Statement and Confirmation of Own Work

A signed copy of this form must be submitted with every assignment.
If the statement is missing your work may not be marked.

Student Declaration

I confirm the following details:

Candidate Name:	SHIHAB MIRZA
Candidate ID Number:	P00190603
Qualification:	NCC L5DC
Unit:	DYNAMIC WEBSITES (DW)
Centre:	ZCAS UNIVERSITY
I have read and understood both NCC Education's *Academic Misconduct Policy* and the *Referencing and Bibliographies* document. To the best of my knowledge my work has been accurately referenced and all sources cited correctly. I confirm that this is my own work and that I have not colluded or plagiarised any part of it.	
Candidate Signature:	
Date:	10/04/2023

TASK 3 (Critical Evaluation)

Introduction to webservices:

A web service is a software component or endpoint that is accessible over the internet and adopts a standard XML or JSON messaging mechanism. The endpoint is usually referred to as an API (application programming interface) which is either REST or SOAP based usually.

XML (Extensible markup language) or JSON (JavaScript object notation) is utilized to represent all interactions with a web service. To clarify, a customer triggers a web service by transmitting an XML or JSON message and awaits an XML or JSON response that corresponds to it.

This response can come from a foreign website/program through HTTP or HTTPS over the internet or it can come from the same website's server.

Potential benefits of web services to GWCS:

- Reduced time to integrate applications or features, leading to improved efficiency.
- Cost savings via consolidation, where common protocols are applicable across all scenarios.
- Capability to integrate existing software modules into one's own applications, leveraging 3rd party expertise.
- Enhanced flexibility to evolve with changing times, rather than being constrained by application version releases.
- Easier data management and accessibility.

How web services add functionality to the GWCS Website:

GWCS has made use of a couple of webservices including the following:

Email JS API:

I used the email-JS tool to allow users to be able to send email messages through the contact form to my email. Email-JS is a JavaScript library that enables sending emails directly from client-side code, using simple and customizable RESTful API calls.
As soon as users fill in and submit the contact us form, firstly, the data is saved in a messages table in the MySQL database then the data is then posted to the email-JS API using a POST request with my public key and the message to send and the username of the user who sent the message and their email as well. the email-JS API then sends the email to me telling me the username of the user and the message they sent.

There is also an auto reply email sent back to the user notifying them that we received the message and that we will get back to them soon.

Google reCAPTCHA API:

I used the google reCAPTCHA API because it is a widely-used solution in online submission forms that prevents spam and abuse from infiltrating the site.

CAPTCHA works by barring spam or bots from entering data into the fields on your website, which can include bogus comments on posts, deceptive emails, deceitful transactions, entries in contact forms, and sham registration submissions.

Un-splash Images API:

This API has helped the GWCS Website come up with images for camping sites and local attraction seamlessly.
The Un-splash images API is a RESTful interface that grants access to a vast collection of premium images.
I used the curl tool in PHP to request for image URLs from this API and then randomly chose images to show for the image galleries and local attractions.

The parameters provided to the API were: my API key, the query string for images, the resolution of the images, how many images I want and a content filter parameter to make sure the images fetched contain **no** inappropriate content.

How HTML was used to develop the website:

HTML or hypertext markup language was used to come up with a skeleton of the web application before making it dynamic. All pages that were needed for the website were created in html first. Of course, the file extensions were ending in PHP because we would have to later on add the PHP code to the pages to make it dynamic, but the point is that html helped me see how the website will look before using dynamic data. At first, I used static data, text and images to populate the html with content. I did this in the following way:
I came up with a header file to include information like the titles etc. in the header tag.
I came up with a navigation bar to navigate between pages and included that in all the files.
I came up with a footer which contained links to social media etc. I started making my text content more structured with the help elements such as h1 and p tags. finally, I started to organize all pages with the help of many non-semantic tags such as the div and span tags.

How CSS was used to standardize the website developed:

After coming up with all the pages in html, I created a CSS folder in which there were 3 files: Index.css, footer.css, and navbar.css. footer.css and navbar.css contained styles for the footer and navbar respectively. And index.css contained styles for all the pages generally.

I started by making the navbar styled and responsive and the footer as well. This was done by help of media queries.

Next, I moved on to styling each page individually, first by adding the necessary CSS selector classes and IDs to the elements on the website. Then styling the classes and IDs individually. The classes and IDs were named carefully in a way so that they don't conflict in the CSS files. I styled using the signature colors of the website which was a dark shade of green and a light shade of green and finally the neutral colors black and white. There mainly 3 colors used on entire website to follow the 3-color rule in designing.

Headings were made bigger in size and longer content was made smaller in size. Border radius, hover effects and box shadows were added to visuals to make the website more

appealing. Images were sized appropriately. Text was positioned in a way that it breaks up visuals. Margin and padding were also added to space out content.

Finally, the website was made responsive for mobile users on all pages using the help of media queries in CSS.

GWCS Website evaluation:

GWCS has a total of 3 web services excluding the RSS feed and the google maps. These are: Email-JS, un-splash images and google reCAPTCHA. All of them work as they should. The emails using email-JS come through seamlessly to my email, this email could be for GWCS staff in reality. Google reCAPTCHA although works on some forms, it completely misses others, so it is not fully functional.
Un-splash images work flawlessly, the API is mostly operational and responds quickly to requests. However, for users with slower internet this may be a bottleneck in rendering pages that make use of this API.

The RSS feed on the home page also works as expected. We can also choose the content to display using the website I use for the RSS feed embed. Google maps works as well, we just need to make sure the coordinates provided are in the correct format.

The rest of the content is either static or coming from the database, which can be debugged and troubleshooted locally if a problem arises.

NUMBER OF WORDS: 1089 (Excluding titles)

Task 4 – (Reflection)

Description:
The process I used to develop the website was simple. Each page has its own PHP file and any PHP logic related to that Page will be written in that file only. There were also the includes files (header, footer, navbar, DB connect etc.) which were included in all pages. The code editor I used was Visual studio Code. I used tools such as jQuery, sweet alert, font awesome icons etc. I also used email-JS along with several other APIs such as the un-splash images and google reCAPTCHA APIs. Almost all tools I used proved to be successful except that the reCAPTCHA was not working on some forms.

Analysis:
I learnt a lot of things from the process I used such as how we have to make sure we sanitize user input to prevent XSS and SQL injection attacks. We use prepared statements and MySQL string escapes to prevent such attacks as well. The process I went through changed the way I understand website development and really made me visualize the power of dynamic websites because I was more used to making static html and CSS websites. I knew a bit of JS, but not much.

Action plan:
When faced with a similar task in the future, I aim to use ajax instead of rendering using plain PHP. I aim to use JavaScript to manipulate the forms more often. I realized I am a bit weak when it comes to making requests directly from JavaScript so I wish to improve on that. I also aim to use a PHP framework to develop websites of this nature in the future as it is more robust to do it that way.

References:

What is an RSS feed?

https://www.spiceworks.com/tech/networking/articles/what-is-an-rss-feed/

Accessed on: 15th April 2023

Web services

https://www.techtarget.com/searchapparchitecture/definition/Web-services

Accessed on: 15th April 2023

Uses of jQuery

https://www.greatinnovus.com/blogs/jquery-important-uses/

Accessed on: 15th April 2023

Sweet alert

https://sweetalert2.github.io/

Accessed on: 15th April 2023

Rest vs soap APIs

https://www.upwork.com/resources/soap-vs-rest-a-look-at-two-different-api-styles

Accessed on: 15th April 2023

HTTP methods

https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods

Accessed on: 15th April 2023