Design Document

of

Domain Specific Information Retrieval System

(Rhythm App)

1. Aim:

The aim is to build a search engine which will accept a query in a given domain from the user and return the ten most relevant search results.

1. Background:

The domain chosen was Music and dataset contains the attributes Artist name, Song name and Lyrics. The searching module is built on Vector space model and uses tf.idf score for ranked retrieval.

1. Language and tools:

The assignment is coded in python and uses nltk, numpy, math, pandas, timeit, tabulate, pickle and flask. Google Collaboratory was used to develop the module.

1. Solution architecture:

Top 10 search result

Methods for pre-processing

Pre-processed data sent to build index

Dataset

User query

IR model

Input

Output

1. IR model description:

The IR model is defined by the class IR\_model which is defined as :

Class data members of class:

1. word\_dict: Datatype : dictionary. Maps each unique words in the corpus to an index.
2. word\_dict\_size: Datatype : Integer. Contains the size of word\_dict.
3. doc\_list\_size: Datatype : Integer. Number of documents in corpus.
4. score\_matrix: Datatype : 2D integer array. Contains tf.idf score of each word corpus.

Class methods:

1. addDocument : arguments : corpus ; No return type. Adds corpus to the matrix and store term frequency (tf) in cells.
2. build\_Vector\_Space : arguments : corpus ; No return type. Calculates tf.idf score for all terms in corpus and builds the index.
3. Search : arguments : user query(list of terms), corpus ; Returns a list of tuples containing tf.idf score of each song in corpus wrt query.

Other methods:

1. tokenize\_sentence: arguments : string. Performs tokenization using TreebankwordTokenizer and returns a list of strings without punctuations.

2. remove\_stopwords : arguments : List of strings. Returns a list of strings after removing stopwords.

3. stem\_words : arguments : List of strings. Using PorterStemmer() returns a list of strings after stemming.

4. preprocess\_sentence : arguments : string. Calls the above methods for pre-processing.

6. Working of the model:

The strings in corpus, lyrics in this case are pre-processed and appended with their row number. The IR\_model first build a dictionary for all terms in the corpus and calls addDocument for each item in the corpus and fills the score\_matrix with tf (The term frequency tf of term t in document d is defined as the number of times that t occurs in d). Then df (df is the number of documents that contain term t) is calculated for all terms and using df, idf (Inverted document frequency) is calculated using the given formula



Then, tf.idf weights are filled in the score\_matrix according to

.

The query q is pre-processed and Search() method is called using an object of the class IR\_model with parameters as q and corpus. Now, Search() method computes tf.idf score according to



between each item in corpus and query. Then, it returns the list query\_score with has the item’s row number and corresponding tf.idf score. Finally, top ten search results with highest tf.idf scores are printed.

7. Running time:

1. For pre-processing the corpus : 12.80527 s

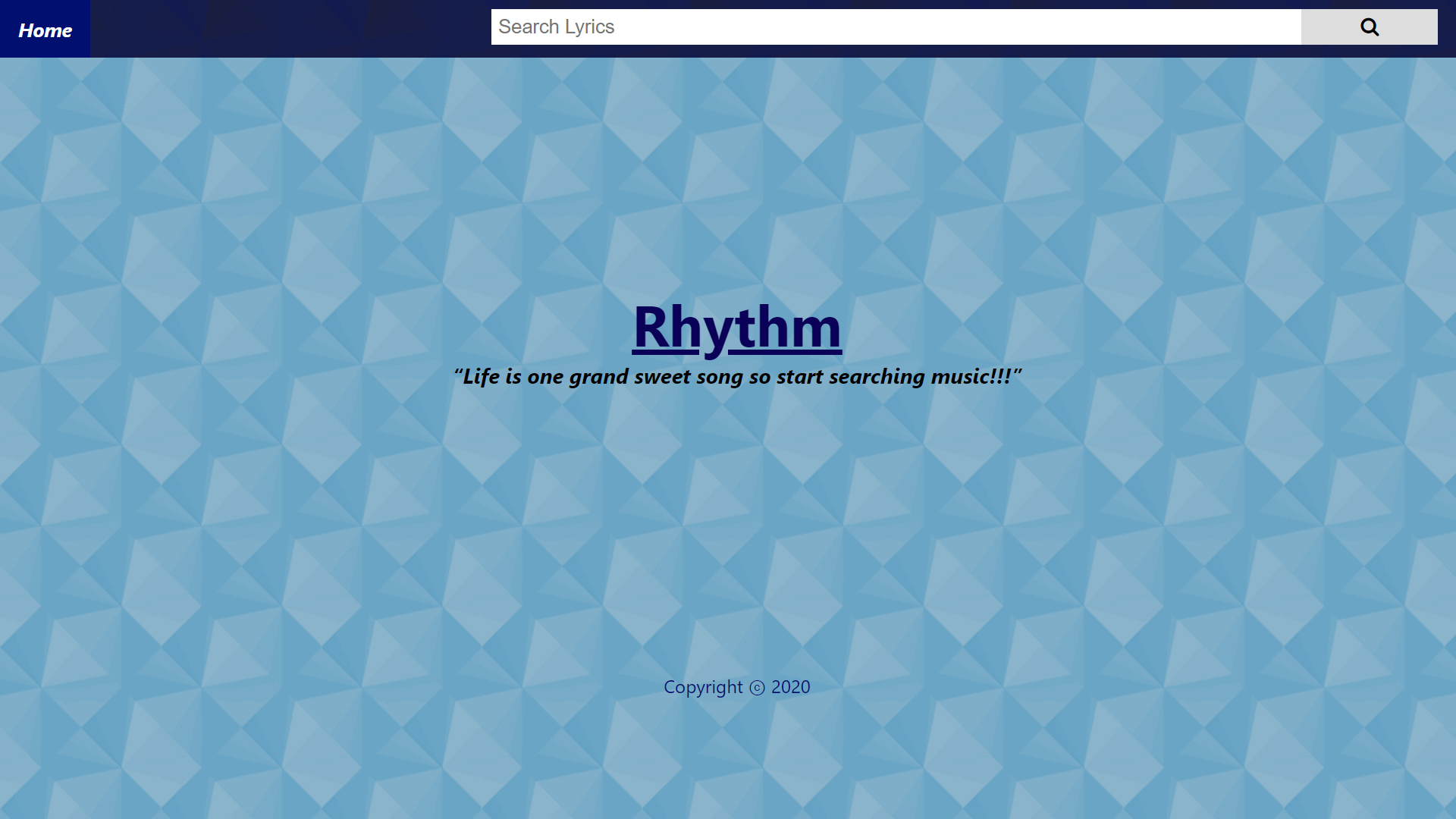
2. For pre-processing the user query : 0.00154 s

