# Web Technologies Report

**Julian Loscombe**
Username: jl14910

**Raul Mihoc**
Username: rm14834

# 1 Introduction

For this coursework we made a website for the game of draughts(checkers). It offers both a local version, in which two people play on the same computer by taking turns and a remote multiplayer version in which a websocket connection allows two people to play each other over the network.

To run the server you can use the npm tasks provided. If you only want to play locally without websockets then you just need to run: `npm run start`.

To run the full game with multiplayer enabled then you should run the following commands, then navigating to `https://localhost` in the browser:

1. `npm run start`
2. `npm run start:matchmaker`
3. `npm run start:router`

We have specifically chosen to only support modern browsers as we wanted to explore features such as websockets and the canvas.

# 2 HTML (claim A)

Our submission contains 3 html pages written without use of any framework. All pages, except for the index page, are served using the content type `application/xhtml+xml`. The index page is not served as xhtml because the Google SignIn button we are using injects non-xhtml content into the page, and therefore the browser returns an error when displaying it.

Additionally, we have an automated way of running the vnu.jar validation script on all of the HTML pages we have developed.

We have used a range of tags, including the `img` tag, the `form` tag and therefore `input` tags. We have also used the html5 sectioning tags, such as `main`, `header`, `aside`, etc. wherever they made sense. We have also used tags to inport scripts and stylesheets from other files.

We used the meta tag to tell supporting browsers the viewport that our page expects, this makes the page work better when we use the chrome dev tools to simulate a mobile browser.

## 3 CSS (claim A)

All the styling for our website, apart from the actual board for the game, has been generated using CSS. Some of the issues we have investigated include: linking the Quicksand font from Google Fonts, use of various selectors such as tags, id's, classes, children, actions such as hover, action, etc. and tag and attribute values. Further, we have also implemented a simple pulsing animation for the logo of our site using the `@keyframes` rule of CSS3.

The pages also use the flexbox layout, therefore making the webpage responsive to different browser window sizes.

## 4 Client-side JS (claim A)

A lot of the work for our submission has been concentrated on this part. We have used plain ES5 in our client side code which consists of a model of the game of checkers, a protocol of message passing using a websocket in order to allow for remote multiplayer games as well as a front-end for the actual game.

The game board has been drawn using canvas 2D and has been animated using window.requestAnimationFrame().

We employed the test driven development strategy for this code and wrote unit tests using the QUnit library.

We use the `localStorage` api to store the parameters for a game that are given by the matchmaker server. As we want to allow multiple clients in the same browser, we store each clients parameters as a key value pair, indexed by the users unique Google id. We then pass this unique id as a parameter in the URL when calling the game view page.

## 5 PNG (claim A)

We have generated PNG screenshots of our index.html page and modified these using GIMP in order to create a short tutorial for the use of our website.

Topics that have been investigated include layers, transparency, converting images and changing resolutions (since the original image was a print screen). In fact, all of the 3 images (localgame.png, logout.png and remotegame.png)

have been obtained from a single .xcf file which contains 4 layers. The transparencies of these have been manipulated in order to obtain the desired highlighting effect of the button.

## 6 SVG

We have generated an SVG crown, using Inkscape, which is used to tell the user that a piece is a king piece. We also use draw the entire board using only the SVG drawing api of the html5 canvas.

Topics that have been investigated include layers, freehand drawing, using both fill and stroke, splines and boolean operations such as the difference operation. This logo is then used to make an `Image` object and is then drawn directly on the canvas using the `drawImage` method.

## 7 Server (claim A)

We have used the initial server provided and adapted/enhanced it to suit our needs. We also wrote two simple websocket servers for matchmaking and routing of messages. Specific issues we have tackled include managing websockets to allow multiplayer games, defining the messaging protocol for the websockets, https and SSL certificates, and dynamically checking the incoming http request headers to see if there is support for xhtml.

## 8 Database (claim A)

Our website uses a database in order to store an "Elo" score for each player which gets updated after every multiplayer game they play. Each player starts at 1000 and his score grows or decreases based on performance in the last game. A leaderboard consisting of the best players is shown at the bottom of the index page.

An important point to make is that the name and ids of each player are provided using the Google SignIn API which has been integrated in our index page. This simplifies the issues with duplicated usernames or ids as we can now use the data from the google account unambiguously.

We have organised all database accesses into a separate server side module

which we have developed in the test-driven development fashion.

We also used some more advanced SQL statements such as `ORDER BY` to get the top ten users with the highest scores from the database.

## 9 Dynamic pages

Our pages are dynamic as they request information from the matchmaker server, or listen for information from the router server.

The index page requests an up to date leaderboard from the matchmaker server and then dynamically adds the results as a table to the page.

The game view is dynamic as players can send messages to one another over the websocket connection, these will then be displayed on the side of the page, using javascript to inject html tags.

There is also a live timer for each move which counts down telling the user how long they have left to make a move.

The user can interact with the board view, selecting a move, and have the move played on the board.

If a remote player registers a move, that move will then be displayed on the opponents board.

## 10 Depth

We feel that we have explored javascript, websockets and the canvas the most.