

Source:

<https://github.com/hiteshchoudhary/nextjs-fullstack-auth>

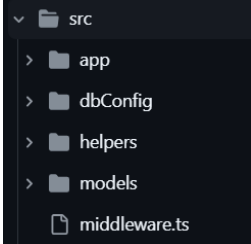
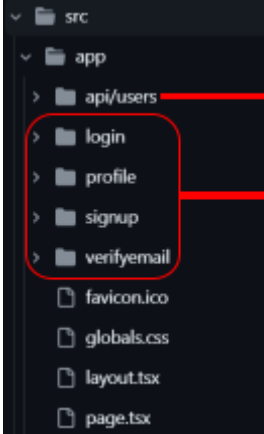
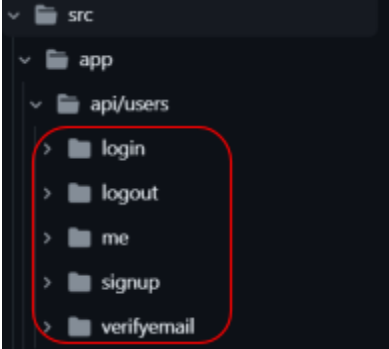
<https://www.youtube.com/playlist?list=PLu71SKxNbfoCXO80Z4miZHTL5GxfFbz7A>

Features:

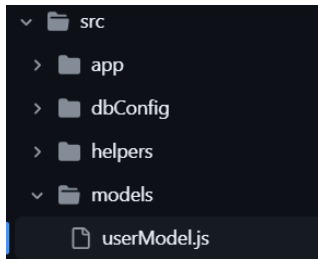
Forget password

User verification

## Folder structures

/src/	/src/app/	/src/app/api/user
 <p>A screenshot of a file explorer showing the /src/ directory. It contains subdirectories: app, dbConfig, helpers, and models. A file named middleware.ts is also visible.</p>	 <p>A screenshot of a file explorer showing the /src/app/ directory. It contains subdirectories: api/users, login, profile, signup, and verifyemail. Files include favicon.ico, globals.css, layout.tsx, and page.tsx. Red arrows point from the api/users directory to the text 'Backend' and from the login, profile, signup, and verifyemail directories to the text 'Frontend'.</p>	 <p>A screenshot of a file explorer showing the /src/app/api/user directory. It contains subdirectories: login, logout, me, signup, and verifyemail.</p>

### src/models/userModel.js



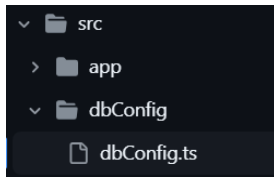
```
import mongoose from "mongoose";

const userSchema = new mongoose.Schema({
  username: {
    type: String,
    required: [true, "Please provide a username"],
    unique: true,
  },
  email: {
    type: String,
    required: [true, "Please provide a email"],
    unique: true,
  },
  password: {
    type: String,
    required: [true, "Please provide a password"],
  },
  isVerified: {
    type: Boolean,
    default: false,
  },
  isAdmin: {
    type: Boolean,
    default: false,
  },
  forgotPasswordToken: String,
  forgotPasswordTokenExpiry: Date,
  verifyToken: String,
  verifyTokenExpiry: Date,
});

const User = mongoose.models.users || mongoose.model("users", userSchema);

export default User;
```

### src/dbConfig/dbConfig.ts



```
import mongoose from 'mongoose';

export async function connect() {
  try {
    mongoose.connect(process.env.MONGO_URI!);
    const connection = mongoose.connection;

    connection.on('connected', () => {
      console.log('MongoDB connected successfully');
    })

    connection.on('error', (err) => {
      console.log('MongoDB connection error. Please make sure MongoDB is running. ' + err);
      process.exit();
    })

  } catch (error) {
    console.log('Something goes wrong!');
    console.log(error);
  }
}
```

