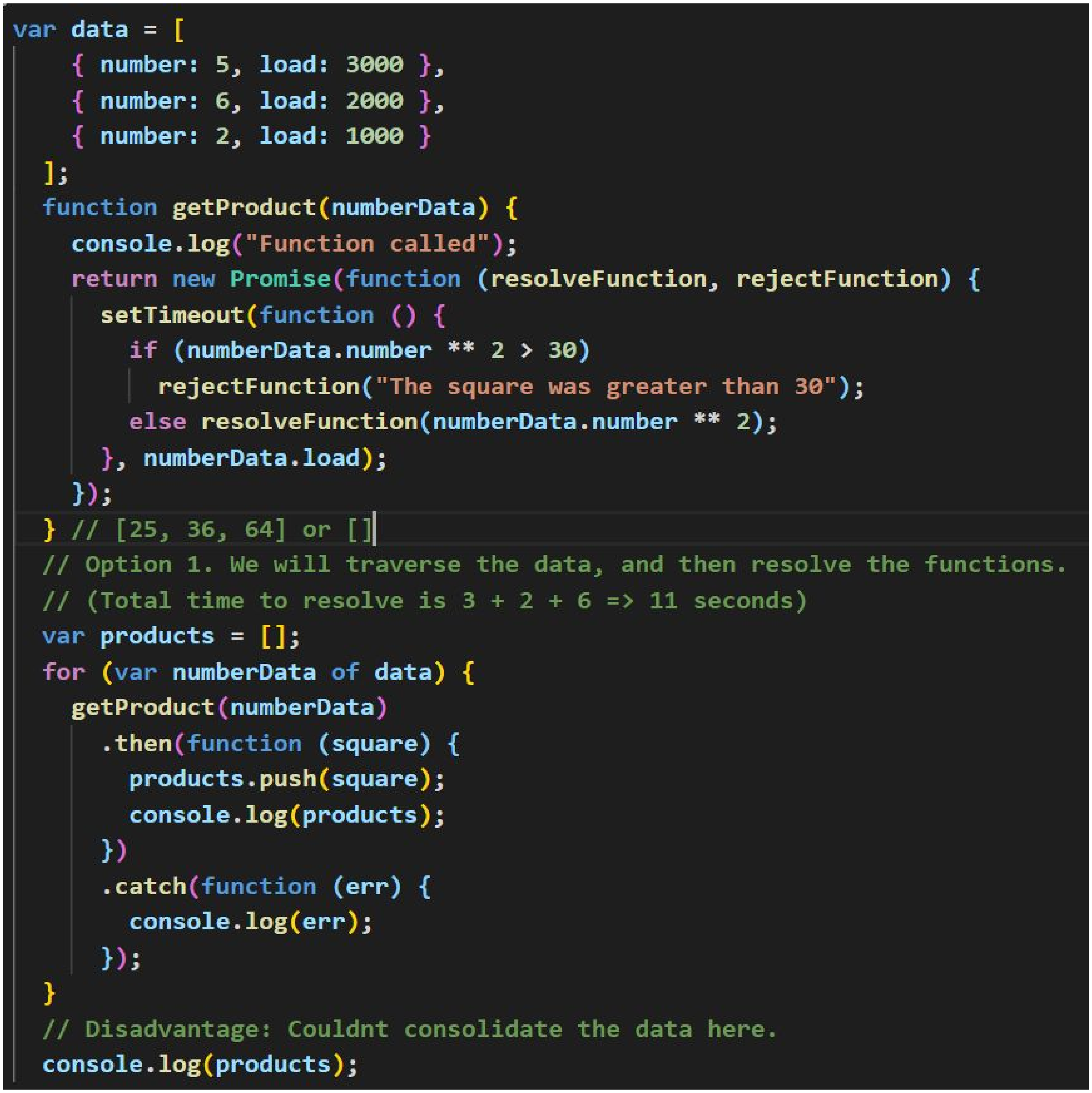
Date : 12-6-2020
Morning Session : 9am – 11 PM
By ~ Sundeep Charan Ramkumar Today

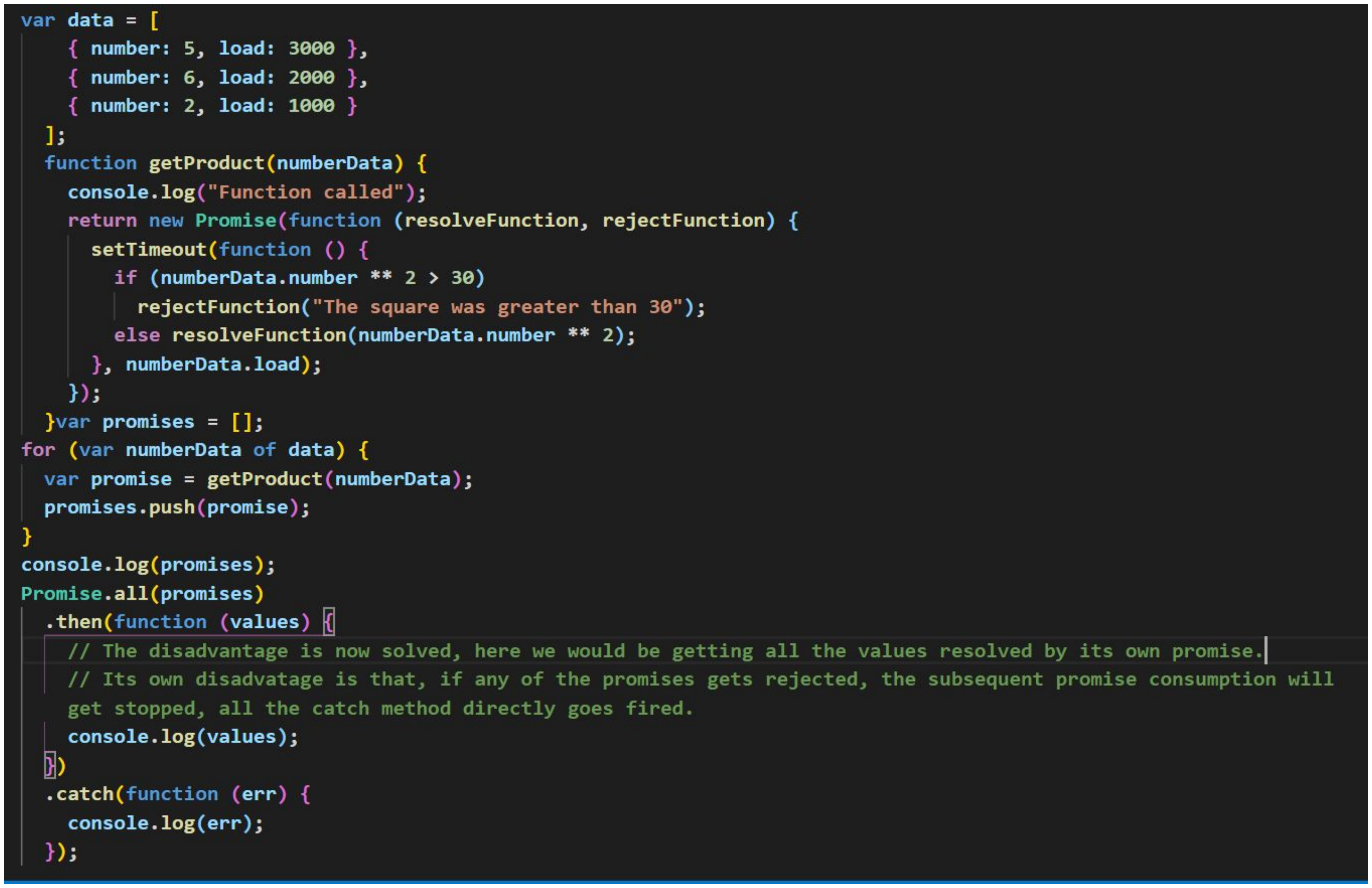
Topics: HTTP Requests and fetch

Array of promises:
Problem statement:



