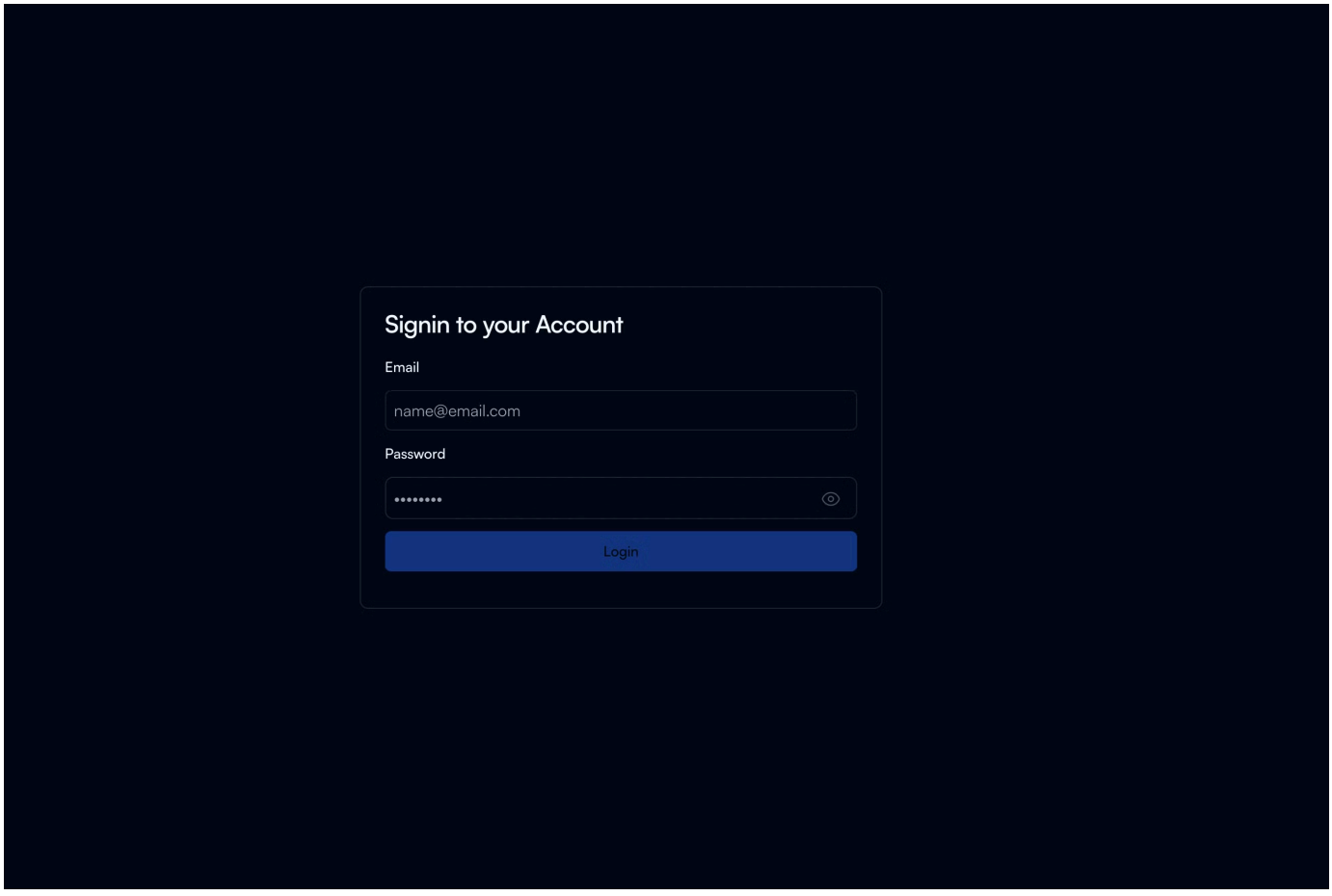
Context

Code - https://github.com/100xdevs-cohort-3/week-6-auth-app

Today we want to understand the most basic way to do authentication in a Node.js app.

What is authentication?

