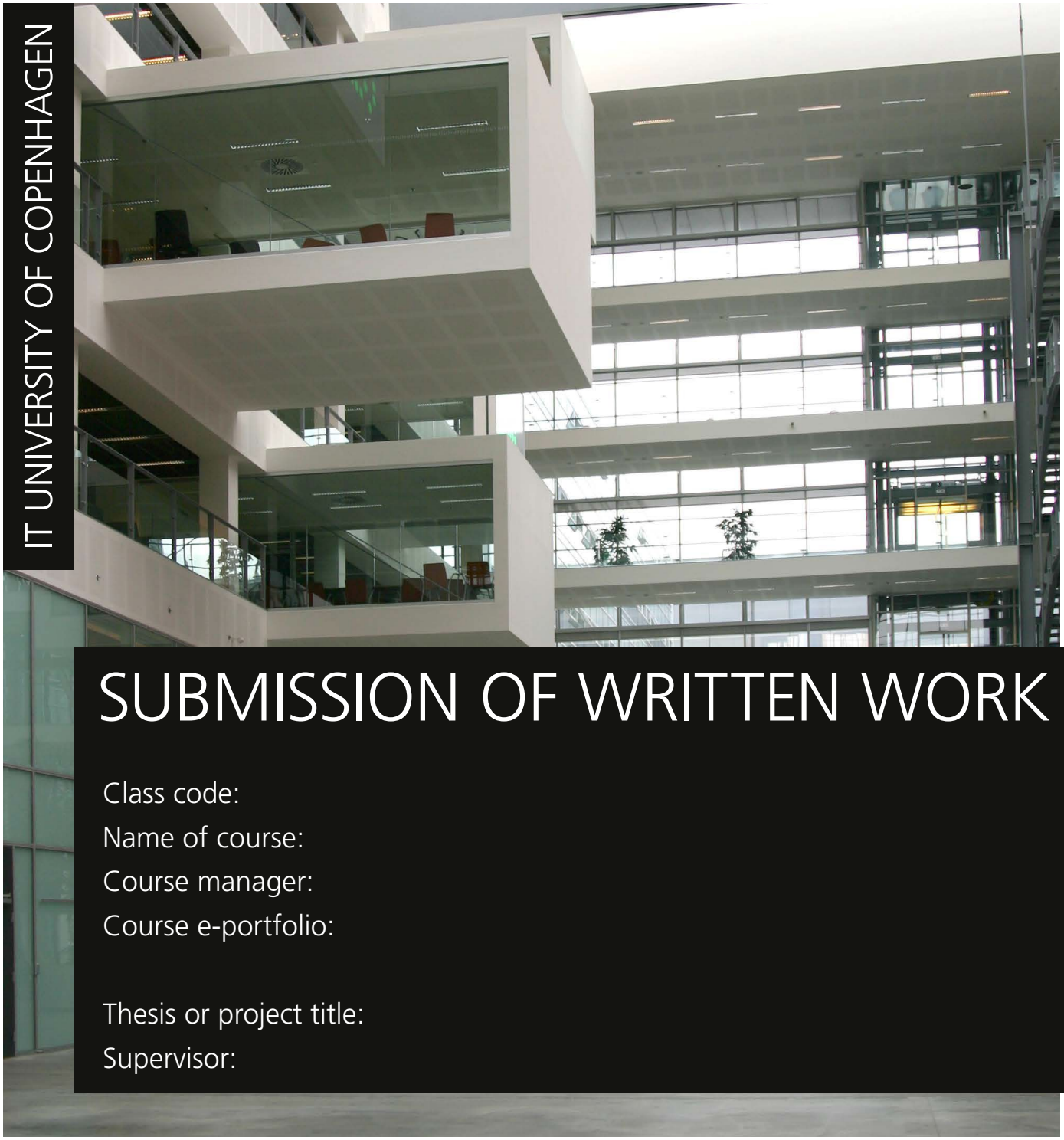
IT UNIVERSITY OF COPENHAGEN



SUBMISSION OF WRITTEN WORK

Class code:

Name of course:

Course manager:

Course e-portfolio:

Thesis or project title:

Supervisor:

