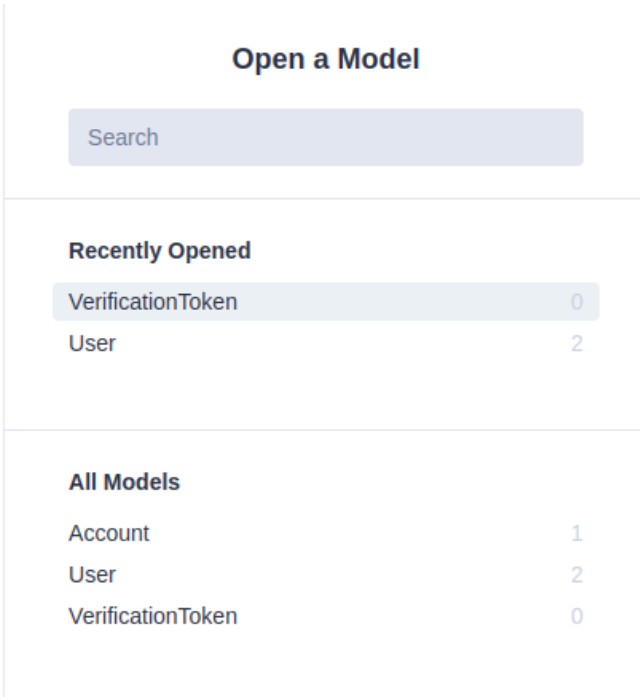
Next Auth

Email Verification for Credentials Registration

```
model VerificationToken {
 id String @id @default(cuid())
 email String
 token String @unique
 expires DateTime

 // Only one unique token for a email
 @@unique([email,token])
}
```

Shut down app , npx prisma generate , npx prisma db push

We can verify in the prisma studio



data/verificationToken

```
import { db } from "@/lib/db";
```

```
export const getVerificationTokenByToken = async (
   token:string
) => {
   try{
       const verificationToken = await db.verificationToken.findUnique({
           // using email to search
           where: {token}
       });
       return verificationToken;
   }catch{
       return null;
   }
}

export const getVerificationTokenByEmail = async (
   email:string
) => {
   try{
       const verificationToken = await db.verificationToken.findFirst({
           // using email to search
           where: {email}
       });
       return verificationToken;
   }catch{
       return null;
   }
}
```

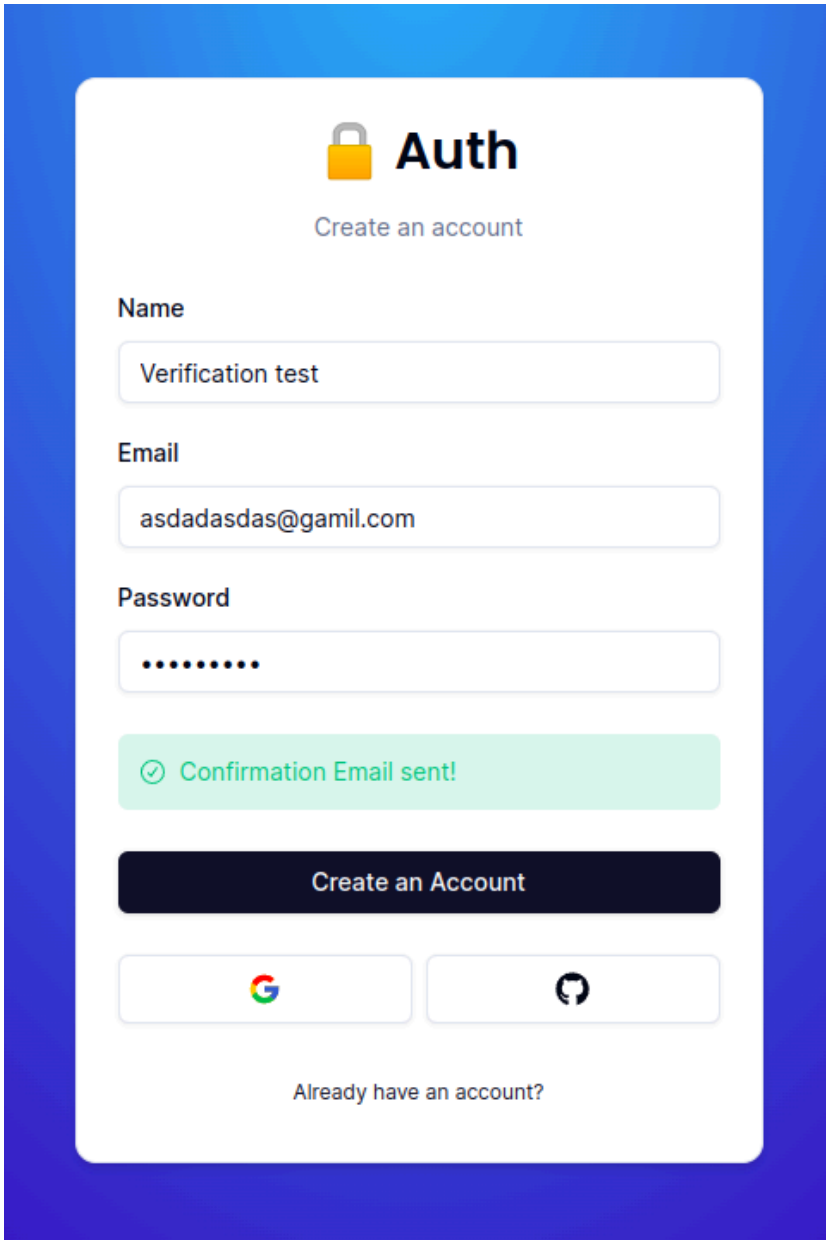
Lets create a lib which will be use to generate these token and if existing token exist then it get removed

npm i –save-dev @types/uuid

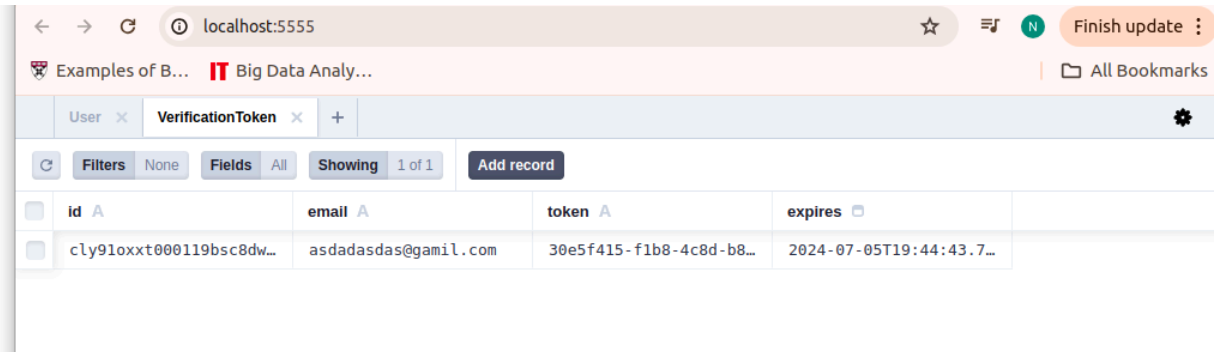
```
import { getVerificationTokenByEmail } from '@/data/verificationToken';
import { v4 as uuidv4 } from 'uuid';
import { db } from './db';
```

```
// now find a place where we going to run this function
export const getVerificationToken = async (email:string) => {
   const token = uuidv4();
   // 1hr from now
   const expires = new Date(new Date().getTime() + 3600*1000);

   // check if existing token sent for this email
   const existingToken = await getVerificationTokenByEmail(email);

   if(existingToken){
       await db.verificationToken.delete({
           where:{
               id:existingToken.id,
           }
       })
   }
   const verificationToken = await db.verificationToken.create({
       data:{
           email,
           token,
           expires,
       }
   });
   return verificationToken;
}
```

Lets go to the register.ts



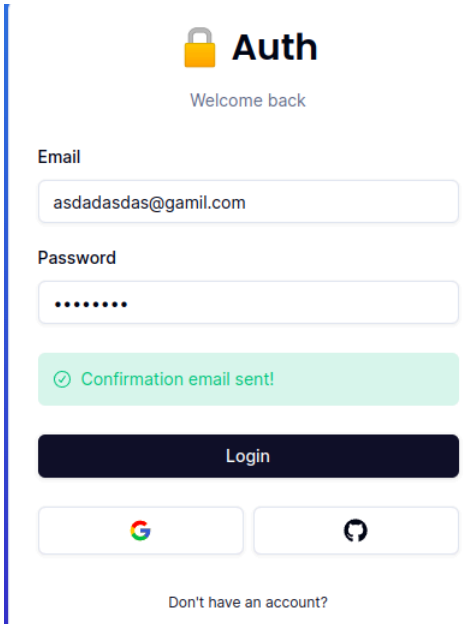
Now we can see that data exist in the Verification Token



Now before generating the token in Register we should check whether email is verified or not in the login section. And we will send them email again

login.ts

```
// since we are using promises
export const login = async (values: z.infer<typeof LoginSchema>) => {
 const validateFields = LoginSchema.safeParse(values);
 if (!validateFields.success) {
   return { error: "Invalid fields" };
 }

 const { email, password } = validateFields.data;
 const existingUser = await getUserByEmail(email);

 // This means they should login using OAuth
 if(!existingUser || !existingUser.email || !existingUser.password){
   return {error:"Email does not Exist!"}
 }

 if(!existingUser.emailVerified){
   const verificationToken = await
getVerificationToken(existingUser.email);
   return {success:"Confirmation email sent!"};
   // Users can still use api to sign in hence we also need to protect it
therw
 }
```



And now if we refresh in the prisma studio we will find that the token has been refreshed and we have created a new token expiry date

