Section 4 – Evaluation

Part (A) – Post Development Testing

As outlined in the design section of my project, I originally planned on using four method of testing:

- White box testing
- Black box testing
- Destructive testing
- Unit testing

However as my project progressed I realised how long it was going to take to finish it. I then chose to ignore unit testing as it was very time intensive and didn't provide many benefits for a project this small scale (compared to a project made and maintained by a team of people over several years). This is bad in terms of maintainability, so if I was continuing to develop this project, writing unit tests would be one of my first priorities to ease future development. I did use all three of the other testing approaches.

I also performed tests that didn't require any user data such as simulated load tests on the server and database. This is useful to see how the website would perform if it had lots of concurrent users.

White box tests

White box testing is when a programmer with access to the source code tests the program for functionality and to locate bugs. They look at the source code and test legal and illegal inputs, as well as checking every section actually does what its supposed to do. I used white box testing in my project as my primary testing method because it's quick and it doesn't require asking anyone else to test it, which is good for pre-release testing. I tested all the scripts that required input using white box testing by checking to code to see what could trip the program up, and then testing it against expected results. The results of my white box testing are below:

Browser tests:

Browser	**Works/Doesn't work**	**Notes**
Mozilla Firefox	Works	None
Google Chrome	Works	None
Internet Explorer 11	Works	None
Microsoft Edge	Unknown	Won't connect to website at all so impossible to test
Mozilla Firefox (Android)	Mostly works	Site hasn't been specifically designed for mobile but most pages work.

Google Chrome (Android)	Mostly works	Site hasn’t been specifically designed for mobile but most pages work

Basic functionality:

Page	**Test**	**Test data**	**Works/Doesn’t work**	**Notes**
signup.php	Can create an account	All valid data	Works	Script creates an account using the data provided
signup.php	Can create an account	No data	Works	Script prompts user to fill in missing fields
signup.php	Can create an account	☺☺☺☺☺	Works	Script prompts the user to try again
login.php	Can log in to existing account	Username and password for an account existing in the database	Works	Script logs user into account
login.php	Can’t log into account with wrong password	Username for an account that exists in the database, wrong password for said account	Works	Script prompts user that username/password combination is invalid
login.php	Can’t log into account that doesn’t exist	Username and password that doesn’t exist in database	Works	Script prompts user that username/password combination is invalid
ask.php	Can ask a question with valid inputs	Valid question title and question body	Works	Script adds a new question to the database
ask.php	Can’t ask a question with nothing filled in	No data	Works	Script prompts the user to fill in every field

ask.php	Can ask a question containing unexpected characters	☺☺☺☺☺	Works	Script adds a new question to the database
searchq.php	Can search for a question with valid inputs	Valid search String	Works	Script shows all questions that match the search query
searchq.php	Can't search for a question without supplying any query	No data	Works	Script prompts the user to search for questions
searchq.php	Can search for a question containing unexpected characters	☺☺☺☺☺	Works	Script shows all questions that match the search query
comment.php	Can comment on a question using valid input	Valid input comprised on standard characters	Works	Script adds a new comment to the database
comment.php	Can't comment on a question without supplying a comment	No data	Works	Script prompts the user to input a comment
comment.php	Can comment on a question using unexpected input	☺☺☺☺☺	Works	Script adds a new comment to the database
findpeople.php	Can post a job listing using valid data	Valid job title, description, location, company, salary and tags	Works	Script adds a new job to the database
findpeople.php	Can't post a job listing with no data	No data	Works	Script prompts the user to fill in all the fields
findpeople.php	Can post a job listing with unexpected characters	☺☺☺☺☺	Works	Script adds a new job to the database

findpeople.php	Can add a valid tag	A valid tag	Works	Script adds a new tag to the filter
findpeople.php	Can’t add a tag with no data	No data	Works	Button to add tag does not appear if the input box is empty
findpeople.php	Can add a tag with unexpected characters	☺☺☺☺☺	Works	Script adds a new tag to the filtered
findPeople.php	Can search for a job with valid input	Valid search String	Works	Script shows all jobs that match the search query
findPeople.php	Can’t search for a jobs without supplying any query	No data	Works	Script prompts the user to search for jobs
findPeople.php	Can search for a job containing unexpected characters	☺☺☺☺☺	Works	Script shows all jobs that match the search query
findjobs.php	Can add a valid tag	A valid tag	Works	Script adds a new tag to the filter
findjobs.php	Can’t add a tag with no data	No data	Works	Button to add tag does not appear if the input box is empty
findjobs.php	Can add a tag with unexpected characters	☺☺☺☺☺	Works	Script adds a new tag to the filtered

Black box tests

Black box testing is when the tester has no knowledge of the internals of a program. They do not have access to the code and they don’t know how it works. Black box tests are often used to test functionality and interfaces. I’m using black box tests because the final version of project is so large and complex that it’s very difficult for me to test every single feature. I also use it because I’m biased in favour of the design and interface, and black box testing allows me to see lots of user’s opinions. This will allow me to see if it meets my success criteria.

In order to black box test my project I created a survey using Microsoft Forms. The survey has seven questions designed to test various aspects of my site. When I was designing the survey I tried to make questions that test the parts of the site that I would struggle to do myself, such as opinions on the user interface and how easy it is to use the site etc.

The testers who will be black box testing my project are a class of 14-15 year old GCSE Computer Science students. This is because they are available for me to use for testing the project. This is a very effective way of me getting 30 people's opinions on my website.

The list of questions in my survey and a link to it are below:

1. Did you like the design of the website?

(yes or no, compulsory)

2. Did you struggle to do anything on the website?

(yes or no, compulsory)

3. Did you find the tutorials easy to follow?

(yes or no, compulsory)

4. Is the interface memorable? (Would you remember how to use the website after a break of a few months?)

(yes or no, compulsory)

5. Was it easy to use each of the features for the first time?

(yes or no, compulsory)

6. What (if any) improvements would you like to see?

(text input, optional)

7. Please list any bugs you saw along with steps to reproduce them

(text input, optional)

Link to survey:
https://forms.office.com/Pages/ResponsePage.aspx?id=PYCwG3TptkKUJQEK3ByuatBasbOGJYRIgymUpAILxi1UOFYyUVZDV05SVkJSTVdLU0JDQ0RZMFVCWS4u

The responses I got from my survey were very positive. In total 21 people responded, which I consider a big enough sample size to draw meaningful conclusions. On average people took exactly three minutes to complete the survey which shows people thought about their answers. This makes me more confident in the data. Analysis of the survey responses are below:

1. Did you like the design of the website?

86% (18/21) of people said they liked the design of the website. This means that the user interface and design of my website was successful. As a programmer I've always struggled with design and making things look nice, and I was please at the feedback in this category. The design of the website is the first thing a user sees when they first use it, and having something that people find visually pleasing is essential for keeping people using the site.

2. Did you struggle to do anything on the website?

71% (15/21) of people said they didn't struggle to do anything on the website. While this is slightly lower than the design feedback, it is still very positive considering the majority of the testers were not the target audience for my project. During my analysis I predicted that most of my users would be amateur and more experienced programmers, mostly employed/looking for work, however the testers are all 14-15 and just starting to study computer science. This means that figure of 71% is even better, which shows that some of the design work I did paid off. If people struggle to use a site they'll give up using something straight away.

3. Did you find the tutorials easy to follow?

86% (18/21) of people agreed that the tutorials I wrote were easy to follow. Unlike the some of the other features in my project, the tutorial pages are aimed directly at the kind of people who were testing my project. I was happy with the high percentage of people that thought the tutorials were easy to follow because it shows that they worked as tutorials. There's no point having hard to follow tutorials as they're frustrating and don't end up helping people.

4. Is the interface memorable? (Would you remember how to use the website after a break of a few months?)

71% (15/21) of the testers considered the interface memorable enough to be able to use the website after a break of a few months). Memorability was one of my success criteria, and I consider 71% to be enough to pass this criteria. Memorability is a hard aspect to test for, because it really needs the testers to take a break of a few months, however there wasn't enough time for this. Because of this I'm not surprised that the same percentage of people that didn't struggle to do anything on the website also considered the interface memorable. I included this in the survey because I wanted to get people's initial impressions, and I was pleased with the feedback from this question.

