Astria

TEAM 06

Contents

I Software Requirement and Specifications 4

1 Introduction 5
1.1 Purpose 5
1.2 Product Scope 5

2 Overall Description 6
2.1 Product Perspective 6
2.2 Product Functions 7
2.3 Operating Environment 7
2.4 User Operating Environment 7
2.5 Assumptions and Dependencies 8

3 External Interface Requirements 9
3.1 User Interfaces 9
3.2 Hardware Interfaces 9
3.3 Software Interfaces 9
3.4 Communication Interfaces 9

4 Functional Requirements 10
4.1 Search 10
4.2 Filters 10
4.3 Read More 10

5 Other Non-functional requirements 11
5.1 Performance Requirements 11
5.2 Security and Privacy Requirements 11
5.3 Reliability 11

II Installation Instructions 12

6 Installing dependencies 13

6.1 Aptitude (apt-get) dependencies . . . 13
6.2 Pip dependencies . . . 14

III User Manual 15

7 Starting the software 16

8 Main Page 17
8.1 Search Card . . . 17
8.1.1 Query Type Buttons . . . 17
8.1.2 Search Bar . . . 17
8.1.3 Filters . . . 17
8.2 Result Cards . . . 18
8.2.1 Table . . . 18
8.2.2 Read More . . . 18
8.2.3 Summary Card . . . 18

IV Algorithms 19

9 Searching 20
9.1 Introduction . . . 20
9.2 Algorithm to Search Database . . . 20
9.2.1 Elasticsearch . . . 20
9.3 Algorithm for Auto-Complete . . . 23
9.3.1 Fuzzy Search(Fuse.js) . . . 23

10 BERT 24
10.1 Introduction . . . 24
10.2 Bert-as-service . . . 24
10.3 Sentence Classification Tasks . . . 25
10.4 Out-of-memory Issues . . . 25

11 Summarization(1000 words) with Restricted Boltzmann Machine(RBM) 26
11.1 Introduction . . . 26
11.2 Pre-Processing . . . 26
11.3 Feature Vector Extraction . . . 27
11.4 Feature Matrix Generation . . . 27
11.5 Deep Learning Algorithm . . . 27
11.6 Summary Generation . . . 27

12 Summarization(200 words) with Gensim 28
12.1 Introduction 28
12.2 Text Rank Model 28
12.3 Keyword Extraction 28
12.4 Sentence Extraction 29

V Architecture 30

13 Activity Diagram 31

Part I

Software Requirement and Specifications

Chapter 1

Introduction

1.1 Purpose

This SRS describes the software functional and non-functional requirements for release of Astria v0.1. This software aims to create a localized search engine for legal documents, which can cater to the need of lawyers as well as non-professionals. Unless otherwise stated, all requirements specified here are of high priority and committed for release v0.1.

1.2 Product Scope

This software consists of following major functions:

