William Joseph

N9937790

Abstract

This document shoes the contributions which have been made to the project by using high quality artefacts and stating how it had contributed.

Portfolio 2

IFB299

Contents

[Artefact 01: Mock-ups 2](#_Toc497503806)

[Artefact 02: Website Django/HTML/CSS/JS 3](#_Toc497503807)

[Artefact 03: Testing 7](#_Toc497503808)

[Artefact 04: Client/Developer changes 8](#_Toc497503809)

[Artefact 05: Update History 9](#_Toc497503810)

[Artefact 06: Backend (from my first portfolio) 10](#_Toc497503811)

Web Links to refer to:

**Main Directory of all Documents:**

<https://github.com/bonanaben/IFB299/tree/master/William/Portfolio%202>

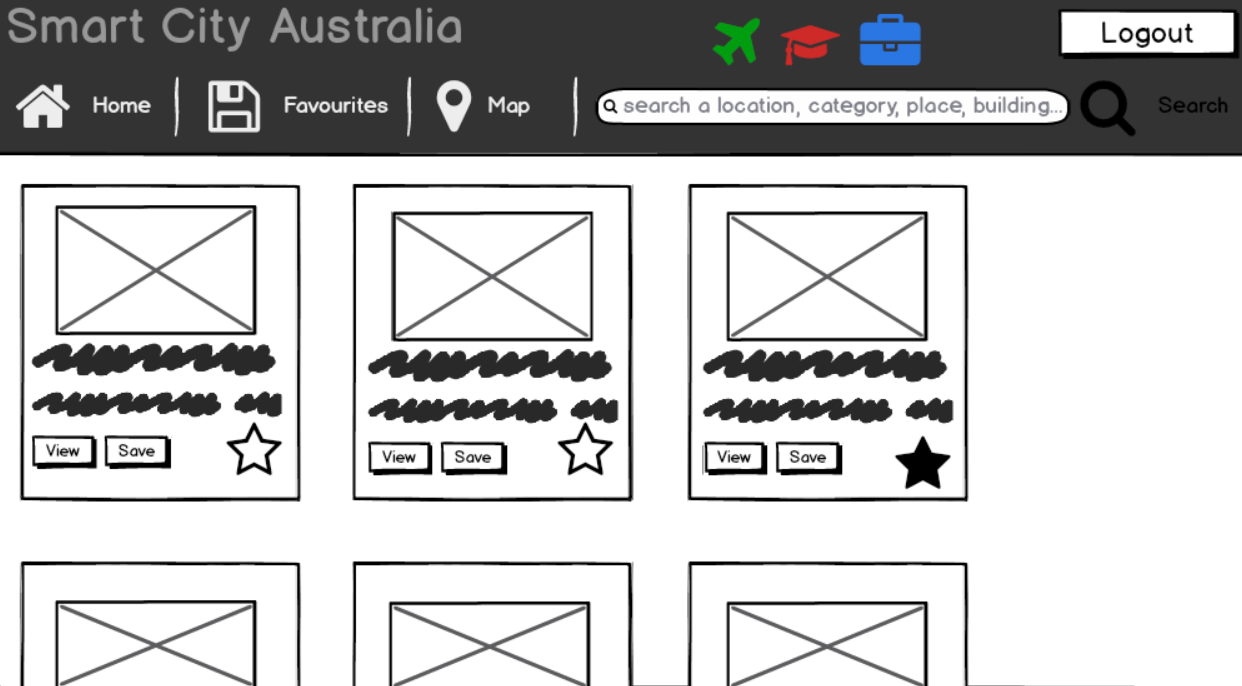
**Website**: Artefact 02: Django/HTML/CSS/JS:

<https://github.com/bonanaben/IFB299/blob/master/Django/FINAL%20WEBSITE/Smart_City299.zip>

