



NEXTAUTH

Adrian Rudnicki

39662

WSTĘP I PRZYGOTOWANIE PROJEKTU

Poznamy *NextAuth* oraz dlaczego jest takie fajne. Przygotujemy przykładowy projekt, który używa bibliotekę *NextAuth* do autentykacji.

PRZYGOTOWANIE NEXTAUTH

Zobaczymy prostą implementację logowania przy pomocy konta Google. Poznamy również strukturę projektu.

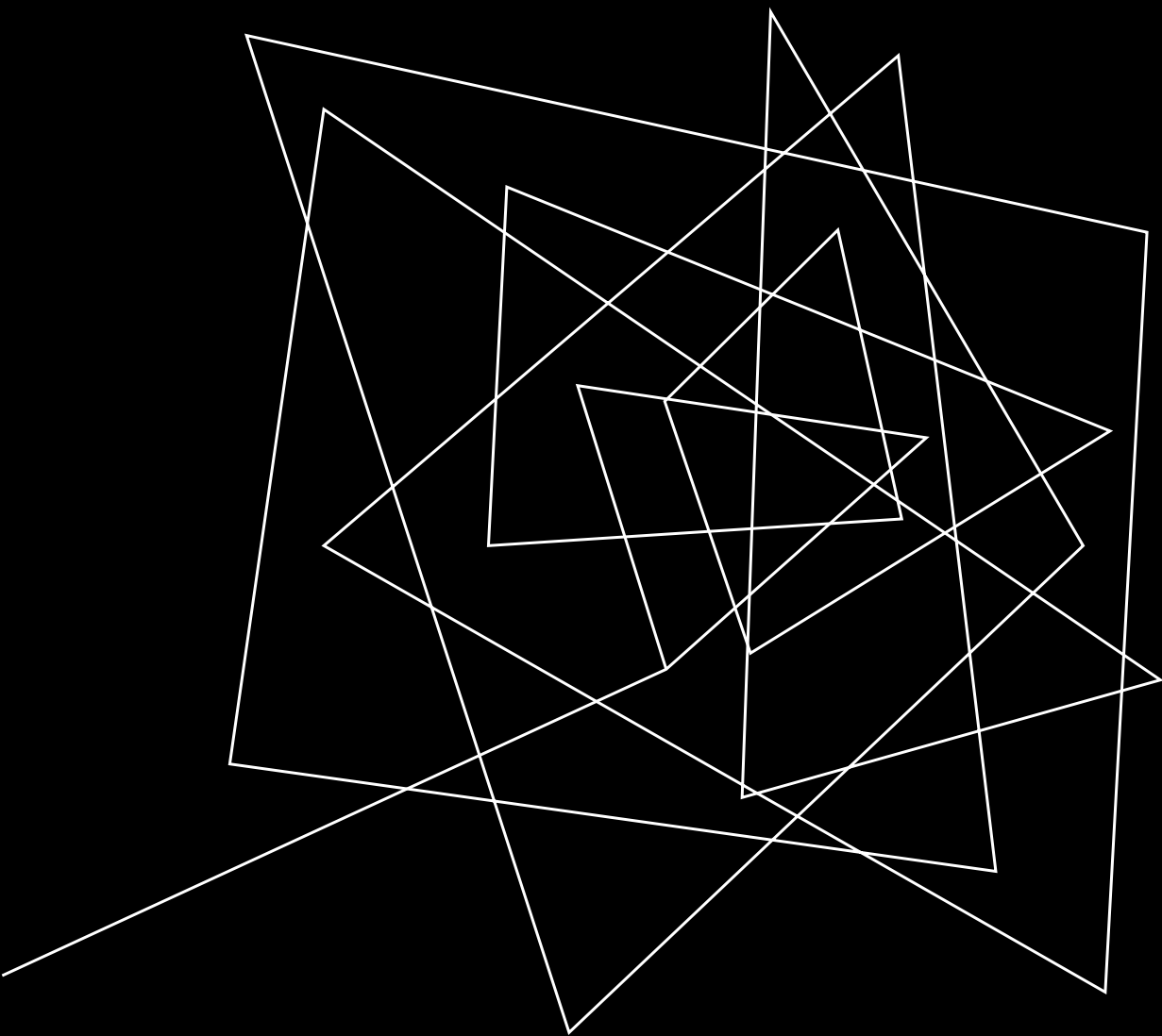
SESJA

Użyjemy sesji dostarczonej przez *NextAuth* do wyświetlenia danych o użytkowniku.

ZABEZPIECZONE DROGI

Zabezpieczymy dostęp do wrażliwych danych naszej aplikacji.

PLAN PREZENTACJI



WSTĘP I PRZYGOTOWANIE PROJEKTU



Czym jest *NextAuth*?

