DECO3801 Test Plan Document

$\ensuremath{\mathsf{THEM}}$ - Typed HTML5 Evaluation Machine

Carl Hattenfels, Scott Heiner, Shen Yong Lau, Robert Meyer, Brendan Miller, David Uebergang

Contents

1	Fun	actional Test Plan	2
	1.1	Testing Strategy	2
	1.2	Implications of Functional Testing	2
	1.3	Test Case Transcript	3
2	Use	er Experience Goals	8
3	Use	er Testing Plan	11
	3.1	User Testing Strategy	11
		Implications of User Testing	
	3.3	User Test Results	14
	3.4	User Test Document	17
4	Apj	pendix A - Python Test Code	19
	4.1	Syntax Tests	19
	4.2	Page Structure Tests	23
	43	JSON-RPC Server Tests	30

1 Functional Test Plan

1.1 Testing Strategy

There are three major testable components of the web application: the front-end website, back-end parser and database. While it was easy to write Python test cases for the back-end parser, it was more difficult to test the front-end website and database with a suite of computer-run tests. Instead, a series of scenarios were drafted that would ensure that the web application was running correctly and as expected. Clearly, all of these scenario tests can be "implemented" as they are merely actions performed by us. This means that a test fails when some functionality is not yet implemented, or when fixing one error creates another error.

The test cases that are being run on the parser can be found in Appendix A within this document. This gives an indication of the tests that are currently implemented in the system. More tests are being added periodically, as different types of HTML5 errors are added to the parser. Each HTML5 error will have its own associated test. Other parser related tests are also contained in Appendix A, including the JSON-RPC server tests. These test for concurrency (can the parser handle 5 concurrent requests?) as well as correctness of JSON-RPC input and output.

1.2 Implications of Functional Testing

The functional testing highlighted some issues with all aspects of the application during initial development. There were several sections of the functionality which were unimplemented initially that are now implemented. The interesting implications of the testing showed that the code was implemented correctly. An appropriate fix for the case of empty files (Website and Server Tests, 11) was implemented post-testing. Many of the tests relating to errors were unimplemented initially, but all are present in this final solution. The test suite that is now being used for this program is complete and thorough.

1.3 Test Case Transcript

Python Parser Tests

Test Number	Test Description	Inputs	Expected Output / Resulting Action	Pass / Fail + How to Fix
1	Testing that a specific error is being reported correctly, given a particular fragment of HTML as the input. Since not all of the error checks have been or can be defined in advance, the implementation of this test case is reactionary and will be continually updated to include new error checks as they are added. The existing tests will have to be run each time a new error check is implemented to ensure that all existing functionality still works as intended.	A tailored fragment of HTML that should cause a specific error to be reported. eg. '' <head> <head> </head> =>Testing for the duplicate set of head tags.</head>	Confirmation that the expected error and associated error code is being returned for the given HTML fragment, in the expected character position of the input fragment.	Pass
2	Test that the JSON-RPC server is running and can respond to a remote function call.	A single client making a function call to the server.	The function call should be processed without an error being raised, implying the server is currently running.	Pass
3	Testing that the JSON-RPC server is able to handle up to a maximum of 5 concurrent remote function calls.	Five concurrent function calls are made to the server.	The test case records the time that each response is received by each of the client instances. The function being called has an internal sleep delay of 2 seconds, so the recorded time for each client should be slightly over 2 seconds, implying all 5 calls were made and processed at the same time.	Pass
4	Testing that a 6th concurrent connection (1 connection over the maximum of 5 concurrent connections) to the JSON-RPC server results in a delayed response.	Six concurrent function calls are made to the server.	As above, the time the response is received is compared to the time the call was made. The first 5 connections should behave as above, receiving a response in just over 2 seconds. The 6th call will receive a response after 4 seconds, 2 seconds after the server is able to respond after the first 5 connections have been responded to.	Pass

5	Testing the parser's response to a case where input of an empty string is supplied.	An empty string.	The parser should respond with a general error stating that the input is empty, preventing other general errors such a missing closing HTML tags or page structure sections (head, body, footer).	Pass
6	Testing the parser's response when the input string doesn't contain any valid HTML.	A garbage string which doesn't contain any HTML tags or tag like elements eg. ¡blah;	The parser should respond with a general error stating that the input doesn't contain any valid HTML.	Pass
7	Testing that a correctly formed JSON-RPC 2.0 request is handled by the server, which should respond with the correct response.	A JSON-RPC 2.0 request containing a small HTML code fragment to be parsed.	A JSON-RPC 2.0 response containing an array of errors related to the given request. The response should also contain the same ID value as the one passed to it with the request.	Pass
8	Testing that a malformed JSON-RPC 2.0 request containing incorrect parameters for a particular function call causes the server to return an error.	A JSON-RPC 2.0 request containing an invalid parameters array for the function call parse_html.	A JSON-RPC 2.0 error response with a message of "Invalid parameters".	Pass
9	Testing that a JSON-RPC 2.0 request calling a function that isn't registered on the server causes the server to send an error response.	A JSON-RPC 2.0 request containing a function name that hasn't been registered on the server.	A JSON-RPC 2.0 error response with a message of "".	Pass
10	Testing the parser response when a tag with a URL attribute is supplied with an valid relative file path.	Html fragment: '' ". File list: [''image.jpg", ''directory/", ''directory/current.html", ''directory/directory2/ image2.jpg"]. Current file: ''directory/current.html"	The parser response should NOT contain an error indicating an invalid file path.	Pass
11	Testing the parser response when a tag with a URL attribute is supplied with an non-existent relative file path.	Html fragment: '' ". File list: [''directory/", ''directory/current.html"]. Current file: ''directory/current.html"	The parser response should contain two errors indicating invalid file paths, associated with the src attributes of the img tags.	Pass

12	Testing the parser response when	Html fragment: '' <img< th=""><th>The parser response should con-</th><th>Pass</th></img<>	The parser response should con-	Pass
	a tag with a URL attribute is sup-	<pre>src=/image.jpg><img< pre=""></img<></pre>	tain two errors indicating invalid	
	plied with an non-existent files in	<pre>src=directory2/ image2.jpg>".</pre>	file paths, associated with the src	
	existing relative filepaths.	File list: [''directory/",	attributes of the img tags.	
		''directory/current.html",		
		''directory/directory2/"].		
		Current file:		
		''directory/current.html"		
13	Testing the parser response when	Html fragment: '' <img< td=""><td>The parser response should NOT</td><td>Pass</td></img<>	The parser response should NOT	Pass
	a tag with a URL attribute is sup-	<pre>src=/image.jpg><img< pre=""></img<></pre>	contain an error indicating an in-	
	plied with an valid absolute file	<pre>src=/directory/directory2/</pre>	valid file path.	
	path.	image2.jpg>". File		
		list: [''directory/",		
		''/image.jpg",		
		''directory/current.html",		
		''directory/directory2/",		
		''directory/directory2/		
		image2.jpg"]. Current file:		
		''directory/current.html".		
14	Testing the parser response when	Html fragment: '' <img< td=""><td>The parser response should con-</td><td>Pass</td></img<>	The parser response should con-	Pass
	a tag with a URL attribute is sup-	<pre>src=/directory3/ image.jpg>".</pre>	tain an error indicating invalid file	
	plied with an non-existent absolute	File list: [''directory/",	paths, associated with the src at-	
	file path.	''directory/current.html"].	tributes of the img tag.	
		Current file:		
		''directory/current.html"		
15	Testing the parser response when	Html fragment: '' <img< td=""><td>The parser response should con-</td><td>Pass</td></img<>	The parser response should con-	Pass
	a tag with a URL attribute is sup-	<pre>src=/directory/ image2.jpg>".</pre>	tain one errors indicating invalid	
	plied with non-existent files in ex-	File list: [''directory/",	file paths, associated with the src	
	isting relative file paths.	''directory/current.html"].	attributes of the img tags.	
		Current file:		
		"'directory/current.html".		

Website and Server Tests

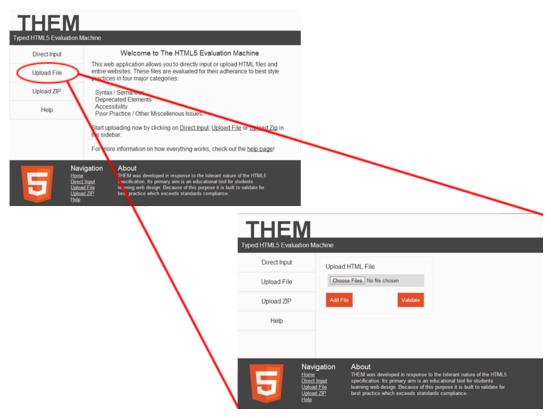
Test Number Test Description		Inputs	Expected Output / Resulting Action	Pass / Fail + How to Fix
1	View Home page	Go to URL, or click link from any page	The home page is displayed.	Pass
2	View Help page	Click link from any page	The user is sent to the help page.	Pass
3	View Direct Input page	Click link from any page	The user is sent to the direct input page.	Pass
4	Validate direct input	The user types their input into the text field on the Direct Input page and clicks Validate.	The input text is saved in a new set with a single file in it. The user is redirected to the Show File page.	Pass
5	View Upload File(s) page	Click link from any page	The user is sent to the Upload File page.	Pass
6	Upload single HTML file	The user selects a file and then clicks Validate.	The file is saved in a new set with a single file in it. The user is redirected to the Show File page.	Pass
7	Upload multiple HTML files individually	The user clicks the Add File button the required number of times, then selects a file for each field. They then click Validate.	Files are saved in a new set, user is redirected to uploaded set page	Pass
8	Upload multiple HTML files together from one dialogue	The user selects multiple files in the dialogue box, then clicks Validate.	Files are saved in a new set, user is redirected to uploaded set page	Pass
9	Upload multiple HTML files, some individually and some from one dialogue box	The user performs a combination of multiple Add Files and selecting multiple files in the dialogue boxes. They then click validate.	Files are saved in a new set, user is redirected to uploaded set page	Pass
10	Upload non-HTML file	The user attempts to upload a file which is not HTML.	The user is redirected to the same page and shown a information box informing them that the file chosen is not a HTML file.	Pass
11	No file selected on upload	The user attempts to upload a file when no file is selected.	Redirect to upload file page with a helpful error message	Pass
12	View Upload Zip page	Click link from any page	The user is sent to the Upload Zip page.	Pass
13	Upload zip file	Zip file selected on previous page, user clicked validate	Zip archive is unpacked, files are saved in a new set, user is redi- rected to uploaded set page	Pass
14	View Uploaded Set page	User either uploads multiple files, or uploads a zip archive	The user is shown the list of files uploaded in this set, with corresponding error bars, except in the case of a single file in a set or zip, in which case the user is redirected directly to the show file page.	Pass