The process of letting users sign in / sign out of your website. Making sure your routes are protected and users can only get back their own data and not the data from a different user



1. Auth basics
2. Authentication 1 of 12
3. Authorization header
4. Creating your own auth middleware
5. localstorage

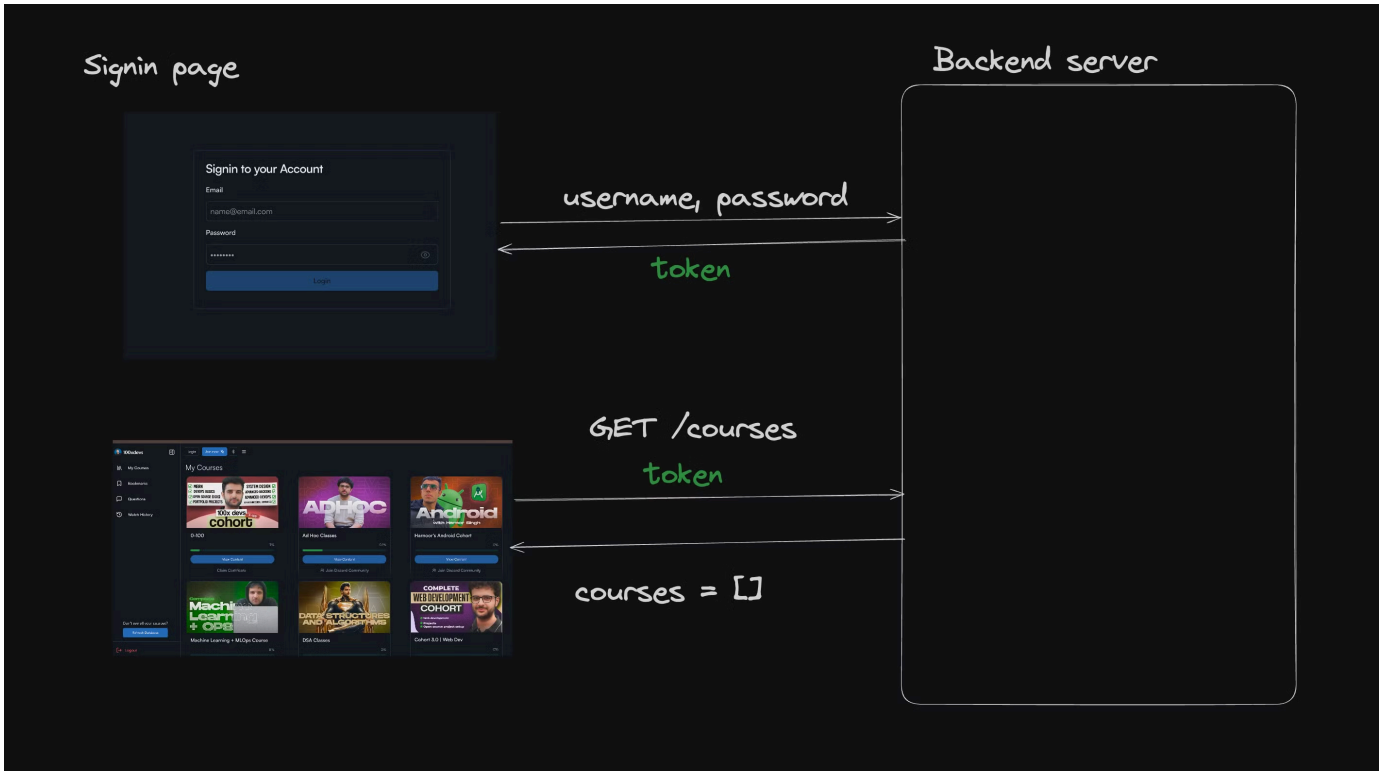
Auth workflow (Bank example)

When you go to open a bank account in a bank, you

1. Go to the bank and give your information.
2. They give you back a cheque book
3. Every time you want to send money, you write it in the cheque book and send it over to the bank
4. That is how the bank identifies you.

