**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**

**ФАКУЛЬТЕТ ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ**

**Кафедра информационных систем управления**

**РАЗРАБОТКА ИНТЕЛЛЕКТУАЛЬНОГО КАТАЛОГИЗАТОРА**

**МУЗЫКАЛЬНЫХ ПРОИЗВЕДЕНИЙ**

Дипломная работа

Минальда Андрея Анатольевича

специальность «информатика»

Научный руководитель:

старший преподаватель

Гутников Сергей Евгеньевич

**СОДЕРЖАНИЕ**

**ВВЕДЕНИЕ5**

**ГЛАВА 1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ6**

1.1 Понятие каталогизатора музыки6

1.2 История развития аудионосителей и музыкальных каталогов 6

1.3 Оффлайн-каталогизаторы8

1.4 Онлайн-каталогизаторы9

1.5 Предпосылки к созданию приложения10

**ГЛАВА 2. ВЫБОР ТЕХНОЛОГИЙ ДЛЯ РЕШЕНИЯ ЗАДАЧИ11**

2.1 Выбор платформы и технологий для разработки11

2.2 Выбор источника данных 13

**ГЛАВА 3. РЕАЛИЗАЦИЯ ПРИЛОЖЕНИЯ16**

3.1 Создание онлайн части web приложения16

3.2 Создание оффлайн части web приложения18

3.3 Реализация поиска20

**ГЛАВА 4. СОЗДАНИЕ СПЕЦИАЛИЗИРОВАННОЙ НЕЙРОСЕТИ22**

4.1 Постановка задачи для нейросети22

4.2 Описание принципа работы23

4.3 Начальная реализация24

**ЗАКЛЮЧЕНИЕ26**

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ27**

**ПРИЛОЖЕНИЕ 128**

**ПРИЛОЖЕНИЕ 229**

**ПРИЛОЖЕНИЕ 330**

**ПРИЛОЖЕНИЕ 431**

**ПРИЛОЖЕНИЕ 532**

**ПРИЛОЖЕНИЕ 634**

**ПРИЛОЖЕНИЕ 737**

# 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**

* 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.

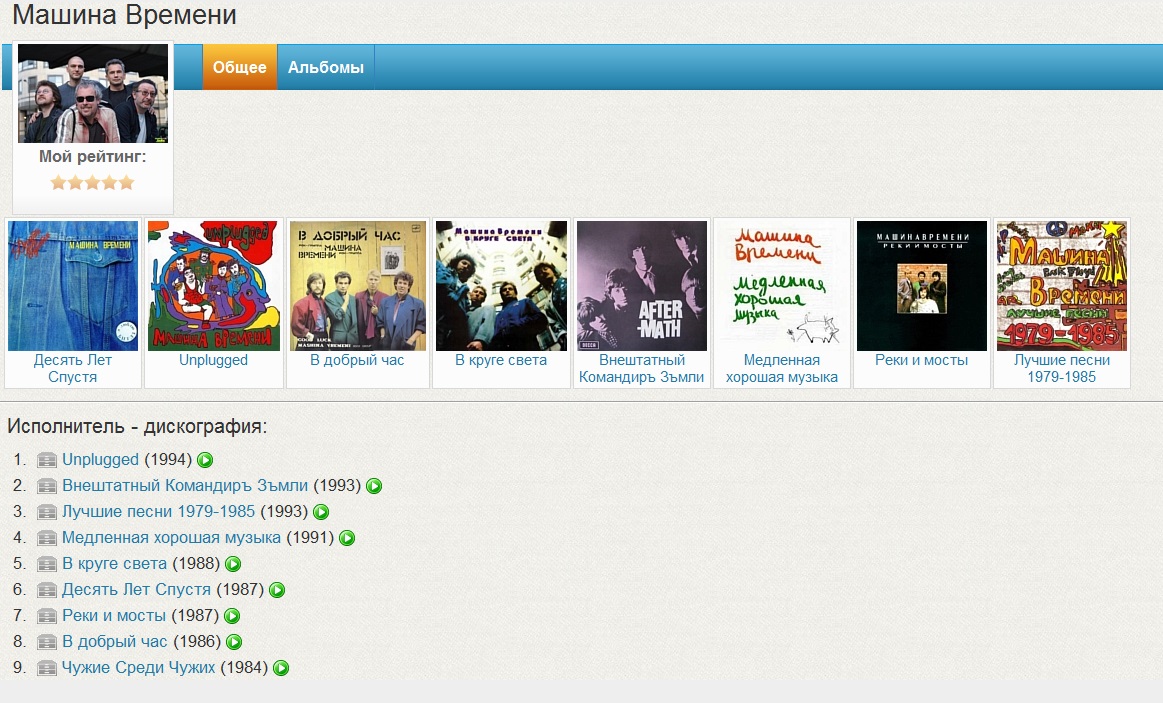


Рисунок 1 – Интерфейс программы Musicnizer

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.

