

Authentication for Next.js

Asst.Prof.Drusawin Vongpramate

Department of Information Technology

Faculty of Science, BRU

Hint

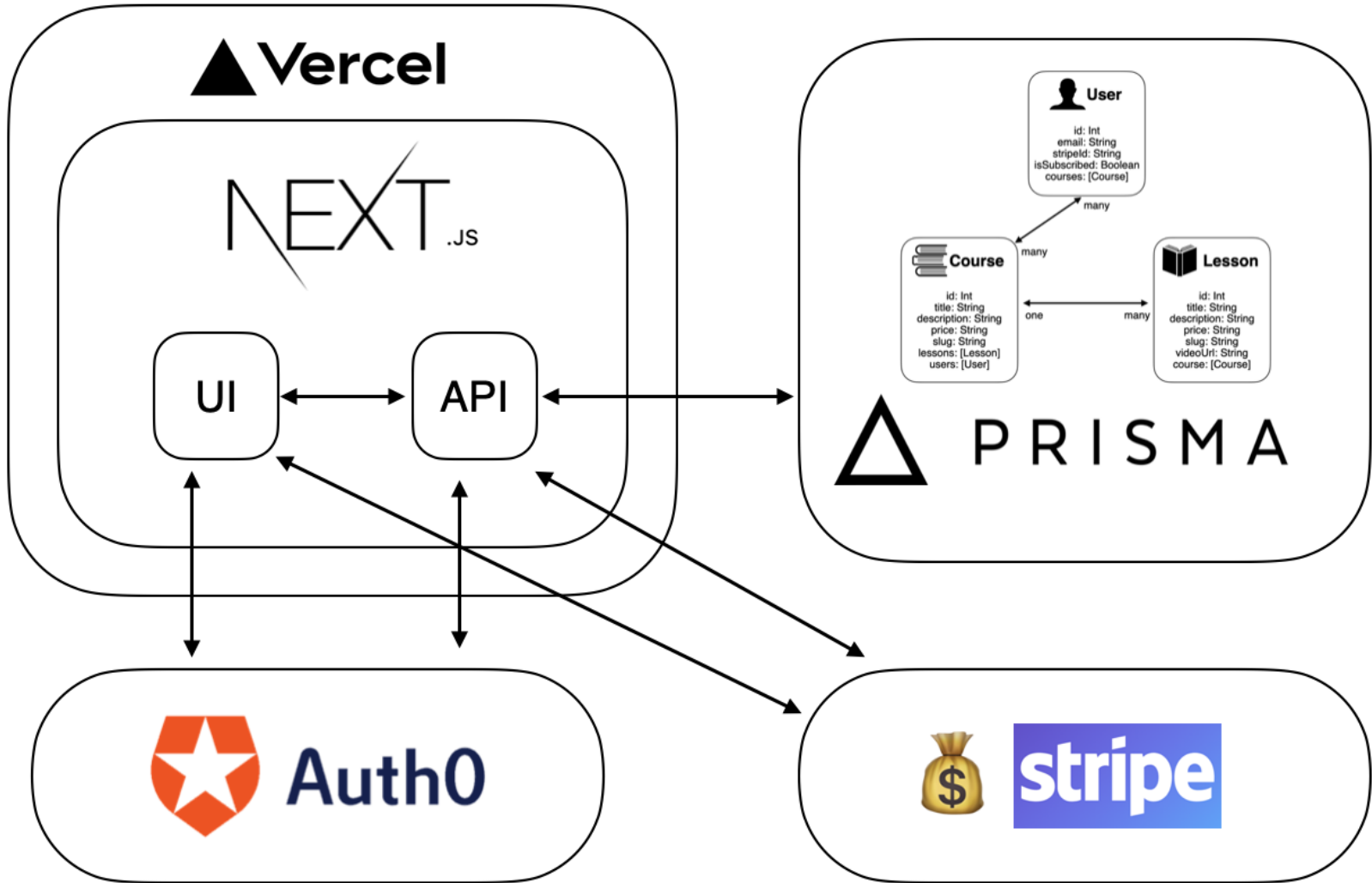


Main Topic



Sub Topic

Software as a Service : SaaS

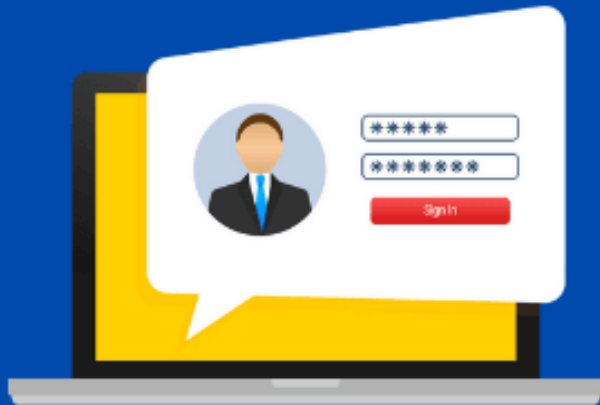


Auth... ?

Authentication

VS

Authorization



Who are you?
Verify the user's identity.



Can you do that?
Determine user permissions.



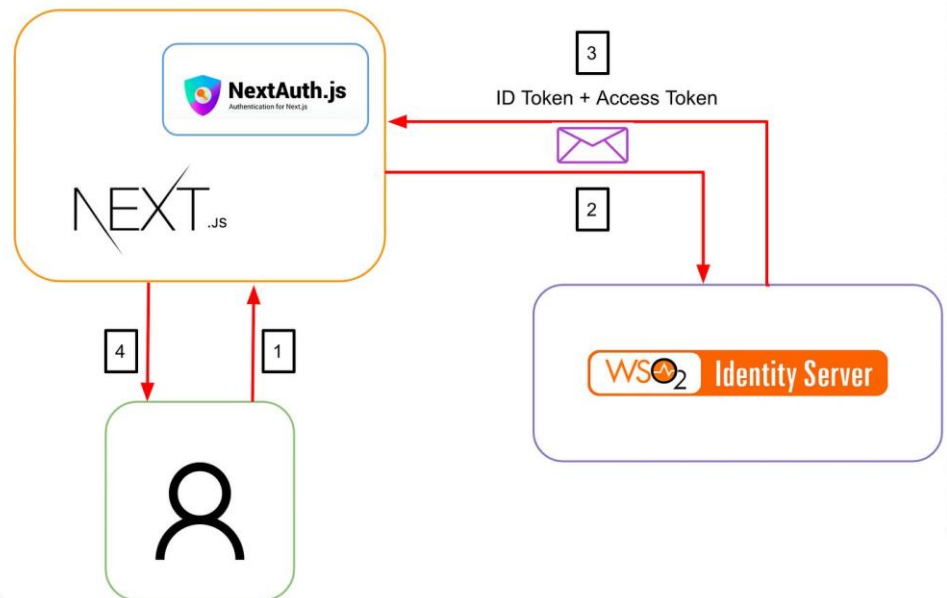
HEIMDAL™
SECURITY

Roles

NextAuth.js

About NextAuth.js (becoming to auth.js)

NextAuth.js is a complete open-source authentication solution for Next.js applications. It is designed from the ground up to support Next.js and Serverless.