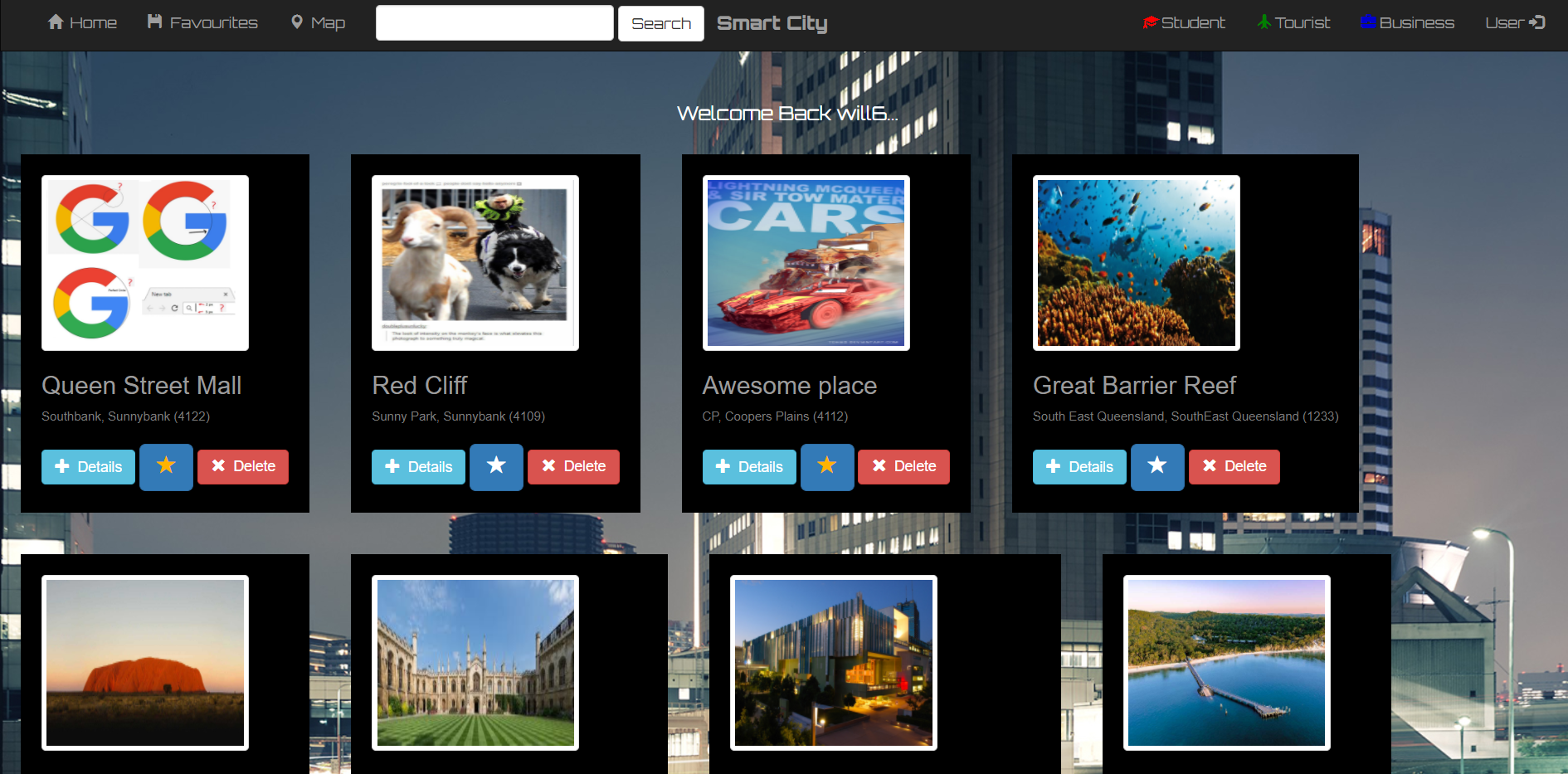
## Artefact 01: Mock-ups

Mock-Ups where very important, especially the second one, as it had a lot of resemblance with the final website, all our mock-ups were based on client feedback and changes (refer to Client/Developer Feedback, Artefact 04). These mock-ups compared to the first ones are a lot more visually closer to the final website. This mock-up was pretty much a layout for the actual website. And a resemblance is noticeable, as the photos shown below:

Mock-up – home page:



Website – home page:



As you can see thanks to the mock-ups they give an awesome representation of the final product, and this mock-up was the closest, as the very website depended on it very much and the similarity is quite clear.

