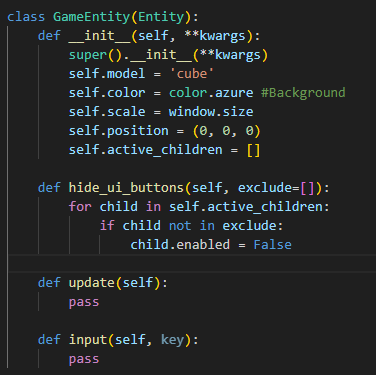
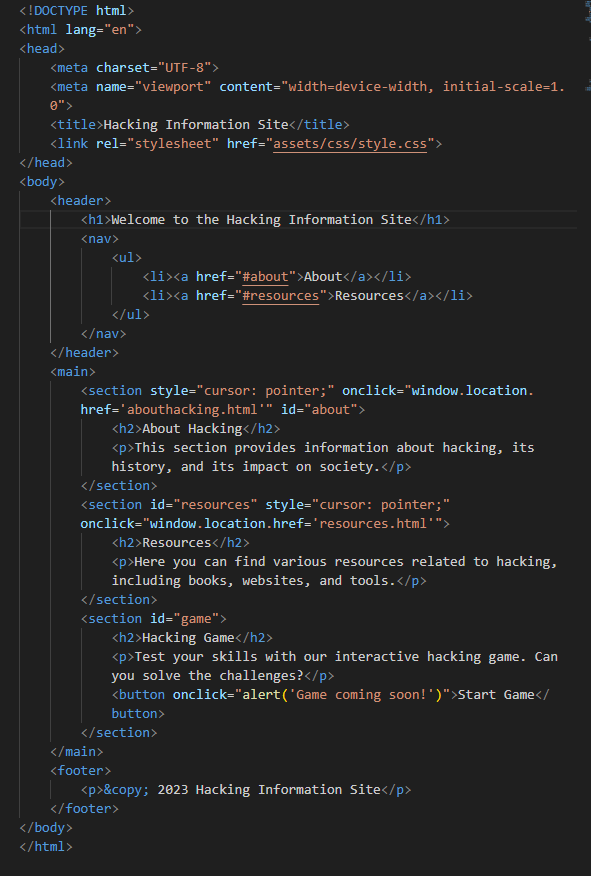
### Documentation of “hackerman” website.

###### The early idea – A catastrophe.



The original idea for the game was to create a virtual gambling game like you find on games like Roblox, the concept of the game would allow you to earn points from answering questions about hacking and then use those points to gamble. In retrospect this was a really bad idea. Therefore, we decided to scrap it for a better idea.

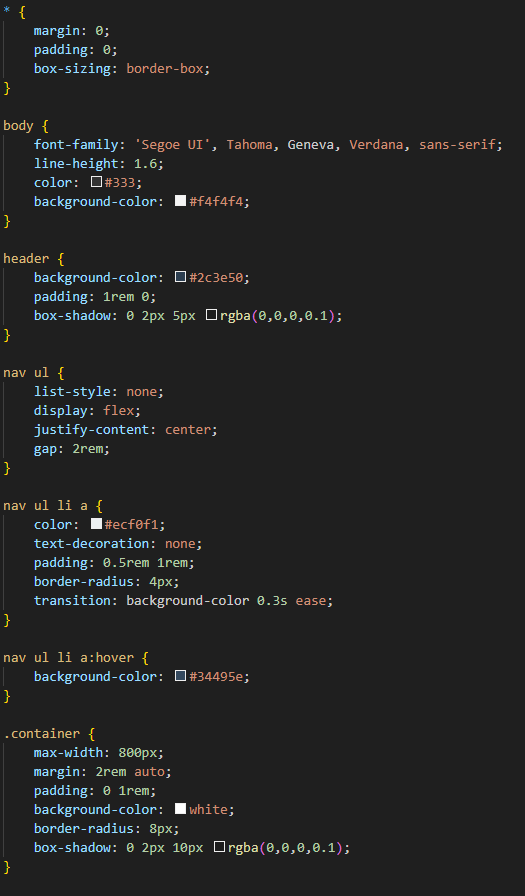
###### The redesign – An arguably incredible replacement.