Sometime auth can redirect to auth/error and hence we need to do the equivalent to it in callbacks also

auth.ts

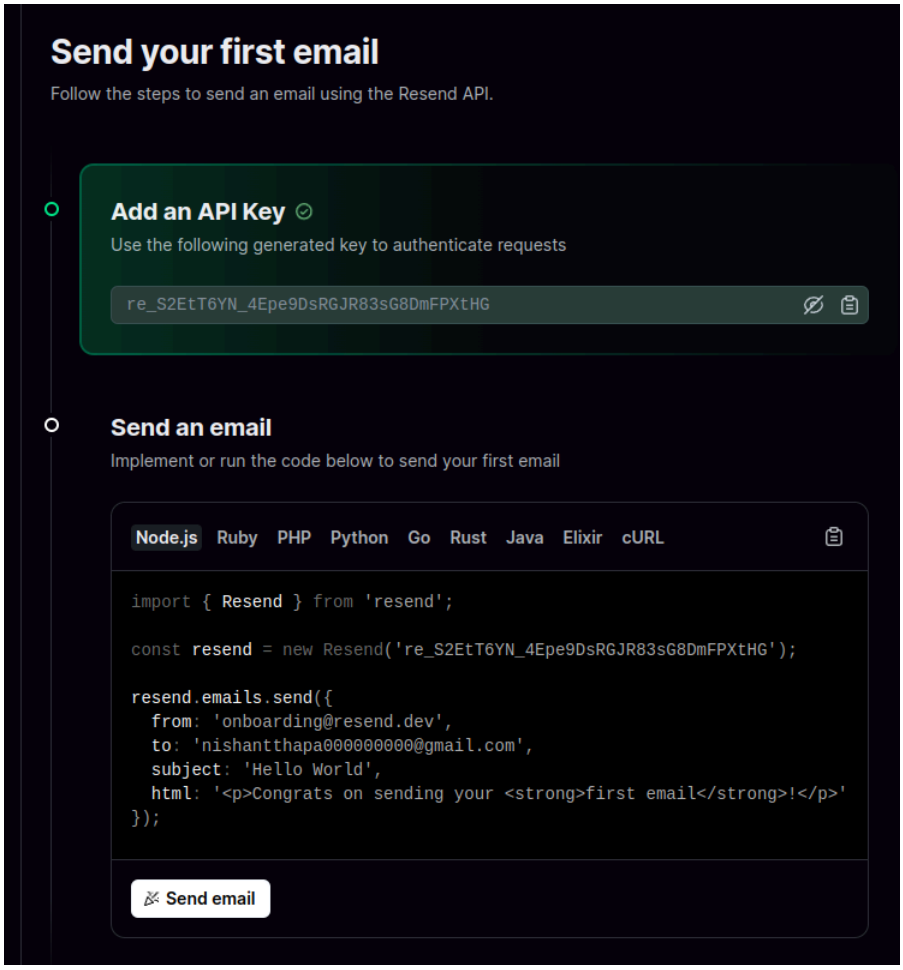
```
    async signIn({user,account}){
      // Allow OAuth without email verifications

      if(account?.provider !== "credentials") return true;

      const existingUser = await getUserById(user.id);

      // Prevent sign in without email verification
      if(!existingUser?.emailVerified) return false;

      // TODO add 2FA check

      return true;
      // allowing to sign in
```

```
    },
```

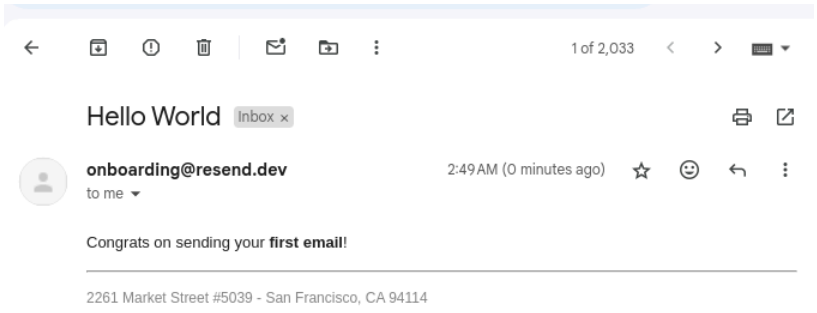
Using **resend**

Login to it, Create a team and then go through onboarding

Add an API Key



And if we click on the send email we will get the email



Go in Docs and see the NextJS quick start

https://resend.com/docs/send-with-nextjs

API Key: re_S2EtT6YN_4Epe9DsRGJR83sG8DmFPXtHG

npm install resend

Currently we are able to send the email to the us only to be able to send emails to other we need our domain

Now lets create a lib/mail.ts

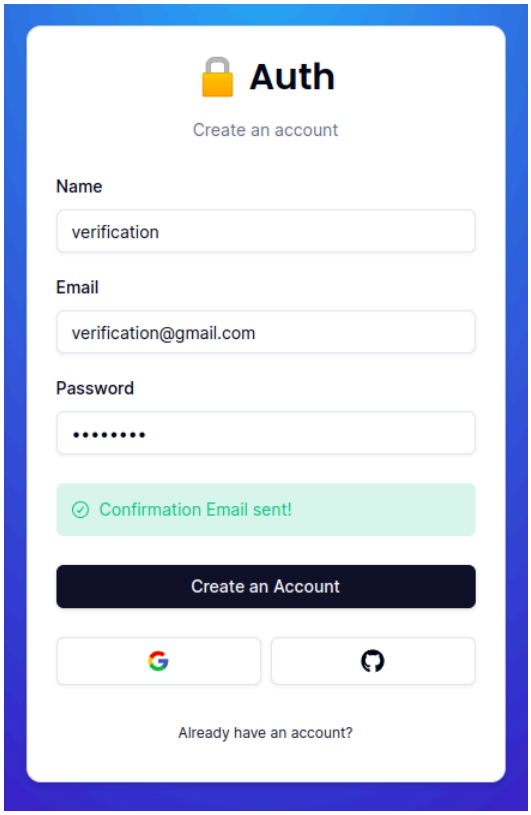
```
import {Resend} from "resend";

const resend = new Resend(process.env.RESEND_API_KEY);

export const sendVerificationEmail = async (
   email:string,
   token:string
)=>{
   const confirmLink
=`http://localhost:3000/auth/new-verification?token=${token}`;
   // new-verification page will be use to verify if everything is okay
   // In production we have to change it(LInk)

   await resend.emails.send({
       from:"onboarding@resend.dev",
       to:email,
       subject:"Confirm your email",
       html:`<p>Click <a href="${confirmLink}">here</a> to confirm the
email.</p>`
   })
}
```

Now lets go to register.ts

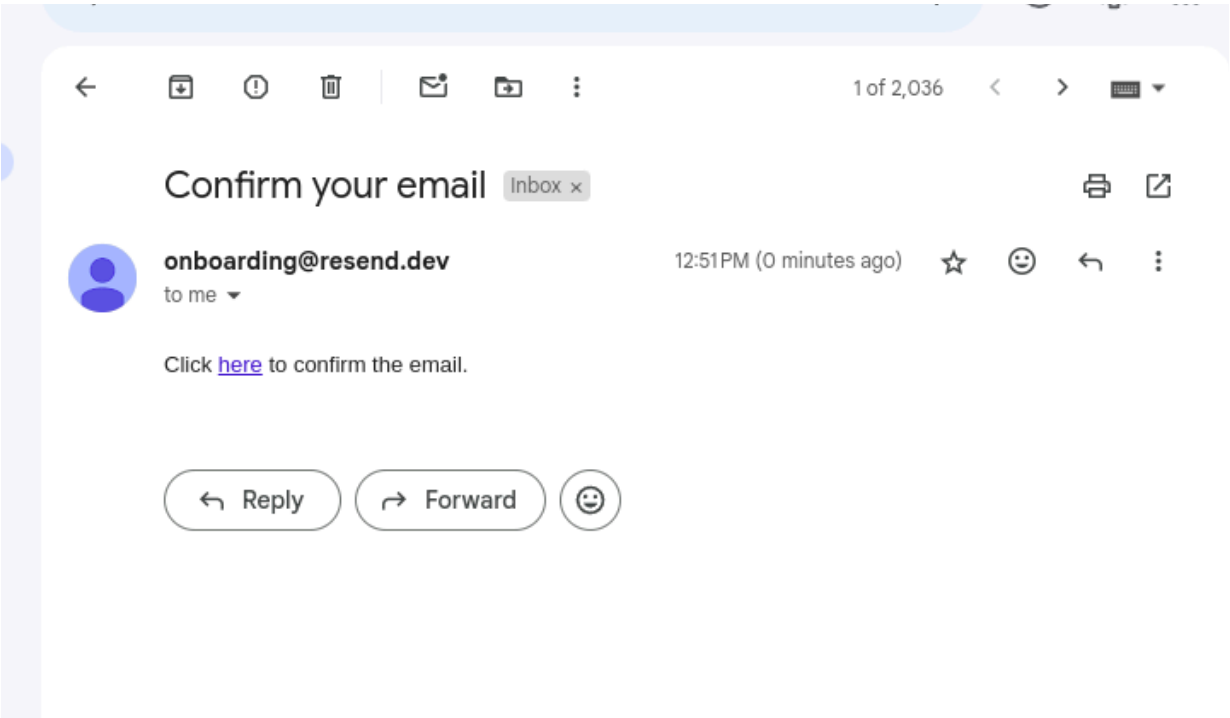
```
   const verificationToken  = await getVerificationToken(email);
   await sendVerificationEmail(
```

```
      verificationToken.email,
      verificationToken.token,
  )
```

Check test this, Create a new email



But we have to create email with login in resend, since other email wont work unless we work on domain



