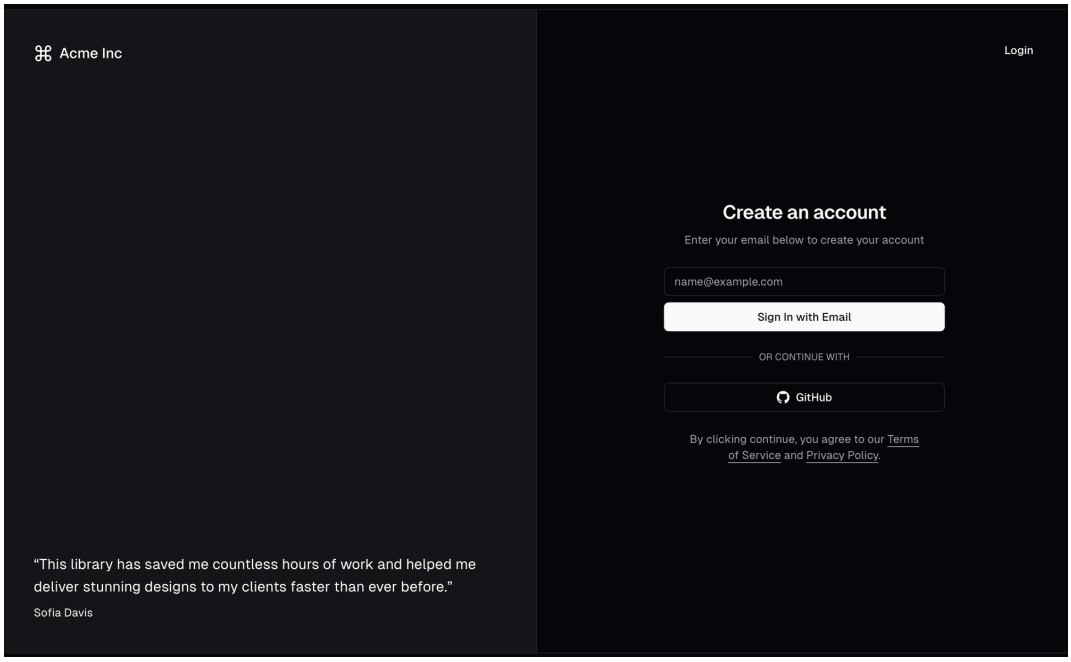
Auth using cookies

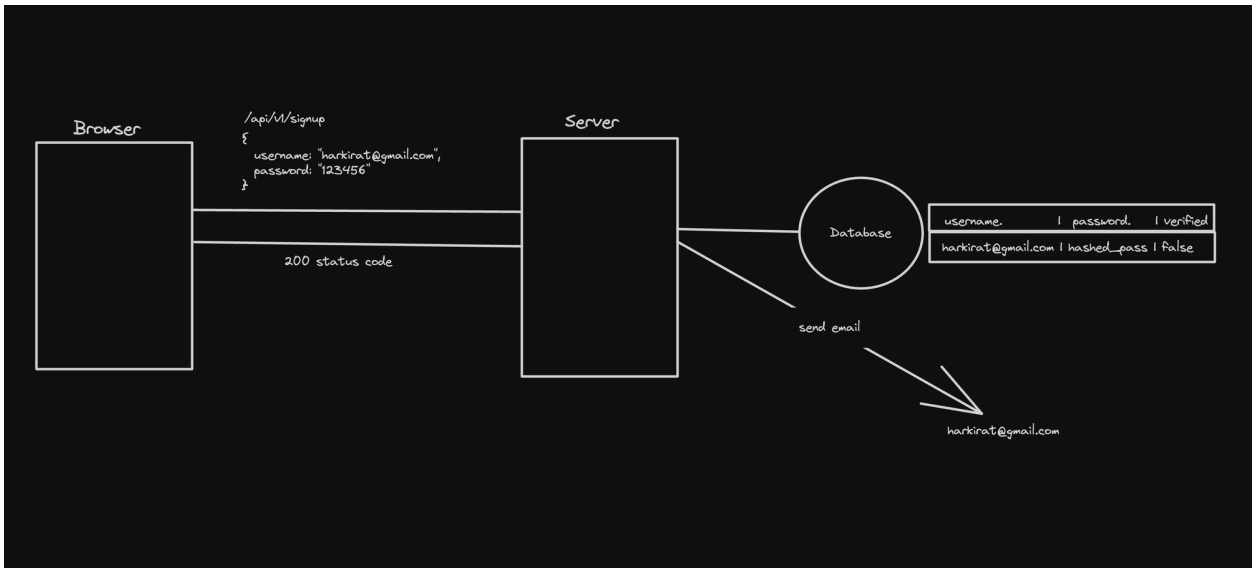
(https://projects.100xdevs.com/tracks/Auth/auth-1)

Authentication is the process of letting users signup/signin into websites via username / password or using SSO (single sign on)