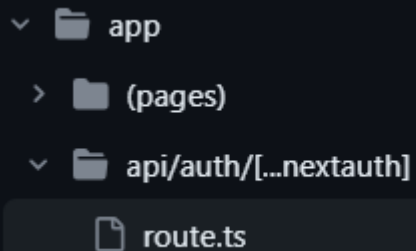
To opensourceowe narzędzie do uwierzytelniania, stworzone specjalnie dla aplikacji opartych na frameworku *Next.js*. Ułatwia implementację różnych strategii uwierzytelniania, takich jak uwierzytelnianie przez konta społecznościowe (Facebook, Google, GitHub itp.)



```
npx create-next-app@latest  
npm install next-auth
```

Od czego zacząć?

Od utworzenia nowego projektu, a następnie zainstalowania biblioteki *NextAuth*. Możemy wykorzystać swój ulubiony *Package Manager* (*npm*, *pnpm*, *yarn*).



```
✓ app  
  > (pages)  
  ✓ api/auth/[...nextauth]  
    route.ts
```

Następnie musimy utworzyć strukturę folderów w odpowiedni sposób i wrzucić do ostatniego z nich odpowiedni plik.

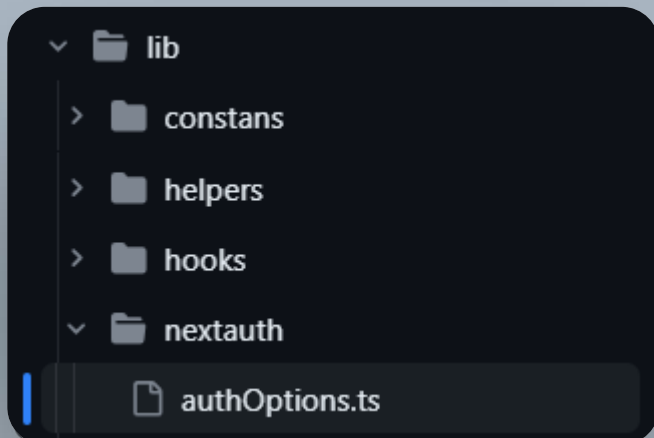
```
import NextAuth from "next-auth/next";
import { authOptions } from "@lib/nextauth/authOptions";

const handler = NextAuth(authOptions);

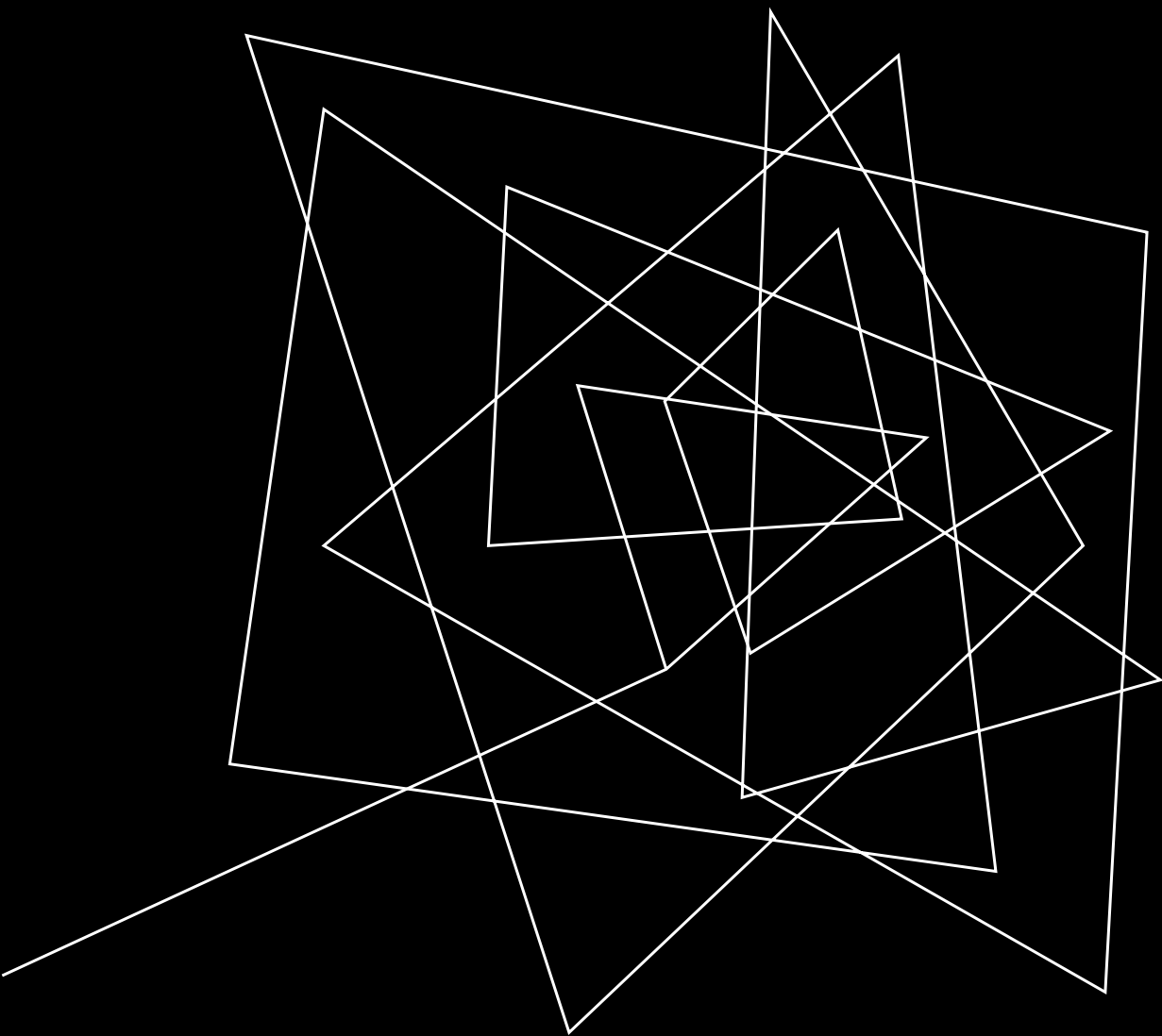
export { handler as GET, handler as POST };
```

