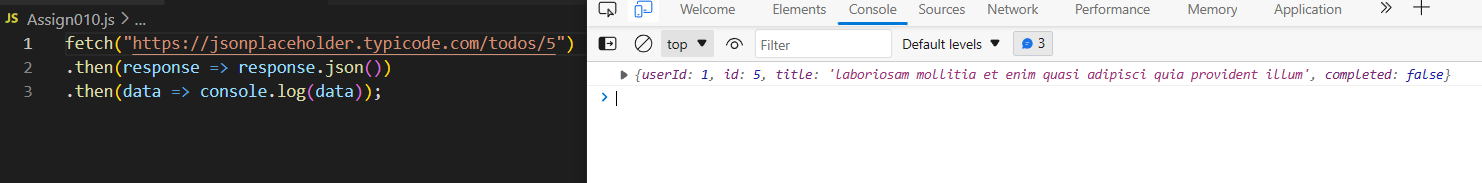
**Assignment 010**

**Fetch ()**: -

**1.Type: -** GET method

**Description: -** The GET method is used to retrieve data from the server. This is a read-only method, so it has no risk of mutating or corrupting the data.

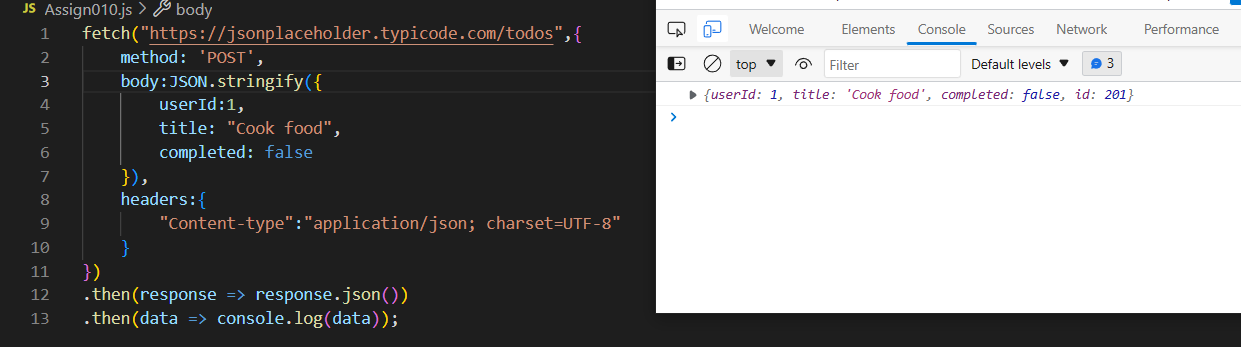
**Code Snippet & output: -**

****

**2.Type: -** POST method

**Description: -** The POST method sends data to the server and creates a new resource. The resource it creates is subordinate to some other parent resource. When a new resource is POSTed to the parent, the API service will automatically associate the new resource by assigning it an ID (new resource URI). In short, this method is used to create a new data entry.

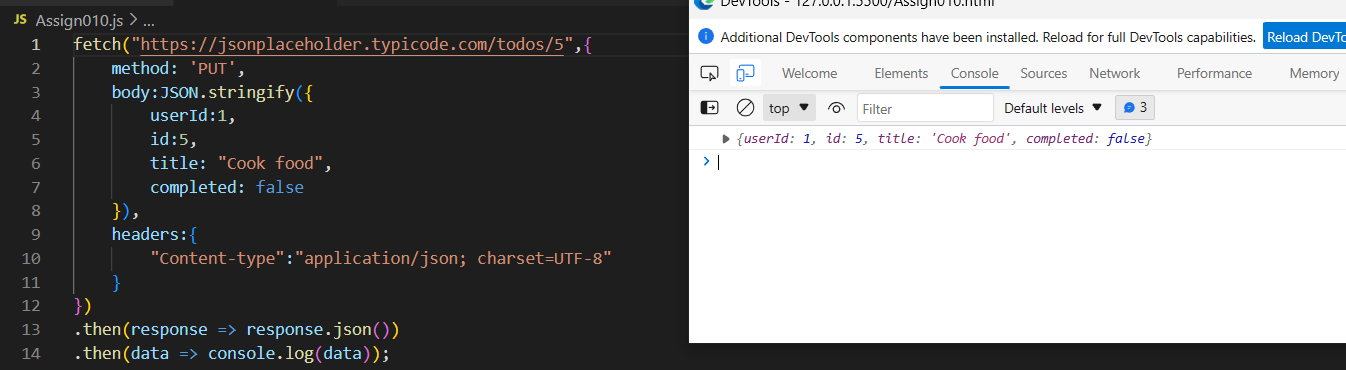
**Code Snippet & output: -**

****

**3.Type: -** PUT method

**Description: -** The PUT method is most often used to update an existing resource. If you want to update a specific resource (which comes with a specific URI), you can call the PUT method to that resource URI with the request body containing the complete new version of the resource you are trying to update.