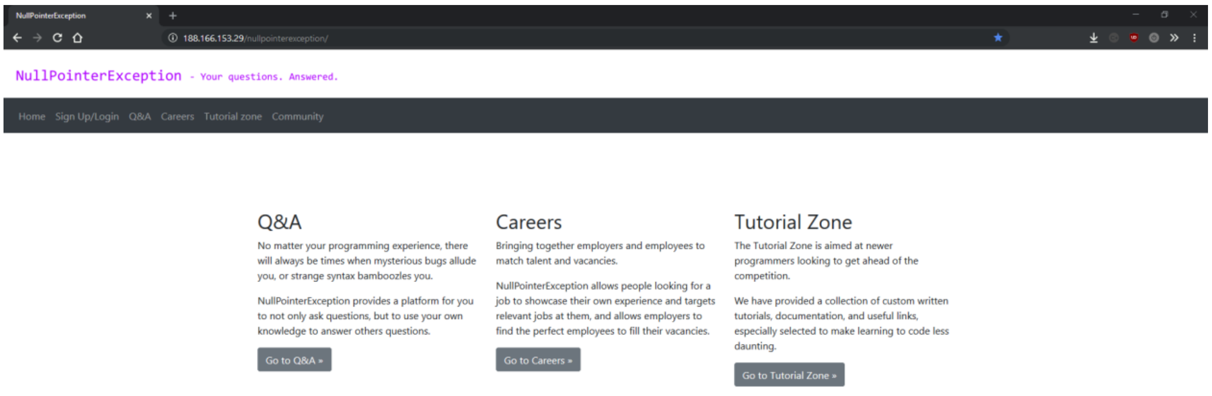
5. Was it easy to use each of the features for the first time?

This is a harder question to analyse because I added a third answer to this question. 67% (14/21) of people thought every feature was easy to use for the first time and a further 29% (6/21) thought most of the features were easy to use for the first time. Only 1 person said that none of the features were easy to use for the first time. This is quite a positive response, but also gives me some insight one what to work on in the future.

6. What (if any) improvements would you like to see?

The final two questions in the survey were both optional and aimed at things for the future rather opinions about the current build. This question received seven responses. Two of theses were about making the layout more interesting and take up more space on the screen. While I see what they're on about, the layout is designed to work well on tablets and phones as well as large computer screens, which means the larger screens end up with more free space. For example, the gap between sections of content and the edges of the screen will be larger on bigger screen resolutions. This can be seen below:

Full size homepage (original image 1920x1080):



NullPointerException - Your questions. Answered.

Q&A

No matter your programming experience, there will always be times when mysterious bugs allude you, or strange syntax bamboozles you.

NullPointerException provides a platform for you to not only ask questions, but to use your own knowledge to answer others questions.

Careers

Bringing together employers and employees to match talent and vacancies.

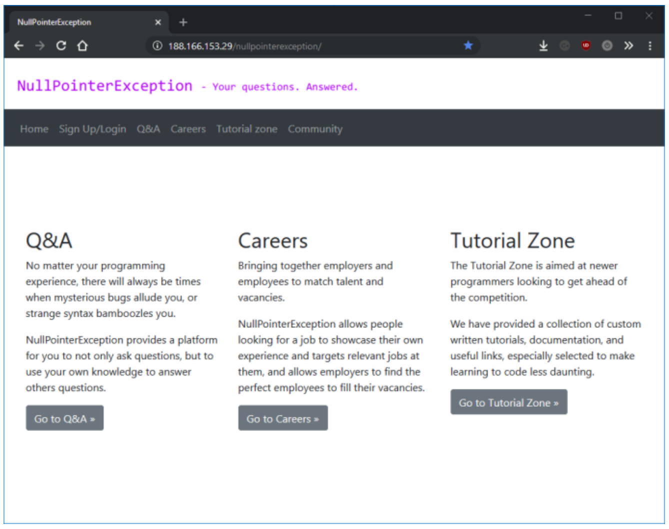
NullPointerException allows people looking for a job to showcase their own experience and targets relevant jobs at them, and allows employers to find the perfect employees to fill their vacancies.

Tutorial Zone

The Tutorial Zone is aimed at newer programmers looking to get ahead of the competition.

We have provided a collection of custom written tutorials, documentation, and useful links, especially selected to make learning to code less daunting.

Small window:


NullPointerException

188.166.153.29/nullpointerexception/

NullPointerException - Your questions. Answered.

Home Sign Up/Login Q&A Careers Tutorial zone Community

Q&A

No matter your programming experience, there will always be times when mysterious bugs allude you, or strange syntax bamboozles you.

NullPointerException provides a platform for you to not only ask questions, but to use your own knowledge to answer others questions.

Go to Q&A »

Careers

Bringing together employers and employees to match talent and vacancies.

NullPointerException allows people looking for a job to showcase their own experience and targets relevant jobs at them, and allows employers to find the perfect employees to fill their vacancies.

Go to Careers »

Tutorial Zone

The Tutorial Zone is aimed at newer programmers looking to get ahead of the competition.

We have provided a collection of custom written tutorials, documentation, and useful links, especially selected to make learning to code less daunting.

Go to Tutorial Zone »

You can see that the gap between the edges of the screen expands on bigger screen sizes. This is because an easy way of creating a responsive website is to design for small screens and expand outwards. This is because a website designed for screen will work on that size and anything bigger (when done properly), but a website designed for a large 4k screen won't work on a small phone. In response to this feedback I would redesign the website on larger screens, as well as creating a separate version for phones, all of which should help address people's criticism.

Another response I got from this question was someone who wanted the tutorials to be broken up a bit and instead of just big paragraphs of text. I completely agree with this point, if you look at any popular tutorials on the internet they are a step by step guide to making a finished project, with the final code shown at the end. This is a better format for tutorials as having big blocks of text can be daunting for some users, and is less interesting. The reason I chose to do this approach is because the tutorials on the site at the moment are more like proof of concepts than finished articles. This is due to the limited time I had to develop this project. In the future I would like to redesign the existing tutorials as well as making more tutorials for other popular languages such as Java/C++.

One tester asked for more colours and a logo. I actually have a logo but since it's made of text some people might not have realised what it was. In the future I might look at creating a more unique design, possibly with a bit more colour, but for now I'm happy with the current design, thanks to the feedback from question one.

7. Please list any bugs you saw along with steps to reproduce them

This was a very useful question to ask. While the other questions have been testing for usability and opinions on design, this question is testing for functionality and robustness. I was expecting either of two options when I asked this question, either the testers would find no bugs meaning that the website is fully functional and works perfectly, or the testers would find bugs that I'd missed during development along with instructions as to how to recreate them. Three users discovered a bug which I hadn't considered, even though it's something I was meaning to address since day one. The bug they found was when you used apostrophes in any of the input field that gets uploaded to the database, the Database class would reject the query and throw an error. This is because I forgot to sanitize my inputs which was very embarrassing. Sanitizing inputs is when you remove or "make safe" ALL the data a user can enter into your database to prevent code injection and MySQL injection. It turns out that when I updated to the project to use the new database class I forgot to re-add the sanitization function. Most of the issues people discovered in my project were quite minor so I'd work on them in the future, but this was such a critical issue with such a simple fix I fixed it immediately. I simply added a new function to the database class which makes any possibly dangerous user data safe.

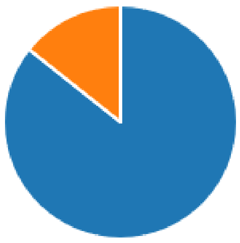
Screenshots of results from the survey:

Irrelevant and inappropriate responses censored in red

1. Did you like the design of the website?

More Details

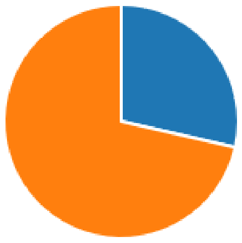
🔵	Yes	18
🟠	No	3



2. Did you struggle to do anything on the website?

More Details

🔵	Yes	6
🟠	No	15



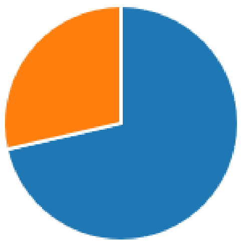
3. Did you find the tutorials easy to follow?