3. For building the index in vector space model : 139.96311 s

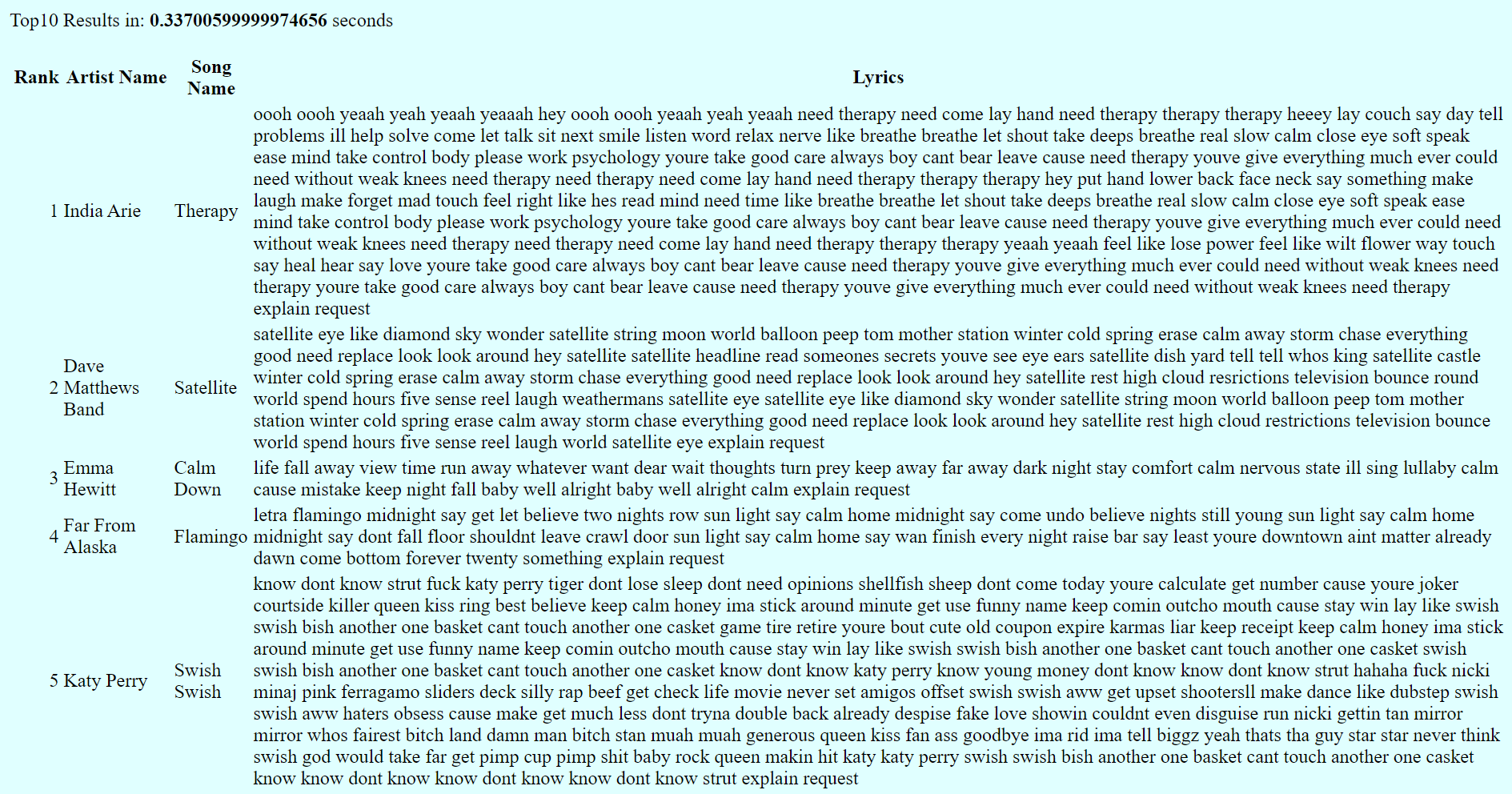
4. For retrieval of top 10 relevant items : 0.34602 s