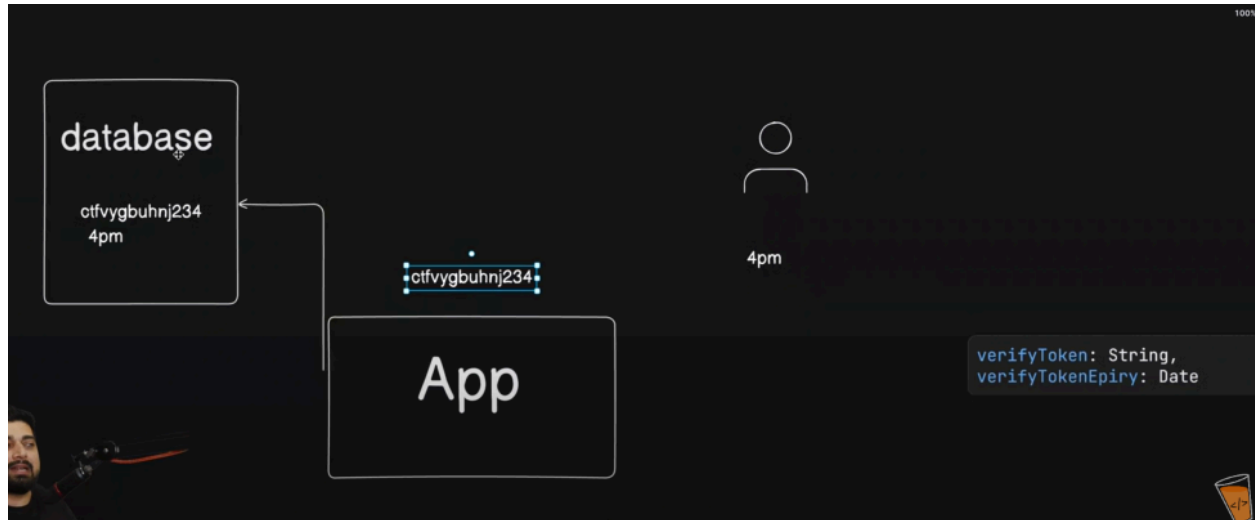
### .env

```
MONGO_URI=mongodb+srv://hitesh:subscribe@cluster0.nwxsjhi.mongodb.net/
TOKEN_SECRET=nexttjsyoutube
DOMAIN=http://localhost:3000
```

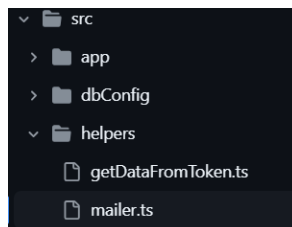
## Verification process:

1. App generate unique token and send one copy to database and another to user in the form of email
2. If user click the button then token goes to app. Then app asked to db either that token is in the db and expire time is matched.
3. If token is matched and not expired then we told them you are verified.

**Note:** Process is same for **forget password**



## src/helpers/mailer.ts



```
npm install nodemailer
```

Kaslaai pathaune

Can be either  
**verifyEmail** or  
**forgetPassword** or  
....

```
export const sendEmail = async({email, emailType, userId}:any) => {
```

**Import Dependencies:** Load `nodemailer`, `User` model, and `bcryptjs` for email handling, database updates, and token hashing.

**Hash User ID:** Generate a hashed token from the user ID for secure email links.

**Update User Model:** Based on email type ("VERIFY" or "RESET"), update the user's verification or reset token and expiry time in the database.

**Set Up Nodemailer Transport:** Use Mailtrap's SMTP service to send test emails, with authentication credentials moved to the `.env` file for security.

**Compose Email:** Create a dynamic email body with a verification or reset link containing the hashed token.

**Send Email:** Use `transport.sendMail()` to send the email.

**Handle Errors:** Catch any errors during the process and throw a descriptive error message.

```
import nodemailer from 'nodemailer';
import User from "@/models/userModel";
import bcryptjs from 'bcryptjs';

export const sendEmail = async({email, emailType, userId}:any) => {
  try {
    // create a hashed token
    const hashedToken = await bcryptjs.hash(userId.toString(), 10)

    if (emailType === "VERIFY") {
      await User.findByIdAndUpdate(userId,
        {verifyToken: hashedToken, verifyTokenExpiry: Date.now() + 3600000})
    } else if (emailType === "RESET"){
      await User.findByIdAndUpdate(userId,
        {forgotPasswordToken: hashedToken, forgotPasswordTokenExpiry: Date.now() + 3600000})
    }

    var transport = nodemailer.createTransport({
      host: "sandbox.smtp.mailtrap.io",
      port: 2525,
      auth: {
        user: "3fd364695517df",
        pass: "7383d58fd399cf"
        //TODO: add these credentials to .env file
      }
    });

    const mailOptions = {
      from: 'hitesh@gmail.com',
      to: email,
      subject: emailType === "VERIFY" ? "Verify your email" : "Reset your password",
      html: `<p>Click <a href="${process.env.DOMAIN}/verifyemail?token=${hashedToken}">here</a> to
    ${emailType === "VERIFY" ? "verify your email" : "reset your password"}
    or copy and paste the link below in your browser. <br>
    ${process.env.DOMAIN}/verifyemail?token=${hashedToken}
```

```

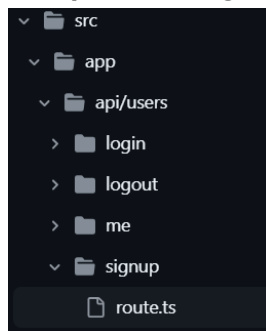
    </p>`
  }

  const mailresponse = await transport.sendMail
    (mailOptions);
  return mailresponse;

} catch (error:any) {
  throw new Error(error.message);
}
}

