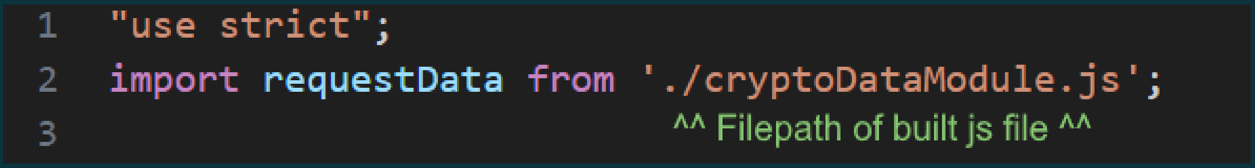
cryptoDataModule.js

The CryptoDataModule is a module that provides functionality for fetching and caching data from an API related to cryptocurrency. It consists of two main classes: ApiClient and CacheProvider.

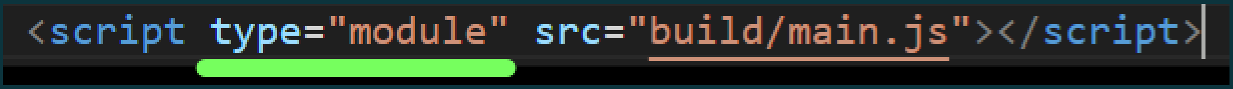
Installation and Usage

To install the module copy cryptoDataModule.js into your project folder or any subfolder. Then import the module to your main js/ts file as shown below:



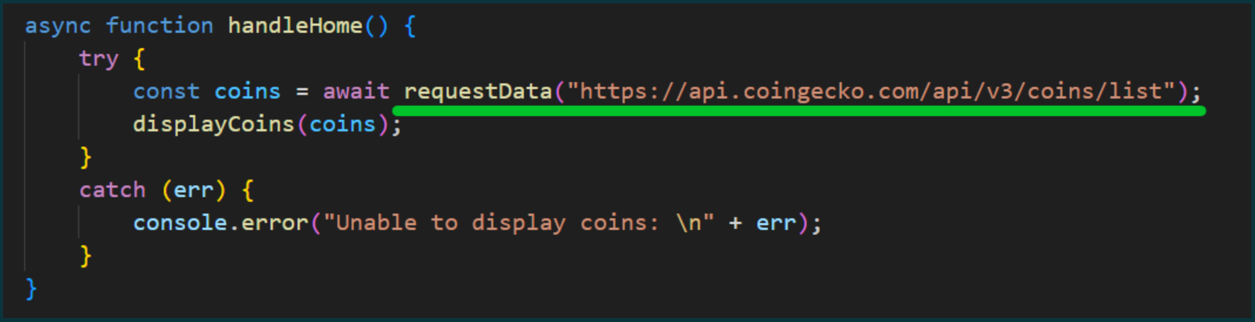
```
"use strict";
import requestData from './cryptoDataModule.js';
```

Now make the necessary adjustment to your main script tag in the HTML code:



```
<script type="module" src="build/main.js"></script>
```

To use the module, you need to invoke the requestData function provided by the module. The requestData function takes two parameters: **url (string)** and **ignoreCache (boolean)** (optional). Here's an example of how to use the module:



```
async function handleHome() {
    try {
        const coins = await requestData("https://api.coingecko.com/api/v3/coins/list");
        displayCoins(coins);
    }
    catch (err) {
        console.error("Unable to display coins: \n" + err);
    }
}
```

In the example above, the requestData function is called with the API URL as the first argument. It fetches data from the API and returns the response as a JSON object. If the request is successful, the fetched data is logged to the console. If an error occurs during the API request or fetching the cached data, an error message is logged to the console.

ApiClient Class

The ApiClient class is responsible for making HTTP requests to the API and fetching the data. It contains a single method called fetchData which is an asynchronous function. The fetchData method takes a URL as a parameter and returns a Promise that resolves to a JSON object representing the fetched data. It internally uses the fetch function to perform the HTTP request.

CacheProvider Class

The CacheProvider class handles caching of the fetched data. It allows storing and retrieving data from the browser's local storage. It contains three methods: get, set, and isValid.

- get(key: string): any: This method retrieves data from the cache based on the provided key. If the data exists in the cache, it returns an object containing the cached data and the timestamp indicating when it was stored. If the data is not found in the cache, it returns null.
- set(key: string, value: any): void: This method stores data in the cache using the provided key and value. The data is stored as an object containing the timestamp of storing and the actual content.
- isValid(data: any): boolean: This method validates the data retrieved from the cache. It checks the timestamp of the data against a timeout value (20,000 milliseconds by default). If the timestamp is within the timeout limit, the data is considered valid, and the method returns true. Otherwise, it returns false.

The CacheProvider class is used by the requestData function to check the validity of cached data before making an API request.

requestData Function

The requestData function is the main function exposed by the module. It encapsulates the logic for fetching data from the API while utilizing caching. It takes the API URL as the first argument and an optional ignoreCache flag. Here's how it works:

1. It creates instances of the ApiClient and CacheProvider classes.
2. It calls the get method of the CacheProvider to retrieve any cached data for the given URL.
3. If cached data exists, and it's valid (according to the isValid method of CacheProvider) and ignoreCache flag is not set to true, the function returns the cached data content.
4. If there is no cached data or it's not valid, the function proceeds to make an API request using the fetchData method of the ApiClient class.
5. If the API request is successful and the response does not contain an error, the fetched data is stored in the cache using the set method of the CacheProvider.
6. Finally, the function returns either the fetched API data or the cached data content, depending on the result of the API request.

By utilizing caching, the requestData function reduces the number of API requests by serving the data from the cache if it's valid and not explicitly ignored.

Future Plans

In the near future I plan to add two top priority features:

1. **Validation:** Expand on validation to include more meaningful data validation, not just timestamp validation.
2. **Garbage collection:** Add a garbage collector class. Storage space in localStorage is limited to just a few MBytes, therefore as time goes on there is a danger of overfilling the localStorage. The garbage collector will handle storage allocation and removal of unused items.