



UNIVERSITY OF HUDDERSFIELD

MENG GROUP PROJECT

Cryptic Crossword Solver

TEST DOCUMENT

Authors:

Mohammad RAHMAN

Leanne BUTCHER

Stuart LEADER

Luke HACKETT

Supervisor:

Dr. Gary ALLEN

Examiner:

Dr. Sotirios BATSAKIS

Moderator:

Dr. Colin VENTERS

Friday, 9th May 2014

Contents

1	Testing	4
1.1	Test Strategy	5
1.1.1	Approaches	5
1.1.2	Environments & Tools	5
1.1.3	Test Schedule	6
1.1.4	Test Priorities	6
1.1.5	Test Groups	7
1.1.6	Test Reports	7
1.2	Unit Testing	8
1.2.1	Core Suite	8
1.2.2	Config Suite	10
1.2.3	Resource Suite	10
1.2.4	Servlet Suite	11
1.2.5	Solver Suite	12
1.2.6	Util Suite	13
1.3	Walk-through	15
1.4	Functional Testing	23
1.4.1	Functional Requirements	23
1.4.2	Usability Requirements	28
	Glossary of Terms	36

List of Figures

1.1	Cryptic clue with results retrieved from web service	23
1.2	Input fields for obtaining characteristics of a cryptic clue	24
1.3	Web page in full screen browser window	24
1.4	Web page in resized browser window	25
1.5	Trace for solution ‘stolen’ with clue; ‘Among library books to lend and not returned’, with solution; ‘stolen’	26
1.6	Trace for solution ‘return’ with clue; ‘Among library books to lend and not returned’, with solution; ‘stolen’	26
1.7	A solution returned with a confidence score displayed of 87% for the clue; ‘Advance in either direction’	27
1.8	Text field to enter clue	28
1.9	Attempting to enter a length of sixteen - the red colour signifies it is invalid	29
1.10	Attempting to enter two words of length fifteen - the green colour signifies it is valid	29
1.11	Attempting to enter three words of valid length with a space and a hyphen - the green colour signifies it is valid	29
1.12	Boxes that have been generated for a solution with the configuration; 2-3,2 .	30
1.13	Results displayed for the clue; ‘A clever rhyme or subtle teaser I constitute’ .	30
1.14	If the user submits without filling in either required fields	31
1.15	If the user submits without filling in the length of the solution	31
1.16	If the user submits without filling in the clue field	31
1.17	Web page in full screen browser window	32
1.18	Web page in resized browser window	32
1.19	Confidence ratings displayed with the solutions returned	33
1.20	Introduction to the user manual	34
1.21	Submit button for user to click	34

List of Tables

1.7	Non-JavaScript enabled walk-through test results	18
1.8	JavaScript enabled walk-through test results	22

Chapter 1

Testing

The testing phase was an ongoing process to adhere to the guidelines of agile development which has been used throughout the project. The development phase was split across three iterations meaning at the end of each, JUnit tests were written and run to ensure all functionality for each iteration was working correctly. This meant any functionality being added in the new iteration using the functionality written in a previous iteration would not be delayed as previous code was reliable and complete.

At the end of development, functional testing and walkthroughs of the user interface were completed. The functional testing took all functional requirements declared at the beginning of the project to ensure they were all fulfilled and working correctly. The walkthroughs of the user interface involved entering data into the system to determine whether the system would fall over with various valid and invalid input entered.

The following chapter will document all the testing discussed above including descriptions of the processes completed and evidence of the system passing various tests with different test strategies.

1.1 Test Strategy

Within this section the proposed test strategy used through the testing stage of the project will be outlined. A test strategy describes the various approaches to testing the development phase of a software development project.

1.1.1 Approaches

The testing approaches that will be used during the testing of this project will be unit testing, integration testing, system testing and requirements testing. Each of these types of tests will individually focus upon one aspect of testing, but when combined together will form a larger, more complete test approach.

Due to the size of the group, the tests will be conducted by those who will have been developing the software — ideally these two tasks would be completed by separate teams.

The unit testing will be completed using the JUnit test framework and will test the various ‘back end’ classes that make up the system. The unit tests will also cover many of the integration tests, which ensures that the classes when combined together are functioning correctly.

A number of full system walkthroughs will also be conducted manually. This will ensure that the user interface has been correctly developed, as well as ensuring that the user interface is as fluid as possible.

Finally the software product will undergo requirements testing. This will compare the original specification against the software product, and will provide evidence of the feature being implemented. If a feature has not been implemented, a reason as to why not will be discussed.

1.1.2 Environments & Tools

The test environment and tools are similar to those required within the development phase. To clarify these environments and tools are:

- Java 7
- Apache Tomcat 7

As well as the above fundamental tools, some additional testing tools and libraries will be required, and are outlined below:

- JUnit 4 library
- Apache HTTP Components library

- MySQL Database

The JUnit 4 library is used extensively throughout the testing phase to ensure that the code is correctly working upon various levels.

The Apache HTTP Components library provides low level access to the HTTP protocol, which enables for responses to be tested against their subsequent requests. For example if the HTTP return type header was XML does the system actually deliver XML through the HTTP protocol.

Finally the MySQL database stores a number of cryptic clues, their solutions and the type of clue it is. This will enable the system to be tested with real data and thus test to ensure the algorithms are correctly generating the correct results. It will also help to test that the results are being ‘ranked’ appropriately using the confidence rating.

1.1.3 Test Schedule

In order to prevent the testing phase becoming too long, a time limit of 20% per phase has been set. This means that up to 20% of the allocated time for a given phase will be devoted to testing that phase.

It must be said that testing will be conducted alongside development, and therefore many of the major and common aspects of the system should have had some form of (basic) testing. The 20% time limit has been put in place to ensure that there is some time set a side for testing.

The time allocation will also be used to help schedule how long a development phase should last, and thus in directly controls how much testing will need to be conducted.

1.1.4 Test Priorities

The test priorities will be in line with the MoSCoW analysis. This will mean that requirements that fall under the ‘Must’ and ‘Should’ categories will by default have a higher testing priority over ‘Could’ categories.

The project is focusing upon the ‘back end’ functionality as much as possible, and therefore there will be no automated user interface tests. This also means that ‘back end’ tests will have a higher priority over user interface tests. However it must be stated that the reduction in priority does not mean that there will be a lack in quality.

