Blog Coursework Report

Richard Borbely
40283185@napier.ac.uk
Edinburgh Napier University - Web Technologies (SET08101)

1 Introduction

This report describes the design and implementation stages of a public travel blog platform using **HTML**, **CSS**, **Javascript** and **NodeJS**. It is made up of various server and client side elements. The client element presents a user interface enabling multiple users to interact with the website but restricting certain functions to an administrative user. Server elements provide a create, read, update, delete (CRUD) API and persist data in a database. Sources of information gathered for this project include: Beginning Node.js by author *Basarat Ali Syed*, Pro Express.js by author *Azat Mardan* and other online forums.

2 Software Design

Generating a PUG template with Express will provide a strong framework to start off with, this implementation will use dynamically generated HTML pages that are reusable. For example, any number of posts are going to use the same template, making it very efficient. Each page will fork off of a layout page that is going to have a title bar and a navigation bar implemented. Planning what routes the server should handle is the next important step. The routing diagram is shown on *Figure 1*.

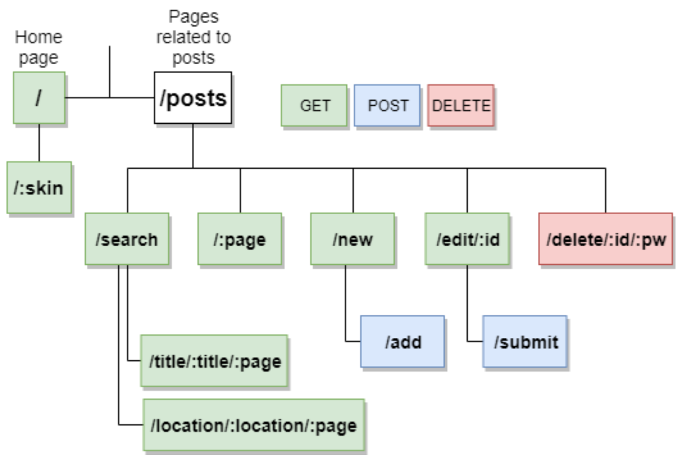

Figure 1: Routing diagram

For easy manipulation of front-end elements a CSS and Javascript file is going to be attached to the main layout page alongside with a Jquery library for additional functionality.

3 Implementation

Navigation Bar The navigation bar is located on the left side of the page, it provides the main means of navigation. When the page is scrolled and the navbar reaches a certain location, its position is changed to 'fixed' using Javascript therefore it will scroll with the page and stay visible at all times. The two rectangles next to it enable the user to switch between a light and a dark themed look of the website.

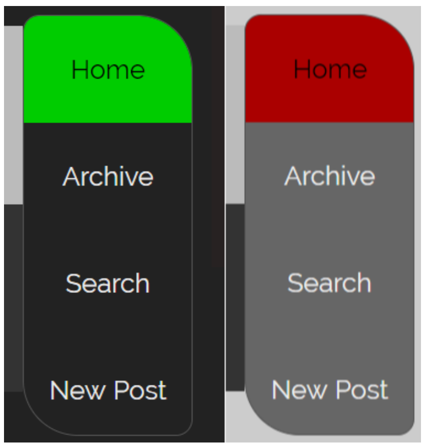

Figure 2: Navigation bar

Home The home page is dominated by a welcome message and a display box that cycles through the images available on the website. The displayed images will always be unique.

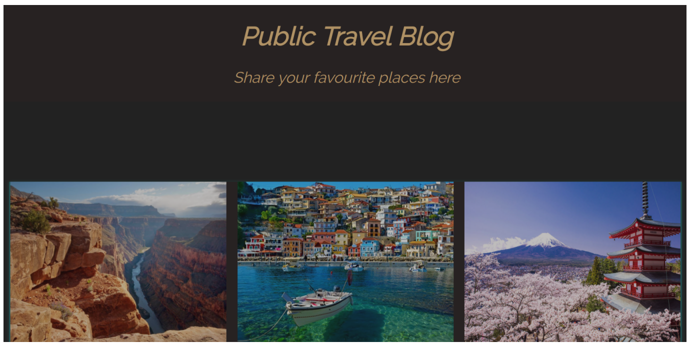

Figure 3: Home page

Archives The contents of the archives are displayed on multiple pages to make browsing convenient. An additional menu is shown below the navigation bar with the available page numbers. This feature has been implemented in the PUG templates using a for loop that will iterate through only the posts that need to be displayed on the current page.

```
// Display 5 posts on a page

 let from = (postCount−1) − (5 ∗ (pageNumber−1))
 let to = from − 5;
 for (let i = from; i > to; i−−) //iterating backwards on the↩
    data, so the latest post will appear first