15	View Uploaded File page	User either selects a file on the Up-	The user is shown their uploaded	Pass
		loaded Set page, or uploads a single	file, with corresponding error bar,	
		file, or validates by direct input	general error information, and up-	
			loaded text with error highlighting.	
16	Remove file after certain period of	A file should be removed from the	The files are removed from the	Pass
	being untouched in the server	server after a period of inactivity.	database after a time. (3 hours)	

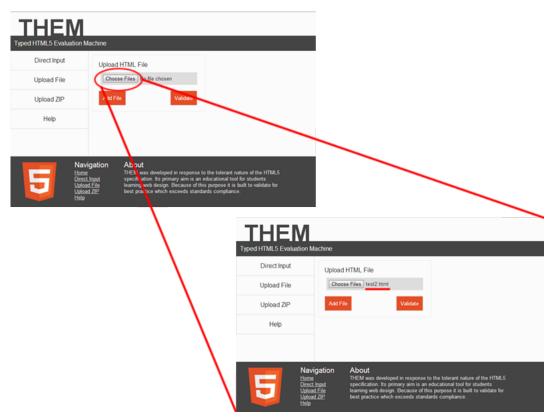
2 User Experience Goals

A clear user experience was in mind while developing this website. Through its ease of use and minimal effort on the part of the user, the application aims to create a very surgical, ambient, and passive experience. The tool should give users almost immediate insight into the issues with their HTML and websites. This is where the user's experience with the tool ends, for this session. The user now can go and fix their file externally, return to the program and almost instantly receive another assessment of their code's validity. There is no aim to get the user invested in the system, and be held on the website for long periods at a time. However, creating a reliable and worthwhile experience is a large focus of this project, brief as this experience with the user is. The user should not be frustrated by the errors the program reveals, with the focus instead on helping the user learn and develop better web practices. It is meant to be a program that a user just "touches", that is, they upload their file they want to check, and then go back and fix it, and then come back to this to validate again, in a cyclic process.

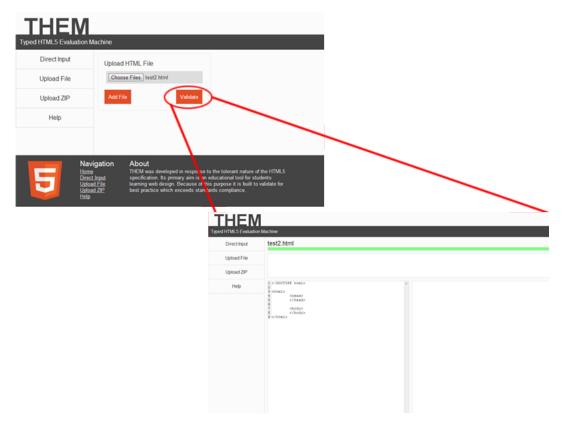
Our priorities are on quick and easy use, which is why all web pages are instantly accessible from all other web pages and users only need a few clicks to navigate. The website has been designed to require as few clicks as possible to access the primary functionality of the system. For example, the following images represent an average user's attempt to verify a file, after brief knowledge of the system's workings.



The first click takes the user to the webpage.



The second click chooses a file to verify. Other clicks may be employed here as the user navigates their file system to find the file they want to upload.



The third click validates the selected file. In this way, the user quickly and easily travels from the opening page to viewing their validated HTML. All throughout the web application, the aim has been to create similar experiences where few clicks are required by the user, and they reach their end goal in minimal time.

Users benefit greatly from this experience. The surgical nature means that they develop a relationship with the website where it is used as an intermediate and reliable tool, much like their text editor or their browser. It is easily inserted as part of their workflow. It is hoped that by using this application, users will experience a great rush of joy when their program finally passes, and that long error bar on the web page turns completely green. Validation, after all, is its own reward.

3 User Testing Plan

3.1 User Testing Strategy

The web application will eventually be utilised by two user groups - students of DECO1400, and students of DECO7140. As such, four major user testing groups were pinpointed:

- Undergraduate students who have already completed DECO1400
- Undergraduate students who have not completed DECO1400 but have worked with computers
- Masters students who have already completed DECO7140; and
- Masters students who have not already completed DECO7140.

However, poor initial consideration due to this highly targeted user base caused us to primarily focus on students who had not done DECO1400, as these users represented students "new" to DECO1400. Focusing also on past DECO1400 students would have allowed us to understand the needs of users who had previously completed the course, and could determine whether the tool would have been worthwhile to them. Poor communication on the part led to us getting very few in this category. Ultimately, information was gathered from nine users - four were undergraduates who had not done DECO1400, one was an undergraduate who had done DECO1400 and four were masters students who had done DECO7140.

Six key scenarios were formulated for the users to undertake. Each scenario was performed on the live prototype at http://underwaterfall.com. Storyboards for each of them can be found in the separately submitted Appendix B.

• Getting started by reading the help page (First Encounter Scenario)

Actor: New User

Goal: To understand how the website works, and understand the

feedback it provides.

Necessity of Scenario: This scenario is required for first time users to understand

how to use and interact with the program.

Preconditions: User has not previously visited the webpage.

• Validating HTML via Direct Input Actor: User

Goal: To check the validity of a piece of copied or typed HTML. Necessity of Scenario: This scenario represents one of the key ways users can get

insight into how to program using HTML5.

Preconditions: User has a clear understanding of the validation the web-

site provides from the help page.

• Validating HTML via uploading a file

Actor: User

Goal: To check the validity of a HTML file.

Necessity of Scenario: This scenario represents one of the key ways users can get

insight into how to program using HTML5.

Preconditions: User has a clear understanding of the validation the web-

site provides from the help page.

• Validating websites or multiple HTML files via uploading a zip

Actor: User

Goal: To check the validity of a zip file of either website files or

HTML.

Necessity of Scenario: This scenario represents one of the key ways users can get

insight into how to program using HTML5.

Preconditions: User has a clear understanding of the validation the web-

site provides from the help page.

• Fixing a file based on the error suggestions, resubmitting and getting a valid file

Actor: User

Goal: To check the validity of a piece of copied or typed HTML. Necessity of Scenario: This scenario is the primary point of the application - users

learning to correct their HTML5 pages.

Preconditions: User has already uploaded a file and determined the errors

relating to their webpage.

• Attempting to upload a non-HTML file (Fringe Case Scenario)

Actor: User

Goal: To check the validity of a non-HTML file.

Necessity of Scenario: Users are fallible and can upload incorrect files. They may

also believe the website is capable of evaluating other types

of files, like Javascript or CSS.

Preconditions: N/A

There was a focus in testing of two metrics: time taken to complete each scenario, and, in keeping with the surgical user experience, number of clicks required to complete each scenario. As there was a need to create an enjoyable environment for the users, any particular emotions and reactions of the users as they undertook the scenarios was also noted. the primary strategy for user testing was as follows:

- 1. Prepare / lay out materials for the participant so that everything is ready.
- 2. Introduce ourselves to the participant and give them a high-level idea of what they will be doing in their tasks today.
- 3. Ask participant to fill in and sign consent form. The test conductor will fill in their parts too.
- 4. Give the participant more detailed instructions about the task they are to do (i.e. access the website, upload file and validate). Ask them to think out loud or to make comments as they work. See if there are any questions from the participants before starting, and answer these where appropriate.
- 5. When participant is ready, ask the participant to start on the task. Start the timer. Be prepared to count the number of clicks they required to complete the task. Take hand notes as the participant works, according to the arrangements you have worked out amongst the non-participant group. If the participant goes a bit quiet, ask, "what are you thinking now?" or "what are you working on now?"
- 6. When they complete the first scenario, move them onto the next one, and so on.
- 7. After completing all six scenarios, ask the participant to fill in the questionnaire. Clarify as necessary.

- 8. When the participant has finished filling in the questionnaire, check over the responses to make sure that all parts have been filled out, and that the answers are legible.
- 9. Tell the participant that the session is at an end. Thank the participant for their time.

The results of testing are below, after the "Implications of User Testing" section, as well as the Questionnaire used for testing.

