Next Auth

Database and Prisma Setup

npm i -D prisma

npm i @prisma/client

db.ts

```
import {PrismaClient} from "@prisma/client"

// We need this because in development we have , whenever we save the file
NExtJS will do the Hot reload and their will be lot of PrismaClient()
declare global{
   var prisma: PrismaClient | undefined;
}

export const db = globalThis.prisma || new PrismaClient();

if(process.env.NODE_ENV !== "production") globalThis.prisma = db;
// export const db = new PrismaClient();
```

To prevent several PrismaClient in development

npx prisma init

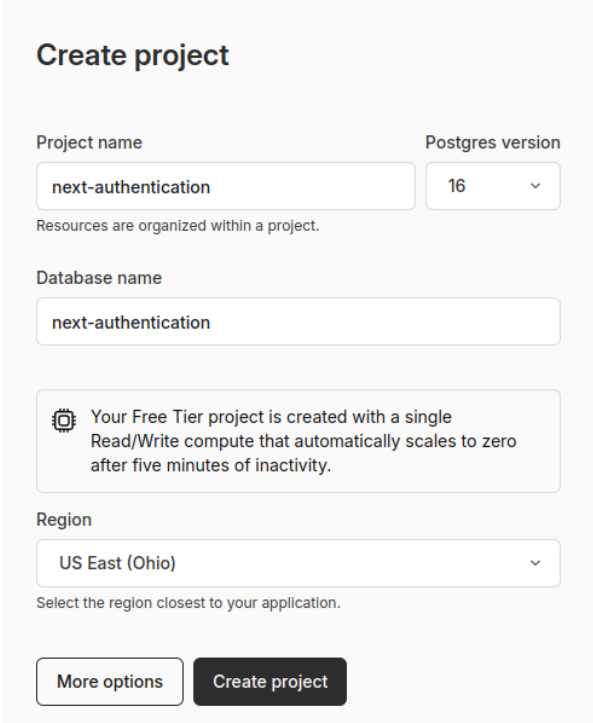
This will create a folder Prisma/schema.prisma

```
// This is your Prisma schema file,
// learn more about it in the docs: https://pris.ly/d/prisma-schema

// Looking for ways to speed up your queries, or scale easily with your
serverless or edge functions?
// Try Prisma Accelerate: https://pris.ly/cli/accelerate-init

generator client {
 provider = "prisma-client-js"
```

```
}

datasource db {
 provider = "postgresql"
 url      = env("DATABASE_URL")
}
```

And **.env** with sample database url

Now lets obtain actual database url, lets use Neon.tech to get postgres db



Connection String

postgresql://next-authentication_owner:9D3hZcMixEFw@ep-fragrant-leaf-a5lvttd1.us-east-2.aws.neon.tech/next-authentication?sslmode=require

Or we can also Choose prisma to what exactly put in Prisma and .env file

schema.prisma

```
// prisma/schema.prisma
datasource db {
 provider  = "postgresql"
 url       = env("DATABASE_URL")
 // uncomment next line if you use Prisma <5.10
 // directUrl = env("DATABASE_URL_UNPOOLED")
}
```

.env

```
DATABASE_URL="postgresql://next-authentication_owner:9D3hZcMixEFw@ep-fragr
ant-leaf-a5lvttd1-pooler.us-east-2.aws.neon.tech/next-authentication?sslmo
de=require"
# uncomment next line if you use Prisma <5.10
#
DATABASE_URL_UNPOOLED="postgresql://next-authentication_owner:9D3hZcMixEFw
@ep-fragrant-leaf-a5lvttd1.us-east-2.aws.neon.tech/next-authentication?ssl
mode=require"
```

In schema.prisma

```
model User{
 id String @id @default(cuid())
 name String
}
```

Now we need to save it

npx prisma generate

Now using lib we can access this User model

Now we will push these changes to db

npx prisma db push

```
ntc@ntc-HP-Pavilion-Laptop-15-eh1xxx:~/Desktop/Projects/authentication_ne
xt$ npx prisma db push
Environment variables loaded from .env
Prisma schema loaded from prisma/schema.prisma
Datasource "db": PostgreSQL database "next-authentication", schema "publi
c" at "ep-fragrant-leaf-a5lvttd1-pooler.us-east-2.aws.neon.tech"

🚀  Your database is now in sync with your Prisma schema. Done in 8.81s

✔ Generated Prisma Client (v5.16.1) to ./node_modules/@prisma/client in 9
6ms
```

Now we have successfully connected to the Postgres using Prisma

Go to the NextAuth documentation to see the

Add authjs prisma adapter

`npm install @prisma/client @auth/prisma-adapter`

Will be used when we setup NextAuth properly

[Auth.js | Prisma](#) For reference to prisma.schema model

Updated model on prisma.schema

```
// prisma/schema.prisma
datasource db {
 provider  = "postgresql"
 url       = env("DATABASE_URL")
 // uncomment next line if you use Prisma <5.10
 // directUrl = env("DATABASE_URL_UNPOOLED")
}

generator client{
```

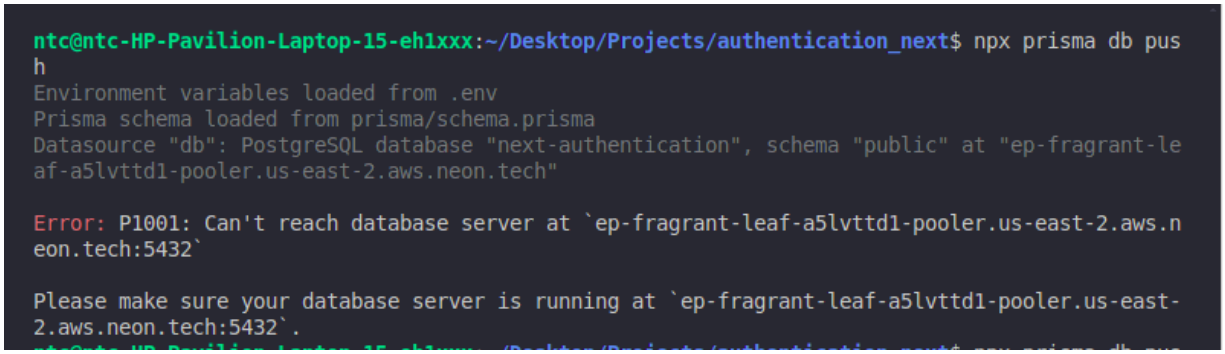
```
 provider = "prisma-client-js"
}

model User {
 id             String    @id @default(cuid())
 name           String?
 email          String?   @unique
 emailVerified DateTime?
 image          String?
 accounts       Account[]
}
model Account {
 id                 String  @id @default(cuid())
 userId             String
 type               String
 provider           String
 providerAccountId  String
 refresh_token      String? @db.Text
 access_token       String? @db.Text
 expires_at         Int?
 token_type         String?
 scope              String?
 id_token           String? @db.Text
 session_state      String?
 user User @relation(fields: [userId], references: [id], onDelete:
Cascade)
//  unque rule for the combination of the provider and the provideAccount
 @@unique([provider, providerAccountId])
}
```

Now we need to push it to the Database.

npx prisma generate

npx prisma db push

Whenever you are facing error which pushing the changes to the Database make sure to refresh/reopen the project



NextAuth by default doesn't support the credential provider , but it support it So in user model after image lets add password optional String , optional since if we use Oauth provide like google , github then there is no need for the password

We need allow our Adapter to create this User model without the need of the password.

Everytime we make some changes in the schema.prisma then we have to generate the client and then push the changes to the Database.

Repository at the moment:
https://github.com/nthapa000/authentication_next/tree/23d69d446f653472cec3908d1b225b86fb0b8e3b

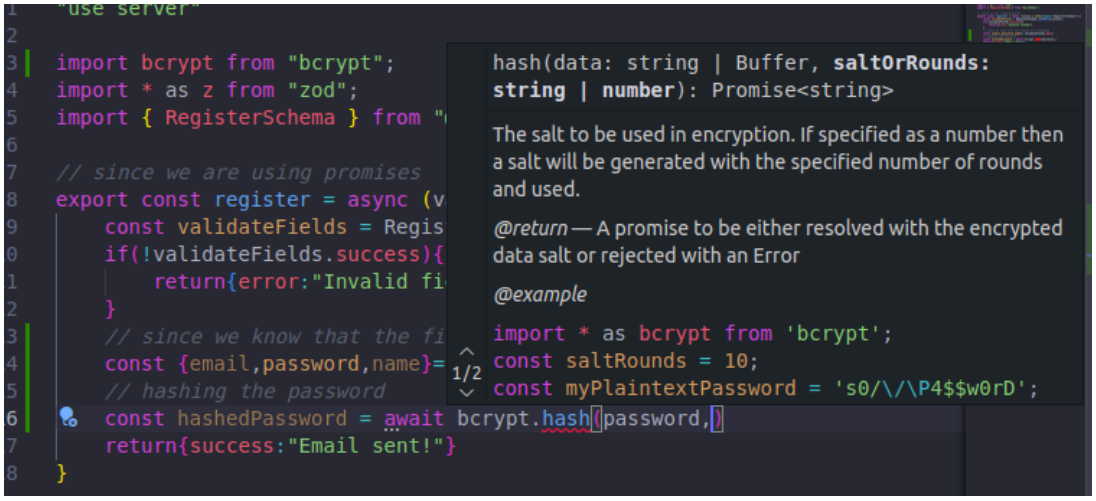
Create User

To store the user information in the database we have to encrypt the password. We will be using **Bcrypt**

