# GUI Documentation

## GUI Design

For the GUI, I opted for a simple HTML file. This is for a number of reasons, the first being that running a full web framework for a single page GUI is overkill and would essentially just be a fancy way of serving the html document. Secondly, the reason I opted not to use a python+tkinter type GUI is that HTML/Javascript is perfectly adapted to making http requests, making it a better fit for our API which is a flask http server. If you take a look at ‘full\_monty.py’, you will also see how easy it is for us to start the html API on a local machine programmatically. It is also not uncommon for websites to serve .html files even today.

The UI itself uses twitter’s bootstrap for styling, making the page not only look nice, but also making it cross platform – the page is useable and aesthetic on both desktop and mobile devices on a vast span of screen sizes.

The UI elements are minimal and clearly labelled, there is an expandable field for entering a movie plot and a button to get the prediction. Clicking the button outputs the prediction and the percentage certainty below. There is also a search box at the top of the page for the user to type a movie name into, it will search Wikipedia for a match so that the user can copy the plot and paste it into the main text-entry box. The background is an artistic wallpaper from the movie “Alien”, chosen because it fits with our theme of movie plot predictions, but also because the darkness of the image makes the text very easy to read and because Alien is a truly awesome film.

## Model Integration

To communicate with the API, a post request must be sent to <http://localhost:5000/prediction> containing the plot in the data. In the first iteration of the GUI, the response to that request was served raw to the user. That meant basically redirecting away from the GUI, to the address above. In the interest of keeping the user on the GUI page, I switched the request method from a form submission, to an AJAX request which meant the user never has to leave the page, and all the work is seemingly done in the background. To help me do this, I included the jQuery code (makes code style prettier than vanilla JS, adds functionality, simplifies existing functionality). Once the AJAX request change was made, the API was rejecting these types of request due to having no CORS policy, to make the API accept ajax requests, I had to add a CORS policy and configure it.

When the model was updated so that it could work out the certainty of a prediction, minor changes were made to the API and the GUI in order to transmit these two pieces of data at once. The GUI was now receiving data in Object form opposed to Array form. The API returned prediction certainty as a number between 0 and 1, in order to make this human readable, I multiplied the number by 100 and rounded it off to 3 decimal places to get a percentage before returning that along with the prediction to the user.

## User Testing

Testing the AJAX requests was when I reached the major hurdle, which was that the API had no CORS policy. Upon further research, I realised this was an easy fix, implemented it, tested and it worked.

I have also tested the layout to make sure it works on various devices and screen widths. When adding the movie title search field, I tested this again to make sure the new element didn’t mess with the current layout or overflow off the page. At first it did, so I limited it to 300 pixels width.

I tested the movie title search function a fair bit. It started off quite primitively by opening a new tab which searched Wikipedia for the movie which the user had entered, the user would then have to copy and paste the plot back into the text field. A few times I tried adding “(film)” or “(movie)” to the end of the Wikipedia search query in order to stop a search for “cars” from redirecting to a page about cars rather than the movie, this proved unsuccessful and messed with the results which worked fine before. I looked at the Wikipedia’s API as well, however this didn’t contain the functionality I needed. Eventually I got access to IMBDs API, which had all the functionality I needed and only contains records on movies and television so “cars” would get the correct information. Upon testing this solution, I realised that searching for movies with a space in them wasn’t working, this is because I’d manually URI encoded the title, and the AJAX request was automatically URI encoding the title again.