WEB APP DEVELOPMENT FRONT END

NIALL MULCAHY 20077039

Overview:

This is the front of the application I made as part of the first assignment. The general principle of the application was a website that would allow the user to create a PC “build” that was comprised of “parts” or PC components (CPUs, GPUs etc). This report will describe the functionality of the web application, as well as show the APIs/frameworks used, use cases, persistence etc.

Navigation:

My method of navigation was by means of a nav bar along the top of the page which I adapted from the version used in the first Vue lab on moodle. The nav bar contains four items, each of which both link to the appropriate page/Vue component and display an appropriate icon which were acquired from “Font Awesome”. The “Nav Brand” itself also functions as a link back to the homepage and to the left of the brand, the “PCMasterRace” logo is displayed, this is a logo that is highly synonymous with PC gaming and PC building, and thus, felt appropriate.

Home:

The home page contains a couple of “Semantic UI Segments” which contain a basic header, a welcome message, and the highest rated build on the web application. The welcome message is just a simple message to grab the attention of the user. It also contains a link to the builds page which I will get on to later. At the bottom of the page, the build with the highest number of upvotes is posted. It is posted by means of a Vue client table which will display all of the components used in the build. This will dynamically change whenever another build gets more upvotes.

Builds:

The builds page simply contains another Vue client table that lists all the builds in the database. This table only displays the title, id, cost and the main components (CPU, GPU) and allows the user to sort and search by specific components of the build. This table also provides the user the ability to upvote as well as delete a build. In the title section of the table, the name of the build itself acts as a button which will trigger a “Vuesax popup” containing a more detailed component breakdown of the build, including the build Id, user, CPU, GPU, RAM, storage, and OS.

Create a build:

The create a build page is fairly simple, it allows a user to create a “default build”. When created, the build does not contain any components, these are added later. What is added however, is the name of the build, the user who created it, and the cost of the build. These were chosen as they will not be updated once submitted. The cost has validation via “Vuelidate” and must cost between 1 and 5000. Both the name and the user defaults to “Unnamed” and “Anon” respectively if neither are changed. When the submit button is pressed, the validations are checked, and if the validation criteria are met, the build is submitted to the database and as such, will be displayed on the builds page.

Parts:

It is on the parts page where the builds are updated to contain specific components. On the bottom half of the page, there is a table which contains all of the parts in the database. This table contains the name, id, and type of part, and allows you to filter and search by each of these. However, the top half of the page is where we will add the parts to a build. Once the id of the part you wish to add has been found from the list below, simply copy that into the “Vue form”, then from the “Vue form select” dropdown menu, choose which type of component your selected id refers to. From here, there will be a second selection dropdown menu which will contain the ids of all the builds (the build id is shown as opposed to the name to account for multiple builds potentially having the same name), select the build you wish to add the selected part to. Once the “Add to build” button is pressed the component will be added to the build and will be displayed as such on the build page.

Overall Design:

As you can see, the overall aesthetic of the web application is quite dark. This is something that I myself prefer as it is easier on the eyes and seems to be quite the trend in terms of both online and offline applications.

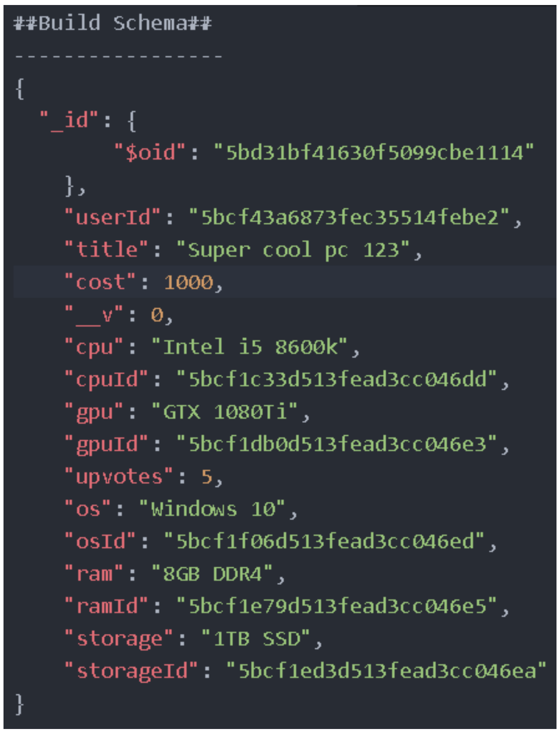
A feature I intended to add was user support such as signing up and signing in. Unfortunately, these had to be scrapped due to time constraints. The plan was for builds that were created by a user that wasn’t signed up/in would default to builds with an anonymous user. But if they were to be signed in, the user would default the correct user id. The user would theoretically have had their own page, which would allow them to view their own builds.