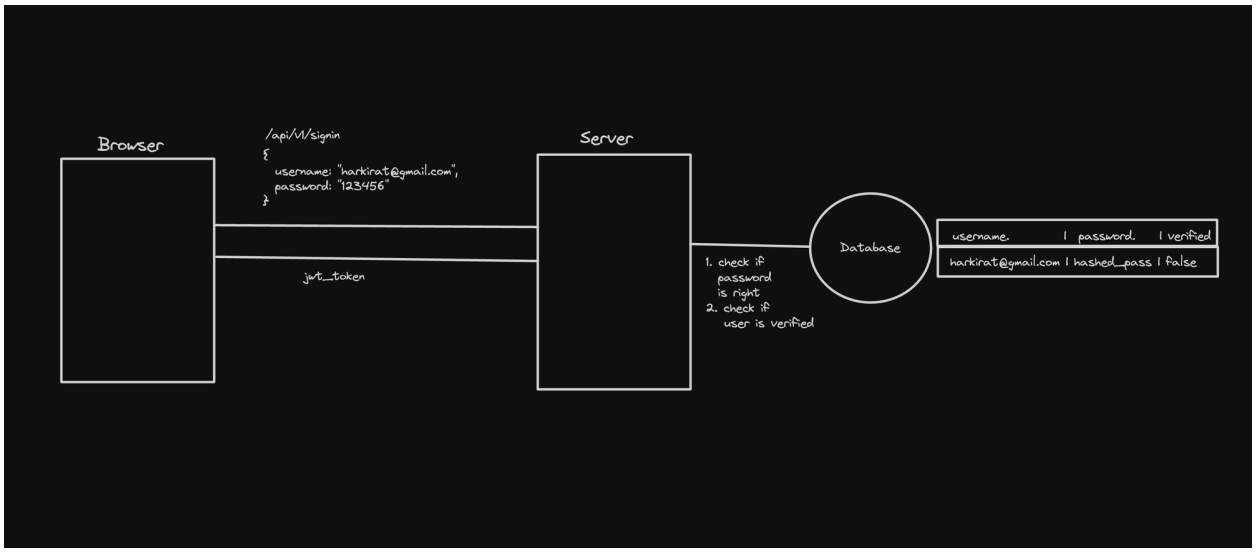
Authentication using jwt + localstorage

Signup



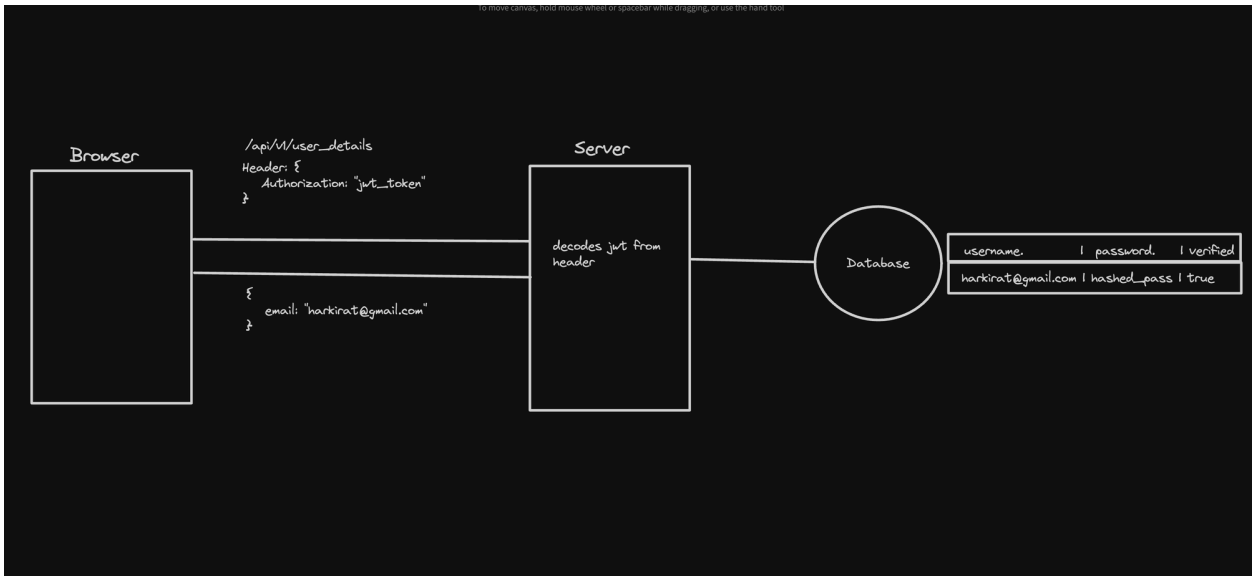
Send out a request username /etc , verify email so that people can’t use other email addresses.

Signin



User send a signin request

Auth request

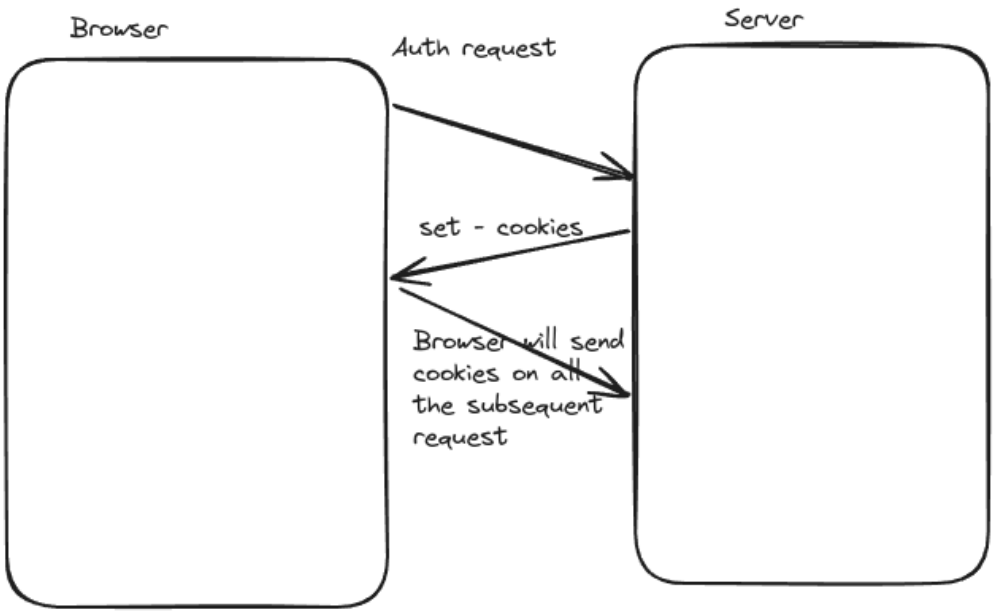


Authentication using cookies

What are cookies

Cookies in web development are small pieces of data sent from a website and **stored on the user's computer by the user's web browser while the user is browsing**. They are designed to be a reliable mechanism for websites to remember things (very similar to local storage)

1. **Session Management**: Cookies allow websites to identify users and track their individual session states across multiple pages or visits.
2. **Personalization**: Websites use cookies to personalize content and ads. For instance, cookies might store information about a user's preferences, allowing the site to tailor content or advertisements to those interests.
3. **Tracking**: Cookies can track users across websites, providing insights into browsing behavior. This information can be used for analytics purposes, to improve website functionality, or for advertising targeting.
4. **Security**: Secure cookies can be used to enhance the security of a website by ensuring that the transmission of information is only done over an encrypted connection, helping to prevent unauthorized access to user data.



