Report on use of Web Technologies

# Samuel Russell

**HTML**

**A**: means you have investigated a variety of different issues and gained a general high level of confidence with the structure of HTML pages (or with generating HTML via a framework)

I used an assortment of block, inline and inline-block elements. I ensured it met the HTML5 standard using my browser. I wrote some static pages by hand and some using EJS templating engine (both server side and client side) which I like as the template is based in html so you have full control and freedom over what you are getting.

**CSS**

**A**: means you have investigated a variety of different issues and gained a general high level of confidence with CSS style (or with generating CSS via a framework)

I defined a site wide style and page specific stylesheets. I made use of hierarchical selectors for bigger pages and also dynamic (e.g. on-hover) styles. I made use of different types of positioning, for example fixing the position of the login modal to the centre of the screen and different types of display, for example inline-block to allow the matches to dynamically layout on the screen. I included CSS animations initiated by JavaScript to provide a fluid user interface. I also used media queries to change the layout of the webpages to suit different screen sizes. I could have used bootstrap, but I wanted to have total control.

**JS**

This is client-side JavaScript for effects or animation or interaction, including use of client-side frameworks, but excluding aspects which are to do with dynamic page construction

**A**: means you have gained a high level of understanding of how client-side JavaScript works, or a high level of expertise in using client-side frameworks

I wrote JavaScript that handled user interaction for example adding a new player and logging in. All these server interactions (POST and GET requests) were done using ajax behind the scenes, as to not disrupt from the browsing experience. I used a variety of event listeners including for clicks and key presses.

For the new player dialog, I used JavaScript to populate a dynamic list of search suggestions of players from the server based on what you are typing. It also dynamically populates a list of rubbers depending on which competition you select.

I animated the landing page logo using JavaScript. There are many libraries to animate SVG elements but since I only wanting a simple motion I opted to create it from scratch.

**PNG**

**A**: means you have gained experience with some more sophisticated tools such as handling layers and transparency, or airbrushing or creating original artwork

I created a banner image from scratch in GIMP with some pictures I took. I used a variety of tools including…



Gradient: The background has a gradient to try and match the court to sky colour scheme.

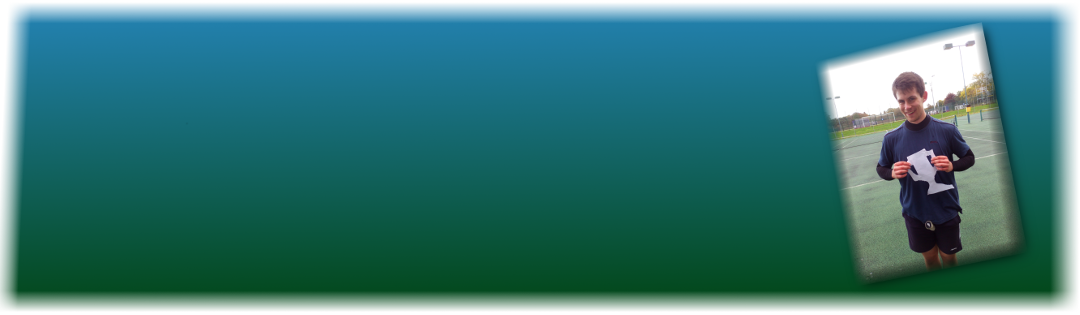


Transparency: the edges fade out in to a transparent background.

Layers: Each photograph had its own layer so I could edit it individually and place it on top of the background. These were then composited together for the final image.

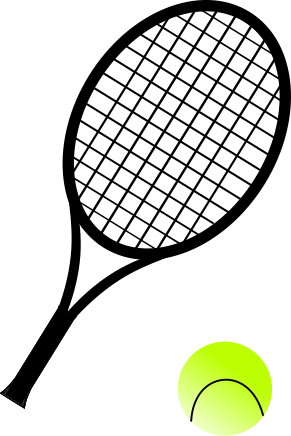


Airbrush: On the middle photo, I used the white air brush to add more spray from the bottle to make it look more dramatic.