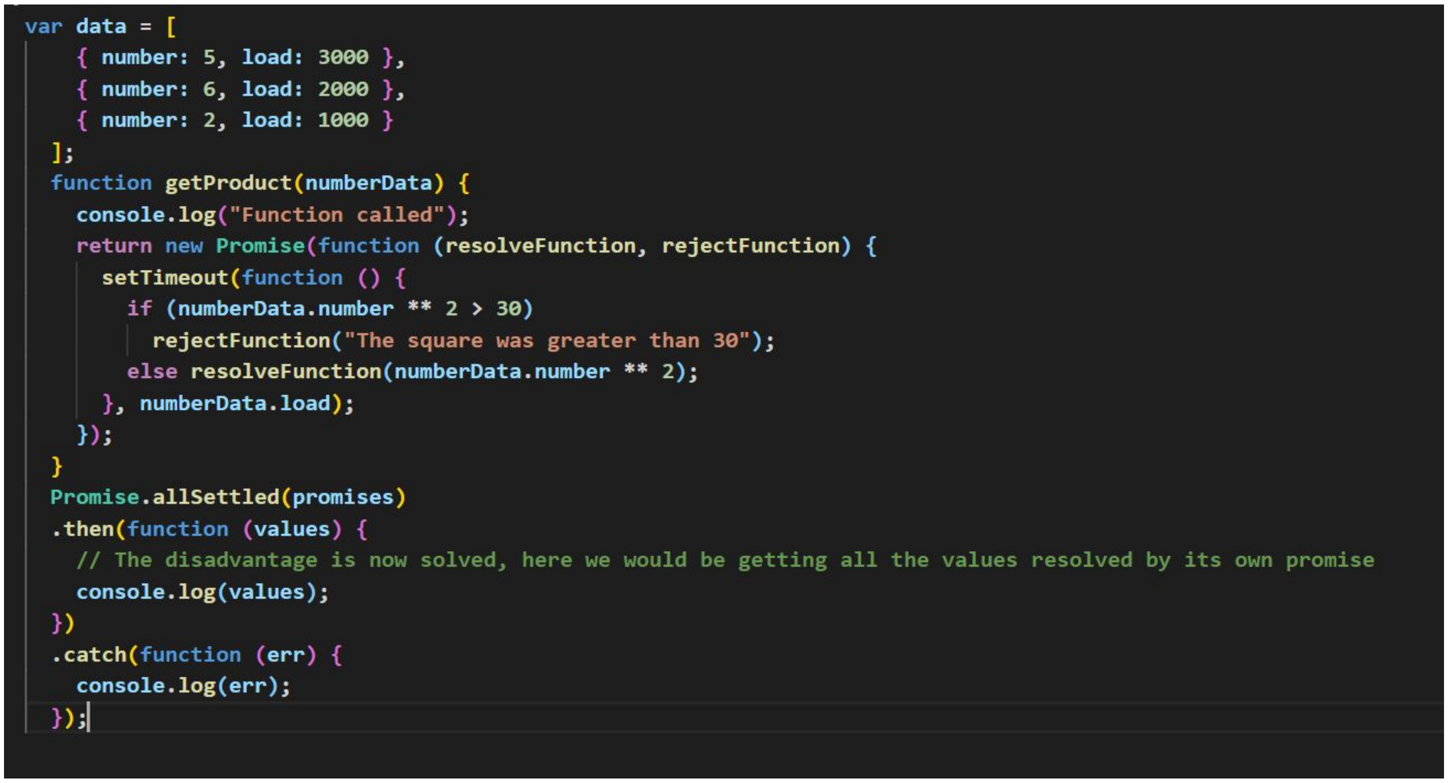
```
var data = [
    { number: 5, load: 3000 },
    { number: 6, load: 2000 },
    { number: 2, load: 1000 }
  ];
  function getProduct(numberData) {
    console.log("Function called");
    return new Promise(function (resolveFunction, rejectFunction) {
      setTimeout(function () {
        if (numberData.number ** 2 > 30)
          rejectFunction("The square was greater than 30");
        else resolveFunction(numberData.number ** 2);
      }, numberData.load);
    });
  } // [25, 36, 64] or []
  // Option 1. We will traverse the data, and then resolve the functions.
  // (Total time to resolve is 3 + 2 + 6 => 11 seconds)
  var products = [];
  for (var numberData of data) {
    getProduct(numberData)
      .then(function (square) {
        products.push(square);
        console.log(products);
      })
      .catch(function (err) {
        console.log(err);
      });
  }
  // Disadvantage: Couldnt consolidate the data here.
  console.log(products);
```

Promise.all: Create promises inside the loop and then call Promise.all()



```javascript
var data = [
    { number: 5, load: 3000 },
    { number: 6, load: 2000 },
    { number: 2, load: 1000 }
  ];
  function getProduct(numberData) {
    console.log("Function called");
    return new Promise(function (resolveFunction, rejectFunction) {
      setTimeout(function () {
        if (numberData.number ** 2 > 30)
          rejectFunction("The square was greater than 30");
        else resolveFunction(numberData.number ** 2);
      }, numberData.load);
    });
  }var promises = [];
for (var numberData of data) {
  var promise = getProduct(numberData);
  promises.push(promise);
}
console.log(promises);
Promise.all(promises)
  .then(function (values) {
    // The disadvantage is now solved, here we would be getting all the values resolved by its own promise.
    // Its own disadvantage is that, if any of the promises gets rejected, the subsequent promise consumption will
    get stopped, all the catch method directly goes fired.
    console.log(values);
  })
  .catch(function (err) {
    console.log(err);
  });
```

Promise.allSelected: **Having all promises run, irrespective of its rejection state.**

