```
import { router, userProcedure } from '@/app/api/trpc/trpc-router';
import { getURL } from '@/shared/utils/helpers';
import mailer from '@/shared/utils/mailer';
import { User } from '@supabase/supabase-js';
import { z } from 'zod';
import { v4 as uuidv4 } from 'uuid';
import { getUserOrganizationRole } from '@/shared/utils/api/organization';
import { Enums, Tables } from '@/types_db';
import { ORG_MEMBER_INVITE_TIMEOUT } from '@/shared/constants/common';

const organizationRouter = router({
  // get user organization info

  getUserPersonalOrganization: userProcedure.query(async ({ ctx }) => {
    const user = ctx.session.data.session?.user as User;

    const org = await ctx.supabase
      .from('swarms_cloud_organizations')
      .select('*')
      .eq('owner_user_id', user.id)
      .single();
    return org;
  }),

  getUserOrganizations: userProcedure.query(async ({ ctx }) => {
    const user = ctx.session.data.session?.user as User;
```

```
const allOrgs: {
  role: Enums<'organization_member_role'> | 'owner';
  organization: Partial<Tables<'swarms_cloud_organizations'>>;
}[] = [];

// personal
const personalOrg = await ctx.supabase
  .from('swarms_cloud_organizations')
  .select('*')
  .eq('owner_user_id', user.id)
  .limit(1);

if (personalOrg.data?.length) {
  allOrgs.push({
    role: 'owner',
    organization: personalOrg.data[0],
  });
}

// other
const members = await ctx.supabase
  .from('swarms_cloud_organization_members')
  .select('*')
  .eq('user_id', user.id)
  .filter('is_deleted', 'not.is', 'true');
```

```
  if (members.data) {
    for (const member of members.data) {
      if (!member.organization_id) {
        continue;
      }
      const org = await ctx.supabase
        .from('swarms_cloud_organizations')
        .select('id,name,public_id')
        .eq('id', member.organization_id)
        .limit(1);
      if (org.data?.length && member.role) {
        allOrgs.push({
          role: member.role,
          organization: org.data[0],
        });
      }
    }
  }
  return allOrgs;
}),

getOrganizationInfo: userProcedure
  .input(
    z.object({
      id: z.string(),
```

```
    }),
  )
  .query(async ({ ctx, input: { id } }) => {
    const user = ctx.session.data.session?.user as User;


    const role = await getUserOrganizationRole(id, user.id);
    if (!role) {
      throw new Error('Access denied');
    }
    const org = await ctx.supabase
      .from('swarms_cloud_organizations')
      .select('id,name,public_id')
      .eq('id', id)
      .limit(1);
    return org.data?.[0];
  }),

// create new organization
createOrganization: userProcedure
  .input(
    z.object({
      name: z.string().min(3),
    }),
  )
  .mutation(async ({ ctx, input: { name } }) => {
    const user = ctx.session.data.session?.user as User;
```

```
    // check user already have organization
    const userOrg = await ctx.supabase
      .from('swarms_cloud_organizations')
      .select('*')
      .eq('owner_user_id', user.id)
      .limit(1);

    if (userOrg.data?.length) {
      throw new Error('User already have organization');
    }

    const org = await ctx.supabase
      .from('swarms_cloud_organizations')
      .insert({ name, owner_user_id: user.id });

    if (org.error) {
      throw new Error(org.error.message);
    }
    return org.data;
  }),
// update name
updateOrganizationName: userProcedure
  .input(
    z.object({
      id: z.string(),
```

```
      name: z.string().min(3),
    }),
  )
  .mutation(async ({ ctx, input: { id, name } }) => {
    const user = ctx.session.data.session?.user as User;

    const userRole = await getUserOrganizationRole(id, user.id);

    if (!userRole || userRole == 'reader') {
      throw new Error('Access denied');
    }
    const org = await ctx.supabase
      .from('swarms_cloud_organizations')
      .update({ name })
      .eq('id', id)
      .select('*');

    if (!org.data?.length) {
      throw new Error('Organization not found');
    }
    return true;
  }),

// members
// list
members: userProcedure
```

```
.input(
  z.object({
    id: z.string(),
  }),
)
.query(async ({ ctx, input: { id } }) => {
  const user = ctx.session.data.session?.user as User;

  // check access: user should be owner or member
  const userRole = await getUserOrganizationRole(id, user.id);

  if (!userRole) {
    throw new Error('Access denied');
  }

  const members: {
    user_id: string;
    name: string;
    role: Enums<'organization_member_role'> | 'owner';
  }[] = [];

  const { data: org } = await ctx.supabase
    .from('swarms_cloud_organizations')
    .select('*')
    .eq('id', id)
    .limit(1);
```

```
if (!org?.length) {
  throw new Error('Organization not found');
}
const { data: orgOwnerRes } = await ctx.supabase
  .from('users')
  .select('*')
  .eq('id', org?.[0]?.owner_user_id ?? '')
