**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра MO ЭВМ**

индиВИДУАЛЬНОЕ ДОМАШНЕЕ ЗАДАНИЕ

**по дисциплине «Разработка программного обеспечения информационных систем»**

Тема: Поиск подходящих цитат из песен

|  |  |  |
| --- | --- | --- |
| Студент гр. 5381 |  | Ильиных И.С. |
| Студент гр. 5381 |  | Скиба А.С. |
| Студент гр. 5382 |  | Шахов А.Д. |
| Преподаватель |  | Заславский М.М. |

Санкт-Петербург

2018

**ЗАДАНИЕ**

**на индивидуальное домашнее задание**

|  |  |  |
| --- | --- | --- |
| Студенты Ильиных И.С. Скиба А.С. Шахов А.Д. | | |
| Группы 5381, 5382 | | |
| Тема проекта: Поиск подходящих цитат из песен | | |
| Исходные данные:  Проект должен быть разработан с использованием базы данных MongoDB | | |
| Содержание пояснительной записки:  Содержание, Введение, Качественные требования к решению, Сценарии использования, Модель данных, Разработанное прилоежение, Заключение, Список использованных источников. | | |
| Предполагаемый объем пояснительной записки:  Не менее 15 страниц. | | |
| Дата выдачи задания: 13.09.2018 . | | |
| Дата сдачи реферата: \_\_\_\_\_\_\_\_\_\_\_\_ | | |
| Дата защиты реферата: \_\_\_\_\_\_\_\_\_\_ | | |
| Студент |  | Ильиных И.С. |
| Студент |  | Скиба А.С. |
| Студентка |  | Шахов А.Д. |
| Преподаватель |  | Заславский М.М. |

**Аннотация**

В индивидуальном домашнем задании реализован веб-сервис на основе СУБД MongoDB, с помощью которого можно найти возможные рифмы из текстов песен, к введённому слову.

**Summary**

In the individual homework implemented a web service based on MongoDB DBMS, with which you can find possible rhymes from the lyrics to the entered word.

**содержание**

[Введение 5](#_Toc533016710)

[1. Качественные требования к решению 6](#_Toc533016711)

[2. Сценарии использования 7](#_Toc533016712)

[2.1. Сценарии использования для задачи импорта, представления, анализа и экспорта данных 7](#_Toc533016713)

[2.2. Вывод 8](#_Toc533016714)

[3. Модель данных 9](#_Toc533016715)

[3.1. Описание структуры 9](#_Toc533016716)

[3.2. Нереляционная модель данных 9](#_Toc533016717)

[3.3. Аналог модели данных для SQL СУБД 9](#_Toc533016718)

[3.4. Запросы 11](#_Toc533016719)

[3.5. Выводы 13](#_Toc533016720)

[4. Разработанное приложение 14](#_Toc533016721)

[4.1. Краткое описание 14](#_Toc533016722)

[4.2. Использованные технологии 14](#_Toc533016723)

[4.3. Ссылки на Приложение 15](#_Toc533016724)

[Список использованных источников 17](#_Toc533016725)

[Приложение А. Документация по сборке и развертыванию приложения 18](#_Toc533016726)

[Приложение B. Инструкция для пользователя 19](#_Toc533016727)

[Приложение С. Снимки экрана приложения 20](#_Toc533016728)

# Введение

У многих начинающих поэтов, существует потребность в поиске рифм, в современном мире множество сервисов позволяют это сделать. Но тогда встаёт вопрос, как узнать, насколько уникально то, что ты придумал, где и кто уже использовал такие рифмы?

Целью проекта является разработка приложения, с помощью которого можно найти рифмы к заданному слову из текстов песен загруженных в БД.

В проекте разработано веб-приложение на основе СУБД MongoDB.

# 1. Качественные требования к решению

Необходимо разработать веб-приложение, позволяющее находить рифмы в текстах песен к заданному слову.

