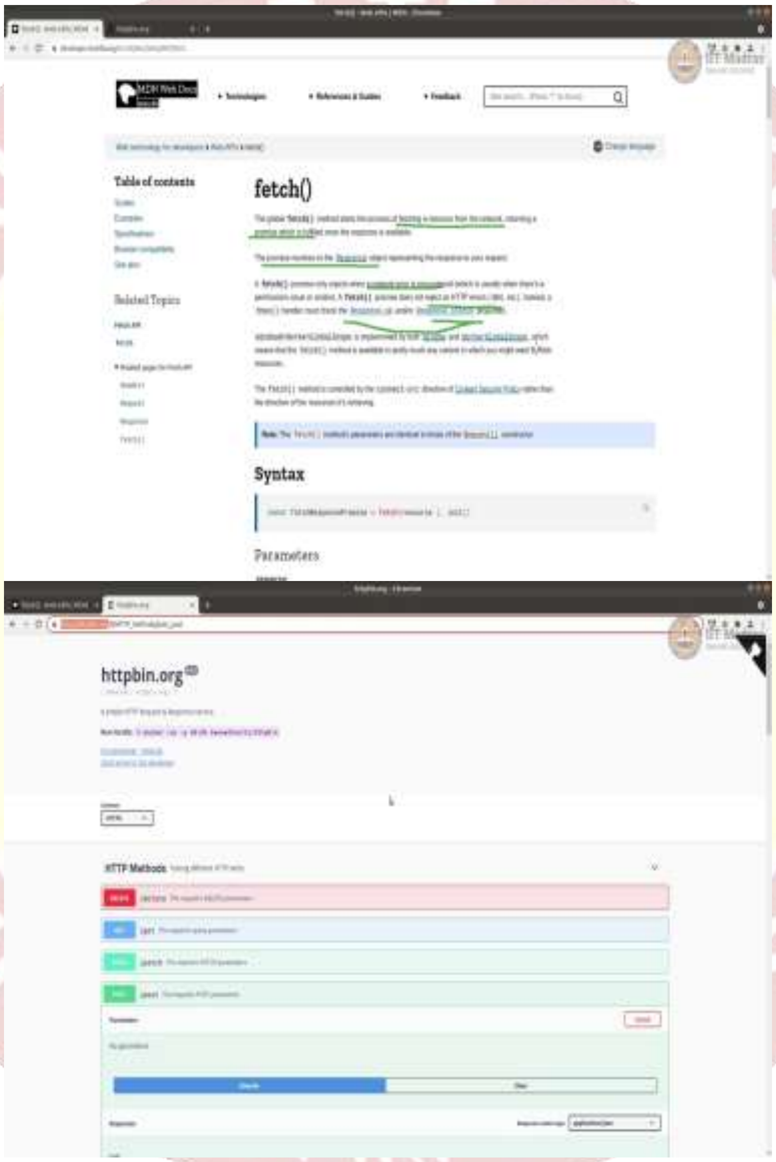
INDIAN INSTITUTE OF TECHNOLOGY MADRAS • सिद्धिर्भवति कर्मजा •

IIT Madras
ONLINE DEGREE

Modern Application Development - II
Professor Thejesh G.N.
Software Consultant,
Indian Institute of Technology, Madras
JavaScript - Fetch API

(Refer Slide Time: 00:14)



Syntax

Parameters

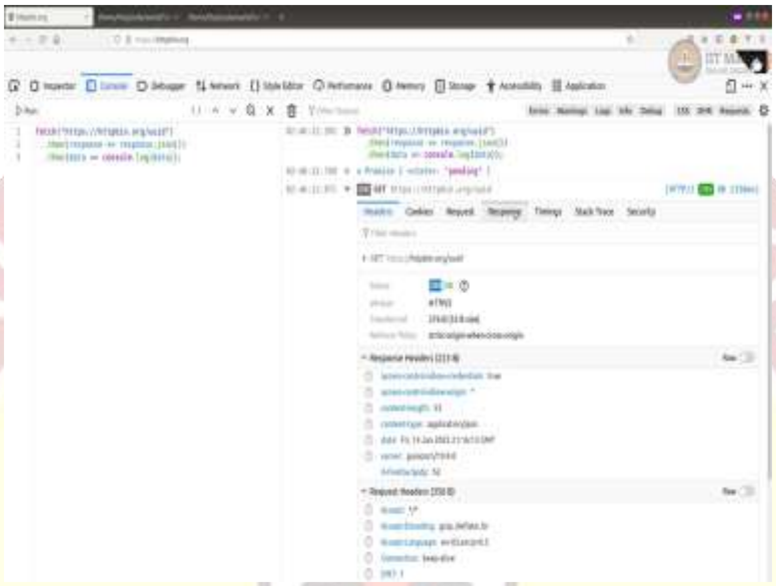
Welcome to the Modern Application Development II screencast. In this screencast, we will explore fetch APIs to request the data from the network. For this, you will need a browser, Firefox or Chrome. I am using Firefox to do all the exploration with developer console open, and then I am also going to use an online tool called httpbin where we can send request and get responses and explore various API methods and formats for us to test our fetch requests. Let us do learn now what fetch means. Let us go to MDN documents here.

Now, fetch is a modern way of doing network access. In the sense, it is used to fetch data from the network. It returns a promise that the document is hence you will have to wait for the promise to fulfill or resolve to get a response object. Once the promise resolves, it gets a response object. Now, the promise never rejects if there is an HTTP error. There are two kinds of errors. One is either the network error or there can be HTTP error.

This specific promise will only reject if there is a network error. It can happen because there is no Internet or there is no permission, etc., etc. But it will never throw a rejection if there is an HTTP error, like page not found or 505 errors like that. Hence, we will always have to do response okay check or the status check or whether the status is 404, 505, checks like that, before we actually try to get the data. It is something for us to remember.

Now, fetch at a basic level takes two parameters. One is a resource, other one is init. Now, the resource can be a string or a request object. We will look at the request object later. It can be a string or which means it can be in URL. And init can take various attributes. By default, you do not need to pass. It will assume certain defaults and call. So, let us look at that minimal HTTP request that we can send using fetch. Let us go back to console here.

(Refer Slide Time: 02:53)



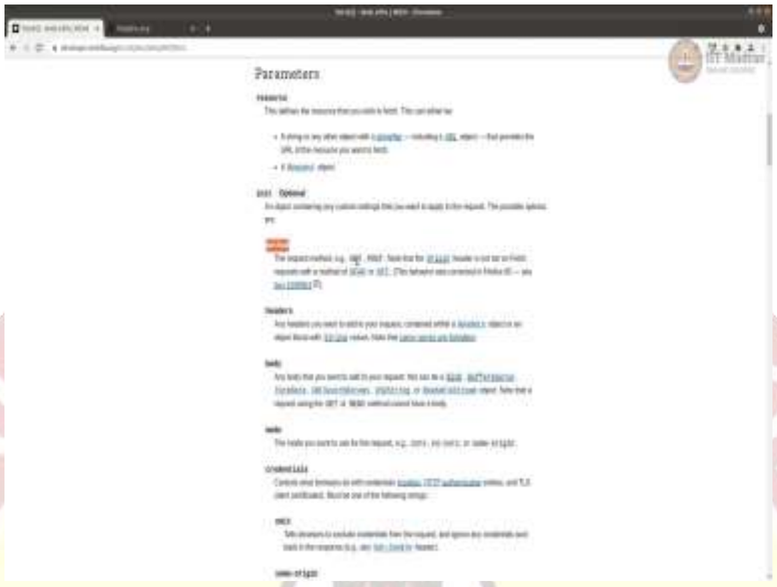
```
fetch("https://httpbin.org/uuid")
  .then(response => response.json())
  .then(data => console.log(data));
```

```
> fetch("https://httpbin.org/uuid")
  .then(response => response.json())
  .then(data => console.log(data));
< Promise { <state>: "pending" }
▼ GET https://httpbin.org/uuid                    [HTTP/1.1 200 OK]
```

So, I am sending a request to this URL. I am not passing any parameters here, just leaving it as it is, just giving a string URL, and then you know handling the first promise, and then that gives me a response, and then calling response.json, which also gives me a promise. I am handling that and printing a data. Now, let us run this. Resolved got the answer.

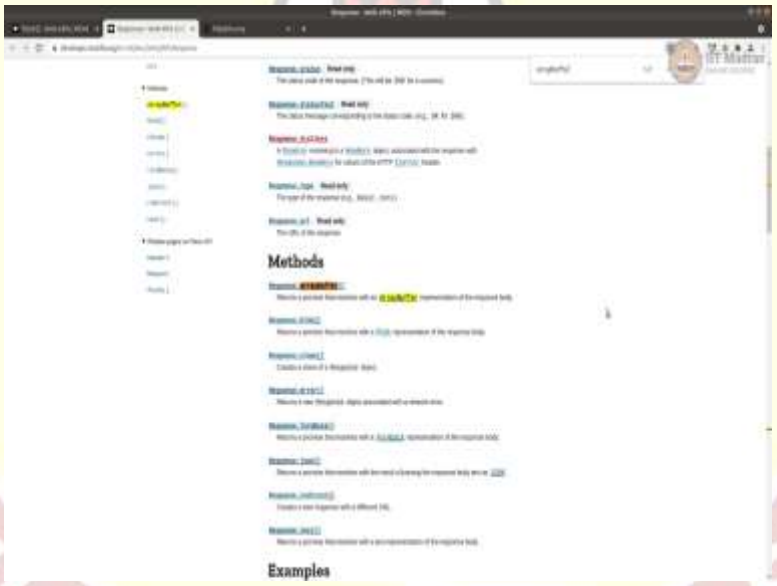
Now, we can also go ahead and check request and response details here. You can check the request. It has no payload because it is doing by default get. You can check in the headers. You can see it's 200. It's doing a get and you can also check the response, which is a json, you can see the json data here. If you click on raw, you can actually see the json. So, this is the very basic HTTP request made using the fetch API.

