



Web Interface for FlexiTerm

Adedamola Agbonyin

May, 2018

Supervisor: Irena Spasic'

Moderator: Jing Wu

CM3203: One Semester Individual Project

40 credits

School of Computer Science and Informatics
Cardiff University

Acknowledgement

I would like to use this medium to express gratitude to family, friends and fellow students who have supported me throughout this project. Special thanks to the Cardiff University IT service desk for their kind assistance towards the technical issues I faced, and my supervisor, Dr. Irena Spasić for her guidance and constructive feedback.

Abstract

This report documents the work done towards developing a web interface for the FlexiTerm Java application. The current method of accessing FlexiTerm requires the user to download the source files and follow a list of instructions to run the program. The aim of this project is therefore to provide an online demo app through which users can directly interact with FlexiTerm.

The outcome of this project is a web application which maintains the functionality of FlexiTerm and provides a web-based user interface through which users can enter input in different formats, view and download output files. The requirements documented in [Section 2.1](#) were converted into a design following in the steps of existing online term recognition tools. After development and comprehensive testing with test cases, usability testing was conducted among a group of users to assess user satisfaction.

The results of the usability testing show that the FlexiTerm web app has higher perceived usability than 70% of existing systems, thus demonstrating that an online demo is not only feasible, but can provide an alternative, more accessible method of using the FlexiTerm tool.

The FlexiTerm web demo can be accessed online at users.cs.cf.ac.uk/AgbonyinAT/demo.html

Table of Contents

1.0 Introduction	7
1.1 Defining the Problem	7
1.2 Similar Tools	8
1.2.1 Termine	8
1.2.2 Terminology Extraction	10
1.3 Target Users and Beneficiaries	12
1.4 Project Scope	12
1.5 Approach	12
1.6 Assumptions	14
1.7 Outcomes	14
2.0 Specification and Design	15
2.1 Requirements Definition and Analysis	15
2.1.1 Functional Requirements	16
2.1.2 Non-functional Requirements	17
2.2 Design	18
2.2.1 System Architecture	18
2.2.2 Database Design (Relevant Tables)	20
2.2.3 Interface Design	21
3.0 Methodology	28
3.1 Development Tools and Languages	28
3.1.1 Client Side	28
3.1.2 Server Side	30
3.2. Project Management	30
3.3. Version Control	31
4.0 Risk Assessment	33
5.0 Implementation	35
5.1 Summary of Features	35
5.2 Features	37
5.2.1 Homepage and User Input	37
5.2.2 Input Validation	38
5.2.3 Input processing	39
5.2.4 Results Page	43
	3

5.2.5 Coding Standards	46
5.2.6 Browser Compatibility	47
6.0 Testing and Evaluation	48
6.1 Functional & Non-functional Testing	48
6.1.1 Functional Testing	49
6.1.2 Non-functional Requirements	50
6.2 Usability Testing	51
6.3 Evaluation	54
7.0 Conclusion	57
8.0 Future Work	58
9.0 Learning and Reflection	59
10.0 References	61
Appendix A	63
Appendix B	68

Table of Figures

Figure 1: Successful run of FlexiTerm from the command line	5
Figure 2: Flexiterm Output files	8
Figure 3: Termine web demo with input	9
Figure 4: Termine output - HTML Format	9
Figure 5: Termine output - Plain text format	10
Figure 6: Termine output - table format	10
Figure 7: Termine output - term details	10
Figure 8: Termine output - term variants for TNF-alpha	10
Figure 9: Terminology Extraction demo with input	11
Figure 10: Terminology Extraction output	11
Figure 11: Waterfall Model	13
Figure 12: Agile Model	13
Figure 13: Waterfall-Agile Hybrid Model	14
Figure 14: Client-server model	18
Figure 15: Single-tier Application	19
Figure 16: Table term_normalised	20
Figure 17: Table term_phrase	20
Figure 18: Table term_termhood	21
Figure 19: Table data_document	21
Figure 20: Table output_table	21
Figure 21: Demo home page wireframe	22
Figure 22: HTML table wireframe	22
Figure 23: Table tab wireframe	23
Figure 24: Download tab wireframe	23
Figure 25: Activity diagram - enter input	25
Figure 26: Activity Diagram - view output	26
Figure 27: Activity Diagram - download output	27
Figure 28: Plain HTML Table	29
Figure 29: DataTable data table	29
Figure 30: Trello Weekly Board	31
Figure 31: GitLab Version Control	32
Figure 32: GitLab milestones	32
Figure 33: Location of demo in homepage menu	37
Figure 34: Demo Homepage	37
Figure 35: File upload - wrong format error	38
Figure 36: File upload - too large file error	38
Figure 37: Text input - input greater than limit error	39
Figure 38: Url input - invalid format	39
Figure 39: Input form - No input	39
Figure 40: Flex.php - semaphore file	30

Figure 41: Server is busy	41
Figure 42: Flex.php - <i>processInput()</i> function	41
Figure 43: Flex.php - <i>saveResultsToSession()</i> function	42
Figure 44: Flex.php - <i>copyOutputFiles()</i> function	42
Figure 45: CORS error	43
Figure 46: PHP CORS headers	43
Figure 47: Results.php - Default view	44
Figure 48: Expanded table row	45
Figure 49: Extracted terms in context	45
Figure 50: Downloads tab	46
Figure 51: Browser market share	47
Figure 52: Diminishing Returns for usability testing	53

1.0 Introduction

1.1 Defining the Problem

Term recognition is a method of extracting technical terms that are relevant to a particular domain or corpus from texts. In this context, a term is a word or set of words used to express a concept in a domain or branch of study e.g. biomedicine. Term recognition can be automated with computer programs, and one of such programs is FlexiTerm.

FlexiTerm is a command-line Java application for automatic recognition of multiword terms in texts such as medical journals and research papers. It works by first performing linguistic filtering to select term candidates, followed by a “frequency-based measure (...) to qualify a candidate as a term” [Spasić et al., 2013, p.1]. One key feature of FlexiTerm is that it is built to allow for term variants, which is suitable for “less formal texts such as patient blogs or medical notes” [Spasić et al., 2013, p.1]. The web page <http://users.cs.cf.ac.uk/I.Spasic/flexiterm/> contains details about FlexiTerm including relevant publications and download links.

To run the application, the user will have to do the following:

1. Download the FlexiTerm source files at <https://sourceforge.net/projects/flexiterm/> - about 63.7mb zipped and 286.8mb unzipped.
2. Install Java locally.
3. Put plain text input files in a folder.
4. Run the start script. This can either be *FlexiTerm.sh* or *FlexiTerm.bat* depending on the user’s operating system. If run successfully, output files *output.html*, *output.txt*, *output.mixup* and *output.csv* are created in the same folder.
5. Click each output file to view it.

```
[pc-65-3:src 4 Adedamola$ sh FlexiTerm.sh
Loading default properties from trained tagger ./models/left3words-wsj-0-18.tagger
er
Reading POS tagger model from ./models/left3words-wsj-0-18.tagger ... done [0.6
sec].
Apr 26, 2018 2:26:20 AM edu.stanford.nlp.process.PTBLexer next
WARNING: Untokenizable: (U+1, decimal: 1)
```

Figure 1: Successful run of FlexiTerm from the command line

Name	Date Modified	Size	Kind
dict	30 Jun 2017 at 1:59 PM	--	Folder
dictionary.txt	Yesterday at 2:14 PM	Zero bytes	Plain Text
FlexiClustering.bat	1 Aug 2017 at 11:33 AM	91 bytes	Batch File
FlexiClustering.class	29 Oct 2017 at 12:59 PM	16 KB	Java class file
FlexiClustering.java	13 Nov 2017 at 4:22 PM	36 KB	Java Source
FlexiClustering.log	29 Oct 2017 at 12:59 PM	12 KB	Log File
FlexiClustering\$Cluster.class	29 Oct 2017 at 12:59 PM	2 KB	Java class file
FlexiTerm.bat	1 Aug 2017 at 11:35 AM	253 bytes	Batch File
FlexiTerm.class	8 Aug 2017 at 9:45 AM	40 KB	Java class file
FlexiTerm.java	8 Aug 2017 at 9:44 AM	88 KB	Java Source
FlexiTerm.log	15 Aug 2017 at 11:47 AM	11 MB	Log File
FlexiTerm.sh	Yesterday at 2:14 PM	234 bytes	Shell Script
flexiterm.sqlite	Today at 2:26 AM	183.2 MB	SQLite database
lib	1 Aug 2017 at 11:35 AM	--	Folder
mixup_demo.bat	31 May 2017 at 1:22 PM	152 bytes	Batch File
models	30 Jun 2017 at 1:59 PM	--	Folder
output.csv	Today at 2:26 AM	40 bytes	Comm...et (.csv)
output.html	Today at 2:26 AM	455 bytes	HTML
output.mixup	Today at 2:26 AM	Zero bytes	Document
output.txt	Today at 2:26 AM	Zero bytes	Plain Text
Porter.class	26 May 2011 at 4:20 PM	5 KB	Java class file
Porter.java	26 May 2011 at 4:20 PM	12 KB	Java Source
README.TXT	15 Aug 2017 at 11:45 AM	4 KB	Plain Text
sample.txt	Yesterday at 3:40 AM	2 KB	Plain Text
settings.txt	15 Aug 2017 at 11:46 AM	159 bytes	Plain Text
stoplist.txt	5 Aug 2017 at 1:02 PM	8 KB	Plain Text
text	Yesterday at 2:16 PM	--	Folder
text.html	Today at 2:26 AM	6 KB	HTML
text.labels	Today at 2:26 AM	1 byte	Document
tmp.txt?	Today at 2:26 AM	27 KB	Document
WordNet.class	4 Jun 2011 at 12:07 PM	2 KB	Java class file
WordNet.java	4 Jun 2011 at 12:07 PM	2 KB	Java Source

Figure 2: Flexiterm Output files

This is a relatively long process and can be complicated for some users. For example, one may not know whether/how to run the *FlexiTerm.sh* or the *FlexiTerm.bat* script, or may be wary of downloading software files off the internet. Regardless, it is beneficial to provide a simpler alternative through which users can make use of FlexiTerm, in addition to having the option to download and run it locally if they wish.

Thus, the aim of this project is to include a web demo in the FlexiTerm homepage at <http://users.cs.cf.ac.uk/I.Spasic/flexiterm/> which will combine the above list of steps into a simple and easy process: users will simply enter their input in one of three formats, submit it and view/download the output, without having to download or run any code.

1.2 Similar Tools

In order to get a clear vision of how the FlexiTerm demo would look and behave, I researched existing online term extraction tools: Termine by NaCTeM [Nactem, 2012] and the Terminology Extraction demo by Translated Labs [Translated, n.d].

1.2.1 Termine

Termine is an automatic term recognition tool developed by the National Centre for Text Mining (NaCTeM), which is operated by the University of Manchester.

Web Demonstration

☒ Plain text (Only ASCII characters allowed)

NF-kappa B-containing plasmids correlated directly with induction of NF-kappa B DNA-binding activity. Although the intact beta interferon promoter was only weakly stimulated by phorbol ester or TNF-alpha, multimers of the PRDII NF-kappa B-binding domain were inducible by both agents. TNF-alpha was able to increase expression of the HIV LTR in T cells, but in monocytic cells, TNF-alpha did not induce the HIV LTR above a constitutive level of activity. This level of NF-kappa B-independent activity appears to be sufficient for virus multiplication, since TNF-alpha treatment had no effect on the kinetics of de novo HIV type 1 (HIV-1) infection and viral RNA production in U937 cells. However, in Jurkat cells, TNF-alpha dramatically enhanced the spread of HIV-1 through the cell population and increased viral RNA synthesis, indicating that in T cells HIV-1 multiplication was stimulated by TNF-alpha treatment.

☐ Local text file (*.txt file in ASCII encoding or *.pdf file; 2MB maximum)
Browse... No file selected.

☐ URL (HTML or PDF content; 2MB maximum)

POS tagger: GENIA Tagger version 2.1 ☒ Preserve break lines

Figure 3: Termine web demo with input

How Termine Web Demonstration works:

1. The user enters input in any of three formats: plain text, file upload or url linking. Alternatively the user can run the demo using sample input provided with options “Try (MEDLINE sample)” and “Try (NaCTeM sample)”.
2. The user selects a POS tagger (optional).
3. The user presses the “Analyze” button and the demo page is redirected to a results page where results can be displayed in three formats: Html, plain text and table formats.

Output

TerMine (C-value) analysis

[Service questionnaire](#)

Found 46 terms in 8.22 seconds - all terms ([in table](#)) ([in text](#)) - threshold: 0 Apply

Cell-specific differences in activation of NF-kappa B regulatory elements of human immunodeficiency virus and beta interferon promoters by tumor necrosis factor. Three aspects of the involvement of tumor necrosis factor in human immunodeficiency virus (HIV) pathogenesis were examined. Tumor necrosis factor alpha (TNF-alpha) mRNA production was analyzed by polymerase chain reaction amplification in monocytic U937 cells and in a chronically HIV infected U937 cell line (U9-IIIB). TNF-alpha RNA was undetectable in U937 cells, whereas a low constitutive level was detected in U9-IIIB cells. Paramyxovirus infection induced a 5- to 10-fold increase in the steady-state level of TNF-alpha RNA in U9-IIIB cells compared with U937 cells, suggesting that HIV-infected monocytic cells produced higher levels of TNF-alpha than did normal cells after a secondary virus infection. The effects of TNF-alpha on gene expression were examined by transient expression assays using reporter chloramphenicol acetyltransferase plasmids linked to regulatory elements from the HIV long terminal repeat (LTR) and the beta interferon promoter. In U937 and Jurkat T lymphoid cells, the inducibility of the different hybrid promoters by TNF-alpha or phorbol ester varied in a cell type- and promoter context-specific manner; the levels of gene activity of NF-kappa B-containing plasmids correlated directly with induction of NF-kappa B DNA-binding activity. Although the intact beta interferon promoter was only weakly stimulated by phorbol ester or TNF-alpha, multimers of the PRDII NF-kappa B-binding domain were inducible by both agents. TNF-alpha was able to increase expression of the HIV LTR in T cells, but in monocytic cells, TNF-alpha did not induce the HIV LTR above a constitutive level of activity. This level of NF-kappa B-independent activity appears to be sufficient for virus multiplication, since TNF-alpha treatment had no effect on the kinetics of de novo HIV type 1 (HIV-1) infection and viral RNA production in U937 cells. However, in Jurkat cells, TNF-alpha dramatically enhanced the spread of HIV-1 through the cell population and increased viral RNA synthesis, indicating that in T cells HIV-1 multiplication was stimulated by TNF-alpha treatment.

Thank you for using TerMine. Please now complete a [questionnaire](#) to let us know your views about this service.

Figure 4: Termine output - HTML Format

```

1  beta interferon promoter 4.754888
2  u937 cell 4
3  human immunodeficiency virus 3.169925
3  tumor necrosis factor 3.169925
5  hiv infected u937 cell line 2.321928
6  nf-kappa b dna-binding activity 2
6  nf-kappa b regulatory element 2
6  hiv long terminal repeat 2
6  jurkat t lymphoid cell 2
6  hiv ltr 2
6  tnfr-alpha rna 2
6  phorbol ester 2
6  prdii nf-kappa b-binding domain 2
6  tumor necrosis factor alpha 2
6  t cell hiv-1 multiplication 2
6  reporter chloramphenicol acetyltransferase plasmid 2
6  polymerase chain reaction amplification 2
6  u9-iiib cell paramyxovirus infection 2
19  secondary virus infection 1.584962
19  transient expression assay 1.584962
19  hiv-infected monocytic cell 1.584962
19  nf-kappa b-containing plasmid 1.584962
19  monocytic u937 cell 1.584962
19  nf-kappa b-independent activity 1.584962
19  viral rna synthesis 1.584962
19  viral rna production 1.584962
27  t cell 1
27  tnfr-alpha treatment 1
27  monocytic cell 1
27  gene activity 1
27  cell population 1
27  agent tnfr-alpha 1
27  regulatory element 1
27  hiv type 1
27  jurkat cell 1
27  mrna production 1
27  u9-iiib cell 1
27  virus multiplication 1
27  gene expression 1
27  normal cell 1
27  hybrid promoter 1
27  cell type- 1

```

Figure 5: Termine output - Plain text format

Rank	Term	Score
1	beta interferon promoter	4.754888
2	u937 cell	4
3	human immunodeficiency virus	3.169925
3	tumor necrosis factor	3.169925
5	hiv infected u937 cell line	2.321928
6	nf-kappa b dna-binding activity	2
6	nf-kappa b regulatory element	2
6	hiv long terminal repeat	2
6	jurkat t lymphoid cell	2
6	hiv ltr	2
6	tnfr-alpha rna	2
6	phorbol ester	2
6	prdii nf-kappa b-binding domain	2
6	tumor necrosis factor alpha	2
6	t cell hiv-1 multiplication	2
6	reporter chloramphenicol acetyltransferase plasmid	2
6	polymerase chain reaction amplification	2
6	u9-iiib cell paramyxovirus infection	2
19	secondary virus infection	1.584962
19	transient expression assay	1.584962
19	hiv-infected monocytic cell	1.584962
19	nf-kappa b-containing plasmid	1.584962
19	monocytic u937 cell	1.584962
19	nf-kappa b-independent activity	1.584962

Figure 6: Termine output - table format

As seen in Figures 4, 5 and 6, Termine output is displayed in three views: HTML, plain text and table respectively. The HTML view is the default and shows the original input text as entered by the user, with the words and phrases identified as terms highlighted in pink. Hovering over any of these terms shows its c-value or termhood score and rank [Figure 7]. Term ranks are determined by the termhood score; terms with the same score are given the same rank. Words with multiple variants, such as abbreviations, are highlighted in green and hovering over them displays a list of “possible expanded forms” [Figure 8].

