

JavaScript Promise:

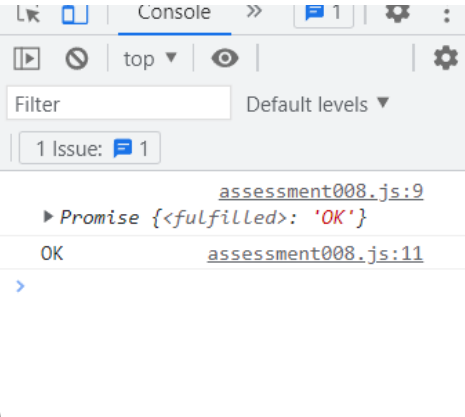
The Promise object represents the eventual completion or failure of an asynchronous operation and its resulting value. Promises are forward direction only; You can only resolve them once.

A Promise is in one of these states:

- pending: initial state, neither fulfilled nor rejected.
- fulfilled: meaning that the operation was completed successfully.
- rejected: meaning that the operation failed.

Code(fulfilled):

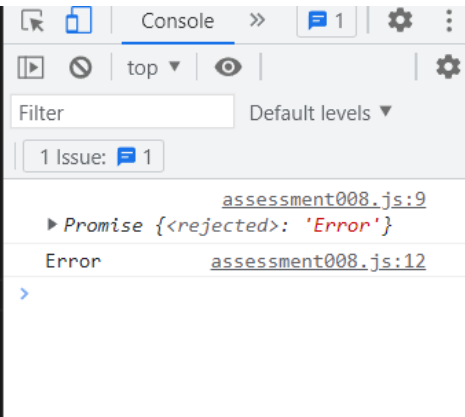
```
let myPromise = new Promise(function(value, error) {
  let x = 0;
  if (x == 0) {
    value("OK");
  } else {
    error("Error");
  }
});
console.log(myPromise)
myPromise.then(
  function(value) {console.log(value)}
).catch(function(error) {console.log(error)})
```



The screenshot shows a web browser's developer console. The top part shows the code being executed. The bottom part shows the console output: 'Promise {<fulfilled>: 'OK'}' and 'OK'. The console also shows the file path 'assessment008.js:9' and 'assessment008.js:11'.

Code(rejected):

```
let myPromise = new Promise(function(value, error) {
  let x = 1;
  if (x == 0) {
    value("OK");
  } else {
    error("Error");
  }
});
console.log(myPromise)
myPromise.then(
  function(value) {console.log(value)}
).catch(function(error) {console.log(error)})
```



The screenshot shows a web browser's developer console. The top part shows the code being executed. The bottom part shows the console output: 'Promise {<rejected>: 'Error'}' and 'Error'. The console also shows the file path 'assessment008.js:9' and 'assessment008.js:12'.

Fetch:

The Fetch API provides a JavaScript interface for accessing and manipulating parts of the protocol, such as requests and responses. It also provides a global fetch() method that provides an easy, logical way to fetch resources asynchronously across the network.

Code:

[illegible]

HTTP methods:

1. GET
2. POST
3. PUT
4. DELETE
5. PATCH

GET method:

The GET method requests a representation of the specified resource. Requests using GET should only retrieve data.

Code:

```
fetch('https://jsonplaceholder.typicode.com/posts/1')
.then(response=> response.json().then(result=>{console.log(result)}));
```

Default levels ▾ 1 Issue: 1

assessment008.js:2

```
{userId: 1, id: 1, title: 'sunt aut facere repellat p
rovident occaecati excepturi optio reprehenderit', bo
dy: 'quia et suscipit\nuscipit recusandae consequunt
ur ...strum rerum est autem sunt rem eveniet architect
o'}
```

POST method:

The POST method submits an entity to the specified resource, often causing a change in state or side effects on the server.