1.1.5 Test Groups

In order to ensure that the product is fit for its intended purpose it was proposed that a number of quality circles would be run. These circles known as ‘test groups’, would identify issues within the product whilst also providing an ‘idealistic solution’. The solutions are branded as idealistic as they may not always be feasible with the resources that are available.

Although one or more test groups would have provided various levels of feedback it was decided that the current levels of resources did not permit a full evaluation of the results. However if more time (and resources) were available the group would undertake one or more test groups as part of the testing cycle.

1.1.6 Test Reports

For each of the highlighted testing areas — unit testing, integration testing, system testing and requirements testing — a dedicated subsection will be made available as part of the testing section that makes up the written report.

Each of the subsections will highlight the specific details of each type of test, and present the results.

It must be stated that the reports will only show the latest phase of tests. Prehistoric tests will not be presented to avoid repetition and outdated information.

1.2 Unit Testing

As part of the test strategy — outlined in the previous section — the system will under go unit testing. The unit tests are designed to test each individual class upon their own as well as in combination with other classes.

The subsections within this section will discuss each of test suites that are designed to test each of the packages that are used within the system.

1.2.1 Core Suite

The core suite focuses upon testing the various classes that make up the core package. Each of the individual core data classes — Clue, Solution and SolutionPattern — are tested upon their individual properties and characteristics.

For example test 8 will test to ensure the given clue is able to return an array of words, whilst test 38 will test to ensure that the SolutionPattern correctly returns the number of words expected in the solution.

The SolutionCollection test will test the various specific methods associated with the SolutionCollection, and will not test the base methods that have been overridden from the base class — Set. For example test 33 ensures that a given SolutionCollection returns the best solution — i.e. the solution with the highest confidence rating.

Finally the Manager test class will test to ensure that a number of solvers are correctly instantiated and spread across as many available threads upon the machine running the code. It will also ensure that the correct solution is returned within the list of solutions.

Test No.	Test	Outcome
Clue Test		
1	testClueStringString	PASS
2	testGetActualSolution	PASS
3	testGetPattern	PASS
4	testGetSolutions	PASS
5	testGetType	PASS
6	testGetClueNoWithPunctuation	PASS
7	testGetClueNoPunctuationNoSpaces	PASS
8	testGetClueWords	PASS
9	testGetBestSolution	PASS
10	testGetClue	PASS
Manager Test		
11	testDistributeAndSolveClue	PASS
Solution Test		

12	testCompareToSameConfidencesAndDifferentSolutions	PASS
13	testCompareToDifferentConfidencesAndSameSolutions	PASS
14	testCompareToDifferentConfidencesAndSolutions	PASS
15	testEqualsDifferentSolutionAndConfidence	PASS
16	testEqualsSameSolutionAndConfidence	PASS
17	testEqualsSameSolutionAndDifferentConfidence	PASS
18	testEqualsDifferentSolutionAndSameConfidence	PASS
19	testGetConfidence	PASS
20	testGetSolution	PASS
21	testSetConfidence	PASS
22	testToString	PASS
23	testAddToTraceAndGetSolutionTrace	PASS
23	testCompareToSameConfidencesAndSolutions	PASS
SolutionCollection Test		
24	testContainsString	PASS
25	testGetSolutionsGreaterThanOrEqualTo	PASS
26	testGetSolutions	PASS
27	testGetSolutionsLessThan	PASS
28	testRemoveAllStrings	PASS
29	testSortSolutions	PASS
30	testAddNewSolution	PASS
31	testAddDuplicate	PASS
32	testAddIncreaseConfidence	PASS
33	testGetBestSolution	PASS
SolutionPattern Test		
34	testToString	PASS
35	testToPattern	PASS
36	testStaticMatch	PASS
37	testGetKnownChars	PASS
38	testGetNumberOfWords	PASS
39	testRecomposeSolution	PASS
40	testGetTotalLength	PASS
41	testHasMultipleWords	PASS
42	testFilterSolutions	PASS
43	testGetIndividualWordLengths	PASS
44	testMatch	PASS
45	testGetPattern	PASS
46	testSeparateSolution	PASS
47	testSplitPattern	PASS

1.2.2 Config Suite

The config suite focuses upon testing the configuration class, settings. The suite ensures that each of the methods are correctly returning the expected values. The tests within this suite are relatively straightforward, as it is a case of ensuring the correct values are returned.

Test No.	Test	Outcome
Settings Test		
48	testGetCustomDictionaryStream	PASS
49	testGetDictionaryExclusionsStream	PASS
50	testGetPropertyStream	PASS
51	testGetPOSModelStream	PASS
52	testGetChunkerModelStream	PASS
53	testGetFilename	PASS
54	testGetIndicatorsURL	PASS
55	testGetDBURL	PASS
56	testGetCategoriserModelStream	PASS
57	testGetSentenceDetectorModelStream	PASS
58	testGetParserModelStream	PASS
59	testGetTokeniserModelStream	PASS
60	testGetThesaurusStream	PASS
61	testGetHomophoneDictionaryStream	PASS
62	testGetDBUsername	PASS
63	testGetDBPassword	PASS
64	testGetDictionarStream	PASS

1.2.3 Resource Suite

The Resource Suite tests all the resources that are used to aid in the solving of clues to ensure the methods interacting with them are returning the correct data. The resources are the various dictionary, thesaurus, and abbreviation files.

By running the tests, the initial reading of the files and population of collections to hold the data is tested. For testing the correct return of data, samples were taken from the files themselves to make is possible to use the ‘assertEquals’ method provided within the JUnit library to get exact expected matches.

Test No.	Test	Outcome
HomophoneDictionary Test		
65	testGetPronunciations	PASS
66	testGetHomonyms	PASS
Dictionary Test		

67	testMatchesWithPrefix	PASS
68	testMatchesWithNonString	PASS
69	testMatchesWithNonPrefix	PASS
70	testDictionaryFilter	PASS
71	testIsDictionaryWord	PASS
72	testWrongDictionaryWord	PASS
73	testBlankDictionaryWord	PASS
74	testMatchingWords	PASS
75	testMatchingWordsNonString	PASS
76	testMatchingWordsNonWord	PASS
77	testPrefixMatch	PASS
78	testPrefixMatchWithNonPrefix	PASS
79	testPrefixMatchWithNonString	PASS
Thesaurus Test		
80	testSynonymMatch	PASS
81	testGetSynonyms	PASS
82	testGetSpecificSynonyms	PASS
83	testGetSpecificSynonymsNoKnownChars	PASS
84	testGetSecondSynonymsWithPattern	PASS
85	testGetSecondSynonymsMinMaxLength	PASS
86	testConfidenceAdjust	PASS
Abbreviations Test		
87	testGetAbbreviationsForWord	PASS
88	testGetAbbreviationsForClue	PASS

1.2.4 Servlet Suite

The servlet suite focuses upon testing the server side request handling and validations. The automated testing of the servlet required the use of the Apache Http Components project, which provides a number of low level tools that enables HTTP protocols to be accessed.