```

### ***src/api/users/Signup***



**Connect to Database:** Ensure the database is connected using `connect()`.

**Parse Request Body:** Extract `username`, `email`, and `password` from the incoming request.

**Check for Existing User:** Use `User.findOne()` to check if a user with the given email already exists. Return an error if so.

**Hash Password:** Generate a salt and hash the password using `bcryptjs`.

**Create New User:** Save the new user with the hashed password to the database.

**Send Verification Email:** Trigger an email with a verification link using `sendEmail()`.

**Return Success Response:** Send a success response with the created user data.

**Handle Errors:** Catch and return any errors that occur during the process.

```

import {connect} from "@dbConfig/dbConfig";
import User from "@models/userModel";
import { NextRequest, NextResponse } from "next/server";
import bcryptjs from "bcryptjs";
import { sendEmail } from "@helpers/mailer";

connect()

export async function POST(request: NextRequest){
  try {
    const reqBody = await request.json()
    const {username, email, password} = reqBody

    console.log(reqBody);

    //check if user already exists
    const user = await User.findOne({email})

    if(user){
      return NextResponse.json({error: "User already exists"}, {status: 400})
    }

    //hash password
    const salt = await bcryptjs.genSalt(10)
    const hashedPassword = await bcryptjs.hash(password, salt)

    const newUser = new User({
      username,
      email,
      password: hashedPassword
    })

    const savedUser = await newUser.save()
    console.log(savedUser);

    //send verification email

    await sendEmail({email, emailType: "VERIFY", userId: savedUser._id})

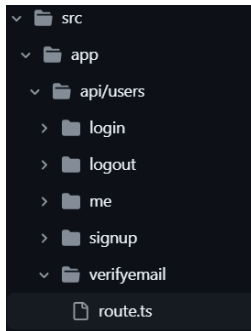
    return NextResponse.json({
      message: "User created successfully",
      success: true,
      savedUser
    })

  } catch (error: any) {
    return NextResponse.json({error: error.message}, {status: 500})
  }
}

```

After signup we want to verify email

**src/app/api/users/verifyemail**



**Connect to Database:** Ensure connection using `connect()`.

**Parse Request Body:** Extract the `token` from the request body.

**Find User by Token:** Search for a user with the provided `verifyToken` and valid `verifyTokenExpiry`.

**Handle Invalid Token:** If no matching user is found or the token is expired, return an error response.

**Verify User:** Set `isVerified` to true and clear the `verifyToken` and `verifyTokenExpiry`.

**Save Updated User:** Save the changes to the user in the database.

**Return Success Response:** Send a success message if email verification is completed.

**Handle Errors:** Catch and return any errors that occur during the process.

```
import {connect} from "@dbConfig/dbConfig";
import { NextRequest, NextResponse } from "next/server";
import User from "@models/userModel";

connect()

export async function POST(request: NextRequest){

  try {
    const reqBody = await request.json()
    const {token} = reqBody
    console.log(token);

    const user = await User.findOne({verifyToken: token, verifyTokenExpiry: {$gt: Date.now()}});

    if (!user) {
      return NextResponse.json({error: "Invalid token"}, {status: 400})
    }
    console.log(user);

    user.isVerified = true;
    user.verifyToken = undefined;
    user.verifyTokenExpiry = undefined;
```



```

    await user.save();

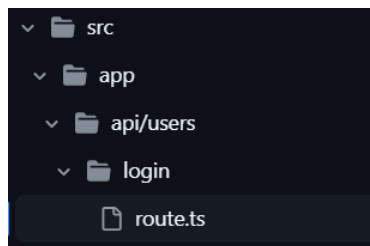
    return NextResponse.json({
      message: "Email verified successfully",
      success: true
    })

  } catch (error:any) {
    return NextResponse.json({error: error.message}, {status: 500})
  }
}

