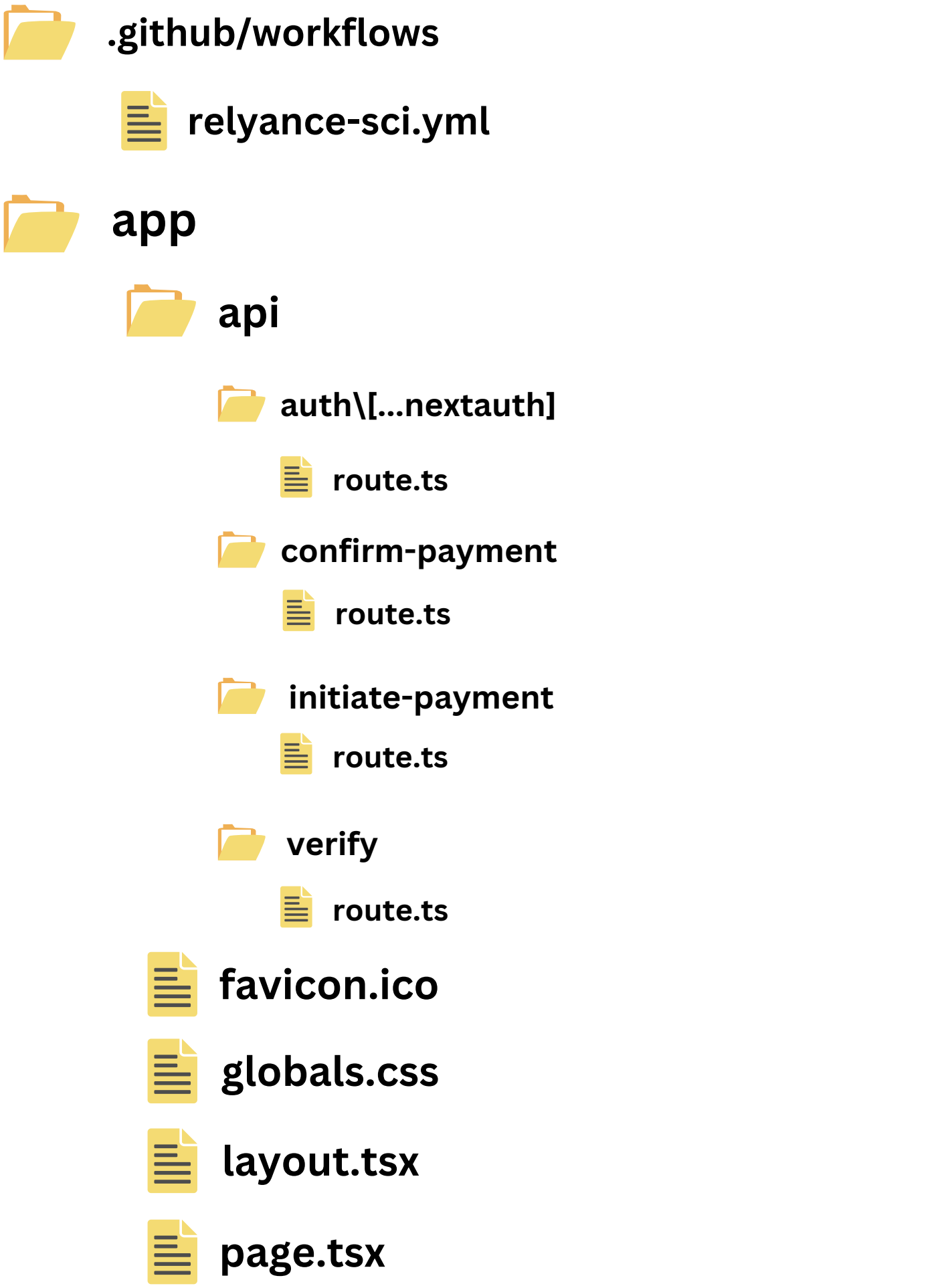