npm i bcrypt

Now lets install its types since it doesn't come by default

npm i -D @types/bcrypt

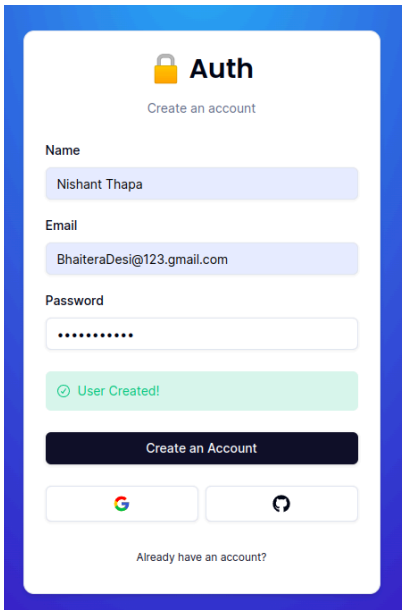


If we hover on the hash function then we can know the different way of using it

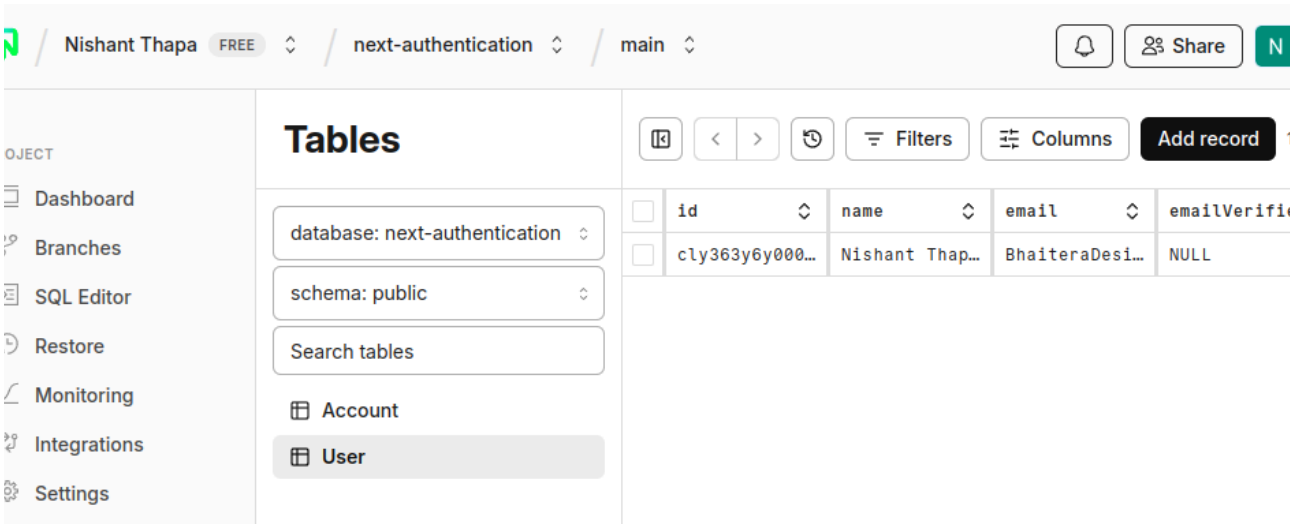
actions/register.ts

```
"use server"

import { db } from "@/lib/db";
import bcrypt from "bcrypt";
import * as z from "zod";
import { RegisterSchema } from "@/schemas";

// since we are using promises
export const register = async (values:z.infer<typeof RegisterSchema>)=>{
   const validateFields = RegisterSchema.safeParse(values);
   if(!validateFields.success){
       return{error:"Invalid fields"}
   }
   // since we know that the fields are validated
   const {email,password,name}= validateFields.data;
   // hashing the password
   const hashedPassword = await bcrypt.hash(password,10)
   // conform whether email is available on not this will be done by
checking on the database
   // finding if there exist a user with the given email address
   const existingUser = await db.user.findUnique({
       where:{
```

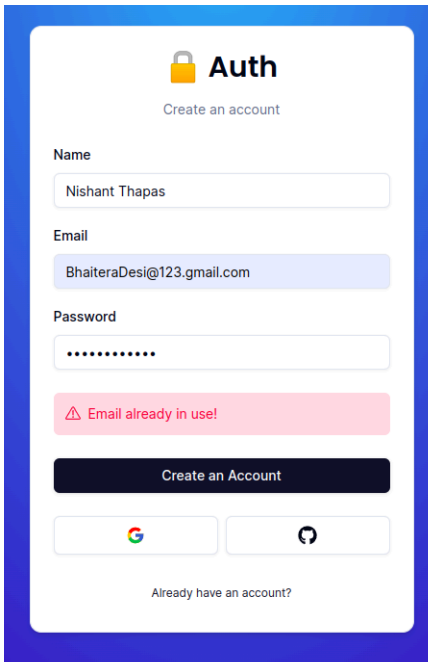
```
            email,
        }
    });

    if(existingUser){
        return {error:"Email already in use!"};
    }
    // since user doesn't exist with the given email address now we can
create a new user
    await db.user.create({
        data:{
            name,
            email,
            password:hashedPassword,
        }
    });

    // TODO: send verification email

    return{success:"User Created!"}
}
```



Now lets check whether the user is updated on the Database



And our password is hashed and now lets try with the same email



Now create a folder called data/user.ts

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

// util
export const getUserByEmail = async (email: string)=>{
   try{
       const user = await db.user.findUnique({where:{email}});
       return user;
```

```
    }catch{
        return null;
    }
}

export const getUserById = async (id: string)=>{
    try{
        const user = await db.user.findUnique({where:{id}});
        return user;
    }catch{
        return null;
    }
}
```

We will use these on the auth callback later where we need more information from the database

Now lets use this in the register function **register.ts**

```
const existingUser = await getUserByEmail(email);

    if(existingUser){
        return {error:"Email already in use!"};
    }
```

Alternatively of **bcrypt** is bcryptjs

npm i bcryptjs

npm i -D @types/bcryptjs

Just import it and the rest of the code will be same

Currently we are not implementing the verification of the email as since we first want to do login and then we will only allow users whose email are verified this done directly by nextAuth , we wont be breaking the login function if the field verification is not there , but we will be telling the nextAuth itself.

Repository at this point:
https://github.com/nthapa000/authentication_next/tree/0a97691333ae0c569af390ee8999f41ed98447c4

Middleware And Login

npm install next-auth@beta

New features

https://authjs.dev/getting-started/migrating-to-v5

And now check the Configuration also on the same page

In root directory create a new file called **auth.ts**

```
// from the Auth.js website
import NextAuth from "next-auth";
import GitHub from "next-auth/providers/github";
import Google from "next-auth/providers/google";

// we have to add GET and POST inside the api for the next auth
// auth to get the current session of the user if any
export const {
 auth,
 handlers: { GET, POST },
} = NextAuth({
 providers: [GitHub, Google],
});
```

Create a new folder inside app

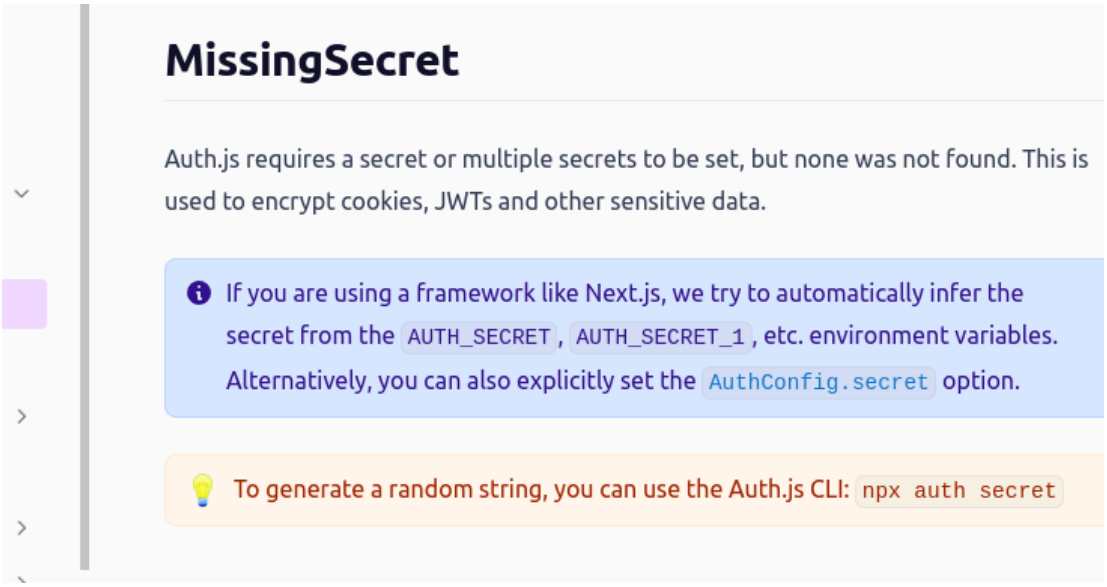
api/auth/[...nextauth]/route.ts

```
export {GET,POST} from "@/auth"
// prisma by default do not support edge
```

Now if we go to http://localhost:3000/api/auth/providers

We will get error


```
{
  "message": "There was a problem with the server configuration. Check the server logs for more information."
}
```

In .env add AUTH_SECRET

Now if we visit it again we get

Pretty print

```
{
  "github": {
    "id": "github",
    "name": "GitHub",
    "type": "oauth",
    "signinUrl": "http://localhost:3000/api/auth/signin/github",
    "callbackUrl": "http://localhost:3000/api/auth/callback/github"
  },
  "google": {
    "id": "google",
    "name": "Google",
    "type": "oidc",
    "signinUrl": "http://localhost:3000/api/auth/signin/google",
    "callbackUrl": "http://localhost:3000/api/auth/callback/google"
  }
}
```

For production it is recommended to generate AUTH_SECRET to be complex

```
npx auth secret
```

Now lets setup the Middleware

To see the difference between older version and newer version of NextAuth here is the page

[Migrate to NextAuth.js v5]

Inside the root of the project create **Middleware.ts**

Middleware is not NextAuth specific , it is nextJs specific hence don't mis spell it

```
import { auth } from "@/auth"
export default auth((req) => {
 // req.auth
})
// Optionally, don't invoke Middleware on some paths
export const config = {
 matcher: ["/((?!api|_next/static|_next/image|favicon.ico).*)"],
}
```

Config matcher is used to invoke the middleware

```
import { auth } from "@/auth"
export default auth((req) => {
 // req.auth
 console.log("ROUTE",req.nextUrl.pathname)
})
// Optionally, don't invoke Middleware on some paths
export const config = {
 matcher: ["/auth/login"],
}
```

Now middle ware is invoke in this route

We can confirm it in terminal

```
 GET /auth/login 200 in 4552ms
ROUTE /auth/login
 GET /auth/login 200 in 29ms
```

If we go to some other route we wont see it

Now replace the matcher with the given one

```
matcher: [ '/((?!.*\\..*|_next).*)', '/', '/(api|trpc)(.*)'],
```

```
// Optionally, don't invoke Middleware on some paths
export const config = {
   matcher: [ '/((?!.*\\..*|_next).*)', '/', '/(api|trpc)(.*)'],
   // Everything except the static files and the next images won't invoke the middleware
}
```

We want middleware on both public and authorized route and then we can decide what to do with it in middleware

```
export default auth((req) => {
 // req.auth
 //   console.log("ROUTE",req.nextUrl.pathname)
 //   We will use the combination of logged in state and pathname to decide what to do with the route
 //   I want my app to be protected by default
 // We will have fewer public route then private route, Currently we will put whole our app to be accessed by the authorized users only then we will allow only certain route to be accessed by public like landing page
 const isLoggedIn = !!req.auth;
 console.log("ROUTE: ", req.nextUrl.pathname);
 console.log("IS LOGGEDIN: ", isLoggedIn);
});
```

```
 ✓ Compiled in 45ms (227 modules)
ROUTE:  /
IS LOGGEDIN:  false
 GET / 200 in 45ms
ROUTE:  /auth/login
IS LOGGEDIN:  false
```

We are using Prisma as ORM since it doesn't support edge then we have to do something, we will not be able to use lot of callbacks and events inside the auth.ts

Solution for it is separating the config and then using it in the middleware (Middleware work on Edge)
https://authjs.dev/getting-started/migrating-to-v5#edge-compatibility

Make a file called **auth.config.ts** in root directory