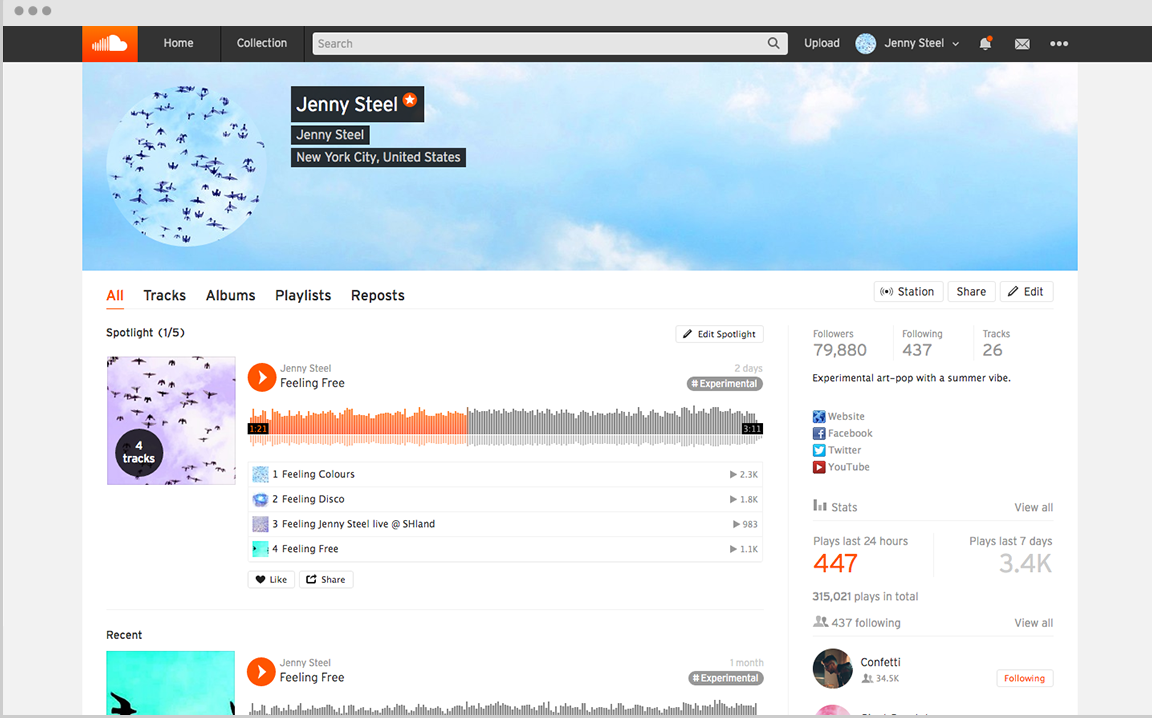


Рисунок 2 – Интерфейс сайта soundcloud.com

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. Предпосылки к созданию приложения

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 идея 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.

**ГЛАВА 2. ВЫБОР ТЕХНОЛОГИЙ ДЛЯ РЕШЕНИЯ ЗАДАЧИ**

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 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.

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

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

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.

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 3.

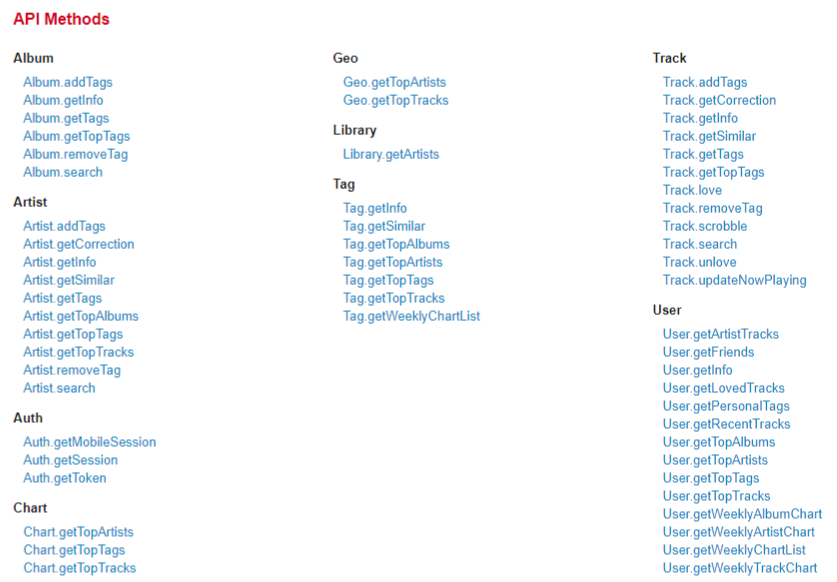


Рисунок 3 – Методы API сайта last.fm

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

1. http://ws.audioscrobbler.com/2.0/ - адрес API, после него по порядку пойдёт различные параметры;

2. ?api\_key=f39425750fc23d743fbf853d9585a46c – знак вопроса означает, что после него начинается перечисление всех параметров запроса, первый из которых, это ключ доступа;

3. method=artist.gettopalbums – непосредственно метод, к которому мы будем обращаться, в данном случае это метод получения самых популярных альбомов артиста;

4. artist=Coldplay – название музыкальной группы, коллектива, дуэта или имя и фамилия одиночного исполнителя;

5. 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 4.



Рисунок 4 – Результаты запроса

Реализация класса LastFMParser, который занимается обработкой ответов на запросы, приведена в приложении 1.

**ГЛАВА 3. РЕАЛИЗАЦИЯ ПРИЛОЖЕНИЯ**

3.1 Структура 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.

Рассмотрим эти папки и файлы подробнее.

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

1. 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.
2. 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.
3. lib – тут хранятся все сторонние библиотеки, которые используются в проекте. Например, в данном проекте используются библиотеки bootstrap, jquery, jquery-validation и jquery-validation-unobtrusive.
4. images – тут хранятся все картинки в различных разрешениях, таких как png, jpeg, gif и svg, которые будут использоваться в проекте.

3.3 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.

Все проделанные изменения содержатся в листинге 6.

Поиск будет выдавать результаты отдельно для онлайн и оффлайн баз, отдельно музыкантов, альбомы и песни. Интерфейс страницы поиска можно увидеть на рисунке 11.

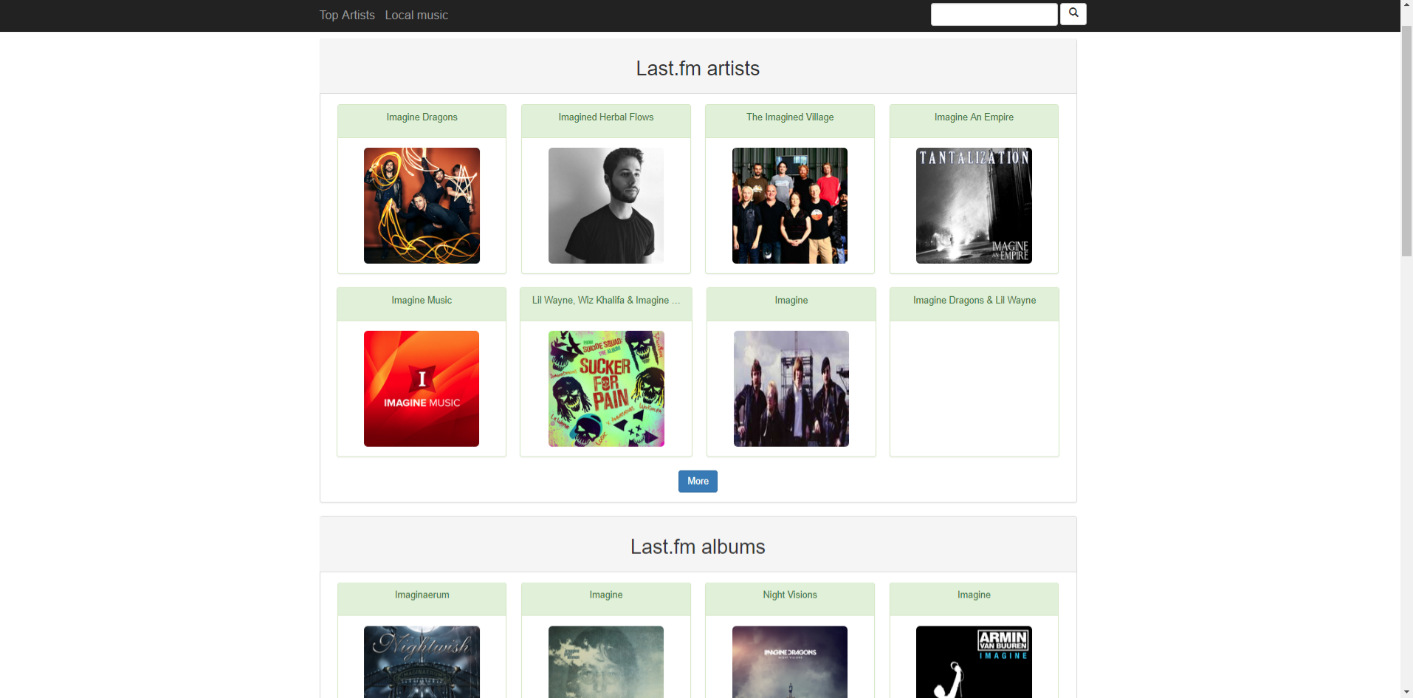


Рисунок 11 – Интерфейс страницы поиска

В конце файла Search.cshtml в приложении 6 написан 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 классу 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.

**ГЛАВА 4. СОЗДАНИЕ СПЕЦИАЛИЗИРОВАННОЙ НЕЙРОСЕТИ**

4.1 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.

4.2 Описание принципа работы

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.

4.3 Начальная реализация

В приложении 7 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 курсового проекта получены следующие результаты: изучены книги и электронные источники по темам «Разработка web приложений ASP.NET MVC 5», «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 MVC 5 / A. Freeman – Apress, 2013
2. Galloway J. Professional ASP.NET MVC 5 / J. Galloway, B. Wilson, K. S. Allen, D. Matson – Wrox, 2014
3. Магдануров Г. ASP.NET MVC Framework / Г. Магдануров, В. Юнев – БХВ, 2010
4. [Электронный ресурс – https://metanit.com/sharp/mvc5/]
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. **public** **class** LastFMParser
2. {
3. **public** **const** **string** СommonUrl = “http://ws.audioscrobbler.com/2.0/?api\_key=f39425750fc23d743fbf853d9585a46c&”;
5. **public** **string** CounrtyForSearch = «belarus»;
7. **public** List<Artist> GetTopArtists(**int** page, **int** limit)
8. {
9. **string** url = СommonUrl + “method=geo.gettopartists&country=” + CounrtyForSearch + “&page=” + page + “&limit=” + limit;
10. List<Artist> artists = **new** List<Artist>();
11. **foreach** (XmlNode artistNode **in** GetXmlDocumentFrom(url).SelectNodes(“//artist”))
12. {
13. Artist artist = **new** Artist(artistNode.SelectSingleNode(“name”).InnerText);
14. artist.SetPictureLink(artistNode.SelectSingleNode(“image[@size=’large’]”).InnerText);
15. artists.Add(artist);
16. }
18. **return** artists;
19. }
21. **public** Artist GetArtist(**string** name)
22. {
23. **string** url = СommonUrl + “method=artist.getinfo&artist=” + name;
24. XmlNode artistInfoMainNode = GetXmlDocumentFrom(url).SelectSingleNode(“//artist”);
25. Artist artist = **new** Artist(name)
26. {
27. Albums = GetAlbumsOfArtist(name, 1, 8),
28. Tracks = GetTracksOfArtist(name, 1, 12),
29. Tags = GetTopTagsFrom(artistInfoMainNode.SelectNodes(“//tags/tag”))
30. };
31. artist.SetPictureLink(artistInfoMainNode.SelectSingleNode(«.//image[@size='large']»).InnerText);
32. artist.SetScrobbles(artistInfoMainNode.SelectSingleNode(“.//stats/playcount”).InnerText);
33. artist.SetListeners(artistInfoMainNode.SelectSingleNode(“.//stats/listeners”).InnerText);
34. artist.SetShortBiography(artistInfoMainNode.SelectSingleNode(“.//bio/summary”).InnerText);
35. **return** artist;
36. }
38. **public** Album GetAlbum(**string** albumName, **string** artistName)
39. {
40. **string** url = СommonUrl + “method=album.getinfo&artist=” + artistName + “&album=” + albumName;
41. XmlNode albumInfoMainNode = GetXmlDocumentFrom(url).SelectSingleNode(“//album”);
42. Album album = **new** Album(albumInfoMainNode.SelectSingleNode(“.//name”).InnerText)
43. {
44. Tags = GetTopTagsFrom(albumInfoMainNode.SelectNodes(“.//tags/tag”)),
45. Artist =**new** Artist(albumInfoMainNode.SelectSingleNode(“.//artist”).InnerText),
46. Tracks =GetTracksOfAlbumFrom(albumInfoMainNode.SelectNodes(“.//tracks/track”))
47. };
49. album.SetPictureLink(albumInfoMainNode.SelectSingleNode(«.//image[@size='large']»).InnerText);
50. album.SetScrobbles(albumInfoMainNode.SelectSingleNode(“.//playcount”).InnerText);
51. album.SetListeners(albumInfoMainNode.SelectSingleNode(“.//listeners”).InnerText);
52. **return** album;
53. }
54. **public** Track GetTrack(**string** trackName, **string** artistName)
55. {
56. **string** url = СommonUrl + “method=track.getInfo&artist=” + artistName + “&track=” + trackName;
57. XmlNode trackInfoMainNode = GetXmlDocumentFrom(url).SelectSingleNode(“//track”);
58. Artist artist = **new** Artist(trackInfoMainNode.SelectSingleNode(“.//artist/name”).InnerText);
59. Album album = GetAlbumOfTrackFrom(trackInfoMainNode, artist);
60. Track track = **new** Track(trackInfoMainNode.SelectSingleNode(“.//name”).InnerText)
61. {
62. Artist = artist,
63. Album = album,
64. Tags = GetTopTagsFrom(trackInfoMainNode.SelectNodes(“.//toptags/tag”))
65. };
66. track.SetPictureLink(trackInfoMainNode.SelectSingleNode(«.//album/image[@size='large']»)?.InnerText ?? «»);
67. track.SetDurationInMilliseconds(trackInfoMainNode.SelectSingleNode(“.//duration”).InnerText);
68. track.SetListeners(trackInfoMainNode.SelectSingleNode(“.//listeners”).InnerText);
69. track.SetScrobbles(trackInfoMainNode.SelectSingleNode(“.//playcount”).InnerText);
70. track.SetInfo(trackInfoMainNode.SelectSingleNode(“.//wiki/summary”)?.InnerText ?? «»);
72. **return** track;
73. }
74. }

**ПРИЛОЖЕНИЕ 2**

HomeController.cs

1. **public** **class** HomeController : Controller
2. {
3. MusicRepository database = **new** MusicRepository();
4. LastFMParser parser = **new** LastFMParser();
5. **public** **const** **int** FirstPage = 1;
6. **public** **int** artistsPerPage = 48;
7. **public** **int** albumsPerPage = 48;
8. **public** **int** tracksPerPage = 48;
9. **public** **int** newSearchElements = 8;
11. **public** ActionResult Index()
12. {
13. List<Artist> artists = parser.GetTopArtists(1, artistsPerPage);
14. **return** View(artists);
15. }
17. **public** ActionResult TopArtists(**int** page)
18. {
19. List<Artist> artists = parser.GetTopArtists(page, artistsPerPage);
20. **return** PartialView(«PartialArtists», artists);
21. }
23. **public** ActionResult ArtistProfile(**string** name)
24. {
25. Artist artist = parser.GetArtist(name);
26. **return** View(artist);
27. }
29. **public** ActionResult ArtistBiography(**string** name)
30. {
31. Artist artist = parser.GetArtistWithBiography(name);
32. **return** View(artist);
33. }
35. **public** ActionResult ArtistAllTracks(**string** name)
36. {
37. Artist artist = parser.GetArtistWithAllTracks(name);
38. **return** View(artist);
39. }
41. **public** ActionResult ArtistTracks(**string** name, **int** page)
42. {
43. List<Track> tracks = parser.GetTracksOfArtist(name, page, tracksPerPage);
44. **return** PartialView(«PartialTracksInPanels», tracks);
45. }
47. **public** ActionResult ArtistAllAlbums(**string** name)
48. {
49. Artist artist = parser.GetArtistWithAllAlbums(name);
50. **return** View(artist);
51. }
53. **public** ActionResult ArtistAlbums(**string** name, **int** page)
54. {
55. List<Album> albums = parser.GetAlbumsOfArtist(name, page, albumsPerPage);
56. **return** PartialView(«PartialAlbums», albums);
57. }
59. **public** ActionResult Album(**string** albumName, **string** artistName)
60. {
61. Album album = parser.GetAlbum(albumName, artistName);
62. **return** View(album);
63. }
65. **public** ActionResult Track(**string** trackName, **string** artistName)
66. {
67. Track track = parser.GetTrack(trackName, artistName);
68. **return** View(track);
69. }
70. }

**ПРИЛОЖЕНИЕ 3**

Album.cs

1. **public** **class** Album
2. {
3. **public** **int** Id { **get**; **set**; }
4. **public** **string** Name { **get**; **set**; }
5. [Column]
6. **private** **string** PictureLink { **get**; **set**; }
7. [Column]
8. **private** **string** Scrobbles { **get**; **set**; }
9. [Column]
10. **private** **string** Listeners { **get**; **set**; }
11. **public** **virtual** List<Track> Tracks { **get**; **set**; }
12. **public** **virtual** List<**string**> Tags { **get**; **set**; }
13. **public** **virtual** Artist Artist { **get**; **set**; }
14. **public** Album() { }
16. **public** Album(**string** name)
17. {
18. Name = name;
19. Tracks = **new** List<Track>();
20. Tags = **new** List<**string**>();
21. }
22. }

**ПРИЛОЖЕНИЕ 4**

ArtistProfile.cshtml

1. @**using** Cataloguer.Models
2. @model Artist
3. @{
4. ViewBag.Title = Model.Name;
5. }
7. <div **class**=»row»>
8. <div **class**=”col-sm-4 col-md-4 col-lg-4”>
9. <img width=”100%” **class**=”img-thumbnail” src=”@Model.GetPictureLink()” />
10. </div>
11. <div **class**=”col-sm-8 col-md-8 col-lg-8”>
12. <h1 **class**=”text-center”>@Model.Name</h1>
13. @Html.Partial(“PartialTags”, Model.Tags)
14. <h4>Scrobbles : @Model.GetScrobbles()</h4>
15. <h4>Listeners : @Model.GetListeners()</h4>
16. @**if** (Model.GetShortBiography() != « «)
17. {
18. <p>@Model.GetShortBiography()</p>
19. <p>@Html.ActionLink(«See more…», «ArtistBiography», «Home», **new** { name = Model.Name }, **new** { })</p>
20. }
21. </div>
22. </div>
23. <h2 **class**=”text-center”>Top tracks</h2>
24. <div **class**=»flex-container»>
25. @Html.Partial(«PartialTracksInPanels», Model.Tracks)
26. </div>
27. @**if** (Model.Tracks.Count > 9)
28. {
29. @Html.ActionLink(«More tracks…», «ArtistAllTracks», «Home», **new** { name = Model.Name }, **new** { })
30. }
31. <h2 **class**=”text-center”>Top albums</h2>
32. <div **class**=»flex-container»>
33. @Html.Partial(«PartialAlbums», Model.Albums)
34. </div>
35. @**if** (Model.Albums.Count > 7)
36. {
37. @Html.ActionLink(«More albums…», «ArtistAllAlbums», «Home», **new** { name = Model.Name }, **new** { })
38. }

PartialTracksInPanels.cshtml

1. @**using** Cataloguer.Models
2. @model List<Track>
3. @**foreach** (Track track **in** Model)
4. {
5. <a **class**=»panel panel-success» href=»/Home/Track?trackName=@track.Name&artistName=@track.Artist.Name»>
6. <div **class**=»panel-heading»><p **class**=»text-center»>@track.Name</p></div>
7. <div **class**=»panel-body»>
8. <img **class**=»img-rounded center-block» src=»@track.GetPictureLink()» />
9. </div>
10. <div **class**=»panel-footer»>
11. <span **class**=»glyphicon glyphicon-headphones»></span>    @track.GetListeners()
12. </div>
13. </a>
14. }

PartialAlbums.cshtml

1. @**using** Cataloguer.Models
2. @model List<Album>
4. @**foreach** (Album album **in** Model)
5. {
6. <a **class**="panel panel-success" href="/Home/Album?albumName=@album.Name&artistName=@album.Artist.Name">
7. <div **class**="panel-heading">
8. <p **class**="text-center">@album.Name</p>
9. </div>
10. <div **class**="panel-body">
11. <img **class**="img-rounded center-block" src="@album.GetPictureLink()" />
12. </div>
13. @**if** (album.GetScrobbles() != **null**)
14. {
15. <div **class**="panel-footer">
16. <p>
17. <span **class**="glyphicon glyphicon-headphones"></span>    @album.GetScrobbles()
18. </p>
19. </div>
20. }
21. </a>
22. }