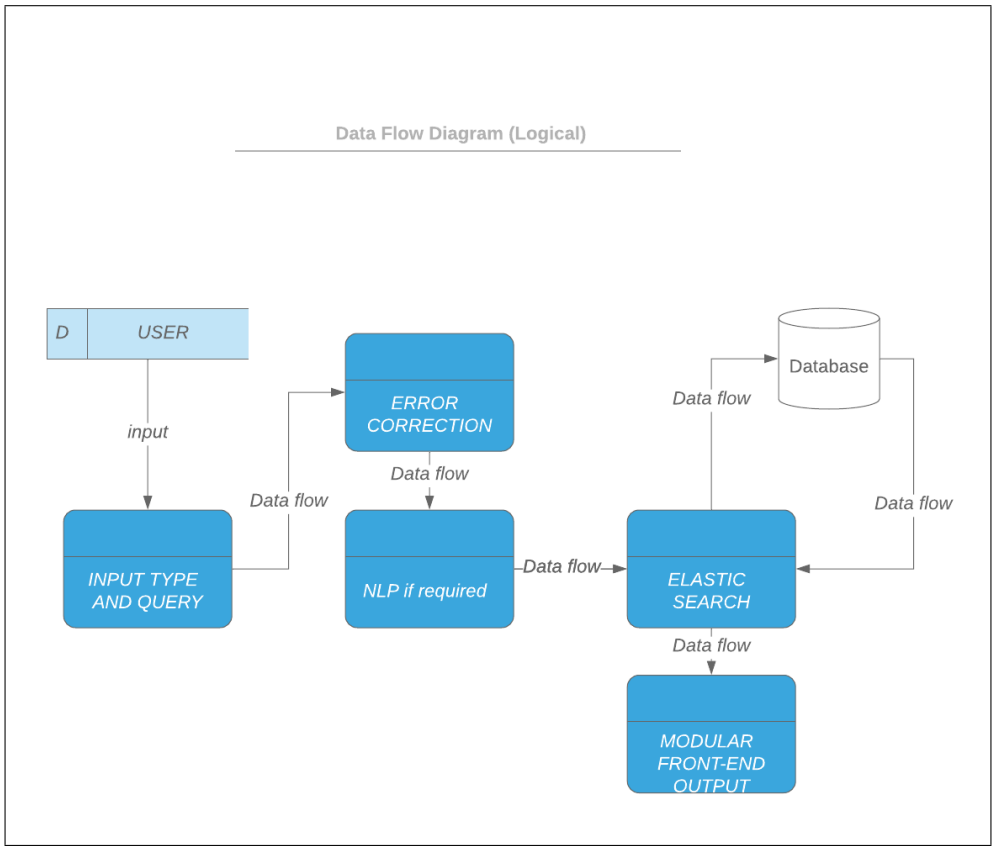
- Taking input, which can be a search query with legal terms and descriptions known to a lawyer, or a common query in simple terms by a common man.
- Generating an appropriate set of results depending on the search parameters.
- Highlighting imortant fields in a tabular form to display the most important facts about the case in a concise manner.
- Generating a brief summary of each case that appears in the result page.

Chapter 2

Overall Description

2.1 Product Perspective

This software aims to generate a fast,efficient search engine exclusively for legal data.Google search has a vast data set and hence its results show data from other unnecessary sources. Moreover, most legal search engines available are efficient as long as legal terms are used. But our goal here is to ensure that relevant results are obtained for both legal queries by a lawyer and queries in common language used by a layman.



2.2 Product Functions

The set of functionalities supported by this software are as follows :

- *Search Page*: The software allows the user to type four types of queries - Act Names, Legal/Non-Legal keywords, Case Title, and Natural Language query.

- *Filtration of Data*: Several filters can be applied according to the user's need like - Judge name, Date(from and to), Category, and Acts.

- *Results Page*: Will try to generate relevant results according to the given search input and filters. Results will be shown in tabular form highlighting the fields - List of Cases, Judgement, Judge, Acts Cited, Case Category and Date.

- *Get Summarization*: Get a brief (100-200 words approx.) summary of each case that shows up in the results.A "Read More" button is there, which when clicked, shows the summary.

2.3 Operating Environment

This software has a front-end and a back-end:

Front-End: The frontend is developed using HTML,CSS and JavaScript.The basic structure is given by HTML,designing is done by CSS and JavaScript is used to add user interactions and other page actions.

Back-End: The backend is developed using mainly Flask(Python).There is a database which is linked to the frontend using AJAX.

2.4 User Operating Environment

The user can use the software on any Operating System including Ubuntu, Windows, Android (phone). The user needs to have a Browser (Chrome/Firefox) on their system.

2.5 Assumptions and Dependencies

This software has been targeted at Debian-based x64 Operating Systems. It depends on Open Source tools and Python libraries like:

- Flask
- Elasticsearch
- Genism
- Requests
- Scipy
- bert-serving-client
- bert-serving-server
- tensorflow
- tensorboard
- Keras
- Werkzeug
- Jinja2
- Jmespath
- Urllib
- Numpy

The required dependencies are all packaged with the installer, although super-user permissions might be necessary to install them. The software cannot be installed on any other operating system apart from Linux-Debian based x64 Architecture.

Chapter 3

External Interface Requirements

3.1 User Interfaces

The user interface is loaded with features and effects to enable a good experience and ease of access. There is a search box with date,case,judge,category filters and radio buttons for 4 types of query. The results are appended to the search page itself and is shown after a rapid scroll on entering the search.The result boxes will have an option to show the summary as well.

