**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**

**ФАКУЛЬТЕТ ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ**

**Кафедра информационных систем управления**

МИНАЛЬД

Андрей Анатольевич

**РАЗРАБОТКА ИНТЕЛЛЕКТУАЛЬНОГО КАТАЛОГИЗАТОРА**

**МУЗЫКАЛЬНЫХ ПРОИЗВЕДЕНИЙ**

Дипломная работа

Научный руководитель:

ст. преподаватель каф. ИСУ,

С.Е. Гутников

Допущена к защите

«\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_ 2019 г.

Зав. кафедрой информационных

систем управления,

доктор тех. наук,

профессор В.В. Краснопрошин

**СОДЕРЖАНИЕ**

**ВВЕДЕНИЕ7**

**ГЛАВА 1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ И ПОСТАНОВКА ЗАДАЧИ8**

1.1 Понятие каталогизатора музыки8

1.2 История развития аудионосителей и музыкальных каталогов 8

1.3 Оффлайн-каталогизаторы9

1.4 Онлайн-каталогизаторы10

1.5 Постановка задачи11

**ГЛАВА 2. ВЫБОР ТЕХНОЛОГИЙ И ОПИСАНИЕ СТРУКТУРЫ ПРИЛОЖЕНИЯ13**

2.1 Выбор платформы и технологий для разработки13

2.2 Выбор источника данных 16

2.3 Описание структуры приложения17

**ГЛАВА 3. РЕАЛИЗАЦИЯ ПРИЛОЖЕНИЯ27**

3.1 Реализация общедоступной части приложения27

3.2 Реализация поиска31

3.3 Постановка задачи для нейросети32

3.4 Создание индивидуальных чартов34

**ЗАКЛЮЧЕНИЕ36**

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ37**

**ПРИЛОЖЕНИЕ 138**

**ПРИЛОЖЕНИЕ 243**

**ПЕРЕЧЕНЬ УСЛОВНЫХ ОБОЗНАЧЕНИЙ**

HTML - HyperText Markup Language

MVC – Model-View-Controller

ORM - [Object-relational mapping](https://ru.wikipedia.org/wiki/ORM)

SQL - Structured Query Language

БД - База данных

API - Application programming interface

**РЕФЕРАТ**

Дипломная работа, 47 страниц, 7 источников, 23 рисунка.

Ключевые слова:КАТАЛОГИЗАТОР, ЧАРТ, ВЕБ-ПРИЛОЖЕНИЕ, НЕЙРОННАЯ СЕТЬ, МУЗЫКА, БАЗА ДАННЫХ, МОДЕЛЬ, ПРЕДСТАВЛЕНИЕ, КОНТРОЛЛЕР.

Объект исследования: нейронные сети в веб-приложениях.

Цель работы: изучить технологии для разработки веб-приложения и разработать веб-приложение для каталогизирования музыкальных произведений.

Методы исследования: анализ теоретической части, разработка приложения.

Область применения: интернет пространство.

За время работы были реализованы следующие задачи: изучена литература по разработке веб-приложений, а также проанализирована актуальность разработки приложения для каталогизирования музыкальных произведений и составления индивидуальных чартов, полученных в результате обучения нейронной сети. В ходе работы были изучены различные технологии, СУБД MS SQL, фреймворки ASP.NET Core и Bootstrap, разработано соответствующее веб-приложение для каталогизирования музыкальных произведений. Работа имеет большое практическое значение для различных музыкальных сайтов, в том числе сайтов радио и социальных сетей, поскольку она помогает составлять более качественные музыкальные чарты с целью дальнейшей каталогизации музыкальных произведений, которые понравились пользователю.

**РЭФЕРАТ**

Дыпломная праца, 47 старонак, 7 крыніц, 23 малюнка.

Ключавыя словы: КАТАЛАГІЗАТАР, ЧАРТ, ВЭБ-ПРЫКЛАДАННЕ, НЕЙРОНАВАЯ СЕТКА, МУЗЫКА, БАЗА ДАНЫХ, МАДЭЛЬ, ПРАДСТАЎЛЕННЕ, КАНТРОЛЕР.

Аб'ект даследавання: нейронавыя сеткі ў вэб-прыкладаннях.

Мэта працы: вывучыць тэхналогіі для распрацоўкі вэб-прыкладання і распрацаваць вэб-прыкладанне для каталагизаванні музычных твораў.

Метады даследавання: аналіз тэарэтычнай часткі, распрацоўка прыкладання.

Вобласць ужывання: інтэрнэт прастора.

За час працы былі рэалізаваны наступныя задачы: вывучана літаратура па распрацоўцы вэб-прыкладанняў, а таксама прааналізавана актуальнасць распрацоўкі прыкладання для каталагизавання музычных твораў і складання індывідуальных чартаў, атрыманых у выніку навучання нейронавай сеткі. У ходзе работы былі вывучаны розныя тэхналогіі, СКБД MS SQL, фреймворкі ASP.NET Core і Bootstrap, распрацавана адпаведнае вэб- прыкладанне для каталагизавання музычных твораў. Праца мае вялікае практычнае значэнне для розных музычных сайтаў, у тым ліку сайтаў радыё і сацыяльных сетак, паколькі яна дапамагае складаць больш якасныя музычныя чарты з мэтай далейшай каталагізацыі музычных твораў, якія спадабаліся карыстальніку.

**ABSTRACT**

The diploma, 47 pages, 7 sources, 23 drawings.

Keywords: CATALOG, CHART, WEB APPLICATION, NEURAL NETWORK, MUSIC, DATABASE, MODEL, VIEW, CONTROLLER.

Object of study: neural networks in web applications.

Objective: to study the technology for developing a web application and develop a web application for cataloging music.

Research methods: analysis of the theoretical part, the development of the application.

Scope: Internet space.

During the work, the following tasks were implemented: the literature on the development of web applications was studied, and the relevance of developing an application for cataloging musical works and compiling individual charts derived from neural network training was analyzed. During the work, various technologies were studied, MS SQL DBMS, ASP.NET Core and Bootstrap frameworks, a corresponding web application for music cataloging was developed. The work is of great practical importance for various music sites, including radio and social networking sites, because it helps to create better-quality music charts for the purpose of further cataloging music that the user liked.

# ВВЕДЕНИЕ

Подавляющее большинство людей любят слушать музыку. У каждого отдельного человека музыкальные предпочтения формируются в зависимости от различных факторов: пола, возраста, типа характера, типа темперамента и многих других. Люди могут слушать различные стили в зависимости от поры года, погоды, настроения и т.д.

Но всем хочется, чтобы любимая музыка всегда была доступна вне зависимости от места, времени и обстоятельств. Также никто не любит искать любимый альбом или трек долго: их должно быть легко найти. Поэтому возникает потребность каким-либо образом хранить музыку. Лучше всего с этой задачей справляются музыкальные каталогизаторы.

Каталогизатор – это программа, которая помогает хранить музыку в удобном формате.

В дипломной работе поднимается тема актуальности объединения онлайн и оффлайн каталогизаторов, всеобщих и индивидуальных чартов в единое приложение, а также подробно рассматривается процесс его создания.

**ГЛАВА 1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ И ПОСТАНОВКА ЗАДАЧИ**

* 1. **Понятие каталогизатора музыки**

Каталог – в общем случае, некий список информации об объектах, составленный с целью облегчения поиска этих объектов по какому-либо признаку. В роли объекта может выступать абсолютно всё: населённые пункты некоторого государства, книги в библиотеке, продукция завода, фильмы или песни. Каталоги бывают двух видов: бумажные и электронные.

Каталогизатор – это программа, которая создаёт электронные каталоги, поддерживает их в актуальном состоянии, позволяет добавлять, удалять, редактировать различную информацию.

* 1. **История развития аудионосителей и музыкальных каталогов**

Самым первым аудионосителем был восковой цилиндр, который был изобретён Томасом Эдисоном в 1877-ом году. Музыки в то время было не так много, как в XXI веке, восковой цилиндр был дорог и редок, поэтому о каталоге или каком-либо сборнике музыки не могло идти и речи.

В 1894-ом году была изобретена граммофонная пластинка. В 1903-ем году фирма «Одеон» научилась делать пластинки двусторонними. Затем пластинки начали производить на конвейерах, и они стали серийным продуктом. С этого момента можно было покупать много различных пластинок, подписывать их и, например, в алфавитном порядке по названию или исполнителю ставить на полке. Это и были первые каталоги.

Затем было ещё несколько качественных скачков в изучении способов хранения музыки:

* в 1940-ых годах была изобретена магнитная лента;
* в 1948-ом – долгоиграющая пластинка;
* в 1963-ем – компакт-кассета;
* в 1969-ом – микрокассета.

Принцип каталогизации за это время не изменился – различные аудионосители подписывались и ставились на полку.

Самый большой прогресс был достигнут при переходе от аналоговых аудионосителей к цифровым:

* в 1976-ом году научились записывать аудиофайлы на компьютер;
* в 1982-ом году был изобретён компакт-диск;
* в 1992-ом – компакт-кассеты;
* в 2000-ом – DVD-Audio.

Начиная с 1976-ого года каталогизировать музыку стало проще: её можно было хранить на компьютере. По мере прогресса в разработке языков программирования начало развиваться программное обеспечение. А по мере развития программного обеспечения начали развиваться и музыкальные каталогизаторы.

С момента изобретения компьютеров и до изобретения интернета единственным видом электронных музыкальных каталогизаторов были оффлайн-каталогизаторы.

