

PC Assembler

Alfie Fitch
92184

A website which allows you to choose what components you wish to put into your computer, notifying you of any compatibility issues and the price of the total build once components have been chosen.

The Sixth Form College Farnborough, Centre No. 58319

Contents

Analysis:

Problem Identification	4
Stakeholders	4
Research	7
Features and Limitations	10
Requirements	13
Computational Methods	14
Success Criteria	15
Decomposition	17

Design:

GUI Design	18
Website Data requirements and Algorithms	21
API Decomposition	23
Full list of planned API Methods	24
API inputs and outputs	25
API Validation and Algorithms	26
API Test Plan	30
Database Decomposition	31
Database Variables and Validation	34
Database Algorithms	37
Database Test Plan	39

Development:

Database Setup	42
Web Server Setup	45
API Development	47
Website Development	54

Testing:

Testing Table	75
Video Testing	78
Usability Testing	81

Evaluation:

Evaluation of Success Criteria	87
Final Limitations of Project	89
Maintenance	90
Limitation fixes	91
Future Development	92
Conclusion	93

Analysis

Problem Identification

Many people who wish to build a computer themselves feel overwhelmed with the large selection of components, They are unsure which components they require or which are the best for their needs. On top of this, novice builders are unaware of any compatibility issues which may cause problems and mean the build does not work. This may put them off building a computer themselves which is very easy with the right components or mean they spend more money using a company to pre-build their computer and select generic components which are not fit for their needs.

Stakeholders

The website is available to all but will be used primarily by those who wish to build a computer themselves. The website allows them to choose the components and provides the estimated cost of the build as well as a compatibility checker. The user wants an easy and sleek experience which requires little or no previous knowledge about how to use the website. The user is looking to get an idea or a plan of what the PC they want and how much it will cost. The information about the build needs to be easy to see and understand for the user.

Amazon would also be a stakeholder in the website if I can add the price provided by Amazon to the website. This would mean rather than people using my website to buy the components, they are sent to amazon.

My website may also be used by people who build pre-builds for other people as it will allow them to provide a price to the customer as well as ensuring they select the correct parts and that there are no compatibility issues. The system that provides issues with compatibility will need to be accurate as it will influence how and what people buy. The system may also not flag a build that has issues causing the user to buy a computer that may not work.

The stakeholders and users of my website might be first time users who have never built a computer before, this means the website needs to be easy to understand and navigate. The website also needs to ensure all components have been selected and nothing is missing that is required, such as ram. The website may also be used by those who have built multiple computers before but just want to ensure that the compatibility is ok and want to work out the total price of the build.

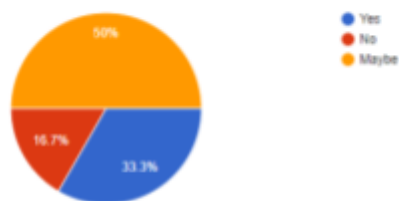
The website is appropriate to the needs of the stakeholders as it allows them to build a computer knowing that the components are compatible and if the finished build does not post, they know that the issue is not to do with any compatibility issues.

Another stakeholder will be the company who hosts the website, They will be in charge of ensuring the website is up at all times, however, this can easily be done from home, removing them as stakeholders.

Stakeholders Survey Responses:

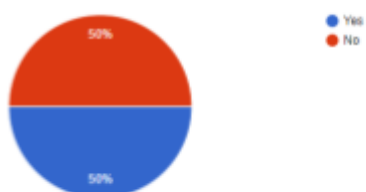
I completed a survey about the features my website will include. The results showed:

Would you use a dedicated website to select computer components?
6 responses



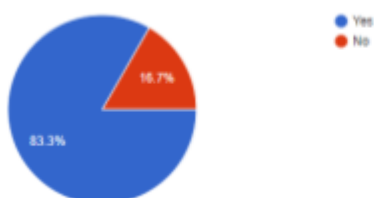
Only 16% of people who responded would not use a website to help choose components in their computer. This shows that the website will be used once completed.

Would you like examples or recommended computer Builds?
6 responses



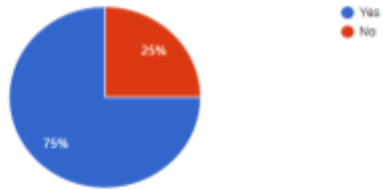
Only 50% of users would like example or recommended builds which shows that it is not a important feature and a lot of time shouldn't be wasted on it if it cannot work.

Would a compatibility checker be useful and help with the build?
6 responses



83% would like a compatibility checker which makes it one of the most demanded features and therefore one of the most important ones to get right to ensure visitors to my website.

Would the total price of the build be useful?
8 responses



The total price of the build is also a highly demanded feature which can be provided by the amazon link to calculate the total price.

Would you like reviews of the components available?
8 responses



Reviews of the components available was 50/50, this means that the feature is not that important and can be skipped if it takes up to much time to code.

Research

Amazon:

Amazon has some features which will be implemented into my program. Amazon has a library of computer components which can be selected and added to a basket which displays the total price of the components.



The image shows an example of the items sold by amazon and the information and layout of how they are displayed to the customer.

Information includes energy rating and customer review data.

The shopping basket in Amazon which shows the components chosen and the total price of the components in the basket. In addition, Amazon recommends other items that are available.

Amazon may also allow for the current price of the components to be embedded into the website which will allow for an easy and always up to date way to buy and provide the price of the final build.

By providing a link to amazon instead of using my own website, people will feel safer buying the product as it is from a well known company.

Amazon does not provide any compatibility checking, it also may recommend items that are not compatible as they sell much more than just computer components.

PC Specialist

PC Specialist is a website for configuring pre built systems. My website will be similar in the fact components for the computer can be chosen but it will not provide the finished build or allow you to buy components on the website directly.



This page allows you to choose all available components as well as operating systems and the case. This will be similar to my website and the system I will use to select the parts.



PC Specialist also runs a compatibility check for the components chosen. My website will also do this. I am planning on doing this through the use of tables and presence checks.

The compatibility checks prevent users selecting components that may not work together causing reduced speed or it not working at all.

PC Specialist may also be useful for the information on the different components and the storage of the information. The way PC specialist allows for selection of the different parts is not easy to use for those who are new to building a computer as it does not show information about the device and does not allow for users to search for all components. The layout of the drop down menu is full and quite hard to read through as the information is close together with other components which may also cause people to read information about a different component

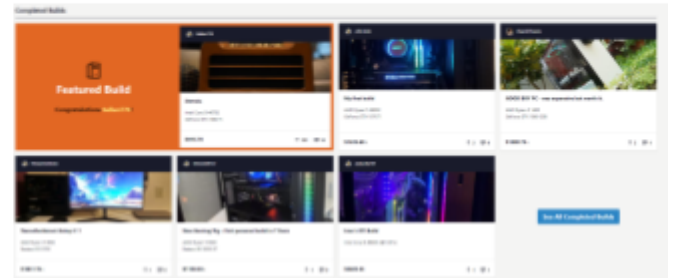
PC Specialist has a different method of selecting components for the build which may be harder to implement. The drop down menu to select the parts is harder to implement as one page will have a large amount of code and repeated code, The other method of giving each component section its own page makes it much easier for both the user to see as well as the developer to integrate.

PC Part Picker

PC Part Picker is similar to the website I plan to build. Many features are similar but I want to make some changes to it that I believe will make it better.



PC Part Picker provides some good pre selected computers for different categories, such as budget and gaming. This provides the user a good idea as to what the best specification is for the price they have in mind.



PC Part Picker also provides people to upload there builds to the website. This provides people with inspiration on what to build and how they want their pc to turn out.

PC Part Picker uses multiple techniques I wish to implement into my website, for example, unlike PC Specialist, in order to select parts its takes you to a seperate page which provides more room and a better layout, as well as the ability to add filters and search through the products. This makes the selection process much easier, especially for someone who is building a computer for their first time.

PC Part Picker also uses amazon embedded links in order to provide the price for the products. This allows the users to easily find the correct place to buy the component as well as an up to date, accurate price which can be used to calculate the final price of the build in the basket area of the website.

Features and Limitations

- Compatibility Checker

The Website will provide a compatibility checking of the selected components. This will allow for peace of mind with the components in the build. I plan to use tables that hold information of what type of CPU Socket the motherboard uses as well as what type of ram is selected to allow for the program to check the information and compatibility. At first, only a small amount of options will be available to ensure the system works as intended before hundreds of different components are added to the tables. This feature is one of the most important and essential as it provides the basis for why people would want to use the website instead of choosing components themselves. The survey completed also showed that over 80% of the respondents would want this feature.

Limitation:

The compatibility checker is one of the most difficult to implement as it requires comparisons and links to multiple databases as well as the information required in the database to decide the compatibility, this may mean that the components that are compatibility checked are reduced, for example, only CPU and motherboard may be checked whilst the PSU may not be compatible but not flagged.

- Total Price of build

The website will provide a total price of the components selected by using the embedded price provided by amazon. This will provide an up to date and accurate price as well as an easy way to purchase the items by clicking the button and taking you to the page. Multiple websites have used the embedded function so it should be available for use by my website. By only using amazon, the user will either have to look up online to find any place that may sell the item for cheaper or just use amazon. In the future, more websites can be added such as ebay. This is one of the most essential features as it makes the website more user friendly as once they complete the build they can see the total price live and immediately rather than having to search each component individually.

Limitation:

The basis of the system which will show the live total price of the build uses amazon, this means that the only way this can work easily is if amazon already has an API system that can be linked to or embedded into the website. Therefore if the amazon system cannot be linked to the website, then a price will have to be selected which was accurate at the time and may have changed since the database was first created.

- Part Selection

The website will use a drop menu with a search bar to allow for the selection of parts. These will be for all components as well as the operating system and storage system. This will allow for an easy way to select the components. However, this may be difficult if you don't know the specification of the components so this may be changed to provide more information about the item. This could be a separate page you are directed to in order to select the item. Similar to that found on PC Part Picker. Part Selection is by far the most important component of the website as without it the basis would not work and other features would become redundant. Without the part selection aspect of the website, the main criteria would not be met and the website would be basically useless.

Limitations:

The part selection feature is another hard to implement but vital aspect of the website. The basis of the part selection requires a large database of 100's of options for each component as well as the different data required such as performance and specs. This may mean that, to begin with, especially whilst the systems are being tested, a small selection of components can be added, this means that during testing, any issues do not have to be corrected on 1000's of components, rather only a manageable amount. Over time, however, more components can be added which will decrease the impact of this limitation.

- The number of available components.

The number of components available for a computer is vast. This provides a problem with how many to store in the tables and compatibility checking. This may take up a lot of storage and will take a lot of time to add each component. Therefore to begin with a smaller selection of components will be available to ensure that the system works as intended. The only way to remove this limitation would be to remove the part selection components which would, as mentioned previously, mean the website does not meet the main aim and would become useless to the user based on the results from the survey.

- Filters

The website will have the ability to filter the results of the components by relevance, price and reviews. This may make the website easier to use for someone who has a set budget or knows exactly what they are looking for in terms of the power of the machine and capabilities. This feature is not important as if it is not implemented, the function of the website is not affected, however, the feature would be a nice addition, and may increase how user-friendly the website is.

Limitations:

Filters are although not vital towards the website an important feature to have due to the large array of possible parts to pick from. Filters may be hard to implement depending on how the data is stored and whether it has been cleaned up so the important data is easy to pick out by either a search term or filter term and formatted correctly. This limitation will be decreased if a smaller number of components are used as it is easier to search through a smaller database if there are any formatting issues as they can be fixed manually easily.

- Part Reviews

Reviews of the specific parts on the website would provide the user with better ideas of what parts to pick and would be beneficial to a new user or first time builder, however, the survey showed that only 50% of the users would like to see this feature, which means that the basic functionality of the website would not be effected without it. In addition, adding the feature may be quite difficult and time-consuming as reviews will need to be collected from another database such as Amazon and implemented into the website and distributed to the relative component correctly. Therefore this feature is not one I am focusing on as others are more important to get the website working as wanted.

Limitations:

Part reviews will depend on the users of the website writing reviews themselves, therefore when the website is first started, the number of reviews will be minimal if any at all, this means the feature is unlikely to be used, however, it may be possible to add a link to Amazon reviews which will mean there are 1000s of reviews already written which can be displayed on the website.

Requirements

The environment we are using is IntelliJ IDEA, therefore this is a requirement in order to program the website. A Java Development Kit is also required to create the executable code and interpret the code written so it can be run on the website without errors and as quickly as possible, meaning the website experience is as fast and efficient as possible. A Web Browser, such as Chrome will be used in order to access development tools provided as well as access the website in general and be able to interact with the user interface and interact with all the features the website will offer. SQLite Studio will also be used to create and manage SQL tables. The website GitHub will be used for storing and updating the code online as well as sharing the code between teachers, college computers and home computers to allow me to work on the code wherever I am, rather than just being able to at college. Bash will be required as the command line interpreter to test the interaction with the server.

The end-user will require a decent web browser such as chrome in order to access the website without any issues as some features of the website may not be compatible with older browsers.

One requirement for the website is the addition of the price provided by Amazon using an embedded price. This will use code created by Amazon added to the code of the website to provide an accurate price of the product. The Amazon link will also direct the user to the correct place to purchase the product and therefore complete the users build experience.

The tables holding the information on each component are required to be linked with each other to allow comparison of the selected components and depending on which table they are in or a single column in the table if they are compatible with the other parts.

My website requires no hardware to develop or operate except for a server to host the website when completed however that is not required to complete the project. The website is online and requires nothing from the outside or other websites except for the price provided by Amazon or other websites in the future.

In order to code the website, a computer with at least 4GB of ram (preferably 8GB), a stable and strong internet connection as well as ideally a quad-core processor and is able to run both IntelliJ and chrome is required. The website when finished should be able to run on most browsers as intended but may require Javascript to be enabled.

Another requirement is the hosting of the website. During development, this can be done on the computer which is being used to code it, however, once the website is deployed and testing has begun, a dedicated computer or server will be required in order to allow other people to visit the website and begin testing and use the finished website.

Computational Methods

- Decomposition, the breaking down of the project into smaller components will be used to help design and code the website easier than as a whole. The pages of the website, tables and code itself can be decomposed. In my project, decomposition will help in the selection process by splitting each component into different web pages where they can be selected and, even further decomposed by giving each component its own web page with details.
- Libraries and Modules will be used in order to save time and reuse code that was created before and is known to work.
- Problem Recognition and Procedure Identification will ensure that the issue and what is required to fix the problem can be done and are in place.
- Inputs, outputs and preconditions are required as what goes in, out and preconditions need to be thought about beforehand and that the code is able to handle it.
- Abstraction will be used in order to better understand what needs to be coded for the intended features to work and what the most important pieces of code are.
- Heuristics could be used to estimate the user's choice for components depending on previous selection by people with the same already selected components in order to speed up the compatibility checking.
- Concurrent processing can be done to help ensure that new code being written works and is compatible with previous code and what needs to be fixed can be spotted easily and quickly.
- Caching will be used to store the most frequently accessed web pages on the site as well as the build that is most often picked to help speed up other processes required.
- Divide and Conquer can be used by the program to work through each component to ensure the compatibility with each other one by one.

Success Criteria

1. Users are able to access and load the website correctly.
2. Users are able to log in to an account and sign up for a new one.
3. Users are able to select components for their computer build.
4. The Users are able to delete their account and change their password, as well as a working forgot password system is implemented.
5. The user can easily navigate and has an understanding of navbars.
6. The user is able to save their build to come back at a later date.
7. Users are able to browse components available and access the information about them.
8. Users can purchase the components on Amazon directly from the webpage.
9. Users are warned if chosen components may not be compatible with each other.
10. Users are able to see recommended and showcased completed builds.
11. Users can select as many or as little components they need.

Design

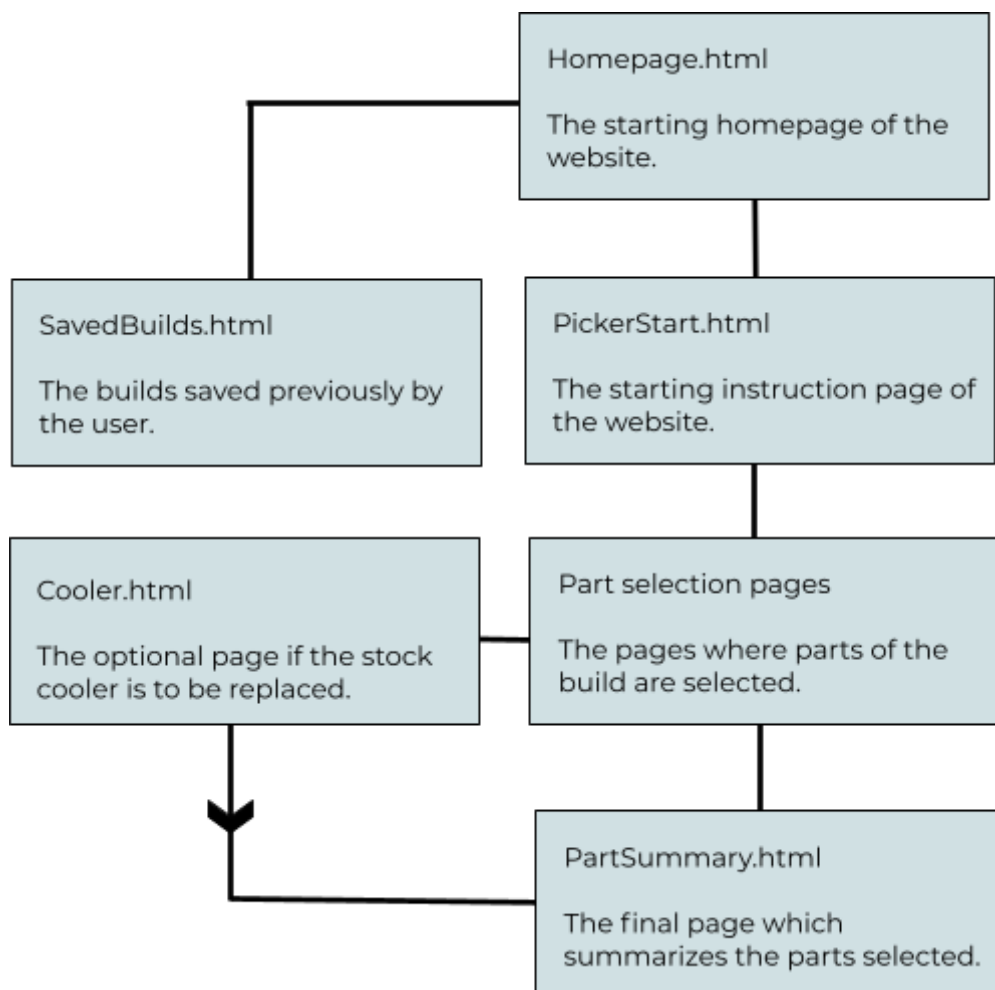
Decomposition

Sitemap:

Pages:

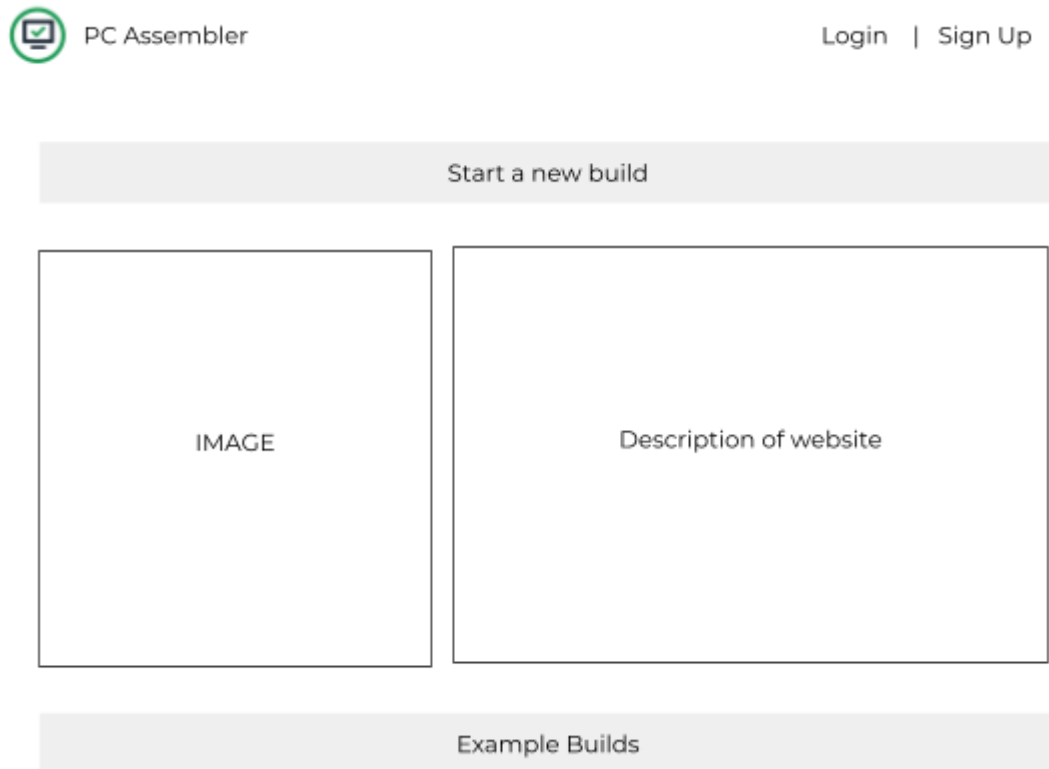
The website will have a homepage which shows the different options the user can do such as looking at recommended builds as well as moving onto the part picker itself. The part picker will consist of multiple pages including a starting page which explains the process and the instructions on how it works, It will then have another page for selecting each component of the computer and a final page after which shows a summary of the selected parts and the compatibility of them. The website will also have a page which allows the user to see previous builds they have created if they decide to save them on the final summary page of the part picker.