Auth workflow

The workflow for authentication usually looks as follows -



1. The user comes to your website (courses.com)
2. The user sends a request to /signin with their username and password

4 In every subsequent request, the user sends the token to identify it self to

💡 Think of the token like a secret that the server has given you. You send that secret back to the server in every request so that the server knows who you are.

Creating an express app

Lets initialise an express app that we will use to generate an authenticated backend today.

▼ Initialise an empty Node.js project

```
npm init -y
```

▼ Create an index.js file, open the project in visual studio code

```
touch index.js
```

▼ Add express as a dependency

```
npm i express
```

▼ Create two new POST routes, one for signing up and one for signing in

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

app.post("/signup", (req, res) => {
```

```
( q, res) => {
});

app.listen(3000);
```

▼ Use express.json as a middleware to parse the post request body

```
app.use(express.json());
```

▼ Create an in memory variable called users where you store the username , password and a token (we will come to where this token is created later.

```
const users = []
```

▼ Complete the signup endpoint to store user information in the in memory variable

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

  users.push({
    username,
    password
  })
  res.send({
    message: "You have signed up"
  })
});
```

▼ Create a function called generateToken that generates a random string for you

```
function generateToken() {
  let options = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v',

  let token = "";
  for (let i = 0; i < 32; i++) {
    // use a simple function here
                                    .random() * options.length)];
```

```
    return token;
}
```

Authentication 1 of 12

▼ Finish the signin endpoint. It should generate a token for the user and put it in the in memory variable for that user

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

    const user = users.find(user => user.username === username && user.pc

    if (user) {
        const token = generateToken();
        user.token = token;
        res.send({
            token
        })
        console.log(users);
    } else {
        res.status(403).send({
            message: "Invalid username or password"
        })
    }
});
```

This can be improved further by
1. Adding zod for input validation
2. Making sure the same user cant sign up twice
3. Persisting data so it stays even if the process crashes
We'll be covering all of this eventually

Creating an authenticated EP

Let's create an endpoint (/me) that returns the user their information `only if they send their