NextAuth.js

Flexible and easy to use

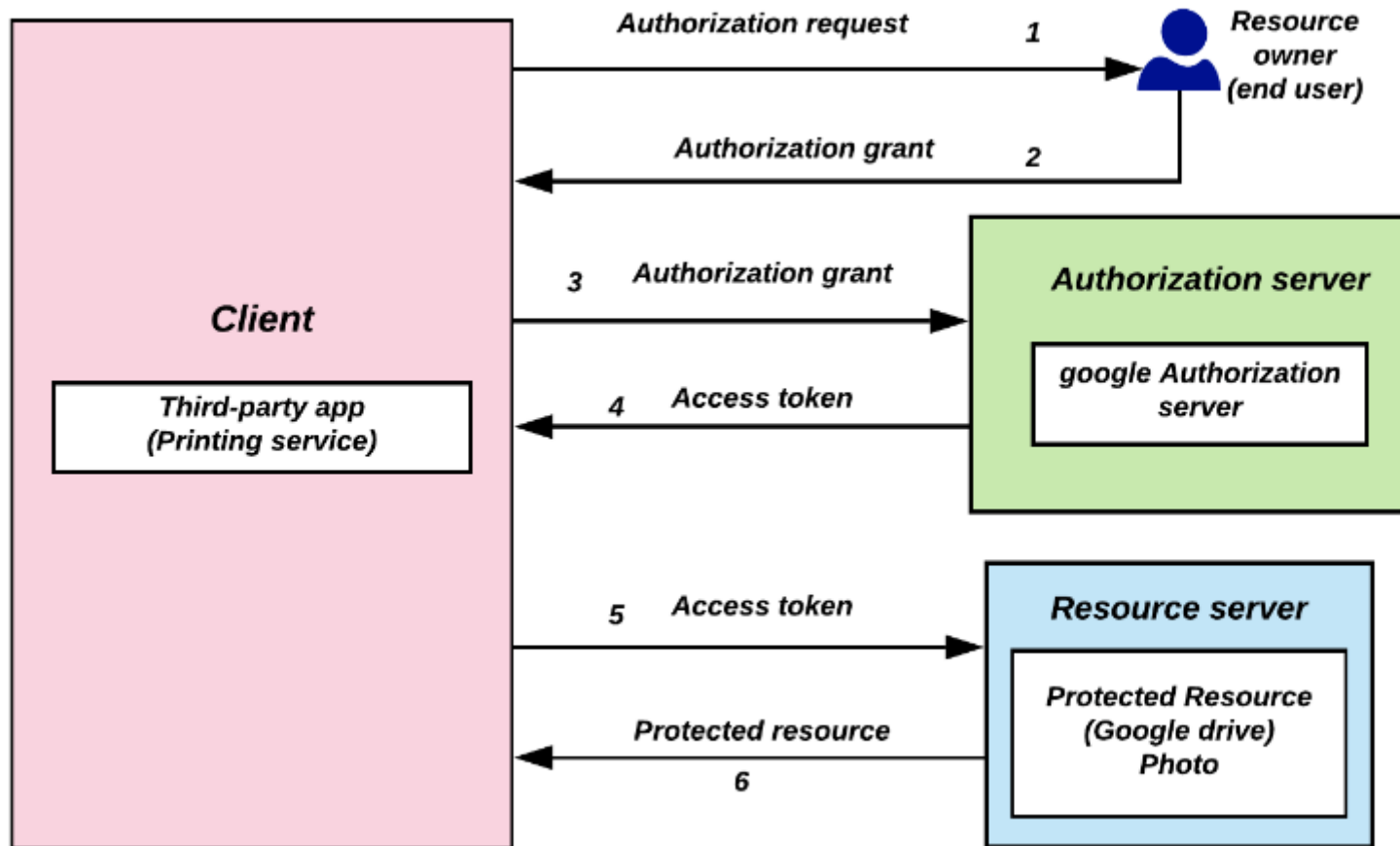
- Designed to work with any OAuth service, it supports [OAuth 1.0](#), [1.0A](#), [2.0](#) and [OpenID Connect](#)
- Built-in support for many popular sign-in services
- Supports email / passwordless authentication
- Supports stateless authentication with any backend (Active Directory, LDAP, etc)
- **Supports** both [JSON Web Tokens](#) and [database sessions](#)
- Designed for Serverless but runs anywhere (AWS Lambda, Docker, Heroku, etc...)

OAuth 2.0 (RFC 6749)

OAuth (short for "[Open Authorization](#)") is an open standard for access delegation, commonly used as a way for internet users to grant websites or applications access to their information on other websites but **without giving them the passwords**. This mechanism is used by companies such as Amazon, Google, Meta Platforms, Microsoft, and Twitter to permit users to share information about their accounts with third-party applications or websites.