Full Name:	Birthdate (dd/mm-yyyy):	E-mail:
1. _____	_____	_____@itu.dk
2. _____	_____	_____@itu.dk
3. _____	_____	_____@itu.dk
4. _____	_____	_____@itu.dk
5. _____	_____	_____@itu.dk
6. _____	_____	_____@itu.dk
7. _____	_____	_____@itu.dk
8. _____	_____	_____@itu.dk

Project: Search Engine

Group G*

Andrei Cobzaru René Haas Yonathan Volpin Maria Guardiola Muñoz
{adco,renha,yovo,mgmu}@itu.dk

December 10, 2018

Figure 1: Simply magical!

*AKA the amazing CodeUnicorns™

Contents

0.1 Statement of Contribution 4

1 Team protocol **4**

1.1 How we work 4

2 Overview of the CodeUnicorn Term-frequency search Engine **4**

2.1 UML diagram 4

3 Inverted Indices **6**

4 Refined Queries **10**

5 Ranking Algorithms **12**

6 Extension: Improving the Client GUI **13**

7 Extension: Webcrawler **13**

8 Extension: Fuzzy-Search **13**

Introduction

This document reports on the search engine project that we developed during the Introductory Programming course at the IT University of Copenhagen. In Sections 1–3 we will report on our solutions to the mandatory tasks posed in the project description. We also solved some of the challenges posed for these mandatory tasks.

- Include Challanges

Their solution is described in the respective section along with the solution to the mandatory task.

Furthermore, we developed the following extensions which are described in detail in the following sections:

- Include extensions

The description for each solution is roughly split up into the following parts:

- **Task:** A short review on the task that we had to solve.
- **Basic Approach:** A basic description on how we solved the task.
- **Technical description:** Description of software architecture used in the solution.
- **Testing considerations:** Description of considerations for testing the correctness of our solution.
- **Benchmarking/Reflection:** Benchmarking results for experiments that allowed us to choose the best data structure for the search engine.

We handed in the source code that accompanies this report as a single zip file called `GroupG.zip`. Furthermore `ip18groupG/src/main/java/searchengine/` contains the source code that solves the mandatory tasks. The full repository for our code is also available on ITU's Github: `https://github.itu.dk/wilr/ip18groupG/`.

0.1 Statement of Contribution

All authors contributed equally to all parts of the solution of the mandatory tasks.

1 Team protocol

1.1 How we work

We use github slack and overleaf.

- **Link to this document :** `https://v2.overleaf.com/project/5c0a4b2ec7d52344f3aee659`