## Artefact 02: Website Django/HTML/CSS/JS

This artefact is very important as it the existent of the project itself, the website was built on Django framework built on top of python that was a web development platform. My focus of the project was in the development of the website’s Skelton, making sure that it was able to connect to the database, process the data, and display the data for the user to see.

Below is an example of some Django code:

def index\_search(request):  
 if not request.user.is\_authenticated():  
 return render(request, 'cityinfo/login.html')  
 else:  
 # this displays all spot\_informations in the database  
 spot\_informations = spot\_information.objects.all()  
 query = request.GET.get("q")  
 # unless the user searches for something then follows the if statement below  
 if query:  
 spot\_informations = spot\_informations.filter(  
 Q(plname\_\_icontains=query)  
 ).distinct()  
 result = spot\_information.objects.filter(plname\_\_icontains=query)  
 return render(request, 'cityinfo/cityinfo\_index.html', {  
 'spot\_informations': result,  
 })  
 else:  
 return render(request, "cityinfo/cityinfo\_index.html")  
  
  
# begins by stating to follow the above function (index\_search) if the user has searched  
def index(request):  
 if request.GET.get('q'):  
 return index\_search(request)  
 # below checks if user is authenticated or not, else redirects to login page, to log back in  
 if not request.user.is\_authenticated():  
 return render(request, 'cityinfo/login.html')  
 else:  
 # this is where filtering happens where the user filters when clicking student, business or tourist  
 # according to category  
 all\_spot\_information = []  
 if request.GET.get('cat'):  
 categoryQueryParam = request.GET.get('cat')  
 if "|" in categoryQueryParam:  
 categories = categoryQueryParam.split('|')  
 all\_spot\_information = spot\_information.objects.filter(  
 reduce(lambda x, y: x | y, [Q(category\_\_iexact=cat) for cat in categories]))  
 else:  
 all\_spot\_information = spot\_information.objects.filter(category\_\_iexact=categoryQueryParam)  
 else:  
 all\_spot\_information = spot\_information.objects.all()  
 context = {  
 "all\_spot\_information": all\_spot\_information  
 }  
 return render(request, 'cityinfo/cityinfo\_index.html', context)

**^^^**These two functions above are from the website itself, *index\_search* was the ability to search for spotinformation (“plname” field). *Index* funtion was the home page of the website, this is an important function as it controls most of the content being displayed, from searching, filtering and displaying all spotinformation

As I have mentioned I created the skeleton of the website, in other words mainly on functionality rather than looks, however I have had some interactions with the looks learning very basics of HTML and CSS which helped me allow the data to be displayed on the web browser. HTML (not really a programming language but a mark-up language, was used to send and request data: communication with the database). CSS on the other hand was used for styling the website, I managed to use CSS to quickly develop a user friendly (basic look) website.

JavaScript was also used for the buttons when favouriting this allowed the user to click that star and it changes colour without having the user refreshing the page, so he/she knows its been favourited and saved to his/her favourites

An example of HTML:

{% extends 'cityinfo/base.html' %}  
{% load cityinfo\_tags %} <!-- Cutesy of https://stackoverflow.com/questions/6451304/django-simple-custom-template-tag-example -->  
{% block body %}  
  
 <!--welcomes the user-->  
 <br>  
 <div class="welcome\_user">  
 <h4 style="color: white" class="heading4"> Welcome Back {{ request.user.username }}... </h4>  
 </div>  
  