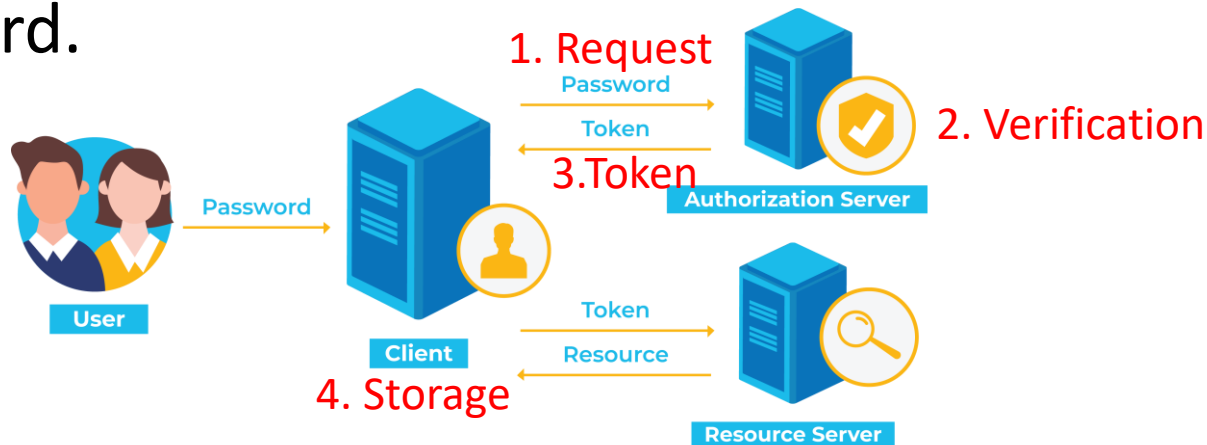


OAuth 2.0 Abstract flow

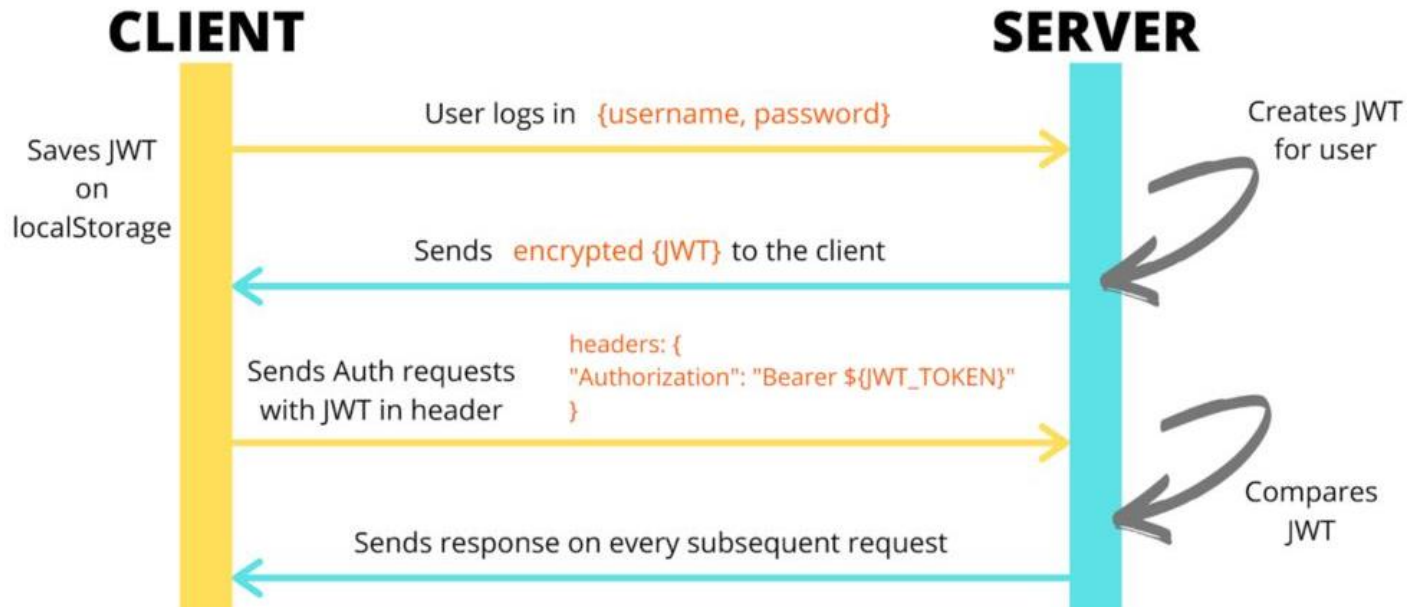


Token Base Authentication

Token-based authentication is a protocol that generates encrypted security tokens. It enables users to verify their identity to websites, which then generates a **unique encrypted authentication token**. That token provides users with access to protected pages and resources for a **limited period of time** without having to re-enter their username and password.



JSON Web Token (JWT)



```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODFiIn0.Y3ODkwliwibmFtZSI6IkpvaG9lIiwiaWF0IjoxNTE2MjM5MDIuXbPfbHMI6arZ3Y922BhjWgQzWXcXNrZ0ogtVhfEd2o
```