2 Overview of the CodeUnicorn Term-frequency search Engine

2.1 UML diagram

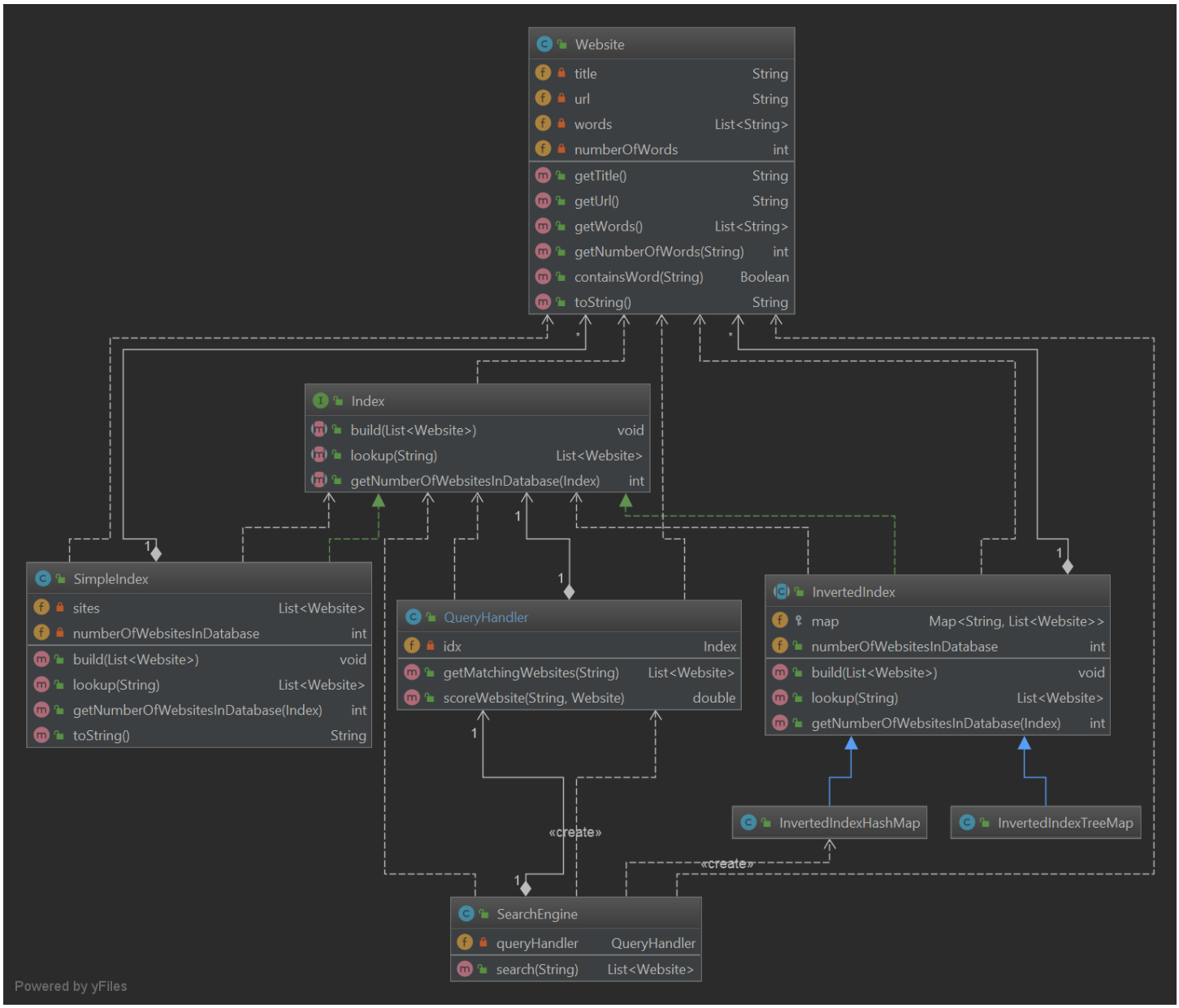


Figure 2: UML Diagram for the SearchEngine

3 Inverted Indices

Task: This task added an index data structure to the search engine. In particular, an inverted index data structure was implemented and experiments have been carried out to de cide which implementation of the inverted index should be used in the search engine.

An inverted index provides a fast way to access the websites that contain a certain word by storing a mapping between words and the list of websites that contain the word. Figure 3 provides an example for such a data structure.

And inverted index is a *map* from a *word* to a list of websites containing that word.

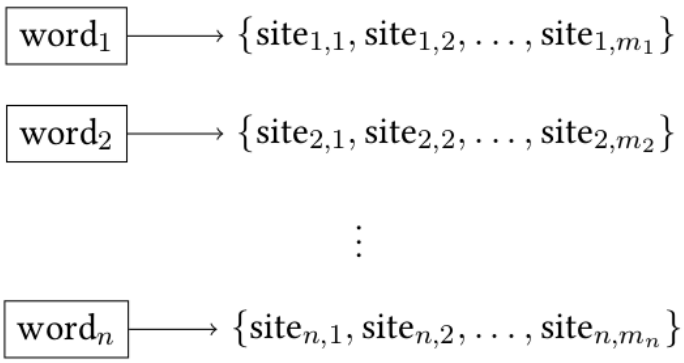


Figure 3: Diagram of inverted indices

Basic Approach: As per the code handout, we generalized the notion of an index into an interface `Index`, that defines the following two methods `build` and `lookup` see the javadoc in figure 4

Method Detail

build

```
void build(java.util.List<Website> sites)
```

The build method processes a list of websites into the index data structure.

Parameters:
`sites` - The list of websites that should be indexed

lookup

```
java.util.List<Website> lookup(java.lang.String query)
```

Given a query string, returns a list of all websites that contain the query.