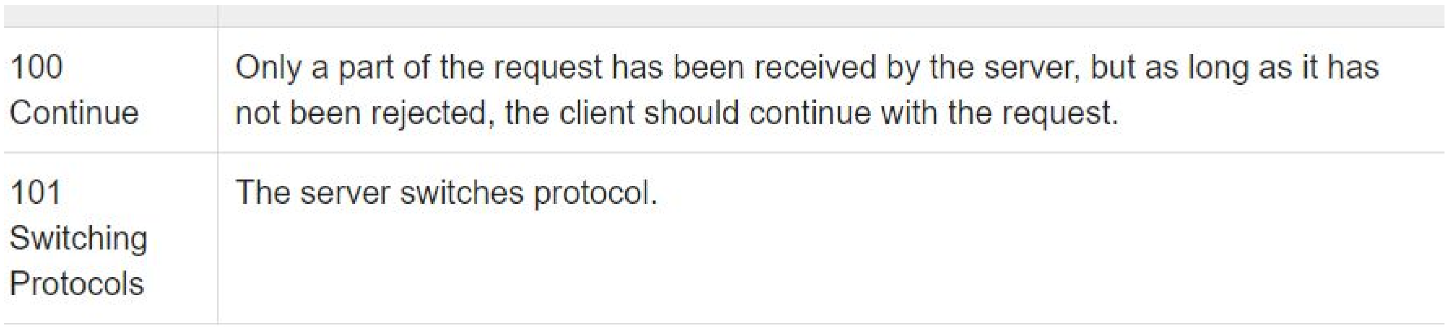


```javascript
var data = [
    { number: 5, load: 3000 },
    { number: 6, load: 2000 },
    { number: 2, load: 1000 }
  ];
  function getProduct(numberData) {
    console.log("Function called");
    return new Promise(function (resolveFunction, rejectFunction) {
      setTimeout(function () {
        if (numberData.number ** 2 > 30)
          rejectFunction("The square was greater than 30");
        else resolveFunction(numberData.number ** 2);
      }, numberData.load);
    });
  }
  Promise.allSettled(promises)
  .then(function (values) {
    // The disadvantage is now solved, here we would be getting all the values resolved by its own promise
    console.log(values);
  })
  .catch(function (err) {
    console.log(err);
  });
```

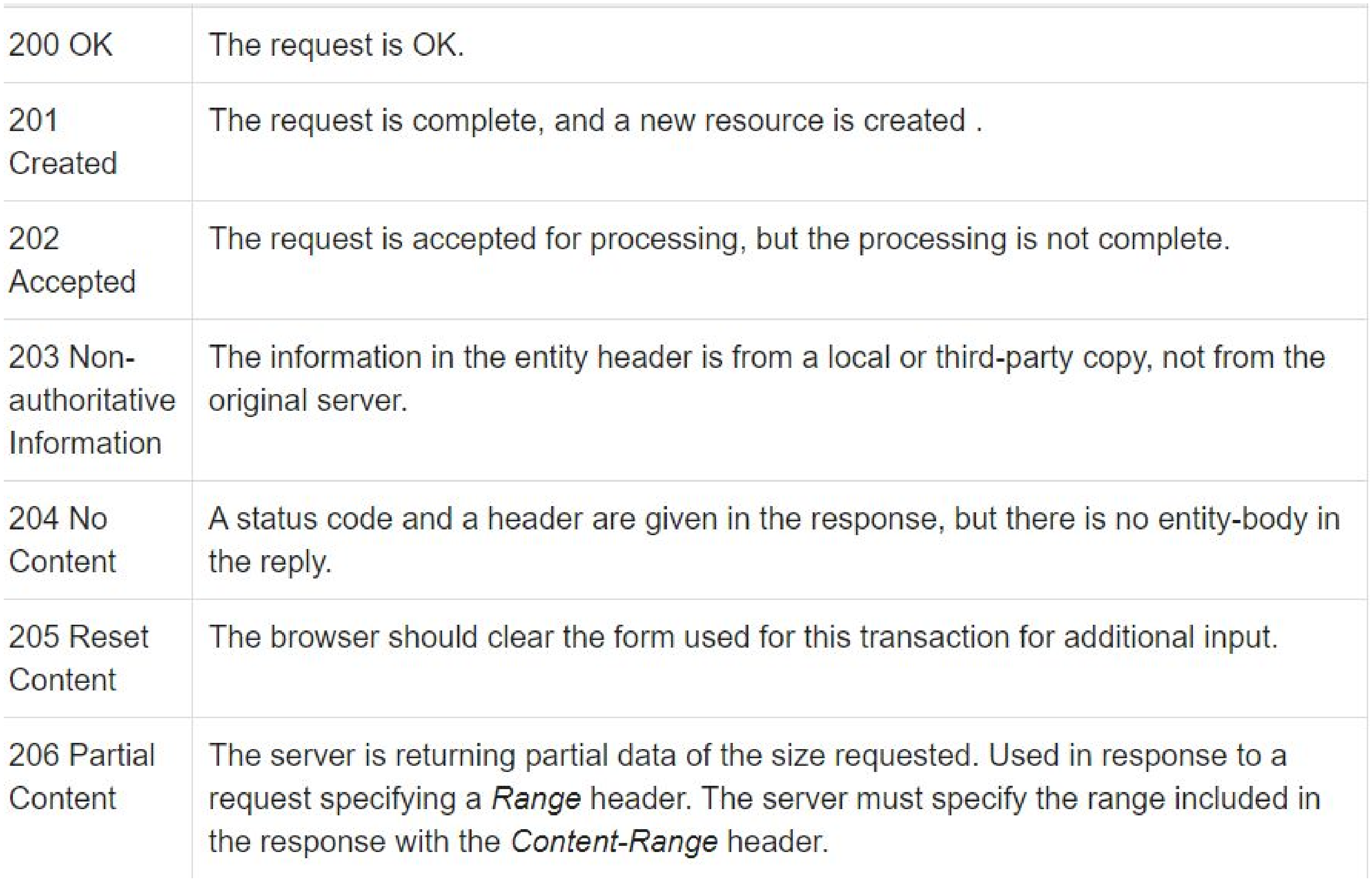
HTTP: Hypertext Transfer Protocol

The HTTP protocol is a request/response protocol based on the client/server based architecture where web browsers, robots and search engines, etc. act like HTTP clients, and the Web server acts as a server.

Status codes:



100 Continue	Only a part of the request has been received by the server, but as long as it has not been rejected, the client should continue with the request.
101 Switching Protocols	The server switches protocol.



