```
import { useOrganizationStore } from '@/shared/stores/organization';
import { useEffect, useMemo } from 'react';
import { FormEvent, useState } from 'react';
import { useToast } from '@/shared/components/ui/Toasts/use-toast';
import confetti from 'canvas-confetti';
import { trpc } from '@/shared/utils/trpc/trpc';
import { useAuthContext } from '@/shared/components/ui/auth.provider';

type FormMutationType<T> = {
  mutateAsync: (data: T) => Promise<null | boolean>;
};

interface FormMutationProps<T> {
  e?: FormEvent<HTMLFormElement>;
  options?: Record<string, any>;
  data?: T;
  mutationFunction: FormMutationType<T>;
  toastMessage?: string;
}

export const useOrganizations = () => {
  const { query } = useQueryMutation();
  const currentOrgId = useOrganizationStore((state) => state.currentOrgId);

  const userOrgData = query?.organization?.data;
  const usersOrgData = query?.organizations?.data;
```

```
  const currentOrganization = useMemo(() => {
    return usersOrgData?.find((org) => org.organization.id === currentOrgId);
  }, [usersOrgData, currentOrgId]);

  const currentId =
    userOrgData?.data?.id || usersOrgData?.[0]?.organization?.id;

  useEffect(() => {
    if (userOrgData?.data?.id) {
      useOrganizationStore.getState().setUserOrgId(userOrgData.data.id);
    }

    if (currentId) {
      useOrganizationStore.getState().setCurrentOrgId(currentId);
    }
  }, [currentId]);

  return {
    userOrgData,
    usersOrgData,
    currentOrganization,
  };
};

// Returns reuseable mutation logic
```

```
// handleFormMutation => forms
// withOrganizationMutation => event handlers
export function useOrganizationMutation() {
  const toast = useToast();
  const { user } = useAuthContext();
  const currentOrgId = useOrganizationStore((state) => state.currentOrgId);
  const { query } = useQueryMutation();

  const [openDialog, setOpenDialog] = useState(false);
  const [openRoleDialog, setOpenRoleDialog] = useState(false);

  async function handleFormMutation<T>({
    e,
    mutationFunction,
    toastMessage,
    options,
  }: FormMutationProps<T>) {
    e?.preventDefault();

    if (!user) {
      toast.toast({
        description: 'Log in to perform this action',
        style: { color: 'red' },
      });
      return;
    }
```

```
const formData = new FormData(e?.currentTarget);
const data = {
  ...Object.fromEntries(formData),
  ...options,
};

for (const [key, value] of Object.entries(data)) {
  if (!value || value.toString().trim().length < 3) {
    toast.toast({
      description: `${key} must be at least 3 characters long`,
      style: { color: 'red' },
    });
    return;
  }
}

useOrganizationStore.getState().setIsLoading(true);

try {
  const response = await mutationFunction.mutateAsync(data as T);
  console.log(response);
  toast.toast({
    description: toastMessage || 'Request is successful',
    style: { color: 'green' },
  });
```

```
    setOpenDialog(false);
    confetti({
      particleCount: 150,
      spread: 90,
      origin: { y: 0.6 },
    });

    query?.organization?.refetch();
    query?.organizations?.refetch();

    e?.currentTarget?.reset();
  } catch (error: any) {
    if (error?.message) {
      toast.toast({
        description: error?.message,
        style: { color: 'red' },
      });
    }
  } finally {
    useOrganizationStore.getState().setIsLoading(false);
  }
}

async function withOrganizationMutation<T>({
  data,
  mutationFunction,
```

```
  toastMessage,
}: FormMutationProps<T>) {
  if (!user) {
    toast.toast({
      description: 'Log in to perform this action',
      style: { color: 'red' },
    });
    return;
  }

  if (!currentOrgId) {
    toast.toast({
      description: 'Organization not found',
      style: { color: 'red' },
    });
    return;
  }

  useOrganizationStore.getState().setIsLoading(true);

  try {
    const response = await mutationFunction.mutateAsync(data as T);
    if (response) {
      setOpenDialog(false);
      toast.toast({ description: toastMessage, style: { color: 'green' } });
      query?.members?.refetch();
```

```
        query?.organizations?.refetch();
      }
    } catch (error) {
      if ((error as any)?.message) {
        toast.toast({ description: (error as any)?.message });
      }
    } finally {
      useOrganizationStore.getState().setIsLoading(false);
    }
  }

  return {
    handleFormMutation,
    withOrganizationMutation,
    openDialog,
    openRoleDialog,
    setOpenRoleDialog,
    setOpenDialog,
  };
}

// return all possible queries and mutations
export function useQueryMutation() {
  const userOrgId = useOrganizationStore((state) => state.userOrgId);
  const currentOrgId = useOrganizationStore((state) => state.currentOrgId);
  const { user } = useAuthContext();
```

```
// queries
const userOrganizationsQuery = user
  ? trpc.organization.getUserOrganizations.useQuery()
  : null;
const userOrganizationQuery = user
  ? trpc.organization.getUserPersonalOrganization.useQuery()
  : null;
const organizationMembersQuery = user
  ? trpc.organization.members.useQuery({
      id: currentOrgId ?? '',
    })
  : null;
const pendingInvitesQuery = user
  ? trpc.organization.pendingInvites.useQuery({
      organization_id: userOrgId ?? '',
    })
  : null;

// mutations
const createOrgMutation = trpc.organization.createOrganization.useMutation();
const updateOrgNameMutation =
  trpc.organization.updateOrganizationName.useMutation();
const inviteEmailMutation =
  trpc.organization.inviteMemberByEmail.useMutation();
const changeRoleMutation = trpc.organization.changeMemberRole.useMutation();
```

```
  const leaveOrganizationMutation =
    trpc.organization.leaveOrganization.useMutation();
  const deleteMemberMutation = trpc.organization.deleteMember.useMutation();
  const cancelledInvitesMutation = trpc.organization.cancelInvite.useMutation();

  const query = {
    organization: userOrganizationQuery,
    organizations: userOrganizationsQuery,
    members: organizationMembersQuery,
    invites: pendingInvitesQuery,
  };

  const mutation = {
    create: createOrgMutation,
    update: updateOrgNameMutation,
    invite: inviteEmailMutation,
    changeRole: changeRoleMutation,
    leave: leaveOrganizationMutation,
    delete: deleteMemberMutation,
    cancel: cancelledInvitesMutation,
  };

  return { query, mutation };
}
```