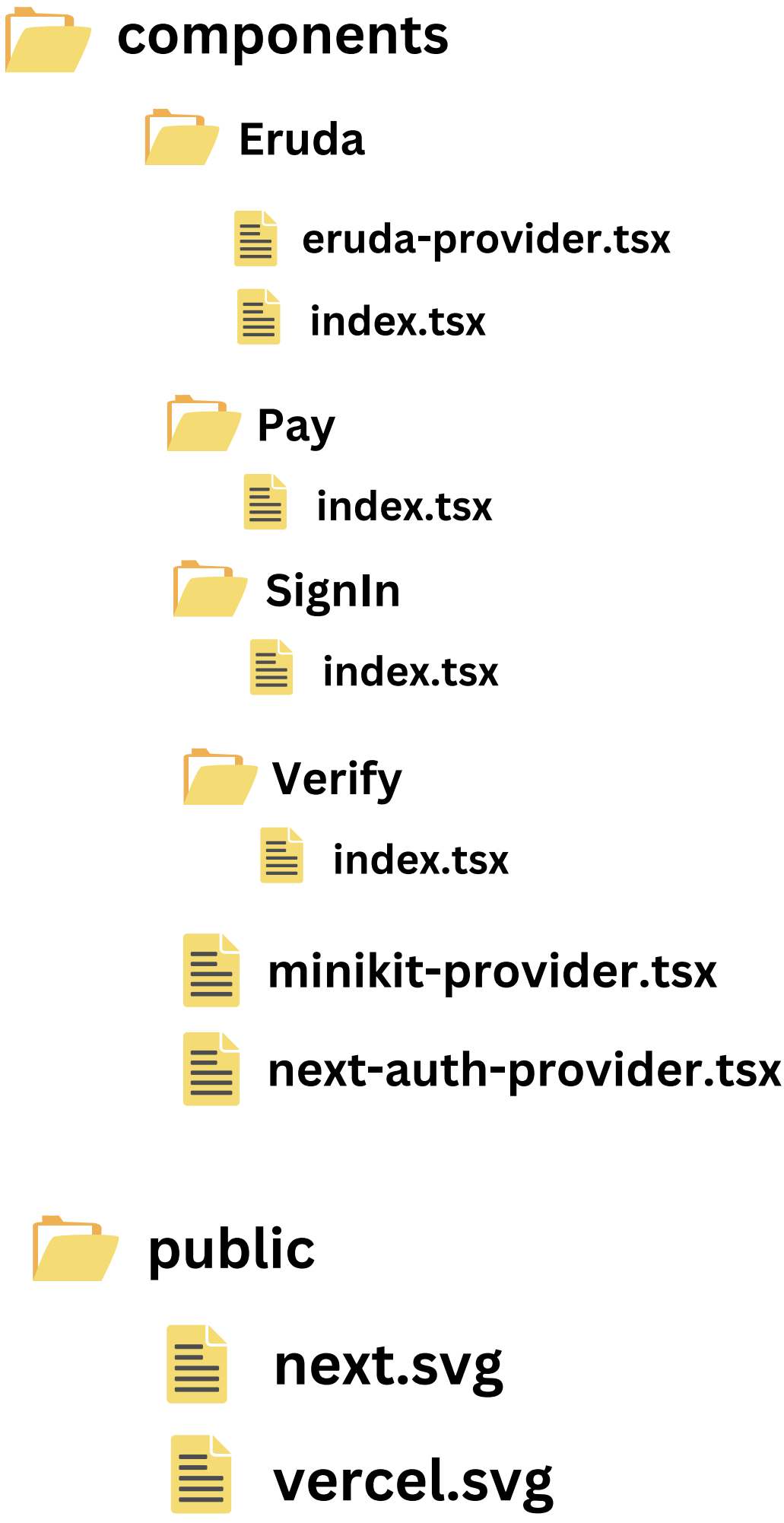

