HyperionDev

Promises, Async, and Await

Model-Answer Approach

Visit our website

Auto-graded task 1

Approach taken

The `fetchPokemon` function uses the Fetch API to retrieve data about a specified Pokémon. The function constructs the URL dynamically based on the Pokémon's name, performs a fetch request, and processes the response. To ensure a successful response, validation is performed, and the data is parsed as JSON. Key attributes, such as the Pokémon's name, weight, and abilities are extracted and stored in an array. Error handling is implemented to capture and log any issues during the fetch operation or data processing.

Why it is a fitting solution

This approach is effective because it combines asynchronous operations with clear error handling. Using fetch with promises allows for efficient network requests and error management. The function's design ensures that only successful responses are processed and provides meaningful error messages if something goes wrong. This makes the code robust and adaptable to varying conditions.

Potential common pitfalls

Frequent requests to the Pokémon API might exceed rate limits, causing errors or blocking of requests. If the API's response format changes, the code may fail to parse the data correctly. Finally, if an invalid or non-existent Pokémon name is used, the function will handle the error, but the output might be misleading or empty.

Auto-graded task 2

Approach taken

The `catApi` function fetches a random cat image URL from the Cat API using the Fetch API. It sends a request to the specified endpoint, checks if the response is successful, and then extracts the URL from the response. The function includes error handling to catch any issues with the fetch operation or data processing, returning a fallback message if an error occurs.

Why it is a fitting solution

This method is suitable, as it uses asynchronous functions to manage network requests efficiently and handle errors gracefully. By checking the response status and parsing the JSON data, it ensures only valid URLs are processed. This approach is ideal for APIs that might return data in various formats or redirect responses, making it resilient to common API response issues.

Potential common pitfalls

The endpoint might change or require authentication, potentially causing fetch errors. If the API response does not match the expected format, it might lead to issues in extracting the image URL. Finally, the API might enforce CORS (cross-origin resource sharing) policies that could block requests from certain domains or require specific headers.