1 Header

```
{  "alg": "HS256",  "typ": "JWT"}
```

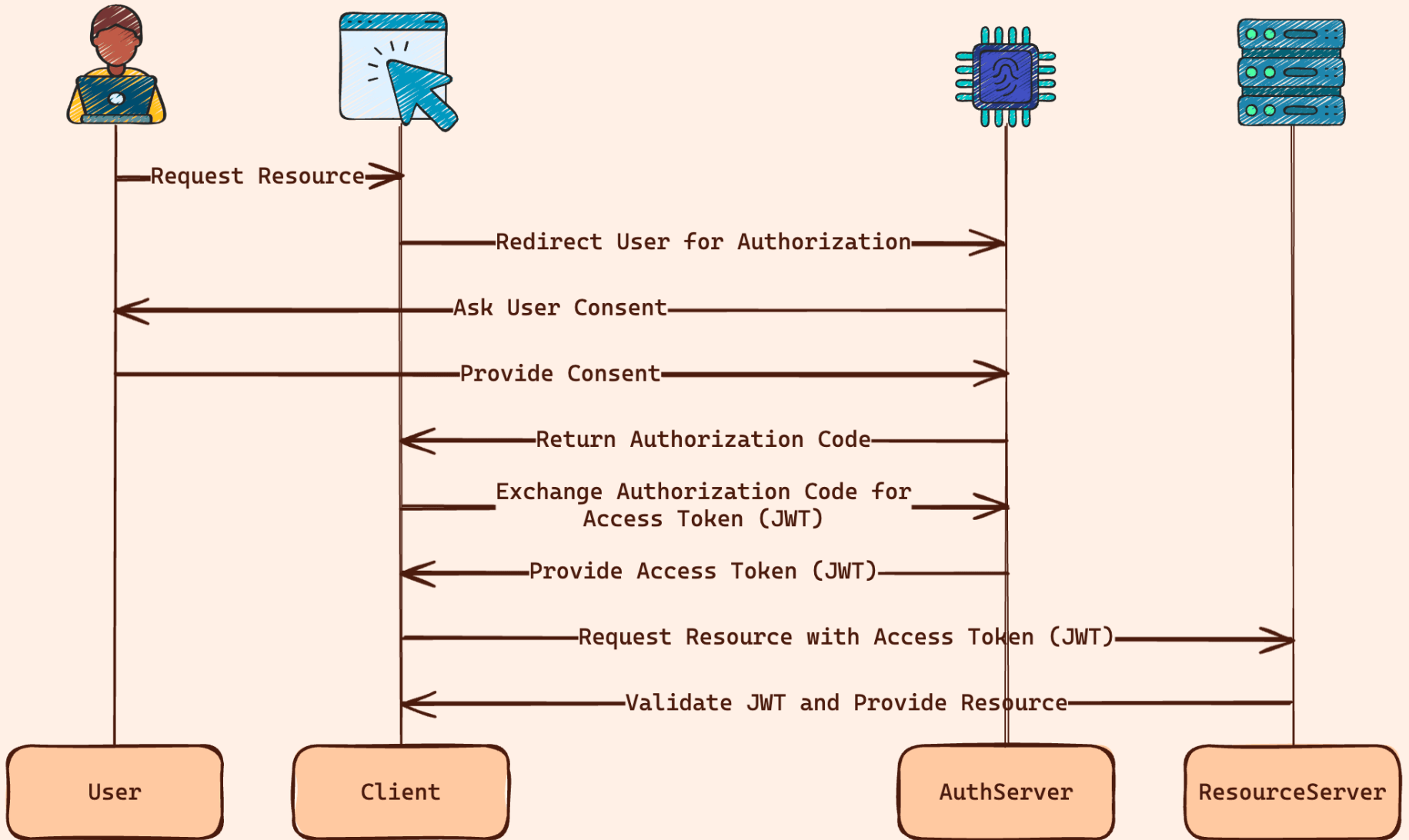
2 Payload

```
{  "sub": "1234567890",  "name": "John Doe",  "iat": 1516239022}
```

3 Signature

```
HMACSHA256 (  BASE64URL(header)  .  BASE64URL(payload) ,  secret)
```

JWT in OAuth flow



Essential Packages

```
✓ Would you like to use TypeScript? ... No / Yes
✓ Would you like to use ESLint? ... No / Yes
✓ Would you like to use Tailwind CSS? ... No / Yes
✓ Would you like to use `src/` directory? ... No / Yes
✓ Would you like to use App Router? (recommended) ... No / Yes
✓ Would you like to customize the default import alias (@/*)? ... No / Yes
```

~~NEXT~~.JS



Auth.js
Authentication for the Web.



Prisma

Get started

Create next app

```
>npx create-next-app@latest
```

Change work directory

```
>cd <your app>
```

Install prisma

```
>npm install prisma --save-dev
```

```
>npm install @prisma/client --save-dev
```

Setup prisma

```
>npx prisma init
```

.env

6

```
7 DATABASE_URL="mysql://root:@localhost:3306/auth"
```



.env

schema.prisma

```
7 generator client {
8   provider = "prisma-client-js"
9 }
10
11 datasource db {
12   provider = "mysql"
13   url      = env("DATABASE_URL")
14 }
15
16 model User{
17   id Int @id @default(autoincrement())
18   name String?
19   email String @unique
20   password String
21   createdAt DateTime @default(now())
22   updatedAt DateTime @updatedAt
23 }
```

prisma migrate

```
> npx prisma migrate dev --name init
```

to turn your database schema into a Prisma schema

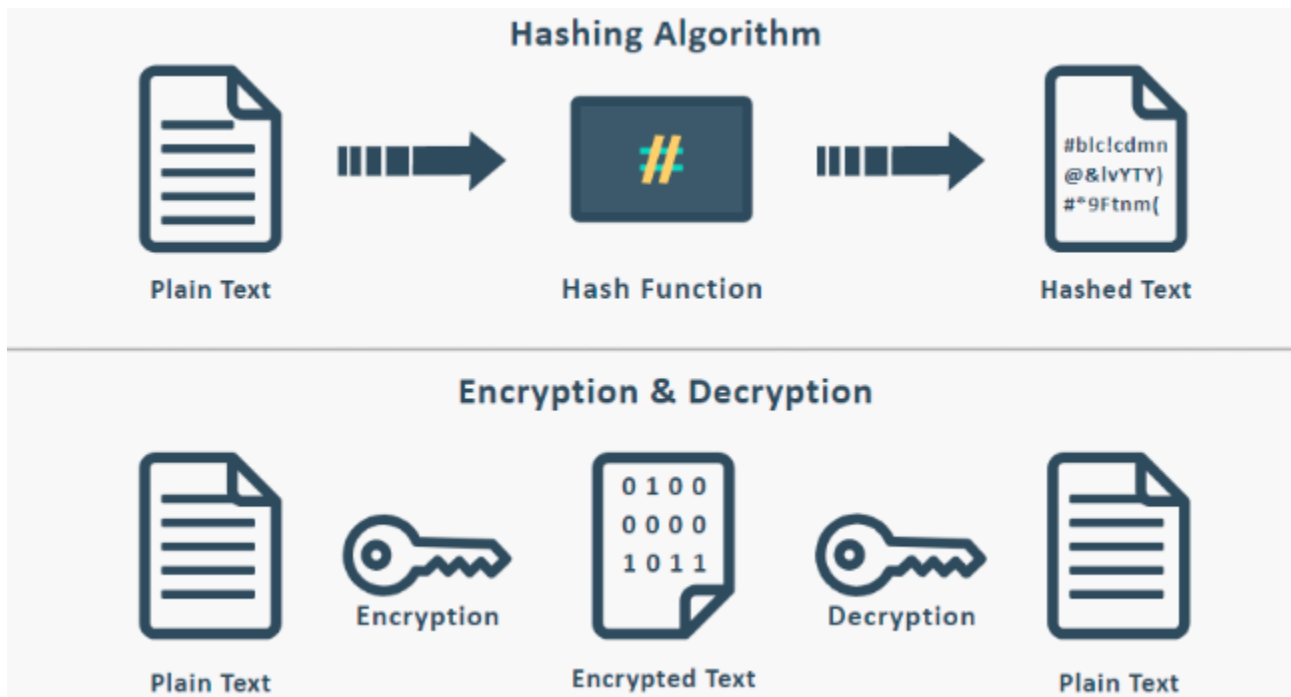
```
> npx prisma db pull
```

to generate Prisma client

```
> npx prisma generate
```

hash

A hash is a function that converts an input of letters and numbers into an encrypted output of a fixed length.