 <!--displays the information (no searching)-->  
 {% comment %}<div class="spot\_information-container">{% endcomment %}  
 {% for spotinformation **in** all\_spot\_information %}  
 <div class="divder">  
 <div class="border">  
 <img class="thumbnail" src="{{ spotinformation.image.url }}"/>  
 <h3 class="heading3">{{ spotinformation.plname }}</h3>  
 <h6 class="heading6">{{ spotinformation.location }}</h6>  
 </div>  
 <div>  
 <div class="index\_options">  
 <a href="{% url 'cityinfo:cityinfo\_details' spotinformation.id %}" class="btn btn-info">  
 <span class="glyphicon glyphicon-plus"></span>&nbsp; Details  
 </a>  
 <a data-spotid="{{ spotinformation.id }}" href="#" class="btn btn-primary btn-lg favourite-btn">  
 {% marked\_favourite request.user.id spotinformation.id **as** active %}  
 <span class="glyphicon glyphicon-star {{ active }}"></span>  
 </a>  
 {% if request.user.is\_superuser %}  
 <a href="#">  
 <button type="button" class="btn btn-danger">  
 <span class="glyphicon glyphicon-remove"></span>&nbsp; Delete  
 </button>  
 </a>  
 {% else %}  
 {% endif %}  
 </div>  
 </div>  
 </div>  
 {% comment %} </div>{% endcomment %}  
 {% endfor %}

**^^^** above is an example of HTML code this in particular was the home page and simply had a little Django code “ {% ‘Django code’ %} ”. this HTML section was for all the spotinformation in the database add the data in a container/box on the screen and add the place name, location and image in the container/box, then add 3 buttons “Details”, favourite star, and “Delete” buttons. This was repeated for all spotsinformations in the database. (refer to Final results section)

CSS example:

body {  
 background: white url("https://images3.alphacoders.com/696/69607.jpg");  
}  
  
.navbar {  
 border-radius: 0;  
}  
  
.navbar-brand {  
 font-family: 'Orbitron', sans-serif;  
 font-weight: 900;  
 text-align: center;

