# Data Mining Project:IBM Watson

## Team introduction

The project that is going to be described in the following document represents the final project of the data mining course of University Babeș-Bolyai. The project takes after the IBM initiative to create a question answering machine to compete in the TV quiz show, Jeopardy, being also known as the ‘default’ project. The team is made up of five members who are studying Distibuted Systems over the Internet and Databases master's programs:

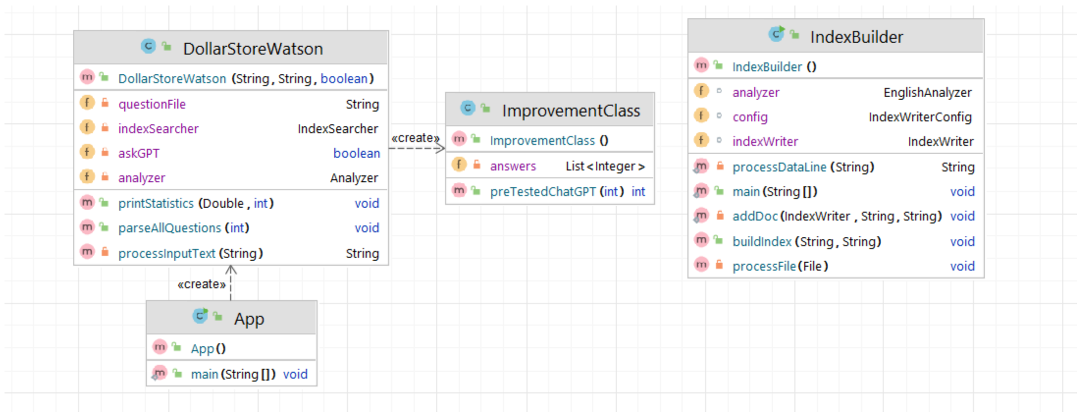
- Sava Radu-Florian
- Roszinecz Norbert
- Urzică Bogdan-Radu
- Băciulescu Raul-Ovidiu
- Stan Alexandru

## Overview

The purpose of the project is to answer 100 quiz-like questions similarly to the Jeopardy quiz show. As for the datasource, the application is using a collection of wikipedia pages. They can be found in the project structure as resources. The wikipedia archive stored as 80 separate pages, each containing multiple documents. The application logic lies in the fact that most of the questions can be answered corrrectly with the title of a wikipedia page. Based on this assumption, we first create the index by using the content of each wikipedia document corresponding to a title. The, after creating the index, we respond to each question and check for the correct answer using the information retrieval system created with Apache Lucene. Finally, we apply accuracy metrics and draw conclusions.

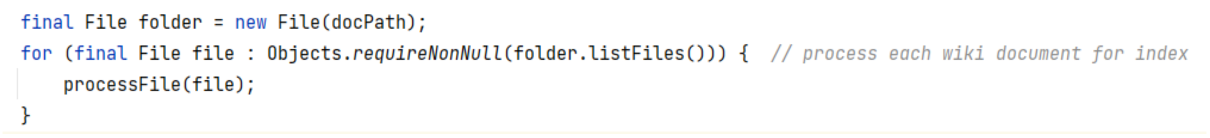
## Project structure

The project was realised in Java 17.0.1 using the Apache Lucene dependencies ( core, query parser and common analyzers). It realies on Maven as an automated build tool. In the following section we will discuss on the class diagram and code structure.



## Index Building

The class IndexBuilder creates the index as a resource by analyzing the given wikipedia pages. It calls upon the buildIndex function which takes two parameters: the index creation path and the path of the wikipedia pages used to create the index. By following this approach, each file in the wikipedia path will be processed while iterating throught the directory files