C-Value score: 2 (#6)
[Search this term with Cheshire3](#)

Figure 7: Termine output - term details

Possible expanded forms: tumor necrosis factor-alpha, tumour necrosis factor-alpha, tumor necrosis factor type alpha, tnfr alpha, tissue necrosis factor-alpha, alpha tumor necrosis factor, transforming necrosis factor alpha, tumor necrotic factor-alpha, tumor necrosis factor-alpha

Figure 8: Termine output - term variants for TNF-alpha

The plain text and table views are similar to each other. They both contain the list of extracted terms and their termhood scores, ordered by rank in descending order

1.2.2 Terminology Extraction

This is a term extraction tool developed by Translated Labs.

How Terminology Extraction Works:

Enter input in the text box and click “Terminology Extraction” to submit.

Insert the text requiring terminology extraction.

For long texts, extraction could take up to a minute.

Examples: [Chemistry](#) | [Computational Linguistics](#) |

Cell-specific differences in activation of NF-kappa B regulatory elements of human immunodeficiency virus and beta interferon promoters by tumor necrosis factor. Three aspects of the involvement of tumor necrosis factor in human immunodeficiency virus (HIV) pathogenesis were examined. Tumor necrosis factor alpha (TNF-alpha) mRNA production was analyzed by polymerase chain reaction amplification in monocytic U937 cells and in a chronically HIV infected U937 cell line (U9-IIIB). TNF-alpha RNA was undetectable in U937 cells, whereas a low constitutive level was detected in U9-IIIB cells. Paramyxovirus infection induced a 5- to 10-fold increase in the steady-state level of TNF-alpha RNA in U9-IIIB cells compared with U937 cells, suggesting that HIV-infected monocytic cells produced higher levels of TNF-alpha than did normal cells after a secondary virus infection. The effects of TNF-alpha on gene expression were examined by transient expression assays using reporter chloramphenicol acetyltransferase plasmids linked to regulatory elements from the HIV long terminal repeat (LTR) and the beta interferon promoter. In U937 and Jurkat T lymphoid cells, the inducibility of the different hybrid promoters by TNF-alpha or phorbol ester varied in a cell type- and promoter context-specific manner; the levels of gene activity of NF-kappa B-containing plasmids correlated directly with induction of NF-kappa B DNA-binding activity. Although the intact beta interferon promoter was only weakly stimulated by phorbol ester or TNF-alpha, multimers of the PRDII NF-kappa B-binding domain were inducible by both agents. TNF-alpha was able to increase expression of the HIV LTR in T cells, but in monocytic cells, TNF-alpha did not induce the HIV LTR above a constitutive level of activity. This level of NF-kappa B-independent activity appears to be sufficient for virus multiplication, since TNF-alpha treatment had no effect on the kinetics of de novo HIV type 1 (HIV-1) infection and viral RNA production in U937 cells. However, in Jurkat cells, TNF-alpha dramatically enhanced

Languages: English Terminology Extraction

Figure 9: Terminology Extraction demo with input

Output:

Top 20 terms		
#	Extracted term	Score
1	tumor necrosis factor	71%
2	tumor necrosis factor alpha	64%
3	polymerase chain reaction amplification	61%
4	beta interferon	61%
5	beta interferon promoter	59%
6	reporter chloramphenicol acetyltransferase	59%
7	phorbol ester	59%
8	tnf-alpha rna	57%
9	intact beta interferon	56%
10	reporter chloramphenicol acetyltransferase plasmid	56%
11	prdii nf-kappa b-binding domain	56%
12	chloramphenicol acetyltransferase plasmids linked	55%
13	viral rna	55%
14	monocytic cells	55%
15	promoter context-specific	55%
16	jurkat t lymphoid cells	55%
17	hiv ltr	54%
18	intact beta interferon promoter	54%
19	assays using reporter chloramphenicol	54%
20	phorbol ester varied	53%

Terms in context	
<p>Cell-specific differences in activation of NF-kappa B regulatory elements of human immunodeficiency virus and beta interferon promoters by tumor necrosis factor. Three aspects of the involvement of tumor necrosis factor in human immunodeficiency virus (HIV) pathogenesis were examined. Tumor necrosis factor alpha (TNF-alpha) mRNA production was analyzed by polymerase chain reaction amplification in monocytic U937 cells and in a chronically HIV infected U937 cell line (U9-IIIB). TNF-alpha RNA was undetectable in U937 cells, whereas a low constitutive level was detected in U9-IIIB cells. Paramyxovirus infection induced a 5- to 10-fold increase in the steady-state level of TNF-alpha RNA in U9-IIIB cells compared with U937 cells, suggesting that HIV-infected monocytic cells produced higher levels of TNF-alpha than did normal cells after a secondary virus infection. The effects of TNF-alpha on gene expression were examined by transient expression assays using reporter chloramphenicol acetyltransferase plasmids linked to regulatory elements from the HIV long terminal repeat (LTR) and the beta interferon promoter. In U937 and Jurkat T lymphoid cells, the inducibility of the different hybrid promoters by TNF-alpha or phorbol ester varied in a cell type- and promoter context-specific manner; the levels of gene activity of NF-kappa B-containing plasmids correlated directly with induction of NF-kappa B DNA-binding activity. Although the intact beta interferon promoter was only weakly stimulated by phorbol ester or TNF-alpha, multimers of the PRDII NF-kappa B-binding domain were inducible by both agents. TNF-alpha was able to increase expression of the HIV LTR in T cells, but in monocytic cells, TNF-alpha did not induce the HIV LTR above a constitutive level of activity. This level of NF-kappa B-independent activity appears to be sufficient for virus multiplication, since TNF-alpha treatment had no effect on the kinetics of de novo HIV type 1 (HIV-1) infection and viral RNA production in U937 cells. However, in Jurkat cells, TNF-alpha dramatically enhanced the spread of HIV-1 through the cell population and increased viral RNA synthesis, indicating that in T cells HIV-1 multiplication was stimulated by TNF-alpha treatment.</p>	

Figure 10: Terminology Extraction output

The output for Terminology Extraction is displayed similarly to Termine. The top 20 terms are ranked according to their scores and also highlighted in the context of the original input text.

With these tools in mind, it was much easier to understand the requirements for the FlexiTerm web demo's functionality and design the user interface accordingly. I decided to adopt the Termine feature which allows users to enter input in url, file upload and plain text formats, single-colour term highlighting in context, and term variants if applicable. I also adopted the ranking system of Terminology Extraction where terms with the same scores will not have the same rank; rank is incremented instead. This appears to be more readable and less confusing for the user.

1.3 Target Users and Beneficiaries

An application is only successful if it brings value to its users. One way of identifying potential users is to draw from a general understanding based on the purpose and functionality of FlexiTerm. FlexiTerm is a term recognition tool, so it is expected that the potential users of the demo will likely consist of data scientists, researchers, linguists, or generally anyone in need of a text mining/recognition tool.

1.4 Project Scope

The scope of this project was limited to only providing an interface for FlexiTerm, with no changes to the underlying Java program. It includes the following:

1. Design and implement a full-stack web application to demonstrate the functionality of FlexiTerm.
2. Users can enter their desired input to be processed by the FlexiTerm application.
3. Output will be displayed in a variety of formats.
4. Output files can be downloaded.

Due to time and feasibility constraints, other features of FlexiTerm such as allowing the user to edit the list of stop words are declared out of scope and have instead been added to future work.

1.5 Approach

To approach this software project, I considered two development methodologies: Agile Methodology and Waterfall Methodology. With Waterfall, progress cascades downwards like a waterfall from requirements analysis, design, implementation, testing and finally, maintenance. The advantage of Waterfall is that it is simple to understand and implement. There is emphasis on clear milestones and the phases do not overlap, so each phase must be fully completed and reviewed before moving on to the next.

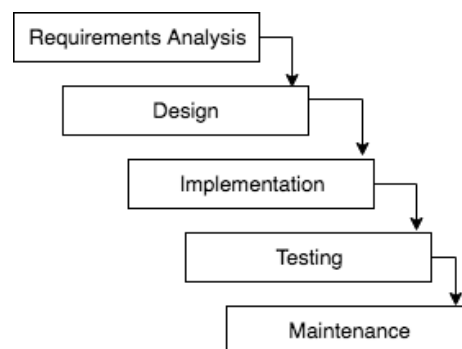


Figure 11: Waterfall Model

Waterfall is best-suited to small projects such as this, but which have clearly defined requirements that are not expected to change down the line. As planning is thorough, it ensures that the requirements and system design are feasible before any development takes place. The main disadvantage of Waterfall is that it is not very flexible [Lonergan, 2016]. All requirements are defined at the initial stage and cannot be revisited; this may make it difficult to make changes to the application if testing - particularly usability testing at the end of the project- reveals poor usability, incomplete implementation of requirements or any serious unhandled errors.

Agile, on the other hand, is a lot more flexible in comparison [Lonergan, 2016]. Progress is iterative such that design, development, implementation and testing take place in short sprints, with client feedback at the end of each sprint. This makes it easier to adapt to any requirement changes and identify and fix bugs early on.

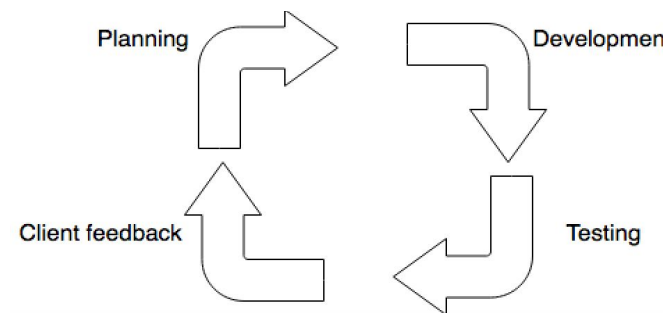


Figure 12: Agile Model

Although my intention was to use Agile as described in my initial report, I eventually went with a different approach. Having had experience with both methodologies, I decided that rather than use either pure Agile or pure Waterfall, to combine the strengths of the two into an iterative-incremental hybrid methodology [Inflectra, 2018]. This combines the rigid incremental feature of Waterfall with the flexible iterative feature of Agile. In this method, the traditional Waterfall approach of initial planning, requirements gathering and design are first followed. Then the development process is carried out in Agile form, with short (1 - 2 week) sprints of development, testing and receiving client feedback. Changes are made according to the feedback and the next sprint starts for the next functionality to be implemented. After all requirements have been implemented, final integration testing is performed and the application is deployed.

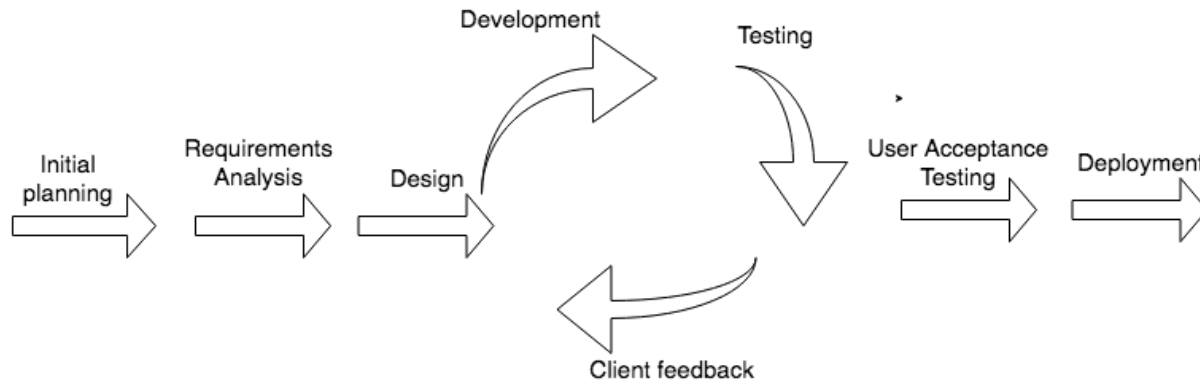


Figure 13: Waterfall-Agile Hybrid Model

1.6 Assumptions

The following assumptions are made regarding this project:

1. Although this is quite unlikely, the FlexiTerm web app is designed to handle more than one user at a time. One user at a time is the typical case as the FlexiTerm Java app utilises a SQLite database file which does not support concurrent write requests.
2. User input is in plain text format regardless of input method. This is to enable easier processing by FlexiTerm.
3. The user is connected to the internet and has Javascript enabled.
4. The server the app is to be hosted on is live.
5. The app is accessed with a desktop computer (as opposed to mobile). To save time, only desktop view was considered. Mobile optimisation will be included in future work.
6. The finished app is to be hosted on the Cardiff University Users web server for public access and integrated into the FlexiTerm homepage at <http://users.cs.cf.ac.uk/I.Spasic/flexiterm/>.

1.7 Outcomes

The main outcomes of this project are:

1. A web application with the functionality of FlexiTerm. FlexiTerm is an open source tool [Spasić et al., 2013, p.3], so it is important to ensure that the demo code is functional, readable, and can easily be maintained and extended by other developers in the future.
2. Documentation of the finished project with this report, including full code base, design and implementation process, complete list of requirements, functionality and usability testing results, and other essential information.

2.0 Specification and Design

To accomplish the purpose of this project which is to develop a web interface for FlexiTerm, the first step is to define the system's needs.

2.1 Requirements Definition and Analysis

In software engineering, this is the first stage and one of the most important aspects; the requirements are first defined, then analysed and categorised as functional or non-functional.

“Requirements definition is the process during which the needs of the customer are translated into a clear, detailed specification of what the system must do” [Landis et al., 1992]. Requirements analysis takes place after the initial specifications have been agreed upon. In this stage, the developer assesses the requirements “for completeness and feasibility” [Landis et al., 1992]. The requirements must be relevant to the project and also specific, realistic, actionable and measurable. Adopting Agile test-driven development is to ensure that the requirements are written to be “individually testable” so user acceptance tests (UATs) are also written during this phase. The finished product will then be tested against the client's requirements to ensure that they have been correctly implemented and are deemed “accepted” [Marsic, 2012].

One way of documenting system requirements is the use of IEEE 830 statements, a.k.a. “The system shall...” sentences. An alternative is Agile user stories wherein the requirements are expressed as a description of a piece of functionality from the perspective of the user [Cohn, 2004]. An example of a user story is: “as a user, I can enter my input in three formats: plain text, url and file upload”. User stories are much lighter than other methods of requirements recording which makes it an ideal method for situations where requirements are expected to change frequently [Hayward, 2013].

In this case however, statement requirements are more appropriate as I already have a clear example of what the end product should look and behave (Termine), so rather than having to hold several client/stakeholder meetings to write user stories, I simply reverse-engineered the requirement definitions from the existing Termine demo instead, which is much more efficient. This way, I already have the main advantage of user stories - having a vision of the finished product and understanding requirements from a user perspective - but with the thoroughness of statement requirements. The defined requirements were then discussed with the client to ensure that all essential features were covered, and then eventually revised into functional and non-functional requirements.

2.1.1 Functional Requirements

Functional requirements are “detailed statements of the project’s desired capabilities” (Stephens, 2015). They define what the system will do when interacting with a user that may or may not be visible to the user. Below are the functional requirements for this application.

Table 1: Functional requirements

ID	Requirement	Acceptance Criteria	Priority
F1	The demo page should be integrated into and accessible from the FlexiTerm homepage.	A link to the demo page should be inserted into the FlexiTerm homepage navigation menu.	High
F2	The application must allow the user to enter input in three formats: url, plain text and file upload.	On the demo page, there should be a form with fields to allow one of three inputs: a url field, a plain text field and a file upload button.	High
F3	The user’s input should first be validated on the client side to ensure it can be processed by the FlexiTerm back end.	<ol style="list-style-type: none"> 1. The user must select one input format and provide an input for the chosen format (no empty text fields or empty uploads). 2. The user must insert input in acceptable format such e.g. word count \leq 500 words for text box or only .txt files for upload. If the above are not adhered to, an error message will be displayed and submission attempts will be unsuccessful.	Medium
F4	The application must provide a means for the user to send input to the server.	The demo page should contain a submit button to submit user input.	High
F5	The user’s input must be processed successfully by the existing FlexiTerm Java application with no changes made to the code.	The back end should return a success message upon successful processing of the input.	Low
F6	The user must be able to view the results produced by FlexiTerm.	Upon successful submission and processing, the user should be redirected to a results page where the output generated by FlexiTerm can be viewed in html, plain text and table formats.	High
F7	The application must provide the user with options to download the output files generated by the FlexiTerm back end.	The results page should also contain a list of links to download the <i>output.txt</i> , <i>output.html</i> , <i>output.mixup</i> and <i>output.csv</i> files generated by FlexiTerm.	High

2.1.2 Non-functional Requirements

Non-functional requirements are no less important than functional requirements. They describe how the system will behave or operate. Below are the non-functional requirements for this application.