In order for the system to be able to correctly compute a list of possible solutions the servlet requires the clue, the solution pattern, and the solution length.

Tests 89 – 95 attempt to give only one or two of the required parameters to ensure that the request is rejected. A rejected request will return an error (in either XML or JSON) explaining what is incorrect.

Tests 96 and 97 give the three required parameters, but make the pattern and solution lengths unmatchable — for example a solution length of 3,4 should not be matched to ?????,?? as that would have a solution length of 5,2.

Test No.	Test	Outcome
SolverServlet Test		
89	testNoParameters	PASS
90	testClueParameterOnly	PASS
91	testClueSolutionParameterOnly	PASS
92	testCluePatternParameterOnly	PASS
93	testSolutionPatternParameterOnly	PASS
94	testSolutionParameterOnly	PASS
95	testPatternParameterOnly	PASS
96	testInvalidPattern	PASS
97	testInvalidSolution	PASS

1.2.5 Solver Suite

Within the Solver Suite each solver has its own test class and each solver is tested the same way. Each test class (e.g. ‘PalindromeTest’, ‘HiddenTest’, etc.) extends the ‘SolverTest’ class and is set up to use the ‘testSolve’ method in the class being extended. The separate solver classes use the ‘BeforeClass’ annotation within JUnit to get the name of the solver, the solver itself, the number of clues to test against (twenty for each solver) and setting the unknown characters to true.

When the suite and each solver test is run, all the details are set and the ‘testSolve’ method is called passing in the details as parameters. This method retrieves twenty clues from the database to test (with the correct clue type assigned to it) and runs one clue on each thread available. For each clue retrieved from the database, the ‘solve’ method is called for the correct solver and the potential solutions returned from the solver are checked to determine whether the actual solution, which is stored in the database with the clue, has been found.

Test No.	Test	Outcome
Acrostic Test		
98	testAlgorithm	PASS
Anagram Test		
99	testAlgorithm	PASS
Hidden Test		
100	testAlgorithm	PASS
Pattern Test		
101	testAlgorithm	PASS
DoubleDefinition Test		
102	testAlgorithm	PASS
Homophone Test		

103	testAlgorithm	PASS
Spoonerism Test		
104	testAlgorithm	PASS
Palindrome Test		
105	testAlgorithm	PASS
Reversal Test		
106	testAlgorithm	PASS
Deletion Test		
107	testAlgorithm	PASS
Container Test		
108	testAlgorithm	PASS
Charade Test		
109	testAlgorithm	PASS

1.2.6 Util Suite

The util suite focuses upon testing the various classes that make up the util package. As the util package mainly provides helper methods, the classes have been tested individually, however many of the classes would have been tested during the other test suites.

The XMLBuilder test class tests to ensure that a valid XML document can be created based upon a number of input values, including solution collections. It is imperative that the XMLBuilder class creates well-formed XML code, as this is used in the responses that are generated by the web service.

The WordUtils and Confidence test classes ensure that these fundamental classes correctly return their data in order to ensure that usage within the system will always be correct.

Test No.	Test	Outcome
DB Test		
111	testGetTestClues7	PASS
112	testGetTestClues8	PASS
113	testGetTestClues1	PASS
114	testGetTestClues2	PASS
115	testGetTestClues3	PASS
116	testGetTestClues4	PASS
117	testGetTestClues5	PASS
118	testGetTestClues6	PASS
WordUtils Test		
119	testHasCharacters	PASS
120	testNormaliseInput	PASS

121	testRemoveSpacesAndHyphens	PASS
122	testCharactersPresentInWord	PASS
Confidence Test		
123	testMultiply	PASS
Cache Test		
124	testPut	PASS
125	testContainsKey	PASS
126	testPrePut	PASS
127	testGet	PASS
XMLBuilder Test		
128	testXMLBuilder	PASS
129	testXMLBuilderStringString	PASS
130	testXMLBuilderStringStringDouble	PASS
131	testAddKeyValue	PASS
132	testAddError	PASS
133	testAddErrors	PASS
134	testAddSolution	PASS

1.3 Walk-through

The previous section focused upon automated unit tests that were able to programmatically deduce the correctness of output. Although unit testing is a major part of the overall testing strategy it is not the only strategy.

Within this section a number of objective based software walk-throughs will be conducted. A software walk-through is an analysis technique that requires all interested parties to ask questions and make comments about possible anomalies, violation of development standards, and other problems (IEEE, 2008).

As described within the development section, the user interface makes use of JavaScript, and hence there will be two walk-throughs, one with JavaScript enabled and another with JavaScript disabled.

Non-JavaScript

As part of the testing process, the user interface's default view will be tested through a series of walk-throughs. These tests are designed to test to ensure that a user is unable to break the user interface.

All of the tests shown in table 1.7 were conducted in a browser that had JavaScript disabled.

Test Number	Test Data	Reason For Test	Expected Outcome	Actual Outcome	Corrective Action Key	Notes
Clue Input						
1	cryptic clue = nil solution length = nil known characters = nil	To ensure that the server rejects incomplete submitted data.	When the page is reloaded, error messages are shown informing the user that required fields have not been given.	As expected.	None.	None.
2	cryptic clue = “We hear twins shave” solution length = nil known characters = nil	To ensure that the server rejects incomplete submitted data.	When the page is reloaded, error messages are shown informing the user that required fields have not been given.	As expected.	None.	None.
3	cryptic clue = nil solution length = “4” known characters = nil	To ensure that the server rejects incomplete submitted data.	When the page is reloaded, three error messages are shown informing that required fields have not been given.	As expected.	None.	None.