```
app.get("/me", (req, res) => {
    const token = req.headers.authorization;
    const user = users.find(user => user.token === token);
    if (user) {
        res.send({
            username: user.username
        })
    } else {
        res.status(401).send({
            message: "Unauthorized"
        })
    }
})
```

Tokens vs JWTs

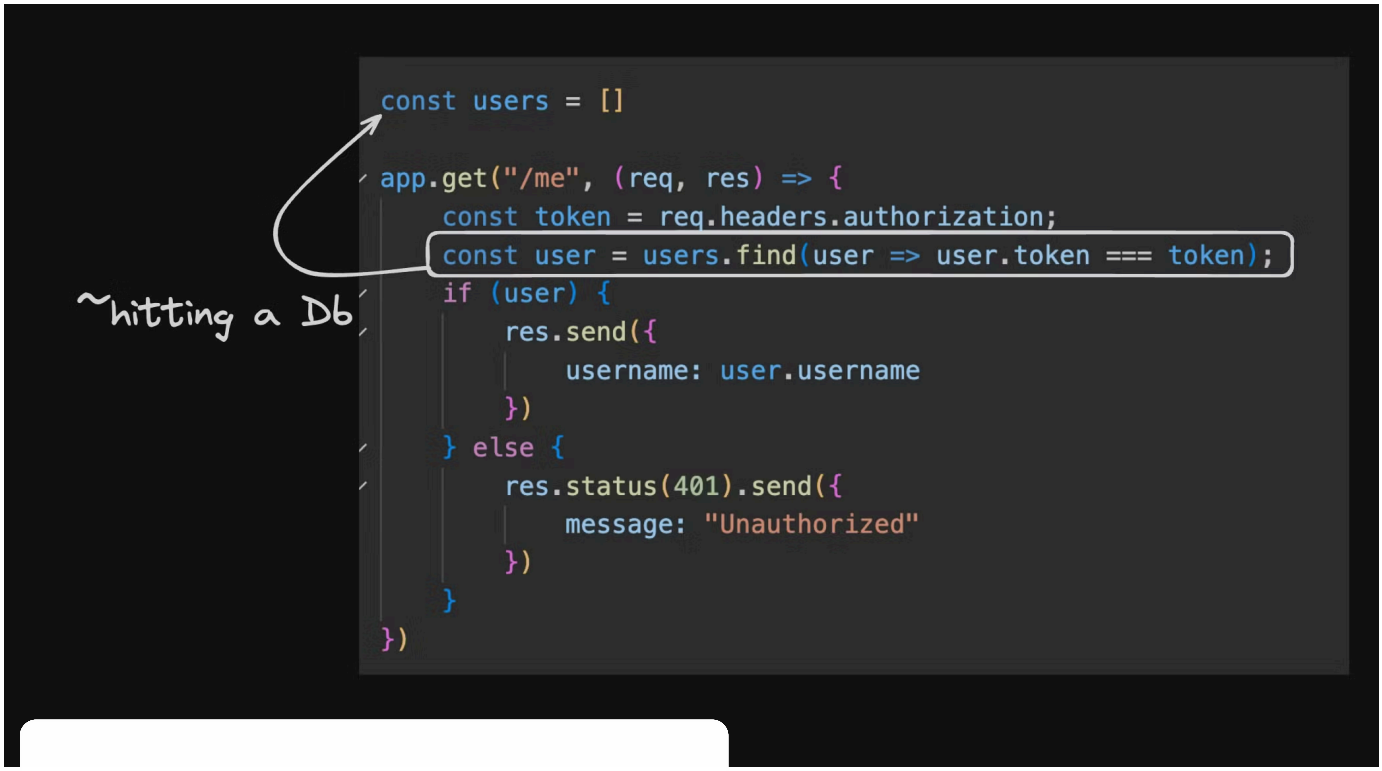
There is a problem with using stateful tokens.

Stateful

By stateful here, we mean that we need to store these tokens in a variable right now (and eventually in a database).

Problem

The problem is that we need to send a request to the database every time the user wants to hit an authenticated endpoint



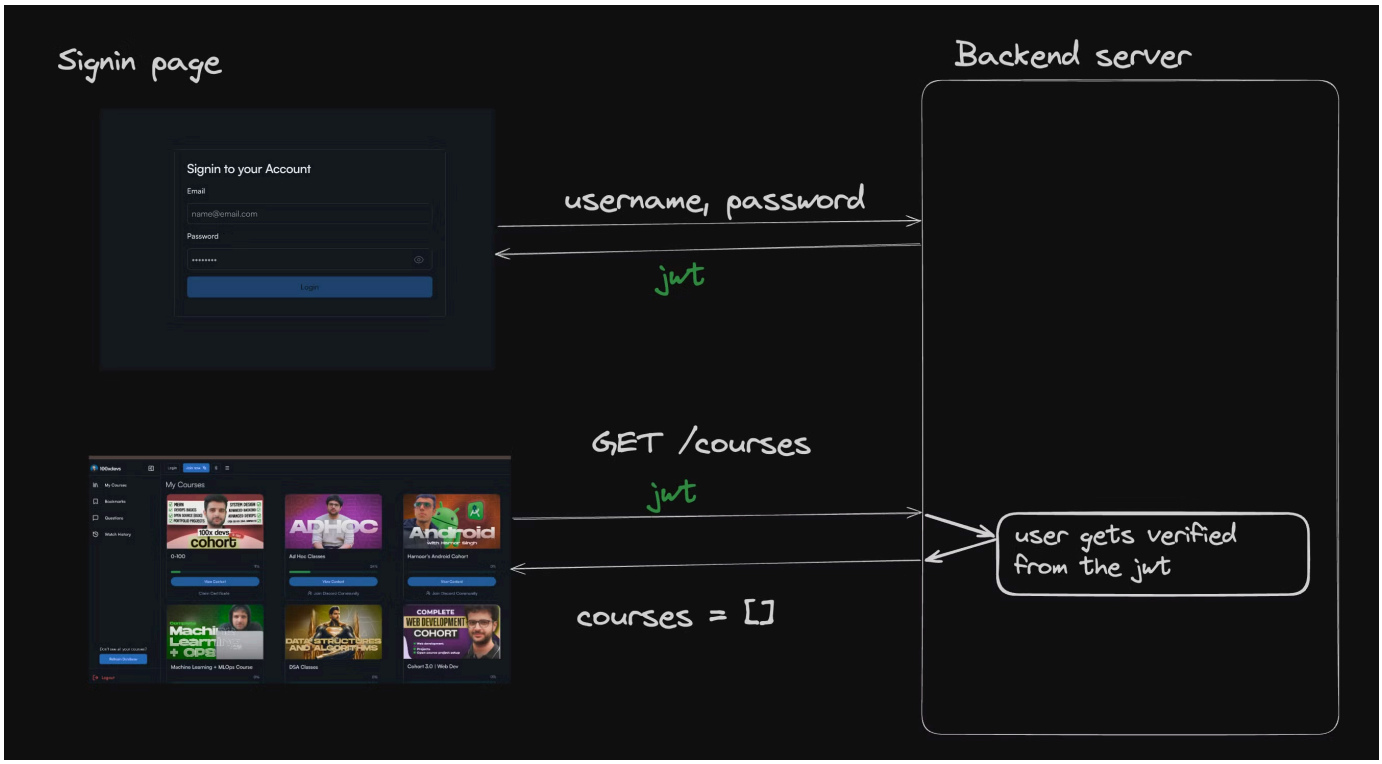
Solution

Authentication 1 of 12

JWTs

JWTs, or JSON Web Tokens, are a compact and self-contained way to represent information between two parties. They are commonly used for authentication and information exchange in web applications.

JWTs are Stateless: JWTs contain all the information needed to authenticate a request, so the server doesn't need to store session data. All the data is stored in the token itself.



Replace token logic with jwt

Lets change the token logic that we had to use jwts

▼ Add the `jsonwebtoken` library as a dependency - https://www.npmjs.com/package/jsonwebtoken