Proposed pages:

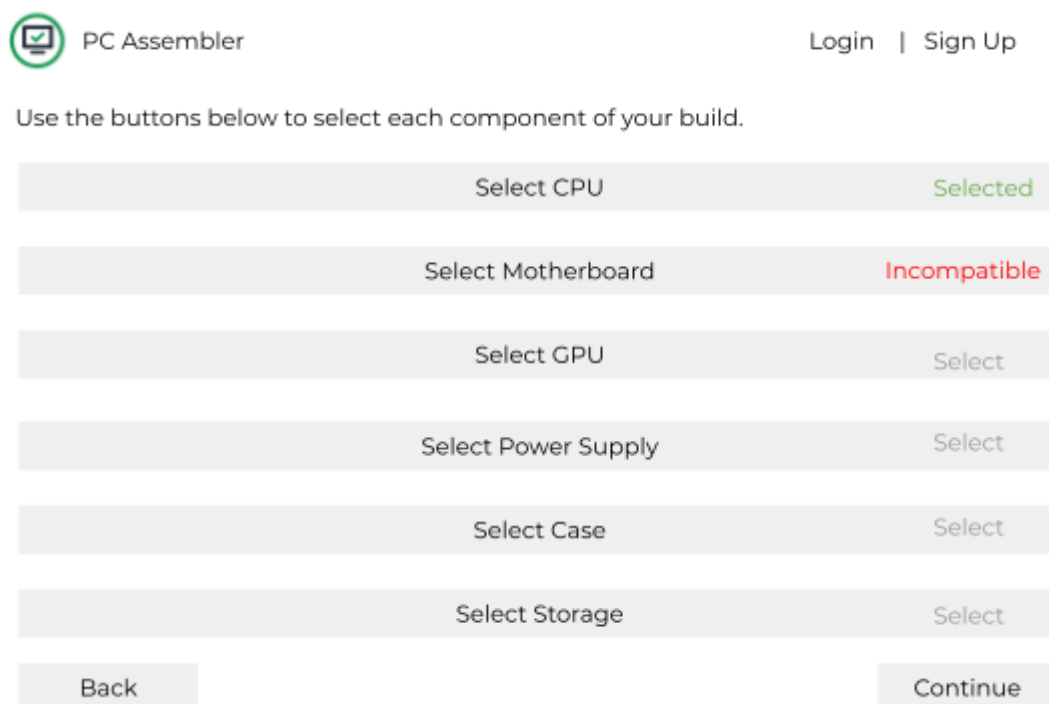


GUI Design


Homepage:



Selection Page:



Specific Part Selection Page:

 PC Assembler


Login | Sign Up

Image	Part Name	Price
Image	Part Name	Price
Image	Part Name	Price
Image	Part Name	Price
Image	Part Name	Price

Back

Next

Review Page:

 PC Assembler

Login | Sign Up

Selected Part 1	Compatible	Amazon Link
Selected Part 2	Compatible	Amazon Link
Selected Part 3	Compatible	Amazon Link
Selected Part 4	Incompatible	Amazon Link
Selected Part 5	Compatible	Amazon Link

Back

Save Build

HTML Tags:

The tags used for the website will include:

- For text which uses multiple tags such as `<h(number)>`
- Hyperlinks which allow the user to find the specific Amazon page for the part selected. The tag used is `<a>`
- Images will be used for multiple parts of the website including the logo in the top left. The tag that is used is ``
- Buttons will be used to select the parts as well as progress the pages. The tag that is used is `<input type="button">` when inside a form and `<button>` outside a form.

Usability:

The website has multiple features which will ensure the website is user friendly. For example:

- The website is easy to navigate with buttons at the bottom of the page to return to the previous pages, The website is also not crowded with lots of options and information that may confuse the user and make it hard to find the information or option they require which means the website will also be easy to use for new users and older people.
- The website will be responsive and be accessible on all types of devices such as iPads and computers. The website has been designed to work with touchscreen displays as well as traditional mice.
- The images used on the website are small and there is not a large amount of them, especially the larger ones which will mean the website is much faster to load and more responsive as less has to be downloaded from the internet into the ram and cache on the device.
- The website will have a consistent colour scheme and design which means the website is easy to understand and follow along with. It also makes it easier to pick out more important information such as a confirmation to delete command which can be coloured differently to draw attention.
- The website has been designed to ensure the user has control of the website by allowing them to continue to the next page rather than automatically change when they have selected their component.

Website Data requirements and Algorithms

Pages	Data at page load	Data on interaction	Algorithms
<p>Index.html</p> <p>The main homepage of the website</p>	<ul style="list-style-type: none"> • Cookies stored in the web browser such as whether the user has logged into the website before and if they have changed any preferences depending on the layout of the website. • Image files need to be available to users not on the LAN and loaded into the webpage so they can be displayed quickly. 	<ul style="list-style-type: none"> • Cookies to be saved to the browser if it is the first time the user is visiting the website. • Login Data should be accessible from the database to allow for login to occur from the login popup if the user has not already logged in. 	<ul style="list-style-type: none"> • JavaScript function that separates and reads the cookies in the browser to collect relevant data and pass it on to the HTML code to change the homepage. • JavaScript function to open the login popup and submit the relevant data to the login function.
<p>NewBuild.html</p> <p>The starting page when beginning a new computer build</p>	<ul style="list-style-type: none"> • The databases storing all available components need to be loaded so the user can make the first selection which will narrow down future results. • Login data so the information that is inputted is saved in the correct user's database to prevent crossover of data and another user being able to access the build. 	<ul style="list-style-type: none"> • The database storing the available components need to be updated on interaction as the users selecting various components will change which components need to be shown and therefore change the HTML respectively. 	<ul style="list-style-type: none"> • JavaScript function to read the users inputs and change the data on the HTML depending on which components the user has selected. • JavaScript function to save the progress of the computer build either every 30 seconds or any time a change to the component selection is detected.

<p>Featuredbuilds.html</p> <p>The page which shows the current featured and recommended builds.</p>	<ul style="list-style-type: none"> • Which specific builds are being displayed along with the images of it and the specific components to be shown. • The date to show that the featured builds page is up to date and changes regularly so same builds are not shown all the time. 	<ul style="list-style-type: none"> • Which component details to show depending on which build the image the user is hovering over. • The correct links to more information about the build when the user clicks the more information link on top of the image. 	<ul style="list-style-type: none"> • CSS function to display information when the user hovers over the image. • Javascript function to pull the build information and data from the database as well as change the link depending on which is shown.
<p>Mybuilds.html</p> <p>Page to access previously started or saved builds.</p>	<ul style="list-style-type: none"> • User details - Userid and password of the user to show correct saved builds and verify the person accessing the information. • Saved builds - the saved builds stored in the user's database which needs to be pulled and shown in the correct locations. 	<ul style="list-style-type: none"> • Which build the user clicks on needs to be loaded into the builder page so the parts selected before are already put in place. • Which button the user presses such as the load button or deletes button which will trigger separate javascript. 	<ul style="list-style-type: none"> • Function to load the correct data and pass it on to the next page. • Function to load the login details of the user and select the correct builds from the database. • Function to delete builds that the user wants to delete.
<p>Checkout.html</p> <p>The page to show the final build price and links to buy the components from.</p>	<ul style="list-style-type: none"> • Which components were selected by the user on the part selection page. • If the user is retrieving a build that was saved previously then all components need to be loaded correctly from the database. 	<ul style="list-style-type: none"> • The amazon API and data loaded from the database to show the correct link when the amazon button is clicked as well as the correct price that is stored in the database updated regularly. 	<ul style="list-style-type: none"> • Javascript function to read information from the stored components database and API from amazon to link into the component page on amazon.

API Decomposition

Page Name	API's on load	API's on request
index.html	<ul style="list-style-type: none"> Link to featured builds database to show the correct featured builds for that time. /featuredbuild/get Check cookies to see if a user is already logged in or if they are a brand new user. /cookies/get 	<ul style="list-style-type: none"> When 'mysavedbuilds' clicked then login status checked so saved builds can be loaded correctly with correct data. /mysavedbuilds/get Collect data from the database of the components on the featured builds when the specific build is clicked and/or hovered over /featuredbuildscomponents/get
checkout.html	<ul style="list-style-type: none"> Amazon API link to show checkout button and the correct price for the specific computer component. /amazon/get Calculate the overall price of the components using each part assigned cost from the most recent date. /totalprice/get 	<ul style="list-style-type: none"> Check cookies if the user has signed in when they click the 'save build' button so it can be saved in the correct place. /cookies/get When 'save build' clicked, data of components sent and saved to the 'savebuilds' database. /savedbuilds/update
newbuild.html	<ul style="list-style-type: none"> Connect to the component database depending on which component tab is selected and load the components images, details and price /componentdatabase/get 	<ul style="list-style-type: none"> Connect to the amazon API when the component is selected to retrieve price and show a link to where the component can be purchased. /amazon/get
mybuilds.html	<ul style="list-style-type: none"> Link to the database where users saved builds are stored and load the correct information into specific fields. /savedbuilds/get 	<ul style="list-style-type: none"> Send the build data and component information to the checkout page when the user wants to edit build or access component amazon links. /savedbuilds/list
featuredbuilds.html	<ul style="list-style-type: none"> Link to the featured builds database to show more builds than on the homepage, may be using a different table or same, depending on if the homepage can show a certain amount of the builds in the database. /featuredbuilds/get 	<ul style="list-style-type: none"> Collect data from the featured builds database about the components in the specific build when either hovered over or clicked on to expand the image. /featuredbuildscomponents/get

Full list of planned API Methods

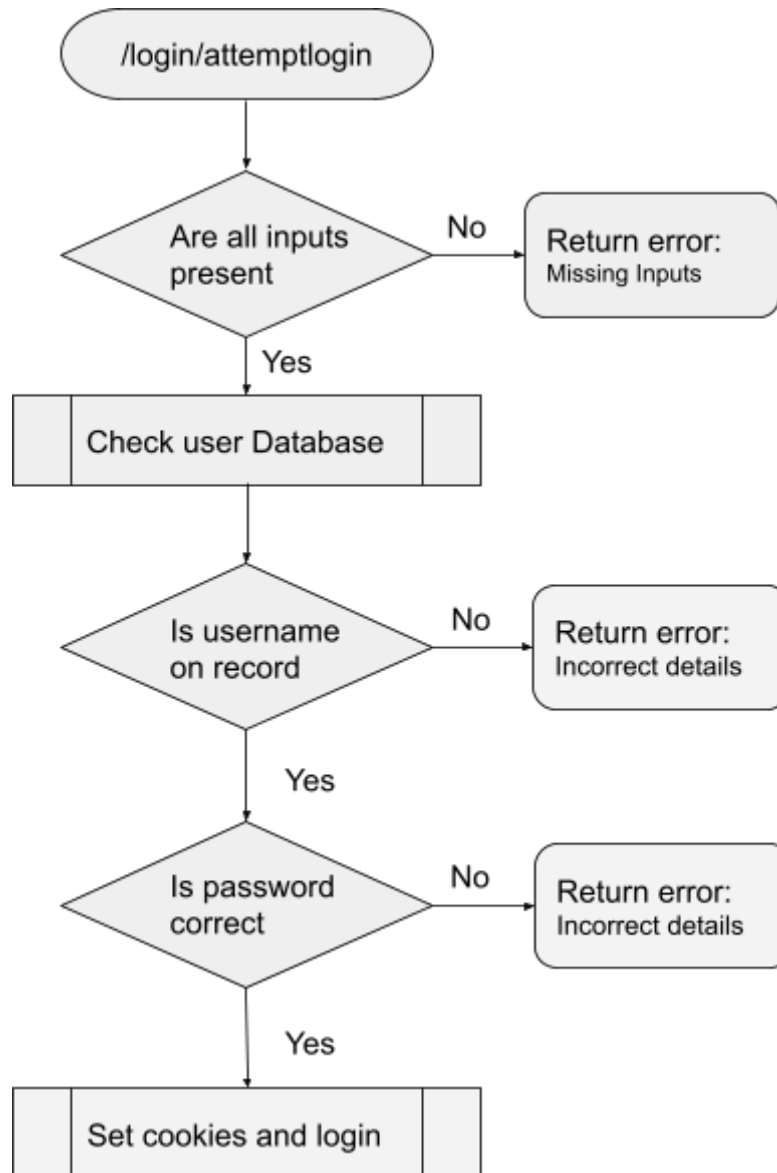
- [/featuredbuilds/](#) - The link to the featured builds table in the database.
 - Get - used to get the information stored and display it on the website.
 - Push - may be used to save data from users builds to produce new featured builds.
- [/login/](#) - Used to link with the users database and collect login information.
 - Get - Used to retrieve the login details to compare with what the user has put in and validate, also saves cookies if login is valid..
 - Push - Used to save new login details to the database when a new user signs up for the first time..
- [/amazon/](#) - Used to connect and link into the amazon database for price.
 - Get - can only use get as data cannot be sent or saved, used to collect data on the price and availability of the product as well as where to buy it from.
- [/savedbuilds/](#) - Links to the personal saved build table for each user to save information on builds that are still in progress or saved.
 - List - Used to collect the information and components on each build the user has when they are logged in and open the saved build page.
 - Update - Used to save new builds and components to the table of the user so it can be retrieved at a later date.
- [/featuredbuildscomponents/](#) - (This could be integrated with the featured build api) Links to the featured builds table, specifically the components of each build to select only the important data for that specific area of the website.
 - Get - Retrieve the component information from the database table to show in the hover over boxes and the expanded images.
 - List - Used to retrieve the component data and display it as a list.
- [/totalprice/](#) - Connects to the javascript and database to calculate the price for the parts that are selected depending on the most recent inputted price for the component.
 - Get - Retrieve the calculated price from the javascript function or the saved build in the database.
 - Update - Send the price of each separate component to the javascript function to calculate the total price.

API inputs and outputs

API Paths	Methods	Inputs	Outputs
/featuredbuilds/get	Get	FeaturedBuild	{“BuildID”: GPU, CPU, MB, PSU, Storage, RAM} Or {‘Error’, Detail of error - ‘BuildID’ not found}
/featuredbuilds/save	Push	NewFeatured	{“BuildID”: GPU, CPU, MB, PSU, Storage, RAM} {‘User’: name} Or {‘Error’, Detail of error - ‘Build details not found’}
/login/attemptlogin	Get	User, Token	{‘User’} {‘Token’} Or {‘Error’, Detail of error - user not found}
/login/save	Save	User, Token	{‘User’} {‘Token’} Or {‘Error’, Detail of error - User already signed up}
/amazon/get	Get	Price	{‘Price’} {‘Link’} Or {‘Error’, Detail of error - Product not found}
/savedbuilds/list	List	SavedBuilds	{‘BuildID’: GPU, CPU, MB, PSU, Storage, RAM} {‘UserID’} Or {‘Error’, detail of error - Cannot find build}
/savedbuilds/update	Update	NewSavedBuild	{‘BuildID’: GPU, CPU, MB, PSU, Storage, RAM} {‘UserID’} Or {‘Error’, detail of error - Cannot save build}
/totalprice/get	Get	TotalPrice	{‘BuildID’: price of components, total price} Or {‘Error’, details of error}

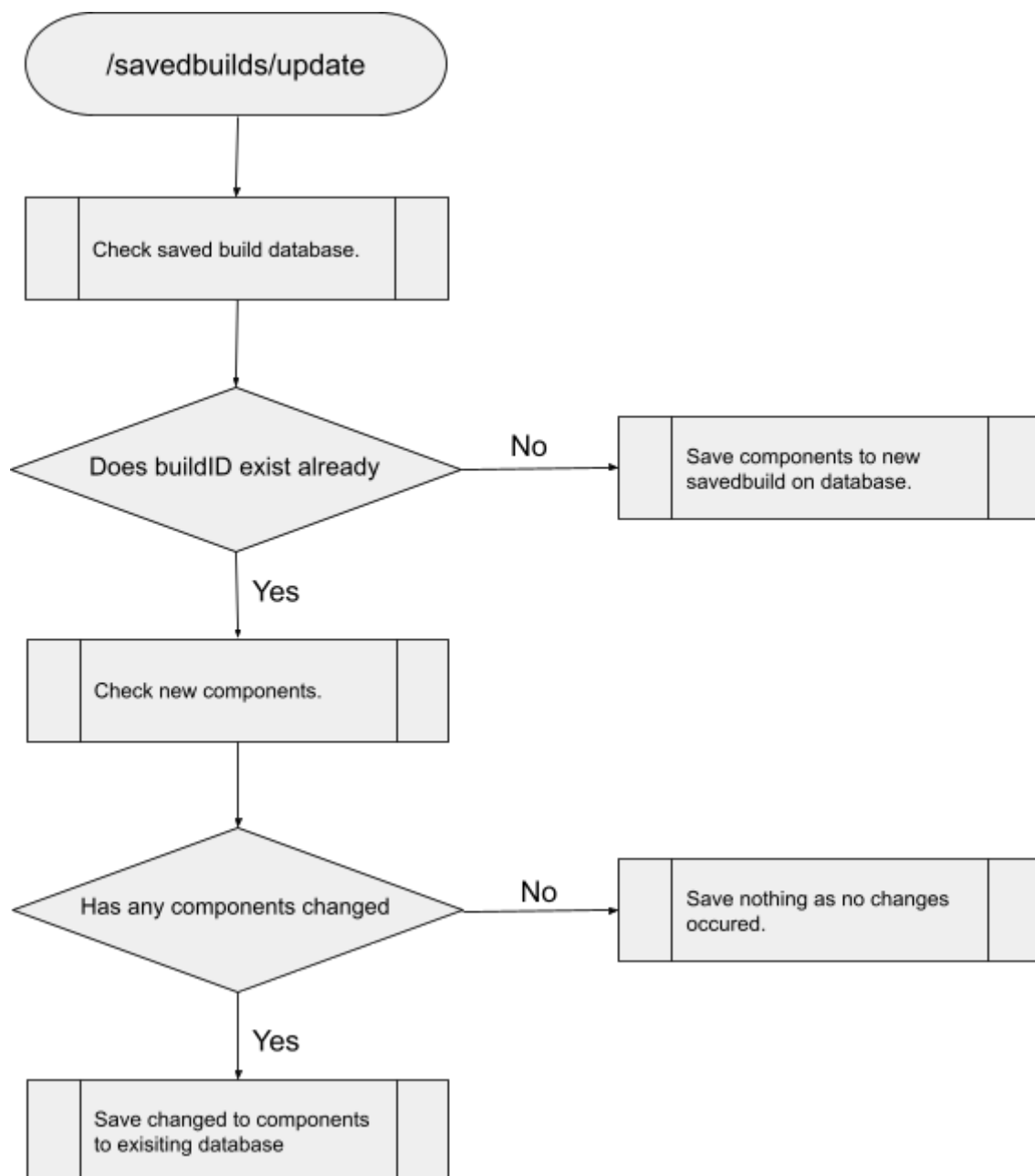
API Validation and Algorithms

/login/attemptlogin:



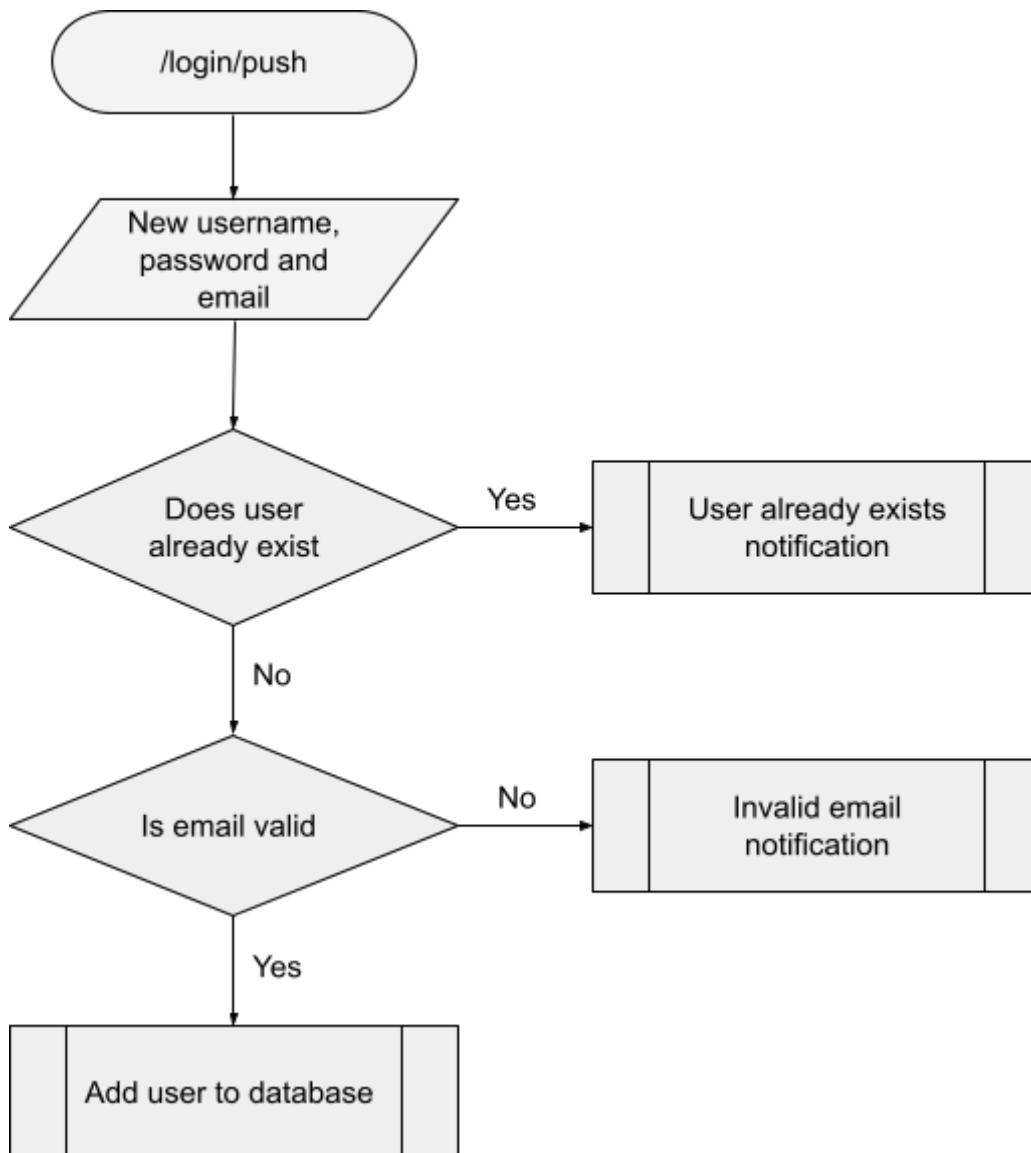
The flowchart shows the basic algorithm and process the code will use to login and validate a user on the website. The code will detect any missing fields that are required and prompt the user to input them, this will reduce the amount of errors from the search and database part as the code is not searching for a blank password or searching for a password in the username fields.