Od czego zacząć?

W pliku *route.ts* wrzucamy odpowiedni kod, który pozwoli nam na obsługę zapytania GET oraz POST, niezbędnego do prawidłowego działania biblioteki.



Jak widać niezbędne są importy. *NextAuth* dostarczony jest przez bibliotekę, ale *authOptions* musimy zrobić sobie sami. W przypadku tego importu nie istotne jest skąd on pochodzi.



PRZYGOTOWANIE
NEXTAUTH

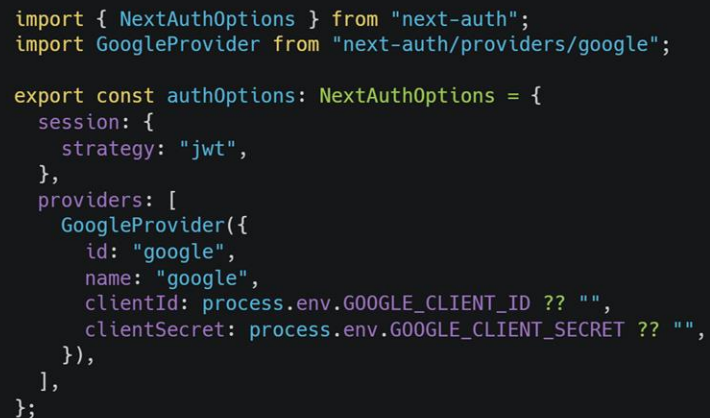


```
42 School, Amazon Cognito, Apple, Atlassian, Auth0,  
Authentik, Azure Active Directory, Azure Active Directory  
B2C, Battle.net, Box, BoxyHQ SAML, Bungie, Coinbase,  
Discord, Dropbox, DuendeIdentityServer6, EVE Online,  
Facebook, FACEIT, Foursquare, Freshbooks, FusionAuth,  
GitHub, GitLab, Google, HubSpot, IdentityServer4,  
Instagram, Kakao, Keycloak, LINE, LinkedIn, Mail.ru,  
Mailchimp, Medium, Naver, Netlify, Okta, OneLogin, Osso,  
osu!, Patreon, Pinterest, Pipedrive, Reddit, Salesforce,  
Slack, Spotify, Strava, Todoist, Trakt, Twitch, Twitter,  
United Effects, VK, Wikimedia, WordPress.com, WorkOS,  
Yandex, Zitadel, Zoho, Zoom,
```

Projekt mamy gotowy teraz najlepsze.

Po utworzeniu projektu można zacząć konfigurację tego co potrzebujemy. *NextAuth* dostarcza wiele *Providerów*, dzięki którym użytkownik może się logować.

Lista jest spora, ale *NextAuth* dostarcza odpowiednią dokumentację do każdego *Providera*. Najpopularniejsze to Google, Facebook, Github.



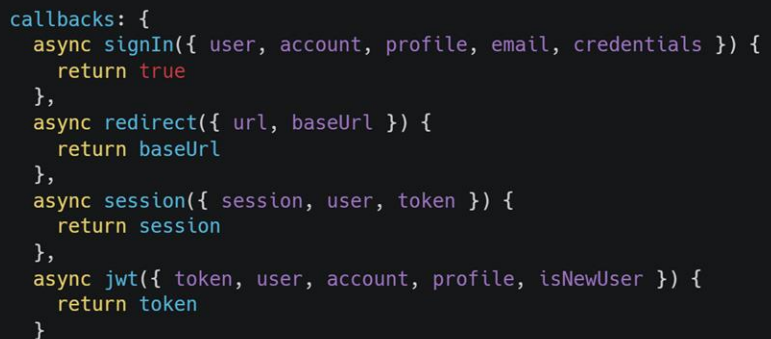
```
import { NextAuthOptions } from "next-auth";
import GoogleProvider from "next-auth/providers/google";

export const authOptions: NextAuthOptions = {
  session: {
    strategy: "jwt",
  },
  providers: [
    GoogleProvider({
      id: "google",
      name: "google",
      clientId: process.env.GOOGLE_CLIENT_ID ?? "",
      clientSecret: process.env.GOOGLE_CLIENT_SECRET ?? "",
    }),
  ],
};
```

Projekt mamy gotowy teraz najlepsze.