```ts
import GitHub from "next-auth/providers/github"
import Google from "next-auth/providers/google";

import type { NextAuthConfig } from "next-auth"
export default { providers: [GitHub,Google] } satisfies NextAuthConfig
```

So Now basically we will use auth.config.ts to trigger the middleware and not the auth.ts which will use the Prisma Adapter

auth.ts

```ts
import NextAuth from "next-auth";
// Adding prisma adapter
import {PrismaAdapter} from "@auth/prisma-adapter"
import { db } from "./lib/db";
import authConfig from "./auth.config";

// we have to add GET and POST inside the api for the next auth
// auth to get the current session of the user if any
export const {
 auth,
```

```
 handlers: { GET, POST },
} = NextAuth({
   // passing the database to the prisma adapter
   // here it contains non edge supported Prisma Adapter hence we need to
make some changes in middleware.ts
   adapter:PrismaAdapter(db),
   session:{strategy:"jwt"},
   ...authConfig,
});
```

npm i @auth/prisma-adapter

middleware.ts

```
import NextAuth from "next-auth";

import authConfig from "./auth.config";

const {auth} = NextAuth(authConfig);

export default auth((req) => {
 // req.auth
 //   console.log("ROUTE",req.nextUrl.pathname)
 //   We will use the combination of logged in state and pathname to
decide what to do with the route
 //   I want my app to be protected by default
 // We will have fewer public route then private route, Currently we will
put whole our app to be accessed by the authorized users only then we will
allow only certain route to be accessed by public like landing page
 const isLoggedIn = !!req.auth;
 console.log("ROUTE: ", req.nextUrl.pathname);
 console.log("IS LOGGEDIN: ", isLoggedIn);
});

// Optionally, don't invoke Middleware on some paths
export const config = {
 matcher: ["/((?!.*\\..*|_next).*)", "/", "/(api|trpc)(.*)"],
```

```
 // Everything except the static files and the next images won't invoke
the middleware
};
```

Now lets do some callbacks(they are extremely useful when we want to trigger something specifically on some next auth action like, signin, signout,authorized,redirect)

Lets just check the protected routes

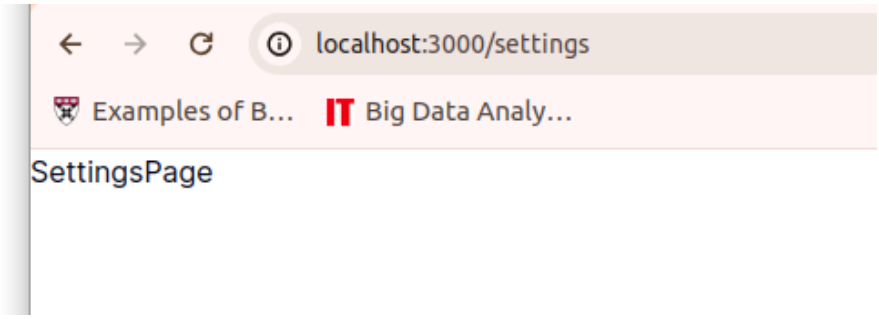
Make a folder in app folder

(protected)/settings/page.tsx

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

export default SettingsPage
```

It is in protected folder such that we can know that the following routes should be protected

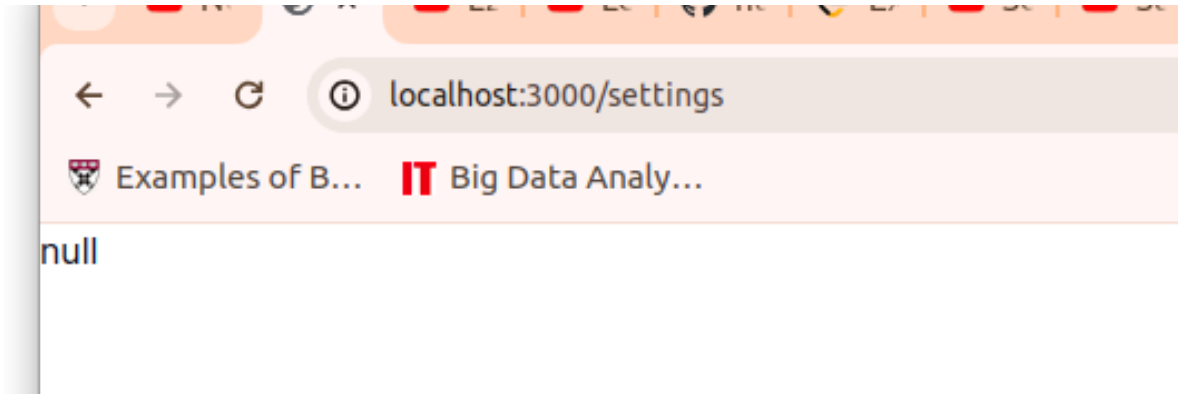
http://localhost:3000/settings




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

```
const SettingsPage = async () => {
 const session = await auth();

 return (
   <div>
       {JSON.stringify(session)}
   </div>
 )
}

export default SettingsPage
```

Now we can see that currently the current session is **null**



Now what I want to do is that if a user is logged out then we don't want them to be able to access the settings route

Now on the root directory create a folder **route.ts**

```
// routes which can be accessed by the logged out users/ public
// JS docs so that when we hover on the publicRoutes we can see these
comments

/**
* An array of routes that are accessible to the public
* These routes do not require authentication
* @type {string[]}
*/
export const publicRoutes = [
```

```
    "/"
];

/**
* An array of routes that are used for authentication
* These routes will redirect logged in users to /settings
* @type {string[]}
*/
export const authRoutes = [
    "/auth/login",
    "/auth/register",
];

// this route should be always allowed for user to login /signup

/**
* The prefix for API authentication routes
* Routes that start with this prefix are used for API authentication
process
* @type {string}
*/
export const apiAuthPrefix = "/api/auth";

// this is the place where we will redirect whenever the user is logged
in, unless specified differently
/**
* The default redirect path after logging in
* @type {string}
*/
export const DEFAULT_LOGIN_REDIRECT = "/settings"
```

Js docs are used here they are shown when we hover on it

Lets use these in the middleware.ts

```
import NextAuth from "next-auth";

import authConfig from "./auth.config";
import {
 DEFAULT_LOGIN_REDIRECT,
 apiAuthPrefix,
 authRoutes,
 publicRoutes,
} from "@/route";

const { auth } = NextAuth(authConfig);

export default auth((req) => {
 // runs in everyroute

 //   console.log("ROUTE",req.nextUrl.pathname)
 //   We will use the combination of logged in state and pathname to
decide what to do with the route
 //   I want my app to be protected by default
 // We will have fewer public route then private route, Currently we will
put whole our app to be accessed by the authorized users only then we will
allow only certain route to be accessed by public like landing page

 const { nextUrl } = req;
 const isLoggedIn = !!req.auth;

 // always allow these routes
```

```
 // example : /api/auth/providers these are required by the nextAuth to
function properly
 const isApiAuthRoute = nextUrl.pathname.startsWith(apiAuthPrefix);
 const isPublicRoute = publicRoutes.includes(nextUrl.pathname);

 // if user is already logged in then we are not gonna allow users to go
to the login section again, and if the user is logged out and try to
access the settings page then they will be redirected to the login page
 const isAuthRoute = authRoutes.includes(nextUrl.pathname);

 // allow every single Api route
 // Make sure you can visit localhost:3000/api/auth/provider
 if (isApiAuthRoute) {
   return null;
 }

 // authroute are technically public route we need to check them first
before we check the public orute
 if (isAuthRoute) {
   if (isLoggedIn) {
     return Response.redirect(new URL(DEFAULT_LOGIN_REDIRECT, nextUrl));
     // so that it create an absolute URL ,localhost:3000/settings
   }
   return null;
 }

 if (!isLoggedIn && !isPublicRoute) {
   return Response.redirect(new URL("/auth/login", nextUrl));
 }

 return null;
 // allowing everyother route
});

// Optionally, don't invoke Middleware on some paths
export const config = {
 matcher: ["/((?!.*\\..*|_next).*)", "/", "/(api|trpc)(.*)"],
 // Everything except the static files and the next images won't invoke
the middleware
};
```

We are allowed to go to route /auth/login , /auth/register and /api/auth/providers
But we are not allowed to go to /setting pages if we are not logged in

But if we add /setting in public routes then we can access it even if we are not logged in.

Repository at this point:
https://github.com/nthapa000/authentication_next/tree/43620d0b96a0eed100a630716a9b695497bf005c

Now lets do the Login
We will currently do the login with the Credentials

auth.config.ts

```
import bcrypt from "bcryptjs"
import type { NextAuthConfig } from "next-auth";
import Credentials from "next-auth/providers/credentials"

import { LoginSchema } from "./schemas";
import { getUserByEmail } from "./data/user";

// it is possible that users can surpass the server actions
// and don't use our login page at all and use api/auth directly
// Hence we have to do login check here also

export default {
   providers: [
       Credentials({
           async authorize(credentials){
               const validatedFields = LoginSchema.safeParse(credentials);
               if(validatedFields.success){
                   const {email,password} = validatedFields.data;

                   const user = await getUserByEmail(email);
```

```
                    // we wont allow the users who have login with gmail to
use the credentials provider
                    if(!user || !user.password) return null;

                    // confirming if the password is correct withour
actually knowing the correct password
                    const passwordsMatch = await bcrypt.compare(
                        password,
                        user.password,
                    );

                    if(passwordsMatch) return user;
                }
                return null;
            }
        })
    ]
} satisfies NextAuthConfig;
```

auth.ts

Export signIn and signOut this can be used in server actions

```
export const {
 handlers: { GET, POST },
 auth,
 signIn,
 signOut
} = NextAuth({
    // passing the database to the prisma adapter
    // here it contains non edge supported Prisma Adapter hence we need to
make some changes in middleware.ts
    adapter:PrismaAdapter(db),
    session:{strategy:"jwt"},
    ...authConfig,
});
```

actions/login.ts

```ts
"use server";

import * as z from "zod";
import { LoginSchema } from "@/schemas";
import { signIn } from "@/auth";
import { DEFAULT_LOGIN_REDIRECT } from "@/route";
import { AuthError } from "next-auth";

// 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;
 try {
   // give the type of signin
   await signIn("credentials", {
     email,
     password,
     // later we will have here callback when we implement it
     redirectTo: DEFAULT_LOGIN_REDIRECT,
   });
 } catch (error) {
   if (error instanceof AuthError) {
     switch (error.type) {
       case "CredentialsSignin":
         return { error: "Invalid Credentials!" };
       default:
         return { error: "Something went wrong!" };
     }
   }
   throw error;
 }
};
```