(Refer Slide Time: 04:13)



Parameters

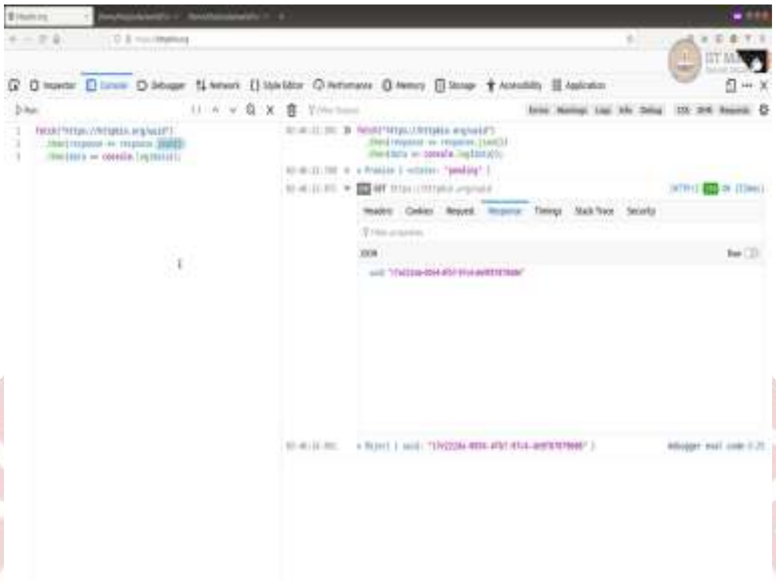
[illegible]



Methods

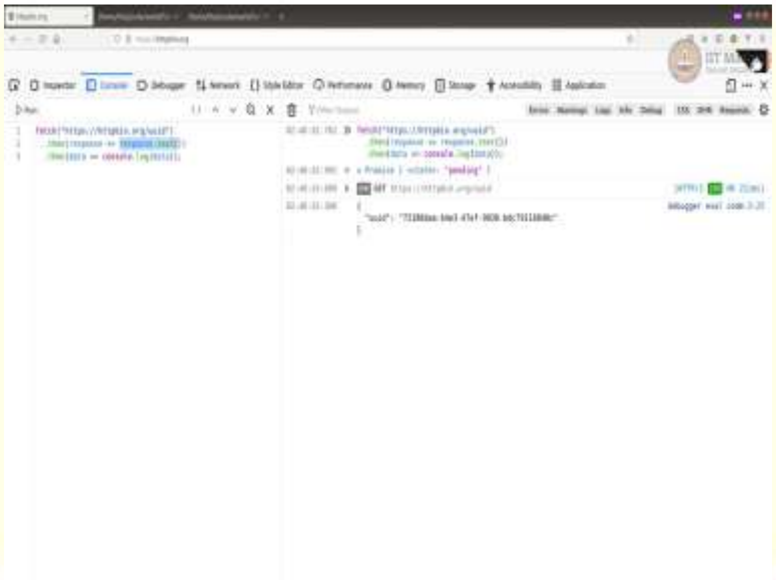
[illegible]

Examples



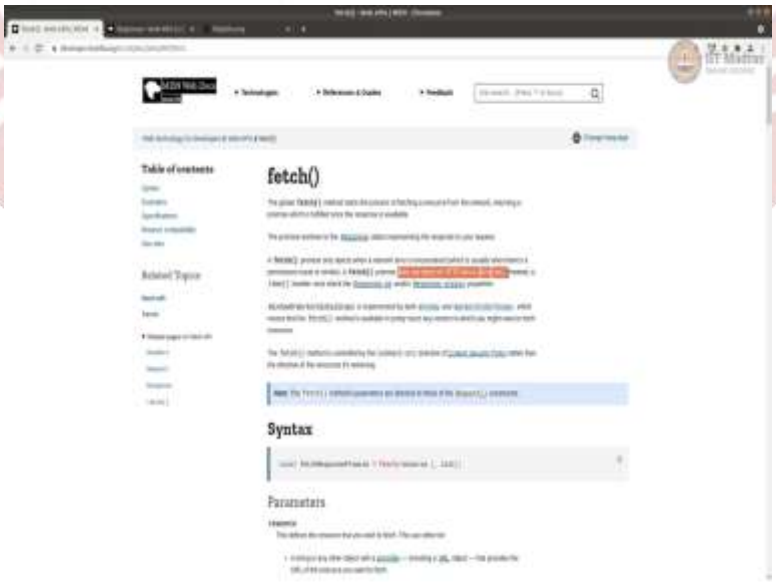
```
fetch("https://httpbin.org/uuid")
  .then(response => response.json())
  .then(data => console.log(data));
```

```
>> fetch("https://httpbin.org/uuid")
     .then(response => response.json())
     .then(data => console.log(data));
<- Promise { <state>: "pending" }
▼ GET https://httpbin.org/uuid                         [HTTP/1.1 200 OK]
   Headers  Cookies  Request  Response  Timings  Stack Trace  Security
   JSON
     uuid: "[illegible]"
   Object { uuid: "[illegible]" }                        debugger eval code:3:25
```



```
fetch("https://httpbin.org/uuid")
  .then(response => response.text())
  .then(data => console.log(data));
```

```
>> fetch("https://httpbin.org/uuid")
     .then(response => response.text())
     .then(data => console.log(data));
<- Promise { <state>: "pending" }
▶ GET https://httpbin.org/uuid                         [HTTP/1.1 200 OK]
   {                                                     debugger eval code:3:25
     "uuid": "[illegible]"
   }
```



fetch()

[illegible]

Syntax

[illegible]

Parameters

[illegible]

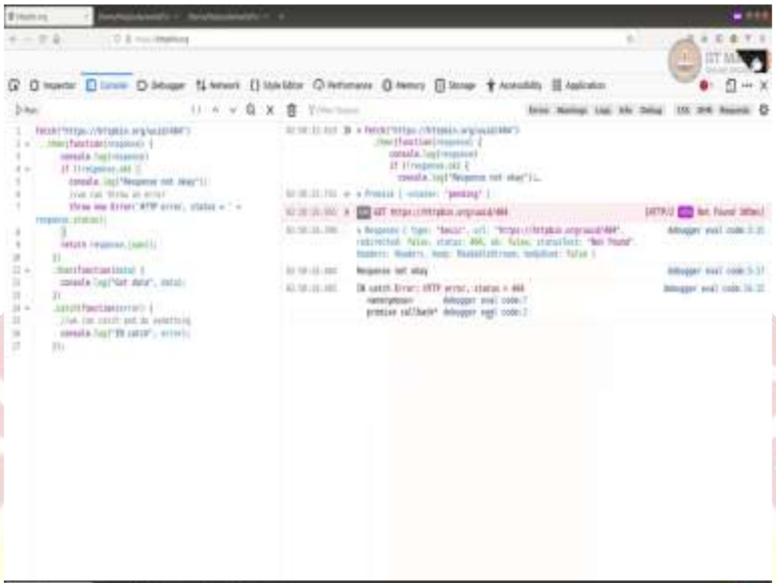
Now, let us go ahead and check what are the parameters that we can send as part of the fetch. Now, here you can see in the second part, you can actually send method which by default is get, we can send other methods like post, put etc. We can also send headers, body and many other details. And also, it can return for example many kinds of data. Let's check whether they have it listed here, no. Let's go to response object.

So, for example, here we have used response.json, but we can get many other data types too. For example, we can get array buffer, we can get a blob, which is binary file type data, usually we get, use it to download a file, which we do not know much about, then you can obviously get json and then you can get text, you can also get form data. These are the different types of data that you can get out of the response or you can also send it as part of the request.

So, for example, I could have done text here. It would have, run and given me the format in the form of text instead of in the format of json. You know that's something to remember. So, those are all the functions available. They all written including, you can see here even the response.text returns a promise and you have to resolve. So, they all written promises. You need to resolve the promises.