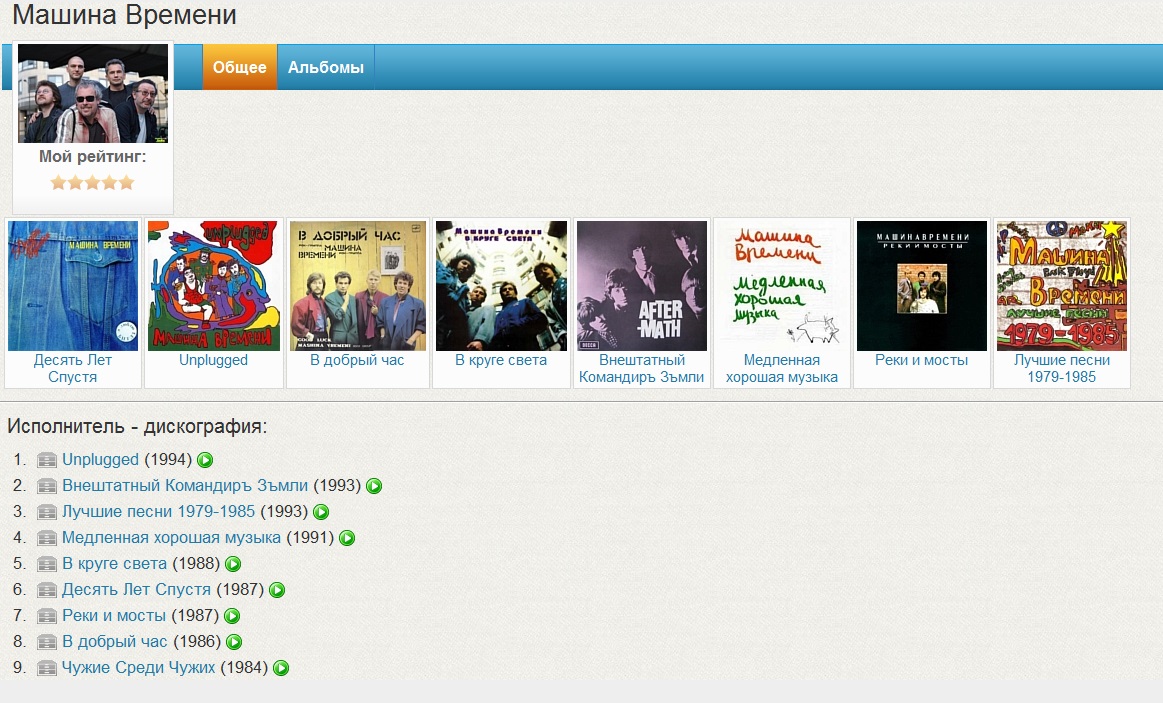
С момента изобретения интернета начали развиваться онлайн-каталогизаторы. Хранить информацию в интернете намного удобнее, чем на компьютере, так как доступ к интернету можно получить из любой точки мира, а к своему компьютеру только в определённом месте, например, дома или на работе. В связи с этим в начале 2000-ых онлайн-каталогизаторы начали набирать популярность и вследствие обходить по популярности оффлайн-каталогизаторы. Но несмотря на такой большой рост популярности оффлайн-каталогизаторы до сих пор пользуются немалым спросом.

* 1. **Оффлайн-каталогизаторы**

Оффлайн-каталогизатор – это программа, которая создаёт каталог на основе локальной базы, т.е. базы, расположенной на компьютере.

Принцип работы оффлайн-каталогизатора таков: на локальном компьютере в определённом месте создаётся папка, в которой хранится вся музыка, фотографии и информация.

Одним из самых популярных представителей такого вида каталогизаторов является программа Musicnizer. Интерфейс окна, отображающего информацию о музыкальном коллективе, изображён на рисунке 1.



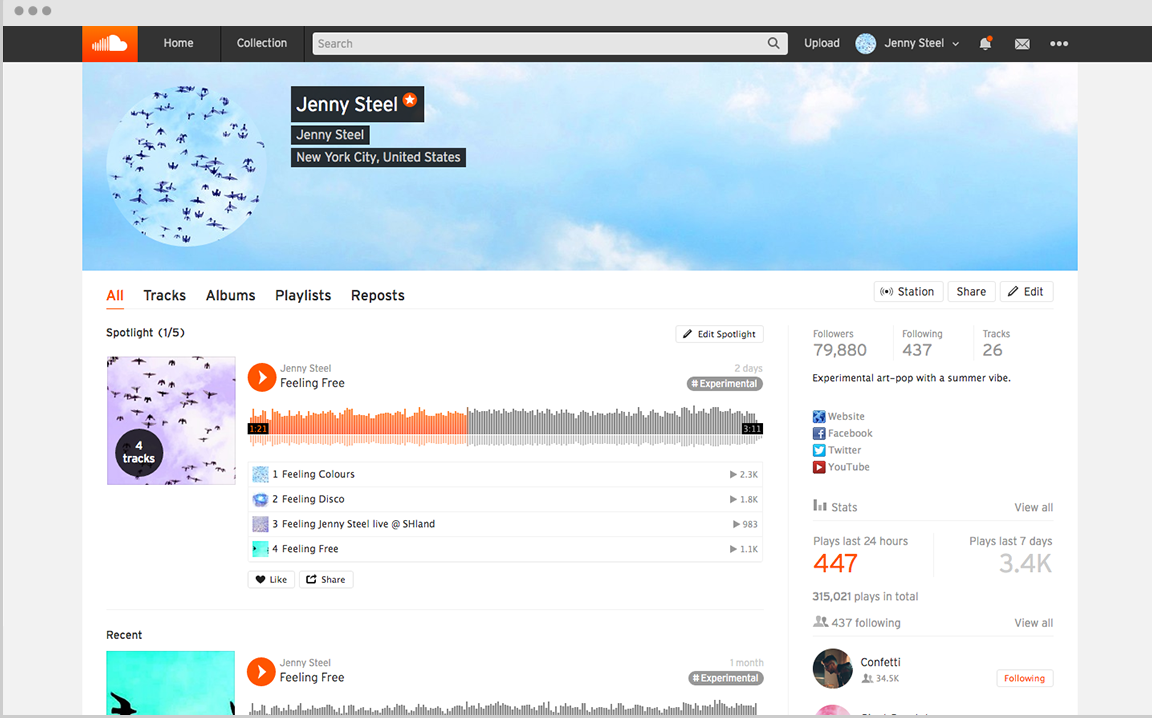
**Рисунок 1.1 – Интерфейс программы Musicnizer**

В верхней части окна находится название группы, затем фотография и рейтинг. Посередине расположен список музыкальных альбомов с их обложками, отсортированный по популярности. Внизу окна также расположены альбомы, но уже отсортированные в хронологическом порядке. По каждой надписи можно нажать и перейти в соответствующее окно.

* 1. **Онлайн-каталогизаторы**

Онлайн-каталогизатор – это программа, которая создаёт каталог на основе удалённой базы, т.е. базы, расположенной на сервере.

Принцип работы онлайн-каталогизатора таков: на некотором удалённом сервере хранится база со всеми исполнителями, альбомами и песнями. Пользователь, предварительно зарегистрировавшись, может прослушивать все композиции, которые хранятся на сервере. А после перехода в личный кабинет может создавать свой собственный каталог.



**Рисунок 1.2 – Интерфейс сайта soundcloud.com**

В верхней части страницы расположено меню, в котором есть поиск, ссылка на главную страницу, собственную коллекцию песен и личный кабинет. Под меню располагается обложка песни, а под обложкой сама песня. В правой части сайта расположена информация о количестве треков и пользователей, следящих за творческой деятельностью данного исполнителя, а также о количестве прослушиваний за последний день и неделю.

* 1. **Постановка задачи**

У обоих видов каталогизаторов есть свои плюсы и минусы.

Плюсы оффлайн-каталогизаторов: доступ к каталогу без доступа к интернету, очень большая скорость загрузки окон; минусы: трудность пополнения коллекции, отсутствие доступа вне компьютера.

Плюсы онлайн-каталогизаторов: доступ к каталогу в любой части мира, лёгкость обновления и пополнения коллекции; минусы: отсутствие доступа к каталогу при отсутствии доступа к интернету.

Главная задача состоит в том, чтобы объединить все плюсы и избавиться от минусов этих двух видов каталогизаторов, добавив системе интеллектуальности за счёт создания индивидуальных музыкальных чартов с помощью нейронной сети.

Пользователи приложения должны иметь возможность изучать большую музыкальную базу данных, которая предоставляется через API сайта last.fm. Также пользователи должны иметь возможность составлять собственные чарты и изучать общие чарты двух видов: один основывается на количестве прослушиваний музыкального произведения на сайте last.fm, а второй основывается на результатах обучения нейронной сети, которая обрабатывает введённые пользователем личные данные и составленные пользователем индивидуальные чарты.

**ГЛАВА 2. ВЫБОР ТЕХНОЛОГИЙ И ОПИСАНИЕ СТРУКТУРЫ ПРИЛОЖЕНИЯ**

**2.1 Выбор платформы и технологий для разработки**

Чтобы выбрать платформу нужно определиться с целями приложения и его целевой аудиторией. Цель – разработать лёгкую в понимании и сопровождении систему. Целевая аудитория – это пользователи телефонов, планшетов и компьютеров различных операционных систем, проживающие в различных странах.

Таким образом приложение должно быть:

* кроссплатформенным, т.е. должно быть в состоянии запуститься в разных операционных системах, например, Windows, UNIX, Mac OS, iOS, Android и других;
* кроссбраузерным, т.е. должно быть в состоянии запуститься в различных браузерах, например, Google Chrome, Internet Explorer, Opera, Microsoft Edge и других;
* адаптивным, т.е. должно выглядеть хорошо и красиво в разных разрешениях, т.е. на телефоне, планшете, ноутбуке или стационарном компьютере.

Под данные требования прекрасно подходит платформа Microsoft .NET.

Microsoft .NET (она же .NET Framework) – программная платформа, выпущенная компанией Microsoft в 2002-ом году. Основой платформы является общеязыковая среда исполнения Common Language Runtime, которая подходит для разных языков программирования.

После выбора платформы нужно выбрать фреймворк. Фреймворк – это программная платформа, определяющая структуру программной системы, программное обеспечение, облегчающее разработку и объединение разных компонентов большого программного проекта.

В Microsoft .NET выбор стоит между ASP.NET MVC Framework и .NET Core Framework. .NET Core – это новый фреймворк с улучшенной по сравнению с .NET Framework архитектурой. Ещё год назад я бы выбрал .NET Framework, так как .NET Core не поддерживал многие программных пакеты, и вследствие этого уступал в надёжности и стабильности. Например, пакеты NuGet или Bower могли быть не перенесены из .NET Framework в .NET Core, и в такой ситуации приходилось бы искать менее популярные и менее удобные пакеты, но которые уже были перенесены. Это являлось большим минусом. Но теперь такой проблемы не существует и выбор .NET Core становится очевидным.

ASP.NET Core - это бесплатная веб-инфраструктура с открытым исходным кодом, обладающая более высокой производительностью, чем ASP.NET, разработанная Microsoft и сообществом. Это модульная структура, которая работает как на полной .NET Framework, на Windows, так и на кроссплатформенном .NET Core. Фреймворк представляет собой полное переписывание, объединяющее ранее отдельные ASP.NET MVC и ASP.NET Web API в единую модель программирования.

Несмотря на то, что это новый фреймворк, построенный на новом веб-стеке, он обладает высокой степенью совместимости концепций с ASP.NET MVC. Приложения ASP.NET Core поддерживают параллельное управление версиями, при котором разные приложения, работающие на одном компьютере, могут ориентироваться на разные версии ASP.NET Core. Это невозможно с предыдущими версиями ASP.NET.

ASP.NET Core MVC – это фреймворк для создания приложений на платформе .NET, который реализует шаблон Model – View – Controller. Актуальной версией ASP.NET Core MVC является ASP.NET Core MVC 2.0.

MVC (Model – View – Controller) – это схема разделения данных приложения, пользовательского интерфейса и управляющей логики на три отдельных компонента: модель, представление и контроллер – таким образом, что модификация каждого компонента может осуществлять независимо друг от друга.

Далее нужно определиться с языком программирования. Microsoft .NET предлагает следующие языки: С#, С++/CLI, F#, Visual Basic .NET, Jscript .NET, а также уже устаревший J#. Самым перспективным и быстроразвивающимся из всех вышеперечисленных является язык C#.