/savedbuilds/update:



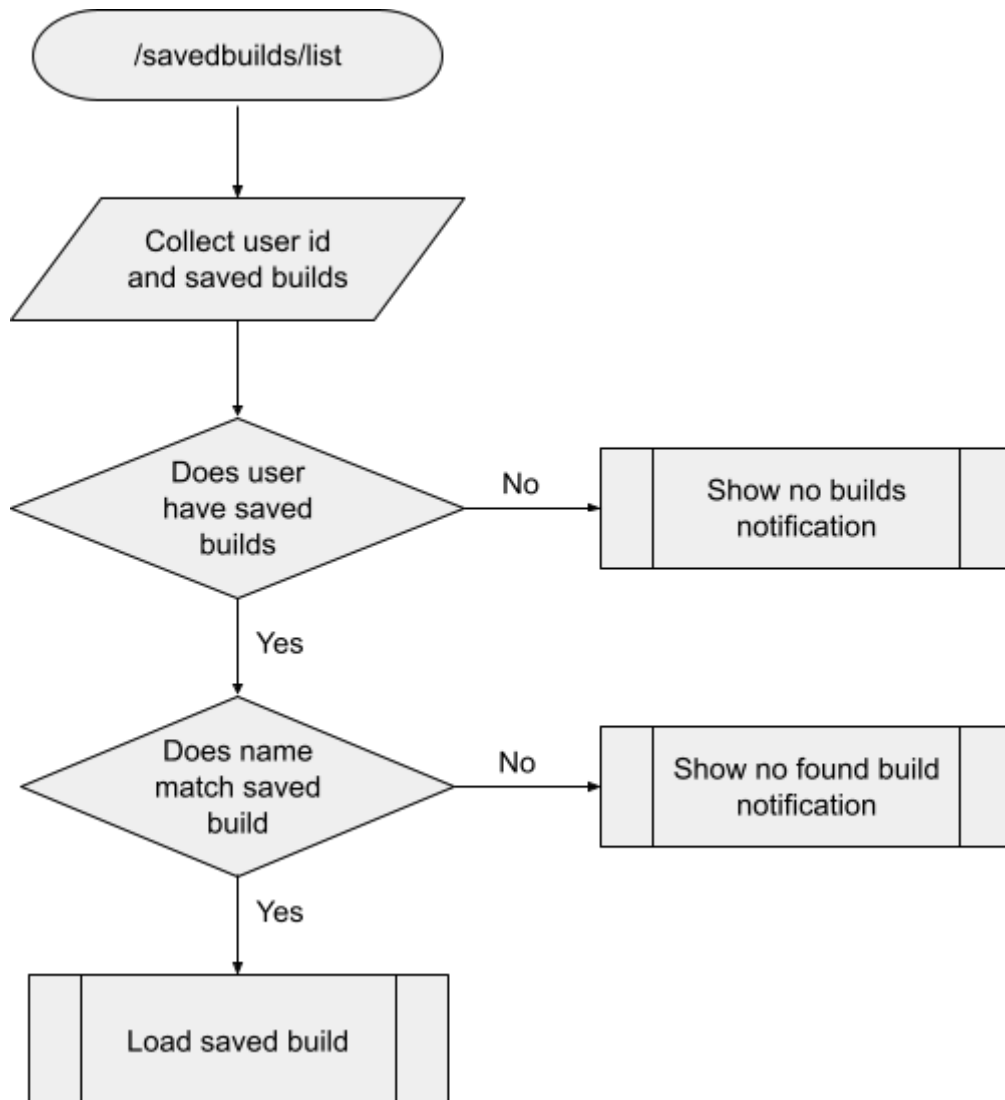
This flowchart shows the process and algorithm used to save new builds to the users account. The algorithm will check the saved build that is stored in the database as that which is shown on the users screen and see if there are any changes between the two and then save the new changes to the database if there are so the user can come back to the build as well as edit it. The algorithm will also not save anything if the user has not made any changes to make it more efficient and not run unnecessary code.

/login/push:



This flowchart shows the process and system used to add a new user to the database and allow them to sign up. The process insures that the user does not exist on the database already which ensure that the log in system has no issues with duplicates and insures the correct user is logged in, in addition, the system ensures that the email is valid and looks like what an email is expected to look like to ensure that a request new password or request password change system has the correct email and that the user is not locked out of their account.

/savedbuilds/list:



This flowchart shows the process that will be used to re open and edit the previously saved builds by the user. The process will select the saved builds from the name that the user enters, it then checks that the database for that user to see if the saved build the user requested exists and if not notifies the user, It also ensures that the user actually has saved builds before hand to ensure that the program is not running unnecessarily.

API Test Plan

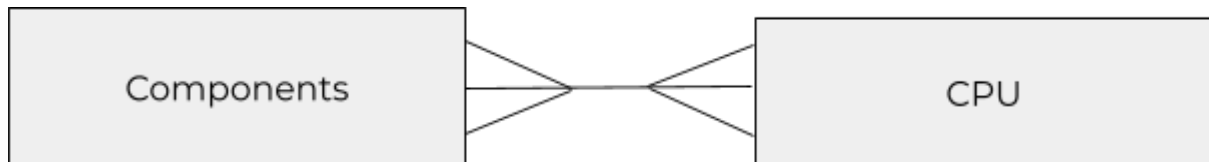
API path and Intention	Curl Test	Expected Results
/featuredbuilds/get (normal behaviour)	Curl -s localhost:8081/featuredbuilds/get	A list of the featured builds for that day or time.
/featuredbuilds/save (normal behaviour)	Curl -s localhost:8081/featuredbuilds/save -F BuildID = 1 -F GPU = 1080ti -F CPU = Ryzen 5600 -F MB = Mothername -- cookie token="a1b2c3"	Status: Saved build under userid where token = token received. (Check the database that new build has been saved correctly.)
/login/attemptlogin (normal behaviour)	Curl -s localhost:8081/login/attemptlogin -F username = "Alfie" -F password ="test"	Token: Random Value (Check database user has been assigned new token and matches cookie token)
/login/attemptlogin (invalid behaviour)	Curl - s localhost:8081/login/attemptlogin -F username = "Alfe" -F password ="test"	Error: Username not found.
/login/save (normal behaviour)	Curl -s localhost:8081/login/push -F email = "alfiefitch@email.com" -F username = "Alfie1" -F password = "test1"	Status: Saved new user (check database that new user has been saved correctly)
/login/save (invalid behaviour)	Curl - s localhost:8081/login/push -F email = " alfiefitch0@gmail.com " -F username = "Alfie1" -F password = "test2"	Error: Username and email already exists in the database.
/savedbuilds/get (normal behaviour)	Curl -s localhost:8081/savedbuilds/get --cookie token ="a1b2c3"	A list of the users previously saved builds OR Error: No Saved Builds
/savedbuilds/update (normal behaviour)	Curl -s localhost:8081/savedbuilds/update --cookie token ="a1b2c3" -F BuildID = 1 -F GPU = 1080ti -F CPU = Ryzen 5600 -F MB = Mothername	Status: Saved New Build (Check database that new build has saved correctly and to the correct user.)
/totalprice/get (normal behaviour)	Curl -s localhost:8081/totalprice/get -F Component type -F Component ID	Returns the price for the correct component (Check database that the price returned is the correct price for the component requested.)

Database Decomposition

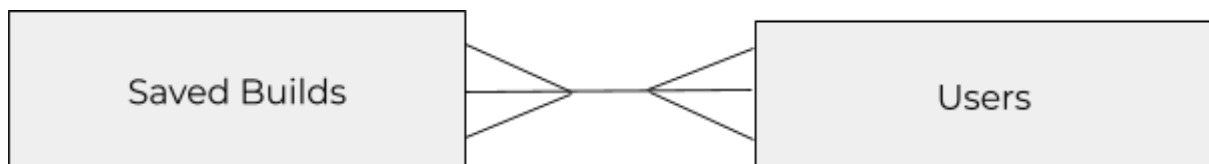
Database Entities:

- Users - To store the user details and login credentials.
- Saved Builds - To store the saved components and builds the user wants to save.
- Prices - To store the live prices from amazon as well as the fallback prices.
- Featured Builds - To store the featured builds that are shown on the website.
- Components
 - CPU - Database where all information about available CPU's are stored.
 - RAM - Database where all information about available RAM is stored.
 - PSU - Database where all information about available PSU's are stored.
 - MB - Database where all information about available MB's are stored.
 - GPU - Database where all information about available GPU's are stored.
 - Case - Database where all information about available cases are stored.
 - Storage - Database where all information about available storage is stored.

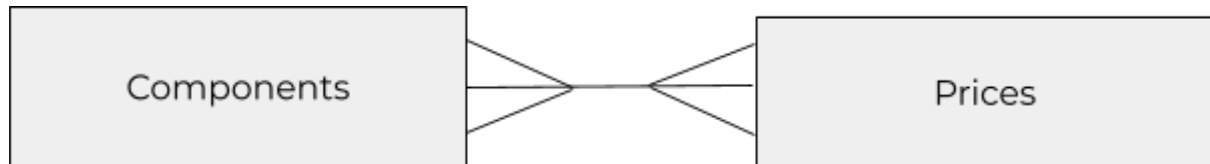
Many to Many Link Tables:



The components table will link to the CPU table which stores many different CPU's. They will be stored in the component table as just a single ID, which will then link back to the same ID in the CPU table and provide the information about the CPU required to create the table where the user can select the component. This is the same for all subsections in the components table.

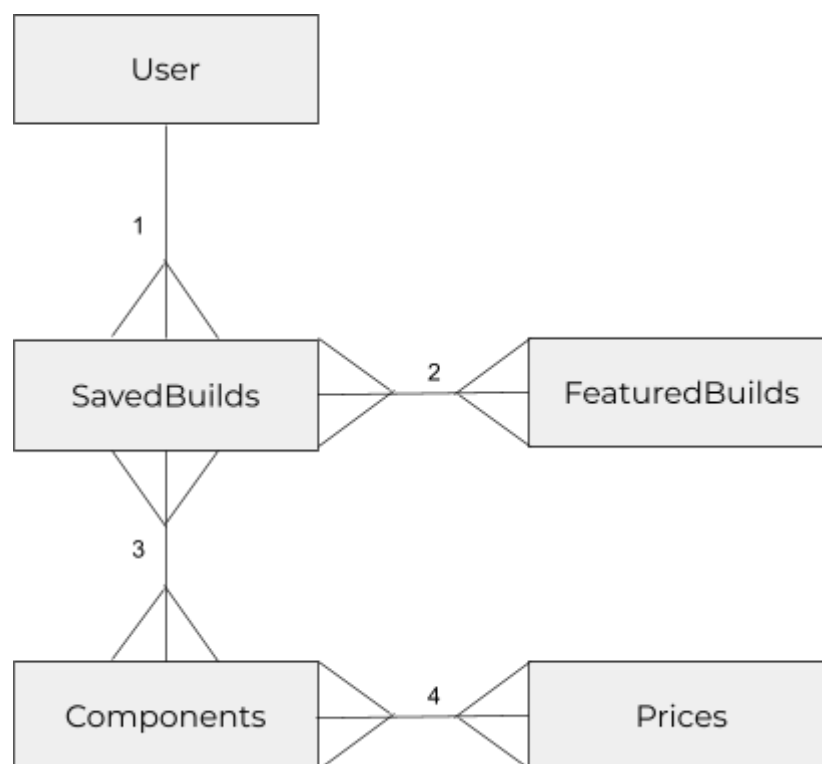


The saved builds database stores many saved builds from all the users, There are also multiple users on the system which makes the link - many to many. The userid, stored in the user table is used in the saved builds table to link the information back to the user without requiring all other data being stored in both databases.



The Components database has a many to many relationship with the price database as all of the components stored in the 'subdatabases' for components each have a corresponding price which is stored in the prices database and linked using the component ID so not all data needs to be stored in both databases.

Entity Relationship Diagram:



- 1) Link 1 is the link between the user database and the saved builds database, This is a one to many relationship as one user is able to have multiple saved builds. The databases are linked using the userid which is stored in the user table and referenced in the savedbuilds database.
- 2) Link 2 is the link between the savedbuilds database and the Featuredbuilds database which is a many to many relationship as the savedbuilds database can store multiple different builds, in addition the featured builds database will store around 5 builds to show. They are linked using the build ID so the featured builds database can retrieve the correct details from the savedbuilds database.

- 3) Link 3 is the link between the components and the savedbuilds database which is similar to link 1 as there is one saved build which has access to the multiple components as well as the subcomponent database. They are linked using the buildid as the selected components from the component database are then stored in the saved build database under that buildID.
- 4) Link 4 is the link between the components database and the price database. This link is a many to many link as multiple components stored have a corresponding price stored in the price database, these tables are linked using the componentID which allows the component id to retrieve either the live amazon price or the fallback price if the live price is unavailable.

Normalisation Checks:

Price:

{“PriceID”:, “amazonprice”:, “fallbackprice”: }

The price database is in first normal form as there is a primary key, PriceID, in addition, all columns are unique and atomic. The price database is also in second normal form as there is no composite key in the database. In addition the database is in third normal form as the information cannot be found from elsewhere, every column depends on the primary key.

Users:

{“UserID”:, “Username”:, “Email”:, “Password”:}

The user database is in first normal form as there is a primary key, which is the UserID, as well as all columns being unique and atomic as they cannot be broken down further, The database is also in second normal form as there is no composite key in the database. In addition, the database is in third normal form as the information cannot be found elsewhere or in another database, every column depends on the primary key.

SavedBuilds:

{“BuildID”:, “CPUID”:, “GPUID”:, “RAMID”: }

The savedbuilds database is in first normal form as there is a primary key, as well as all columns being unique and atomic as they cannot be further broken down. The database is in second normal form as there are no composite keys in the database, in addition the database is in third normal form as every column depends on the primary key which is “BuildID”.

Database Variables and Validation

Users:

Field	Data Type	Validation	Additional Notes
UserID - PK	Integer	-	AutoIncremented when new user is added
Username	String	Unique, not found elsewhere in the table	-
Email	String	Must have correct formatting - @ symbol as well as a .com/.co.uk	Also must not be present elsewhere in the table
Password	String	At least 5 characters with a numbers or symbol	May be stored using hashing further down development
Token	String	-	Will be randomly generated anytime the user logs into the system.

Price:

Field	Data Type	Validation	Additional Notes
PriceID - PK	Integer	-	AutoIncremented when new price is added
Amazon Price	String	-	Will be in £, may be able to change in future.
Fallback Price	String	-	Used if the amazon price is unavailable.

Components (Table repeated for all components of the build - CPU used as example):

Field	Data Type	Validation	Additional Notes
ComponentID - PK	Integer	-	AutoIncremented when new component is added
PriceID - FK (from price table)	String	-	-
CPUname	String	Unique	Name of the CPU / Component
Clock	String	Formatted in MHz	The Clock speed of the component to show in the table.
Codename	String	-	Codename of the component
Cores	String	-	Number of cores the CPU has
Process	String	-	The process used by the CPU
L3Cache	String	-	The amount of L3Cache the cpu has
ReleaseDate	String	Not in Future	Formatted as Date

FeaturedBuilds:

Field	Data Type	Validation	Additional Notes
FeaturedID - PK	Integer	-	AutoIncremented when new build is added
UserID - FK (from user table)	String	-	-
BuildID - FK (from savedbuilds table)	String	-	-

Savedbuilds:

Field	Data Type	Validation	Additional Notes
BuildID - PK	Integer	-	AutoIncremented when new build is added
PriceID - FK (from price table)	String	-	-
CPUID -FK (from CPU table)	String	-	-
GPUID - FK (from GPU table)	String	-	-
RAMID - FK (from RAM table)	String	-	-
PSUID - FK (from PSU table)	String	-	-
CaselD - FK (from case table)	String	-	-
StorageID - FK (from storage table)	String	-	-

Database Algorithms

[/login/attemptlogin:](#)

Check if the details entered by the user match the details stored in the database:

```
SELECT username, password, FROM users WHERE username = [?](retrieved from form)
```

If password = [F] generate a new token for the user:

```
UPDATE users SET token = ? - (randomly generated)  
WHERE username = ? - (username from first query)
```

[/featuredbuilds/get:](#)

Select the component information relating to the buildid decided by the javascript code.

```
SELECT components FROM featuredbuilds WHERE buildid = [?](from javascript)
```

[/login/save:](#)

Retrieve the new login details from the form and save them as a new user.

```
INSERT INTO user  
  (username, email address, password,)  
VALUES (?, ?, ?)
```

[/price/get:](#)

Retrieve the price for a component from the price database.

```
SELECT price FROM componentprice WHERE componentid = [?](from form)
```

[/savedbuilds/save:](#)

Save a new build to the savedbuilds database under a user's id.

```
INSERT INTO savedbuilds WHERE userid = [?] (from javascript)
    (buildid, cpuid, gpuid, ramid, psuid, caseid, storageid, mbid,)
VALUES (?, ?, ?, ?, ?, ?, ?, ?)
```

Used to check if the build id already exists.

```
SELECT buildid FROM savedbuilds WHERE buildid = [?] (from javascript)
```

[/savedbuilds/list:](#)

List the users saved builds so they can be edited and accessed.

```
SELECT cpuid, gpuid ,ramid, psuid, caseid, storageid, mbid, FROM
savedbuilds WHERE userid = [?](provided by javascript)
```

Database Test Plan

Prices:

PriceID[PK]	AmazonPrice	FallbackPrice
1	£57.98	£60
2	£123	£120
3	£32.76	£30
4	£45	£50
...

Components (CPU as example):

ComponentID[PK]	PriceID[FK]	CPUname	Clock	Cores	Codename	Processes
1	17	Ryzen 5	4GHz	4	Sandy bridge	7nm
2	54	Ryzen 7	5GHz	8	Comet lake	14nm
3	4	I7	2GHz	2	Picasso	8nm
4	22	I9	16GHz	1	Picasso	2nm
...

SavedBuilds:

BuildID[PK]	CPUID[FK]	GPUID[FK]	RAMID[FK]	PSUID[FK]	CaseID[FK]	StorageID[FK]
1	17	27	3	22	90	11
69	1	56	55	17	12	5
28	2	76	45	32	90	99
55	67	2	30	45	22	1
...

Users:

UserID[PK]	Username	Password	Email Address
1	Alfie	Test	alfie@email.com
2	Alfie2	Test2	alfie2@email.com
3	Alfie3	Test3	alfie3@email.com
4	Alfie4	Test4	alfie4@email.com
...

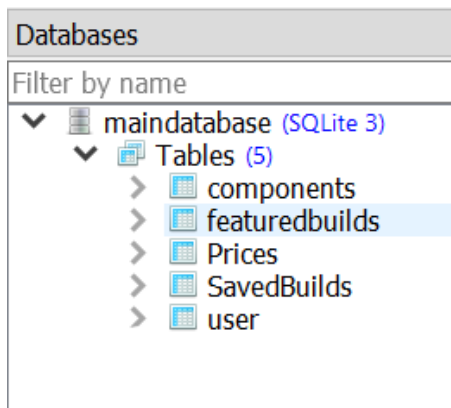
Featured Builds:

featuredID[PK]	CPUID[FK]	GPUID[FK]	RAMID[FK]	PSUID[FK]	CaseID[FK]	StorageID[FK]
1	17	27	3	22	90	11
2	1	56	55	17	12	5
3	2	44	67	88	9	21
4	661	2	91	66	20	22
...

Development

Database Setup

To set up the database, I used SQLite which allowed me to create, edit and manage multiple tables inside the database. I used this software as it was free and easy to use, it was also not too complicated and did everything that I required in the initial database setup.



To begin the database setup, I created a new database named 'maindatabase' which is the overall database and will hold all of the tables required for the project. By using a single main database, it meant that all of the tables were in one place and can communicate with each other easily, such as using primary and foreign keys. Once I created the main database I was able to add the tables I had designed previously and be able to access them from IntelliJ.

After creating the initial database and the tables, with filler columns to allow for the creation as SQLite would not allow a table to be created without a column, I then created columns in each table, I started with the user table as this would be the first table I would require as I was planning on developing the login system before anything else.

	Name	Data type	P	F	U	H	N	C	
1	userID	INTEGER	🔑		🧩		🚫		NULL
2	username	STRING							NULL
3	email	STRING							NULL
4	password	STRING							NULL
5	Token	STRING			🧩				NULL

The User table had 5 columns in total, which were userID, username, email, password, and Token. Each column had a default value of Null which meant that the api would return an error if the user

failed to fill an input so the user could be prompted to do so, adding a level of validation to the website.



The userID column was an integer as it would be a number which incremented by one as a new user was added, in addition the column was a primary key as it was used in other tables in the database. userID was also set as a unique character as each user needed to have a separate id in order to ensure the correct information was being shown to them and not the information stored under someone else who happened to have the same ID.

The username column was a string as it could include letters or numbers in any arrangement, since taking the screenshot, I have made this column 'unique' to insure the username is not the same as any others as this is how a user will login to the website therefore it needs to be different than any others on the database.

The email column was also a string as it would include numbers, symbols such as the '@' and the '.' as well as normal characters, in addition to the username, email should be 'unique' as if someone tries to sign up with an email already on the database it is likely they have already signed up and can therefore be prompted they have done so or can reset their password.

The password column was labeled a string as it can include any character, numbers and symbols, however, this should not be labeled as unique as it is possible for two users to have the same password, If the system notifies the user that there password is already stored on the database can be seen as a security flaw as it tells the user someones password.

The token was a string as it is made of a random assortment of numbers and letters, the token should also be 'unique' to ensure that it is not shared with others and it is ensured that the information being presented to the user is the correct information, as a validation, the username and the token can be used together to collect the stored information to add redundancy to any duplications of the token or the username.