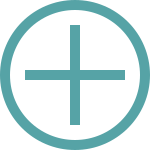
I used a drop shadow filter on each photograph to give it a 3D effect.

I used tinypng.com to reduce the file size of the image.

**SVG**

**A**: means you have gained a higher level of experience, e.g. with path editing, grouping, transformations, gradients, patterns, etc., or put a lot of effort into vector artwork.

I made this (right) SVG from scratch. I used the shape tool and gradient tool to make the ball. The racket consists of several splines. The racket and ball are two separate groups. I also made all the icons for the site shown below.

    **** ****

**Server**

**A**: means you've dealt with things like port numbers, URL validation, content negotiation for old browsers, sending redirections to browsers, handling UTF-8, https and certificates, or web sockets, or cloud hosting, or security issues beyond URL validation, or auto-testing, or cookies, or running under reduced privilege

I started with the given express server and extended it in a modular way. One module serves up static resources (images, CSS, client-side JavaScript, client-side templates). Another module serves up dynamic templated webpages. The final module serves up an API for interacting with the data using JSON objects.

The system handles UTF-8 character set, so text can include emoji.

I have set up content negotiation so that it will serve up XHTML if the browser supports it.

When visiting the root address, initially the landing page is shown, however if the site has been visited before (cookie set) it redirects directly to the home page for convenience.

If you try and visit a player’s page that does not exist, then you will be redirected to a special error page.

I have set up a http sever which redirects all requests to the secure https server. This uses TLS to authenticate itself with a self-signed certificate and encrypt traffic so passwords and other sensitive data cannot be stolen.

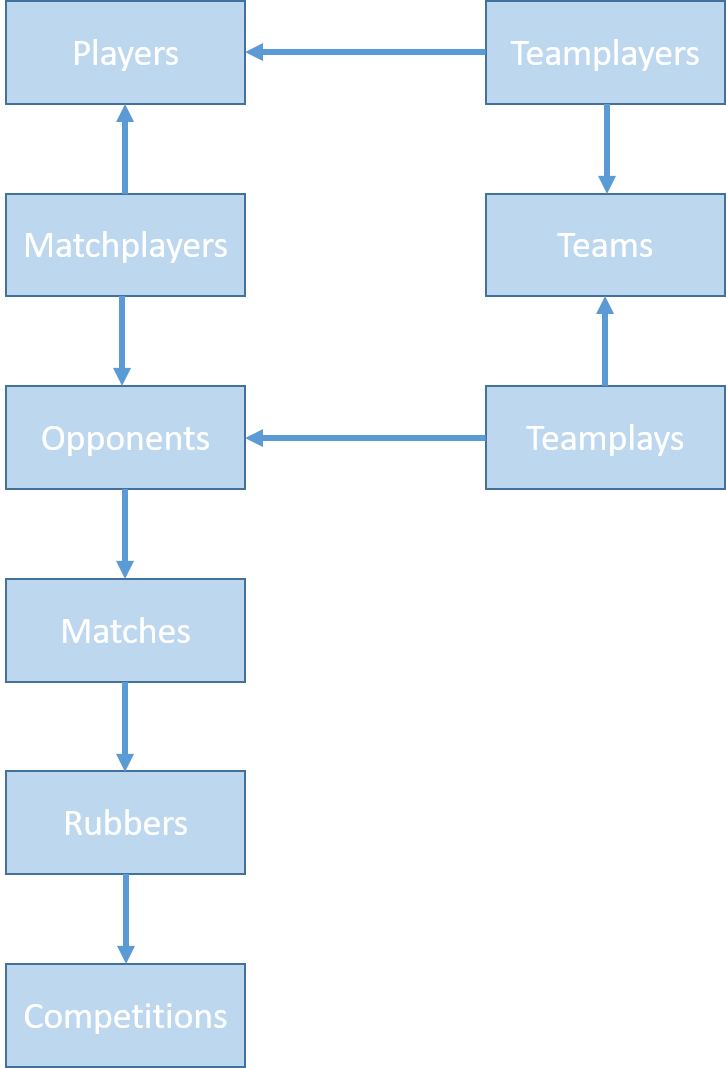
I have used the express cookie parser to extract the cookies from the request headers and set them again in the response. I don’t store much information in them though, only the visited flag as mentioned above and the session id number.