After a lot of debate, we decided to move onto using a website to help the year 10s have easy access to resources that help them with learning about hacking, the ethical ways and unethical ways. This site provides varying information from general terminology and even a side game that guides the person playing common ways that ethical hackers break into the systems of companies.

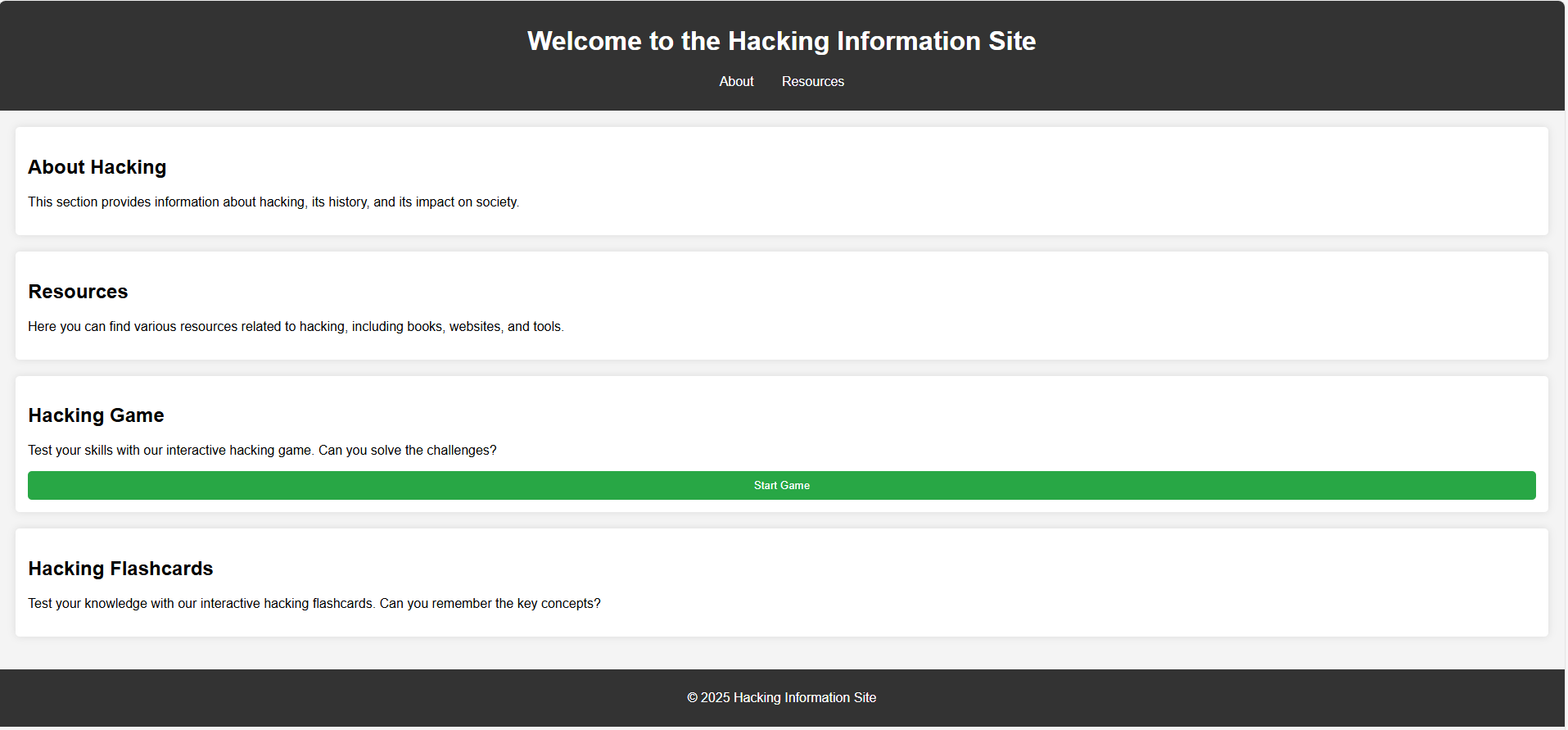
The end goal is to get them excited, youth now a days are often quite rebellious (apart from Phil) and therefore the thrill of doing something bad will most likely get them excited for computing.

