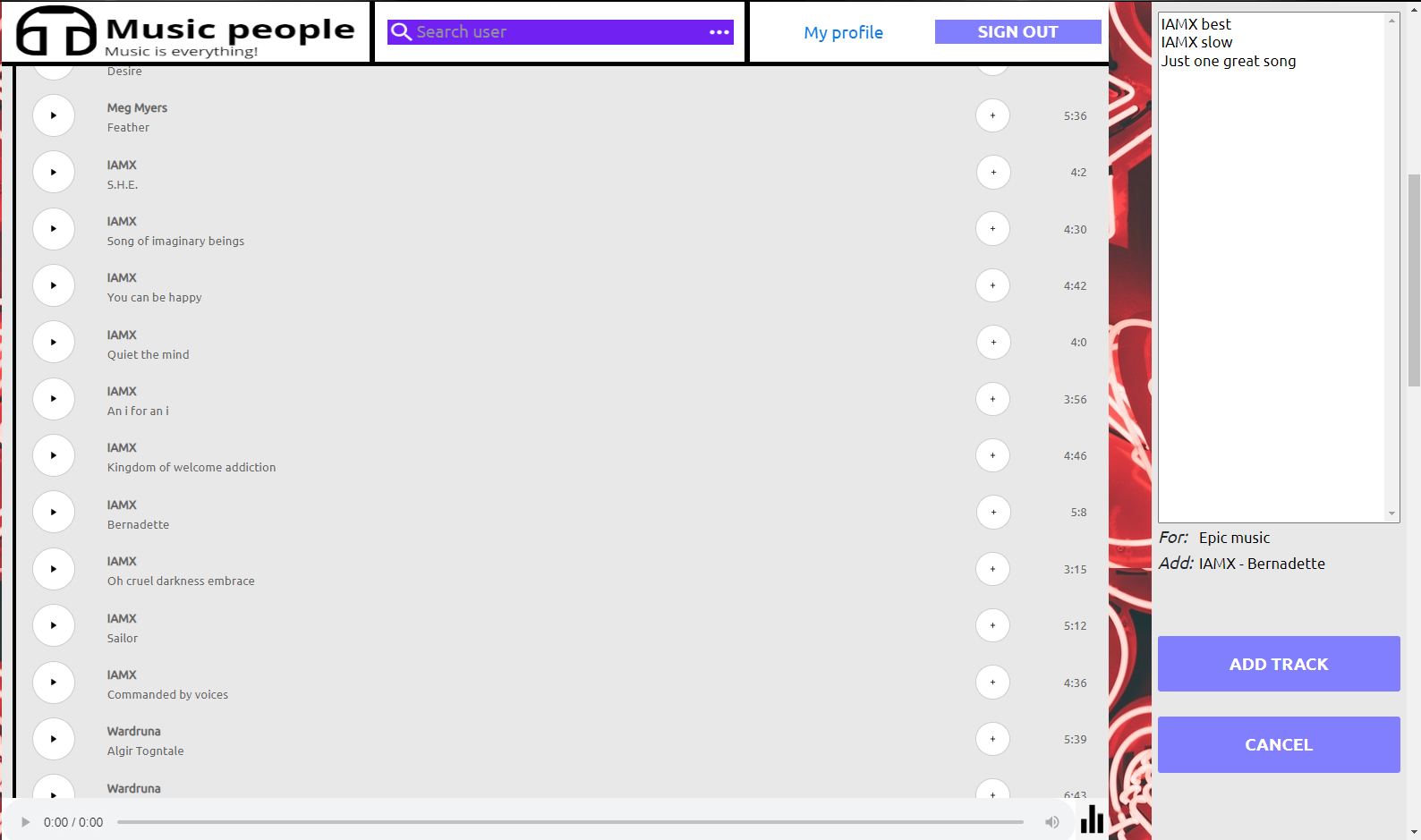


Рисунок 4.1.18 – Страница поиска с боковой панелью

При нажатии на кнопку проигрывания, трек запускается в плеере, расположенном в нижней части страницы.



Рисунок 4.1.19 – отображение плеера

При нажатии на крайнюю правую кнопку в плеере, происходит переход на страницу эквалайзера.

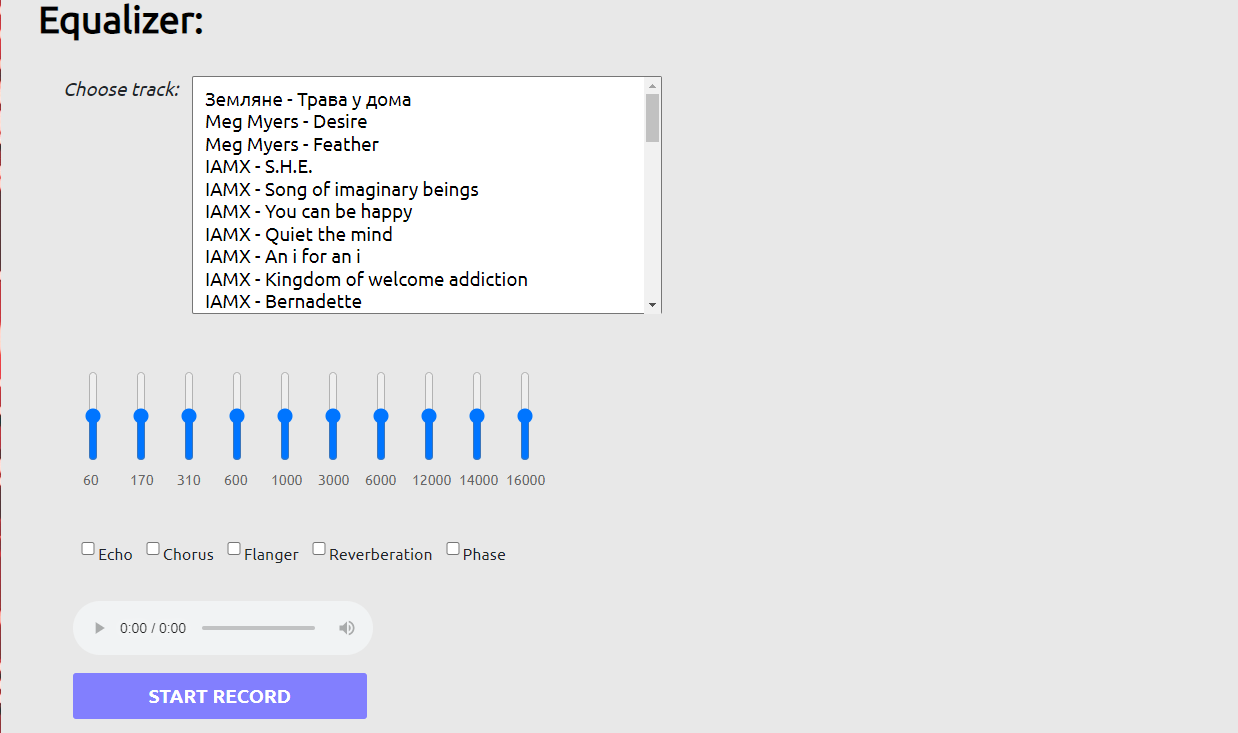


Рисунок 4.1.20 – Страница эквалайзера (часть 1)



Рисунок 4.1.21 – Страница эквалайзера (часть 2)

Заключение

В ходе выполнения курсовой работы было разработано веб-приложение «MovieMan», которое имеет простой интерфейс, приятные навигацию и интерфейс. В процессе разработки использовались языки программирование C# и JavaScript. В качестве СУБД было использована «MS SQL».

В ходе выполнения курсовой работы мною было разработано веб-приложение “MovieMan”. В процессе разработки испльзовались языки программирования С# и JavaScript, а также фреймворк Entity Framewrok Core. В качестве СУБД использовались «MS SQL».

При тестировании программы не было выявлено случаев некорректной работы программы и появления ошибок.

Написанный код легко модифицируется, а также адаптирован к добавлению нового функционала или данных.

Итогами выполнения данной работы является опыт разработки веб-приложений на языках программирования C# и JavaScript и опыт работы с базами данных.

Список использованной литературы

[1] Д. Рихтер – CLR via C#, 2019г. – 896с.

[2] Дж. Албахари – C# 7.0. Карманный справочник, 2017г. – 224с.

[3] ASP.NET Documentation [Электронный ресурс] – Режим доступа: <https://docs.microsoft.com/en-us/aspnet/core/?view=aspnetcore-3.1> – Дата доступа: 20.03.2020.

[4] React Documentation [Электронный ресурс] – Режим доступа: https://ru.reactjs.org/docs. – Дата доступа: 23.03.2020.

[5] Transact-SQL Documentation [Электронный ресурс] – Режим доступа: https://docs.microsoft.com/en-us/sql/t-sql/language-reference?view=sql-server-ver15. – Дата доступа: 29.04.2020.