Salt (cryptography)

In cryptography, a salt is **random data fed** as an additional input to a one-way function that hashes data, a password or passphrase. **Salting helps defend against attacks** that use precomputed tables

Username	String to be hashed	Hashed value = SHA256
user1	password123	EF92B778BAFE771E89245B89ECBC08A44A4E166C06659911881F383D4473E94F
user2	password123	EF92B778BAFE771E89245B89ECBC08A44A4E166C06659911881F383D4473E94F

Instead, a salt is generated and appended to each password, which causes the resultant hash to output different values for the same original password.

Username	Salt value	String to be hashed	Hashed value = SHA256 (Password + Salt value)
user1	D;%yL9TS:5PaIS/d	password123D;%yL9TS:5PaIS/d	9C9B913EB1B6254F4737CE947EFD16F16E916F9D6EE5C1102A2002E48D4C88BD
user2)<,-<U(jLezy4j>*	password123)<,-<U(jLezy4j>*	6058B4EB46BD6487298B59440EC8E70EAE482239FF2B4E7CA69950DFBD5532F2

Sign up API (/api/auth/signup/route.js)

2

2

2

1

1

```
1 import { PrismaClient } from "@prisma/client";
2 import { hashSync } from "bcrypt";
3
4 const prisma = new PrismaClient();
5
6 export async function POST(req) {
7   try {
8     const { name, email, password } = await req.json();
9     const hashedPass = hashSync(password, 10);
10    const newUser = await prisma.user.create({
11      data: {
12        name,
13        email,
14        password: hashedPass,
15      },
16    });
17    return Response.json({
18      mgs: "user created!",
19      data: newUser,
20    });
21  } catch (err) {
22    return Response.json(err, { status: 500 });
23  }
24 }
```

step

No

Sign in

Next-Auth

```
> npm install next-auth --save-dev
```

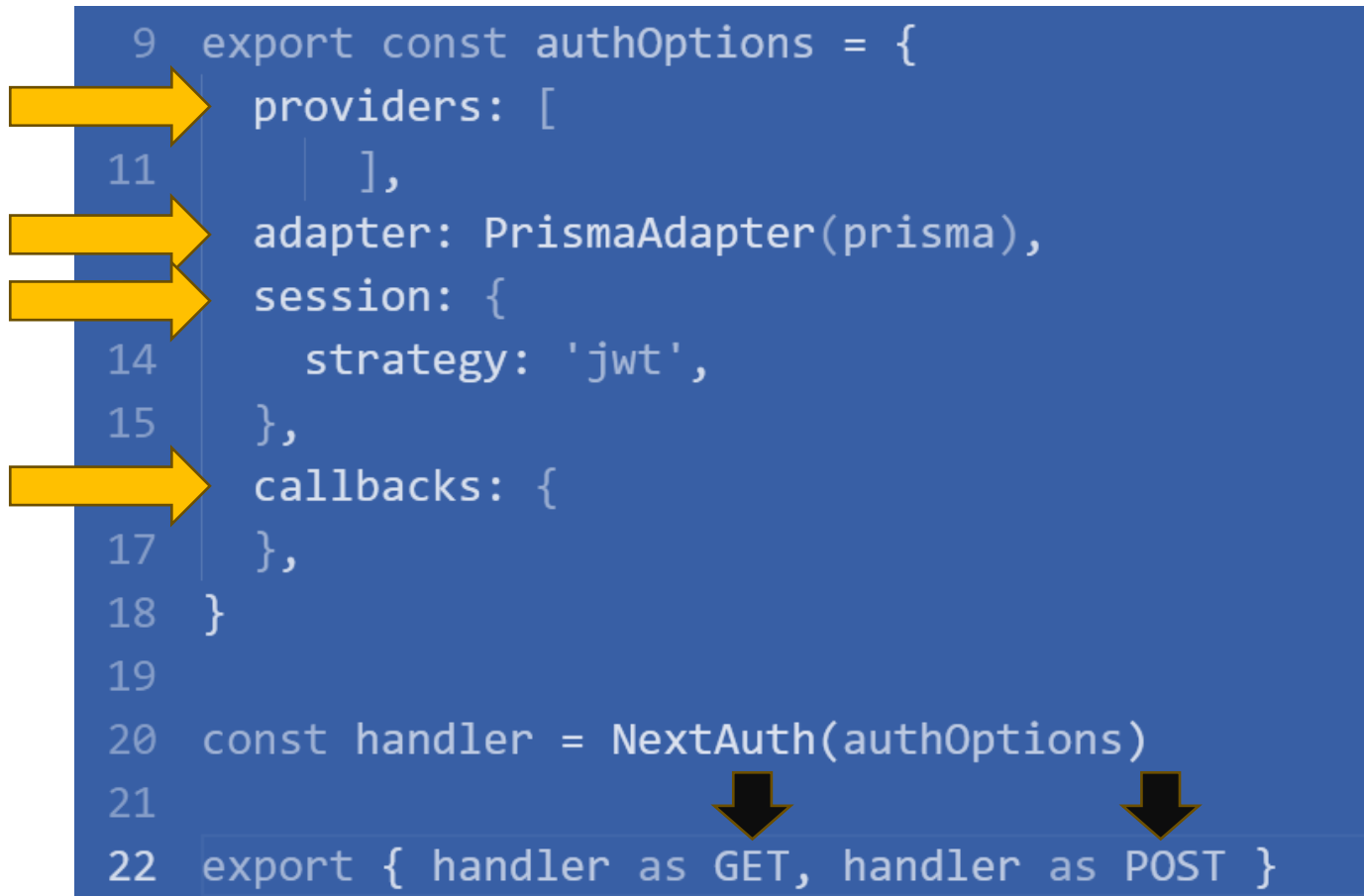
Adapter

```
> npm install @auth/prisma-adapter --save-dev
```

Sign in (API route structure)