###### Styling and making the pages beautiful.



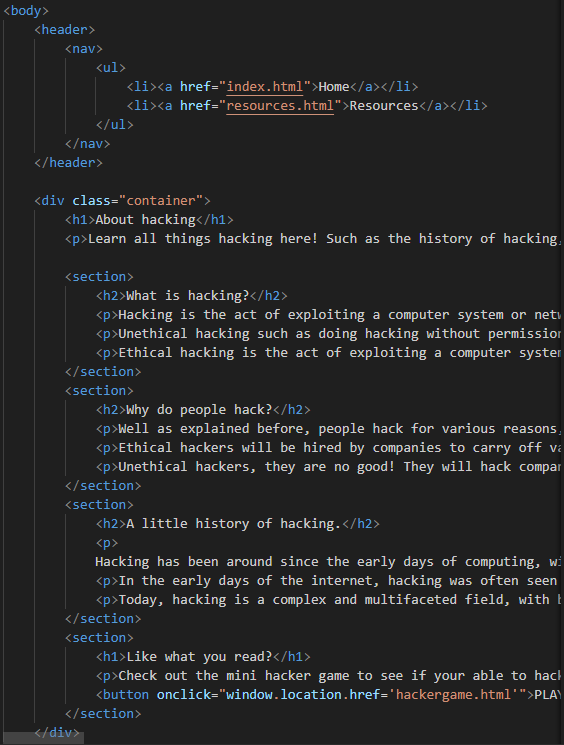
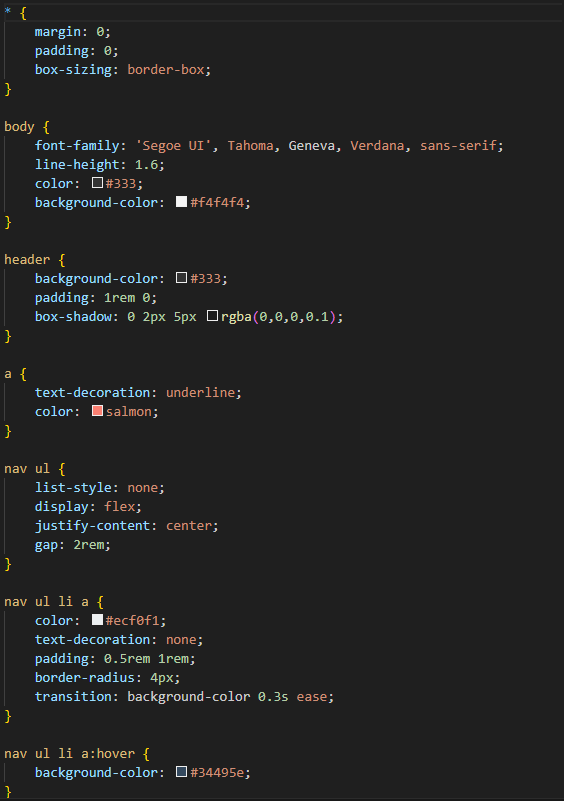
Of course, to make sure that the website looks good I implemented some CSS styling elements so that the website looks more structured and nicer, in the end the UI looked rather dashing.

*A look at the UI*.



This is what the UI looks like with the html and the CSS applied, of course to make the buttons and all that to work we needed to implement some JavaScript so that it can interface with the website and listen for clicks on each element, due to the JavaScript not needing to be that intensive it more plausible to just add the JavaScript onto the tags themselves with the “onclick” function which redirects the user to the correct webpage.

*The resources page – A look into hacking*.