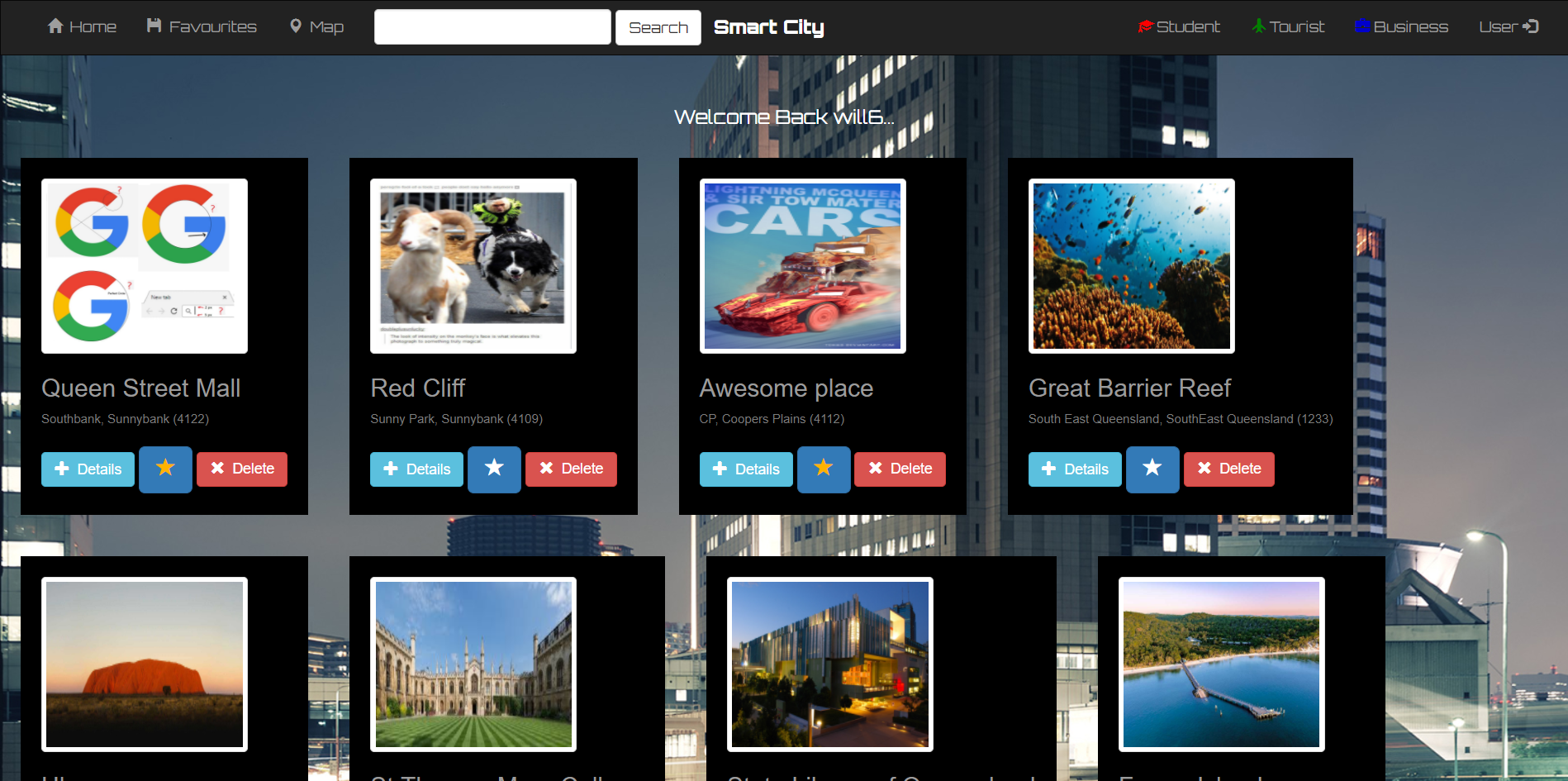
...  
}

**^^^**above is a CSS code example, the first was the background which used a URL from the internet and displayed on the website, if the image was non-existent then the background by default would be white. The other two functions were related to the navigation bar, a bootstrap navigation bar. the first function was about the borders around the navigation bar radius which is set to 0, which is basically 90 degrees corners. Lastly was the adjustment of the brand name of the navigation bar including the font, font weight was how think the font was, alignment where the brand is to appear, in this case in the middle of the navigation bar

JS example:

**var** SpotListPage = {  
 init: **function** () {  
 **this**.render();  
 **this**.bindEvents();  
 },  
  
 render: **function** () {  
  
 },  
  
 bindEvents: **function** () {  
 $('.favourite-btn').on('click', **function** (e) {  
 e.preventDefault();  
  
 **var** self = $(**this**);  
 **var** spotId = $(**this**).data('spotid');  
  
 $.post( "/cityinfo/favorite", JSON.stringify({spotId: spotId}))  
 .done(**function**( data ) {  
 $('.glyphicon-star', self).toggleClass('active');  
 });  
  
 **return false**;  
 });  
 }  
};  
  
$(document).ready(**function** () {  
 SpotListPage.init();  
});

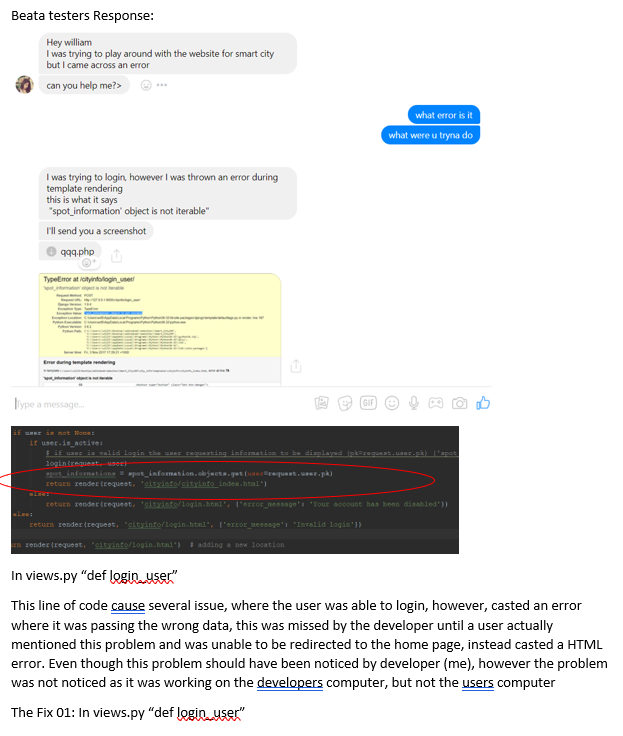
**^^^** above is an example of JS this effects the favourite button, this is how the website changes the colour of the star to yellow when clicking without having to refresh the page when favouriting, and toggles on and off (favourite and un-favourite)



