Session

We will first try to accept data. We can do it without form as well, but trying to save time, we will copy from the bootstrap-form.

```
<form>
  <div class="mb-3">
    <label for="exampleInputEmail1" class="form-label">Email address</label>
    <input type="email" class="form-control" id="exampleInputEmail1" aria-
describedby="emailHelp">
    <div id="emailHelp" class="form-text">We'll never share your email with anyone else.</div>
  </div>
  <div class="mb-3">
    <label for="exampleInputPassword1" class="form-label">Password</label>
    <input type="password" class="form-control" id="exampleInputPassword1">
  </div>
  <div class="mb-3 form-check">
    <input type="checkbox" class="form-check-input" id="exampleCheck1">
    <label class="form-check-label" for="exampleCheck1">Check me out</label>
  </div>
  <button type="submit" class="btn btn-primary">Submit</button>
</form>
```

The final thing is to link up bootstrap-css. We can download the css file from bootstrap, put inside the css folder and got it from the server, but ‘cdn’ is a preferred way.

We will put a div element with container class and inside it we will put another div element with,

div.d-flex.justify-content-center.align-items-center

```
<div class="container">
    <div class="d-flex justify-content-center align-content-center">
        <form action="/signIn">
```

```
            <div class="mb-3">
                <label for="exampleInputEmail1" class="form-label">Email address</label>
                <input type="email" class="form-control" id="exampleInputEmail1" aria-
describedby="emailHelp">
                <div id="emailHelp" class="form-
text">We'll never share your email with anyone else.</div>
            </div>

            <div class="mb-3">
                <label for="exampleInputPassword1" class="form-label">Password</label>
                <input type="password" class="form-control" id="exampleInputPassword1">
            </div>

            <div class="mb-3 form-check">
                <input type="checkbox" class="form-check-input" id="exampleCheck1">
                <label class="form-check-label" for="exampleCheck1">Check me out</label>
            </div>

            <button type="submit" class="btn btn-primary">Submit</button>
        </form>

    </div>
</div>
```

Whenever we were able to define url inside form tag and it was sending a query parameter inside our url. In the same, we will first use this method. In form we will use action. So,

```
<form action="/signIn">
```

When we click on submit button, it goes to signIn but will not show any details as we have not defined the signIn

```
app.get('/signIn', (req, res) => {
    res.send('ASD')
})
```

HTML is able to get all the date and send it a particular route because we have defined it in our index.js file. We notice after question mark there is nothing. So, to solve that we will write in the input tag as

```
<input type="email" name="email" class="form-control" id="exampleInputEmail1" aria-
describedby="emailHelp">
```

And will do the same thing for the password as well,

```
<input type="password" name="password" class="form-control" id="exampleInputPassword1">
```

In the last input tag for the checkbox as well, we will put the attribute name,

```
<input type="checkbox" name="remember" class="form-check-input" id="exampleCheck1">
```

Now, let us give in some details in the form and we can see in the address bar,

localhost:3000/signIn?email=yashasvi.kumar.sinha%40gmail.com&pass=qw123123qasd&remember=on

So, this means that we can accept those requests inside the GET handler, for signIn. Let us console.log in the GET handler to see if we are able to get the query parameters. So, in the index.js file, we will add,

```
app.get('/signIn', (req, res) => {
    console.log(req.query)
    res.send('ASD')
})
```

In the terminal we can see,

```
{
    email: 'yashasvi.kumar.sinha%40gmail.com'
    pass: '123143234534534564643252 3
    remember: 'on'
}
```

Now that we have data logged into the server environment, we can use that data and what we do with that data, frontend will not be aware of it.

By default, all the data is in the GET and visible in the url. So to override something, in the form tag we can add, method="POST". The default value is GET so we have over written it to POST. Now if we check and give values and signIn, it will not show the query parameters in the URL, but will show an error message as **`Cannot POST /signIn`.** This error is show because we have not created a POST method. We will create a POST request for the same,

```
app.post('/signIn', (req, res) => {
    console.log(req.query)
    res.send('Post Route')
})
```

It is will give the output, but the query parameters would not be mentioned in the URL or in the terminal. So, to check if the data is there or not, we can write,

```
app.post('/signIn', (req, res) => {
    console.log(req.query)
    console.log(req.body)
    console.log(req.params)
    res.send('Post Route')
})
```

The data is still not visible. So, in order to fix this, we will be using a middleware.

```
const express = require('express')
const app = express();

app.use(express.static('/public'))
app.use(express.urlencoded({extended: false}))

app.get('/', (req, res) => {
    res.sendFile(__dirname + '/public/home.html')
```

```
})

app.get('/signIn', (req, res) => {
    console.log(req.query)
    res.send('ASD')
})

app.post('/signIn', (req, res) => {
    console.log(req.query)
    console.log(req.body)
    console.log(req.params)
    res.send('Post Route')
})

app.listen(3000, () => console.log('Server Started'))
```