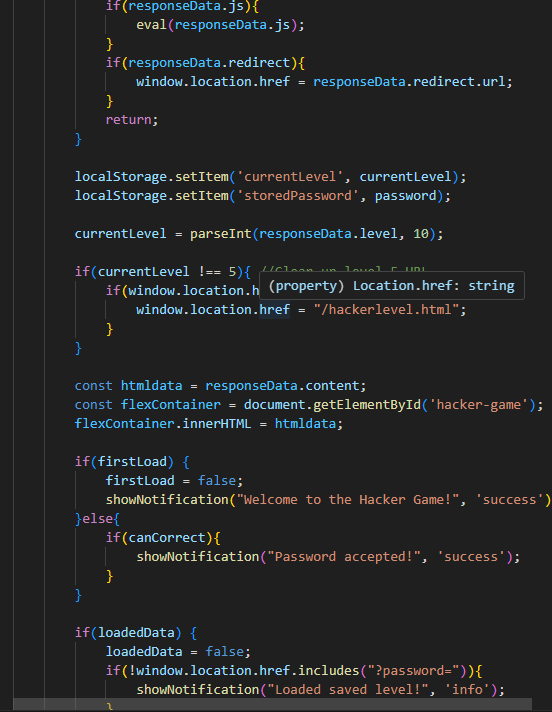
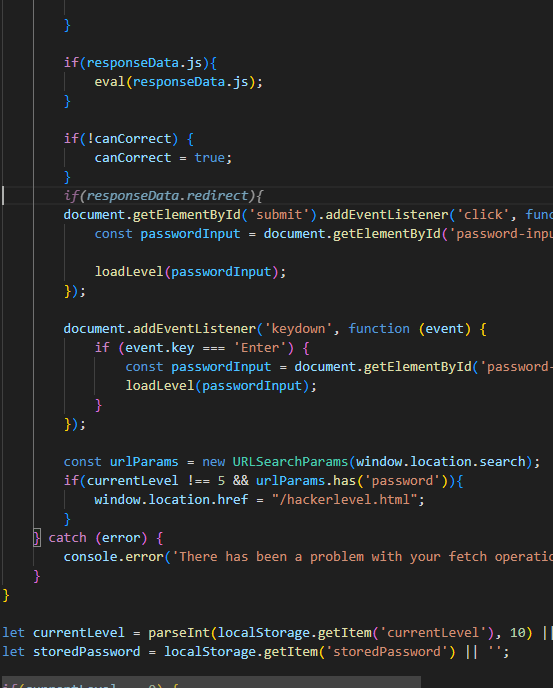
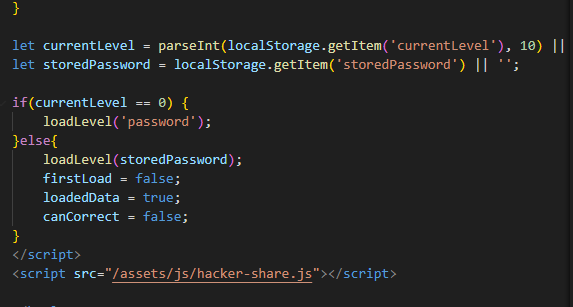
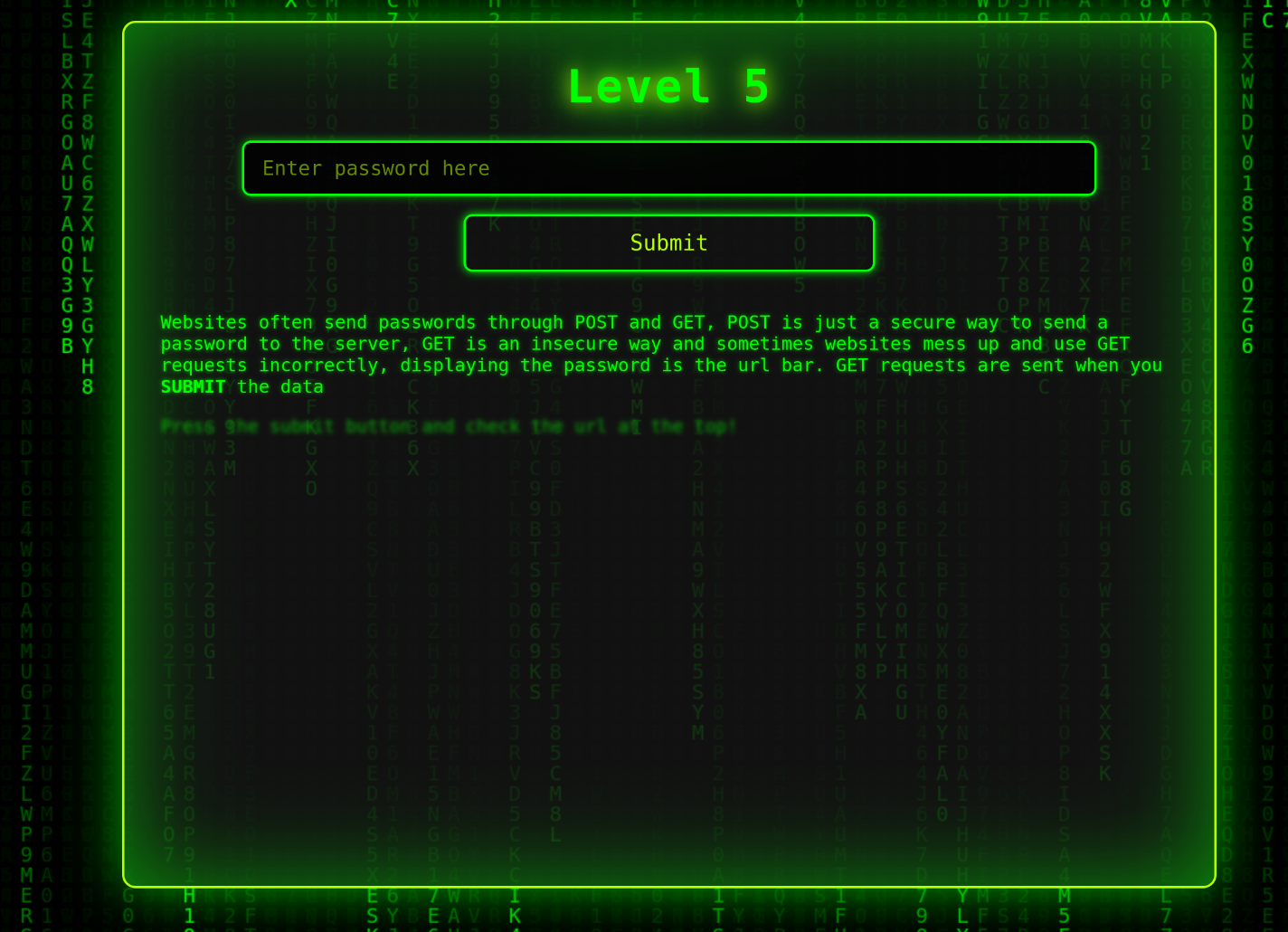