Parameters:
`query` - The query

Returns:
the list of websites that contains the query word.

Figure 4: The javadoc for the `build` and `lookup` methods in the `InvertedIndex` class

We then implemented the inverted index by using Java's Map data structure. Here, Java provides different implementations most obvious a treemap and a hashmap respectively. We implemented and tested both solutions.

Technical description We build the index data structures as described in Figure 5.

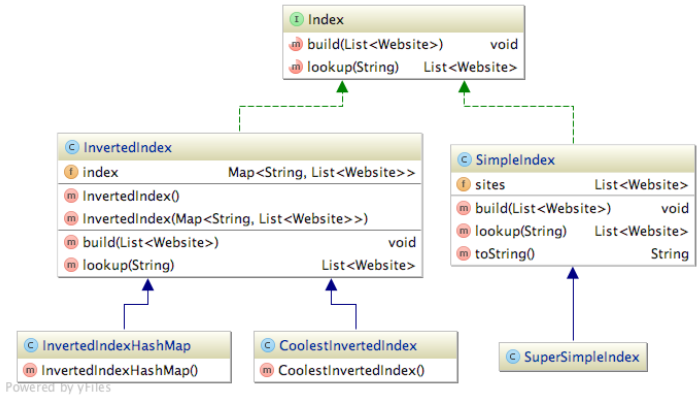


Figure 5: TEMPTLATE UML Diagram for the Software Architecture of Index data structures.

We allowed to switch the implementation of the Map data structure by adding a field to the search engine class and a if-else statement in the constructor see figure 6.

```
        //  add index switch
         if(indexType == "simple") {
             Index idx = new SimpleIndex();
         } else if (indexType == "tree") {
             Index idx = new InvertedIndexTreeMap();
         } else if (indexType == "hash") {
             Index idx = new InvertedIndexHashMap();
         }
```

Figure 6: Option to change indextype in search engine

With that in mind we are ready to define the `build` and `lookup` methods.

```
@Override
public void build(List<Website> sites) {
    // Task 1.
    for (Website website: sites) {
        for (String word: website.getWords()) {
            List<Website> existingWebsites = map.get(word);
            if (existingWebsites == null) {
                existingWebsites = new ArrayList<>();
            }
            existingWebsites.add(website);
            this.map.put(word, existingWebsites);
        }
    }
}

6 tests completed, 1 failed
```

Figure 7: andrei `build` build the map from websided method.

We implement the inverted index by adding a GETWORDS method to WEBSITE class see figure 9

```
    @Override
    public void build(List<Website> sites) {
        numberOfWebsInDB = sites.size();
        for (Website site: sites){
            List<String> allWords = site.getWords();
            for (String word : allWords){
                List<Website> sitesWithWord = map.get(word);
                if (sitesWithWord == null) sitesWithWord = new ArrayList<>();
                if (!sitesWithWord.contains(site)) sitesWithWord.add(site);
                this.map.put(word, sitesWithWord);
            }
        }
    }
```

Figure 8: maria / rene build

```
    public List<String> getWords() {
        return words;
    }
```

Figure 9: getWords

```
@Override
public List<Website> lookup(String query) {
    return this.map.get(query);
}
```

Figure 10: the `lookup` method

Testing considerations We verified the correctness of the `build` and `lookup` method with unit tests.

These tests can be found in the file SRC/TEST/JAVA/SEARCHENGINE/INDEXTEST.JAVA. Here, the `build` method of the `InvertedIndex` class was tested by

```
Test Duration Result
parseBadFile() 0.003s failed
parseGoodFile() 0.019s passed
```

The `lookup` method of all three indexes was checked by carefully creating a list of test websites and checking whether the lookup methods returns the expected results by looking at the following properties of the returned list: (i) its size and (ii) ... ? (Add Java code?)

Benchmarking/Reflection We tested which implementation provides the fastest running time for lookup operation by ... (describe your benchmarking setup / approach).

We choose the following query words for the benchmark: "denmark, archipelago, by, okhotsk, population, amager, the, uppsala, danish nordic, sea, climate, immigration, million, copenhagen, selfgoverning, km, which, urban, zealand".

