Blog Central Coursework Report

Alex McGill

40276245@live.napier.ac.uk

Edinburgh Napier University - Web Technologies - SET08101

1 Introduction

The brief of this assignment was to build a blog platform. It should provide the user with an interface that can allow them to add, view, edit and delete blog posts. The blog posts must be persistent and have a server that provides specific data to the user interface using a CRUD API to utilise the blog features. The blog platform should have additional features to provide more functionality and usability for the user.

I decided to create a blog platform that can have multiple user accounts and allow anybody with an account to create blog posts. The name of the blog platform is 'Blog Central', as it is a hub for different types of blog posts where users can look up articles they are interested in. Once the user posts an article they become an author and are featured on the authors page. The homepage displays articles from all categories, sorted by most recently posted. If a user wants to see a specific category of blog post then they can select a category from the drop down which displays the various categories. All of the data for the website is stored on MongoDB using mLab[1] and is retrieved dynamically by the server.

When first researching CRUD API's with Node.JS I found a very informative article[2] that breaks down how a simple implementation works on Node.JS step by step. This was a great first step which lead me onto YouTube. To better understand the capabilities of Node.JS I followed along a few YouTube tutorial video series', with 'Node.JS & Express From Scratch'[3] having the most impact on the structure of the blog platform. I found that this was a solid foundation of knowledge for my blog platform and allowed me to get the initial stages of development completed while learning at the same time. Although helpful I found that the tutorials are fairly basic in terms of functionality, sticking only to core features and needed to be greatly expanded upon for the site to properly fulfil its purpose.

2 Software Design

Before starting the implementation of the website I planned out what the key pages were should look like. The homepage should be a feed of articles from every category, with the most recently posted article at the top to keep content fresh. The concept for the page can be seen in Appendix A. The nav bar at the top of the page allows the user to easily switch between pages of the website. Observing what other blog sites were doing I realised that the ones with images always looked the best and made the content far more attention grabbing. Due to this observation I decided that every blog post to the website requires a thumbnail that is displayed along with a short text preview of the article. The thumbnails should all be exactly the same height, but to stop aspect ratio scaling issues where images are stretched, the width is to be automatically set within the limits of the blog post content section.

I felt that it would be important to include the author of the article on not only the actual article page but even the article previews so that readers can become familiar with specific authors. This means that if they've read an article from that author before that they enjoyed, it's likely they would read certain articles knowing that they are written by the same author. The author's name should be clickable, redirecting the user to the authors profile where they can see all of the posts that the author has made. The design for the author profile page is shown in Appendix B. This means that every author has their own blog with just their posts, as well as there being a collection posts from every author.

Once I had designed the author profile page I realised that it would be convenient for the user if there was a page dedicated to displaying the different authors. This means that the user wouldn't have to go back and find the article to go to an authors page, instead they can just select the author from the list. The initial design for the authors page can be seen in Appendix C. The idea was to make it as simple as possible to quickly access a profile, so it should have large clickable profile pictures along with author names associated to them. In a future where it is possible there are lots of registered authors on the blog platform, it would become appropriate to include some kind of author search feature but as it is on a smaller scale it would be unnecessary.

The actual article page is similar to the post preview that is listed on the homepage but only includes the selected post content. The title and thumbnail are displayed at the top of the blog post, with the blog content following. This can be seen in Appendix D. The blog content should be able to include images, videos, audio etc. making the blog as engaging as possible to retain user attention. After the design concept was created I decided the blog post category should also be included in the footer of the blog post in line with the author and date. This would be clickable so that after reading the article, the user can select click on the category and find similar articles.

After having most of the core design concepts completed I was ready to start an implementation but first I had to figure out the website navigation hierarchy. The final version of the website navigation diagram can be seen in Appendix E. The 'Articles' tab on the navigation bar should have a large drop down menu with all of the available categories displayed.

When a category is selected the user should be brought to a page with all of the articles within that category. A user should be able to edit a page if they are either logged in and the author of that post or they are an administrator of the website.

