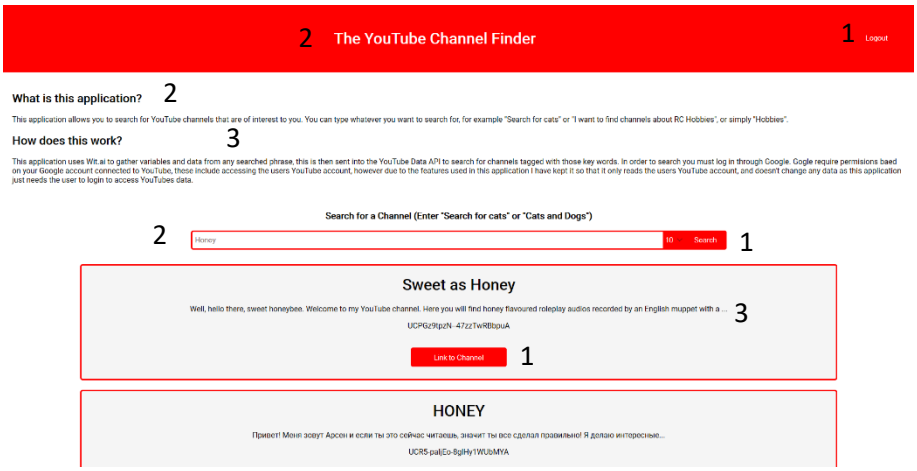
Design documentation:



Design documentation: With my project I made sure that the touch areas were consistently styled and large, in order to help older people use the site, whilst keeping the buttons relatively simple and easy to use (see 1).

I also followed the KISS analogy (Keep It Simple Stupid), to make sure that the interface around the entire project is simplistic and minimal for easy understanding and use. It can be seen that it is set into 3 sections, the heading which includes the login button, the explanation, then the application, all labelled for easy interpretation (see 2).

I followed the repetition rule with my design, where every section of the screen is similar to unify the overall website, the repetition features used include the colours used throughout (YouTube Red, White and WhiteSmoke), as well as the buttons being the same throughout the application. Fonts used are the same (Google Font Roboto, same font used on the YouTube website to keep it themed).

The text and background colour also has a high contrast allowing for the text to be easily read on the page. This contrast is white text on a red background, and black text on a white background. Text was also aligned to the left to make it easier to read (For the description as it was multiple lines, see 3).

Technical documentation:

The APIs I used were Wit.ai and the Google YouTube Data API. Wit.ai is a Facebook owned API that allows you to send the API data, and according to how you taught the Ai of the API is send back information regarding what strings were sent to it. For example with the application I made, the application sends it strings regarding what the user has searched for, e.g. “Search for Cats”. I have taught the AI to find search queries inside the strings it is sent, therefore it returns “Cats” as a search query, allowing me to use that as a variable later.

The second API I used was the Google API (GAPI). I used this API for the OAuth which was needed for the YouTube Data API. This API allowed me to let the user log into the application, then search for YouTube channels. I did this by calling return gapi.client.youtube.search.list which included the data that was being searched and the amount of results, the data was channel tags, which was gained from the Wit.ai.

The most difficult challenge that I had to overcome in implementing this application was using React.js. For what I was doing it would have probably been more beneficial to use jQuery, but for criteria I used React.js and was able to achieve mostly the same result. It was very difficult to figure out at the start, but I eventually got the hang of it. The only problem that I still have with it is you can’t use jQuery to get when a certain ID of the React.js is clicked, like what we did with the Flickr tasks. I originally wanted to use a feature of the YouTube data API where you could click a button to subscribe to the channel, but I wasn’t able to figure out how to get an on click event using React.js, as the click attribute gave data for all of the React.js components in the array and jQuery was unable to access the React.js components.