```

After verify email we want to login

**src/app/api/users/login**



**Connect to Database:** Establish connection using `connect()`.

**Parse Request Body:** Extract `email` and `password` from the request body.

**Check if User Exists:** Search for the user in the database; return an error if not found.

**Validate Password:** Compare the provided password with the hashed one; return an error if invalid.

**Create Token Data:** Prepare user data (`id`, `username`, `email`) for the token.

**Generate JWT:** Sign the token with `jwt` and set it to expire in 1 day.

**Set Token in Cookies:** Store the token in an HTTP-only cookie.

**Return Success Response:** Send a success message if login is successful.

**Handle Errors:** Catch and return any errors encountered during the process.

```

import {connect} from "@dbConfig/dbConfig";
import User from "@models/userModel";
import { NextRequest, NextResponse } from "next/server";
import bcryptjs from "bcryptjs";
import jwt from "jsonwebtoken";

connect()

export async function POST(request: NextRequest){
  try {

```

```

const reqBody = await request.json()
const {email, password} = reqBody;
console.log(reqBody);

//check if user exists
const user = await User.findOne({email})
if(!user){
    return NextResponse.json({error: "User does not exist"}, {status: 400})
}
console.log("user exists");

//check if password is correct
const validPassword = await bcryptjs.compare(password, user.password)
if(!validPassword){
    return NextResponse.json({error: "Invalid password"}, {status: 400})
}
console.log(user);

//create token data
const tokenData = {
    id: user._id,
    username: user.username,
    email: user.email
}

//create token
const token = await jwt.sign(tokenData, process.env.TOKEN_SECRET!, {expiresIn: "1d"})

const response = NextResponse.json({
    message: "Login successful",
    success: true,
})
response.cookies.set("token", token, {
    httpOnly: true,
})

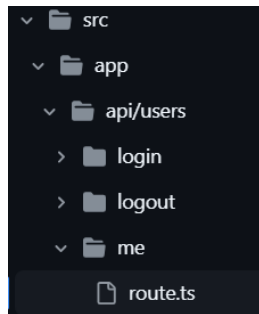
return response;

} catch (error: any) {
    return NextResponse.json({error: error.message}, {status: 500})
}
}

```

After login we want to get our information or profile

**src/app/api/users/me**



**Connect to Database:** Ensure connection using `connect()`.

**Extract User ID from Token:** Retrieve the user ID using `getDataFromToken()`.

**Find User by ID:** Query the database for the user by ID, excluding the `password` field.

**Return User Data:** Send a response with the user's data if found.

**Handle Errors:** Catch and return any errors encountered, with a status of `400`.

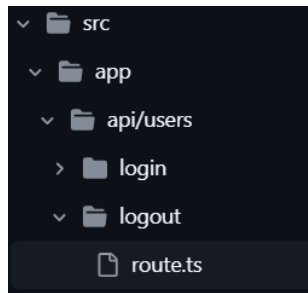
```
import { getDataFromToken } from "@helpers/getDataFromToken";

import { NextRequest, NextResponse } from "next/server";
import User from "@models/userModel";
import { connect } from "@dbConfig/dbConfig";

connect();

export async function GET(request: NextRequest) {
  try {
    const userId = await getDataFromToken(request);
    const user = await User.findOne({ _id: userId }).select("-password");
    return NextResponse.json({
      mesaaage: "User found",
      data: user
    });
  } catch (error: any) {
    return NextResponse.json({ error: error.message }, { status: 400 });
  }
}
```

At last user want to logout  
**src/app/api/users/logout**



**Create Logout Response:** Construct a success response with the message "Logout successful".

**Clear JWT Cookie:** Set the `token` cookie to an empty value and set its expiration date to the past, effectively removing it.