To store the data from users, blog posts and authors in the MongoDB database I needed to create a schema for each object. I noted the key bits of information that would be required for each object such as first name, last name, email address and then combined them into separate models. Each model is stored in its own collection on MongoDB so that all the users are stored together, all posts together and all authors together. An extra variable I decided to include in the user model was the boolean 'isAdmin'. If it is true, then the user is an administrator and will be able to edit / delete any blog post and can edit any author profile. It can only be set by manually from the database to ensure that a user doesn't find a way through the website to give themselves administrator privileges.

3 Implementation

One of the first implemented features of the website was the navigation bar. By placing the navigation bar in a layout.pug file, every other web page on the site has the exact same bar at the top of the page. The navigation bar items change, dependant on whether the user is logged into the website or not. The difference can be seen below when comparing Figure 1 and Figure 2.
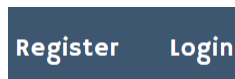

Figure 1: **Navigation Bar Section** - Logged Out

Figure 2: **Navigation Bar Section** - Logged In

By switching the navigation bar items dynamically depending on the user being logged in, there is a reduced amount of wasted space as they are only visible when needed.

Before the user can login (shown in Appendix F) they need a way to create an account. The 'Register' item on the navigation bar takes the user to a sign up page which can be seen in Appendix G. The registration form has input validation to make sure that all of the data inputted by the user meets the requirements. A key feature on this page is checking whether the username or email address already exists in the database. If this wasn't checked beforehand then there would be issues logging in with incorrect details linked to different accounts. Before the passwords are stored in the database they are hashed using 'bcrypt'[4] to ensure that they cannot be accessed in plain English text. This was a very important feature to ensure the security of user accounts and trusting that their information is stored safely.

The website homepage is populated with recent articles in the centre, but on the right side column of the page there is a 'Suggested Article' section. This can be seen in Appendix H. The server randomly chooses an article from the list of posts excluding the top page results and suggests it to the user. This is a convenient feature especially as the website grows and there are more articles posted as it would be easy to miss one that they may have wanted to read. The post previews that are used on the home page show the first 45 words of the article, giving the reader just enough content to decide whether it is worth continuing to read on.

The search page allows the user to search for a tag and match it with any articles that have that tag assigned to them. This is shown in Appendix I, where the search term 'Jupiter' is entered, and any articles with a 'Jupiter' are displayed. To ensure maximum usability, all tags and search queries are converted to lower case to avoid case specific matching which could lead to articles being left out.

When a blog post preview is clicked on, the full post is opened onto a separate page. Unlike other pages, on the full article page there is a 'Related Posts' section on the right hand side. This shows all of the blog posts that have similar tags assigned to them, filtering out any others. This is another convenience feature that aims towards getting the reader to visit more blog posts and staying on the website longer. The blog content and related articles section can be seen in Appendix J. In the example, the article is about the Tesla Model S. In the 'Related Articles' section, there are other articles about Tesla and the Model S.

Before a user can submit a new blog post it needs to meet all of the requirements. For example, the thumbnail must be a valid URL and the title cannot be more than 60 characters. The requirement enforcement is shown in Appendix K. Any fields that are not valid are highlighted red and a small pop up error message is displayed in the input for that field.

The authors page shows all the users that have posted articles, as they are now considered authors. This is shown in Appendix L. The idea when designing this page was to have it look like cards, with the rounded edges and simple format of each author. Clicking on the 'card' sends the user to that author page, as shown in Appendix M. On wider displays, the layout can expand allowing more authors to be displayed on the same line. The author profile is customisable by the author, where they can change their own profile picture and bio. This can be seen in Appendix N.

4 Critical Evaluation

In comparison to the requirements set in brief, I have met all of them. The CRUD functionality was a core feature of the site and so it was one of the first features to be implemented, even before a proper user interface. The website also has many additional features, such as user accounts, author profiles, search functionality, hover animations + colour changes, mega drop-down menu (Appendix O), related articles, categories, page scaling media queries, administrator accounts and more. Considering all of this I think that the website has gone beyond the core learning of the module

and exceeds the blog framework requirements, leading to an overall rewarding user experience.