Jeśli już się zdecydujemy na odpowiednie dla nas formy logowania - implementujemy wybrane opcje. Wybieramy strategię, którą obraliśmy (do wyboru mamy ich kilka, a popularną opcją jest *jwt*).

Następnie rejestrujemy *Providerów* i odpowiednią dla nich konfigurację (zazwyczaj jakieś *id* lub *secret*)



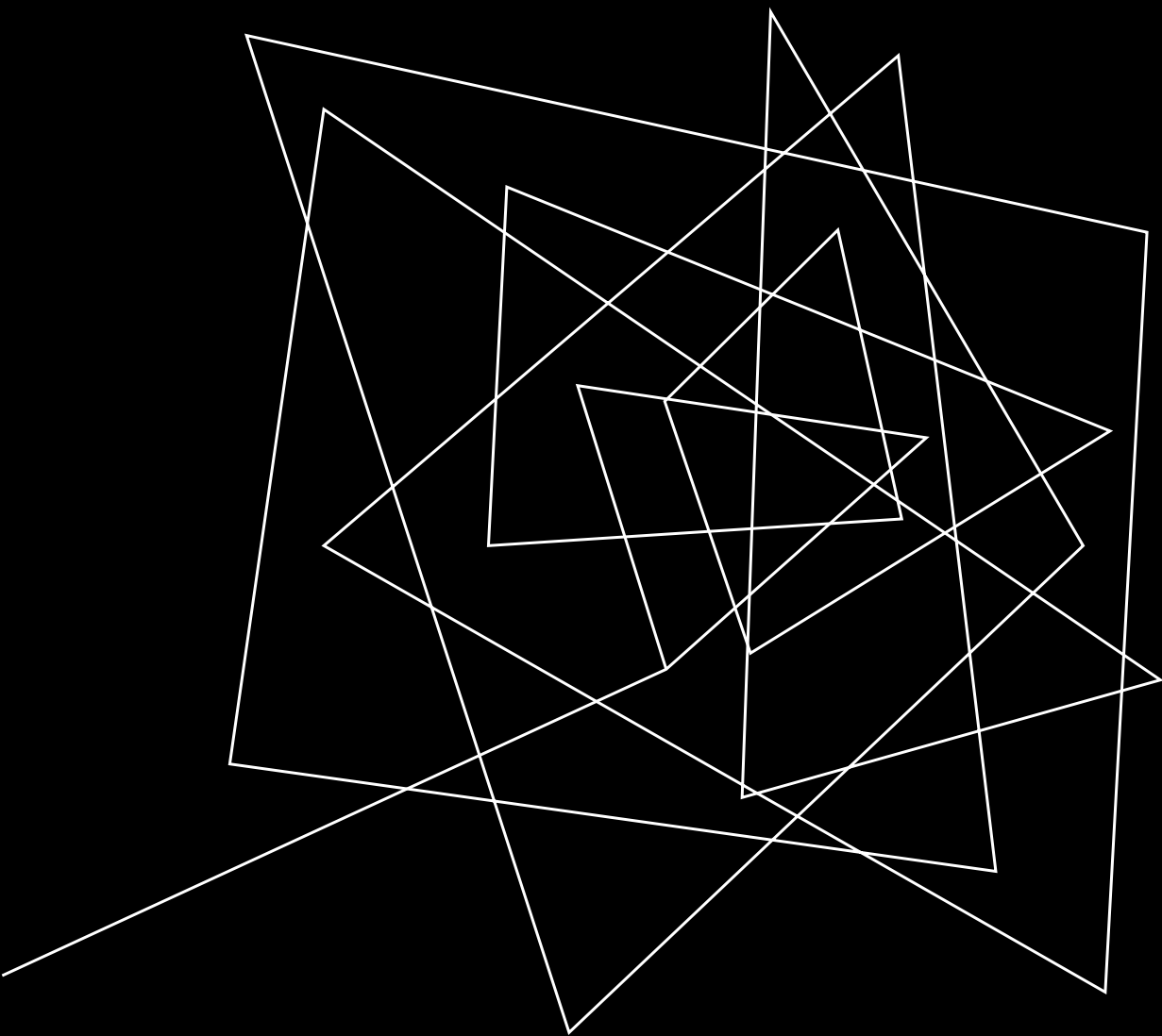
```
callbacks: {  
  async signIn({ user, account, profile, email, credentials }) {  
    return true  
  },  
  async redirect({ url, baseUrl }) {  
    return baseUrl  
  },  
  async session({ session, user, token }) {  
    return session  
  },  
  async jwt({ token, user, account, profile, isNewUser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

No i to wszystko!

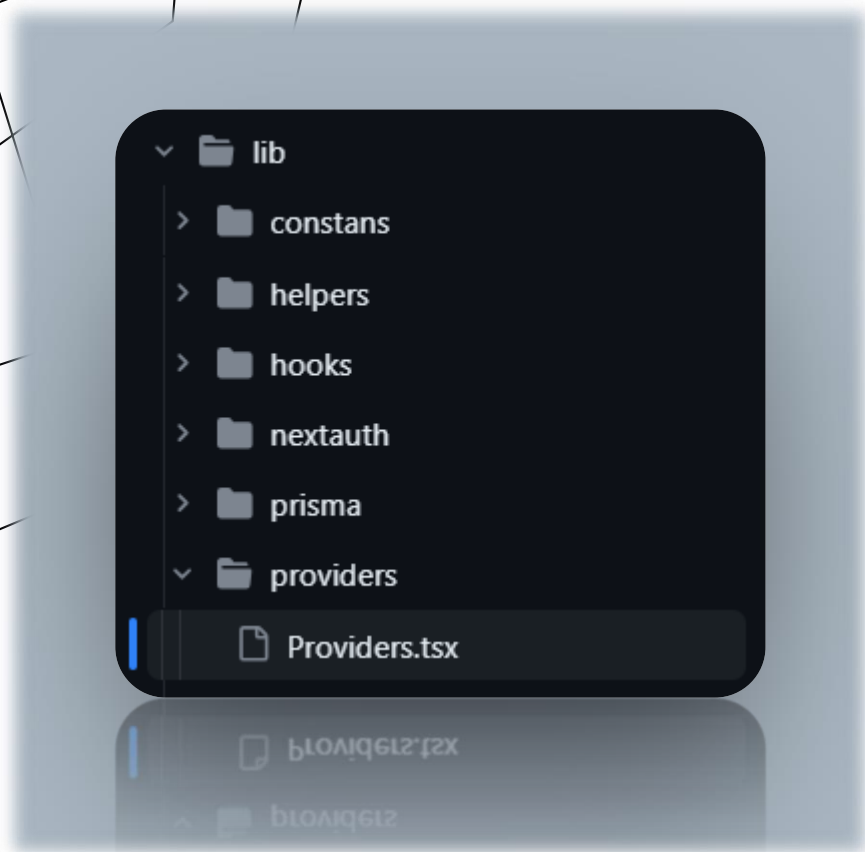
W tym momencie nasza aplikacja jest gotowa do przyjmowania użytkowników. Cała implementacja autentykacji jest już gotowa.

Oczywiście warto sprawdzić gdzieś po drodze czy dany użytkownik istnieje już w naszej bazie danych.

Można to zrobić w *callback'ach*, które dostarcza *NextAuth*.



SESJA



Sesja. Bo już się zalogowaliśmy.

Na początku musimy podzielić naszą aplikację na część, która się dzieje na serwerze oraz na część, która śmiga w przeglądarce.

Jednak przede wszystkim musimy utworzyć sobie *SessionProvidera*. Najlepiej w pliku *Providers.tsx*. Tak jest to komponent.