3.2 Implications of User Testing

In general, users had no trouble navigating the system. The average rating for how easy the system was to navigate was 4.78 / 5 on a Likert scale. The user experience goal of the small number of clicks required for each operation was met. No user (from those who had click data registered) needed to click more than ten times. However, the system requires improvement in several key areas. On recommendation from the users, here is a key list of changes that were implemented in the completion of this project:

- Invite the users to click on the error tags. Users often did not realise that they could click on the highlighted text to find out more about the error. Users are now provided with information on the help page about this, as well as told on the file page to "Click on highlighted text for more information."
- Errors with non-HTML files. As part of the user testing, users were asked to upload a non-HTML file, a common action which could be performed by a user. Many were surprised that the file was actually parsed. The mimetype of the uploaded file is now checked, and the parser is not passed non-HTML code, except in the case of code which is plain-text and could possibly be HTML code that lacks structure. The user is presented with an error on the Upload File screen when they attempt to upload a non-HTML file. The good news is that for the most part, the website did not break, except when a user attempted to upload a Powerpoint file. This is prevented by the previously mentioned fix.
- Add multiple files via Add File button. Users noted that an attempt to add a second file via the Add File button after selecting the first file caused their previously selected file to be forgotten. This needs to be fixed. One user had trouble due to the similarity of the Validate and Add File buttons, so these now differ in colour to highlight the difference functions.
- Blank file uploading. Users choosing no file in a file field of the form, along with some file fields being filled, were parsed as if they were files with no name. As such, they were sent to a set page showing bars with blank file names and error bars. These files are no longer parsed, and if a user clicks Validate with no file selected, they are redirected back to the upload file page and asked to upload a file.

There was no plan to redefine any test plans, but for future testing, working closer together to complete it would be a priority. The differences between the methods of testing used by members of the group was obvious, and it lead to less data than expected. All in all, the web application was well received, with many users who had experience with HTML5 stating they would find the tool beneficial to their studies. It received a rating of 4.6 / 5 on the Likert scale for "How likely would you use this tool to assist you in your study?" among those students who had previous experience with HTML5. The Typed HTML5 Evaluation tool is a fantastic tool for users to evaluate their HTML5.

3.3 User Test Results

Metric Results

	Get Informed	Direct Input	Upload a File	Fixing a File	Upload Zip	Upload Non-HTML File
Tester 1 - DECO1400						
Time taken (secs)		49.03	39.9	32.7	13.3	6.03
Clicks						
Reaction		Confused about the highlighting word, unsure it is clickable	Feel confused when add file buton cancel previous upload entry	Learn the error quickly, getting used to the system presentation	Feeling comfortable	Feeling surprised when the result is as expected
Tester 2 - DECO7140						
Time taken (secs)		56	22.3	27	11.2	6.5
Clicks						
Reaction		Unsure about what the error bar representing	Frustrated as keep mispressing add file instead of validate due to same colour button	Learn the error quickly	Feeling comfortable	Feeling surprised when the result is as expected
Tester 3 - DECO7140						
Time taken (secs)		43.5	19.2	23.3	9.9	6.9
Clicks						
Reaction		Feeling unimpressed as the error is not val- idated correctly	Feeling satisfied with the simple way to up- load file	Learn the error quickly and fixed it	Feeling comfortable	Feeling surprised when the result is as expected
Tester 4 - DECO7140						
Time taken (secs)		51.2	23.1	31.78	11.67	8.3
Clicks						
Reaction		Feeling that the presentation of errors is good	Frustrating when try- ing to upload multi- ple file, the add file button cancel previ- ous entry	Learn the error quickly, feeling good	Feeling comfortable	Feeling surprised when the result is as expected
Tester 5 - DECO7140						
Time taken (secs)		48.12	17	20.1	10.8	5.87
Clicks						
Reaction		Unsure about the highlighting words are clickable	Feeling good as it is easy to upload single file	Learn the error quickly	Feeling comfortable	Feeling surprised when the result is as expected

Tester 6 - Undergraduate Time taken (secs)	7	11	24.8	10	8	N/A
	1		6	5		5
		Didn't realise you could click on errors	Multiple files added but no files given - still shows the bars		4	Powerpoint file uploaded "max_allowed_packets error given
Tester 7 - Undergraduate						
Time taken (secs)	2	45	29	20	18	20
Clicks:	1	5	5	3	5	5
Reaction						Note, inf file still was parsed.
Tester 8 - Undergraduate						
Time taken (secs)	2	41	14.8	63	22	13
Clicks:	1	3	6	8	7	5
Reaction				Backtracked to copy		
				files, didn't get to put		
				in entire tag		
Tester 9 - Undergraduate						
Time taken (secs)	2	60	28	211	9	15
Clicks:	1	5	6	9	4	3
Reaction			Clicked add file accidentally	Not intuitive to click highlighted text		

Questionnaire

Tester	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10
Tester 1	Yes (DECO1400, Programming)	Yes (Java, Javascript, PHP, Python)	4	5	4	3	5	No	Yes	If you upload a non-zip file to the upload zip no error message is shown. Selecting file then clicking add file clears previous additions. Upload invalid file to upload file (specifically provided zip) shows the file. Adding '' html ." (note full stop) causes errors to appear (raw output). Clicking add file then only uploading 1 file takes you to collections screen instead of the single file screen. Leaving uploads blank causes blank error bars to appear.
Tester 2	Yes (DECO7140)	Yes (Python, Java, Actionscript)	3	2	4	5	4	No	Yes	
Tester 3	Yes (DECO7140)	No	4	4	4	4	4	Yes	Yes	
Tester 4	Yes (B InfTech)	Yes (Java, PHP, C‡, HTML5, CSS, Javascript, MYSQL, etc)	5	4	4	3	5	No	No	Confused about the "upload" buttons. Color theme is nice. Single file ->upload file, multiple files ->zip it and upload. Highlight the code in the correction section will be more user friendly (Upload File). General instruction on the UI can be added. Maybe do more market research about existing validation tools. Line mumber is good to be placed in direct input.
Tester 5	Yes (DECO7140)	Yes (Actionscript, Python)	3	5	4	5	5	No	No	Simple & clean layout, I would like if I could copy the text and paste the text again to modify it.
Tester 6	No	Yes (Python)	2	4	4	2	5	Yes	Yes	
Tester 7	No	Yes (Python, Java, Matlab)	2	4	4	3	5	Yes	No	No way to gauge the effects of the error based on the error message. Didn't initially realise you can click on highlighted text to see error notes nor which colours associated to errors (thought colour was gauging error intensity.)
Tester 8	No	Yes (Python, Matlab)	3	5	4	4	5	Yes	No	All good :)
Tester 9	No (though did make website in primary school)	Yes (Python, Matlab)	1	4	4	5	5	Yes	Yes	It was fun.

3.4 User Test Document

	DECO3801 User Testing Document	
Name:		
Program or Degree:		

Please wait for your instructions from supervisors before completing these scenarios. Time will be taken between each scenario to write down key information.

Get Informed

Navigate to the help page. Since this is your first time using the software, read up on what the error bars mean here.

Direct Input

Copy some basic HTML text into the Direct Input page, and validate it for errors. This can be from a file you have locally, or you can use the following provided code:

```
<html>
<head></head>
<body></body>
</html>
```

Upload File

Upload a file to the Upload File page, and validate it for errors. If you do not have a file of your own, we can provide you with one.

Fixing a File

Based on the error messages provided, fix your uploaded file and resubmit it.

Upload Zip

Upload a zipped website to the Upload Zip page, and validate it for errors. If you do not have a website of your own, we can provide you with one.

Upload Non-HTML File

Try uploading any file you like. Does the application behave well?

Questionnaire

1.	Have you taken DECO1400/DECO7140? If not, do you have any past learning experience in web design (such as HTML4, Javascript, etc). ☐ Yes ☐ No Past learning experience:
2.	Do you have any programming background, if so what languages you have been using? ☐ Yes ☐ No Past learning experience:
3.	How likely would you use this tool to assist you in your study?
	Very likely $\begin{vmatrix} & & & \\ 5 & 4 & 3 & 2 & 1 \end{vmatrix}$ Very unlikely
4.	How visually appealing is the website to you?
	Very appealing $\begin{vmatrix} & & & \\ 5 & 4 & 3 & 2 & 1 \end{vmatrix}$ Not very appealing
5.	How well did this tool meet your expectations?
	Met my expectations very well $\begin{vmatrix} & & \\ 5 & 4 & 3 & 2 \end{vmatrix}$ Did not meet my expectations
6.	Do you feel comfortable with the presentation of the error message(s)?
	Very comfortable $\begin{vmatrix} & & & \\ 5 & 4 & 3 & 2 & 1 \end{vmatrix}$ Not very comfortable
7.	How easy did you find the tool to navigate?
	$\begin{array}{c c c c} \text{Very easy} & & & \text{Very difficult} \\ \hline & 5 & 4 & 3 & 2 & 1 \end{array}$
8.	Is this your first time using HTML5 in developing a website? \Box Yes \Box No
9.	Would you prefer the option to select multiple files at once (in the explorer window) when uploading? \square Yes \square No
10.	If you have any general feedback/suggestions, feel free to use the space below to tell us.

