What is the intended purpose of your application?

The purpose of the application is for a user to check if an item they are looking for is in the Fortnite item shop currently and when it last appeared to give them a reference for when it might come back. I've run into situations in which me or my friends want a popular item and check the Fortnite shop constantly. I thought it would be easy and interesting if a user would input items they want and check on only those items. If Heruko were to store data for a long period of time, I would consider using this application frequently.

What data will be stored and delivered by the API?

- Cosmetics added by user
    - Cosmetic name
    - When the cosmetic was last seen in the shop
    - If the item is in the item shop (both at time of addition and when get is called)
    - Cosmetic image
    - Rest of cosmetic data from Fortnite API
    - Addition message

What went right in the development of this project?

Pretty much everything went right in the development of the project. Everything was completed in a timely manner, I figured out how to use the Fortnite API, and I was able to make what I believe to be a good looking and well functioning application.

What went wrong in the development of this project?

The only major thing that went wrong in development in this project was that my laptop force reset and completely wiped almost all data from my laptop on the final stretch of the project. This led to complications with Git, where I wasn't able to use version control on the final day of the project. This also led to not being able to be deployed at the end of the project. I attached a picture of the factory reset prompt below.

What did you learn while developing this project?

I learned how to utilize external API's and how they affect your API/server. I also finally clicked on how the client and server relationship works, and I understand how to handle responses now.

If you were to continue, what would you do to improve your application?

If I were to improve on the application I would add an intelligence factor to the "add cosmetic" bar. This would make it so that if the user's input wasn't perfect, it would give options of correct inputs. This would make the application much more forgiving and user friendly.

If you went above and beyond, how did you do so?

I went above and beyond by applying an external API into my own API. I use this API to validate user inputs, see if the user's input is in the item shop (the crucial functionality), and to get other metadata about the item the user has input (like when the item was last seen). I also used external fonts to style the page, which is handled in htmlResponses.js.

If you used any borrowed code or code fragments, where did you get them from? What do the code fragments do? Where are they in your code?

```
return fetch(`${COSMETIC_URL}/?name=${body.name}`).then((apiResponse) => fetch
(SHOP_URL).then((shopResponse) => {
  if (apiResponse.status !== 200) {
    responseJSON.message = 'Cosmetic not found';
    responseJSON.id = 'missingParams';
    return respondJSON(request, response, 400, responseJSON);
  }
```

- https://www.geeksforgeeks.org/how-to-use-the-javascript-fetch-api-to-get-data/
- Used as reference for fetch operations (refresher).
- 62 jsonResponse.js

```
const day = date[2].startsWith('0') ? parseInt(date[2][1]) : parseInt
(date[2]);
```

- Not a particular reference, but I used W3 Schools as a reference for operator documentation.
- 70 client.html

```
const lsString = jsObj.data.shopHistory[jsObj.data.shopHistory.length - 1].
substring(0, 10);
```

- https://www.sitepoint.com/string-substrings-javascript/
- Learned about substring() when looking for how to only get a certain number of characters from a string and used this for documentation.
- 77 jsonResponses.js

Endpoint Documentation

URL: /getIndex
Supported Methods: GET
Query Params: request , response
Descriptions: Sends Index to server
Return Types: html

URL: /getCSS
Supported Methods: GET
Query Params: request , response
Descriptions: Sends CSS to server
Return Types: css

URL: /getFont
Supported Methods: GET
Query Params: request , response
Descriptions: Sends custom font to server
Return Types: otf

URL: /getCosmetics
Supported Methods: GET
Query Params: request , response
Descriptions: Sends input cosmetics to the client
Return Types: JSON

URL: /notFound
Supported Methods: request , response
Query Params: request , response
Descriptions: 404 Error
Return Types: JSON

URL: /notFoundMeta
Supported Methods: request , response
Query Params: request , response
Descriptions: 404 Error using head request
Return Types: JSON

URL: /addCosmetics
Supported Methods: request , response
Query Params: request , response
Descriptions: Adds the user’s input to the server
Return Types: None (writes JSON to server)