This webpage talks about general hacking such as what is hacking and the history of hacking, which the user doesn’t need to know but it’s a fun read as it tells them about famous examples of hacking and how they were achieved. Of course, we needed some more CSS so that the webpage looks nice, because the webpage would use the same elements from other webpages, I decided to put the main and separate resources in the same CSS file saving a bit of complexity. Due to the file not need extensive JavaScript, here I also just embedded the onclick events within the divs themselves allowing them to redirect to the respective page they pointed to.

###### The hackerman levels – An awesome game idea.



The hackerman levels were thought of as a minigame that the users can play as a fun little tutorial into hacking, each level gets progressively harder (but provides hints that pretty much tell you how to complete the level) making it a fun and exciting challenge for them!



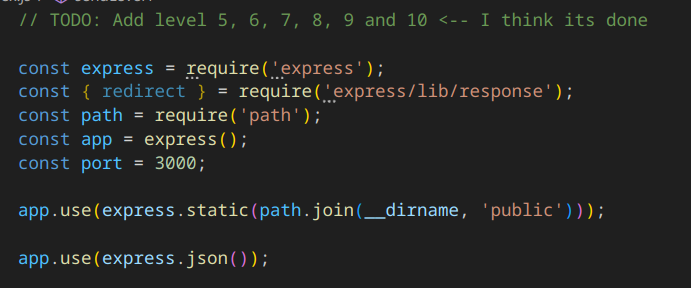
The Code here was very expansive as it required the JavaScript to dynamically load each level onto the page without refreshing it so that it would be sleek and effective even on slow networks, as each level was only around 20kb in size meaning that nearly every internet speed can effectively load the levels.