```
npm install jsonwebtoken
```

▼ Get rid of our `generateToken` function

```
function generateToken() {
  let options = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v',

  let token = "";
  for (let i = 0; i < 32; i++) {
    // use a simple function here
    token += options[Math.floor(Math.random() * options.length)];
  }
  return token;
}
```

▼ Create a `JWT_SECRET` variable

```
const JWT_SECRET = "USER_APP";
```

▼ Create a jwt for the user instead of generating a token

```
const username = req.body.username;
```

```
const password = req.body.password;

Authentication 1 of 12 ind(user => user.username === username && user.pc

if (user) {
    const token = jwt.sign({
        username: user.username
    }, JWT_SECRET);

    user.token = token;
    res.send({
        token
    })
    console.log(users);
} else {
    res.status(403).send({
        message: "Invalid username or password"
    })
}
});
```

Notice we put the `username` inside the token. The `jwt` holds your

You no longer need to store the `token` in the global `users` variable

▼ In the `/me` endpoint, use `jwt.verify` to verify the token

```
app.get("/me", (req, res) => {
    const token = req.headers.authorization;
    const userDetails = jwt.verify(token, JWT_SECRET);

    const username =  userDetails.username;
    const user = users.find(user => user.username === username);

    if (user) {
        res.send({
            username: user.username
        })
```

```
        message: "Unauthorized"
    })
```


JWTs can be DECODED by anyone

JWTs can be decoded by anyone. They can be verified by only the server that issued them.

Ref - https://jwt.io/

Try creating a jwt and decoding it on the website. You'll notice it does decode. But that is fine

Comparision to a cheque.

If you ever sign a cheque , you can show it to everyone and everyone can see that you are transferring $20 to a friend. But only the BANK needs to verify before debiting the users account.

Doesnt matter if everyone sees the cheque, they cant do anything with this information.

But the bank can verify the signature and do whatever the end users asked to do

JWTs can be verified by only the person who issued them (using the JWT s

Assignment: Creating an auth middleware

Can you try creating a `middleware` called `auth` that verifies if a user is logged in and ends the request early if the user isn't logged in?

▼ Solution