```

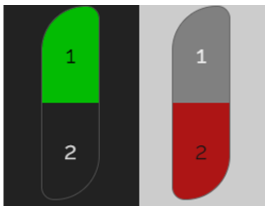

Figure 4: Pages menu

Editing and deleting posts is available with the icons next to the titles. Editing will redirect to the new post page with the form being filled out with the post data. Deleting a post requires a password.

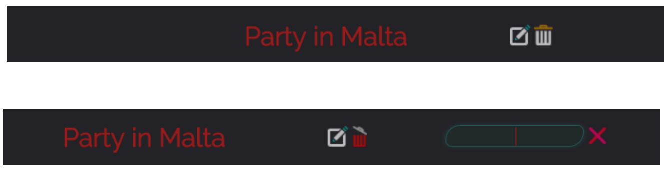

Figure 5: Editing and deleting a post

Search Looking up posts by their title or location is available in the search menu. If the search results in multiple posts, they will be displayed in the same way as in the archive.

New Post Creating a new post is done by filling out a form, it is required to fill out the title, location and description fields. Uploading an image to is optional.

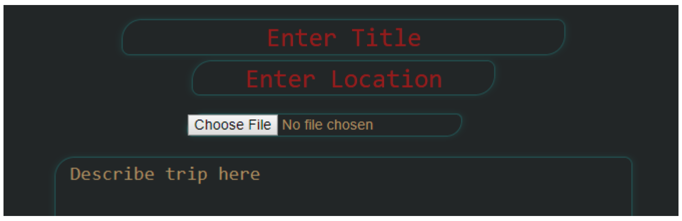

Figure 6: Form for creating a new post

4 Critical Evaluation

Security This platform is meant to be open to the public, where anyone can share their travel experience. This requires a high level of security for protecting the posts from malicious editing or deleting. Currently only a low level security is implemented to make sure only people who know the password can delete entries. This password is static and is stored on the server. To improve the platforms security, at least three types of users should be implemented. An Administrator who is able to change the most fundamental features, Members who are trusted users and are allowed to create, edit and delete posts. Guests who are allowed to browse and submit queries for creating a post.

Comments and Likes The next big feature that should be implemented is to allow users to leave comments and likes under their favourite posts.

5 Personal Evaluation

I have faced many challenges throughout this project since I had no prior experience with any kind of server-side development. Out of the pool of challenging tasks I have enjoyed tackling a few interesting ones.

Callbacks When I started implementing the multiple page feature in the archive I had to query the database for the count of the posts and work with that to determine how many pages needed to be displayed. That is when I hit the first obstacle, receiving the result for that query took a few milliseconds and the code that was running on the server would not wait, so it went on with the count variable that was undefined. After some research I learned that using an asynchronous callback is the solution to my problem.

Delete Button Each post has got their own delete button next to the title, they are all objects generated from a pug template with the same parameters. This proved to be an issue when I was implementing the functionality; when a delete button is pressed, its background image changes, a textbox for password input and a cancel button appears. How do I determine which button was pressed? Which post needs to be deleted? And where to spawn the textbox and cancel button?
Buttons had to store more information than just their own id's. Each button would get an attribute that stored their corresponding post id's from the database, this way the buttons got linked to the posts. Adding the extra objects on button press is very simple to do in Javascript:

```
    //this = the pressed button

    this.parentNode.appendChild(OBJECT);

```

Where *parentNode* is the title area of a post. This way the extra objects would get placed right next to the delete button.

Delete Route Developing the routing of the application has been straight forward up until the delete route. I came to understand that for security reasons it required a bit more work than a GET or a POST method. I used Jquery to capture the button press and Ajax to send the delete request to the server with the post id and password. In the routing function the entered password would get compared against the correct one and only proceed with the deletion of the entry if the passwords match.