Приложение А

(обязательное)

Исходный код программы

using System.Collections.Generic;

namespace MovieFan.Core.Entities

{

public class Country

{

public int Id { get; set; }

public string Name { get; set; }

public bool IsActual { get; set; }

public ICollection<FilmCountry> FilmCountries { get; set; }

public ICollection<FilmName> FilmNames { get; set; }

}

}

using System;

using System.Collections.Generic;

namespace MovieFan.Core.Entities

{

public class Episode

{

public Guid Id { get; set; }

public int Season { get; set; }

public int EpisodeNumber { get; set; }

public string Name { get; set; }

public string Description { get; set; }

public double Views { get; set; }

public double RatingsSum { get; set; }

public Guid FilmId { get; set; }

public Film Film { get; set; }

public ICollection<WatchedEpisodes> WatchedEpisodes { get; set; }

}

}

using System;

using System.Collections.Generic;

using System.Text;

namespace MovieFan.Core.Entities

{

public class Film

{

public Guid Id { get; set; }

public string OriginalName { get; set; }

public string Description { get; set; }

public DateTime? WorldPremier { get; set; }

public long? Budget { get; set; }

public long? Fees { get; set; }

public int? Duration { get; set; }

public string PosterUrl { get; set; }

public bool IsSeries { get; set; }

public double Views { get; set; }

public double RatingsSum { get; set; }

public ICollection<FilmCountry> FilmCountries { get; set; }

public ICollection<FilmName> FilmNames { get; set; }

public ICollection<FilmGenre> FilmGenres { get; set;}

public ICollection<FilmParticipaterRole> FilmParticipaterRoles { get; set; }

public ICollection<WatchedFilms> WatchedFilms { get; set; }

public string GetGenresToString()

{

if (FilmGenres.Count == 0)

{

return "";

}

StringBuilder stringBuilder = new StringBuilder();

foreach(var item in FilmGenres)

{

var genre = item.Genre;

stringBuilder.Append(genre.Name + ", ");

}

stringBuilder.Remove(stringBuilder.Length - 2, 2);

return stringBuilder.ToString();

}

public string GetWorldPremierString() => WorldPremier.HasValue ? WorldPremier.Value.ToShortDateString() : string.Empty;

public string GetActorRolesToString(Guid actorId)

{

StringBuilder stringBuilder = new StringBuilder();

stringBuilder.Append("");

foreach(var item in FilmParticipaterRoles)

{

if (item.Participater.Id == actorId)

{

stringBuilder.Append(item.ParticipaterRole.Name + ", ");

}

}

if (stringBuilder.Length > 2)

{

stringBuilder.Remove(stringBuilder.Length - 2, 2);

}

return stringBuilder.ToString();

}

}

}

using System;

namespace MovieFan.Core.Entities

{

public class FilmCountry

{

public Guid FilmId { get; set; }

public int CountryId { get; set; }

public Film Film { get; set; }

public Country Country { get; set; }

}

}

using System;

namespace MovieFan.Core.Entities

{

public class FilmGenre

{

public Guid FilmId { get; set; }

public int GenreId { get; set; }

public Film Film { get; set; }

public Genre Genre { get; set; }

}

}

using System;

namespace MovieFan.Core.Entities

{

public class FilmName

{

public Guid FilmId { get; set; }

public int CountryId { get; set; }

public string Name { get; set; }

public Film Film { get; set; }

public Country Country { get; set; }

}

}

using System;

namespace MovieFan.Core.Entities

{

public class FilmParticipaterRole

{

public Guid FilmId { get; set; }

public Guid ParticipaterId { get; set; }

public int ParticipaterRoleId { get; set; }

public Film Film { get; set; }

public Participater Participater { get; set; }

public ParticipaterRole ParticipaterRole { get; set; }

}

}

using System.Collections.Generic;

namespace MovieFan.Core.Entities

{

public class Genre

{

public int Id { get; set; }

public string Name { get; set; }

public ICollection<FilmGenre> FilmGenres { get; set; }

}

}

using System;

using System.Collections.Generic;

using System.Text;

namespace MovieFan.Core.Entities

{

public class Participater

{

public Guid Id { get; set; }

public string FirstName { get; set; }

public string LastName { get; set; }

public int? Growth { get; set; }

public DateTime? Birthday { get; set; }

public string Gender { get; set; }

public string PhotoUrl { get; set; }

public ICollection<FilmParticipaterRole> FilmParticipaterRoles { get; set; }

public string GetBirthdayString() => Birthday.HasValue ? Birthday.Value.ToShortDateString() : string.Empty;

public string GetParticipaterRolesNamesToString()

{

if (FilmParticipaterRoles.Count == 0)

{

return "";

}

StringBuilder stringBuilder = new StringBuilder();

List<ParticipaterRole> addedRoles = new List<ParticipaterRole>();

foreach(var item in FilmParticipaterRoles)

{

bool flag = true;

foreach(var role in addedRoles)

{

if (role.Id == item.ParticipaterRoleId)

{

flag = false;

}

}

if (flag)

{

stringBuilder.Append(item.ParticipaterRole.Name + ", ");

addedRoles.Add(item.ParticipaterRole);

}

}

stringBuilder.Remove(stringBuilder.Length - 2, 2);

return stringBuilder.ToString();

}

}

}

using System.Collections.Generic;

namespace MovieFan.Core.Entities

{

public class ParticipaterRole

{

public int Id { get; set; }

public string Name { get; set; }

public ICollection<FilmParticipaterRole> FilmParticipaterRoles { get; set; }

}

}

using System;

using System.Collections.Generic;

namespace MovieFan.Core.Entities

{

public class User

{

public Guid Id { get; set; }

public string Email { get; set; }

public string Password { get; set; }

public string Nickname { get; set; }

public string FirstName { get; set; }

public string LastName { get; set; }

public DateTime? Birthday { get; set; }

public string Gender { get; set; }

public DateTime RegistrationDate { get; set; }

public int RoleId { get; set; }

public UserRole UserRole { get; set; }

public ICollection<WatchedFilms> WatchedFilms { get; set; }

public ICollection<WatchedEpisodes> WatchedEpisodes { get; set; }

public ICollection<WatchPostUser> WatchPostUsers { get; set; }

public string GetBirthdayString() => Birthday.HasValue ? Birthday.Value.ToShortDateString() : string.Empty;

public string GetRegistrationDateString() => RegistrationDate.ToLongDateString();

}

}

namespace MovieFan.Core.Entities

{

public class UserRole

{

public int Id { get; set; }

public string Name { get; set; }

}

}

using System;

namespace MovieFan.Core.Entities

{

public class WatchedEpisodes

{

public Guid UserId { get; set; }

public Guid EpisodeId { get; set; }

public double? Rating { get; set; }

public User User { get; set; }

public Episode Episode { get; set; }

}

}

using System;

namespace MovieFan.Core.Entities

{

public class WatchedFilms

{

public Guid UserId { get; set; }

public Guid FilmId { get; set; }

public double? Rating { get; set; }

public User User { get; set; }

public Film Film { get; set; }

}

}

namespace MovieFan.Core.Entities

{

public class WatchPlace

{

public int Id { get; set; }

public string Name { get; set; }

}

}

using System;

using System.Collections.Generic;

namespace MovieFan.Core.Entities

{

public class WatchPost

{

public Guid Id { get; set; }

public string Name { get; set; }

public string Description { get; set; }

public DateTime Time { get; set; }

public int MaxSize { get; set; }

public string GeoLong { get; set; }

public string GeoLat { get; set; }

public string MarkerInfo { get; set; }

public DateTime PublishDate { get; set; }

public Guid AuthorId { get; set; }

public User User { get; set; }

public Guid FilmId { get; set; }

public Film Film { get; set; }

public int PlaceId { get; set; }

public WatchPlace WatchPlace { get; set; }

public ICollection<WatchPostUser> WatchPostUsers { get; set; }

}

}

using System;

namespace MovieFan.Core.Entities

{

public class WatchPostQuery

{

public Guid WatchPostId { get; set; }

public Guid UserId { get; set; }

public WatchPost WatchPost { get; set; }

public User User { get; set; }

}

}

using System;

namespace MovieFan.Core.Entities

{

public class WatchPostUser

{

public Guid WatchPostId { get; set; }

public Guid UserId { get; set; }

public bool DoesParticipate { get; set; }

public WatchPost WatchPost { get; set; }

public User User { get; set; }

}

}

using MovieFan.Core.Entities;

namespace MovieFan.Core.Repositories

{

public interface ICountryRepository : IRepository<Country, int>

{

}

}

using MovieFan.Core.Entities;

using System;

using System.Collections.Generic;

namespace MovieFan.Core.Repositories

{

public interface IEpisodeRepository : IRepository<Episode, Guid>

{

IEnumerable<Episode> FindEpisodesByFilm(Guid filmId);

