Project 1 Documentation

John Pionzio

The purpose of my application is to allow users to create a collection of tanks on a web site which consists of 3 individual pages. The first page contains a form where users fill out information about a tank (its name, nation of origin, year designed, number produced, and optionally a description and image URL). The second page will display tanks currently in the collection based on data retrieved from the API, which are sorted based on the sorting method currently selected by the user. The third page displays documentation for the application.

The data stored and delivered by the API is a JSON object containing each tank entry the user adds to the database, with each tank having their name, nation of origin, year designed, number produced, a description, and image URL stored in this object. If the user does not provide a description or image URL, a default description and image will be stored; note that images themselves are not stored by the API, just URL links to images on the web. All stored data is delivered to the client by request on the second page.

During the development of this project, everything regarding development went right. There were very few problems regarding code during development, and those that did occur were fixed promptly and with little difficulty. Adding the sorting method was expected to cause trouble, but ultimately went smoothly, as did splitting the server functionality between pages and adding new types of media to be handled.

The closest the project's development came to something going wrong was in two instances, one being a slight hiccup when implementing the sorting method, as I had initially not stored the data on the client in the correct type of object. The second instance was creating working directory links to images, as I initially thought that doing so would be more similar to how we linked mp4's and mp3's during exercises, but some experimentation on my end revealed that the process was far simpler.

While developing this project, I learned how to properly store the elements of a JSON object in an array, and I also learned how to sort and array of JSON objects via any field shared by the JSON objects within the array. I learned how to manually add links to a Bulma navbar-burger, something I had done automatically in the past; it sounds weird to say I knew how to do it automatically and not manually, but life is strange like that. I also learned how to return an image and a PDF from a server directory.

If I were to continue working on my application, I would further improve the page's styling and color scheme, as while I do believe the current state is satisfactory, I feel as though it could be better. I would also like to add a more responsive message system when sending and retrieving data, as the application currently relies on basic text messages.

This project went above and beyond by utilizing Bulma to make well designed webpages, splitting the functionality of the application through several webpages, display documentation on a separate page, and allowing the user to sort displayed information based on specific criteria (sort tanks by name, nation, year, and number produced).

This project was started using the completed code I wrote for HTTP API Assignment 2, which has been modified to function as I require. Most code in my project's files bears resemblance to the code from that assignment, but nearly all code has been modified at least slightly by this point, leaving no remaining borrowed code fragments (to my knowledge).

Endpoint Documentation:

URL: addTank

Supported Method(s): POST

Query Parameter(s): body (body.name, body.nation, body.year, body.produced, and *optionally* body.description and body.image)

Description: Posts data regarding a new tank entry to the server to be stored by the server-side 'tanks' object; if no entry exists for the tank entry being posted, a new entry is created and a status code 201 is returned, and if one does exist, the existing entry is updated with the new data being posted and a status code 204 is returned. If insufficient data is provided by the 'body' (no name, nation, year produced, or number produced), a status code 400 response is returned.

Return Type(s): application/json

URL: getTanks

Supported Method(s): GET, HEAD

Query Parameter(s): N/A

Description: Returns the current value of the server-side 'tanks' object which stores all tank data; the returned object contains multiple JSON objects for each tank currently stored, with each entry having its own name, nation, year produced, number produced, description, and image.

Return Type(s): application/json

URL: notReal

Supported Methods: GET, HEAD

Query Parameters: N/A

Description: Returns a status code 404 to the client whenever an invalid URL is used.

Return Type(s): application/json