"use server";

import { createAdminClient, createSessionClient } from "@/lib/appwrite";

import { appwriteConfig } from "@/lib/appwrite/config";

import { Query, ID } from "node-appwrite";

import { parseStringify } from "@/lib/utils";

import { cookies } from "next/headers";

//import { avatarPlaceholderUrl } from "@/constants";

import { redirect } from "next/navigation";

const getUserByEmail = async (email: string) => {

  const { databases } = await createAdminClient();

  const result = await databases.listDocuments(

    appwriteConfig.databaseId,

    appwriteConfig.usersCollectionId,

    [Query.equal("email", [email])],

  );

  return result.total > 0 ? result.documents[0] : null;

};

const handleError = (error: unknown, message: string) => {

  console.log(error, message);

  throw error;

};

export const sendEmailOTP = async ({ email }: { email: string }) => {

  const { account } = await createAdminClient();

  try {

    const session = await account.createEmailToken(ID.unique(), email);

    return session.userId;

  } catch (error) {

    handleError(error, "Failed to send email OTP");

  }

};

export const createAccount = async ({

  fullName,

  email,

}: {

  fullName: string;

  email: string;

}) => {

  const existingUser = await getUserByEmail(email);

  const accountId = await sendEmailOTP({ email });

  if (!accountId) throw new Error("Failed to send an OTP");

  if (!existingUser) {

    const { databases } = await createAdminClient();

    await databases.createDocument(

      appwriteConfig.databaseId,

      appwriteConfig.usersCollectionId,

      ID.unique(),

      {

        fullName,

        email,

        avatar: avatarPlaceholderUrl,

        accountId,

      },

    );

  }

  return parseStringify({ accountId });

};

export const verifySecret = async ({

  accountId,

  password,

}: {

  accountId: string;

  password: string;

}) => {

  try {

    const { account } = await createAdminClient();

    const session = await account.createSession(accountId, password);

    (await cookies()).set("appwrite-session", session.secret, {

      path: "/",

      httpOnly: true,

      sameSite: "strict",

      secure: true,

    });

    return parseStringify({ sessionId: session.$id });

  } catch (error) {

    handleError(error, "Failed to verify OTP");

  }

};

export const getCurrentUser = async () => {

  try {

    const { databases, account } = await createSessionClient();

    const result = await account.get();

    const user = await databases.listDocuments(

      appwriteConfig.databaseId,

      appwriteConfig.usersCollectionId,

      [Query.equal("accountId", result.$id)],

    );

    if (user.total <= 0) return null;

    return parseStringify(user.documents[0]);

  } catch (error) {

    console.log(error);

  }

};

export const signOutUser = async () => {

  const { account } = await createSessionClient();

  try {

    await account.deleteSession("current");

    (await cookies()).delete("appwrite-session");

  } catch (error) {

    handleError(error, "Failed to sign out user");

  } finally {

    redirect("/sign-in");

  }

};

export const signInUser = async ({ email }: { email: string }) => {

  try {

    const existingUser = await getUserByEmail(email);

    // User exists, send OTP

    if (existingUser) {

      await sendEmailOTP({ email });

      return parseStringify({ accountId: existingUser.accountId });

    }

    return parseStringify({ accountId: null, error: "User not found" });

  } catch (error) {

    handleError(error, "Failed to sign in user");

  }

};

"use client";

import { z } from "zod";

import { zodResolver } from "@hookform/resolvers/zod";

import { useForm } from "react-hook-form";

import { Button } from "@/components/ui/button";

import {

  Form,

  FormControl,

  FormField,

  FormItem,

  FormLabel,

  FormMessage,

} from "@/components/ui/form";

import { Input } from "@/components/ui/input";

import { useState } from "react";

import Image from "next/image";

import Link from "next/link";

import { createAccount, signInUser } from "@/lib/actions/user.actions";

import OtpModal from "@/components/OTPModal";

type FormType = "sign-in" | "sign-up";

const authFormSchema = (formType: FormType) => {

  return z.object({

    email: z.string().email(),

    fullName:

      formType === "sign-up"

        ? z.string().min(4).max(50)

        : z.string().optional(),

  });

};

const AuthForm = ({ type }: { type: FormType }) => {

  const [isLoading, setIsLoading] = useState(false);

  const [errorMessage, setErrorMessage] = useState("");

  const [accountId, setAccountId] = useState(null);

  const formSchema = authFormSchema(type);

  const form = useForm<z.infer<typeof formSchema>>({

    resolver: zodResolver(formSchema),

    defaultValues: {

      fullName: "",

      email: "",

    },

  });

  const onSubmit = async (values: z.infer<typeof formSchema>) => {

    setIsLoading(true);

    setErrorMessage("");

    try {

      const user =

        type === "sign-up"

          ? await createAccount({

              fullName: values.fullName || "",

              email: values.email,

            })

          : await signInUser({ email: values.email });

      setAccountId(user.accountId);

    } catch {

      setErrorMessage("Failed to create account. Please try again.");

    } finally {

      setIsLoading(false);

    }

  };

  return (

    <>

      <Form {...form}>

        <form onSubmit={form.handleSubmit(onSubmit)} className="auth-form">

          <h1 className="form-title">

            {type === "sign-in" ? "Sign In" : "Sign Up"}

          </h1>

          {type === "sign-up" && (

            <FormField

              control={form.control}

              name="fullName"

              render={({ field }) => (

                <FormItem>

                  <div className="shad-form-item">

                    <FormLabel className="shad-form-label">Full Name</FormLabel>

                    <FormControl>

                      <Input

                        placeholder="Enter your full name"

                        className="shad-input"

                        {...field}

                      />

                    </FormControl>

                  </div>

                  <FormMessage className="shad-form-message" />

                </FormItem>

              )}

            />

          )}

          <FormField

            control={form.control}

            name="email"

            render={({ field }) => (

              <FormItem>

                <div className="shad-form-item">

                  <FormLabel className="shad-form-label">Email</FormLabel>

                  <FormControl>

                    <Input

                      placeholder="Enter your email"

                      className="shad-input"

                      {...field}

                    />

                  </FormControl>

                </div>

                <FormMessage className="shad-form-message" />

              </FormItem>

            )}

          />

          <Button

            type="submit"

            className="form-submit-button"

            disabled={isLoading}

          >

            {type === "sign-in" ? "Sign In" : "Sign Up"}

            {isLoading && (

              <Image

                src="/assets/icons/loader.svg"

                alt="loader"

                width={24}

                height={24}

                className="ml-2 animate-spin"

              />

            )}

          </Button>

          {errorMessage && <p className="error-message">\*{errorMessage}</p>}

          <div className="body-2 flex justify-center">

            <p className="text-light-100">

              {type === "sign-in"

                ? "Don't have an account?"

                : "Already have an account?"}

            </p>

            <Link

              href={type === "sign-in" ? "/sign-up" : "/sign-in"}

              className="ml-1 font-medium text-brand"

            >

              {" "}

              {type === "sign-in" ? "Sign Up" : "Sign In"}

            </Link>

          </div>

        </form>

      </Form>

      {accountId && (

        <OtpModal email={form.getValues("email")} accountId={accountId} />

      )}

    </>

  );

};

export default AuthForm;