WebTech - Paw Pals

Iwan Pettifer-Col, Eleanor Cox

Introduction

Paw Pals is a website for dog owners to meet up, chat in an IRC-esque chat, view each others profiles, and make new connections. Users signup and are then presented with a chat window in which they can chat to all other online users. They can also view their own profile (and delete their account), or view other online members profiles by clicking the other members names in the chat. The default colour theme of the site is pink, but this can be changed by clicking the palette icon in the bottom left corner.

HTML

Assessment: A
Main pages are written in HTML and validated using a XHTML validator online. We use HTML5 semantic tags such as footer to add context to the structure of our page. Custom elements are generated dynamically and inserted onto a page (on the server-side), details of this are discussed below. We include appropriate header tags and used element classes extensively to style the page.

We use a custom 404 page to catch all unknown requests, which can be reached by navigating to *http://localhost:8080/* followed by any nonsense.

CSS

Assessment: A
We extensively use CSS3 tags such as transform, and transition delay to animate many elements of the website. All our animations are done using CSS tags, where the base class has a “transition-duration” tag, so when we add the specific animation class, the web page renderer uses the transition-duration to play the animation. This makes keeping track of element’s states easier, and offloads the frame rendering onto the GPU (as in CSS3).

We make use of parent > child selectors to style certain specifics that match criterias, allowing us to reuse classes more readily. State-selectors are used to animate button clicks, hovers, and text entry. Pseudo-selectors are used to style things such as placeholder text.
A list of notable animations on our site:

- Click “Signup” on home screen, watch form appear
- Loading “spinner” animation on signup/login
- Chat messages appearing in the chat window
- Validation error messages appearing when signing up

We have responsive styling for the navigation elements, to make the site usable on a smaller browser window. However this site is not designed for mobile users.

JavaScript

Assessment: A

We use JQuery for making client-side operations simpler, such as using $ selectors. We do not use a framework such as React or Angular, and instead opted to use JQuery to help manipulate the state of each page.

Our client-side javascript is almost entirely validated by Lint (aside from a few formatting corrections which don't make sense, such as indentation of object brackets). We use client-side scripts to attach event handlers to buttons and perform POST requests when signing up, logging in, logging out and deleting your account. We also use JS to add, remove, and toggle classes in order to display animations. We use well-defined functions to perform state changes to the page, as by doing it this way we ensure the state is always perfectly changed, and no elements are left in the incorrect state. This is especially useful when there are more than one triggers to changing the state of a page.

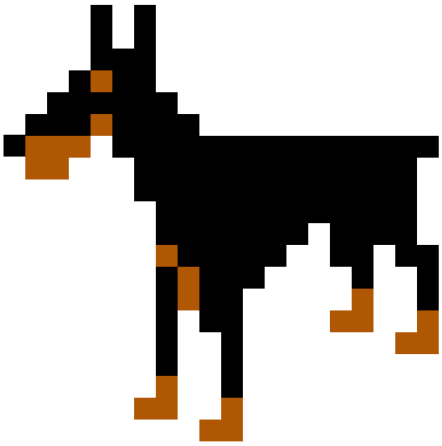
Our site cannot be used without JS enabled, and so if JS is disabled a full-page cover is shown instructing the user on how to enable JS. We separate functions into two categories: handlers, and operators. Handlers are used to add event handlers to all desired triggers for an operation, such as on click, on input etc. The operator functions are then used to perform the operation. Callbacks are used in POST and socket functions to handle successes and failures of operations, and notify the user correctly. Validation is performed primarily server-side, as this allows us to thoroughly validate data which may be untrusted from a client. This is discussed below.

PNG

Assessment: A

The handmade PNG images for our website are the four navigation bar icons, the favicon and the three pixel art dog images on the Images page. The Images page is solely to show off some additional PNG and SVGs, and so does not link from the website itself. It can be reached by navigating to *http://localhost:8080/images*. The icons were created using quite basic techniques, mainly creating, filling and combining simple shapes. However they use transparency cleverly to match the changing of colour theme for the website meaning that different icons do not need to be loaded every time you change the theme.

The additional three images were created to highlight additional skills learnt while using GIMP. To create these we first took a picture of a dog and made this the base layer of the image, cropped to an appropriate size and with opacity at 40%. This was used as a guideline for the "pixel" placements, where each pixel is actually 20x20px. We chose a few colours for each image then by eye we created the original pixel artwork. An example can be seen below, with the original photo and the artwork we created from this.



SVG

Assessment: A

We used SVG to create the logo for the website and an additional drawing to highlight some alternative skills. Both of these are shown on the Images page with the extra PNGs.
We decided to make the logo an SVG as this would make it scalable for future use, something that is desirable when any new website logo is made. The logo uses a mixture of basic shapes and paths to create a simple but memorable design of a dog's face with a heart for a nose, symbolising the relationships built through our website. To create the eyes, nose and tongue we used shape tools to create circles, rectangles and semicircles. The ears and mouth were created using the Bezier curve tool and then editing the nodes in the paths as appropriate. The right sides were created, then duplicated, flipped horizontally and aligned in the middle to create the symmetry. The alignment tool was particularly useful when creating the logo, especially when used with groups of paths (such as, aligning the whole face with the centre of the background circle). The logo is shown below on a black background.