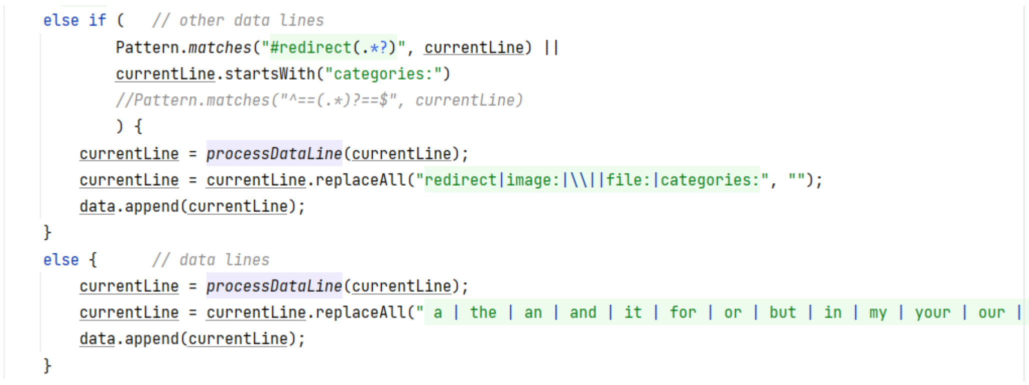


```
final File folder = new File(docPath);
for (final File file : Objects.requireNonNull(folder.listFiles())) {  // process each wiki document for index
    processFile(file);
}
```

When parsing a file, the name of the file which is being parsed will be printed and each each document will be built when reading the file line by line, trim it and convert to lowercase as a form of normalization. A document starts with its title enclosed in double square brackets like this [[Example]], so if the line of text matches this pattern then the document which was being created previously will be added to the collection and a new document will be created for the new article.

Otherwise, the line will be parsed by a more complex logic. In this case, we make sure to remove the wikipedia sections like the subtitles ‘references’, ‘see also’, ‘bibliography’, ‘external links’ ‘further reading’, ‘notes’, ‘gallery’ ‘footnotes’ or file/image annotations, which do not represent consistent data neither do they posess semantic meaning to be contained by our index.

Finally, the redirect links (which contain the structure ‘#redirect’) and regular text are treated in a similar manner: they get the link annotations removed ( if present) when calling the method processDataLine, then redirect links have their metadata removed, while regular text has around thirteen stopwords removed.

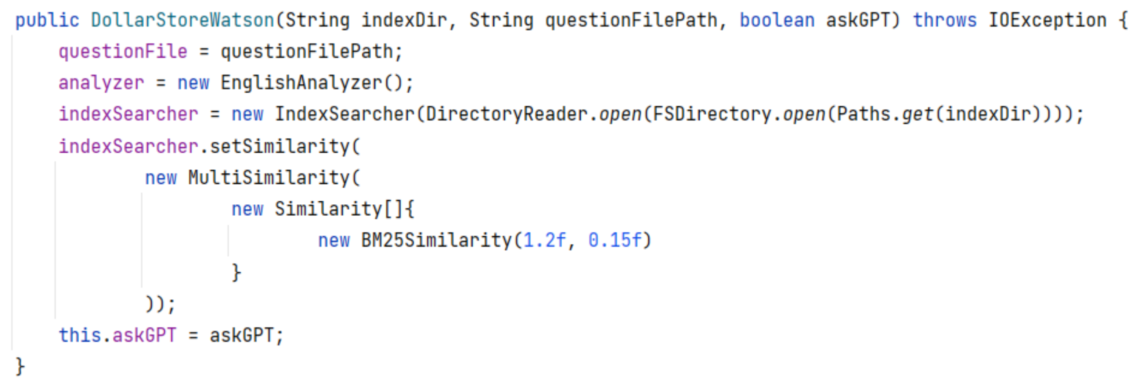


```
else if (   // other data lines
        Pattern.matches("#redirect(.*?)", currentLine) ||
        currentLine.startsWith("categories:")
        //Pattern.matches("^==(.*)?==$", currentLine)
        ) {
    currentLine = processDataLine(currentLine);
    currentLine = currentLine.replaceAll("redirect|image:|\\||file:|categories:", "");
    data.append(currentLine);
}
else {      // data lines
    currentLine = processDataLine(currentLine);
    currentLine = currentLine.replaceAll(" a | the | an | and | it | for | or | but | in | my | your | our | their ", " ");
    data.append(currentLine);
}
```