	Name	Data type	P	F	U	H	N	C	
1	cpuID	INTEGER							NULL
2	CPUName	STRING							NULL
3	Codename	STRING							NULL
4	Cores	INTEGER							NULL
5	Clock	STRING							NULL
6	Socket	STRING							NULL
7	Process	STRING							NULL
8	L3Cache	STRING							NULL
9	TDP	STRING							NULL
10	ReleaseDate	STRING							NULL

The next table I made was the CPU's table. This table included the information to be shown to the user to determine which CPU to select. The first column, cpuID is an integer as it will increment by 1 when a new component is added, in addition it is a primary key as it is used in other tables elsewhere such as savedbuilds. The other columns are strings as they

include measurements, such as MHz which means they have both letters and numbers, however, the cores column is an integer as it can only have numbers as the number of cores the cpu has.

Each table for the other components follow a similar pattern, with a componentid, which is the primary key and unique to prevent other components having the same id and therefore a risk the incorrect information is saved. The tables will also have the other columns to store the specific information and determine which column it will be shown in on the website. The majority being strings rather than integers.





The next table was Prices which stores the amazon linked price and the fallback price for the components.

	Name	Data type	P	F	U	H	N	C	
1	priceID	INTEGER							NULL
2	amazonprice	STRING							NULL
3	fallbackprice	STRING							NULL

The first column, priceID is a primary key and unique as it is linked to other tables so each component has the correct price linked to it, it is unique to prevent other prices having the

same id and therefore a risk of mixing the prices between components. The second column is the amazonprice column which is a string as it will include the number price, symbols such as the '.' and the currency sign. The second column is the fallback price which will be used if the amazonprice is unavailable. In addition it is a string as a price will include both a symbol as well as numbers.










The featurebuilds table is used to store the information to be shown on the featured build section of the homepage.

	Name	Data type	P	F	U	H	N	C	
1	featuredID	INTEGER							NULL
2	UserID	STRING							NULL
3	BuildID	STRING							NULL

The featuredID column is a primary and unique column as it identifies the build to be used on the website. In addition, it is an integer. The second column is the userID, used

to find the correct user information to display on the website alongside the build information, UserID is a string as it links to another table where the information is a number. Both the UserID and BuildID is a foreign key as it links to other tables to collect the correct information without having to store it again in a different table. BuildID is also an integer as it is linked to a column in the savedbuilds table where the BuildID is an integer.

SavedBuilds is used to store the build information about builds that have been decided by the users and they have requested to save.

	Name	Data type	P	F	U	H	N	C	
1	UserID	INTEGER							NULL
2	BuildID	INTEGER							NULL
3	CPUID	INTEGER							NULL
4	GPUID	INTEGER							NULL
5	RAMID	INTEGER							NULL
6	PSUID	INTEGER							NULL
7	CaseID	INTEGER							NULL
8	StorageID	INTEGER							NULL

The UserID column is a foreign id which links to the userid in the users table. The BuildID Column is a primary and unique key as it is the unique identifier for the build made by the user and used to select which one the user wants to display. The next columns are all foreign keys as they link to the respective component table and provide

the information for the build components without having to restore all the details of the component the user selects.

Web Server Setup

The web server is used to allow access to the website using port 8080, this means the website can connect to the database as well as javascript which would not be possible if the file was opened as a plain html file.

```
public static void main(String[] args) {

    openDatabase("maindatabase.db"); - This line connects to the database and allows for
    communication between it and the code.

    ResourceConfig config = new ResourceConfig(); - This line prepares jersey which is used
    as the servlet library for this website.

    config.packages("controllers"); - configs and sets up the api for use.

    config.register(MultiPartFeature.class); - to allow support for multipart forms on the
    website.
    ServletHolder servlet = new ServletHolder(new ServletContainer(config)); - Instantiate
    the Servlet
    Server server = new Server(7268); - prepare our Jetty Server to listen on port 7268
    ServletContextHandler context = new ServletContextHandler(server, "/"); - instantiate
    the Server
    context.addServlet(servlet, "/*"); - connect the Servlet to the Server

    try {
        server.start(); - starts the server
        System.out.println("Server successfully started.");
        server.join(); - Wait for the server to rejoin the main thread, program waits here
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

The code above is responsible for starting, and setting up the necessary connections with api's and databases to allow the code to retrieve and send information, therefore being the most important aspect of the code which, without, would prevent the entire code from working. During development of this, I had issues with the code which would result in an error being shown, and therefore the server not starting, therefore fixing this was a priority. I worked out that the error was being caused by the original port, 8080, already being occupied by another program, therefore I changed the port to 7268 which was free and therefore the code began to run as expected. In addition, on my laptop, if running the program, and then exiting intellij without ending the main.java process first, the port remained occupied therefore, when re opening, the code would not run, this could be fixed by restarting the computer which freed up the port. Other times the code would not present an error message, rather would just stop which was difficult to diagnose however was fixed by copying the code into a new project environment.

The Second code that was required for the server to work properly was client.java. This code dealt with the requests for file and mime types such as providing images, css and javascript files when requested by the users browser. This ensured the user got the most up to date and working code that was stored on the server.

```
@GET
@Path("/Images/{path}")
@Produces({"image/jpeg,image/png"})
public byte[] getImageFile(@PathParam("path") String path) {
    return getFile("client/Images/" + path);
}

@GET
@Path("/Javascript/{path}")
@Produces({"text/javascript"})
public byte[] getJavaScriptFile(@PathParam("path") String path) {
    return getFile("client/Javascript/" + path);
}

@GET
@Path("/lib/{path}")
@Produces({"text/javascript"})
public byte[] getJavaScriptLibraryFile(@PathParam("path") String path) {
    return getFile("client/lib/" + path);
}

@GET
@Path("/css/{path}")
@Produces({"text/css"})
public byte[] getCSSFile(@PathParam("path") String path) {
    return getFile("client/css/" + path);
}

@GET
@Path("/{path}")
@Produces({"text/html"})
public byte[] getIHTMLFile(@PathParam("path") String path) {
    return getFile("client/" + path);
}
```

The changes required to the code above was to change the paths of the files that were being provided, for example, the images folder was changed to match the folder location in my project as this was different from the original code. This was the same for the css path and the javascript path. I put these in a folder rather than keeping them in the client folder where the HTML was located. This meant my code was easier to understand and find the correct files. Although this was not vital to the code and would have worked if the files remained in the same folder, it meant the project was easier to code and was quick and easy to fix in the client.java file.

API Development

Multiple API's are required for the website to work correctly and communicate with the databases stored.

Users:

The first API's created were for the login and signup system as this was the first system I worked on because it was one of the first things a user will use on the website.

```
@POST
@Path("login")
public String loginUser(@FormParam("username") String username,
    @FormParam("password") String password) {
    System.out.println("Invoked loginUser() on path user/login");
    try {
        PreparedStatement ps1 = Main.db.prepareStatement("SELECT password FROM
user WHERE username = ?");
        ps1.setString(1, username);
        ResultSet loginResults = ps1.executeQuery();
        if (loginResults.next() == true) {
            String correctPassword = loginResults.getString(1);
            if (password.equals(correctPassword)) {
                String token = UUID.randomUUID().toString();
                System.out.println(token);
                PreparedStatement ps2 = Main.db.prepareStatement("UPDATE user SET
token = ? WHERE username = ?");
                ps2.setString(1, token);
                ps2.setString(2, username);
                ps2.executeUpdate();
                JSONObject userDetails = new JSONObject();
                userDetails.put("username", username);
                userDetails.put("token", token);
                String temp = userDetails.toString();
                System.out.println(temp);
                return temp;
            } else {
                return "{\"Error\": \"Incorrect password!\"}";
            }
        } else {
            return "{\"Error\": \"Username and password are incorrect.\"}";
        }
    } catch (Exception exception) {
        System.out.println("Database error during /user/login: " +
exception.getMessage());
        return "{\"Error\": \"Server side error!\"}";
    }
}
```

The API above is used to validate a user's login attempt and check the data the user inputted is the correct data and matches what is stored in the database. The first thing the API does is find the password that is linked to the username inputted. This is in a try statement as if the password or username cannot be found in the database, the code will run the else statement which will return an error that the username and password is incorrect, which is then read by the javascript and will display the incorrect information message to the user. If the details are found, the code will create a random token and store it in the database as well as return the token and username to the javascript.

Curl Test:

This was tested using the curl command -

Curl -s localhost:7268/users/login -F username=Alfie -F password=test

```
alfie@DESKTOP-FLCBNCD MINGW64 ~  
$ curl -s localhost:7269/users/login -F username=Alfie -F password=test
```

This did nothing in the curl prompt itself however, did create a new random token in the database and therefore could be checked by looking in the database physically, in addition, no error was returned.

```
@POST  
@Path("logout")  
public static String logout(@CookieParam("token") String token){  
    try{  
        System.out.println("users/logout " + token);  
        PreparedStatement ps = Main.db.prepareStatement("SELECT userID FROM user  
WHERE token=?");  
        ps.setString(1, token);  
        ResultSet logoutResults = ps.executeQuery();  
        if (logoutResults.next()){  
            int userID = logoutResults.getInt(1);  
            //Set the token to null to indicate that the user is not logged in  
            PreparedStatement ps1 = Main.db.prepareStatement("UPDATE user SET  
token = NULL WHERE userID = ?");  
            ps1.setInt(1, userID);  
            ps1.executeUpdate();  
            return "{\"status\": \"OK\"}";  
        } else {  
            return "{\"error\": \"Invalid token!\"}";  
        }  
    }  
    catch (Exception ex) {  
        System.out.println("Database error during /users/logout: " +  
ex.getMessage());  
        return "{\"error\": \"Server side error!\"}";  
    }  
}
```


The code above is used to logout a user when they click the logout button. The token is sent to the api by the javascript and then used to select the userID from the users table where the token is the token sent. This insures the correct user is being logged out, and that if the user is logged in elsewhere, they are also logged out. The code will then set the token to null when they are logged out so it can be set to a new random string when the user logs back in. The api returns the status ok when the log out has occurred so the webpage can be reloaded and the users cookies and information is removed. The code may also return an error if the token is not found or is invalid, however the user will still be logged out if the token is invalid through other API's which ensure the user only logged in if the token in their browser matches the token in the database.

Curl Test:

This was tested using the following curl command -

Curl -s localhost:7268/users/logout -F token=test

```
alfie@DESKTOP-FLCBNCD MINGW64 ~  
$ curl -s localhost:7268/users/logout -F token=test
```

This logged out the user which could be checked in the database as the token was changed and became null, as well as the curl prompt not responding with an error which showed that the API works as it is supposed to.

```
@GET  
@Path("getbyusern/{username}")  
public String GetUser(@PathParam("username") String username) {  
    System.out.println("Invoked Users.GetUser() with UserID " + username);  
    try {  
        PreparedStatement ps = Main.db.prepareStatement("SELECT  
userID,username,email,password,token FROM user WHERE username = ?");  
        ps.setString(1, username);  
        ResultSet results = ps.executeQuery();  
        JSONObject response = new JSONObject();  
        if (results.next() == true) {  
            response.put("userID", results.getInt(1));  
            response.put("username", results.getString(2));  
            response.put("email", results.getString(3));  
            response.put("password", results.getString(4));  
            response.put("token", results.getString(5));  
        }  
        System.out.println(response.toString());  
        return response.toString();  
    } catch (Exception exception) {  
        System.out.println("Database error: " + exception.getMessage());  
        return "{\"Error\": \"Unable to get item, please see server console  
for more info.\"}";  
    }  
}
```

The API below was used to ensure the user that is logged in is the correct user and that the correct information is shown, this api prevents people from being able to change the username cookie in the browser and therefore access other people's information. The code is called by javascript which sends the username of the user stored in the cookies, this is then checked with the database, if it is not found then the code returns an error which causes the javascript to call the logout api and therefore log out the user, if the username is found, the code will compare the token stored in the cookies with the token stored in the database, if they match then nothing will be done and the user can continue, however, if they differ then it suggests the user has logged in elsewhere or changed the username in the cookies and they are therefore logged out and cannot access the features anymore without logging back in.

Curl Test:

The code above was tested in curl by using the following command:

Curl -s localhost:7268/users/getbyusern/Alfie

```
alfie@DESKTOP-FLCBNCD MINGW64 ~
$ curl -s localhost:7268/users/getbyusern/Alfie
{"password":"test","userID":98,"email":"test","username":"Alfie","token":"test"}
```

The test responded with the stored information for the username that was sent to the API, therefore showing it worked well and that no errors were found or produced.

```
@POST
@Path("add")
public String UsersAdd(@FormDataParam("username") String username,
@FormDataParam("email") String email, @FormDataParam("password") String
password) {
    System.out.println("Invoked Users.UsersAdd()");
    try {
        PreparedStatement ps = Main.db.prepareStatement("INSERT INTO user
(username, email,password) VALUES (?, ?,?)");
        ps.setString(1, username);
        ps.setString(2, email);
        ps.setString(3, password);

        ps.execute();
        return "{\"OK\": \"Added user.\"}";
    } catch (Exception exception) {
        System.out.println("Database error: " + exception.getMessage());
        return "{\"Error\": \"Unable to create new item, please see server
console for more info.\"}";
    }
}
```

The API above is used to add new users to the database, The code receives the username, email and password from the sign up form, The information is checked against the database in the javascript beforehand to ensure the user does not already have an account and that the information entered is in the correct format, once the validation checks have been completed, the API adds the information to the users table in the database, returns an ok message which prompts the javascript to show the successfully signed up message and then the user is able to log into their account. The API also returns an error which prompts the javascript to show a generic error with details message.

Curl Test:

The curl command - `Curl -s localhost:7268/users/add -F (details)` was used to test the api works as expected.

```
alfie@DESKTOP-FLCBNCD MINGW64 ~  
$ curl -s localhost:7268/users/add -F username=Alfie2 -F email=alfietest -F password=test1  
{"OK": "Added user."}
```

By running the command, the API responded with the expected OK message and there was no error message provided, In addition, by manually checking the database, the new user was added correctly with the right details therefore the API code can be shown as working as intended.

Components:

```
@GET  
@Path("list")  
public String CPUList() {  
    JSONArray response = new JSONArray();  
    try {  
        PreparedStatement ps = Main.db.prepareStatement("SELECT  
CPUname,Codename,Cores,Clock,Socket,Process,L3Cache,TDP,ReleaseDate,GPUname,Chip,GPUrelease,GPUbus,GPUmemory,GPUclock,GPUmemclock,shaders FROM components ");  
        ResultSet results = ps.executeQuery();  
        while (results.next()==true) {  
            JSONObject row = new JSONObject();  
            row.put("CPUname", results.getString(1));  
            row.put("Codename", results.getString(2));  
            row.put("Cores", results.getString(3));  
            row.put("Clock", results.getString(4));  
            row.put("Socket", results.getString(5));  
            row.put("Process", results.getString(6));  
            row.put("L3Cache", results.getString(7));  
            row.put("TDP", results.getString(8));  
            row.put("ReleaseDate", results.getString(9));  
            row.put("GPUname", results.getString(10));  
            row.put("Chip", results.getString(11));  
            row.put("GPUrelease", results.getString(12));  
            row.put("GPUbus", results.getString(13));  
            row.put("GPUmemory", results.getString(14));  
            row.put("GPUclock", results.getString(15));  
            row.put("GPUmemclock", results.getString(16));  
            row.put("shaders", results.getString(17));  
  
            response.add(row);  
            System.out.println(response);  
        }  
        return response.toString();  
    } catch (Exception exception) {  
        System.out.println("Database error: " + exception.getMessage());  
        return "{\"Error\": \"Unable to list items. Error code xx.\"}";  
    }  
}
```

The API above is used to collect the cpu data from the database to display in the table on the website and allow the user to select the component. The API is repeated whilst the next row in the table is populated to ensure that all the components are shown. The code collects the information from the columns of each row and places that in the row to be shown on the website which ensures that the information is shown in the correct place. The API above is used only for the cpu table, another similar one is required for each other component as column names and number of columns differ between component types.

Curl Test:

To test the api, the curl test - `Curl -s localhost:7268/components/list` can be used.

```

a1f1e@DESKTOP-PLCBNCD MINGW64 ~
$ curl -s localhost:7268/components/list
[{"TDP": "65 W", "GPUbus": "4", "shaders": "4864 \u221a 152 \u221a 80", "L3Cache": "32MB", "GPUname": "RTX 3060 Ti", "Process": "7 nm", "Clock": "4.6 GHz", "ReleaseDate": "Nov 5th, 2020", "Chip": "GA104", "Cores": "6", "GPUmemclock": "1750", "Codename": "Vermeer", "CPUname": "Ryzen 5 5600X", "GPUclock": "1410", "GPUrelease": "Dec 1st, 2020", "Socket": "Socket AM4", "GPUmemory": "8 GB"}, {"TDP": "65 W", "GPUbus": "4", "shaders": "8704 \u221a 272 \u221a 96", "L3Cache": "32MB", "GPUname": "RTX 3080", "Process": "7 nm", "Clock": "4.2 GHz", "ReleaseDate": "Jul 7th, 2019", "Chip": "GA102", "Cores": "6", "GPUmemclock": "1188", "Codename": "Matisse", "CPUname": "Ryzen 5 3600", "GPUclock": "1440", "GPUrelease": "Sep 1st, 2020", "Socket": "Socket AM4", "GPUmemory": "10 GB"}, {"TDP": "65 W", "GPUbus": "4", "shaders": "10496 \u221a 328 \u221a 112", "L3Cache": "16MB", "GPUname": "RTX 3090", "Process": "7 nm", "Clock": "3.9 GHz", "ReleaseDate": "Apr 24th, 2020", "Chip": "GA102", "Cores": "10496", "GPUmemclock": "1219", "Codename": "Matisse", "CPUname": "Ryzen 3 3100", "GPUclock": "1395", "GPUrelease": "Sep 1st, 2020", "Socket": "Socket AM4", "GPUmemory": "24 GB"}, {"TDP": "65 W", "GPUbus": "4", "shaders": "5888 \u221a 184 \u221a 96", "L3Cache": "32MB", "GPUname": "RTX 3070", "Process": "7 nm", "Clock": "4.1 GHz", "ReleaseDate": "Sep 24th, 2019", "Chip": "GA104", "Cores": "6", "GPUmemclock": "1750", "Codename": "Matisse", "CPUname": "Ryzen 5 3500X", "GPUclock": "1500", "GPUrelease": "Sep 1st, 2020", "Socket": "Socket AM4", "GPUmemory": "8 GB"}, {"TDP": "65 W", "GPUbus": "4", "shaders": "5120 \u221a 320 \u221a 128", "L3Cache": "32MB", "GPUname": "RTX 6800 XT", "Process": "7 nm", "Clock": "4.1 GHz", "ReleaseDate": "Jul 7th, 2019", "Chip": "GA102", "Cores": "8", "GPUmemclock": "1900", "Codename": "Vermeer"}]

```

This responded with a list of all components and information that is stored in the components database, this shows that the API is working as expected, no errors were produced as well as no changes to the database occurring, therefore the code can be shown to be working well.

```

@GET
@Path("/getcpu/{id}")
public String Getname(@PathParam("id") String id) {
    System.out.println("Invoked Cpu.getid() with id = " + id);
    try {
        PreparedStatement ps = Main.db.prepareStatement("SELECT CPUname,Codename,Cores,Clock,Socket,Process,L3Cache,TDP,ReleaseDate FROM components WHERE cpuID = ?");
        ps.setString(1, id);
        ResultSet results = ps.executeQuery();
        JSONObject response = new JSONObject();
        if (results.next() == true) {
            response.put("CPUname", results.getString(1));
        }
        System.out.println(response.toString());
        return response.toString();
    } catch (Exception exception) {
        System.out.println("Database error: " + exception.getMessage());
        return "{\"Error\": \"Unable to get item, please see server console for more info.\"}";
    }
}

```

The API above is short and used only once the user selects the component from the table. The user selects the component from the table which stores the id of the selected component, in order to collect the information again in the savedbuilds table, the API above and the Id stored are used to collect the correct data rather than collect all the data that is stored.

Curl Test:

The API can be tested using the curl command - Curl -s localhost:7268/components/getcpu/2

```
alfie@DESKTOP-FLCBNCD MINGW64 ~  
$ curl -s localhost:7268/components/getcpu/2  
{"CPUname":"Ryzen 5 3600"}
```

The code above responded with the correct cpu name that was requested which can be checked by seeing which component has the id 2 in the database. No errors were produced and the expected output was seen therefore the API works as required.

Most of the API's are repeated for each component with only names changed.