After the login we are able to login

We can’t go to login/register route

Now we need to add a logout button to clear the cache

Now inside **app/(protected)/settings** add a button

```
import { auth, signOut } from "@/auth"

const SettingsPage = async () => {
 const session = await auth();

 return (
   <div>
       {JSON.stringify(session)}
       <form action={async()=>{
         "use server";
         await signOut();
       }}>
         <button type="submit">
           Sign out
         </button>
       </form>
   </div>
 )
}

export default SettingsPage
```

Now if we click on the Sign out it will logout

In the session we can see that there is not id etc , hence we will use callbacks

Repository at this moment

https://github.com/nthapa000/authentication_next/tree/6c6abcd0b670f3317624cc4d991faf0bc4bd7c10

Callbacks

Lets learn more about callbacks

https://next-auth.js.org/configuration/callbacks

We need to extend the session and for it we need to extend jwt

Example

```
export const {
 handlers: { GET, POST },
 auth,
 signIn,
 signOut
} = NextAuth({
   callbacks:{
     async jwt({token}){
       console.log({token});
       return token;
     }
   },
   // passing the database to the prisma adapter
   // here it contains non edge supported Prisma Adapter hence we need to
make some changes in middleware.ts
   adapter:PrismaAdapter(db),
   session:{strategy:"jwt"},
   ...authConfig,
```

```
});
```

Here in terminal

```
 ✓ Compiled in 129ms (786 modules)
{
  token: {
    name: 'test',
    email: 'test@gmail.com',
    picture: null,
    sub: 'cly70v5y70000a9ifrqqbeaoc',
    iat: 1720084634,
    exp: 1722676634,
    jti: 'dbd1c1a9-4d09-4274-9280-4faf9e948f69'
  }
}
 GET /settings?_rsc=1gko6 200 in 66ms
```

sub is id

, it is not everything

Npx prisma studio to see your database

```
callbacks:{
    async session({token,session}){
      console.log({
        sessionToken: token,
        session,
      })
      session.user.customField = token.customField;
      return session;
    },
    async jwt({token}){
      // token is much more reliable to use for the logged in user
      console.log({token});
      token.customField="test";
      // it will be available in both sessionToken adn token
      return token;
    }
  },
```

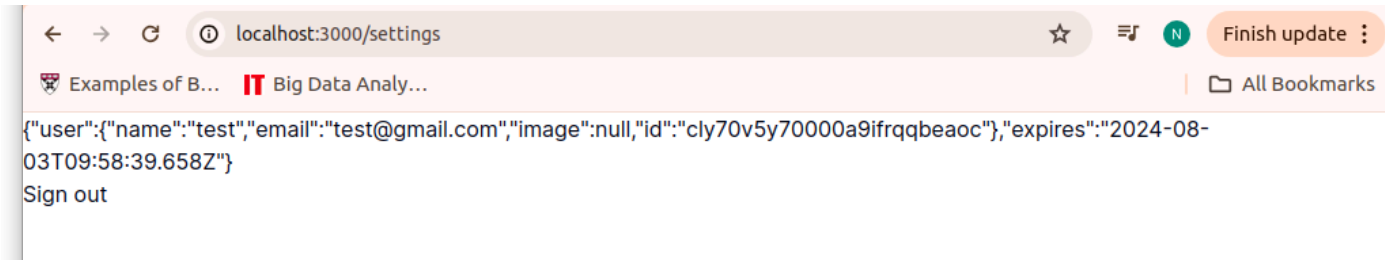
```
 }
  GET /settings?_rsc=1gko6 200 in 65ms
  ✓ Compiled in 167ms (756 modules)
 {
   token: {
     name: 'test',
     email: 'test@gmail.com',
     picture: null,
     sub: 'cly70v5y70000a9ifrqqbeaoc',
     iat: 1720086882,
     exp: 1722678882,
     jti: '524bab3c-b985-4be2-a082-b9c13cc037e1'
   }
 }
 {
   sessionToken: {
     name: 'test',
     email: 'test@gmail.com',
     picture: null,
     sub: 'cly70v5y70000a9ifrqqbeaoc',
     iat: 1720086882,
     exp: 1722678882,
     jti: '524bab3c-b985-4be2-a082-b9c13cc037e1',
     customField: 'test'
   },
   session: {
     user: { name: 'test', email: 'test@gmail.com', image: null },
     expires: '2024-08-03T09:54:45.490Z'
   }
 }
```

auth.ts

Getting the id in the settings page

```
import NextAuth from "next-auth";
// Adding prisma adapter
import {PrismaAdapter} from "@auth/prisma-adapter"
import { db } from "./lib/db";
import authConfig from "./auth.config";

// we have to add GET and POST inside the api for the next auth
// auth to get the current session of the user if any
export const {
 handlers: { GET, POST },
 auth,
 signIn,
 signOut
} = NextAuth({
```

```
 // how we can transfer id from token to the session
   callbacks:{
     async session({token,session}){
       // console.log({
       //   sessionToken: token,
       //   session,
       // })
       // session.user.customField = token.customField;
       if (token.sub && session.user){
         session.user.id = token.sub;
       }
       // Now we can access the id whetehr it is server side componetn or
client side
       return session;
     },
     async jwt({token}){
       // token is much more reliable to use for the logged in user
       // console.log({token});
       // token.customField="test";
       // it will be available in both sessionToken adn token
       return to
ken;
     }
   },
   // passing the database to the prisma adapter
   // here it contains non edge supported Prisma Adapter hence we need to
make some changes in middleware.ts
   adapter:PrismaAdapter(db),
   session:{strategy:"jwt"},
   ...authConfig,
});
```



Now lets add more field , go to the schema.primsa and add field

```
enum UserRole{
 ADMIN
 USER
}

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

Now do

npx prisma generate

npx prisma migrate reset : will reset the entire database and all data will be lost

Do it only in development

npx prisma db push

auth.ts

import getUserById form user.ts

```
export const {
 handlers: { GET, POST },
 auth,
 signIn,
 signOut
} = NextAuth({
 // how we can transfer id from token to the session
   callbacks:{
     async session({token,session}){
       console.log({
         sessionToken: token,
```

```
      //   session,
      })
      // session.user.customField = token.customField;
      if (token.sub && session.user){
        session.user.id = token.sub;
      }

       if(token.role && session.user){
         session.user.role = token.role;
      }

      // Now we can access the id whetehr it is server side componetn or
client side
      return session;
    },
    async jwt({token}){
      // token is much more reliable to use for the logged in user
      // console.log({token});
      // token.customField="test";
      // it will be available in both sessionToken adn token
      if(!token.sub)  return token;

      const existingUser = await getUserById(token.sub);
      if(!existingUser) return token;
      token.role = existingUser.role;
      return token;
    }
  },
  // passing the database to the prisma adapter
  // here it contains non edge supported Prisma Adapter hence we need to
make some changes in middleware.ts
  adapter:PrismaAdapter(db),
  session:{strategy:"jwt"},
  ...authConfig,
});
```

Now our session token will have role