## Main Application

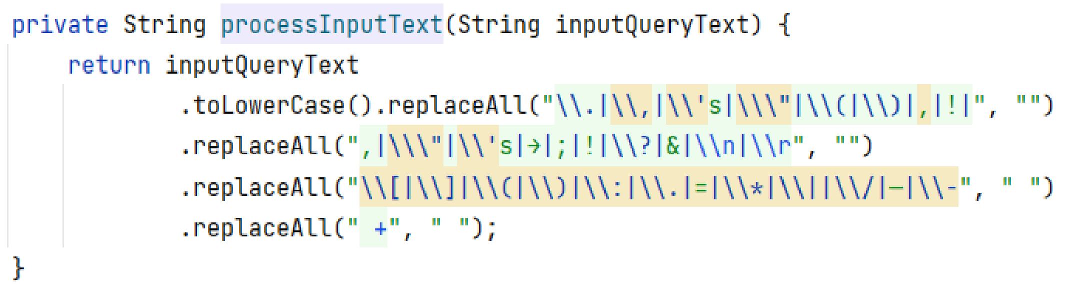
The application starts by creating a new DollarStoreWatson object which requires the path of the index, the path to the questions file and a boolean called askGPT. This boolean value was used for  the improvement of the system, the default value if ‘false’ which means that it will follow the Jeopardy logic with one correct answer, while the ‘true’ value allows for the application to pick one of the first ten answers retrieved by our system with the help of chatGPT.

The DollarStoreWatson object is using an English Analyzer applied on the results retrieved by the index. The similarity used was created in a trial and error process, by trying out multiple similarity score formulas ( boolean, Jelinek-Mercer, etc .. ) and them summing them by appling a MultiSimilarity score. Starting from Jelinek-Mercer then moving to a hybrid MultiSimilarity with Jelinek-Mercer and boolean similarity, the best results were obtained with a BM25 similarity with k1 = 1.2 and b = 0.15 reaching a 44% P@1 score.



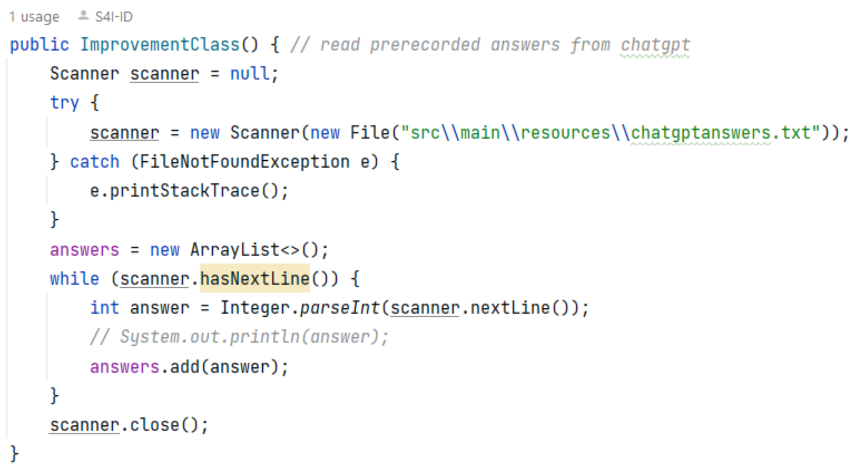
```java
public DollarStoreWatson(String indexDir, String questionFilePath, boolean askGPT) throws IOException {
    questionFile = questionFilePath;
    analyzer = new EnglishAnalyzer();
    indexSearcher = new IndexSearcher(DirectoryReader.open(FSDirectory.open(Paths.get(indexDir))));
    indexSearcher.setSimilarity(
            new MultiSimilarity(
                    new Similarity[]{
                            new BM25Similarity(1.2f, 0.15f)
                    }
            ));
    this.askGPT = askGPT;
}
```