More Details

4. Is the interface memorable? (Would you remember how to use the website after a break of a few months?)

More Details

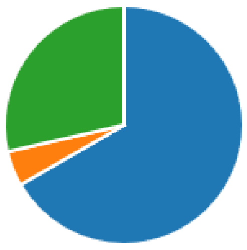
🔵	Yes	15
🟠	No	6

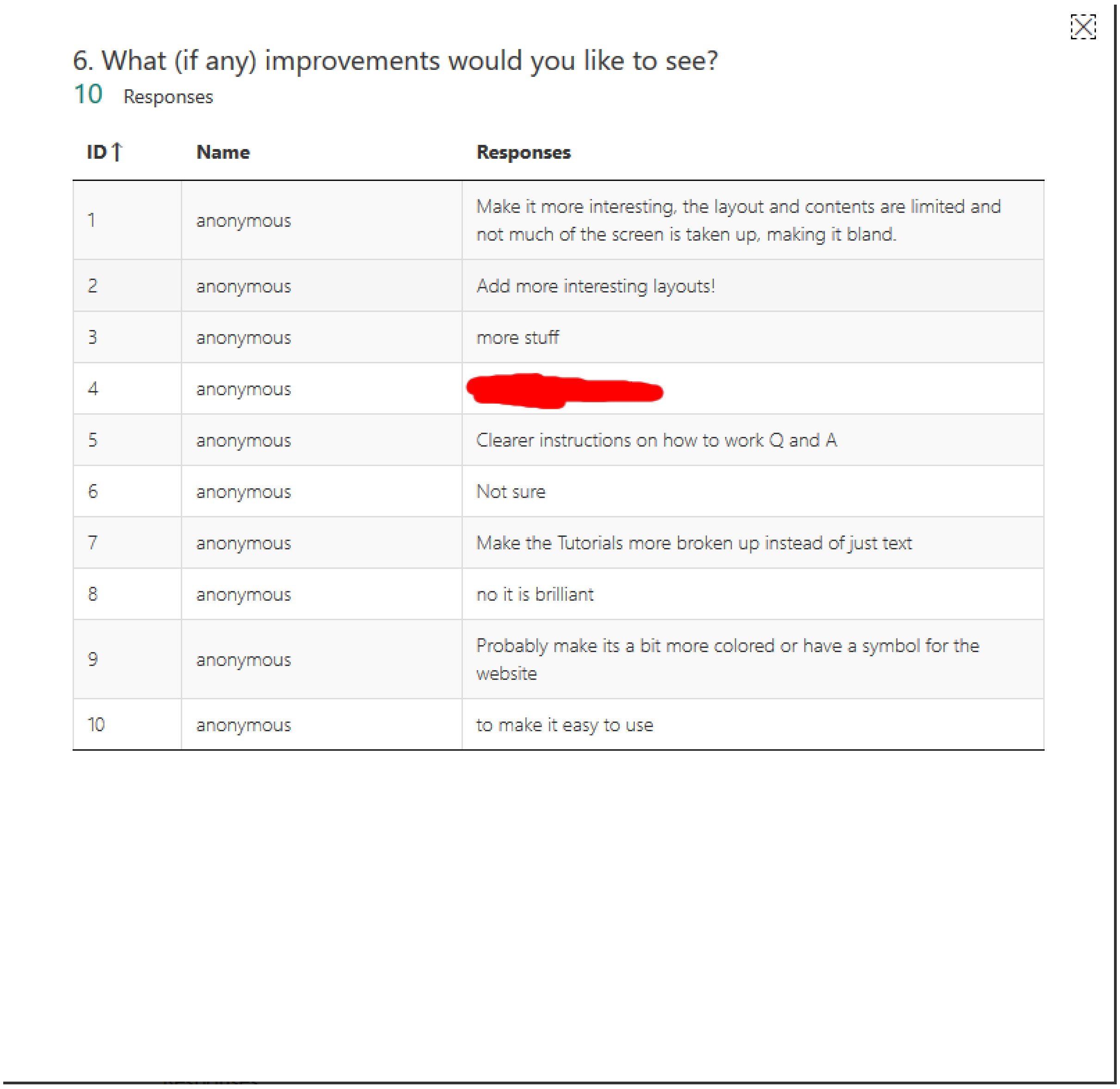


5. Was it easy to use each of the features for the first time?

More Details

●	Yes	14
●	No	1
●	Sometimes	6

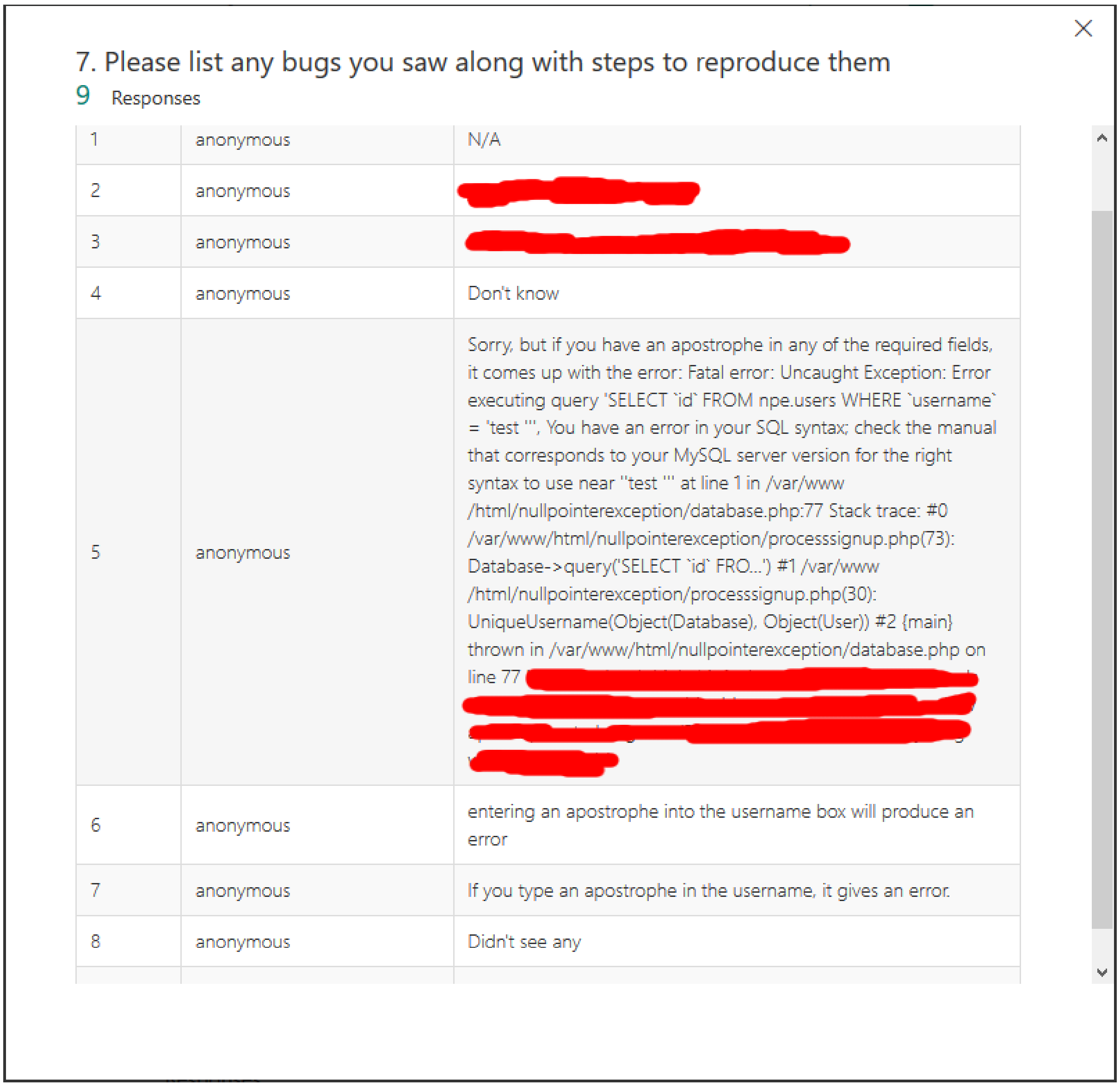




6. What (if any) improvements would you like to see?

10 Responses

ID ↑	Name	Responses
1	anonymous	Make it more interesting, the layout and contents are limited and not much of the screen is taken up, making it bland.
2	anonymous	Add more interesting layouts!
3	anonymous	more stuff
4	anonymous	[redacted]
5	anonymous	Clearer instructions on how to work Q and A
6	anonymous	Not sure
7	anonymous	Make the Tutorials more broken up instead of just text
8	anonymous	no it is brilliant
9	anonymous	Probably make its a bit more colored or have a symbol for the website
10	anonymous	to make it easy to use



7. Please list any bugs you saw along with steps to reproduce them

9 Responses

1	anonymous	N/A
2	anonymous	[redacted]
3	anonymous	[redacted]
4	anonymous	Don't know
5	anonymous	Sorry, but if you have an apostrophe in any of the required fields, it comes up with the error: Fatal error: Uncaught Exception: Error executing query 'SELECT \`id\` FROM npe.users WHERE \`username\` = 'test ''', You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near ''test ''' at line 1 in /var/www /html/nullpointerexception/database.php:77 Stack trace: #0 /var/www/html/nullpointerexception/processsignup.php(73): Database->query('SELECT \`id\` FRO...') #1 /var/www /html/nullpointerexception/processsignup.php(30): UniqueUsername(Object(Database), Object(User)) #2 {main} thrown in /var/www/html/nullpointerexception/database.php on line 77 [redacted]
6	anonymous	entering an apostrophe into the username box will produce an error
7	anonymous	If you type an apostrophe in the username, it gives an error.
8	anonymous	Didn't see any

Part (B) – Evaluation of success criteria

When I was coming up with my success criteria during the analysis section of my project, I came up with one main objective, and six success criteria. First I'll list the success criteria, the I'll evaluate the six success criteria against the finished project and then finally I'll compare the finished project to the main objective.