```
 GET /settings?_rsc=1gko6 200 in 834ms
 ✓ Compiled in 251ms (716 modules)
{
  sessionToken: {
    name: 'test',
    email: 'test@gmail.com',
    picture: null,
    sub: 'cly7o9oh8000014i7u0hv9cd0',
    iat: 1720122747,
    exp: 1722714747,
    jti: 'b10d20ab-a19a-4baf-a50d-a8ccf67fad8c',
    role: 'USER'
  }
}
 GET /settings?_rsc=1gko6 200 in 340ms
```

Now we need to see how we can add typescript in the session

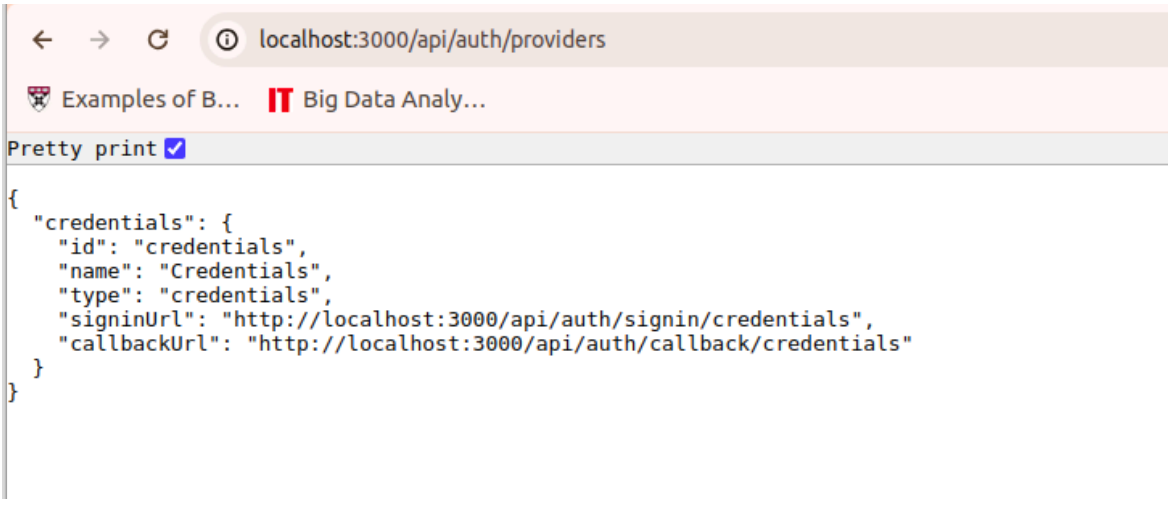
https://authjs.dev/getting-started/typescript

Repository at this point:

https://github.com/nthapa000/authentication_next/tree/0420ec1610419bb90c023181fe1013a4c9b13521

Adding Google and Github

Currently when we go to localhost:3000/api/auth/providers we can see the following

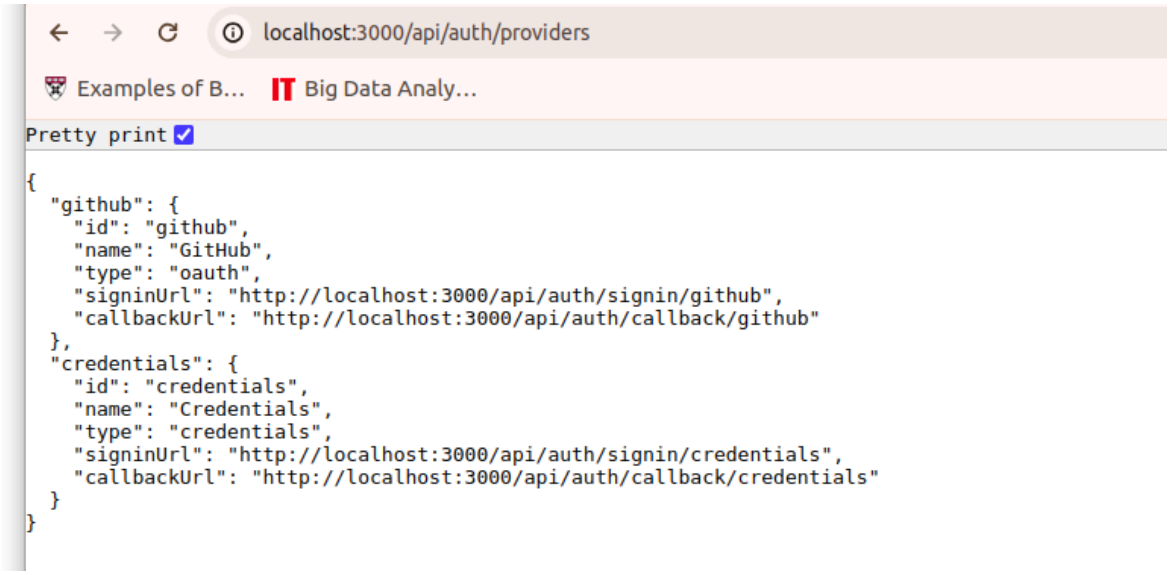


We can see the Credentials provider

auth.config.ts

Importing the Github provider from the next-auth/providers/github

We can then see the github in providers



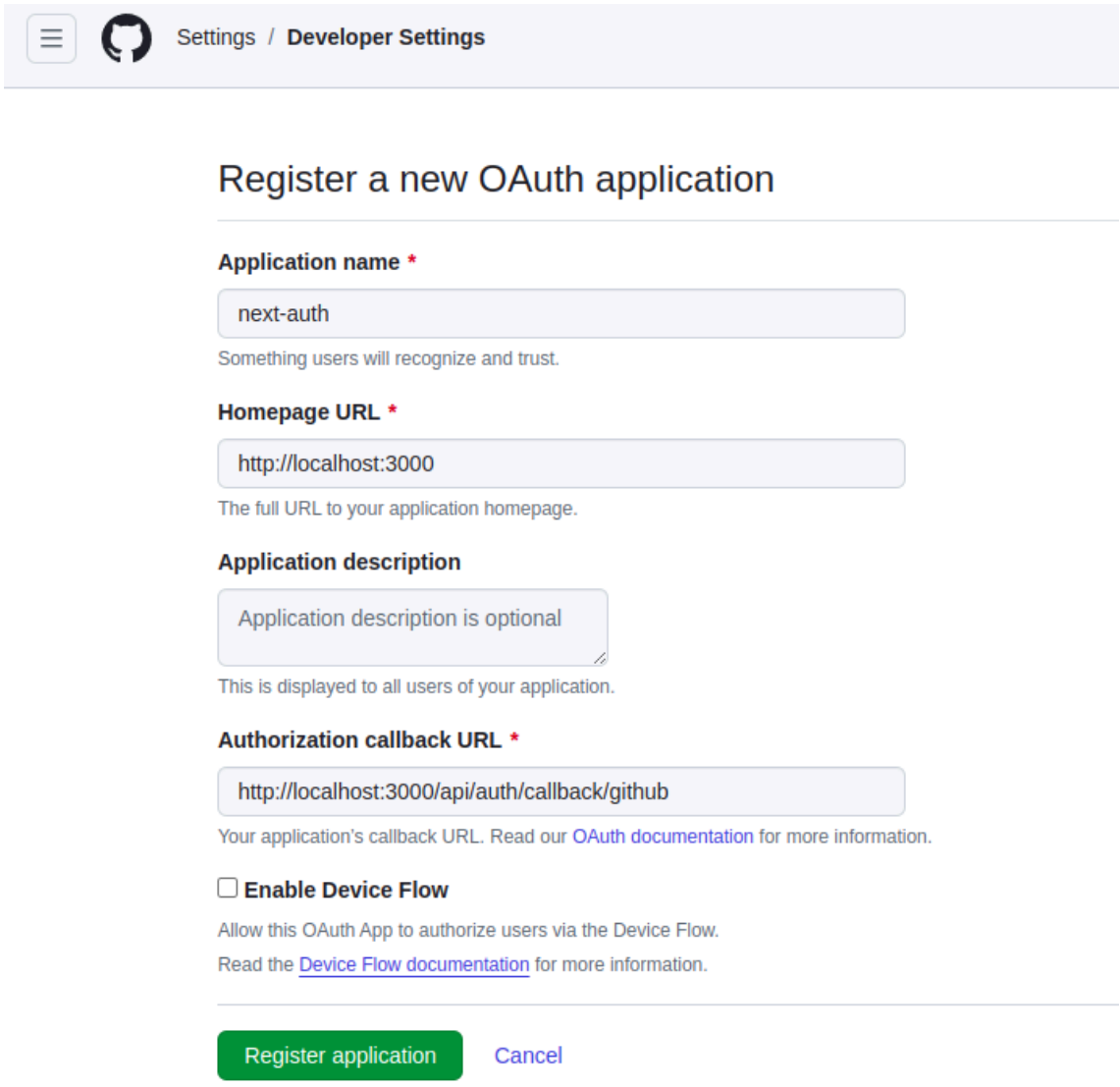