We then created another SVG to show off some more original artwork, in particular freehand drawing. This drawing is shown above right, and features a dog in sunglasses and some interesting background details. We used the freehand line tool and a graphics tablet to draw a picture of a dog, then used the simplify tool on these paths to lower the number of nodes.

We then tweaked each path via the node editor tool and once we were happy with the outline we used the combine tool to create one outline for the dog. We followed a similar procedure to create the sunglasses and background smear. To colour the dog we duplicated the lineart on a separate layer and block filled sections with colours using the fill tool. For the sunglasses and circle we used the gradient tool, and the smear uses the pattern tool for a checkerboard effect.

Server

Assessment: A

Our Express Node server performs extensive validation on all data received. We found the safest way to do this was to first verify whether the exact data fields existed (checking if cookies/keys are undefined), before checking the exact expected format of the data. The server stores a key-value pair object for session management, meaning it's quick to check is a session key sent by a client is valid.

We use correct HTTP status codes for all responses to the client, such as specifying 403 for attempting to access the chat page when logged out, 303 when redirecting to certain areas, and 500 for file IO errors. Express routing made catching different URL requests easy, and included catch-all routing for invalid requests. By moving sensitive HTML pages out of the /public directory, we avoid the issue of clients being able to access restricted pages such as dashboard.html via going to *http:localhost:8080/dashboard.html*.

To serve restricted pages, the server authenticates the clients session key before reading the file, and serving it back. Users can navigate to *http://localhost:8080/me* to access their personal profile page, as the server redirects them to the appropriate page. To access any other profile, we use queries in the URL such as /profile?id=0000000000. The server matches the ID with the correct page, returning 404 otherwise.

On the server, important cookies such as session keys and login tokens are set. By avoiding setting/getting cookies from the client side, we minimise the risk of inconsistent setting/getting performed by various browsers. Some less important cookies (such as theme selection) are set on the client side.

The server uses sockets to enable chat functionality. When it receives a message, it validates the session key and gathers the senders name and profile ID from the database. It then sends the message back to *all* connected clients. We added handlers to ensure clean closing of sockets on both client and server side should errors occur. The server is also very verbose in its logging.

In the chat window, try sending "@server joke" for an easter egg. We use the website www.icanhazdadjoke.com and their random joke API to pull a joke whenever someone types "@server joke" into the chat. The joke is only sent to the user who typed the command, not all connected users. This taught us how to format requests including their headers to external APIs.

Database

Assessment: B

The database is filled with user credentials, such as their email, a password hash, salt, and login token. We make calls to the DB for example when authenticating users and pulling profile information, and use the async callbacks to handle the responses being sent to the clients. We use SELECT WHERE queries to only pull the relevant rows from the DB, and use the DELETE command to remove a user from the DB.

Dynamic Pages

Assessment: A

We have HTML templates stored in a folder separate to public/, and these are loaded by the server as needed. Using asynchronous callbacks, we fill the navbar with the correct currently logged in user's name. In the chat window, we also use client side cookies to determine if an incoming message was sent from that client, and if so we style it differently. Profile pages are generated on the server, by calling the database for profile details before sending the completed template back to the client.

Another example of dynamic pages, is that on the /about page, the navigation bar is only visible to currently logged in users. We also have custom notification popups which are generated dynamically on the client side. These are used to relay service status information to users. On the profile pages, the "delete account" button is only inserted into the profile template when viewing your own profile. Our favorite dynamic element of the site is the chat feature.

You can also change the colour theme of the entire site without reloading the page. By using a self-contained JS function, we swap out elements with certain classes to have new classes. All the theme colours are stored in a stylesheet separate to the main one, to make changing the themes much easier. This can be seen in colours.css.

Profiles also have a randomly assigned profile picture. With more time, we would have developed this into something the user could select, by uploading their own image.

Depth

Assessment: ~A

Aside from taking design inspiration from Google's Material Design, all styling, templating, validation, socket handling, and client side animations were designed by us. We opted not to use a framework such as React, which allows easier templating and database management, as we decided that by implementing these features from scratch we would gain a deeper understanding of their ins and outs.

Our favorite parts of the site that we designed include the custom notifications, the site-wide theme switching, the signup form animation, the loading animation, the chat bubbles, and the 404 page. With more time, we would like to have added some basic chatbot features to the chat, where users could talk to the server.

To try the full site, in two separate incognito windows (not tabs) signup using two different sets of details. Once both windows are signed up and on the chat window, you could change the colour theme of one to help differentiate them. You can chat back and forth, with different colour chat bubbles to indicate if you sent the message. Click on the other senders message

icon to view their profile. Click on the home icon to view your own profile. From here you can delete your account. Click the padlock to logout, or the “i” to go to the about page.

The chat feature was tested with multiple users over a WLAN (not eduroam due to firewalls) and it worked great; feel free to try it with multiple devices! Overall we really enjoyed and were proud with our results of creating a basic social media / chat platform in a small amount of time.

A list of notable aspects:

- Thorough server-side data validation
- Signup and login animations
- /me redirecting
- Chat features
- Theme switching
- Custom notification popups
- Custom 404 page
- Animated input validation error messages
- Send the message “@*server joke*”