Table 2: Non functional requirements

ID	Category	Requirements	Acceptance Criteria	Priority
NF 1	Security	The application should be secure from external attacks.	<ol style="list-style-type: none"> 1. In all modern browsers, the entire FlexiTerm page must be HTTPS secure - no external links should be HTTP. 2. No personal information such as username or password of the User server account the app is hosted on should be revealed in the code. 	Low
NF 2	Reliability	The application must run as intended, any user actions must not break the performance of the apps.	The system must be fault-tolerant and catch and handle errors so that the application remains live.	High
NF 3	Maintainability	As mentioned earlier, FlexiTerm is an open source application that would likely be updated regularly for the foreseeable future. The web application code must therefore be maintainable in event of future updates.	As this is a straightforward application, the lines of code in each file (relevant to this application) must not exceed 1000 lines each.	Low
NF 5	Performance	Nielsen states that “webpages have to be designed with speed in mind” [Nielsen, 1997]. To ensure maximum user satisfaction, response times should be kept to a minimum.	Demo and results page should load within 10 seconds, as this is around the limit for keeping the user’s attention . Any longer, and the user may think an error has occurred.	Low
NF 6	Consistency	The application design must be consistent with the rest of the FlexiTerm homepage.	The existing CSS files used for the FlexiTerm homepage must also be used for the demo page.	Medium
NF 7	Usability	The application must be easy to use, learnable and overall, user-friendly.	1. The application must adhere strictly to Jakob Nielsen’s 10 heuristics for user interface	Medium

			design [Nielsen, 1995]. 2. The application must score an above average score in usability testing	
NF 8	Compatibility	The application must be compatible with most, if not all modern browsers.	Application should maintain functionality and look when run on 5 of the most commonly used browsers.	Medium

2.2 Design

Following requirements analysis, the next step is to transform these requirements into a high-level design. As mentioned earlier, this application is not concerned with how FlexiTerm works, but simply with passing input to FlexiTerm and making use of the output.

2.2.1 System Architecture

This system architecture of a web application defines the interaction between its components. The two main structural components of a typical web application are the client and server sides which run concurrently [Yaskevich, 2017]. When a user requests a web page such as the FlexiTerm demo page from a web browser, the client sends a response through the browser. If the request is successful, the page will be displayed on the user's browser.

The client side is the functionality that a user interacts with. It lives on the user's web browser and is developed in HTML for layout, JavaScript for functionality and CSS for styling. The server side lives on the server and responds to HTTP requests. It can store user data but cannot be seen by the user. It is developed with any of the following languages: PHP, Python, Java, Ruby on Rails, .NET or Node.js /JavaScript [Stringfellow, 2017].

The client-server model is used in this application: the client accepts user input and sends it to the server which then processes the input with FlexiTerm and sends the output back to the client.

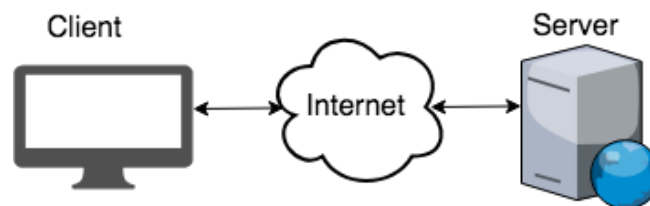


Figure 14: Client-server model

The SQLite database file *FlexiTerm.sqlite* is hosted directly on the server, rather than communicating with an externally hosted database, so all of the code is hosted on one server. This is called a single-tier architecture: front end and back end exist on the same server.

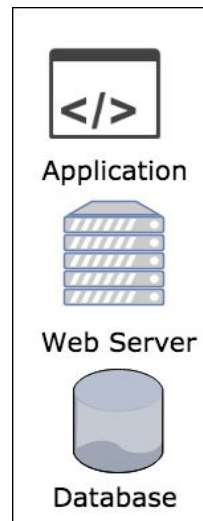


Figure 15: Single-tier Application

An alternative to single-tier architecture is multi-tier architecture where the database or back end and the application or front end live on different servers. A single-tier architecture is ideal for this application being a simple, low-traffic app and because a database file is used to hold data rather than an external database server. However, the disadvantage of this is that if the server is down, the entire FlexiTerm site, including the demo, will not be accessible.

There are two main types of client side architecture for web applications: Single page and Multi page models. A single page application “is an app that works inside a browser and does not require page reloading during use” [Neoteric, 2016]. It is a single web page which loads additional content using JavaScript. Examples of SPAs are Gmail, Facebook, Twitter etc. SPAs aim to improve user experience by eliminating page reloads and wait time [Neoteric, 2016]. JavaScript-based SPA frameworks include AngularJS, Ember.js, Meteor.js, Knockout.js, React.js, Vue.js. A multi page application is the traditional web application. Each change e.g. submitting data back to the server requests and renders a new page from the server to the browser. Nowadays, however, the use of Ajax or Asynchronous JavaScript eliminates the need to send large amounts of data back and forth between browser and client - only the relevant parts of the application are refreshed.

From past experience building both single and multi page applications with React.js, a multi page model is easier to set up and maintain for smaller projects. It allows the code base to be improved and maintained by other developers without having to learn an SPA framework, some of which have very steep learning curves [Naumovski, 2017]. In addition, the Users web server which is to host the finished application does not support SPA frameworks and the app would have to be hosted externally otherwise. As this goes against a fundamental requirement of the project, the MPA model is more suitable in this case.

2.2.2 Database Design (Relevant Tables)

Within this demo app, data is not being stored on any database. Rather, data *already* stored in the SQLite database by the FlexiTerm Java app is used. The FlexiTerm Java app utilises an SQLite database of 22 tables, however, only the following four tables are used in the demo to obtain relevant information and results:

Table 3: Relevant Database Tables

	Table	Purpose	Database Structure						
1	term_normalised	This is used to obtain the normalised version, <i>expanded</i> , of each term.	<table><tr><th colspan="2">term_normalised</th></tr><tr><td>PK</td><td><u>normalised</u></td></tr><tr><td></td><td>expanded len</td></tr></table> <p>Figure 16: Table term_normalised</p>	term_normalised		PK	<u>normalised</u>		expanded len
term_normalised									
PK	<u>normalised</u>								
	expanded len								
2	term_phrase	This is used to obtain the original term (before normalisation), <i>phrase</i> , exactly as extracted from the input text.	<table><tr><th colspan="2">term_phrase</th></tr><tr><td>PK</td><td><u>id</u></td></tr><tr><td></td><td>sentence_id token_start token_length phrase normalised</td></tr></table> <p>Figure 17: Table term_phrase</p>	term_phrase		PK	<u>id</u>		sentence_id token_start token_length phrase normalised
term_phrase									
PK	<u>id</u>								
	sentence_id token_start token_length phrase normalised								

3	term_termhood	This is used to obtain the term hood score, <i>c</i> , for each normalised term.	<table><tr><th colspan="2">term_termhood</th></tr><tr><td>PK</td><td>expanded</td></tr><tr><td></td><td>representative len f s nf c</td></tr></table>	term_termhood		PK	expanded		representative len f s nf c
term_termhood									
PK	expanded								
	representative len f s nf c								
Figure 18: Table term_termhood									
4	data_document	This is used to obtain the entire input text, <i>document</i> .	<table><tr><th colspan="2">data_document</th></tr><tr><td>PK</td><td><u>id</u></td></tr><tr><td></td><td>document verbatim</td></tr></table>	data_document		PK	<u>id</u>		document verbatim
data_document									
PK	<u>id</u>								
	document verbatim								
Figure 19: Table data_document									
5	output_table	This is used to obtain the <i>rank</i> and <i>variant</i> (if applicable) of each term.	<table><tr><th colspan="2">output_table</th></tr><tr><td>PK</td><td><u>id</u></td></tr><tr><td></td><td>rank representative variant f c</td></tr></table>	output_table		PK	<u>id</u>		rank representative variant f c
output_table									
PK	<u>id</u>								
	rank representative variant f c								
Figure 20: Table output_table									

2.2.3 Interface Design

Following requirements analysis, I designed the wireframes for the user interface based on the established requirements. Wireframes show the basic layout of the intended design, how the app will work and what content it will have. They also help to catch feasibility and usability errors early on. In addition, the design can be viewed by the client and revised according to any feedback. For this project, Balsamiq Mockups 3 was used as the wireframing tool.

The FlexiTerm web demo, which is following in the steps of Termine, consists of two pages: the demo home page where users enter input and the results page where users view and download input.

Demo Home Page

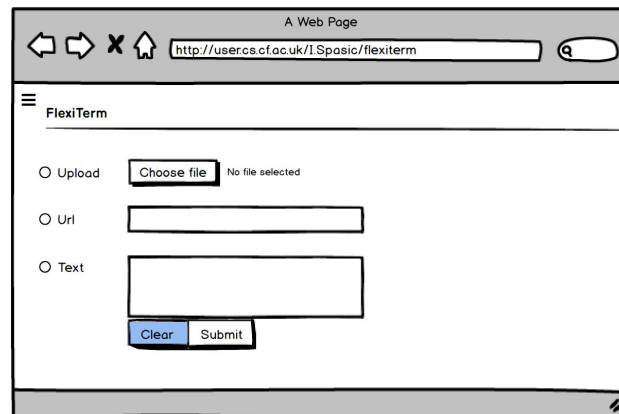


Figure 21: Demo home page wireframe

On the main page, the user can enter input in one of three methods: by uploading a plain text file, entering a url link to a text document or simply entering text in the text box. The 'Clear' button allows the user to remove any entered input including uploaded files. The design is minimalistic with only relevant information presented in simple language.

Results

The results page is a single-view page with the number of results at the top right corner and three tabs: View as HTML page, View as Table and Download. In the HTML view, which is the default, the extracted terms are highlighted in the context of the original input submitted by the user, similar to Termine and Term Extraction. Hovering the mouse over any highlighted term will display a tooltip containing details about it such as the the normalised version of the term or its termhood score.

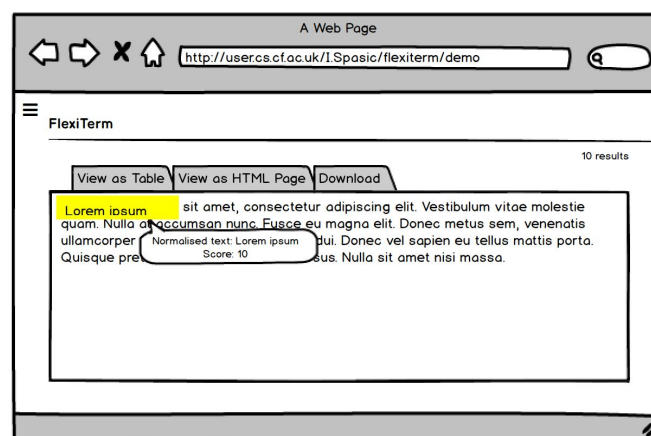


Figure 22: HTML table wireframe

In the table view, results are displayed in a data table complete with search, filter and pagination features. The wireframe uses a placeholder data table to represent this.

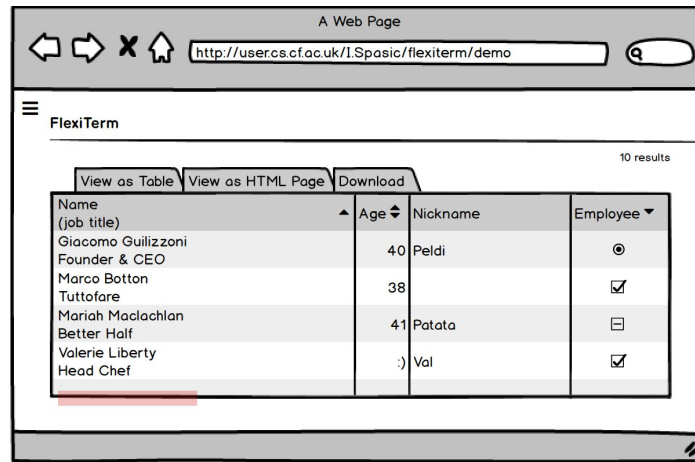


Figure 23: Table tab wireframe

The Downloads tab shows a list of links for downloading the output files produced by FlexiTerm: *Output.html*, *Output.mixup*, *Output.csv* and *Output.txt*. Clicking any of the links prompts a download dialog box.

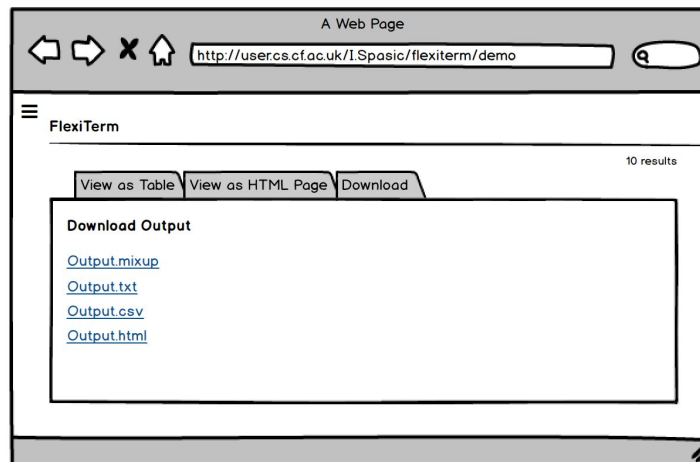


Figure 24: Download tab wireframe

As evident by the above wireframes, this demo is designed to be similar to Termine and Terminology Extraction in terms of functionality and user interface.

Use Cases

This section will describe possible interactions between a user and the system to achieve certain goals. For all use cases, there is one actor role simply described as “FlexiTerm user”. This refers to any member of the stakeholders or intended users.

Use case 1

User case name:	Enter input
User type:	FlexiTerm user
Description:	The user will enter input on the demo page and submit it
Pre-conditions:	The user has accessed the FlexiTerm site and navigated to the demo page
Basic flow:	<ol style="list-style-type: none"> 1. The user enters input in their desired format: plain text, file upload or url 2. The user presses the Submit button
Exception flow 1:	<u>Description</u> User input is invalid e.g. empty text box, too large file size, invalid url format <u>Termination outcome</u> An error message is displayed under the affected radio-selected input
Exception flow 2:	<u>Description</u> User does not select a radio button or enter any input at all <u>Termination outcome</u> An error message is displayed above the form prompting the user to select and enter an input
Post-conditions:	Input is successfully submitted for processing

Activity Diagram: Enter input

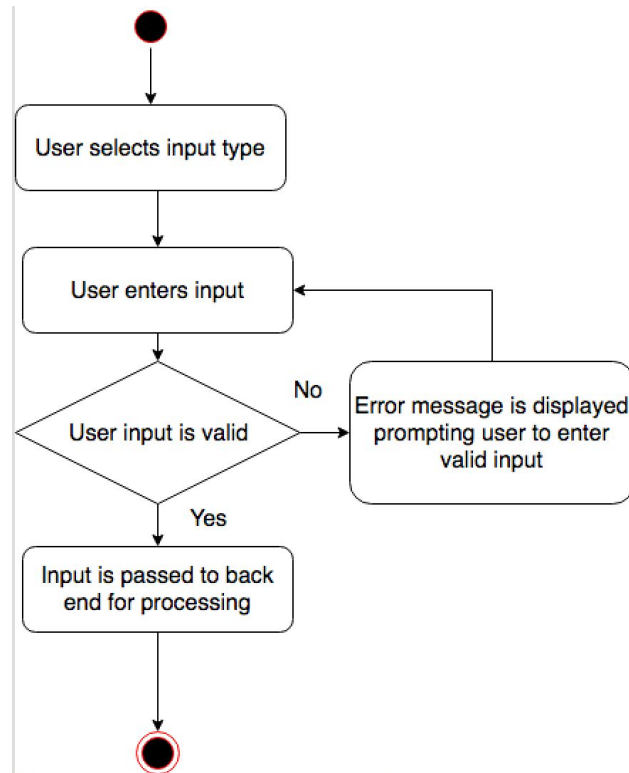


Figure 25: Activity diagram - enter input

Use Case 2

User case name:	View output
User type:	FlexiTerm user
Description:	The user will view the output produced by FlexiTerm
Pre-conditions:	User input has been successfully submitted and processed by the FlexiTerm back end
Basic flow:	The results page is displayed, default view is DataTable view
Alternative flow 1:	User clicks on the HTML page tab to view the HTML view
Alternative flow 2:	The input produces no output so the results page is blank

Activity Diagram: View Output

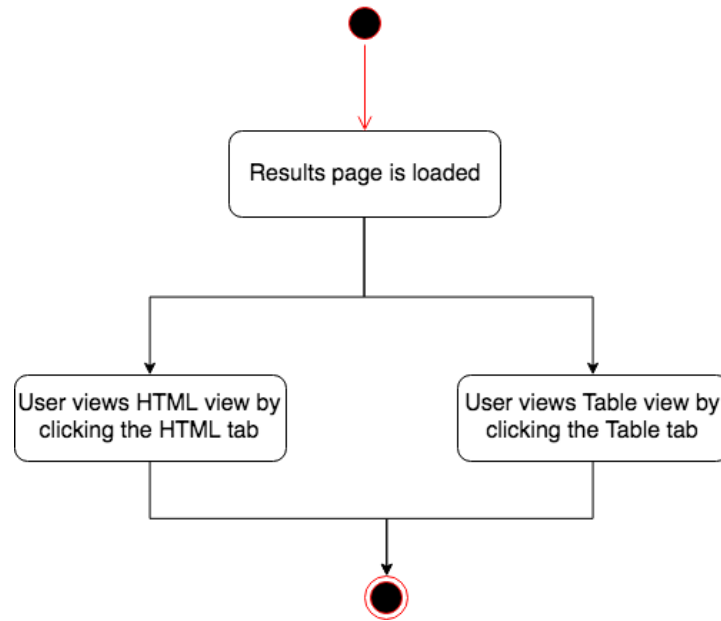


Figure 26: Activity Diagram - view output

Use Case 3

User case name:	Download output
User type:	FlexiTerm user
Description:	The user will download the output produced by FlexiTerm
Pre-conditions:	User input has been submitted and successfully processed by the FlexiTerm back end and the results page has loaded
Basic flow:	<ol style="list-style-type: none"> 1. The user clicks on the "Download" tab to view the Download links 2. The user clicks on the desired link (e.g output csv file) 3. A download prompt is displayed on the browser (or not, depending on the browser) 4. The file is downloaded onto the user's device.
Alternative flow 1:	The input produces no output so the results page is blank

Activity Diagram: Download output

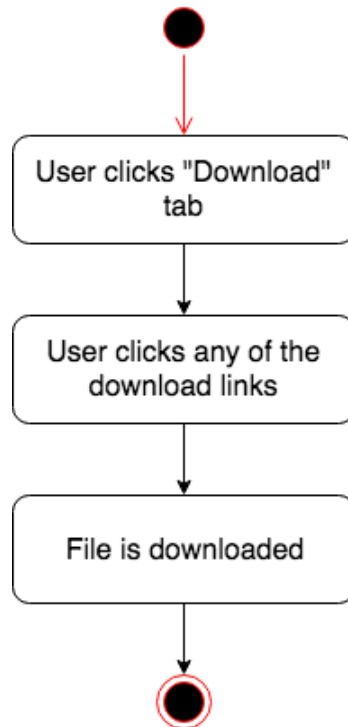


Figure 27: Activity Diagram - download output

3.0 Methodology

3.1 Development Tools and Languages

Having worked on numerous web development projects in the past, it was relatively easy for me to identify what tools and languages would be most appropriate for this project. These were divided into two categories: client side and server side.