/api/auth/[...nextauth]/route.js

```
9 export const authOptions = {  
10   providers: [  
11     ],  
12   adapter: PrismaAdapter(prisma),  
13   session: {  
14     strategy: 'jwt',  
15   },  
16   callbacks: {  
17     },  
18 }  
19  
20 const handler = NextAuth(authOptions)  
21  
22 export { handler as GET, handler as POST }
```



Providers

The **Credentials provider** allows you to handle signing in with arbitrary credentials, such as a username and password, two-factor authentication or hardware device (e.g. YubiKey U2F / FIDO).

```
import CredentialsProvider from "next-auth/providers/credentials"
...
providers: [
  CredentialsProvider({
    // The name to display on the sign in form (e.g. 'Sign in with...')
    name: 'Credentials',
    // The credentials is used to generate a suitable form on the sign in page.
    // You can specify whatever fields you are expecting to be submitted.
    // e.g. domain, username, password, 2FA token, etc.
    // You can pass any HTML attribute to the <input> tag through the object.
    credentials: {
      username: { label: "Username", type: "text", placeholder: "jsmith" },
      password: { label: "Password", type: "password" }
    },
    async authorize(credentials, req) {
      // You need to provide your own logic here that takes the credentials
      // submitted and returns either a object representing a user or value
      // that is false/null if the credentials are invalid.
      // e.g. return { id: 1, name: 'J Smith', email: 'jsmith@example.com' }
      // You can also use the `req` object to obtain additional parameters
      // (i.e., the request IP address)
      const res = await fetch("/your/endpoint", {
        method: 'POST'
```

Providers

Authentication

Providers in NextAuth.js are **OAuth** definitions that allow your users to sign in with their favorite preexisting logins. You can use any of our many predefined providers, or write your own custom OAuth configuration. Using a built-in OAuth Provider (e.g. GitHub, Twitter, Google, etc...) or Using a custom OAuth Provider

```
import Auth0Provider from "next-auth/providers/auth0"

Auth0Provider({
  clientId: process.env.CLIENT_ID,
  clientSecret: process.env.CLIENT_SECRET,
  issuer: process.env.ISSUER,
  authorization: { params: { scope: "openid your_custom_scope" } },
})
```

```
import GoogleProvider from "next-auth/providers/google"

GoogleProvider({
  clientId: process.env.GOOGLE_CLIENT_ID,
  clientSecret: process.env.GOOGLE_CLIENT_SECRET,
  profile(profile) {
    return {
      // Return all the profile information you need.
      // The only truly required field is `id`
      // to be able identify the account when added to a database
    }
  },
})
```

Callbacks

Callbacks are asynchronous functions you can use to control what happens when an action is performed. Callbacks are extremely powerful, especially in scenarios involving JSON Web Tokens as they allow you to implement access controls without a database and to integrate with external databases or APIs.

```
...
  callbacks: {
    async signIn({ user, account, profile, email, credentials }) {
      return true
    },
    async redirect({ url, baseUrl }) {
      return baseUrl
    },
    async session({ session, user, token }) {
      return session
    },
    async jwt({ token, user, account, profile, isNewUser }) {
      return token
    }
  }
  ...
}
```

Auth Code

/app/api/auth/[...nextauth]/route.js

```
1 import NextAuth from "next-auth"
2 import CredentialsProvider from "next-auth/providers/credentials"
3 import { PrismaClient } from "@prisma/client"
4 import { PrismaAdapter } from "@auth/prisma-adapter"
5 import bcrypt from 'bcrypt'
6
7 const prisma = new PrismaClient()
```



```
9   export const authOptions = {
10     providers: [
11       CredentialsProvider({
12         name: "Credentials",
13         credentials: {
14           email: { label: "Email", type: "email", placeholder: "user@bru.ac.th" },
15           password: { label: "Password", type: "password" },
16         },
17         async authorize(credentials, req) {
18           if (!credentials) return null;
19           const user = await prisma.user.findUnique({
20             where: { email: credentials.email },
21           });
22
23           if (
24             user &&
25             (await bcrypt.compare(credentials.password, user.password))
26           ) {
27             return {
28               id: user.id,
29               name: user.name,
30               email: user.email,
31             };
32           } else {
33             throw new Error("Invalid email or password");
34           }
35         },
36       }),
37     ],
38     adapter: PrismaAdapter(prisma),
```

```
38 adapter: PrismaAdapter(prisma),
39 session: {
40   strategy: "jwt",
41 },
42 callbacks: {
43   jwt: async ({ token, user }) => {
44     if (user) {
45       token.id = user.id;
46     }
47     return token;
48   },
49   session: async ({ session, token }) => {
50     if (session.user) {
51       session.user.id = token.id;
52     }
53     return session;
54   },
55 },
56 };
57
58 const handler = NextAuth(authOptions);
59
60 export { handler as GET, handler as POST };
```

Elements Recorder Console Sources **Network** Performance Memory >>
2 1 ⚙️ ⋮

Preserve log Disable cache No throttling ⌵ 📶 ⬆️ ⬇️

Filter Invert Hide data URLs Hide extension URLs

All Fetch/XHR Doc CSS JS Font Img Media Manifest WS Wasm Other
 Blocked response cookies Blocked requests

3rd-party requests

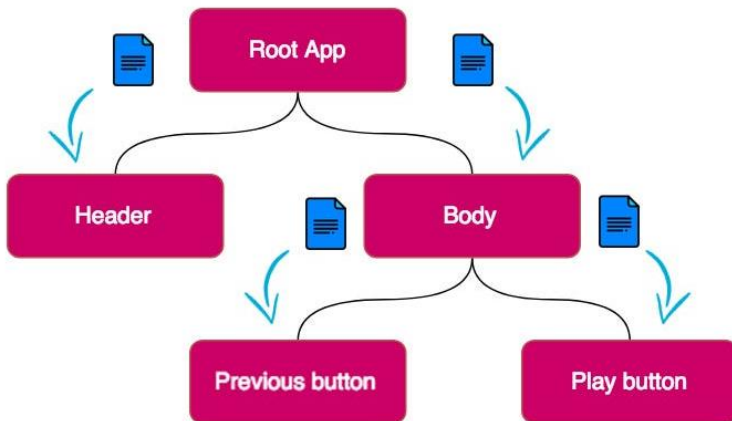
10 ms	20 ms	30 ms	40 ms	50 ms	60 ms	70 ms	80 ms	90 ms	100 ms
-------	-------	-------	-------	-------	-------	-------	-------	-------	--------