Основные функции:

* Поиск рифм по заданному слову, сочетанию слов;
* Отображение общей статистики по всем найденным рифмам;
* Добавление новых песен;
* Импорт и экспорт данных.

# 2. Сценарии использования

## 2.1. Сценарии использования для задачи импорта, представления, анализа и экспорта данных

## Произвести поиcк рифм по слову

Пользователь имеет возможность произвести поиск рифм по слову, результатом поиска будет сама песня, также некоторые статистики (сколько раз повторялось слово в песне, сколько там рифм было вообще, сколько среднее кол-во раз повторялись слова, сколько разных слов), также будет статистика по всем словам и песням, она будет показывать похожие метрики, только они будут собраны по всем песням.

*Порядок действий*:

1. Пользователь заходит на сайт
2. Далее он вводит слово, рифмы к которому он хочет найти вводит в input
3. Далее при нажатии на button на сервер отправляется запрос
4. Пользователю выдается список песен с выделенными в нем рифмами и статистикой

## Добавить песню

Пользователь имеет возможность добавлять песню в базу данных, чтобы по ней также проводился поиск.

*Порядок действий*:

1. Пользователь заходи на сайт
2. Переходит на страницу добавления песни
3. Вводит информацию о песне в некоторый input
4. Отправляет песню на сервер нажатием на button, где она сохраняется, и пользователь видит подтверждение успеха, либо неудачи

## Экспортировать песни

Пользователь может экспортировать всю БД по которой осуществляется поиск

*Порядок действий*:

1. Пользователь заходи на сайт
2. Нажимает кнопку экспорта данных
3. Получает файл .txt

## Посмотреть статистику

Пользователь может посмотреть статистику для данного запроса по всей БД

*Порядок действий*:

1. Пользователь заходи на сайт
2. Осуществляет поиск рифм
3. Нажимает кнопку глобальной статистики
4. Открывается окно со статистикой по всей БД

## 2.2. Вывод

Для решения преобладают операции чтения, так как для Пользователя реализован поиск рифм, то есть поиск и вывод данных, а добавление новых песен – скорее дополнительная функция.

# 3. Модель данных

## 3.1. Описание структуры

## Данные хранятся в MongoDB, как 2 коллекции: english & russian - их структура идентична, так что рассматривать их отдельно не имеет смысла, разница в том, что алгоритмы обрабатывают их по-разному. Для русского языка используется небольшое преобразование символов на созвучные, затем удаляются согласные и рифма смотрится по последней гласной, в английском за преобразование на схожие звуки отвечает известный алгоритм metaphone2, затем сверяются 2 последние звука.

## Будем считать, что песни на 2 языках идентичны по потреблению памяти.

## *Почему все так как есть?*

## Большая часть логики у нас заключена не в самой СУБД, а в сервере непосредственно, т.к. СУБД не имеет возможности как-то искать рифмы к словам, есть идея построить индекс рифм. Это с одной стороны хорошая идея, индекс всегда ускорит поиск, но мы не можем предсказать по какому слову пользователю понадобятся рифмы, поэтому нам придется считать с запасом. Для русского языка все более или менее понятно, надо для всех гласных найти рифмы в песне, сохранить их рядом с песней и получится огромное кол-во данных, с каждой песней, это не очень хороший подход. Для английского все еще сложней, т.к. мы смотрим на последние 2 символа, что делает эту задачу еще более плачевной, т.к. комбинаций расстановки этих 2 символов еще больше, чем гласных в русском языке, поэтому мы решили вынести всю логику в сервер и считать статистику на нем же.

## 3.2. Нереляционная модель данных

Для определения размера документа условимся, что у нас используются отлько ASCII-символы, чтобы было проще считать место, занимаемое полем документа, т.е. один символ будет занимать 1-байт.