}

}

using MovieFan.Core.Entities;

using System;

using System.Collections.Generic;

namespace MovieFan.Core.Repositories

{

public interface IFilmRepository : IRepository<Film, Guid>

{

IEnumerable<Film> FindFilmsByGenre(int genreId, bool isNone = false);

IEnumerable<Film> FindFilmsByName(string name);

}

}

using MovieFan.Core.Entities;

namespace MovieFan.Core.Repositories

{

public interface IGenreRepository : IRepository<Genre, int>

{

}

}

using MovieFan.Core.Entities;

using System;

using System.Collections.Generic;

namespace MovieFan.Core.Repositories

{

public interface IParticipaterRepository : IRepository<Participater, Guid>

{

IEnumerable<Participater> FindParticipatersByParticipaterRole(int participaterRoleId, bool isNone = false);

IEnumerable<Participater> FindParticipatersByFullName(string fullname);

}

}

using MovieFan.Core.Entities;

namespace MovieFan.Core.Repositories

{

public interface IParticipaterRoleRepository : IRepository<ParticipaterRole, int>

{

}

}

using System.Collections.Generic;

namespace MovieFan.Core.Repositories

{

public interface IRepository<TEntity, TKey>

{

TEntity Find(TKey id);

IEnumerable<TEntity> List();

}

}

using System;

namespace MovieFan.Core.Repositories

{

public interface IUnitOfWork : IDisposable

{

IFilmRepository FilmRepository { get; }

ICountryRepository CountryRepository { get; }

IGenreRepository GenreRepository { get; }

IParticipaterRepository ParticipaterRepository { get; }

IParticipaterRoleRepository ParticipaterRoleRepository { get; }

IEpisodeRepository EpisodeRepository { get; }

IUserRepository UserRepository { get; }

IUserRoleRepository UserRoleRepository { get; }

IWatchPostRepository WatchPostRepository { get; }

IWatchPlaceRepository WatchPlaceRepository { get; }

void Commit();

}

}

using MovieFan.Core.Entities;

using System;

using System.Collections.Generic;

namespace MovieFan.Core.Repositories

{

public interface IUserRepository : IRepository<User, Guid>

{

User FindUserByEmail(string email);

User FindUserByEmailAndPassword(string email, string password);

IEnumerable<User> FindUsersByUserRole(int userRoleId);

IEnumerable<User> FindUsersByNickname(string nickname);

void Add(User user, bool saveChanges = true);

}

}

using MovieFan.Core.Entities;

namespace MovieFan.Core.Repositories

{

public interface IUserRoleRepository : IRepository<UserRole, int>

{

UserRole FindByName(string name);

}

}

using MovieFan.Core.Entities;

namespace MovieFan.Core.Repositories

{

public interface IWatchPlaceRepository : IRepository<WatchPlace, int>

{

}

}

using MovieFan.Core.Entities;

using System;

using System.Collections.Generic;

namespace MovieFan.Core.Repositories

{

public interface IWatchPostRepository : IRepository<WatchPost, Guid>

{

IEnumerable<WatchPost> FindWatchPostsByWatchPlace(int watchPlaceId);

IEnumerable<WatchPost> FindWatchPostsByNickname(string nickname);

IEnumerable<WatchPost> FindWatchPostsByFilmName(string filmName);

void Add(WatchPost watchPost, bool saveChanges = true);

void Remove(Guid watchPostId, bool saveChanges = true);

void WatchPostSendRequest(Guid watchPostId, Guid userId, bool saveChanges = true);

void WatchPostRejectUser(Guid watchPostId, Guid userId, bool saveChanges = true);

void WatchPostAcceptUser(Guid watchPostId, Guid userId, bool saveChanges = true);

}

}

using MovieFan.Core.Entities;

using Microsoft.EntityFrameworkCore;

using Microsoft.EntityFrameworkCore.Metadata.Builders;

namespace MovieFan.Infrastructure.Configurations

{

internal sealed class CountryConfiguration : IEntityTypeConfiguration<Country>

{

public void Configure(EntityTypeBuilder<Country> builder)

{

builder.ToTable(nameof(Country));

builder.HasKey(x => x.Id);

builder.Property(x => x.Name).IsRequired().HasMaxLength(255);

builder.Property(x => x.IsActual).IsRequired().HasDefaultValue(1);

}

}

}

using MovieFan.Core.Entities;

using Microsoft.EntityFrameworkCore;

using Microsoft.EntityFrameworkCore.Metadata.Builders;

namespace MovieFan.Infrastructure.Configurations

{

internal sealed class EpisodeConfiguration : IEntityTypeConfiguration<Episode>

{

public void Configure(EntityTypeBuilder<Episode> builder)

{

builder.ToTable(nameof(Episode));

builder.HasKey(x => x.Id);

builder.Property(x => x.Id).ValueGeneratedNever();

builder.Property(x => x.Season).IsRequired().HasDefaultValue(0);

builder.Property(x => x.EpisodeNumber).IsRequired().HasDefaultValue(0);

builder.Property(x => x.Name).IsRequired().HasMaxLength(255);

builder.Property(x => x.Description);

builder.Property(x => x.Views).IsRequired().HasDefaultValue(0);

builder.Property(x => x.RatingsSum).IsRequired().HasDefaultValue(0);

builder.HasOne(x => x.Film).WithMany().HasForeignKey(x => x.FilmId)

.OnDelete(DeleteBehavior.Cascade);

}

}

}

using MovieFan.Core.Entities;

using Microsoft.EntityFrameworkCore;

using Microsoft.EntityFrameworkCore.Metadata.Builders;

namespace MovieFan.Infrastructure.Configurations

{

internal sealed class FilmConfiguration : IEntityTypeConfiguration<Film>

{

public void Configure(EntityTypeBuilder<Film> builder)

{

builder.ToTable(nameof(Film));

builder.HasKey(x => x.Id);

builder.Property(x => x.Id).ValueGeneratedNever();

builder.Property(x => x.OriginalName).IsRequired().HasMaxLength(255);

builder.Property(x => x.Description);

builder.Property(x => x.WorldPremier);

builder.Property(x => x.Budget);

builder.Property(x => x.Fees);

builder.Property(x => x.Duration);

builder.Property(x => x.PosterUrl).HasMaxLength(1024);

builder.Property(x => x.IsSeries).IsRequired();

builder.Property(x => x.Views).IsRequired().HasDefaultValue(0);

builder.Property(x => x.RatingsSum).IsRequired().HasDefaultValue(0);

}

}

}

using MovieFan.Core.Entities;

using Microsoft.EntityFrameworkCore;

using Microsoft.EntityFrameworkCore.Metadata.Builders;

namespace MovieFan.Infrastructure.Configurations

{

internal sealed class FilmCountryConfiguration : IEntityTypeConfiguration<FilmCountry>

{

public void Configure(EntityTypeBuilder<FilmCountry> builder)

{

builder.ToTable(nameof(FilmCountry));

builder.HasKey(x => new { x.FilmId, x.CountryId });

builder.HasOne(x => x.Film).WithMany(x => x.FilmCountries).HasForeignKey(x => x.FilmId)

.OnDelete(DeleteBehavior.Cascade);

builder.HasOne(x => x.Country).WithMany(x => x.FilmCountries).HasForeignKey(x => x.CountryId)

.OnDelete(DeleteBehavior.Cascade);

}

}

}

using MovieFan.Core.Entities;

using Microsoft.EntityFrameworkCore;

using Microsoft.EntityFrameworkCore.Metadata.Builders;

namespace MovieFan.Infrastructure.Configurations

{

internal sealed class FilmGenreConfiguration : IEntityTypeConfiguration<FilmGenre>

{

public void Configure(EntityTypeBuilder<FilmGenre> builder)

{

builder.ToTable(nameof(FilmGenre));

builder.HasKey(x => new { x.FilmId, x.GenreId });

builder.HasOne(x => x.Film).WithMany(x => x.FilmGenres).HasForeignKey(x => x.FilmId)

.OnDelete(DeleteBehavior.Cascade);

builder.HasOne(x => x.Genre).WithMany(x => x.FilmGenres).HasForeignKey(x => x.GenreId)

.OnDelete(DeleteBehavior.Cascade);

}

}

}

using MovieFan.Core.Entities;

using Microsoft.EntityFrameworkCore;

using Microsoft.EntityFrameworkCore.Metadata.Builders;

namespace MovieFan.Infrastructure.Configurations

{

internal sealed class FilmNameConfiguration : IEntityTypeConfiguration<FilmName>

{

public void Configure(EntityTypeBuilder<FilmName> builder)

{

builder.ToTable(nameof(FilmName));

builder.HasKey(x => new { x.FilmId, x.CountryId });

builder.Property(x => x.Name).IsRequired().HasMaxLength(255);

builder.HasOne(x => x.Film).WithMany(x => x.FilmNames).HasForeignKey(x => x.FilmId)