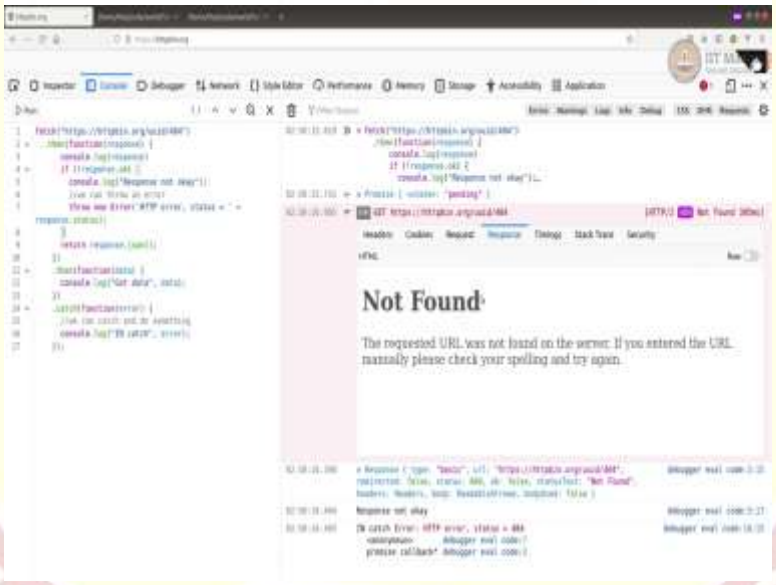
Now, how do we handle error? For example, if it returned like a 404 or 505, how do we handle the error. Like I told you before that fetch actually doesn't throw rejection if there is a 404 error. We will have to check response, okay, or response status. So, we will, let us do that and see how that looks.

(Refer Slide Time: 06:39)



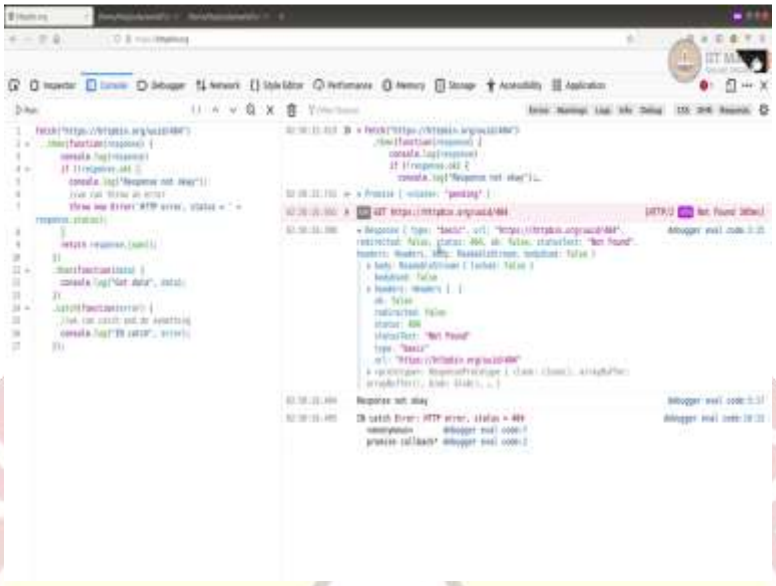
```
fetch("https://httpbin.org/uuid/404")
  .then(function(response) {
    console.log(response)
    if (!response.ok) {
      console.log("Response not okay");
      //we can throw an error
      throw new Error("HTTP error, status = " +
response.status);
    }
    return response.json();
  })
  .then(function(data) {
    console.log("Got data", data);
  })
  .catch(function(error) {
    //we can catch and do something
    console.log("IN catch", error);
  });
```

```
>> fetch("https://httpbin.org/uuid/404")
     .then(function(response) {
       console.log(response)
       if (!response.ok) {
         console.log("Response not okay");…
<- Promise { <state>: "pending" }
GET https://httpbin.org/uuid/404                       [HTTP/2 404 Not Found 385ms]
Response { type: "basic", url: "https://httpbin.org/uuid/404",      debugger eval code:3:21
redirected: false, status: 404, ok: false, statusText: "Not Found",
headers: Headers, body: ReadableStream, bodyUsed: false }
Response not okay                                                   debugger eval code:5:17
IN catch Error: HTTP error, status = 404                            debugger eval code:16:22
    <anonymous>      debugger eval code:7
    promise callback* debugger eval code:2
```



Not Found

The requested URL was not found on the server. If you entered the URL manually please check your spelling and try again.



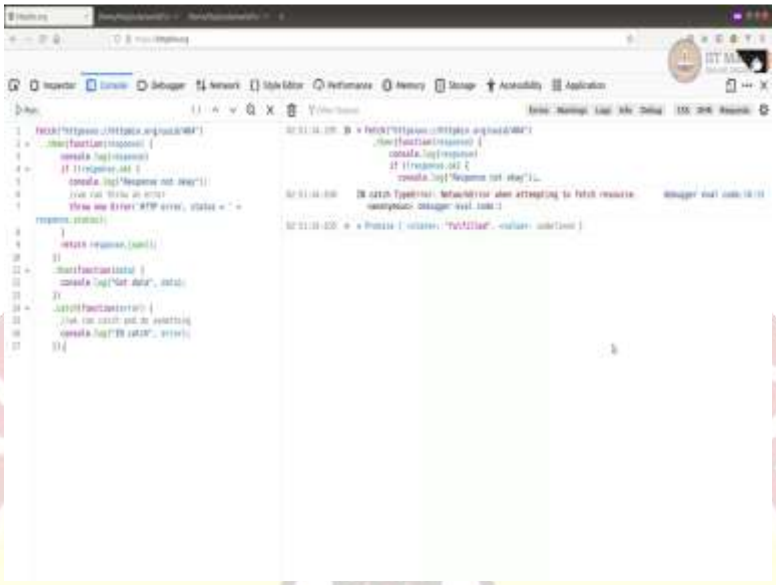
```
fetch("https://httpbin.org/status/404")
  .then(function(response) {
    console.log(response)
    if (!response.ok) {
      console.log("Response not okay");
      //we can throw an error
      throw new Error("HTTP error, status = " +
response.status);
    }
    return response.json();
  })
  .then(function(data) {
    console.log("Got data", data);
  })
  .catch(function(error) {
    //we can catch and do something
    console.log("IN catch", error);
  });
```

```
» fetch("https://httpbin.org/status/404")
    .then(function(response) {
      console.log(response)
      if (!response.ok) {
        console.log("Response not okay");…
← ▸ Promise { <state>: "pending" }
▸ GET https://httpbin.org/status/404                [HTTP/2 404 Not Found [illegible]ms]
▾ Response { type: "basic", url: "https://httpbin.org/status/404",      debugger eval code:3:[illegible]
redirected: false, status: 404, ok: false, statusText: "Not Found",
headers: Headers, body: ReadableStream, bodyUsed: false }
  ▸ body: ReadableStream { locked: false }
    bodyUsed: false
  ▸ headers: Headers { … }
    ok: false
    redirected: false
    status: 404
    statusText: "Not Found"
    type: "basic"
    url: "https://httpbin.org/status/404"
  ▸ <prototype>: ResponsePrototype { clone: clone(), arrayBuffer:
  arrayBuffer(), blob: blob(), … }
Response not okay                                      debugger eval code:5:[illegible]
IN catch Error: HTTP error, status = 404               debugger eval code:[illegible]
    <anonymous>       debugger eval code:7
    promise callback* debugger eval code:2
```

Let me paste a piece of code. Here I am actually calling an URL which is not found, which is a 404 URL, then once it is called then I get a promise, I am checking the response. I am actually checking here whether response is not okay, then I am actually throwing an error. Or if I am not, I can just do something else. But here, if the response is not okay, then I am throwing an error. If the response is okay, then I am, actually I am getting response dot json. If the response dot json gets called, then it goes here into this part. If this error throw happens, then it goes into this part and prints this. Let's run this.

Let's say the API was called, the request was sent to XHR, let us expand this and you can see the response was not found. So, 404 was thrown. But in our response, you can see that it has actually not failed, it has resolved. So, you can see status is 404. We could have checked for that also. But we just checked okay and not okay. Okay is false. So, it comes here. And says response not okay, just printed here, then it actually says HTTP error status is equal to 404, which we are printing here, and then throws an error you know. So, this error has been caught by this catch. And hence it goes into in catch and the logging of the error. So, this is how the error is handled in case you have a 404 or 505 and that kind of HTTP errors. The network error, if there was a network error, it would directly catch it here.

(Refer Slide Time: 08:38)

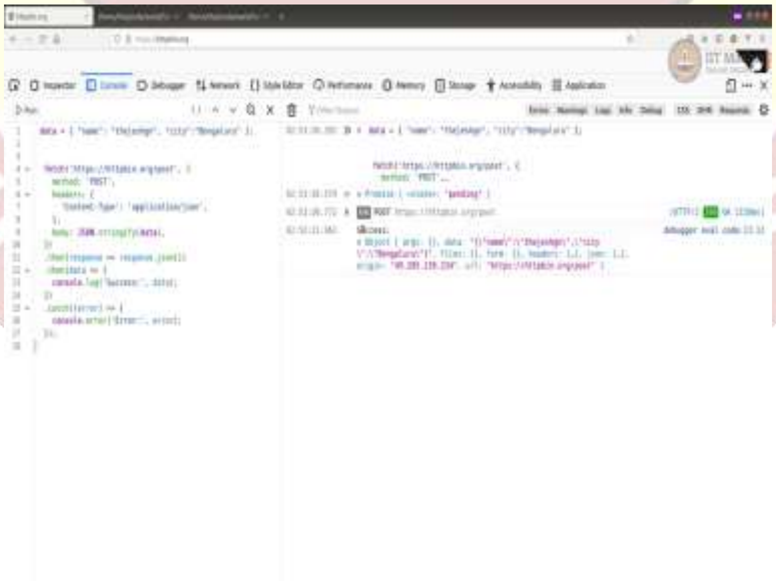


```
fetch("htttpsss://httpbin.org/status/404")
  .then(function(response) {
    console.log(response)
    if (!response.ok) {
      console.log("Response not okay");
      //we can throw an error
      throw new Error('HTTP error, status = ' +
response.status);
    }
    return response.json();
  })
  .then(function(data) {
    console.log("Got data", data);
  })
  .catch(function(error) {
    //we can catch and do something
    console.log("IN catch", error);
  });
```

```
[illegible] >> ▸ fetch("htttpsss://httpbin.org/status/404")
                   .then(function(response) {
                     console.log(response)
                     if (!response.ok) {
                       console.log("Response not okay");…
[illegible]      IN catch TypeError: NetworkError when attempting to fetch resource.      debugger eval code:16:11
                   <anonymous> debugger eval code:1
[illegible] ← ▸ Promise { <state>: "fulfilled", <value>: undefined }
```