Now also send email in login functionality,

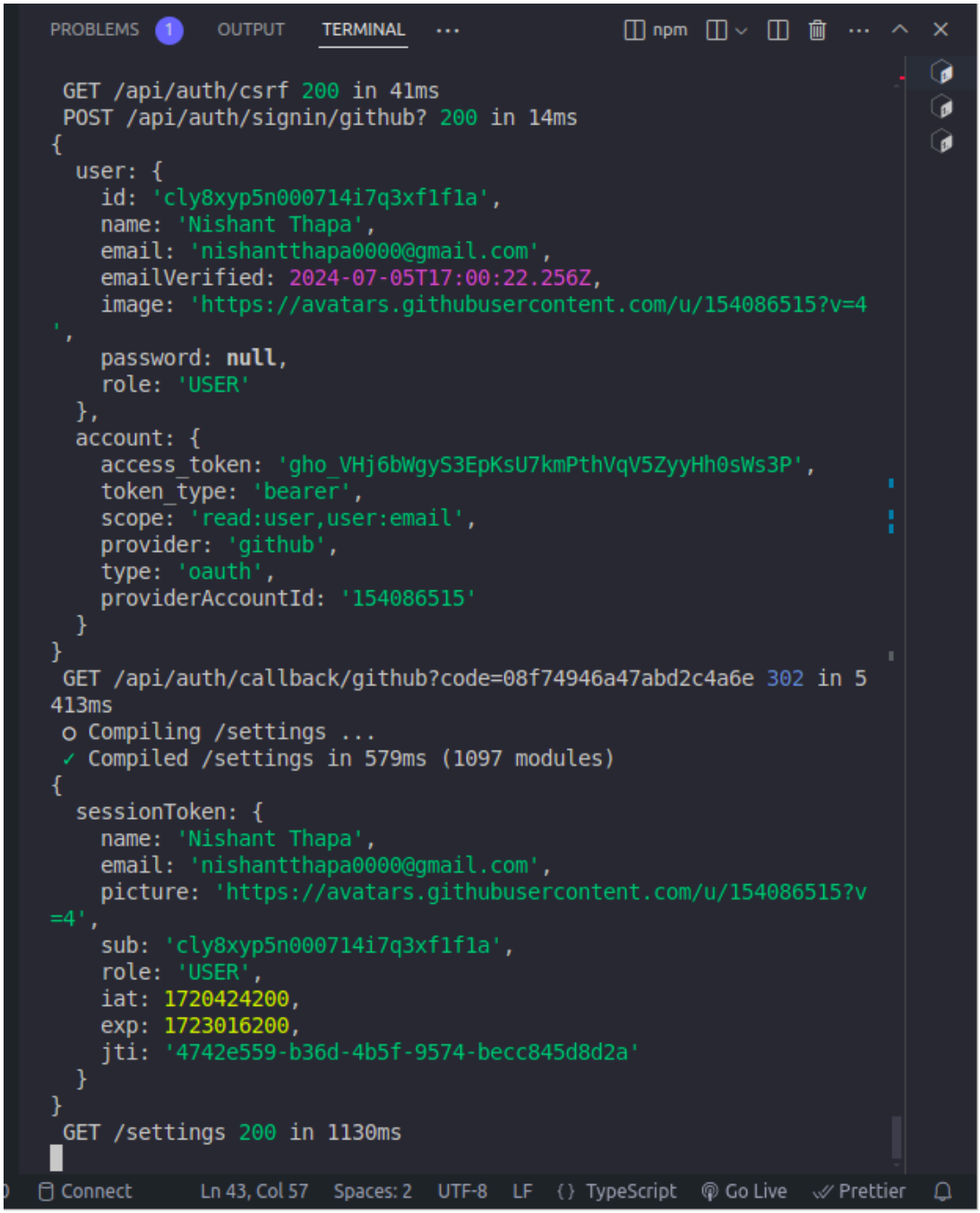
```
 if(!existingUser.emailVerified){
   const verificationToken = await
getVerificationToken(existingUser.email);

   await sendVerificationEmail(
     verificationToken.email,
     verificationToken.token,
   )

   return {success:"Confirmation email sent!"};
   // Users can still use api to sign in hence we also need to protect it
therw
 }
```

Now if we try to login with not verified email then it will send us a confirmation email

Repository till this point:

https://github.com/nthapa000/authentication_next/tree/a357cf1f39deb43fb98bfab69ff9f55ecf6df065

Login with Github we get following info:



Now lets Login with Credentials but first comment out the login.ts

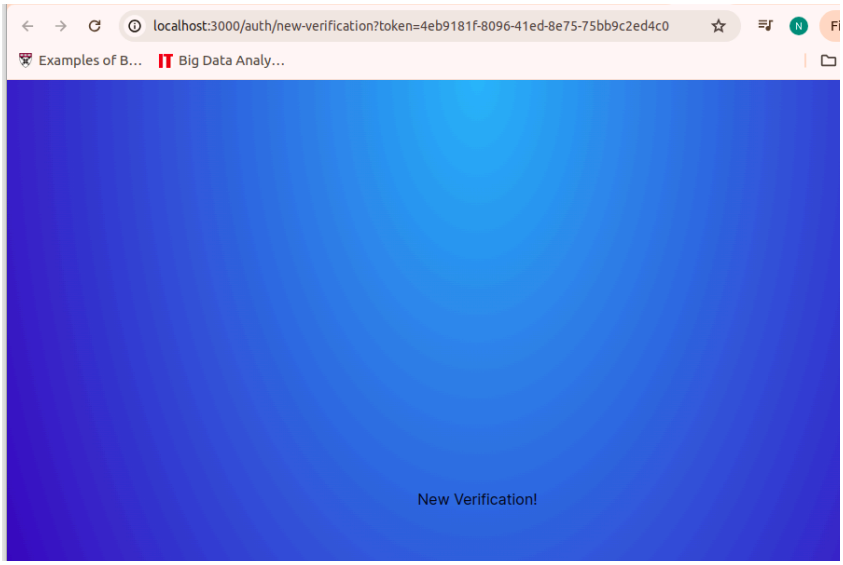
Currently what happening is that if user try to Confirm the email it is getting directed to the login page so insert the route in **route.ts**

Email Verification

We will create auth/new-verification

```
const NewVerificationPage = () => {
    return (
        <div>
            <NewVerificationForm />
        </div>
    )
}

export default NewVerificationPage;
```

And we will be redirected to this page when we click on verify

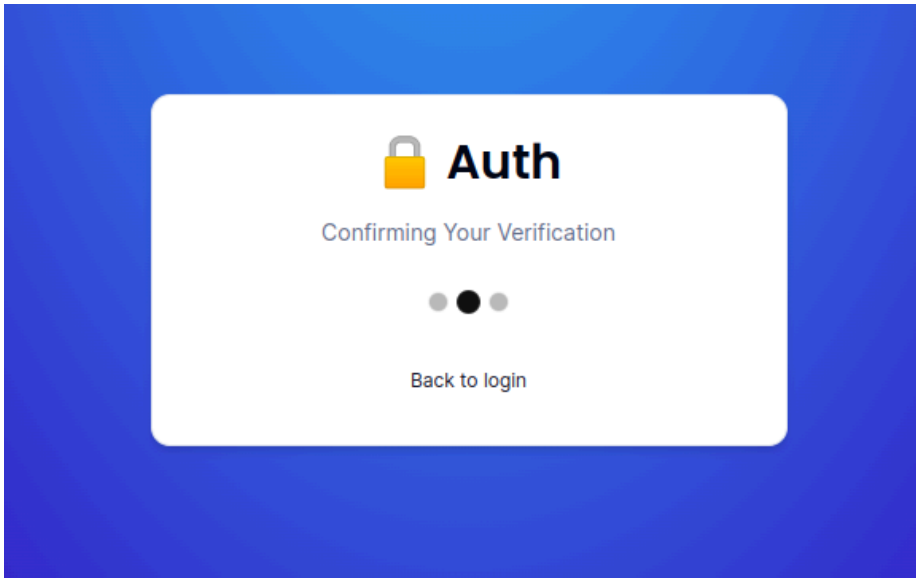


We want to add a loader for which we need React Spinner

npm i react-spinners

components/auth/new-verification-form.tsx

```tsx
"use client"

import { useCallback, useEffect } from "react"
import { BeatLoader} from "react-spinners"
import { CardWrapper } from "./card-wrapper"
import { useSearchParams } from "next/navigation"

export const NewVerificationForm = () =>{
   const searchParams = useSearchParams();
   const token = searchParams.get("token");
   const onSubmit = useCallback(() =>{
       console.log(token);
       // in Browser we can see the token being the logging
   },[token]);

   useEffect(()=>{
       onSubmit();
   },[onSubmit])

   return(
       <CardWrapper
           headerLabel="Confirming Your Verification"
           backButtonLabel="Back to login"
           backButtonHref="/auth/login"
       >
           <div className="flex items-center w-full justify-center">
               <BeatLoader/>
           </div>

       </CardWrapper>
   )
}
```



Now action/new-verfication.ts

```
"use server";

import { db } from "@/lib/db";
import { getUserByEmail } from "@/data/user";
import { getVerificationTokenByToken } from "@/data/verificationToken";

export const newVerification = async (token:string)=>{
   const existingToken = await getVerificationTokenByToken(token);
   if(!existingToken){
       return {error:"Token does not exist!!"};
   }
   const hasExpired = new Date(existingToken.expires) < new Date();

   if(hasExpired){
       return {error:"Token has expired"};
   }

   // Find user whom we got to validate
   const existingUser  = await getUserByEmail(existingToken.email);
   if(!existingUser){
       return {error:"Email does not exist!"};
   }
```