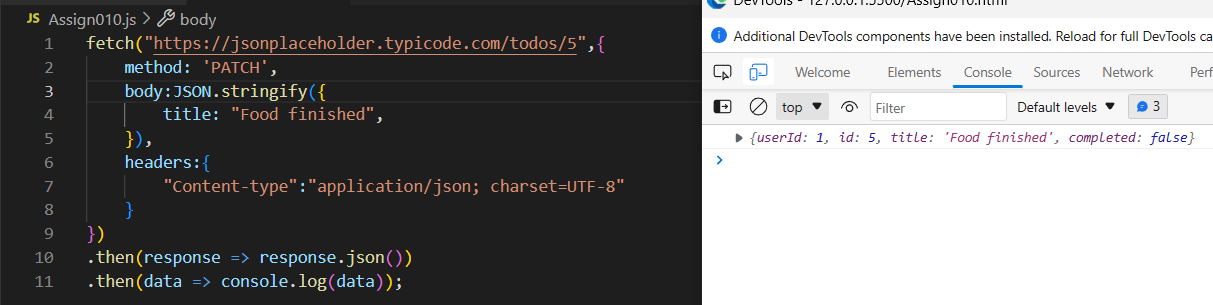
**Code Snippet & output: -**

****

**4.Type: -** PATCH method

**Description: -** The PATCH method is very similar to the PUT method because it also modifies an existing resource. The difference is that for the PUT method, the request body contains the complete new version, whereas for the PATCH method, the request body only needs to contain the specific changes to the resource, specifically a set of instructions describing how that resource should be changed, and the API service will create a new version according to that instruction.

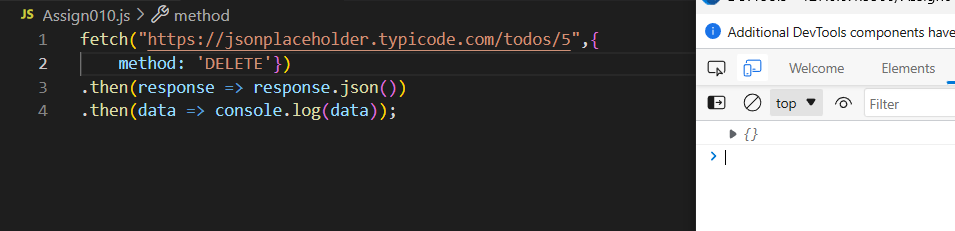
**Code Snippet & output: -**

****

**5.Type: -** DELETE method

**Description: -** The DELETE method is used to delete a resource specified by its URI.

**Code Snippet & output: -**

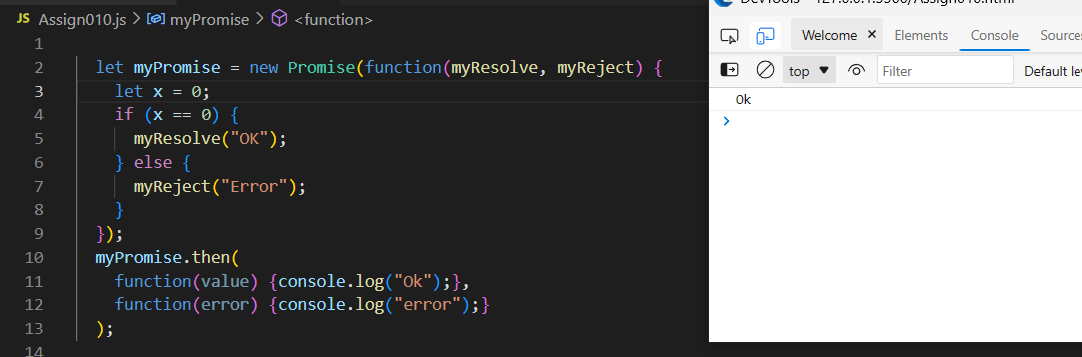
****

**Promise(): -**

**1.Type: -** resolve()

**Description: -** The promise.resolve() method in JS returns a Promise object that is resolved with a given value. If the value is a promise, then the promise is returned. If the value has a “then” attached to the promise, then the returned promise will follow that “then” to till the final state. The promise fulfilled with its value will be returned.

