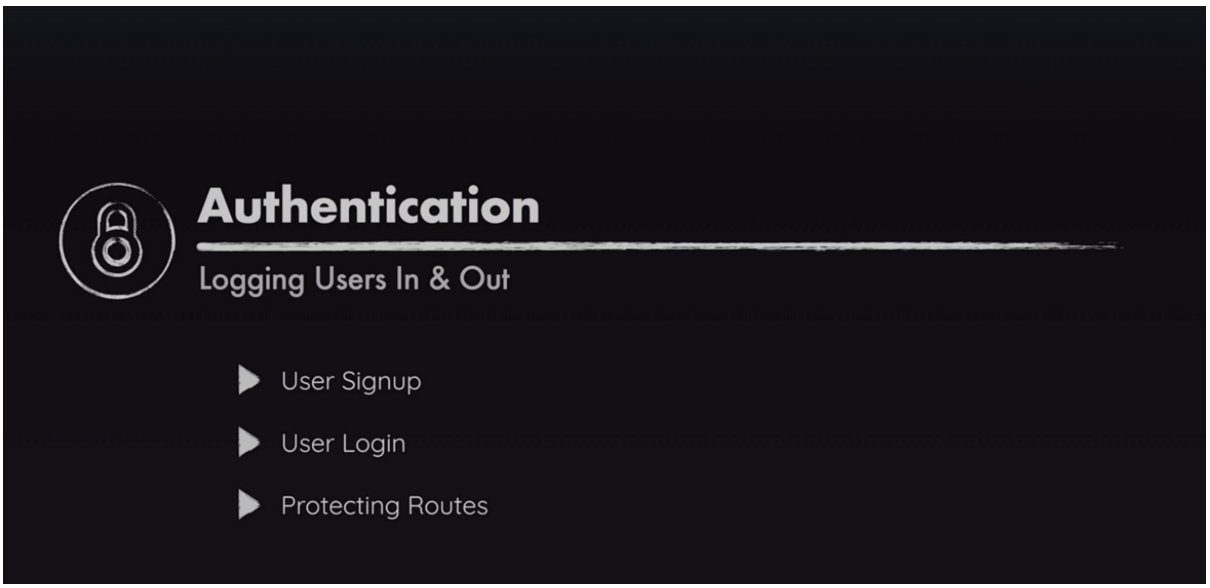
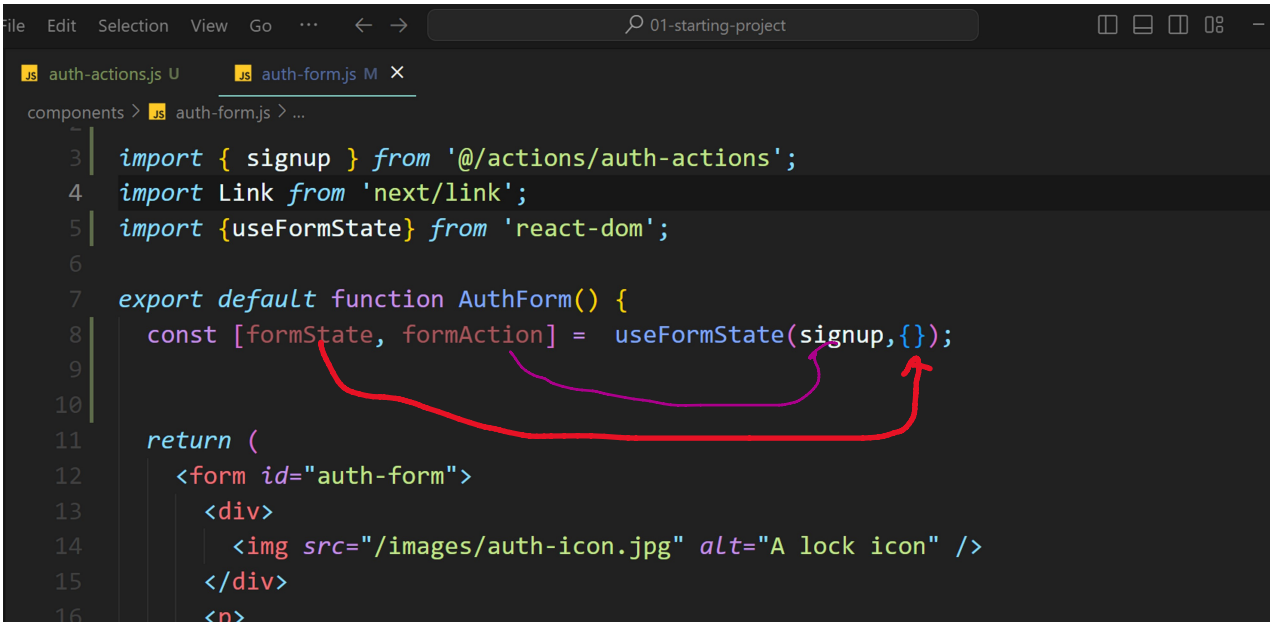