When comparing the blog platform to other competitors available online, I think that for the most part it holds up against them. There are a few standard features that are not implemented that I presume users would expect such as a comment thread for each blog post where they can share their opinions on the article. Typically the comment sections are linked with Facebook accounts, giving the user easy access and the website more information about the user. Despite some competitors providing trending or popular articles, I haven't seen a 'Suggested Article' feature implemented before. In practise this could lead to outdated articles being pushed to the front page but with the right filters and logic applied I feel this could be a convenient feature.

Having a trending articles section would be another key difference in comparison to the top blog platforms. This allows more users to see the best articles and bring more traffic to their site. To implement something like this, each blog post would need a view counter that would require some spam prevention to ignore cases where users spam the refresh option on the page. This could potentially be in the form of unique IP address views on an article.

Possible improvements could include implementing a safeguard / filter in the add article content input section. The input allows authors to add HTML into the blog post, giving them freedom to style their posts and the content within it the way they like. However, I feel like this could potentially allow certain users with the knowledge to inject some malicious HTML that would alter the behaviour of the blog content. One way of overcoming this would be to have a user interface on the content input where they use tools to configure the post for them, for example centring an image or text by selecting a 'centre' option rather than reading in any HTML input.

Another improvement that would be important for administrative purposes would be database triggers. This would leave an audit trail when anything happens - for example, if a rogue admin deletes multiple blog posts and it is unclear who deleted them then the trigger would log each event and would show exactly what happened, who did it and when. Currently these actions are logged to the console but a permanent, uneditable database collection would be a suitable place for this.

One improvement I would also make would be splitting the pages with multiple articles into multiple pages. Currently all of the articles are on the same page which means they scroll down quite far and load more data. It seems to be common practise to split up the search results after a certain amount of posts into multiple pages, with the user being able to switch between each page and access the articles on that page. This would become more important as the site grew more popular and there were hundreds of posts being loaded at the same time rather than smaller chunks being served as required.

5 Personal Evaluation

I think I performed well in this assignment because there are lots of additional features implemented and they all work as intended. Implementing all of these features took a long time and a lot of effort. Early on in development it was a bit overwhelming having written out a huge broken down to-do list of things that needed fixed/implemented and making slow progress but the more familiar I got with Node.JS and MongoDB the quicker I was able to solve the different problems that were faced.

One of the big problems I was facing was the asynchronous nature of Node.JS in relation to waiting until after a MongoDB query had returned to do the next task. This lead me down the slope of callbacks within callbacks and huge functions that should really be split into separate functions. The typical problem was I would write out the function and then realise that an empty array was being passed to the res.render display layout because it was being called before the MongoDB query returned the results of the array. The next steps were adding callback functions and if after an hour or so I still hadn't figured it out I would leave it and come back to it later on in the day. For whatever reason I would then upon returning to the problem quickly find a solution which usually involved setting the array in res.render equal to a multitude of function calls that returned an array from the database.

One problem that wasn't actually a problem at the time but could potentially be exploited in the future was that in the blog post previews, HTML line breaks ($<$br $/>$) were being added normally. This meant that previews could be longer than they should be and what I realised was that if somebody wanted to then they could post multiple articles with a long spam of HTML line breaks and the articles preview listings would take up way too much of the screen than they should. The solution I found for this was getting the text in the post preview and removing any occurrences of '$<$br $/>$'. This did not impact the actual content of the post at all once the user had it opened but it meant that all post preview heights were limited within the word count and character restriction.

As this project had a lot to it I spent quite a bit of time reading over relevant threads and forum posts online about the different problems I needed to solve, gaining a better understanding at what was possible and how others went about solving them. Having that further understanding allowed me to incorporate my own implementations and approaches towards developing the blog platform.

I learned a lot during this assignment, going from a very basic understanding of Node.JS to creating an entire blog platform within a limited time frame was definitely a big learning experience full of surprises. The biggest point that I took away from this project was stepping back from a problem and going back to it later, rather than working on it until it is solved. I found that often when I revisited the problem I looked at it a different way and the solution was more clear. I also realised just how complex huge social media sites, such as Facebook and Reddit, must be to achieve serving endless amounts of dynamic content among millions of users daily in comparison to my relatively simple blog platform.