8	cryptic clue = “We hear twins shave” solution length = “4” known characters = “????”	To ensure that the correct result is returned upon the submission of a valid form with a solvable clue.	A number of solutions should be returned, with the correct solution – “pair” – holding the highest confidence rating.	As expected.	None.	41 solutions were returned.
9	cryptic clue = “We hear twins shave” solution length = “4” known characters = “p??r”	To ensure that the correct result is returned upon the submission of a valid form with a solvable clue.	A reduced result set in comparison to test #8, however the correct solution – “pair” – should have the highest confidence rating.	As expected.	None.	2 solutions were returned.
10	cryptic clue = “We hear twins shave” solution length = “4” known characters = “????”	To ensure that the all results as shown, with their solution trace’s open, and viewable.	All solution traces are open, and viewable by default.	As expected.	None.	None.

Table 1.7: Non-JavaScript enabled walk-through test results

JavaScript

As described within the development section, the user interface by default will use JavaScript to reduce the amount of input the user is required to do. It also adds to the user experience, by removing the requirement for some pages to be ‘reloaded’.

Within this subsection, a number of software walk-throughs will be conducted with JavaScript enabled. This will test the additional layer of functionality that is brought by using JavaScript. Table 1.8 outlines the results of each of the tests.

Test Number	Test Data	Reason For Test	Expected Outcome	Actual Outcome	Corrective Action Key	Notes
Clue Input						
11	cryptic clue = nil solution length = nil known characters = nil	To ensure that the JavaScript prevents submission of an invalid form.	The ‘cryptic clue’ and the ‘solution length’ inputs should become red. The ‘known characters’ input fields should not appear.	As expected.	None.	None.
12	cryptic clue = “We hear twins shave” solution length = nil known characters = nil	To ensure that the JavaScript prevents submission of an invalid form.	The ‘cryptic clue’ input should become green, the ‘solution length’ input should become red. The ‘known characters’ input fields should not appear.	As expected.	None.	None.
13	cryptic clue = nil solution length = “4” known characters = nil	To ensure that the JavaScript prevents submission of an invalid form.	The ‘solution length’ input should become green, the ‘cryptic clue’ input should become red. The ‘known characters’ input fields should not appear.	As expected.	None.	None.

14	cryptic clue = “We hear twins shave” solution length = “4” known characters = nil	To ensure that the JavaScript allows the submission of a valid form.	The ‘cryptic clue’ and the ‘solution length’ inputs should become green, and a “Solving Clue” message dialogue appears.	As expected.	None.	None.
15	cryptic clue = “We hear twins shave” solution length = “4” known characters = “p”, “”, “”, “I”	To ensure that the JavaScript allows the submission of a valid form.	The ‘cryptic clue’ and the ‘solution length’ inputs should become green, and a “Solving Clue” message dialogue appears.	As expected.	None.	None.
Solution Output						
16	cryptic clue = “We hear twins shave” solution length = “4” known characters = nil	To ensure that the correct result is returned upon the submission of a valid form with a solvable clue.	A number of solutions should be returned, with the correct solution – “pair” – holding the highest confidence rating.	As expected.	None.	41 solutions were returned.

17	cryptic clue = “We hear twins shave” solution length = “4” known characters = “p”, “,” , “,” , “r”	To ensure that the correct result is returned upon the submission of a valid form with a solvable clue.	A reduced result set in comparison to test #14, however the correct solution – “pair” – should have the highest confidence rating.	As expected.	None.	2 solutions were returned.
18	cryptic clue = “We hear twins shave” solution length = “4” known characters = nil	To ensure that the results are paginated to 10 solutions per page.	41 solutions are expected, and therefore there should be 5 pages of results with no more than 10 solutions per page	As expected.	None.	None.
19	cryptic clue = “We hear twins shave” solution length = “4” known characters = nil	To ensure that the top solution’s trace is opened, and viewable by default.	The top solution’s solution’s trace is viewable by default.	As expected.	None.	None.
20	cryptic clue = “We hear twins shave” solution length = “4” known characters = nil	To ensure that other solution traces can be viewed when clicked upon, and any open solution traces are closed.	Clicking upon a second solution opens the associated trace and closes the original trace.	As expected.	None.	None.

Table 1.8: JavaScript enabled walk-through test results

1.4 Functional Testing

The purpose of the functional testing is to take all the requirements apart from the non-functional requirements and determine whether they have been achieved from the final system.

This section will take all the requirements from the Functional Requirements and Usability Requirements found within the MoSCoW analysis section and will provide evidence of the requirements being fulfilled. If the requirements have not been implemented, an explanation will be delivered as to the reasons why.

1.4.1 Functional Requirements

Retrieve and process cryptic clues and output possible solutions A website and a web service have been implemented. The website allows the input of cryptic clues and clue details from the user which are then passed to the web service for all the solver algorithms to find potential solutions. These potential solutions are then sent back to the website to be displayed for the user.

The screenshot displays a web application interface for solving cryptic clues. It includes a 'Clue' section with the text 'A pain? Have some tea, Rachel', a 'Pattern' section with '??????', and a status message '54 solutions generated in 7.63 seconds'. A light blue box contains a 'Heads up!' message: 'Try supplying known characters to reduce the number of possible solutions.' Below this, a scrollable list of solutions is shown. The first solution, 'earache', is highlighted in blue and includes a 'hidden' tag and a '52%' confidence rating. A 'Solution Trace' box for 'earache' lists two points: '1. Solution hidden in the clue in a forward direction.' and '2. Confidence rating increased as the clue contains indicator word(s) suggesting the solution is of type "hidden".' The second solution, 'travail', is highlighted in green and includes a 'double definition' tag and a '50%' confidence rating.

Clue
A pain? Have some tea, Rachel

Pattern
??????

54 solutions generated in 7.63 seconds

Heads up! Try supplying known characters to reduce the number of possible solutions.

earache **hidden** 52%

Solution Trace:

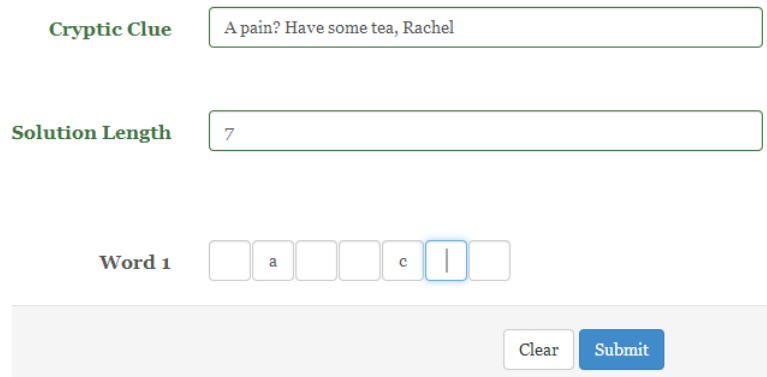
1. Solution hidden in the clue in a forward direction.
2. Confidence rating increased as the clue contains indicator word(s) suggesting the solution is of type "hidden".

travail **double definition** 50%

Figure 1.1: Cryptic clue with results retrieved from web service

Overall: Fully Implemented

Obtain characteristics of the clue's solution (e.g. number of words, word lengths and known letters) and match these against possible solutions A web interface has been implemented with the necessary controls for the user to successfully implement the required and optional information associated with a cryptic clue.