We are getting the data in the req.body. We can see that in req.query and req.params are shown but are empty. We are getting data in POST route. So, if we have our form sending request or data to any of our server route, by default it is GET, we will get data as object in query parameters. If it changed to POST, then, it will automatically will try to fetch or send request to POST route, that we have define. In POST, our data will not be in req.query. In such case, we have to first define a middleware;

```
app.use(express.urlencoded({extended: false}))
```

This helps express to understand form related data in case of POST method. After this, once we add, we will get the entire form data, we will receive it inside our req.body.

When we signIn, we are getting redirected to another URL. We send some html data. For now, we will send, about.html file;

```
app.post('/signIn', (req, res) => {
    console.log(req.body)
    res.send(__dirname + '/public/about.html')
})
```

This is refreshing the page and the entire page is getting loaded again. If anytime in the browser, in the TAB, if you get a spinner thing, that means entire page is getting loading or is getting another set of page from the server. For a brief time there would be a spinner. This tells us that the entire page is new. Instead of actually having URL defined and browser sending data, we will keep user in the same page and will be sending data to the server and getting back response from server without ever reloading the page. In order to do that, we need to have browser JavaScript, something will be working inside the browser. So, in the home.html, we will link the browser.js using the script tag.

Inside the browser we will attach an eventListener on the form and when user clicks on submit button, that will initiate submit event. We will stop default behaviour and create our own .json and then send that to the route. In browser.js file,

```
const form = document.getElementByTagName('form')[0]

form.addEventListener('submit', handleSubmit)

function handleSubmit() {
    alert('Works)')
}
```

When we click on submit, the alert box is shown and once we click on ok, then it would take us to the about.html page.

```
const form = document.getElementByTagName('form')[0]

form.addEventListener('submit', handleSubmit)

function handleSubmit() {
    alert('Works)')
    console.log('details log')
}
```

As the name suggest, preventDefault() will prevent the submit event. Now, if we click on submit, the page is not being reloaded. After preventing the form from getting submitted, we will put console.log to see if it is working or not. When we click on submit, it logs in the console.

We can get these data and then use fetch() to send that data rather than refreshing the page. Now, we want to get the data that user as given. We will reference all the input tags, that gives you more control.

```
const form = document.getElementByTagName('form')[0]
const emailInput = document.getElementById('exampleInputEmail1')
const passInput = document.getElementById('exampleInputPassword1')

form.addEventListener('submit', handleSubmit)

function handleSubmit(e) {
    e.preventDefault()
    console.log(emailInput.value)
    console.log(passInput.value)
}
```

Now that we have the value, we have a request and then manually we will send it to req.body. Since, I'm going to be use fetch(), we will put async so that we can use await keyword.

```
const form = document.getElementByTagName('form')[0]
const emailInput = document.getElementById('exampleInputEmail1')
const passInput = document.getElementById('exampleInputPassword1')

form.addEventListener('submit', handleSubmit)

async function handleSubmit(e) {
    e.preventDefault()
    console.log(emailInput.value)
    console.log(passInput.value)
    const request = fetc('/signIn')
}
```

Fetch request is a GET request so will override it into a POST request. We can provide second parameter as object and it will receive all the data.

```
async function handleSubmit(e) {
    e.preventDefault()
    console.log(emailInput.value)
    console.log(passInput.value)

    const request = fetc('/signIn', { method: 'POST'})
}
```

We will get a response object. In this, we will get the html body. But the req.body is still empty. We want to send data, emailInput and passInput into json.

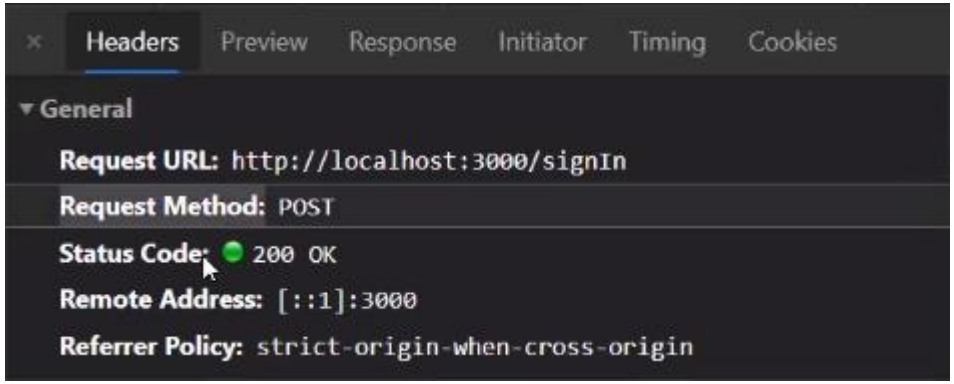
```
const form = document.getElementByTagName('form')[0]
const emailInput = document.getElementById('exampleInputEmail1')
const passInput = document.getElementById('exampleInputPassword1')

form.addEventListener('submit', handleSubmit)

async function handleSubmit(e) {
    e.preventDefault()
    console.log(emailInput.value)
    console.log(passInput.value)

    const userCred = {
        email: emailInput.value,
        password: passInput.value
    }

    const request = fetch('/signIn', { method: 'POST', body: userCred })
}
```