We can see cookie if we inspect and go to the application tab.

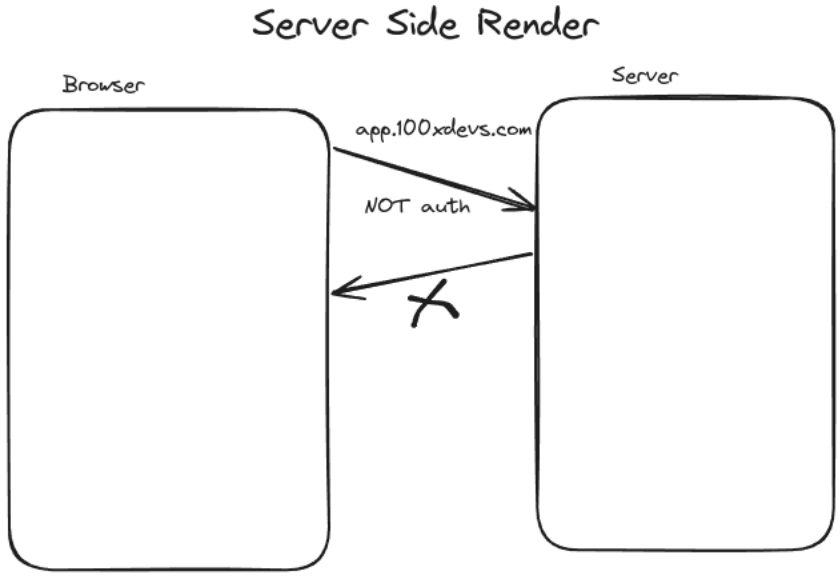
Why not local Storage?

Cookies and localStorage both provides ways to store data on the client-side, but they serve different purposes and have different characteristics

1. Cookies are send with every request to the websites (by the browser) (you don't have to explicitly add a header to the fetch call)
2. Cookies can have an expiry attached to them
3. Cookies can be restricted to only Https and to certain domains

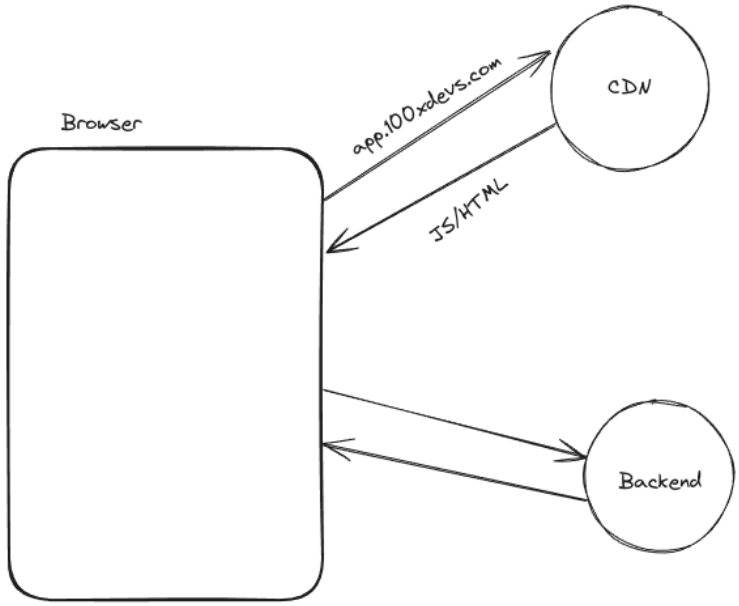
What is Benefit in cookies?

- We can't use localStorage like we did in React in NextJS . NextJS does the SSR the very first request that comes from the browser , has the user data. If this authorization doesn't have header then how will NextJs app know who is there. We can't have Authorization header/local storage from the very first request send by the Browser.



- In react Browser get frontend bundle from cdn , it doesn't matter if it doesn't receive authorization header , it is currently getting html, js file

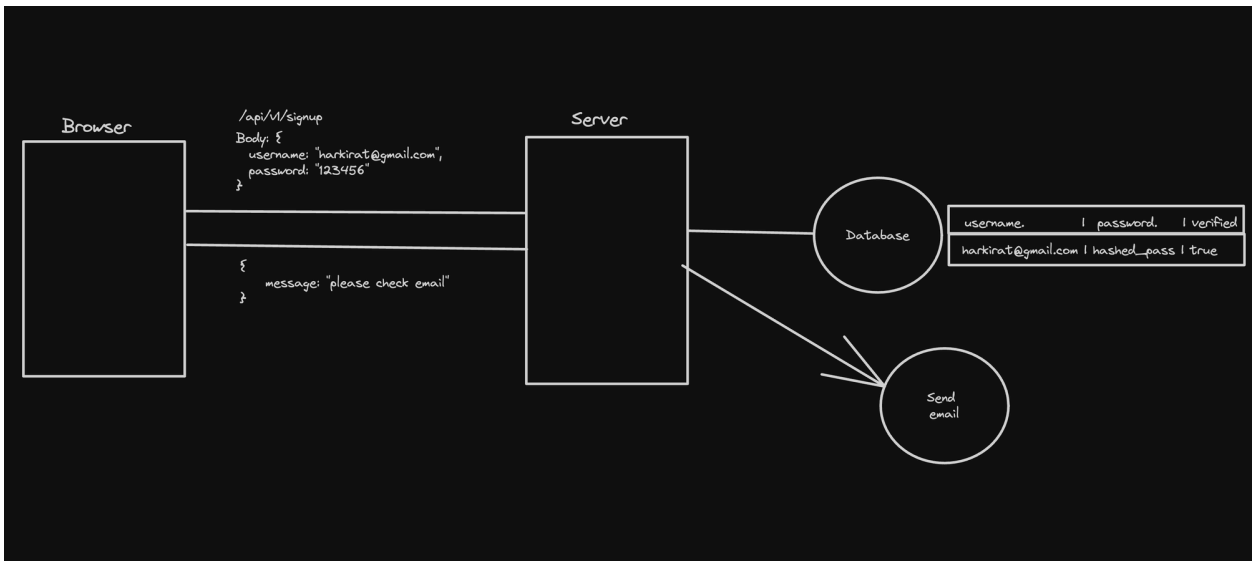
which are same for everyone and when it get the bundle then it send the subsequent request