^^^ above is the final result all the help from DJANGO(PYTHON)/HTML/CSS/JS created this website.

## Artefact 03: Testing

Testing was a very important to me, which had a lot to do with the website. Firstly was **Dummy Data** is what I called this test. This testing allowed me to check if the data displays on the screen which included some populating of the database (use of very little data), this helps a lot when checking if my Django code was working as it was a new language that I had learnt since the beginning of the unit, so seeing if data actually displayed was actually very important which shows that the data works. The second test was **Beta Testing**, this was basically user testing however within the team the website was given to my team members for testing the website, this was to check whether it was accessible for others, but mainly used to test for bugs/problems which others found and the developer myself did not see, and then told to me via person or in the chat what was wrong or issues that they found. This information was collected and then fixed.

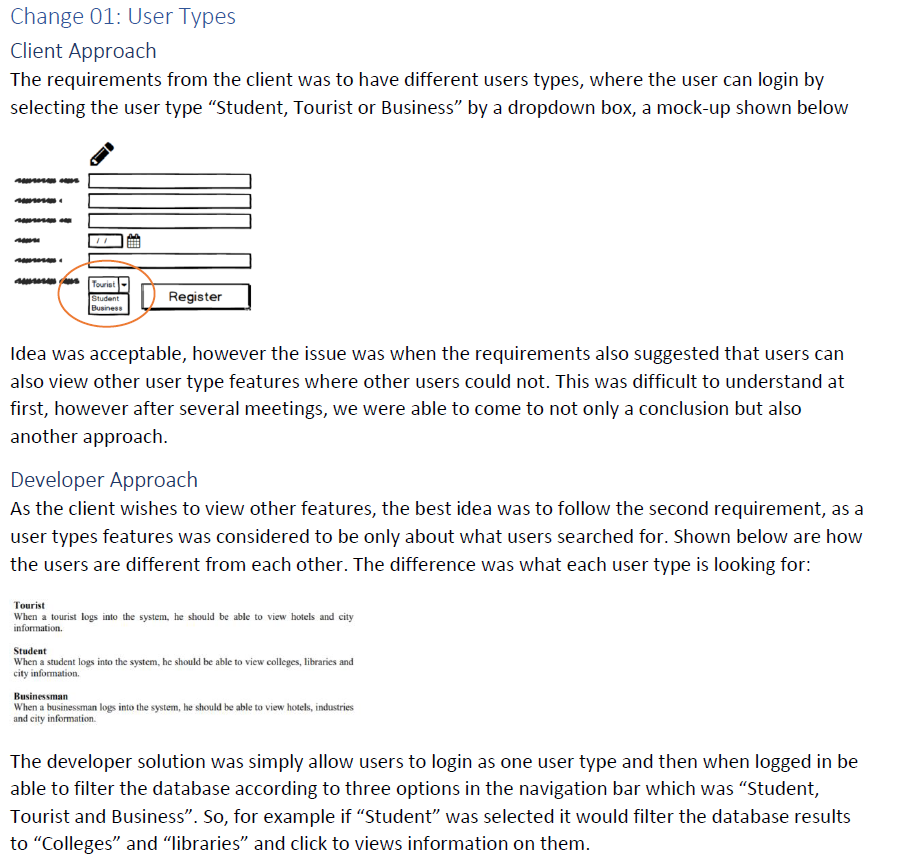


**^^^** above is an example of the document where the developer was told by the user of the tester that the website had an issue when logging in and alerted the developer during the chat, furthermore, the artefact also shows where the developer found the problem to the error and fixed and recorded it in the document, some errors were found and spoken to in person.