.github/workflows
relyance-sci.yml
app
api
auth\[...nextauth]
route.ts
confirm-payment
route.ts
initiate-payment
route.ts
verify
route.ts
favicon.ico
globals.css
layout.tsx
page.tsx


components
Eruda
eruda-provider.tsx
index.tsx
Pay
index.tsx
SignIn
index.tsx
Verify
index.tsx
minikit-provider.tsx
next-auth-provider.tsx
public
next.svg
vercel.svg

.env.example

.eslintrc.json

.gitignore

README.md

next.config.mjs

package.json

pnpm-lock.yaml

postcss.config.mjs

tailwind.config.ts

tsconfig.json

.github/workflows	Add .github/workflows/relyance-sci.yml	2 months ago
app	fix: vercel build error (#2)	last week
components	Init template (#1)	last month
public	Init template (#1)	last month
.env.example	Init template (#1)	last month
.eslintrc.json	Init template (#1)	last month
.gitignore	Init template (#1)	last month
README.md	Init template (#1)	last month
next.config.mjs	Init template (#1)	last month
package.json	Init template (#1)	last month
pnpm-lock.yaml	Init template (#1)	last month
postcss.config.mjs	Init template (#1)	last month
tailwind.config.ts	Init template (#1)	last month
tsconfig.json	Init template (#1)	last month

relyance-sci.yml:

```
name: Relyance SCI Scan

on:
  schedule:
    - cron: "0 20 * * *"
  workflow_dispatch:

jobs:
  execute-relyance-sci:
    name: Relyance SCI Job
    runs-on: ubuntu-latest

    steps:
      - name: Checkout
        uses: actions/checkout@v4

      - name: Pull and run SCI binary
        run: |-
          docker pull gcr.io/relyance-ext/compliance_inspector:release && \
          docker run --rm -v `pwd`:/repo --env API_KEY='${{ secrets.DPP_SCI_KEY }}'
gcr.io/relyance-ext/compliance_inspector:release
```

auth\[...nextauth]

route.ts

```
import NextAuth, { NextAuthOptions } from "next-auth";

const authOptions: NextAuthOptions = {
  secret: process.env.NEXTAUTH_SECRET,

  providers: [
    {
      id: "worldcoin",
      name: "Worldcoin",
      type: "oauth",
      wellKnown: "https://id.worldcoin.org/.well-known/openid-
configuration",
      authorization: { params: { scope: "openid" } },
      clientId: process.env.WLD_CLIENT_ID,
      clientSecret: process.env.WLD_CLIENT_SECRET,
      idToken: true,
      checks: ["state", "nonce", "pkce"],
      profile(profile) {
        return {
          id: profile.sub,
          name: profile.sub,
          verificationLevel:
            profile["https://id.worldcoin.org/v1"].verification_level,
        };
      },
    },
  ],
  callbacks: {
    async signIn({ user }) {
      return true;
    },
  },
  debug: process.env.NODE_ENV === "development",
};

const handler = NextAuth(authOptions);
export { handler as GET, handler as POST };
```

confirm-payment

route.ts

```
import { MiniAppPaymentSuccessPayload } from "@worldcoin/minikit-js";
import { cookies } from "next/headers";
import { NextRequest, NextResponse } from "next/server";

interface IRequestPayload {
  payload: MiniAppPaymentSuccessPayload;
}

export async function POST(req: NextRequest) {
  const { payload } = (await req.json()) as IRequestPayload;

  // IMPORTANT: Here we should fetch the reference you created in /initiate-payment to ensure the
transaction we are verifying is the same one we initiated
  //   const reference = getReferenceFromDB();
  const cookieStore = cookies();

  const reference = cookieStore.get("payment-nonce")?.value;

  console.log(reference);

  if (!reference) {
    return NextResponse.json({ success: false });
  }
  console.log(payload);
  // 1. Check that the transaction we received from the mini app is the same one we sent
  if (payload.reference === reference) {
    const response = await fetch(
      `https://developer.worldcoin.org/api/v2/minikit/transaction/${payload.transaction_id}?
app_id=${process.env.APP_ID}`,
      {
        method: "GET",
        headers: {
          Authorization: `Bearer ${process.env.DEV_PORTAL_API_KEY}`,
        },
      }
    );
    const transaction = await response.json();
    // 2. Here we optimistically confirm the transaction.
    // Otherwise, you can poll until the status == mined
    if (transaction.reference == reference && transaction.status != "failed") {
      return NextResponse.json({ success: true });
    } else {
      return NextResponse.json({ success: false });
    }
  }
}
```

initiate-payment

route.ts

```
import { cookies } from "next/headers";
import { NextRequest, NextResponse } from "next/server";

export async function POST(req: NextRequest) {
  const uuid = crypto.randomUUID().replace(/-/g, "");

  // TODO: Store the ID field in your database so you can verify the payment later
  cookies().set({
    name: "payment-nonce",
    value: uuid,
    httpOnly: true,
  });

  console.log(uuid);

  return NextResponse.json({ id: uuid });
}
```