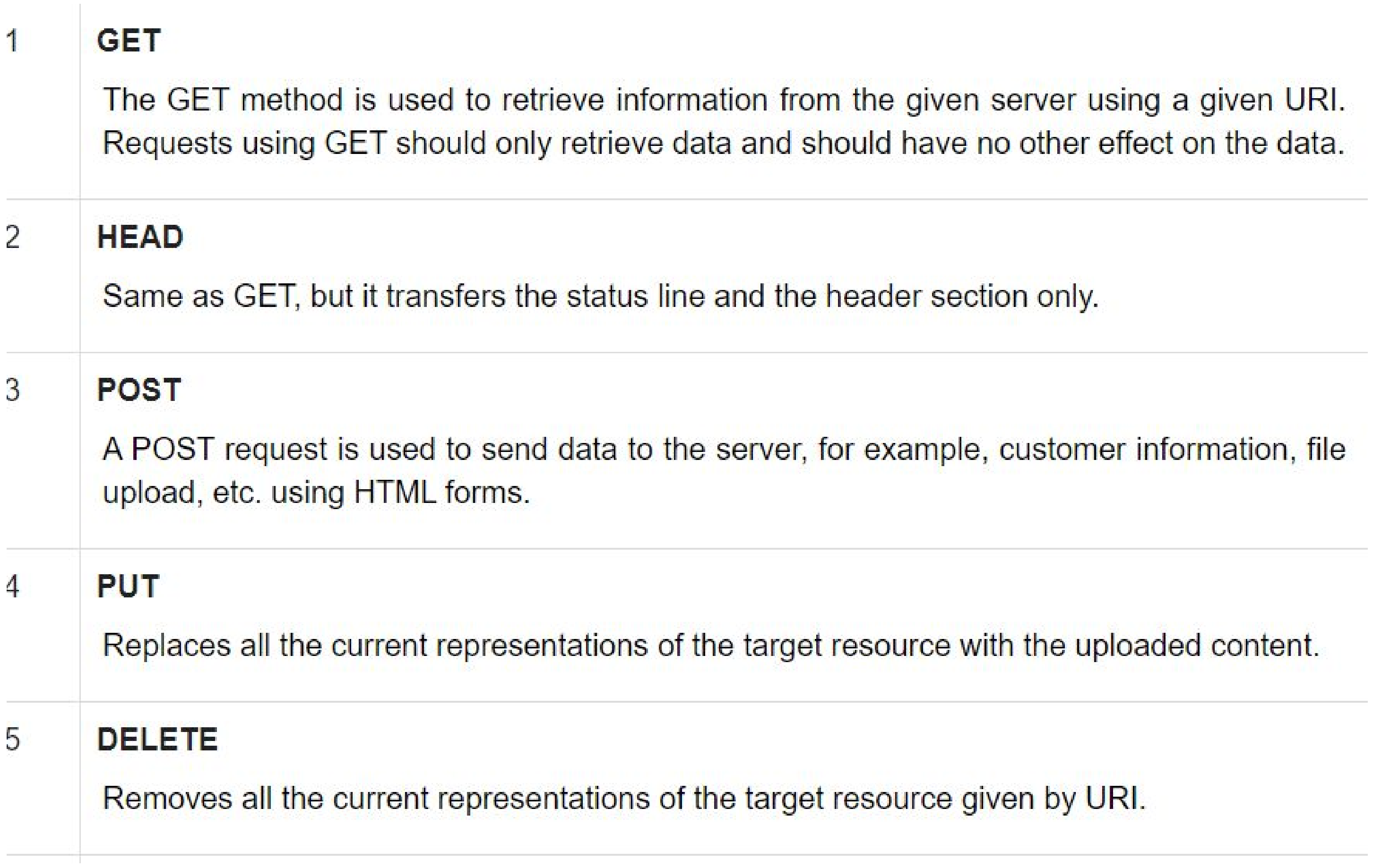
200 OK	The request is OK.
201 Created	The request is complete, and a new resource is created .
202 Accepted	The request is accepted for processing, but the processing is not complete.
203 Non-authoritative Information	The information in the entity header is from a local or third-party copy, not from the original server.
204 No Content	A status code and a header are given in the response, but there is no entity-body in the reply.
205 Reset Content	The browser should clear the form used for this transaction for additional input.
206 Partial Content	The server is returning partial data of the size requested. Used in response to a request specifying a *Range* header. The server must specify the range included in the response with the *Content-Range* header.

300 Multiple Choices	A link list. The user can select a link and go to that location. Maximum five addresses .
301 Moved Permanently	The requested page has moved to a new url .
302 Found	The requested page has moved temporarily to a new url .
303 See Other	The requested page can be found under a different url .
304 Not Modified	This is the response code to an *If-Modified-Since* or *If-None-Match* header, where the URL has not been modified since the specified date.
305 Use Proxy	The requested URL must be accessed through the proxy mentioned in the *Location* header.
306 *Unused*	This code was used in a previous version. It is no longer used, but the code is reserved.
307 Temporary Redirect	The requested page has moved temporarily to a new url.

400 Bad Request	The server did not understand the request.
401 Unauthorized	The requested page needs a username and a password.
402 Payment Required	*You can not use this code yet.*
403 Forbidden	Access is forbidden to the requested page.
404 Not Found	The server can not find the requested page.
405 Method Not Allowed	The method specified in the request is not allowed.
406 Not Acceptable	The server can only generate a response that is not accepted by the client.
407 Proxy Authentication Required	You must authenticate with a proxy server before this request can be served.
408 Request Timeout	The request took longer than the server was prepared to wait.
409 Conflict	The request could not be completed because of a conflict.
410 Gone	The requested page is no longer available .
411 Length Required	The "Content-Length" is not defined. The server will not accept the request without it .

500 Internal Server Error	The request was not completed. The server met an unexpected condition.
501 Not Implemented	The request was not completed. The server did not support the functionality required.
502 Bad Gateway	The request was not completed. The server received an invalid response from the upstream server.
503 Service Unavailable	The request was not completed. The server is temporarily overloading or down.
504 Gateway Timeout	The gateway has timed out.
505 HTTP Version Not Supported	The server does not support the "http protocol" version.

Types of requests



1	**GET** The GET method is used to retrieve information from the given server using a given URI. Requests using GET should only retrieve data and should have no other effect on the data.
2	**HEAD** Same as GET, but it transfers the status line and the header section only.
3	**POST** A POST request is used to send data to the server, for example, customer information, file upload, etc. using HTML forms.
4	**PUT** Replaces all the current representations of the target resource with the uploaded content.
5	**DELETE** Removes all the current representations of the target resource given by URI.

HTTP header: HTTP header fields provide required information about the request or response, or about the object sent in the message body. There are four types of HTTP message headers:

- General-header: These header fields have general applicability for both request and response messages.
- Client Request-header: These header fields have applicability only for request messages.
- Server Response-header: These header fields have applicability only for response messages.
- Entity-header: These header fields define meta information about the entity-body or, if nobody is present, about the resource identified by the request.

Fetch: **fetch method is something is a function which return promise whether it got successful or rejected**

The Fetch provides a fetch() method defined on the window object, which you can use to perform requests. This method returns a Promise that you can use to retrieve the response of the request.

```
fetch('https://www.reddit.com/r/javascript/top/.json?limit=5')
.then(res => console.log(res));
```

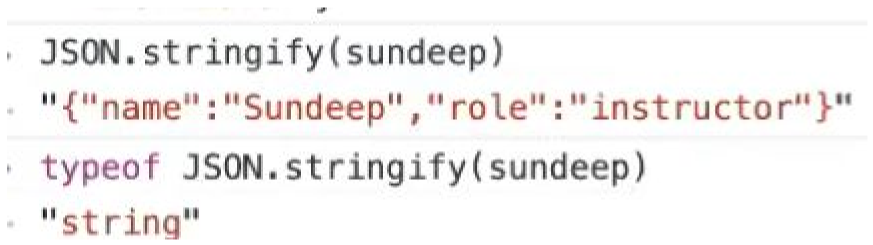
If you inspect the response in your browser's console, you should see a Response object with several properties:

```
{
  body: ReadableStream
  bodyUsed: false
  headers: Headers {}
  ok: true
  redirected: false
  status: 200
  statusText: ""
  type: "cors"
  url: "https://www.reddit.com/top/.json?count=5"
}
```