Документ:

1. Идентификатор - поле, которое автоматически генерируется MongoDB, его размер 12-байт, судя по [документации](https://docs.mongodb.com/v3.0/reference/bson-types/#objectid).
2. Название песни - будем считать название песни размером в 20 символов, т.е. его ориентировочный размер будет 20-байт.
3. Название исполнителя - также будет считать как 20 символов, т.е. его размер будет около 20-байт, также как и название песни.
4. Текст песни - кол-во слов в песне лежит где-то между 100-300 слов, зависит от жанра и многих других условий, примем его равным 300, т.е. размер песни будет составлять 300-байт.

Пример JSON:

{

"artist": "...",

"text": "...",

"title": "...",

"\_id": 123

}

Приняв во внимание то, что написано выше, мы получим, что средний размер документа: sizeof(id) + sizeof(title) + sizeof(artist) + sizeof(text) = 12 + 20 + 20 + 300 = 352, следовательно размер одного документа будет равен 352-байта. Что довольно мало, это хорошо.

## 3.3. Аналог модели данных для SQL СУБД

## Нам понадобилось бы несколько сущностей: Artist, Song, также надо учесть, что исполнители могут писать одну песню вместе, т.е. между этими сущностями будет отношение many-to-many, т.е. составим таблицы:

CREATE TABLE Artist (

id SERIAL,

name VARCHAR(20) NOT NULL,

CONSTRAINT PK\_ARTIST PRIMARY KEY (id)

);

CREATE TABLE Song (

id SERIAL,

name VARCHAR(20) NOT NULL,

text VARCHAR(1000) NOT NULL,

CONSTRAINT PK\_SONG PRIMARY\_KEY (id)

)

CREATE TABLE SongToArtist (

song\_id INTEGER NOT NULL REFERENCES Song,

artist\_id INTEGER NOT NULL REFERENCES Artist,

)

Можно было сделать структуру оптимизированную для нашей задачи, но мы сделали её такой, какой требует этого рациональный смысл, а не глупое желание оптимизировать все подряд.

## 3.4. Запросы

*Поиск*:

* NoSQL

collection.find({})

* SQL

SELECT song\_id, name, text, name FROM (

SELECT \* FROM SongTOArtist

JOIN Song ON Song.id == sond\_id

JOIN ARtist ON Artist.id == artist\_id

);

Тут видно, что в MongoDB мы делаем это за O(n), а в SQL будет O(n + (n / k)log(n / k)) ~ O(nlog(n))

*Обновление*:

* NoSQL

collection.update\_one({

'\_id': id

}, {

'$set': song

})

* SQL

UPDATE TABLE Song SET text = $song WHERE id == $id;

В двух случаях оценка сложности O(n), т.к. нет JOIN.

*Удаление*:

* NoSQl

collection.delete\_one({

'\_id': id

})

* SQL

DELETE TABLE Song WHERE id == $id;

В двух случаях оценка сложности O(n), т.к. нету JOIN.

Для сравнения допустим, что у нас 1000 песен и 20 исполнителей, т.е. у исполнителя примерно 50 песен, это значит, что n = 1000 , а k = 20.

## 3.5. Выводы

##### Память

Размеры данных:

* NoSQL

n \* 352 = 1000 \* 352 = 352000

* SQL

n \* (20 + 300 + 8) + k \* (20 + 8) + n \* (8 + 8) = n \* 344 + k \* 28 = 1000 \* 328 + 20 \* 28 + 1000 \* 16 = 344560

Даже здесь SQL уже выигрывает, но это довольно плотные данные.

Из таблиц видно, что в случае SQL, мы бы потребляли больше памяти в случае однородных данных, т.е. если бы у нас было много одиниковых исполнителей и их песен, то мы бы выйграли по памяти, н оу нас такое не гарантируется, данные у нас будут разнообразные, т.к. песни добавляются спонтанно, значит случай NoSQL нам подходит больше, но нельзя сказать, что он лучше.