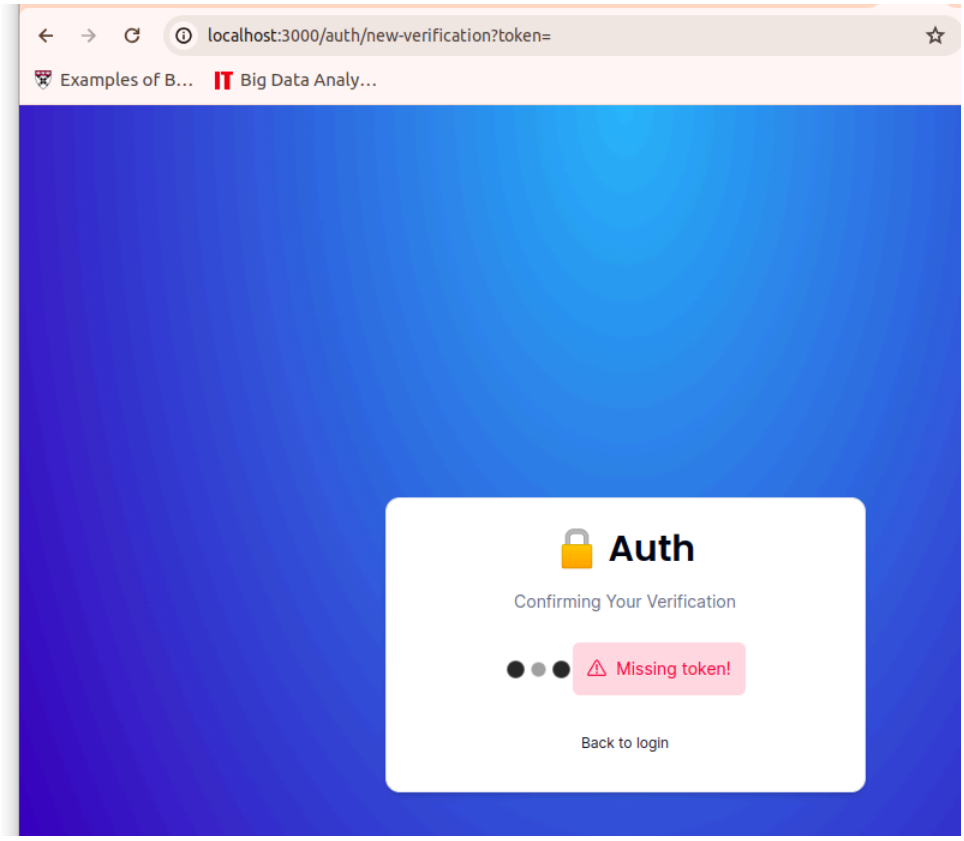
```
    await db.user.update({
        where:{id:existingUser.id},
        data:{
            emailVerified: new Date(),
            // Updating email value in future
            email: existingToken.email
        }
    })
    await db.verificationToken.delete({
        where:{id:existingToken.id}
    })

    return {success:"Email verified"}
}
```

Now import this server action in the new-verification-form.tsx

```
"use client"

import { useCallback, useEffect, useState } from "react"
import { BeatLoader} from "react-spinners"
import { CardWrapper } from "./card-wrapper"
import { useSearchParams } from "next/navigation"
import { newVerification } from "@/actions/new-verification"
import { FormError } from "../form-errors"
import { FormSuccess } from "../form-success"

export const NewVerificationForm = () =>{
    const [error,setError] = useState<string | undefined>();
    const [success,setSuccess] = useState<string | undefined>();

    const searchParams = useSearchParams();
    const token = searchParams.get("token");
    const onSubmit = useCallback(() =>{
        if(!token){
            setError('Missing token!');
            return;
        }
```

```
        newVerification(token)
            .then((data)=>{
                setSuccess(data.success);
                setError(data.error);
            })
            .catch(()=>{
                setError("Something went wrong");
            })
        // in Browser we can see the token being the logging
    },[token]);

    useEffect(()=>{
        onSubmit();
    },[onSubmit])

    return(
        <CardWrapper
            headerLabel="Confirming Your Verification"
            backButtonLabel="Back to login"
            backButtonHref="/auth/login"
        >
            <div className="flex items-center w-full justify-center">
                <BeatLoader/>
                <FormSuccess message={success} />
                <FormError message={error} />
            </div>

        </CardWrapper>
    )
}
```

We are getting missing token since we removed the token from the query parameter



Now if we again create a new token then it will be first verifies and when we delete it then it show Token does not exist since we have deleted it from the website

And now if we login we will be allowed to login

Repository till this point

https://github.com/nthapa000/authentication_next/tree/99afe3d6a959acc64e852efd96b2a27a68139fd0

Reset password email

Forgot password functionality

Login-form.tsx

```
<FormField
                    control={form.control}
```

```
                    name="password"
                    render={({field})=>(
                        <FormItem>
                            <FormLabel>Password</FormLabel>
                            <FormControl>
                                <Input
                                {...field}
                                placeholder="*********"
                                type="password"
                                disabled={isPending}
                                />
                            </FormControl>

                           <Button
                            size="sm"
                            variant="link"
                            asChild
                            className="px-0 font-normal"
                            >
                              <Link href="/auth/reset">
                                Forgot Password
                              </Link>
                           </Button>
                            <FormMessage />
                        </FormItem>
                    )}
                />
```

Currently on pressing the Forgot password we are being redirected auth/login, add this route to Auth routes in route.ts

```
export const authRoutes = [
   "/auth/login",
   "/auth/register",
   "/auth/error",
   "/auth/reset"
];
```

Lets create this page now app/auth/reset

```
const ResetPage = () => {
 return (
   <ResetForm />
 )
}

export default ResetPage
```

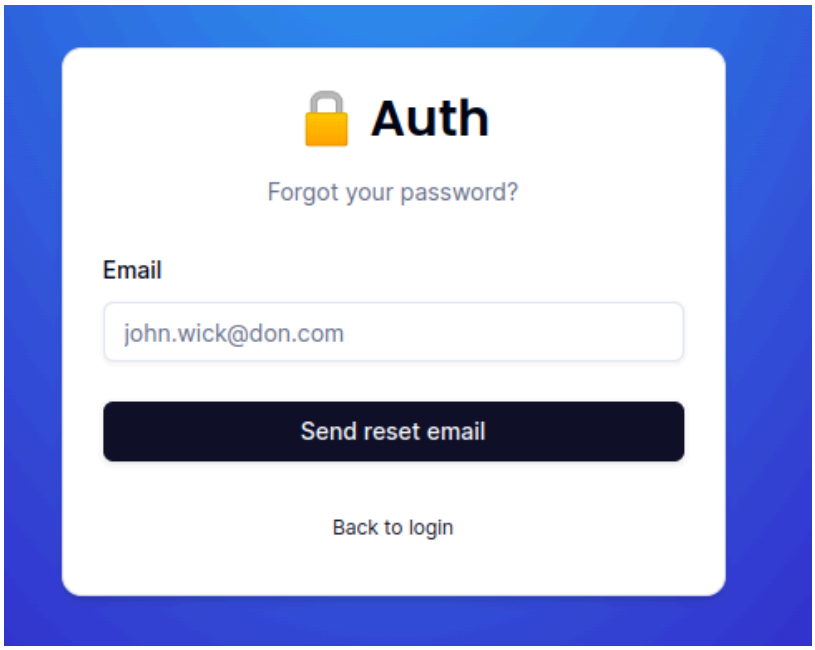
auth/component/reset-form.tsx

```
// we are not exporting as default since it is NOT a page and it is just a
component
"use client";
// since we are using hook
import * as z from "zod";

import { useForm } from "react-hook-form";
import { zodResolver } from "@hookform/resolvers/zod";
import { useState, useTransition } from "react";

import {
 Form,
 FormControl,
 FormField,
 FormItem,
 FormLabel,
 FormMessage,
} from "@/components/ui/form";

import { LoginSchema } from "@/schemas";
import { CardWrapper } from "./card-wrapper";
import { Input } from "../ui/input";
import { Button } from "../ui/button";
import { FormError } from "../form-errors";
import { FormSuccess } from "../form-success";
import { login } from "@/actions/login";
import Link from "next/link";
```

```tsx
export const ResetForm = () => {
 const [error,setError] = useState<string | undefined>("");
 const [success,setSuccess] = useState<string | undefined>("");
 const [isPending,startTransition]=useTransition();

 const form = useForm<z.infer<typeof LoginSchema>>({
   resolver: zodResolver(LoginSchema),
   defaultValues: {
     email: "",
     password: "",
   },
 });

 const onSubmit = (values:z.infer<typeof LoginSchema>)=>{
   // Everytime we hit the new submit then we will clear all successs and
error message
   setError("");
   setSuccess("");

   startTransition(()=>{
     login(values)
       .then((data)=>{
         setError(data?.error);
         // Add when we add 2FA
         // 2 factor code has been sent
         setSuccess(data?.success);
       })
   })
   // if we didn't want to do the server action then we could do in such a
way
   // axios.post("/your/api/route",values)
 }

 return (
   <CardWrapper
     headerLabel="Forgot your password?"
     backButtonLabel="Back to login"
     backButtonHref="/auth/login"
   >
     <Form {...form}>
```

```
        <form
            onSubmit={form.handleSubmit(onSubmit)}
            className="space-y-6"
        >
            <div className="space-y-4">
                <FormField
                    control={form.control}
                    name="email"
                    render={({field})=>(
                        <FormItem>
                            <FormLabel>Email</FormLabel>
                            <FormControl>
                                <Input
                                {...field}
                                disabled={isPending}
                                placeholder="john.wick@don.com"
                                type="email"
                                />
                            </FormControl>
                            {/* Message of the form , we can also change
this message by going in the Login Schema*/
                            // email:z.string().email({
                            //  message:"Email is Required"
                            // })
                           }
                            <FormMessage />
                        </FormItem>
                    )}
                />
            </div>
            {/* only one of the message will be visible */}
            {/* <FormError message="Something went Wrong"/> */}
            {/* <FormSuccess message="Email Sent"/> */}
            <FormError message={error }/>
            <FormSuccess message={success}/>
            {/* Button component */}
            <Button
                type="submit"
                className="w-full"
                disabled={isPending}
```

```
            >
              Send reset email
            </Button>
        </form>
      </Form>
    </CardWrapper>
  );
};
```



Now we will define schema

```
export const ResetSchema = z.object({
  email: z.string().email({
    message: "Email is required",
  }),
});
```

Now change accordingly in the reset-form.tsx

Server action reset.ts

```
"use server"
```

```
import { ResetSchema } from "@/schemas"
import { getUserByEmail } from "@/data/user"
// To do the server side validation
import * as z from 'zod';

export const reset = async (values: z.infer<typeof ResetSchema>) =>{
   const validatedFields = ResetSchema.safeParse(values);

   if(!validatedFields.success){
       return {error:"Invalid email"};
   }
   const {email} = validatedFields.data;
   const existingUser = await getUserByEmail(email);

   if(!existingUser){
       return { error: "Email not found!"};
   }

   // TODO generate token and send email

   return {success:"Reset email sent!"};
}
```