Name	Headers	Payload	Preview	Response	Initiator	Timing	Cookies
localhost	▼ General						
webpack.js?v=1719824353345	Request URL:	http://localhost:3000/api/auth/callback/credentials					
main-app.js?v=1719824353345	Request Method:	POST					
app-pages-internals.js	Status Code:	● 401 Unauthorized					
page.js	Remote Address:	[::1]:3000					
layout.js	Referrer Policy:	strict-origin-when-cross-origin					
styles-760a7b596bc0696605d...	▼ Response Headers <input type="checkbox"/> Raw						
asset-manifest.json	Connection:	keep-alive					
session	Content-Type:	application/json					
webpack-hmr	Date:	Mon, 01 Jul 2024 08:59:22 GMT					
favicon.ico	Keep-Alive:	timeout=5					
providers	Transfer-Encoding:	chunked					
csrf	Vary:	RSC, Next-Router-State-Tree, Next-Router-Prefetch					
credentials							

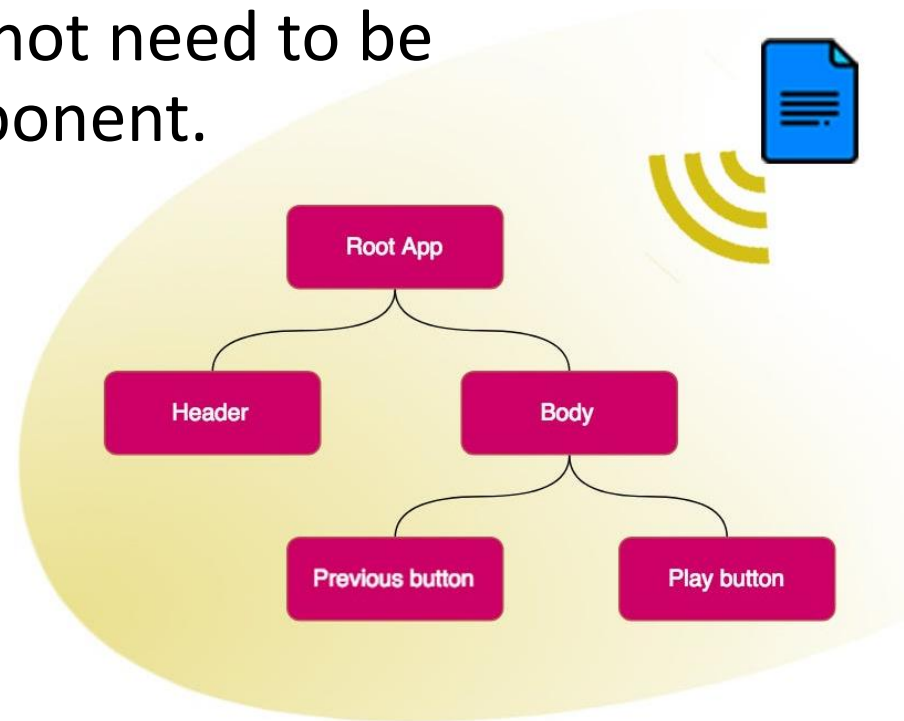
14 requests | 1.7 MB transferred

Context API (React)

To be able to run the **Session** from anywhere, call the **SessionProvider component** from the Layout, but so that the Layout does not need to be converted into a Client component.



Passing prop to each level



The data in Context available to every component

SessionProvider

/app/components/SessionProvider.js

```
1 'use client'  
2 import { SessionProvider } from 'next-auth/react'  
3 export default SessionProvider
```



<Client Component>

<Server Component/>

</Client Component>

Layout

/app/layout.js

```
1 import SessionProvider from "../components/SessionProvider";
2 import { getServerSession } from "next-auth";
3
4 export default async function RootLayout({ children }) {
5   const session = await getServerSession();
6   return (
7     <html lang="en">
8       <body>
9         <SessionProvider session={session}>{children}</SessionProvider>
10      </body>
11    </html>
12  );
13 }
14
```