  .limit(1);
if (!orgOwnerRes?.length) {
  throw new Error('Organization owner not found');
}
const orgOwner = orgOwnerRes?.[0];
if (orgOwner) {
  members.push({
    user_id: orgOwner.id,
    name: orgOwner.full_name || '',
    role: 'owner',
  });
}
// members
const orgMembers = await ctx.supabase
  .from('swarms_cloud_organization_members')
  .select('*')
  .eq('organization_id', id)
  .filter('is_deleted', 'not.is', 'true');
```

```
    if (orgMembers.data) {
      for (const member of orgMembers.data) {
        if (!member.user_id || !member.role) {
          continue;
        }
        const { data: memberRes } = await ctx.supabase
          .from('users')
          .select('*')
          .eq('id', member.user_id)
          .limit(1);
        if (!memberRes?.length) {
          continue;
        }
        const member_user = memberRes[0];
        members.push({
          user_id: member.user_id,
          name: member_user.full_name || '',
          role: member.role,
        });
      }
    }

    return members;
  }),
// , invite by email
```

```
inviteMemberByEmail: userProcedure
  .input(
    z.object({
      id: z.string(),
      email: z.string().email(),
      role: z.enum(['manager', 'reader']),
    }),
  )
  .mutation(async ({ ctx, input: { id, email, role } }) => {
    // check access: user should be owner or member with manager role

    const user = ctx.session.data.session?.user as User;
    const userRole = await getUserOrganizationRole(id, user.id);

    if (!userRole || userRole == 'reader') {
      throw new Error('Access denied');
    }
    if (userRole !== 'owner' && role == 'manager') {
      throw new Error('you can not invite manager');
    }
    const orgRes = await ctx.supabase
      .from('swarms_cloud_organizations')
      .select('id,name,public_id')
      .eq('id', id)
      .limit(1);
    const org = orgRes.data?.[0];
```

```
if (!org) {
  throw new Error('Organization not found');
  return;
}

// check email already have account
const { data, error } = await ctx.supabase.rpc(
  // @ts-ignore
  'get_user_id_by_email',
  {
    email,
  },
);
// @ts-ignore
const invitedUser = (data as { id: string }[])?.[0];
if (error) {
  throw new Error('Failed to check email');
}
if (!invitedUser) {
  // Todo: fix soon
  throw new Error('User need to signup first');
}

// check duplicate member
const { data: member } = await ctx.supabase
```

```
  .from('swarms_cloud_organization_members')
  .select('*')
  .eq('organization_id', id)
  .eq('user_id', invitedUser.id)
  .filter('is_deleted', 'not.is', 'true')
  .limit(1);

if (member?.length) {
  throw new Error('User already member of this organization');
}

// inviter only can invite 10 times in hour
const { data: invites } = await ctx.supabase
  .from('swarms_cloud_organization_member_invites')
  .select('*')
  .eq('invite_by_user_id', user.id)
  .gte('created_at', new Date(new Date().getTime() - 60 * 60 * 1000));

if ((invites?.length ?? 0) >= 10) {
  throw new Error('You can only invite 10 times in an hour');
}

// check last invite record
const { data: lastInvites } = await ctx.supabase
  .from('swarms_cloud_organization_member_invites')
  .select('*')
```

```
  .eq('organization_id', id)
  .eq('email', email)
  .eq('status', 'waiting')
  .order('created_at', { ascending: false })
  .limit(1);

const lastInvite = lastInvites?.[0];

if (lastInvite) {
  // check if its not  expired
  if (
    new Date().getTime() - new Date(lastInvite.created_at).getTime() <
    ORG_MEMBER_INVITE_TIMEOUT
  ) {
    throw new Error('Invite already sent');
  } else {
    // update status to expired
    await ctx.supabase
      .from('swarms_cloud_organization_member_invites')
      .update({ status: 'expired' })
      .eq('id', lastInvite.id);
  }
}
// make new invite

const host_url = getURL();
```

```
    const mail = mailer();

    const secret_code = uuidv4();
    const accept_code_api = 'organization/accept-invite';
      const html = `You have been invited to join ${org?.name} Organization. Click here to accept: ${host_url}api/${accept_code_api}?code=${secret_code}`;

    try {
      const sendEmail = await mail.sendMail({
        from: `kye@apac.ai`,
        to: email,
        subject: 'Invitation to join organization',
        html,
      });
      if (!sendEmail) {
        throw new Error('Failed to send email');
      }

      const res = await ctx.supabase
        .from('swarms_cloud_organization_member_invites')
        .insert({
          organization_id: id,
          email,
          secret_code,
          user_id: invitedUser.id,
          invite_by_user_id: user.id,
```

```
          status: 'waiting',
          role: role as Enums<'organization_member_role'>,
        });

      return true;
    } catch (error) {
      throw new Error('Failed to send email');
    }
  }),