.OnDelete(DeleteBehavior.Cascade);

builder.HasOne(x => x.Country).WithMany(x => x.FilmNames).HasForeignKey(x => x.CountryId)

.OnDelete(DeleteBehavior.Cascade);

}

}

}

using MovieFan.Core.Entities;

using Microsoft.EntityFrameworkCore;

using Microsoft.EntityFrameworkCore.Metadata.Builders;

namespace MovieFan.Infrastructure.Configurations

{

internal sealed class FilmParticipaterRoleConfiguration : IEntityTypeConfiguration<FilmParticipaterRole>

{

public void Configure(EntityTypeBuilder<FilmParticipaterRole> builder)

{

builder.ToTable(nameof(FilmParticipaterRole));

builder.HasKey(x => new { x.FilmId, x.ParticipaterId, x.ParticipaterRoleId });

builder.HasOne(x => x.Film).WithMany(x => x.FilmParticipaterRoles).HasForeignKey(x => x.FilmId)

.OnDelete(DeleteBehavior.Cascade);

builder.HasOne(x => x.Participater).WithMany(x => x.FilmParticipaterRoles).HasForeignKey(x => x.ParticipaterId)

.OnDelete(DeleteBehavior.Cascade);

builder.HasOne(x => x.ParticipaterRole).WithMany(x => x.FilmParticipaterRoles).HasForeignKey(x => x.ParticipaterRoleId)

.OnDelete(DeleteBehavior.Cascade);

}

}

}

using MovieFan.Core.Entities;

using Microsoft.EntityFrameworkCore;

using Microsoft.EntityFrameworkCore.Metadata.Builders;

namespace MovieFan.Infrastructure.Configurations

{

internal sealed class GenreConfiguration : IEntityTypeConfiguration<Genre>

{

public void Configure(EntityTypeBuilder<Genre> builder)

{

builder.ToTable(nameof(Genre));

builder.HasKey(x => x.Id);

builder.Property(x => x.Name).IsRequired().HasMaxLength(255);

}

}

}

using MovieFan.Core.Entities;

using Microsoft.EntityFrameworkCore;

using Microsoft.EntityFrameworkCore.Metadata.Builders;

namespace MovieFan.Infrastructure.Configurations

{

internal sealed class ParticipaterConfiguration : IEntityTypeConfiguration<Participater>

{

public void Configure(EntityTypeBuilder<Participater> builder)

{

builder.ToTable(nameof(Participater));

builder.HasKey(x => x.Id);

builder.Property(x => x.Id).ValueGeneratedNever();

builder.Property(x => x.FirstName).IsRequired().HasMaxLength(255);

builder.Property(x => x.LastName).IsRequired().HasMaxLength(255);

builder.Property(x => x.Growth);

builder.Property(x => x.Birthday);

builder.Property(x => x.Gender).HasMaxLength(255);

builder.Property(x => x.PhotoUrl).HasMaxLength(1024);

}

}

}

using MovieFan.Core.Entities;

using Microsoft.EntityFrameworkCore;

using Microsoft.EntityFrameworkCore.Metadata.Builders;

namespace MovieFan.Infrastructure.Configurations

{

internal sealed class ParticipaterRoleConfiguration : IEntityTypeConfiguration<ParticipaterRole>

{

public void Configure(EntityTypeBuilder<ParticipaterRole> builder)

{

builder.ToTable(nameof(ParticipaterRole));

builder.HasKey(x => x.Id);

builder.Property(x => x.Id);

builder.Property(x => x.Name).IsRequired().HasMaxLength(255);

}

}

}

using MovieFan.Core.Entities;

using Microsoft.EntityFrameworkCore;

using Microsoft.EntityFrameworkCore.Metadata.Builders;

using System;

namespace MovieFan.Infrastructure.Configurations

{

internal sealed class UserConfiguration : IEntityTypeConfiguration<User>

{

public void Configure(EntityTypeBuilder<User> builder)

{

builder.ToTable(nameof(User));

builder.HasKey(x => x.Id);

builder.Property(x => x.Id).ValueGeneratedNever();

builder.Property(x => x.Email).IsRequired().HasMaxLength(1024);

builder.Property(x => x.Password).IsRequired().HasMaxLength(1024);

builder.Property(x => x.Nickname).IsRequired().HasMaxLength(255);

builder.Property(x => x.FirstName).HasMaxLength(255);

builder.Property(x => x.LastName).HasMaxLength(255);

builder.Property(x => x.Birthday);

builder.Property(x => x.Gender).HasMaxLength(255);

builder.Property(x => x.RegistrationDate).IsRequired().HasDefaultValue(DateTime.Now);

builder.HasOne(x => x.UserRole).WithMany().HasForeignKey(x => x.RoleId)

.OnDelete(DeleteBehavior.Cascade);

}

}

}

using MovieFan.Core.Entities;

using Microsoft.EntityFrameworkCore;

using Microsoft.EntityFrameworkCore.Metadata.Builders;

namespace MovieFan.Infrastructure.Configurations

{

internal sealed class UserRoleConfiguration : IEntityTypeConfiguration<UserRole>

{

public void Configure(EntityTypeBuilder<UserRole> builder)

{

builder.ToTable(nameof(UserRole));

builder.HasKey(x => x.Id);

builder.Property(x => x.Id);

builder.Property(x => x.Name).IsRequired().HasMaxLength(255);

}

}

}

using MovieFan.Core.Entities;

using Microsoft.EntityFrameworkCore;

using Microsoft.EntityFrameworkCore.Metadata.Builders;

namespace MovieFan.Infrastructure.Configurations

{

internal sealed class WatchedEpisodesConfiguration : IEntityTypeConfiguration<WatchedEpisodes>

{

public void Configure(EntityTypeBuilder<WatchedEpisodes> builder)

{

builder.ToTable(nameof(WatchedEpisodes));

builder.HasKey(x => new { x.UserId, x.EpisodeId });

builder.Property(x => x.Rating);

builder.HasOne(x => x.User).WithMany(x => x.WatchedEpisodes).HasForeignKey(x => x.UserId)

.OnDelete(DeleteBehavior.Cascade);

builder.HasOne(x => x.Episode).WithMany(x => x.WatchedEpisodes).HasForeignKey(x => x.EpisodeId)

.OnDelete(DeleteBehavior.Cascade);

}

}

}

using MovieFan.Core.Entities;

using Microsoft.EntityFrameworkCore;

using Microsoft.EntityFrameworkCore.Metadata.Builders;

namespace MovieFan.Infrastructure.Configurations

{

internal sealed class WatchedFilmsConfiguration : IEntityTypeConfiguration<WatchedFilms>

{

public void Configure(EntityTypeBuilder<WatchedFilms> builder)

{

builder.ToTable(nameof(WatchedFilms));

builder.HasKey(x => new { x.UserId, x.FilmId });

builder.Property(x => x.Rating);

builder.HasOne(x => x.User).WithMany(x => x.WatchedFilms).HasForeignKey(x => x.UserId)

.OnDelete(DeleteBehavior.Cascade);

builder.HasOne(x => x.Film).WithMany(x => x.WatchedFilms).HasForeignKey(x => x.FilmId)

.OnDelete(DeleteBehavior.Cascade);

}

}

}

using MovieFan.Core.Entities;

using Microsoft.EntityFrameworkCore;

using Microsoft.EntityFrameworkCore.Metadata.Builders;

namespace MovieFan.Infrastructure.Configurations

{

internal sealed class WatchPlaceConfiguration : IEntityTypeConfiguration<WatchPlace>

{

public void Configure(EntityTypeBuilder<WatchPlace> builder)

{

builder.ToTable(nameof(WatchPlace));

builder.HasKey(x => x.Id);

builder.Property(x => x.Name).IsRequired().HasMaxLength(255);

}

}

}

using MovieFan.Core.Entities;

using Microsoft.EntityFrameworkCore;

using Microsoft.EntityFrameworkCore.Metadata.Builders;

using System;

namespace MovieFan.Infrastructure.Configurations

{

internal sealed class WatchPostConfiguration : IEntityTypeConfiguration<WatchPost>

{

public void Configure(EntityTypeBuilder<WatchPost> builder)

{

builder.ToTable(nameof(WatchPost));

builder.HasKey(x => x.Id);

builder.Property(x => x.Id).ValueGeneratedNever();

builder.Property(x => x.Name).IsRequired().HasMaxLength(255);

builder.Property(x => x.Description);

builder.Property(x => x.Time).IsRequired();

builder.Property(x => x.MaxSize).IsRequired();

builder.Property(x => x.GeoLong).IsRequired().HasMaxLength(255);

builder.Property(x => x.GeoLat).IsRequired().HasMaxLength(255);

builder.Property(x => x.MarkerInfo).HasMaxLength(255);

builder.Property(x => x.PublishDate).IsRequired().HasDefaultValue(DateTime.Now);

builder.HasOne(x => x.User).WithMany().HasForeignKey(x => x.AuthorId)

