Creative Project 3 Specification

Due: Saturday, May 20th 11:30PM PST

Overview

For your third Creative Project, you will explore different API's available as public web services and use the Fetch API to asynchronously request and process response data on your own website. Learning how to find and use public APIs is an extremely important skill in modern web development (and working with public datasets is one of the most fun parts of being a web developer!).

There are a ton of APIs on the web today - you can find ones for dog breeds, government datasets, dictionary web services, Spotify data, etc. You can find more APIs here. With that, even when the data they respond with may be very interesting, some APIs can be frustrating to work with due to poor documentation (what? documentation is actually useful?) or outdated response formats (XML, HTML, etc.). The APIs for CP3 should all return data in JSON or plaintext format since we cover that more in this class and these formats tend to be easier to work with.

Specifically, you will choose a public API (a list of some recommended ones are below) to request and process interesting data with and integrate into your own webpage:

- Country Data API
- Dog API
- jService Trivia Questions
- Movie DB API (requires API key)
- News Headlines API (requires API key)
- Webster-Merriam Dictionary (requires API Key)
- Spotify API (requires OAuth)

For CP3, you are **not allowed use the following APIs (unless you add them with a second API):**

- NASA Astronomy Photo of the Day (other NASA APIs are fine!)
- Weather Data API endpoints from CS11 Winter 2021 (other endpoints are fine)

We want to make sure you are exploring new APIs and not copy/pasting code that you have seen from lecture or a previous course.

Note: You must visibly cite the API you are using in your page (e.g. in a page footer).

Some students do have specific APIs they may be wanting to use in their websites - you are allowed to use another public API, however you should ask on Discord to use the API(s) to ensure they meet the requirements and note that **the TAs will only be expected to support projects that use the above APIs**. EI will be available to answer quick questions about other APIs.

Note about API Keys

- Some of these APIs require API keys - an API key is helpful to ensure the service isn't overloaded with requests from clients, and most public APIs you work with as a web developer will require some sort of key or authentication. Each API has sufficient documentation to get access to your key once you register for a developer account (required for a key). That said, feel free to ask on Discord/OH if you have any questions about getting/using API keys! Remember that you should **not** be opting into any paid subscription plan.

Ideas for CP3

As long as you meet the requirements outlined below, you have freedom in what kind of website you create. Here are some ideas for CP3:

- Use the Country Data API to analyze or present facts about different countries, currencies, languages, etc.
- Use the Dictionary or Thesaurus API to help implement a text analysis tool or a type of word game
- Implement a personalized news tool based on user input and the News Headlines API
- Analyze data and present statistics found in an API dataset by exploring the different keys in the response data
- Take two APIs and use them together on the same page in a creative way!
    - Note: This is recommended practice for HW3! You can also earn an additional point on CP3 (up to 9 points) for incorporating 2 APIs.
- Ask us if you'd like help coming up with more ideas for your project!

We encourage you to explore the new material covered in class, as well as related (but optional) content we may link to along the way, as long as you follow the CS 132 Code Quality guidelines. This is your chance to:

1. Continue to build a website that you can link to on your resume or code portfolio (CPs can be public, most HWs cannot be).
2. Ask EI and/or TAs about features you want to learn how to implement (we can provide more support for CPs than HWs) and ask for feedback/ideas on Discord.
3. Apply what you're learning in CS 132 to projects you are personally interested in and may use outside of just a CS 132 assignment.

4. Get feedback on code quality when learning new technologies in CS 132 to implement for the corresponding HW, which will be worth more points.

You may choose to do a new website for each CP, or build on a single project, as long as you meet the CP requirements. **You may not copy/paste code from lecture/Canvas. If you refer to example code, you are expected to briefly comment what was referred to in your source code and possibly, page footer** (we are more lenient in such cases, and it helps to know what material is useful as a resource)**.**

Development Strategy for Using Fetch with an API

Students often jump into code to fetch data from APIs without having enough understanding of how Promises and the fetch pipelines works, and/or how to properly request and process data from a particular API. In Week 6, EI reviewed strategies for getting started with a website that fetches data from the web which can help students distinguish between bugs in their UI, bugs in their fetch code, and errors in fetch requests caused by their code *or* the API itself having an issue (which does happen!). Here is a general strategy we recommend for this assignment:

1. Design your page (either with a front-end or wireframe) to plan for your implementation, imagining if you had the data you wanted from the API. **Do this before you write any Fetch call(s)**.
2. Find out how to build a URL to fetch from your chosen API (most APIs will have examples in their documentation).
    - What is the base URL? For example, the base URL of the NASA Astronomy Photo of the Day (APOD) API used in lecture is https://api.nasa.gov/planetary/apod.
    - What are any required parameters (often called query parameters) you need to add to the URL? For example, the APOD API requires a query parameter of "api_key" which accepts a value of an API key you can register for on the API's home page. Without registering for an API key, the APOD API conveniently lets you provide a value "DEMO_KEY" for a limited number of daily requests. Using this required parameter, you can make a request to https://api.nasa.gov/planetary/apod?api_key=DEMO_KEY to get JSON data for the photo of the current day (try it!).
    - Are there any optional parameters you can choose from to request specific information from the API? In the APOD API, you can also use an optional parameter called "date" documented on their API page to specify the date. For example, you can make a request to https://api.nasa.gov/planetary/apod?api_key=DEMO_KEY&date=2000-01-30 to get the Astronomy photo of the day for January 30th, 2000. Some other APIs let you choose the date when filtering out response data, but they may use a different parameter name than date. All of the listed APIs will specify what possible paths and parameters are available for you to use.