Code:

```
fetch("https://jsonplaceholder.typicode.com/todos", {
  method: "POST",
  headers: {"Content-type": "application/json; charset=UTF-8"},
  body: JSON.stringify({
    userId: 1,
    title: "Fix my bugs",
    completed: false
  })
}).then((response) => response.json()).then(res=>console.log(res))
```

Default levels ▾ | 1 Issue: 1

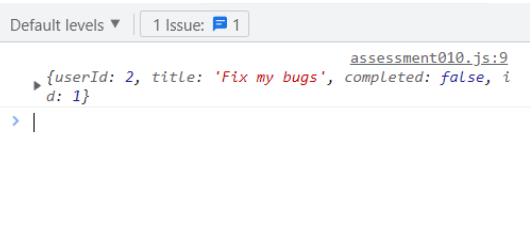
assessment010.js:9
▶ {userId: 1, title: 'Fix my bugs', completed: false, id: 201}

PUT method:

The PUT method replaces all current representations of the target resource with the request payload.

Code:

```
fetch("https://jsonplaceholder.typicode.com/posts/1", {
  method: "PUT",
  headers: {"Content-type": "application/json; charset=UTF-8"},
  body: JSON.stringify({
    userId: 2,
    title: "Fix my bugs",
    completed: false
  })
}).then((response) => response.json()).then(res=>console.log(res))
```

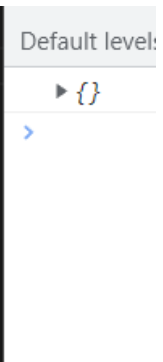


DELETE method:

The DELETE method deletes the specified resource.

Code:

```
fetch("https://jsonplaceholder.typicode.com/posts/1", {
  method: "DELETE",
  headers: {"Content-type": "application/json; charset=UTF-8"},
  body: JSON.stringify({
    userId: 2,
    title: "Fix my bugs",
    completed: false
  })
}).then((response) => response.json()).then(res=>console.log(res))
```

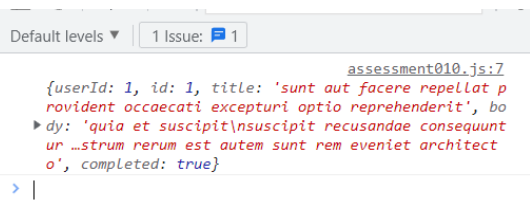


PATCH method:

The PATCH method is used to apply partial modifications to a resource.

Code:

```
fetch("https://jsonplaceholder.typicode.com/posts/1", {
  method: "PATCH",
  headers: {"Content-type": "application/json; charset=UTF-8"},
  body: JSON.stringify({
    completed: true
  })
}).then((response) => response.json()).then(res=>console.log(res))
```



HTTP Status Code:

1. Successful responses (200 – 299)
2. Redirection messages (300 – 399)
3. Client error responses (400 – 499)
4. Server error responses (500 – 599)

200: OK

The most standard response that was for a long time the only used code for websites. You can use this code for every success case if you wish, skipping the others here would be completely acceptable.

201: Created

These are the second most used of the 200s. It is for POST requests where new entries are created.

202: Accepted

This is to indicate that you have accepted a request but may not have necessarily completed it.

204: No Content

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.

301: Moved permanently

The URL of the requested resource has been changed permanently. The new URL is given in the response.

302: Found

This response code means that the URI of requested resource has been changed temporarily. Further changes in the URI might be made in the future. Therefore, this same URI should be used by the client in future requests.

400: Bad request

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.

401: Unauthorized

Although the HTTP standard specifies "unauthorized", semantically this response means "unauthenticated". That is, the client must authenticate itself to get the requested response.

403: Forbidden

The client does not have access rights to the content; that is, it is unauthorized, so the server is refusing to give the requested resource. Unlike 401 Unauthorized, the client's identity is known to the server.

404: Not found

The server cannot find the requested resource. In the browser, this means the URL is not recognized. In an API, this can also mean that the endpoint is valid but the resource itself does not exist.

500: Internal Server Error

The server has encountered a situation it does not know how to handle.

501: Not Implemented

The request method is not supported by the server and cannot be handled. The only methods that servers are required to support (and therefore that must not return this code) are GET and HEAD.

502: Bad Gateway

This error response means that the server, while working as a gateway to get a response needed to handle the request, got an invalid response.