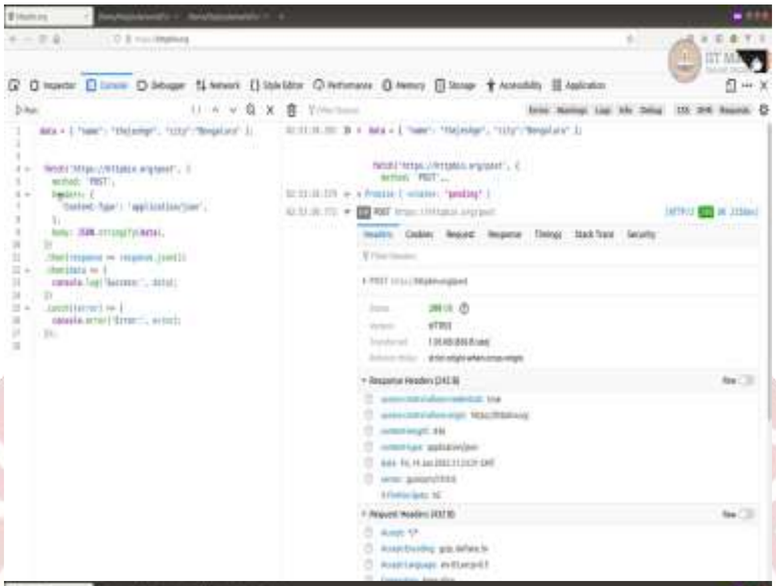
Let us do something else. I am just going to give completely wrong URL and see if it throws a network error. See here. You can already see in catch, it has already thrown network error. So, network error is thrown, but HTTP error actually comes into resolves and then we have to catch it and throw it. So, that is how you handle errors. In case you if you are making async call without an await word. We will see how to use await with respect to fetch at the end.

(Refer Slide Time: 09:17)

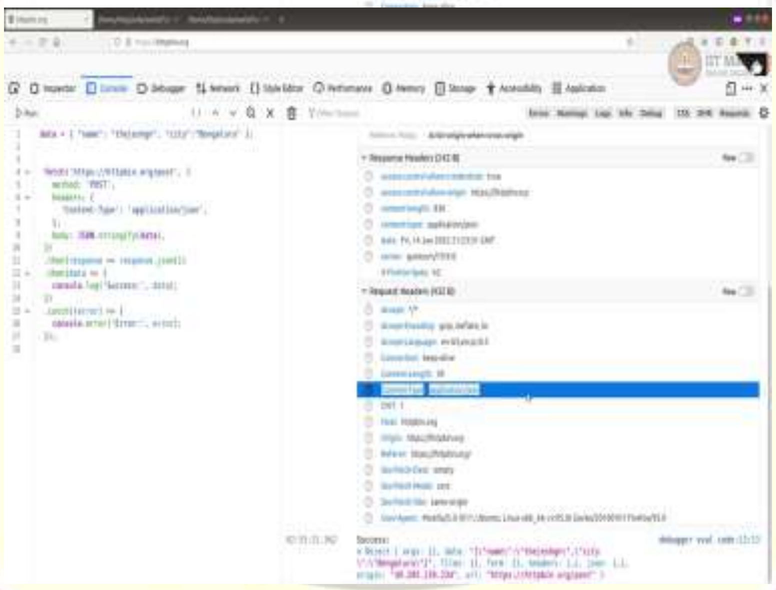


```
data = { "name": "Thejeshgn", "city":"Bengaluru" };


fetch("https://httpbin.org/post", {
  method: 'POST',
  headers: {
    'Content-Type': 'application/json',
  },
  body: JSON.stringify(data),
})
.then(response => response.json())
.then(data => {
  console.log('Success:', data);
})
.catch((error) => {
  console.error('Error:', error);
});
}
```

```
[illegible] >> data = { "name": "Thejeshgn", "city":"Bengaluru" };


                  fetch("https://httpbin.org/post", {
                    method: 'POST'…
[illegible] ← ▸ Promise { <state>: "pending" }
[illegible]   ▸ XHR POST https://httpbin.org/post                 [HTTP/1.1 200 OK 1139ms]
[illegible]     Success:                                          debugger eval code:13:11
                ▸ Object { args: {}, data: "{\"name\":\"Thejeshgn\",\"city
                \":\"Bengaluru\"}", files: {}, form: {}, headers: {…}, json: {…},
                origin: "49.205.139.234", url: "https://httpbin.org/post" }
```



```
data = { "name": "thejeshgn", "city":"Bengaluru" };


fetch("https://httpbin.org/post", {
  method: 'POST',
  headers: {
    'Content-Type': 'application/json',
  },
  body: JSON.stringify(data),
})
.then(response => response.json())
.then(data => {
  console.log('Success:', data);
})
.catch((error) => {
  console.error('Error:', error);
});
```



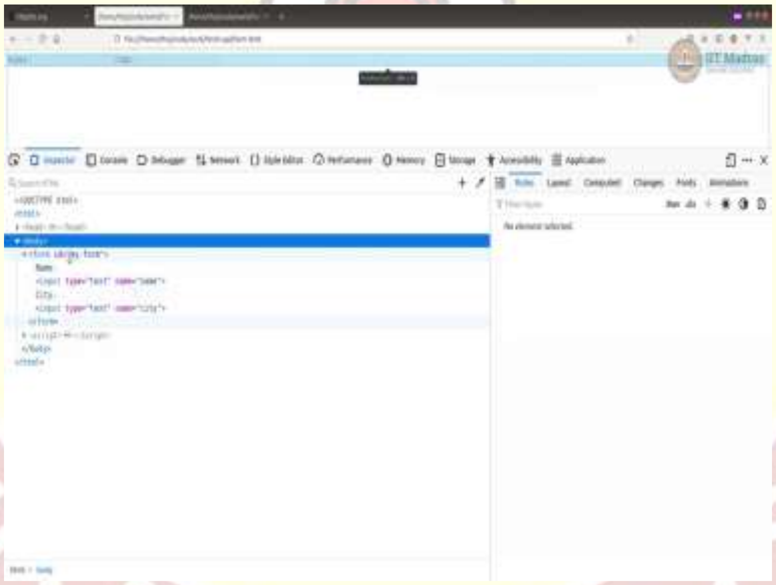
```
data = { "name": "thejeshgn", "city":"Bengaluru" };


fetch("https://httpbin.org/post", {
  method: 'POST',
  headers: {
    'Content-Type': 'application/json',
  },
  body: JSON.stringify(data),
})
.then(response => response.json())
.then(data => {
  console.log('Success:', data);
})
.catch((error) => {
  console.error('Error:', error);
});
```

Now, you can also send json data using fetch. It is pretty straightforward. We need to do some small changes. I will show what are those. Just give me 1 second. Here I have a json object, data, I am passing name and city, then I am making a post. Here in this specific case, I am actually adding an extra object with the parameter set where method is equal to post. I am setting header content is equal to application json. This is very important if you are actually sending the json data.

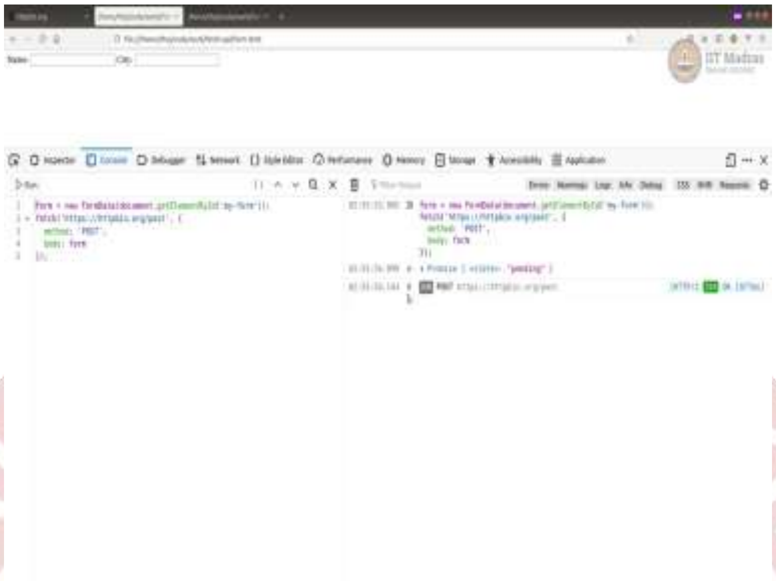
And the body we have to send which will have to contain the data, but this data has to be stringified. It has to convert this object into a string, a json string, and then set it to body to be sent. Hence, setting these headers is very, very important. If you do not set the headers, it would not know that you are sending a json data. Then I am just looking for the response.