This can be seen in the SRC/JMH/JAVA/SEARCHENGINE/INDEXBENCHMARK.JAVA file. The running times of different index implementations can be found in Table 1 (or a plot).

Index	Simple Index	InvertedIndex w/ HashMap	InvertedIndex w/ TreeMap
File: DATA/ENWIKI-TINY.TXT	35711.496	536.435	2171.606
File: DATA/ENWIKI-SMALL.TXT	15330842.249	550.919	4236.984
File: DATA/ENWIKI-MEDIUM.TXT	364731186.336	565.580	4898.113

Table 1: Running times for different index implementations.

These benchmark results indicate that ... . Based on these results, we decided that we use ... as the index in our data structure.

4 Refined Queries

Task:

- The or statement

Basic Approach: src/main/java/searchengine/QueryHandler.java

Technical description: this code takes the srearch sting input and returns a list of websides.

```
1  public List<Website> getMatchingWebsites(String line) {
2      List<Website> finalResult = new ArrayList<>();
3      String[] parts = line.split(" OR ");
4
5      // We will have to combine each of the individual results.
6      for (String part: parts) {
7          String[] words = part.split(" ");
8
9          // Initialize the partial result as the first search.
10         List<Website> partialResult = idx.lookup(words[0]);
11
12         // Intersect all the rest.
13         for (int i = 1; i < words.length; i++) {
14             List<Website> individualResult = idx.lookup(words[i]);
15             partialResult = intersectResults(partialResult, individualResult);
16         }
17
18         // Combine the results.
19         finalResult = combineResults(partialResult, finalResult);
20     }
21
22     return finalResult;
23 }
```

ADREI SOLUTION

```
1  private List<Website> intersectResults(List<Website> list1, List<Website> list2) {
2      List<Website> result = new ArrayList<>();
3
4      for (Website site: list1) {
5          if (list2.contains(site)) {
6              result.add(site);
7          }
8      }
9
10     return result;
11 }
```

ADREI SOLUTION

```
1  private List<Website> combineResults(List<Website> list1, List<Website> list2) {
2      List<Website> result = new ArrayList<>();
3      result.addAll(list1);
4
5      for (Website site: list2) {
6          if (! list1.contains(site)) {
7              result.add(site);
8          }
```

```
    }

    return result;
}
```

ADREI SOLUTION

Testing considerations:

Benchmarking/Reflection:

5 Ranking Algorithms

Task: 1. Add an interface Score to your program. It should have a method getScore that takes a string, a Website object, and an Index object, and returns a floating point number, for example double or float . 2. Implement the term frequency score in a class TFScore . 3. Implement the term frequency-inverse document score in a class TFIDFScore . 4. Modify your program such that each website that matches the query is assigned a score. For a multiple-word query, the score of the website is the sum of its score values for all words in the query. For an OR between two multiple-word queries, the score of the website is the maximum of the scores in these two queries. 5. Modify your program such that the list of websites that match a query is sorted by their score to the query in descending order. (Use methods provided in java.util!) 6. Provide queries and their results that show differences in the result quality, obtained by using these two score functions. Give a short description of how they differ.

Basic Approach:

Technical description: In AbstractScore

```
protected double tf(String w, Website S) {
    // This method counts the number of times
    // the word w occours in S.
    int count = 0;
    for(String word: S.getWords()) {
        if(w.equals(word)) {
            count++;
        }
    }
    return count;
}
```

Figure 11: Or implementation of the term frequency method. ADREI SOLUTION

```
private double idf(String w, List<Website> D) {
    int d = D.size(); // The total number of websites.
    int n = 0; // The number of websites the word occurs on.
    for (Website site: D) {
        if (site.containsWord(w)) {
            n++;
        }
    }

    return Math.log10(1.0 * n / d ); // TODO check the base of the log
}
```

Figure 12: Or implementation of the inverse document frequency method. ADREI SOLUTION

Testing considerations:

Benchmarking/Reflection:

6 Extension: Improving the Client GUI

We put some effort into optimizing the visual side of our Search engine by Improving the Client GUI. This was done by editing the STATIC/INDEX.HTML file

7 Extension: Webcrawler

8 Extension: Fuzzy-Search