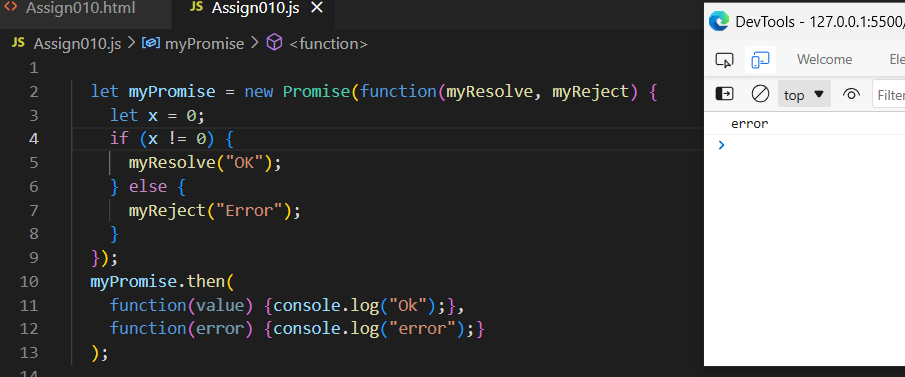
**Code Snippet: -**



**2.Type: -** reject()

**Description: - reject()** method is used to return a rejected Promise object with a given reason for rejection. It is used for debugging purposes and selective error catching. The catch() method can be used for logging the output of the reject () method to the console that is catch() method acts as a career which carries the rejected message from Promise.reject() method and displays that in user’s console.

**Code Snippet: -**



**Status code: -**

**1.Status number: -**200 (OK)

**Description: -**The request succeeded. The result meaning of "success" depends on the HTTP method:

GET: The resource has been fetched and transmitted in the message body. HEAD: The representation headers are included in the response without any message body. PUT or POST: The resource describing the result of the action is transmitted in the message body. TRACE: The message body contains the request message as received by the server.

**2.Status number: -** 201(Created)

**Description: -** The request succeeded, and a new resource was created as a result. This is typically the response sent after POST requests, or some PUT requests.

**3.Status number: -** 202(Accepted)

**Description: -**The request has been received but not yet acted upon. It is noncommittal, since there is no way in HTTP to later send an asynchronous response indicating the outcome of the request. It is intended for cases where another process or server handles the request, or for batch processing.

**4.Status number: -** 204(No content)

**Description: -**There is no content to send for this request, but the headers may be useful. The user agent may update its cached headers for this resource with the new ones.

**5.Status number: -** 301(Moved Permanently)

**Description: -** It indicates that the URL has been moved permanently. This can lead clients (such as a browser) to 'cache' the new URL and make them automatically call the new URL instead of trying to look up the previous URL. Since you have said it was permanent.

**6.Status number: -** 302(Found)

**Description: -** It is the same as the 301, indicating that the client should redirect. Instead, you're declaring that it is a temporary redirect and that they should come back to looking up this URL at some point.

**7.Status number: -** 400(Bad request)

**Description: -** Send this when there is an error with the request such as invalid parameters, missing data, etc. It is also a generic and default code for errors that do not have any other appropriate response code.

**8.Status number: -** 401(unauthorized)

**Description: -** The code 401 has to do with authentication. It is often incorrectly interchangeably used with 403. The appropriate time to send a 401 response is to indicate that the client did not send any authentication credentials.

**9.Status number: -** 403(forbidden)

**Description: -** This error code has to do with authorization. It means they have credentials but they are wrong or insufficient. There are two common cases when a user fails authorization.

1. They have sent credentials but the credentials are invalid (bad token, wrong password).
2. They have valid credentials but do not have the privilege and authorization
   * A User trying to access Admin stuff
   * Changing the password of an account that isn't theirs

**10.Status number: -** 404(Not found)

**Description: -** The most well-known that doesn't need much explanation. For the server-side, the use case is when the client sends an unknown entity ID (which can either be in the URL or request body). For example:

* User ID not found
* Add user to a project but the project ID is not found

**11.Status number: -** 422(Unprocessable entity)

**Description: -** This is often used interchangeably with 400 as a generic error code as Unprocessable entity can mean both I can't read it and I can read but I can't understand it. Both these codes are generally acceptable as a generic error codes. However, I prefer the use case where this code is for formatting errors, such as sending an XML request body when you expect JSON and using 400 instead for generic errors.

**12.Status number: -** 500(Internal server error)

**Description: -** This is the code that the error handler sends when there is an unintentional error in your code. You can be very sure it is to do with a server-side coding error or config when throwing this error. The only place where you should return this is in the error handler of your back-end. Never explicitly throw this status code.