**Implementing Role-Based Access Control in Next.js, Next-Auth with Prisma and MongoDB**

[[](https://medium.com/@chsherryy?source=post_page-----324f1929cf93--------------------------------)](https://medium.com/@chsherryy?source=post_page-----324f1929cf93--------------------------------)

[Chaudhary Shaharyar Tariq](https://medium.com/@chsherryy?source=post_page-----324f1929cf93--------------------------------)

·

Follow

3 min read

·

Jan 3

44

**Introduction:**

In modern web applications, ensuring that users have the right permissions to access various parts of your app is crucial. This blog post will guide you through setting up role-based access control (RBAC) in a Next.js application, utilizing Prisma for data handling, MongoDB as the database, and Next-Auth for authentication.

**Scenario:**

Revolutionizing healthcare data management with our EHR system built on Next.js, Prisma, MongoDB, and NextAuth. Experience seamless navigation and robust security through role-based authentication, ensuring precise access controls for healthcare professionals. Elevate efficiency and reliability in patient care with our state-of-the-art solution

End Users:

1. Doctor
2. Pathologist
3. Nurse
4. Receptionist

**Getting Started with NextAuth**

NextAuth is a complete open-source authentication solution for Next.js applications. It’s easy to integrate and simplifies handling user authentication.

**Configuring NextAuth for Authentication**

Here’s a basic configuration for NextAuth in your Next.js app. Save this in your api/[...nextauth].js file:

import CredentialsProvider from "next-auth/providers/credentials";  
import prisma from "@app/utils/prismadb";  
import bcrypt from "bcrypt";  
  
  
export const options = {  
 providers: [  
 CredentialsProvider({  
 name: "Credentials",  
 credentials: {  
 email: {  
 label: "email:",  
 type: "text",  
 },  
 password: {  
 label: "password:",  
 type: "password",  
 },  
 },  
 async authorize(credentials) {  
 try {  
 const user = await prisma.user.findUnique({  
 where: {  
 email: credentials.email,  
 },  
 });  
  
 if (!user || !user?.hashedPassword) {  
 throw new Error("Invalid credentials");  
 }  
  
 const isCorrectPassword = await bcrypt.compare(  
 credentials.password,  
 user.hashedPassword  
 );  
  
 if (!isCorrectPassword) {  
 throw new Error("Invalid credentials");  
 }  
 return { ...user, password: null, role: user.role };  
 } catch (error) {  
 console.log(error);  
 }  
 return null;  
 },  
 }),  
 ],  
 secret: process.env.SECRET,  
 debug: process.env.NODE\_ENV === "development",  
 callbacks: {  
 async jwt({ token, user }) {  
 if (user) token.role = user.role;  
 return token;  
 },  
 async session({ session, token }) {  
 if (session?.user) session.user.role = token.role;  
 return session;  
 },  
 },  
};

This snippet sets up a Credentials Provider and uses Prisma to interact with your database. Users are authenticated by checking the provided credentials against the stored user data.

**Middleware for Role-Based Authorization**

Middleware in Next.js allows you to run code before a request is completed. By leveraging this, we can implement a RBAC system.

Here’s how you can set up a middleware for role-based authorization:

import { NextResponse } from "next/server";  
import { getToken } from "next-auth/jwt";  
  
export async function middleware(request) {  
 const token = await getToken({ req: request, secret: process.env.SECRET });  
  
 if (!token) return NextResponse.redirect(new URL("/login", request.url));  
  
 // Check the role and redirect based on the role  
 switch (token.role) {  
 case "RECEPTIONIST":  
 if (!request.nextUrl.pathname.startsWith("/profile")) {  
 return NextResponse.redirect(new URL("/profile", request.url));  
 }  
 break;  
 case "DOCTOR":  
 if (  
 !request.nextUrl.pathname.startsWith("/patients") &&  
 !request.nextUrl.pathname.startsWith("/patientprofile") &&  
 !request.nextUrl.pathname.startsWith("/complain") &&  
 !request.nextUrl.pathname.startsWith("/report")  
 ) {  
 return NextResponse.redirect(new URL("/patients", request.url));  
 }  
 break;  
 case "NURSE":  
 // Add the paths that the nurse can access here  
 if (!request.nextUrl.pathname.startsWith("/vitals")) {  
 return NextResponse.redirect(new URL("/vitals", request.url));  
 }  
 break;  
 case "PATHOLOGIST":  
 // Add the paths that the pathologist can access here  
 if (!request.nextUrl.pathname.startsWith("/image")) {  
 return NextResponse.redirect(new URL("/image", request.url));  
 }  
 break;  
 default:  
 return NextResponse.redirect(new URL("/login", request.url));  
 }  
}  
  
export const config = {  
 matcher: [  
 // Match all routes except the ones that start with /login and api and the static folder  
 "/((?!api|\_next/static|\_next/image|favicon.ico|login).\*)",  
 ],  
};

In this setup, the middleware checks the user’s role stored in the JWT and redirects them based on their role.

**Breaking Down the Middleware Logic**

1. Fetching the JWT Token: Using getToken, we retrieve the JWT token which contains the user's role.
2. Redirecting Based on Role: We check the user’s role and redirect them to different paths based on their role (e.g., DOCTOR, NURSE).

**Conclusion**

Implementing RBAC in a Next.js app using NextAuth, Prisma, and MongoDB is a robust way to handle user permissions. This setup provides a secure and flexible way to manage user access based on their roles.

Remember, while this setup provides a basic structure, you should tailor the logic to fit your specific application requirements.