verify

route.ts

```
import {
  verifyCloudProof,
  IVerifyResponse,
  ISuccessResult,
} from "@worldcoin/minikit-js";
import { NextRequest, NextResponse } from "next/server";

interface IRequestPayload {
  payload: ISuccessResult;
  action: string;
  signal: string | undefined;
}

export async function POST(req: NextRequest) {
  const { payload, action, signal } = (await req.json()) as IRequestPayload;
  const app_id = process.env.APP_ID as `app_${string}`;
  const verifyRes = (await verifyCloudProof(
    payload,
    app_id,
    action,
    signal
  )) as IVerifyResponse; // Wrapper on this

  console.log(verifyRes);

  if (verifyRes.success) {
    // This is where you should perform backend actions if the verification succeeds
    // Such as, setting a user as "verified" in a database
    return NextResponse.json({ verifyRes, status: 200 });
  } else {
    // This is where you should handle errors from the World ID /verify endpoint.
    // Usually these errors are due to a user having already verified.
    return NextResponse.json({ verifyRes, status: 400 });
  }
}
```

globals.css

```
@tailwind base;
@tailwind components;
@tailwind utilities;

:root {
  --foreground-rgb: 0, 0, 0;
  --background-start-rgb: 214, 219, 220;
  --background-end-rgb: 255, 255, 255;
}

@media (prefers-color-scheme: dark) {
  :root {
    --foreground-rgb: 255, 255, 255;
    --background-start-rgb: 0, 0, 0;
    --background-end-rgb: 0, 0, 0;
  }
}

body {
  color: rgb(var(--foreground-rgb));
  background: linear-gradient(
      to bottom,
      transparent,
      rgb(var(--background-end-rgb))
    )
    rgb(var(--background-start-rgb));
}

@layer utilities {
  .text-balance {
    text-wrap: balance;
  }
}
```

layout.tsx

```
import type { Metadata } from "next";
import { Inter } from "next/font/google";
import "./globals.css";
import MiniKitProvider from "@/components/minikit-provider";
import dynamic from "next/dynamic";
import NextAuthProvider from "@/components/next-auth-provider";

const inter = Inter({ subsets: ["latin"] });

export const metadata: Metadata = {
  title: "Create Next App",
  description: "Generated by create next app",
};

export default function RootLayout({
  children,
}: Readonly<{
  children: React.ReactNode;
}>) {
  const ErudaProvider = dynamic(
    () => import("../components/Eruda").then((c) => c.ErudaProvider),
    {
      ssr: false,
    }
  );
  return (
    <html lang="en">
      <NextAuthProvider>
        <ErudaProvider>
          <MiniKitProvider>
            <body className={inter.className}>{children}</body>
          </MiniKitProvider>
        </ErudaProvider>
      </NextAuthProvider>
    </html>
  );
}
```

page.tsx

```
import { PayBlock } from "@/components/Pay";
import { SignIn } from "@/components/SignIn";
import { VerifyBlock } from "@/components/Verify";

export default function Home() {
  return (
    <main className="flex min-h-screen flex-col items-center justify-center p-24 gap-y-3">
      <SignIn />
      <VerifyBlock />
      <PayBlock />
    </main>
  );
}
```

eruda-provider.tsx

```
"use client";

import eruda from "eruda";
import { ReactNode, useEffect } from "react";

export const Eruda = (props: { children: ReactNode }) => {
  useEffect(() => {
    if (typeof window !== "undefined") {
      try {
        eruda.init();
      } catch (error) {
        console.log("Eruda failed to initialize", error);
      }
    }
  }, []);

  return <>{props.children}</>;
};
```

Eruda

eruda-provider.tsx

index.tsx

```
"use client";

import dynamic from "next/dynamic";
import { ReactNode } from "react";

const Eruda = dynamic(() => import("./eruda-provider").then((c) => c.Eruda), {
  ssr: false,
});

export const ErudaProvider = (props: { children: ReactNode }) => {
  if (process.env.NEXT_PUBLIC_APP_ENV === "production") {
    return props.children;
  }
  return <Eruda>{props.children}</Eruda>;
};
```