6   Node.JS Packages

I used multiple Node.JS Packages throughout the development of my blog platform. They are all listed on Github in the packages.json file within the main 'blog' directory.
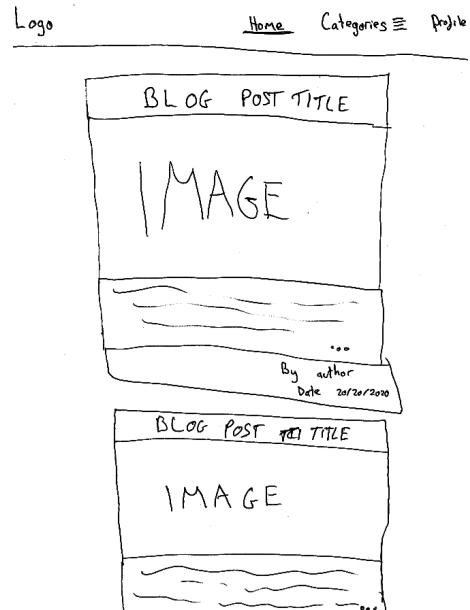
- bcryptjs
- body-parser
- connect-flash
- express
- express-messages
- espress-session
- express-validator
- mongodb
- mongoose
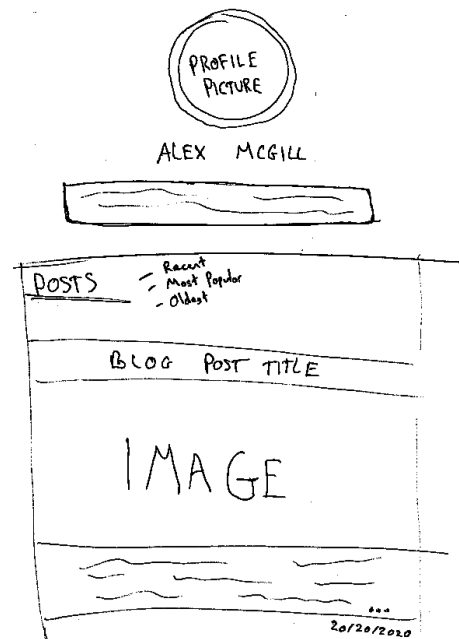- passport
- passport-local
- pug

References

[1] "mlab mongodb host provider."

[2] S. Domes, "Build a node.js api in under 30 minutes," Jan. 2017.

[3] B. Traversy, "Node.js & express from scratch," May 2017.

[4] D. Wirtz, "bcrypt.js package."