Now we need to get the environment keys

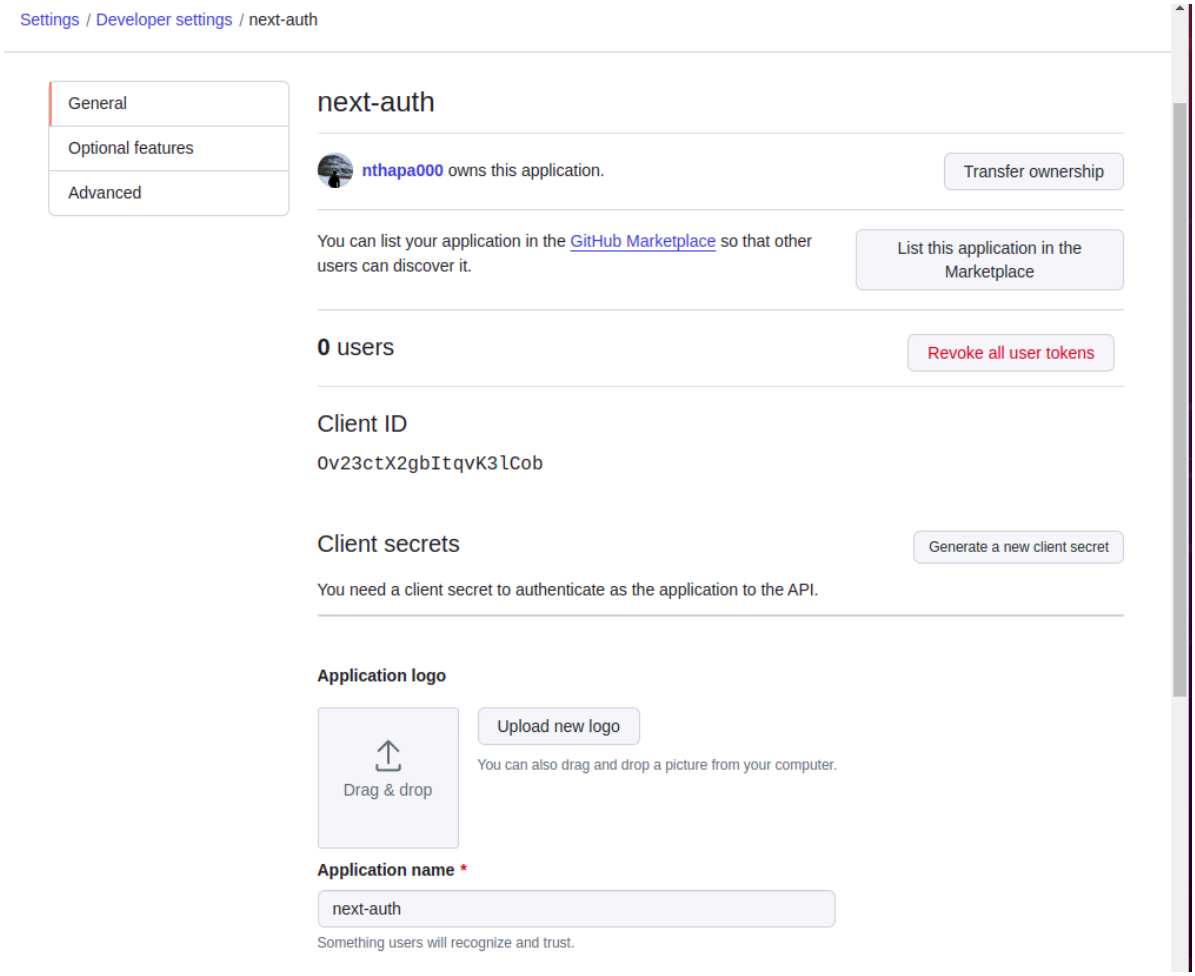
Let do for GIthub First

Go to your profile then settings and then scroll down to Developer settings and inside it go to the OAuth Apps and then click on the Create new OAuth application

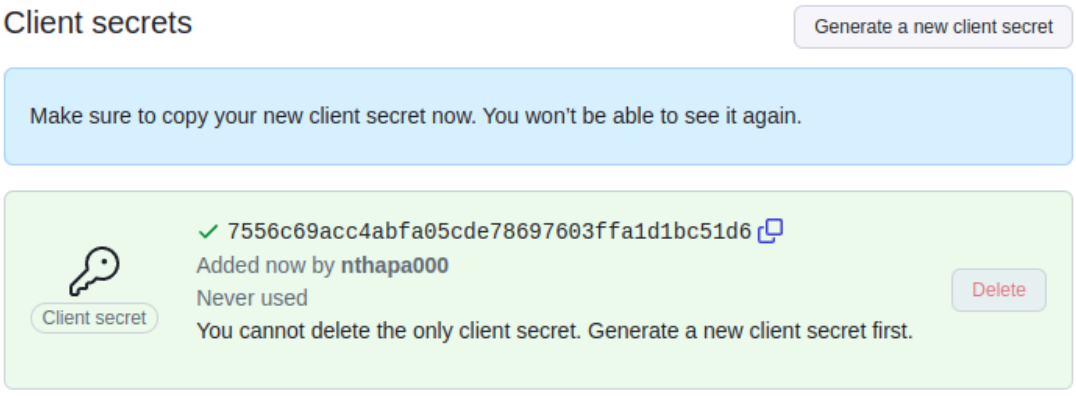
For Authorization callBackUrl we can find it in

http://localhost:3000/api/auth/providers


Settings / Developer Settings
Register a new OAuth application
Application name *
next-auth
Something users will recognize and trust.
Homepage URL *
http://localhost:3000
The full URL to your application homepage.
Application description
Application description is optional
This is displayed to all users of your application.
Authorization callback URL *
http://localhost:3000/api/auth/callback/github
Your application's callback URL. Read our OAuth documentation for more information.
Enable Device Flow
Allow this OAuth App to authorize users via the Device Flow.