.OnDelete(DeleteBehavior.Cascade);

builder.HasOne(x => x.Film).WithMany().HasForeignKey(x => x.FilmId)

.OnDelete(DeleteBehavior.Cascade);

builder.HasOne(x => x.WatchPlace).WithMany().HasForeignKey(x => x.PlaceId)

.OnDelete(DeleteBehavior.Cascade);

}

}

}

using MovieFan.Core.Entities;

using Microsoft.EntityFrameworkCore;

using Microsoft.EntityFrameworkCore.Metadata.Builders;

namespace MovieFan.Infrastructure.Configurations

{

internal sealed class WatchPostUserConfiguration : IEntityTypeConfiguration<WatchPostUser>

{

public void Configure(EntityTypeBuilder<WatchPostUser> builder)

{

builder.ToTable(nameof(WatchPostUser));

builder.HasKey(x => new { x.WatchPostId, x.UserId });

builder.Property(x => x.DoesParticipate).IsRequired().HasDefaultValue(0);

builder.HasOne(x => x.WatchPost).WithMany(x => x.WatchPostUsers).HasForeignKey(x => x.WatchPostId)

.OnDelete(DeleteBehavior.Cascade);

builder.HasOne(x => x.User).WithMany(x => x.WatchPostUsers).HasForeignKey(x => x.UserId)

.OnDelete(DeleteBehavior.NoAction);

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using MovieFan.Core.Entities;

using MovieFan.Core.Repositories;

using Microsoft.EntityFrameworkCore;

namespace MovieFan.Infrastructure.Repositories

{

internal sealed class CountryRepository : ICountryRepository

{

private readonly MovieFanContext \_context;

public CountryRepository(MovieFanContext context)

{

\_context = context;

}

public IEnumerable<Country> List()

{

return \_context.Countries.Include(x => x.FilmCountries).ThenInclude(x => x.Film)

.Include(x => x.FilmNames).ThenInclude(x => x.Film)

.OrderBy(x => x.Name);

}

public Country Find(int id)

{

return \_context.Countries.Include(x => x.FilmCountries).ThenInclude(x => x.Film)

.Include(x => x.FilmNames).ThenInclude(x => x.Film)

.SingleOrDefault(x => x.Id == id);

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using MovieFan.Core.Entities;

using MovieFan.Core.Repositories;

using Microsoft.EntityFrameworkCore;

namespace MovieFan.Infrastructure.Repositories

{

internal sealed class EpisodeRepository : IEpisodeRepository

{

private readonly MovieFanContext \_context;

public EpisodeRepository(MovieFanContext context)

{

\_context = context;

}

public IEnumerable<Episode> List()

{

return \_context.Episodes.Include(x => x.Film).OrderBy(x => x.Season).ThenBy(x => x.EpisodeNumber);

}

public Episode Find(Guid id)

{

return \_context.Episodes.Include(x => x.Film)

.SingleOrDefault(x => x.Id == id);

}

public IEnumerable<Episode> FindEpisodesByFilm(Guid filmId)

{

return List().Where(x => x.Film.Id == filmId);

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using MovieFan.Core.Entities;

using MovieFan.Core.Repositories;

using Microsoft.EntityFrameworkCore;

namespace MovieFan.Infrastructure.Repositories

{

internal sealed class FilmRepository : IFilmRepository

{

private readonly MovieFanContext \_context;

public FilmRepository(MovieFanContext context)

{

\_context = context;

}

public IEnumerable<Film> List()

{

return \_context.Films.Include(x => x.FilmCountries).ThenInclude(x => x.Country)

.Include(x => x.FilmGenres).ThenInclude(x => x.Genre)

.Include(x => x.FilmNames).ThenInclude(x => x.Country)

.Include(x => x.FilmParticipaterRoles).ThenInclude(x => x.Participater)

.Include(x => x.FilmParticipaterRoles).ThenInclude(x => x.ParticipaterRole)

.Include(x => x.WatchedFilms).ThenInclude(x => x.User)

.OrderBy(x => x.OriginalName);

}

public Film Find(Guid id)

{

return \_context.Films.Include(x => x.FilmCountries).ThenInclude(x => x.Country)

.Include(x => x.FilmNames).ThenInclude(x => x.Country)

.Include(x => x.FilmGenres).ThenInclude(x => x.Genre)

.Include(x => x.FilmParticipaterRoles).ThenInclude(x => x.Participater)

.Include(x => x.FilmParticipaterRoles).ThenInclude(x => x.ParticipaterRole)

.Include(x => x.WatchedFilms).ThenInclude(x => x.User)

.SingleOrDefault(x => x.Id == id);

}

public IEnumerable<Film> FindFilmsByGenre(int genreId, bool isNone = false)

{

if (isNone)

{

return List().Where(x => x.FilmGenres.Count == 0);

}

return List().Where(x => x.FilmGenres.Where(x => x.Genre.Id == genreId).ToList().Count != 0);

}

public IEnumerable<Film> FindFilmsByName(string name)

{

return List().Where(x => x.OriginalName.Contains(name));

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using MovieFan.Core.Entities;

using MovieFan.Core.Repositories;

using Microsoft.EntityFrameworkCore;

namespace MovieFan.Infrastructure.Repositories

{

internal sealed class GenreRepository : IGenreRepository

{

private readonly MovieFanContext \_context;

public GenreRepository(MovieFanContext context)

{

\_context = context;

}

public IEnumerable<Genre> List()

{

return \_context.Genres.Include(x => x.FilmGenres).ThenInclude(x => x.Film)

.OrderBy(x => x.Name);

}

public Genre Find(int id)

{

return \_context.Genres.Include(x => x.FilmGenres).ThenInclude(x => x.Film)

.SingleOrDefault(x => x.Id == id);

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using MovieFan.Core.Entities;

using MovieFan.Core.Repositories;

using Microsoft.EntityFrameworkCore;

namespace MovieFan.Infrastructure.Repositories

{

internal sealed class ParticipaterRepository : IParticipaterRepository

{

private readonly MovieFanContext \_context;

public ParticipaterRepository(MovieFanContext context)

{

\_context = context;

}

public IEnumerable<Participater> List()

{

return \_context.Participaters.Include(x => x.FilmParticipaterRoles).ThenInclude(x => x.ParticipaterRole)

.Include(x => x.FilmParticipaterRoles).ThenInclude(x => x.Film)

.OrderBy(x => x.FirstName).ThenBy(x => x.LastName);

}

public Participater Find(Guid id)

{

return \_context.Participaters.Include(x => x.FilmParticipaterRoles).ThenInclude(x => x.ParticipaterRole)

.Include(x => x.FilmParticipaterRoles).ThenInclude(x => x.Film)

.SingleOrDefault(x => x.Id == id);

}

public IEnumerable<Participater> FindParticipatersByParticipaterRole(int participaterRoleId, bool isNone = false)

{

if (isNone)

{

return List().Where(x => x.FilmParticipaterRoles.Count == 0);

}

return List().Where(x => x.FilmParticipaterRoles.Where(x => x.ParticipaterRole.Id == participaterRoleId).ToList().Count != 0);

}

public IEnumerable<Participater> FindParticipatersByFullName(string fullname)

{

return List().Where(x => (x.FirstName + " " + x.LastName).Contains(fullname));

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using MovieFan.Core.Entities;

using MovieFan.Core.Repositories;

using Microsoft.EntityFrameworkCore;

namespace MovieFan.Infrastructure.Repositories

{

internal sealed class ParticipaterRoleRepository : IParticipaterRoleRepository

{

private readonly MovieFanContext \_context;

public ParticipaterRoleRepository(MovieFanContext context)

{

\_context = context;

}

public IEnumerable<ParticipaterRole> List()

{

return \_context.ParticipaterRoles.Include(x => x.FilmParticipaterRoles).ThenInclude(x => x.Participater)

.Include(x => x.FilmParticipaterRoles).ThenInclude(x => x.Film)

.OrderBy(x => x.Name);

}

public ParticipaterRole Find(int id)

{

return \_context.ParticipaterRoles.Include(x => x.FilmParticipaterRoles).ThenInclude(x => x.Participater)

.Include(x => x.FilmParticipaterRoles).ThenInclude(x => x.Film)

.SingleOrDefault(x => x.Id == id);

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using MovieFan.Core.Entities;

using MovieFan.Core.Repositories;

using Microsoft.EntityFrameworkCore;

namespace MovieFan.Infrastructure.Repositories

{

internal sealed class UserRepository : IUserRepository

{

private readonly MovieFanContext \_context;

public UserRepository(MovieFanContext context)

{

\_context = context;

}

public IEnumerable<User> List()

{

return \_context.Users.Include(x => x.UserRole)