7   Appendices

A   Home Screen Sketch

B   Author Profile Sketch

C   Authors Sketch
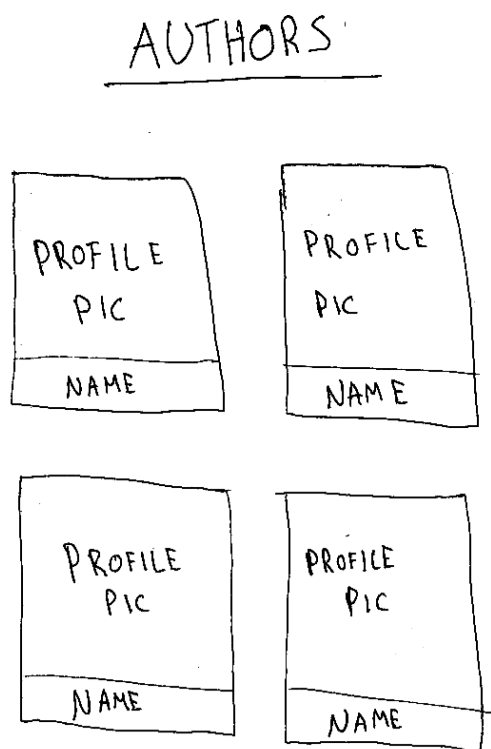
D   Article Sketch
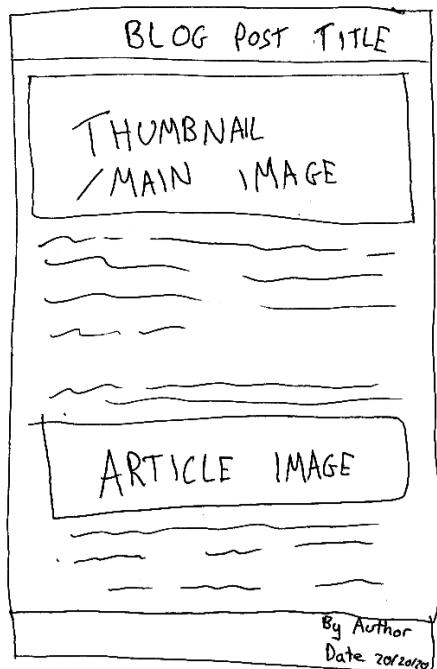
E   Website Navigation
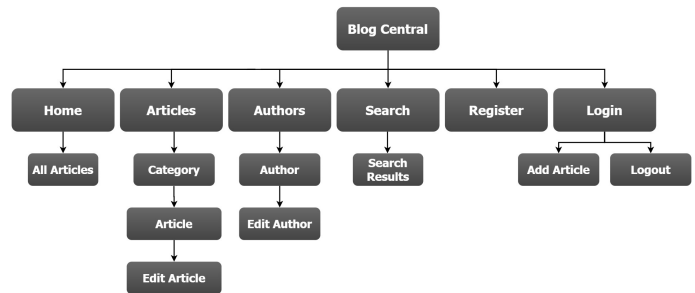
F   Login Form

G   Register Form

H   Home Page

SUGGESTED ARTICLE

Tesla Model 3 Sales Boom As Production Efficiency Increases

Elon Musk Test Drives The Tesla Roadster 2

Model 3 sales are now the current market leader in new vehicles. Movet timeam an pri, sed ei urbanitas complectitur. Id sale homero legere usu, eum at mollis imperdiet consequuntur. Commune fastidii ad vis, at solet munere oblique cum, nibh definiebas ad nam. No errem...

Category: Business   By Michael Scott | 2018-4-8 16:06:02

Startup Company 'Pied Piper' Is The Future Of Silicon Valley

I   Search Page

Showing search results for 'jupiter'

Elon Musk Has Purchased Jupiter Again

It was announced yesterday that Elon has bought the rights to Jupiter. Lorem ipsum dolor sit amet, maluisset salutatus ea mei, id vim erant tincidunt instructior. Probo ornatus partiendo mei eu, nonumy partiendo democritum mei ei, mollis iisque eam no.

J   Article Page

Inside the Model S: A Closer Look

RELATED ARTICLES

The modern interior of the Model S is unlike any other.

Lorem ipsum dolor sit amet, maluisset salutatus ea mei, id vim erant tincidunt instructior. Probo ornatus partiendo mei eu, nonumy partiendo democritum mei ei, mollis iisque eam no. Quod aeterno malorum mei cu partem mandamus. Sit causae iriure at, qui habeo oblique epicuri eu. Mutat facilis id pri, ea qui vocibus volutpat, eruditi noluisse per te.

Ea cum quem quod tincidunt, mea latine efficiendi persequeris cu.

Elon Musk Test Drives The Tesla Roadster 2

Tesla Model S Performance Test

Tesla Model 3 Sales Boom As Production Efficiency Increases
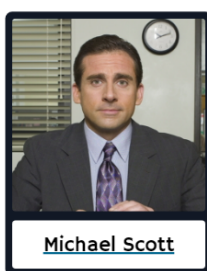
K   Add Article

Title: Title

Thumbnail: https://i.imgur.com/IgoYptY.png

Content:

Tags: Seperate tags with com...   ⚠ Please fill in this field.

Category: Business ▾

Submit

L Authors Page

M Author Profile

N Edit Author Profile

O Screenshot