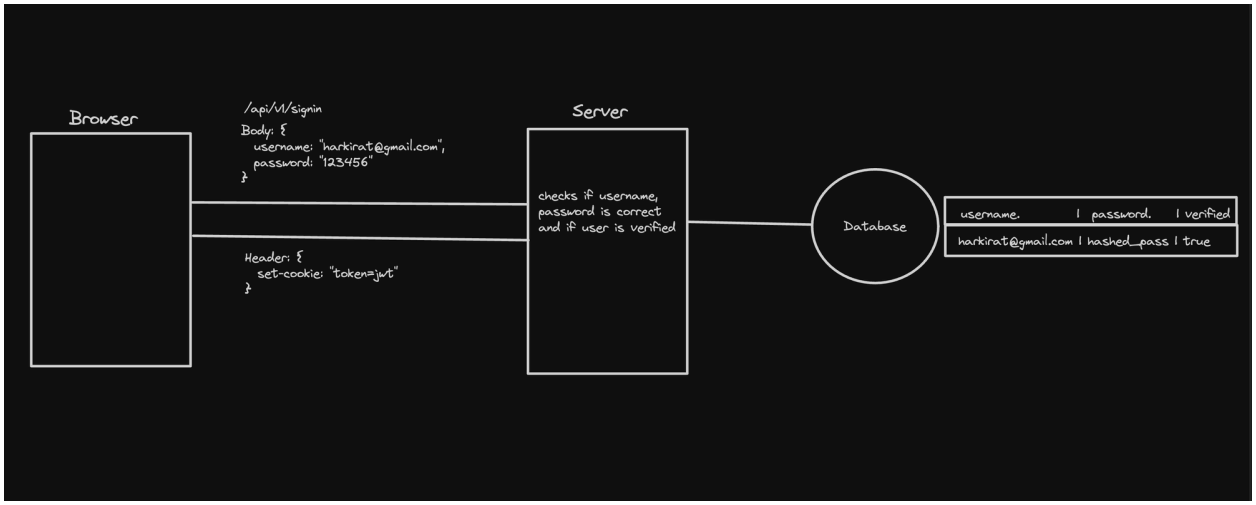
Hence in NextJs we need to send cookies they can be send from the start also Even in the first request the cookies goes in 100xdevs.com website , so that we know what courses a user is signed in to.

How authentication works with cookies

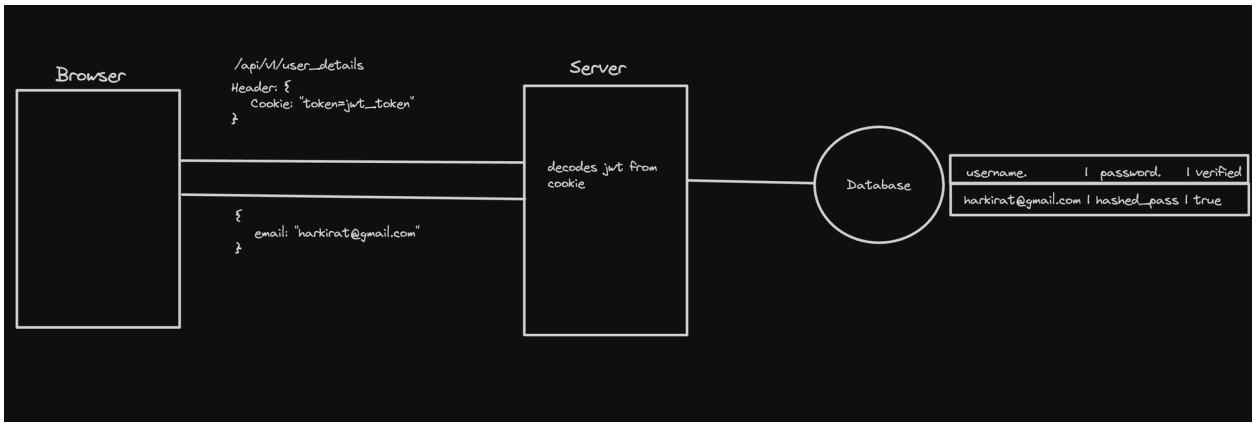
Signup



Signin



Auth endpoints



You don’t need to explicitly set the cookie header in the browser. It’s automatically set by the browser in every request. Even the request which doesn’t need the cookie

Properties of cookies

Types of Cookies

1. Persistent – Stay even if you close the window

Example Authentication cookie should be persistent , because even if we close the 100xdev.com window and if we reopen it then we are stil connected

2. Session - will go away after the window is closed
3. Secure – Sent only over secure, encrypted connections (HTTPs)

Properties of Cookies

- HttpOnly - Can not be accessed by client side scripts (JS to access the script) (Safety measure to ensure that it cannot be accessed)
- SameSite - Ensures cookies are not send on cross origin requests
    1. Strict
    2. Lax - **Only GET requests and on top level navigation**
    3. None

