```
import React, { memo, useEffect, useMemo, useRef, useState } from 'react';
import { ChevronDown } from 'lucide-react';
import { Button } from '@/shared/components/ui/Button';
import useToggle from '@/shared/hooks/toggle';
import { useOnClickOutside } from '@/shared/hooks/onclick-outside';
import { cn } from '@/shared/utils/cn';
import {
  ExcludeOwner,
  MemberProps,
  UserOrganizationsProps,
} from '../../../types';
import { useOrganizationStore } from '@/shared/stores/organization';
import { useToast } from '@/shared/components/ui/Toasts/use-toast';
import ModalPrompt from '../../prompt';
import { ROLES } from '@/shared/constants/organization';
import { useOrganizationTeam } from '../../../hooks/team';
import { User } from '@supabase/supabase-js';
import { useOrganizationMutation } from '../../../hooks/organizations';
import LoadingSpinner from '@/shared/components/loading-spinner';

interface TeamMemberProps {
  member: MemberProps;
  user: User | null;
  currentOrganization: UserOrganizationsProps;
}
```

```
function TeamMember({ member, user, currentOrganization }: TeamMemberProps) {
  const { handleRoleChange, handleLeaveOrg, handleDeleteMember } =
    useOrganizationTeam();
  const { openDialog, openRoleDialog, setOpenRoleDialog, setOpenDialog } =
    useOrganizationMutation();

  const toast = useToast();
  const memberRef = useRef(null);
  const { isOn, setOff, setOn } = useToggle();
  const isLoading = useOrganizationStore((state) => state.isLoading);

  useOnClickOutside(memberRef, setOff);

  const ownerRole = member.role === 'owner';
  const isCurrentUser = member.user_id === user?.id;
  const isOrgOwner = currentOrganization?.role === 'owner';
  const isReader = currentOrganization?.role === 'reader';
  const isManager = currentOrganization?.role === 'manager';

  const [memberRole, setMemberRole] = useState('');
  const [isMemberRoleLoading, setIsMemberRoleLoading] = useState(false);

  const allMemberRoles = useMemo(() => {
    const excludedRoles = ['Team roles', 'owner'];
    return ROLES.filter((role) => !excludedRoles.includes(role.value)).map(
      (role) => role.value,
```

```
  );
}, [member.role]);

function handleMemberRole(role: ExcludeOwner) {
  setMemberRole(role);
}

async function handleUserRole() {
  if (!memberRole || memberRole === member?.role)
    return toast.toast({
      description: 'Role already exists',
      style: { color: 'red' },
    });
  setIsMemberRoleLoading(true);
  await handleRoleChange?.(member.user_id, memberRole as ExcludeOwner);
  setIsMemberRoleLoading(false);
  setOpenRoleDialog(false);
}

function handleModal() {
  if (ownerRole) return;
  setOn();
}

return (
   <div className="flex flex-col sm:flex-row items-center sm:items-start sm:justify-between border
```

```
rounded-md px-2 py-4 sm:p-4 text-card-foreground hover:opacity-80 w-full max-sm:gap-2">
      <div className="flex items-center gap-2 basis-1/2">
            <span  className="h-7  w-7  sm:w-10  sm:h-10  text-sm  sm:text-base  flex  justify-center
items-center bg-slate-500 text-white rounded-full uppercase">
        {member?.name.charAt(0)}
       </span>
       <p>{member?.name}</p>
      </div>
      <div className="relative basis-1/3">
       <div
        className={cn(
              'border  w-28  p-2  cursor-pointer  text-center  rounded-md  capitalize  flex  justify-between
items-center',
          !(!ownerRole && isOrgOwner) && 'justify-center cursor-not-allowed',
        )}
        onClick={handleModal}
       >
        {isMemberRoleLoading ? (
          <LoadingSpinner />
        ) : (
          <>
            <span>{member.role}</span>
            {!ownerRole && isOrgOwner && <ChevronDown size={20} />}
          </>
        )}
       </div>
```

```
{isOrgOwner && (
  <ul
    ref={memberRef}
    className={cn(
          'absolute list-none border bg-secondary w-32 flex flex-col items-center rounded-md bottom-8 left-14 transition-all invisible',
      isOn && 'visible',
    )}
  >
    {allMemberRoles?.map((role) => (
      <li
        key={role}
        onClick={() => handleMemberRole(role as ExcludeOwner)}
        className="hover:text-secondary hover:bg-foreground capitalize w-full py-2 text-center"
      >
        <ModalPrompt
          content={`Do you wish to change the role for ${member.name || ''}?`}
          isLoading={isLoading}
          openDialog={openRoleDialog}
          setOpenDialog={setOpenRoleDialog}
          handleClick={() => handleUserRole()}
        >
          <Button
            variant="ghost"
            className="capitalize hover:bg-transparent hover:text-secondary transition-none"
            aria-label={role}
```

```
            >
              {role}
            </Button>
          </ModalPrompt>
        </li>
      ))}
    </ul>
  )}
</div>
<div className="w-full sm:max-w-[86px] flex justify-center mt-2 md:mt-0">
  {(isCurrentUser && isReader) || (isCurrentUser && isManager) ? (
    <ModalPrompt
      content={`Can you confirm you're leaving?`}
      isLoading={isLoading}
      handleClick={handleLeaveOrg as () => void}
      openDialog={openDialog}
      setOpenDialog={setOpenDialog}
    >
      <Button className={cn('h-7 sm:h-10 sm:w-full')}>Leave</Button>
    </ModalPrompt>
  ) : !isCurrentUser &&
    ((isManager && member.role !== 'manager') || isOrgOwner) &&
    !ownerRole ? (
    <ModalPrompt
      content={`Would you like to remove ${member.name || ''} from this organization?`}
      isLoading={isLoading}
```

```
            handleClick={() => handleDeleteMember?.(member.user_id)}
            openDialog={openDialog}
            setOpenDialog={setOpenDialog}
          >
            <Button className={cn('h-7 sm:h-10 sm:w-full')}>Remove</Button>
          </ModalPrompt>
        ) : null}
      </div>
    </div>
  );
}

export default memo(TeamMember);
```