**Return Success Response:** Return the response confirming successful logout.

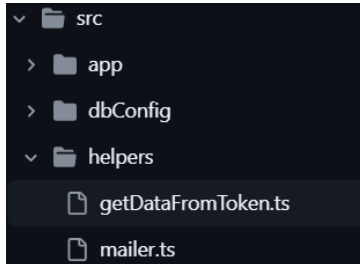
**Handle Errors:** Catch any errors and return an error response with status `500`.

```
import { NextResponse } from "next/server";

export async function GET() {
  try {
    const response = NextResponse.json(
      {
        message: "Logout successful",
        success: true,
      }
    )
    response.cookies.set("token", "",
      { httpOnly: true, expires: new Date(0)
    });
    return response;
  } catch (error: any) {
    return NextResponse.json({ error: error.message }, { status: 500 });
  }
}
```

Login gardaa hami cookies maa token (jwt token) set garekaa thiyau.  
Aba “me” bhanne route maa token bata data (id) extract garera database bata data nikaalxau.  
Data extract ko laagi xutai helper function banaayako xa.

### ***src/helper/getDataFromToken.ts***



**Retrieve Token from Cookies:** Extract the JWT from the `token` cookie in the request.

**Verify Token:** Use `jwt.verify()` to decode the token using the secret stored in `process.env.TOKEN_SECRET`.

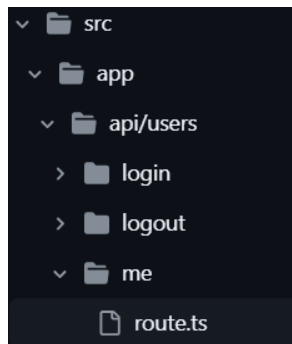
**Return User ID:** Return the user ID (`decodedToken.id`) from the decoded token.

**Handle Errors:** If an error occurs during token verification, throw an error with the error message.

```
import { NextRequest } from "next/server";
import jwt from "jsonwebtoken";

export const getDataFromToken = (request: NextRequest) => {
  try {
    const token = request.cookies.get("token")?.value || '';
    const decodedToken:any = jwt.verify(token, process.env.TOKEN_SECRET!);
    return decodedToken.id;
  } catch (error: any) {
    throw new Error(error.message);
  }
}
```

### src/app/api/me



**Connect to Database:** Establish connection using `connect()`.

**Extract User ID from Token:** Call `getDataFromToken(request)` to retrieve the user ID from the token.

**Find User by ID:** Query the `User` model for the user with the matching ID, excluding the password.

**Return User Data:** Send a response with the user's data and a success message if found.

**Handle Errors:** Catch and return any errors encountered, with an error message and `status: 400`.

```
import { getDataFromToken } from "@helpers/getDataFromToken";

import { NextRequest, NextResponse } from "next/server";
import User from "@models/userModel";
import { connect } from "@dbConfig/dbConfig";

connect();

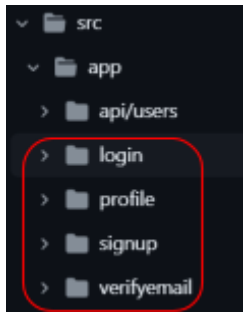
export async function GET(request: NextRequest) {

  try {
    const userId = await getDataFromToken(request);
    const user = await User.findOne({ _id: userId }).select("-password");
    return NextResponse.json({
      mesaaage: "User found",
      data: user
    })
  } catch (error: any) {
    return NextResponse.json({ error: error.message }, { status: 400 });
  }
}
```

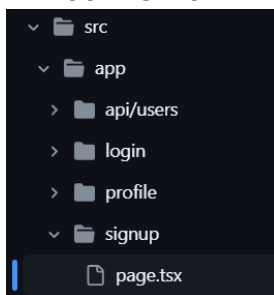
# Front-End

## Tools:

- Axios
- React-hot-toast



## *src/app/signup*



```
"use client";
import Link from "next/link";
import React, { useEffect } from "react";
import { useRouter } from "next/navigation";
import axios from "axios";
import { toast } from "react-hot-toast";

export default function SignupPage() {
  const router = useRouter();
  const [user, setUser] = React.useState({
    email: "",
    password: "",
    username: "",
  });
  const [buttonDisabled, setButtonDisabled] = React.useState(false);
  const [loading, setLoading] = React.useState(false);

  const onSignup = async () => {
```

```

    try {
      setLoading(true);
      const response = await axios.post("/api/users/signup", user);
      console.log("Signup success", response.data);
      router.push("/login");
    } catch (error:any) {
      console.log("Signup failed", error.message);

      toast.error(error.message);
    } finally {
      setLoading(false);
    }
  }
}

useEffect(() => {
  if(user.email.length > 0 && user.password.length > 0 && user.username.length > 0) {
    setButtonDisabled(false);
  } else {
    setButtonDisabled(true);
  }
}, [user]);

return (