The form consists of three main sections. The first section, labeled 'Cryptic Clue', contains a text input field with the value 'A pain? Have some tea, Rachel'. The second section, labeled 'Solution Length', contains a text input field with the value '7'. The third section, labeled 'Word 1', contains a row of seven input boxes. The second box contains the letter 'a', the fifth box contains the letter 'c', and the sixth box contains a vertical bar '|'. Below these sections is a light gray bar containing two buttons: 'Clear' and 'Submit'.

Figure 1.2: Input fields for obtaining characteristics of a cryptic clue

Overall: Fully Implemented

Provide a web interface for users to interact with the system A web interface has been implemented to allow users to interact with the web service.

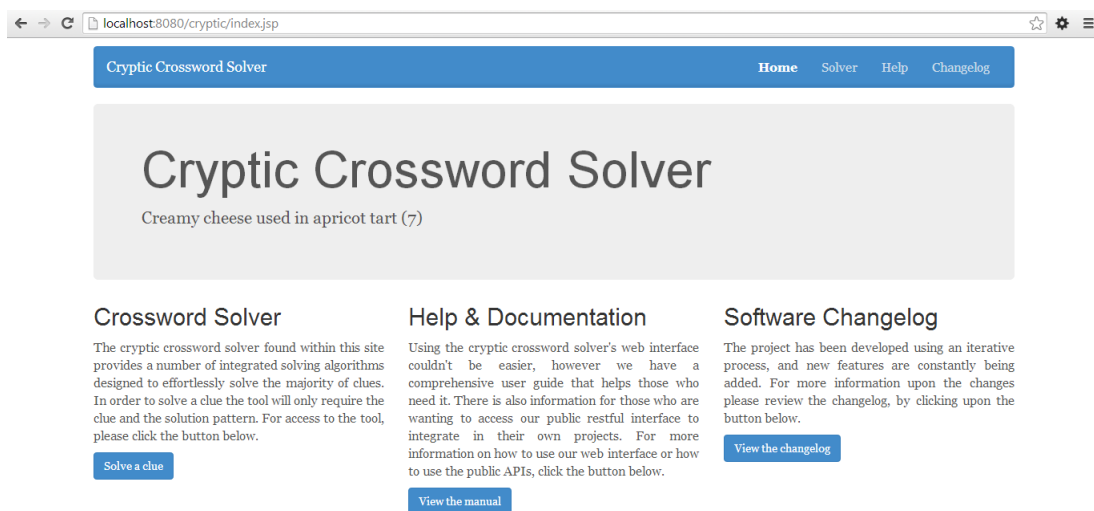


Figure 1.3: Web page in full screen browser window

Overall: Fully Implemented

Provide a mobile-friendly interface for users to interact with the system The web interface that has been implemented has been implemented in such a way that it resizes with the browser window. Therefore, even though it was not made for mobile devices it has a mobile-friendly interface with as limited scrolling as possible.

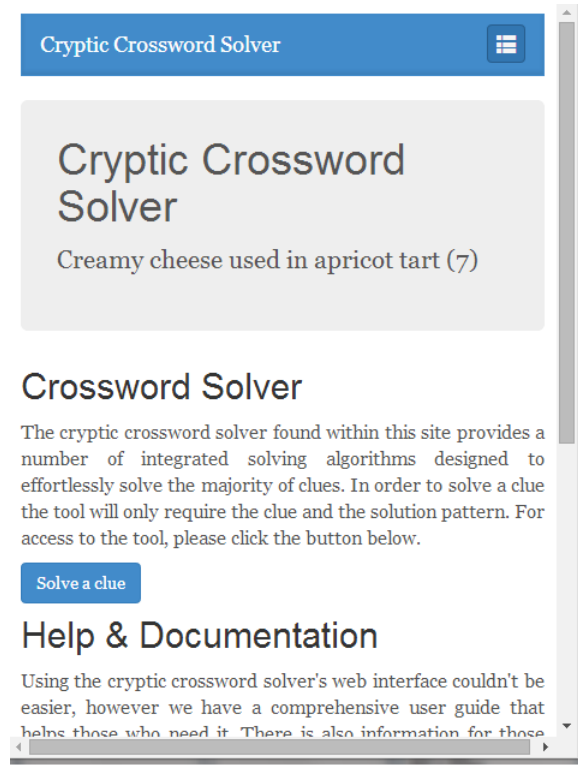


Figure 1.4: Web page in resized browser window

Overall: Partially Implemented

Store clues and their corresponding solutions for future retrieval, including the clue type and solution length The database used within the project is purely for testing purposes. It was decided during the implementation stage that a database to store clues along with their details for the use of the web service before passing the clue to the solvers was not necessary because it may not be entirely beneficial. To store exact clues in the database would mean the user would have to input the exact same clue (wording and punctuation) for this to be beneficial.

Overall: Not Implemented

Take feedback from a user on a solution's accuracy and use this to rank solutions for a given clue As a database has not been implemented for the uses other than testing, this requirement was not possible.

Overall: Not Implemented

Relay the process (solution trace) followed to arrive at a proposed solution to the user Whilst the solvers are finding potential solutions, each solution builds up its own trace. This trace holds important aspects of how the associated solution has been found. These traces are then passed back to the web page to be displayed with the solution for the user.