Main objective

"Can I produce a useable and intuitive solution that provides users with an alternative to StackOverflow, while providing additional social and career features not found in mainstream programming sites".

Success criteria

- User friendly and intuitive

- Functional and robust question and answer system

- Functional and professional career system

- Functional tutorial zone

- Fast and reliable database and back-end

- Secure back-end

User friendly and intuitive

This was quite an important success criteria for me as its very important for a user to have a good first impression when they use a website for the first time. In my analysis I said I'd be able to quantify this by getting a class of younger students to test the project. 86% of the testers who responded to my survey said that they liked the design of the website, which I think makes the project meet the user friendly success criteria. 71% of people also said they had no problem trying to use any of the features, which is a big enough percentage for me to consider this criteria met.

Functional and robust question and answer system

In my analysis I split this success criteria up into subsections as I felt it was to broad and vague on its own. These subcategories and an analysis of each of them are listed below:

Success criteria	**Met/Partially met/Not met**	**Notes**
User can ask questions	Met	Proved by both my white box tests and the functionality tests performed by independent testers
Users can view asked questions	Met	Proved by both my white box tests and the functionality tests performed by independent testers
Users can search for questions	Met	Proved by both my white box tests and the functionality tests performed by independent testers

Users can filter questions by the three categories “Hot”, “Top” and “New”	Met	Proved by both my white box tests and the functionality tests performed by independent testers
Users can tag questions with the language they are about	Not met	Not met due to time constraints. This feature will be added in the future for improved user experience
Users can filter questions by their language tag	Not met	Not met due to time constraints. This feature will be added in the future for improved user experience
Users can comment on questions	Met	Proved by both my white box tests and the functionality tests performed by independent testers
Users can edit comments	Met	Proved by both my white box tests and the functionality tests performed by independent testers
Users can vote on questions	Met	Proved by both my white box tests and the functionality tests performed by independent testers
Users can report inappropriate questions	Not met	Not met due to time constraints. This feature will be added in the future for improved user experience
Users can save their favourite questions	Not met	Not met due to time constraints. This feature will be added in the future for improved user experience
Users can answer questions	Partially met	Answer system merged into comment system for a more forum style approach
Users can edit answers	N/A	Answer system merged into comment system for a more forum style approach

Users can vote on answers	Partially met	Answer system merged into comment system for a more forum style approach
Users can accept answers	Partially met	Answer system merged into comment system for a more forum style approach
Users can report answers	Partially met	Answer system merged into comment system for a more forum style approach
Users can comment on answers	Partially met	Answer system merged into comment system for a more forum style approach
Users can report comments	Not met	Not met due to time constraints. This feature will be added in the future for improved user experience

Overall I think that I met enough of these subsections to class this success criteria as passed. One thing I ended up doing differently during development to what I drew up in my analysis was merging comments and answers. This is due to more research into StackOverflow, which has both comments and answers, which I feel makes the whole process more complicated than it needs to be. Merging them also cut down the already extensive development time.

I feel that all the subsections that were fully met were well executed and worked exactly how I intended them to. I was particularly pleased with the design and functionality of the Q&A system as it was the first section of the website I completed and showed me that the time I'd spent planning and designing it has not been wasted.

Functional and professional careers system

This criteria is also split into categories. An analysis of each of these categorises is below:

Success criteria	**Met/Partially met/Not met**	**Notes**
Employers can create adverts for jobs	Met	Proved by both my white box tests and the functionality tests performed by independent testers
Employers can remove adverts for jobs	Not met	Not met due to time constraints. This feature will be

		added in the future for improved user experience
Adverts will be taken down once the position is filled	Not met	Not met due to time constraints. This feature will be added in the future for improved user experience
Employers can view analytics data for their advert	Not met	Not met due to time constraints. This feature will be added in the future for improved user experience
Employers can tag jobs	Met	Proved by both my white box tests and the functionality tests performed by independent testers
Employers can specify job location	Met	Proved by both my white box tests and the functionality tests performed by independent testers
Users can report jobs	Not met	Users can apply for jobs if the employer adds their contact details (email address/phone number) to the job description
Users can apply for jobs	Met	Users can apply for jobs if the employer adds their contact details (email address/phone number) to the job description
Users can have their job displayed on their profile	Not met	Not met due to time constraints. This feature will be added in the future for improved user experience
Users can contact employers	Met	Users can apply for jobs if the employer adds their contact details (email address/phone number) to the job description
Users can search for jobs based on a search query	Met	Proved by both my white box tests and the functionality tests performed by independent testers

Users can search for jobs based on tags	Met	Proved by both my white box tests and the functionality tests performed by independent testers
Users can search for jobs based on location	Met	Proved by both my white box tests and the functionality tests performed by independent testers
Users can view jobs on an interactive map	Not met	Not met due to time constraints. This feature will be added in the future for improved user experience

I was happier with how I met the success criteria for this section than in the Q&A section. I met more of the subsections and I think the career system is a more polished section of the website than the question and answer system. I also enjoyed the programming for the careers more than any other area of the site because it involved some interesting problem solving and approaches to problems I couldn't imagine when I was planning the project.

The responses from my survey back up my opinions on the careers system. 86% of people said they liked the design, 71% of people said they had no problem using any of the features, and 95% of people thought at least most of the features were easy to use for the first time. These are all high percentages which I think

Adding an interactive map would've been a very simple task program using the free Google API but I didn't add it because of the complexity it would add to both the site and the write-up. This is the main reason for not adding seemingly simple features to the project.

Functional tutorial zone

This criteria is also split into categories. An analysis of each of these categorises is below:

Success Criteria	**Met/Partially met/Not met**	**Notes**
User friendly UI	Met	Proved by external testers during black box tests
Good UX design	Met	Proved by external testers during black box tests
Detailed and easy to use tutorials	Met	Proved by external testers during black box tests
No dead links	Met	Passed at time of writing

No spelling mistakes	Met	Proved by external testers during black box tests

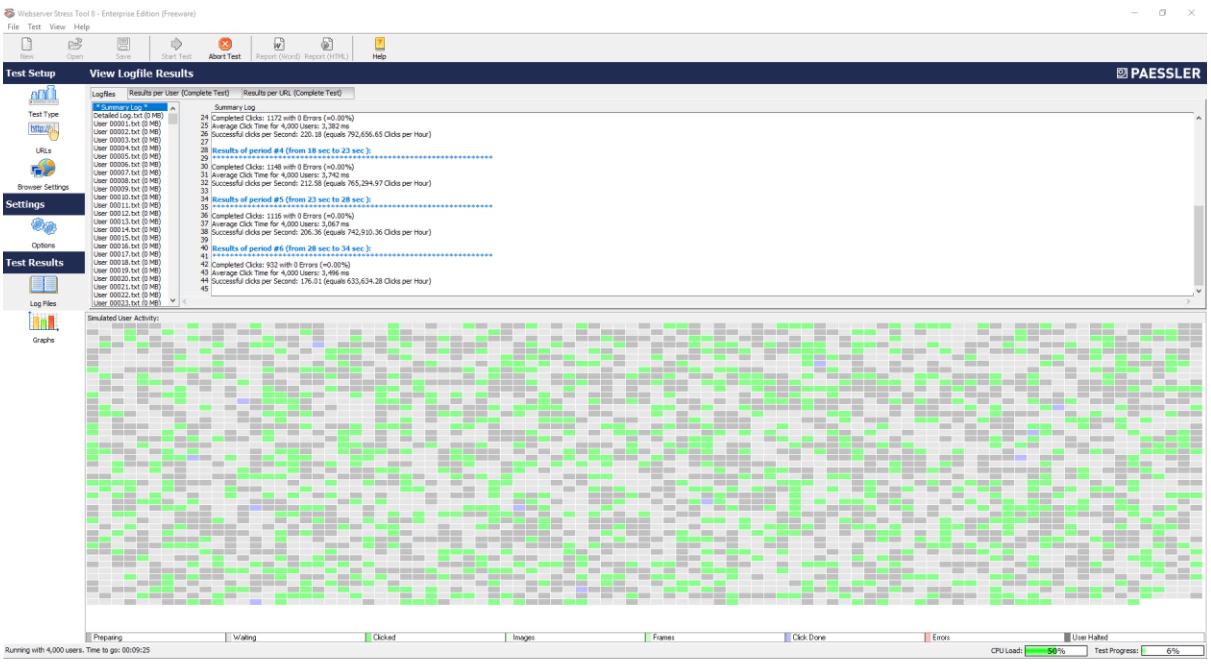
Since the tutorial zone was the easiest section to write and contains only markup, this had the least success criteria and all of them were met. It was also the easiest to test because every subsection could be tested during my black box tests using my survey. 86% of people agreed that the tutorials were easy to follow which is a very high percentage, and another 86% of people said that they liked the design of the website. This is the category that passed the most success criteria and I feel that it's a very strong addition to the site.