If the response is, I am just printing here, I am not checking status okay, all of that. You should actually do it in when you are doing and then I am printing success and errors if there were. So, let us run this. Post is happening, post was sent and success. Now, let’s check here, headers. So, you can see that it was post what we said here, and then we will say request headers, we will check the requested headers, you can check content type application json, and then request body also you can see here. This is the request body you can make it raw, if you want to. Here is the request body that we are sending. Then whatever the response httpbin got, usually it sends the same response back. So, you can see that it has sent the same json back you know to us. So, this is how you send a json data, it can be post, it can be put as well.

(Refer Slide Time: 11:40)



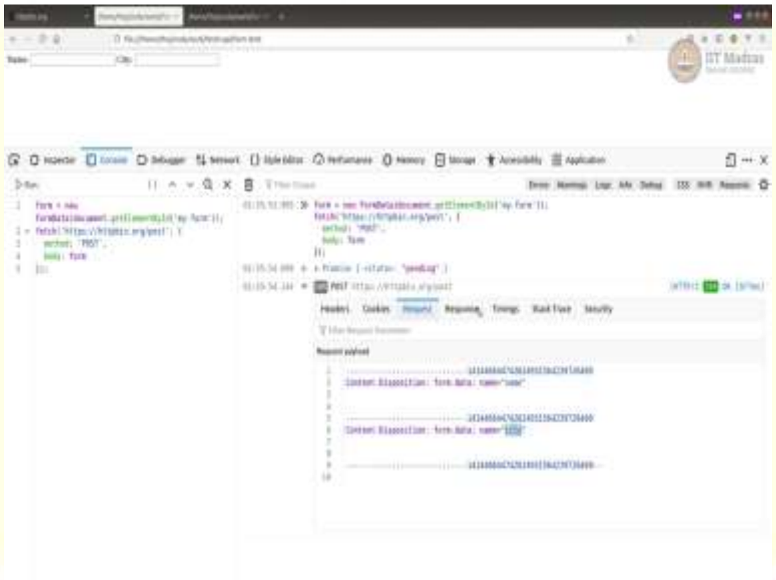
```
<!DOCTYPE html>
<html>
 <head>…</head>
 <body>
  <form id="my_form">
    Name
    <input type="text" name="name">
    City
    <input type="text" name="city">
  </form>
  <script>…</script>
 </body>
</html>
```



Name: City:

```
form = new FormData(document.getElementById('my-form'));
fetch('https://httpbin.org/post', {
  method: 'POST',
  body: form
});
```

```
[illegible] » form = new FormData(document.getElementById('my-form'));
              fetch('https://httpbin.org/post', {
                method: 'POST',
                body: form
              });
[illegible] ← ▸ Promise { <state>: "pending" }
[illegible] ▸ [illegible] POST https://httpbin.org/post        [HTTP/1.1 200 OK [illegible]ms]
```



Name: City:

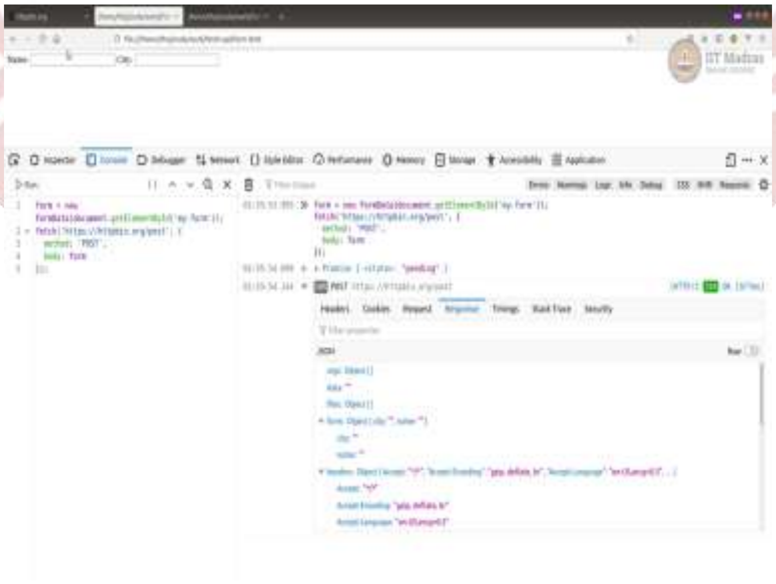
```
form = new FormData(document.getElementById('my-form'));
fetch('https://httpbin.org/post', {
  method: 'POST',
  body: form
});
```

```
[illegible] » form = new FormData(document.getElementById('my-form'));
              fetch('https://httpbin.org/post', {
                method: 'POST',
                body: form
              });
[illegible] ← ▸ Promise { <state>: "pending" }
[illegible] ▾ [illegible] POST https://httpbin.org/post        [HTTP/1.1 200 OK [illegible]ms]
```

Headers Cookies Request Response Timings Stack Trace Security

Request payload

```
-----------------------------[illegible]
Content-Disposition: form-data; name="name"


-----------------------------[illegible]
Content-Disposition: form-data; name="city"


-----------------------------[illegible]--
```



Name: City:

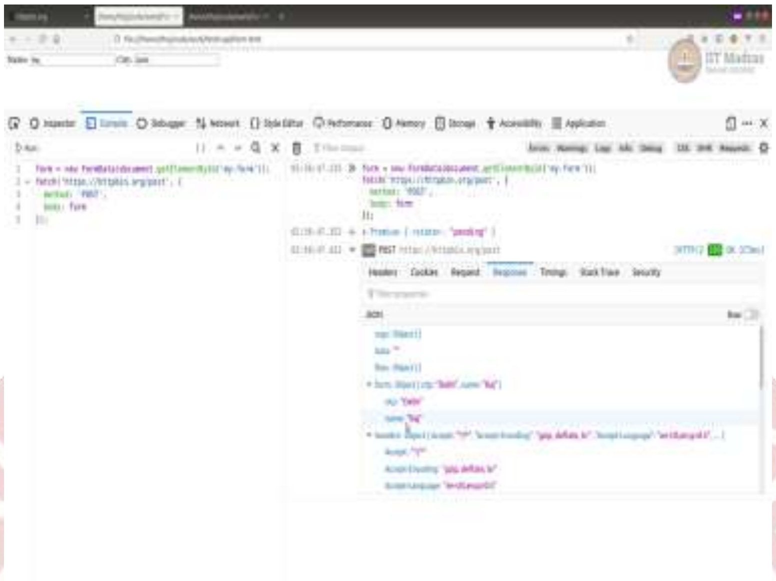
```
form = new FormData(document.getElementById('my-form'));
fetch('https://httpbin.org/post', {
  method: 'POST',
  body: form
});
```

```
[illegible] » form = new FormData(document.getElementById('my-form'));
              fetch('https://httpbin.org/post', {
                method: 'POST',
                body: form
              });
[illegible] ← ▸ Promise { <state>: "pending" }
[illegible] ▾ [illegible] POST https://httpbin.org/post        [HTTP/1.1 200 OK [illegible]ms]
```

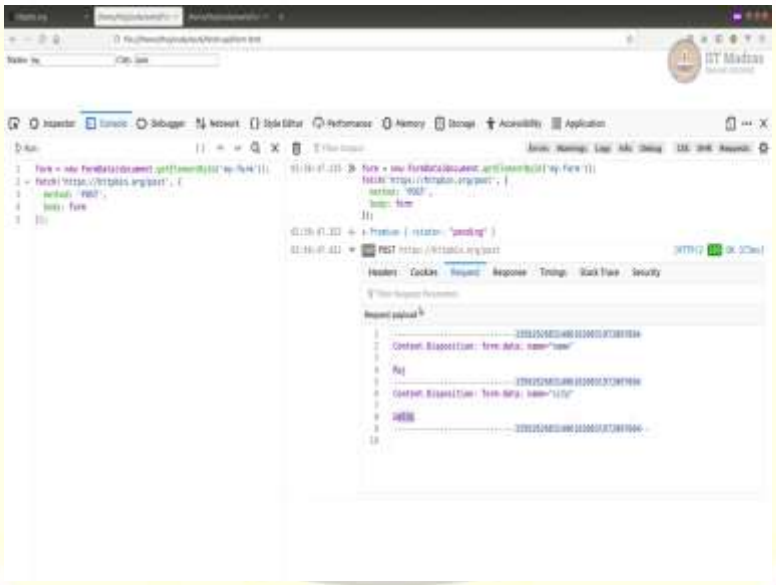
Headers Cookies Request Response Timings Stack Trace Security

JSON

```
args: Object {}
data: ""
files: Object {}
▾ form: Object { city: "", name: "" }
    city: ""
    name: ""
▾ headers: Object { Accept: "*/*", "Accept-Encoding": "gzip, deflate, br", "Accept-Language": "en-US,en;q=0.5", … }
    Accept: "*/*"
    Accept-Encoding: "gzip, deflate, br"
    Accept-Language: "en-US,en;q=0.5"
```



```
form = new FormData(document.getElementById('my-form'));
fetch('https://httpbin.org/post', {
  method: 'POST',
  body: form
});
```



```
form = new FormData(document.getElementById('my-form'));
fetch('https://httpbin.org/post', {
  method: 'POST',
  body: form
});
```