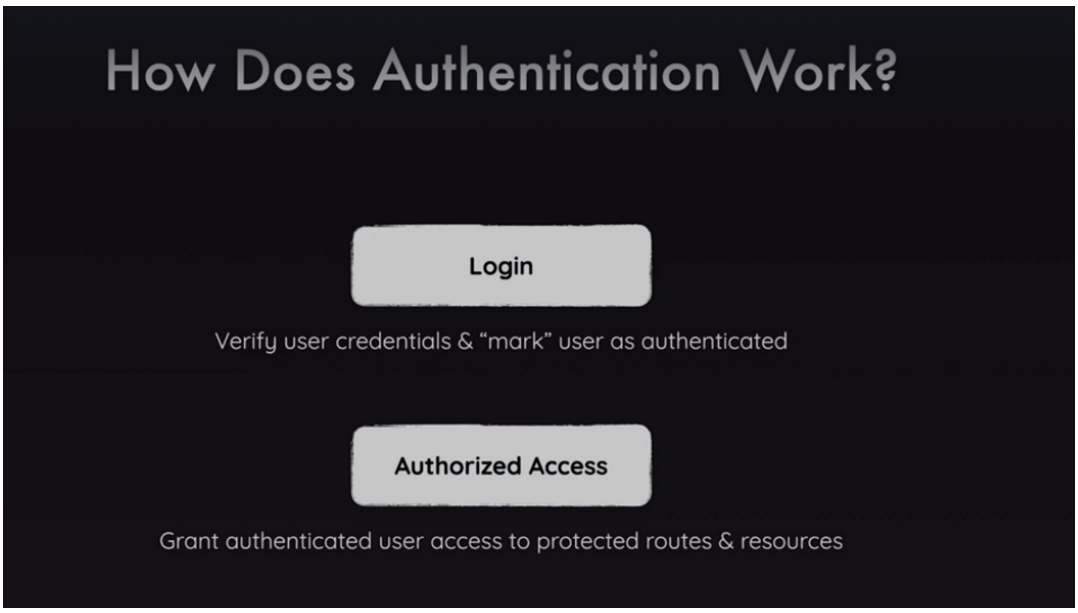
Section 9: User Authentication