The code may look daunting but in reality, it’s just running some checks to see if the server accepted the password or not and then updating the onclick function of the submit button to handle the new level data.

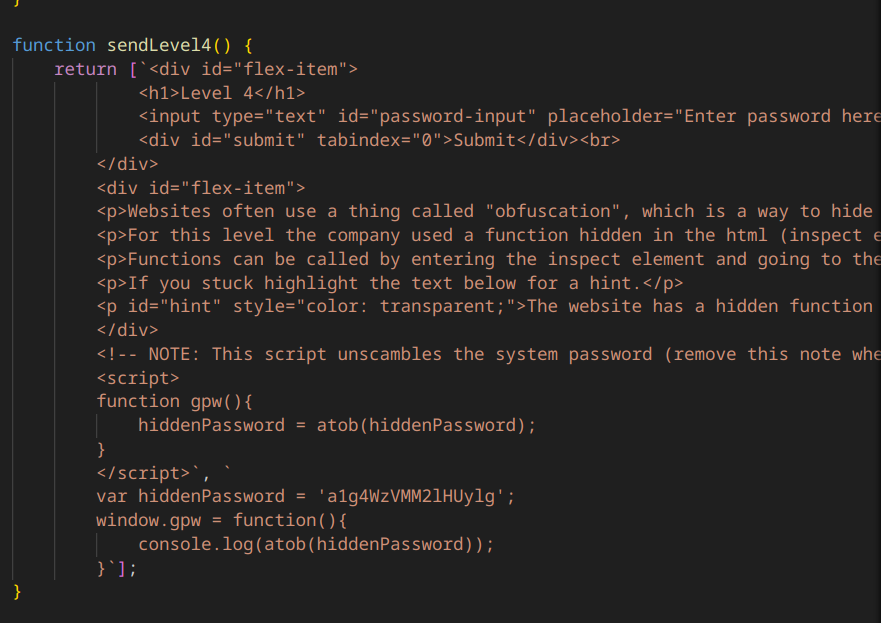
Using the server backend, it allows the passwords to be stored securely so that if a real hacker attempts to get the passwords they can’t unless there is a hidden bug that hasn’t been found.

###### The server - How it all works.

Each level is then handled using the backend script which uses the JavaScript runtime known as NodeJS, this allows the user to interact with the server, allowing them to send GET and POST requests that contain the data that is needed to validate the passwords and load onto the next level.

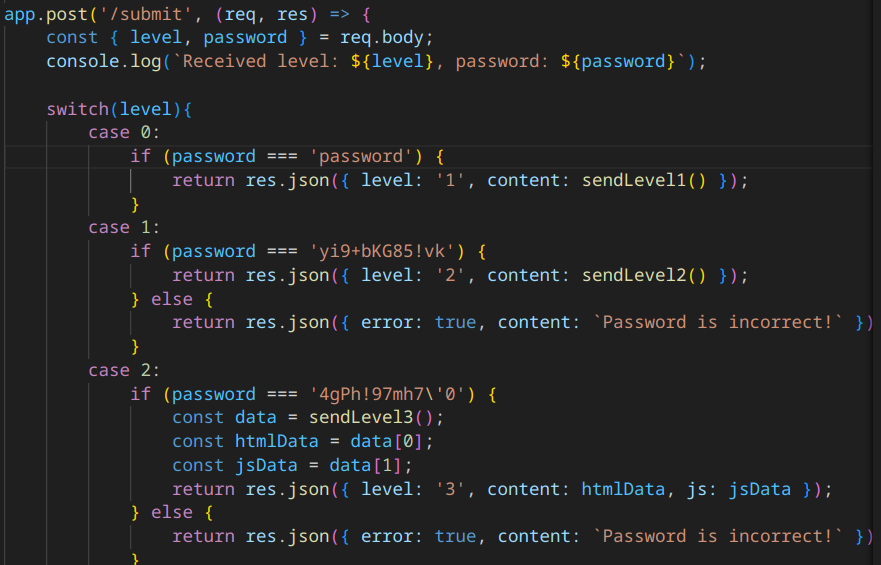


Here is the server script where it imports all the needed libs and sets up the port and file path that we will be serving the site on, from the code it tells the server to start a local website hosted on the port “3000” which isn’t used by many applications so the chance of a collision is very low unlike using something like port 21 which is used for FTP. Next the code tells the server to use the directory path of the server and then the “public” folder so that when we serve the html files, inside them it knows where to receive the files from.