Reset-form.tsx

```
// we are not exporting as default since it is NOT a page and it is just a
component
"use client";
// since we are using hook
import * as z from "zod";

import { useForm } from "react-hook-form";
import { zodResolver } from "@hookform/resolvers/zod";
import { useState, useTransition } from "react";

import {
 Form,
 FormControl,
```

```
 FormField,
 FormItem,
 FormLabel,
 FormMessage,
} from "@/components/ui/form";

import { ResetSchema } from "@/schemas";
import { CardWrapper } from "./card-wrapper";
import { Input } from "../ui/input";
import { Button } from "../ui/button";
import { FormError } from "../form-errors";
import { FormSuccess } from "../form-success";
import { reset } from "@/actions/reset";

export const ResetForm = () => {
 const [error,setError] = useState<string | undefined>("");
 const [success,setSuccess] = useState<string | undefined>("");
 const [isPending,startTransition]=useTransition();

 const form = useForm<z.infer<typeof ResetSchema>>({
   resolver: zodResolver(ResetSchema),
   defaultValues: {
     email: "",
   },
 });

 const onSubmit = (values:z.infer<typeof ResetSchema>)=>{
   // Everytime we hit the new submit then we will clear all successs and
error message
   setError("");
   setSuccess("");

   console.log(values);
   startTransition(()=>{
     reset(values)
       .then((data)=>{
         setError(data?.error);
         // Add when we add 2FA
         // 2 factor code has been sent
         setSuccess(data?.success);
```

```
      })
   })
   // if we didn't want to do the server action then we could do in such a
way
   // axios.post("/your/api/route",values)
 }

 return (
   <CardWrapper
     headerLabel="Forgot your password?"
     backButtonLabel="Back to login"
     backButtonHref="/auth/login"
   >
     <Form {...form}>
       <form
           onSubmit={form.handleSubmit(onSubmit)}
           className="space-y-6"
       >
           <div className="space-y-4">
               <FormField
                   control={form.control}
                   name="email"
                   render={({field})=>(
                       <FormItem>
                           <FormLabel>Email</FormLabel>
                           <FormControl>
                               <Input
                               {...field}
                               disabled={isPending}
                               placeholder="john.wick@don.com"
                               type="email"
                               />
                           </FormControl>
                           {/* Message of the form , we can also change
this message by going in the Login Schema*/
                           // email:z.string().email({
                           //  message:"Email is Required"
                           // })
                       }
                           <FormMessage />
```

```
                    </FormItem>
                )}
            />
        </div>
        {/* only one of the message will be visible */}
        {/* <FormError message="Something went Wrong"/> */}
        {/* <FormSuccess message="Email Sent"/> */}
        <FormError message={error }/>
        <FormSuccess message={success}/>
        {/* Button component */}
        <Button
            type="submit"
            className="w-full"
            disabled={isPending}
        >
          Send reset email
        </Button>
      </form>
    </Form>
  </CardWrapper>
 );
};
```

Now we need to create Schema in prisma for the

```
model PasswordResetToken{
 id String @id @default(cuid())
 email String
 token String @unique
 expires DateTime

 @@unique([email,token])
}
```

npx prisma generate

npx prisma db push

Now inside data folder create password-reset-token.ts

```
import { db } from "@/lib/db";

export const getPasswordResetTokenByToken = async(token:string)=>{
   try{
       const passwordResetToken = await db.passwordResetToken.findUnique({
           where:{token}
       });
       return passwordResetToken;
   }catch{
       return null;
   }
};

export const getPasswordResetTokenByEmail = async(email:string)=>{
   try{
       const passwordResetToken = await db.passwordResetToken.findFirst({
           where:{email}
       });
       return passwordResetToken;
   }catch{
       return null;
   }
};
```

lib/token.ts : Checks whether there exist the token or not

```
import { getVerificationTokenByEmail } from '@/data/verificationToken';
import { v4 as uuidv4 } from 'uuid';
import { db } from './db';
import { getPasswordResetTokenByEmail } from
'@/data/password-reset-token';

export const generatePasswordResetToken = async (email:string)=>{
   const token = uuidv4();
   const expires = new Date(new Date().getTime() + 3600 * 1000);

   const existingToken = await getPasswordResetTokenByEmail(email);
   if(existingToken){
```

```
        await db.passwordResetToken.delete({
            where:{id:existingToken.id}
        })
    }

    const passwordResetToken = await db.passwordResetToken.create({
        data:{
            email,
            token,
            expires
        }
    });

    return passwordResetToken;
}
```

lib/mail.ts

```
import {Resend} from "resend";

const resend = new Resend(process.env.RESEND_API_KEY);

export const sendPasswordResetEmail = async(
    email:string,
    token:string,
)=>{
    const resetLink =
`http://localhost:3000/auth/new-password?token=${token}`

    await resend.emails.send({
        from:"onboarding@resend.dev",
        to:email,
        subject:"Reset your password",
        html:`<p>Click <a href="${resetLink}">here</a> to reset
password</p>`
    })
}
```

actions/reset.ts

```
    // TODO generate token and send email
    const passwordResetToken = await generatePasswordResetToken(email);
    await sendPasswordResetEmail(
        passwordResetToken.email,
        passwordResetToken.token,
    );


    return {success:"Reset email sent!"};
```

Repository at this point:
https://github.com/nthapa000/authentication_next/tree/da02ecfb73607d0e99ee602c9f8e4ecae6b0e18f

Reset password form:

Include the route in authRoutes

auth/new-password.tsx

```
const NewPasswordPage = () =>{
    return (
        <NewPasswordPage/>
        // <div>HI</div>
    )
}

export default NewPasswordPage;
```

Make a new password schema

schemas/index.ts

```
import * as z from "zod";

export const NewPasswordSchema = z.object({
 password: z.string().min(6, {
   message: "Minimum 6 character required",
 }),
});
```

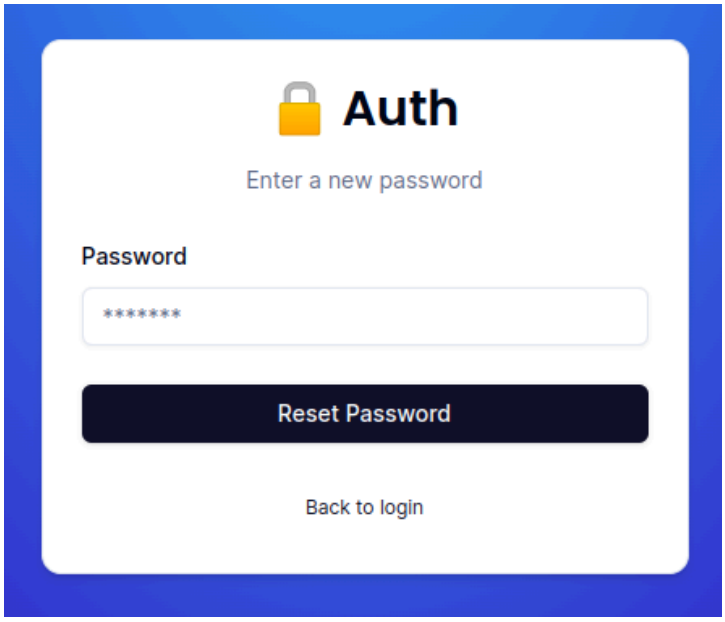
components/auth/new-password-form.tsx

```
// we are not exporting as default since it is NOT a page and it is just a
component
"use client";
// since we are using hook
import * as z from "zod";

import { useForm } from "react-hook-form";
import { zodResolver } from "@hookform/resolvers/zod";
import { useState, useTransition } from "react";

import {
 Form,
 FormControl,
 FormField,
 FormItem,
 FormLabel,
 FormMessage,
} from "@/components/ui/form";

import { NewPasswordSchema } from "@/schemas";
import { CardWrapper } from "./card-wrapper";
import { Input } from "../ui/input";
import { Button } from "../ui/button";
import { FormError } from "../form-errors";
import { FormSuccess } from "../form-success";
import { newPassword } from "@/actions/new-password";
import { useSearchParams } from "next/navigation";
```