3.1.1 Client Side

HTML/CSS

The front end of the application is designed using HTML5 to structure the web page and CSS to style it. Rather than designing the CSS from scratch, the web pages written for the demo use the existing CSS template file for the rest of the FlexiTerm homepage. This was a massive advantage as the CSS template is already designed to be fully responsive and contains predefined styling for base HTML elements such as paragraphs, buttons, navigation etc., thus allowing for consistency throughout the homepage. The template is also customisable so I was able override or create new styling for some HTML elements when the need arose.

JavaScript, AJAX and jQuery

JavaScript, jQuery and Ajax are used to add functionality to the HTML page e.g. causing actions to occur when a user clicks a button. jQuery is a lightweight JavaScript library that makes it faster and easier to manipulate HTML elements and CSS and make Ajax calls, compared to plain/vanilla JavaScript.

Example: Selecting an HTML element with ID “paragraph”

Plain JS: `document.getElementById('paragraph')`

jQuery: `$('#paragraph')`

jQuery also reduces the need to reinvent the wheel by writing custom scripts which may take time, as these already exist in libraries. It provides the capability for creating plugins, which are useful custom methods that are used to perform operations on an object, and is also excellent for cross-browser support provided a recent enough version is used. Although jQuery can easily be substituted for plain JavaScript in this project, it is required for the *DataTables* plugin, hence its inclusion. The jQuery version used in the FlexiTerm demo is v3.3., released Jan 20, 2018.

Ajax - or asynchronous JavaScript - is an asynchronous, speedy, client side script that allows communication between the client and the server. The main feature of Ajax is that only necessary data, such as user input, is passed to the server for processing, so there is no need to reload the entire page when output is returned. The user input will need to be passed to and processed by the FlexiTerm Java back end hence the need for Ajax.

DataTables

DataTables is a jQuery plugin that adds advanced functionality and responsive styling to HTML tables. Based on the wireframe in Figure 23, the results are to be displayed in a table, but rather than a simple table, it is more beneficial to allow the user to be able to sort, filter and display the tabular results as they wish - these are the features a data table provides. Figures 11 and 12 below show the difference between a data table and a plain HTML table.

Name	Position	Office	Age	Start date	Salary
Tiger Nixon	System Architect	Edinburgh	61	2011/04/25	\$320,800
Garrett Winters	Accountant	Tokyo	63	2011/07/25	\$170,750
Ashton Cox	Junior Technical Author	San Francisco	66	2009/01/12	\$86,000
Cedric Kelly	Senior Javascript Developer	Edinburgh	22	2012/03/29	\$433,060
Airi Satou	Accountant	Tokyo	33	2008/11/28	\$162,700
Brielle Williamson	Integration Specialist	New York	61	2012/12/02	\$372,000
Herrod Chandler	Sales Assistant	San Francisco	59	2012/08/06	\$137,500
Rhona Davidson	Integration Specialist	Tokyo	55	2010/10/14	\$327,900
Colleen Hurst	Javascript Developer	San Francisco	39	2009/09/15	\$205,500
Sonya Frost	Software Engineer	Edinburgh	23	2008/12/13	\$103,600
Jena Gaines	Office Manager	London	30	2008/12/19	\$90,560

Figure 28: Plain HTML Table

Show 10 entries				Search: <input type="text"/>	
Name	Position	Office	Age	Start date	Salary
Airi Satou	Accountant	Tokyo	33	2008/11/28	\$162,700
Angelica Ramos	Chief Executive Officer (CEO)	London	47	2009/10/09	\$1,200,000
Ashton Cox	Junior Technical Author	San Francisco	66	2009/01/12	\$86,000
Bradley Greer	Software Engineer	London	41	2012/10/13	\$132,000
Brenden Wagner	Software Engineer	San Francisco	28	2011/06/07	\$206,850
Brielle Williamson	Integration Specialist	New York	61	2012/12/02	\$372,000
Bruno Nash	Software Engineer	London	38	2011/05/03	\$163,500
Caesar Vance	Pre-Sales Support	New York	21	2011/12/12	\$106,450
Cara Stevens	Sales Assistant	New York	46	2011/12/06	\$145,600
Cedric Kelly	Senior Javascript Developer	Edinburgh	22	2012/03/29	\$433,060
Name	Position	Office	Age	Start date	Salary
Showing 1 to 10 of 57 entries					
			Previous	1	2 3 4 5 6 Next

Figure 29: DataTable data table

Font Awesome

Font awesome is a free toolkit that provides a variety of icons for web projects. Some Font Awesome icons were used to represent visuals in the application to aid user experience. I initially considered using ASCII symbols, but certain browsers such as some versions of Internet Explorer and Google

Chrome have issues rendering ASCII symbols, displaying a “□” character or some variation instead. Font Awesome is however guaranteed to have cross-browser support and is currently used on over 100 million websites across the world.

3.1.2 Server Side

PHP

For this application, the main function of the server side is to simply run the FlexiTerm Java app in the background, extract information from the SQLite database and then display it on the browser. Rather than having to learn new technologies, I only considered the back end technologies I had previous experience with: Node.JS (server side Javascript) and PHP. The main advantage of Node.JS compared to PHP for this application is to maintain only JavaScript as a single scripting language across the application, thus making it more maintainable and easier to work with. PHP is another excellent, rich language which is extremely portable.

However, Node.JS is currently not supported by the Users web server, and due to account quotas on the server is not an effective solution as it requires large NPM modules to be installed during environment set up in addition to the FlexiTerm source files which are already over 200mb large. I selected PHP as the server side language because it is already installed on the server and meets the requirements for this project.

SQLite

The FlexiTerm Java app stores its input and output data in a SQLite database file. SQLite is a piece of software that provides a relational database stored in a single file e.g. *FlexiTerm.sqlite*. The SQLite database behaves like any other relational database management software (RDBMS): it stores and manages data, and processes queries. The application back end communicates with this database to display output to the browser. When the Java application is run, the input files and the database is wiped and replaced with the new data. However, this works best when it is being written to by one user at a time so I needed to tackle the problem of multiple concurrent users.

SQLite Manager

This is a Database Management System (DBMS) tool for viewing, editing and managing SQLite files. Being able to view what tables were contained in the database file was useful in identifying useful data to extract and writing and testing queries to extract them.

3.2. Project Management

As with any other software project, effective project management is very important, even with a project as seemingly straightforward as this. Trello was used as the main project management tool

for this project; it is a project management tool that organises projects into Kanban boards, which can be labelled however the user wishes. My weekly Trello boards were labelled as: “To Do”, “Doing” and “Done” and tasks were created and rearranged as progress was made.

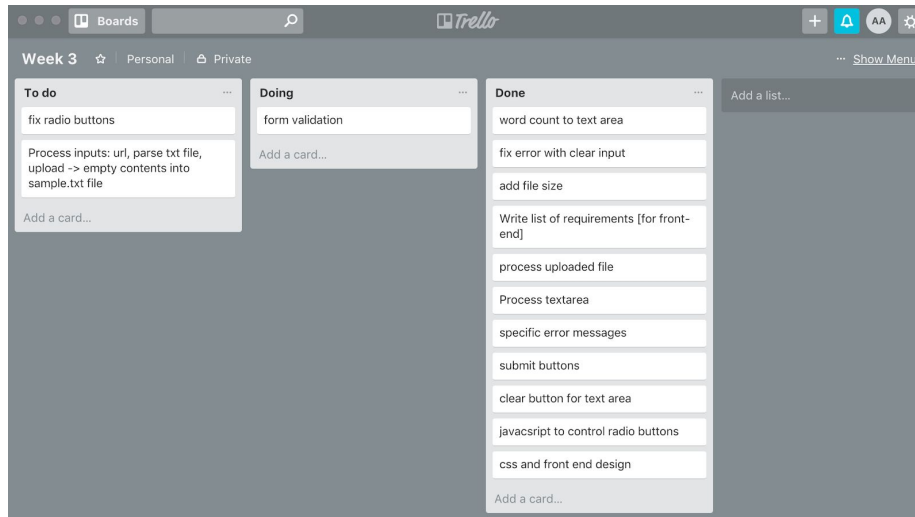


Figure 30: Trello Weekly Board

Following the Gantt Chart in the initial plan closely, work was divided on a weekly basis such that each week resulted in a working version of the application with newly added features or changes according to the weekly plan, following the Waterfall development method. I met with my supervisor, Irena Spasić, who is also the main client of the project as well the developer of the FlexiTerm tool to request feedback and provide updates on my progress following Agile standards. Trello proved to be an incredibly useful tool with regards to maintaining a Waterfall-Agile methodology and ensuring that the software project and the documentation of the final report was completed in due time.

3.3. Version Control

As with any software development project, developing without having a code backup system in place is risky. To mitigate this risk, I made use of a version control tool called Gitlab. GitLab is a Git-repository manager, which is basically a version control system for maintaining code such that changes are effectively managed over time. Any time a change is pushed, a new revision is made, while still maintaining old versions of the code. This is a much better solution than creating several multiple versions of the code anytime a change is made.

My private Cardiff University Gitlab account at gitlab.cs.cf.ac.uk was used to maintain the codebase throughout this project. Following Agile standards, weekly milestones were set and corresponding

branches were created for each one. When each milestone was finished, the code was pushed from my computer to the Gitlab server and the milestone was marked complete on GitLab.

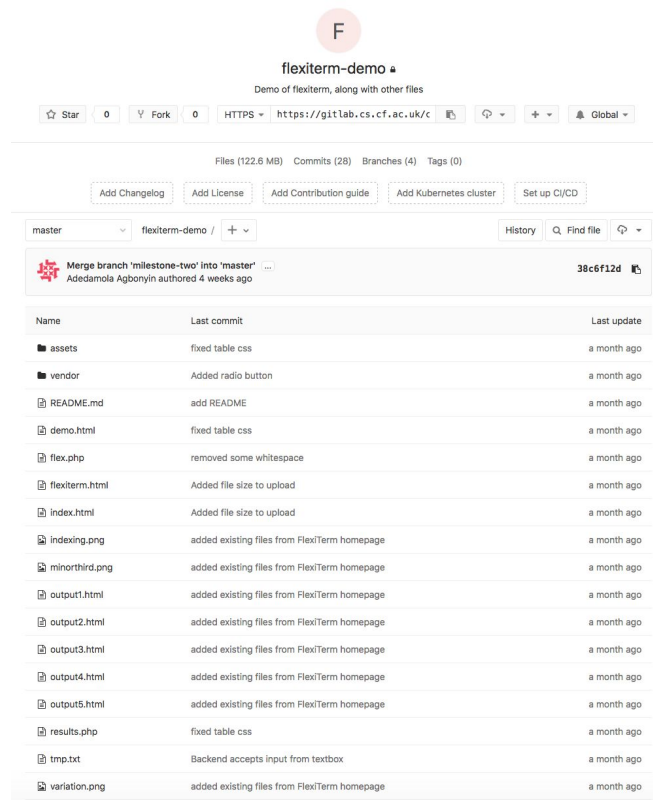


Figure 31: GitLab Version Control

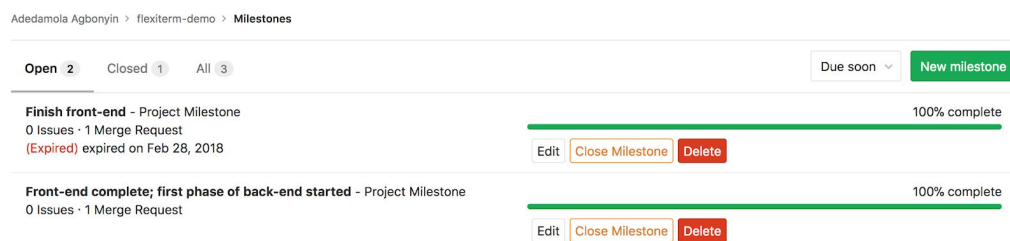


Figure 32: GitLab milestones

Effective version control with tools like GitLab proved to be incredibly beneficial due to changing requirements throughout the project; a few times, code snippets from older revisions were reused and some code files were restored when a particular change could not be undone. In addition to GitLab, I created a GitHub account to save fully-functional versions of the program weekly, in case the GitLab server was down for maintenance. GitHub is a publicly available version of Gitlab.

4.0 Risk Assessment

There is always an element of risk and uncertainty with any software development project. Risk assessment is a process whereby possible risks that could affect the entire project are identified and mitigated, ensuring that all goals defined initially are met as effectively as possible.

Table 4: Risk Assessment

Category	Risk	Severity	Likelihood	Action to mitigate risk
Technical knowledge	Project may require technical expertise beyond my level.	Low	Low	Liaise with supervisor to ensure that requirements for this project are within my technical expertise or can be learned within a short time. If not, compromise can be made to change requirements.
Timing and project management	<ol style="list-style-type: none"> 1. Initial plan may have underestimated amount of work to be done. 2. Changes to requirements can delay the project. 	Medium	Medium	The use of Trello and regular development milestones on GitLab further breakdown tasks to be completed so that timing issues can be caught early on.
Technical requirements	Inadequate or incomplete requirements analysis resulting in certain features being missing.	Medium	Medium	The Waterfall-Agile methodology ensures that the user is prepared to adapt to any sudden changes to the system.
Usability	<ol style="list-style-type: none"> 1. Code may break during use after final deployment. 2. Issues from browser incompatibility. 3. Poor feedback during usability testing, resulting in late-stage changes. 	Medium	Medium	<ol style="list-style-type: none"> 1. Ensure errors are properly handled and do not break the system 2. Thorough testing to ensure cross-browser support. 3. Design UI according to Jakob Nielsen's usability heuristics [Nielsen, 1995]
Server issues	Server may crash or be offline when users are attempting to access the	High	Medium	Nothing to do in this case except wait for the issue to be fixed.

	app.			
Data loss	Code base may accidentally deleted from local computer.	Medium	Low	Code base will be maintained on the cloud on GitLab and GitHub in event of local data loss.
Human factors	Any personal extenuating circumstances such as illness which may delay the project.	Low	Low	The initial plan allows enough spare time in case any unforeseen issues arise.
Maintainability	Future changes to the underlying FlexiTerm app may break the demo code.	Low	Low	<ol style="list-style-type: none"> 1. The demo app is designed to be independent of the inner workings of FlexiTerm and will function correctly as long as the SQLite database structure is maintained. 2. Code should be well-commented to allow for readability.
Portability	Difficulties may arise when the app is eventually deployed to the Users server.	Medium	Low	UAT testing will be carried out on the Project and Users web servers prior to final deployment.

5.0 Implementation

This section will detail the execution of the initial plan using the methodology described in [section 3.0](#), as well as issues encountered during development and how they were resolved.

5.1 Summary of Features

The initial plan of this project contained a list of main and additional features expected to be implemented in the FlexiTerm web application. The table below will detail which of those features have been implemented, which were exempted and additional details.