Each level has its own function, I consider this function to be helper functions as otherwise the code that sends the html content back would get really messy and therefore not very maintainable. The helper function simply just returns the html content for that level and any JavaScript that the level needs.

###### Handling fetches and sending data.



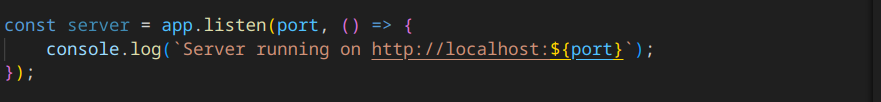
Right now, we are setting up a path for handling the “fetch” request that the user makes to the server to load the files for the next level, the fetch will pass the level that the user is on and the password that they have inputted, we then use a switch case statement\* to figure out what code should be run based on the level that has been received.

After it matches the level number to the correct case statement it runs a simple selection to check if the password that the user inputted is the correct password hardcoded on the server (you can see the password in the screenshot)

Once the password is validated and is correct (or incorrect) the data is sent to the user where the function that we discussed earlier either displays the error notification or loads the next level.

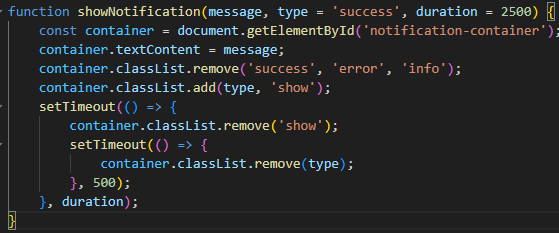
\*Switch case is just a more efficient way of using if else in languages like JavaScript as they provide direct access to the code that you want to run rather than checking multiple if statements

*How we start the server*

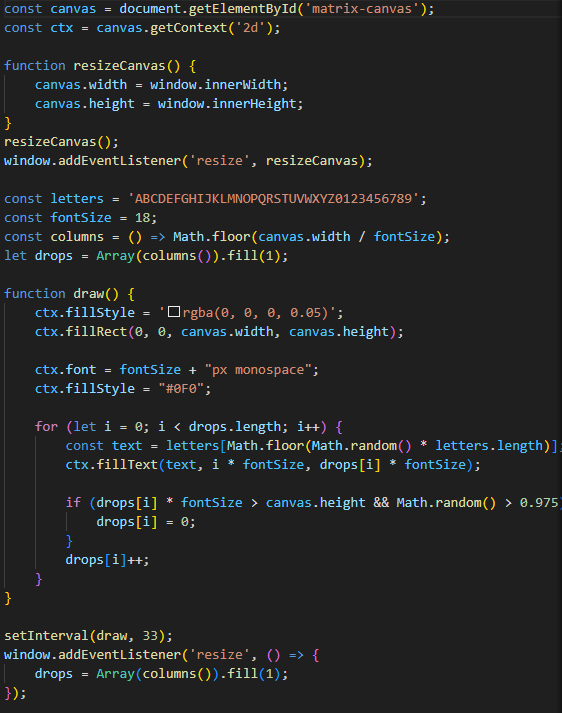


Self-explanatory, this creates the server and assigns the constant variable “server” to it, so it starts the server on localhost with the port that we set earlier, we assign the variable in case we need to terminate the server at any point.

