COMS32500: Web Technologies - Report

Mark Ergus Nicholl - mn16660
Michael Rollins - mr16338

Overview

Our website is a review site for cocktails. It allows the user to browse certain pages without an account/being logged in, but also allows them to signup for an account with the site, from which they can login and access their personal profile. The profile allows them to update their profile picture and post reviews of drinks. The posts allow pictures, descriptions and locations of where they purchased their drink. To test the website, a dummy profile has been made with the email 'a@a.com' and password 'qwertyuiop'. Alternatively, you could sign up to a new profile and add your profile picture and posts.

HTML (Estimate: A)

Frameworks - We did not use any frameworks, we created a total of 10 html pages to allow us to practice and understand html.
XHTML - All pages were developed with the xmnls tag set to xhtml so that the browser can provide feedback for errors in formats.
Tags And Format - We ensured that proper tags were used and attributes are correctly assigned (ie. 'email', 'password') on each tag. We needed to use div tags instead of span tags as it was the requirement for our CSS framework of choice. The language of the pages are set properly to 'en-GB' to allow for better support for page translations.
Modular Components - Components like the navigation bar and post object that are used frequently are written in separate files, these are then loaded with javascript to improve modularity. In order to change or fix issues for these modules, we only need to change 1 file.

CSS (Estimate: A)

Framework - We decided to use the Bulma framework.
SCSS - In order to use our CSS framework of choice while still being able to customise the theme and look of our website, we set up an SCSS file to import the framework. The benefit of this is that we can customise the themes and aspects of the framework, and as the framework is modular, we only import the components we need into our main

CSS file. This would benefit the efficiency of the website as we do not need to load unnecessary data.
Additional CSS - Despite using a framework, we also wrote a lot of additional CSS properties in order to further refine the look of our website. We dealt with issues such as z-indexes, opacity, keyframe animations, hover effects, flex-boxes, uniform font on our website and fitting pages to screen size.
Mobile Support - Our website supports both large screens and mobiles, and testing this with the developer mobile emulator on Google Chrome took a lot of time. The Bulma framework allowed us to implement different resources (ie. the navigation bar) for different screen sizes. All pages were designed to allow both mobile and larger screen users to have a good experience.

Javascript (Estimate: A)

Framework - No frameworks were used for the client-side Javascript. All functions were written ourselves, or we followed a tutorial and re-wrote the code to better understand it.
Interactive Page - On the ‘fixmeadrink’ page, a user is prompted with options and when answered, the page displays a drink suggestion. Javascript functions were used to transition between options, toggle animations and provide results to the user. The animations relied on promises being created and taking the input of the users mouse to toggle play/stop.
Navigation Bar - In mobile mode, the navigation bar relies on javascript to open the selection menu and resize the options.
Load Posts - On the ‘bars’ page, the posts for each bar is loaded by Javascript. The script adds the modular post html page as it receives content to only use the space it needs.
Tab Selector - Users can select between login and sign up tabs on the sign up page.
Login Animation - After clicking ‘login’, an animation is displayed and a timeout function is set off before redirection to the user’s profile.
Load More - On the ‘profile’ page, there is a load more button to display more posts. The button disappears when there are no more posts to be shown.

PNG (Estimate: A)

Images Produced - On the ‘fix me a drink’ page, the images for the results of the mini interactive game are produced with Gimp.
Removing Background - We managed to isolate the subject of images from the background. The initial images were downloaded from public sharing sources online.

We added a layer in gimp, selected the foreground with the selection tool, inversed the selections and deleted the background. The layer was then applied to produce the PNG images.

SVG (Estimate: A)

Images Produced - Our logo, the martini glass with olive, was created in Adobe XD whereas all other SVG images on the ‘fix me a drink’ page were created in Inkscape.
Ellipses - Ellipses were used for lines in most artwork to give more character to the objects.
Path editing - We imported reference images into Inkscape and the trace the objects with the Bezier curve tool. We then simplified the curve if we need to smoothen it. We then edited the paths to follow the curves of the reference object.
Gradients - We filled some artwork with gradient fills with different opacities. An example can be seen on the ‘strong arm’ artwork around the muscle on the ‘fix me a drink’ page.
Grouping - In order to produce SVG animations, we had to group the shapes in order to manipulate them easily.
Animation - We used a software called Spirit to animate the SVG drawings. This is a payware, but we utilised it on a trial. This allowed us to move separate groupings with different transformations.