4 Appendix A - Python Test Code

4.1 Syntax Tests

```
1 | from __future__ import absolute_import, division, unicode_literals
3
   #from . import support
   import unittest, html5lib
4
   from html5lib import treebuilders
5
6
7
   class TestSyntax(unittest.TestCase):
8
       Provides a number of test cases to test the syntax used
9
10
       in the document.
       0.00
11
12
       def setUp(self):
13
           self.parser = html5lib.HTMLParser(tree=treebuilders.
14
               getTreeBuilder("etree"))
15
16
       def test_malformed_tag_name(self):
17
           Test that the tag name isn't an invalid symbol.
18
19
20
           Input:
           A HTML fragment containing a tag with an invalid tag name.
21
22
           Expected Results:
23
           An error should be thrown reporting an invalid tag name.
24
25
26
27
           inputFragmentEmptyName = "<html>< ></body>"
           inputFragmentQuestionMark = "<html><?></body>"
28
           inputFragmentRightBracket = "<html><></html>"
29
30
           self.parser.parse(inputFragmentEmptyName)
31
32
           self.assertIn(((6, 6), u'expected-tag-name', {u'data': u''}),
33
                self.parser.errors, "Failed to report invalid tag name. Get "
34
                   )
35
           self.parser.reset()
36
           self.parser.parse(inputFragmentQuestionMark)
37
38
39
           self.assertIn(((6, 6), u'expected-tag-name-but-got-question-mark')
               , {}),
               self.parser.errors, "Failed to report valid tag name. Got
40
                   question mark instead.")
41
           self.parser.reset()
42
           self.parser.parse(inputFragmentRightBracket)
43
44
           self.assertIn(((6, 7), u'expected-tag-name-but-got-right-bracket'
45
               , {}),
               self.parser.errors, "Failed to report valid tag name. Got
46
                   question mark instead.")
```

```
47
       def test_self_closing_end_tag(self):
48
49
           Test that a closing tag with a misplaced forwardslash
50
           raises an error.
51
52
53
           Input:
           A HTML fragment containing a closing tag with a misplaced
54
               forwardslash.
55
           Expected Results:
56
57
           An error should be thrown reporting an invalid tag name.
58
59
           inputFragment = "<html><a></a /></html>"
60
61
           self.parser.parse(inputFragment)
62
63
           self.assertIn(((9, 14), u'self-closing-flag-on-end-tag', {}),
64
65
                self.parser.errors, "Failed to report misplaced forwardslash
                   in closing tag.")
66
67
       def test_invalid_self_closing_tag(self):
68
69
           Test that the use of a self closing tag for a tag
           which isn't considered a self closing tag returns
70
71
           an error.
72
73
           Input:
           A HTML fragment containing a start tag with a trailing
74
               forwardslash
           (self-closing) for a tag type which isn't a self closing tag.
75
76
77
           Expected Results:
           An error should be thrown reporting the given tag type isn't a
78
               self-closing
79
           tag.
80
81
82
           inputFragment = "<html><a /></html>"
83
           self.parser.parse(inputFragment)
84
85
           self.assertIn(((6, 10), u'non-void-element-with-trailing-solidus')
86
               , {u'name': u'a'}),
87
                self.parser.errors, "Failed to report invalid self-closing
                   tag.")
88
89
       def test_attributes_in_end_tag(self):
           0.00
90
91
           Test that attributes occuring in a closing tag are
92
           reported as an error.
93
94
           Input:
           A HTML fragment containing a closing tag which contains
95
96
           at least one attribute.
97
```

```
98
            Expected Results:
            An error should be thrown reporting that the closing tag shouldn'
99
                t contain
            attributes.
100
101
102
            inputFragment = '<html><a></a src="blah"></html>'
103
104
105
            self.parser.parse(inputFragment)
106
            self.assertIn(((9, 23), u'attributes-in-end-tag', {}),
107
                 self.parser.errors, "Failed to report attributes in closing
108
                    tag.")
109
        def test_duplicate_h1_tags(self):
110
111
            Test that any duplicate h1 tags are reported as errors.
112
113
114
            Input:
115
            A HTML fragment containing more than one set of h1 tags.
116
117
            Expected Results:
118
            An error should be thrown reporting that duplicate h1 tags were
                found in the document.
119
120
            inputFragment = """
121
122
   <!DOCTYPE html>
   <html>
123
124
   <head>
   </head>
125
126 | <body >
127 <h1></h1>
128 <h1></h1>
129 <footer>
130 </footer>
   </body>
131
   </html>
132
133
134
135
            self.parser.parse(inputFragment)
136
            self.assertIn(((56, 59), u'duplicate-h1-element', {u'name': u'h1'
137
                }),
                 self.parser.errors, "Failed to report duplicate h1 tags.")
138
139
140
        def test_heading_order(self):
141
            0.00
142
            Test that heading elements maintain correct order within the
                document. The
            order follows from h1-h6 and they must appear in that order
143
                during the document.
144
            Input:
145
            A HTML fragment containing a set of h2 tags appearing before a
146
                set of h1 tags.
147
```

```
Expected Results:
148
149
            An error should be thrown reporting that the h2 tags are out of
               order.
150
151
152
            inputFragment = """
153 <!DOCTYPE html>
154 <html>
155 <head>
156 </head>
157 <body>
158 <h2></h2>
159 <h1></h1>
   <footer>
160
161
   </footer>
162
   </body>
   </html>
163
164
165
            self.parser.parse(inputFragment)
166
167
168
            self.assertIn(((46, 49), u'invalid-heading-order', {u'name': u'h2
                ', u'missing': u'h1'}),
                self.parser.errors, "Failed to report heading tags out of
169
                    order.")
170
171 | if __name__ == '__main__':
172 unittest.main()
```