.Include(x => x.WatchedFilms).ThenInclude(x => x.Film)

.Include(x => x.WatchedEpisodes).ThenInclude(x => x.Episode)

.Include(x => x.WatchPostUsers).ThenInclude(x => x.WatchPost)

.OrderBy(x => x.Nickname);

}

public User Find(Guid id)

{

return \_context.Users.Include(x => x.UserRole)

.Include(x => x.WatchedFilms).ThenInclude(x => x.Film)

.Include(x => x.WatchedEpisodes).ThenInclude(x => x.Episode)

.Include(x => x.WatchPostUsers).ThenInclude(x => x.WatchPost)

.SingleOrDefault(x => x.Id == id);

}

public User FindUserByEmail(string email)

{

if (email == null)

{

throw new ArgumentNullException(nameof(email));

}

return \_context.Users.Include(x => x.UserRole)

.Include(x => x.WatchedFilms).ThenInclude(x => x.Film)

.Include(x => x.WatchedEpisodes).ThenInclude(x => x.Episode)

.Include(x => x.WatchPostUsers).ThenInclude(x => x.WatchPost)

.SingleOrDefault(x => x.Email == email);

}

public User FindUserByEmailAndPassword(string email, string password)

{

if (email == null)

{

throw new ArgumentNullException(nameof(email));

}

if (password == null)

{

throw new ArgumentNullException(nameof(password));

}

return \_context.Users.Include(x => x.UserRole)

.Include(x => x.WatchedFilms).ThenInclude(x => x.Film)

.Include(x => x.WatchedEpisodes).ThenInclude(x => x.Episode)

.Include(x => x.WatchPostUsers).ThenInclude(x => x.WatchPost)

.SingleOrDefault(x => x.Email == email && x.Password == password);

}

public IEnumerable<User> FindUsersByUserRole(int userRoleId)

{

return \_context.Users.Include(x => x.UserRole)

.Include(x => x.WatchedFilms).ThenInclude(x => x.Film)

.Include(x => x.WatchedEpisodes).ThenInclude(x => x.Episode)

.Include(x => x.WatchPostUsers).ThenInclude(x => x.WatchPost)

.Where(x => x.UserRole.Id == userRoleId);

}

public IEnumerable<User> FindUsersByNickname(string nickname)

{

return List().Where(x => x.Nickname.Contains(nickname));

}

public void Add(User user, bool saveChanges = true)

{

if (user == null)

{

throw new ArgumentNullException(nameof(user));

}

if (user.Email == null)

{

throw new ArgumentNullException(nameof(user.Email));

}

if (user.Password == null)

{

throw new ArgumentNullException(nameof(user.Password));

}

if (user.Nickname== null)

{

throw new ArgumentNullException(nameof(user.Nickname));

}

if (user.RegistrationDate == null)

{

throw new ArgumentNullException(nameof(user.RegistrationDate));

}

\_context.Users.Add(user);

if (saveChanges)

{

\_context.SaveChanges();

}

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using MovieFan.Core.Entities;

using MovieFan.Core.Repositories;

using Microsoft.EntityFrameworkCore;

namespace MovieFan.Infrastructure.Repositories

{

internal sealed class UserRoleRepository : IUserRoleRepository

{

private readonly MovieFanContext \_context;

public UserRoleRepository(MovieFanContext context)

{

\_context = context;

}

public IEnumerable<UserRole> List()

{

return \_context.UserRoles.OrderBy(x => x.Name);

}

public UserRole Find(int id)

{

return \_context.UserRoles.SingleOrDefault(x => x.Id == id);

}

public UserRole FindByName(string name)

{

if (name == null)

{

throw new ArgumentNullException(nameof(name));

}

return \_context.UserRoles.SingleOrDefault(x => x.Name == name);

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using MovieFan.Core.Entities;

using MovieFan.Core.Repositories;

using Microsoft.EntityFrameworkCore;

namespace MovieFan.Infrastructure.Repositories

{

internal sealed class WatchPlaceRepository : IWatchPlaceRepository

{

private readonly MovieFanContext \_context;

public WatchPlaceRepository(MovieFanContext context)

{

\_context = context;

}

public IEnumerable<WatchPlace> List()

{

return \_context.WatchPlaces.OrderBy(x => x.Name);

}

public WatchPlace Find(int id)

{

return \_context.WatchPlaces.SingleOrDefault(x => x.Id == id);

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using MovieFan.Core.Entities;

using MovieFan.Core.Repositories;

using Microsoft.EntityFrameworkCore;

namespace MovieFan.Infrastructure.Repositories

{

internal sealed class WatchPostRepository : IWatchPostRepository

{

private readonly MovieFanContext \_context;

public WatchPostRepository(MovieFanContext context)

{

\_context = context;

}

public IEnumerable<WatchPost> List()

{

return \_context.WatchPosts.Include(x => x.Film)

.Include(x => x.User)

.Include(x => x.WatchPlace)

.Include(x => x.WatchPostUsers).ThenInclude(x => x.User)

.OrderByDescending(x => x.Time);

}

public WatchPost Find(Guid id)

{

return \_context.WatchPosts.Include(x => x.Film)

.Include(x => x.User)

.Include(x => x.WatchPlace)

.Include(x => x.WatchPostUsers).ThenInclude(x => x.User)

.SingleOrDefault(x => x.Id == id);

}

public IEnumerable<WatchPost> FindWatchPostsByWatchPlace(int watchPlaceId)

{

return List().Where(x => x.PlaceId == watchPlaceId);

}

public IEnumerable<WatchPost> FindWatchPostsByNickname(string nickname)

{

return List().Where(x => x.User.Nickname.Contains(nickname));

}

public IEnumerable<WatchPost> FindWatchPostsByFilmName(string filmName)

{

return List().Where(x => x.Film.OriginalName.Contains(filmName));

}

public void Add(WatchPost watchPost, bool saveChanges = true)

{

\_context.WatchPosts.Add(watchPost);

if (saveChanges)

{

\_context.SaveChanges();

}

}

public void Remove(Guid watchPostId, bool saveChanges = true)

{

var watchPost = Find(watchPostId);

\_context.WatchPosts.Remove(watchPost);

if (saveChanges)

{

\_context.SaveChanges();

}

}

public void WatchPostSendRequest(Guid watchPostId, Guid userId, bool saveChanges = true)

{

var newWatchPostUser = new WatchPostUser { WatchPostId = watchPostId, UserId = userId, DoesParticipate = false };

var watchPost = Find(watchPostId);

watchPost.WatchPostUsers.Add(newWatchPostUser);

if (saveChanges)

{

\_context.SaveChanges();

}

}

public void WatchPostAcceptUser(Guid watchPostId, Guid userId, bool saveChanges = true)

{

var watchPostUser = Find(watchPostId).WatchPostUsers.Where(x => x.UserId == userId).Single();

watchPostUser.DoesParticipate = true;

if (saveChanges)

{

\_context.SaveChanges();

}

}

public void WatchPostRejectUser(Guid watchPostId, Guid userId, bool saveChanges = true)

{

var watchPost = Find(watchPostId);

var watchPostUser = watchPost.WatchPostUsers.Where(x => x.UserId == userId).Single();

watchPost.WatchPostUsers.Remove(watchPostUser);

if (saveChanges)

{

\_context.SaveChanges();

}

}

}

}

using MovieFan.Core.Entities;

using MovieFan.Infrastructure.Configurations;

using Microsoft.EntityFrameworkCore;

namespace MovieFan.Infrastructure

{

internal class MovieFanContext : DbContext

{

private const string Connection =

"Data Source=(local);Initial Catalog=MovieFan;Integrated Security=True";

public DbSet<Film> Films { get; set; }

public DbSet<Country> Countries { get; set; }

public DbSet<Genre> Genres { get; set; }

public DbSet<Participater> Participaters { get; set; }

public DbSet<ParticipaterRole> ParticipaterRoles { get; set; }

public DbSet<Episode> Episodes { get; set; }

public DbSet<User> Users { get; set; }

public DbSet<UserRole> UserRoles { get; set; }

public DbSet<WatchPost> WatchPosts { get; set; }

public DbSet<WatchPlace> WatchPlaces { get; set; }

protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)

{

optionsBuilder.UseSqlServer(Connection);

}

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

base.OnModelCreating(modelBuilder);

modelBuilder.ApplyConfiguration(new FilmConfiguration());

modelBuilder.ApplyConfiguration(new CountryConfiguration());

modelBuilder.ApplyConfiguration(new FilmCountryConfiguration());

modelBuilder.ApplyConfiguration(new FilmNameConfiguration());

modelBuilder.ApplyConfiguration(new GenreConfiguration());

modelBuilder.ApplyConfiguration(new FilmGenreConfiguration());

modelBuilder.ApplyConfiguration(new ParticipaterConfiguration());

modelBuilder.ApplyConfiguration(new ParticipaterRoleConfiguration());

modelBuilder.ApplyConfiguration(new FilmParticipaterRoleConfiguration());

modelBuilder.ApplyConfiguration(new EpisodeConfiguration());

modelBuilder.ApplyConfiguration(new UserRoleConfiguration());

modelBuilder.ApplyConfiguration(new UserConfiguration());

modelBuilder.ApplyConfiguration(new WatchedFilmsConfiguration());

modelBuilder.ApplyConfiguration(new WatchedEpisodesConfiguration());

modelBuilder.ApplyConfiguration(new WatchPlaceConfiguration());

modelBuilder.ApplyConfiguration(new WatchPostConfiguration());

//modelBuilder.ApplyConfiguration(new WatchPostQueryConfiguration());

modelBuilder.ApplyConfiguration(new WatchPostUserConfiguration());

}

}

}

using System;

using MovieFan.Core.Repositories;

using MovieFan.Infrastructure.Repositories;

namespace MovieFan.Infrastructure

{

public sealed class UnitOfWork : IUnitOfWork

{

private MovieFanContext \_context;

private IFilmRepository \_filmRepository;

private ICountryRepository \_countryRepository;

private IGenreRepository \_genreRepository;

private IParticipaterRepository \_participaterRepository;

private IParticipaterRoleRepository \_participaterRoleRepository;

private IEpisodeRepository \_episodeRepository;

private IUserRoleRepository \_userRoleRepository;

private IUserRepository \_userRepository;

private IWatchPlaceRepository \_watchPlaceRepository;

private IWatchPostRepository \_watchPostRepository;

private MovieFanContext Context => \_context ??= new MovieFanContext();

public IFilmRepository FilmRepository => \_filmRepository ??= new FilmRepository(Context);

public ICountryRepository CountryRepository => \_countryRepository ??= new CountryRepository(Context);

public IGenreRepository GenreRepository => \_genreRepository ??= new GenreRepository(Context);

public IParticipaterRepository ParticipaterRepository => \_participaterRepository ??= new ParticipaterRepository(Context);

public IParticipaterRoleRepository ParticipaterRoleRepository => \_participaterRoleRepository ??= new ParticipaterRoleRepository(Context);

public IEpisodeRepository EpisodeRepository => \_episodeRepository ??= new EpisodeRepository(Context);

public IUserRoleRepository UserRoleRepository => \_userRoleRepository ??= new UserRoleRepository(Context);

public IUserRepository UserRepository => \_userRepository ??= new UserRepository(Context);

public IWatchPlaceRepository WatchPlaceRepository => \_watchPlaceRepository ??= new WatchPlaceRepository(Context);

public IWatchPostRepository WatchPostRepository => \_watchPostRepository ??= new WatchPostRepository(Context);

public void Commit()

{

if (\_isDisposed)

{

throw new ObjectDisposedException("UnitOfWork");

}

Context.SaveChanges();

}

private bool \_isDisposed;

public void Dispose()

{

if (\_context == null)

{

return;

}

if (!\_isDisposed)

{

Context.Dispose();

}

\_isDisposed = true;

}

}

}

using System.Threading.Tasks;

using Microsoft.AspNetCore.Mvc;

using MovieFan.Web.ViewModels;

using MovieFan.Core.Entities;

using Microsoft.AspNetCore.Identity;

using System.Security.Claims;

using System.Collections.Generic;

using Microsoft.AspNetCore.Authentication;

using Microsoft.AspNetCore.Authentication.Cookies;

using MovieFan.Core.Repositories;

using System;

namespace MovieFan.Web.Controllers

{

public class AccountController : Controller

{

private readonly IUnitOfWork \_uow;

public AccountController(IUnitOfWork uow) => \_uow = uow;

[HttpGet]

public IActionResult Login()

{

ViewBag.Username = User.Identity.Name ?? "anonymous";

return View();

}

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Login(LoginViewModel model)

{

if (ModelState.IsValid)

{

User user = \_uow.UserRepository.FindUserByEmailAndPassword(model.Email, model.Password);

if (user != null)

{

await Authenticate(user); // аутентификация

ViewBag.Username = User.Identity.Name ?? "anonymous";

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

}

ViewBag.Username = User.Identity.Name ?? "anonymous";

ModelState.AddModelError("", "Incorrect email and (or) password!");

}

else

{

ViewBag.Username = User.Identity.Name ?? "anonymous";

}

return View(model);

}

[HttpGet]

public IActionResult Register()

{

ViewBag.Username = User.Identity.Name ?? "anonymous";

return View();

}

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Register(RegisterViewModel model)

{

ViewBag.Username = User.Identity.Name ?? "anonymous";

if (ModelState.IsValid)

{

User user = \_uow.UserRepository.FindUserByEmail(model.Email);

if (user == null)

{

// добавляем пользователя в бд

User newUser = new User

{

Id = Guid.NewGuid(),

Email = model.Email,

Password = model.Password,

Nickname = model.Nickname,

RegistrationDate = DateTime.Now

};

var userRole = \_uow.UserRoleRepository.FindByName("user");

if (userRole == null)

{

throw new ArgumentNullException(nameof(userRole));

}

newUser.UserRole = userRole;

\_uow.UserRepository.Add(newUser, false);

\_uow.Commit();

await Authenticate(newUser); // аутентификация

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

}

else

ModelState.AddModelError("Password", "Incorrect email and (or) password!");

}

return View(model);

}

private async Task Authenticate(User user)

{

// создаем один claim

var claims = new List<Claim>

{

new Claim(ClaimsIdentity.DefaultNameClaimType, user.Email),

new Claim(ClaimsIdentity.DefaultRoleClaimType, user.UserRole.Name)

};

// создаем объект ClaimsIdentity

ClaimsIdentity id = new ClaimsIdentity(claims, "ApplicationCookie", ClaimsIdentity.DefaultNameClaimType,

ClaimsIdentity.DefaultRoleClaimType);

// установка аутентификационных куки

await HttpContext.SignInAsync(CookieAuthenticationDefaults.AuthenticationScheme, new ClaimsPrincipal(id));

}

public async Task<IActionResult> Logout()

{

await HttpContext.SignOutAsync(CookieAuthenticationDefaults.AuthenticationScheme);

ViewBag.Username = User.Identity.Name ?? "anonymous";

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

}

}

}

using System;

using System.Linq.Expressions;

using MovieFan.Core.Entities;

using MovieFan.Core.Repositories;

using MovieFan.Web.ViewModels;

using Microsoft.AspNetCore.Mvc;

using Microsoft.AspNetCore.Authorization;

using System.Collections.Generic;

using System.Linq;

using Microsoft.AspNetCore.Routing;

namespace MovieFan.Web.Controllers

{

[Authorize]

public class FilmController : Controller

{

private readonly IUnitOfWork \_uow;

public FilmController(IUnitOfWork uow) => \_uow = uow;

[Route("[controller]/[action]")]

public IActionResult List()

{

var model = new FilmListViewModel();

model.Genres = \_uow.GenreRepository.List();

model.Films = \_uow.FilmRepository.List();

ViewBag.FilterByNameValue = null;

return View(model);

}

[Route("[controller]/[action]")]

public IActionResult FilterByGenre(FilmListViewModel model)

{

model.Genres = \_uow.GenreRepository.List();

switch (model.GenreId)

{

case "All":

model.Films = \_uow.FilmRepository.List();

break;

case "None":

model.Films = \_uow.FilmRepository.FindFilmsByGenre(1, true);

break;

default:

model.Films = \_uow.FilmRepository.FindFilmsByGenre(int.Parse(model.GenreId));

break;

}

ViewBag.FilterByNameValue = null;

return View("List", model);

}

[Route("[controller]/[action]")]

public IActionResult FilterByName(string searchString)

{

var model = new FilmListViewModel();

model.Genres = \_uow.GenreRepository.List();

if (string.IsNullOrEmpty(searchString))

{

model.Films = \_uow.FilmRepository.List();

}

else

{

model.Films = \_uow.FilmRepository.FindFilmsByName(searchString);

}

ViewBag.FilterByNameValue = searchString;

return View("List", model);

}

[Route("[controller]/[action]/{id:guid}")]

public IActionResult Details(Guid id)

{

var model = new FilmDetailsViewModel();

model.Film = \_uow.FilmRepository.Find(id);

model.Episodes = new List<Episode>(\_uow.EpisodeRepository.FindEpisodesByFilm(id));

var user = \_uow.UserRepository.FindUserByEmail(User.Identity.Name);

var watchedFilm = user.WatchedFilms.Where(x => x.FilmId == id).SingleOrDefault();

if (watchedFilm == null)

{

ViewBag.DidUserWatch = false;

}

else

{

ViewBag.DidUserWatch = true;

}

return model.Film == null ? (IActionResult) NotFound() : View(model);

}

[Route("[controller]/[action]/{id:guid}")]

public IActionResult EpisodeDetails(Guid id)

{

var episode = \_uow.EpisodeRepository.Find(id);

var user = \_uow.UserRepository.FindUserByEmail(User.Identity.Name);

var watchedEpisode = user.WatchedEpisodes.Where(x => x.EpisodeId == id).SingleOrDefault();

if (watchedEpisode == null)

{

ViewBag.DidUserWatch = false;

}

else

{

ViewBag.DidUserWatch = true;

}

return episode == null ? (IActionResult)NotFound() : View(episode);

}

[Route("[controller]/[action]/{id:guid}")]

public IActionResult AddToWatchedFilms(Guid id)

{

var user = \_uow.UserRepository.FindUserByEmail(User.Identity.Name);

var watchedFilm = user.WatchedFilms.Where(x => x.FilmId == id).SingleOrDefault();

if (watchedFilm == null)

{

user.WatchedFilms.Add(new WatchedFilms { FilmId = id, UserId = id });

}

else

{

user.WatchedFilms.Remove(watchedFilm);

}

\_uow.Commit();

return RedirectToAction("Details", new RouteValueDictionary( new { controller = "Film", action = "Details", id = id }));

}

[HttpGet("[controller]/[action]/{id:guid}")]

public IActionResult RateFilm(Guid id)

{

var user = \_uow.UserRepository.FindUserByEmail(User.Identity.Name);

var film = \_uow.FilmRepository.Find(id);

if (user == null)

{

return NotFound();

}

var watchedFilm = user.WatchedFilms.Where(x => x.FilmId == id).SingleOrDefault();

var ratingString = "";

if (watchedFilm != null)

{

ratingString = watchedFilm.Rating == null ? "" : watchedFilm.Rating.ToString();

}

var model = new RateFilmViewModel

{

Id = user.Id,

FilmName = film.OriginalName,

Rating = ratingString

};

return View(model);

}

[HttpPost("[controller]/[action]/{id:guid}")]

public IActionResult RateFilm(RateFilmViewModel model)

{

if (!ModelState.IsValid)

{

return View(model);

}

var user = \_uow.UserRepository.FindUserByEmail(User.Identity.Name);

if (user == null)

{

return NotFound();

}

var watchedFilm = user.WatchedFilms.Where(x => x.FilmId == model.Id).SingleOrDefault();

double? rating;

if (model.Rating == null || model.Rating == "")

{

rating = null;

}

else

{

rating = double.Parse(model.Rating);

}

if (watchedFilm == null)

{

user.WatchedFilms.Add(new WatchedFilms { UserId = user.Id, FilmId = model.Id, Rating = rating });

}

else

{

watchedFilm.Rating = rating;

}

\_uow.Commit();

return RedirectToAction("Details", new RouteValueDictionary(new { controller = "Film", action = "Details", id = model.Id }));

}

[Route("[controller]/[action]/{id:guid}")]

public IActionResult AddToWatchedEpisodes(Guid id)

{

var user = \_uow.UserRepository.FindUserByEmail(User.Identity.Name);

var watchedEpisode = user.WatchedEpisodes.Where(x => x.EpisodeId == id).SingleOrDefault();

if (watchedEpisode == null)

{

user.WatchedEpisodes.Add(new WatchedEpisodes { EpisodeId = id, UserId = id });

}

else

{

user.WatchedEpisodes.Remove(watchedEpisode);

}

\_uow.Commit();

return RedirectToAction("EpisodeDetails", new RouteValueDictionary(new { controller = "Film", action = "EpisodeDetails", id = id }));

}

[HttpGet("[controller]/[action]/{id:guid}")]

public IActionResult RateEpisode(Guid id)

{

var user = \_uow.UserRepository.FindUserByEmail(User.Identity.Name);

var episode = \_uow.EpisodeRepository.Find(id);

if (user == null)

{

return NotFound();

}

var watchedEpisode = user.WatchedEpisodes.Where(x => x.EpisodeId == id).SingleOrDefault();

var ratingString = "";

if (watchedEpisode != null)

{

ratingString = watchedEpisode.Rating == null ? "" : watchedEpisode.Rating.ToString();

}

var model = new RateEpisodeViewModel

{

Id = user.Id,

EpisodeName = episode.Name,

Rating = ratingString

};

return View(model);

}

[HttpPost("[controller]/[action]/{id:guid}")]

public IActionResult RateEpisode(RateEpisodeViewModel model)

{

if (!ModelState.IsValid)

{

return View(model);

}

var user = \_uow.UserRepository.FindUserByEmail(User.Identity.Name);

if (user == null)

{

return NotFound();

}

var watchedEpisode = user.WatchedEpisodes.Where(x => x.EpisodeId == model.Id).SingleOrDefault();

double? rating;

if (model.Rating == null || model.Rating == "")

{

rating = null;

}

else

{

rating = double.Parse(model.Rating);

}

if (watchedEpisode == null)

{

user.WatchedEpisodes.Add(new WatchedEpisodes { UserId = user.Id, EpisodeId = model.Id, Rating = rating });

}

else

{

watchedEpisode.Rating = rating;

}

\_uow.Commit();

return RedirectToAction("EpisodeDetails", new RouteValueDictionary(new { controller = "Film", action = "EpisodeDetails", id = model.Id }));

}

}

}

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Mvc;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

namespace MovieFan.Web.Controllers

{

[Authorize]

public class HomeController : Controller

{

[AllowAnonymous]

public IActionResult Index()

{

ViewBag.Username = User.Identity.Name ?? "anonymous";

return View();

}

}

}

using System;

using System.Linq.Expressions;

using MovieFan.Core.Entities;

using MovieFan.Core.Repositories;

using MovieFan.Web.ViewModels;

using Microsoft.AspNetCore.Mvc;

using Microsoft.AspNetCore.Authorization;

namespace MovieFan.Web.Controllers

{

[Authorize]

public class ParticipaterController : Controller

{

private readonly IUnitOfWork \_uow;

public ParticipaterController(IUnitOfWork uow) => \_uow = uow;

[Route("[controller]/[action]")]

public IActionResult List()

{

var model = new ParticipaterListViewModel();

model.Participaters = \_uow.ParticipaterRepository.List();

model.ParticipaterRoles = \_uow.ParticipaterRoleRepository.List();

ViewBag.FilterByFullNameValue = null;

return View(model);

}

[Route("[controller]/[action]")]

public IActionResult FilterByParticipaterRole(ParticipaterListViewModel model)

{

model.ParticipaterRoles = \_uow.ParticipaterRoleRepository.List();

switch (model.ParticipaterRoleId)

{

case "All":

model.Participaters = \_uow.ParticipaterRepository.List();

break;

case "None":

model.Participaters = \_uow.ParticipaterRepository.FindParticipatersByParticipaterRole(1, true);

break;

default:

model.Participaters = \_uow.ParticipaterRepository.FindParticipatersByParticipaterRole(int.Parse(model.ParticipaterRoleId));

break;

}

ViewBag.FilterByFullNameValue = null;

return View("List", model);

}

[Route("[controller]/[action]")]

public IActionResult FilterByFullName(string searchString)

{

var model = new ParticipaterListViewModel();

model.ParticipaterRoles = \_uow.ParticipaterRoleRepository.List();

if (string.IsNullOrEmpty(searchString))

{

model.Participaters = \_uow.ParticipaterRepository.List();

}

else

{

model.Participaters = \_uow.ParticipaterRepository.FindParticipatersByFullName(searchString);

}

ViewBag.FilterByFullNameValue = searchString;

return View("List", model);

}

[Route("[controller]/[action]/{id:guid}")]

public IActionResult Details(Guid id)

{

var participater = \_uow.ParticipaterRepository.Find(id);

return participater == null ? (IActionResult)NotFound() : View(participater);

}

}

}

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Обозначение | | | | Наименование | | | | Дополнительные сведения | | | |
|  | | | | Текстовые документы | | | |  | | | |
|  | | | |  | | | |  | | | |
| БГУИР КП 1–40 01 01 01 ПЗ | | | | Пояснительная записка | | | | 97 с. | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | | Графические документы | | | |  | | | |
|  | | | |  | | | |  | | | |
| БГУИР 751003 01 СП | | | | Схема базы данных приложения | | | | Формат А1 | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  |  |  |  |  | БГУИР КП 1-40 01 01 01 Д1 | | | | | | |
|  |  |  |  |  |
|  |  |  |  |  |
| Изм. | Л. | № докум. | Подп. | Дата | Веб-приложение “MovieFan”  Ведомость курсового  проекта |  | | | | Лист | Листов |
| Разраб. | | Абраменко А.В. |  |  |  |  |  | |  | 97 | 97 |
| Пров. | | Марина И. М. |  |  |  | Кафедра ПОИТ  гр. 751003 | | | | | |