Server (Estimate: A)

Framework - For our server we expanded upon the provided basic Node.js server to implement the web application framework Express.js. By using Node it allowed us to use unify our code around Javascript rather than having to use separate languages for client and server side development. Express gave us access to a robust API to support various HTTP routing methods which was far easier to use than with Node alone.
Port Number - Within out server we managed to change the port number the website was accessed through to one of our own choosing. We chose 3443 as 443 is the HTTPS default, but on most systems requires root privileges, so for simplicity's sake we chose a port number above 1024 where root privileges are not required.
URL Validation - To increase the security of our server we made sure access to the server would only be granted to safe URLs, to do this we “blocked” all incoming GET requests that did we did not specify as safe, by handling them separately. As the site contains a relatively small number of pages it was reasonable to handle a whitelist of correct URLs and redirect all others to an error page. We also used redirection if a user

who was not signed in tried to access a page which required the user to be logged in, for example this prevent the user accessing “/profile” if they were not signed in, and redirected them to the login page.
Sessions - To allow users to be logged in, we used sessions. We decided not to use a framework such as Passport.js as we wanted to explore this avenue and understand more about how states are stored on websites. We decided to use sessions as they are stored server side, thus making them much more secure than states that live on the client’s side, such as cookies. We store a session when a user logs in and set it to null when the user logs out. The session is automatically ended when the browser is closed. The session is checked when a user tries to access a privileged area, such as the profile. It is also used to determine who’s profile to display.
HTTPS - We use HTTPS to deliver our webpage. To do this, we self-signed an SSL certificate and key with OpenSSL. We then add and require HTTPS on our server, adding the path to the certificate and key stored in our file system.

Database (Estimate: A)

SQL - We used a SQLite database with two tables for users and posts. Each user has a unique ID primary key, which can be used as a foreign key in the post table to link posts to a specific user, a first and last name, an email address, a URL to their profile photograph and a password which has been hashed and salted with a unique salt which is also stored in the database. The posts have an ID of their own, as well as the UID associated with their author, the location of the where the drink was purchase, a photograph URL and a comment.
The database allows for new users and posts to be inserted, given that they pass the validation required. This validation prevents users signing up with an email address which is already registered to our system, as well as basic formatting restrictions such as a regular expression to define what an email address should look like, and length of a password and that names should contain letters only.
Cloud Database - Along with our SQL database, we utilised a cloud storage, Cloudinary, to store the photos uploaded by users. This allowed us to crop/transform uploaded photos to fit the image templates and store urls on the SQL database.

Dynamic Pages (Estimate: A)

Framework - We decided not to use a framework and implement the dynamic nature of our website ourselves. Although this was more challenging, we experienced a lot of issues that helped us understand how frameworks create dynamic websites today. We

gathered ideas on how to make our website dynamic by inspecting published websites, studying frameworks and seeking advice on forums to see how other developers approach the issue.
Process - When a user signs in, whenever they visit the “/profile” their profile is dynamically loaded. This includes inserting their name into the profile, as well as the title of the page, their first four posts (with more being able to be requested and displayed at the click of a button), and their profile picture. Each post is also dynamically loaded and displays the associated photograph, their location and their comment. The posts are loaded in random order so a user will be prompted with different posts every time they visit the page. To load posts on the ‘bars’ page, the script iterates through the http request and creates new user post objects while dynamically adding to them. This creates just as many post objects as required and reduces wasted white space occupied by empty objects.
Modularity - The post object is written in a separate html file, thus making it very modular. We could easily add post objects to other parts of the website without copying and pasting large amounts of code and without creating new CSS files for them.

Depth

Along with the details provided above, these are some of the other areas we spent some time on in our development.
Uniform Theme - throughout the website, we tried to keep a uniform theme and an identity to our website. We decided on the colour and font early on so that we can develop a coherent page together.
Browser Support - Although the website was developed primarily for Chrome, we tried to ensure a high level of support for Firefox and Safari. We ensure features used for animation and styling is available to all 3 browsers. We also spent time on these 3 browsers to fix issues among them. Although there are still some issues such as the date of birth selector on the sign up form for Safari, the website generally loads well in these 3 browsers.
Mobile Support - As we were interested in developing websites for mobile clients, we decided to design this website for both mobile and wide screen users from the beginning. This took quite some time to test as we had to adapt our CSS and test with the list of popular mobile devices available on Google’s developers tool. An example of the support for mobile is the list of posts on a user’s profile. They will display 4 in a row on desktops and change to a vertically stacked section on mobile.