```
@GET  
@Path("list/{id}")  
public String BuildList() {  
    JSONArray response = new JSONArray();  
    try {  
        PreparedStatement ps = Main.db.prepareStatement("SELECT Name FROM  
savedbuilds WHERE UserID = id");  
        ResultSet results = ps.executeQuery();  
        while (results.next() == true) {  
            JSONObject row = new JSONObject();  
            row.put("Name", results.getString(1));  
            response.add(row);  
            System.out.println(response);  
        }  
        return response.toString();  
    } catch (Exception exception) {  
        System.out.println("Database error: " + exception.getMessage());  
        return "{\"Error\": \"Unable to list items. Error code xx.\"}";  
    }  
}
```

The API above is used to select the saved build of the user when they are requested. The API receives the user id and selects the list of saved builds of the user whilst there is a next row. This ensures that all of the saved builds are shown to the user on the webpage. The API is similar as the one used to select and show the components once the user has selected which build they would like to edit. The API also only selects the saved builds with the same userID so that only the saved builds of that user are shown and not everyone's to select from and edit.

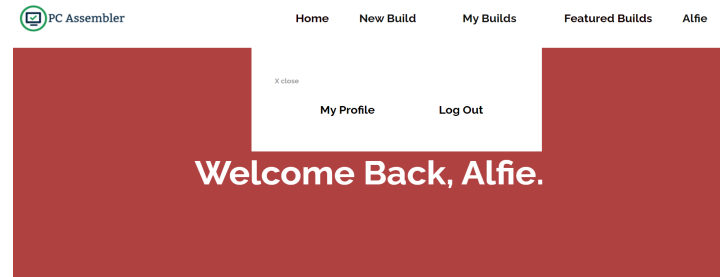
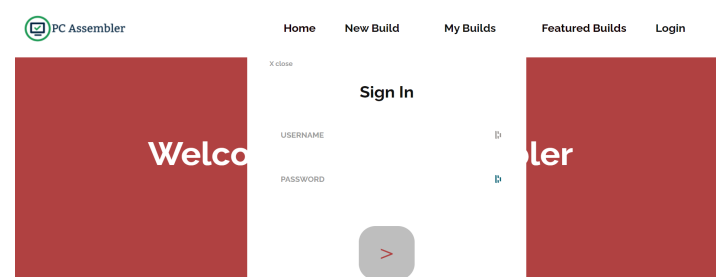
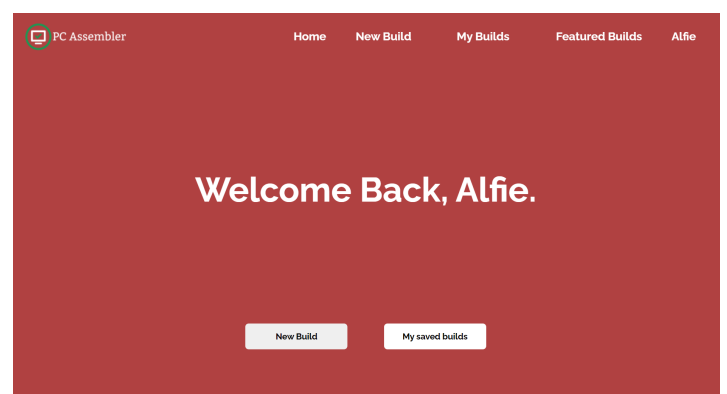
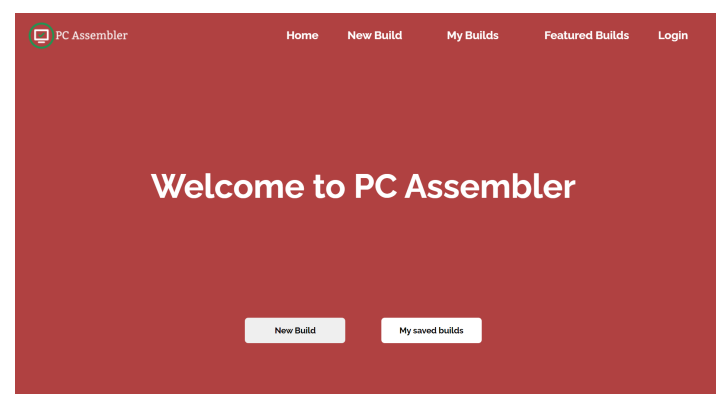
Website Development

Login and Account Management system:

I created Index.html which is the homepage and first page a user will visit on the website. The homepage contains links to the login, signup and other pages of the website. The title of the homepage contains a dynamic title which changes depending on if the user is signed into the website or not. This was done through the use of the javascript below:

```
function pageload(){
  loggedout = readCookie('notific');
  if(loggedout == 'true'){
    document.getElementById("noti").style.display = "block";
    document.cookie="notific=" + "false";
  }else{
    document.getElementById("noti").style.display = "none";
  }
  personname = readCookie('username');
  if(personname != "Login"){
    document.getElementById("loggedin").style.display = "block";
    document.getElementById("signinstuff").style.display="none"
    document.getElementById('maintitle').innerHTML = "Welcome Back, " + personname + ".";
  }else{
    document.getElementById("loggedin").style.display = "none";
    document.getElementById("signinstuff").style.display = "block";
    document.getElementById('maintitle').innerHTML = "Welcome to PC Assembler";
  }
}
```

The code above runs when the page loads and changes certain aspects of the website depending on the login status. A global variable of personname is originally set as Login, this then changes when the user logs into the website to the username, therefore the code will run as it no longer = Login. This changes the login button to the username of the user and hides the login and signup forms, showing the logout and profile buttons instead, in addition it changes the title from 'Welcome to PC Assembler' to 'Welcome Back, username'. The changes to the website when logged in are shown below:



The Login System uses the form shown in the image above and the javascript below.

```
function postUserLogin() {  
  
    console.log("Invoked postUserLogin() ");  
    var formData = new FormData(document.getElementById('loginpopupform'));  
    var url = "/users/login";  
  
    fetch(url, {  
        method: "POST",  
        body: formData,  
    }).then(response => {  
        return response.json(); //now return that promise to JSON  
    }).then(response => {  
        if (response.hasOwnProperty("Error")) {  
            document.getElementById("wrongdeets").style.display="flex";  
        } else {  
            document.cookie = "token=" + response.token;  
            document.cookie = "username=" + response.username;  
            openafterlogin();  
        }  
    });  
}
```


X close

Sign In

Incorrect username or password.Try Again.

USERNAME
Alfies

PASSWORD
.....



CREATE ACCOUNT

The JavaScript is stored in the resources/js folder along with all other javascript for this project. Once Invoked the javascript collects the data inputted in the form 'loginpopupform'. This is the username and password entered by the user on the homepage. This sends the form data to the login api which checks it with the values stored on the database. The Javascript will then receive a response from the API, which if it includes the word 'Error' the javascript will show the incorrect details message so the user can try again, if there is no error message, the javascript will set a cookie for the returned token and username value so the user can be identified. Finally the javascript will run the openafterlogin function which redirects the user to the correct page.

The Login feature was one of the main requirements for the project and required to allow each user to have their own savedbuilds list and be able to select and create personal builds. This would not be possible without a login system as the builds would have to be public and therefore can be changed by anybody who is on the website. The log in system went through multiple interactions in order to be fast and reliable as it is now. The javascript includes validating the users details are correct before allowing them access, this ensures that only people who know the details can access the account and that inputting any random value will result in a error and the user not being able to enter the portion of the website which requires users to be logged in.

The login and validation system also includes code which will periodically check that the cookie details stored on the users browser matches the details stored on the database. This prevents a user from being able to change their username cookie in order to access other users details.

```
function statuscheck(){
  console.log("invoked status check");
  user = readCookie('username');
  var url = "/users/getbyusern/" + user;
  fetch(url,{
    method: "GET",
  }).then (response => {
    return response.json();
  }).then(response => {
    if(response.hasOwnProperty("Error")){
      alert(JSON.stringify(response));
    }else{
      idcompare(response.token);
    }
  })
}
```

The code retrieves the username stored in the cookie by using the readcookie function

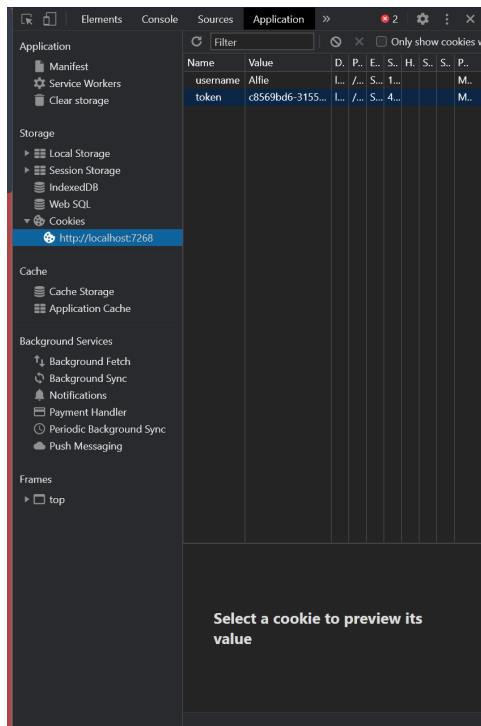
```
function readCookie(name) {
  var nameEQ = name + "=";
  var ca = document.cookie.split(';');
  for(var i=0;i < ca.length;i++) {
    var c = ca[i];
    while (c.charAt(0)==' ') c = c.substring(1,c.length);
    if (c.indexOf(nameEQ) == 0) return c.substring(nameEQ.length,c.length);
  }
  return filename = "Login";
}
```

The javascript then calls the getbyusern api which retrieves the data stored in the database under that username, If an error response is provided then the user is alerted and logged out, if a user is found then the token is forwarded to the id compare function.

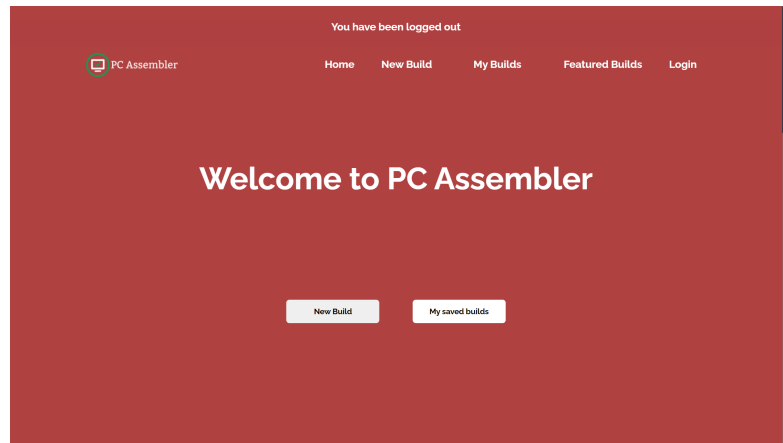
```
function idcompare(loggedtoken){
  var actualtoken = readCookie('token');
  if(actualtoken != 'loggedout') {
    if (loggedtoken == actualtoken) {
      alert(toopen);
      window.open(toopen + ".html", "_self");
    } else {
      console.log("Token incorrect - logging out");
      logout();
    }
  }
}
```

The id compare function will then retrieve the token that is stored in the cookie and compare it with the token sent from the api and that stored in the database. If they

match then the user is redirected to the website they want to go to, however if they are different then the user is notified and logged out of their account. This ensures that the correct user is able to access the correct information and further secures their account and details stored on the system.



This could be tested by manually changing the token in chrome using the developer menu. By changing the saved cookie value, the code would read a difference with the one stored on the database and therefore log out the user shown in the screenshot below by the notification at the top.



```
function addUser(forminput) {
  console.log("Invoked AddUser()");
  const formData = forminput;
  let url = "/users/add";
  fetch(url, {
    method: "POST",
    body: formData,
  }).then(response => {
    return response.json()
  }).then(response => {
    if (response.hasOwnProperty("Error")) {
      alert(JSON.stringify(response));
    } else {
      document.getElementById("signuptrue").style.display = "block";
    }
  });
}
```

The Javascript above is used to add a user to the database and allow them to log in. The code retrieves the details imputed into the signupform and then pushes it through to the api which will check the database for any other users with the same username or email, if this is found then the user is notified that they should use sign in instead of sign up. If no user is found then the information is saved, a new userID is created and the user is added. The javascript reads the response and displays a successful sign up message so the user is aware and can now sign in.

This is an important feature to ensure users can add themselves to the system and be able to create builds and use other features of the website which require a login. On occasion, I would have issues with the code where it would input the data in incorrect locations in the database, such as putting the email in the password column. This was fixed by ensuring that the data was inputted in the order that it is to be added to the database.

The Login form, added using HTML, uses javascript as well in order to show and hide depending on what the user clicks. By default the HTML below is hidden until the user selects the login button which in turn triggers javascript code which changes the section 'loginformoverall' to be shown and therefore allowing the user to login or sign up to the website.

The Login system works as intended and all features I would have liked work well, This therefore completes Success Criteria 2 as my website now has a working login and sign up system.

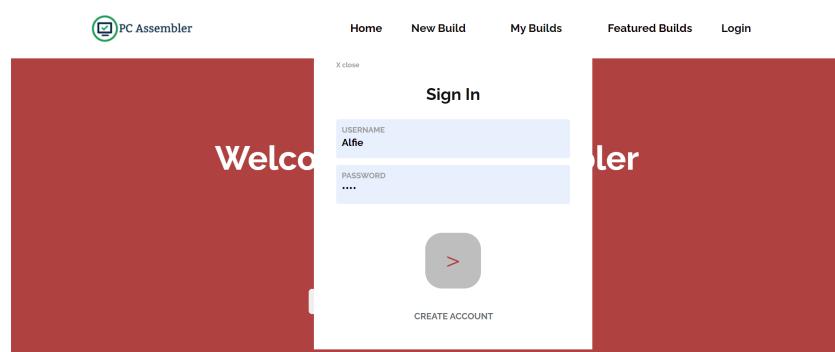
```
function loginmodalshow(){
    window.open("login.html", "_self");
}

function closesignin(){
    document.getElementById("signupformoverall").style.display = "none";
}
```

```
<section class="login-form" id="loginformoverall" style="display: none">
  <form method="post" id="loginpopupform">
    <div class = "signedin" id = "loggedinnew" style="display: none">
      <input type='button' class="closebutton" onclick="closeloginpage()" value='X close'
style="margin-bottom: 20px" />
      <br>
      <button type="button" class="profile">My Profile</button>
      <button type="button" class="profile" onclick="logout()">Log Out</button>
    </div>
    <div id="signinstuff">
      <input type='button' class="closebutton" onclick="closeloginpage()" value='X close' />
      <h1> Sign In </h1>
      <div class = "incorrect" id = "wrongdeets" style="display: none;">
        <h7>Incorrect username or password.</h7><br>
        <h7>Try Again.</h7>
      </div>
      <div class="textb">
        <input id="username" name ="username" type="text" required>
        <div class="placeholder">Username</div>
      </div>

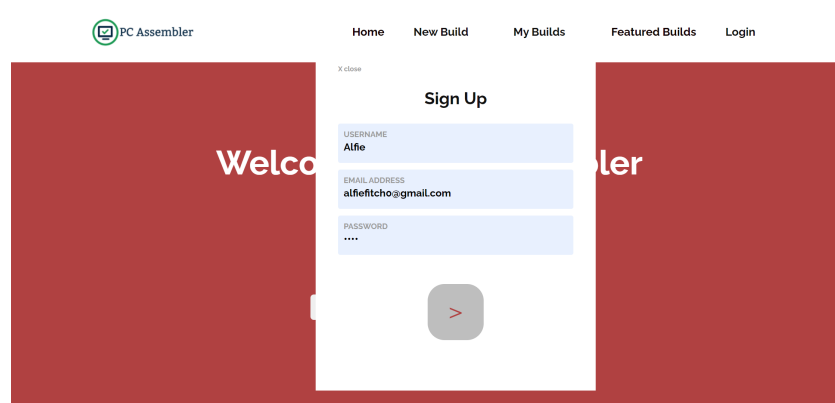
      <div class="textb">
        <input id = "password" name = "password" type="password" required>
        <div class="placeholder">Password</div>
      </div>
    </form>
    <input type='button' class="btn fas fa-arrow-right" onclick="postUserLogin()" value=""/>
    <br>
    <input type='button' class="signuptext" onclick="createaccountshow()" value='CREATE ACCOUNT' />
  </section>
```

The login system uses a form to collect the data from the user and an input which is a button type as I found when using a standalone button it would cause the page to refresh and therefore cause the login modal to disappear and therefore not be able to notify the user of an incorrect password as when the page refreshes the notification defaults back to not being shown. The fix makes no difference to the end user as they both look identical and act no differently other than no refreshing.



The login form uses css to show over the top of the other elements on the website, this allows the login form to be opened up anywhere on the web page as the menu bar is also sticky and always remains on top of the page no matter how low the user

has scrolled. The login modal also contains the button to open the sign up form which triggers javascript to hide the login form and show the sign up form which is identical, but with added details, such as email.



I decided that creating the login and sign up forms like this was the most user friendly as it is easy to understand and can be accessible from anywhere so users do not have to hunt from it, although harder to code, it is easier to keep track of as it is in one document for the whole page and its

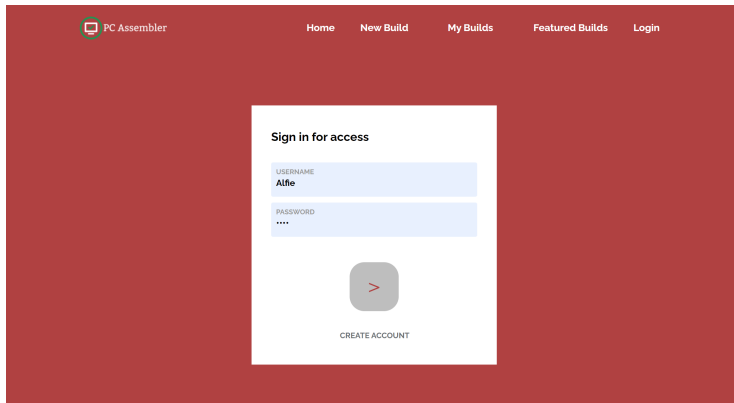
features rather than another html file for just a login page.

```
function createaccountshow(){
    document.getElementById("signupformoverall").style.display="block";
    document.getElementById("loginformoverall").style.display="none";
    $(".nav").addClass("sticky");
}
```

This code is reliable for making the login form hide when the user selects the create account button, ensuring the correct form is being shown to the user. Adding the line to make the navbar sticky is cosmetic as it makes the menu bar white when the form is opened up by the user.

By separating the forms so only one can be shown at a time ensures the user is using the correct form for what they want to do and not trying to log in using the sign up form, this makes the website for user friendly and easier to understand, it also prevents any possible errors with the code by inputting incorrect information.

```
function logincheck(){
    console.log("running login check");
    var personname = readCookie('username');
    if(personname == "Login"){
        loginmodalshow();
    }else{
        statuscheck();
    }
}
```



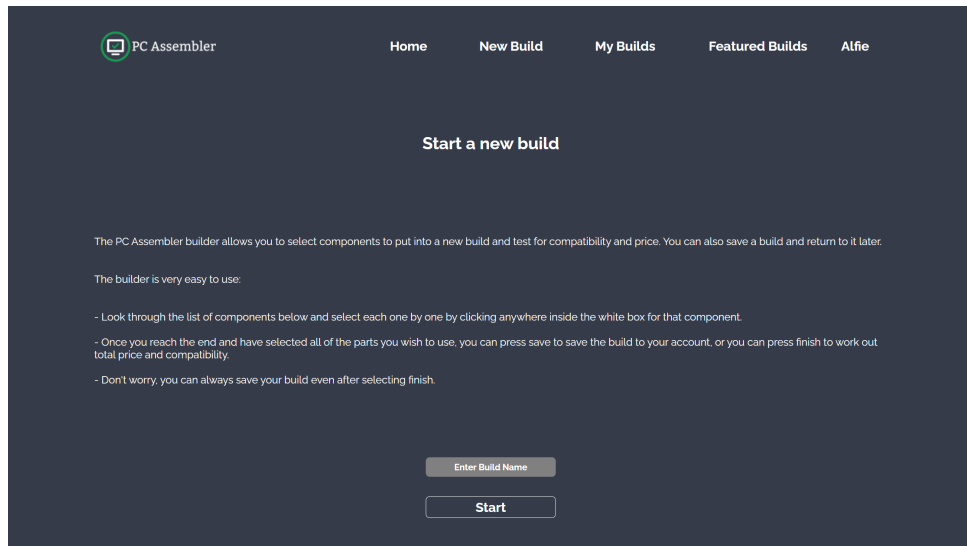
The code above is triggered when a user changes a page, such as clicking the new build button. It's purpose is to read the username cookie stored in the users browser, If it is login, this shows the user is yet to login and therefore redirects the user to a login page, if the cookie is not login then the statuscheck javascript is ran to check the users token is correct

and then they are redirected to the page they originally wanted to visit. This ensures that there is always a user logged in when visiting web pages such as the new build page which use information personalised and retrieved from specific users database. This also prevents the user from accessing a page whilst being logged out and it shows a blank page or incorrect information which the user will not understand and therefore not continue using the website. This also prevents other users' information being shown to someone who is logged out adding security to the website. The code can be changed to remove the requirement for the loginmodalshow() function as it is only a single line:

```
function loginmodalshow(){
    window.open("login.html", "_self");
}
```

By removing this, the code is easier to read and understand as it will remove unnecessary code. However, the code will have to be repeated if it is used elsewhere which is not a significant problem as it is one line:

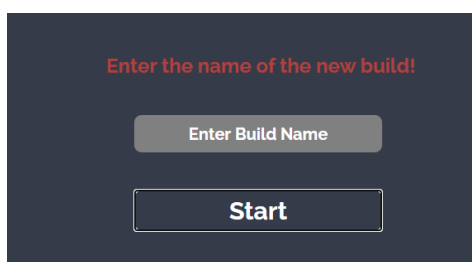
```
function logincheck(){
    console.log("running login check");
    var personname = readCookie('username');
    if(personname == "Login"){
        window.open("login.html", "_self");
    }else{
        statuscheck();
    }
}
```



When a user passes the login and user validation checks, they are able to access the new build page which allows them to start a new build and select components. They are first shown a page with the instructions. On this page are also the tables for each component however, they are by default hidden so they cannot be seen until they are called.

The user must input a build name before they are able to start, this uses validation to ensure that the box is not empty and has some form of identification. The start button triggers the function to start listing:

```
function startlisting(){
  let tempname = document.getElementById("buildname").value;
  name = String(tempname);
  if(name == ""){
    document.getElementById("noname").style.display="block";
  }else{
    document.getElementById("noname").style.display="none";
    showall(name);
    ListComponents("CPU");
    ListComponents("GPU");
  }
}
```



The startlisting function firstly retrieves the data inputted by the user in the buildname text box, this is then turned into a string so it can be compared in the if statement to ensure that it is not empty. If the box is empty then the code triggers the no name notification to be shown, which is by default hidden. This validation ensures that the build the user is

creating can be saved under a name that is easily recognisable by the user so they can retrieve the build in the future, it is not vital in storing the build as the main identification of the builds are done by the BuildID.

The else statement is run if the user has inputted a name for the build, this hides the no name notification if it is shown beforehand, then triggers the showall name function.

```
function showall(name){
    document.getElementById("titletochange").innerHTML ="Editing Build - " +
name;
    document.getElementById("newbuildcpu").style.display="block";
    document.getElementById("instruct").style.display="none";
    document.getElementById("buildname").style.display="none";
}
```

The function is triggered and the name, inputted by the user, is sent to the function, this changes the title from New Builds to 'Editing Build' + the name of the build the user entered. In addition, the function hides the build name input box as well as the instructions and shows the cpu selection table.

The startlisting function also triggers the List Components function for each component. This function calls the API and retrieves and formats the information to be shown in the component tables.