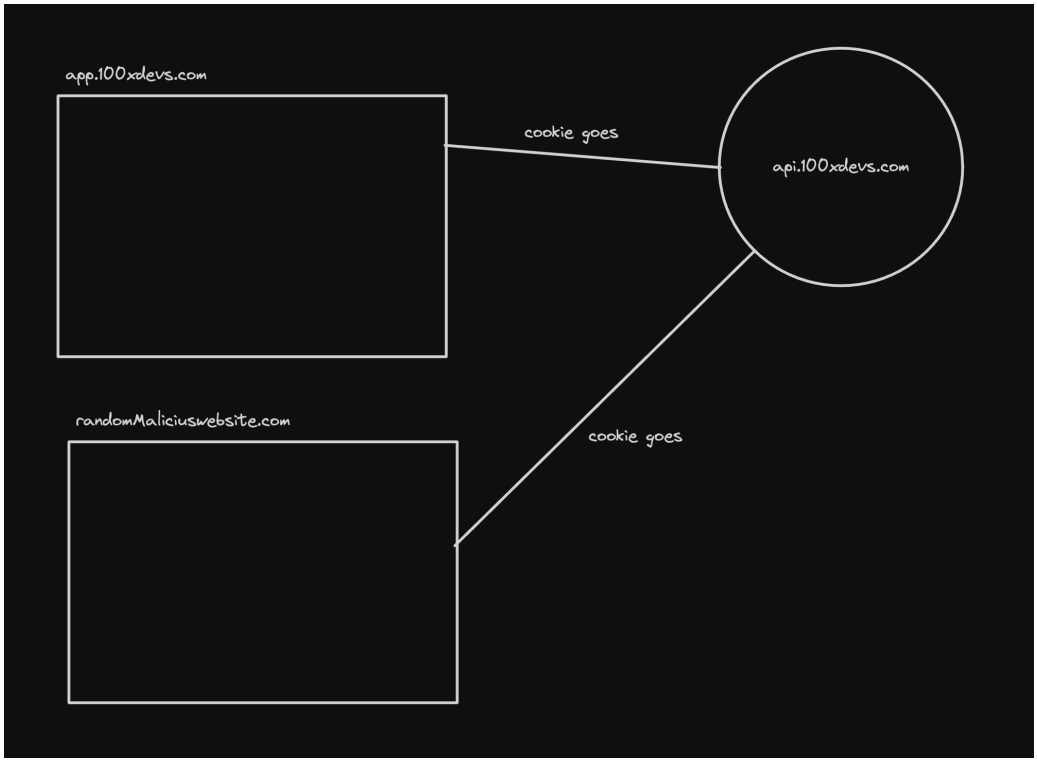
  Ref - .
  https://portswigger.net/web-security/csrf/bypassing-samesite-restrictions#:~:text=SameSite%20is%20a%20browser%20security,leaks%2C%20and%20some%20CORS%20exploits

- Domains - You can also specify what all domains should the cookie be sent from

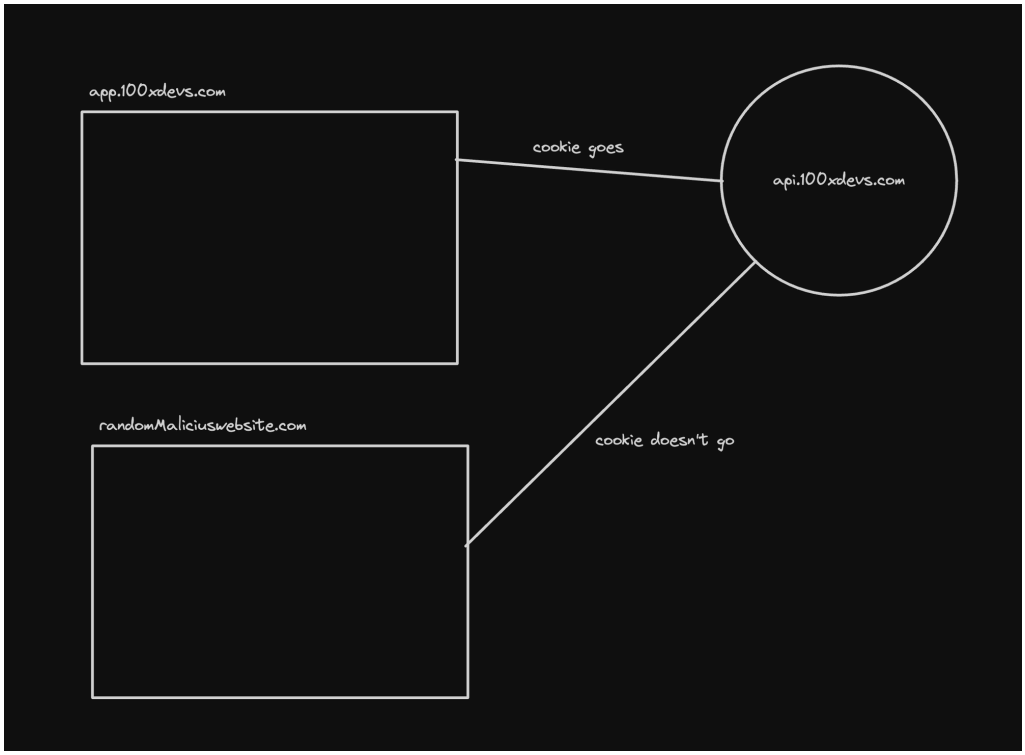
CSRF attacks

Cross site request forgery attacks were super common because of cookies and hence the **SameSite** attribute was introduced
Example

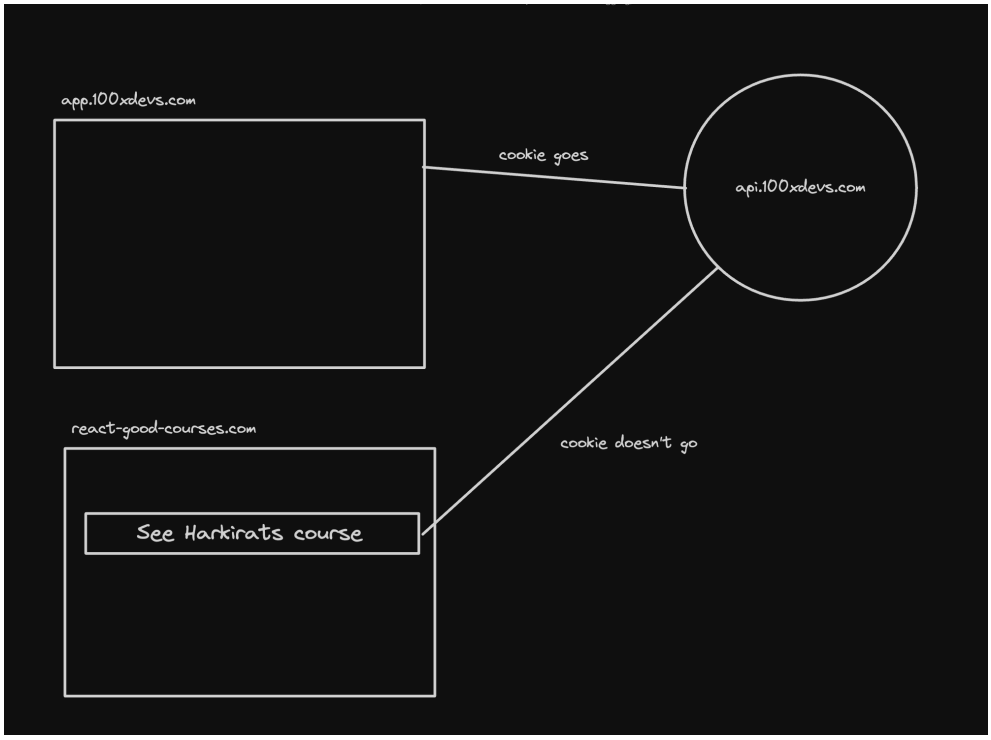
SameSite: none
This is problem because we can do cross site request forgery. Hence whenever we set cookies we need to be strict and restrict domain



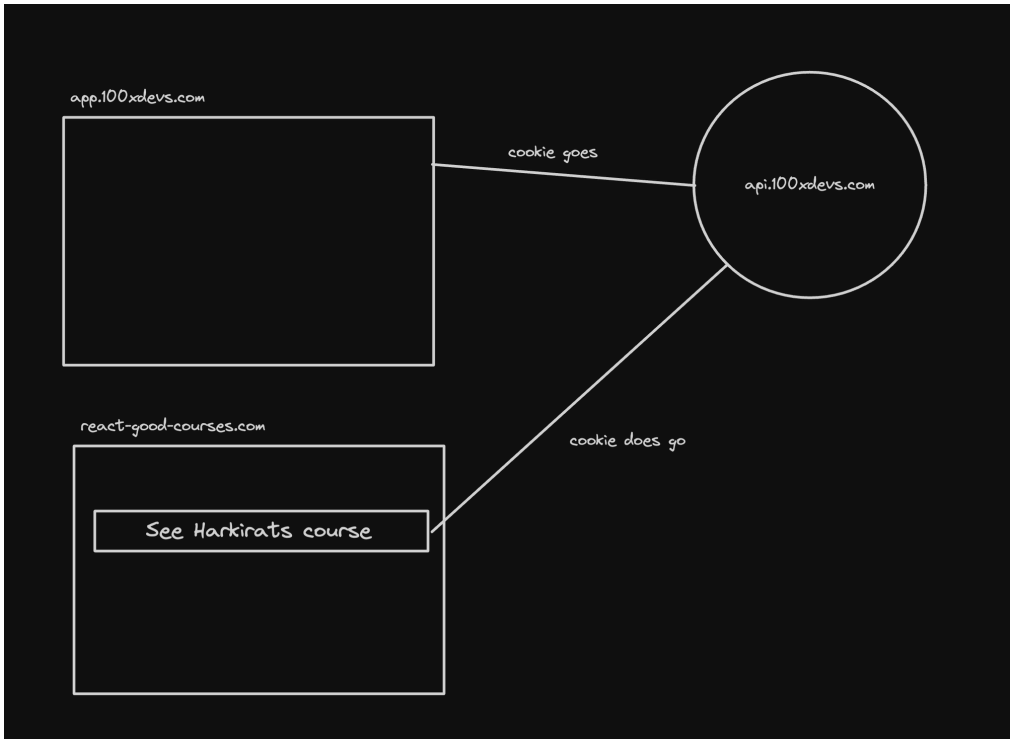
SameSite: Strict

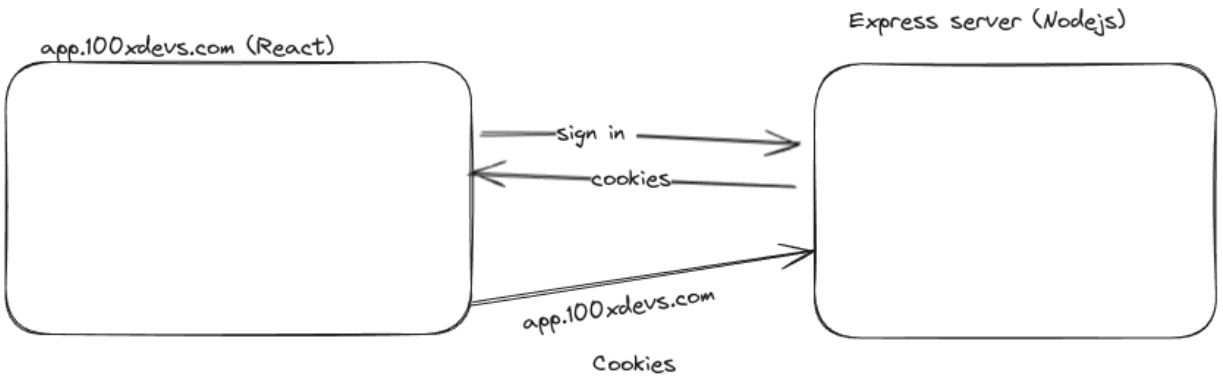


But there’s a problem\

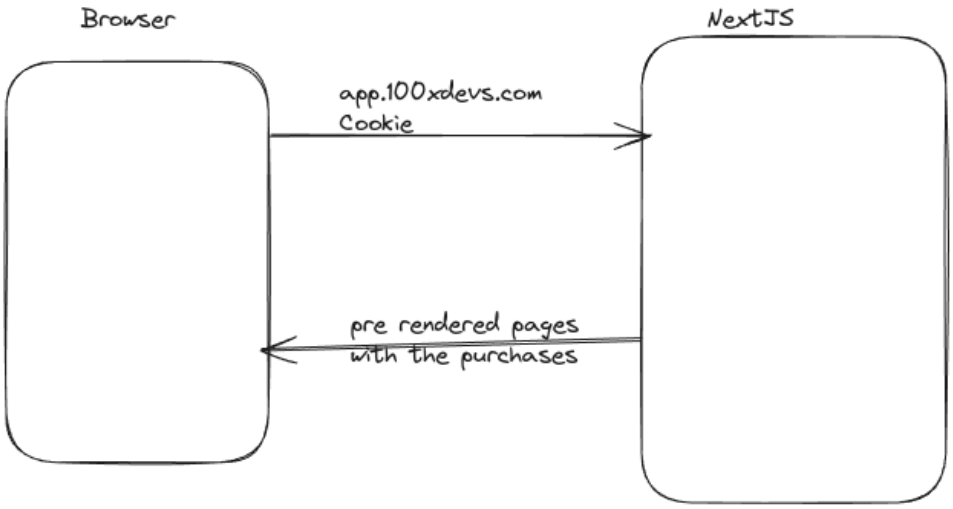


SameSite: Lax





It is property of browser that if it get back a set cookie header from server then it send cookies in every subsequent request



If same side is strict then suppose 100xdevs.com we wont reach the page where our courses are seen we will be directed to signup/sign landing page because the referer of the request was the third party website.

Examples in express (Backend)

Initialize an empty TS project

```
npm init -y
npx tsc --init
```

Update rootDir and OurDir

```
"roodDir": "./src"
"outDir": "./dist"
```

Add required Libraries

```
import express from "express";
import cookieParser from "cookie-parser";
import cors from "cors";
import jwt, { JwtPayload } from "jsonwebtoken";
import path from "path";
```

Initialize express app, add middlewares

```
const app = express();
app.use(cookieParser());
app.use(express.json());
app.use(cors({
   credentials: true,
   origin: "http://localhost:5173"
}));
```

Add a dummy signin endpoint

```
app.post("/signin", (req, res) => {
   const email = req.body.email;
   const password = req.body.password;
   // do db validations, fetch id of user from db
   const token = jwt.sign({
       id: 1
   }, JWT_SECRET);
   res.cookie("token", token);
   res.send("Logged in!");
});
```

Add a protected backend route

```
app.get("/user", (req, res) => {
   const token = req.cookies.token;
   const decoded = jwt.verify(token, JWT_SECRET) as JwtPayload;
   // Get email of the user from the database
   res.send({
```

```
        userId: decoded.id
    })
});
```

Add a logout route

```
app.post("/logout", (req, res) => {
    res.cookie("token", "ads");
    res.json({
        message: "Logged out!"
    })
});
```

Listen to port 3000

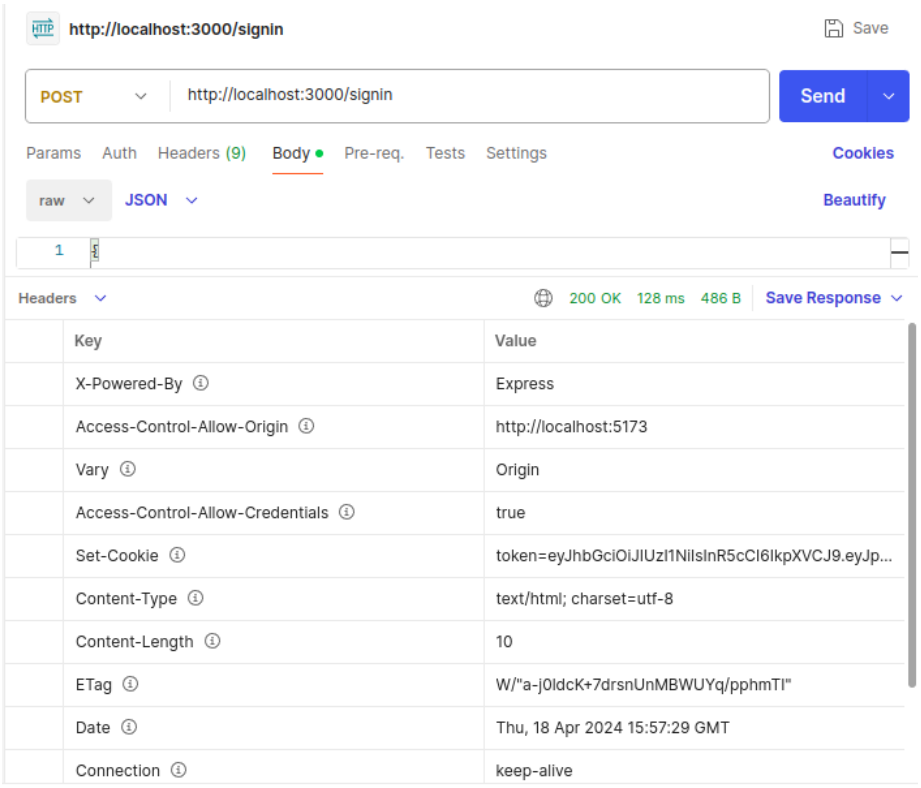
index.ts

```
import express from "express";
import cookieParser from "cookie-parser"; //parses a very long cookie
string
import cors from "cors";
import jwt, { JwtPayload } from "jsonwebtoken";
import path from "path";

const JWT_SECRET = "test123";

const app = express();
app.use(cookieParser());
app.use(express.json());
app.use(cors({
    credentials: true,
    origin: "http://localhost:5173" //at this place cookie will be set
}));

app.post("/signin", (req, res) => {
    const email = req.body.email;
    const password = req.body.password;
```

```
    // do db validations, fetch id of user from db
    const token = jwt.sign({
        id: 1
    }, JWT_SECRET);
    res.cookie("token", token);
    res.send("Logged in!");
});

app.get("/user", (req, res) => {
    const token = req.cookies.token;//get cookie from user
    const decoded = jwt.verify(token, JWT_SECRET) as JwtPayload;
    // Get email of the user from the database
    res.send({
        userId: decoded.id
    })
});


app.post("/logout", (req, res) => {
res.ClearCookie('token')
res.cookie("token", "ads");//will put cookie in the set-cookie header
    res.json({
        message: "Logged out!"
    })
});


app.get("/", (req, res) => {
    res.sendFile(path.join(__dirname, "../src/index.html"))
})

app.listen(3000);
```