Fast and reliable database and back-end

Unlike all the previous success criteria, the next two are not broken up into subsections as they are quantifiable themselves. I wanted to have a fast and reliable back-end system and database to keep the website running smoothly. I also wanted to keep the cost down as I was paying for everything myself, so I chose a relatively inexpensive Digital Ocean Ubuntu Virtual Machine to run both the web server and the database. The peek usage on the site was during the black box tests with over 30 concurrent users, and the website was fast and smooth to use. In a production environment however, there could a much larger number of concurrent users which would be an issue for the cheap server. To solve this, I'd upgrade the hardware the website is running on if the website started to become more popular. This would be easy to do because of the way I've set up the deployment of the website. I simply run a script to install the packages needed to setup the server, then clone the git repo into the apache public directory.

To test how fast the current server is, I performed a load test on the website using a tool called Load Impact (loadimpact.com). This test lasted 10 minutes and had up to 135 requests per second. The minimum time taken to perform a request was 3ms and the average was 43ms. There were no errors. 43ms is a perfectly acceptable response time for the server, and shows that the back-end can cope with well over 135 requests per second. For this reason I consider this criteria met. I also used another tool called Webserver Stress Tool 8 as it allowed me to test up to 4000 concurrent users for 10 minutes, a much higher number than the original test. At 4000 concurrent users, the server was really starting to struggle. 0.5% of requests didn't complete due to a load induced error, and average request time went up to 4000ms. This is an example of a use case where I'd upgrade the server to make it faster and more reliable, however my project isn't going to get 4000 concurrent users, so this is more of an academic test. The results can be seen below:

Test running:

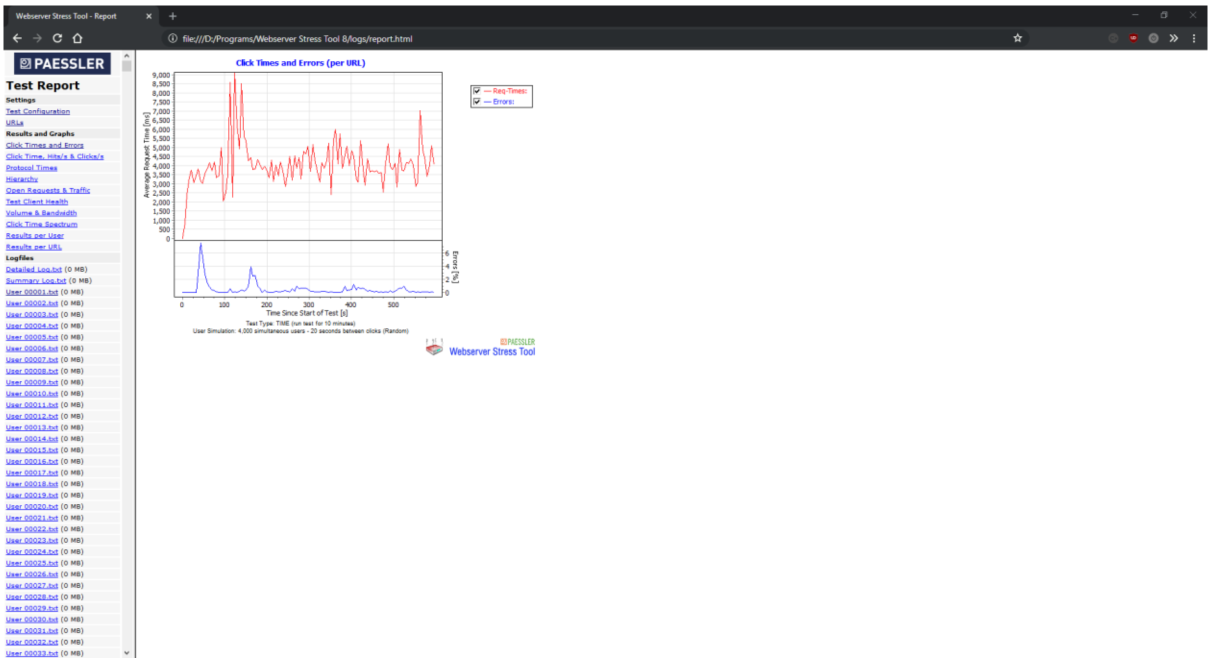


```
Detailed Log.txt (0 MB)
User 00001.txt (0 MB)
User 00002.txt (0 MB)
User 00003.txt (0 MB)
User 00004.txt (0 MB)
User 00005.txt (0 MB)
User 00006.txt (0 MB)
User 00007.txt (0 MB)
User 00008.txt (0 MB)
User 00009.txt (0 MB)
User 00010.txt (0 MB)
User 00011.txt (0 MB)
User 00012.txt (0 MB)
User 00013.txt (0 MB)
User 00014.txt (0 MB)
User 00015.txt (0 MB)
User 00016.txt (0 MB)
User 00017.txt (0 MB)
User 00018.txt (0 MB)
User 00019.txt (0 MB)
User 00020.txt (0 MB)
User 00021.txt (0 MB)
User 00022.txt (0 MB)
User 00023.txt (0 MB)
```

```
Completed Clicks: 1172 with 0 Errors (=0.00%)
Average Click Time for 4,000 Users: 3,382 ms
Successful clicks per Second: 220.18 (equals 792,656.65 Clicks per Hour)

Results of period #4 (from 18 sec to 23 sec ):
*****************************************************************
Completed Clicks: 1148 with 0 Errors (=0.00%)
Average Click Time for 4,000 Users: 3,742 ms
Successful clicks per Second: 212.58 (equals 765,294.97 Clicks per Hour)

Results of period #5 (from 23 sec to 28 sec ):
*****************************************************************
Completed Clicks: 1116 with 0 Errors (=0.00%)
Average Click Time for 4,000 Users: 3,067 ms
Successful clicks per Second: 206.36 (equals 742,910.36 Clicks per Hour)

Results of period #6 (from 28 sec to 34 sec ):
*****************************************************************
Completed Clicks: 932 with 0 Errors (=0.00%)
Average Click Time for 4,000 Users: 3,496 ms
Successful clicks per Second: 176.01 (equals 633,634.28 Clicks per Hour)
```

Running with 4,000 users. Time to go: 00:09:25

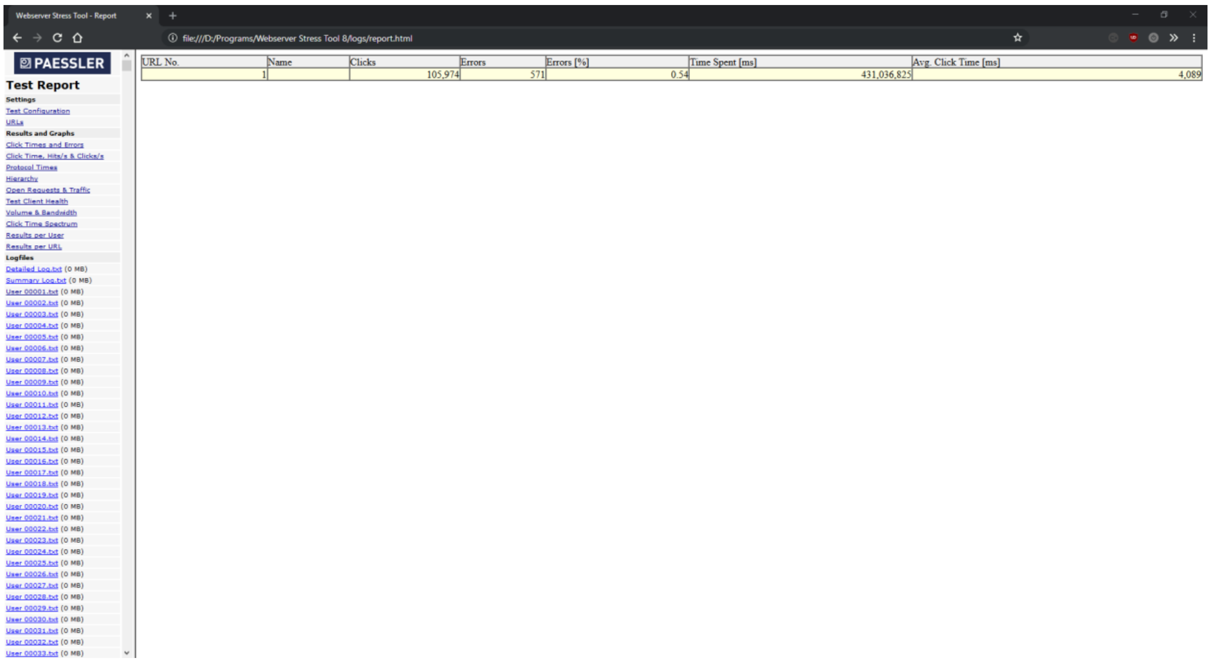
Test results:



Test Type: TIME (run test for 10 minutes)
User Simulation: 4,000 simultaneous users - 20 seconds between clicks (Random)

```
Detailed Log.txt (0 MB)
Summary Log.txt (0 MB)
User 00001.txt (0 MB)
User 00002.txt (0 MB)
User 00003.txt (0 MB)
User 00004.txt (0 MB)
User 00005.txt (0 MB)
User 00006.txt (0 MB)
User 00007.txt (0 MB)
User 00008.txt (0 MB)
User 00009.txt (0 MB)
User 00010.txt (0 MB)
User 00011.txt (0 MB)
User 00012.txt (0 MB)
User 00013.txt (0 MB)
User 00014.txt (0 MB)
User 00015.txt (0 MB)
User 00016.txt (0 MB)
User 00017.txt (0 MB)
User 00018.txt (0 MB)
User 00019.txt (0 MB)
User 00020.txt (0 MB)
User 00021.txt (0 MB)
User 00022.txt (0 MB)
User 00023.txt (0 MB)
User 00024.txt (0 MB)
User 00025.txt (0 MB)
User 00026.txt (0 MB)
User 00027.txt (0 MB)
User 00028.txt (0 MB)
User 00029.txt (0 MB)
User 00030.txt (0 MB)
User 00031.txt (0 MB)
User 00032.txt (0 MB)
User 00033.txt (0 MB)
```

More test results:


Webserver Stress Tool - Report

file:///D:/Programs/Webserver Stress Tool 8/logs/report.html

PAESSLER

Test Report

Settings
Test Configuration
URLs
Results and Graphs
Click Times and Errors
Click Time, Hits/s & Clicks/s
Protocol Times
Hierarchy
Open Requests & Traffic
Test Client Health
Volume & Bandwidth
Click Time Spectrum
Results per User
Results per URL
Logfiles
Detailed Log.txt (0 MB)
Summary Log.txt (0 MB)
User 00001.txt (0 MB)
User 00002.txt (0 MB)
User 00003.txt (0 MB)
User 00004.txt (0 MB)
User 00005.txt (0 MB)
User 00006.txt (0 MB)
User 00007.txt (0 MB)
User 00008.txt (0 MB)
User 00009.txt (0 MB)
User 00010.txt (0 MB)
User 00011.txt (0 MB)
User 00012.txt (0 MB)
User 00013.txt (0 MB)
User 00014.txt (0 MB)
User 00015.txt (0 MB)
User 00016.txt (0 MB)
User 00017.txt (0 MB)
User 00018.txt (0 MB)
User 00019.txt (0 MB)
User 00020.txt (0 MB)
User 00021.txt (0 MB)
User 00022.txt (0 MB)
User 00023.txt (0 MB)
User 00024.txt (0 MB)
User 00025.txt (0 MB)
User 00026.txt (0 MB)
User 00027.txt (0 MB)
User 00028.txt (0 MB)
User 00029.txt (0 MB)
User 00030.txt (0 MB)
User 00031.txt (0 MB)
User 00032.txt (0 MB)
User 00033.txt (0 MB)

URL No.	Name	Clicks	Errors	Errors [%]	Time Spent [ms]	Avg. Click Time [ms]
1		105,974	571	0.54	431,036,825	4,089

Secure back-end

This is hard to quantify as security vulnerabilities can come from anywhere. In my analysis I said that if no one had managed to gain access to my project during it's development I would consider this success criteria met. No one managed to gain access to my project, however during my black box testing, several people accidentally discovered a vulnerability that would have allowed anyone who knew what they were doing to perform SQL injection on any of the login pages. This is because I forgot to re-add the function to sanitize user inputs to my database class. This meant that when the testers were typing in usernames that contained apostrophes, it would end the MySQL query and then anything afterwards could be ran against the database. This is a massive security vulnerability, however as soon as I saw the feedback from the tests, I re-added the sanitization function to the database class to "make safe" all data entered by the user. Without this function this criteria wouldn't be met, however since I managed to add it so quickly after the test results, I consider this success criteria to be met.

Main objective

Overall I think I met my main objective with this project. All the components worked both individually and combined on the website, and all the feedback I got from my testers was positive. Breaking down the main objective, the project has proven to be both useable and intuitive based on the survey responses. I think the project has the potential to be called a viable alternative to StackOverflow, providing it had a big enough user base to be useful. While it does provide some social features (comments and questions), I don't think there are enough social features at the moment, so I would need to add a "community" section to the site in order to make it fully meet the "social features" criteria. The careers features however are my favourite part of the site and the thing I'm proudest of developing in the whole project. I think the careers part of the main objective is definitely met.

Addressing partially met and unmet success criteria

The biggest issue I had with this project was a not being able to foresee the implications of minor changes I made when I was first planning the project. I think this is due to looking at everything initially from a purely programming sense, assuming that I'd just spend the whole two years developing the project so I could be as ambitious as I want. This is because whenever I was thinking about new features, I was only thinking about what they would bring to the finished project and how long would take to develop. Since I've been programming for over five years, I saw changes such as adding an interactive map for job locations as a five line change instead of a brand new feature that would take hours to plan, design and write up. This is the main reason there are unmet success criteria left at the end of the project. With more time, every single one of my predefined success criteria would have been added, and possibly lots more than I couldn't even envisage when I was planning the project.

When I realised that I had to start cutting features to ensure I finished the project in time to prioritise my other A Levels which I found much harder than Computer Science, I ranked all the success criteria from most critical to least critical in order to prioritise the most essential features to make the best website I could. The main features that were cut were ones that would involve lots of designing and writing up but that didn't include much code. These were often features that would require making new pages or pages that didn't follow the standard layout I defined for the site, because these took the most time to implement (including designing and writing up). These were also the least critical features for the project's success.

Since I worked on the question and answer section of the project first, the first success criteria I cut was tagging questions. This is because the tagging system for the careers section of the project was written almost a year after I finished the question and answer system, and at the time I didn't know how to do some of the things I ended up needing to do. I also cut this success criteria because most people would write about which language their question is about in the question, meaning that the question search engine would pick up on this and provide a way of the user finding questions about a specific language. This is not a perfect solution and in the future I would port the tagging system from the careers section of the site to the question and answer section in order to meet this success criteria and provide a better user experience.

Reporting questions, comments and answers was the next success criteria I removed. This is because all the reporting criteria all ranked lowest on my list of priorities. I'm not saying that being able to report questions, answers and jobs are unimportant, because irrelevant or offensive content on the website will annoy users and could drive them away from the site. However this is a more long term issue to try and solve, so I added to it the list of features I'll add in future releases.

The largest feature I cut was the social features. This was highest up on my list of features to cut and I was disappointed not to be able to add this to the latest version of my project as I feel it would have added a lot to the website. This would be the first feature I'd work on in the future. To make this I'd replicate the process I used to develop the question and answer system and the career system. First I'd make a homepage for the community section, with links and descriptions to different parts of the community system. Then I'd code each one of these features, and finally integrate them into the rest of the website, such as being able to view someone's profile by clicking their username on a comment they have written. This would be a massive upgrade to the site, but also the most complicated system to write.

Part (C) – Justification of usability features

Success of usability features

Since my project is a website, usability was a very important thing for me to think about. This is because if people find things hard to use, or find using some of the features frustrating, they simply won't use the site. I designed my website from the ground up to include lots of usability features.

I split my the usability features of my project into five sections so that I could build features for each one. These are "learnability", "efficiency", "memorability", "dealing with errors", "user satisfaction". I will analyse the usability features and discuss how successful each of these were below:

Learnability

Learnability is a measure of how easy it is for users to use my project for the first time, and how long it takes them to learn how to use a feature before they are competent in it. To increase the learnability of my website, I spent lots of designing the layout until I came to the interface I used. This comprised of breaking the website down into it's main feature: question and answer, careers, and the tutorial zone. I did this to reduce the number of steps it took users to get to the features they want to use, and reduce the amount of irrelevant information they saw while they trying to get the features they want in order to stop them being overwhelmed. In order to quantify how well this design worked, I added two questions to the survey as part of my black box testing phase. These were: "Did you like the design of the website?", "Did you struggle to do anything on the website?", "Was the interface memorable?" and "Was it easy to use each of these features for the first time?".

The responses to these questions makes me think this design work worked and that the learnability features I added were successful. 86% of peopled liked the design of the website, 71% of people had no problems using any of the features, 71% also found the interface memorable, which is important because it shows that the interface is both unique and also well designed and understandable. Finally, 95% of people said that at least most of features were easy to use for the first time. This is an incredibly high percentage, and this also the most important survey question for learnability. Because of this I consider that the learnability features I added were very successful.