Read the Device Flow documentation for more information.
Register application
Cancel

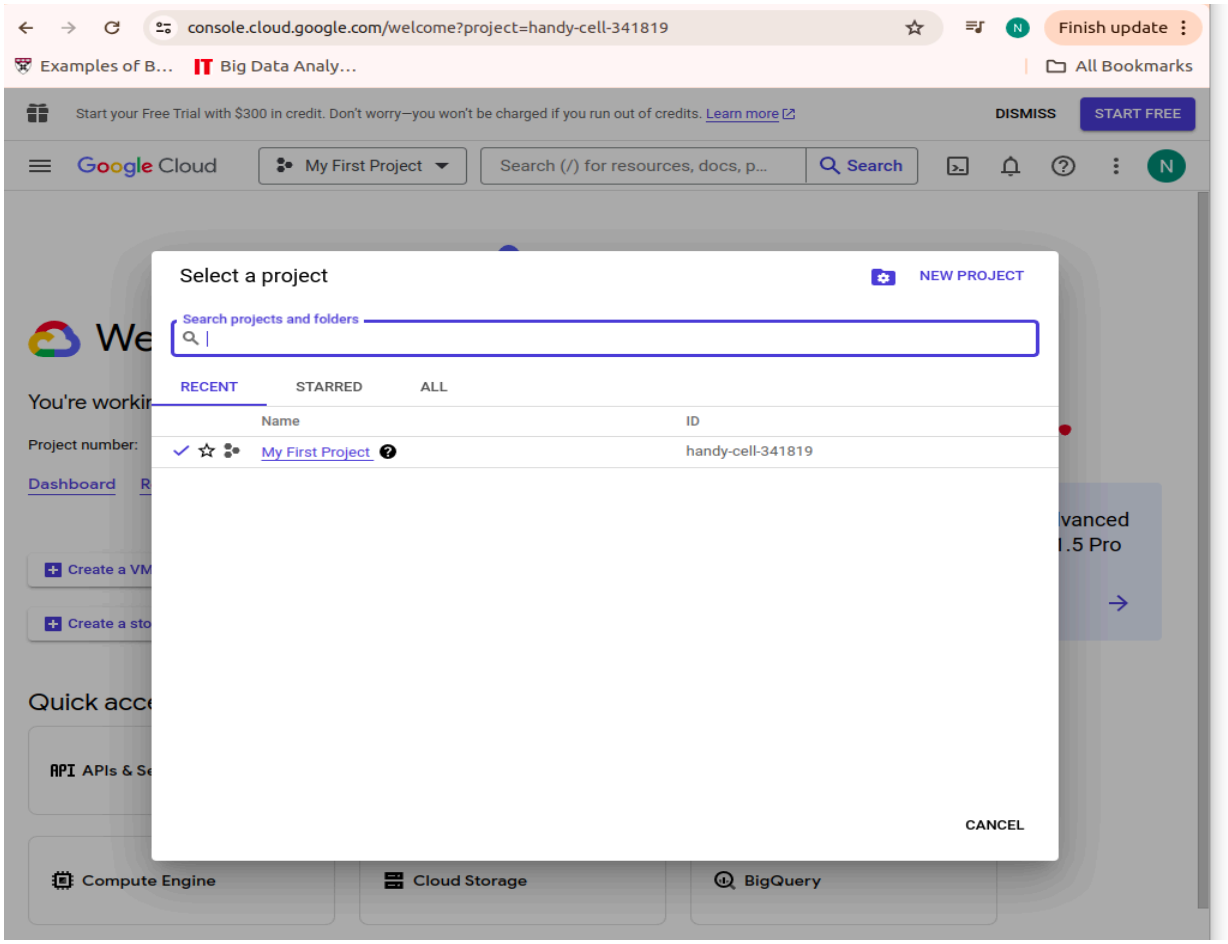


Generate a new client secret

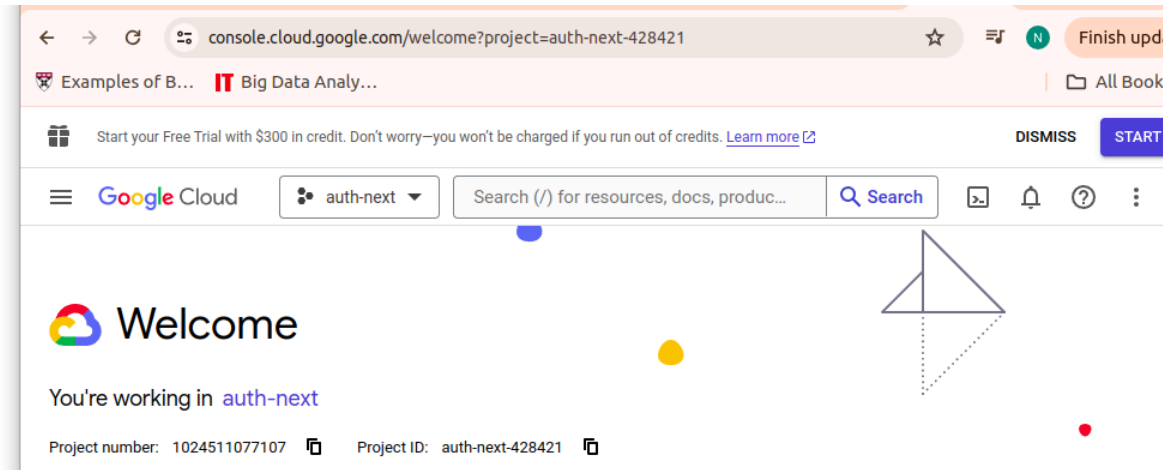


Now lets do the samething for the Google

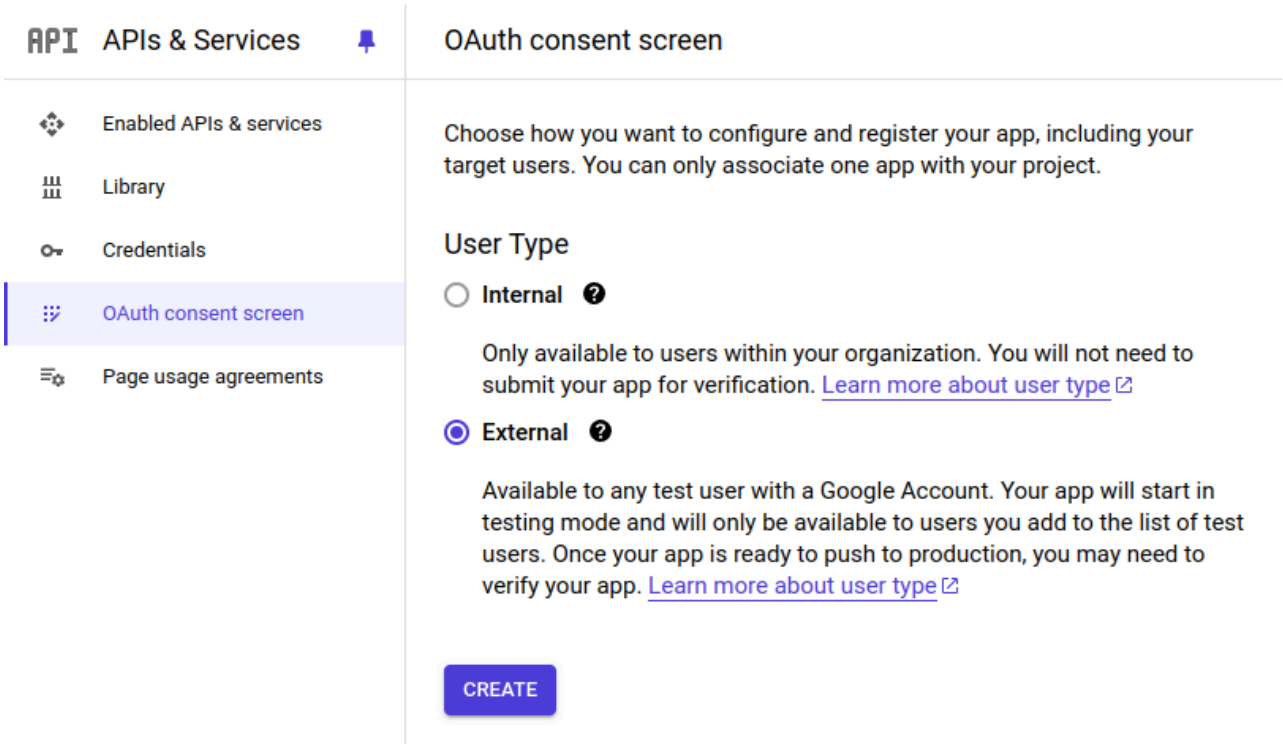
Type Google api console and create a new project



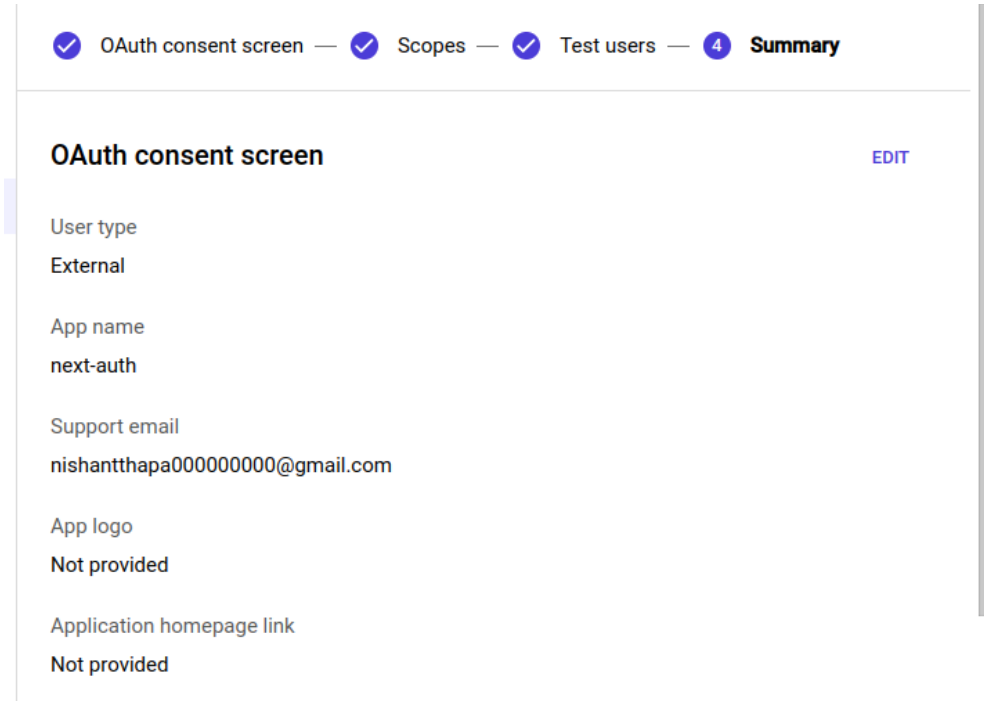
Homepage of the project



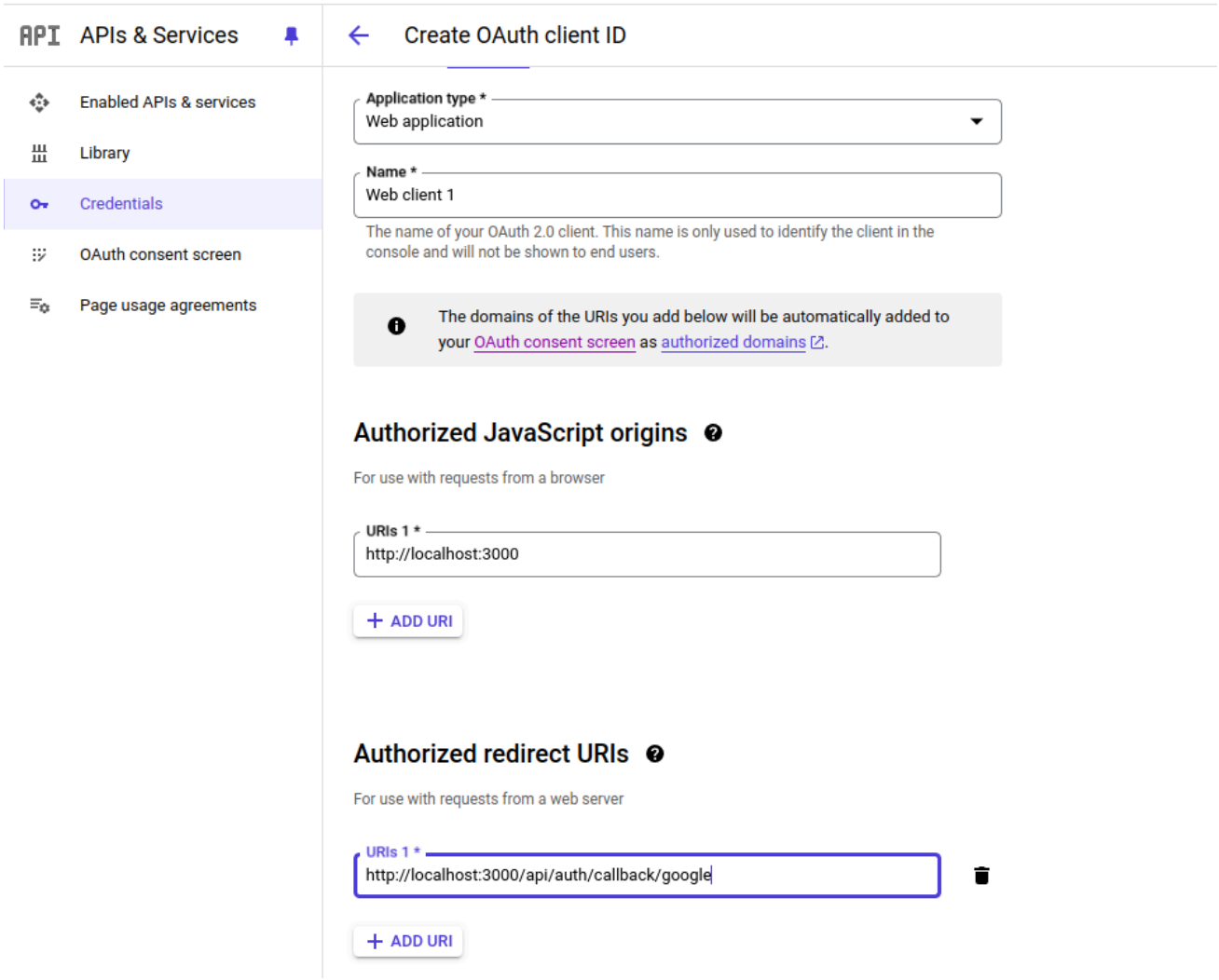
And now search for API and services and then create **OAuth consent screen**



Finish the form

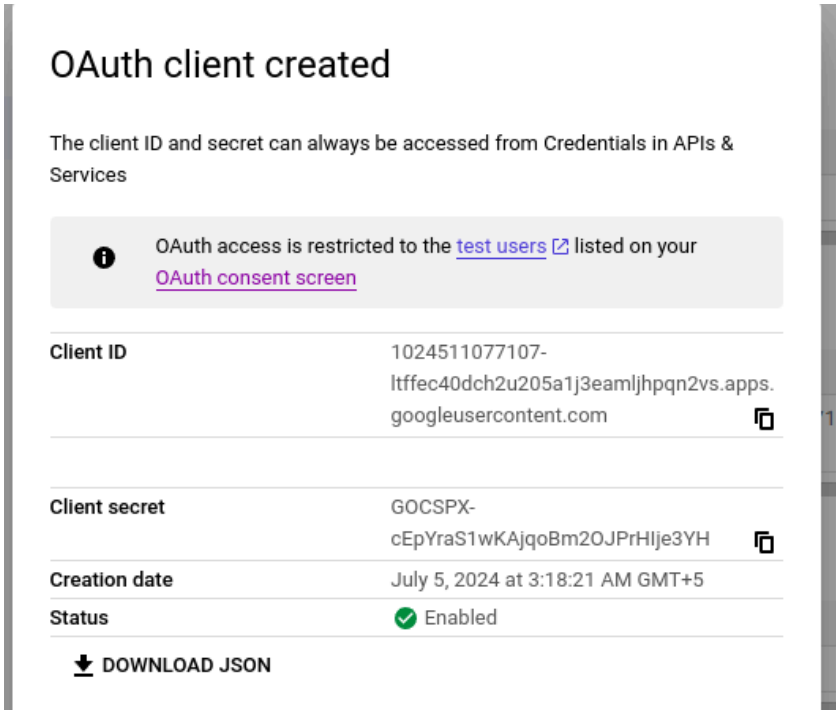


Now go to the Credentials and then Create Credentials and create OAuth client Id



Client Id : 1024511077107-ltffec40dch2u205a1j3eamljhpqn2vs.apps.googleusercontent.com

and Secret : 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



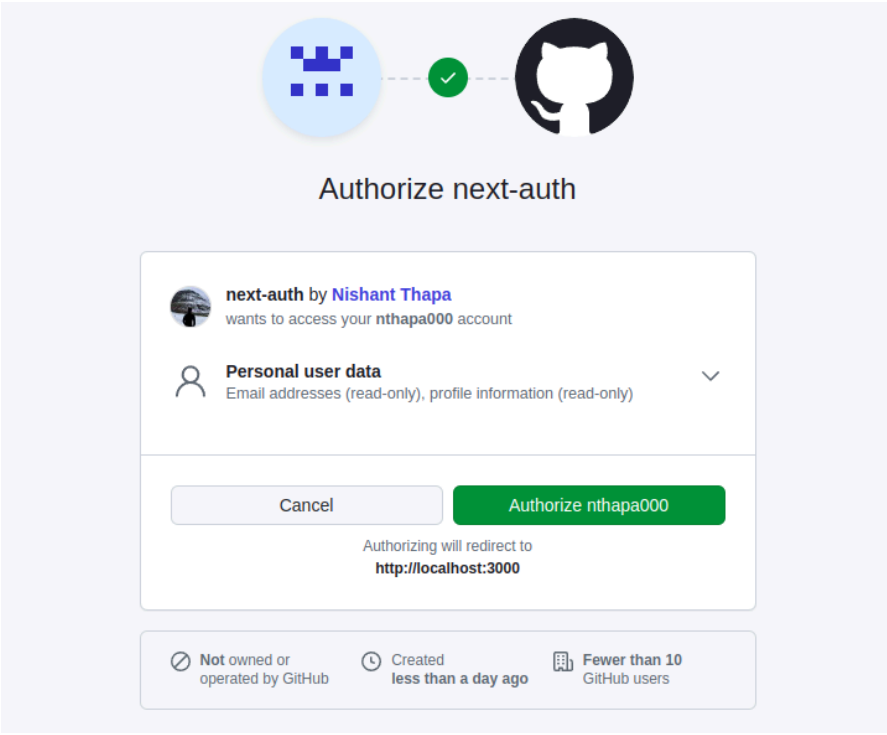
Now we have gotten the Client Id and secret now implement sign in functionality

components/auth/social.tsx