4.2 Page Structure Tests

```
1 | from __future__ import absolute_import, division, unicode_literals
2
3
   #from . import support
  import unittest, html5lib
4
  from html5lib import treebuilders
5
6
7
   class TestPageStructure(unittest.TestCase):
8
9
       Provides a number of test cases related to basic page structure
10
       for html5 documents.
11
       0.00
19
13
       def setUp(self):
14
           self.parser = html5lib.HTMLParser(tree=treebuilders.
               getTreeBuilder("etree"))
15
16
       def test_singular_tags(self):
           \Pi_{i}\Pi_{j}\Pi_{j}
17
           Test that the multiple-instance-singular-tag error is thrown
18
19
           for cases where more than one instance of a singular tag block is
20
           present.
21
22
           Input:
           Nested blocks of singular tags (html, body, head).
23
           eg. <html></html></html>
24
25
26
           Ouput:
27
           All three test cases should report a multiple instance of both
           the start and closing tags for each of the three singular tags.
28
29
           multipleHTMLInstances = "<html></html></html>"
30
           multipleHeadInstances = "<html><head></head></head></head></od>
31
               body > </html > "
32
           multipleBodyInstances = "<html><head></head><body></body></
               body > </html > "
33
           self.parser.parse(multipleHTMLInstances)
34
35
           self.assertIn(((6, 11), u'multiple-instance-singular-tag', {u'}
36
               name': u'html'}),
37
               self.parser.errors, "Multiple instances of starting HTML tag
                   not reported.")
38
           self.assertIn(((12, 18), u'incorrect-placement-html-end-tag', {u'}
39
               name': u'html'}),
               self.parser.errors, "Multiple instances of closing HTML tag
40
                   not reported.")
41
42
           self.parser.reset()
           self.parser.parse(multipleHeadInstances)
43
44
           self.assertIn(((12, 17), u'multiple-instance-singular-tag', {u'
45
               name': u'head'}),
46
               self.parser.errors, "Multiple instances of starting HTML tag
                   not reported.")
```

```
47
           self.assertIn(((25, 31), u'incorrect-placement-singular-end-tag',
48
                {u'name': u'head'}),
               self.parser.errors, "Multiple instances of closing head tag
49
                   not reported.")
50
           self.parser.reset()
51
           self.parser.parse(multipleBodyInstances)
52
53
           self.assertIn(((25, 30), u'multiple-instance-singular-tag', {u'
54
               name': u'body'}),
55
               self.parser.errors, "Multiple instances of starting HTML tag
                   not reported.")
56
           self.assertIn(((38, 44), u'unexpected-end-tag-after-body', {u'
57
               name': u'body'}),
               self.parser.errors, "Multiple instances of closing body tag
58
                   not reported.")
59
       def test_missing_doctype(self):
60
61
62
           Test that the expected-doctype-but-got-start-tag error is thrown
63
           for cases where no DOCTYPE is declared.
64
65
           Input:
           Nested blocks of singular tags (html, body, head), all of which
66
           are missing the DOCTYPE declaration.
67
68
           eg. <html></html></html>
69
70
           Expected Results:
           All test cases should report a missing DOCTYPE declaration.
71
72
           startTagBeforeDoctype = "<html></html></html>"
73
           endTagBeforeDoctype = "</head></head>"
74
75
           eofBeforeDoctype = ""
76
77
           self.parser.parse(startTagBeforeDoctype)
78
           self.assertIn(((0, 5), u'expected-doctype-but-got-start-tag', {u'
79
               name': u'html'}),
               self.parser.errors, "Failed to report missing DOCTYPE
80
                   declaration (start tag before doctype.")
81
           self.parser.reset()
82
           self.parser.parse(endTagBeforeDoctype)
83
84
           self.assertIn(((0, 6), u'expected-doctype-but-got-end-tag', {u'
85
               name': u'head'}),
               self.parser.errors, "Failed to report missing DOCTYPE
86
                   declaration (closing tag before doctype.")
87
           self.parser.reset()
88
           self.parser.parse(eofBeforeDoctype)
89
90
91
           self.assertIn(((-1, -1), u'expected-doctype-but-got-eof', {}),
               self.parser.errors, "Failed to report missing DOCTYPE
92
                   declaration (EOF before doctype.")
```

```
93
94
95
        def test_closing_html(self):
96
            Test that a missing HTML closing tag is reported when none
97
            are present in the document.
98
99
            Input:
100
101
            Nested blocks of singular tags (head, body).
102
103
            Expected Results:
104
            Report whether the the closing HTML tag is present.
105
            multipleHeadInstances = "<head></head></head>"
106
            multipleBodyInstances = "<body></body></body>"
107
108
109
            self.parser.parse(multipleHeadInstances);
110
            self.assertIn(((-1, -1), u'no-closing-html-tag', {}),
111
112
                self.parser.errors, "Failed to report missing closing HTML
                    tag.")
113
            self.parser.reset()
114
115
            self.parser.parse(multipleBodyInstances)
116
            self.assertIn(((-1, -1), u'no-closing-html-tag', {}),
117
118
                self.parser.errors, "Failed to report missing closing HTML
                    tag.")
119
        def test_misplaced_tags_before_head(self):
120
121
            Test that both start and closing tags occurring before the head
122
123
            section are reported as being misplaced.
124
125
            Input:
126
            A number of instances of start and closing tags being placed
                before
            the head section.
127
128
129
            Expected Results:
            Report whether or not the tags preceding the head section are
130
                reported
            as being misplaced.
131
132
            misplacedHeadTags = "<html><body></body><head></head></html>"
133
            misplacedLinkTags = "<html><a></a><head></head><body></body></
134
               html>"
135
136
            self.parser.parse(misplacedHeadTags)
137
            self.assertIn(((6, 11), u'incorrect-start-tag-placement-before-
138
                head', {u'name': u'body'}),
                self.parser.errors, "Failed to report start body tag before
139
                    head section.")
140
            self.assertIn(((12, 18), u'incorrect-end-tag-placement-before-
141
                head', {u'name': u'body'}),
```

```
142
                self.parser.errors, "Failed to report closing body tag before
                     head section.")
143
144
            self.parser.reset()
            self.parser.parse(misplacedLinkTags)
145
146
147
            self.assertIn(((6, 8), u'incorrect-start-tag-placement-before-
                head', {u'name': u'a'}),
                self.parser.errors, "Failed to report start link (a) tag
148
                    before head section.")
149
            self.assertIn(((9, 12), u'incorrect-end-tag-placement-before-head
150
                ', {u'name': u'a'}),
                self.parser.errors, "Failed to report closing link (a) tag
151
                    before head section.")
152
153
        def test_incorrect_tags_in_head(self):
154
            Test that tags which don't belong in the head section
155
            are reported as misplaced using the 'incorrect-start-tag-
156
                placement - in - head'
157
            and 'incorrect-end-tag-placement-in-head' errors.
158
159
            Input:
160
            A HTML fragment with a pair of head tags enclosing a tag
            pair which doesn't belong in the head phase.
161
162
163
            Expected Results:
            Inclusion of the 'incorrect-start-tag-placement-in-head'
164
            and 'incorrect-end-tag-placement-in-head' errors being reported
165
166
            as part of the returned array of error codes.
167
168
            inputFragment = "<html><head><a></a></head></html>"
169
170
            self.parser.parse(inputFragment)
171
172
            self.assertIn(((12, 14), u'incorrect-start-tag-placement-in-head'
                , {u'name': u'a'}),
                self.parser.errors, "Failed to report starting tag which
173
                    doesn't belong in the head section.")
174
175
        def test_tags_after_eof(self):
176
177
            Tests that starting and closing tags occurring after the last
            instace of a closing HTML tag are reported as an error.
178
179
180
181
            A HTML fragment with a start and closing tag pair occurring
182
            after the start and closing HTML pair.
183
184
            Expected Results:
            An error being thrown for both the start and closing tags
185
                occurring
            after the HTML tags.
186
187
188
            inputFragment = "<html></html><a></a>"
189
```

```
190
            self.parser.parse(inputFragment)
191
192
            self.assertIn(((13, 15), u'expected-eof-but-got-start-tag', {u'
193
                name': u'a'}),
                self.parser.errors, "Failed to report start tag after closing
194
                     HTML tag.")
195
196
            self.assertIn(((16, 19), u'expected-eof-but-got-end-tag', {u'name
                ': u'a'}),
                self.parser.errors, "Failed to report closing tag after
197
                    closing HTML tag.")
198
        def test_missing_start_tag(self):
199
200
            Tests that a missing start tag is reported in the case
201
            that a closing tag is found without a matching start tag.
202
203
204
            Input:
205
            A HTML fragment containing a closing tag without a matching
            start tag.
206
207
            Expected Results:
208
209
            An error being thrown reporting that the matching start tag
            is missing.
210
211
212
            inputFragment = "<html><head></head><body></a></body></html>"
213
214
            self.parser.parse(inputFragment)
215
216
            self.assertIn(((25, 28), u'unexpected-end-tag', {u'name': u'a'}),
217
218
                self.parser.errors, "Failed to report the lack of a matching
                    start tag.")
219
220
        def test_misplaced_tags_after_body(self):
221
222
            Tests that any tags occurring after the body phase
            are reported as being incorrectly placed.
223
224
225
            Input:
            A HTML fragment with a pair of start and closing tags placed
226
227
            after the closing body tag.
228
            Expected Results:
229
230
            An error should be thrown for both the start and closing
            tags found after the closing body tag.
231
232
            0.00
233
            inputFragment = "<html><head></head></body></body><a></a></html>"
234
235
            self.parser.parse(inputFragment)
236
237
            self.assertIn(((32, 34), u'unexpected-start-tag-after-body', {u'
238
                name': u'a'}),
239
                self.parser.errors, "Failed to report misplaced starting tag
                    found after the closing body tag.")
```

```
240
            self.assertIn(((35, 38), u'unexpected-end-tag-after-body', {u'
241
               name': u'a'}),
                self.parser.errors, "Failed to report misplaced closing tag
242
                   found after the closing body tag.")
243
244
        def test_missing_closing_html_tag(self):
245
246
            Test that a missing closing HTML tag is reported.
247
248
            Input:
            A HTML fragment missing a closing HTML tag.
249
250
            Expected Results:
251
            An error should be thrown stating that the closing HTML tag is
252
               missing.
253
254
            255
256
            self.parser.parse(inputFragment)
257
258
259
            self.assertIn(((-1, -1), u'no-closing-html-tag', {}),
                self.parser.errors, "Failed to report missing closing HTML
260
                   tag.")
261
262
        def test_early_termination_before_head(self):
263
            Test that an early closing HTML tag before the head phase
264
265
            is reported as an error.
266
267
            Input:
            A HTML fragment with the head and body sections placed after
268
269
            a closed set of HTML tags.
270
271
            Expected Results:
            An error should be thrown stating that the closing HTML tag
272
273
            has been found before the head phase.
274
275
276
            inputFragment = "<html></html><head></head><body></body>"
277
278
            self.parser.parse(inputFragment)
279
            self.assertIn(((6, 12), u'early-termination-before-head', {u'name
280
               ': u'html'}),
281
                self.parser.errors, "Failed to report early termination
                   before head section.")
282
        def test_early_termination_in_head(self):
283
284
            Test that an early closing HTML tag in the head phase
285
286
            is reported as an error.
287
288
            Input:
289
            A HTML fragment with the closing HTML tag placed within
290
           the set of head tags.
```

```
291
292
            Expected Results:
            An error should be thrown stating that the closing HTML tag
293
294
            has been found in the head phase.
295
296
            inputFragment = "<html><head></html></head><body></body>"
297
298
299
            self.parser.parse(inputFragment)
300
            self.assertIn(((12, 18), u'early-termination-in-head', {u'name':
301
                u'html'}),
                self.parser.errors, "Failed to report early termination
302
                    before head section.")
303
304
        def test_early_termination_before_body(self):
305
            Test that an early closing HTML tag before the body phase
306
307
            is reported as an error.
308
309
310
            A HTML fragment with the closing HTML tag placed before the body
311
            section.
312
313
            Expected Results:
            An error should be thrown stating that the closing HTML tag
314
315
            has been found before the body phase.
316
317
318
            inputFragment = "<html><head></head></html><body></body>"
319
            self.parser.parse(inputFragment)
320
321
322
            self.assertIn(((19, 25), u'early-termination-before-body', {u'
                name': u'html'}),
323
                self.parser.errors, "Failed to report early termination
                    before head section.")
324
        def test_early_termination_in_body(self):
325
326
            Test that an early closing HTML tag in the body phase
327
328
            is reported as an error.
329
            Input:
330
            A HTML fragment with the closing HTML tag placed within
331
332
            the set of body tags.
333
334
            Expected Results:
335
            An error should be thrown stating that the closing HTML tag
            has been found in the head phase.
336
337
338
            inputFragment = "<html><head></head><body></html></body>"
339
340
341
            self.parser.parse(inputFragment)
342
```

```
343
            self.assertIn(((25, 31), u'early-termination-in-body', {u'name':