```
function auth(req, res, next) {
  const token = req.headers.authorization;

  if (token) {
    jwt.verify(token, JWT_SECRET, (err, decoded) => {
      if (err) {
        res.status(401).send({
          message: "Unauthorized"
        })
      } else {
        req.user = decoded;
        next();
      }
    })
  } else {
    res.status(401).send({
      message: "Unauthorized"
    })
  }
```

```
}

app.get("/me", (req, res) => {
  const user = req.user;

  res.send({
    username: user.username
  })
})
```


Writing the frontend for it

Until now, we've been using POSTMAN to send out all of our requests.

Now, lets create a full stack application. Lets write the frontend that lets you

1. Signup
2. Signin
3. Get your information
4. Logout

Writing the frontend

▼ Create a public/index.html file

```
mkdir public
cd public
touch index.html
```

```
<div>
  Authentication 1 of 12
  <input type="text" name="username" placeholder="Username">
  <input type="password" name="password" placeholder="Password">
  <button onclick="signup()">Submit</button>
</div>
```

▼ Create a signin section

```
<div>
  Signin
  <input type="text" name="username" placeholder="Username">
  <input type="password" name="password" placeholder="Password">
  <button onclick="signin()">Submit</button>
</div>
```

▼ Create a User information section

```
<div>
  User information:
  <div id="information"></div>
</div>
```

▼ Create a logout button

```
<div>
  <button onclick="logout()">Logout</button>
</div>
```

Writing the onclick handlers

▼ Add the axios external library

```
<script src="https://cdnjs.cloudflare.com/ajax/libs/axios/1.7.7/axios.min.js">
```

▼ Write the signup function

```
async function signup() {
  const username = document.getElementById("signup-username").value;
                                  mentById("signup-password").value;
```

```
    const response = await axios.post("http://localhost:3000/signup", {
        username,
        password
    })
    alert("Signed up successfully");
}
```

▼ Write the signin function

```
async function signin() {
    const username = document.getElementById("signin-username").value;
    const password = document.getElementById("signin-password").value;

    const response = await axios.post("http://localhost:3000/signin", {
        username: username,
        password: password
    });

    localStorage.setItem("token", response.data.token);

    alert("Signed in successfully");
}
```

▼ Write the logout function

```
async function logout() {
    localStorage.removeItem("token");
}
```

▼ Write the `getUserInformation` function

```
async function getUserInformation() {
    const token = localStorage.getItem("token");

    if (token) {
        const response = await axios.get("http://localhost:3000/me", {
            headers: {
                Authorization: token
            }
        });
                                    nation").innerHTML = response.data.us
```

```
    }
}
```


Updating the backend

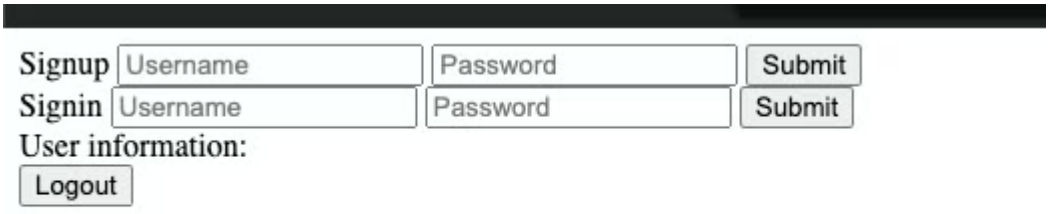
Lets serve the `index.html` file directly from the backend

▼ Approach #1

```
app.get("/", function(req, res) {
  res.sendFile("./public/index.html")
})
```

▼ Approach #2

```
app.use(express.static("./public"))
```



Firing the `getUserInformation` call

▼ Call the `getUserInformation` function when the page loads

```
getUserInformation();
```

Assignment

Conditionally render the `logout` or the `signin` / `signup` pages based on if the user is already logged in or not

Assignment: Creating a TODO app

Try to create a TODO application where

1. User can signup/signin
2. User can create/delete/update TODOs
3. User can see their existing todos and mark them as done