APPENDIX I

Here we have attached the codes for various functions used during the project.

Code-1

The code that splits the document into tokens(words). A list of all distinct words is prepared for each of the documents

```
public class TfIdf
    //<editor-fold defaultstate="collapsed" desc="TF Calculator">
    /**
     * Calculated the tf of term termToCheck
     * @param totalterms : Array of all the words under processing document
     * @param termToCheck : term of which tf is to be calculated.
     * @return tf(term frequency) of term termToCheck
     */
    public double tfCalculator(String[] totalterms, String termToCheck) {
        double count = 0;  //to count the overall occurrence of the term termToCheck
        for (String s : totalterms) {
            if (s.equalsIgnoreCase(termToCheck)) {
                count++;
            }
        }
        return count / totalterms.length;
    }
    //</editor-fold>

    //<editor-fold defaultstate="collapsed" desc="Idf Calculator">
    /**
     * Calculated idf of term termToCheck
     * @param allTerms : all the terms of all the documents
     * @param termToCheck
     * @return idf(inverse document frequency) score
     */
    public double idfCalculator(List<String[]> allTerms, String termToCheck) {
        double count = 0;
        for (String[] ss : allTerms) {
            for (String s : ss) {
                if (s.equalsIgnoreCase(termToCheck)) {
                    count++;
                    break;
                }
            }
        }
        return 1+Math.log(allTerms.size() / count);
    }
//</editor-fold>
}
```

Code 2:

The code shown below computes the tf-idf weights for a text document.

```
package docsimilarity;

import java.io.FileNotFoundException;
import java.io.IOException;
import javax.swing.JFileChooser;

public class DocSimilarity {

    /**
     * @param args the command line arguments
     */
    /**
     * Main method
     * @param args
     * @throws FileNotFoundException
     * @throws IOException
     */
    public static void main(String[] args) throws FileNotFoundException, IOException
    {
        // TODO code application logic here
        /*
        String foldername = new String();
        JFileChooser chooser = new JFileChooser();
        chooser.setCurrentDirectory(new java.io.File("."));
        chooser.setDialogTitle("Select Folder Containing Documents");
        chooser.setFileSelectionMode(JFileChooser.DIRECTORIES_ONLY);
        chooser.setAcceptAllFileFilterUsed(false);

        int returnVal = chooser.showOpenDialog(chooser);
        if(returnVal == chooser.APPROVE_OPTION) {

            foldername = chooser.getSelectedFile().getAbsolutePath();
        }
else
        {

        }*/
          //System.out.println("C:\\Users\\ Desktop\\PROJECT\\folder");

        DocumentParser dp = new DocumentParser();
        dp.parseFiles("C:\\Users\\Desktop\\PROJECT\\folder");
        dp.tfIdfCalculator(); //calculates tfidf
        dp.getCosineSimilarity(); //calculated cosine similarity

    }

}
```

Code 3:

The code shown below computes the cosine similarity between two text documents:

```
package docsimilarity;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;

public class DocumentParser {

   //This variable will hold all terms of each document in an array.
   private List<String[]> termsDocsArray = new ArrayList<String[]>();
   private List<String> allTerms = new ArrayList<String>(); //to hold all terms
   private List<double[]> tfidfDocsVector = new ArrayList<double[]>();

   /**
    * Method to read files and store in array.
    * @param filePath : source file path
    * @throws FileNotFoundException
    * @throws IOException
    */
   public void parseFiles(String filePath) throws FileNotFoundException, IOException {
      File[] allfiles = new File(filePath).listFiles();
      BufferedReader in = null;
      for (File f : allfiles) {
         if (f.getName().endsWith(".txt")) {
            in = new BufferedReader(new FileReader(f));
            StringBuilder sb = new StringBuilder();
            String s = null;
            while ((s = in.readLine()) != null) {
               sb.append(s);
            }
            String st1 = sb.toString().replaceAll("[\\W&&[^\\s]]", "");
            st1 = st1.replaceAll("0", "");
            st1 = st1.replaceAll("1", "");
            st1 = st1.replaceAll("2", "");
            st1 = st1.replaceAll("3", "");
            st1 = st1.replaceAll("4", "");
            st1 = st1.replaceAll("5", "");
            st1 = st1.replaceAll("6", "");
            st1 = st1.replaceAll("7", "");
            st1 = st1.replaceAll("8", "");
            st1 = st1.replaceAll("9", "");
            String[] tokenizedTerms = st1.split(" ");       //\\W+");   to get individual terms
            //sb.toString().replaceAll(s, filePath)
            for (String term : tokenizedTerms) {
               if (!allTerms.contains(term)) {  //avoid duplicate entry
                  allTerms.add(term);
               }
            }
            termsDocsArray.add(tokenizedTerms);
         }
      }

   }
```

```
/**
 * Method to create termVector according to its tfidf score.
 */
public void tfIdfCalculator() {
    double tf; //term frequency
    double idf; //inverse document frequency
    double tfidf; //term requency inverse document frequency
    for (String[] docTermsArray : termsDocsArray) {
        double[] tfidfvectors = new double[allTerms.size()];
        int count = 0;
        for (String terms : allTerms) {
            tf = new TfIdf().tfCalculator(docTermsArray, terms);
            idf = new TfIdf().idfCalculator(termsDocsArray, terms);
            tfidf = tf * idf;
            //System.out.println(tfidf);
            tfidfvectors[count] = tfidf;
            count++;
        }
        tfidfDocsVector.add(tfidfvectors);  //storing document vectors;
        //System.out.println(" ");
        //System.out.println("count = "+count);
        //System.out.println(" ");
    }}

/**
 * Method to calculate cosine similarity between all the documents.
 */
public void getCosineSimilarity() {
    //for (int i = 0; i < tfidfDocsVector.size(); i++) {
    //    for (int j = 0; j < tfidfDocsVector.size(); j++) {
            System.out.println("between " + 1 + " and " + 2 + " = "
                    + new CosineSimilarity().cosineSimilarity
                    (
                     tfidfDocsVector.get(0),
                     tfidfDocsVector.get(1)
                    )
                );
    //    }
    //}
}

}
```

```
 * Method to calculate cosine similarity between two documents.
 * @param docVector1 : document vector 1 (a)
 * @param docVector2 : document vector 2 (b)
 * @return
 */
public double cosineSimilarity(double[] docVector1, double[] docVector2) {
    double dotProduct = 0.0;
    double magnitude1 = 0.0;
    double magnitude2 = 0.0;
    double cosineSimilarity = 0.0;
```

```
//System.out.println(" docVector1.length = "+docVector1.length);
        //System.out.println(" docVector2.length = "+docVector2.length);
        for (int i = 0; i < docVector1.length; i++) //docVector1 and docVector2 must be of same length
        {
           dotProduct += docVector1[i] * docVector2[i];  //a.b
           magnitude1 += Math.pow(docVector1[i], 2);  //(a^2)
           magnitude2 += Math.pow(docVector2[i], 2); //(b^2)
         System.out.println(" docVector1[" + i + "]: "+docVector1[i] + " docVector2[" + i + "]: "+docVector2[i]);
        }

        magnitude1 = Math.sqrt(magnitude1);//sqrt(a^2)
        magnitude2 = Math.sqrt(magnitude2);//sqrt(b^2)

        if (magnitude1 != 0.0 | magnitude2 != 0.0) {
           cosineSimilarity = dotProduct / (magnitude1 * magnitude2);
        } else {
           return 0.0;
        }
        return cosineSimilarity;
    }
}
```