```
import { createClient } from '@/shared/utils/supabase/server';

import { supabaseAdmin } from '../supabase/admin';

import axios from 'axios';


interface TwentyCrmUser {

  name: string;

  email: string;

  signUpIncomplete: boolean;

}


// we use this method as patch , we added trigger for new users so their auth email will sync with
postgres to users table.

export const syncUserEmail = async (id: string, email: string) => {

  if (!email || !id) return;

  try {

    const user = await supabaseAdmin

      .from('users')

      .select('id,email')

      .eq('id', id)

      .single();

    if (user.data?.id && !user.data?.email) {

      await supabaseAdmin.from('users').update({ email }).eq('id', id);

    }

  } catch (error) {

    console.error('syncUserEmail', id, error);

  }
```

```
};


export const createTwentyCRMUser = async (user: TwentyCrmUser) => {
  const response = await axios.post(
    `${process.env.TWENTY_CRM_API_URL}/swarmsWebsiteUsers`,
    user,
    {
      headers: {
        Authorization: `Bearer ${process.env.TWENTY_CRM_TOKEN}`,
      },
    },
  );
  return response.data?.data?.createSwarmsWebsiteUser?.id;
};


export async function syncTwentyCRMId(userId: string, twenty_crm_id: string) {
  if (!userId || !twenty_crm_id) return;


  try {
    const { error } = await supabaseAdmin
      .from('users')
      .update({ twenty_crm_id })
      .eq('id', userId);
  } catch (error) {
    console.error('syncTwentyCRMId', userId, error);
  }
```

```
}


export const getUserById = async (id: string) => {
  if (!id) return;
  try {
    const user = await supabaseAdmin
      .from('users')
      .select('id, email, twenty_crm_id')
      .eq('id', id)
      .single();

    return user.data;
  } catch (error) {
    console.error('getUserById', id, error);
  }
};


export async function updateTwentyCrmUser(id: string, data: any) {
  if (!id) return;
  try {
    const user = await supabaseAdmin
      .from('users')
      .select('id, email, twenty_crm_id')
      .eq('id', id)
      .single();
    if (!user.data?.twenty_crm_id) {
      console.log('User does not have a twenty_crm_id');
      return;
    }

    const response = await axios.patch(
      `${process.env.TWENTY_CRM_API_URL}/swarmsWebsiteUsers/${user.data?.twenty_crm_id}`,
      data,
      {
        headers: {
          Authorization: `Bearer ${process.env.TWENTY_CRM_TOKEN}`,
        },
      },
    );
    return response.data;
  } catch (error) {
    console.error('updateTwentyCrmUser', id, error);
  }
}

export const updateFreeCreditsOnSignin = async (id: string): Promise<void> => {
  if (!id) {
    return; // Early return for invalid input
  }

  try {
```

```
// Fetch user data and check for existing free credits
const { data: user } = await supabaseAdmin
  .from('users')
  .select('had_free_credits')
  .eq('id', id)
  .single();

if (user?.had_free_credits) {
  return; // User already has free credits
}

// Check for email confirmation
const { data } = await createClient().auth.getUser();

if (!data?.user?.email_confirmed_at) {
  return; // User email not confirmed
}

// Update free credits and user flag
const { error: creditError } = await supabaseAdmin
  .from('swarms_cloud_users_credits')
  .select('*')
  .eq('user_id', id)
  .single();

if (creditError) {
      if (creditError.code === 'PGRST116') {
        // Insert a new entry with initial credit values
        await supabaseAdmin.from('swarms_cloud_users_credits').insert([
          {
            user_id: id,
            free_credit: 20,
          },
        ]);
      }
    } else {
      await supabaseAdmin
        .from('swarms_cloud_users_credits')
        .update({ free_credit: 20 })
        .eq('user_id', id);
    }

    await supabaseAdmin
      .from('users')
      .update({ had_free_credits: true })
      .eq('id', id);
  } catch (error) {
    console.error('Error updating free credits:', error);
  }
};
```