```

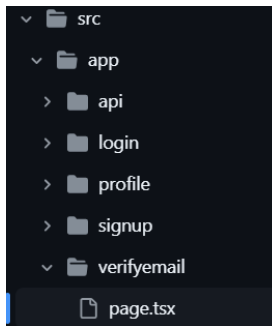


```

        onChange={(e) => setUser({...user, password: e.target.value})}
        placeholder="password"
      />
      <button
        onClick={onSignup}
        className="p-2 border border-gray-300 rounded-lg mb-4 focus:outline-none
focus:border-gray-600">{buttonDisabled ? "No signup" : "Signup"}</button>
      <Link href="/login">Visit login page</Link>
    </div>
  )
}

```

### ***src/app/verifyemail***



```

"use client";

import axios from "axios";
import Link from "next/link";
import React, { useEffect, useState } from "react";

export default function VerifyEmailPage() {

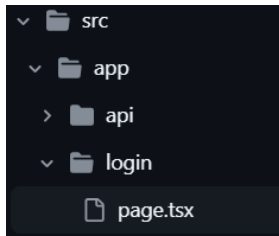
  const [token, setToken] = useState("");
  const [verified, setVerified] = useState(false);
  const [error, setError] = useState(false);

  const verifyUserEmail = async () => {
    try {
      await axios.post('/api/users/verifyemail', {token})
      setVerified(true);
    } catch (error:any) {
      setError(true);
      console.log(error.reponse.data);
    }
  }
}

```



## src/app/login



```
"use client";
import Link from "next/link";
import React, {useEffect} from "react";
import {useRouter} from "next/navigation";
import axios from "axios";
import { toast } from "react-hot-toast";

export default function LoginPage() {
  const router = useRouter();
  const [user, setUser] = React.useState({
    email: "",
    password: "",
  })

  const [buttonDisabled, setButtonDisabled] = React.useState(false);
  const [loading, setLoading] = React.useState(false);

  const onLogin = async () => {
    try {
      setLoading(true);
      const response = await axios.post("/api/users/login", user);
      console.log("Login success", response.data);
      toast.success("Login success");
      router.push("/profile");
    } catch (error:any) {
      console.log("Login failed", error.message);
      toast.error(error.message);
    } finally{
      setLoading(false);
    }
  }

  useEffect(() => {
    if(user.email.length > 0 && user.password.length > 0) {
      setButtonDisabled(false);
    } else{
      setButtonDisabled(true);
    }
  }, [user]);

  return (
```

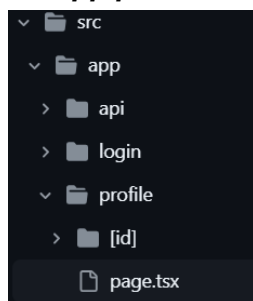
```

<div className="flex flex-col items-center justify-center min-h-screen py-2">
  <h1>{loading ? "Processing" : "Login"}</h1>
  <hr />

  <label htmlFor="email">email</label>
  <input
    className="p-2 border border-gray-300 rounded-lg mb-4 focus:outline-none focus:border-gray-600 text-black"
    id="email"
    type="text"
    value={user.email}
    onChange={(e) => setUser({...user, email: e.target.value})}
    placeholder="email"
  />
  <label htmlFor="password">password</label>
  <input
    className="p-2 border border-gray-300 rounded-lg mb-4 focus:outline-none focus:border-gray-600 text-black"
    id="password"
    type="password"
    value={user.password}
    onChange={(e) => setUser({...user, password: e.target.value})}
    placeholder="password"
  />
  <button
    onClick={onLogin}
    className="p-2 border border-gray-300 rounded-lg mb-4 focus:outline-none
focus:border-gray-600">Login here</button>
  <Link href="/signup">Visit Signup page</Link>
</div>
)
}

```

### **src/app/profile**



```

"use client";
import axios from "axios";
import Link from "next/link";
import React, {useState} from "react";
import {toast} from "react-hot-toast";
import {useRouter} from "next/navigation";