```

"use client";

import { SessionProvider } from "next-auth/react";

export default function Providers({ children }: { children:
React.ReactNode }) {
  return (
    <SessionProvider>
      {children}
    </SessionProvider>
  );
}

```

```

import Providers from "@lib/providers/Providers";

export default function RootLayout({
  children,
}): {
  children: React.ReactNode;
}) {
  return (
    <html lang="pl-PL">
      <body>
        <Providers>{children}</Providers>
      </body>
    </html>
  );
}

```

Sesja. Bo już się zalogowaliśmy.

„*use client*” jest tutaj najważniejsze. Ponieważ chcemy mieć dostęp do sesji z poziomu komponentów klienta (tych, które są renderowane w przeglądarce).

Oczywiście musimy zaimportować *Providera* w rootowym elemencie aplikacji. Wtedy cała aplikacja będzie miała dostęp do sesji.

```
"use client";

import { signIn, signOut, useSession } from "next-auth/react";

function ClientComponent() {
  const { data: session } = useSession();

  if (session) {
    return (
      <>
        {session?.user?.name} <br />
        <button onClick={() => signOut()}>Sign out</button>
      </>
    );
  }
  return (
    <>
      Not signed in <br />
      <button onClick={() => signIn()}>Sign in</button>
    </>
  );
}

export default ClientComponent;
```

Sesja w *client component*.

Użycie sesji jest bardzo proste. *NextAuth* dostarcza odpowiedni *hook* – *useSession*, dzięki któremu mamy dostęp do danych użytkownika.

Możemy również sprawdzić czy sesja istnieje i wyświetlić odpowiednie *UI* dla akcji użytkownika.

W tym przykładzie pozwalamy zalogować się lub wylogować się w zależności od stanu sesji, dzięki czemu poznajemy kolejne funkcję *signIn* oraz *signOut*.