* NoSQL:0,  
  SQL: 1

##### Скорость

##### Поиск у нас происходит за 1 команду, а в случае SQL, нам бы пришлось сделать JOIN, что, как известно является не самой быстрой операцией, т.к. внутри используется сортировка ключей, обычно она работает за nlog(n). А наше решение работает за линию, что в любом случае будет быстрее, т.к. выигрыш по кол-ву записей в таблице Song будет на константу, сравнивая с нашей коллекцией, а сложность будет возрастать не на константу, а на логарифм, с увеличением кол-ва песен, также там поиск будет проводиться по всем песням, следовательно сложность будет (n+ m)log(n + m),а у нас n что показывает выигрыш NoSQL базы данных.

Тут также считаем, что видно еще только на просмотре сложностей операций.

* NoSQL:1  
  SQL: 1

##### Удобство

##### В NoSQL мыс сможем не закреплять структуру нашего документа, что поможет сохранять нам в БД некоторую мета-информацию и проще расширяться, можно будет делать некйи кэш, прямо в mongodb, также мыможем захотеть еще хранить альбомы, в SQL это сильно устранит структуру, а у нас просто добавится одно поле.

* NoSQL: 2,  
  SQL: 1

# 4. Разработанное приложение

## 4.1. Краткое описание

Разработанное приложение осуществляет поиск рифм в текстах песен. Приложение состоит из страниц:

1. Главная страница. На данной странице Пользователь может задать данные для поиска.
2. Страница для добавления новых песен в БД

## 4.2. Использованные технологии

При написании приложения использовались следующие технологии:

* Python
* NodeJS
* Redis
* Flask
* MongoDB;
* HTML;
* CSS;
* Java Script.

## 4.3. Ссылки на Приложение

Исходный код приложения и инструкция по установке находятся по ссылке:

https://github.com/moevm/nosql2018-song\_quotes Заключение

Разработано приложение для поиска рифм к заданному слову в текстах песен с использованием базы данных MongoDB. Приложение работает корректно, но имеет широкие возможности по улучшению алгоритма подбора рифм, как для русского языка, так и для английского.

# Список использованных источников

1. MongoDB Documentation

URL: https://docs.mongodb.com/ (дата обращения: 19.12.2018).

1. Flask Documentation

URL: http://flask.pocoo.org/docs/1.0/ (дата обращения: 19.12.2018).

1. NodeJS Documentation

URL: https://nodejs.org/en/docs/ (дата обращения: 19.12.2018).

# Приложение А. Документация по сборке и развертыванию приложения

Инструкция по сборке и запуску:

1. Скачать проект из репозитория.
2. Проверить наличие на компьютере технологий:
   1. Python
   2. Flask
   3. Redis
   4. NodeJS
   5. NodeJS-express (*npm install*)
   6. MongoDB
3. Запустить back-end при помощи python-flask, например, при помощи IDE PyCharm
4. Запустить front-end при помощи NodeJS (*node app*)
5. Запустить Redis
6. Запустить MongoDB
7. Перейти в браузере по адресу: <http://localhost:3000/>
8. Импортировать песни

# Приложение B. Инструкция для пользователя

**Главный экран**

При входе на сайт отображается главная страница, на которой пользователь может экспортировать БД, посмотреть статистику и осуществить поиск

Для выполнения поиска Пользователь должен ввести слово, к которому он хочет подобрать рифму и нажать кнопку «Search»

**Страница добавления песен**

Пользователь вводит исполнителя, название песни и её текст. Если Пользователь не знает текста песни, то нажатием кнопки «Search text» можно осуществить его поиск. Когда все обязательные поля заполнены можно осуществить добавление песни к БД.

# Приложение С. Снимки экрана приложения

На рисунках 1-4 изображены снимки экрана приложения.