Table 5: Summary of Implemented features

Main Features			
Description	Implemented	Exempted	Details
The user must be able to input their text file by pasting into a textbox, drag and drop or upload, or by URL.	✓		The demo home page contains an input form through which users can enter their input by uploading a file, entering a url to a document or simply pasting or entering text in a text box.
The user's input must be validated first, and then processed by the FlexiTerm back end.	✓		<p>All three forms of user input are validated before processing:</p> <ol style="list-style-type: none"> 1. One input format must be selected - value cannot be empty in all three cases. 2. Uploaded files must be plain text and no larger than 4kb in size. 3. Url must be valid and link to a plain text document. 4. Text box cannot allow more than 600 words. <p>If input is invalid, the user is prompted to enter a valid input.</p>
The output must be returned in a web page such that the recognised terms are highlighted, with the option to also view in plain text, table format, ranked according to termhood scores.	✓		<p>Output is returned in only two formats: HTML view in which terms are highlighted in context, and table view ranked according to termhood scores. The plain text view was deemed redundant in the design stage and exempted.</p>

The user must be able to download the output in appropriate formats, e.g. csv for the table, plain text etc.	✓		Download links for the <i>.mixup</i> , <i>.csv</i> , <i>.txt</i> and <i>.html</i> output files are made available to the user.
Additional Features			
The application should use cookies such that when the page is refreshed, the data/output is not lost.	✓		In addition to default browser caching, results from the database are saved in user sessions on the server to allow multiple users to access their respective results.
The application should be able to handle several users at the same time.	✓		The FlexiTerm back end is designed to process input once at a time due to limitation on the SQLite database, however, measures were put in place to allow multiple users to view and download results nonetheless.
The application should include basic search engine optimisation so that users can easily find the FlexiTerm site when searching for text recognition tools.		✓	This feature was not discussed with the client before-hand and was therefore exempted during development.
The application should be compatible with different browser types and versions.	✓		The application was tested for browser compatibility to ensure a satisfactory result.
The application should be lightweight enough to load with little delay.	✓		One reason Input limits were put in place is that processing time by the back end is greatly reduced.
The web page should contain additional details such as source code, terms of use and a simple quick start guide for new users.		✓	The FlexiTerm homepage already contains details about the application itself, therefore this feature was deemed redundant and exempted.
The code should strictly adhere to good coding standards that can be easily reused or expanded by others.	✓		A Linter plugin was installed in Sublime Text 3 - the text editor used throughout this project - in order to auto-format code into correct syntax.
The user should be able to adjust some FlexiTerm settings.		✓	Due to limitations of the SQLite database to only allow one write at a time, allowing users to change apart from input

			would have been quite difficult to implement and could potentially break the code. As this feature is not necessarily fundamental to the functioning of FlexiTerm, it was exempted.
--	--	--	---

The implemented features are described in further detail in section 5.2.

5.2 Features

5.2.1 Homepage and User Input

On the FlexiTerm homepage, a link to the demo page was inserted in the navigation menu (as seen below) as the final item.

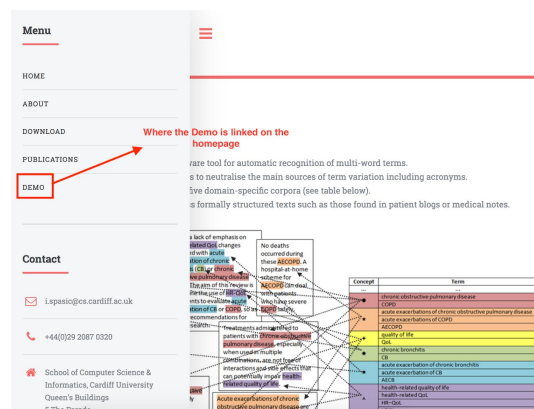


Figure 33: Location of demo in homepage menu

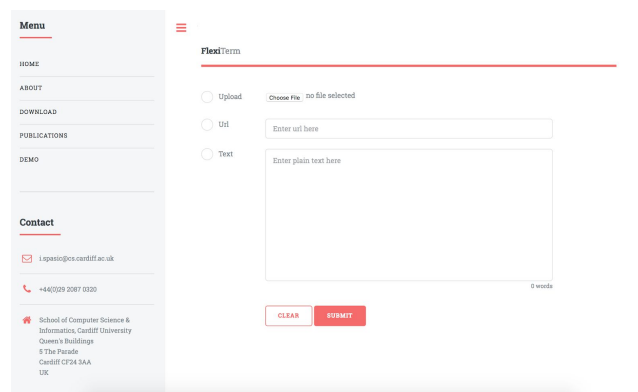


Figure 34: Demo Homepage

The home page was designed following the wireframe closely [Figure 20], with some minor changes. The page consists of a simple form with three radio buttons for selecting an input format, and three corresponding form elements into which input is entered. The user can select their desired input format by either clicking a radio button or simply entering input directly. A jQuery event listener function automatically checks the corresponding radio button for whichever input the user selects, so if the user enters input in the url textbox for example, the “Url” radio button is automatically selected. If after entering input in one format the user enters input in another format e.g. file upload after entering a url, the “Upload” radio button is then checked automatically instead. This way, upon submission, only one input - for which a radio button is selected - is uploaded to the server.

The text box for entering plain text input has a word count on the bottom right corner that is updated whenever any changes are made in the textbox e.g. when text is pasted, typed or erased. The

validation function for the text box does not allow more than 600 characters so the visible word count makes it easier for the user to erase some of the text to fit the limit.

The Clear button clears all input in case the user makes an error. It is styled using the default button style class of the CSS template, which makes it less prominent than the coloured Submit button. This is to prevent users from accidentally clearing their input instead of submitting it.

5.2.2 Input Validation

Validating user input is necessary to prevent the FlexiTerm code from breaking while processing invalid data or having to process too large data. When developing a method for validating user input, two ideas were considered: writing custom validation functions or using a form validation framework called FormValidation.io.

The advantage of using a framework is that it typically utilises responsive design and already contains feedback messages and icons. Initially the framework approach was used in an attempt to save time, but styling features embedded within it clashed with the FlexiTerm CSS template, resulting in a scattered design. Attempts made to reconcile the two templates proved futile, and seeing as only three input elements were to be validated, the framework was scrapped in place of custom functions. Eventually, writing custom functions did not take as long as expected, as some of the code was reused for the three input formats. A different form validation function was written for each of them:

1. Upload: The validation function for this input format only allows files less or equal to 4 kb in size and plain text or .txt files for easy processing. Uploads are sent to the server if and only if they meet these conditions.



Figure 35: File upload - wrong format error



Figure 36: File upload - too large file error

2. Textbox: Textbox values are validated against word length; it is invalid if the value is greater than 600 words or empty. Similar to file upload, input greater than 635 words were truncated by the server during processing so a limit was put in place on the client side as a work-around.

Text

Purification of TCF-1 alpha, a T-cell-specific transcription factor that activates the T-cell receptor C alpha gene enhancer in a context-dependent manner. The differentiation of T cells into functionally diverse subpopulations is controlled in part, by transcriptional activation and silencing; however, little is known in detail about the proteins that influence this developmental process. We have purified a new T-cell-specific factor, TCF-1 alpha, that is implicated in the activation of genes encoding a major component of the human T-cell receptor (TCR). TCF-1 alpha, originally identified and purified through its binding sites on the HIV-1 promoter, was found to bind to the TCR alpha enhancer and to promoters for several genes expressed at significantly earlier stages of T-cell development than the TCR alpha gene (e.g., p56lck and CD3 delta). Sequences related to the TCF-1 alpha binding motif (5'-GGCACCCTTTGA-3') are also found in the human TCR delta (and possibly TCR beta) enhancers. Southwestern and gel renaturation experiments with the use of purified protein fractions revealed that TCF-1 alpha activity is derived from a family of 57- to 53-kD proteins that are abundantly expressed in mature and immature T-cell lines (Jurkat, CCRF-CEM) and not in mature B cells (JY, Namalwa) or nonlymphoid (HeLa) cell lines. A small 95-bp fragment of the TCR alpha control region that

Invalid input - max of 600 words allowed

780 words

Figure 37: Text input - input greater than limit error

3. Url: Url values are checked for a match against the following regular expression:

```
/^(http:\\\\|www\\.|https:\\\\|www\\.|http:\\\\|https:\\\\|) ?[a-z0-9]+([\\-\\.]{1}[a-z0-9]+)*\\. [a-z]{2,5}(:[0-9]{1,5})?(\\/\\.*)?\\.txt$/. If the value is empty, not a valid url or the url links to a non-plain text file, an error message is displayed.
```

Url

http://users.cs.cf.ac.uk/AgbonyinAT/example.csv

Invalid input - please enter a valid .txt url

Figure 38: Url input - invalid format

If no input is entered or selected at all, an error message is displayed above the form instructing the user to enter an input.

FlexiTerm

Please enter an input

Figure 39: Input form - No input

5.2.3 Input processing

When input is successfully submitted, three outcomes are possible:

- An error occurs on the server
- The server is busy
- The POST request is successful

An error occurs on the server if for any reason invalid input slipped through client side validation. If there is no text, url or file input, the server returns an error message. The server is busy if another user's input is currently being processed and returns a Busy message to the client. The POST request

is successful if the server is not busy and the input is valid. If this is the case, the input is saved as a *sample.txt* file in the Text folder within the FlexiTerm source folder.

Sessions and Handling Multiple Users

As mentioned, FlexiTerm uses a SQLite database file which can only allow one write request at a time. This means that if two users submit their input a few milliseconds apart for example, the results for the first user are overwritten by the second user's results. Therefore, there was need to handle this scenario, however unlikely.

To ensure that only one input set is processed at a time, a semaphore file is used to lock the server while input is being processed. Upon input submission, the server checks to see if a *semaphore.txt* file exists in the source files folder. If not, the file is created in that folder and the *processInput()* function is called to run FlexiTerm on the *sample.txt* input and a *Success* response is sent to the client. If the file exists, however, the server responds with a *Busy* message instead. If an error occurs due to invalid input, the semaphore file is deleted to allow another user and the server sends an error response to that client.

```
if (file_exists('./assets/semaphore.txt') == 0) {
    $semaphore = fopen('./assets/semaphore.txt', 'w') or die('Unable to open file!');
    fclose($semaphore);
    // Copy textbox input to sample.txt
    if (isset($_POST['textbox'])) {
        $data = $_POST['textbox'];
        $_SESSION['data'] = $data;
        $file = fopen('./assets/src/text/sample.txt', 'w') or die('Unable to open file!');
        fwrite($file, $data);
        fclose($file);
        processInput();
    }
    else if (isset($_POST['url'])) {
        // Copy text in url text file to sample.txt
        $path = $_POST['url'];
        $file1 = file_get_contents($path);
        $path2 = './assets/src/text/sample.txt';
        $file2 = file_get_contents($path2);
        file_put_contents($path2, $file1);
        processInput();
    }
    elseif (isset($_FILES['file'])) {
        // Copy uploaded file as sample.txt
        $target = './assets/src/text/sample.txt';
        move_uploaded_file($_FILES['file']['tmp_name'], $target);
        processInput();
    }
    else {
        echo 'Error occurred';
        if (file_exists('./assets/semaphore.txt') == 1) unlink('./assets/semaphore.txt');
    }
}
else {
    echo 'Busy';
}
```

Figure 40: Flex.php - semaphore file

On the client, side, the ajax function does not only send data to the server, but receives a response as well. If a *Busy* response is received, the user is alerted on the homepage.

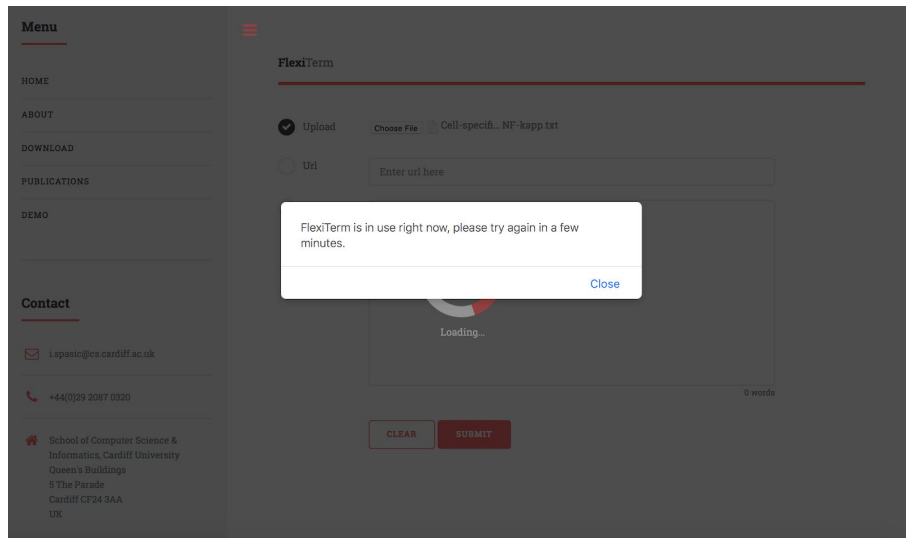


Figure 41: Server is busy

If a success response is received by the client, the demo page is automatically redirected to the results page. The *processInput()* function executes the FlexiTerm start script *FlexiTerm.sh*. If an error occurs for any reason, the semaphore file is deleted and an error response is sent to the client. Otherwise, to ensure that results are still maintained even after being overwritten in the database, results are saved to the user session. Upon input submission, a new session is created - if not already- and assigned a session id. In PHP, a session is a way to store information on the server for a particular user, and the information can be retrieved and used across multiple pages.

After the FlexiTerm start script has been run successfully, the *saveResultsToSession()* and *copyOutputFiles()* functions are called.

```
// Run flexiTerm Java app on sample.txt
function processInput() {
    chdir('/usr/local/www/user/AgbonyinAT/assets/src/');
    exec("sh FlexiTerm.sh", $output, $return);
    if(!$output){
        saveResultsToSession();
        copyOutputFiles();
        echo 'Success';
    }
    else {
        if (file_exists('./assets/semaphore.txt') == 1) unlink('./assets/semaphore.txt');
        echo 'Error'; //Error processing input
    }
}
```

Figure 42: Flex.php - processInput() function

The *saveResultsToSession()* function runs a set of queries on the database to extract relevant information for the results page, and save them in session variables.

```
function saveResultsToSession() {
    $conn = new PDO('sqlite:/usr/local/www/user/AgbonyinAT/assets/src/flexitem.sqlite') or die("cannot open the database");
    $tableResult = $conn->query("SELECT DISTINCT LOWER(phrase) as phrase, t1.expanded, ROUND(c,3) as c FROM term_termhood t1
        LEFT JOIN term_normalised t2 on t1.expanded = t2.expanded
        LEFT JOIN term_phrase t3 on t3.normalised = t2.normalised
        GROUP BY t1.expanded
        ORDER BY c DESC");
    //Get number of results
    $countResult = $conn->query("SELECT count(*) AS count FROM term_termhood ;");
    // Get original text
    $textResult = $conn->query("SELECT document FROM data_document ;");
    // Get term variants
    $termVariantResult = $conn->query("SELECT rank, representative, variant, c FROM output_table ORDER BY rank");
    $table = array();
    $variants = array();
    $count = $countResult->fetch();
    $count = $count["count"];
    $text = $textResult->fetch();
    $text = $text["document"];
    while ($row = $tableResult->fetch(PDO::FETCH_ASSOC)) {
        $table[] = $row;
    }
    while ($row = $termVariantResult->fetch(PDO::FETCH_ASSOC)) {
        $variants[$row["representative"]]["var"][] = $row["variant"];
    }
    // Save data as Session variables
    $_SESSION["count"] = $count;
    $_SESSION["text"] = $text;
    $_SESSION["table"] = $table;
    $_SESSION["variant"] = $variants;
}
```

Figure 43: Flex.php - *saveResultsToSession()* function

The *copyOutputFiles()* function preserves the output files produced by FlexiTerm by copying them to a new folder and naming them after the session id for that user. For example, if the session id is 12345, the output.html file is saved as *12345_output.html*. Multiple output files for different users can therefore exist on the server rather than being overwritten.

```

function copyOutputFiles() {
    // Copy all output files to new folder (to prevent overwriting by subsequent users)
    if (file_exists('./assets/src/output.html') == 1) {
        copy('./assets/src/output.html', './assets/output/' . $_SESSION["id"] . '_output.html');
    }
    if (file_exists('./assets/src/output.mixup') == 1) {
        copy('./assets/src/output.mixup', './assets/output/' . $_SESSION["id"] . '_output.mixup');
    }
    if (file_exists('./assets/src/output.csv') == 1) {
        copy('./assets/src/output.csv', './assets/output/' . $_SESSION["id"] . '_output.csv');
    }
    if (file_exists('./assets/src/output.txt') == 1) {
        copy('./assets/src/output.txt', './assets/output/' . $_SESSION["id"] . '_output.txt');
    }
}

```

Figure 44: Flex.php - copyOutputFiles() function

Issues with CORS

Cross-Origin Resource Sharing CORS is a mechanism that allows a client to access server resources from a different domain. “A user agent makes a cross-origin HTTP request when it requests a resource from a different domain, protocol, or port than the one from which the current document originated” [MDN Web Docs, n.d.].

After validation on the client side, all input is passed to the *Flex.php* endpoint for processing. For security purposes, most browsers such as Google Chrome and Mozilla Firefox among others, prevent cross-origin HTTP requests initiated from scripts such as the FlexiTerm client side script *Flex.js* and an error similar to that in Figure 42 is displayed in the browser console.