I have created a session system from scratch. When a user visits a page, they are given a random, unique 32-byte key which is set as a cookie so that is sent with every request (including the API calls). This cookie is a session cookie so that the user is logged out when they close the browser. I also use another cookie that has a moderate life so they are also logged out if they leave for too long. This life is lengthened when they make a new request. When the user logs in the user id number is linked with the session id in the database, so that for each request the current user is known. Unused sessions are deleted after a time as they are no longer needed.

As mentioned I have also implemented user accounts, to restrict some actions to only certain users; i.e. the club organiser. I created this myself from scratch ensuring to salt and hash passwords before storing in a database. The login is done by interaction with special login API which if successful associates the current session with the user id.

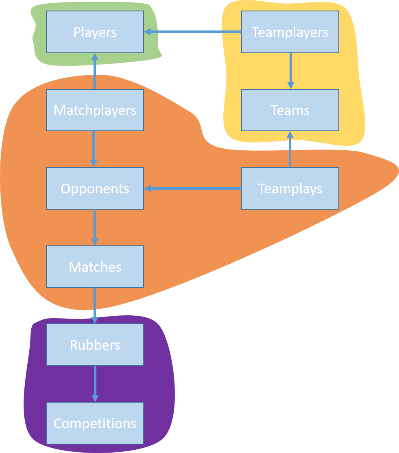
I have hosted the site on my raspberry pi at <http://ubstess.ddns.net>.

The account details are username: ‘ian’ and password: ‘webtech’.

**Database**

A: means you've gained a lot of experience with SQL, or you've put a lot of effort into organising database access (e.g. into a separate server-side module) or you've put a lot of effort into database design or details of handling your data

The match data is spread across several tables due to the one-many or many-many relationships as shown in the diagram. To extract data, I had to JOIN (left and full) multiple tables. In other places where it was more appropriate I used multiple requests to extract all the data I needed.

I created separate data base controllers for each portion of the system to create interfaces for the different operations such as creating a match.

I also, implemented user accounts and sessions from scratch storing the data in tables.

To create the tables I used a library called *db-migrate*. This is normally used as a database migration tool that means you can change the database without starting from scratch. I used it since it gives a clean way of defining the tables and also in the future when I am handling real data, I can transition to a new version seamlessly. I also created a *DatabaseSeed* script to pre-populate the tables with example content.

I ensure all SQL queries are sanitized by using the library’s variable inject system.

**Dynamic pages**: this is *either* inserting data into templates on the server side and delivering dynamic pages *or* requesting data from the server and inserting into existing pages on the client side

A: means you've put in a lot of programming effort or become very fluent in using your chosen framework

As mentioned, I used EJS templating framework. It was mainly used on the server and served up a page populated with content from template. It also added extra features if you were logged in; e.g. edit dialogs. It enabled me to have a central set of shared resources for example the common imports in the head and the navigation bar, so they were consistent over all pages. I also used EJS templating on the client. For adding a player, the html element was relatively simple to populate and dynamically insert in to the DOM, however a match element is a lot more complicated (laborious) to construct. Therefore, I was able to use AJAX to fetch the same match template from the sever and populate it with the new data much more cleanly.

Depth

I have tried to make a system that is functional and well laid out on computers, smartphones and tablets for instance the navigation bar shrinks to a hamburger menu on smaller screen sizes. The full functionality is not all there on the front end, for instance you cannot create new teams or competitions, but they would use the same process as for players and matches and the backend API is all there.

I used a prebuilt cookie parser, JSON parser and templating system, but built my own session and user account system.

When submitting data to the server, as well as server validation of the data, some checks are done client side, to provide quicker feedback.