                u'html'}),
                self.parser.errors, "Failed to report early termination
344
                    before head section.")
345
        def test_tags_between_head_body(self):
346
347
            Test that a set of tags placed after the head section
348
349
            but before the body section is reported as an error.
350
351
            Input:
352
            A HTML fragment with a set of tags between the head
353
            and body sections.
354
            Expected Results:
355
356
            An error should be thrown stating that the set of tags
            can't be placed between the head and body sections.
357
358
359
            inputFragment = "<html><head></head><a></a><body></body></html>"
360
361
362
            self.parser.parse(inputFragment)
363
            self.assertIn(((19, 21), u'start-tag-before-body-after-head', {u'})
364
                name': u'a'}),
                self.parser.errors, "Failed to report start tag after head
365
                    phase but before body phase.")
366
            self.assertIn(((22, 25), u'end-tag-before-body-after-head', {u'
367
                name': u'a'}),
                self.parser.errors, "Failed to report closing tag after head
368
                    phase but before body phase.")
369
370
        def test_missing_starting_html_tag(self):
371
372
            Test that a missing starting HTML tag is reported as an error.
373
374
            Input:
            A HTML fragment missing a starting HTML tag.
375
376
            Expected Results:
377
            An error should be thrown indicating that the fragment doesn't
378
379
            contain a starting HTML tag.
380
381
            inputFragment = "<head></head><body></body></html>"
382
383
384
            self.parser.parse(inputFragment)
385
            self.assertIn(((-1, -1), u'no-starting-html-tag', {}),
386
                self.parser.errors, "Failed to report missing starting HTML
387
                    tag.")
388
        def test_unknown_doctype(self):
389
390
391
            Test that a doctype with an invalid name is reported as being
            an unknown doctype.
392
```

```
393
394
            Input:
            A HTML fragment containing an invalid doctype name.
395
396
            Expected Results:
397
            An error should be thrown reporting that the doctype name is
398
                invalid.
399
400
401
            inputFragment = '<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN
                " "http://www.w3.org/TR/html4/strict.dtd">'
402
403
            self.parser.parse(inputFragment)
404
            self.assertIn(((0, 89), u'unknown-doctype', {}),
405
                 self.parser.errors, "Failed to report unknown doctype.")
406
407
        def test_space_after_doctype(self):
408
409
410
            Test that a doctype tag has a space between the doctype
                declaration
411
            and the doctype name.
412
413
            Input:
            A HTML fragment containing a doctype with no space between the
414
415
            declaration and the doctype name.
416
417
            Expected Results:
            An error should be thrown reporting that there is no space
418
            the doctype declaration and the doctype name.
419
420
421
422
            inputFragment = '<!DOCTYPEhtml>'
423
424
            self.parser.parse(inputFragment)
425
            self.assertIn(((0, 8), u'need-space-after-doctype', {}),
426
427
                 self.parser.errors, "Failed to report missing space after the
                     doctype declaration.")
428
429
        def test_end_tag_before_doctype(self):
430
            Test that a closing tag isn't placed before the doctype
431
               declaration.
432
433
434
            A HTML fragment containing a single closing tag.
435
436
            Expected Results:
            An error should be thrown reporting that a closing tag has been
437
438
            placed before the doctype declaration.
439
440
441
            inputFragment = '</html>'
442
```

```
443
            self.parser.parse(inputFragment)
444
            self.assertIn(((0, 6), u'expected-doctype-but-got-end-tag', {u'
445
                name': u'html'}),
                self.parser.errors, "Failed to report closing tag before
446
                    doctype declaration.")
447
448
        def test_EOF_before_doctype(self):
449
450
            Test that an error is reported if the document is blank.
451
452
            Input:
453
            A blank document containing no characters of any kind.
454
455
            Expected Results:
            An error should be thrown reporting that the EOF was reached
456
                before
            a doctype was declared.
457
458
459
460
            inputFragment = """
461
            0.00
462
463
            self.parser.parse(inputFragment)
464
465
466
            self.assertIn(((-1, -1), u'expected-doctype-but-got-eof', {}),
                 self.parser.errors, "Failed to report the EOF occurring
467
                    before the doctype declaration.")
468
469
        def test_form_element_not_in_form(self):
470
471
            Test that form elements must be contained in wrapping form tags.
472
473
            Input:
474
            A HTML fragment containing form elements which aren't wrapped in
475
            form tags.
476
            Expected Results:
477
            An error should be thrown reporting that the form elements aren't
478
                 contained
479
            in wrapping form tags.
480
481
            inputFragment = """
482
   <!DOCTYPE html>
483
484 <html>
485 | <head>
486
   </head>
   <body>
487
   <datalist></datalist>
488
   <fieldset></fieldset>
489
   <input></input>
490
   <label></label>
491
   <output></output>
492
493
   </body>
494 </html>
```

```
495
496
            self.parser.parse(inputFragment)
497
498
            self.assertIn(((46, 55), u'form-element-not-in-form', {u'name': u
499
                'datalist'}),
500
                self.parser.errors, "Failed to report datalist tag outside of
                     wrapping form tags.")
501
502
            self.assertIn(((68, 77), u'form-element-not-in-form', {u'name': u
                'fieldset'}),
                self.parser.errors, "Failed to report fieldset tag outside of
503
                     wrapping form tags.")
504
            self.assertIn(((90, 96), u'form-element-not-in-form', {u'name': u
505
                'input'}),
                self.parser.errors, "Failed to report input tag outside of
506
                    wrapping form tags.")
507
            self.assertIn(((106, 112), u'form-element-not-in-form', {u'name':
508
                u'label'}),
                self.parser.errors, "Failed to report label tag outside of
509
                    wrapping form tags.")
510
            self.assertIn(((122, 129), u'form-element-not-in-form', {u'name':
511
                u'output'}),
                self.parser.errors, "Failed to report output tag outside of
512
                    wrapping form tag.")
513
        def test_duplicate_id_value(self):
514
515
            Test that the occurrence of duplicate id values is reported as an
516
517
518
            Input:
519
            A HTML fragment containing 2 elements with the same value for
               their id attribute.
520
            Expected Results:
521
522
            An error should be thrown reporting that an id attribute has the
               same value of a previously declared id
523
            attribute.
524
525
            inputFragment = """
526
   <!DOCTYPE html>
527
   <html>
529
   <head>
530
   </head>
   <body>
531
   532
   533
534
   </body>
   </html>
535
536
537
            self.parser.parse(inputFragment)
538
```

```
539
            self.assertIn(((64, 76), u'duplicate-id-attribute', {u'name': u'p
540
                ', u'original': u'p'}),
                 self.parser.errors, "Failed to report duplicate id value
541
                    usage.")
542
543
        def test_duplicate_page_title(self):
544
            Test that a duplicate title element is reported as an error.
545
546
547
            Input:
            A HTML fragment containing duplicate title elements.
548
549
550
            Expected Results:
            An error should be thrown reporting that a duplicate instance of
551
                the page
            title has been found.
552
553
554
            inputFragment = """
555
   <!DOCTYPE html>
557
   <html>
558 <head>
559 <title>blah</title>
   <title>blah</title>
560
   </head>
561
   <body>
562
563
    </body>
564
    </html>
            . . . .
565
566
            self.parser.parse(inputFragment)
567
568
569
            self.assertIn(((51, 57), u'duplicate-title-in-head', {u'name': u'
                title'}),
                 self.parser.errors, "Failed to report duplicate title element
570
                    .")
571
        def test_missing_title_element(self):
572
573
            Test that a missing title element as part of the head section
574
            is reported as missing.
575
576
577
            Input:
            A HTML fragment containing a basic page structure but missing the
578
                 required
579
            title element in the head section.
580
581
            Expected Results:
            An error should be thrown reporting that the title element is
582
                missing from
            the head sectionself.
583
584
585
            inputFragment = """
586
587
   <!DOCTYPE html>
588 <html>
```

```
589 <head>
590
    </head>
591
    <body>
592
    </body>
593
    </html>
             0.00
594
595
596
            self.parser.parse(inputFragment)
597
598
            self.assertIn(((-1, -1), u'title-element-missing-from-head', {}),
                 self.parser.errors, "Failed to report missing title element."
599
600
601
        def test_img_missing_alt_attribute(self):
602
            Test that an img tag missing the required alt attribute is
603
                reported
            as an error.
604
605
606
            Input:
            A HTML fragment containing an img tag missing the required alt
607
                attribute.
608
            Expected Results:
609
            An error should be thrown reporting that the alt attribute is
610
                missing for the
611
            given img tag.
612
613
614
            inputFragment = """
   <!DOCTYPE html>
615
616 | <html>
617 | <head>
618 </head>
619 <body>
620
   <img>
621
    </body>
622
    </html>
623
624
625
            self.parser.parse(inputFragment)
626
             self.assertIn(((46, 50), u'img-element-missing-alt-attribute', {u
627
                'name': u'img'}),
                 self.parser.errors, "Failed to report missing alt attribute
628
                    for the given img tag.")
629
630
        def test_img_alt_attribute_empty(self):
631
            Test that an img tag's alt attribute, when empty, is reported as
632
                an error.
633
634
            A HTML fragment containing an img tag with an empty alt attribute
635
636
            Expected Results:
637
```

```
638
            An error should be thrown reporting that the alt attribute is
                empty.
             . . .
639
640
            inputFragment = """
641
   <!DOCTYPE html>
642
   <html>
643
644
   <head>
645
   </head>
646
   <body>
    <img alt="">
647
648
    </body>
649
    </html>
650
651
            self.parser.parse(inputFragment)
652
653
             self.assertIn(((46, 57), u'img-alt-attribute-empty', {u'attr': u'
654
                 self.parser.errors, "Failed to report empty alt attriubte for
655
                      img tag.")
656
657
        def test_missing_closing_tag_before_footer(self):
658
            Test that any open tags (missing the closing tag) are reported if
659
660
             footer section occurs before closing tag.
661
662
            Input:
            A HTML fragment containing an open 'a' tag which is missing the
663
                closing tag,
            followed by the footer section.
664
665
666
            Expected Results:
667
            An error should be thrown reporting that the closing tag wasn't
                found before
668
            the footer section.
669
670
            inputFragment = """
671
672
    <!DOCTYPE html>
    <html>
673
674
    <head>
675
    </head>
   <body>
676
677
   <a>>
   <footer>
    </footer>
680
    </body>
    </html>
681
682
683
684
            self.parser.parse(inputFragment)
685
            self.assertIn(((50, 57), u'missing-end-tag-before-footer', {u'
686
                name': u'a'}),
```

```
687
                 self.parser.errors, "Failed to report missing end tag before
                    footer section.")
688
        def test_missing_closing_tags_footer_section(self):
689
690
            Test that missing closing tags in the footer section are reported
691
692
693
            Input:
694
            A HTML fragment containing an 'a' tag with a missing closing tag.
695
            Expected Results:
696
            An error should be thrown reporting that the closing tag is
697
                missing within the footer
698
            section.
            0.00
699
700
            inputFragment = """
701
702 <!DOCTYPE html>
703 | <html>
704 <head>
705 </head>
706 | <body >
707 <footer>
   <a>>
708
   </footer>
709
710
    </body>
711
    </html>
            ....
712
713
            self.parser.parse(inputFragment)
714
715
            self.assertIn(((59, 67), u'missing-closing-tags-in-footer', {u'
716
                name': u'a'}),
717
                self.parser.errors, "Failed to report missing closing tag in
                    footer section.")
718
        def test_invalid_tag_name(self):
719
720
721
            Test that tags with invalid tag names are reported as errors.
722
723
            Input:
724
            A HTML fragment containing a tag with an invalid name "blah".
725
            Expected Results:
726
727
            An error should be thrown reporting that the tag name is invalid.
728
729
            inputFragment = """
730
731 <!DOCTYPE html>
732 | <html>
   <head>
733
   </head>
734
   <body>
735
   <blah></blah>
736
   <footer>
737
738 </footer>
```

```
739
    </body>
740
    </html>
            ....
741
742
            self.parser.parse(inputFragment)
743
744
            self.assertIn(((46, 51), u'invalid-element-name', {u'name': u'
745
                blah'}),
746
                self.parser.errors, "Failed to report invalid tag name.")
747
        def test_missing_closing_tag(self):
748
749
            Test that any missing closing tags are reported as errors.
750
751
752
            Input:
            A HTML fragment containing an opening 'a' tag with a missing
753
                closing tag.
754
            Expected Results:
755
            An error should be thrown reporting that the closing tag is
756
                missing.
757
758
            inputFragment = """
759
   <!DOCTYPE html>
760
   <html>
761
   <head>
762
763
    </head>
   <body>
764
765
   <a>>
766
   <footer>
767
   </footer>
768
   </body>
769
   </html>
            ....
770
771
772
            self.parser.parse(inputFragment)
773
            self.assertIn(((46, 48), u'missing-end-tag', {u'name': u'a'}),
774
                self.parser.errors, "Failed to report missing closing tag.")
775
776
   if __name__ == '__main__':
777
    unittest.main()
```