It seems that the request was successful,

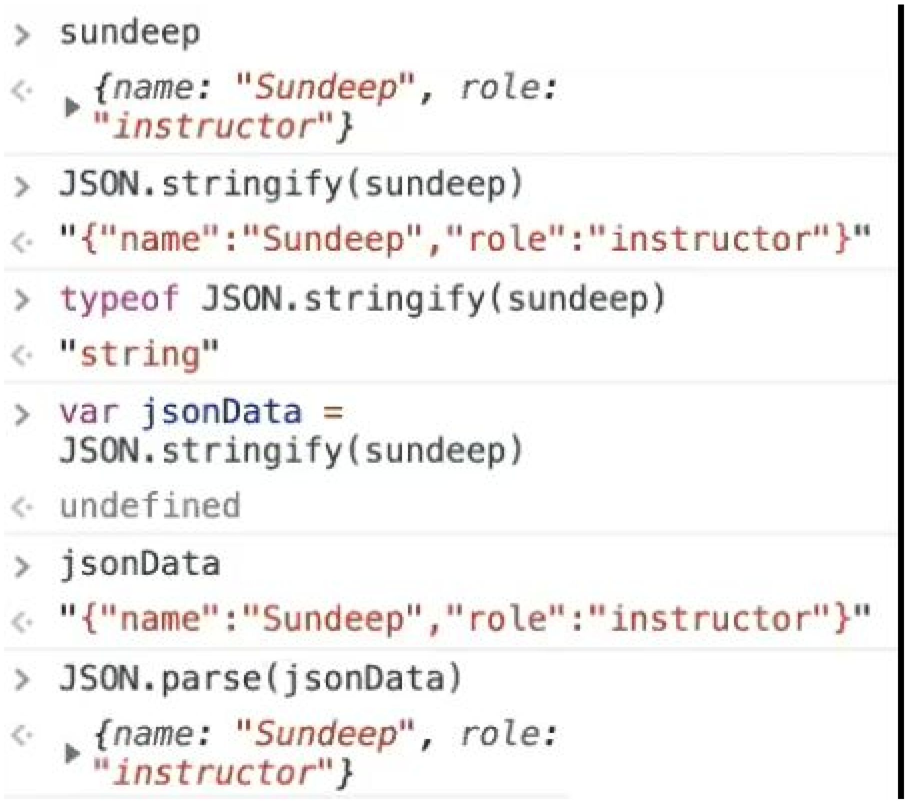
API : Application programming interface, API which will give you some relevant data which will be valuable to you some sorts.

JSON: Means Javascript object notation, javascript object consists of Key value pair , **JSON** is String **(every property and value have string)**



```
JSON.stringify(sundeep)
"{"name":"Sundeep","role":"instructor"}"
typeof JSON.stringify(sundeep)
"string"
```

To convert to javascript object



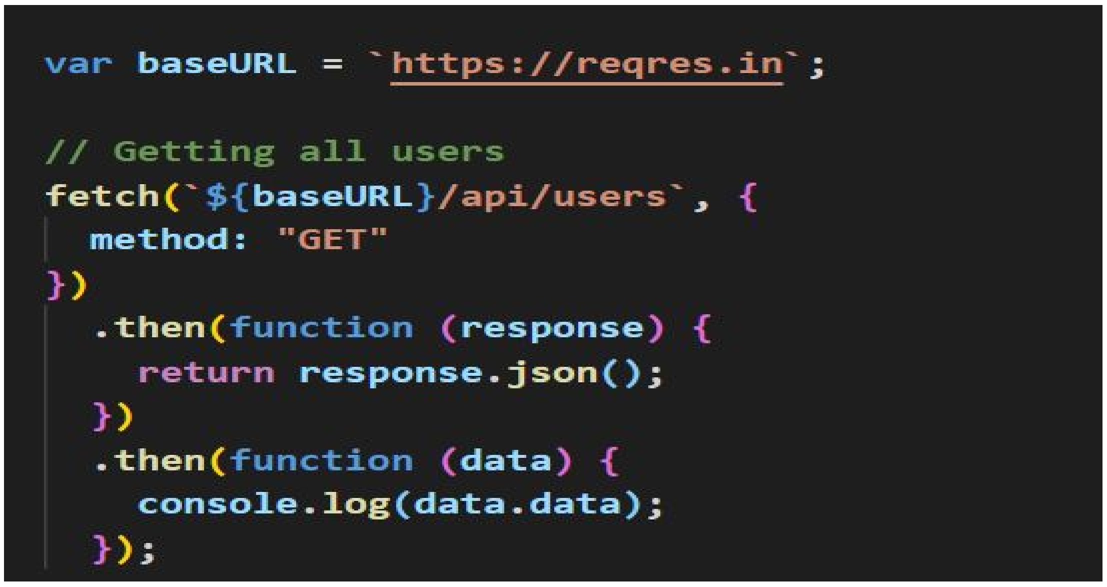
```
> sundeep
<· ▸ {name: "Sundeep", role:
   "instructor"}
> JSON.stringify(sundeep)
<· "{"name":"Sundeep","role":"instructor"}"
> typeof JSON.stringify(sundeep)
<· "string"
> var jsonData =
  JSON.stringify(sundeep)
<· undefined
> jsonData
<· "{"name":"Sundeep","role":"instructor"}"
> JSON.parse(jsonData)
<· ▸ {name: "Sundeep", role:
   "instructor"}
```

https://reqres.in/

https://reqres.in/api/users?page=2

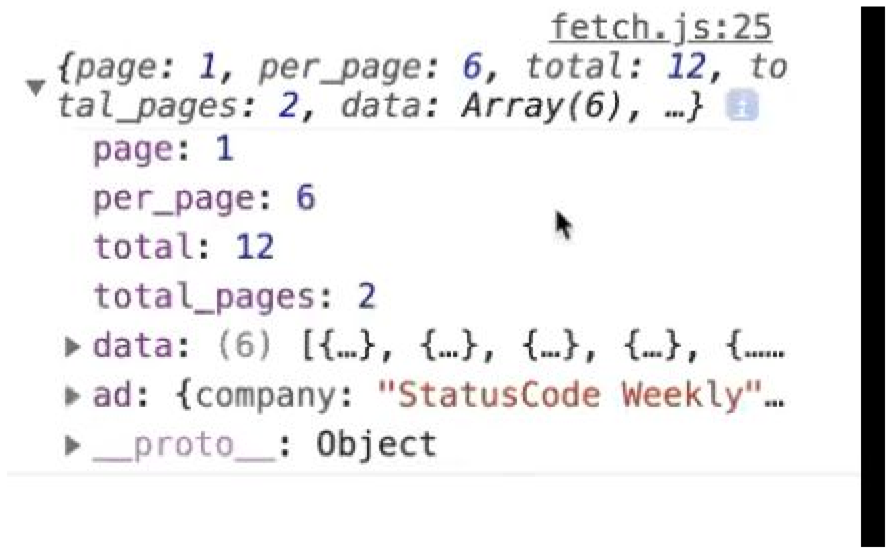
JSON Viewer Extension
https://chrome.google.com/webstore/detail/json-viewer/gbmdgpbipfallnflgajpaliibnhdgobh?utm_source=chrome-ntp-icon

Fetch GET Request :