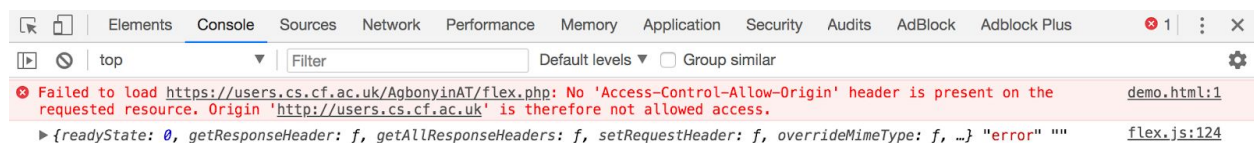


Figure 45: CORS error

One way to circumvent this in PHP is to use CORS preflight requests in form of headers ahead of the actual request to inform the server that the client request is safe.

```

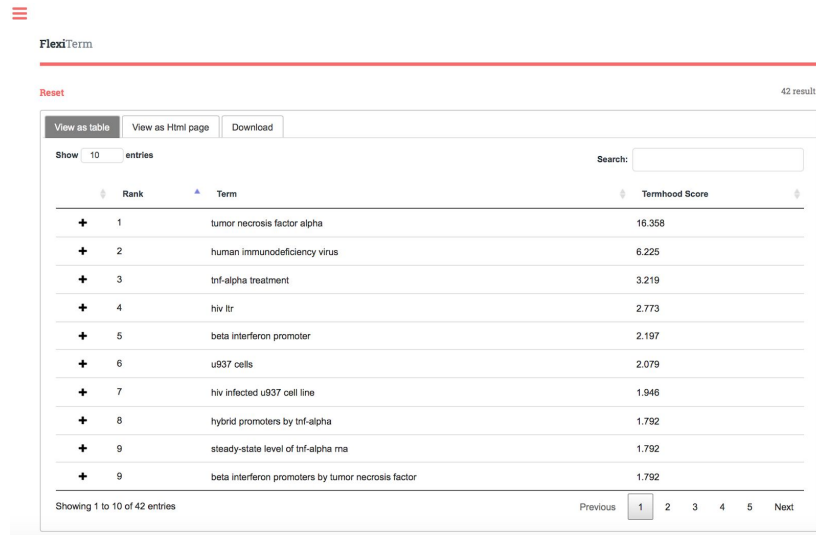
header('Access-Control-Allow-Origin: *');
header('Access-Control-Allow-Credentials: true');
header('Access-Control-Allow-Methods: GET,HEAD,OPTIONS,POST,PUT');
header('Access-Control-Allow-Headers: Access-Control-Allow-Headers, Origin,Accept, X-Requested-With,Content-Type, Access-Control-Request-Method, Access-Control-Request-Headers');

```

Figure 46: PHP CORS headers

5.2.4 Results Page

The results page *results.php* is automatically loaded in the same browser tab as soon as user input is successfully submitted, processed and saved to a session in *Flex.php*.



FlexTerm

Reset 42 results

View as table View as HTML page Download

Show 10 entries Search:

Rank	Term	Termhood Score
1	tumor necrosis factor alpha	16.358
2	human immunodeficiency virus	6.225
3	tnf-alpha treatment	3.219
4	hiv ltr	2.773
5	beta interferon promoter	2.197
6	u937 cells	2.079
7	hiv infected u937 cell line	1.946
8	hybrid promoters by tnf-alpha	1.792
9	steady-state level of tnf-alpha rna	1.792
9	beta interferon promoters by tumor necrosis factor	1.792

Showing 1 to 10 of 42 entries Previous 1 2 3 4 5 Next

Figure 47: Results.php - Default view

The Results page contains the number of results on the top right corner and a reset button on the top left corner. The Reset button redirects the page to the demo page for the user to enter new input. Below these is a container with three tabs: View as table (default view), View as HTML page and Download.

When the page loads, the session is continued and the *semaphore.txt* file is first deleted to allow any other users to make use of the application as the results have already been read from the database file and can now be freely overwritten. The session variables, which contain the list of results, table data, original input text and list of variants, are used to dynamically create the HTML and table views. As the results are being read from the session variables, a regex string is also dynamically created from the list of phrases identified by Flexterm as possible terms from the original input text. This regex string is used to highlight terms in the HTML view.

Table Tab

The table tab contains a data table of the extracted terms, their ranks and termhood (c-value) scores. The data table is searchable and sortable (default is by rank in ascending order), and the number of

results per page can be increased or decreased. This gives the user more freedom than a plain HTML table.

About halfway through development, a new version of FlexiTerm was released and the demo had to be updated to reflect this. This was a relatively easy transition to make as the demo was written to be independent of the FlexiTerm Java app, and the format of the output files and database tables which were used in the demo were maintained. The main difference between the old and new versions is that the new version produces term variants as part of the output. The old table tab simply contained a data table with rank, term and score columns but I had to make changes to the table view to display the term variants after replacing the FlexiTerm Java app. Not all terms have term variants; in fact, usually a very small number of terms in the input texts used in testing produced term variants as part of the output. So rather than including a column that would be blank majority of the time, I figured it was more appropriate to hide the additional data, but provide the user with an option to view it if they want.

To achieve this, I made use of the child row feature of the DataTables plugin. Each table row contains additional data hidden in child rows. The user can click the “+” symbol on each row to display the child row which contains both the normalised version of the term and the list of term variants.

–	4	patients with cf	2.079
<u>Normalised Term</u>		<u>Term Variants</u>	
cf patient		patients with cf cf patients	

Figure 48: Expanded table row

Html Tab

The HTML view shows the original input as entered by the user, with the extracted terms highlighted within the text. Upon hovering a mouse over any of the highlighted terms, a tooltip appears below showing the termhood score with the rank in brackets, the normalised version of the term and the list of variants, similar to the tabular view.

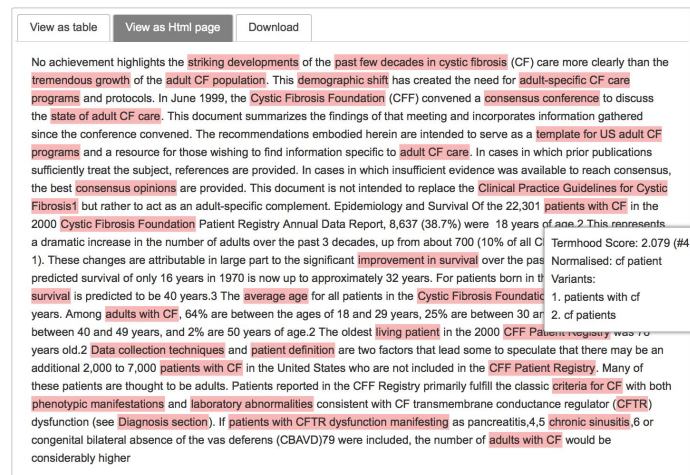


Figure 49: Extracted terms in context

The term-highlighting feature is performed by adding a span of class “highlight” around every word that matches the regex string - which consists of every phrase from which a term is extracted. Then, a JavaScript function is used to dynamically create a string of the variants of that phrase (e.g. the term CFF can have a variant of Cystic Fibrosis Foundation) combined with the normalised version of the term. A tooltip element is then added to each of the words with the “highlight” class containing the string, which has been punctuated with styling elements to format it in a visually appealing way.

Download Tab

The download tab contains a list of four links for each output file. Clicking any of the links will prompt a download dialog box (or not, depending on the browser) with which to download the respective files.

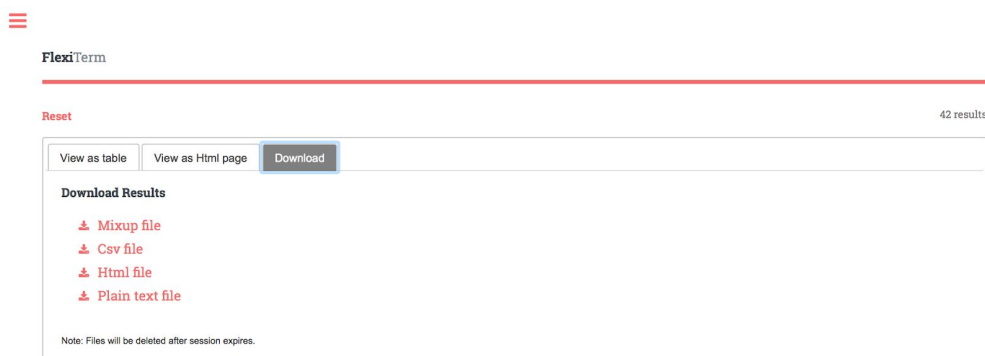


Figure 50: Downloads tab

As mentioned in [Section 5.2.3](#), the output files are named based on the session ID of each user, so rather than being overwritten each time a different user uses the app, the output files remain intact and accessible for the duration of the session.

5.2.5 Coding Standards

After all coding and initial debugging was finished, the client side JavaScript code was refactored to follow JavaScript coding standards using the Airbnb JavaScript style guide [github.com/airbnb/javascript]. A linter was installed in the text editor used to write the code (Sublime Text 3) for automatic formatting. A linter is a tool that analyses code to identify syntax errors, bugs and programming errors. The linter used in this project is Standard JS [standardjs.com]. The JavaScript and CSS files were correctly formatted using Dirty Markup [dirtymarkup.com].

PHP code was completed last but could not be formatted using a linter as there were some difficulties setting it up. To save time, the formatting was done by hand, with unnecessary spaces, redundant/repetitive code removed and variables appropriately named. After formatting and bug fixing, the demo app was deployed onto the Users web server for UAT testing.

Issues with Deployment

As expected during Risk Assessment [[Section 4.0](#)], there was an issue with account quotas on the Users Web Server. File quotas were set to about 60mb, but the total FlexiTerm source files and the demo code was about 281mb, several times over the limit. I contacted the IT service desk about this issue and was eventually able to get an increase on my account quota to 360mb. This issue delayed deployment and testing by about three days. However, this delay was quite minor as the development process took several days shorter than expected, so there was enough time to spare.

5.2.6 Browser Compatibility

The HTML5+CSS template FlexiTerm uses is designed to be fully responsive, giving the advantage of cross-browser compatibility. The 5 most popularly used desktop browsers are Google Chrome, Apple Safari, Mozilla Firefox, Microsoft Internet Explorer and Edge [StatCounter, 2018].

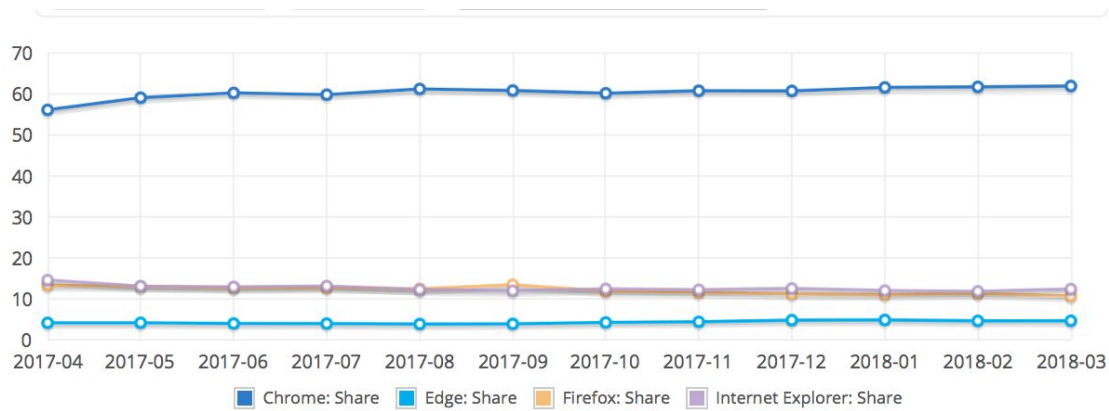


Figure 51: Browser market share [Netmarketshare, 2018]

The demo home and results pages were therefore tested for compatibility with these five browsers only. As seen in [Appendix A](#), all modern browsers except IE 7 are fully compatible with the demo home and results pages. In IE7, the functionality is unaffected but the menu icon on the top left corner on the page is absent. A compromise has to be made in this regard by ignoring browsers older than IE version 10. This is justified by the statistic that only 2.63% of the browser market share is IE11 [Statista, 2018], so we can safely assume that the number of users using older versions - especially versions as old as version 7 - is insignificant.

The FlexiTerm demo app is fully compatible with modern browsers; this highlights the benefit of using responsive design when creating web pages.

6.0 Testing and Evaluation

With regards to the Agile-Waterfall methodology applied to this project, testing was performed after each new feature was implemented to ensure no existing functionality is broken by new changes. This also reduces the burden of final integration testing after all features have been implemented. During integration testing, three types of testing were performed to evaluate this application: functional testing, non-functional testing and usability testing.

Functional testing is used to verify the functionality of the application against the functional requirements. It can be automated or done manually. For this application, functional testing was done manually with test cases. An alternative to this is unit testing in which sections of code are tested for correctness. However, unit tests will need to cover as much of the code as possible in order to be effective, so to save time writing and running possibly hundreds of unit tests, test cases were used instead. Non-functional testing on the other hand, is used to test non-functional requirements. In both testing methods, the acceptance criteria as documented in [Section 2.1](#) are used to decide whether or not a test fails or passes.

Usability testing is a method of evaluating how user-friendly and effective a user interface is. For this application, usability testing is conducted using user testing where a group of users were given sample input and asked to perform tasks with the application. After testing the application, they were then provided a questionnaire with which to provide feedback.

6.1 Functional & Non-functional Testing

In order to perform functional/non-functional testing, the following steps are followed [Bartlett, 2015]:

1. Identify test input if applicable
2. Complete the expected outcomes with the test input if applicable
3. Execute test cases
4. Compare expected and actual outcomes

The purpose of using test cases is to test the implemented features rather than the requirements defined at the beginning of the project. Although test cases should ideally be derived from requirements, this is not always possible as it is often difficult to document all the client's expectations into requirements, so the requirement set is often incomplete. Test cases are written after the system has undergone some progress so they can be more specific and detailed and therefore not always traceable to initial requirements.

6.1.1 Functional Testing

Summary of Test Case Results

Table 6: Functional Test Cases

ID	Name	Acceptance Criteria	Actual Outcome	PASS /FAIL
F1	Access demo from FlexiTerm homepage	A link to the demo page should be inserted into the FlexiTerm homepage navigation menu.	A link to the demo is present in the FlexiTerm homepage menu (see Figure 33).	PASS
F2	Enter input	On the demo page, there should be a form with fields to allow one of three inputs: a url field, a plain text field and a file upload button.	The demo has a form through which users can enter input in file upload, text and url formats (see Figure 34).	PASS
F3	Validate input	1. The user must select one input format and provide an input for the chosen format (no	Input is validated to ensure it meets certain criteria. Invalid input prompts an error message	PASS

		<p>empty text fields or empty uploads).</p> <p>2. The user must insert input in acceptable format such e.g. word count \leq 500 words for text box or only .txt files for upload.</p> <p>If the above are not adhered to, an error message will be displayed and submission attempts will be unsuccessful.</p>	(see Section 5.2.2)	
F4	Submit input	The demo page should contain a submit button to submit user input.	The demo page contains a submit button for input submission (see figure 34).	PASS
F5	Process input	The back end should return a success message upon successful processing of the input.	The <i>processInput()</i> function returns a “Success” message if input has been successfully submitted (see figure 41).	PASS
F6	View results	Upon successful submission and processing, the user should be redirected to a results page where the output generated by FlexiTerm can be viewed in html, plain text and table formats.	The results page is loaded after input is successfully processed. This page displays data in html, plain text and table formats in different tabs (see figure 46).	PASS
F7	Download Results	The results page should also contain a list of links to download the <i>output.txt</i> , <i>output.html</i> , <i>output.mixup</i> and <i>output.csv</i> files generated by FlexiTerm.	The downloads tab contains a list of links to downloads output files (see figure 49).	PASS

In this case, it was possible to trace test cases directly to initial requirements, thus showing that by reverse-engineering existing systems, we are able to form a close-to-complete set of requirements as opposed to having to visualise the user’s requirements only. We can see from the results summary that the functionality works as expected.

6.1.2 Non-functional Requirements

The Non-functional requirements in [Section 2.1.2](#) will be tested based on the acceptance criteria which have already been written in a quantifiable way, rather than using test cases as user actions are not involved.

Table 7: Non-functional test cases

ID	Category	Acceptance Criteria	Actual Outcome	PASS /FAIL
NF1	Security	In all modern browsers, the entire FlexiTerm page must be HTTPS secure - no external links should be HTTP.	Browser console was used to identify all occurrences of mixed content (HTTP + HTTPS). All external links such as and web tracking links are HTTPS.	PASS
		No personal information such as username or password of the User server account the app is hosted on should be revealed in the code.	No usernames or passwords or any valuable information are saved in the code.	PASS
NF2	Reliability	The system must be fault-tolerant and catch and handle errors so that the application remains live.	The system is designed to not only catch errors on the client side, but to simply return 0 results if any errors occur on the server side rather than crashing the application.	PASS
NF3	Maintainability	As this is a straightforward application, the lines of code in each file (relevant to this application) must not exceed 1000 lines each.	All code written with regards to this application are < 300 lines each. Demo.html - 136 lines Flex.php - 116 lines Results.php - 279 lines	PASS
NF5	Performance	Demo and results page should load within 10 seconds, as this is around the limit for keeping the user's attention . Any longer, and the user may think an error has occurred.	The demo app loads for as long as it takes for FlexiTerm to process the user input. Average processing time is 7.048. Seconds [Appendix B] , which is less than the 10 second limit.	PASS
NF6	Consistency	The existing css files used for the FlexiTerm homepage must also be used for the demo page.	The CSS and HTML template was maintained throughout this project, as well as custom css in a separate file - <i>secondary.css</i> .	PASS

NF7	Usability	The application must adhere strictly to Jakob Nielsen's 10 Heuristics for User Interface design [Nielsen, 1995]	Details are documented in Section 6.2	PASS
		The application must score an above average score in usability testing		PASS
NF8	Compatibility	Application should maintain functionality and look when run on 5 of the most commonly used browsers	Details are documented in Appendix A	PASS

The above table shows that all non-functional requirements have been successfully implemented.