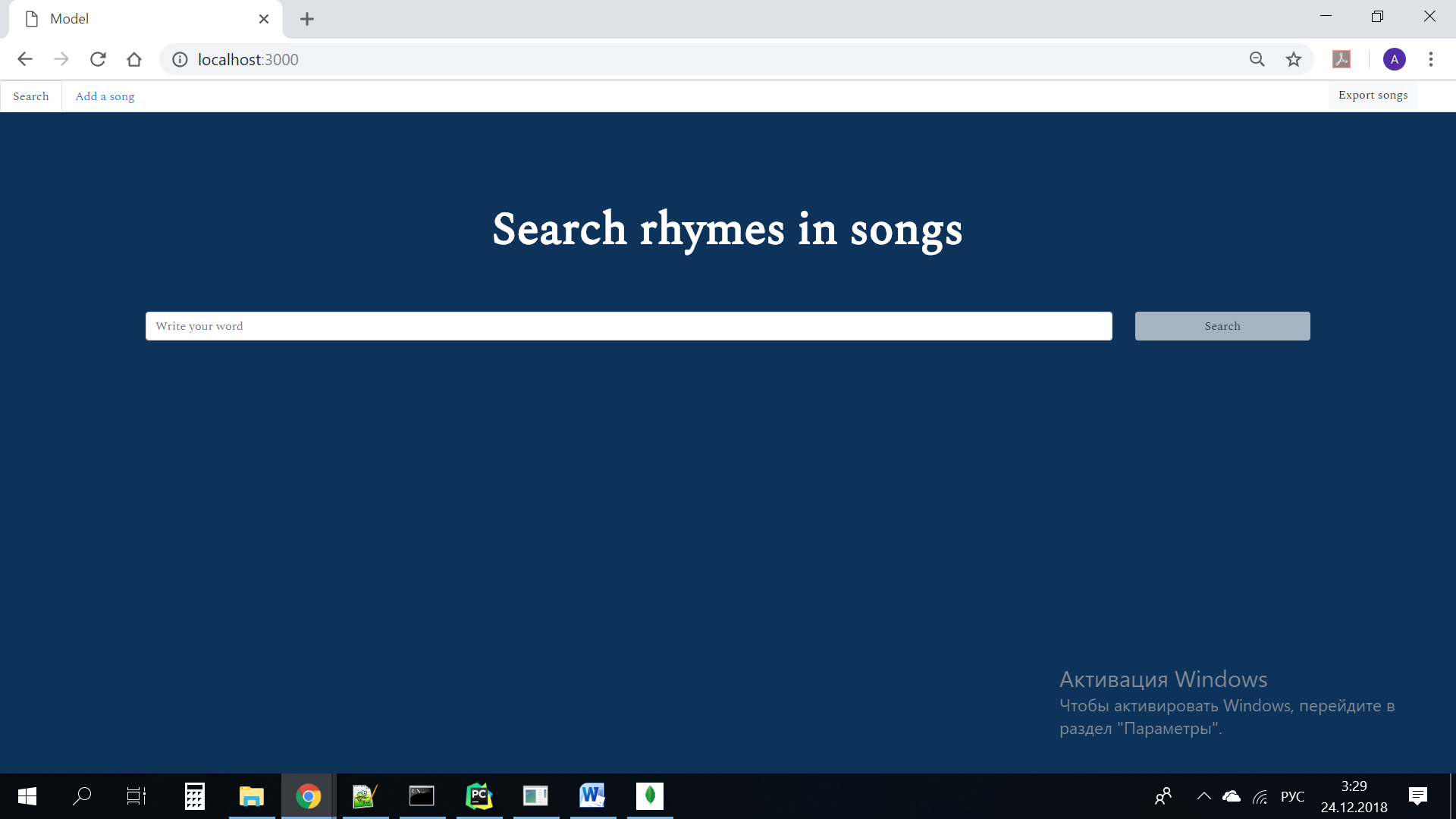


Рисунок 1 – Главная страница