3.2 Hardware Interfaces

No special hardware is needed for the functioning of this software. A computer with a monitor, a keyboard and a mouse suffices.

3.3 Software Interfaces

The software consists of a frontend and backend. The frontend displays the data and options to the user. When the user sends a search request, the request is sent to the backend which searches the database and generates a result page that is appended to the search page. Flask is used for backend and mostly JavaScript,HTML,CSS sufficed for frontend.

3.4 Communication Interfaces

Internet connection is necessary for working of the software.

Chapter 4

Functional Requirements

4.1 Search

There are four radio buttons for 4 types of query,namely Act Names,Legal/Non-Legal keywords,Cases titles,Natural language query.The user selects the type of query and enters the search in the search bar.

4.2 Filters

There is a Date Filter which allows the user to set the start and end date. Three placeholders are also available for the filters Judge,Category and Acts.

4.3 Read More

Every search result is in the form of a card which shows tabular data.There is a "Read More" button which when clicked, shows the summary of the case and gets converted into a "Read Less" button. The summary can be hidden again by clicking the "Read Less" button.

Chapter 5

Other Non-functional requirements

5.1 Performance Requirements

The software should take less than 2 seconds to load the results. For this efficient search algorithms, namely Elasticsearch and Fuse.js are employed.

5.2 Security and Privacy Requirements

The software does not require submission of sensitive information and hence no security and privacy requirements have been identified.

5.3 Reliability

The results are expected to have a good accuracy and match the entered query. Only results which match the search input above a certain percentage are shown to maintain the reliability of the search engine.

Part II

Installation Instructions

Chapter 6

Installing dependencies

1. Open terminal. Press `Ctrl + Alt + T` on ubuntu to open it.
2. Navigate to the code repository and follow the instructions given in `install.md` file.
3. Press enter whenever prompted during installation.

The following packages are supposed to be installed :

6.1 Aptitude (apt-get) dependencies

- build-essential
- python3-dev
- python3-setuptools
- python3-pip
- apt-transport-https
- elasticsearch

6.2 Pip dependencies

- flask
- boto3
- botocore
- bz2file
- certifi
- chardet
- Click
- docutils
- idna
- itsdangerous
- jmespath
- MarkupSafe
- numpy
- tensorflow
- tensorboard
- pkg-resources
- python-dateutil
- requests
- s3transfer
- six
- Keras
- bert-serving-client
- bert-serving-server
- smart-open
- urllib3
- Werkzeug
- gensim
- requests
- Jinja2
- scipy

Part III

User Manual

Chapter 7

Starting the software

This software is operated using the web browser. To start the server, open terminal and execute server.py using python3. Run the command
`python3 server.py`

Open the internet browser and open the following URL from the address bar :
`http://127.0.0.1:8000`

Chapter 8

Main Page

Main Page is where all the information of the software resides. The following are the most important areas of the main page:

8.1 Search Card

The Search Card contains all the tools required to send a search request.It consist of the following parts:

8.1.1 Query Type Buttons

There are four radio buttons for four types of search queries: Act names, Case titles, Legal/Non-Legal keywords, and Natural Language query.

8.1.2 Search Bar

This search bar is where the user types in the search query. A list of search recommendations also appear.

8.1.3 Filters

There are filters for searches which include a date filter(from-to).There is a dropdown button which enables the user to select the appropriate date.Placeholders for Judge,Acts and Category are also available, where the user manually enters the data.A list of previous and recommended filters is also displayed in dropdown.

8.2 Result Cards

There are multiple result cards which are displayed according to the search parameters.Each result card consists of:

8.2.1 Table

A table of data is displayed in each result card. For every field, a small number of items are displayed.In case there are several items in a single column, the extra data is truncated, and can be seen as a tooltip on hovering over the column.The table consists of the following fields:

1. List of Cases
2. The Final Judgement for the case
3. The Judge for the case
4. The Acts cited in the case
5. Date of beginning and ending of the case

8.2.2 Read More

There is a "Read More" button, which when pressed will show the summary of the case in a dropdown card. This gets converted into a "Read Less" button on clicking.On clicking the "Read Less" button, the summary is hidden again.