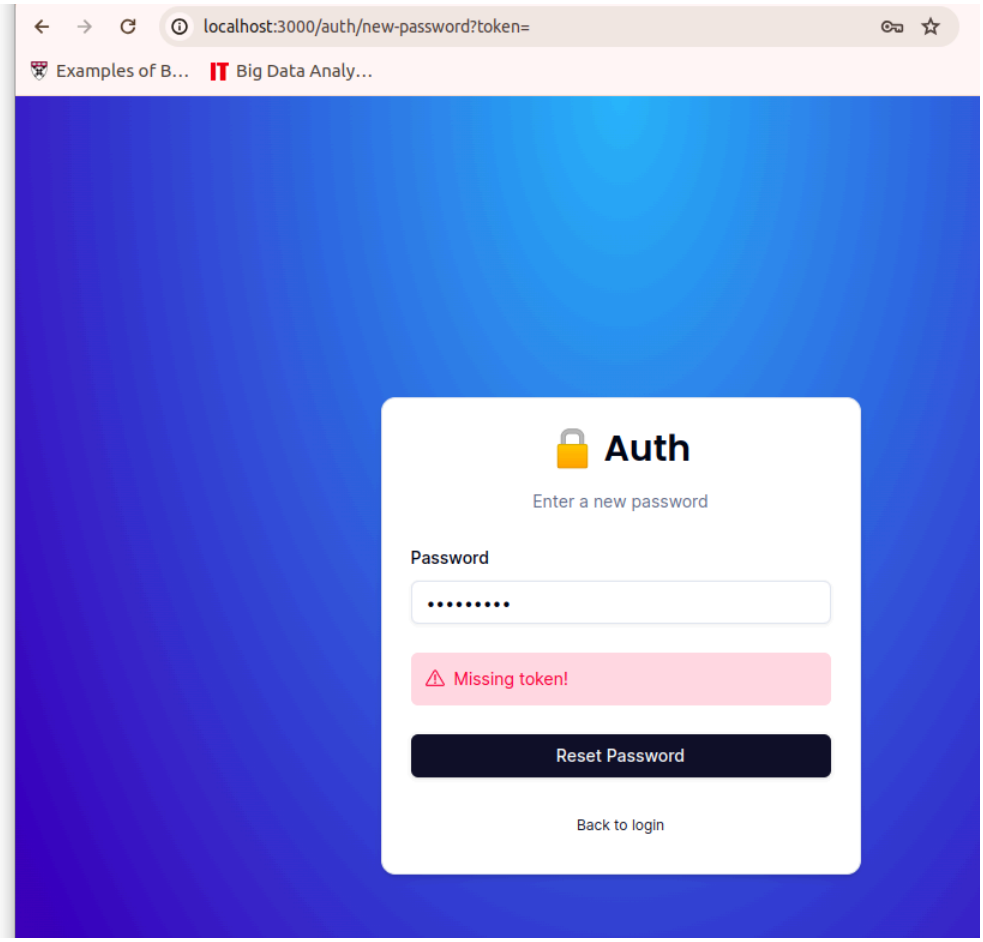
```tsx
export const NewPasswordForm = () => {
 const searchParams = useSearchParams();
 const token = searchParams.get("token")

 const [error,setError] = useState<string | undefined>("");
 const [success,setSuccess] = useState<string | undefined>("");
 const [isPending,startTransition]=useTransition();

 const form = useForm<z.infer<typeof NewPasswordSchema>>({
   resolver: zodResolver(NewPasswordSchema),
   defaultValues: {
     password: "",
   },
 });

 const onSubmit = (values:z.infer<typeof NewPasswordSchema>)=>{
   // Everytime we hit the new submit then we will clear all successs and
error message
   setError("");
   setSuccess("");

   console.log(values);
   startTransition(()=>{
     newPassword(values,token)
       .then((data)=>{
         setError(data?.error);
         // Add when we add 2FA
         // 2 factor code has been sent
         setSuccess(data?.success);
       })
   })
   // if we didn't want to do the server action then we could do in such a
way
   // axios.post("/your/api/route",values)
 }

 return (
   <CardWrapper
     headerLabel="Enter a new password"
     backButtonLabel="Back to login"
```

```
    backButtonHref="/auth/login"
  >
    <Form {...form}>
      <form
          onSubmit={form.handleSubmit(onSubmit)}
          className="space-y-6"
      >
          <div className="space-y-4">
              <FormField
                  control={form.control}
                  name="password"
                  render={({field})=>(
                      <FormItem>
                          <FormLabel>Password</FormLabel>
                          <FormControl>
                              <Input
                              {...field}
                              disabled={isPending}
                              placeholder="*******"
                              type="password"
                              />
                          </FormControl>
                          {/* Message of the form , we can also change
this message by going in the Login Schema*/
                          // email:z.string().email({
                          //  message:"Email is Required"
                          // })
                         }
                          <FormMessage />
                      </FormItem>
                  )}
              />
          </div>
          {/* only one of the message will be visible */}
          {/* <FormError message="Something went Wrong"/> */}
          {/* <FormSuccess message="Email Sent"/> */}
          <FormError message={error }/>
          <FormSuccess message={success}/>
          {/* Button component */}
          <Button
```

```
                type="submit"
                className="w-full"
                disabled={isPending}
            >
                Reset Password
            </Button>
        </form>
    </Form>
  </CardWrapper>
 );
};
```



If we try to change the password without token



actions/new-password.ts

```
"use server";
import * as z from "zod";
// Hashing the new password
import bcrypt from "bcryptjs"

import { NewPasswordSchema } from "@/schemas";
import { getPasswordResetTokenByToken } from
"@/data/password-reset-token";
import { getUserByEmail } from "@/data/user";
import { db } from "@/lib/db";

export const newPassword = async (
   values:z.infer<typeof NewPasswordSchema>,
   token?: string | null,
```

```
)=>{
    if(!token){
        return {error:"Missing token!"}
    }

    const validatedFields = NewPasswordSchema.safeParse(values);

    if(!validatedFields.success){
        return {error:"Invalid Fields!!"};
    }
    const {password} = validatedFields.data;
    const existingToken = await getPasswordResetTokenByToken(token);
    if(!existingToken){
        return {error:"Invalid token!"}
    }

    const hasExpired = new Date(existingToken.expires) < new Date();

    if(hasExpired){
        return {error:"Token has expired"};
    }

    const existingUser = await getUserByEmail(existingToken.email);

    if(!existingUser){
        return {error:"Email does not exist"};
    }

    const hashedPassword = await bcrypt.hash(password,10);

    await db.user.update({
        where:{id:existingUser.id},
        data:{password:hashedPassword}
    })

    await db.passwordResetToken.delete({
        where:{id: existingToken.id}
    });

    return {success:"Password updated"};
```

```
}
```

We successfully implemented forgot password implementation

Repository at this moment:
https://github.com/nthapa000/authentication_next/tree/cd4ca0dee5a2604663046d5985fa4e3c6fee43dc

Two Factor Authentication

schema.prisma

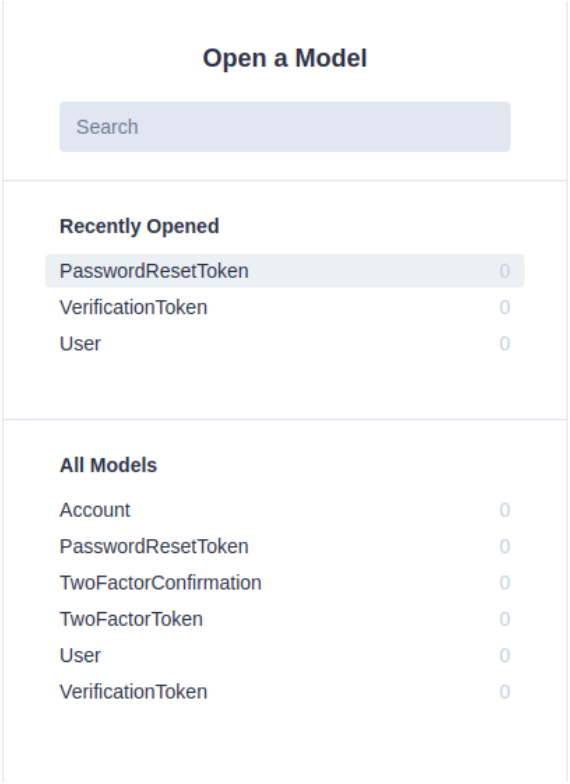
```
model User {
 id            String    @id @default(cuid())
 name          String?
 email         String?   @unique
 emailVerified DateTime?
 image         String?
 password      String?
 role          UserRole @default(USER)
 accounts      Account[]
 isTwoFactorEnabled Boolean @default(false)
 twoFactorConfirmation TwoFactorConfirmation?
}


model TwoFactorToken{
 id String @id @default(cuid())
 email String
 token String @unique
 expires DateTime

 @@unique([email,token])
}

model TwoFactorConfirmation{
 id String @id @default(cuid())
```

```
 userId String
 // if user get deleted its 2FA also get deleted
 user User @relation(fields: [userId],references: [id],onDelete: Cascade)
 @@unique([userId])
}
```

npx prisma generate

npx prisma migrate reset

npx prisma db push



Create new user with whom you have ‘resent’(https://resend.com/domains) login.

data/two-factor.ts

```
import { db } from "@/lib/db";
```

```
export const getTwoFactorTokenByToken = async (token:string) => {
    try{
        const twoFactorToken = await db.twoFactorToken.findUnique({
            where:{token}
        });
        return twoFactorToken;
    }catch{
        return null;
    }
}

export const getTwoFactorTokenByEmail = async (email:string) => {
    try{
        const twoFactorToken = await db.twoFactorToken.findFirst({
            where:{email}
        });
        return twoFactorToken;
    }catch{
        return null;
    }
}
```

Now create data for two factor confirmation

```
import { db } from "@/lib/db";

export const getTwoFactorConfirmationByUserId = async (
    userId: string
) => {
    try{
        const twoFactorConfirmation = await
db.twoFactorConfirmation.findUnique({
            where: {userId}
        })

        return twoFactorConfirmation;
    }catch{
        return null;
    }
}
```

Now we need to create lib to generate two factor authentication , we will be using crypto

```
import { getVerificationTokenByEmail } from '@/data/verificationToken';
import { v4 as uuidv4 } from 'uuid';
import { db } from './db';
import { getPasswordResetTokenByEmail } from
'@/data/password-reset-token';
import crypto from "crypto";
import { getTwoFactorTokenByEmail } from '@/data/two-factor';

export const generateTwoFactorToken = async (email:string)=>{
   const token = crypto.randomInt(100000,1000000).toString();
   // We can change this time to lesser
   const expires = new Date(new Date().getTime() + 3600*1000);

   const existingToken = await getTwoFactorTokenByEmail(email);

   if(existingToken){
       await db.twoFactorToken.delete({
           where:{
               id:existingToken.id,
           }
       })
   }

   const twoFactorToken = await db.twoFactorToken.create({
       data:{
           email,
           token,
           expires,
       }
   })

   return twoFactorToken;
}
```

Now lets create a mail util for it

lib/mail.ts

```
import {Resend} from "resend";

const resend = new Resend(process.env.RESEND_API_KEY);

export const sendTwoFactorTokenEmail = async(
   email:string,
   token:string
)=>{
   await resend.emails.send({
       from:"onboarding@resend.dev",
       to:email,
       subject:'2FA code',
       html:`<p>Your 2FA code: ${token}</p>`
   })
};
```

Now we have to manually enable the two factor Authentication from the prisma studio from false to true

Now since we have enabled the Two factor authentication we should not be able to login such easily

Lets go to **auth.ts** and add a 2FA check

```
async signIn({user,account}){
       // Allow OAuth without email verifications
       console.log({
         user,
         account
       })

       if(account?.provider !== "credentials") return true;
       // @ts-ignore
       const existingUser = await getUserById(user.id);

       // Prevent sign in without email verification
       if(!existingUser?.emailVerified) return false;
```

```
      // TODO add 2FA check
    //  This is not usefull since how will be know that the user has
verified
      // if(existingUser.isTwoFactorEnabled) {
      //   return false;
      // }
      if(existingUser.isTwoFactorEnabled){
        const twoFactorConfirmation =
getTwoFactorConfirmationByUserId(existingUser.id);

        // console.log({
        //   twoFactorConfirmation
        // })

        if(!twoFactorConfirmation) return false;

        // Delete two factor confirmation for next sign in
        //  so that next time they have to login they have to go through
that again, we can also add expires
        await db.twoFactorConfirmation.delete({
          // @ts-ignore
          where:{id:twoFactorConfirmation.id}
        });
      }

      return true;
      // allowing to sign in
    },
```