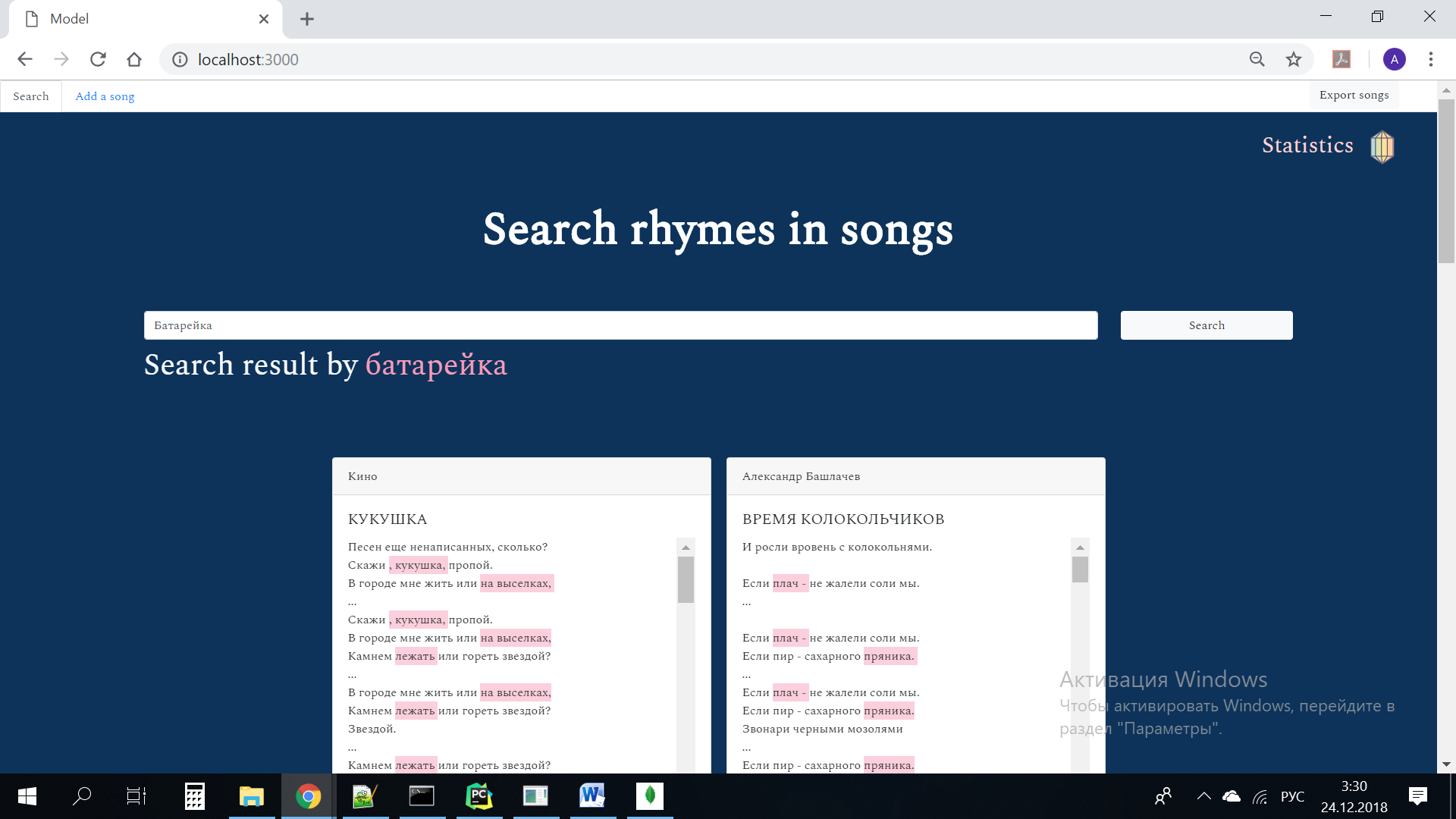


Рисунок 2 – Страница с отображением найденных рифм