После выбора платформы, главного фреймворка и языка программирования появляется вопрос о выборе СУБД. Система управления базами данных (СУБД) – совокупность программных и лингвистических средств общего или специального назначения, обеспечивающих управление созданием и использованием баз данных. Лучшим выбором является Microsoft SQL Server, так как это тоже продукт Microsoft и с ним будет проще работать. Для удобства работы с СУБД можно использовать Entity Framework Core.

ADO.NET Entity Framework Core — объектно-ориентированная технология доступа к данным, является [object-relational mapping](https://ru.wikipedia.org/wiki/ORM) (ORM) решением для [.NET Framework](https://ru.wikipedia.org/wiki/.NET_Framework) от [Microsoft](https://ru.wikipedia.org/wiki/Microsoft). Предоставляет возможность взаимодействия с объектами как посредством [LINQ](https://ru.wikipedia.org/wiki/LINQ) в виде LINQ to Entities, так и с использованием Entity SQL. Для облегчения построения web-решений используется как [ADO.NET Data Services](https://ru.wikipedia.org/wiki/ADO.NET_Data_Services) ([Astoria](https://ru.wikipedia.org/wiki/ADO.NET_Data_Services)), так и связка из [Windows Communication Foundation](https://ru.wikipedia.org/wiki/Windows_Communication_Foundation) и [Windows Presentation Foundation](https://ru.wikipedia.org/wiki/Windows_Presentation_Foundation), позволяющая строить многоуровневые приложения, реализуя один из шаблонов проектирования [MVC](https://ru.wikipedia.org/wiki/Model-View-Controller), [MVP](https://ru.wikipedia.org/wiki/Model-View-Presenter) или [MVVM](https://ru.wikipedia.org/wiki/Model-View-ViewModel).

В виду отсутствия разнообразия языками разметки выбраны HTML5 и CSS3. Для удобства разработки дизайна страниц следует использовать CSS-фреймворк Bootstrap версии 3.3.7.

Bootstrap — свободный набор инструментов для создания [сайтов](https://ru.wikipedia.org/wiki/%D0%A1%D0%B0%D0%B9%D1%82) и [веб-приложений](https://ru.wikipedia.org/wiki/%D0%92%D0%B5%D0%B1-%D0%BF%D1%80%D0%B8%D0%BB%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D0%B5). Включает в себя [HTML](https://ru.wikipedia.org/wiki/HTML) и [CSS](https://ru.wikipedia.org/wiki/CSS)-шаблоны оформления для [типографики](https://ru.wikipedia.org/wiki/%D0%A2%D0%B8%D0%BF%D0%BE%D0%B3%D1%80%D0%B0%D1%84%D0%B8%D0%BA%D0%B0), веб-форм, кнопок, меток, блоков навигации и прочих компонентов веб-интерфейса, включая [JavaScript](https://ru.wikipedia.org/wiki/JavaScript)-расширения.

Для придания страницам интерактивности логично использовать JavaScript и JavaScript библиотеку jQuery, а для работы с асинхронными методами и увеличения скорости работы сайта – AJAX.

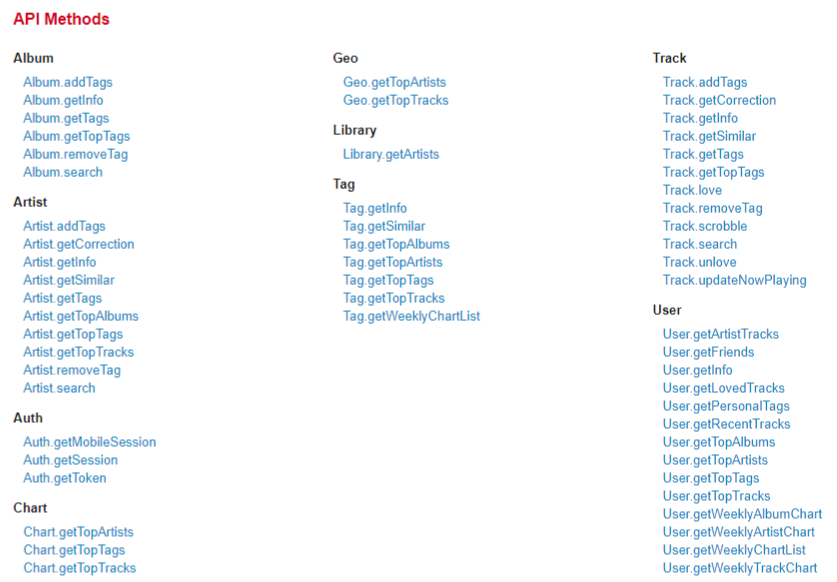
jQuery —  [библиотека JavaScript](https://ru.wikipedia.org/wiki/%D0%91%D0%B8%D0%B1%D0%BB%D0%B8%D0%BE%D1%82%D0%B5%D0%BA%D0%B0_JavaScript), фокусирующаяся на взаимодействии [JavaScript](https://ru.wikipedia.org/wiki/JavaScript) и [HTML](https://ru.wikipedia.org/wiki/HTML). Библиотека jQuery помогает легко получать доступ к любому элементу [DOM](https://ru.wikipedia.org/wiki/Document_Object_Model), обращаться к атрибутам и содержимому элементов [DOM](https://ru.wikipedia.org/wiki/Document_Object_Model), манипулировать ими. Также библиотека jQuery предоставляет удобный [API](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D1%80%D1%84%D0%B5%D0%B9%D1%81_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F_%D0%BF%D1%80%D0%B8%D0%BB%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D0%B9) для работы с [AJAX](https://ru.wikipedia.org/wiki/AJAX).

**2.2 Выбор источника данных**

Для начала нужно выбрать удалённую онлайн-базу. Выбор пал на базу сайта last.fm, так как это одна из крупнейших музыкальных баз в мире.

Last.fm – это сайт, посвящённый музыке. С помощью плагинов к медиаплеерам он собирает информацию о музыке, которую слушают пользователи, и на основе полученных данных автоматически составляет общие и индивидуальные чарты.

Чтобы получать информацию от этого сайта нужно делать запросы к его API.



**Рисунок 2.1 – Методы API сайта last.fm**

API (программный интерфейс приложения, интерфейс прикладного программирования от английского application programming interface) – это набор готовых классов, процедур, функций, структур и констант, предоставляемых приложением (библиотекой, сервисом) или операционной системой для использования во внешних программных продуктах. API определяет функциональность, которую предоставляет программа, при этом API позволяет абстрагироваться от того, как именно эта функциональность реализована.

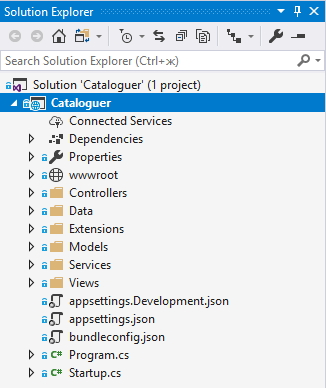
Чтобы сайт возвращал информацию в запрос надо вставить ключ доступа, который называется API\_KEY. Чтобы его получить нужно зарегистрироваться на сайте как разработчик и затем его можно будет найти в личном кабинете. Ключ доступа - это строка, состоящая из цифр и букв в нижнем регистре, например, «f39425750fc23d743fbf853d9585a46c».

После регистрации и получения ключа доступа можно делать запросы на сайт. Полный список всех методов API можно найти на рисунке 2.1.

**2.3 Описание структуры приложения**

Всё приложение состоит из семи папок, каждая из которых отвечает за некоторую функциональность приложения, и нескольких общих файлов, которые нельзя чётко отнести к какой-либо папке.

Структура проекта представлена на рисунке 2.2.

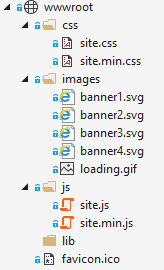


**Рисунок 2.2 – Структура проекта**

Рассмотрим эти папки и файлы подробнее.

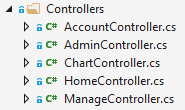
Папка wwwroot (рисунок 2.3). В этой папке содержится всё статическое содержимое веб-приложения, такое как картинки, файлы стилей и файлы сторонних библиотек. Папка wwwroot состоит из 4 подпапок и одного файла:

* css – тут хранится файл site.css, в котором содержатся все стили css специфичные для данного проекта. Также здесь хранится файл site.min.css, который является минимизированной версией вышеупомянутого файла site.css. Файл site.min.css нужен для того, чтобы уменьшить объём данных, передаваемых клиенту от сервера.
* js - тут хранится файл site.js, в котором содержатся весь специфичный для данного проекта код на языке JavaScript. Также здесь хранится файл site.min.js, который является минимизированной версией файла site.js. Файл site.min.js также нужен для того, чтобы уменьшить объём данных, передаваемых клиенту от сервера.
* lib – тут хранятся все сторонние библиотеки, которые используются в проекте. Например, в данном проекте используются библиотеки bootstrap, jquery, jquery-validation и jquery-validation-unobtrusive.
* images – тут хранятся все картинки в различных разрешениях, таких как png, jpeg, gif и svg, которые будут использоваться в проекте.
* favicon.ico – это иконка приложения, которая будет отображаться во вкладке в браузере.



**Рисунок 2.3 – Структура папки wwwroot**

Папка Controllers (рисунок 2.4). В этой папке хранятся все контроллеры приложения. Контроллер – это самый главный компонент MVC, это сердце приложения. Контроллер обеспечивает связь между пользователем и системой. Он контролирует и направляет данные от пользователя к системе и наоборот. Контроллер использует модель и представление для реализации необходимого действия.

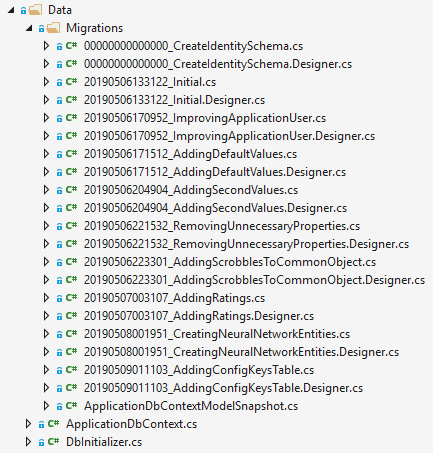


**Рисунок 2.4 – Структура папки Controllers**

В папке Controllers находится 5 контроллеров:

* AccountController – это контроллер, который отвечает за авторизацию и аутентификацию. В нём определены действия Register, Login, LoginWithRecoveryCode, Logout, ChangePassword и др.
* AdminController – это контроллер, который доступен только администраторам веб-приложения. В нём определены действия Index и Learn, открывающие доступ к изучению, изменению параметров и обучению нейронной сети.
* ChartController – это контроллер, который доступен только для авторизированных пользователей. В нём определены действия Index, FullChart, ChangeChart, AddToClientChart и RemoveFromClientChart, которые предоставляют функциональность для чтения и изменения индивидуальных и всеобщих музыкальных чартов.
* HomeController – это контроллер, который доступен для всех пользователей. В нём определены действия TopArtists, ArtistTracks, DownloadTrack, Search и др. Эти действия предоставляют функциональность для изучения музыкальной базы данных last.fm с помощью запросов к API.
* ManageController – это контроллер, который отвечает за функциональность, которая предоставляется зарегистрированному и авторизированному пользователю для изменения данных собственного аккаунта. В этом контроллере определены такие действия как SendVerificationEmail, TwoFactorAuthentication, SetPassword, ShowRecoveryCodes и др.

Папка Data (рисунок 2.5). В этой папке находятся файлы, которые связывают приложение с БД.



**Рисунок 2.5 – Структура папки Data**

В подпапке Migrations находятся все файлы миграций проекта. Система миграций в Entity Framework Core – это система наподобие системы контроля версий, только для структуры базы данных.

При каждом изменении моделей или контекста данных нужно запустить команду add-migration MigrationName из окна Package Manager Console в Visual Studio. После запуска команды автоматически сгенерируется файл на языке C#, который будет отражать изменения, которые произошли в моделях или контексте данных. После создания файла миграции нужно непосредственно обновить структуру БД. Для этого нужно запустить команду update-database из окна Package Manager Console в Visual Studio. Эта команда сгенерирует SQL-скрипты на основе файлов миграций на языке C# и запустит эти скрипты на SQL Server.

Как можно заметить на рисунке 2.6, сгенерированный класс содержит два метода: Up и Down. Метод Up отвечает за действия, которые должны быть проделаны с БД при применении миграции, а метод Down за действия, которые должны быть проделаны при откате миграции в первоначальное состояние. Например, в миграции AddingConfigKeysTable в методе Up создаётся таблица с именем ConfigKeys, а в методе Down она удаляется.



**Рисунок 2.6 – Файл миграции AddingConfigKeysTable**

Также в папке Data помимо миграций также содержатся файл контекста данных ApplicationDbContext.cs, в котором перечислены все сущности, таблицы для которых должны быть созданы, и файл инициализатора базы данных DbInitializer.cs, в котором указаны данные, которыми должна быть заполнена база данных, в том случае, если по какой-то причине таблицы из этой БД оказались пустыми.

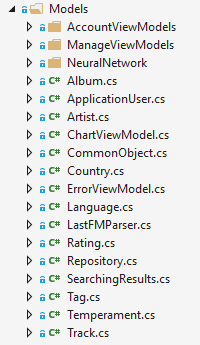
Папка Extensions (рисунок 2.7). В этой папке находятся все классы с методами расширений, которые необходимы в проекте.



**Рисунок 2.7 – Структура папки Extensions**

Класс EmailSenderExtensions содержит методы расширений для объектов классов, которые реализуют интерфейс IEmailSender. А класс UrlHelperExtensions содержит методы расширений для объектов классов, которые реализуют интерфейс IUrlHelper.

Папка Models (рисунок 2.8). В этой папке хранятся все модели приложения.



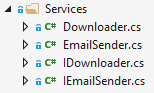
**Рисунок 2.8 – Структура папки Models**

Модель – это один из компонентов MVC. Модель предоставляет данные и методы работы с ними: запросы в базу данных, проверка на корректность. Модель не зависит от представления (не знает, как данные визуализировать) и контроллера (не имеет точек взаимодействия с пользователем) просто предоставляя доступ к данным и управлению ими. Модель строится таким образом, чтобы отвечать на запросы, изменяя своё состояние, при этом может быть встроено уведомление [наблюдателей](https://ru.wikipedia.org/wiki/%D0%9D%D0%B0%D0%B1%D0%BB%D1%8E%D0%B4%D0%B0%D1%82%D0%B5%D0%BB%D1%8C_(%D1%88%D0%B0%D0%B1%D0%BB%D0%BE%D0%BD_%D0%BF%D1%80%D0%BE%D0%B5%D0%BA%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F)). Модель, за счёт независимости от визуального представления, может иметь несколько различных представлений для одной модели.

В папке Models содержится 3 подпапки: NeuralNetwork содержит классы, необходимые для работы с нейронной сетью, AccountViewModels и ManageViewModels содержат модели, которые специально предназначены для передачи представлению, так называемые вью-модели. Также в папке Models содержится множество других классов, которые будут более подробно рассмотрены в главе 3.

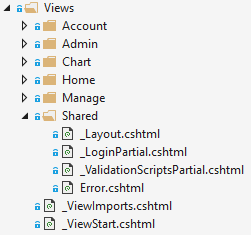
Папка Services (рисунок 2.9). Эта папка содержит классы-микросервисы, которые образуют, отделённую от остального приложения, оболочку с микросервисной архитектурой.

Микросервисная архитектура  — вариант [сервис-ориентированной архитектуры](https://ru.wikipedia.org/wiki/%D0%A1%D0%B5%D1%80%D0%B2%D0%B8%D1%81-%D0%BE%D1%80%D0%B8%D0%B5%D0%BD%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D0%B0%D1%8F_%D0%B0%D1%80%D1%85%D0%B8%D1%82%D0%B5%D0%BA%D1%82%D1%83%D1%80%D0%B0) программного обеспечения, ориентированный на взаимодействие насколько это возможно небольших, слабо связанных и легко изменяемых модулей — микросервисов.



**Рисунок 2.9 – Структура папки Services**

Папка Views (рисунок 2.10).



**Рисунок 2.10 – Структура папки Views**

В папке Views содержатся все представления в проекте. Представление отвечает за получение необходимых данных из модели и отправляет их пользователю. Представление не обрабатывает введённые данные пользователя.

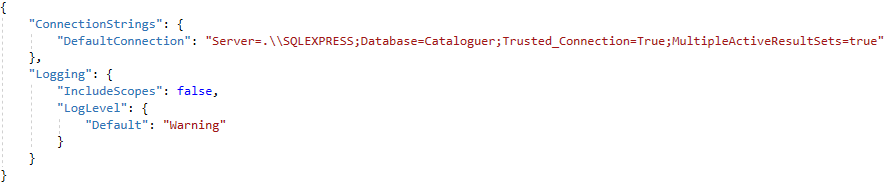
Папке Views состоит из 6 подпапок и 2 файлов. 5 папок (Account, Admin, Chart, Home, Manage) содержат представления для 5 контроллеров проекта, которые были описаны ранее. Папка Shared содержит общие представления, самым главным из которых является \_Layout.cshtml, разметка которого продемонстрирована на рисунке 2.11.



**Рисунок 2.11 – Файл Views\Shared\\_Layout.cshtml**

Также в папке Views содержатся файл \_ViewImports.cshtml, который целиком добавляется в каждый вывод разметки, чтобы импортировать все пространства имён для распознавания механизмом визуализации Razor, и файл \_ViewStart.cshtml, который состоит из одной строки «@{ Layout = "\_Layout";}», которая указывает имя файла мастер-страницы, которое будет использоваться по умолчанию.

Файлы appsettings.json (рисунок 2.12) нужны для того, чтобы сконфигурировать приложение. В них можно указать строку подключения, настройки логирования, различные конфигурационные параметры и многое другое.



**Рисунок 2.12 – Файл appsettings.json**

Помимо appsettings.json также определён файл appsettings.Development.json, который имеет такую же структуру и переопределяет конфигурационные секции из главного файла appsettings.json, если значение переменной, отвечающей за среду выполнения, равно Development.

Файл bundleconfig.json (рисунок 2.13) нужен для того, чтобы сконфигурировать способ, которым будут создаваться минимизированные версии статических файлов.



**Рисунок 2.13 – Файл bundleconfig.json**

Класс Startup.cs (рисунок 2.14) содержит 2 метода: ConfigureServices и Configure. ConfigureServices нужен для того, чтобы объявить сервисы, которые будут использованы в приложении. А метод Configure нужен для того, чтобы сконфигурировать объявленные сервисы каким-либо способом. Способ конфигурирования может зависеть от различных параметров, таких как среда выполнения приложения.



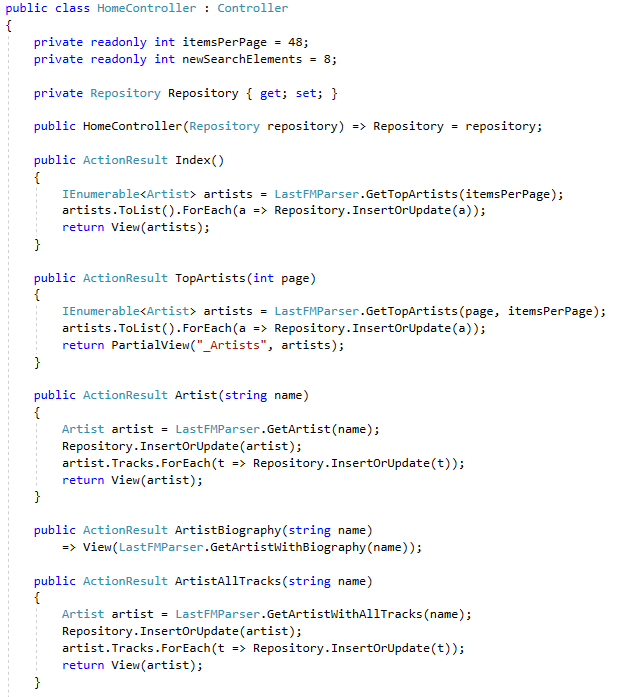
**Рисунок 2.14 – Файл Startup.cs**

Класс Program.cs содержит метод Main, который выполняется при запуске приложения. Сначала метод Main создаёт объект класса, который реализует интерфейс IWebHost, с помощью конфигурирующего класса Startup. Затем метод Main запускает приложение посредством выполнения метода Run у сконфигурированного объекта IWebHost.

**ГЛАВА 3. РЕАЛИЗАЦИЯ ПРИЛОЖЕНИЯ**

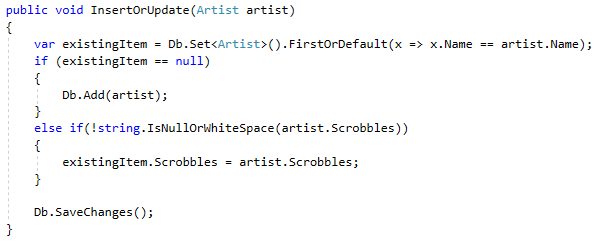
**3.1 Реализация общедоступной части приложения**

В самом начале нужно реализовать функциональность, которая будет доступна всем пользователям веб-приложения: как зарегистрированным и авторизированным, так и незарегистрированным. За это отвечает контроллер Home. Основные действия контроллера Home приведены на рисунке 3.1. Рассмотрим их подробнее.



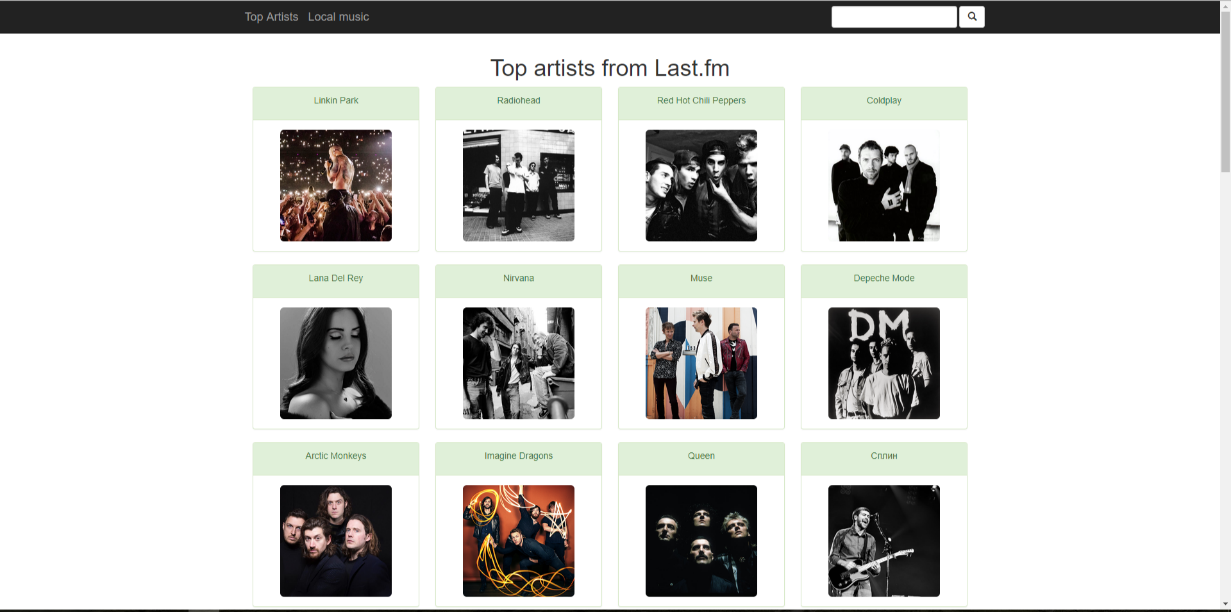
**Рисунок 3.1 – Основные действия контроллера Home**

Большинство действий этого контроллера сначала вызывает соответствующий метод класса LastFMParser, код которого приведён в Приложении 1, затем заполняет базу данных отсутствующими исполнителями и треками или обновляет информацию, если такие исполнители и треки уже присутствуют в базе, и в конце передаёт полученные данные соответствующему представлению.



**Рисунок 3.2 – Метод InsertOrUpdate класса Repository**

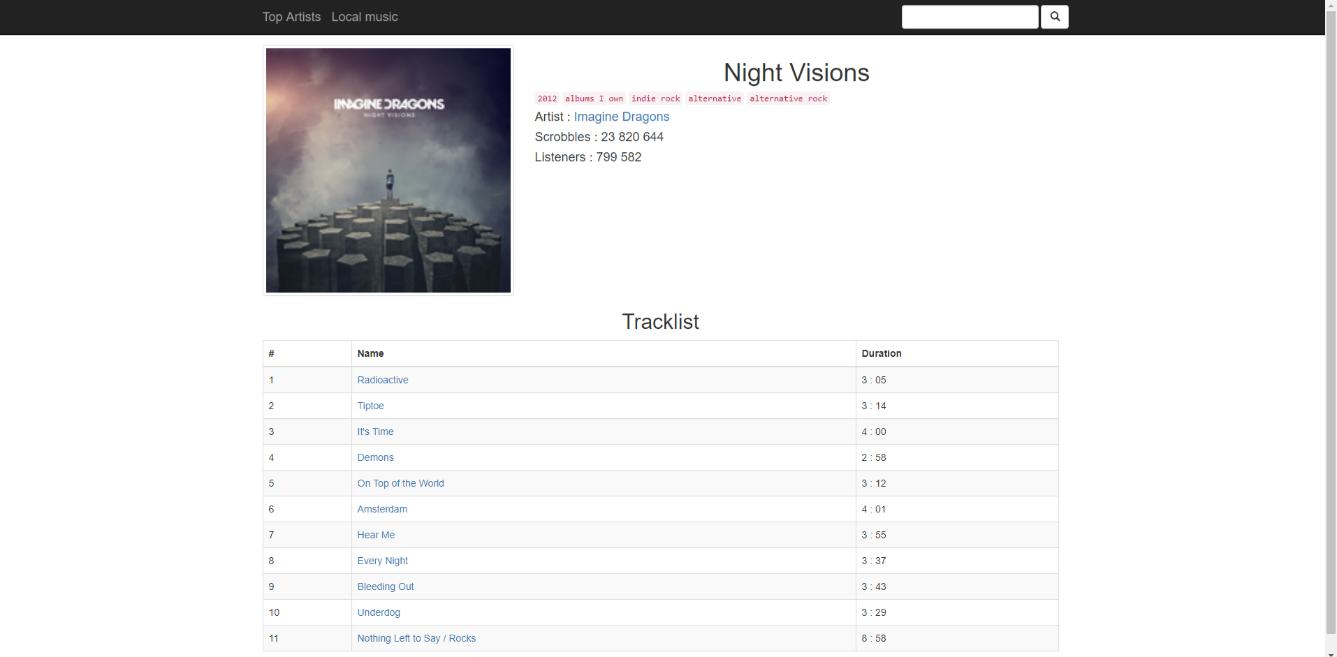
Для примера рассмотрим действие Index. Сначала вызывается метод GetTopArtists класса LastFMParser: с помощью API запроса получается список 48 самых популярных исполнителей по мнению сайта last.fm. Затем этот список передаётся в метод InsertOrUpdate класса Repository, приведённый на рисунке 3.2, который синхронизирует полученные данные с данными в БД.



**Рисунок 3.3 – Интерфейс главной страницы**

В самом конце список исполнителей передаётся представлению, который визуализирует его в виде вертикальной панели, наполненной карточками с названием и фотографией исполнителя, как показано на рисунке 3.3.

На рисунке 3.4 приведён интерфейс страницы, отвечающей за предоставление информации об альбоме Night Visions группы Imagine Dragons. На этой странице можно увидеть обложку альбома, имя исполнителя, количество прослушиваний этого альбома пользователями last.fm, а также список треков, входящих в данный альбом.



**Рисунок 3.4 – Интерфейс страницы альбома**

Как было сказано ранее, практически все действия выполняют вызов метода, который в свою очередь выполняет API запрос.

Рассмотрим структуру запроса на примере запроса «http://ws.audioscrobbler.com/2.0/?api\_key=f39425750fc23d743fbf853d9585a46c&method=artist.gettopalbums&artist=Coldplay&page=1&limit=8»:

* http://ws.audioscrobbler.com/2.0/ - адрес API, после него по порядку пойдёт различные параметры;
* ?api\_key=f39425750fc23d743fbf853d9585a46c – знак вопроса означает, что после него начинается перечисление всех параметров запроса, первый из которых, это ключ доступа;
* method=artist.gettopalbums – непосредственно метод, к которому мы будем обращаться, в данном случае это метод получения самых популярных альбомов артиста;
* artist=Coldplay – название музыкальной группы, коллектива, дуэта или имя и фамилия одиночного исполнителя;
* page=1&limit=8 – два параметра, которые определяют какие именно результаты надо вывести, limit говорит, что результатов должно быть не больше 8, а page – что должны быть выведены самые первые результаты.

Итого: должны быть возвращены 8 самых популярных альбомов группы Coldplay. Результаты приведены на рисунке 3.5.



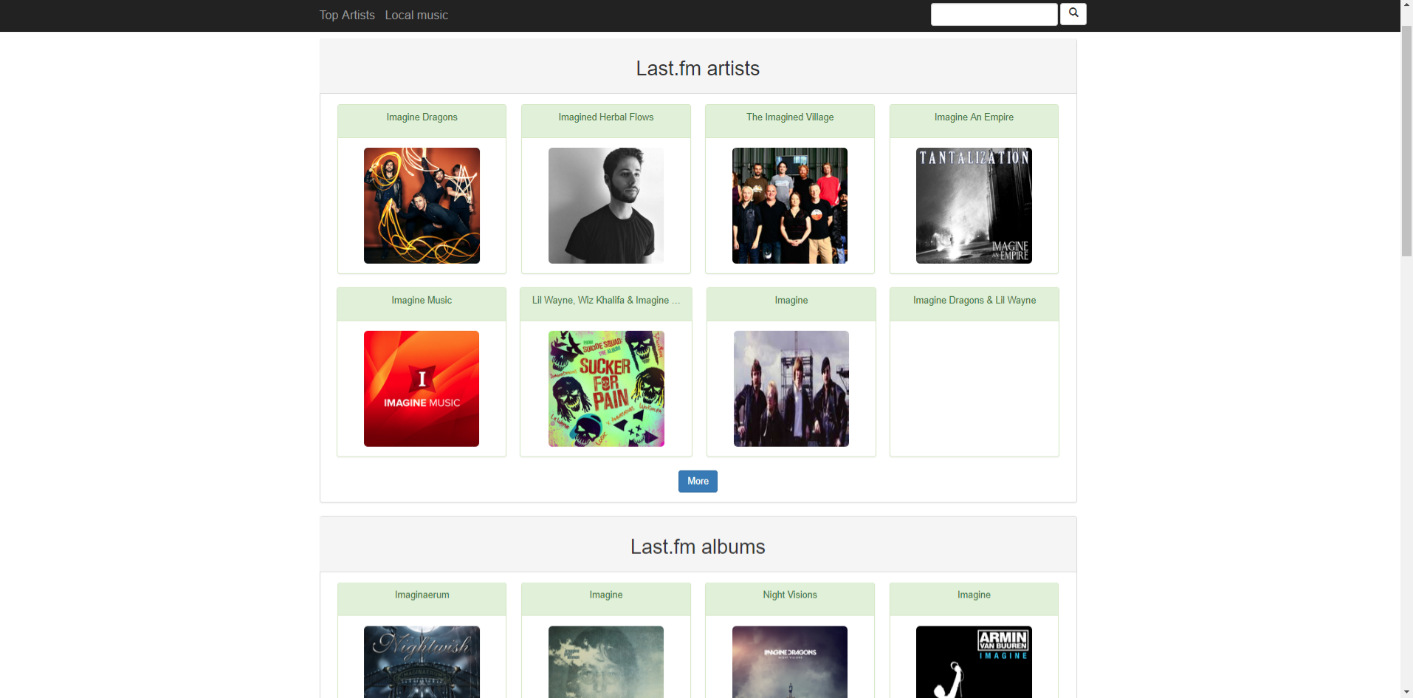
**Рисунок 3.5 – Результаты запроса**

**3.2 Реализация поиска**

Для реализации поиска нужно добавить:

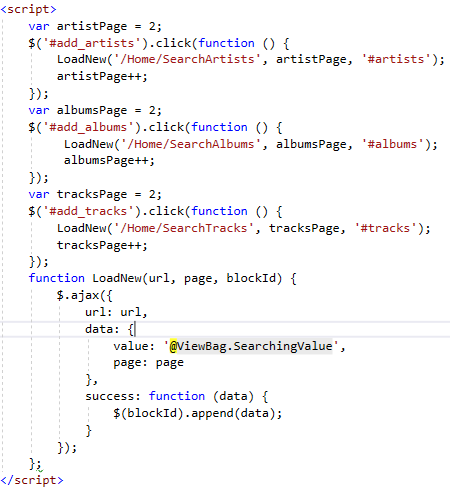
* новые методы в класс LastFMParser, чтобы делать запросы к API, связанные с поиском по подстроке;
* новые методы в контроллер, чтобы искать в локальной базе подходящие записи и выдавать их по нажатию на кнопку поиска;
* новые представления, которые будут отвечать именно за вид отображаемых ответов.

Поиск будет выдавать результаты отдельно для музыкантов, альбомов и песен. Интерфейс страницы поиска можно увидеть на рисунке 3.6.



**Рисунок 3.6 – Интерфейс страницы поиска**

На рисунке 3.7 приведён скрипт, который по нажатию на кнопку «More» («Ещё») выводит следующие 8 результатов на страницу. Он работает так: по нажатию на кнопку отправляется асинхронный ajax-запрос действиям SearchArtists, SearchAlbums и SearchTracks класса LastFMParser, который в свою очередь делает запрос на сайт last.fm с параметром page равным числу на единицу больше предыдущего, что значит, что должно быть возвращено следующие несколько записей. Во время отправления запроса и приёма ответа отображается колесо загрузки, которое демонстрирует тот факт, что страница не зависла и запрос выполняется.



**Рисунок 3.7 – Скрипт в файле Views\Home\Search.cshtml**

**3.3 Постановка задачи для нейросети**

Главное предназначение нейронной сети в этом приложении – это помощь пользователю в поиске исполнителей, которые должны ему понравится. Обычно для этого на главных страницах музыкальных сайтов располагаются чарты, т.е. списки наиболее популярных в определённый период музыкальных произведений или исполнителей. Они составляются путём всеобщего голосования. Это удобно, но чарты не учитывают особенностей конкретного пользователя, чарты одни для всех. Нейросеть должна помочь составлять индивидуальные чарты.

В самом начале создания нейронной сети нужно определиться с тем, какие данные будут подаваться на вход и что будет получаться на выходе.

На вход должны подаваться данные пользователя, которые могут каким-либо образом влиять на его музыкальные предпочтения. Такими данными являются возраст, пол, национальность, тип характера. Так же можно учесть профессию и языки, которыми владеет или хотя бы которые понимает пользователь.

Все входные данные должны быть числами, но из этого списка только возраст является числом. Остальные параметры нужно каким-то образом представить в виде чисел.

Пол легко можно представить в виде числа: можно допустить, что 1 означает мужской пол, а -1 – женский.

Тип характера также не сложно представить в виде числа: нужно всем типам присвоить целые числа от 1 до N, где N – количество типов.

Сложнее дело обстоит с национальностью, профессией и языками. Нужно придумать функцию, которая будет принимать на вход строку, означающую национальность, профессию или язык, а на выходе будет выдавать число в некотором диапазоне. Эта функция должна обладать таким свойством: чем ближе профессии, тем ближе значения функции. Для простоты можно взять некоторое количество самых популярных профессий, проинициализировать их начальными значениями, а менять их и добавлять новые по мере обучения нейросети.

На выходе должен получиться массив нейронов, где каждый нейрон связан с каким-то определённым исполнителем и его значение должно варьироваться от 0 до 1, где 0 означает, что данный исполнитель не должен понравиться пользователю, а 1, что, наоборот, должен понравится.

В самом начале можно выбрать 100 самых популярных исполнителей из любого чарта, а добавлять новые по мере обучения нейросети.

При регистрации на сайте пользователь обязан ввести данные, которые нужны для нейросети. Для удобства это можно сделать с помощью предоставления информации через социальные сети: они предоставляют такую возможность, а пользователю не придётся ничего вводить.

После регистрации пользователь сможет изучить тот индивидуальный чарт, который для него подсчитала нейросеть. Если его удовлетворит результат, то он может просто продолжить пользоваться сайтом. А если не удовлетворит, то он может скорректировать результаты сам, тем самым передать нейронной сети информацию о том, каким должен был быть результат. Такие индивидуальные чарты будут храниться в базе данных.

У пользователей с одинаковыми параметрами всё равно могут быть разные музыкальные вкусы, поэтому идеальным результатом будет считаться усреднённые значения позиции исполнителей в чартах для пользователей с одинаковым набором параметров.

Наилучшее количество скрытых слоёв, количество нейронов во всех слоях и скорость обучения неизвестны в начале обучения нейронной сети. Эти параметры должны подбираться эмпирически. Поэтому администратор должен иметь возможность влиять на них, а также на выбор функции активации, для того, чтобы ускорить обучение нейронной сети и повысить её эффективность.

**3.4 Создание индивидуальных чартов**

В приложении 2 приведён листинг класса начальной реализации нейронной сети. В нём, для простоты, только один скрытый слой.

Основными характеристиками нейросети являются количество нейронов на входном, скрытом и выходном слоях, а также скорость обучения. Они описаны в строках 3 – 7.

Далее, в соответствии с основными характеристиками, инициализируются матрицы весов и сдвигов (строки 8 - 16).

В качестве функции активации выбрана сигмоида (строка 60).

Чтобы получить значения выходных нейронов, нужно воспользоваться функциями, описанными в строках 24 – 52. Первая из них подсчитывает значения скрытых нейронов, а вторая на основе результатов первой функции – значения выходных нейронов.

Чтобы посчитать значение функции стоимости нейронной сети можно воспользоваться методом, описанным в строках 62 – 77. Этот метод проходит по всем элементам в верном наборе данных, для каждого считает его функцию стоимости конкретного элемента, а затем считает среднее арифметическое. Таким образом получается, что меньшее значение функции стоимости означает более высокую обученность системы.

И наконец, для обучения нейронной сети можно использовать функцию, описанную в строках 79 – 127. Эта функция некоторое количество раз, в данном случае тысячу, выполняет итерацию метода обратного распространения ошибки с помощью метода градиентного спуска.

Суть метода обратного распространения ошибки заключается в обратном ходе по нейронной сети и подсчёте значений, на которые должны изменяться значения активации нейронов, веса между слоями, а также сдвиги.

А суть метода градиентного спуска заключается в вычислении градиента какой-либо функции, в данном случае функции стоимости, и небольшом изменении каждой компоненты текущего вектора значений в направлении обратном градиенту, чтобы в конечном итоге оказаться в локальном минимуме функции.

Метод градиентного спуска использует параметр скорости обучения. Он, как уже упоминалось ранее, подбирается эмпирически. Если выбрать его слишком большим, например, 0.3, то метод не будет сходиться. А если слишком маленьким, например, 0.01, то метод будет сходиться, но очень медленно.

**ЗАКЛЮЧЕНИЕ**

В процессе выполнения дипломной работы получены следующие результаты: изучены книги и электронные источники по темам «Разработка web приложений ASP.NET Core MVC», «Архитектура клиент-серверных приложений», «Нейронные сети», а также непосредственно разработано приложение, использующее нейронные сети и совмещающее задачи и цели онлайн и оффлайн каталогизаторов.

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. Freeman A. Pro ASP.NET Core MVC 2 / A. Freeman – Apress, 2017
2. Galloway J. Professional ASP.NET Core 2.0 / J. Galloway, B. Wilson, K. S. Allen, D. Matson – Wrox, 2018
3. Магдануров Г. ASP.NET MVC Framework / Г. Магдануров, В. Юнев – БХВ, 2010
4. [Электронный ресурс – https://metanit.com/sharp/aspnet5/]
5. [Электронный ресурс – https://www.last.fm/api]
6. [Электронный ресурс – https://msdn.microsoft.com/ru-ru/]
7. [Электронный ресурс – https://proglib.io/p/neural-network-course/]

**ПРИЛОЖЕНИЕ 1**

LastFMParser.cs

1. **using** System;
2. **using** System.Collections.Generic;
3. **using** System.IO;
4. **using** System.Linq;
5. **using** System.Net;
6. **using** System.Text;
7. **using** System.Xml;
9. **namespace** Cataloguer.Models
10. {
11. //Documentation https://www.last.fm/ru/api
12. **public** **static** **class** LastFMParser
13. {
14. // Alternative key - 96d047d302a8707f3a7410873466dbfd
15. **private** **const** **string** СommonUrl = "http://ws.audioscrobbler.com/2.0/?api\_key=f39425750fc23d743fbf853d9585a46c&";
17. **private** **readonly** **static** **string** CounrtyForSearch = "belarus";
19. **public** **static** IEnumerable<Artist> GetTopArtists(**int** limit, **int** page = 1)
20. {
21. **string** url = $"{СommonUrl}method=geo.gettopartists&country={CounrtyForSearch}&page={page}&limit={limit}";
22. **foreach** (XmlNode artistNode **in** GetXmlDocumentFrom(url).SelectNodes("//artist"))
23. {
24. var artist = **new** Artist(artistNode.SelectSingleNode("name").InnerText);
25. artist.SetPictureLink(artistNode.SelectSingleNode("image[@size='large']").InnerText);
26. yield **return** artist;
27. }
28. }
30. **public** **static** Artist GetArtist(**string** name)
31. {
32. **string** url = $"{СommonUrl}method=artist.getinfo&artist={name}";
33. XmlNode artistInfoMainNode = GetXmlDocumentFrom(url).SelectSingleNode("//artist");
34. List<Album> albums = GetAlbumsOfArtist(name, 8).ToList();
35. List<Track> tracks = GetTracksOfArtist(name, 12).ToList();
36. List<Tag> tags = GetTopTagsFrom(artistInfoMainNode.SelectNodes("//tags/tag")).ToList();
37. var artist = **new** Artist(name) { Albums = albums, Tracks = tracks, Tags = tags };
38. artist.SetPictureLink(artistInfoMainNode.SelectSingleNode(".//image[@size='large']").InnerText);
39. artist.SetScrobbles(artistInfoMainNode.SelectSingleNode(".//stats/playcount").InnerText);
40. artist.SetListeners(artistInfoMainNode.SelectSingleNode(".//stats/listeners").InnerText);
41. artist.SetShortBiography(artistInfoMainNode.SelectSingleNode(".//bio/summary").InnerText);
42. **return** artist;
43. }
45. **public** **static** Artist GetArtistWithAllTracks(**string** name)
46. {
47. List<Track> tracks = GetTracksOfArtist(name, 48).ToList();
48. var artist = **new** Artist(name) { Tracks = tracks };
49. artist.SetPictureLink(GetPictureLinkOfArtist(name));
50. **return** artist;
51. }
53. **public** **static** IEnumerable<Track> GetTracksOfArtist(**string** name, **int** limit, **int** page = 1)
54. {
55. **string** url = $"{СommonUrl}method=artist.gettoptracks&artist={name}&page={page}&limit={limit}";
56. XmlDocument artistTracksDocument = GetXmlDocumentFrom(url);
57. var artist = **new** Artist(artistTracksDocument.SelectSingleNode("//toptracks").Attributes["artist"].Value);
58. **foreach** (XmlNode nodeWithTrack **in** artistTracksDocument.SelectNodes("//track"))
59. {
60. **string** trackName = nodeWithTrack.SelectSingleNode(".//name").InnerText;
61. **int** rank = Convert.ToInt32(nodeWithTrack.Attributes["rank"].Value);
62. var track = **new** Track(trackName) { Rank = rank , Artist = artist };
63. track.SetPictureLink(nodeWithTrack.SelectSingleNode(".//image[@size='large']").InnerText);
64. track.SetScrobbles(nodeWithTrack.SelectSingleNode(".//playcount").InnerText);
65. track.SetListeners(nodeWithTrack.SelectSingleNode(".//listeners").InnerText);
66. yield **return** track;
67. }
68. }
70. **public** **static** Artist GetArtistWithAllAlbums(**string** name)
71. {
72. List<Album> albums = GetAlbumsOfArtist(name, 48).ToList();
73. Artist artist = **new** Artist(name) { Albums = albums };
74. artist.SetPictureLink(GetPictureLinkOfArtist(name));
75. **return** artist;
76. }
78. **public** **static** IEnumerable<Album> GetAlbumsOfArtist(**string** name, **int** limit, **int** page = 1)
79. {
80. **string** url = $"{СommonUrl}method=artist.gettopalbums&artist={name}&page={page}&limit={limit}";
81. XmlDocument artistAlbumsDocument = GetXmlDocumentFrom(url);
82. var artist = **new** Artist(artistAlbumsDocument.SelectSingleNode("//topalbums").Attributes["artist"].Value);
83. **foreach** (XmlNode nodeWithTopAlbum **in** artistAlbumsDocument.SelectNodes("//album"))
84. {
85. **string** albumName = nodeWithTopAlbum.SelectSingleNode(".//name").InnerText;
86. var album = **new** Album(albumName) { Artist = artist};
87. album.SetPictureLink(nodeWithTopAlbum.SelectSingleNode(".//image[@size='large']").InnerText);
88. album.SetScrobbles(nodeWithTopAlbum.SelectSingleNode(".//playcount").InnerText);
89. yield **return** album;
90. }
91. }
93. **public** **static** **string** GetPictureLinkOfArtist(**string** name)
94. {
95. **string** url = $"{СommonUrl}method=artist.getinfo&artist={name}";
96. **return** GetXmlDocumentFrom(url).SelectSingleNode("//artist/image[@size='large']").InnerText;
97. }
99. **public** **static** Artist GetArtistWithBiography(**string** name)
100. {
101. **string** url = $"{СommonUrl}method=artist.getinfo&artist={name}";
102. XmlNode artistInfoMainNode = GetXmlDocumentFrom(url).SelectSingleNode("//artist");
103. var artist = **new** Artist(name);
104. artist.SetPictureLink(artistInfoMainNode.SelectSingleNode(".//image[@size='large']").InnerText);
105. artist.SetFullBiography(artistInfoMainNode.SelectSingleNode(".//bio/content").InnerText);
106. **return** artist;
107. }
108. **public** **static** Album GetAlbum(**string** albumName, **string** artistName)
109. {
110. **string** url = $"{СommonUrl}method=album.getinfo&artist={artistName}&album={albumName}";
111. XmlNode albumInfoMainNode = GetXmlDocumentFrom(url).SelectSingleNode("//album");
112. List<Tag> tags = GetTopTagsFrom(albumInfoMainNode.SelectNodes(".//tags/tag")).ToList();
113. var artist = **new** Artist(albumInfoMainNode.SelectSingleNode(".//artist").InnerText);
114. List<Track> tracks = GetTracksOfAlbumFrom(albumInfoMainNode.SelectNodes(".//tracks/track")).ToList();
115. **string** name = albumInfoMainNode.SelectSingleNode(".//name").InnerText;
116. var album = **new** Album(name) { Tags = tags, Artist = artist, Tracks = tracks };
117. album.SetPictureLink(albumInfoMainNode.SelectSingleNode(".//image[@size='large']").InnerText);
118. album.SetScrobbles(albumInfoMainNode.SelectSingleNode(".//playcount").InnerText);
119. album.SetListeners(albumInfoMainNode.SelectSingleNode(".//listeners").InnerText);
120. **return** album;
121. }
122. **private** **static** IEnumerable<Track> GetTracksOfAlbumFrom(XmlNodeList nodesWithTracks)
123. {
124. **foreach** (XmlNode nodeWithTrack **in** nodesWithTracks)
125. {
126. **string** name = nodeWithTrack.SelectSingleNode(".//name").InnerText;
127. **int** rank = Convert.ToInt32(nodeWithTrack.Attributes["rank"].Value);
128. var track = **new** Track(name) { Rank = rank };
129. track.SetDuration(nodeWithTrack.SelectSingleNode(".//duration").InnerText);
130. yield **return** track;
131. }
132. }
133. **public** **static** Track GetTrack(**string** searchedTrackName, **string** artistName)
134. {
135. **string** url = $"{СommonUrl}method=track.getInfo&artist={artistName}&track={searchedTrackName}";
136. XmlNode trackInfoMainNode = GetXmlDocumentFrom(url).SelectSingleNode("//track");
137. var artist = **new** Artist(trackInfoMainNode.SelectSingleNode(".//artist/name").InnerText);
138. Album album = GetAlbumOfTrackFrom(trackInfoMainNode, artist);
139. List<Tag> tags = GetTopTagsFrom(trackInfoMainNode.SelectNodes(".//toptags/tag")).ToList();
140. **string** name = trackInfoMainNode.SelectSingleNode(".//name").InnerText;
141. var track = **new** Track(name) { Artist = artist, Album = album, Tags = tags };
142. track.SetPictureLink(trackInfoMainNode.SelectSingleNode(".//album/image[@size='large']")?.InnerText ?? "");
143. track.SetDurationInMilliseconds(trackInfoMainNode.SelectSingleNode(".//duration").InnerText);
144. track.SetListeners(trackInfoMainNode.SelectSingleNode(".//listeners").InnerText);
145. track.SetScrobbles(trackInfoMainNode.SelectSingleNode(".//playcount").InnerText);
146. track.SetInfo(trackInfoMainNode.SelectSingleNode(".//wiki/summary")?.InnerText ?? "");
147. **return** track;
148. }
149. **private** **static** Album GetAlbumOfTrackFrom(XmlNode trackInfoMainNode, Artist artist)
150. {
151. **string** albumName = trackInfoMainNode.SelectSingleNode(".//album/title")?.InnerText ?? "";
152. Album album = **null**;
153. **if** (!String.IsNullOrWhiteSpace(albumName))
154. {
155. album = **new** Album(albumName) { Artist = artist};
156. album.SetPictureLink(trackInfoMainNode.SelectSingleNode(".//album/image[@size='large']").InnerText);
157. }
158. **return** album;
159. }
160. **private** **static** IEnumerable<Tag> GetTopTagsFrom(XmlNodeList nodesWithTags)
161. {
162. **foreach** (XmlNode nodeWithTag **in** nodesWithTags)
163. {
164. **string** tag = nodeWithTag.SelectSingleNode(".//name").InnerText;
165. yield **return** **new** Tag(tag);
166. }
167. }
168. **public** **static** IEnumerable<Artist> SearchArtists(**string** value, **int** limit, **int** page = 1)
169. {
170. **string** url = $"{СommonUrl}method=artist.search&artist={value}&page={page}&limit={limit}";
171. **foreach** (XmlNode artistNode **in** GetXmlDocumentFrom(url).SelectNodes("//artistmatches/artist"))
172. {
173. var artist = **new** Artist(artistNode.SelectSingleNode(".//name").InnerText);
174. artist.SetPictureLink(artistNode.SelectSingleNode(".//image[@size='large']").InnerText);
175. yield **return** artist;
176. }
177. }
178. **public** **static** IEnumerable<Album> SearchAlbums(**string** value,**int** limit, **int** page=1)
179. {
180. **string** url = $"{СommonUrl}method=album.search&album={value}&page={page}&limit={limit}";
181. **foreach** (XmlNode albumNode **in** GetXmlDocumentFrom(url).SelectNodes("//albummatches/album"))
182. {
183. var artist = **new** Artist(albumNode.SelectSingleNode(".//artist").InnerText);
184. **string** name = albumNode.SelectSingleNode(".//name").InnerText;
185. var album = **new** Album(name) { Artist = artist };
186. album.SetPictureLink(albumNode.SelectSingleNode(".//image[@size='large']").InnerText);
187. yield **return** album;
188. }
189. }
190. **public** **static** IEnumerable<Track> SearchTracks(**string** value,**int** limit, **int** page=1)
191. {
192. **string** url = $"{СommonUrl}method=track.search&track={value}&page={page}&limit={limit}";
193. **foreach**(XmlNode trackNode **in** GetXmlDocumentFrom(url).SelectNodes("//trackmatches/track"))
194. {
195. var artist=**new** Artist(trackNode.SelectSingleNode(".//artist").InnerText);
196. **string** name = trackNode.SelectSingleNode(".//name").InnerText;
197. var track = **new** Track(name) { Artist = artist };
198. track.SetPictureLink(trackNode.SelectSingleNode(".//image[@size='large']").InnerText);
199. track.SetListeners(trackNode.SelectSingleNode(".//listeners").InnerText);
200. yield **return** track;
201. }
202. }
203. **private** **static** XmlDocument GetXmlDocumentFrom(**string** url)
204. {
205. HttpWebRequest request = (HttpWebRequest)WebRequest.Create(url);
206. HttpWebResponse response = (HttpWebResponse)request.GetResponse();
207. **string** result = **new** StreamReader(response.GetResponseStream(), Encoding.UTF8).ReadToEnd();
208. XmlDocument document = **new** XmlDocument();
209. **using** (var stringReader = **new** StringReader(result))
210. {
211. **using** (var xmlTextReader = **new** XmlTextReader(stringReader) { Namespaces = **false** })
212. {
213. document.Load(xmlTextReader);
214. }
215. }
216. **return** document;
217. }
218. **public** **static** **string** GetPictureLinkOfAlbum(**string** albumName, **string** artistName)
219. {
220. **string** url = $"{СommonUrl}method=album.getinfo&artist={artistName}&album={albumName}";
221. **return** GetXmlDocumentFrom(url).SelectSingleNode("//album/image[@size='large']").InnerText;
222. }
223. }
224. }

**ПРИЛОЖЕНИЕ 2**

**NeuralNetwork.cs**

1. **using** System;
2. **using** System.Collections.Generic;
3. **using** System.Linq;
5. **namespace** Cataloguer.Models.NeuralNetwork
6. {
7. **public** **class** NeuralNetwork
8. {
9. #region Properties
11. **private** Repository Repository { **get**; **set**; }
13. /// <summary>
14. /// Learning rate
15. /// </summary>
16. **private** **readonly** **float** a;
17. **private** **readonly** **int** IterationsAmount;
18. **private** **readonly** **int** InputLayerLength = 7;
19. **private** **readonly** **int** HiddenLayerLength;
20. **private** **int** OutputLayerLength { **get**; **set**; }
21. **private** **float**[] InputLayer;
22. **private** **float**[] HiddenLayer;
23. **private** **float**[] ExpectedHidden;
24. **private** **float**[] OutputLayer;
25. **private** **float**[] ExpectedOutput;
26. **private** **float**[,] WeightsInputHidden;
27. **private** **float**[] BiasesHidden;
28. **private** **float**[,] WeightsHiddenOutput;
29. **private** **float**[] BiasesOutput;
30. **private** List<DatasetItem> Dataset { **get**; **set**; } = **new** List<DatasetItem>();
32. #endregion
34. **public** NeuralNetwork(Repository repository)
35. {
36. Repository = repository;
37. a = Convert.ToSingle(repository.GetConfigKeyValue("LearningRate"));
38. IterationsAmount = Convert.ToInt32(repository.GetConfigKeyValue("LearningIterations"));
39. HiddenLayerLength = Convert.ToInt32(repository.GetConfigKeyValue("HiddenLayerLength"));
40. OutputLayerLength = repository.GetTracksAmount();
41. InputLayer = **new** **float**[InputLayerLength];
42. HiddenLayer = **new** **float**[HiddenLayerLength];
43. ExpectedHidden = **new** **float**[HiddenLayerLength];
44. OutputLayer = **new** **float**[OutputLayerLength];
45. ExpectedOutput = **new** **float**[OutputLayerLength];
47. LoadBiases();
48. LoadWeights();
49. List<Rating> ratings = repository.GetRatings();
50. List<Track> tracks = repository.GetTracks();
51. LoadDataset(ratings, tracks);
52. }
54. **private** **void** LoadBiases()
55. {
56. BiasesHidden = **new** **float**[HiddenLayerLength];
57. BiasesOutput = **new** **float**[OutputLayerLength];
58. List<Bias> Biases = Repository.GetBiases();
59. **if** (Biases.Count == 0)
60. {
61. BiasesHidden.InitializeRandomVector();
62. BiasesOutput.InitializeRandomVector();
63. }
64. **else**
65. {
66. BiasesHidden = Biases.Where(b => b.Layer == 1).OrderBy(b => b.Number).Select(b => b.Value).ToArray();
67. BiasesOutput = Biases.Where(b => b.Layer == 2).OrderBy(b => b.Number).Select(b => b.Value).ToArray();
68. }
69. }
71. **private** **void** LoadWeights()
72. {
73. WeightsInputHidden = **new** **float**[InputLayerLength, HiddenLayerLength];
74. WeightsHiddenOutput = **new** **float**[HiddenLayerLength, OutputLayerLength];
75. List<Weight> Weights = Repository.GetWeights();
76. **if** (Weights.Count == 0)
77. {
78. WeightsInputHidden.InitializeRandomMatrix();
79. WeightsHiddenOutput.InitializeRandomMatrix();
80. }
81. **else**
82. {
83. List<Weight> weightsInputHidden = Weights.Where(w => w.FromLayer == 0).ToList();
84. **for** (**int** i = 0; i < InputLayerLength; i++)
85. {
86. **for** (**int** j = 0; j < HiddenLayerLength; j++)
87. {
88. WeightsInputHidden[i, j] = weightsInputHidden
89. .FirstOrDefault(w => w.FromNumber == i && w.ToNumber == j)?.Value ?? 0;
90. }
91. }
93. List<Weight> weightsHiddenOutput = Weights.Where(w => w.FromLayer == 1).ToList();
94. **for** (**int** i = 0; i < HiddenLayerLength; i++)
95. {
96. List<Weight> weightsFromI = weightsHiddenOutput.Where(w => w.FromNumber == i).ToList();
97. **for** (**int** j = 0; j < OutputLayerLength; j++)
98. {
99. WeightsHiddenOutput[i, j] = weightsFromI
100. .FirstOrDefault(w => w.ToNumber == j)?.Value ?? 0;
101. }
102. }
103. }
104. }
105. **private** **void** LoadDataset(List<Rating> ratings, List<Track> tracks)
106. {
107. IEnumerable<ApplicationUser> users = ratings.Select(r => r.ApplicationUser).Distinct();
108. **foreach** (ApplicationUser user **in** users)
109. {
110. List<Rating> userRatings = ratings.Where(r => r.ApplicationUser.Id == user.Id).ToList();
111. Dataset.Add(**new** DatasetItem(user,userRatings,tracks,OutputLayerLength));
112. }
113. }
114. **public** **void** CalculateHiddenLayer()
115. {
116. **for** (**int** j = 0; j < HiddenLayerLength; j++)
117. {
118. **float** result = 0;
119. **for** (**int** i = 0; i < InputLayerLength; i++)
120. {
121. result += InputLayer[i] \* WeightsInputHidden[i, j];
122. }
123. result += BiasesHidden[j];
124. HiddenLayer[j] = CalculateSigmoid(result);
125. }
126. }
127. **public** **void** CalculateOutputLayer()
128. {
129. **for** (**int** j = 0; j < OutputLayerLength; j++)
130. {
131. **float** result = 0;
132. **for** (**int** i = 0; i < HiddenLayerLength; i++)
133. {
134. result += HiddenLayer[i] \* WeightsHiddenOutput[i, j];
135. }
136. result += BiasesOutput[j];
137. OutputLayer[j] = CalculateSigmoid(result);
138. }
139. }
140. **public** List<KeyValuePair<Track, **float**>> GetAssumptiveRating(ApplicationUser user)
141. {
142. InputLayer = user.GetInputData();
143. CalculateHiddenLayer();
144. CalculateOutputLayer();
145. Dictionary<Track, **float**> assumptiveRating = **new** Dictionary<Track, **float**>();
146. List<Track> tracks = Repository.GetTracks().OrderBy(t => t.Id).ToList();
147. **for** (**int** i = 0; i < OutputLayerLength; i++)
148. {
149. assumptiveRating.Add(tracks[i], OutputLayer[i]);
150. }
151. **return** assumptiveRating.OrderByDescending(r=>r.Value).Take(20).ToList();
152. }
153. **private** **float** CalculateSigmoid(**float** x) => 1 / (**float**)(1 + Math.Exp(-x));
154. **public** **double** CalculateCostFunction()
155. {
156. **double** value = 0;
157. **foreach** (var datasetItem **in** Dataset)
158. {
159. InputLayer = datasetItem.InputData;
160. CalculateHiddenLayer();
161. CalculateOutputLayer();
162. ExpectedOutput = datasetItem.OutputData;
163. **for** (**int** i = 0; i < OutputLayerLength; i++)
164. {
165. value += Math.Pow(OutputLayer[i] - ExpectedOutput[i], 2);
166. }
167. }
168. **return** value / Dataset.Count;
169. }
170. **public** **void** Learn()
171. {
172. **for** (**int** iteration = 0; iteration < IterationsAmount; iteration++)
173. {
174. **foreach** (var datasetItem **in** Dataset)
175. {
176. InputLayer = datasetItem.InputData;
177. CalculateHiddenLayer();
178. CalculateOutputLayer();
179. ExpectedOutput = datasetItem.OutputData;
180. ExpectedHidden = Enumerable.Repeat(0.0f,HiddenLayerLength).ToArray();
181. **for** (**int** j = 0; j < OutputLayerLength; j++)
182. {
183. **float** currentOutputNeuron = OutputLayer[j];
184. **float** DcDa = 2 \* (currentOutputNeuron - ExpectedOutput[j]);
185. **float** DaDz = currentOutputNeuron\* (1 - currentOutputNeuron);
186. **for** (**int** i = 0; i < HiddenLayerLength; i++)
187. {
188. **float** DzDw = HiddenLayer[i];
189. **float** DcDw = DcDa \* DaDz \* DzDw;
190. WeightsHiddenOutput[i, j] -= MaxByModule(a \* DcDw);
191. **float** DcDb = DcDa \* DaDz;
192. BiasesOutput[j] -= MaxByModule(a \* DcDb);
193. **float** DzDaMinus1 = WeightsHiddenOutput[i, j];
194. **float** DcDaMinus1 = DcDa \* DaDz \* DzDaMinus1;
195. ExpectedHidden[i] -= MaxByModule(DcDaMinus1);
196. }
197. }
198. **for** (**int** j = 0; j < HiddenLayerLength; j++)
199. {
200. **float** currentHiddenNeuron = HiddenLayer[j];
201. **float** DcDa = -2 \* ExpectedHidden[j];
202. **float** DaDz = currentHiddenNeuron\* (1 - currentHiddenNeuron);
203. **for** (**int** i = 0; i < InputLayerLength; i++)
204. {
205. **float** DzDw = InputLayer[i];
206. **float** DcDw = DcDa \* DaDz \* DzDw;
207. WeightsInputHidden[i, j] -= MaxByModule(a \* DcDw);
208. **float** DcDb = DcDa \* DaDz;
209. BiasesHidden[j] -= MaxByModule(a \* DcDb);
210. }
211. }
212. }
213. }
214. }
215. **private** **float** MaxByModule(**float** x)
216. {
217. **float** maxModule = (**float**)Math.Max(Math.Abs(x), 0.00001);
218. **return** maxModule \* Math.Sign(x);
219. }
220. **public** **void** SaveAsync()
221. {
222. Repository.RemoveAllBiasesAndWeights();
223. List<Bias> biasesHidden = ConvertFromArray(BiasesHidden, 1).ToList();
224. List<Weight> weightsInputHidden = ConvertFromMatrix(WeightsInputHidden, 0).ToList();
225. Repository.AddBiasesAndWeights(biasesHidden, weightsInputHidden);
226. List<Bias> biasesOutput = ConvertFromArray(BiasesOutput, 2).ToList();
227. List<Weight> weightsHiddenOutput = ConvertFromMatrix(WeightsHiddenOutput, 1).ToList();
228. Repository.AddBiasesAndWeights(biasesOutput, weightsHiddenOutput);
229. }
230. **private** IEnumerable<Bias> ConvertFromArray(**float**[] biases, **byte** layer)
231. {
232. **for** (**int** i = 0; i < biases.Length; i++)
233. {
234. yield **return** **new** Bias(layer, i, biases[i]);
235. }
236. }
237. **private** IEnumerable<Weight> ConvertFromMatrix(**float**[,] weights, **byte** layer)
238. {
239. **int** n = weights.GetLength(0);
240. **int** m = weights.GetLength(1);
241. **for** (**int** i = 0; i < n; i++)
242. {
243. **for** (**int** j = 0; j < m; j++)
244. {
245. yield **return** **new** Weight(layer, i, j, weights[i, j]);
246. }
247. }
248. }
249. }
250. }