```
import { getSession } from "next-auth";

export default async function Home() {
  const session = await getSession();

  return (
    <div>
      {session?.user?.name ? (
        <div>{session?.user?.name}</div>
      ) : (
        <div>Not logged in</div>
      )}
    </div>
  );
}
```

Sesja w *server component*.

Podobnie jak w *client component*, możemy uzyskać dostęp do sesji w komponentach serwerowych. Jednak w takich komponentach nie możemy używać *hooków*. Rozwiązanie jest inne, również dostarczone przez *NextAuth*.

getSession to funkcja, która umożliwia nam dostęp do sesji podobnie jak *useSession*.



```
import { getServerSession } from "next-auth";
import { NextResponse } from "next/server";

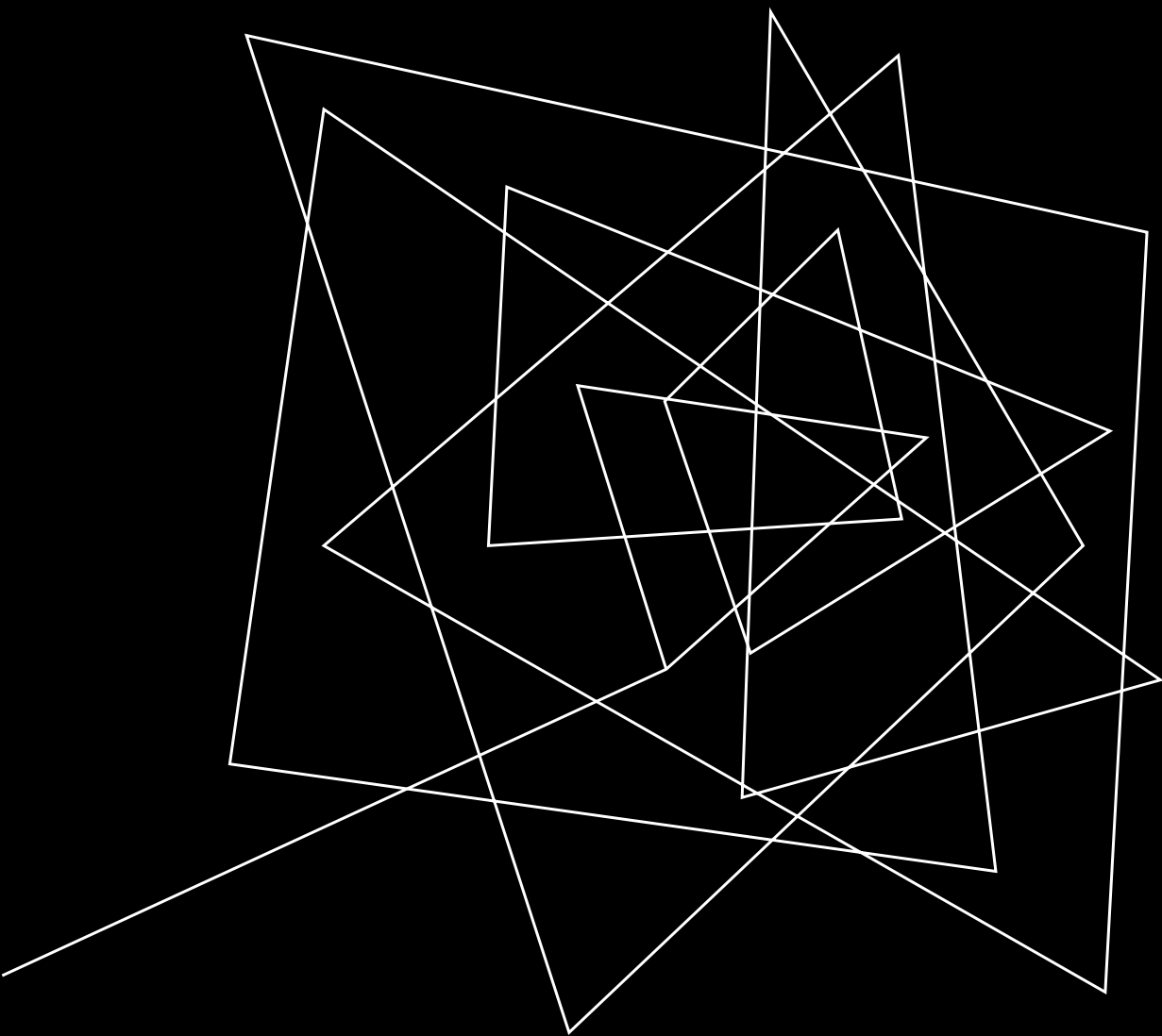
import { authOptions } from "@/lib/nextauth/authOptions";

export async function GET() {
  const session = await getServerSession(authOptions);

  return NextResponse.json(
    { name: session?.user?.name ?? "Not Logged In" }
  );
}
```

Sesja w *api route*.

NextJs umożliwia tworzenie *api route*. *NextAuth* również tutaj pomaga dostać się do sesji, aby pobrać dane o użytkowniku. Jest jednak mała różnica. Aby wszystko poprawnie działało należy przekazać jako argument funkcji *getServerSession* *authOptions*, które tworzyliśmy na samym początku.



ZABEZPIECZONE
DROGI