```
function ListComponents() {
    //debugger;
    document.getElementById("startbutton").style.display = "none";
    document.getElementById("legend").style.display = "block";
    console.log("invoked List"); //console.log your BFF for debugging client
side - also use debugger statement
    const url = "/components/list"; // API method on web server will be
in Users class, method list
    fetch(url, {
        method: "GET", //Get method
    }).then(response => {
        return response.json(); //return response as JSON
    }).then(response => {
        if (response.hasOwnProperty("Error")) { //checks if response from the web
server has an "Error"
            alert(JSON.stringify(response)); // if it does, convert JSON
object to string and alert (pop up window)
        } else {
            formatList(response); //this function will create an HTML
table of the data (as per previous lesson)
            console.log(response);
        }
    });
}
```

The ListComponents function is run for every component. It hides the start button from the user, as well as showing the legend. The function then communicates with the list api which will retrieve the data from the database. This function will then call the format list function if the data is retrieved correctly and no errors are found.

```

function formatList(myJSONArray){
    console.log("Formatting")
    let CPUhtml = "";
    let GPUhtml = "";
    var i = 1;
    for (let item of myJSONArray) {
        let stringi = String(i);
        let ident = "id = 'cpuname" + stringi + "' onclick = 'selectedcpu(" +
stringi + ")';";
        let identgpu = "id = 'gpuname" + stringi + "' onclick = 'selectedgpu(" +
stringi + ")';";
        CPUhtml += "<tr class = 'tablediv'><td>" + item.CPUname + "</td><td>" +
item.Cores + "</td><td>" + item.Clock + "</td><td>" + item.Socket + "</td><td>" +
item.Process + "</td><td>" + item.TDP + "</td><td>" + item.L3Cache +
"</td><td>" + item.ReleaseDate + "</td><td " + ident + ">Select</td></tr>";
        GPUhtml += "<tr class = 'tablediv'><td>" + item.GPUname + "</td><td>" +
item.Chip + "</td><td>" + item.shaders + "</td><td>" + item.GPUBus + "</td><td>" +
item.GPUMemory + "</td><td>" + item.GPUClock + "</td><td>" + item.GPUMemclock +
"</td><td>" + item.GPUrelease + "</td><td " + identgpu + ">Select</td></tr>";
        i++;
    }
    document.getElementById("CPUtable").innerHTML = CPUhtml;
    document.getElementById("GPUtable").innerHTML = GPUhtml;
}

```

The FormatList function creates and populates the tables to be shown to the user. The information is retrieved from the List Component api which is then turned into a string and placed into the correct row in the html. In addition, the code is in a for statement to ensure that the correct amount of rows is created and populated depending on the amount of rows in the database. The for statement also allows for each row of the table to be given a unique ID which is done through incrementing i by 1 on each iteration of the statement, This means when the user selects the component the code can find out which component that is easily and therefore save it in the database.

Once formatted the code is then sent to the html file and is filled in to create the table on the page. The code is sent to each component table which is identified by the name of the component + table.

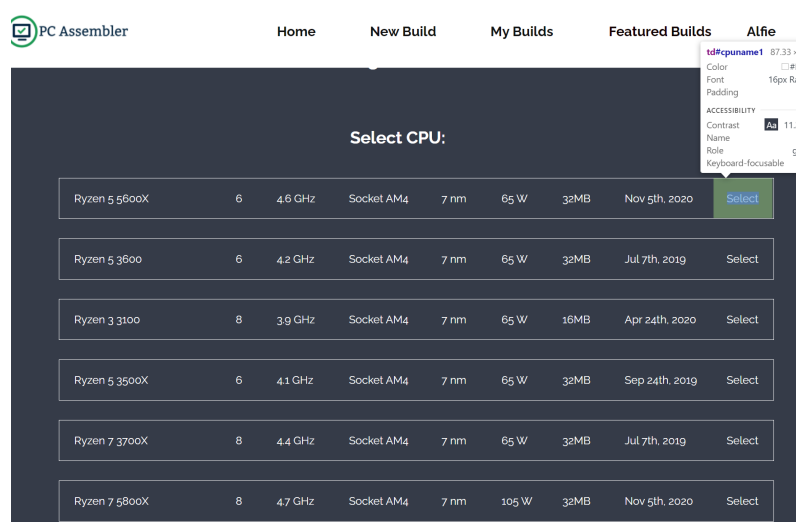
```

<section id="newbuildcpu" class="dark" style="min-height: 479px; overflow-y: hidden; display: none;">
    <div id="identifier"></div>
    <div class="innerwidth">
        <div class = "overalltable">
            <legend id = "legend" class="tabletitle" style="display: none">Select CPU:</legend>
            <table class = "tabletable" id="CPUtable"></table>
        </div>

    </div>
    <div id = "cpuselected" class = "currentlyselected">Nothing Selected</div>
    <div style="background-color: #353b48; height: 30px;"></div>
    <div class = "nothingselected" id="notselect">Please Select a component</div>
    <input type='button' id="showmorecpu" class="showmore" onclick="showgpu()" value='Next' />
    <div style="background-color: #353b48; height: 60px;"></div>
</section>

```

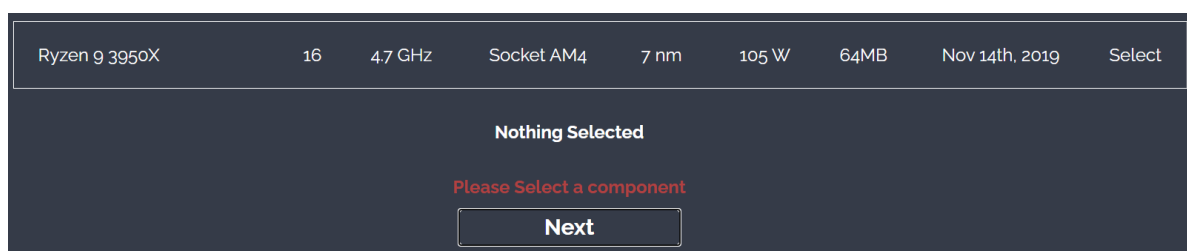
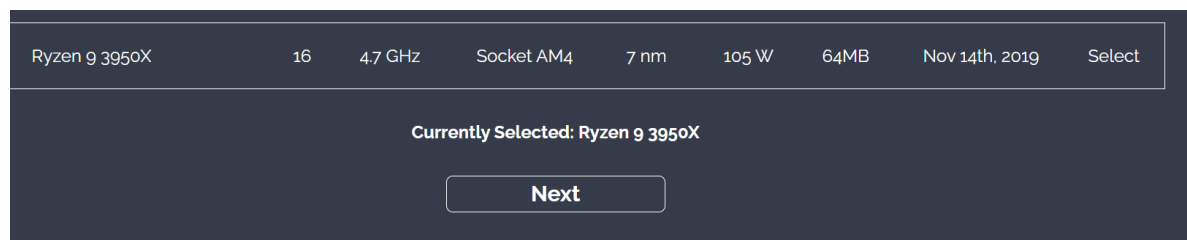
The HTML code above shows where the code created in the javascript is put and how the javascripts are called using onclick commands.



The screenshot shows that a row is made for each component in the cpu database, in addition, the code has created and appended an id to the select column which matches the component id stored in the database and therefore can be linked back so the code can run and find the information about which component the user selected without having to save all the details. This also makes it

easier to save in the saved builds database as all that needs to be done is the appended id variable needs to be saved as it is the same as the component id.

The bottom of the page displays the component that the user selected as well as the next button so the user can move on to the next component. In addition, by clicking the next button, the code will ensure that a component has been selected before progressing onto the next component.



This ensures that the code following this has no issues by trying to save null values in the database which will result in an error as the database is programmed to not have any null values in the saved build columns. This is easy to add and in the long run makes it easier and reduces the amount of possible errors a user could experience on the website. The code to do this is repeated in every 'show'component' function, This could be changed so that it was in another function however this may be harder to do as the variable and identifiers change between pages and tables, therefore these would have to change in the code as well.


```
function showgpu(){

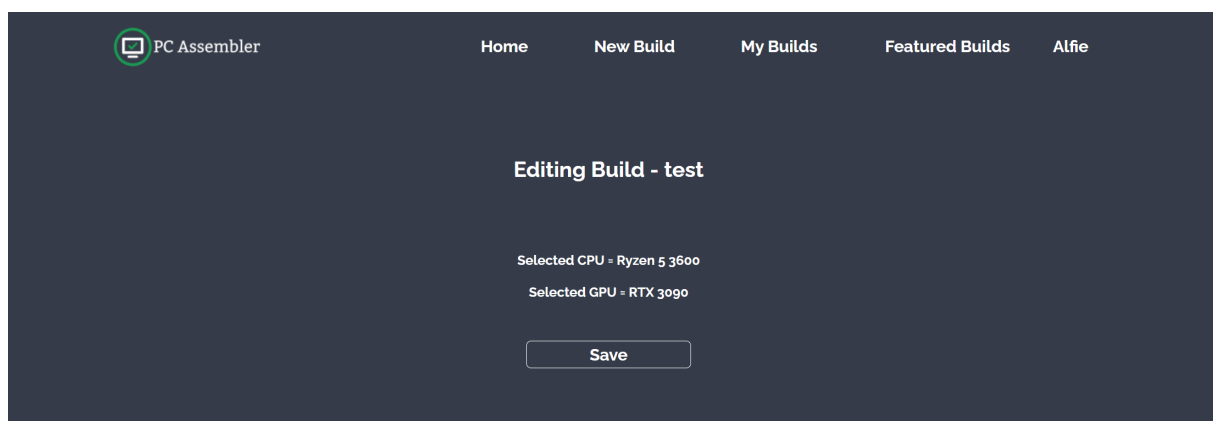
    if(document.getElementById("cpuselected").innerHTML == "Nothing Selected"){
        document.getElementById("notselect").style.color = "#b04141";
    }else {
        window.scrollTo(0, 0);
        document.getElementById("newbuildcpu").style.display = "none";
        document.getElementById("newbuildgpu").style.display = "block";
    }
}
```

The function will also reset the users view to the top of the page, this is not necessary but makes the webpage more user friendly. The code will also show the no selection message if no component is selected. Lastly, the code will hide the current component table and show the next one, such as in the code above which hides the cpu table and shows the gpu table.

On the last component page, the next button points to a final overview which gives the users an overview of what parts they have selected and the final save button.

```
function savethebuild(){
    document.getElementById("save").style.display="block";
    document.getElementById("newbuildgpu").style.display="none";
    document.getElementById("savedata").innerHTML = "Selected CPU = " +
selectedcpuvar + "<br><br> Selected GPU = " + selectedgpuvar +
"<br><br>";
}
```

The code above hides the final component table, which in this case is gpu by changing the display style to none, it also shows the save div which contains the overview. The code also created the overview html from the variables which are defined globally and changed in the respective code when the user selects a component.

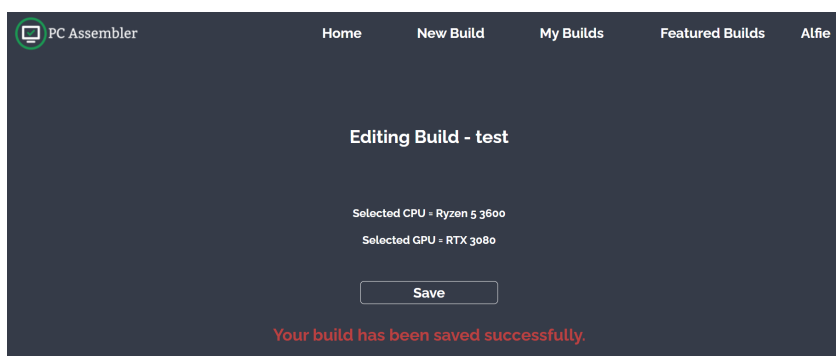


The save button triggers the finalsave function which pushes the build information to the database to be saved. The code hides the save button so the user does not trigger the function and therefore save multiple times.

```
function finalsave(){
    document.getElementById("showmorecpu").style.display="none";
    pushbuild(selectedcpuvar, selectedgpuvar, userid);
}
```

```
function pushbuild(cpu,gpu,userid) {
    console.log("Invoked AddUser()");
    const cpu1 = cpu;
    const gpu1 = gpu;
    const userid1 = userid;
    let url = "/savedbuilds/add";
    fetch(url, {
        method: "POST",
        body: userid1, cpu1, gpu1,
    }).then(response => {
        return response.json()
    }).then(response => {
        if (response.hasOwnProperty("Error")) {
            alert(JSON.stringify(response));
        } else {
            document.getElementById("savedconfirm").style.display="block";
        }
    });
}
```

The code above communicates with the API in savedbuilds which adds the component id's to the database. The code waits for a response from the API, which if it contains the keyword Error, will inform the user and log the error, if not, then the code will display the successfully saved notification. This code can be checked by manually checking the database for the correct information and that it is stored in the correct location.



The correct information and data is stored in the database in the correct locations:

	SaveID	UserID	CPUID	GPUID
1	1	1	7	12

Saved Builds:

The next page that i developed was the saved builds page which is accessible for a logged in user to be able to select their previously saved builds and read details about it as well as edit the components they have saved. The basis of the page is similar to the component selection page, using the same table method, as well as the same selection method, just displaying different information and pointing to different areas of the website.

```
<section id="savedbuild" class="dark">
  <div style="background-color: #353b48; height: 60px;"></div>
  <div id="identifier"></div>
  <div class="innerwidth">
    <div class = "newbuilddtitle" id="titletochange">Select one of your saved
builds below.</div>
  </div>
  <div style="background-color: #353b48; height: 30px;"></div>
</section>

<section id="savedbuild" class="dark" style = "min-height: 479px; overflow-y:
hidden; display: block;">
  <div id="identifier"></div>
  <div class="innerwidth">
    <div class = "overalltable">
      <legend id = "legend" class="tabletitle" style="display: none">Your
Builds:</legend>
      <table class = "tabletable" id="buildtable"></table>
    </div>

  </div>

  <div style="background-color: #353b48; height: 60px;"></div>
</section>
```

The HTML above displays the table and title for the saved builds webpage. The table element is empty in the html as this is filled in the javascript using the information retrieved from the API and database. The id of the table element must be the same as the table id in the javascript, which was causing issues previously due to a spelling error which meant the javascript was not pointing to the table element and therefore not filling in the information. Unlike the component selection page, every section on this page is shown by default as rather than hiding sections that the user should not see yet, they are on another html page. This reduces the complexity of the code as hiding and showing the correct sections can be hard to get right. The HTML for the web page also includes the same menu html which is used on every web page as well as the html for the custom side scroll bar, which again is the same on every page.

The javascript on the page is triggered when the user selects the my build button from any page on the menu bar, the javascript, once triggered will not only direct the user to the savedbuilds html page but begin to collect the information from the database and trigger other javascript to allow the build selection to work as expected.

```

function startsaved() {
    window.open("savedbuilds.html", "_self");
    //debugger;
    statuscheck();
    let id = readCookie("userid");
    console.log("invoked List"); //console.log your BFF for debugging client
side - also use debugger statement
    const url = "/savedbuilds/list/" + id; // API method on web server
will be in Users class, method list
    fetch(url, {
        method: "GET", //Get method
    }).then(response => {
        return response.json(); //return response as JSON
    }).then(response => {
        if (response.hasOwnProperty("Error")) { //checks if response from the web
server has an "Error"
            alert(JSON.stringify(response)); // if it does, convert JSON
object to string and alert (pop up window)
        } else {
            savedbuilds(response); //this function will create an HTML
table of the data (as per previous lesson)
            console.log(response);
        }
    });
}

```

The javascript above is used. When triggered, the code first redirects the user to the savedbuilds.html file in the same tab. It will then trigger the statuscheck() function to ensure the user is logged in and that the stored information in the cookies is correct as this information is used to display the saved build and is therefore vital to prevent the incorrect or other users information being shown to the wrong person. Once the status check has been completed and the user has not been logged out, the code will read the cookie for userid. This is then passed to the API using the url /savedbuilds/list + the id read from the cookie. This gives the API the information required to pull the correct saved builds that the user has requested. Once the API has responded, The code will change depending on the status of the response. If the response has the property of an error, the user will be notified and the error will be logged. On the other hand, the code will start the next javascript, savedbuilds, pushing with it the response from the API to format the saved builds table and display the information to the user.

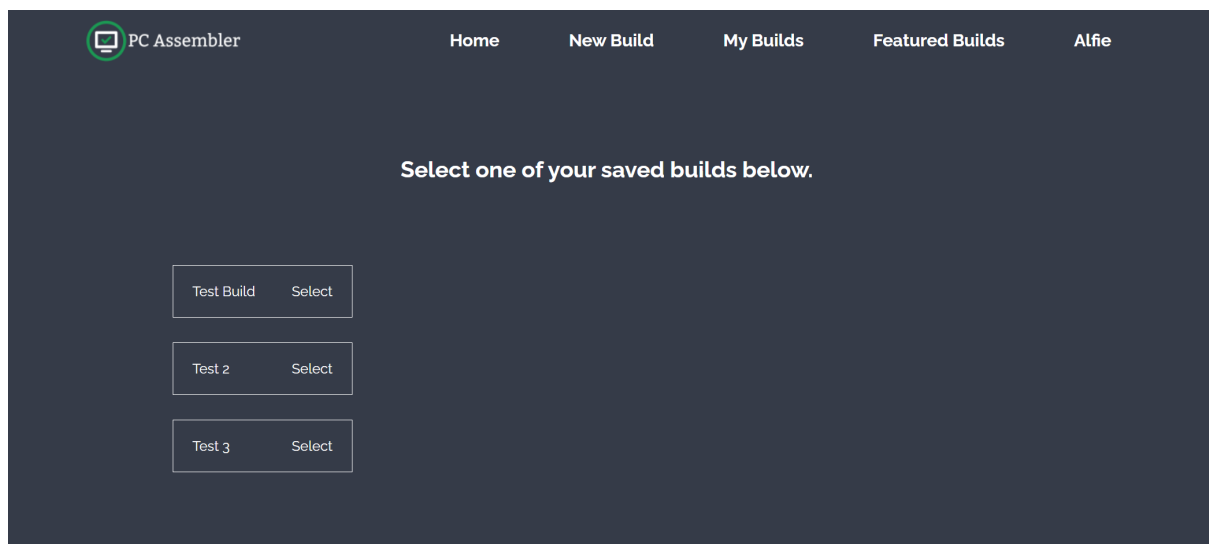
```

function savedbuilds(myJSONArray){
    console.log("Formatting")
    let buildhtml = "";
    var i = 1;
    for (let item of myJSONArray) {
        let stringi = String(i);
        let ident = "id = 'build" + stringi + "' onclick = 'selectedbuild(" + stringi + ")';";
        buildhtml += "<tr class = 'tablediv'><td>" + item.Name + "</td><td>Select</td></tr>";
        i++;
    }
    document.getElementById("buildtable").innerHTML = buildhtml;
}

```

The javascript above is used to format the html table. When called, the function receives the information retrieved from the API which is formatted by the javascript, by creating the variable, buildhtml. This variable is the html string that will be placed inside the table element in the savedbuilds html. This also adds the select button which will be used to tell the code which build the user has selected. The code also creates a unique id for the build by incrementing i by 1 on each iteration of the for loop. This means the id of the row will be build + a number which equates to the build id of the same build in the saved builds table. The final line of code in the javascript is used to push the final HTML string to the table element and therefore display it to the users and allow them to select and view their own saved builds.

Issues I had whilst coding this included a multiple misspellings in both the url for the API, which resulted in 404 errors and therefore the code not working, and secondly, an error in the spelling of 'buildhtml' meant that the code pointed to an incorrect html element, therefore not formatting and displaying the table correctly.



Once the code was working correctly, the user was shown a table with their saved builds. I added a new saved build to the database manually with a different userid to ensure that only the users saved builds were being shown. The select button worked correctly by triggering another function on click. The unique id each had was pushed to the function so it can be read and the code is aware of which build the user has selected. The code received the unique id to give the build from the API which collected the buildid from the database. By incrementing the unique id by 1 each time would not have worked as if another users saved builds have a build id of 15 as they are further down the database, the code would give the unique id of 1, 2, 3 etc which would mean the user would be displayed the saved builds of the user whose buildid's are 1, 2 or 3 when they select a build they want to edit.

The CSS for the table could also be edited to make the table show in the centre of the page, however, although this may be easy for navigability and user experience, it does not affect the code or the functionality of the webpage and system.

The next function I have coded is triggered when the user selects the saved build they want to view or edit. The code will retrieve the component information from the database and show this to the user a dedicated page. This webpage will show the option to edit the build, which will reopen the component selection page under the same name as the original build, which will be overwritten and updated depending on what the user selects and decides to do.

```
function selectedbuild(buildid){
  console.log("invoked retrieve using buildid = " + buildid);    //console.log
  //your BFF for debugging client side - also use debugger statement
  const url = "/savedbuilds/retrieve/" + buildid;                // API method on
  //web server will be in Users class, method list
  fetch(url, {
    method: "GET",          //Get method
  }).then(response => {
    return response.json();          //return response as JSON
  }).then(response => {
    if (response.hasOwnProperty("Error")) { //checks if response from the web
      server has an "Error"
      alert(JSON.stringify(response));    // if it does, convert JSON
      object to string and alert (pop up window)
    } else {
      alert(JSON.stringify(response.GPUID));
      showselectedbuild(response.GPUID, response.CPUID);          //this
      //function will create an HTML table of the data (as per previous lesson)
      console.log(response);
    }
  });
}
```

The Javascript above collects the id's and information stored in the database about the selected build. This code requests the API, including the id of the build selected in the url. The API will then return the id of the selected GPU and CPU stored in the database. This then provides the information required to retrieve the full information, such as the component name from the component API, and therefore means the information does not need to be stored in the saved build database. Once the code receives the id of the components, it will forward them to the showselectedbuild javascript which will retrieve the information and display it to the user.

The Javascript below is triggered by the code above, it is sent the ID of the components, which is then used to call the relevant API's in order to retrieve the name of the specific components that were saved for this build. The API for the CPU is sent the id of the saved cpu, this will then search the database for the cpu with that id and respond with the information. The code will take the name of the cpu out of the response, and save it to the variable, selectedcpuinfo. This variable is then used on the dedicated saved builds page to show the name of the cpu that the user had selected for this specific saved build. The code does the same for each component, including the gpu shown below.