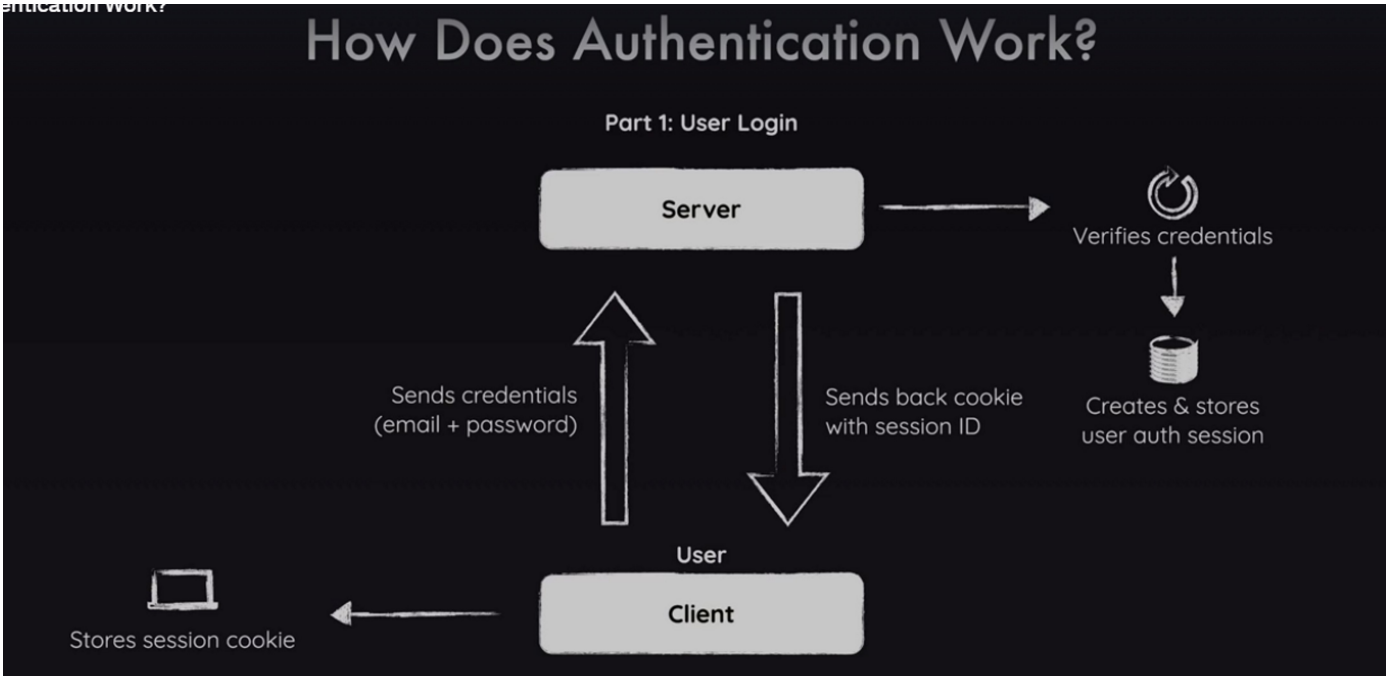
15 September 2024 15:38

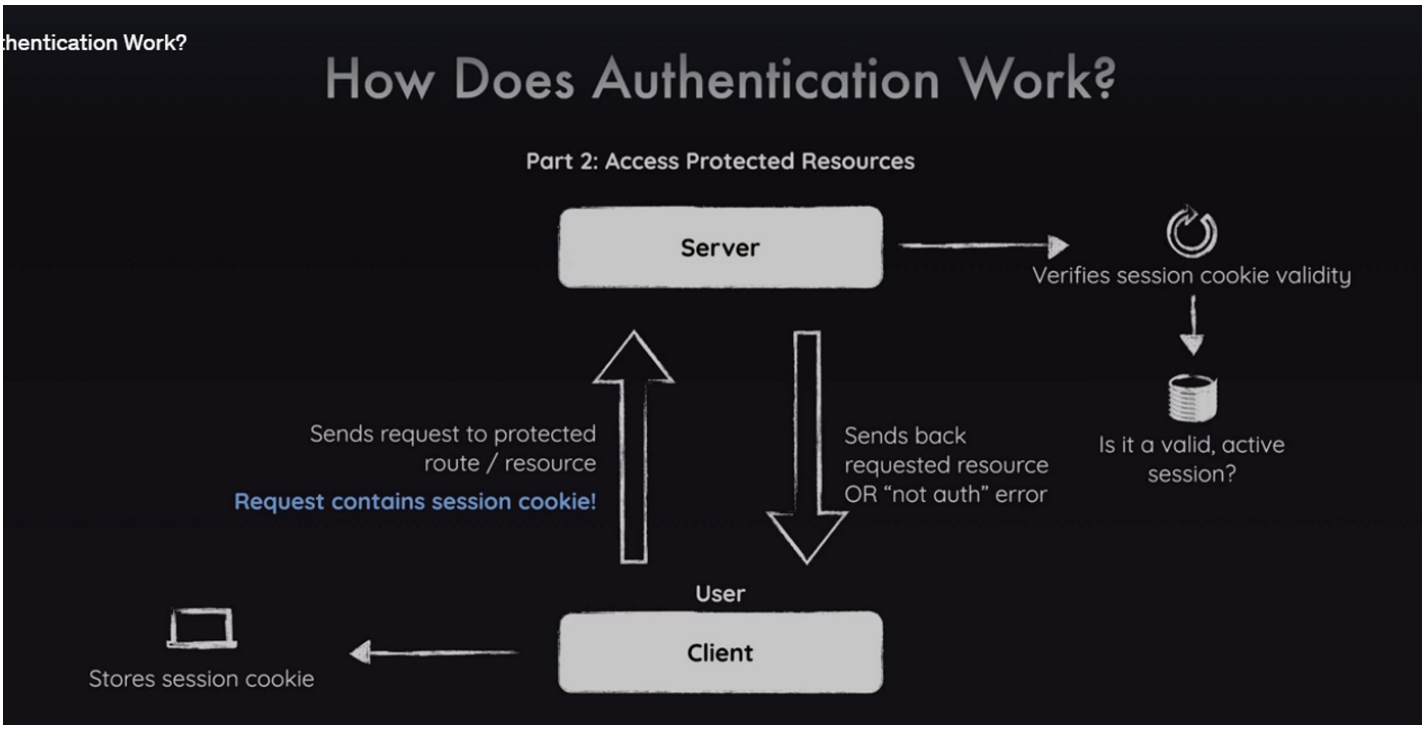




First element will be the form state and the second argument will be the action. The signup action but actually wrapped with an extra function essentially that allows React to listen to that action and manage this **formState** for us.







So we got that user signup implemented, but as mentioned, that isn't really the authentication process yet. Instead, we want to make sure that users, for example, can't visit /training if they didn't just log in, so if they're not authorized. And therefore we need to answer one important question,

how exactly does authentication work?

What do we have to do as a next step? Well, in the end, authentication is a two part process,
and the first part is that we must be able to log users in, which in the end means that we wanna be able to verify
user credentials, email and password, check if they're valid, if they match the ones that were used
during account creation and, and that's very important,

we want to then mark a user who did provide valid credentials as authenticated.

We need some way of remembering that this user who just sent this request is authenticated
and should be granted access to protected resources, because that's the second part, that we have certain resources that should be protected and that should only be made accessible to authenticated users, to users who logged in
and who are marked as authenticated. Now let's start with that first part, logging users in and marking them as authenticated. It all starts with a client, a user visiting our website, who in the end sends a request to our server,
a request where they filled out that authentication form to either create a new user or to log in with an existing user
and either way, they send some credentials along with that request, because they submitted that form.

So they sent an email and a password to the server. On that server, we then verify those credentials,
either because we just create a new user and we check whether we have a valid email,
a valid password and the user doesn't exist yet or because the user tried to log in, we take a look at the existing users
that we stored in the database and we check whether that email password combination
that was submitted is valid. If we consider it valid, so if we determine that the user did provide correct credentials,
we create a so-called authentication session, which in the end is simply a database entry
that's stored in a dedicated table on the server, that's how we remember that a certain user is now considered to be authenticated, we store a new entry in a table.

Now that entry, in a table, will in the end have an ID, a so-called session ID,and we then send back that ID, typically in a cookie, to the user. So the response to that authentication request,
to that login request, contains such, a session cookie, a cookie with such a session ID, if the credentials were valid and the client, the browser, will then automatically store that session cookie, that's the first part.