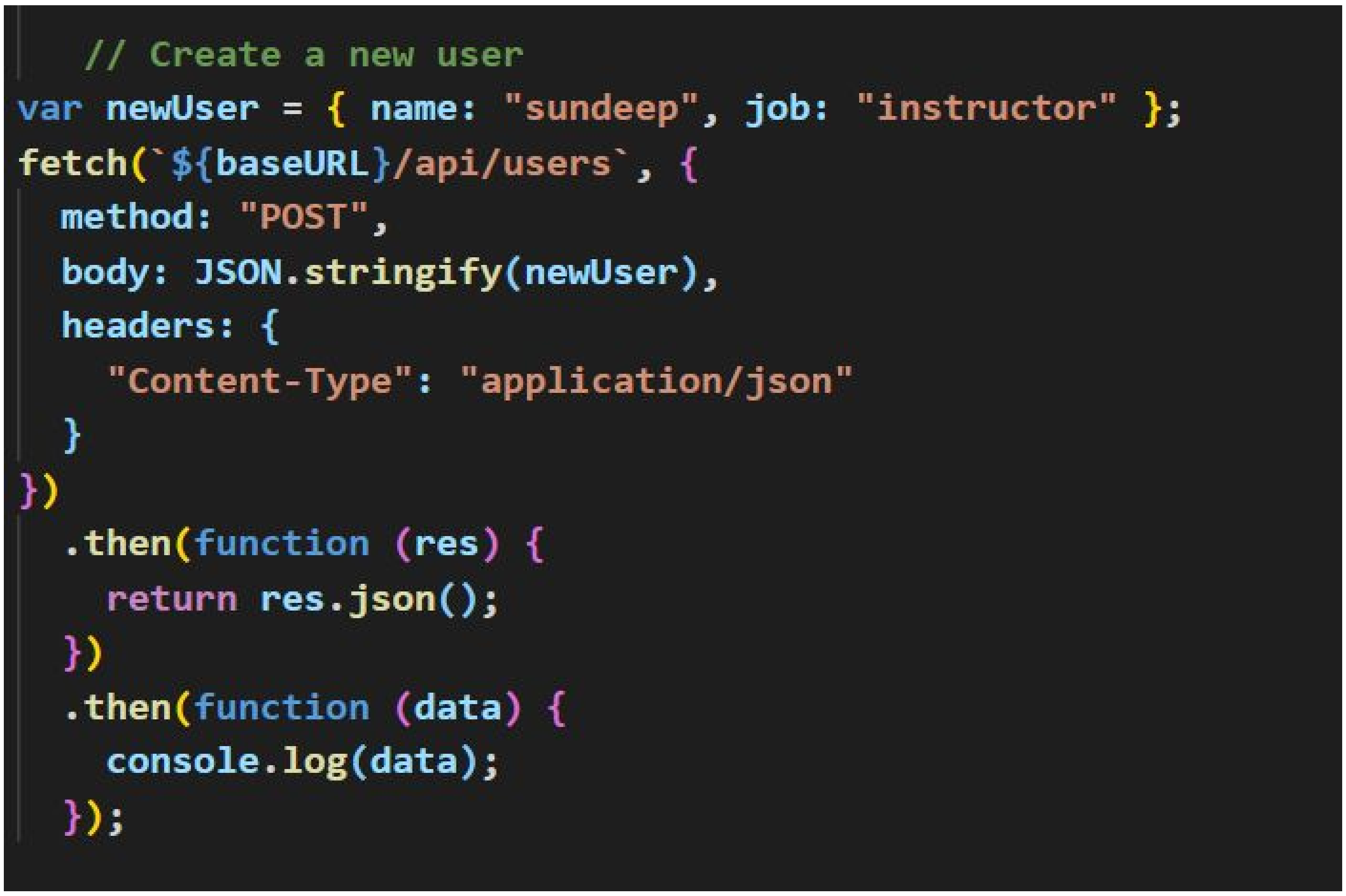
```
var baseURL = `https://reqres.in`;

// Getting all users
fetch(`${baseURL}/api/users`, {
  method: "GET"
})
  .then(function (response) {
    return response.json();
  })
  .then(function (data) {
    console.log(data.data);
  });
```



```
                                       fetch.js:25
▼ {page: 1, per_page: 6, total: 12, to
  tal_pages: 2, data: Array(6), …}
    page: 1
    per_page: 6
    total: 12
    total_pages: 2
  ▶ data: (6) [{…}, {…}, {…}, {…}, {……
  ▶ ad: {company: "StatusCode Weekly"…
  ▶ __proto__: Object
```

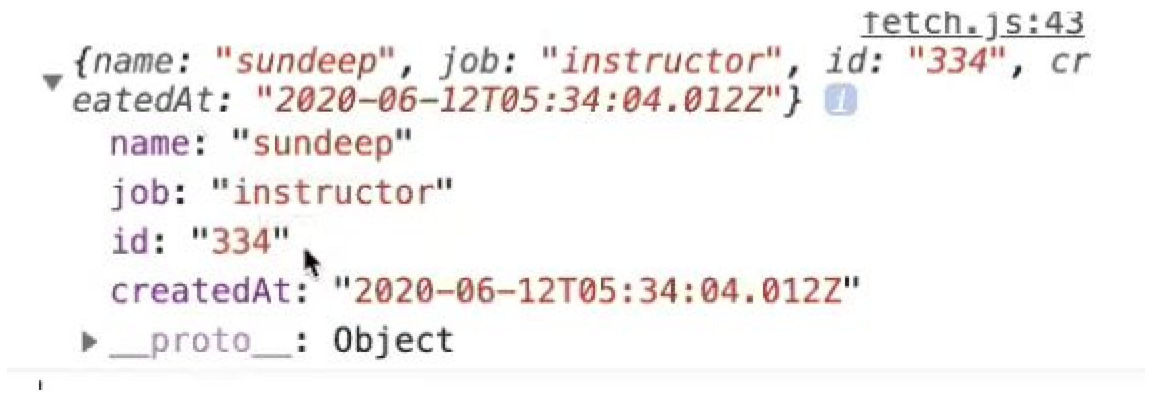
Fetch POST Request:



```
  // Create a new user
var newUser = { name: "sundeep", job: "instructor" };
fetch(`${baseURL}/api/users`, {
  method: "POST",
  body: JSON.stringify(newUser),
  headers: {
    "Content-Type": "application/json"
  }
})
  .then(function (res) {
    return res.json();
  })
  .then(function (data) {
    console.log(data);
  });
```