8.2.3 Summary Card

The Summary Card has a text area where the summary is shown in 100-200 words. It is preceded by a heading describing the case, or the case title.

Part IV

Algorithms

Chapter 9

Searching

9.1 Introduction

In this part we apply some efficient search algorithms.

9.2 Algorithm to Search Database

9.2.1 Elasticsearch

Elasticsearch is an open-source, RESTful, distributed search and analytics engine built on Apache Lucene.One can send data in the form of JSON documents to Elasticsearch using the API or ingestion tools such as Logstash and Amazon Kinesis Firehose. Elasticsearch automatically stores the original document and adds a searchable reference to the document in the cluster's index. The motivation to use Elasticsearch is as follows. While one can drive a car by turning a wheel and stepping on some pedals, highly competent drivers typically understand at least some of the mechanics of the vehicle. The same is true for search engines. Elasticsearch provides APIs that are very easy to use, and it will get one started and take one far without much effort.
The various components of Elasticsearch include:

- *Near Real Time(NRT)*: Elasticsearch is a near real time search platform. What this means is there is a slight latency (normally one second) from the time you index a document until the time it becomes searchable.
- *Cluster*: A cluster is a collection of one or more nodes (servers) that together holds the entire data and provides federated indexing and search capabilities across all nodes.
- *Node*: A node is a single server that is part of the cluster, stores the data, and participates in the cluster's indexing and search capabilities.A

node can be configured to join a specific cluster by the cluster name.In a single cluster, one can have as many nodes as needed.

- *Index*: An index is a collection of documents that have somewhat similar characteristics. For example, we can have an index for customer data, another index for a product catalog, and yet another index for order data.

- *Type*: Within an index, we can define one or more types. A type is a logical category/partition of the index whose semantics is completely up to the coder. In general, a type is defined for documents that have a set of common fields.

- *Document*: A document is a basic unit of information that can be indexed. For example, we can have a document for a single customer, another document for a single product, and yet another for a single order. This document is expressed in JSON (JavaScript Object Notation) which is an ubiquitous internet data interchange format.

- *Shards and Replicas*: Elasticsearch provides the ability to subdivide the index into multiple pieces called shards. When we create an index, we can simply define the number of shards that we want. Each shard is in itself a fully-functional and independent "index" that can be hosted on any node in the cluster.

We can have a better idea at :
`https://www.elastic.co/blog/found-elasticsearch-from-the-bottom-up#inverted-indexes-and-index-terms`

Once we have our node (and cluster) up and running,we need to communicate with it. For this, Elasticsearch provides a very comprehensive and powerful REST API that we can use to interact with our cluster. The Elastic Search API for our software runs at localhost:9200 and it generates a JSON output similar to this:


```
[
  {
    "_index": "cases",
    "_type": "case",
    "_id": "230",
    "_score": 7.8923335,
    "_source": {
      "date": "2  May  2014",
      "filename": "2014_S_72",
      "title": "Selvam v State Through Inspector of Police",
      "cit-id": "2014 Indlaw SC 325",
      "case-cat": "",
      "subject": "Criminal| Practice & Procedure",
      "legal-key": "Murder",
      "parts": "",
      "judge": "B. S. Chauhan, J.",
      "act-used": "",
      "result": "Appeal disposed of\n"
    }
  },
  {
    "_index": "cases",
    "_type": "case",
    "_id": "52985",
    "_score": 7.8923335,
    "_source": {
      "date": "13 March 2014",
      "filename": "2014_S_374",
      "title": "Bhagwan Tukaram Dange v State of Maharashtra",
      "cit-id": "2014 Indlaw SC 165",
      "case-cat": "",
      "subject": "Criminal| Practice & Procedure",
      "legal-key": "Murder",
      "parts": "",
      "judge": "K. S. Radhakrishnan, J.",
      "act-used": "",
      "result": "Appeal disposed of\n"
    }
  },
  {
    "_index": "cases",
    "_type": "case",
    "_id": "7421",
    "_score": 7.752153,
    "_source": {
      "date": "13 March 2014",
      "filename": "2014_S_431",
      "title": "Kanhaiya Lal v State of Rajasthan",
      "cit-id": "2014 Indlaw SC 164",
      "case-cat": "",
      "subject": "Criminal| Practice & Procedure",
      "legal-key": "Murder",
      "parts": "",
      "judge": "C. Nagappan, J.",
      "act-used": "",
      "result": "Appeal allowed\n"
    }
  },
```