4.3 JSON-RPC Server Tests

```
1 | from __future__ import absolute_import, division, unicode_literals
2
  #from . import support
3
4 import unittest
  import time
5
  import jsonrpclib
6
7
  from multiprocessing import Pool
   from jsonrpclib import Server
9
  import httplib
10
  import simplejson as json
11 | import base64
12
13
14 Calls the test_concurrency method on the server. Required to be external
  TestJsonServer class as it was causing a "pickling" error when used as a
15
      method.
   0.00
16
   def getTime(time):
17
18
           return Server("http://localhost:8080").test_concurrency(time)
19
20
   class TestJsonServer(unittest.TestCase):
21
22
       Provides a number of test cases related to the functionality of the
           json
23
       rpc server.
24
25
       These tests require that the server is currently running. The first
       test checks that the server is running.
26
       0.00
27
28
       def setUp(self):
29
           self.startTime = time.time()
30
31
       def resetCurrentTime(self):
32
           self.startTime = time.time()
33
34
       def getExecutionTimes(self, numProcesses):
35
36
37
           Attempts to call the getTime function with startTime as the
38
           only argument in a concurrent manner using numProcesses as the
           number of concurrent calls to make. The resulting times returned
39
           by the remote function, test_concurrency, are added to a list
40
           and returned.
41
42
           The timeout for each call attempt is currently set to 5 seconds.
43
           This will only allow for numProcesses to go up to 10. After that
44
           the processing times at the server side will trigger the timeout
45
46
           and result in an exception being thrown.
47
           0.00
48
           results = []
49
50
           times = []
51
           pool = Pool(processes=numProcesses)
```

```
for i in range(numProcesses):
53
                results.append(pool.apply_async(getTime, (self.startTime,)))
54
55
            for result in results:
56
                times.append(result.get(timeout=5))
57
58
59
            return times
60
61
        def test_client(self):
62
            Test that the server is currently running. Required for the
63
64
            remaining server tests to run.
65
            Input: Attempt to execute a known function on the server.
66
67
            Expected Result: No exception being raised, implying that the
68
                server
            is currently running.
69
70
71
72
            exceptionRaised = False;
73
            try:
                getTime(self.startTime)
74
75
            except:
76
                exceptionRaised = True
77
            self.assertFalse(exceptionRaised, "The server isn't running.")
78
79
80
        def test_concurrent_connections(self):
81
            Test that the server can handle the maxmimum number of concurrent
82
            connections while receiving a response in a similar time frame
83
84
            for all requests.
85
86
            Input: 5 concurrent function calls to the server.
87
            Expected Result: The remote function, test_concurrency, contains
88
            second sleep call. The sum of the times taken to complete each of
89
            the function calls, relative to self.startTime should be between
90
            range 11 > totalTime >= 10.
91
92
93
            self.resetCurrentTime()
94
95
96
            totalTime = 0
97
98
            for time in self.getExecutionTimes(5):
                totalTime += time
99
100
            self.assertTrue(totalTime >= 10 and totalTime < 11, "Failed to "</pre>
101
                "execute all 5 concurrent function calls within the expected
102
103
                "time frame.")
104
```

```
105
        def test_max_concurrent_connections(self):
106
107
            Tests that the server processes excess function calls after the
            initial batch of calls.
108
109
            Input: 6 concurrent function calls to the server.
110
111
112
            Expected Result: The remote function, test_concurrency, contains
113
            second sleep call. The server has a maximum number of concurrent
            connections of 5, so the 6th call will take slightly over 4
114
                seconds
115
            to complete. The sum of the times for all 6 calls should be
                within
116
            the range 15 > totalTime >= 14.
117
118
            self.resetCurrentTime()
119
120
121
            totalTime = 0
122
123
            for time in self.getExecutionTimes(6):
124
                totalTime += time
125
            self.assertTrue(totalTime >= 14 and totalTime < 15, "Failed to "</pre>
126
127
                "execute all 6 concurrent function calls within the expected
                "time frame.")
128
129
        def test_json_rpc_correct_response(self):
130
131
132
            Tests that the server responds as expected to a correctly
133
            formed JSON-RPC 2.0 request.
134
135
            Input: A correctly formed JSON-RPC 2.0 request containing
            an empty array of file names, an empty file name (direct
136
            input method) and a HTML fragment consisting of '<html></html>'.
137
138
139
            Expected Result: The returned JSON-RPC 2.0 response string
            should match the string expectedResponse, which contains
140
141
            the expected array of errors.
142
            conn = httplib.HTTPConnection("127.0.0.1:8080")
143
            fragment = base64.b64encode(b'<html>')
144
            params = [{"files": [], "document": fragment, "filename": ""}]
145
            request = json.JSONEncoder().encode({"jsonrpc": "2.0", "method":
146
                "parse_html",
                "params": params, "id": "A3s23"})
147
            header = {"Content-type": "application/json"}
148
149
            conn.request("POST", "", request, header)
150
151
            response = conn.getresponse()
            conn.close()
152
153
            expectedResponse = '{"jsonrpc": "2.0", "result": [[1, 0, 5, {"
154
               name": "html"}], [25, 6, 12, {"name": "html"}]], "id": "A3s23
```

```
155
156
            self.assertEqual(response.read(), expectedResponse, "Wrong
                response.")
157
        def test_json_rpc_malformed_parameters(self):
158
159
            Tests that the server responds with an error when
160
161
            a request contains incorrect parameters.
162
            Input: A JSON-RPC 2.0 request containing incorrectly
163
            formatted parameters to be passed on to the requested
164
165
            function.
166
            Expected Result: The returned JSON-RPC 2.0 response string
167
168
            should match the string expectedResponse, which contains
            a response representing an invalid parameters error.
169
170
            conn = httplib.HTTPConnection("127.0.0.1:8080")
171
172
            fragment = base64.b64encode(b'<html></html>')
            params = []
173
            request = json.JSONEncoder().encode({"jsonrpc": "2.0", "method":
174
                "parse_html",
                "params": params, "id": "A3s23"})
175
            header = {"Content-type": "application/json"}
176
177
            conn.request("POST", "", request, header)
178
            response = conn.getresponse()
179
180
            conn.close()
181
            expectedResponse = '{"error": {"message": "Invalid parameters.",
182
                "code": -32602}, "jsonrpc": "2.0", "id": "A3s23"}'
183
            self.assertEqual(response.read(), expectedResponse, "Wrong
184
                response.")
185
186
        def test_json_rpc_unsupported_method(self):
187
            Tests that the server responds with an error when
188
            a client attempts to make a function call for a function
189
            which hasn't been registered to the server.
190
191
            Input: A JSON-RPC 2.0 request containing a function name
192
            which hasn't been registered on the server.
193
194
            Expected Result: The returned JSON-RPC 2.0 response
195
            string should match the string expectedResponse, which
196
197
            contains a response representing an unsupported method
198
            error.
            0.00
199
            conn = httplib.HTTPConnection("127.0.0.1:8080")
200
            fragment = base64.b64encode(b'<html>')
201
            params = [{"files": [], "document": fragment, "filename": ""}]
202
            request = json.JSONEncoder().encode({"jsonrpc": "2.0", "method":
203
                "not_registered",
                "params": params, "id": "A3s23"})
204
205
            header = {"Content-type": "application/json"}
```

```
206
            conn.request("POST", "", request, header)
207
208
            response = conn.getresponse()
209
            conn.close()
210
            expectedResponse = '{"error": {"message": "Method not_registered
211
                not supported.", "code": -32601}, "jsonrpc": "2.0", "id": "
                A3s23"}'
212
213
            self.assertEqual(response.read(), expectedResponse, "Wrong
                response.")
214
215
        def test_invalid_filepath(self):
216
217
            Tests that the server responds with an error when
218
            a client attempts to make a function call for a function
            which hasn't been registered to the server.
219
220
            Input: A JSON-RPC 2.0 request containing a function name
221
222
            which hasn't been registered on the server.
223
224
            Expected Result: The returned JSON-RPC 2.0 response
225
            string should match the string expectedResponse, which
226
            contains a response representing an unsupported method
227
            error.
228
229
            conn = httplib.HTTPConnection("127.0.0.1:8080")
            fragment = base64.b64encode(b'<img src=../image.jpg><img src=</pre>
230
                directory2/image2.jpg>')
            params = [{"files": ["image.jpg", "directory/", "directory/
231
                current.html"], "document": fragment, "filename": "directory/
                current.html"}]
            request = json.JSONEncoder().encode({"jsonrpc": "2.0", "method":
232
                "not_registered",
                "params": params, "id": "A3s23"})
233
234
            header = {"Content-type": "application/json"}
235
            conn.request("POST", "", request, header)