3. Get an example JSON representation from the API by visiting the URL with query parameters. Copy/paste the JSON into your browser console and expand the result to understand the hierarchy of an example JSON response for the API. Most APIs will have examples that you can copy/paste in a new page and output the JSON, similar to what El did with the menu JSON in in Lectures 11/12. Make sure you use your API key as a query parameter if using an API that requires one. For testing, you may find it helpful to store the JSON as a temporary global variable so you can focus on working with the JSON data parsing and DOM manipulation before working with any Fetch calls (similar to the approach in lecture). **Do not have any JSON stored as global or module-global variables in your final submission**.
4. Go back to the API documentation for any clarifications on what any field names in the JSON mean. Some will be intuitive, some may need a bit of clarification in the documentation.
5. **Note: You can get up to this step without using any Fetch calls in your assignment.** Once you know how to access the field names you want for a response (this exercise will be helpful to review), finish your JS response function to use the JSON and update your page with the data you want to use to meet the external requirements.
6. To test possible errors (you will need to display a descriptive error message on the page if an API returns an error caught in the catch statement), you can temporarily replace your response function with the error-handling function in the fetch. **The Network tab should be your best friend in this assignment!**
7. Can you think of any other ways to use the API (e.g. other query parameters or endpoints) or incorporate another API? You can go for the extra API challenge!

External Requirements

Overall External Requirements

- Your project must include the following three files at a minimum:
    - index.html - main page of your website
    - styles.css - file to style your .html file
    - *.js - a JS file (named appropriately of your choosing) containing your JavaScript code. You may include more than one.
- You may also include the provided `helpers.js` with `checkStatus` to factor out alias/utility functions.
- Similar to other assignments, a**ll file names and links in your project must be relative and lower-cased without spaces** (e.g. img/puppy.jpg but not img/puppy.JPG or img/Puppy.jpg). This is enforced to avoid broken links commonly occurring in CP/HW submissions due to case-insensitivity of file names on Windows machines.

- Your page should include appropriate content and copyrights and citations. If we find plagiarism in CPs or inappropriate content, **you will be ineligible for any points on the CP**. Ask if you're unsure if your work is appropriate.

HTML/CSS Requirements

- Your website must contain at least one HTML page linked to your JS file to respond to page events as we've discussed in class.
- You must have a `styles.css` file linked from your `index.html`, but there are no additional external requirements (your CSS will still be graded on Internal Requirements below which come from CP1/CP2)

JavaScript External Requirements

Your website must somehow dynamically load information from the web API(s) you've chosen and present information from that response on the page. This requires that you must:

- Respond to some event (whether it's the window load event or any UI event) to determine when to fetch the data
- Dynamically integrate the data returned from the API into the page by manipulating the DOM elements in some non-trivial way using `element.appendChild`, `element.removeChild`, or `element.replaceChild`
- Use the `checkStatus` function from Lecture 12 to throw an Error if the `fetch` response status is not ok before processing the data (refer to this example). This is a helper function we are allowing (encouraging) you to use (with JSDoc) in your AJAX programs, but the rest of your functions must be your own. As mentioned above, you can add this to the `helpers.js` function we've shared for other JS assignments.
- Handle any errors caused in the `fetch` request/response process by displaying a *helpful* message to the user on the page (i.e. **without** using `alert`, `console.log`, or `console.error`). To do so, you should define a function (e.g. `handleError`) to implement the error-message-displaying and pass that function in the `fetch` call chain's `catch` statement. Be mindful of what users see (e.g. don't dump error details on the page, give a client some useful note).

Internal Requirements

Any *internal* (code style) requirements from previous assignments are also applicable in CP3. We have repeated some of the most commonly-missed ones, as well as new requirements for working with a web API using JS.

Overall

- Your HTML, CSS, and JavaScript should demonstrate good code quality as demonstrated in class and detailed in the CS 132 Code Quality Guidelines. Part of your grade will come from using consistent indentation, proper naming conventions, curly brace locations, etc. Lecture examples demonstrate JS naming/whitespace conventions in this course, but there are more examples in the guide. Don't forget about the Prettifier VSCode extension mentioned in class in Week 2 to help with formatting (though it is not guaranteed to get things perfect, especially if you have syntax errors, such as mismatched HTML tags or curly braces).