Elasticsearch is built on top of Lucene. Every shard is simply a Lucene index. Lucene index, if simplified, is the inverted index. Every Elasticsearch index is a bunch of shards or Lucene indices. When you query for a document, Elasticsearch will subquery all shards, merge results and return it to you. When you index document to Elasticsearch, the Elasticsearch will calculate in which shard document should be written using the formula.

$$shard = hash(routing)\%(number\_of\_primary\_shards)$$

It gives us the list of case files relevant to the query ranked according to the pertinence with the help of scores. Each case file which is returned contains the essential information of the case such the title, citation id, judge and result.

For more information,visit :
https://www.elastic.co/guide/en/elasticsearch/reference/1.4/getting-sta
html

9.3 Algorithm for Auto-Complete

9.3.1 Fuzzy Search(Fuse.js)

A fuzzy search is a process that locates Web pages that are likely to be relevant to a search argument even when the argument does not exactly correspond to the desired information. A fuzzy search is done by means of a fuzzy matching program, which returns a list of results based on likely relevance even though search argument words and spellings may not exactly match. Exact and highly relevant matches appear near the top of the list. Subjective relevance ratings, usually as percentages, may be given.

Fuzzy matching programs usually return irrelevant hits as well as relevant ones. Superfluous results are likely to occur for terms with multiple meanings, only one of which is the meaning the user intends. If the user has only a vague or general idea of the topic, or does not know exactly what to look for, the ratio of relevant hits to irrelevant hits tends to be low. (The ratio is even lower, however, when an exact matching program is used in this situation.)

Fuzzy searching is much more powerful than exact searching when used for research and investigation. Fuzzy searching is especially useful when researching unfamiliar, foreign-language, or sophisticated terms, the proper spellings of which are not widely known. Fuzzy searching can also be used to locate individuals based on incomplete or partially inaccurate identifying information.

Hence, in our project we have used Fuse.js to provide relevant suggestions based on typed data. Different JSONs were created for different types of queries. When the user selects the type of query, and starts typing, the Fuzzy Search runs on the respective JSON and provides relevant autocomplete options.

For more information,visit :
https://whatis.techtarget.com/definition/fuzzy-search

Chapter 10

BERT

10.1 Introduction

BERT, or Bidirectional Encoder Representations from Transformers, is a new method of pre-training language representations which obtains state-of-the-art results on a wide array of Natural Language Processing (NLP) tasks. It is a method of pre-training language representations, meaning that we train a general-purpose "language understanding" model on a large text corpus (like Wikipedia), and then use that model for downstream NLP tasks that we care about (like question answering). BERT outperforms previous methods because it is the first unsupervised, deeply bidirectional system for pre-training NLP. Unsupervised means that BERT was trained using only a plain text corpus, which is important because an enormous amount of plain text data is publicly available on the web in many languages.

10.2 Bert-as-service

Bert-as-service uses BERT as a sentence encoder and hosts it as a service via ZeroMQ, allowing you to map sentences into fixed-length representations in just two lines of code. It is State-of-the-Art, Fast, Scalable, Reliable and Easy to use.We use this in our project for the autocomplete feature. The results are really accurate. It generates vectors for each word and is more powerful than word2vec and other NLP implementation.

10.3 Sentence Classification Tasks

We run a python code, 'run_classifier.py' for training our sentence classifier.The first run shows an accuracy of about 84.55%. After 3-4 runs, we see results between 84% and 88%.Once we have trained our classifier we can use it in inference mode.