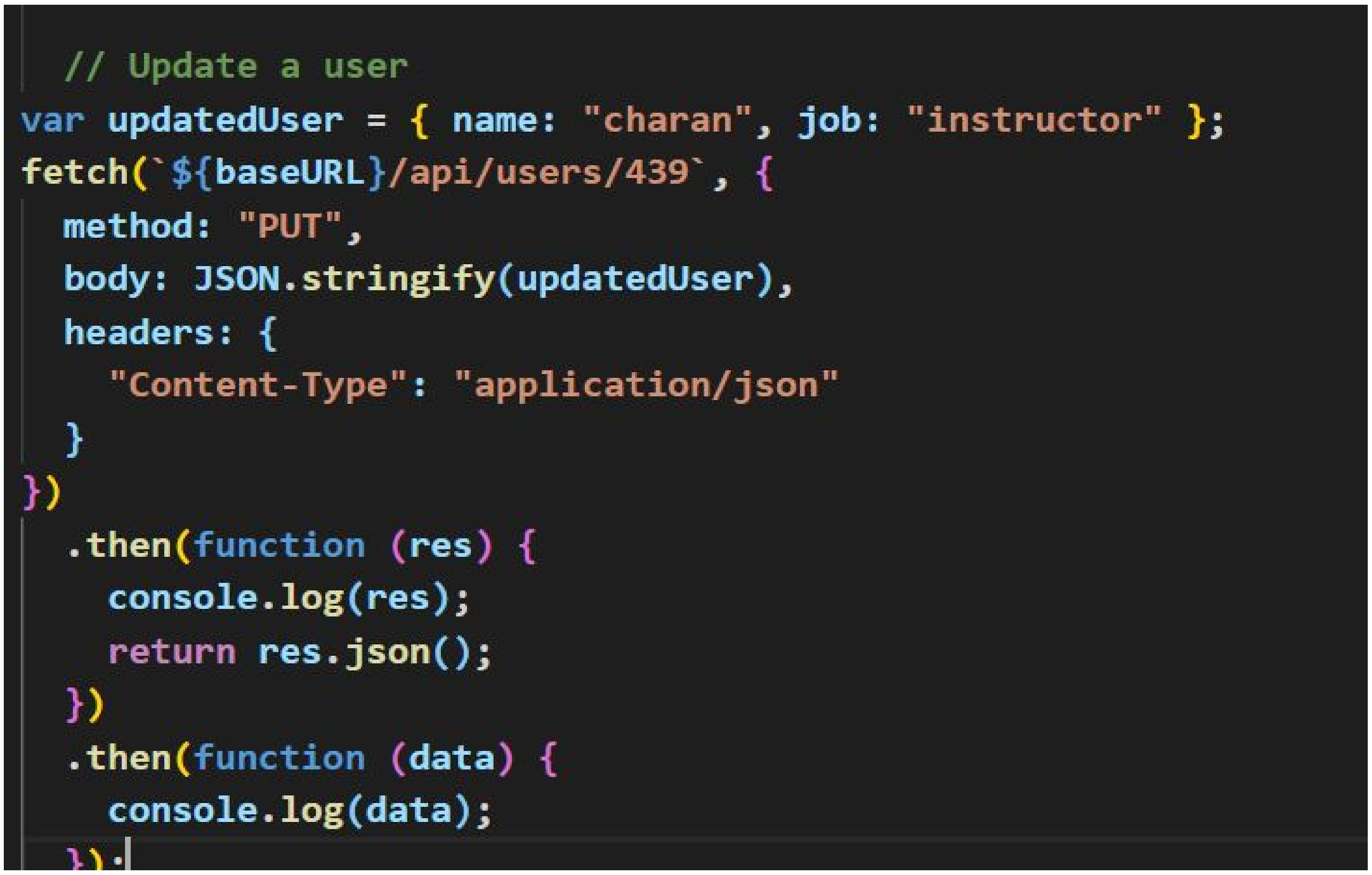
Content-Type : The Content-Type entity header is used to indicate the media type of the resource.

```
Content-Type: text/html; charset=UTF-8
Content-Type: multipart/form-data; boundary=something
```

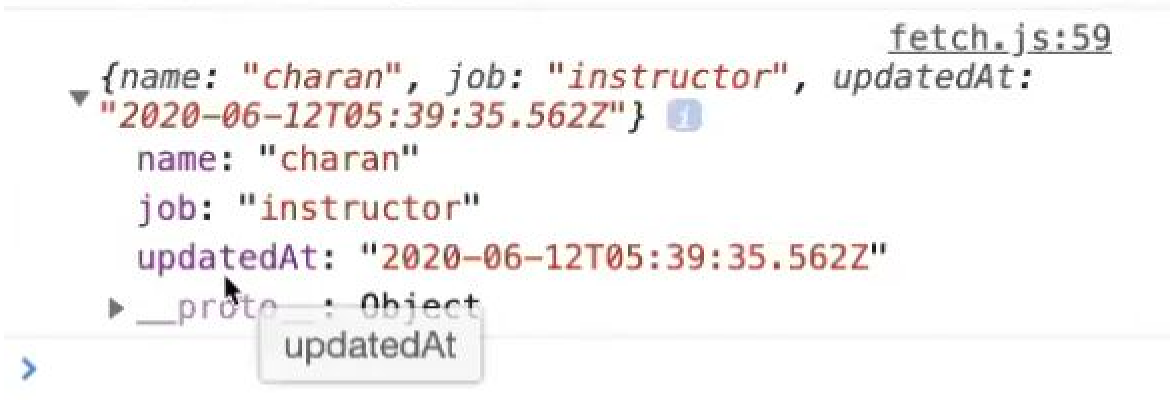


```
                                                    fetch.js:43
▼ {name: "sundeep", job: "instructor", id: "334", cr
  eatedAt: "2020-06-12T05:34:04.012Z"}
    name: "sundeep"
    job: "instructor"
    id: "334"
    createdAt: "2020-06-12T05:34:04.012Z"
  ▶ __proto__: Object
```

Fetch Put Request:

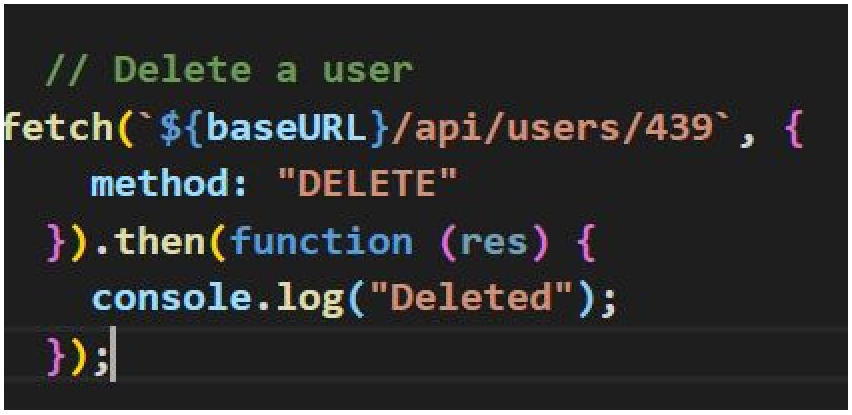


```
  // Update a user
var updatedUser = { name: "charan", job: "instructor" };
fetch(`${baseURL}/api/users/439`, {
  method: "PUT",
  body: JSON.stringify(updatedUser),
  headers: {
    "Content-Type": "application/json"
  }
})
  .then(function (res) {
    console.log(res);
    return res.json();
  })
  .then(function (data) {
    console.log(data);
```



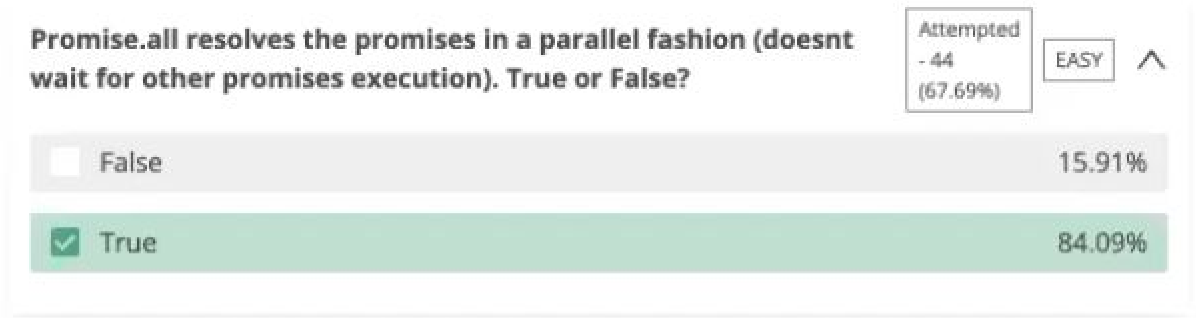
```
                                                    fetch.js:59
▼ {name: "charan", job: "instructor", updatedAt:
  "2020-06-12T05:39:35.562Z"}
    name: "charan"
    job: "instructor"
    updatedAt: "2020-06-12T05:39:35.562Z"
  ▶ __proto__: Object
>
```