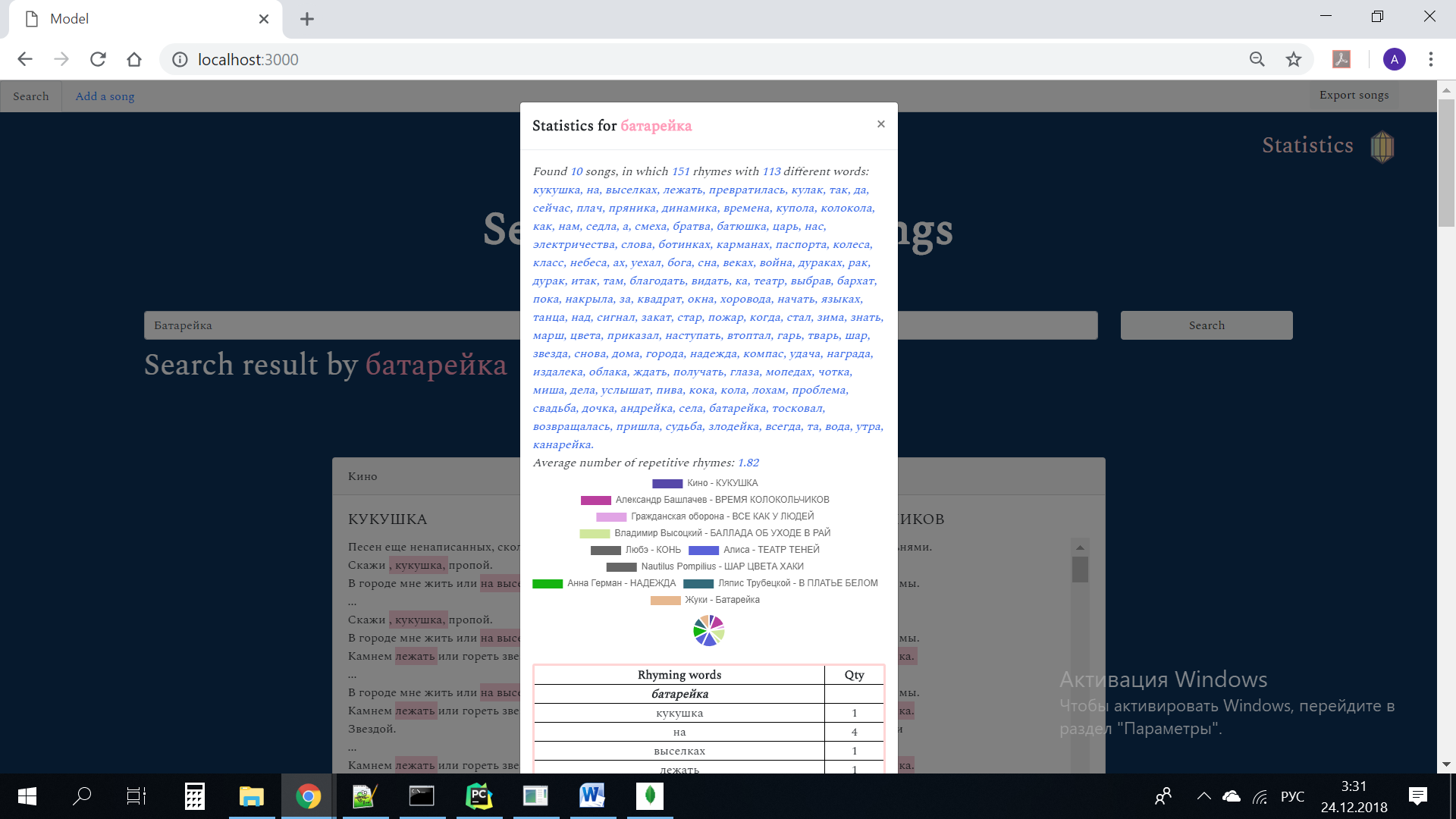


Рисунок 3 – Страница с отображением общей статистики