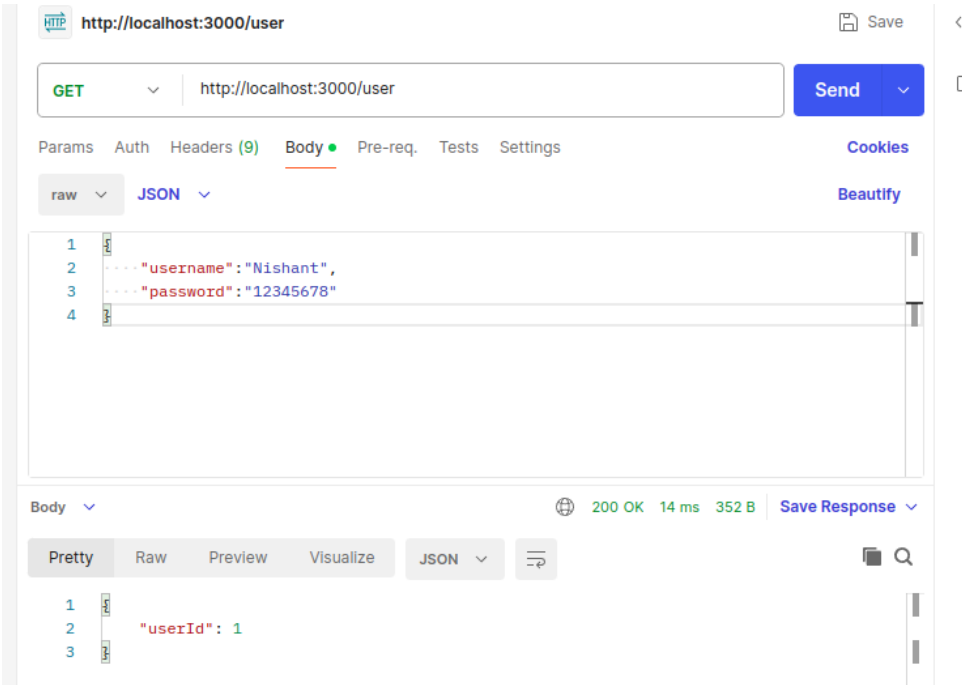
Cookie become difficult when we do cross origin , frontend and backend at different places

npm run build

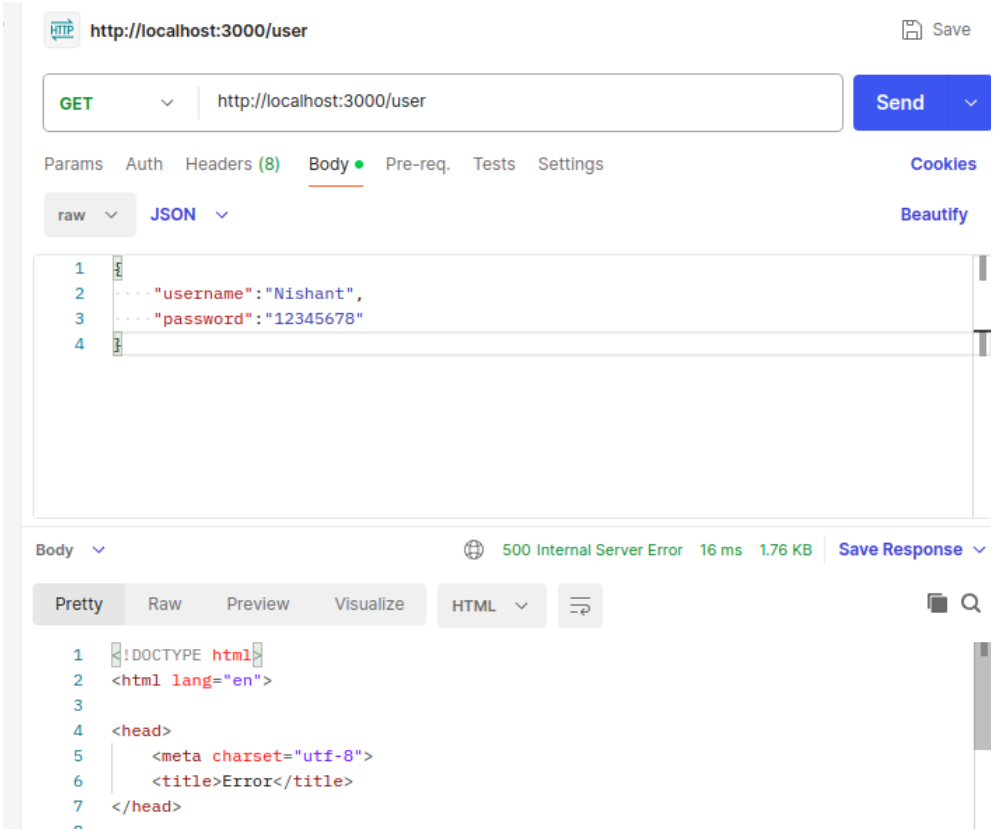


Testing the backend

Let send the GET request

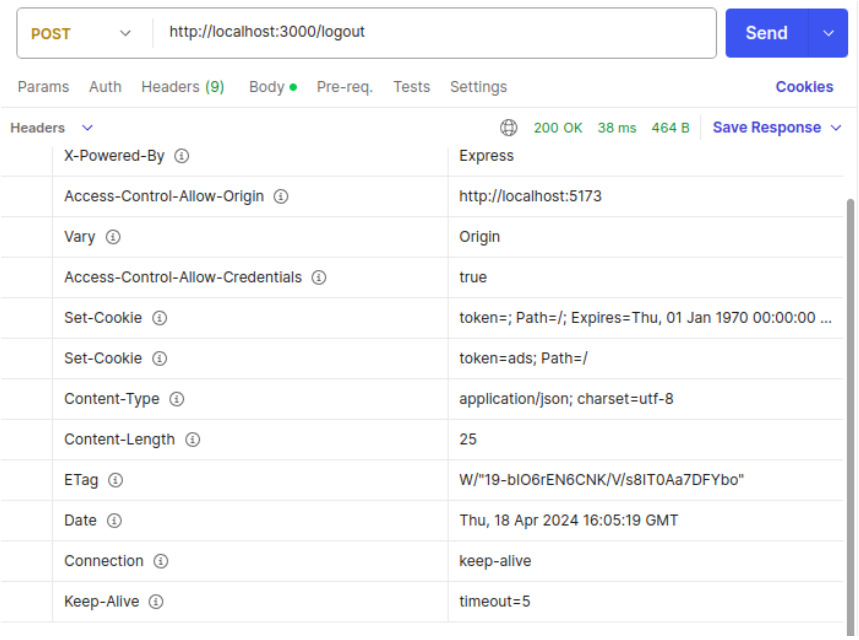


We receive an error if we remove the Cookies



POSTMAN cache cookies like the browser

logout endpoint



Now lets see frontend