```

import { redirect } from "next/navigation";
import { getServerSession } from "next-auth";

export default async function ProtectedRoute() {
  const session = await getServerSession();
  if (!session || !session.user) {
    redirect("/api/auth/signin");
  }

  return (
    <div>
      This is a protected route.
      <br />
      You will only see this if you are authenticated.
    </div>
  );
}

```

Można się zabezpieczyć.

NextAuth pomaga zabezpieczyć drogi naszej aplikacji. Możemy dzięki temu sprawdzić czy dany użytkownik może przeglądać części aplikacji.

```

import { getServerSession } from "next-auth";
import { NextResponse } from "next/server";

import { authOptions } from "@lib/nextauth/authOptions";

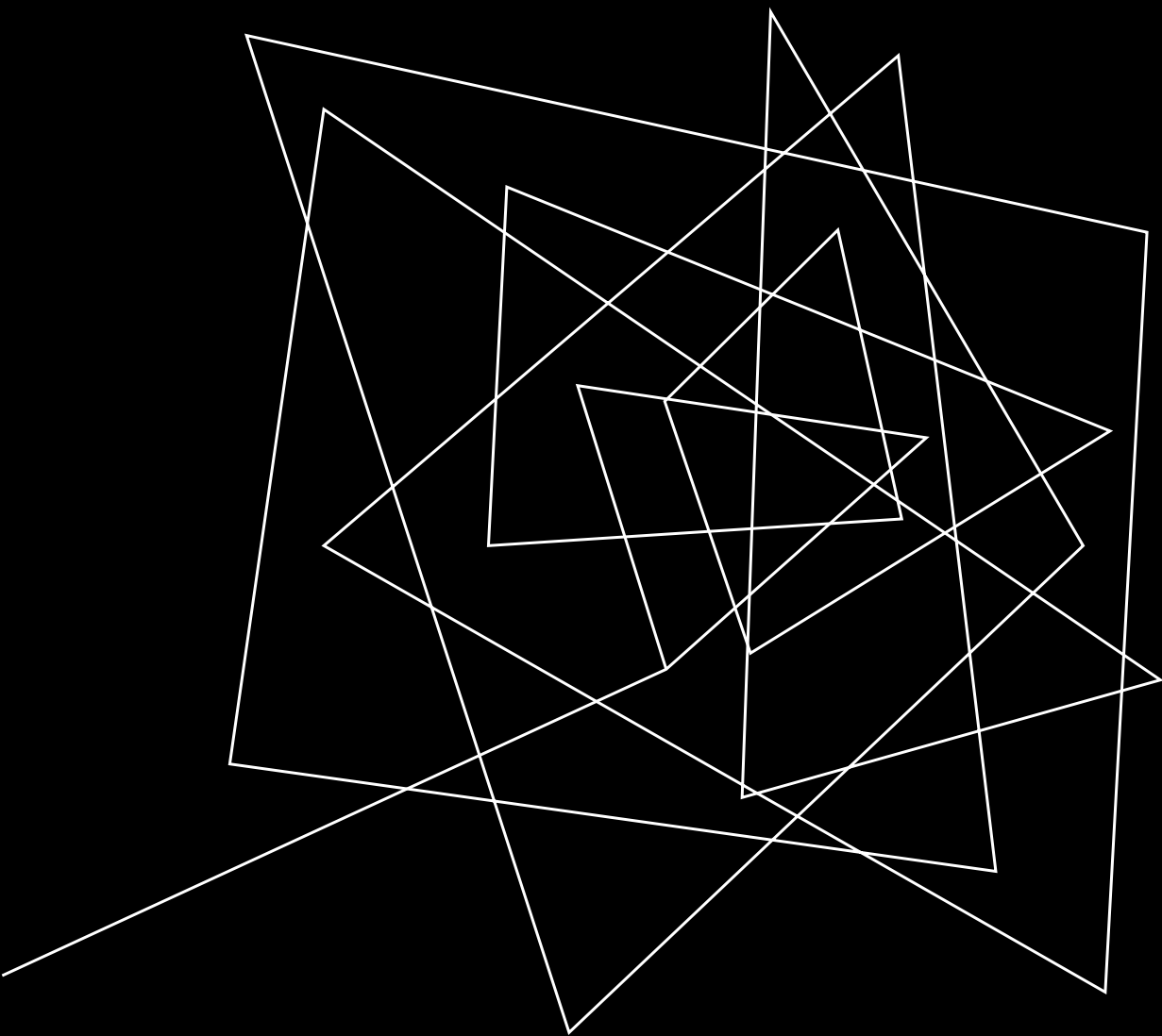
export async function GET() {
  const session = await getServerSession(authOptions);