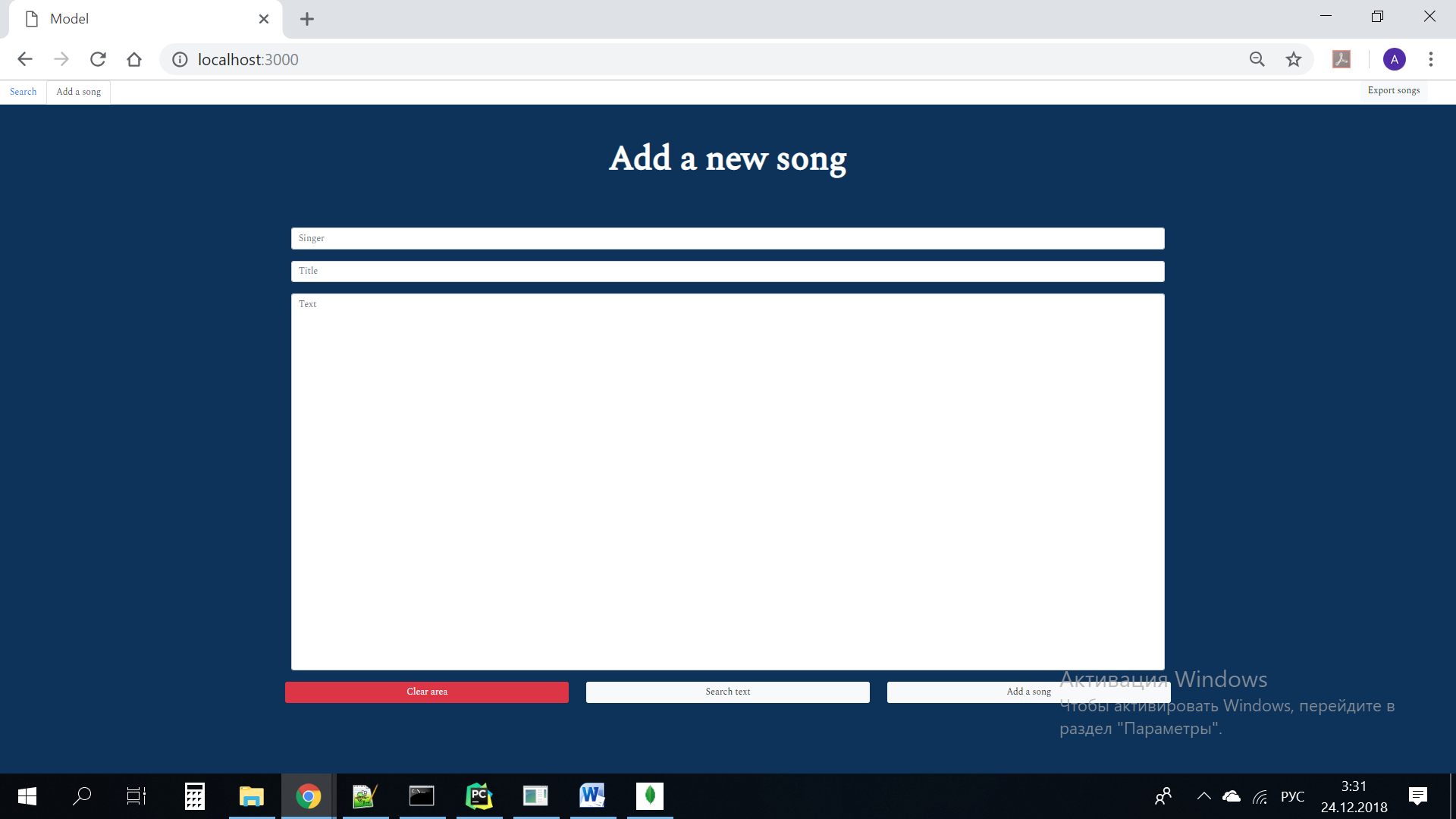


Рисунок 4 – Страница с добавлением новой песни