Efficiency

Efficiency is about how fast someone can accomplish something, and the amount of effort it requires. As with learnability, I didn't want people to have to click through several pages before finding the function or content they wanted to use. To achieve this I planned and designed a layout which broke up the site three groups of related content. As discussed in the learnability analysis above, the feedback from my survey shows me how successful this approach was. Efficiency is very important for my site because if people are using my site as part of their job for multiple hours a day (like StackOverflow), you don't want them to feel frustrated with the procedures they have to do to use your site. If they get too frustrated, they'll leave your site or service for someone else's. Due to the responses from my survey, I think the features I added to the site to improve efficiency were beneficial to the usability of my project, and therefore were successful.

Memorability

When someone comes back from a holiday, or from some time off, they want to be able to pick up exactly where they left off without having to relearn how to use all their tools again. Because of this I made several principles and features to help improve the memorability of the website. The main principle I made when I was thinking about how to make the website memorable was that the general layout should never change after the website went live for the first time. I stuck to this principle throughout the design and development phases of my project. This is because if you change major user interface features on a website, you force existing users to either relearn how to use the site, or drive them to away to a rival site. Neither of these are good for my project.

Memorability is a hard thing to quantify because you can only really test it after a period of a few months, but I added a question to the survey asking people if they thought they'd be able to remember how to use all of the features again after a break of a few months, and I had very good feedback. 71% of the people who took my survey said they'd be able to use every feature after a break of a few months, which tells me that the memorability features and principles I used were successful.

Dealing with errors

There are two things to think about when it comes with dealing with errors. The first is error prevention, where you try and stop the user from being in a position where they can cause errors, and the second is error handling, how you deal with errors if they occur.

Error prevention is important because you want to make the experiences of using your software as painless and easy as possible for the user. If they having to go back because they've done something wrong or made a mistake, they can become frustrated and may even eventually stop using your software. When I was thinking about how to implement error prevention in my project, I came up with three methods: clear and intuitive design and user interface, well labelled inputs, and only allowing users to enter necessary data.

Having a clear and intuitive design helps to reduce errors by making it really easy for the user to see how they can accomplish a specific task. As I explained above, 86% of my testers said they liked the layout of the site, and 95% of people agreed that it was easy to use at least most of the features for the first time. These are high percentages which show me that having a well designed website helped prevent uses from causing errors and making mistakes.

Labelling inputs well is important to help prevent users from causing errors because it makes it very easy for them to see what information needs to be entered where. If the inputs on my website were vague or confusing, people might enter the wrong data in the wrong place, thus causing an error. As I said above, 95% of people said they had no trouble using at least most of the features for the first time, which tells me that my inputs were clearly labelled.

Only allowing users to enter necessary data is a very effective way of preventing errors. A web form with 50 input boxes is much more likely to contain mistakes than a web form with 3 input boxes. I used this principle when designing my website to make sure than I only asked explicitly asked the user for input when it was needed. I developed approaches to assume that the user might want to do, or to default to predefined parameters whenever possible, reducing the chances of users making errors. For example, the question and answers homepage defaults to “hot” as the filter because that’s what the majority of users would want to do.

I also made lots of the inputs buttons instead of text boxes, because with buttons I, as the programmer, was in complete control of the outcome, instead of text boxes which could have infinite outcomes. This was a very effective approach which I utilised throughout the site.

User satisfaction

As I mentioned in most of the sections above, I believe a lot of user satisfaction comes from a mixture of a pleasing user interface and an efficient workflow. The pleasing user interface is proven by one of the questions on my survey: “Did you like the design of the website?”. Having a pleasing user interface is important to my project because it improves the user experience and makes them more likely to continue to use your site.

Having an efficient workflow was a harder task however. I achieved this by planning how users will interact with the site, and reducing the number of clicks they need to complete a task. The more clicks someone has to do to complete a task, the more time they are wasting and the more likely they are to become frustrated and eventually stop using your website. To reduce the number of clicks it takes to achieve a goal or complete a task I split the site up into three distinct subsections of related content, with links on the homepage and all the headers. For example, the careers section ran almost as a separate site to the question and answer system, which means that users trying to apply from a job don’t have to navigate around a section of the site that isn’t relevant for what they’re trying to do, thus making them more efficient.

The results of my survey suggest that this process was effective at increasing user satisfaction because 86% of my testers said they liked the design and layout of the website, and lots of the people testing my project said that they liked the website and enjoyed using it.

Part (C) – Maintenance

About half way through this project I realised that most of the code I’d written was poor quality, hard to read, and hard to maintain. This led me to one of the most important release of this project where I simply optimised and improved all the code for the project, using best practices and added new tools to the workflow. This re-write made the project modular, which is very important for making a maintainable project. In this section I’ll discus the steps and changes I made to make NullPointerException easy to maintain and develop in the future.

One important thing I added to the project is an easy way of scaling up the server that the project runs on to accommodate more users. This is thanks to using Linux and Vagrant which allowed me to create a single ruby script with the install commands needed to setup both the development and production environments needed to run my project, including fetching the latest version of the code and the passwords from the remote repository hosted on GitHub as part of the setup. This is an important feature to have in order to make a project easy to maintain because it works exactly the same on the £4 per month Ubuntu virtual machine the project is currently hosted on as an enterprise grade server. It also does this all automatically which gives the programmers more time to develop the actual features.

Another feature I added to the project was the addition of several core classes which can be included by any script running on the website which contain lots of generic functions that are used throughout the website. For example in the core.js JavaScript script there is a function used to wipe an HTML table (passed by reference as an argument) but leave the table headings. This is used by every page which has uses an HTML table to display the of a database query (such as job listings or question searches). This is important for the maintainability of the project for two reasons. The first is that all the functions that I've already written don't need to be re-written which saves developers time and reduces the size of the project, thus reducing page load time. The second is that it makes it very easy for future developers to add their own classes and functions because the project has been designed to use them.

The project is also very easy to deploy, since it's hosted on a remote git repository (at GitHub). This makes it easier to deploy because they developers can simply SSH into the production server and run a "git pull" command. The whole git/GitHub infrastructure makes deployment easier by allowing developers to work on features using their local machine without effecting the actual code stored in the remote repository, then uploading the changes when a feature/bug fix it complete. git also allows you to revert changes if they cause an unexpected bug or conflict in production. Easy deployment is good for maintainability because developers don't have to set up anything, they can just code, commit, and then pull the changes onto the server when they're ready.

Using containerised development in the form of Vagrant was major step forward in terms of maintainability. This allows developers to have a local version of the project on their local machine which runs in an exact replica of the production server (maybe with some additional debugging tools), that is also identical for every member of the team developing the software. This means developers working as part of a team won't have errors where code runs on their computer but not on someone else's, or not in production. This is incredibly useful for maintainability for several reasons. Firstly, it's already been set up. The ruby script Vagrant uses to create the container has already been written and added to the repository. Any developer working on the project just needs run the "vagrant up" command and Vagrant creates a container identical to anyone else working on the project. Secondly, it means everyone is working on exactly the same machine, so they can have complete faith that every feature that works on their computer will work on everyone else's and in production. Finally, thanks to version control, if a programmer makes changes to the Vagrantfile, everyone else's will apply the same changes whenever they pull from the repository, ensuing everyone is working in the same environment.

When I was developing the project I was very careful about variable and function names. They have to be descriptive and relevant, but also short enough to make the code readable. Sensible and well thought out variable and function names make the job of maintaining and developing the project much easier as they make it a lot easier to understand how the code works and what it actually does. Along with comments, having good variable and names are standard practice when developing any kind of professional project, regardless of the size of the project or development team.

Adding git and GitHub to the workflow for the project was essential for ensuring that the finished product is maintainable. They provide a powerful way of collaborating with other developers working on the same project. These features include issue tracking, pull requests, a comments system for discussions and hosting the source code. These are all very important for maintaining a project because they allow developers to work together and communicate. They also provide tools for tracking changes and showing which developer has done which change, features which I'll go into more detail on below.

Probably the most basic and important feature I added to aid maintainability was adding comments to every file in the code base. Comments are a way of a programmer explaining what their code does, inside the source file. Anyone working on the same file can see these comments and use them to deduce what code is doing, without needing to analyse and understand all the code or function. It also shows what and how the programmer was thinking when they wrote their code. This can be useful information when trying to fix an obscure bug. Comments are essential for creating a maintainable project.

The penultimate thing I did to increase the maintainability of my project was using git and GitHub to track changes in the code and even for project management. Tracking changes makes the code more maintainable because it shows every single revision every programmer has made to the project. You can see which lines have been added and which line have been removed, when changes were made, and a description of every change. This helps programmers keep track of how the project is progressing, and helps them understand why certain decisions were made and how they affect the rest of the project. It keeps a permanent record of every change made to the project which means developers can revert back to any older version in the event of an error. It also features branches which allow different people to be working on different features at the same time, without interfering with each other's code. For example, I made a "production" branch for NullPointerException which contains the current code build that's running on the live server. When code is ready to be deployed you can merge the changes into the production branch and they'll get pushed to the live server. All of these features make the project more maintainable.