pendingInvites: userProcedure
  .input(z.object({ organization_id: z.string() }))
  .query(async ({ ctx, input: { organization_id } }) => {
    const user = ctx.session.data.session?.user as User;
    const userRole = await getUserOrganizationRole(organization_id, user.id);

    if (!userRole || userRole == 'reader') {
      throw new Error('Access denied');
    }

    const invites = await ctx.supabase
      .from('swarms_cloud_organization_member_invites')
      .select('id,email,role,created_at')
      .eq('organization_id', organization_id)
      .eq('status', 'waiting');

    return invites.data;
```

```
  }),

cancelInvite: userProcedure
  .input(
    z.object({
      organization_id: z.string(),
      email: z.string().email(),
    }),
  )
  .mutation(async ({ ctx, input: { organization_id, email } }) => {
    const user = ctx.session.data.session?.user as User;

    // check access: user should be owner or member with manager role

    const userRole = await getUserOrganizationRole(organization_id, user.id);

    if (!userRole || userRole == 'reader') {
      throw new Error('Access denied');
    }

    const invites = await ctx.supabase
      .from('swarms_cloud_organization_member_invites')
      .select('*')
      .eq('organization_id', organization_id)
      .eq('email', email)
      .eq('status', 'waiting')
```

```
      .limit(1);

    if (!invites.data?.length) {
      throw new Error('Invite not found');
    }

    await ctx.supabase
      .from('swarms_cloud_organization_member_invites')
      .update({ status: 'canceled' })
      .eq('id', invites.data[0].id);

    return true;
  }),

leaveOrganization: userProcedure
  .input(
    z.object({
      organization_id: z.string(),
    }),
  )
  .mutation(async ({ ctx, input: { organization_id } }) => {
    const user = ctx.session.data.session?.user as User;

    const userRole = await getUserOrganizationRole(organization_id, user.id);

    if (!userRole) {
```

```
      throw new Error('Access denied');
    }

    if (userRole == 'owner') {
      throw new Error('Owner can not leave organization');
    }

    const member = await ctx.supabase
      .from('swarms_cloud_organization_members')
      .select('*')
      .eq('organization_id', organization_id)
      .eq('user_id', user.id)
      .filter('is_deleted', 'not.is', 'true')
      .limit(1);

    if (!member.data?.length) {
      throw new Error('User not member of this organization');
    }
    await ctx.supabase
      .from('swarms_cloud_organization_members')
      .update({ is_deleted: true })
      .eq('id', member.data[0].id);

    return true;
  }),
deleteMember: userProcedure
```

```
.input(
  z.object({
    organization_id: z.string(),
    user_id: z.string(),
  }),
)
.mutation(async ({ ctx, input: { organization_id, user_id } }) => {
  const user = ctx.session.data.session?.user as User;

  const userRole = await getUserOrganizationRole(organization_id, user.id);

  if (!userRole || userRole == 'reader') {
    throw new Error('Access denied');
  }
  // managers cant delete other managers
  if (userRole == 'manager') {
    const memberRole = await getUserOrganizationRole(
      organization_id,
      user_id,
    );
    if (memberRole == 'manager') {
      throw new Error('Access denied');
    }
  }

  const member = await ctx.supabase
```

```
      .from('swarms_cloud_organization_members')
      .select('*')
      .eq('organization_id', organization_id)
      .eq('user_id', user_id)
      .filter('is_deleted', 'not.is', 'true')
      .limit(1);

    if (!member.data?.length) {
      throw new Error('User not member of this organization');
    }
    await ctx.supabase
      .from('swarms_cloud_organization_members')
      .update({ is_deleted: true, deleted_by_user_id: user.id })
      .eq('id', member.data[0].id);

    return true;
  }),
changeMemberRole: userProcedure
  .input(
    z.object({
      organization_id: z.string(),
      user_id: z.string(),
      role: z.enum(['manager', 'reader']),
    }),
  )
  .mutation(async ({ ctx, input: { organization_id, user_id, role } }) => {
```

```
  const user = ctx.session.data.session?.user as User;

  const userRole = await getUserOrganizationRole(organization_id, user.id);

  if (!userRole || userRole == 'reader') {
    throw new Error('Access denied');
  }

  const member = await ctx.supabase
    .from('swarms_cloud_organization_members')
    .select('*')
    .eq('organization_id', organization_id)
    .eq('user_id', user_id)
    .filter('is_deleted', 'not.is', 'true')
    .limit(1);

  if (!member.data?.length) {
    throw new Error('User not member of this organization');
  }
  await ctx.supabase
    .from('swarms_cloud_organization_members')
    .update({ role })
    .eq('id', member.data[0].id);

  return true;
}),
```

```
});


export { organizationRouter };
```