**ПРИЛОЖЕНИЕ 5**

MusicContext.cs

1. **public** **class** MusicContext : DbContext
2. {
3. **public** DbSet<Artist> Artists { **get**; **set**; }
5. **public** DbSet<Album> Albums { **get**; **set**; }
7. **public** DbSet<Track> Tracks { **get**; **set**; }
9. **protected** **override** **void** OnModelCreating(DbModelBuilder modelBuilder)
10. {
11. modelBuilder.Conventions.Add(**new** NonPublicColumnAttributeConvention());
12. }
13. }
15. **public** **sealed** **class** NonPublicColumnAttributeConvention : Convention
16. {
17. **public** NonPublicColumnAttributeConvention()
18. {
19. Types().Having(NonPublicProperties).Configure((config, properties) =>
20. {
21. **foreach** (PropertyInfo prop **in** properties)
22. {
23. config.Property(prop);
24. }
25. });
26. }
28. **private** IEnumerable<PropertyInfo> NonPublicProperties(Type type)
29. {
30. var matchingProperties = type.
31. GetProperties(BindingFlags.SetProperty | BindingFlags.GetProperty | BindingFlags.NonPublic | BindingFlags.Instance)
32. .Where(propInfo => propInfo.GetCustomAttributes(**typeof**(ColumnAttribute), **true**).Length > 0).ToArray();
33. **return** matchingProperties.Length == 0 ? **null** : matchingProperties;
34. }
35. }
37. **public** **class** MusicRepository
38. {
39. **private** MusicContext db = **new** MusicContext();
41. **public** List<Artist> GetArtists() => db.Artists.ToList();
43. **public** List<Artist> GetArtistsByName(**string** name) => db.Artists.Where(a => a.Name == name).ToList();
45. **public** List<Album> GetAlbums() => db.Albums.ToList();
47. **public** List<Album> GetAlbumsByName(**string** name) => db.Albums.Where(a => a.Name == name).ToList();
49. **public** List<Track> GetTracks() => db.Tracks.ToList();
51. **public** List<Track> GetTracksByName(**string** name) => db.Tracks.Where(a => a.Name == name).ToList();
53. **public** Artist GetArtist(**string** name) => db.Artists.First(a => a.Name == name);
55. **public** **void** UpdateArtist(Artist artist)
56. {
57. db.Entry(artist).State = EntityState.Modified;
58. }
60. **public** Album GetAlbum(**string** albumName, **string** artistName)
61. {
62. **return** db.Artists.First(a => a.Name == artistName).Albums.First(a => a.Name == albumName);
63. }
65. **public** **void** UpdateAlbum(Album album)
66. {
67. db.Entry(album).State = EntityState.Modified;
68. }
70. **public** **void** AddArtist(Artist artist)
71. {
72. db.Artists.Add(artist);
73. }
75. **public** **void** AddAlbum(Album album)
76. {
77. db.Albums.Add(album);
78. }
80. **public** **void** AddAlbumToArtist(Album album, **string** artistName)
81. {
82. db.Artists.First(a => a.Name == artistName).Albums.Add(album);
83. }
85. **public** **void** AddTrackToAlbumOfArtist(Track track, **string** albumName, **string** artistName)
86. {
87. db.Artists.First(a => a.Name == artistName).Albums.First(a => a.Name == albumName).Tracks.Add(track);
88. }
90. **public** **void** AddTrackToArtist(Track track, **string** artistName)
91. {
92. db.Artists.First(a => a.Name == artistName).Tracks.Add(track);
93. }
95. **public** **bool** ArtistExists(**string** name) => db.Artists.Any(a => a.Name == name);
97. **public** **bool** AlbumExists(**string** albumName, **string** artistName) =>
98. db.Albums.Any(a => a.Name == albumName && a.Artist.Name == artistName);
100. **public** **bool** TrackExists(**string** trackName, **string** artistName) =>
101. db.Tracks.Any(t => t.Name == trackName && t.Artist.Name == artistName);
103. **public** **bool** TrackExists(**string** trackName, **string** albumName, **string** artistName) =>
104. db.Tracks.Any(t => t.Name == trackName && t.Album.Name == albumName && t.Album.Artist.Name == artistName);
106. **public** **void** Save()
107. {
108. db.SaveChanges();
109. }
110. }

**ПРИЛОЖЕНИЕ 6**

HomeController.cs

1. [HttpGet]
2. **public** ActionResult Search(**string** value)
3. {
4. SearchingResults results = **new** SearchingResults
5. {
6. LastFMArtists = parser.SearchArtists(value, FirstPage, newSearchElements),
7. LastFMAlbums = parser.SearchAlbums(value, FirstPage, newSearchElements),
8. LastFMTracks = parser.SearchTracks(value, FirstPage, newSearchElements),
9. LocalArtists = database.GetArtistsByName(value),
10. LocalAlbums = database.GetAlbumsByName(value),
11. LocalTracks = database.GetTracksByName(value)
12. };
13. ViewBag.SearchingValue = value;
14. **return** View(results);
15. }
17. **public** ActionResult SearchArtists(**string** value, **int** page)
18. {
19. List<Artist> artists = parser.SearchArtists(value, page, newSearchElements);
20. **return** PartialView("PartialArtists", artists);
21. }
23. **public** ActionResult SearchAlbums(**string** value, **int** page)
24. {
25. List<Album> albums = parser.SearchAlbums(value, page, newSearchElements);
26. **return** PartialView("PartialAlbums", albums);
27. }
29. **public** ActionResult SearchTracks(**string** value, **int** page)
30. {
31. List<Track> tracks = parser.SearchTracks(value, page, newSearchElements);
32. **return** PartialView("PartialTracksInPanels", tracks);
33. }