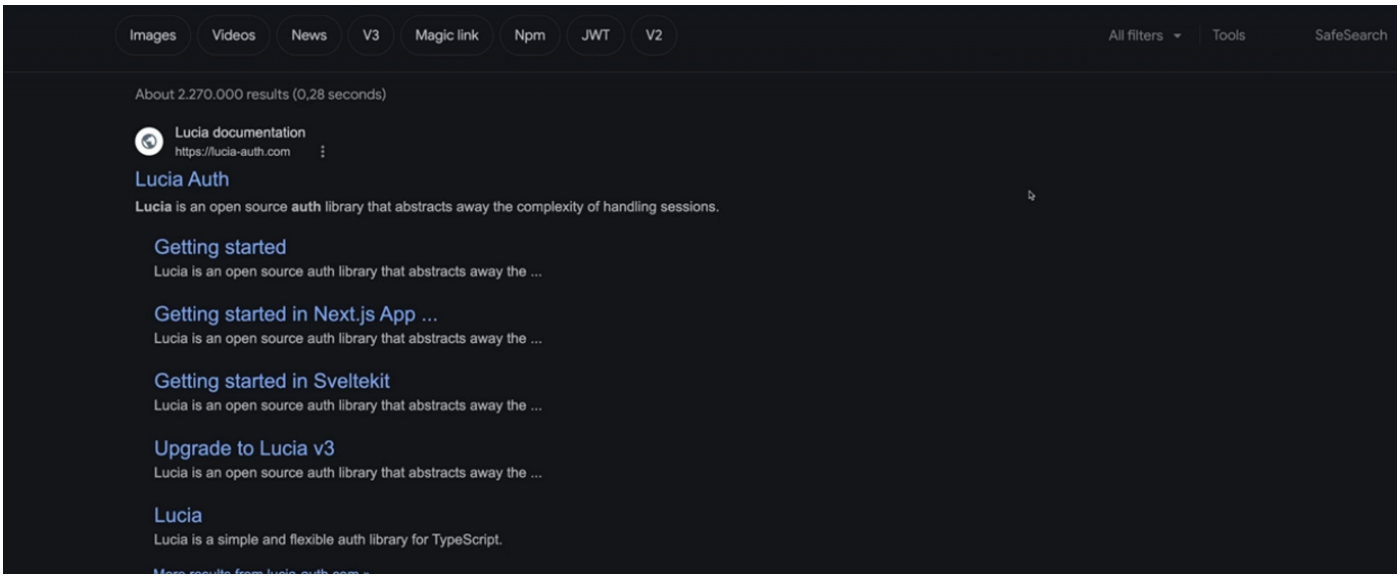
Now that we have this cookie, and now that we remembered on the server that this user is considered to be authenticated,
we can move on to the second part, which is about accessing protected resources. Because let's now say a user who just logged in and who has that cookie now sends a request to some route that should be protected and we'll write some code that will protect the route, and you will see how that works in that section. What's important about that request
is that the browser will automatically add cookies that belong to our website to those requests, so that session cookie will automatically be attached to that request and will be sent to the server. On the server, we can then take a look
at that session cookie and verify its validity. So if it in the end contains a session ID,
we also have in our database, if it's a valid session id. And therefore those session IDs also aren't just some kind of numbers, but instead more complex strings that can't be easily made up.

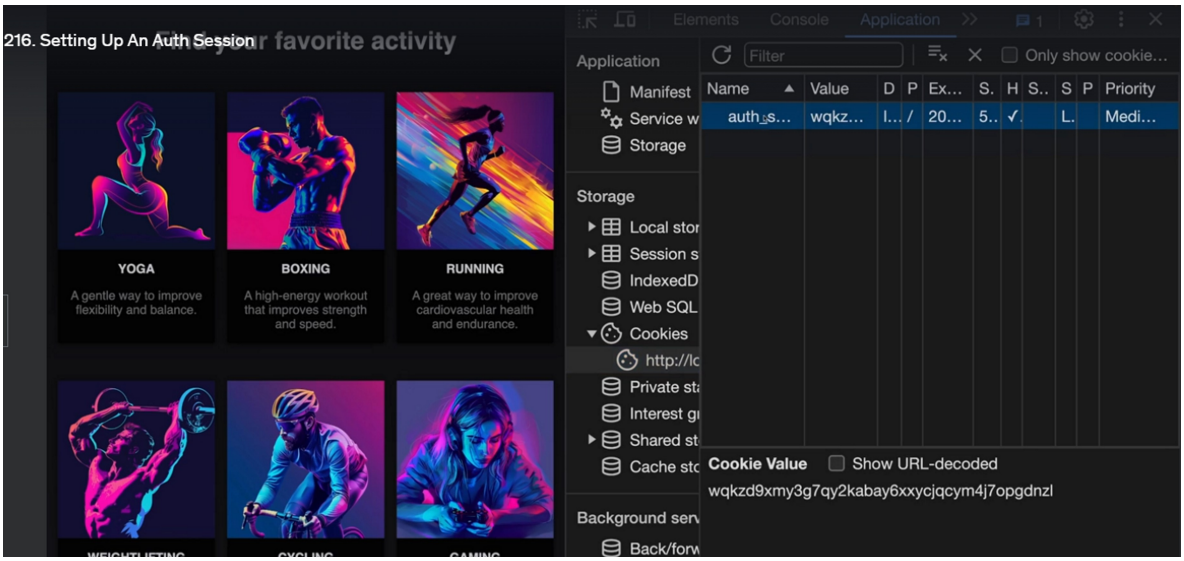
Now, once we determined that we got a request with a valid active session cookie, we sent back the requested resource.