Fetch DELETED Request:



```
  // Delete a user
fetch(`${baseURL}/api/users/439`, {
    method: "DELETE"
  }).then(function (res) {
    console.log("Deleted");
  });
```

MCQ1:

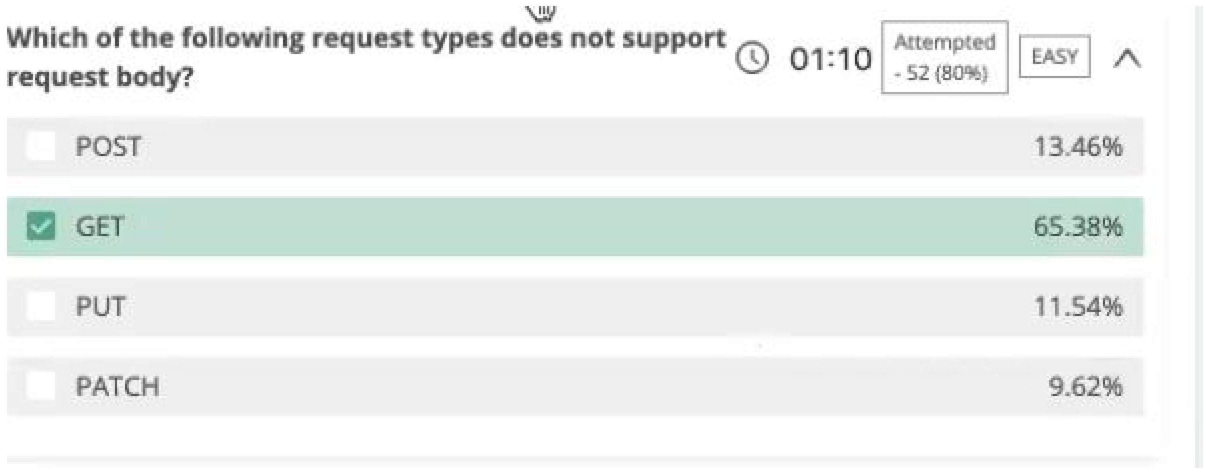


Promise.all resolves the promises in a parallel fashion (doesnt wait for other promises execution). True or False?

Attempted - 44 (67.69%) EASY

- [ ] False 15.91%
- [✓ True] 84.09%

MCQ2:

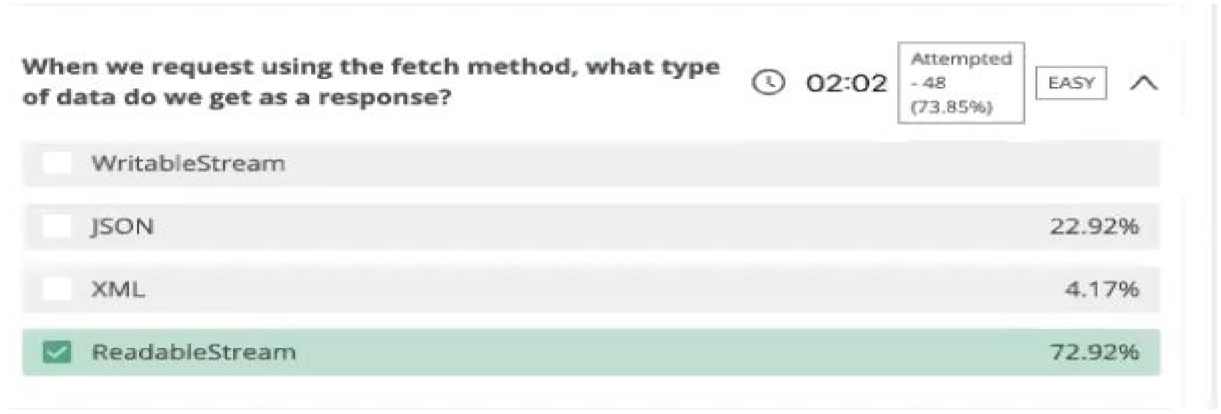


Which of the following request types does not support request body?

01:10 Attempted - 52 (80%) EASY

- [ ] POST 13.46%
- [✓ GET] 65.38%
- [ ] PUT 11.54%
- [ ] PATCH 9.62%

MCQ3:

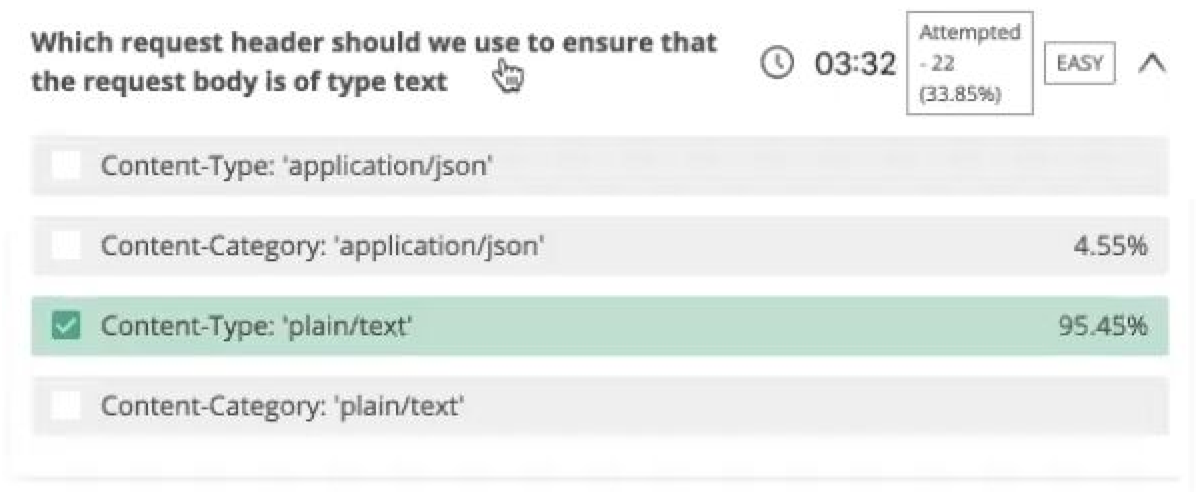


When we request using the fetch method, what type of data do we get as a response?

02:02 Attempted - 48 (73.85%) EASY

- [ ] WritableStream
- [ ] JSON 22.92%
- [ ] XML 4.17%
- [✓ ReadableStream] 72.92%

MCQ4:



Which request header should we use to ensure that the request body is of type text

03:32

Attempted - 22 (33.85%)

EASY

- [ ] Content-Type: 'application/json'
- [ ] Content-Category: 'application/json' 4.55%
- [x] Content-Type: 'plain/text' 95.45%
- [ ] Content-Category: 'plain/text'

Async Programming Resources :

https://www.youtube.com/watch?v=PoRJizFvM7s

https://www.youtube.com/watch?v=gB-OmN1egV8

https://www.youtube.com/watch?v=_8gHHBlbziw