```

function showselectedbuild(CPU, GPU){
    let selectedgpuinfo = "";
    let selectedcpuinfo = "";

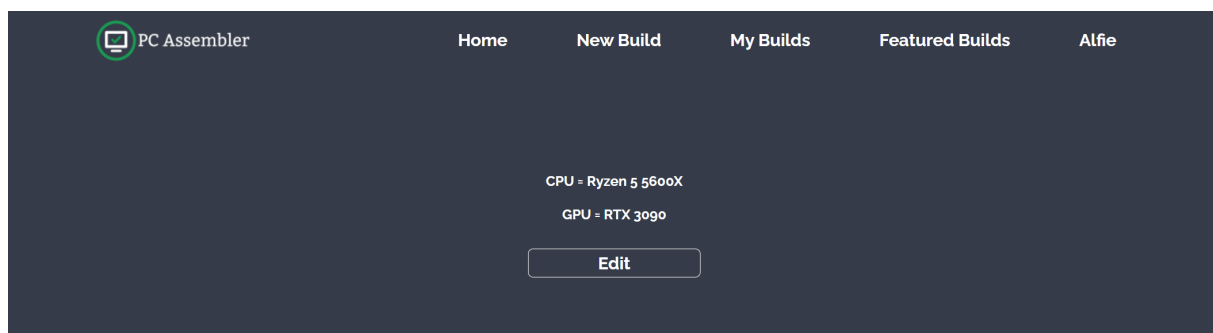
    var url = "/components/getcpu/" + CPU;
    fetch(url, {
        method: "GET",
    }).then (response => {
        return response.json();
    }).then(response => {
        if(response.hasOwnProperty("Error")){
            alert(JSON.stringify(response));
        }else {
            alert(response.CPUname);
            selectedcpuinfo = response.CPUname;
        }
    })

    var url = "/components/getgpu/" + GPU;
    fetch(url, {
        method: "GET",
    }).then (response => {
        return response.json();
    }).then(response => {
        if(response.hasOwnProperty("Error")){
            alert(JSON.stringify(response));
        }else {
            alert(response.GPUname);
            selectedgpuinfo = response.GPUname;
        }
    })

    document.getElementById("selectedbuildcpu").innerHTML = selectedcpuinfo;
    document.getElementById("selectedbuildgpu").innetHTML = selectedgpuinfo;

}

```



The Image above shows the information page the user gets when they have selected a build showing what the components are as well as a button which triggers the edit function which allows the user to change the components in the build.