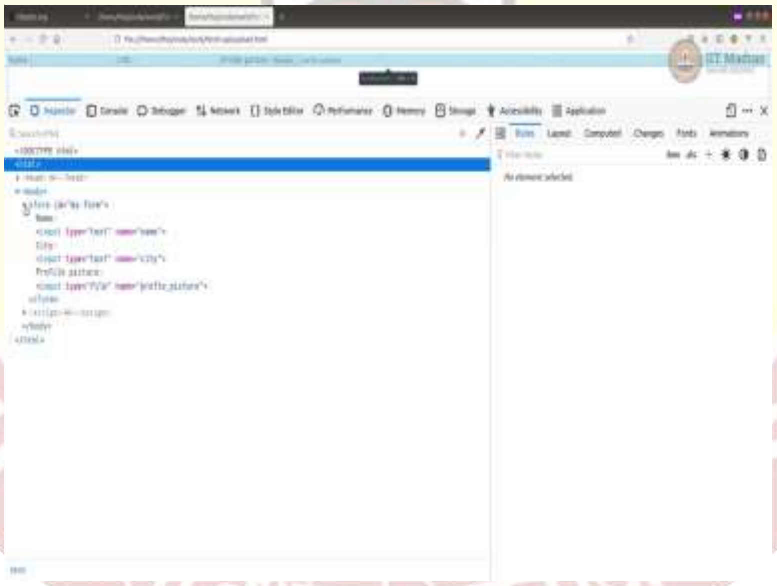
Now, you can also do form submissions using the fetch. Let us see a form, a simple form. Here I have a simple form which has name and city. I can check the HTML here. So, it is a simple form. You can see it here. There is no script, nothing. This is something else, do not worry. Name and city. Name is a tech type text, city of type text, name of the variable is or the parameter is city, name of this is also name. You can also see that. I have ID set for my form. So, I can get the reference to it.

Now, let us go to console and get the reference to form. So, there is form data object that you can create by giving a reference to a form. This you know is a reference to form by ID. Here is my form by ID. And then I am passing that form as a body to my fetch and just making a post request. I am not even waiting for the response here. I am just posting it. This is like a fire and forget. I do not worry about what response I am getting. Now, let’s run. So, it made a post. You can see that 200 was written which is, which means success. We can check what all was sent.

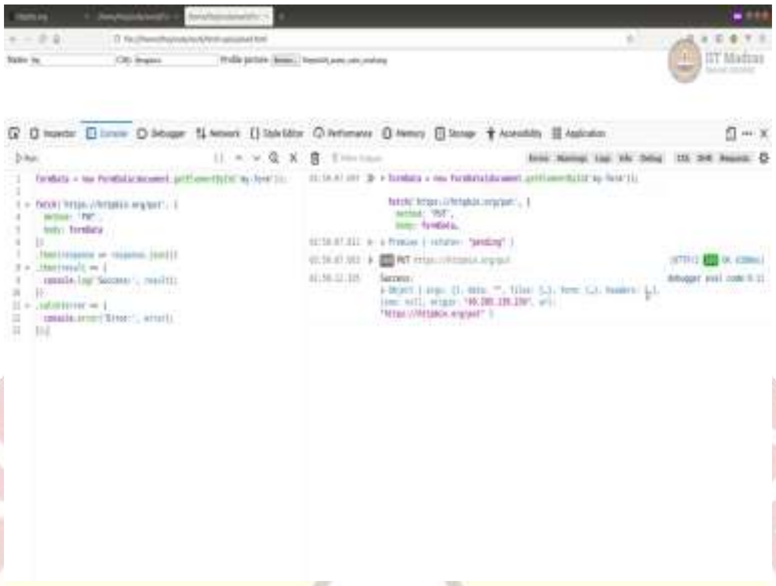
So, here you can see let us say request. Request you as you know form is sent as part of the body. So you can see in the body the name is sent, the city is also sent. You can see the response, json is null. You can see form object is empty. Let’s fill something and send, Raj, Delhi. Now, let’s run this again. Let me clear this, run this again. Sent response has been got.

Now, we can see, you can see the form object here the response, but we will see request also. Here you can see as part of the body you can see it is been set to Raj and Delhi the request has been sent. You can check the headers too and see the post and the other details. And in the response httpbin since the same form object back. You can see that the form object has been sent back with city as Delhi and name as Raj.

(Refer Slide Time: 14:41)

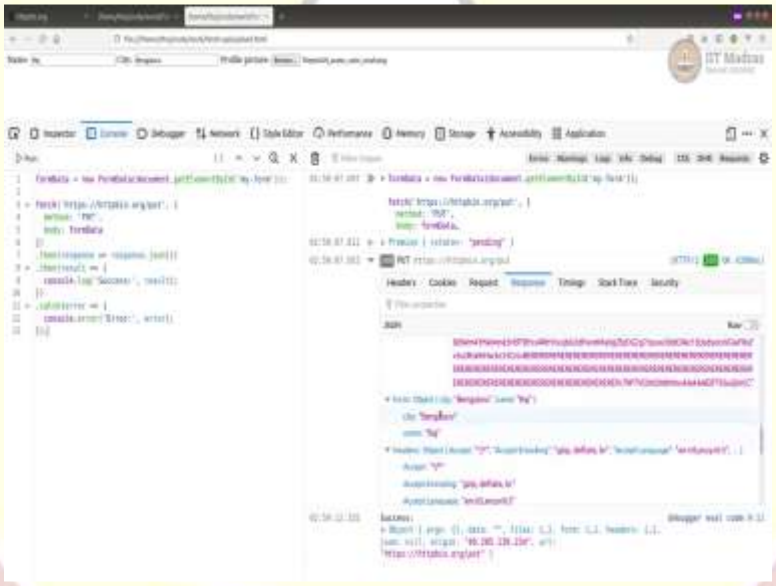


```
<!DOCTYPE html>
<html>
  <head>…</head>
  <body>
    <form id="my_form">
      Name:
      <input type="text" name="name">
      City:
      <input type="text" name="city">
      Profile picture:
      <input type="file" name="profile_picture">
    </form>
    <script>…</script>
  </body>
</html>
```



```
formData = new FormData(document.getElementById('my-form'));

fetch('https://httpbin.org/put', {
  method: 'PUT',
  body: formData
})
.then(response => response.json())
.then(result => {
  console.log('Success:', result);
})
.catch(error => {
  console.error('Error:', error);
});
```

```
Promise { <state>: "pending" }
PUT https://httpbin.org/put                    [HTTP/1.1 200 OK]
Success:
▶ Object { args: {}, data: "", files: {…}, form: {…}, headers: {…},
json: null, origin: "[illegible]", url:
"https://httpbin.org/put" }
```

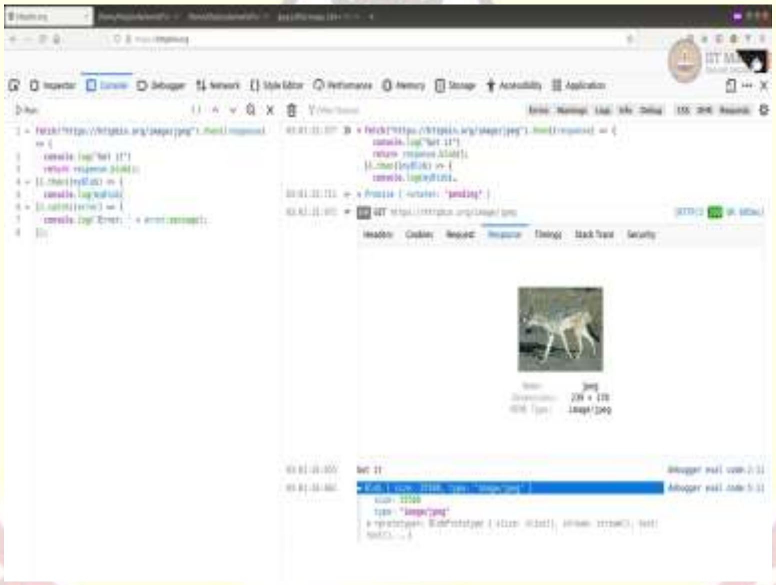


```
▼ form: Object { city: "Bengaluru", name: "Raj" }
    city: "Bengaluru"
    name: "Raj"
```

Similarly, you can do uploading files as well. Here I have a form with three elements with the same ID form, my form and three elements; one is name, city and a profile picture. But the profile picture is of type file. So, since it is of type file, you can actually select a file from your local files system. So, I can do Raj or Bengaluru and you can select some profile picture, selected some profile picture, and then I can submit it. Let me just, so here is the code. So, I am getting a reference to my form, like I said, and then I am trying to do a put, you can also do post to this URL endpoint and sending the form data as part of form body of the fetch, now, body of the request when I am calling a fetch API.