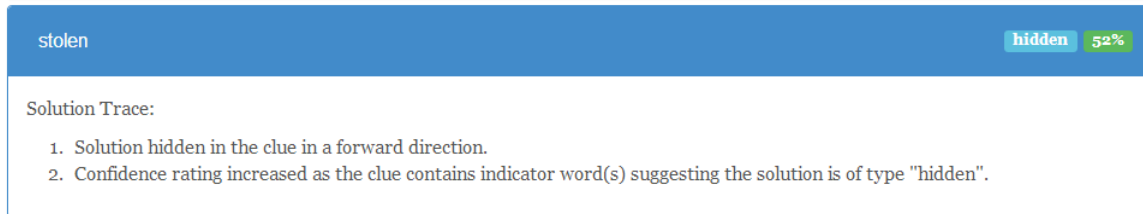


Figure 1.5: Trace for solution 'stolen' with clue; 'Among library books to lend and not returned', with solution; 'stolen'

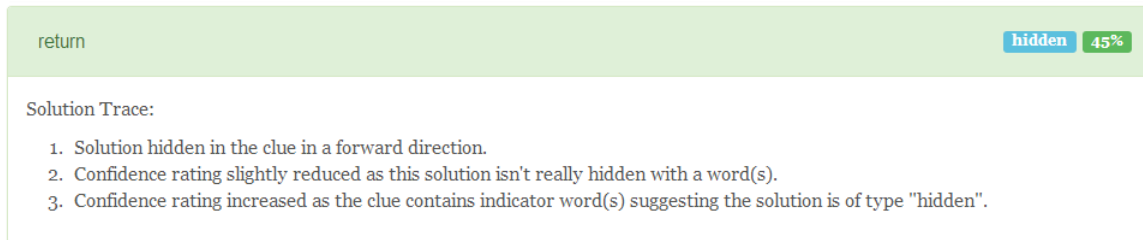


Figure 1.6: Trace for solution 'return' with clue; 'Among library books to lend and not returned', with solution; 'stolen'

Overall: Fully Implemented

Holistically determine a confidence score for each proposed solution, and relay this to the user The confidence score is increased if indicator words or synonyms that match clue words are found. Confidence scores can also be decreased, as shown in Figure 1.6.

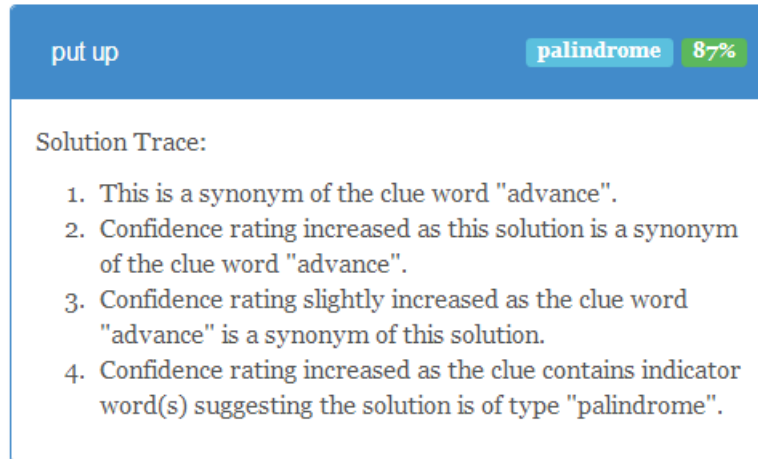


Figure 1.7: A solution returned with a confidence score displayed of 87% for the clue; ‘Advance in either direction’

Overall: Fully Implemented

Document and store the process (solution trace) followed to solve a given clue

As a database has not been implemented for the uses other than testing, this requirement was not possible.

Overall: Not Implemented

Solve clues which take cues from the clue’s orientation or clue number in the crossword

As the chance of this occurring is rare, it was not seen as beneficial to implement.

Overall: Not Implemented

Provide real-time feedback of the processes being followed by the application in an attempt to calculate the correct solution

Although this would have been an ideal feature it was decided that this feature would break the over all underlying architecture of a web service. A web service should be a stateless service, meaning that a request is made and only one response is made.

In order to implement this feature it would require some form of server side session, so that a client can ‘poll’ for results, until an end of results response is received . It was decided that the time spent implementing this feature could be spent else where within the project.

Overall: Not Implemented

Take annotations provided by the user of the known aspects of a clue, such as the definition word(s), fodder or indicator word(s) Although this would have been a beneficial requirement to implement for the project, unfortunately development time was limited. This could have been integrated with the Categoriser to increase confidence scores and possibly bring in the functionality to pass the clue to specific solvers rather than all of them.

Overall: Not Implemented

Store a clue's orientation and clue number in the database As a database has not been implemented for the uses other than testing, this requirement was not possible.

Overall: Not Implemented

Use stored data to generate complete cryptic crosswords As a database has not been implemented for the uses other than testing, this requirement was not possible. It was also stated as a 'Not' requirement within the MoSCoW analysis.

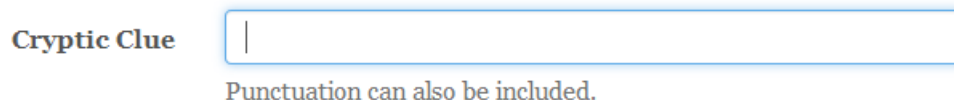
Overall: Not Implemented

Provide a personalised service, including a history of a user's interactions with the system As a database has not been implemented for the uses other than testing, this requirement was not possible. It was also stated as a 'Not' requirement within the MoSCoW analysis.

Overall: Not Implemented

1.4.2 Usability Requirements

Provide a text field for the user to input the cryptic clue to be A text field has been added to the interface to allow the user to enter a cryptic clue.

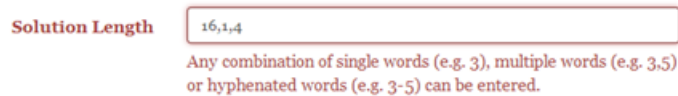


The image shows a user interface element for entering a cryptic clue. On the left, the text "Cryptic Clue" is displayed in a dark blue font. To its right is a rectangular text input field with a light blue border and a vertical cursor line at the beginning. Below the input field, the text "Punctuation can also be included." is written in a smaller, brown font.

Figure 1.8: Text field to enter clue

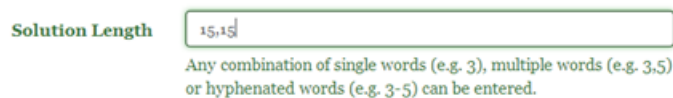
Overall: Fully Implemented

Provide a drop-down box allowing the user to input the number of words in the solution, with an initial upper- limit of 10 words A text field has been provided instead of a drop-down box which allows the user to input the number of words as well as the length of the words. The limit on the amount of words input by the user has not been implemented, however the user can only input words of length up to fifteen. This is because a traditional cryptic crossword has a typical size of 15x15.