The project management tools on GitHub are also important for making the project easier to maintain. These include issue boards, milestones, task lists and releases. Issue boards are used so that developers can keep track of every single bug they discover, and a way of ensuring that people don't try and fix something that's already been fixed due to communication errors. Milestones are used to add time limits to a set of features so that everyone can see the progression of a build or update. The fact that all these have already been set up make the task of maintaining the project a lot easier. Releases are exactly what you'd expect, binary builds of the project when it hits predefined milestones. This is useful for keeping track of builds, especially when using an agile methodology which involves regular builds.

The final aspect of the project designed to make it more maintainable is that it's open source. This means that anyone in the world can access the source code for free and contribute to the project. This is a massive boost for maintainability because it allows a huge team of people working for free in their spare time to help maintain the project. I chose to make this project open source because it helps with maintainability and I also agree with and support the principles of open source development and collaboration. Opening a project up to other developers allows more people to work on the project while also cutting the costs down. More people developing the project means features reach production quicker, improving the site for everyone. It also allows for multiple people to give their opinions on the direction they feel is best for project. This all means that the project is easier to maintain and also better and more polished for the users.

Section (E) – Limitations and remedial actions

Limitations

The main limitation in my finished project is the lack of the social section of the site that I originally planned to do. Although it ranked lowest in my list of priorities list and therefore was the main section to be cut, it would still be a major addition to the site. I was very disappointed when I had to cut this feature, because I'd done all the planning and designing for the more important features of it and it felt like a waste to not use it, however I'll definitely be adding it in the future so this planning and design won't go to waste.

The main reason I cut the social section from my site wasn't because of the time to develop the features, but the complexities it would have added to the write-up process. It would have also required lots of new pages adding which all have to be designed and developed. This is a time consuming task as designing front-end pages is not my speciality, and takes me a lot longer than writing some PHP back-end functions.

The lack of some social features on the website shouldn't be imminently noticeable to new users as they won't have seen my original plan. However, as they become more familiar with the site they might feel that some features are missing. However as the social section is the top priority post-release, by the time users start to feel like the site is lacking some of theses features, the social system should be released.

Another limitation of my project is the lack a moderation and reporting system for questions, comment and jobs. The reporting system would involve adding small buttons next to each question, comment and job listing that would take the user to a page where they can report the question, comment or job listing to the sites administrator or moderation team along with an explanation of what they were reporting it for. Further details of this system and how it could be implemented are in the remedial actions section below.

Features missing from careers system (google maps, applying for jobs, etc.)

The careers system didn't have any features cut, however there are some features users might want added in future updates of the site. These include an interactive map showing the location of a job, a button employers can add to their job listing which redirects the user to an application form on the employers website, a system where users be notified via email whenever a job relevant to them is posted. These features would enhance the careers system and make it more useful for the users. Further details of these changes and how I'd go about implementing them are in the remedial actions section below.

The final missing feature that I feel is a limitation to the site is the lack of a tagging system for the question and answers section. The only reason this wasn't added in the final release was due to time constraints, since all the code had already been written for the careers section. The search for questions feature does all searching for specific languages etc. so no functionality is lost by not including the tagging system, but some users might want it for convenience. I'll go into detail on what it would take to implement this in section below.

The last limitation is completely out of control. For a website such as mine to work I would need a high user-base from day one. My site completely relies on user driven content, and without the users it's basically just a static webpage. For my project to ever take off, there would have to be enough initial interest from developers and employers to get the site off the ground. From then on however it should continue to support itself. I'll talk about how I'd attempt to overcome this in the section below.

Remedial actions

Most of the new remedial actions needed to address the limiting features are relatively simple. This is because they're all changes that I didn't have time to add the final project, but they have all been planned and designed. This means that should be straightforward to implement.

The first set of features I'd add would be a social/community section. This would include user profile pages, direct messaging between users and voting on comments/answers. Most of the work on this section would involve the profile page in some way, so the first thing I'd do would be to start work the user profiles. There would need to be two types of profile pages, one for viewing someone else's profile, and one for viewing your own profile. The reason you'd need separate pages is because your own page would need to be customisable and would therefore contain lots of edit buttons and input boxes. My initial design for profile pages can be seen in the design drawings section of the Design document. I'd create some new classes and functions in the core.php file in order to keep the website modular for maintainability.

After developing the profile page I'd need integrate it with the other sections of the website. For example, the usernames next to the comments on questions would need to link to the user's profile page when clicked. The profile would also need to add up the total number of votes given to a user to produce a score which would be displayed next to the user, as an incentive to answer more people's questions.

Finally I'd add direct messaging. This be accessed by a button on a user's profile that says "Contact this user". Clicking this button would redirect the user to a real time messaging page. This would be quite simple to develop. It would work on the same principle as all the other pages that send and receive data to and from the database, but would make an asynchronous request every second to check for new messages, and download them if necessary.

After implement the community/social features, I'd port the tagging system from the careers section to the question and answer section. This would be a very easy change to make, because all the code has been written and due to the modular way I've developed the code, I'd just have to copy over the layout and include the relevant JavaScript and PHP scripts.

The next missing feature is an interactive map showing the location of each job on job.php. This would be done using the free (at time of writing) Google Maps API which works by including the JavaScript API in a <script> tag, specifying both your API key and the callback function you want to create the map using. This callback function needs to first create a JavaScript object containing data about the map such as the zoom level, longitude and latitude you want the map to be set to, then it needs to create the map using the google.maps.Map() API function, specifying the <div> you want to create the map in, and the JavaScript object with containing all the arguments. Since the PHP script that runs on this page already downloads the location of the job from the database, the only difficulty would be turning the location returned form the database into a latitude and longitude. This can also be achieved using the Google Maps API.

Another notable missing feature which I could remedy is the lack of a dedicated button to apply for a job. This could be fixed by giving the employer an option to add a URL for the application page on their website. I would then use this URL to generate an "Apply for this job" button which would link the URL they specified. This would be a very quick fix.

An important feature that didn't make it into the final release is a reporting/moderation system. This would be important as the user base grows, as if people see content that isn't relevant or offends them, they might stop using the site. The reporting system would work by including report buttons on questions, comments and job listing. Since comments are quite small, the user would have to hover over the comment to display the button. On mobile the same would be achieved with a long press on the comment. When the user clicks the report button, it will redirect them to a report page that contains a compulsory input box asking the user why they reported the question/comment/job listing. When the user has described why they reported the content, a script will add the report to the database along with the id and type of content that was reported.

In order for the site administrator and the moderation team to be able to see and judge the reports, I'd add a moderation page to the password protected admin page. Moderators would log into this page and then see a table (similar to the table on qa.php) containing a list of reports. The table would have filter buttons for things filtering out reports that are in progress or marked as complete. When the moderators click on one of the reports in the table, they would see both the reported material (comment, question or job listing), as well as a link to the material and the user's report. There would be two buttons at the bottom of the page, one for removing the content from the site, and one for deleting the report (if the material was ok).

The last limitation is not a missing feature and is also the hardest limitation to remedy. As I described above, the biggest issue my website at launch would be the low number of users. All the content on my website is entirely user driven, and without any users there won't be enough content on the site to either drive new users to the site or keep existing users using it. While I can't directly control the number of users at the start, Search Engine Optimisation, a domain name and some advertising would be a good start.

Search Engine Optimisation is when you deliberately make your website more appealing for search engines, meaning you rank higher on search queries, driving more traffic to your website. I can do this using techniques such as adding <meta> tags to the website with keywords that I want to redirect users to my site, and by having as many links to my website on other websites to increase its Page Rank score.

Having a domain name is something I only didn't do due to cost. A domain can be upwards of £20 a year (and much more depending on how desirable it is), and this was too much for me to spend on this project, however I would definitely get one in the future. Having a domain would help drive users to the website because it would legitimise it. It would go from a small project to a proper website in most people's eyes. People can also be hesitant about connecting to websites without domain names because they are often scam sites.

Finally, paying for some adverts could help drive traffic towards my site. Again this is something I didn't do due to cost, but I would definitely do in the future.

Conclusion

This has be an incredible project to work on. The sheer scale of what I've achieved is bigger than anything I've ever done in the last five years of programming. If I was doing this project again there are lots of things I'd do differently, such as using git GitHub and Vagrant from day one which would have saved tens of hours of development time. I'd also use an Object Oriented style of programming as the modularity it provided has helped me in lots of different ways since I adopted it.

Overall I think I met enough of my success criteria to class my project as a success and I'm very proud of my final outcome and what I've learned along the way.