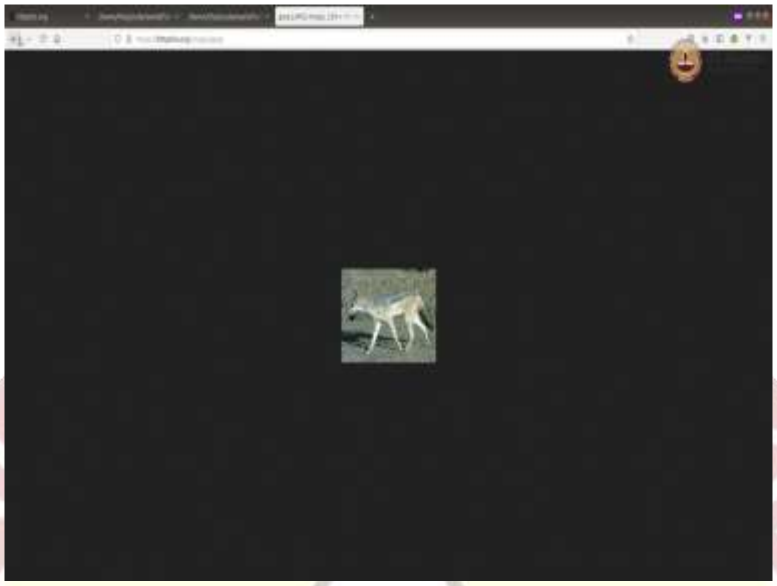
My form has three elements; one file type; and two regular text types. And I am waiting for the response, printing the response, whatever I am getting. So, let us run. It’s been sent. It took some time as you can see that is because it is uploading this whole image to the server. And depending on the size of the image, it might take more time. Let us check this thing. You can see that form contents attribute. This is like a basic for data type. And then other objects from the form, attributes from the form Bengaluru and Raj, it is all been sent as part of body and as part of response we are getting, as part of requests also it was sent, just hold on, took a while second to say, because it is a binary data. And we got response. And this is all the base 64 data. So, that is what httpbin sent. You can check in the headers, all the details as well what all we send with some multi-part form data. This is how you upload file. It is pretty straightforward pretty clean.

(Refer Slide Time: 17:13)



```
fetch("https://httpbin.org/image/jpeg").then((response)
=> {
  console.log("Got it")
  return response.blob();
}).then((myBlob) => {
  console.log(myBlob);
}).catch((error) => {
  console.log("Error: " + error.message);
});
```

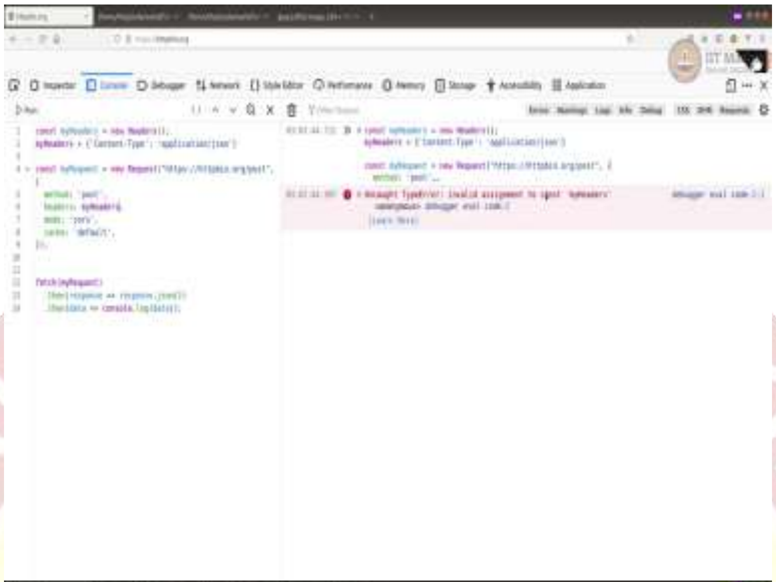
```
Promise { <state>: "pending" }
GET https://httpbin.org/image/jpeg
Got it
Blob { size: [illegible], type: "image/jpeg" }
  size: [illegible]
  type: "image/jpeg"
```



Now, we can also download the file or you can also download the blob format. For example, I am trying to download a picture from URL endpoint. This URL endpoint has a picture. If I go here, it should show a picture. Yeah, it has a picture. Now, I can also get it asynchronously using fetch. I am getting a picture. I printing a message got it. And this picture is of type blob. And I am just log in the blob.

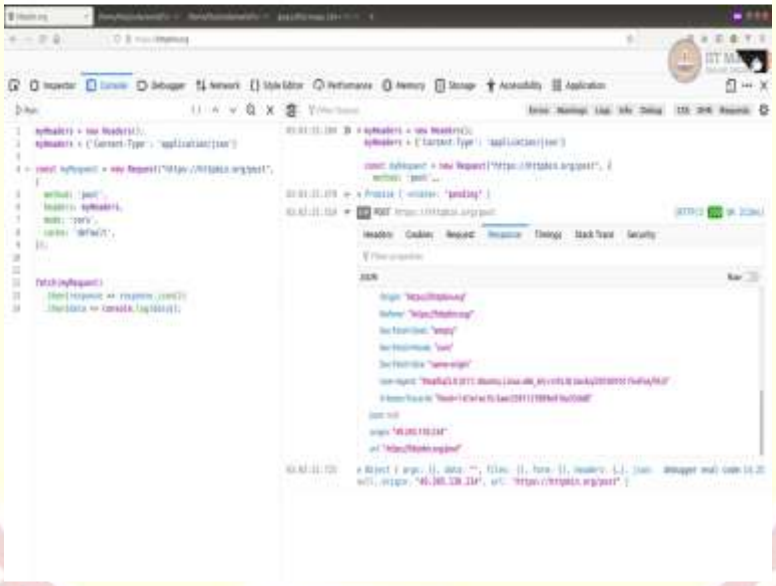
You could do many other things. You can create a URL out of it. You can create a element and set the image source as this blob URL and then make it appear in the page. But here I am just going to log it to console. Let me just run. So, you can see that we have got a blob image, and the size is 35k. It is of type image slash jpeg. You can also see that here if you see the raw data. You can see that is the image that we have got as part of the response. That is how you get a blob.

(Refer Slide Time: 18:35)



```
const myHeaders = new Headers();
myHeaders = {'Content-Type': 'application/json'}

const myRequest = new Request("https://httpbin.org/post",
{
  method: 'post',
  headers: myHeaders,
  mode: 'cors',
  cache: 'default',
});


fetch(myRequest)
  .then(response => response.json())
  .then(data => console.log(data));
```

```
» const myHeaders = new Headers();
  myHeaders = {'Content-Type': 'application/json'}

  const myRequest = new Request("https://httpbin.org/post", {
    method: 'post'…
❶ ▸ Uncaught TypeError: invalid assignment to const 'myHeaders'          debugger eval code:2:1
    <anonymous> debugger eval code:2
    [Learn More]
```



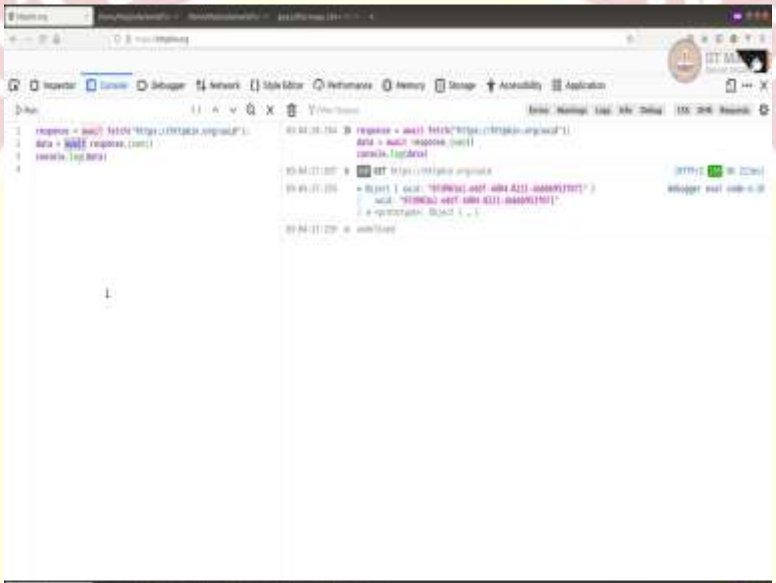
```
myHeaders = new Headers();
myHeaders = {'Content-Type': 'application/json'}

const myRequest = new Request("https://httpbin.org/post",
{
  method: 'post',
  headers: myHeaders,
  mode: 'cors',
  cache: 'default',
});


fetch(myRequest)
  .then(response => response.json())
  .then(data => console.log(data));
```

```
» myHeaders = new Headers();
  myHeaders = {'Content-Type': 'application/json'}

  const myRequest = new Request("https://httpbin.org/post", {
    method: 'post'…
← ▸ Promise { <state>: "pending" }
▾ [illegible] POST https://httpbin.org/post                          [HTTP/2 200 OK [illegible]]
    Headers  Cookies  Request  Response  Timings  Stack Trace  Security
    JSON
      Origin: "https://httpbin.org"
      Referer: "https://httpbin.org/"
      [illegible]
    json: null
    origin: "[illegible]"
    url: "https://httpbin.org/post"
▸ Object { args: {}, data: "", files: {}, form: {}, headers: {…}, json:     debugger eval code:14:25
  null, origin: "[illegible]", url: "https://httpbin.org/post" }
```

Now, you can also instead of sending this separately URL and parameters, you can also send the request and header object created beforehand. For example, here, I am creating headers which has application dot json, then I am actually creating a request object with those headers and other details and then I am making a call. Let me just replace this URL with real httpbin URL so that we can just do send it and see if it works. We can do post and we can send that data. So, let's remove this constant, because you cannot change. I just have to reload because I cannot redefine that same constant again.