###### The user scripts – How the JavaScript works



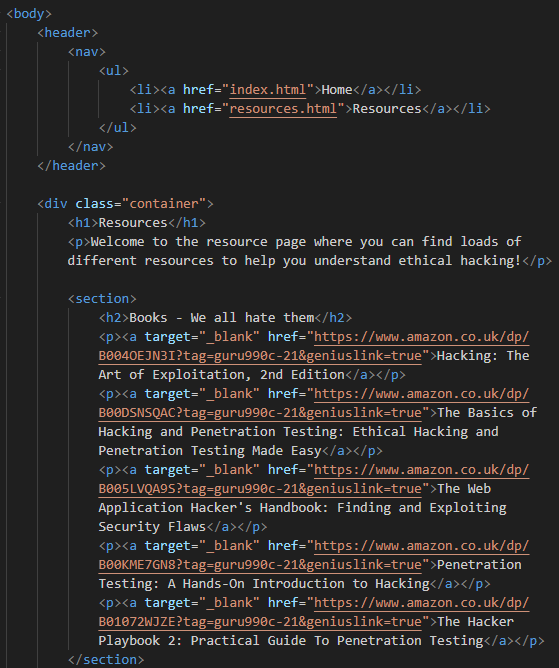
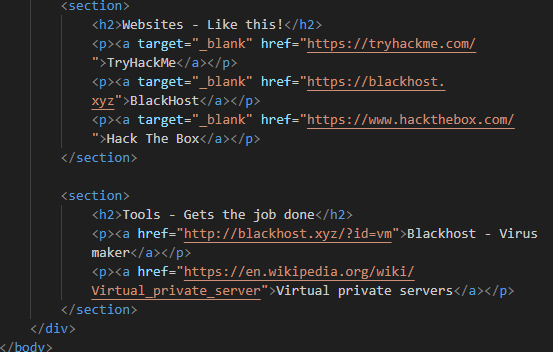
Here we have a basic function that displays a notification so when the user submits the password and the server sends back its response the users webpage can display a box saying whether or not they got the password correct. (It also shows if a saved level was loaded.) after a little bit it then fades out the notification, talk about cool animations!



Knowing what happens here isn’t really that important as it just creates the matrix effect in the background of the hackerman levels:

###### An overview – A fast review of the HTML files

Each page has its own associated HTML content that the user loads and views using their browser of choice.



The resource page uses the CSS we made before from the “abouthacking” webpage to help display the contents of the page, essentially this page just provides some links to learning hacking such as various hacking books, hacking sites and more.

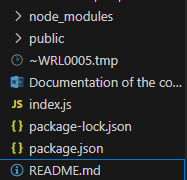
This allows the user to quickly access excellent learning tools from the website.

The webpage takes advantage of the “section” tag to define each separate method of learning, books, websites and then tools, within each section it has a header and then a paragraph with most likely an anchor tag that leads to the book/website/tool.

Each webpage, other than the hackerman levels, has the same nav bar at the top that helps the user reach each webpage directly without having to retype the URL like a caveman. Because it’s a separate element like I said before its able to run on each page by just adding it and the associated CSS.

###### File structure – How neat can it get?

###### The basics



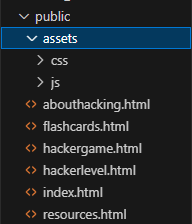
Very simply the basic layout looks like this where we have out $ROOT directory which contains the follow files:

* node\_modules
* public
* index.js
* package-lock,json
* package.json
* README.md

The files/folders; node\_modules, public, package-lock and package.json help setup the node server so that it can handle server requests from the user’s browser, displaying the correct web contents for each URL that they reach.

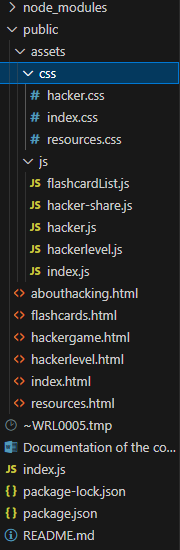
*What about index.js?*

Index.js is the main server file, that we explored before at [the server script](#_The_server_-) which we tell node/nodemon that that is the script that should process all the html files and fetch requests that the user can make to the server.



###### Served files – The meat and potatoes.

Looking into the folder “public” reveals where the html files and the related JS/CSS files are found that the server serves to the user such as abouthacking.html, hackergame.html and more.



In the end all this makes the server files look very organised and easily maintainable due to the structure, if I hadn’t done this I, myself, would’ve most definitely messed up somewhere setting up the paths for the files and folders when serving the HTML files.