Pay

index.tsx

```
"use client";
import {
  MiniKit,
  tokenToDecimals,
  Tokens,
  PayCommandInput,
} from "@worldcoin/minikit-js";

const sendPayment = async () => {
  try {
    const res = await fetch(`/api/initiate-payment`, {
      method: "POST",
    });

    const { id } = await res.json();

    console.log(id);

    const payload: PayCommandInput = {
      reference: id,
      to: "0x0c892815f0B058E69987920A23FBb33c834289cf", // Test address
      tokens: [
        {
          symbol: Tokens.WLD,
          token_amount: tokenToDecimals(0.5, Tokens.WLD).toString(),
        },
        {
          symbol: Tokens.USDCE,
          token_amount: tokenToDecimals(0.1, Tokens.USDCE).toString(),
        },
      ],
      description: "Watch this is a test",
    };
    if (MiniKit.isInstalled()) {
      return await MiniKit.commandsAsync.pay(payload);
    }
    return null;
  } catch (error: unknown) {
    console.log("Error sending payment", error);
    return null;
  }
};

const handlePay = async () => {
  if (!MiniKit.isInstalled()) {
    console.error("MiniKit is not installed");
    return;
  }
  const sendPaymentResponse = await sendPayment();
  const response = sendPaymentResponse?.finalPayload;
  if (!response) {
    return;
  }

  if (response.status == "success") {
    const res = await fetch(`${process.env.NEXTAUTH_URL}/api/confirm-payment`, {
      method: "POST",
      headers: { "Content-Type": "application/json" },
      body: JSON.stringify({ payload: response }),
    });
    const payment = await res.json();
    if (payment.success) {
      // Congrats your payment was successful!
      console.log("SUCCESS!");
    } else {
      // Payment failed
      console.log("FAILED!");
    }
  }
};

export const PayBlock = () => {
  return (
    <button className="bg-blue-500 p-4" onClick={handlePay}>
      Pay
    </button>
  );
};
```

SignIn

index.tsx

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

export const SignIn = () => {
  const { data: session } = useSession();
  if (session) {
    return (
      <>
        Signed in as {session?.user?.name?.slice(0, 10)} <br />
        <button onClick={() => signOut()}>Sign out</button>
      </>
    );
  } else {
    return (
      <>
        Not signed in <br />
        <button onClick={() => signIn()}>Sign in</button>
      </>
    );
  }
};
```

Verify

index.tsx

```
"use client";
import {
  MiniKit,
  VerificationLevel,
  ISuccessResult,
  MiniAppVerifyActionErrorPayload,
  IVerifyResponse,
} from "@worldcoin/minikit-js";
import { useCallback, useState } from "react";

export type VerifyCommandInput = {
  action: string;
  signal?: string;
  verification_level?: VerificationLevel; // Default: Orb
};

const verifyPayload: VerifyCommandInput = {
  action: "test-action", // This is your action ID from the Developer Portal
  signal: "",
  verification_level: VerificationLevel.Orb, // Orb | Device
};

export const VerifyBlock = () => {
  const [handleVerifyResponse, setHandleVerifyResponse] = useState<
    MiniAppVerifyActionErrorPayload | IVerifyResponse | null
  >(null);

  const handleVerify = useCallback(async () => {
    if (!MiniKit.isInstalled()) {
      console.warn("Tried to invoke 'verify', but MiniKit is not installed.");
      return null;
    }

    const { finalPayload } = await MiniKit.commandsAsync.verify(verifyPayload);

    // no need to verify if command errored
    if (finalPayload.status === "error") {
      console.log("Command error");
      console.log(finalPayload);

      setHandleVerifyResponse(finalPayload);
      return finalPayload;
    }

    // Verify the proof in the backend
    const verifyResponse = await fetch(`/api/verify`, {
      method: "POST",
      headers: {
        "Content-Type": "application/json",
      },
      body: JSON.stringify({
        payload: finalPayload as ISuccessResult, // Parses only the fields we need to verify
        action: verifyPayload.action,
        signal: verifyPayload.signal, // Optional
      }),
    });

    // TODO: Handle Success!
    const verifyResponseJson = await verifyResponse.json();

    if (verifyResponseJson.status === 200) {
      console.log("Verification success!");
      console.log(finalPayload);
    }

    setHandleVerifyResponse(verifyResponseJson);
    return verifyResponseJson;
  }, []);

  return (
    <div>
      <h1>Verify Block</h1>
      <button className="bg-green-500 p-4" onClick={handleVerify}>
        Test Verify
      </button>
      <span>{JSON.stringify(handleVerifyResponse, null, 2)}</span>
    </div>
  );
};
```

minikit-provider.tsx

```
"use client"; // Required for Next.js

import { MiniKit } from "@worldcoin/minikit-js";
import { ReactNode, useEffect } from "react";

export default function MiniKitProvider({ children }: { children: ReactNode }) {
  useEffect(() => {
    MiniKit.install();
    console.log(MiniKit.isInstalled());
  }, []);

  return <>{children}</>;
}
```

next-auth-provider.tsx

```
"use client";

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

export default function NextAuthProvider({
  children,
}: {
  children: ReactNode;
}) {
  return <SessionProvider>{children}</SessionProvider>;
}
```