HTML and CSS

- Continue to follow the standards for your HTML/CSS, including consistent whitespace/indentation, proper use of classes/ids, separation of languages, avoiding redundancy in CSS, etc.
- Links to your .html, .css and .js files should be **relative links**, not absolute.
- HTML and CSS files must be well-formed and pass W3C validation. This also includes any generated HTML as a result of DOM manipulation in JS (e.g. remember to include descriptive `alt` attributes when creating `img` elements dynamically).

JavaScript

- When adding interactivity to your website, you should handle any events (like a mouse event, keyboard event, timer, etc.) by responding to them using a JavaScript function(s) in your `.js` file. You should not have any JavaScript code in your HTML and you should not have any HTML tags as strings in your JavaScript code (e.g. `el.innerHTML = "<p>Foo</p>";`).
- Your *.js file should be linked to your `index.html` or other `.html` files using `<script defer src="...">` in the HTML `<head>`.
- Minimize styling in JS (e.g. changing the `.style` property of elements) - prefer adding/removing classes to DOM elements instead, and style the classes in your CSS. Remember that there is an exception when dynamically generating values for styles or positions that are not reasonably factored out in CSS (see Week 5 material for some discussion/notes on this).

- Any `.js` code you use must use the module-global pattern (recall the module-global template) and `"use strict";` within.
- Decompose your JS by writing smaller, more generic functions that complete one task rather than a few larger "do-everything" functions. HW2 and the Skittles case study are very useful resources to refer to for a clean interactive program breakdown.
- Use an `init` function to factor out anything specific to initialize page set up (eg event listeners); remember to keep anonymous functions <= 3 lines. Do not reassign any event listeners unnecessarily.
- Localize your variables as much as possible. You should not use any global variables (outside the module pattern) - see Code Quality Guide for reference. Don't forget to utilize the `this` keyword and optional event arguments for callback functions to simplify logic in some cases (but make sure not to catch an event parameter if it doesn't help simplify/is not used).
- Only use module-global variables whenever absolutely necessary. Program constants are also good to remember to use (refer to the Skittles Case Study from Week 4/5)
- Use const with UPPER_CASED naming conventions for program constants (e.g. BASE_URL for APIs, a file path to your images if you are working with many images in your JS, etc.).
- Do not leave any debugging code/commented-out code in your submissions (including `console.log/alert`)
- Your JS code must pass ESLint with no errors (note that added some configurations in this version that will be useful for students for common errors mentioned in the spec/Code Quality Guide, such as requiring camelCase/etc.)
- **New:** Any AJAX requests in your JS code must use the Fetch API. You can find an example skeleton for reference here, and you will see more examples with different APIs in lecture. Do not use any method of making AJAX requests not covered in class (e.g. XMLHTTPRequest or jQuery). **jQuery is strictly unallowed in CP3.**
- **New**: Do not make unnecessary requests to the API. That is, there should be no code in your JS that requests from an API and **never** does anything with the response. Furthermore, be mindful about how frequently you are making requests to a web service. Some APIs will have request limits (e.g. 1000/day), so you'll want to make sure you aren't making redundant requests in loops/timers (it is normal for many pages to make one or few more requests to get the data needed).
- If you want to explore other JS features beyond what we've taught in class, you must cite what resources you used to learn them in order to be eligible for credit, and may not use alternative methods for things that are taught in CS 132 (e.g. XMLHTTPRequest instead of fetch). We strongly encourage students to ask the staff for resources instead of finding online tutorials on their own (some are better than others).

Documentation

Place a comment header in each file with your name, section, a brief description of the assignment, and the corresponding file/program. Examples:

HTML File:

```
<!--
    Author: Lorem Hovik
    CS 132 Spring 2023
    Date: April 1st, 2023

    This is the index.html page for my portfolio of web development work.
    It includes links to side projects I have done during CS 132,
    including an AboutMe page, a blog template, and a cryptogram generator.
-->
```

CSS File:

```
/*
   Author: Lorem Hovik
   CS 132 Spring 2023
   Date: April 1st, 2023

   This is the styles.css page for my portfolio of web development work.
   It is used by all pages in my portfolio to give the site a consistent
   look and feel.
*/
```

Use JSDoc to document *all* of your JS functions with @param, @returns as discussed in the Code Quality Guide.

```
/**
 * Brief description of the function (including requirements, important
 * exceptional cases, and avoiding implementation details).
 * @param {datatype} parameterName1 - parameter description
 * @param {datatype} parameterName2 - parameter description
 * @returns {datatype} - Description of the return value (if applicable)
 */
function functionName(parameterName1, parameterName2) {
  ...
}
```

Your JS file should also have your student information and an overview of the program in a file comment.

Grading

Our goal is to give you feedback, particularly on the internal requirements and style and documentation, so you can incorporate this feedback in your homework assignments which will be worth more towards your final grade.

This CP will be out of 9 points and will be distributed as:

- External Correctness (4 pts)
- Internal Correctness (4 pts)
- Documentation (1 pt)

You can earn one extra point (max of 9 points) if you implement a second fetch request that meets the same requirements above and is useful (e.g. doesn't implement something you can do without it). This could be two fetch calls to the same API (with different parameters or endpoints) or to two different APIs.