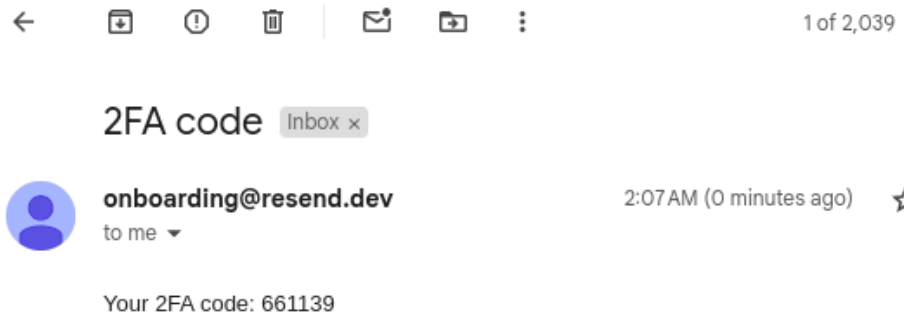
Currently on the email where the 2FA is enabled when we are trying to login we are getting something went wrong error we need to correct it.

action/login.ts

```
if(existingUser.isTwoFactorEnabled && existingUser.email){
```

```
  const twoFactorToken = await generateTwoFactorToken(existingUser.email)
  await sendTwoFactorTokenEmail(
    twoFactorToken.email,
    twoFactorToken.token,
  );

  return {twoFactor: true};
}
```

Now if we login with 2FA enabled it wont show error message instead it will send the 2FA code in the email



Now when the frontend receives twoFactor : true

We need to modify the input so that the we can enter the 2FA code

schemas/index.ts

Login schema

```
export const LoginSchema = z.object({
 email: z.string().email({
   message: "Email is required",
 }),
 password: z.string().min(1, {
   message: "Password is required",
```

```
   // Here default message is String must contain at least 1 character
this should not be the message since the password is already decided by
the users.
 }),
 code: z.optional(z.string())
 // for registering new users we will also add the minimum length of 6 but
we should keep it in login since it is possible before we keep min(6)
there would have been users who created the password of length less than 6
 //
});
```

Login-form.tsx

Create a new state variable

```
// we are not exporting as default since it is NOT a page and it is just a
component
"use client";
// since we are using hook
import * as z from "zod";

import { useForm } from "react-hook-form";
import { zodResolver } from "@hookform/resolvers/zod";
import { useState, useTransition } from "react";
import { useSearchParams } from "next/navigation";

import {
 Form,
 FormControl,
 FormField,
 FormItem,
 FormLabel,
 FormMessage,
} from "@/components/ui/form";

import { LoginSchema } from "@/schemas";
import { CardWrapper } from "./card-wrapper";
import { Input } from "../ui/input";
import { Button } from "../ui/button";
import { FormError } from "../form-errors";
```

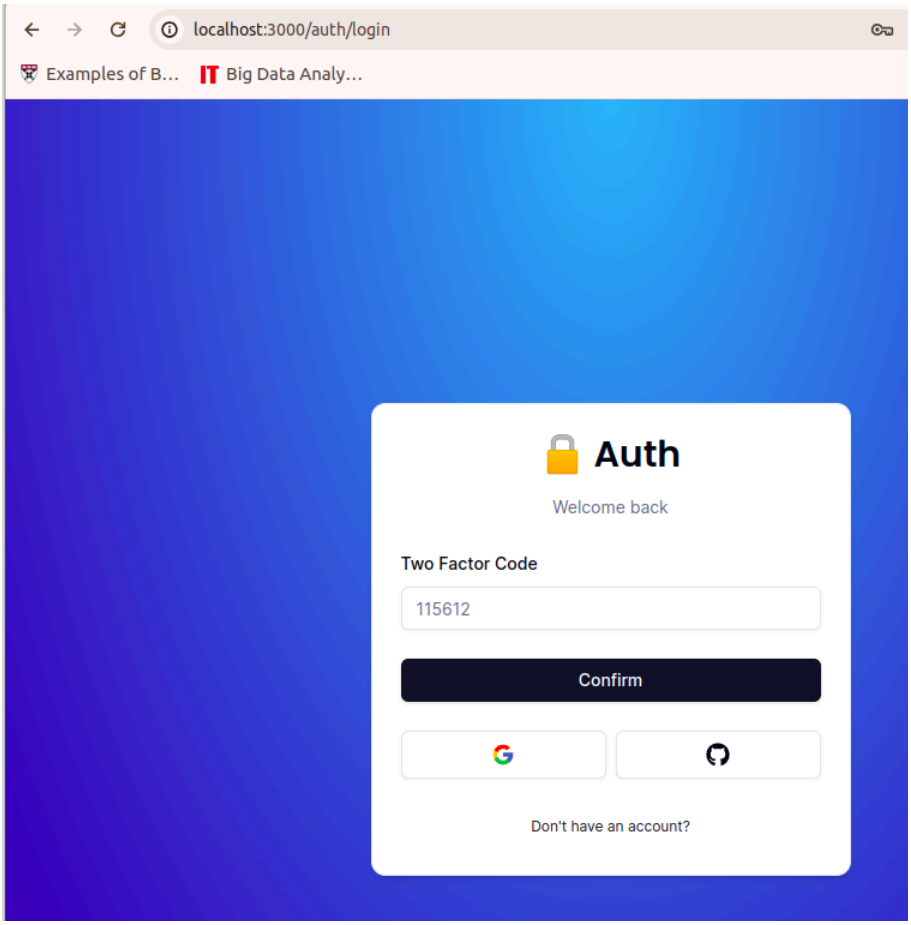
```tsx
import { FormSuccess } from "../form-success";
import { login } from "@/actions/login";
import Link from "next/link";

export const LoginForm = () => {
 const searchParams = useSearchParams();
 const urlError =
   searchParams.get("error") === "OAuthAccountNotLinked"
     ? "Email already in use with the different provider "
     : "";

 const [showTwoFactor, setShowTwoFactor] = useState(false);
 const [error, setError] = useState<string | undefined>("");
 const [success, setSuccess] = useState<string | undefined>("");
 const [isPending, startTransition] = useTransition();

 const form = useForm<z.infer<typeof LoginSchema>>({
   resolver: zodResolver(LoginSchema),
   defaultValues: {
     email: "",
     password: "",
   },
 });

 const onSubmit = (values: z.infer<typeof LoginSchema>) => {
   // Everytime we hit the new submit then we will clear all successs and
error message
   setError("");
   setSuccess("");

   startTransition(() => {
     login(values)
       .then((data) => {
         if (data?.error) {
           form.reset();
           setError(data.error);
         }
         if (data?.success) {
           form.reset();
           setSuccess(data.success);
```

```
          }

          if (data?.twoFactor) {
            // we wont reset the form
            setShowTwoFactor(true);
          }
        })
        .catch(() => {
          setError("Something went wrong!");
        });
    });
    // if we didn't want to do the server action then we could do in such a
way
    // axios.post("/your/api/route",values)
  };

  return (
    <CardWrapper
      headerLabel="Welcome back"
      backButtonLabel="Don't have an account?"
      backButtonHref="/auth/register"
      showSocial
    >
      <Form {...form}>
        <form onSubmit={form.handleSubmit(onSubmit)} className="space-y-6">
          <div className="space-y-4">
            {showTwoFactor && (
              <FormField
              control={form.control}
              name="code"
              render={({ field }) => (
                <FormItem>
                  <FormLabel>Two Factor Code</FormLabel>
                  <FormControl>
                    <Input
                      {...field}
                      disabled={isPending}
                      placeholder="115612"
                    />
                  </FormControl>
```

```
                {
                  /* Message of the form , we can also change this
message by going in the Login Schema*/
                  // email:z.string().email({
                  //  message:"Email is Required"
                  // })
                }
                <FormMessage />
              </FormItem>
            )}
          />
          )}
          {!showTwoFactor && (
            <>
              <FormField
                control={form.control}
                name="email"
                render={({ field }) => (
                  <FormItem>
                    <FormLabel>Email</FormLabel>
                    <FormControl>
                      <Input
                        {...field}
                        disabled={isPending}
                        placeholder="john.wick@don.com"
                        type="email"
                      />
                    </FormControl>
                    {
                      /* Message of the form , we can also change this
message by going in the Login Schema*/
                      // email:z.string().email({
                      //  message:"Email is Required"
                      // })
                    }
                    <FormMessage />
                  </FormItem>
                )}
              />
              <FormField
```

```
                control={form.control}
                name="password"
                render={({ field }) => (
                  <FormItem>
                    <FormLabel>Password</FormLabel>
                    <FormControl>
                      <Input
                        {...field}
                        placeholder="*********"
                        type="password"
                        disabled={isPending}
                      />
                    </FormControl>

                    <Button
                      size="sm"
                      variant="link"
                      asChild
                      className="px-0 font-normal"
                    >
                      <Link href="/auth/reset">Forgot Password</Link>
                    </Button>
                    <FormMessage />
                  </FormItem>
                )}
              />
            </>
          )}
        </div>
        {/* only one of the message will be visible */}
        {/* <FormError message="Something went Wrong"/> */}
        {/* <FormSuccess message="Email Sent"/> */}
        <FormError message={error || urlError} />
        <FormSuccess message={success} />
        {/* Button component */}
        <Button type="submit" className="w-full" disabled={isPending}>
          {showTwoFactor ? "Confirm":"Login"}
        </Button>
      </form>
    </Form>
```

```
    </CardWrapper>
  );
};
```



actions/login.ts

Repository this moment
https://github.com/nthapa000/authentication_next/tree/bd75e1e94ba4c4c86e134f03b167e24a48fadd32