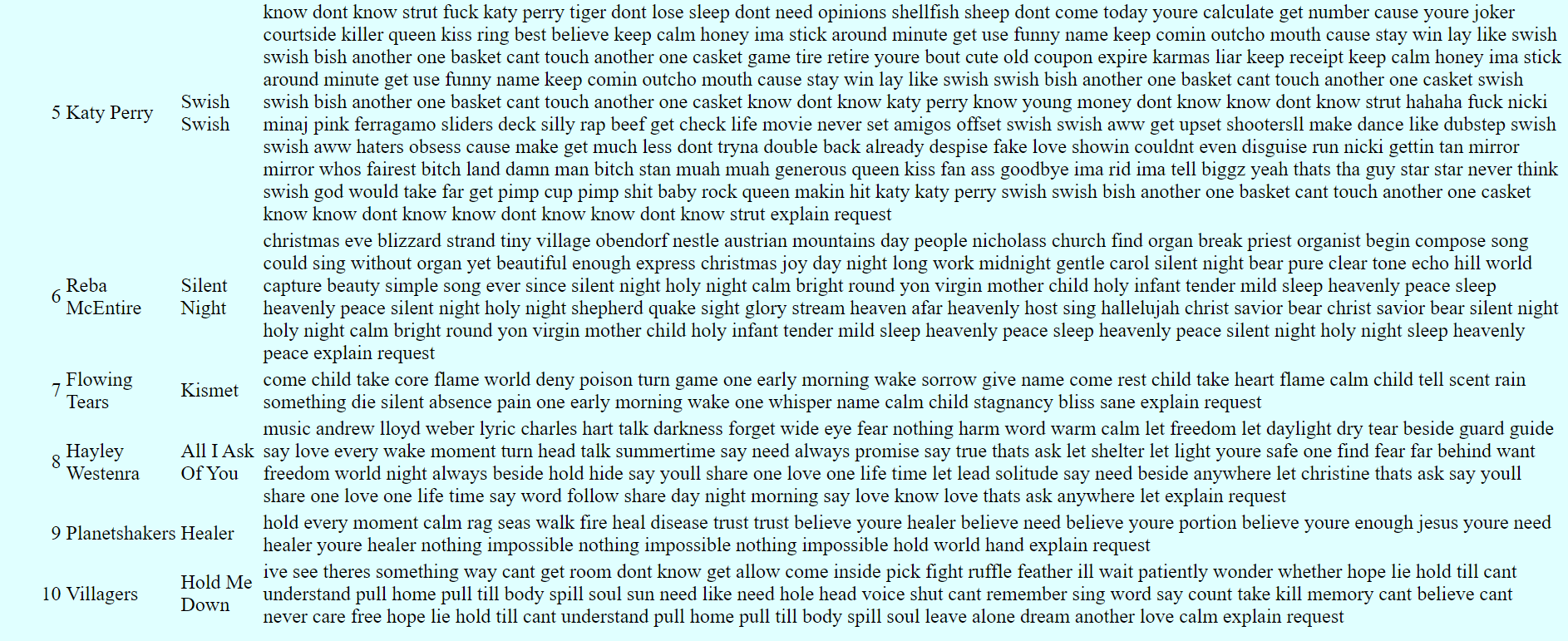
8. User Interface:

* Home screen



* Search results for query - “You need to calm down”





* App returns “Sorry, no match found.” for queries not in the dataset.
* Example queries include “star”, “start”, “love” and “boom”.

9. Team members:

* Akshat Bajpai - 2018A7PS0498H
* Anirudh G - 2018A7PS0217H
* Aryan Sharma - 2018A7PS0245H
* Thakur Shivank Singh - 2018A7PS0439H