```

```

export default function ProfilePage() {
  const router = useRouter()
  const [data, setData] = useState("nothing")
  const logout = async () => {
    try {
      await axios.get('/api/users/logout')
      toast.success('Logout successful')
      router.push('/login')
    } catch (error:any) {
      console.log(error.message);
      toast.error(error.message)
    }
  }

  const getUserDetails = async () => {
    const res = await axios.get('/api/users/me')
    console.log(res.data);
    setData(res.data.data._id)
  }

  return (
    <div className="flex flex-col items-center justify-center min-h-screen py-2">
      <h1>Profile</h1>
      <hr />
      <p>Profile page</p>
      <h2 className="p-1 rounded bg-green-500">{data === 'nothing' ? "Nothing" : <Link
href={` /profile/${data}`}>{data}
      </Link></h2>
      <hr />
      <button
onClick={logout}
className="bg-blue-500 mt-4 hover:bg-blue-700 text-white font-bold py-2 px-4 rounded"
>Logout</button>

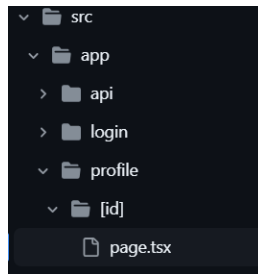
      <button
onClick={getUserDetails}
className="bg-green-800 mt-4 hover:bg-blue-700 text-white font-bold py-2 px-4 rounded"
>Get User Details</button>

    </div>
  )
}

```

This is for dynamic route

***src/app/id***



```
export default function UserProfile({params}: any) {  
  return (  
    <div className="flex flex-col items-center justify-center min-h-screen py-2">  
      <h1>Profile</h1>  
      <hr />  
      <p className="text-4xl">Profile page  
      <span className="p-2 m1-2 rounded bg-orange-500 text-black">{params.id}</span>  
      </p>  
    </div>  
  )  
}
```

## Middleware

**For route protection:** user logout bhayapaxi profile route maa jaana nadine

### Some facts

- Should be in **home** or **src** directory
- Name should be “**middleware**”

### OR Convention

Use the file **middleware.ts** (or **.js**) in the root of your project to define Middleware. For example, at the same level as **pages** or **app**, or inside **src** if applicable.

For more info:

<https://nextjs.org/docs/app/building-your-application/routing/middleware>

## Matcher

matcher allows you to filter Middleware to run on specific paths. (or matcher maa diyako path maa jaana bhandi agaadi middleware apply garne)

We need **path**: First we have to track user location. I.e, `const path = request.nextUrl.pathname`

We have **two paths**: 1. Public & 2. Private

```
const isPublicPath = path === '/login' || path === '/signup' || path === '/verifyemail'
```

**Rule 1:** User login xa bhane public path access garna dina bhayana.

To check, either the user is logged in or not. We have to take **token** . haami user ko token cookies maa raakhekaa thiyau, so haami cookies bata lina parsu.

```
const token = request.cookies.get('token')?.value || ''
```

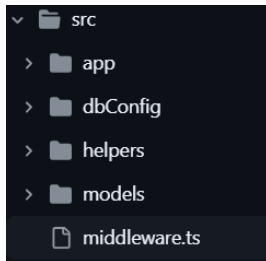
**Usecase:** token xa or logged in xa bhane public path access garna nadine

```
if(isPublicPath && token) {
  return NextResponse.redirect(new URL('/', request.nextUrl))
}

if (!isPublicPath && !token) {
  return NextResponse.redirect(new URL('/login', request.nextUrl))
}

// See "Matching Paths" below to learn more
export const config = {
  matcher: [
    '/',
    '/profile',
    '/login',
    '/signup',
    '/verifyemail'
  ]
}
```

## src/middleware.ts



```
import { NextResponse } from 'next/server'
import type { NextRequest } from 'next/server'

export function middleware(request: NextRequest) {
  const path = request.nextUrl.pathname

  const isPublicPath = path === '/login' || path === '/signup' || path === '/verifyemail'

  const token = request.cookies.get('token')?.value || ''

  if(isPublicPath && token) {
    return NextResponse.redirect(new URL('/', request.nextUrl))
  }

  if (!isPublicPath && !token) {
    return NextResponse.redirect(new URL('/login', request.nextUrl))
  }
}

// See "Matching Paths" below to learn more
export const config = {
  matcher: [
    '/',
    '/profile',
    '/login',
    '/signup',
    '/verifyemail'
  ]
}
```