6.2 Usability Testing

It is often not enough to only assess the functionality of the system against initial requirements. To assess the quality and usability of a system, we employ usability testing. This can be done using standardised satisfaction questionnaires administered after the usability test session. It measures the user's impression of how easy it was to use the system in question [Mifsud, 2015]. Four different questionnaires can be used:

1. SUS: System Usability Scale (10 questions)
2. SUPR-Q: Standardized User Experience Percentile Rank Questionnaire (13 questions)
3. CSUQ: Computer System Usability Questionnaire (19 questions)
4. QUIS: Questionnaire For User Interaction Satisfaction (24 questions)
5. SUMI: Software Usability Measurement Inventory (50 questions)

To determine what questionnaire to use, we consider if there is enough budget, and how important satisfaction is. If the measurement of user satisfaction is important but there is not a large allocated budget, SUS is most appropriate [Mifsud, 2015]. SUS has become an "industry standard with references in over 1300 articles and publications" [Mifsud, 2015]. It consists of a short, simple scale that is easy to administer to participants, making it ideal for a small set of testers. In fact, As this is the case for this project, SUS was chosen as the questionnaire format for testing usability

When an SUS is used after a testing session, participants are asked to rate the system based on the following 10 questions with 5-point Likert scale ranging from “Strongly disagree” to “Strongly agree”. [Usability, n.d.].

1. I think that I would like to use this system frequently.
2. I found the system unnecessarily complex.
3. I thought the system was easy to use.
4. I think that I would need the support of a technical person to be able to use this system.
5. I found the various functions in this system were well integrated.
6. I thought there was too much inconsistency in this system.
7. I would imagine that most people would learn to use this system very quickly.
8. I found the system very cumbersome to use.
9. I felt very confident using the system.
10. I needed to learn a lot of things before I could get going with this system.

For this application, five participants were selected. This number was selected based on Nielsen's user testing research that found that if a system is tested with up to 5 users, new observations are made, with some repetitions but more than 5 users is unnecessary as only repetitions of past observations will abound [Nielsen, 2000].

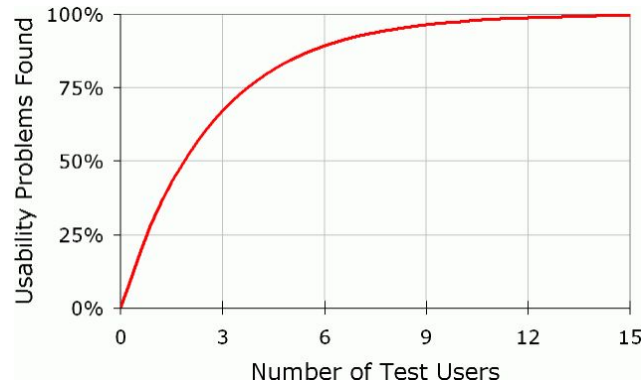


Figure 52: Diminishing Returns for usability testing [Nielsen, 2000]

The above graph shows diminishing returns for usability testing as more users are added. The curve starts to bend round 5 users - which is the recommended number of participants [Nielsen, 2000].

The participants were then asked to test the main functions of FlexiTerm: enter and submit input, and view and download output. After testing, they accessed the questionnaire and answered the questions as objectively as possible. The usability score is then calculated as follows [Usability, n.d.]:

For each odd number question, subtract 1 from score

1. For each even number question, subtract score from 5

2. Sum all 10 numbers up and multiply by 2.5 to convert to a scale of 0 - 100, from the original 0-100

Out of a total of 100, a SUS score greater than 68 would be considered above average and anything below 68 is below average [Usability, n.d.].

The results for the FlexiTerm usability test are as follows:

Tester 1

Question	1	2	3	4	5	6	7	8	9	10
Score	3	1	5	2	5	2	5	1	5	2
Converted score	2	4	4	3	4	3	4	4	4	3
Total	$31 * 2.5 = 77/100$									

Tester 2

Question	1	2	3	4	5	6	7	8	9	10
Score	5	2	4	1	4	2	5	1	5	1
Converted score	4	3	3	4	3	3	4	4	4	4
Total	$32 * 2.5 = 80/100$									

Tester 3

Question	1	2	3	4	5	6	7	8	9	10
Score	2	2	4	1	4	1	4	3	3	2
Converted score	1	3	3	4	3	4	3	2	2	3
Total	$28 * 2.5 = 70/100$									

Tester 4

Question	1	2	3	4	5	6	7	8	9	10
Score	4	1	4	1	4	1	4	1	4	3
Converted score	3	4	3	4	3	4	3	4	3	2
Total	$33 * 2.5 = 82/100$									

Tester 5

Question	1	2	3	4	5	6	7	8	9	10
Score	4	4	4	2	4	2	4	3	4	2
Converted score	3	1	3	3	3	3	3	2	3	2
Total	$26 * 2.5 = 65/100$									

Average

User	1	2	3	4	5
Score	77	80	70	82	65
Total	74.8				

Full survey results available at: www.surveymonkey.com/results/SM-YLLDJW27L/

6.3 Evaluation

The usability score for this application is calculated as $74.8/100$. This is an above-average score, indicating that this application is of good usability. One issue with this score is that the system was tested by up to 4 users at the same time. FlexiTerm can only handle 1 user at once, meaning other users were alerted that the server was busy and kept waiting, while FlexiTerm was processing another user's input. Another issue is that there was a bug with loading results from sessions which lead to no results, even for valid input. This has now been corrected in the final version of the demo app.

Another issue with the results is the potential factor of bias. It was made clear to the participants to answer as objectively as possible but due to the implications of too-low scores, it is possible the users scored it higher than they would have intended to.

Evaluation Against Nielsen's Heuristics

Table 8: Evaluation Against Nielsen's Heuristics

	Heuristic	Description	Evaluation
1	Visibility of system status	The application must provide informative feedback to the user.	<ol style="list-style-type: none"> 1. When the server is busy, an alert message is displayed to the user informing them. 2. When input is being

			processed, a loading bar is displayed for the duration.
2	Match between the system and the real world	The application must use real world language, or, language that can be easily understood by its users.	Even though this demo app is aimed at academics, the language used throughout is simple English
3	User control and freedom	The system must be fault-tolerant and allow users to reverse actions if need be.	<ol style="list-style-type: none"> 1. The Clear button on the input form allows users to remove entered input 2. The Reset button on the results page allows users to reset the output and run the application again
4	Consistency and standards	The application must use identical formatting and terminology throughout and the application; layout should be consistent for the entire webpage including the FlexiTerm home page.	The same CSS template is used throughout the entire site.
5	Recognition rather than recall	Where appropriate, the application must have important information and options visible at all times so that the user does not have to memorise anything.	There are no hidden features or information through the demo app. On the results page, the intuitive “+” icon is clearly visible for users who want to view additional data.
6	Flexibility and efficiency of use	Where appropriate, the system should be able to cater to users of all skill level.	The demo is designed to be as simple and straightforward as possible so that it is user-friendly to users of all skill level.
7	Error recovery	The app must display clear error messages that can help the user rectify their error.	When the user enters invalid input, they are prompted to enter valid input.
8	Error prevention	The app must have measures in place to ensure that errors are preventable	User input is validated against carefully chosen criteria to prevent errors in the back end.
9	Aesthetic and minimalist design	As this is a demo aimed primarily at scientists and researchers, only relevant information should be present and design should be kept	The demo page keeps strictly to requirements and no unnecessary features or information are added.

		to a minimum.	
10	Help and documentation	Any relevant information regarding FlexiTerm should be made available to the user.	This is already available on the FlexiTerm homepage.

Following usability testing, the system was evaluated against Nielsen's Usability Heuristics to ensure they were adhered to. Table 8 shows that all 10 heuristics were applied to the application. The above average usability score even with the limitations of the application demonstrates that following usability guidelines when designing and developing a system was a smart choice.

7.0 Conclusion

The aim of this project was to develop a means for users to process their text with FlexiTerm through their browsers. I believe I have achieved this to the best of my ability given the time and feasibility constraints.

Nearly all the features described in the initial plan were eventually implemented in the application, and those that were not have been added to future plans instead. The project was carried out using a combination of Agile and Waterfall methodologies in which initial planning, requirements analysis and user interface design were first completed, then development was carried out in short sprints.

The application maintains the functionality of FlexiTerm while making it more accessible and easier to use. It is a full stack web application consisting of a server and client side. The application simply provides a web interface through which users can pass input to FlexiTerm for processing and view the output in an appealing, readable format. The output files produced can also be downloaded by the users. User friendliness was kept in mind throughout development, with Nielsen's heuristics used as a guideline. Testing was carried out against the functional and nonfunctional requirements to ensure they were all implemented. Usability testing was also performed and the results showed a score of 74.2, indicating the system is of good usability, according to the System Usability Scale.

In conclusion, even with the minor issues encountered, this application has fulfilled its main requirements as described in the project brief.

The FlexiTerm web demo can be accessed online at users.cs.cf.ac.uk/AgbonyinAT/demo.html

8.0 Future Work

Table 9: Future Work

	Feature	Description
1	Sample data	As with Termine, sample data should be provided for users who simply want to try out FlexiTerm without having to search for possible sample data which may not always produce any results.
2	Allow html and pdf inputs	Currently, the demo only allows plain text inputs in all three input types (url, upload and text box). Future versions should be flexible enough so users can enter different file formats which can be converted to ASCII behind the scenes to be used by FlexiTerm.
3	Allow more input and multiple file uploads	The demo currently only allows short text input and one file upload per use due to server truncation issues. This can be improved upon in future work allowing multiple file uploads to demonstrate the full capabilities of FlexiTerm.
3	Allow users to customise FlexiTerm. Users should be able to do the following: <ul style="list-style-type: none"> - edit stoplist - edit distance threshold - edit minimum term candidate frequency - edit minimum (implicit) acronym frequency - edit acronym recognition mode (explicit/implicit) 	The demo should possess all the features of the back end application so users should be able to calibrate FlexiTerm according to their preference. This could not be implemented in this version because only one version of Flexiterm and consequently only one SQLite database file serves all users, and this can potentially cause more serious errors if the inner workings of flexiterm are being changed by several users at once.
4	Use a full SQL-enabled database	Rather than simply using an SQLite database file, Flexiterm data should be uploaded to a database hosted on an external server. This will be able to handle several users at a time for multiple concurrent writes, and increase the amount of input data acceptable by FlexiTerm.

9.0 Learning and Reflection

Project Management

Throughout this project, it was my duty to manage my time and tasks as effectively as possible. To aid this, I created a Gantt chart in my initial plan to schedule and organise the software development part of the project, as well as the report writing. In the early stages of development, however, I found that the Gantt chart was not very granular as it was written before the requirements and designs of the project were documented. So rather than containing details of specific tasks, it simply consisted of high level descriptions such as “complete front end” and this was not very useful to me. I prefer to use to-do lists in my everyday life, so I attempted to apply them to this project as well, but rather than using paper lists, I decided to try a different method: Kanban boards.

Despite only using Gantt charts for academic software projects, I have also used JIRA Kanban boards in the workplace and it has proved to be very useful and effective for me. JIRA is a paid tool, so I researched free Kanban board tools and decided upon Trello. The main advantage of Trello is that it is accessible online and locally on my desktop and mobile devices so I could keep track of my tasks wherever and whenever. For each week in the Gantt chart, I created a “To Do” board in Trello containing the list of tasks to be accomplished that week. This made it a lot easier to track my progress as I moved items from the To Do board to the “Doing” and “Done” boards. The use of Trello Kanban boards proved successful for this project and I will continue to use it in future projects, rather than only Gantt charts. This shows me that it is more realistic to tailor project management methods to my needs and preferences and the particular project I am working on, rather than going with a single default option each time.

Version Control

In the past, I have only used Tortoise SVN - a Windows desktop tool - for version control in my software projects. Git is widely used among development teams, so it is a valuable skill for any developer to have. Although my previous attempts to learn Git had proved unsuccessful, I saw this as another opportunity for me to attempt learning it once again. I discovered the Cardiff University GitLab and followed multiple online guides to learn how to use it. Although it was initially difficult, I was able to grasp it quickly enough and make full use of it for this project. I am working towards a career in software development so this eventually proved an enormous advantage as I am now able to include Git in my current repertoire of technologies.

Development and Deployment

As mentioned earlier, I was able to apply my past experience to this project, cutting down time that would have been otherwise spent learning new languages and technologies. This led to some amount of over-confidence. I did not spend as much time as was necessary planning out the project because I

wanted to start coding as soon as possible and this led to a few issues along the way. One mistake I made early in development was attempting to use too many frameworks at a time. For example, a fair amount of time was wasted attempting to apply FormValidation.io to validate the input forms and Bootstrap styling, rather than to simply write my own custom CSS and code. I should have also made test deployments to the Users server rather than simply running it locally. When I was about to start user acceptance testing, I attempted to deploy the app to the Users server only to run into several upload failures. Only after several failed attempts did I realise the error was due to 60mb limits on each user account on the server. If I had run mock deployments, I would have noticed this issue earlier. Fortunately, this was resolved on time. In the future, I will aim to properly plan my projects to prevent future problems.

Communication Skills

I made sure to attend the weekly meetings with my supervisor/client as often as I could in order to ensure that I was on the right track. When I was unavailable for a meeting, I emailed my supervisor with updates on my progress. The meetings enabled me to not only consult my supervisor - who is also the project client - but to liaise with other students during round-the-table meetings. In fact, usability testing for the FlexiTerm demo was conducted in one of these meetings.

As I was fairly confident in my abilities with regards to this project based on prior experience, I did not need to arrange personal meetings so the weekly meetings were sufficient. One thing I could have done differently was ask as many questions as possible to ensure the client's requirements were met early on. This may have prevented the issue of requirements changing in the later stages of development.

10.0 References

- Bartlett, J. (2015). *What is Functional Testing? - TestLodge Blog*. [online] TestLodge Blog. Available at: <https://blog.testlodge.com/what-is-functional-testing> [Accessed 10 Apr. 2018].
- Cohn, M. (2004). *Project Advantages of User Stories as Requirements*. [online] Mountain Goat Software. Available at: <https://www.mountaingoatsoftware.com/articles/advantages-of-user-stories-for-requirements> [Accessed 12 Apr. 2018].
- Hayward, A. (2013). *When to use User Stories, Use Cases and IEEE 830 Part 1*. [online] BA Times. Available at: <https://www.batimes.com/articles/when-to-use-user-stories-use-cases-and-ieee-830-part-1.html> [Accessed 12 Apr. 2018].
- Inflectra.com. (2018). *What is Waterfall & Hybrid Development?*. [online] Available at: <https://www.inflectra.com/methodologies/waterfall.aspx> [Accessed 12 Apr. 2018].
- Landis, L., Waligora, S., McGarry, F., Pajerski, R., Stark, M., Johnson, K. and Cover, D. (1992). *Recommended Approach to Software Development, Revision 3*. Software Engineering Laboratory Series. [online] Maryland: National Aeronautics and Space Administration, pp.6-7. Available at: <https://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/19930009672.pdf> [Accessed 20 Apr. 2018].
- Lonergan, K. (2016). *The Pros and Cons of Agile and Waterfall*. [online] Pmis-consulting.com. Available at: <https://www.pmis-consulting.com/agile-versus-waterfall> [Accessed 13 Apr. 2018].
- Marsic, I. (2012). *Software Engineering*. New Brunswick, New Jersey: Rutgers University, p.70.
- MDN Web Docs. (n.d.). *Cross-Origin Resource Sharing (CORS)*. [online] Available at: <https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS> [Accessed 20 Apr. 2018].
- Mifsud, J. (2015). *Usability Metrics – A Guide To Quantify The Usability Of Any System*. [online] Usability Geek. Available at: <https://usabilitygeek.com/usability-metrics-a-guide-to-quantify-system-usability> [Accessed 2 May 2018].
- Nactem.ac.uk. (2012). *Termine Web Demonstrator*. [online] Available at: <http://www.nactem.ac.uk/software/termine> [Accessed 8 Apr. 2018].
- Naumovski, A. (2017). *Straightening out the React/Redux learning curve part 1 - Intro to React*. [online] The Practical Dev. Available at: <https://dev.to/andrejnaumovski/straightening-out-the-reactredux-learning-curve-part-1---intro-to-react-18b> [Accessed 20 Apr. 2018].
- Neoteric. (2016). *Single-Page Application vs. Multiple-Page Application - Neoteric*. [online] Available at: <https://neoteric.eu/single-page-application-vs-multiple-page-application> [Accessed 26 Apr. 2018].
- Netmarketshare (2018). *Browser market share*. [online] Available at: <https://netmarketshare.com/browser-market-share.aspx> [Accessed 1 May 2018].

Nielsen, J. (1995). *10 Heuristics for User Interface Design: Article by Jakob Nielsen*. [online] Nielsen Norman Group. Available at: <https://www.nngroup.com/articles/ten-usability-heuristics> [Accessed 1 May. 2018].

Nielsen, J. (1997). *The Need for Speed*. [online] Nielsen Norman Group. Available at: <https://www.nngroup.com/articles/the-need-for-speed> [Accessed 26 Apr. 2018].

Nielsen, J. (2000). *Why You Only Need to Test with 5 Users*. [online] Nielsen Norman Group. Available at: <https://www.nngroup.com/articles/why-you-only-need-to-test-with-5-users> [Accessed 2 May 2018].

Spasić, I., Greenwood, M., Preece, A., Francis, N. and Elwyn, G. (2013). FlexiTerm: a flexible term recognition method. *Journal of Biomedical Semantics*, [online] 4(1), p.27. Available at: <https://jbiomedsem.biomedcentral.com/articles/10.1186/2041-1480-4-27> [Accessed 25 Apr. 2018].