```tsx
"use client";

// If we want to do SignIn in client component
import { signIn } from "next-auth/react";
import { FcGoogle } from "react-icons/fc";
import { FaGithub } from "react-icons/fa";
import { Button } from "../ui/button";
import { DEFAULT_LOGIN_REDIRECT } from "@/route";

export const Social = () => {
   const onClick = (provider:"google" | "github")=>{
       signIn(provider,{
           callbackUrl:DEFAULT_LOGIN_REDIRECT,
       })
   }
   return (
       <div className="flex items-center w-full gap-x-2">
          {/* Google */}
          <Button
```

```
            size="lg"
            className="w-full"
            variant="outline"
            onClick={()=>onClick("google")}
          >
              <FcGoogle className="h-5 w-5"/>
            </Button>

            {/* Github */}
            <Button
            size="lg"
            className="w-full"
            variant="outline"
            onClick={()=>onClick("github")}
          >
              <FaGithub className="h-5 w-5"/>
            </Button>
        </div>
    )
}
```

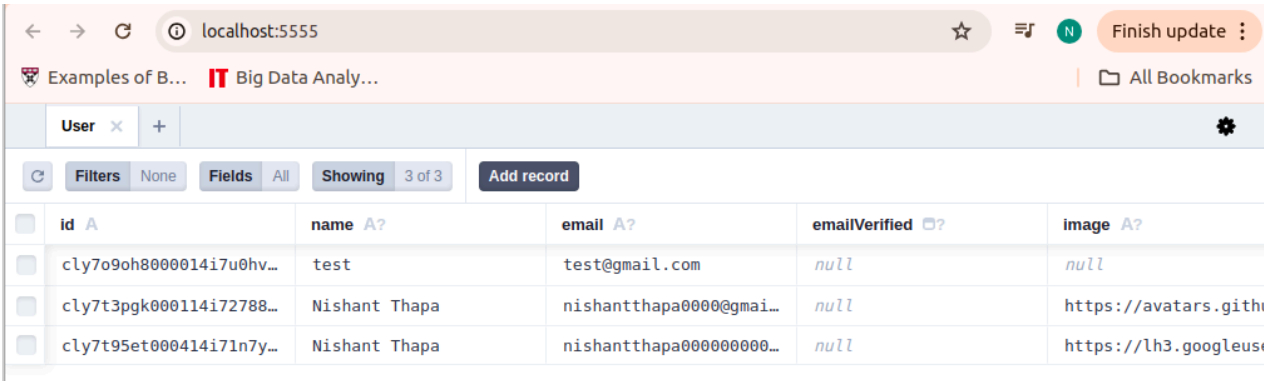
Lets test these things



Now on successfully signed in we will be redirected to the settings page

We can also confirm the user in the npx prisma studio and we also have a account connected to it

Now lets try with the Google , make sure to try with the different email which is not present in the database



We can see the user

Email verification will be done only for Credentials and will not be done for Google/ Github as in case for Google itself does it (has two factor authentication) Hence now we will use **linkAccount** event which mean thas OAuth provider was used and then we will automatically verify email verification field.

Now delete those two record from the database we can do it form the prisma studio

Events

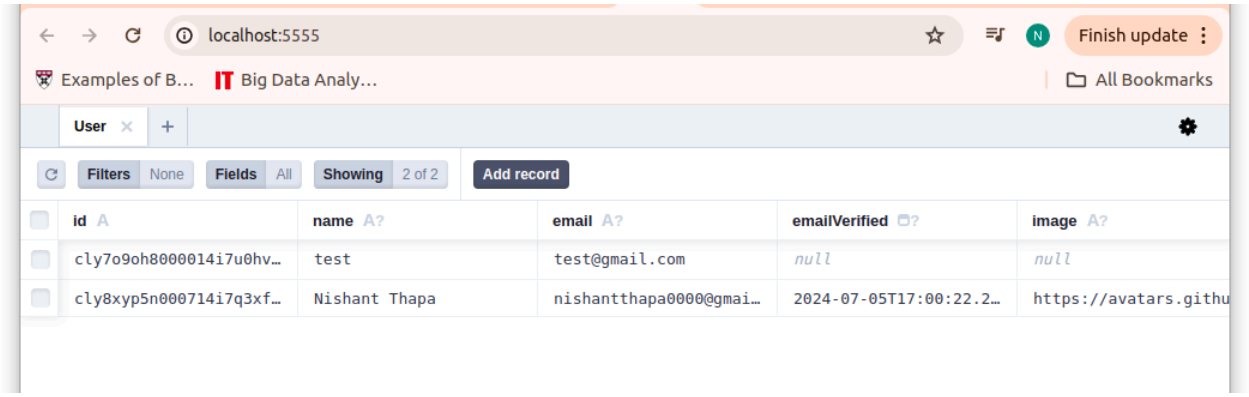
https://next-auth.js.org/configuration/events

auth.ts

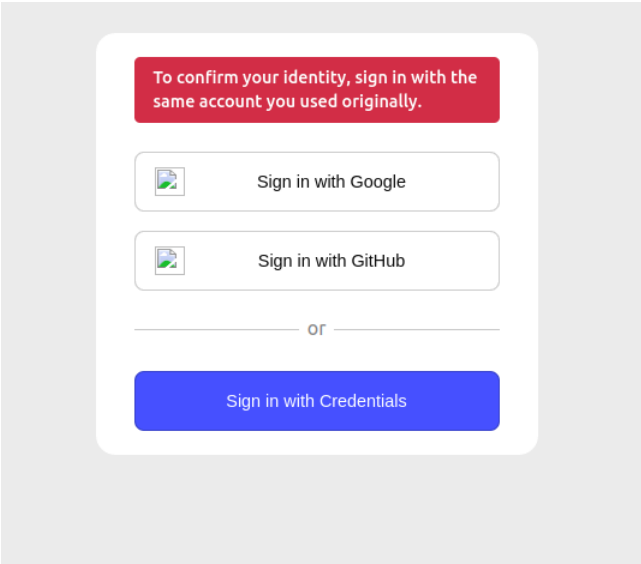
```
events:{
 async linkAccount({user}){
    await db.user.update({
      where: {id:user.id},
      data: {emailVerified:new Date()}
    })
  }
},
```

Now again use Github

And then when we see on prisma studio will have emailVerified section done



If we use the same email as github to access through gmail then we will get this error ,



Add pages in auth.ts

```
pages:{
  signIn:"/auth/login",
  error:"/auth/error"
},
```

Now nextAuth will go to this page if anything goes wrong

Lets create auth/error page

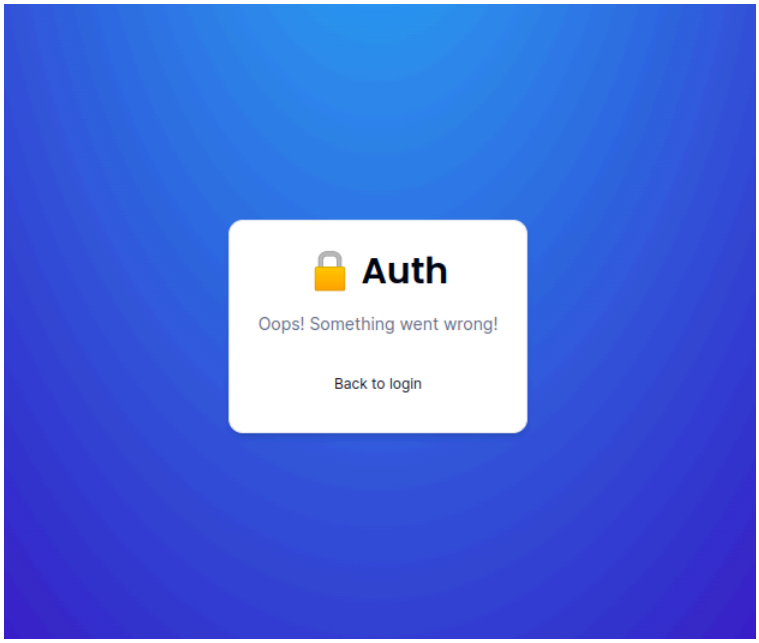
```
const AuthErrorPage = () => {
 return (
    <ErrorCard />
 )
}

export default AuthErrorPage;
```

Add this route to route.ts

Lets create ErrorCard component

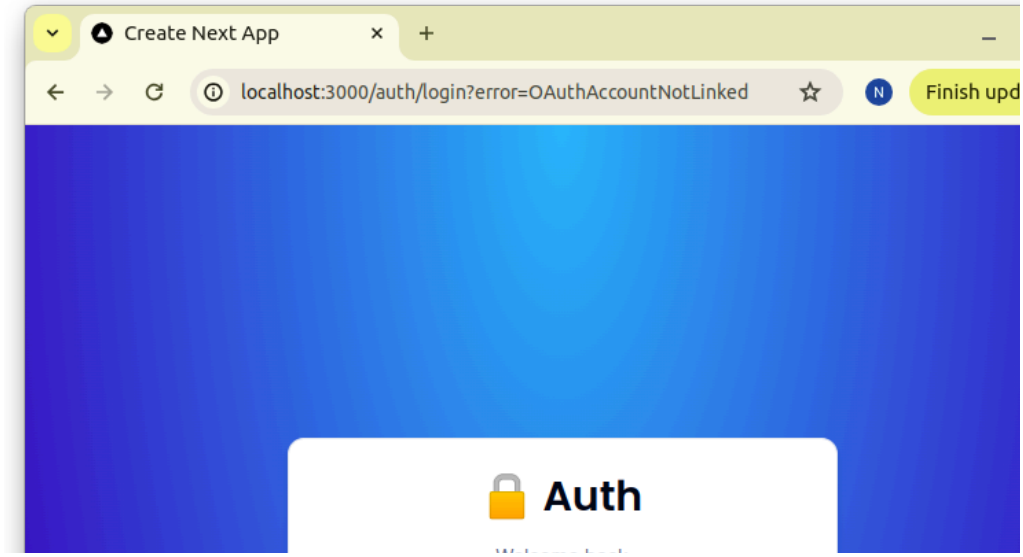
```
import { Header } from "./header";
import { BackButton } from "./back-button";
import {
```

```
    Card,
    CardFooter,
    CardHeader
} from "@/components/ui/card"

export const ErrorCard = () =>{
    return (
        <Card className="w=[400px] shadow-md">
            <CardHeader>
                <Header label="Oops! Something went wrong!"/>
            </CardHeader>
            <CardFooter>
                <BackButton
                    label="Back to login"
                    href="/auth/login"
                >
                </BackButton>
            </CardFooter>
        </Card>
    )
}
```



Check at 3hr 42 min to find another way to write the error-card component

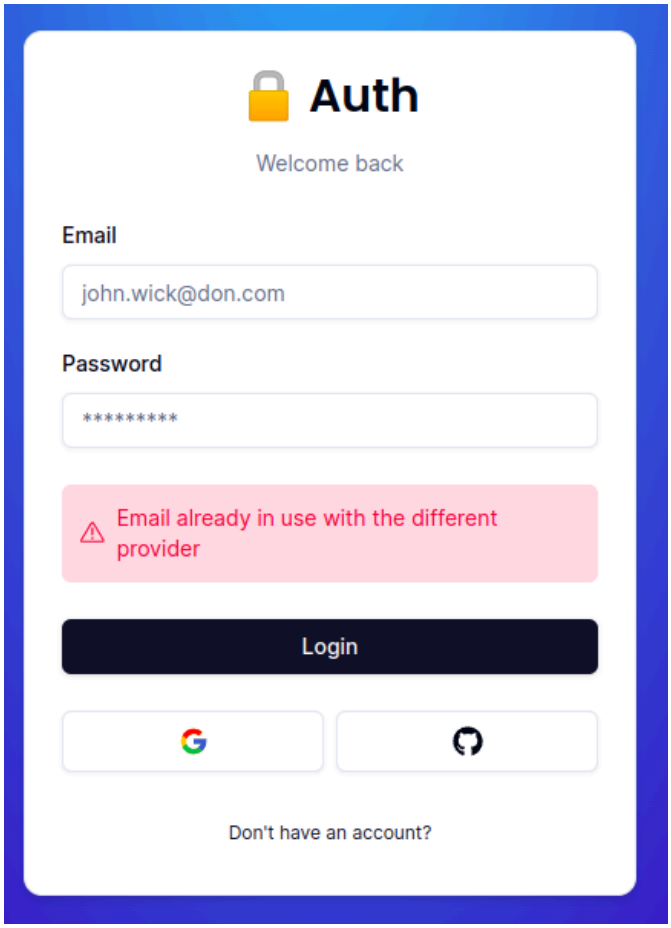
Now when we try to login with the same email as github then we get redirected to sign in page



Focus on the url, we will use it to show the error in this page itself

components/auth/login-form.tsx

```
import { useSearchParams } from "next/navigation";

export const LoginForm = () => {
 const searchParams = useSearchParams();
 const urlError = searchParams.get("error") === "OAuthAccountNotLinked"
 ? "Email already in use with the different provider "
:"";
…..
…
..
<FormError message={error || urlError}/>
```



Repository at this point

https://github.com/nthapa000/authentication_next/tree/a7ee2c7bc74219c31fe85dd60c1b244d0d0c16d9