LastFMParser.cs

1. **public** List<Artist> SearchArtists(**string** value, **int** page, **int** limit)
2. {
3. **string** url = СommonUrl + "method=artist.search&artist=" + value + "&page=" + page + "&limit=" + limit;
4. List<Artist> artists = **new** List<Artist>();
5. **foreach** (XmlNode artistNode **in** GetXmlDocumentFrom(url).SelectNodes("//artistmatches/artist"))
6. {
7. Artist artist = **new** Artist(artistNode.SelectSingleNode(".//name").InnerText);
8. artist.SetPictureLink(artistNode.SelectSingleNode(".//image[@size='large']").InnerText);
9. artists.Add(artist);
10. }
12. **return** artists;
13. }
15. **public** List<Album> SearchAlbums(**string** value, **int** page, **int** limit)
16. {
17. **string** url = СommonUrl + "method=album.search&album=" + value + "&page=" + page + "&limit=" + limit;
18. List<Album> albums = **new** List<Album>();
19. **foreach** (XmlNode albumNode **in** GetXmlDocumentFrom(url).SelectNodes("//albummatches/album"))
20. {
21. Album album = **new** Album(albumNode.SelectSingleNode(".//name").InnerText)
22. {
23. Artist = **new** Artist(albumNode.SelectSingleNode(".//artist").InnerText)
24. };
25. album.SetPictureLink(albumNode.SelectSingleNode(".//image[@size='large']").InnerText);
26. albums.Add(album);
27. }
29. **return** albums;
30. }
32. **public** List<Track> SearchTracks(**string** value, **int** page, **int** limit)
33. {
34. **string** url = СommonUrl + "method=track.search&track=" + value + "&page=" + page + "&limit=" + limit;
35. List<Track> tracks = **new** List<Track>();
36. **foreach**(XmlNode trackNode **in** GetXmlDocumentFrom(url).SelectNodes("//trackmatches/track"))
37. {
38. Track track = **new** Track(trackNode.SelectSingleNode(".//name").InnerText)
39. {
40. Artist = **new** Artist(trackNode.SelectSingleNode(".//artist").InnerText)
41. };
42. track.SetPictureLink(trackNode.SelectSingleNode(".//image[@size='large']").InnerText);
43. track.SetListeners(trackNode.SelectSingleNode(".//listeners").InnerText);
44. tracks.Add(track);
45. }
47. **return** tracks;
48. }

SearchingResults.cs

1. **public** **class** SearchingResults
2. {
3. **public** List<Artist> LastFMArtists { **get**; **set**; }
4. **public** List<Album> LastFMAlbums { **get**; **set**; }
5. **public** List<Track> LastFMTracks { **get**; **set**; }
6. **public** List<Artist> LocalArtists { **get**; **set**; }
7. **public** List<Album> LocalAlbums { **get**; **set**; }
8. **public** List<Track> LocalTracks { **get**; **set**; }
10. **public** SearchingResults() { }
11. }

Search.cshtml

1. @**using** Cataloguer.Models
2. @model SearchingResults
3. @{
4. ViewBag.Title = ViewBag.SearchingValue;
5. List<MusicObject> musicObjects = **new** List<MusicObject>();
6. }
8. <h1 **class**="text-center">@ViewBag.SearchingValue</h1>
9. @{
10. **foreach**(Artist artist **in** Model.LastFMArtists)
11. {
12. MusicObject musicObject = **new** MusicObject
13. {
14. Name = artist.Name,
15. PageLink = "/Home/ArtistProfile?name=" + artist.Name,
16. PictureLink = artist.GetPictureLink()
17. };
18. musicObjects.Add(musicObject);
19. }
20. }
21. @Html.Partial("PartialSearchView", **new** ViewObject
22. {
23. Heading = "Last.fm artists",
24. NoElementsMessage = "There are no artists in last.fm with this name",
25. ContainerId = "artists",
26. PanelCssClass = "panel panel-success",
27. MusicObjects = musicObjects,
28. ButtonId = "addArtists"
29. })
30. <script>
31. var artistPage = 2;
32. $('#addArtists').click(function () {
33. LoadNew('/Home/SearchArtists', artistPage, '#artists');
34. artistPage++;
35. });
36. var albumsPage = 2;
37. $('#addAlbums').click(function () {
38. LoadNew('/Home/SearchAlbums', albumsPage, '#albums');
39. albumsPage++;
40. });
41. var tracksPage = 2;
42. $('#addTracks').click(function () {
43. LoadNew('/Home/SearchTracks', tracksPage, '#tracks');
44. tracksPage++;
45. });
46. function LoadNew(url, page, blockId) {
47. $.ajax({
48. url: url,
49. data: {
50. value: '@ViewBag.SearchingValue',
51. page: page
52. },
53. success: function (data) {
54. $(blockId).append(data);
55. }
56. });
57. };
58. </script>

**ПРИЛОЖЕНИЕ 7**

**NeuralNetwork.cs**

1. **public** **class** NeuralNetwork
2. {
3. **public** **static** **double** a = 0.05;
4. **public** **static** **int** InputLayerLength = 10;
5. **public** **static** **int** HiddenLayerLength = 12;
6. **public** **static** **int** OutputLayerLength = 100;
7. **public** **int**[] InputLayer = **new** **int**[InputLayerLength];
8. **public** **double**[] HiddenLayer = **new** **double**[HiddenLayerLength];
9. **public** **double**[] ExpectedHidden = **new** **double**[HiddenLayerLength];
10. **public** **double**[] OutputLayer = **new** **double**[OutputLayerLength];
11. **public** **double**[] ExpectedOutput = **new** **double**[OutputLayerLength];
12. **public** **double**[,] WeightsInputHidden = **new** **double**[InputLayerLength, HiddenLayerLength];
13. **public** **double**[] BiasesHidden = **new** **double**[HiddenLayerLength];
14. **public** **double**[,] WeightsHiddenOutput = **new** **double**[HiddenLayerLength, OutputLayerLength];
15. **public** **double**[] BiasesOutput = **new** **double**[OutputLayerLength];
17. **public** NeuralNetwork()
18. {
19. LoadWeightsAndBiases();
20. LoadDataset();
21. }
23. **public** **void** CalculateHiddenLayer()
24. {
25. **for** (**int** j = 0; j < HiddenLayerLength; j++)
26. {
27. **double** result = 0;
28. **for** (**int** i = 0; i < InputLayerLength; i++)
29. {
30. result += InputLayer[i] \* WeightsInputHidden[i, j];
31. }
33. result += BiasesHidden[j];
34. HiddenLayer[j] = CalculateSigmoid(result);
35. }
36. }
38. **public** **void** CalculateOutputLayer()
39. {
40. **for** (**int** j = 0; j < OutputLayerLength; j++)
41. {
42. **double** result = 0;
43. **for** (**int** i = 0; i < HiddenLayerLength; i++)
44. {
45. result += HiddenLayer[i] \* WeightsHiddenOutput[i, j];
46. }
48. result += BiasesOutput[j];
49. OutputLayer[j] = CalculateSigmoid(result);
50. }
51. }
53. **public** **int** GetAssumptiveResult()
54. {
55. **double** maximum = OutputLayer.Max();
56. **return** Array.FindIndex(OutputLayer, x => x == maximum);
57. }
59. **private** **double** CalculateSigmoid(**double** x) => 1 / (1 + Math.Exp(-x));
61. **public** **double** CalculateCostFunction()
62. {
63. **double** value = 0;
64. **foreach** (var dataItem **in** Dataset)
65. {
66. InputLayer = InitializeInputLayer();
67. CalculateHiddenLayer();
68. CalculateOutputLayer();
69. InitializeExpectedResults();
70. **for** (**int** i = 0; i < OutputLayerLength; i++)
71. {
72. value += Math.Pow(OutputLayer[i] - ExpectedOutput[i], 2);
73. }
74. }
75. **return** value / Dataset.Count;
76. }
78. **public** **void** Learn()
79. {
80. **int** iterationsAmount = 1000;
81. **for** (**int** iteration = 0; iteration < iterationsAmount; iteration++)
82. {
83. **foreach** (var dataItem **in** Dataset)
84. {
85. InputLayer = InitializeInputLayer();
86. CalculateHiddenLayer();
87. CalculateOutputLayer();
88. InitializeExpectedResults();
89. ExpectedHidden = Enumerable.Repeat(0.0, HiddenLayerLength).ToArray();
90. **for** (**int** j = 0; j < OutputLayerLength; j++)
91. {
92. **double** currentOutputNeuron = OutputLayer[j];
93. **double** DcDa = 2 \* (currentOutputNeuron - ExpectedOutput[j]);
94. **double** DaDz = currentOutputNeuron \* (1 - currentOutputNeuron);
95. **for** (**int** i = 0; i < HiddenLayerLength; i++)
96. {
97. **double** DzDw = HiddenLayer[i];
98. **double** DcDw = DcDa \* DaDz \* DzDw;
99. WeightsHiddenOutput[i, j] -= a \* DcDw;
101. **double** DcDb = DcDa \* DaDz;
102. BiasesOutput[j] -= a \* DcDb;
104. **double** DzDaMinus1 = WeightsHiddenOutput[i, j];
105. **double** DcDaMinus1 = DcDa \* DaDz \* DzDaMinus1;
106. ExpectedHidden[i] -= DcDaMinus1;
107. }
108. }
109. **for** (**int** j = 0; j < HiddenLayerLength; j++)
110. {
111. **double** currentHiddenNeuron = HiddenLayer[j];
112. **double** DcDa = -2 \* ExpectedHidden[j];
113. **double** DaDz = currentHiddenNeuron \* (1 - currentHiddenNeuron);
114. **for** (**int** i = 0; i < InputLayerLength; i++)
115. {
116. **double** DzDw = InputLayer[i];
117. **double** DcDw = DcDa \* DaDz \* DzDw;
118. WeightsInputHidden[i, j] -= a \* DcDw;
119. **double** DcDb = DcDa \* DaDz;
120. BiasesHidden[j] -= a \* DcDb;
121. }
122. }
123. }
124. }
125. }
126. }