So for example, the page content the user wanted to see, or if the cookie doesn't look all right,
if the session expired or something like that, we send back an error,that's in the end how authentication works and that's what we'll now implement over the next lectures.

Choosing a Third-Party Auth Package

npm I lucia @lucia-auth/adapter-sqlite

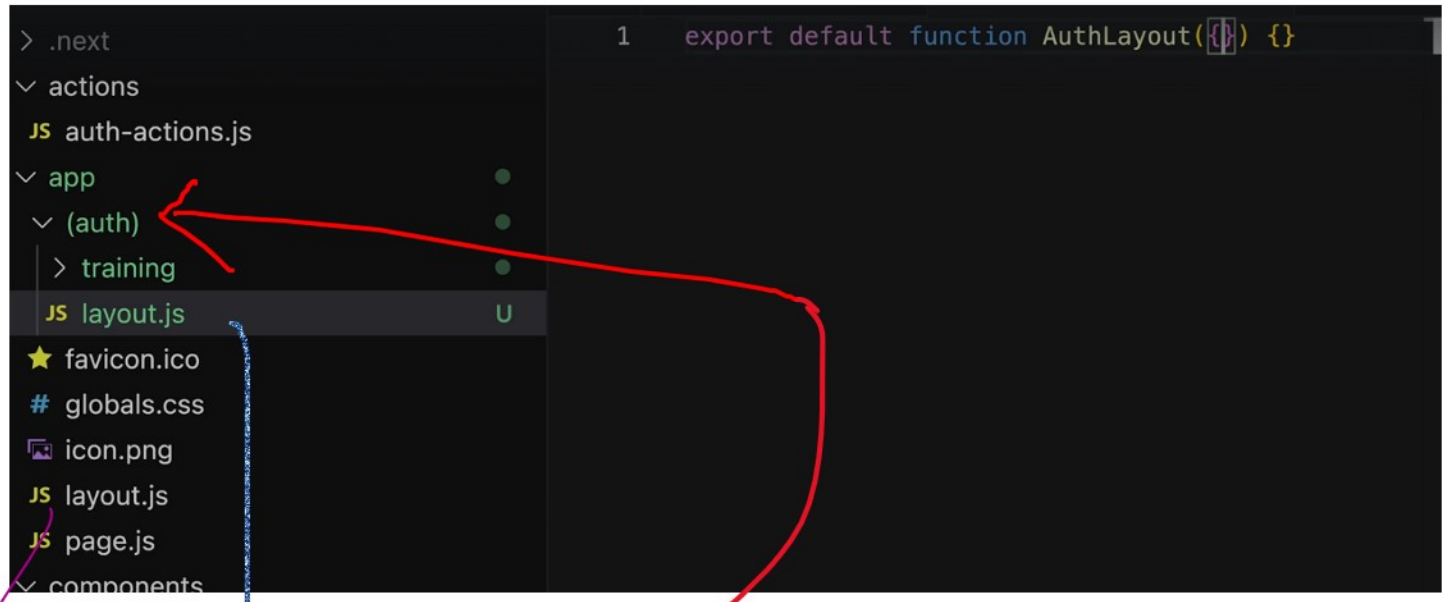




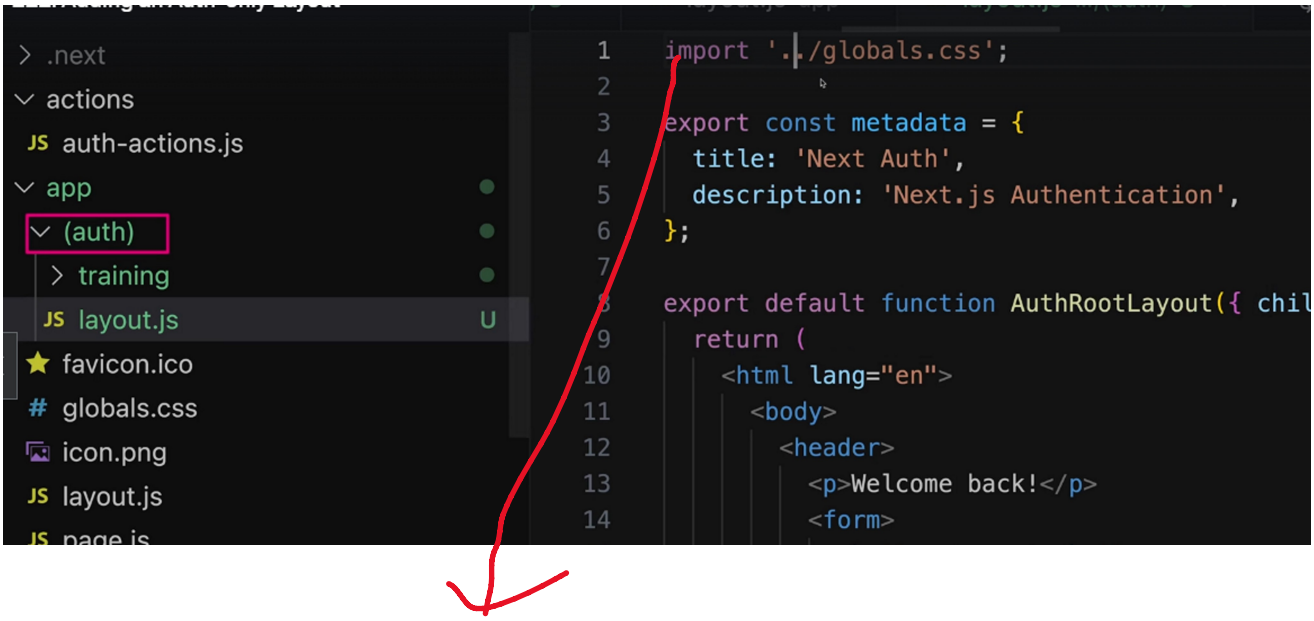
We cannot just create auth sessions, but that we can also verify auth sessions

so that we can verify whether a request is coming from a user who did log in. And for that, in that lib auth.js file, I'll export a new function, a new async function which I'll name verifyAuth. The name of course is up to you. This function should check whether an incoming request is coming from an authenticated user.
So if incoming request has that authentication cookie and if it's a valid cookie, so a cookie with a valid session ID that we stored on the server in the database because just the existence of the cookie alone is not enough.
It could be a faked cookie. We definitely also need to validate it. Now first, we need to retrieve the cookie.

```
31  export async function verifyAuth() {
32
33  }
```



(auth) is route group. This folder will not create extra path segment.



even though (auth) will not add extra path segment, but still it's a physical folder in our file system, that's why I have added ../global.css

One Root Layout vs Multiple Root Layouts

In the previous lecture, we added the (auth) route group and an auth-only layout (by adding a layout.js file in that (auth) folder). Currently, this (auth)/layout.js layout will actually **not replace the main root layout** (i.e., the layout provided by app/layout.js). Because route group layouts are actually **nested into the main root layout** unless there is no such root layout. I.e., if you would create another route group (e.g., (unauth)) and move the layout.js file into that folder, you would end up with **multiple root layouts**.

From <https://www.udemy.com/course/nextjs-react-the-complete-guide/learn/lecture/43341220#overview>