```

function editbuild(){
    let buildid = globalbuildid;
    console.log("invoked get name using id "+buildid);    //console.log your BFF
    for debugging client side - also use debugger statement
    const url = "/savedbuilds/getname/" + buildid;        // API method on web
    server will be in Users class, method list
    fetch(url, {
        method: "GET",          //Get method
    }).then(response => {
        return response.json();    //return response as JSON
    }).then(response => {
        if (response.hasOwnProperty("Error")) { //checks if response from the web
            server has an "Error"
            alert(JSON.stringify(response));    // if it does, convert JSON
            object to string and alert (pop up window)
        } else {
            let stringres = String(response.Name);
            selectedbuildname = stringres;
            console.log(response);
            document.cookie="editing?=" + selectedbuildname;
            window.open("newbuild.html", "_self");
            startlisting();
        }
    });
}

```

The Javascript above is used to begin the edit build process when the user selects the edit button. When triggered, by clicking the edit button, the code will retrieve the name of the build from the build id using the getname API. The API receives the id of the build in the url and will check the database for saved builds with the id and will retrieve the name of the build to send back to the javascript. Once the javascript receives the response with the name of the build, the code creates a cookie with the name of the build, which when the user first logs in is set to be false, the code will then open the new build html page, which begins the selection process, bypassing the instructions and requirement to put in the name of the build as the startlisting code has been edited with an if statement to check that the cookie is not false, which if it is, it will show instruction and name box as normal, however if it is not false, it will go directly to the component selection with the name that was stored in the cookie and selected by the user. The cookie is changed back to false if the user visits the new build page by selecting any other buttons on the website such as the new build button on the nav bar, this prevents the user from being stuck in a loop of editing the same build and being able to create new ones.

The Screenshot below shows what the user sees when they click the edit button by selecting a build in the saved build list. This can be improved by adding a highlight or message showing the user what the currently saved component is, This may have made the webpage more user friendly however, may have caused issues with overwriting the new component rather than changing it as the user had expected if it is not coded correctly.


PC Assembler

Home
New Build
My Builds
Featured Builds
Alfie

Editing Build - Test Build

Select CPU:

Ryzen 5 5600X	6	4.6 GHz	Socket AM4	7 nm	65 W	32MB	Nov 5th, 2020	Select
Ryzen 5 3600	6	4.2 GHz	Socket AM4	7 nm	65 W	32MB	Jul 7th, 2019	Select
Ryzen 3 3100	8	3.9 GHz	Socket AM4	7 nm	65 W	16MB	Apr 24th, 2020	Select
Ryzen 5 3500X	6	4.1 GHz	Socket AM4	7 nm	65 W	32MB	Sep 24th, 2019	Select
Ryzen 7 3700X	8	4.4 GHz	Socket AM4	7 nm	65 W	32MB	Jul 7th, 2019	Select

This was tested by going through the normal part selection process and saving the build. Once that was shown as successful, by manually checking the savedbuilds database, showed that the code had worked as expected because the build the user selected to edit had been updated to match the newly selected parts, overwriting the old build, and therefore editing the selected builds as expected.

	BuildID	userID	CPUID	Name	GPUID
1	1	98	17	Test Build	48
2	2	98	3	Test 2	4
3	3	98	4	Test 3	7
4	4	99	4	Test - Incorrect User	6

	BuildID	userID	CPUID	Name	GPUID
1	1	98	1	Test Build	3
2	2	98	3	Test 2	4
3	3	98	4	Test 3	7
4	4	99	4	Test - Incorrect User	6

The Screenshots above show the savedbuilds database, the left screenshot was taken before the first build, with the id of 1 was edited, the second picture shows after, both the CPUID and GPUID had changed to the id of the new components selected and therefore showing the new selections in the selected build overview page.

Testing


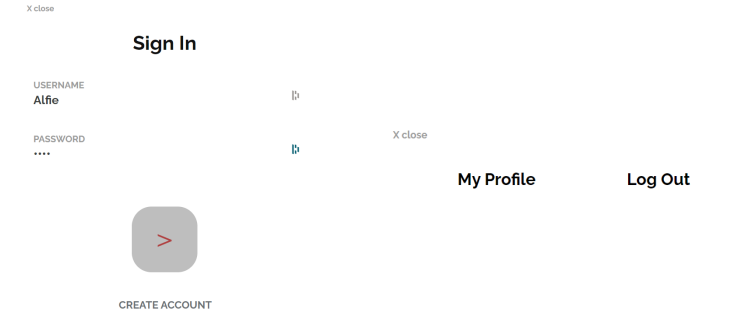
Testing Table


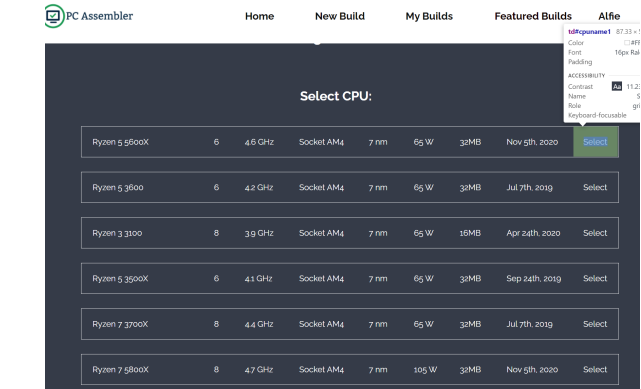
Test	Module/ Subroutine	Inputs	Expected Output	Pass/Fail
1	Menu bar	Press the menu buttons.	The correct page or form to be shown depending on what is clicked.	Pass
2	Login Systems	Username, password, enter button.	If correct login details are entered, the user should be logged in and can access features, if details are incorrect, the user should be notified.	Pass
3	New Build naming system	Name of build.	The name of the build is able to be entered and is read by the system. If the user does not enter a name then they should be notified. If a name is entered the build process should begin.	Pass
4	Selection of Components	Buttons pressed by each component	When a user clicks the select button, the system should save the component that is clicked on. If no component is picked the user should be notified.	Fail
5	Saving of new build	The save button is pressed	The users selected components should be saved in the database under the usersid and then notify the user of a successful save.	Fail
6	Display the saved builds of the user	Saved builds page is shown	The user's saved build name should be shown allowing the user to select a previously saved build.	Pass
7	Display the information about the saved builds.	Saved build is selected.	Information about the users selected saved build is shown including component names.	Fail
8	Allow User to edit saved builds	Edit button is selected	The user can edit the selected components of the build and save it under the same saved build name.	Pass
9	Login Checks	Page is loaded	The user's details stored in the cookies are matched with the details stored in the database and the user is logged out if there is a mismatch.	Pass

Remedies:

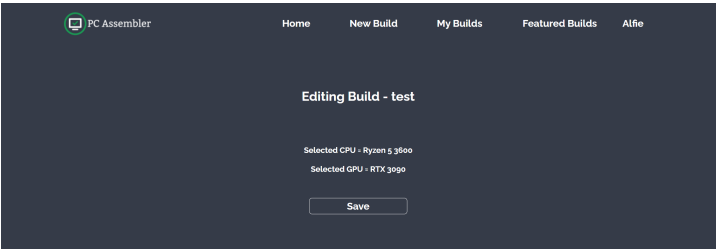
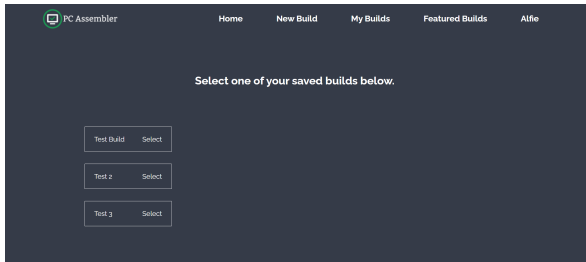
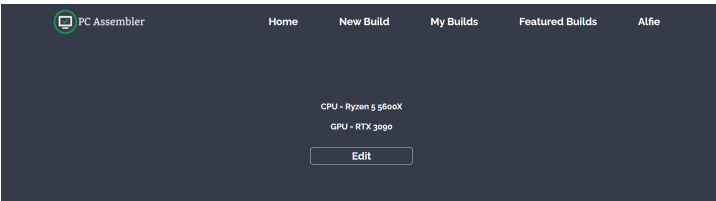
Test	Action taken	Justification	Pass/Fail
4	Function which gives the component its unique id was changed to properly increment by 1 so the id's would differ as before they would show the same id and therefore component.	This was the best way to fix the issue as the basis of the code was already written out and was a 1 line fix, doing it another way would require the whole function to be re-written and therefore not worthwhile.	Pass
5	Similar issue as above where the same id is given to the build so all builds are saved under the same id and therefore rewrite the previous build.	This fix was one of the only ways to implement this as the build id was to increase by 1, It could also not be done another way as other code, which relies on the id incrementing by 1 would have to be re-written	Pass
6	Incorrect API url for the name of the build, this meant the API was not triggered, Easy fix by changing the url of the API in the code.	This was best done this way as rewriting the code would have been a waste of time and may not have been done another way without using the API.	Pass

UI and Code for modules/routines:

Test	Code	UI
1	<pre> <nav class="nav"> <div class="innerwidth"> <button class="menubutton"> </button> <div class="barmen"> <button class="loginbutton" onclick="homepage()" id = "otherloginbutton">Home</button> <button class="loginbutton" onclick="newbuild()" id = "otherloginbutton">New Build</button> <button class="loginbutton" onclick="startsaved()" id = "otherloginbutton">My Builds</button> <button class="loginbutton" onclick="startlistingsaved()" id = "otherloginbutton">Featured Builds</button> <button class="loginbutton" onclick="openform()" id = "loginbutton"> Login </button> </div> </div> </nav> </pre>	
2	<pre> function postUserLogin() { console.log("Invoked postUserLogin() "); var formData = new FormData(document.getElementById('loginnewform')); var url = "/users/login"; fetch(url, { method: "POST", body: formData, }).then(response => { return response.json(); //now return that promise to JSON }).then(response => { if (response.hasOwnProperty("Error")) { document.getElementById("errorMessage").style.display="flex"; } else { document.cookie = "token=" + response.token; document.cookie = "username=" + response.username; openafterlogin(); } }); } </pre>	

3	<pre>function startlisting(){ let tempname = document.getElementById("buildname").value; name = String(tempname); if(name == ""){ document.getElementById("noname").style.display="block"; }else{ document.getElementById("noname").style.display="none"; showall(name); ListComponents("CPU"); ListComponents("GPU"); } }</pre>	
4	<pre>function formatlist(myJSONArray){ console.log("Formatting") let CPUhtml = ""; let GPUhtml = ""; var i = 1; for (let item of myJSONArray) { let stringi = String(i); let ident = "id = 'cpuname' + stringi + " ' onclick = 'selectedcpu(" + stringi + ")';"; let identgpu = "id = 'gpuname' + stringi + " ' onclick = 'selectedgpu(" + stringi + ")';"; CPUhtml += "<tr class = 'tablediv'><td>" + item.CPUname + "</td><td>" + item.Cores + "</td><td>" + item.Clock + "</td><td>" + item.Socket + "</td><td>" + item.Process + "</td><td>" + item.TDP + "</td><td>" + item.L3Cache + "</td><td>" + item.ReleaseDate + "</td><td>" + ident + ">Select</td></tr>"; GPUhtml += "<tr class = 'tablediv'><td>" + item.GPUname + "</td><td>" + item.Chip + "</td><td>" + item.shaders + "</td><td>" + item.GPUBus + "</td><td>" + item.GPUMemory + "</td><td>" + item.GPUClock + "</td><td>" + item.GPUMemclock + "</td><td>" + item.GPURelease + "</td><td>" + identgpu + ">Select</td></tr>"; i++; } document.getElementById("CPUtable").innerHTML = CPUhtml; document.getElementById("GPUtable").innerHTML = GPUhtml; }</pre>	
5	<pre>function savethebuild(){ document.getElementById("save").style.display="block"; document.getElementById("newbuildgpu").style.display="none"; document.getElementById("savedata").innerHTML = "Selected CPU = " + selectedcpuvar + "

 Selected GPU = " + selectedgpuvar + "

"; }</pre>	
6	<pre>function savedbuilds(myJSONArray){ console.log("Formatting") let buildhtml = ""; var i = 1; for (let item of myJSONArray) { let stringi = String(i); let ident = "id = 'build' + stringi + " ' onclick = 'selectbuild(" + stringi + ")';"; buildhtml += "<tr class = 'tablediv'><td>" + item.Name + "</td><td>Select</td></tr>"; i++; } document.getElementById("buildtable").innerHTML = buildhtml; }</pre>	
7	<pre>function showselectedbuild(CPU, GPU){ let selectedcpuinfo = ""; let selectedgpuinfo = ""; var url = "/components/getcpu/" + CPU; fetch(url, { method: "GET", }).then(response => { return response.json(); }).then(response => { if(response.hasOwnProperty("Error")){ alert(JSON.stringify(response)); }else { alert(response.CPUname); selectedcpuinfo = response.CPUname; } }) var url = "/components/getgpu/" + GPU; fetch(url, { method: "GET", }).then(response => { return response.json(); }).then(response => { if(response.hasOwnProperty("Error")){ alert(JSON.stringify(response)); }else { alert(response.GPUname); selectedgpuinfo = response.GPUname; } }) document.getElementById("selectedbuildcpu").innerHTML = selectedcpuinfo; document.getElementById("selectedbuildgpu").innerHTML = selectedgpuinfo; }</pre>	

Video Testing

Tester Name: Ryan Adams

Test 1 - Login System Test.

Ryan was given the task to set up a new account and test that the login system worked. He also was asked to test that the login check and validation worked by only allowing the user to access certain pages if they are logged in.

<https://drive.google.com/file/d/1RzdkUPa4siRyPNAaVJ0SpEYukZPMcCrL/view?usp=sharing>

Testers Thoughts:

Ryan said that the login system worked well and was relatively easy to use and understandable. Ryan also said that he had no issues with the user validation with the cookies and that it logged him out when expected and did not log him out when he did not want to be. Ryan also mentioned that when having to login on the dedicated login page, it would be better to redirect to the target page after login, rather than back to the homepage. This is hard to do as the javascript is reset when the new page is loaded, therefore variables are reset.

Test 2 - Create a New Build.

Ryan was then given the task to create and save a new build by selecting the components.

https://drive.google.com/file/d/1zKscfz_g5ggGa_Wp_fBlZWd094TW1mpJ/view

Tester Thoughts:

Ryan said that the new build system worked as expected, the name validation was working correctly as well as the part selection and displaying the selected components at the end worked well. However, Ryan said that before he had pressed the save button, the successfully saved build notification was showing, this was fixed easily by making the element default display style be none, and adding a line to the function triggered by the save button to change the display style to block and therefore show it. After Ryan had completed his test, I manually checked the database to ensure that the build Ryan had saved was present and was showing the correct parts and user id.

Task 3 - Access and Edit a previously saved build.

Ryan then went to the saved builds page to see all the saved builds under his userid and edit the build he had just created to include different parts.

<https://drive.google.com/file/d/1m9sw8pi93ixtLfD2fP335XfVi1AejCZw/view>

Tester Thoughts:

Ryan said that the website showed all the saved builds well and that it was easy to understand. Ryan also said that when he clicked on the build he had created, it showed the correct parts that were selected during the new build process. Ryan said that the page which showed the build information worked, however, he said it could be improved by adding a title which shows the user which build they had selected. This could be done by bringing the name finding function earlier before the page is shown, rather than after the edit button is selected. In addition, Ryan said that it would be better if the instruction screens were skipped when editing a previously saved build as it can be confusing to the user if the webpage is asking for a new name. This could be done by removing the name input button if the user is visiting from the my builds page, however, this did not affect the function of the code as it still allowed ryan to start even though he did not input a name.

Tester Final Thoughts:

Overall, Ryan said that the website worked well and as expected, he said it was user friendly and relatively easy to navigate. In addition, Ryan said that the website was good looking and laid out well so it was easy to find what the user wanted. Ryan said that the systems worked well, such as saving a build and logging in, he mentioned that the process was quick and the website was fast so the user would not be waiting around for long. Suggestions made by ryan included a more rounded login system, for example, adding a forgotten password button, the ability to change passwords and the ability to delete your account. This can all be added with further development time without major rewrite or changes to the original code. Ryan also suggested that making sure elements of the HTML were visible only when they are needed, for example, the successfully saved build notification is shown at all times. This was an error with a misspelling of the id of the notification element which meant it was not hidden. Other than small fixes that are easily fixed with small tweaks, the main functionality and key components of the website are working well and are reliable, meaning there are no times where the code may work once and then stop.

Robustness testing:

Tester Name: Jacob Doven

Jacob was used to test the robustness of the website, this is because he completed the robustness testing for his own project, and, as a computer science student is aware of what to look out for.

Jacob was given a login, and the code of the website and told to test every feature of it.

<https://drive.google.com/file/d/1EZnEJVL75zXec2FHCLMQ5ZGBQBwznrGI/view?usp=sharing>

Tester Thoughts:

Jacob had no issues with the login system, both account creation and logging in worked well. The validation would ensure that there were no spaces in the username, If a value was added to the username, the login was not approved either, this meant the only way to log in to the website was by using the exact details that were inputted when the user set up the account. The homepage had no issues, except for the my saved builds button, which would reload the homepage rather than go to the saved builds page. This was fixed by changing the function that was triggered on click to the show saved builds function. Other buttons on the homepage worked well, and elements that had user interaction such as the change colour on hover worked as intended. The featured build section on the homepage worked well, the details being shown when the user hovers over the build. The components were also different for each build, which shows the unique id creation of the featured builds worked and prevented the same build being shown. The new build page worked as expected with the validation and all buttons working, however, the validation of the build name could be improved as issues arose when Jacob chose a 'space' as the name of the build. This caused issues with the retrieval of the build when Jacob wanted to edit the components, however the build was saved correctly in the database. This can be fixed by adding further validation to the name of the build which prevents the user from being able to have spaces as the name of the build. The build edit system worked with the other saved builds which suggests that the issue is with the name of the build rather than another fundamental code error.

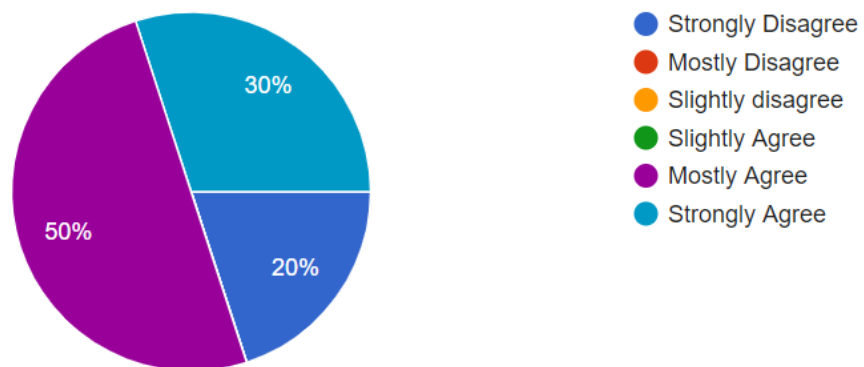
Overall, the robustness testing of the website revealed small errors which can be easily fixed, as well as other issues with validation, such as the name of the new build, however, overall the main features of the code were reliable and worked well every time, including the login systems validation which worked well and prevented any erroneous inputs as well as inputs which did not conform to the validation rules and requirements of the login.

Usability Testing

Including Jacob and Ryan above, the code was sent to 10 people who were asked to use the website and complete a survey. This gave the best idea as to what someone who is seeing the website for the first time thought about usability and ease.

The text is easy to read and large enough

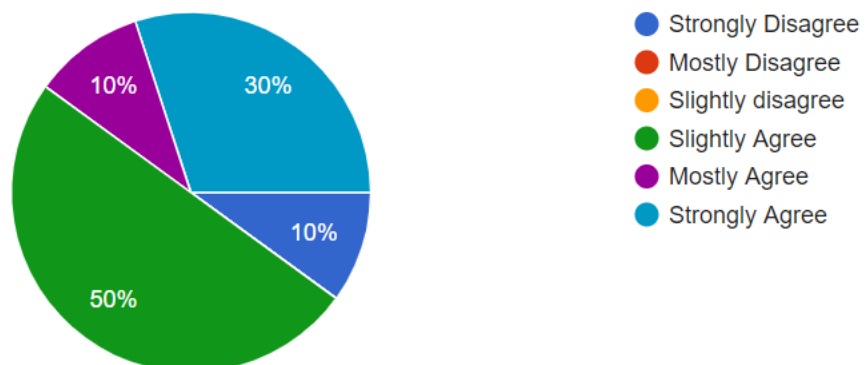
10 responses



The graph above shows that the people surveyed, the majority of people agreed that the text was easy to read and large enough. 5 people mostly agreed with this statement. However, 20% of respondents strongly disagreed with the question. This could be fixed by adding a larger text mode which would change the css to make the text font larger.

The layout of the website is easy to understand

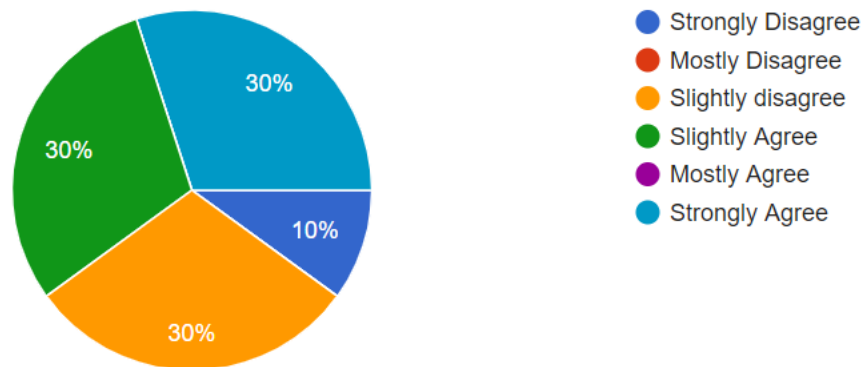
10 responses



Out of the 10 respondents, 70% agreed with the statement that the layout of the website is easy to understand somewhat. This shows that the layout works well and means that it is user friendly, even for people who have not visited the website before, nor are they familiar with using websites often.

The website is easy to navigate

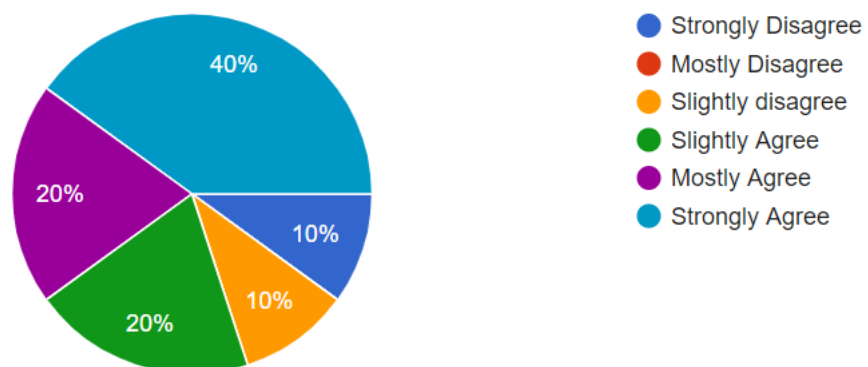
10 responses



The statement that the website was easy to navigate split the respondents. The majority however agreed that the website was easy to navigate. This could be improved, however will require a large recode of the HTML for the web page or a rename of the menu bar buttons so they make more sense to the end user.

There is enough guidance on the webpage so the users will not be confused.

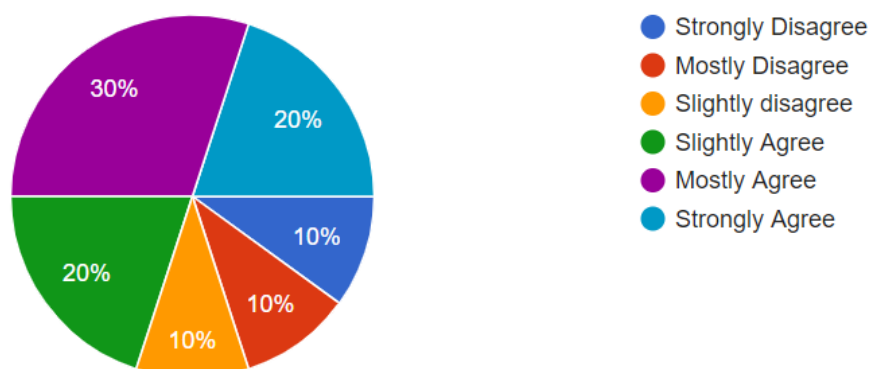
10 responses



Most people felt that the guidance on the webpage was enough so they understood what to do and were not confused. This could be improved by re writing or adding further instructions for those who may not understand the terms used, such as computer science key words.

Starting a New Build is easy.

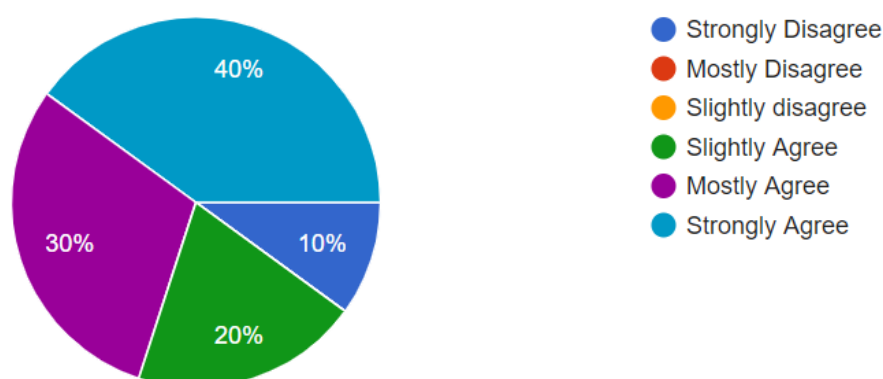
10 responses



The majority of respondents thought that starting a new build was relatively easy. This is likely due to the instructions and that the build process would not start until the user had selected begin after reading the instructions. The webpage would also validate inputs which would reduce the likelihood of errors which someone who is not familiar with using websites may find confusing.

Selecting components and reading information is easy.

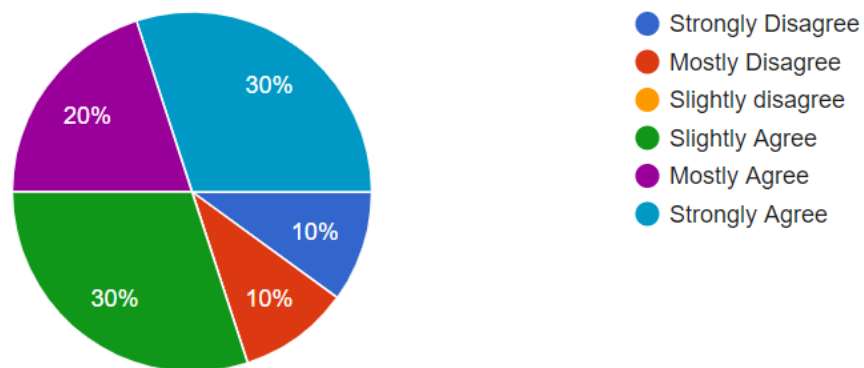
10 responses



90% of the respondents thought that the component selection process and information about the components was easy to understand and read, however this could be improved and is the likely reason for 1 person responding strongly disagree, by adding column titles to the component tables so the user can further understand what the information is such as clock speed or release date. This was probably reduced by the fact that many of the respondents were computer science students, therefore they understood what the information is showing from the units and context of the webpage.

Saving the build and overview is easy to understand.

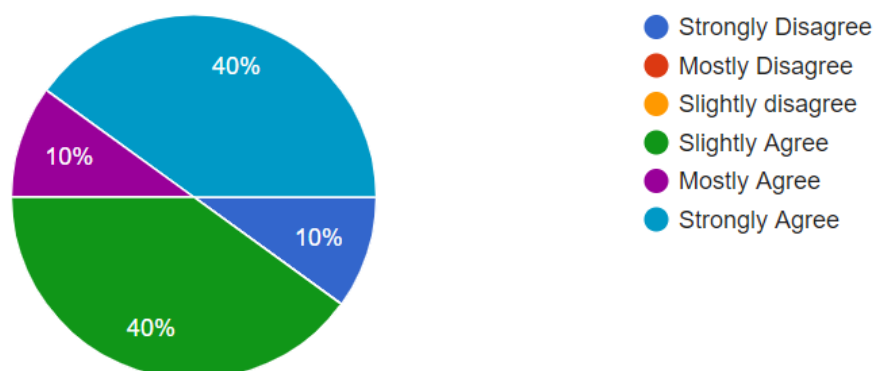
10 responses



The saved build page was easy to understand, shown in the chart above. 80% of the people surveyed agreed with the statement. 20% disagreed, however this would be hard to improve as the page shows the name of the component as well as the component itself, it is likely that the 2 people who answered this are unaware of the purpose and therefore unlikely to use the website.

Viewing previously saved builds is easy.

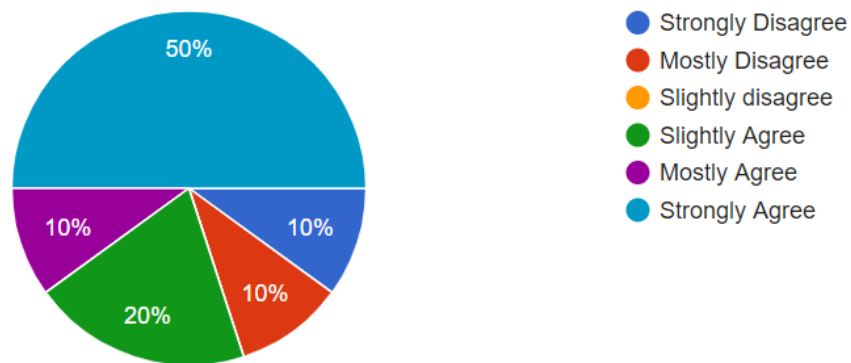
10 responses



Only 10% disagreed with the statement that viewing of previously saved builds is easy, this shows that the layout and design does not need to be changed and that this section of the website is user friendly. The layout of the saved builds table is similar to the layout of the components selection, with the same select method and code.

Selecting your builds is easy.

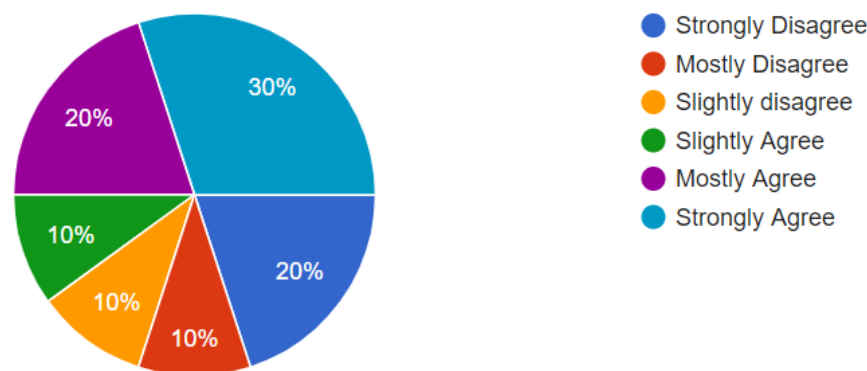
10 responses



50% of the respondents strongly agreed with this statement. The select mechanism is identical to the select method of the individual components. This shows that it is easy for the user to understand and select their saved builds.

Editing saved builds is easy and understandable.

10 responses



The editing of the saved builds is the same system that is used when originally creating the new build. The majority of users understood this system and therefore most agreed with the statement. Therefore, not much needs to be done to make the page more user friendly. It could be improved by bypassing the instruction screen when the selects their build, however this will mean the user may have forgotten the instructions since they last used the website.

Evaluation

Evaluation of Success Criteria

- Users are able to access and load the website correctly.
 - This has been met shown by the responses from the user in the usability testing. The website was hosted on a server whilst running the usability testing which allowed users to access the website from the url and not require them to download the code and setup servers. This meant I could use anyone as the testers and resulted in a broader range of users and therefore results. The Website was responsive and changed depending on the size of the browser. This meant the website was displayed correctly at all times and the usability of the website was not lost if someone was using a smaller screen.
- Users are able to log in to an account and sign up for a new one.
 - The login system and account creation works well and as intended, this is shown in both the robustness and usability testing. No users complained that the login systems resulted in errors. The login system is vital to the functioning of the website. I can also say that the system is secure as validation checks would ensure the correct user is access the information and that no cookie information was changed.
- Users are able to select components for their computer build.
 - The component selection system works and is functional therefore I believe that this success criteria was met well. The usability testing also showed that the selection process was relatively easy to use, but could be improved through the addition of column titles. The selection system saves the correct component and information in the correct place therefore, this system works.
- The Users are able to delete their account and change their password, as well as a working forgot password system is implemented..
 - This success criteria was not met. The user is currently unable to change their password or delete their account, however this can easily be done with the addition of a profile page which uses existing api's to change the details. The forgot password system could also be implemented but would require the ability for the code to send emails, which can be done, as well as provide links that point to the correct account so only that users password can be changed by them.

- The user is able to save their build to come back at a later date.
 - This success criteria has been met to a certain extent. A user is able to save builds and that is a working system. The user can also edit these builds afterwards to change the components. This system works well and is robust as shown in the robustness testing. However, the user is unable to save a build midway through the selection process. This can be added, but may result in further errors with other code where there are null values in the database.
- Users are able to browse components available and access the information.
 - This success criteria has been met. The user has access to all the information and components stored in the database during the component selection process. This information is displayed well and easy to understand as shown in the responses from the usability testing. This can be improved by adding a page that solely displays the information about the components and a list of all available components using existing api and removing the select button. This will allow the user to access the information without starting the build process
- Users can purchase the components on Amazon directly from the webpage.
 - This success criteria was not met as the user cannot access the amazon page from the website. This can be implemented by using the embed method amazon provides, storing this information in the database and retrieving it when the user selects a component, However this will be very time consuming as each of the 100 components currently in the database require a link which has to be done manually.
- Users are warned if chosen components may not be compatible with each other.
 - This success criteria was also not met. This was one of the hardest features to try and implement. This is because in order to reliably integrate it and make it robust, it would require most of the already written code to be rewritten as well as a redesign of the component database system. The compatibility would also have to be done manually which would have been extremely time consuming and not viable.
- Users are able to see recommended and showcased completed builds.
 - This success criteria has been met as the user is able to see featured builds on the webpage homepage, as well as the components used to build them.
- Users can select as many or as little components they need.
 - This success criteria has not been met as the user is required to select all the required components, otherwise, issues may be caused with the database and including null values, although there is a possibility this can be implemented.

Final Limitations of Project

I believe that there are 3 limitations to my project.

1. Size of database and information stored.
 - The database sizes for the components is very large, and will expand as more components and component types are added. This causes issues with layout of the code to display the information and could have been done with more efficient code. The design of the database also prevented some key features such as amazon pricing, and compatibility checking to be viable without a large redesign or overcomplicated, unreliable code. The way the component table was designed also meant that all the components were stored in one table, although this meant only 1 API was required to read the database, the formatting of the table was much harder and not as efficient as if each component was split into another table, this would have also made it easier for creating unique ids to allow for the price table to be separated and work, as well as the compatibility checking, by adding another table for this information.
2. Page redirect after logging in.
 - Another limitation of the project was the page redirection when logging in using the dedicated login page. I had tried to implement this system by creating global variables which were set to the page name when the menu button was clicked, which could then be read by the login function to redirect to the correct page, however, when the login page was loaded, the variables were reset and always pointed back to the index page. This could have been fixed by storing the page name in the browser cookies, or by creating a hidden by default section on the index page which shows when a user is not logged in and tries to access locked pages. This would mean the page was not reloaded and the variables would remain as they were set.
3. Lack of validation for inputs.
 - Most of the input have validation which prevents the code from running if there is not input at all, this is a level of validation which helps prevents errors with the code trying to save null values in the database which will result in errors, however, Input validation could be improved by adding duplication checks to make sure that the name is not already used, or the user does not already exist, In addition password validation could be added to improve the strength of security on the website. Another validation issue occurred with the build name input which should prevent the ability for users to have a space as the name of the build, which caused issues further down the line as seen in the robustness testing.

Maintenance

SQL is a skill that is required by the admin of the website. This is due to the way most information which is shown on the website is stored. All the API's use SQL statements to retrieve and push data. Therefore in order to add more information to the webpage, the admin will need to use and understand SQL statements. For example if a new component is added to the database, this should be done by using SQL insert statements with multiple values for each of the columns, such as part name. Likewise, the Admin should be able to use delete statements in order to delete components and other information from the database that may now be outdated or no longer needed, and Update string can also be used to change certain information in the database or a column without having to recreate the entire row of information. An Admin may be able to work the website without this knowledge, however this will take a lot more time and may result in a higher rate of errors. The admin may also be able to update the website to show any price changes in the components database, as well as any stock information. This would require the use of the UPDATE SQL string as manually changing this information in the database is more confusing and time consuming.

The Admin would also need to be able to understand both javascript and java, this is to allow them to debug and solve any problems that may occur with the functionality of the website. The Admin should therefore be able to find any errors, such as spelling errors or API url errors which will result in errors in the website and it not working. The Admin is not required to understand html or css in depth as it is unlikely any future errors with the website will be due to html as this just receives and displays any dynamic information from the javascript and java code.

Instructions for stakeholders:

Admin:

Instructions for the admin or website maintainer are not much different from what is mentioned in the section above, therefore as long as they have the skills and are able to do some of the tasks above they will require little instruction on how to make sure the website keeps running and working efficiently.

Website Users:

Instructions for the users are displayed throughout the website, such as at the beginning of the new build process. The users should therefore be able to navigate the website and use all of the features on the website through the instructions. This is shown by the usability testing which showed that the website was user friendly and easy to navigate by the majority of the testers.

Limitation fixes

Information from page 89, shown in the same order.

1. The limitations which are caused by the design and size of the database can be fixed in the long run by redesigning the database, ensuring that all the information is split into the most amount of tables. This will allow for additional features such as prices and compatibility checking. For example, by splitting the component table into smaller tables for each component type will allow a unique id to be given to all components and therefore a separate price table will be possible without issue with it pointing to incorrect components, in addition, an extra table could be added for compatibility, grouping those components which are compatible with each other which can then be shown on the website through the Javascript and Java API's.
2. The page redirect issue is a limitation caused by the reset of javascript variables as a new page is shown or the current page is reloaded. This could be improved or solved by setting the value as a new cookie rather than a variable as this does not reset on refresh, however this may cause issues if a user leaves the website then comes back, prompting the website to read the cookie and automatically redirect the user to the incorrect page. This could also be solved by creating the a section on the index page for the dedicated login when a user is logged out, this could be hidden as a default and shown when needed, hiding the other sections so it looks as if it is a new page, however, this may cause issues if the user is logged out on a different page and has to be redirected to the index page before they can log back in.
3. Validation systems can be implemented to help improve this limitation of the website. For example, the code should check the details inputted by a user when signing up to ensure that the user does not already exist, making sure that there are no issues with other users being able to access the incorrect information. In addition, password validation could be introduced such as ensuring there is a capital letter, number and special character to improve the security of the website and the users information. An email validation can be introduced to ensure that the user is inputting something in the format of an email address to reduce the risk of an account being lost as the user will not be able to reset their password if they do not have access to the email address they had signed up with. Another place validation could be added is the name entry for a new build, although a null check is already in place to ensure that a null value is not trying to be saved, a check to ensure the name is not just a space can be introduced to help reduce the chance of errors shown in the robustness testing which would be hard to debug if this check was not introduced.

Future Development

If more time was available for the development of my website, I could implement further improvements and features to the website.

- The ability to share and collaborate on new builds with other users. This can be done through using a link that is given to the other user meaning they have access to the build and can change components, or just view it depending on what the owner sets the link to be.
- Constant automatic saving of the build throughout the process, which means users do not have to select all components in one go and can come back to the build process at any stage. This will also help implement collaboration as the builds will update in real time for both users. This would also prevent a loss of builds when a user may not press the save button or another error was to occur.
- A dedicated information page, or expanded box in the table to show more information about the selected component such as an image, and performance information by users from websites such as pc benchmark. This page could also have user reviews from other websites or the ability for users to add their own reviews of the component.
- The ability to make your build public, allowing people to find it on a dedicated build page and view the components used, as well as add a comment section which allows users to communicate about the build or other things.
- Profiles could be improved, introducing profile images and the ability to follow others, with a feed which will show all the latest published builds of the people you follow, similar to instagram or reddit with the ability to like certain builds.
- A step by step tutorial could be added to the website on the user's first visit, highlighting the important information as well as guiding the user through their first build. This will increase user friendliness and usability as it reduces the risk of a user not being able to understand what to do next or getting stuck throughout the process. This could also be called at any point using a button on the left hand side of the website.

Conclusion

In conclusion, my webpage, having gone through a design, development, and analysis phase, can be shown to be working well, and is robust. This is shown in the multiple stages of testing carried out in both the development and final analysis phases. However, the website could still be improved by redesigning the database and editing the javascript to make the code more efficient and introduce new functionality such as compatibility checking. As well as small improvements, shown in the usability testing, which will help improve the user friendliness of the website. I believe that although the basic functionality of the website is working well and robust, more cycles of both the development and analysis phase would result in a more polished and functional website before it would be open to use by the public.