Home

/app/page.js

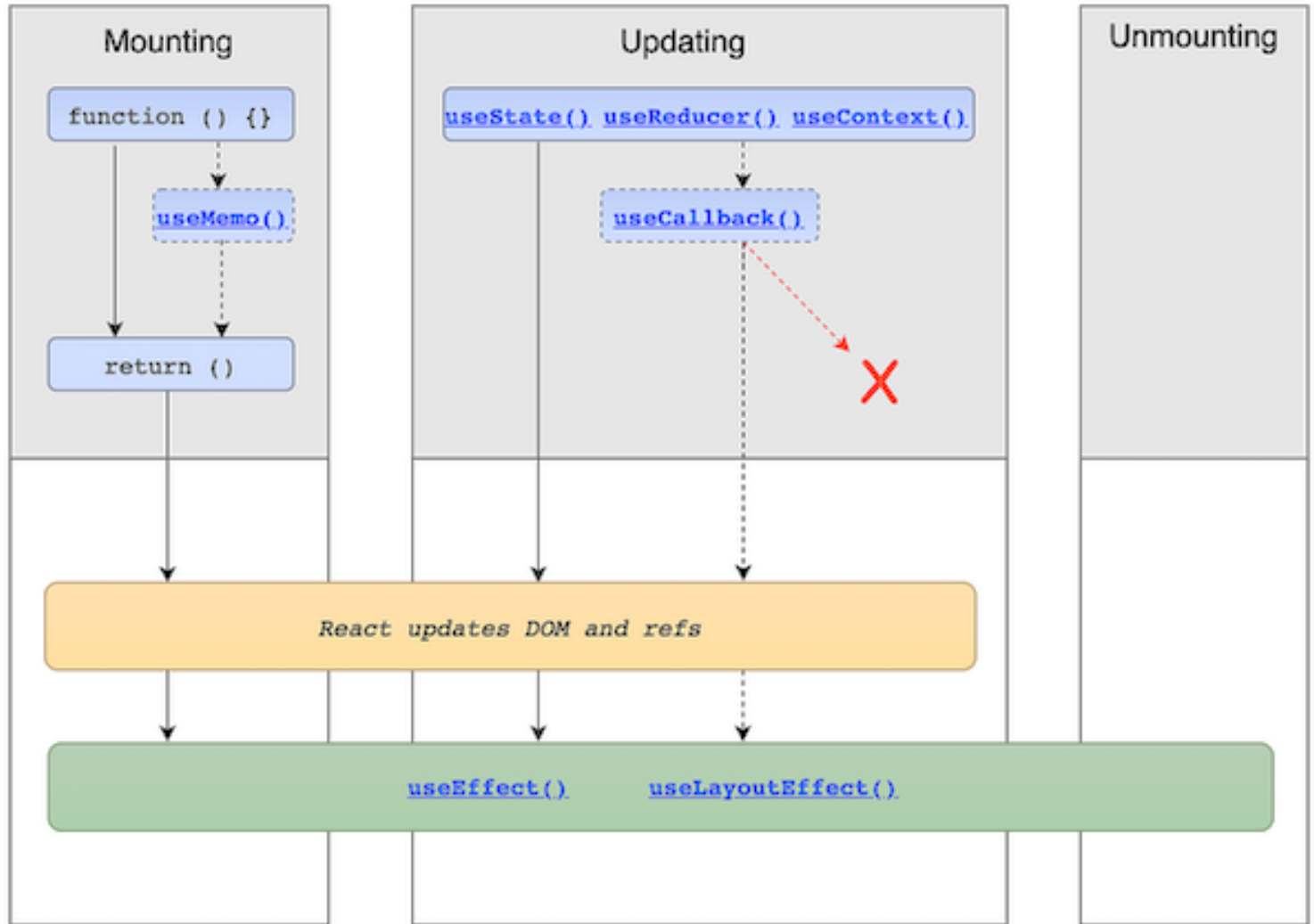
```
1 "use client";
2
3 import { useState } from "react";
4 import { signIn } from "next-auth/react";
5 import { useRouter } from "next/navigation";
6
7 export default function SignIn() {
8   const [email, setEmail] = useState("");
9   const [password, setPassword] = useState("");
10  const router = useRouter();
11
12  const handleSubmit = async (e) => {
13    e.preventDefault();
14    try {
15      const result = await signIn("credentials", {
16        redirect: false,
17        email,
18        password,
19      });
20
21      if (result.error) {
22        console.error(result.error);
23      } else {
24        router.push("/profile");
25      }
26    } catch (error) {
27      console.log("error", error);
28    }
29  };
30 }
```

```
30
31 return (
32   <div>
33     <form onSubmit={handleSubmit}>
34       <div>
35         <label htmlFor="email">Email</label>
36         <input
37           id="email"
38           type="email"
39           value={email}
40           onChange={(e) => setEmail(e.target.value)}
41           required
42         />
43       </div>
44       <div>
45         <label htmlFor="password">Password</label>
46         <input
47           id="password"
48           type="password"
49           value={password}
50           onChange={(e) => setPassword(e.target.value)}
51           required
52         />
53       </div>
54       <button type="submit">Sign In</button>{" "}
55     </form>
56   </div>
57 );
58 }
59
```


React Hook Lifecycle

"Render phase"

Pure and has no side effects. May be paused, aborted or restarted by React



"Commit phase"

Can work with DOM, run side effects, schedule updates.

Profile file

```
1 "use client";
2 import { useSession, signOut } from "next-auth/react";
3 import { useRouter } from "next/navigation";
4 import { useEffect } from "react";
5
6 export default function Profile() {
7   const { data: session, status } = useSession();
8   const router = useRouter();
9
10  useEffect(() => {
11    if (status === "unauthenticated") {
12      router.push("/");
13    }
14  }, [status, router]);
15
16  return (
17    status === "authenticated" &&
18    session.user && (
19      <div>
20        <div>
21          <p>
22            Welcome, <b>{session.user.name}</b>
23          </p>
24          <p>Email: {session.user.email}</p>
25          <button onClick={() => signOut({ callbackUrl: "/" })}>Logout</button>
26        </div>
27      </div>
28    )
29  );
30 }
```

/app/profile/page.js

Name	Headers	Payload	Preview	Response	Initiator	Timing	Cookies
<ul style="list-style-type: none"> styles-760a7b596bc069660... asset-manifest.json session webpack-hmr 1d6bf32dba3bbbed.webpac... webpack.1d6bf32dba3bbbe... providers csrf credentials session profile?_rsc=1wtp7 profile?_rsc=1got7 page.js asset-manifest.json 	<p>Request URL: http://localhost:3000/api/auth/callback/credentials</p> <p>Request Method: POST</p> <p>Status Code: 200 OK</p> <p>Remote Address: [::1]:3000</p> <p>Referrer Policy: strict-origin-when-cross-origin</p> <p>Response Headers</p> <p>Connection: keep-alive</p> <p>Content-Type: application/json</p> <p>Date: Mon, 01 Jul 2024 09:05:54 GMT</p> <p>Keep-Alive: timeout=5</p> <p>Set-Cookie: next-auth.session-token=evJhbGciOiJkaXIiLCJlbmMiOiJBMiU2R0NNIn0..kFmrRIEfoNH</p>						

Name	Headers	Preview	Response	Initiator	Timing	Cookies
<ul style="list-style-type: none"> styles-760a7b596bc069660... asset-manifest.json session webpack-hmr 1d6bf32dba3bbbed.webpac... webpack.1d6bf32dba3bbbe... providers csrf credentials session profile?_rsc=1wtp7 profile?_rsc=1got7 page.js asset-manifest.json 	<p>Request URL: http://localhost:3000/api/auth/session</p> <p>Request Method: GET</p> <p>Status Code: 200 OK</p> <p>Remote Address: [::1]:3000</p> <p>Referrer Policy: strict-origin-when-cross-origin</p> <p>Response Headers</p> <p>Connection: keep-alive</p> <p>Content-Type: application/json</p> <p>Date: Mon, 01 Jul 2024 09:05:54 GMT</p> <p>Keep-Alive: timeout=5</p> <p>Set-Cookie: next-auth.session-token=eyJhbGciOiJkaXIiLCJlbmMiOiJBMiU2R0NNIn0..TzKgOp9upN3xRC4E.Ph5o8Xsa8JNq3tft56hMOjl2m5oH3F_ryEX6JDHWZHa1szX55</p>					

20 requests | 1.7 MB transferred

Create roles



Let's Try

Guideline

1. Which attribute is in the database?
2. What variable in token and session?
3. How to use role?

Q & A