236
237
            response = conn.getresponse()
238
            conn.close()
239
            expectedResponse = '("error": ("message": "Method not_registered
240
                not supported.", "code": -32601}, "jsonrpc": "2.0", "id": "
                A3s23"}'
241
242
            self.assertEqual(response.read(), expectedResponse, "Wrong
                response.")
243
244
        def test_invalid_filepath(self):
            0.00
245
246
            Tests that the server responds with an error when
            a client attempts to make a function call for a function
247
248
            which hasn't been registered to the server.
249
            Input: A JSON-RPC 2.0 request containing a function name
250
251
            which hasn't been registered on the server.
252
```

```
253
            Expected Result: The returned JSON-RPC 2.0 response
254
            string should match the string expectedResponse, which
255
            contains a response representing an unsupported method
256
            error.
257
            conn = httplib.HTTPConnection("127.0.0.1:8080")
258
259
            fragment = base64.b64encode(b'<img src=../image.jpg><img src=
                directory2/image2.jpg>')
260
            params = [{"files": ["image.jpg", "directory/", "directory/
                current.html"], "document": fragment, "filename": "directory/
                current.html"}]
            request = json.JSONEncoder().encode({"jsonrpc": "2.0", "method":
261
                "not_registered",
262
                "params": params, "id": "A3s23"})
            header = {"Content-type": "application/json"}
263
264
265
            conn.request("POST", "", request, header)
            response = conn.getresponse()
266
            conn.close()
267
268
            expectedResponse = '("error": ("message": "Method not_registered
269
                not supported.", "code": -32601}, "jsonrpc": "2.0", "id": "
                A3s23"}'
270
            self.assertEqual(response.read(), expectedResponse, "Wrong
271
                response.")
272
273
        def test_invalid_filepath(self):
274
275
            Tests that the server responds with an error when
276
            a client attempts to make a function call for a function
277
            which hasn't been registered to the server.
278
279
            Input: A JSON-RPC 2.0 request containing a function name
280
            which hasn't been registered on the server.
281
282
            Expected Result: The returned JSON-RPC 2.0 response
283
            string should match the string expectedResponse, which
284
            contains a response representing an unsupported method
285
            error.
286
            conn = httplib.HTTPConnection("127.0.0.1:8080")
287
            fragment = base64.b64encode(b'<img src=../image.jpg><img src=
288
                directory2/image2.jpg>')
            params = [{"files": ["image.jpg", "directory/", "directory/
289
                current.html"], "document": fragment, "filename": "directory/
                current.html"}]
290
            request = json.JSONEncoder().encode({"jsonrpc": "2.0", "method":
                "not_registered",
                "params": params, "id": "A3s23"})
291
292
            header = {"Content-type": "application/json"}
293
            conn.request("POST", "", request, header)
294
            response = conn.getresponse()
295
296
            conn.close()
297
```

```
298
            expectedResponse = '("error": ("message": "Method not_registered
                not supported.", "code": -32601}, "jsonrpc": "2.0", "id": "
                A3s23"}'
299
            self.assertEqual(response.read(), expectedResponse, "Wrong
300
                response.")
301
        def test_invalid_filepath(self):
302
303
            Tests that the server responds with an error when
304
305
            a client attempts to make a function call for a function
306
            which hasn't been registered to the server.
307
            Input: A JSON-RPC 2.0 request containing a function name
308
309
            which hasn't been registered on the server.
310
            Expected Result: The returned JSON-RPC 2.0 response
311
            string should match the string expectedResponse, which
312
313
            contains a response representing an unsupported method
314
315
316
            conn = httplib.HTTPConnection("127.0.0.1:8080")
317
            fragment = base64.b64encode(b'<img src=../image.jpg><img src=</pre>
                directory2/image2.jpg>')
            params = [{"files": ["image.jpg", "directory/", "directory/
318
                current.html"], "document": fragment, "filename": "directory/
                current.html"}]
            request = json.JSONEncoder().encode({"jsonrpc": "2.0", "method":
319
                "not_registered",
                "params": params, "id": "A3s23"})
320
            header = {"Content-type": "application/json"}
321
322
            conn.request("POST", "", request, header)
323
324
            response = conn.getresponse()
325
            conn.close()
326
327
            expectedResponse = '("error": ("message": "Method not_registered
                not supported.", "code": -32601}, "jsonrpc": "2.0", "id": "
                A3s23"}'
328
            self.assertEqual(response.read(), expectedResponse, "Wrong
329
                response.")
330
331
        def test_invalid_filepath(self):
332
333
            Tests that the server responds with an error when
            a client attempts to make a function call for a function
334
335
            which hasn't been registered to the server.
336
            {\tt Input: A JSON-RPC 2.0 \ request \ containing \ a \ function \ name}
337
338
            which hasn't been registered on the server.
339
340
            Expected Result: The returned JSON-RPC 2.0 response
            string should match the string expectedResponse, which
341
342
            contains a response representing an unsupported method
343
            error.
344
```

```
345
            conn = httplib.HTTPConnection("127.0.0.1:8080")
            fragment = base64.b64encode(b'<img src=../image.jpg><img src=
346
                directory2/image2.jpg>')
            params = [{"files": ["image.jpg", "directory/", "directory/
347
                current.html"], "document": fragment, "filename": "directory/
                current.html"}]
            request = json.JSONEncoder().encode({"jsonrpc": "2.0", "method":
348
                "not_registered",
                "params": params, "id": "A3s23"})
349
350
            header = {"Content-type": "application/json"}
351
            conn.request("POST", "", request, header)
352
353
            response = conn.getresponse()
            conn.close()
354
355
356
            expectedResponse = '{"error": {"message": "Method not_registered
                not supported.", "code": -32601}, "jsonrpc": "2.0", "id": "
                A3s23"}'
357
            self.assertEqual(response.read(), expectedResponse, "Wrong
358
                response.")
359
360
        def test_invalid_filepath(self):
361
362
            Tests that the server responds with an error when
            a client attempts to make a function call for a function
363
            which hasn't been registered to the server.
364
365
366
            Input: A JSON-RPC 2.0 request containing a function name
367
            which hasn't been registered on the server.
368
            Expected Result: The returned JSON-RPC 2.0 response
369
370
            string should match the string expectedResponse, which
            contains a response representing an unsupported method
371
372
            error.
373
374
            conn = httplib.HTTPConnection("127.0.0.1:8080")
375
            fragment = base64.b64encode(b'<img src=../image.jpg><img src=
                directory2/image2.jpg>')
            params = [{"files": ["image.jpg", "directory/", "directory/"]
376
                current.html"], "document": fragment, "filename": "directory/
                current.html"}]
            request = json.JSONEncoder().encode({"jsonrpc": "2.0", "method":
377
                "not_registered",
                "params": params, "id": "A3s23"})
378
379
            header = {"Content-type": "application/json"}
380
381
            conn.request("POST", "", request, header)
382
            response = conn.getresponse()
            conn.close()
383
384
            expectedResponse = '("error": ("message": "Method not_registered
385
                not supported.", "code": -32601}, "jsonrpc": "2.0", "id": "
                A3s23"}'
386
            self.assertEqual(response.read(), expectedResponse, "Wrong
387
                response.")
```

```
388
        def test_invalid_filepath(self):
389
390
391
            Tests that the server responds with an error when
392
            a client attempts to make a function call for a function
            which hasn't been registered to the server.
393
394
395
            Input: A JSON-RPC 2.0 request containing a function name
396
            which hasn't been registered on the server.
397
            Expected Result: The returned JSON-RPC 2.0 response
398
399
            string should match the string expectedResponse, which
400
            contains a response representing an unsupported method
401
402
            conn = httplib.HTTPConnection("127.0.0.1:8080")
403
            fragment = base64.b64encode(b'<img src=../image.jpg><img src=</pre>
404
                directory2/image2.jpg>')
            params = [{"files": ["image.jpg", "directory/", "directory/
405
                current.html"], "document": fragment, "filename": "directory/
                current.html"}]
406
            request = json.JSONEncoder().encode({"jsonrpc": "2.0", "method":
                "not_registered",
                "params": params, "id": "A3s23"})
407
            header = {"Content-type": "application/json"}
408
409
            conn.request("POST", "", request, header)
410
            response = conn.getresponse()
411
            conn.close()
412
413
            expectedResponse = '{"error": {"message": "Method not_registered
414
                not supported.", "code": -32601}, "jsonrpc": "2.0", "id": "
                A3s23"}'
415
416
            self.assertEqual(response.read(), expectedResponse, "Wrong
                response.")
417
   if __name__ == '__main__':
418
       unittest.main()
419
```