10.4 Out-of-memory Issues

BERT heavily relies on memory. So,it is genreally run on systems which have strong GPUs, and about 6GB or 12GB memory. Since such large amounts of memory are handled, out-of-memory issues are common. The factors that affect memory usage include: max_seq_length, train_batch_size, Model type(BERT-Base or BERT-Large), and the Optimizer used.
For more information, visit:

- https://github.com/google-research/bert?fbclid=IwAR2bQ1euL4suq8OAj
- https://github.com/hanxiao/bert-as-service?fbclid=IwAR2uhE1w_ZOieRGDq5m5IanGWM4xjcXT2Us0un0VvLkIA_EXaqvmV-KUUY4

Chapter 11

Summarization(1000 words) with Restricted Boltzmann Machine(RBM)

11.1 Introduction

Restricted Boltzmann Machine (RBM) is proposed which is a type of unsupervised deep learning. In unsupervised learning, the input data is not labeled. It is a type of machine learning algorithm, in which inferences are drawn from datasets consisting of unlabeled input data. There are two types:

- Extractive Text Summarization
- Abstractive Text Summarization

This algorithm uses Extractive text summarization.

In this method, textsummarization based on some features like title similarity,term frequency, sentence ranking etc are extracted to form summary.

11.2 Pre-Processing

The text is preprocessed before the algorithm is applied. This is done in order to structure the document by applying myriads of techniques by which the density of document is reduced and document is made light weight. This makes the further processing of document easier.This includes stop-word removal, part of speech tagging,stemming, punctuation mark removal and so on.

11.3 Feature Vector Extraction

The document which is made light weight in the preprocessing phase is now structured into a matrix. A sentence matrix M of order n*v contains the features for every sentence of a matrix. Here, 'n' is the number of sentences in the document and 'v' is the number of features. Four features are extracted of a sentence of text document namely Title Similarity, Positional Feature, Term Weight, Sentence Length, Proper Noun Score.

11.4 Feature Matrix Generation

The above calculated features' values are then stored in a matrix form where the columns represent the features and rows represent the sentences.

11.5 Deep Learning Algorithm

The sentence matrix containing a set of feature vectors is given as an input to the RBM phase as a visible layer. The Deep Learning Algorithm produces a refined matrix after several phases of refining. The refined matrix obtained from the deep learning phase is used for further summary generation phase.

11.6 Summary Generation

Now, in two ways summary can be generated. One is a generalized summary of the whole document and the other way is based on the user query entered by the user(using the sentence score).

In our project, we generate a generalized summary of each case file with 1000 words. This summary file consists of the Case facts in the beginning, followed by the description and then at the end, we have the verdict.

For more information, visit:

- `https://thescipub.com/pdf/10.3844/jcssp.2014.1.9`
- `http://ijsart.com/Content/PDFDocuments/IJSARTV4I623858.pdf`

Chapter 12

Summarization(200 words) with Gensim

12.1 Introduction

Gensim is a free Python library designed to automatically extract semantic topics from documents. The gensim implementation is based on the popular TextRank algorithm. It is an open-source vector space modelling and topic modelling toolkit, implemented in the Python programming language, using NumPy, SciPy and optionally Cython for performance. We use the summarization.summarizer from gensim. This summarising is based on ranks of text sentences using a variation of the TextRank algorithm.

12.2 Text Rank Model

The basic idea implemented by a graph-based ranking model is that of voting or recommendation.

When one vertex links to another one, it is basically casting a vote for that vertex. The higher the number of votes cast for a vertex, the higher the importance of that vertex.

TextRank includes two NLP tasks,namely the Keyword extraction task,Sentence extraction task.

12.3 Keyword Extraction

The task of keyword extraction algorithm is to automatically identify in a text a set of terms that best describe the document.The simplest method is to use frequency criterion,but it yields poor results. The TextRank keyword extraction algorithm is fully unsupervised. No training is necessary.

12.4 Sentence Extraction

To apply TextRank, we first build a graph associated with the text, where the graph vertices are representative for the units to be ranked. The goal is to rank entire sentences, therefore, a vertex is added to the graph for each sentence in the text.

In our project gensim is used for the 200 words summarization. The input files are the original case files. This summarization produces good results with facts and the verdicts of the cases.