Esentially, DollarStoreWatson responds to all 100 questions read from file, normalizes each query ( by removing special characters and leaving only tokens which contain semantic meaning ) and then returns a certain number of results based on the similarity score. If the chatGPT option is set to ‘false’ initially it is going to be only one result, staying true to the Jeopardy rules, otherwise it will ask chatGPT for a hint out of the top 10 answers of our system.



```java
private String processInputText(String inputQueryText) {
    return inputQueryText
            .toLowerCase().replaceAll("\\.|\\,|\\'s|\\\"|\\(|\\)|,|!|", "")
            .replaceAll(",|\\\"|\\'s|→|;|!|\\?|&|\\n|\\r", "")
            .replaceAll("\\[|\\]|\\(|\\)|\\:|\\.|=|\\*|\\||\\/|—|\\-", " ")
            .replaceAll(" +", " ");
}
```

The ImprovementClass was used initially to call chatGPT using an API key, but due to financial limitations that logic has been replaced with the memorised answers read from a local file originally produced by chatGPT.



```java
1 usage   S4I-ID
public ImprovementClass() { // read prerecorded answers from chatgpt
    Scanner scanner = null;
    try {
        scanner = new Scanner(new File("src\\main\\resources\\chatgptanswers.txt"));
    } catch (FileNotFoundException e) {
        e.printStackTrace();
    }
    answers = new ArrayList<>();
    while (scanner.hasNextLine()) {
        int answer = Integer.parseInt(scanner.nextLine());
        // System.out.println(answer);
        answers.add(answer);
    }
    scanner.close();
}
```

## Results

A pre-chatGPT improvement model has reached a precision at one (P@1) of 44%, meaning that out of 100 questions 44 were answered correctly by the system. This metric was chosen in order to stay true to our initial objective and make it as similar as possible to replicate the Jeopardy environment. For the improved version, chatGPT has been included to re-evaluate the top 10 results retrieved by our system similar to the possibility of ‘calling a friend’ which is widely used nowadays in quiz shows. By adding this improvement, the system precision has reached 64%.

## Code-related Questions

The output of the code is printed to console and says explicitly if the result was correct or not.

```
--- QUESTION 1

newspapers the dominant paper in our nation capital it among the top 10 us papers in circulation

   Correct answer  : the washington post

   INCORRECT answer: the new york sun

--- QUESTION 8

ucla celebrity alumni this woman who won consecutive heptathlons at the olympics went to ucla on a basketball scholarship

   Correct answer  : jackie joyner-kersee

   Returned answer : jackie joyner-kersee
```

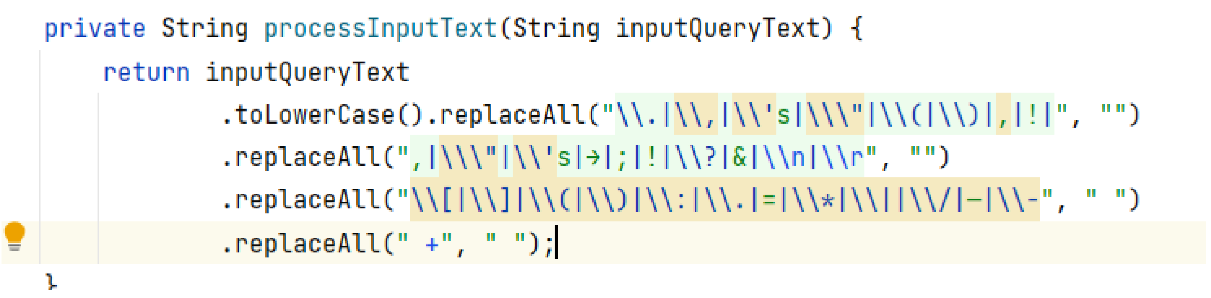
In the case where an alternate solution was possible ( e.g. Kennedy and JFK are both acceped as valid answers) , they are followed as such:

```
--- QUESTION 93

notes from the campaign trail the pulitzer winning the making of the president 1960 covered this man successful presidential campaign

   Correct answer  : jfk

   INCORRECT answer: hunter s. thompson

notes from the campaign trail the pulitzer winning the making of the president 1960 covered this man successful presidential campaign

   Correct answer  : john f. kennedy

   INCORRECT answer: hunter s. thompson
```

## Non-code Questions

1) Indexing and retrieval:
   Q: What issues specific to Wikipedia content did you discover, and how did you address them?
   A: The format used is slightly inconsistent and difficult to analyze via code by simply applying certain transformation rules like replacement. The problematic elements were the visual elements which offer way less information in plain text, especially the annotations. The annotations specifically hold no semantic meaning and must be disposed of in order to increase the quality of the index.The solution was to add each replacement rule specifically and check for the content of the document saved to index periodically to check if the annotated structures were modified as expected. In the code each such structure is matched by a regular expression, handled accordingly and explained in a comment.

   Q: Implement the retrieval component, which takes as query the Jeopardy clue (and, optionally, the question category) and returns the title of the Wikipedia page that is most similar. Describe how you built the query from the clue. For example, are you using all the words in the clue or a subset? If the latter, what is the best algorithm for selecting the subset of words from the clue?
   A: The category is concatenated to the beginning of the clue, converted to lowercase and then parantheses are removed, as they create additional semantic context which is not relevant, and special characters are removed



```
private String processInputText(String inputQueryText) {
    return inputQueryText
            .toLowerCase().replaceAll("\\.|\\,|\\'s|\\\"|\\(|\\)|,|!|", "")
            .replaceAll(",|\\\"|\\'s|->|;|!|\\?|&|\\n|\\r", "")
            .replaceAll("\\[|\\]|\\(|\\)|\\:|\\.|=|\\*|\\||\\/|-|\\-", " ")
            .replaceAll(" +", " ");
}
```

   Q: Are you using the category of the question?
   A: Initally, we used the category as a relevant factor and adapted our index by adding the document category when it was present. This approach appears to be similar to having multiple smaller indexes, but it appers to perform worse than our approach of adding the category to the beginning of the question and applying a custom normalization.

2) Measuring performance:
   Q: Measure the performance of your Jeopardy system, using at least one of the metrics discussed in class. Justify your choice and then report performance using the metric(s) of your choice.
   A: We used the P@1 metric because it is the closest to measuring the desired result, as in the Jeopardy contest there is only one answer which can be offered by the participant per question. In this case, it makes sense that the metric chosen is one that resembles the reality of the situation for which it was created. The system is able to

reach a P@1 of 44%, which means that out of 100 questions it is able to produce a perfect result for 44 of them.

3) Error analysis:

Q: Perform an error analysis of your best
system. How many questions were answered correctly/incorrectly?
A: There were 44 questions which were correctly answered out of 100. By this logic, there remain 56 questions which were answered incorrectly.

Q: Why do you think the correct questions can be answered by such a simple system?
A: Most likely yes, but the problem with a system that performs 'well' would be that it must be tested with diverse question pools since there is the posiblility that it could perform well on a very specific set of question and perform unsatisfactory on another set of questions, much like the overfitting problem met when dealing with artificial intelligence.

Q: What problems do you observe for the questions answered incorrectly?
A: The problems in which the clues were insufficient. The question was too vague for the system to produce an accurate result or the focus of the question is not on the desired subject.

Q: Aim to group the errors into a few classes and discuss them.
A: The errors can be categorised into two groups:
(a) Too vague: the question is too generic for the retrieval system to answer correctly
(b) Overly specific off topic: the question emphasyses on being very detailed when describing a term in the question which holds little relevance to the desired answer ( e.g. describing Cambodia will overlap with 'history of Cambodia' even though it may not be the desired answer)

4) Improving retrieval:

Q: Improve the above standard IR system using natural language processing and/or machine learning. For this task you have more freedom in choosing a solution and I

encourage you to use your imagination. For example, you could implement a posi-tional index instead of a bag of words. Or, you could use ChatGPT to rerank the top

K pages produced by your information retrieval system. What is the performance of your system after this improvement?

A: By using chatGPT in order to rerank the top 10 pages produced by our information retrieval system the number of correct questions answered has increased from 44 to 64.