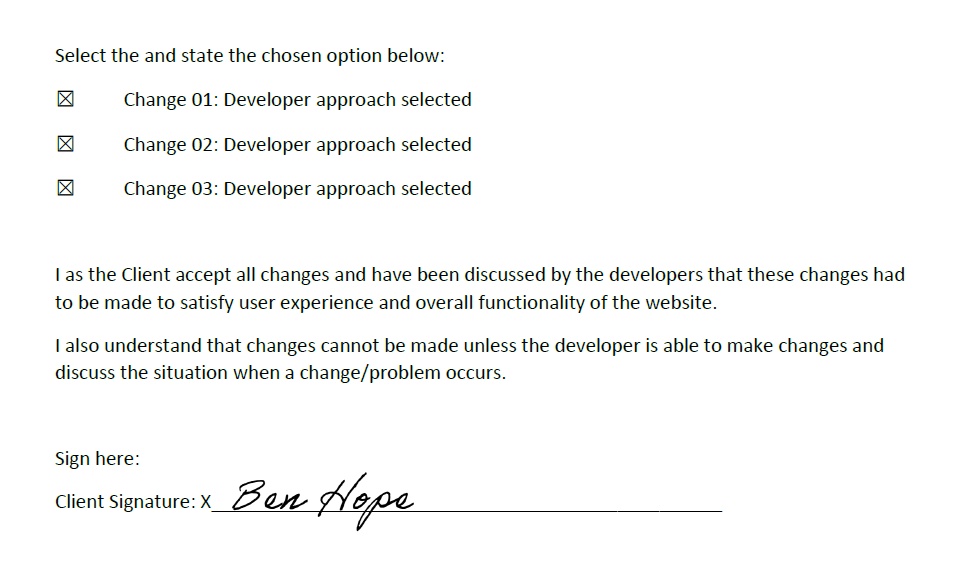
## Artefact 04: Client/Developer changes

Paired Artefact: Benjamin Hope

Client developer changes document was done with developer (William) and the Client (Ben) was all about the interview within meetings and social media about the changes made that were needed or could not be done, it states the client and developer approaches to the problems raised by either side, all approaches where satisfied with the Client and most changes were made as he had made a signature that he had accepted the changes.