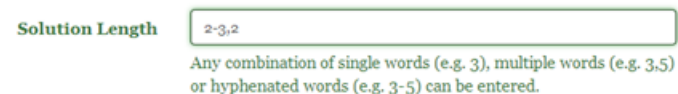
Solution Length 16,1,4
Any combination of single words (e.g. 3), multiple words (e.g. 3,5) or hyphenated words (e.g. 3-5) can be entered.

Figure 1.9: Attempting to enter a length of sixteen - the red colour signifies it is invalid



Solution Length 15,15
Any combination of single words (e.g. 3), multiple words (e.g. 3,5) or hyphenated words (e.g. 3-5) can be entered.

Figure 1.10: Attempting to enter two words of length fifteen - the green colour signifies it is valid



Solution Length 2-3,2
Any combination of single words (e.g. 3), multiple words (e.g. 3,5) or hyphenated words (e.g. 3-5) can be entered.

Figure 1.11: Attempting to enter three words of valid length with a space and a hyphen - the green colour signifies it is valid

Overall: Alternatively Implemented

Dynamically provide individual text boxes for each word of the solution, which allow the length of the words to be specified. Also provide check-boxes between these text- boxes to define whether they are separated by a space or a hyphen See ‘Provide a drop-down box allowing the user to input the number of words in the solution, with an initial upper- limit of 10 words’.

Overall: Alternatively Implemented

Dynamically provide text boxes to represent each character of each word of the solution, allowing the user to input any known characters Text boxes are dynamically created depending on the values input into the ‘Solution Length’ field.

Solution Length

Any combination of single words (e.g. 3), multiple words (e.g. 3,5) or hyphenated words (e.g. 3-5) can be entered.

Word 1 -

Word 2

Figure 1.12: Boxes that have been generated for a solution with the configuration; 2-3,2

Overall: Fully Implemented

Provide a table of results which allow the user the to view the possible solutions
When the results are returned to the user, they are displayed within a table underneath the fields containing the users input.

acrostic	acrostic	52%
Solution Trace: 1. Initial letters taken from clue words, starting with "a", to clue word "constitute". 2. Confidence rating increased as the clue contains indicator word(s) suggesting the solution is of type "acrostic".		
boulders	anagram	52%
troubles	anagram	52%
fanciful	double definition	50%
graceful	double definition	50%
guileful	double definition	50%

Figure 1.13: Results displayed for the clue; ‘A clever rhyme or subtle teaser I constitute’

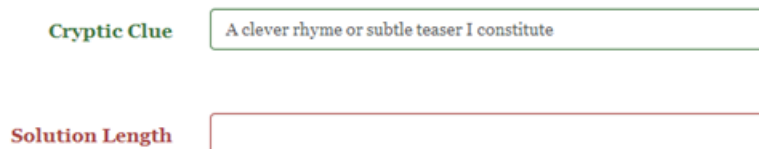
Overall:: Fully Implemented

Alert the user to required fields with red asterisks A design decision was made to change the colour of the label text and the outline of the text box if the user clicks ‘Submit’ without filling in required fields.



A screenshot of a web form with two input fields. The first field is labeled "Cryptic Clue" in red text and is empty. The second field is labeled "Solution Length" in red text and is also empty. Both fields have a thin red border.

Figure 1.14: If the user submits without filling in either required fields



A screenshot of a web form. The "Cryptic Clue" field, labeled in red, contains the text "A clever rhyme or subtle teaser I constitute" and has a green border. The "Solution Length" field, labeled in red, is empty and has a red border.

Figure 1.15: If the user submits without filling in the length of the solution



A screenshot of a web form. The "Cryptic Clue" field, labeled in red, is empty and has a red border. The "Solution Length" field, labeled in red, contains the number "8" and has a green border.

Figure 1.16: If the user submits without filling in the clue field

Overall: Alternatively Implemented

Have a consistent layout avoid unnecessary scrolling All pages within the website have the same colour scheme and fonts with an additional consistency with menu items and bars. The web page has also been implemented to resize to the browser window. This means there is no scrolling for full screen browser windows and limited horizontal scrolling for smaller browser windows as the controls are moved into a linear position.