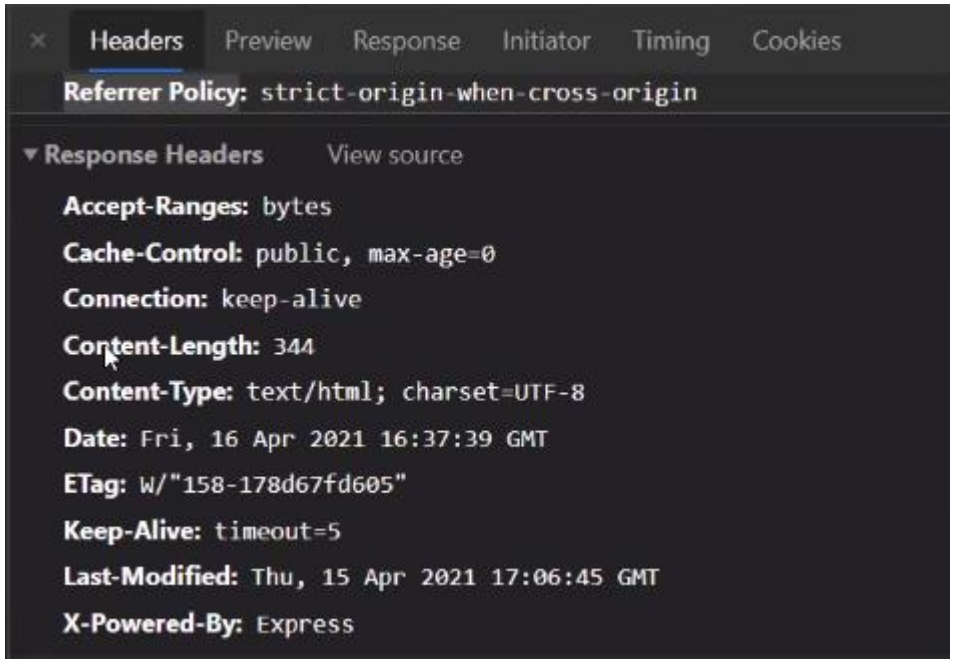
Like we had the urlencoded for the form element, express knows that all data is coming from a form and is put into an object. In order to accept the data, we will add another middleware, in the main.js

```
const form = document.getElementByTagName('form')[0]
const emailInput = document.getElementById('exampleInputEmail1')
const passInput = document.getElementById('exampleInputPassword1')

form.addEventListener('submit', handleSubmit)

async function handleSubmit(e) {
    e.preventDefault()
    console.log(emailInput.value)
    console.log(passInput.value)

    const userCred = {
        email: emailInput.value,
        password: passInput.value
    }

    const request = fetc('/signIn', {
        method: 'POST',
        body: JSON.stringify(userCred),
        headers: { 'Content-Type': 'applicaiton/json; charset=UTF=8'} })

    console.log(request)
}
```