  if(session?.user?.role !== "ADMIN")
    return NextResponse.json(
      { error: "Not Logged In As ADMIN" }
    );

  return NextResponse.json(
    { success: "Logged In As ADMIN" }
  );
}

```

Możemy również autoryzować działania przeznaczone wykonywane przez użytkownika i jeśli ten nie ma uprawnień do wykonania takiego działania – możemy taką akcję zablokować.



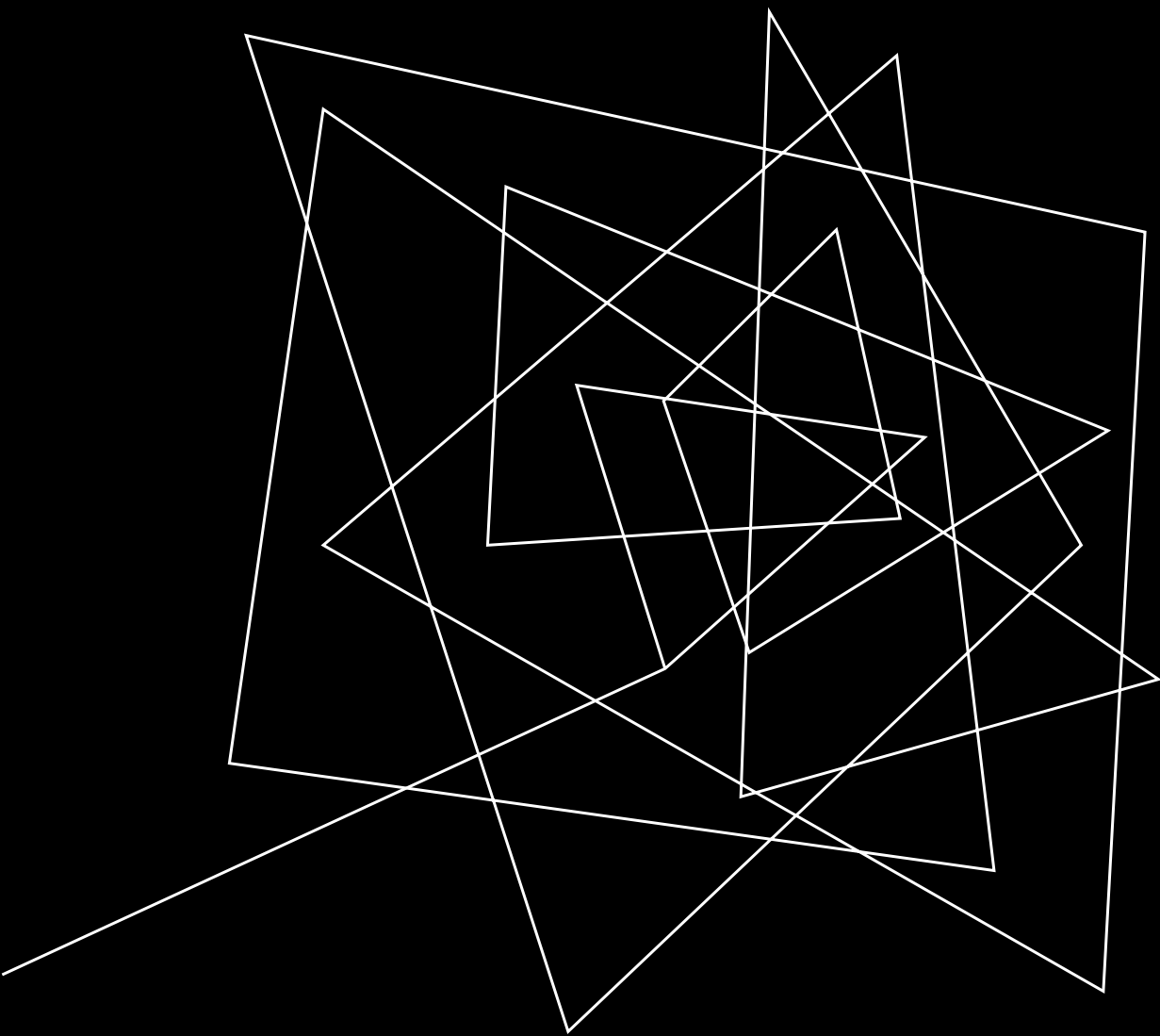
PODSUMOWANIE



Podsumowanie.

NextAuth jest fajnym narzędziem do autentykacji użytkowników. Ta prezentacja to jedynie wierzchołek góry lodowej. Jednak, aby poznać bardziej zaawansowane możliwości biblioteki odsyłam do jej dokumentacji.

Jest napisana całkiem przystępnie, a z powodu jej popularności dostępnych jest mnóstwo poradników na jej temat.



BIBLIOGRAFIA



Bibliografia.

1. <https://next-auth.js.org/>
2. <https://nextjs.org/>
3. <https://www.youtube.com/@jherr>
4. <https://www.youtube.com/@DaveGrayTeachesCode>
5. <https://github.com/parkingbezplatny/app/>