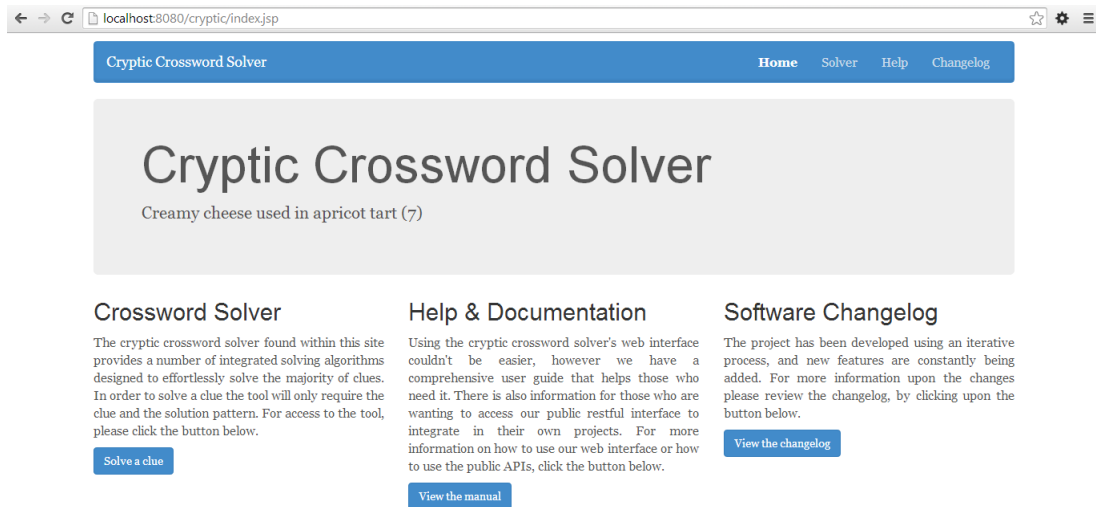


Figure 1.17: Web page in full screen browser window

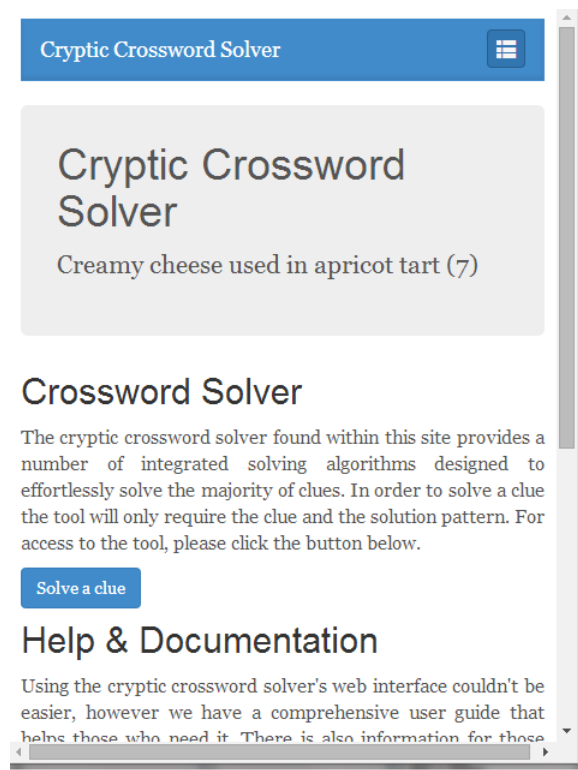


Figure 1.18: Web page in resized browser window

Overall: Fully Implemented

Alert the user to invalid input through validation checks with error messages
 When the user clicks 'Submit' without completing the required fields the styling of the label

and text fields change colour to red and the page does not proceed to pass invalid data to the back end code.

See ‘Alert the user to required fields with red asterisks’ for evidence’.

Overall: Alternatively Implemented

Provide a confidence rating in the table of possible solutions The confidence rating calculated whilst retrieving possible solutions is displayed alongside the solution.

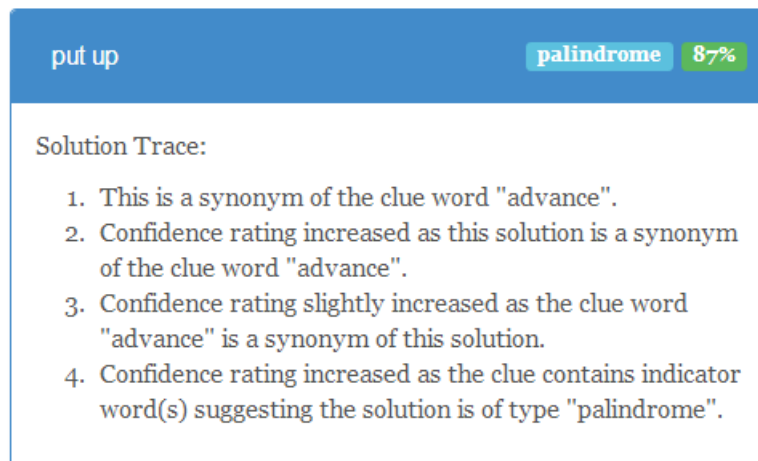


Figure 1.19: Confidence ratings displayed with the solutions returned

Overall: Fully Implemented

Allow the user to select a proposed solution in the corresponding table and mark this as correct **Overall:** Not Implemented

Display help buttons to indicate to the user the purpose and use of each control Instead of implementing buttons to display help for each of the controls on the web page, a user manual has been created which can be accessed through the menu bar. This explains the web page functionality as well as some additional information on the web service.

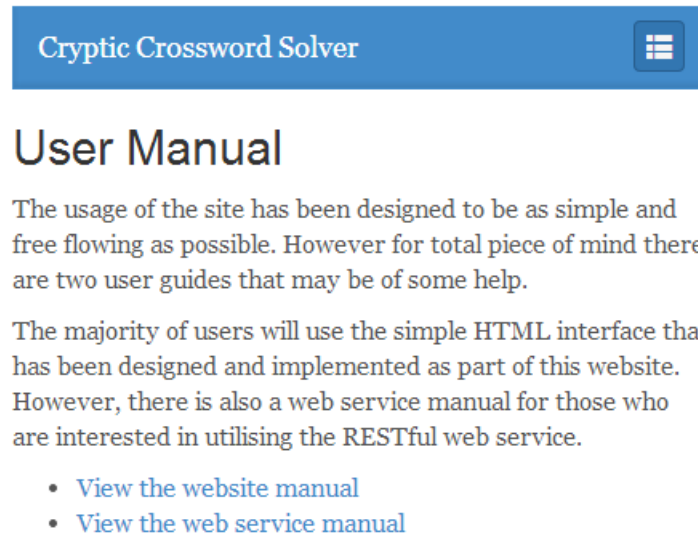


Figure 1.20: Introduction to the user manual

Overall: Alternatively Implemented

Provide a button to submit the user input to the application for processing A submit button has been provided for the user to click when they have input the clue itself and the clues length.

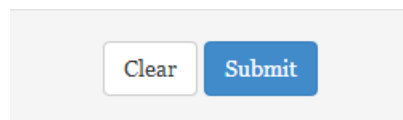


Figure 1.21: Submit button for user to click

Overall: Fully Implemented

Provide a group of radio buttons for the user to select the clue's orientation in its containing crossword There has been no functionality implemented to use the clue's orientation within the solver algorithms, therefore it was unnecessary to implement radio buttons for the user to select the orientation of the clue.

Overall: Not Implemented

Provide a text box to input the clue's number within its containing crossword There has been no functionality implemented to use the clue's grid number within the solver algorithms, therefore it was unnecessary to implement a text box for the user to input the grid number of the clue.

Overall: Not Implemented

Provide mechanisms to accommodate for users with difficulties, such as colour-blindness or poor eyesight There has been no specific functionality implemented to accommodate for users with difficulties. However, the user can disable CSS on their browser to view the web page with no styling.

Overall: Not Implemented

Glossary of Terms

The following section contains a glossary with the meanings of all names, acronyms, and abbreviations used by the stakeholders.

Term/Acronym	Definition
The Guardian	A national UK newspaper that prints daily cryptic crosswords
Android	A mobile phone software platform by Google Inc.
BlackBerry	A mobile phone hardware and software platform developed by BlackBerry Limited
iOS	A mobile phone software platform developed by Apple Inc.
iPhone	A smart phone developed by Apple Inc.
iPod	A portable digital music player developed by Apple Inc.
iPad	A tablet developed by Apple Inc.
NLP	Natural Language Processing
SRS	Software Requirements Specification
App	Shorthand for application

Bibliography

IEEE (2008). IEEE Standard for Software Reviews and Audits. *IEEE STD 1028-2008*, pages 1–52.