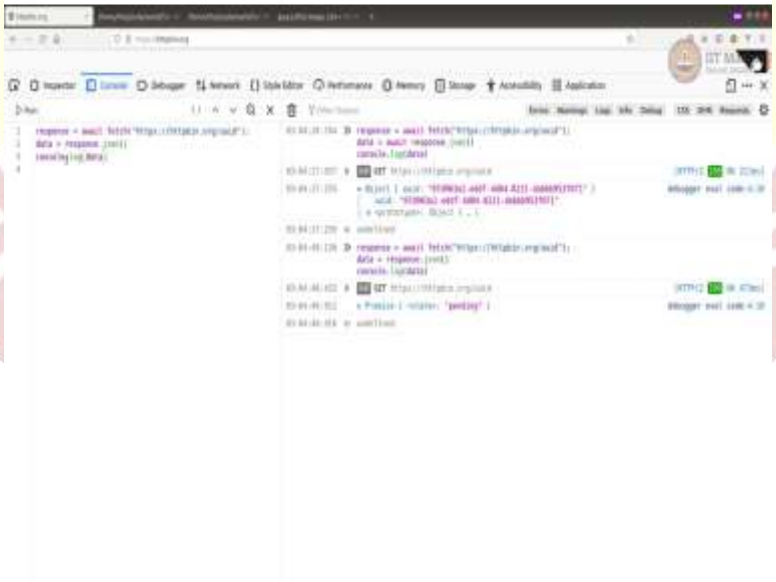
Now, if I run it, since I am using same console window again and again, I am just trying to not to make them constant, because you can't change the values. Anyway, here you can see that it is been posted and it works. So, now, if you are a lot of cases where you are going to repeat the same headers or repeat the same request, you can pre-create them and keep reusing them for various fetch requests. So, that is how you create headers and request to reuse them again and again.

(Refer Slide Time: 20:31)



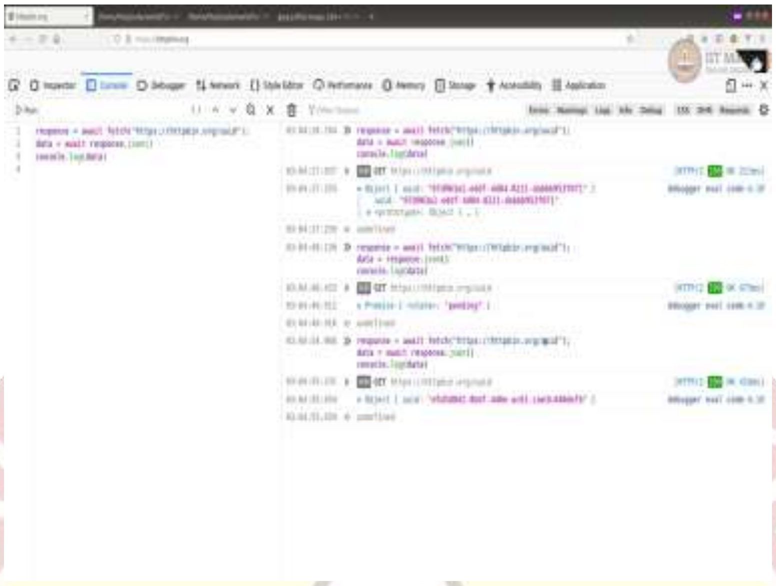
```
response = await fetch("https://httpbin.org/uuid");
data = await response.json()
console.log(data)
```

```
» response = await fetch("https://httpbin.org/uuid");
  data = await response.json()
  console.log(data)
GET https://httpbin.org/uuid
▸ Object { uuid: "[illegible]" }
    uuid: "[illegible]"
  ▸ <prototype>: Object { … }
← undefined
```



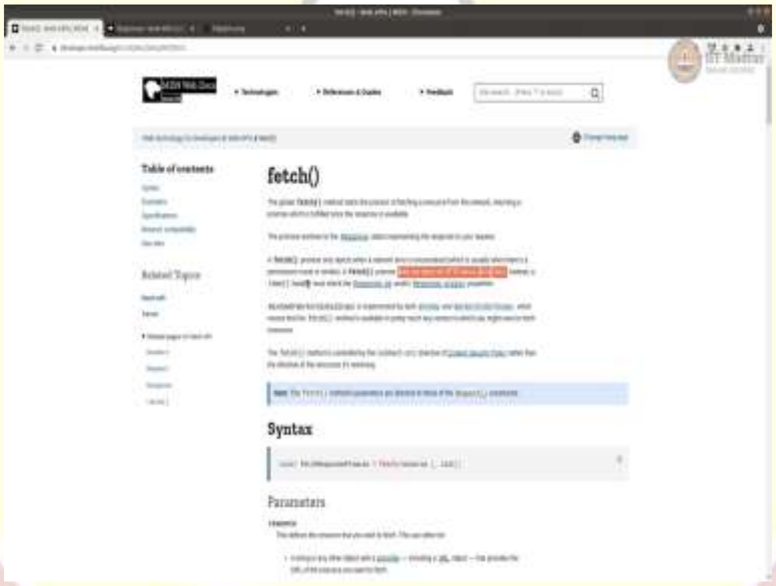
```
response = await fetch("https://httpbin.org/uuid");
data = response.json()
console.log(data)
```

```
» response = await fetch("https://httpbin.org/uuid");
  data = await response.json()
  console.log(data)
GET https://httpbin.org/uuid
▸ Object { uuid: "[illegible]" }
    uuid: "[illegible]"
  ▸ <prototype>: Object { … }
← undefined
» response = await fetch("https://httpbin.org/uuid");
  data = response.json()
  console.log(data)
GET https://httpbin.org/uuid
▸ Promise { <state>: "pending" }
← undefined
```



```
response = await fetch("https://httpbin.org/uuid");
data = await response.json()
console.log(data)
```

```
[illegible] » response = await fetch("https://httpbin.org/uuid");
              data = await response.json()
              console.log(data)
[illegible] ▸ GET https://httpbin.org/uuid                    [HTTP/2 200 OK [illegible]]
[illegible]   ▾ Object { uuid: "[illegible]" }                debugger eval code:[illegible]
                 uuid: "[illegible]"
               ▸ <prototype>: Object { … }
[illegible] ← undefined
[illegible] » response = await fetch("https://httpbin.org/uuid");
              data = response.json()
              console.log(data)
[illegible] ▸ GET https://httpbin.org/uuid                    [HTTP/2 200 OK [illegible]]
[illegible]   ▸ Promise { <state>: "pending" }                debugger eval code:[illegible]
[illegible] ← undefined
[illegible] » response = await fetch("https://httpbin.org/uuid");
              data = await response.json()
              console.log(data)
[illegible] ▸ GET https://httpbin.org/uuid                    [HTTP/2 200 OK [illegible]]
[illegible]   ▸ Object { uuid: "[illegible]" }                debugger eval code:[illegible]
[illegible] ← undefined
```



fetch()

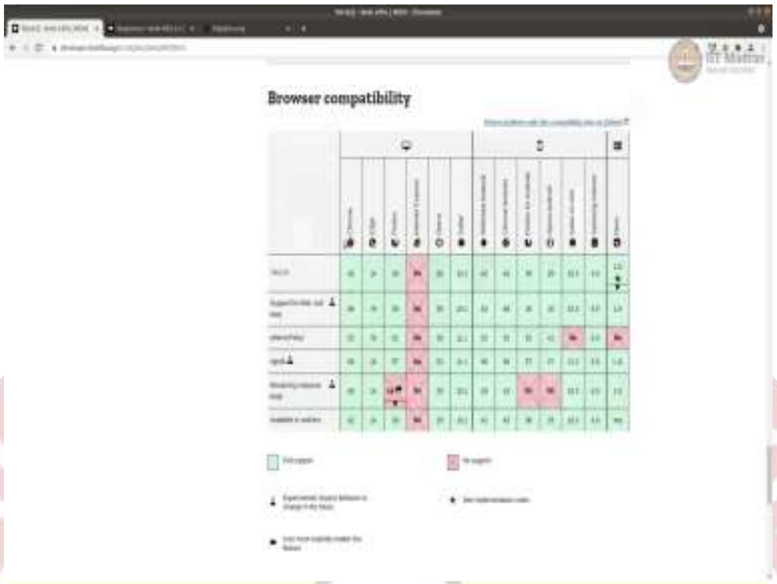
[illegible]

Syntax

```
[illegible]
```

Parameters

[illegible]



Browser compatibility

Now, we know that we can also use async, await instead of making it resolved in the promise manually by ourselves. So, like any other async function, you can add the keyword await. But here, it that returns a response. Response is also a promise. So, you have to add await for that as well and then you print a log data. So, when you run this, you can see that the data is printed directly. Now, if I do not do await here, and then print, this is a promise. Here, you can see that is a promise. Hence, we have to add await here too. So, there are at least usually two promises to be resolved before you can access the data. That’s all actually.

You can go to fetch API page on MDN web docs to read more about it. Also look at the chart for browser support details. At the end of the page it says what browser support almost all modern browsers support so you should be able to use it and use it very efficiently. And that is all for today's screencast. Have a good day. Thank you.