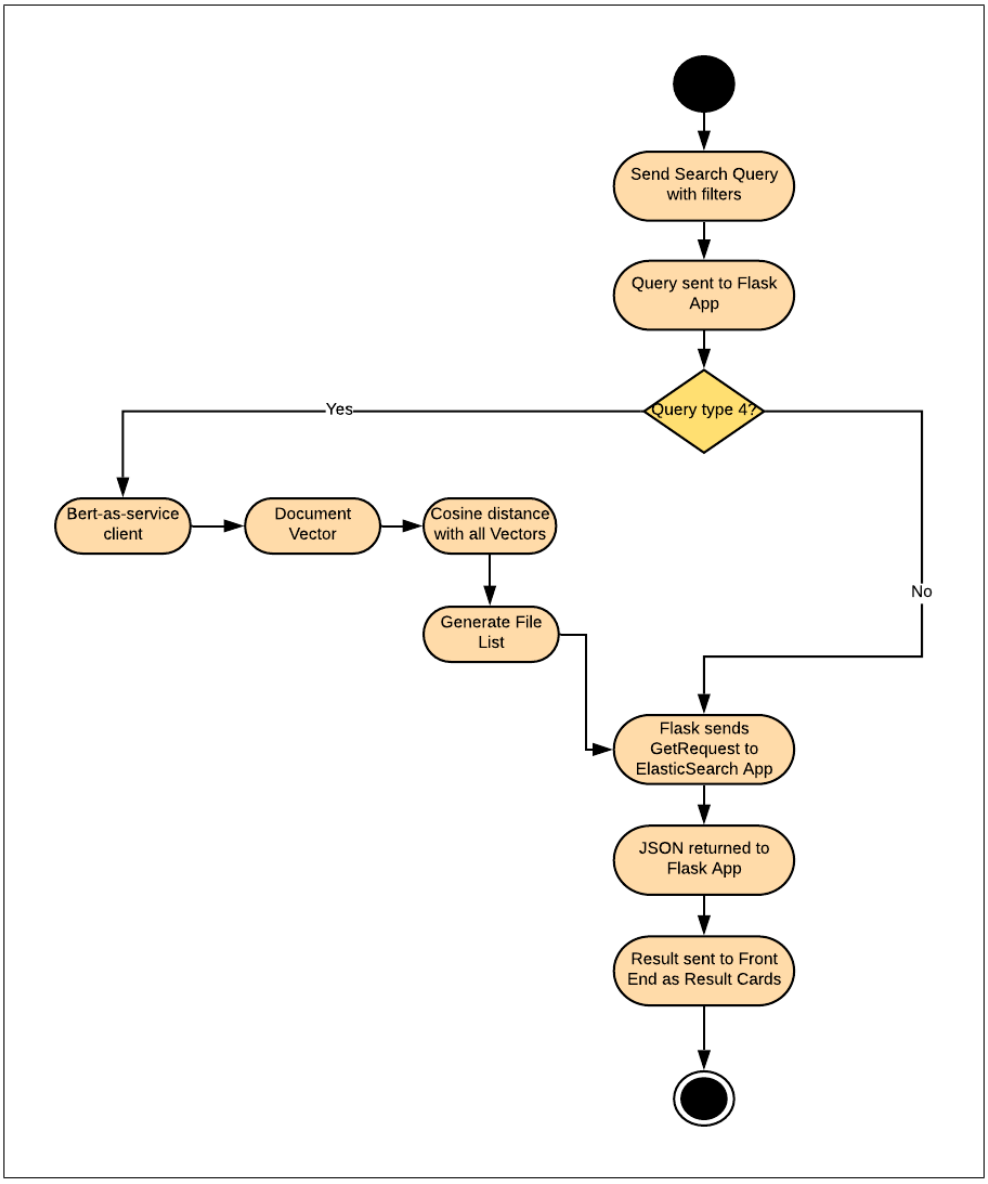
For more information, visit:
`https://radimrehurek.com/gensim/summarization/summariser.html`

Part V

Architecture

Chapter 13

Activity Diagram



Analysis :

- The diagram above shows the various activities involved in the execution of the software.

- The software starts with the user sending a search query.

- Then the query type is detected by employing Fuzzy Search.

- Then the query is sent to the appropriate Flask app endpoint.

- Now, if the query is of type 4, that is it is a natural language query, then the Bert-as-service client generates document vectors and uses cosine similarity to generate a list of file names,which in turn is sent to the Flask App.If the query is not of type 4, then step 6 is directly executed.

- Then, the Flask app sends GetRequest to the Elastic search app.

- The Elastic Search app returns the JSON generated to the Flask App.

- The Flask App generates the search results and generates the result html including the result cards.