My usage of a popup window for the specific builds was mainly because I felt that each build having their own page was a bit pointless, seeing as each page would store only a simple table. Furthermore, I feel the popup is in and of itself, a nice feature to have.

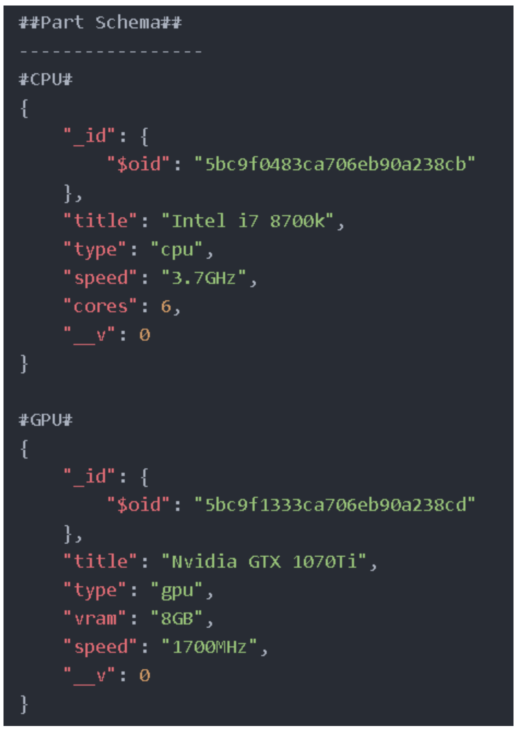
I opted not to allow the user to create/add a part to the database from the front end. This may seem like a strange choice, but I feel like this is a feature that should only be available to the backend. For example, take an application like Spotify, which allows you to create a playlist (in my case a build), in this playlist, you can then add several songs from their vast library (in my case, adding components/parts). Spotify does not (to my knowledge) allow the user to add or create their own songs.

Furthermore, my method of adding parts to the build might also seem like a strange choice, but my reasoning for this action taking place on the parts page as opposed to the builds page was because all of the ids were made readily available on the parts page.

Database Schemas:



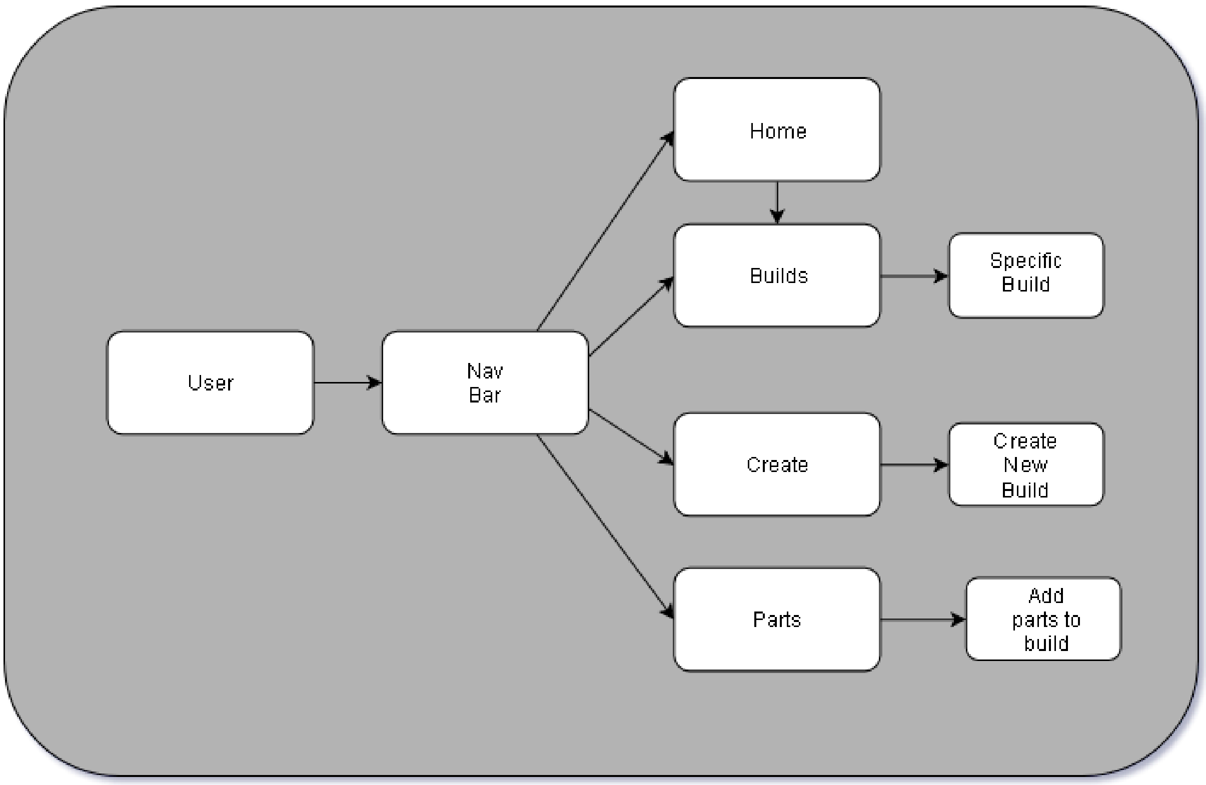
```
##Build Schema##
-----------------
{
  "_id": {
        "$oid": "5bd31bf41630f5099cbe1114"
    },
    "userId": "5bcf43a6873fec35514febe2",
    "title": "Super cool pc 123",
    "cost": 1000,
    "__v": 0,
    "cpu": "Intel i5 8600k",
    "cpuId": "5bcf1c33d513fead3cc046dd",
    "gpu": "GTX 1080Ti",
    "gpuId": "5bcf1db0d513fead3cc046e3",
    "upvotes": 5,
    "os": "Windows 10",
    "osId": "5bcf1f06d513fead3cc046ed",
    "ram": "8GB DDR4",
    "ramId": "5bcf1e79d513fead3cc046e5",
    "storage": "1TB SSD",
    "storageId": "5bcf1ed3d513fead3cc046ea"
}
```



```
##Part Schema##
-----------------
#CPU#
{
    "_id": {
        "$oid": "5bc9f0483ca706eb90a238cb"
    },
    "title": "Intel i7 8700k",
    "type": "cpu",
    "speed": "3.7GHz",
    "cores": 6,
    "__v": 0
}

#GPU#
{
    "_id": {
        "$oid": "5bc9f1333ca706eb90a238cd"
    },
    "title": "Nvidia GTX 1070Ti",
    "type": "gpu",
    "vram": "8GB",
    "speed": "1700MHz",
    "__v": 0
}
```

Use Case:



Access:

Front end deployed on Firebase can be accessed here:

https://pc-builder-web-app.firebaseapp.com/#/

GitHub repository for Front end can be hound here:

https://github.com/niallmul97/Web_Dev_Front_End

Back end deployed on Heroku can be accessed here:

https://pc-builder-web-app.herokuapp.com/

GitHub repository for Back end can be hound here:

https://github.com/niallmul97/PC-Web-App

APIs/Frameworks and References:

Font Awesome:

https://fontawesome.com/

Semantic UI:

https://semantic-ui.com/

Vue:

https://vuejs.org/

Vue tables:

https://bootstrap-vue.js.org/docs/components/table/

Vue form-group:

https://bootstrap-vue.js.org/docs/components/form-group/

Vue form-select:

https://bootstrap-vue.js.org/docs/components/form-select/

Vuesax:

https://lusaxweb.github.io/vuesax/

Vuelidate:

https://monterail.github.io/vuelidate/

Firebase:

https://firebase.google.com/

David Drohan’s labs:

https://ddrohan.github.io/wit-wad-2-2018/