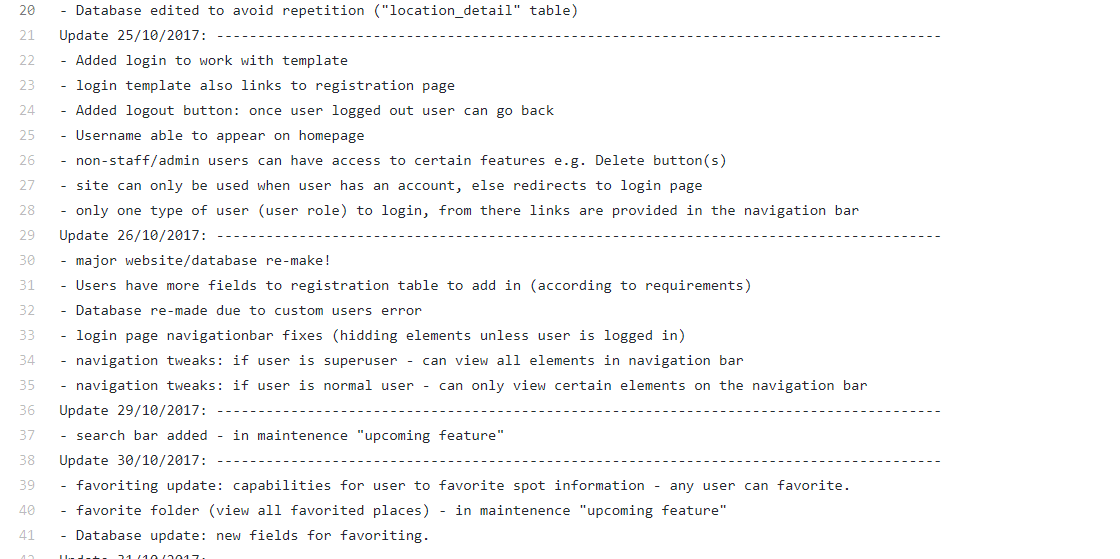


**^^^** the above example is about two approaches one was the clients requirements and the second was an alternative approach by the developer, usually meaning that the developer was unable to understand the clients needs or had a better way of doing it, this was then discussed with the client and recorded where at the end of the day of discussion he had signed the changes to all of them (below).



## Artefact 05: Update History

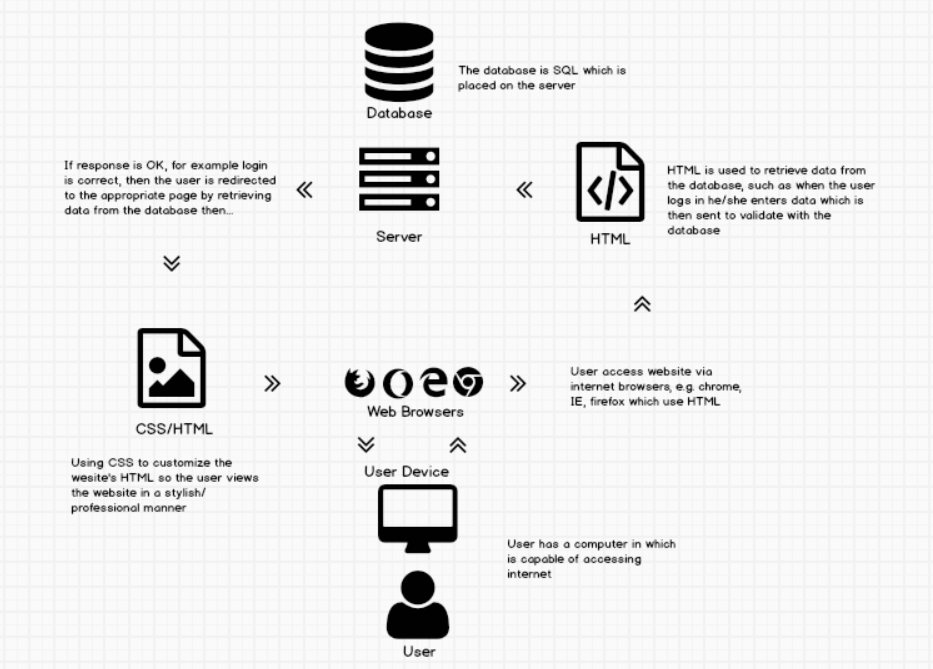
Update history was most important for all team members, as a developer you had the one site and added a feature every so often, but one thing is that other members on your team don’t know what you did on the website, especially when there is more than one developer on the project, update history was edited every time a new change/edit/addition was made to the website itself and recorded when and what happened. This allowed those who download the website from git to know where the website was at, but also what to expect when the team members were using it



The version of the website was versioned according to the date; the day of edit and upload, and specified exactly what changes were made to the website as the above image shows all fixes, additions, database adjustments and in progress/maintenance/upcoming features etc.

## Artefact 06: Backend (from my first portfolio)

backend is a document which shows how the website runs in the backend of the development, this is something a normal user will not see as it is “behind the scenes” of how the website works. This helps a developer or user understand how the website runs when a user interacts with the website in any manner, for example retrieving data when logging in.



This backend diagram helps developers understand how the website works on the internet visually