StatCounter Global Stats. (2018). *Desktop Browser Market Share United Kingdom* / StatCounter Global Stats. [online] Available at: <http://gs.statcounter.com/browser-market-share/desktop/united-kingdom> [Accessed 3 May 2018].

Statista. (2018). *Most popular internet browser versions 2018* / Statistic. [online] Available at: <https://www.statista.com/statistics/268299/most-popular-internet-browsers> [Accessed 3 May 2018].

Stephens, R. (2015). *Beginning software engineering*. Indianapolis, Indiana: John Wiley & Sons, Inc., p.63.

Stringfellow, A. (2017). *What is Web Application Architecture? How It Works, Trends, Best Practices and More*. [online] stackify.com. Available at: <https://stackify.com/web-application-architecture> [Accessed 26 Apr. 2018].

Translated Labs (n.d). *Terminology Extraction*. [online] Available at: <https://labs.translated.net/terminology-extraction> [Accessed 8 Apr. 2018].

Usability.gov. (n.d.). *System Usability Scale (SUS)* / Usability.gov. [online] Available at: <https://www.usability.gov/how-to-and-tools/methods/system-usability-scale.html> [Accessed 4 May 2018].

Yaskevich, A. (2017). *Web application architecture: Components, models and types*. [online] Scnsoft.com. Available at: <https://www.scnsoft.com/blog/web-application-architecture> [Accessed 26 Apr. 2018].

Appendix A

Browser Compatibility

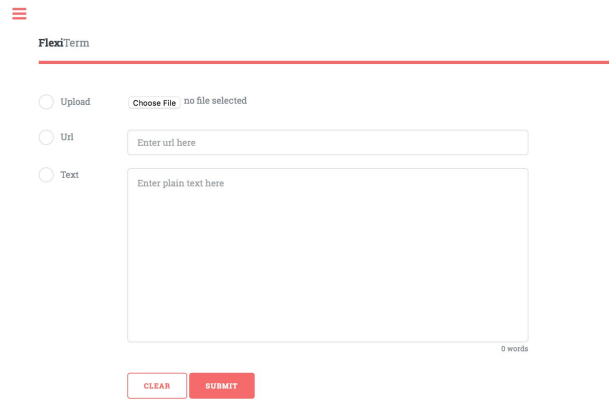


Figure 27: Demo homepage - Chrome

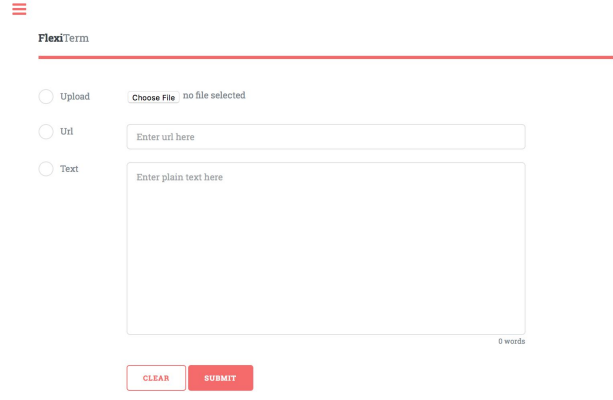


Figure 28: Demo homepage - Firefox

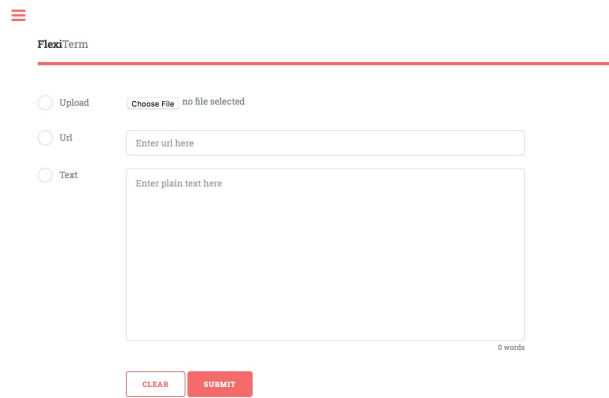


Figure 29: Demo homepage - IE11

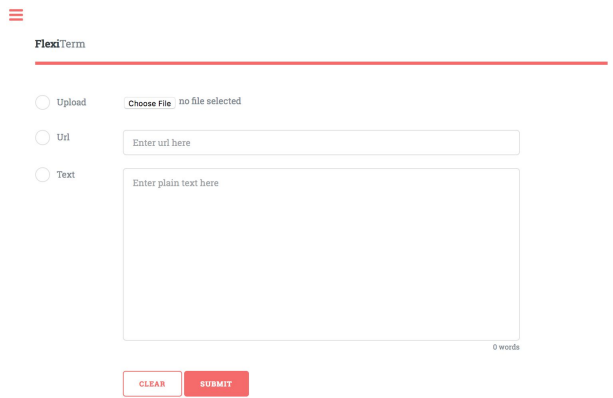


Figure 30: Demo homepage - Edge

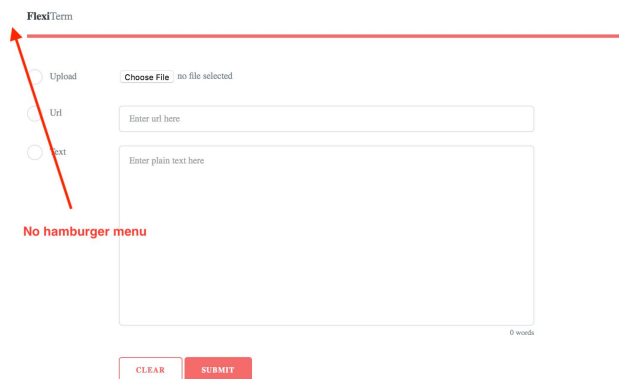


Figure 31: Demo homepage - IE7

Results View

IE7: Menu icon is absent but everything else displays as expected.

FlexiTerm

Reset42 results

View as tableView as HTML pageDownload

Show: 10entries

Search:

Rank	Term	Termhood Score
1	tumor necrosis factor alpha	16.358
Normalised TermTerm Variants		
alpha factor necrosis tumortumor necrosis factor alpha		
2	human immunodeficiency virus	6.225
3	trif-alpha treatment	3.219
4	hiv tr	2.773
5	beta interferon promoter	2.187
6	u937 cells	2.079
7	hiv infected u937 cell line	1.946
8	hybrid promoters by trif-alpha	1.792
9	steady-state level of trif-alpha rna	1.792
9	beta interferon promoters by tumor necrosis factor	1.792

Showing 1 to 10 of 42 entriesPrevious12345Next

Figure 32: Table Results - IE7

FlexiTerm

Reset42 results

View as tableView as HTML pageDownload

Download Results

▲ Mixup file

▲ Csv file

▲ HTML file

▲ Plain text file

Note: Files will be deleted after session expires.

Figure 34: Download Links - IE7

IE11:

FlexiTerm

Reset42 results

View as tableView as HTML pageDownload

Show: 10entries

Search:

Rank	Term	Termhood Score
1	tumor necrosis factor alpha	16.358
Normalised TermTerm Variants		
alpha factor necrosis tumortumor necrosis factor alpha		
2	human immunodeficiency virus	6.225
3	trif-alpha treatment	3.219
4	hiv tr	2.773
5	beta interferon promoter	2.187
6	u937 cells	2.079
7	hiv infected u937 cell line	1.946
8	hybrid promoters by trif-alpha	1.792
9	steady-state level of trif-alpha rna	1.792
9	beta interferon promoters by tumor necrosis factor	1.792

Showing 1 to 10 of 42 entriesPrevious12345Next

Figure 35: Table results - IE7

FlexiTerm

Reset42 results

View as tableView as HTML pageDownload

Cell-specific differences in activation of NF-kappa B regulatory elements of human immunodeficiency virus and beta interferon promoters by tumor necrosis factor. Three aspects of the mechanism of tumor necrosis factor (TNF) in human immunodeficiency virus (HIV) pathogenesis were examined. Tumor necrosis factor alpha (TNF-alpha) mRNA production was

Figure 33: HTML Results - IE7

FlexiTerm

Reset42 results

View as tableView as HTML pageDownload

Cell-specific differences in activation of NF-kappa B regulatory elements of human immunodeficiency virus and beta interferon promoters by tumor necrosis factor. Three aspects of the mechanism of tumor necrosis factor (TNF) in human immunodeficiency virus (HIV) pathogenesis were examined. Tumor necrosis factor alpha (TNF-alpha) mRNA production was

Figure 36: HTML results - IE7

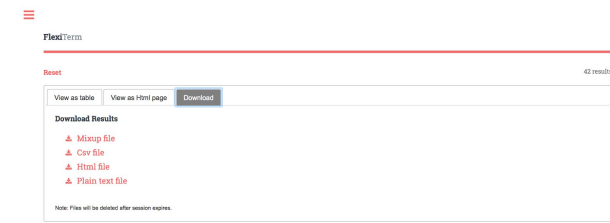


Figure 37: Download Links - IE7

Google Chrome:

The screenshot shows the FlexTerm web application with the 'View as table' tab selected. It displays a table of search results. The table has columns for Rank, Term, and Termhood Score. The first result is 'tumor necrosis factor alpha' with a score of 16.368. Below the table, there's a section for 'Normalized Term' and 'Term Variants'.

Rank	Term	Termhood Score
1	tumor necrosis factor alpha	16.368
2	human immunodeficiency virus	6.225
3	Tnf-alpha treatment	3.219
4	hiv tr	2.773
5	beta interferon promoter	2.197
6	u937 cells	2.079
7	hiv infected u937 cell line	1.946
8	hybrid promoters by Tnf-alpha	1.792
9	steady-state level of Tnf-alpha ms	1.792
9	beta interferon promoters by tumor necrosis factor	1.792

Figure 35: Table results - IE7

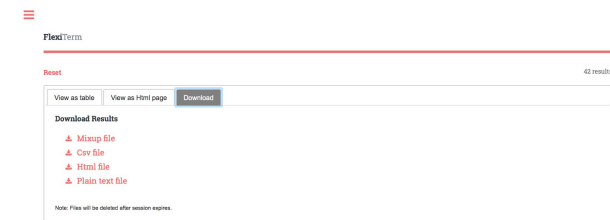


Figure 37: Download Links - IE7

Firefox:

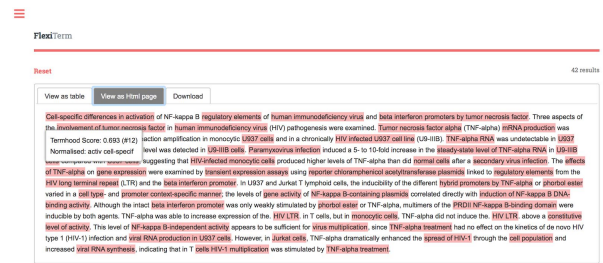


Figure 36: HTML results - IE7

FlexiTerm

Reset 42 results

View as table View as HTML page Download

Show: 10 entries Search:

Rank	Term	Termhood Score
1	tumor necrosis factor alpha	16.358
Normalised Term Term Variants		
alpha factor necrosi tumor tumor necrosis factor alpha		
2	human immunodeficiency virus	6.225
3	trif-alpha treatment	3.219
4	hiv tr	2.773
5	beta interferon promoter	2.197
6	u937 cells	2.079
7	hiv infected u937 cell line	1.946
8	hybrid promoters by trif-alpha	1.792
9	steady-state level of trif-alpha ma	1.792
9	beta interferon promoters by tumor necrosis factor	1.792

Showing 1 to 10 of 42 entries Previous 1 2 3 4 5 Next

Figure 35: Table results - IE7

FlexiTerm

Reset 42 results

View as table View as HTML page Download

Download Results

- Mixup file
- Csv file
- HTML file
- Plain text file

Note: Files will be deleted after session expires.

Figure 37: Download Links - IE7

Edge:

FlexiTerm

Reset

View as table View as HTML page Download

Show: 10 entries Search:

Rank	Term	Termhood Score
1	tumor necrosis factor alpha	16.358
Normalised Term Term Variants		
alpha factor necrosi tumor tumor necrosis factor alpha		
2	human immunodeficiency virus	6.225
3	trif-alpha treatment	3.219
4	hiv tr	2.773
5	beta interferon promoter	2.197
6	u937 cells	2.079
7	hiv infected u937 cell line	1.946
8	hybrid promoters by trif-alpha	1.792
9	steady-state level of trif-alpha ma	1.792
9	beta interferon promoters by tumor necrosis factor	1.792

Showing 1 to 10 of 42 entries Previous 1 2 3 4

Figure 35: Table results - IE7

FlexiTerm

Reset 42 results

View as table View as HTML page Download

Cell-specific differences in activation of NF-kappa B regulatory elements of human immunodeficiency virus and beta interferon promoters by tumor necrosis factor. Three aspects of the involvement of tumor necrosis factor in human immunodeficiency virus (HIV) pathogenesis were examined. Tumor necrosis factor alpha (TNF-alpha) mRNA production was detected in chronically HIV infected U937 cells and in a chronically HIV infected U937 cell line (U9-118). TNF-alpha RNA was undetectable in U937 cells. In U9-118 cells, TNF-alpha RNA was detected in U9-118 cells. Paramyxovirus infection induced a 5- to 10-fold increase in the steady-state level of TNF-alpha RNA in U9-118 cells. These results suggest that HIV-infected monocytes produce higher levels of TNF-alpha than do normal cells after a secondary virus infection. The effects of TNF-alpha on gene expression were examined by transient expression assays using reporter chloramphenicol acetyltransferase plasmids linked to regulatory elements from the HIV long terminal repeat (LTR) and the beta interferon promoter. In U937 and Jurkat T lymphoid cells, the inducibility of the different hybrid promoters by TNF-alpha or phorbol ester varied in a cell-type- and promoter-context-specific manner; the levels of gene activity of NF-kappa B-containing plasmids correlated directly with induction of NF-kappa B DNA binding activity. Although the intact beta interferon promoter was only weakly stimulated by phorbol ester or TNF-alpha, mutants of the pRBD-NF-kappa B-binding domain were inducible by both agents. TNF-alpha was able to increase expression of the HIV LTR in T cells, but in monocytes cells, TNF-alpha did not induce the HIV LTR, above a control level of activity. This level of NF-kappa B-independent activity appears to be sufficient for virus multiplication, since TNF-alpha treatment had no effect on the kinetics of de novo HIV type 1 (HIV-1) infection and viral RNA production in U937 cells. However, in Jurkat cells, TNF-alpha dramatically enhanced the spread of HIV-1 through the cell population and increased viral RNA synthesis, indicating that in T cells HIV-1 multiplication was stimulated by TNF-alpha treatment.

Figure 36: HTML results - IE7

FlexiTerm

Reset

View as table View as HTML page Download

Cell-specific differences in activation of NF-kappa B regulatory elements of human immunodeficiency virus and beta interferon promoters by tumor necrosis factor. Three aspects of the involvement of tumor necrosis factor in human immunodeficiency virus (HIV) pathogenesis were examined. Tumor necrosis factor alpha (TNF-alpha) mRNA production was detected in chronically HIV infected U937 cells and in a chronically HIV infected U937 cell line (U9-118). TNF-alpha RNA was undetectable in U937 cells. In U9-118 cells, TNF-alpha RNA was detected in U9-118 cells. Paramyxovirus infection induced a 5- to 10-fold increase in the steady-state level of TNF-alpha RNA in U9-118 cells. These results suggest that HIV-infected monocytes produce higher levels of TNF-alpha than do normal cells after a secondary virus infection. The effects of TNF-alpha on gene expression were examined by transient expression assays using reporter chloramphenicol acetyltransferase plasmids linked to regulatory elements from the HIV long terminal repeat (LTR) and the beta interferon promoter. In U937 and Jurkat T lymphoid cells, the inducibility of the different hybrid promoters by TNF-alpha or phorbol ester varied in a cell-type- and promoter-context-specific manner; the levels of gene activity of NF-kappa B-containing plasmids correlated directly with induction of NF-kappa B DNA binding activity. Although the intact beta interferon promoter was only weakly stimulated by phorbol ester or TNF-alpha, mutants of the pRBD-NF-kappa B-binding domain were inducible by both agents. TNF-alpha was able to increase expression of the HIV LTR in T cells, but in monocytes cells, TNF-alpha did not induce the HIV LTR, above a control level of activity. This level of NF-kappa B-independent activity appears to be sufficient for virus multiplication, since TNF-alpha treatment had no effect on the kinetics of de novo HIV type 1 (HIV-1) infection and viral RNA production in U937 cells. However, in Jurkat cells, TNF-alpha dramatically enhanced the spread of HIV-1 through the cell population and increased viral RNA synthesis, indicating that in T cells HIV-1 multiplication was stimulated by TNF-alpha treatment.

Figure 36: HTML results - IE7

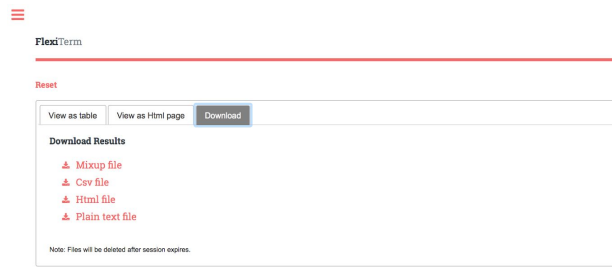


Figure 37: Download Links - IE7

Appendix B

The average input processing time for 5 runs of the FlexiTerm demo is calculated below:

Run	Speed (s)
1	7.13
2	7.02
3	7.22
4	6.79
5	7.08
Average	$34.24 / 5 = 7.048$. seconds