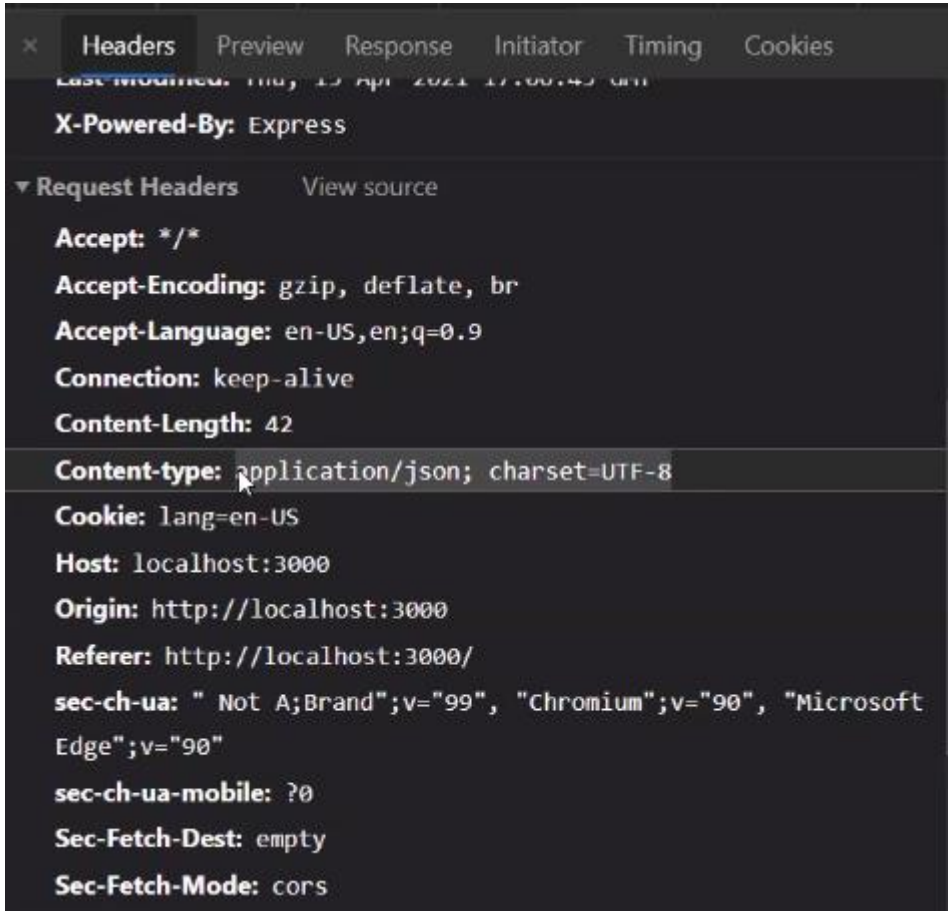
In the fetch if we want to create a post request, we will provide a method: ‘POST’ and in body whatever data that you want to send ot the server, we need to stringify your object, because we need to convert it and lot of things needs to be attached to the object behind the scenes andw e cannot send all this data, so we will convert it into string format, **JSON.stringify.** This is not object but a string and it is string which seems like an object. This string gets transferred through the internet. Then express will convert it into JSON because of the middleware, app.use(express.json()).

Now another part, once the body is defined, header; just data about, some more metadata about the entire request. We can see 3 sections of headers. These are all properties defined in the headers.





All the details from the server. In request header, we have defined the content-type as it was not defined properly. If we click on submit, we should be able to see the details in the terminal as well.



Now the data is inside the server, so we send this entire data through that http request but we have never left the particular request.

That is the idea of being heavily used. There will be change in URL, but that would not be an entire page load.

```
app.post('/signIn', (req, res) => {
    console.log(req.body)
    res.json{ success: true}
})
```

Right now we are getting the data inside the server but we are not able to read the data. In the response tab of network, we have data, that came from the server. To use this data,

In index.js

```
app.post('/signIn', (req, res) => {
    console.log(req.body)
    user.push(req.body)
    res.json({ error: 0})
    console.log(users)
})
```

We can see the details coming for req.body in an array format. If we signIn with different data, then we can see, the details getting appended to the user object.

But the user.json file itself is empty. the moment we turn off the server all the data is lost. So, in order to do file writing, we can append it to the file or write it directly into the file. We need to import the file system.

```
const express = require('express')
const app = express();
const user = require('./db/user.json');
const fs = require('fs');


app.use(express.static('/public'))
app.use(express.json())
app.use(express.urlencoded({extended: false}))

app.get('/', (req, res) => {
    res.sendFile(__dirname + '/public/home.html')
})

app.get('/signIn', (req, res) => {
    console.log(req.query)
    res.send('ASD')
})

app.post('/signIn', (req, res) => {
```

```
    console.log(req.body)
    user.push(req.body)
    res.json({ error: 0})
    console.log(users)
    fs.writeFileSync(__dirname + './db/users.json', users)
})

app.listen(3000, () => console.log('Server Started'))
```

In the output we can see an error; '**The "data" argument must be of type string or an instance of Buffer, TypeArray or DataView.**' We either need to convert it into binary format or string representation.

The best way to do is using JSON.stringify.

```
fs.writeFileSync(__dirname + './db/users.json', JSON.stringify(users))
```

This data would be saved in the users.json file. One thing we can notice is all the details would be coming in one line. This JSON.stringify, also accept node parameters. The second parameter is a replacer purpose, so in that we will say 'null' and the third parameter will be spacing, so that the it would be readable for us.

```
fs.writeFileSync(__dirname + './db/users.json', JSON.stringify(users, null, 4))
```

We are able to append the data we are getting from the front-end into the backend.

When the server is terminated, at that particular time, we will write everything inside the json file.