Making setting page a client component

```
"use client";

import { logout } from "@/actions/logout";
import { useSession } from "next-auth/react";

const SettingsPage = () => {
 // it need to be wrapepd in session provide
 const session = useSession();

 const onClick = () =>{
   logout()
 }

 return (
   <div>
     {/* to access user we need to do this everytime hence we will create
a reusable hooks */}
       {JSON.stringify(session.data?.user)}
         <button onClick={onClick} type="submit">
           Sign out
         </button>
   </div>
 )
}

export default SettingsPage
```

actions/logout.ts

```
"use server";

import { signOut } from "@/auth";

export const logout = async () =>{
```

```
   // If we want to do some server stuff before logout , Removing user ,
clearing user etc
   await signOut();
}
```

Create new folder hooks

Use-current-user

```
import { useSession } from "next-auth/react";

export const useCurrentUser = () =>{
   const session = useSession();
   return session.data?.user;
}
```

protected/Layout

```
interface ProtectedLayoutProps{
   children:React.ReactNode;
}

const ProtectedLayout = ({children}:ProtectedLayoutProps) => {
 return (
   <div className="h-full w-full flex flex-col gap-y-10 items-center
justify-center
bg-[radial-gradient(ellipse_at_top,_var(--tw-gradient-stops))]
   from-sky-400 to-blue-800">
       <Navbar />
       {children}
   </div>
 )
}

export default ProtectedLayout
```

app/(protected)/_components/navbar.tsx

```
"use client";
```

```jsx
import { Button } from "@/components/ui/button";
import Link from "next/link";
import { usePathname } from "next/navigation";

export const Navbar = () => {
 // Now we will always know which path we are in
 const pathname = usePathname();

 return (
   <nav className="bg-secondary flex justify-between items-center p-4
rounded-xl w-[600px] shadow-sm">
     <div className="flex gap-x-2">
       <Button
         asChild
         variant={pathname === "/server" ? "default" : "outline"}
       >
         <Link href="/server">
           Server
           {/* Clicking on it we get 404 */}
         </Link>
       </Button>
       <Button
         asChild
         variant={pathname === "/client" ? "default" : "outline"}
       >
         <Link href="/client">
           Client
           {/* Clicking on it we get 404 */}
         </Link>
       </Button>
       <Button
         asChild
         variant={pathname === "/admin" ? "default" : "outline"}
       >
         <Link href="/admin">
           Admin
           {/* Clicking on it we get 404 */}
         </Link>
       </Button>
```

```
      {/* Now we want to dynamically change the button type to indicate
that we are in this page or NOT */}
      <Button
        asChild
        variant={pathname === "/settings" ? "default" : "outline"}
      >
        <Link href="/settings">Settings</Link>
      </Button>
    </div>
    <p>User button</p>
  </nav>
 );
};
```

Now lets create a reusable Logout button

```
"use client"

import { logout } from "@/actions/logout";

interface LogoutButtonProps {
  children?:React.ReactNode;
}

export const LoginButton= ({
  children
}:LogoutButtonProps)=>{
  const onClick = () => {
      logout();
  }
  return (
      <span onClick={onClick} className="cursor-pointer">
          {children}
      </span>
  )
}
```

npx shadcn-ui@latest add dropdown-menu

npx shadcn-ui@latest add avatar

Now make a user-button.tsx

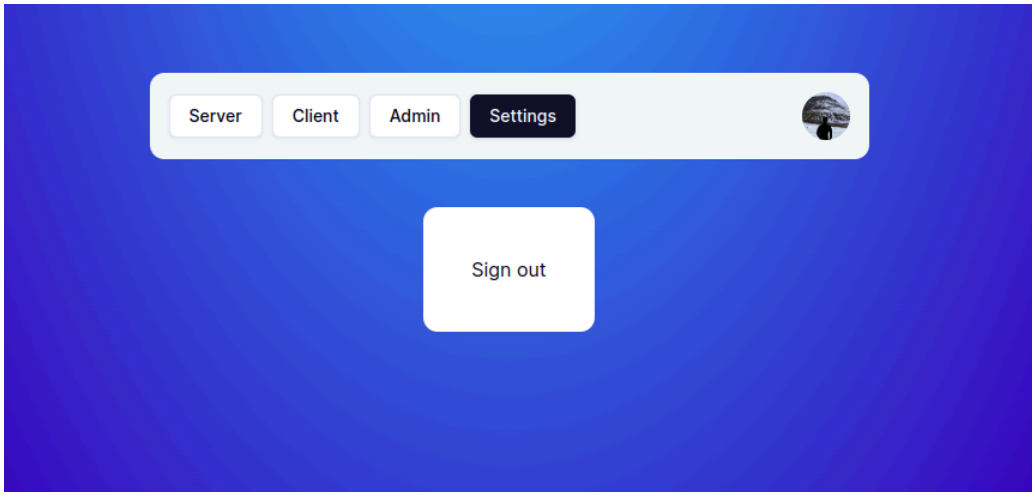
```
"use client"

import {
   DropdownMenu,
   DropdownMenuContent,
   DropdownMenuItem,
   DropdownMenuTrigger
} from '@/components/ui/dropdown-menu'

import {
   Avatar,
   AvatarImage,
   AvatarFallback
} from '@/components/ui/avatar'

export const UserButton = () =>{
   return(
       <div>
           UserButton
       </div>
   )
}
```

Complete

```
"use client"

import { FaUser } from 'react-icons/fa'
import { ExitIcon} from '@radix-ui/react-icons'

import {
   DropdownMenu,
```

```
    DropdownMenuContent,
    DropdownMenuItem,
    DropdownMenuTrigger
} from '@/components/ui/dropdown-menu'

import {
    Avatar,
    AvatarImage,
    AvatarFallback
} from '@/components/ui/avatar'
import { useCurrentUser } from '@/hooks/use-current-user'
import { LogoutButton } from './logout-button'

export const UserButton = () =>{
    const user = useCurrentUser();
    return(
        <DropdownMenu>
            <DropdownMenuTrigger>
                <Avatar>
                    <AvatarImage src={user?.image || ""}/>
                    <AvatarFallback className='bg-sky-500'>
                        <FaUser className='text-white'/>
                    </AvatarFallback>
                </Avatar>
            </DropdownMenuTrigger>
            <DropdownMenuContent className='w-40' align='end'>
                <LogoutButton>
                    <DropdownMenuItem>
                        <ExitIcon className='h-4 w-4 mr-2'/>
                        Logout
                    </DropdownMenuItem>
                </LogoutButton>
            </DropdownMenuContent>
        </DropdownMenu>
    )
}
```

It looks like this



Repository at this point

https://github.com/nthapa000/authentication_next/tree/ecd8d34fb9cfe0367738ca81386a5bebb8b76bdf

Server and Client example

Lets create lib for user

lib/auth.ts

```
import { auth } from "@/auth";

export const currentUser = async () =>{
   const session = await auth();

   return session?.user;
}
```

app/(protected)/server/page.tsx

```
import { UserInfo } from "@/components/user-info";
import { currentUser } from "@/lib/auth";

const ServerPage = async () => {
 const user = await currentUser();
 return (
```

```
    <UserInfo
        label="Server Component"
        user={user}
    />
 )
};

export default ServerPage;
```

npx shadcn-ui@latest add badge

components/user-info.tsx

```
// either server or client depending on the parent

import { ExtendedUser } from "@/next-auth";
import { Card, CardContent, CardHeader } from "./ui/card";
import { Badge } from "./ui/badge";

interface UserInfoProps{
    user?:ExtendedUser;
    label:string;
}

export const UserInfo = ({
    user,
    label,
}:UserInfoProps)=>{
    return (
        <Card className="w-[600px] shadow-md">
            <CardHeader>
                <p className="text-2xl font-semibold text-center">
                    {label}
                </p>
            </CardHeader>
            <CardContent className="space-y-4">
                <div className="flex flex-row items-center justify-between
rounded-lg border p-3 shadow-sm">
```

```
                    <p className="text-sm font-medium">
                        ID
                    </p>
                    <p className="truncate text-xs max-w-[180px] font-mono
p-1 bg-slate-100 rounded-md">
                        {user?.id}
                    </p>
                </div>
                <div className="flex flex-row items-center justify-between
rounded-lg border p-3 shadow-sm">
                    <p className="text-sm font-medium">
                        Name
                    </p>
                    <p className="truncate text-xs max-w-[180px] font-mono
p-1 bg-slate-100 rounded-md">
                        {user?.name}
                    </p>
                </div>
                <div className="flex flex-row items-center justify-between
rounded-lg border p-3 shadow-sm">
                    <p className="text-sm font-medium">
                        Email
                    </p>
                    <p className="truncate text-xs max-w-[180px] font-mono
p-1 bg-slate-100 rounded-md">
                        {user?.email}
                    </p>
                </div>
                <div className="flex flex-row items-center justify-between
rounded-lg border p-3 shadow-sm">
                    <p className="text-sm font-medium">
                        Role
                    </p>
                    <p className="truncate text-xs max-w-[180px] font-mono
p-1 bg-slate-100 rounded-md">
                        {user?.role}
                    </p>
                </div>
                <div className="flex flex-row items-center justify-between
rounded-lg border p-3 shadow-sm">
```

```
                    <p className="text-sm font-medium">
                        Two Factor Authentication
                    </p>
                    <Badge variant={user?.isTwoFactorEnabled ?
"success":"destructive"}>
                    {/* Since we don't have success variant in the Badge go
inside the Badge and add the success variant */}
                        {user?.isTwoFactorEnabled ? "ON" : "OFF"}
                    </Badge>
                </div>
            </CardContent>
        </Card>
    )
}
```

app/(protected)/client/page.tsx

```
"use client"

import { UserInfo } from "@/components/user-info";
import { useCurrentUser } from "@/hooks/use-current-user";

const ServerPage = () => {
 const user =  useCurrentUser();
 return (
   <UserInfo
       label="🌎 Client Component"
       user={user}
   />
 )
};

export default ServerPage;
```

Repository at this moment: