SOLUTION Completing the profile service functionality

Profile API

1. Create endpoints for the following:
    1. Get list of profiles that takes a sortBy alphabetical or reputation
    2. Get individual profile
    3. Edit profile

In order to test the sorting you might want to update the reputation score of the user that comes last alphabetically to 1. If using pgWeb you can use the following query to achieve this:

```sql
UPDATE "UserProfiles"
SET "Reputation" = 1
WHERE "Id" = '73569a32-6903-4538-89cc-9a126132ab59';
```

List:

```csharp
app.MapGet("profiles", async (string? sortBy, ProfileDbContext db) =>
{
    var query = db.UserProfiles.AsQueryable();

    query = sortBy == "reputation"
        ? query.OrderByDescending(x => x.Reputation)
        : query.OrderBy(x => x.DisplayName);

    return await query.ToListAsync();
});
```

Individual

```csharp
app.MapGet("profiles/{id}", async (string id, ProfileDbContext db) =>
{
    var profile = await db.UserProfiles.FindAsync(id);

    return profile is null ? Results.NotFound() : Results.Ok(profile);
});
```

Edit. We need a DTO for this one.

```csharp
namespace ProfileService.DTOs;

public record EditProfileDto(string? DisplayName, string? Description);
```

```csharp
app.MapPut("/profiles/edit", async (EditProfileDto dto, ClaimsPrincipal user,
ProfileDbContext db) =>
{
    var userId = user.FindFirstValue(ClaimTypes.NameIdentifier);
    if (userId is null) return Results.Unauthorized();

    var profile = await db.UserProfiles.FindAsync(userId);
    if (profile is null) return Results.NotFound();

    profile.DisplayName = dto.DisplayName ?? profile.DisplayName;
    profile.Description = dto.Description ?? profile.Description;

    await db.SaveChangesAsync();

    return Results.NoContent();
}).RequireAuthorization();
```

Test the functionality by running the postman requests in Postman and confirm working before moving on.

NextJS App

1. Create a new link in the SideMenu.tsx to go to the profiles page.tsx

```
export default function SideMenu() {
    const pathname = usePathname();
    const navLinks = [
        {key: 'home', icon: HomeIcon, text: 'Home', href: '/'},
        {key: 'questions', icon: QuestionMarkCircleIcon, text: 'Questions', href: '/questions'},
        {key: 'tags', icon: TagIcon, text: 'Tags', href: '/tags'},
        {key: 'session', icon: UserIcon, text: 'User Session', href: '/session'},
        {key: 'profiles', icon: UserIcon, text: 'Profiles', href: '/profiles'},
    ]

```

2. Create an editProfileSchema. The user can update the DisplayName and the Description.

```
import {z} from "zod";

const required = (name: string) => z.string().trim().min(1,
    {message: `${name} is required`});

export const editProfileSchema = z.object({
    displayName: required('Display name'),
    description: required('Description'),
});

export type EditProfileSchema = z.infer<typeof editProfileSchema>;
```

3. Create a new actions file called profile-actions.ts and create the following functions:
    1. getProfiles(string? sortBy)
    2. getProfileById(string id)
    3. editProfile(EditProfileSchema data)

```
'use server';

import {fetchClient} from "@/lib/fetchClient";
import {Profile} from "@/lib/types";
import {revalidatePath} from "next/cache";
import {EditProfileSchema} from "@/lib/schemas/editProfileSchema";

export async function getUserProfiles(sortBy?: string) {
    let url = '/profiles';
    if (sortBy) url += '?sortBy=' + sortBy;
    return fetchClient<Profile[]>(url, 'GET');
}

export async function getProfileById(id: string) {
    return fetchClient<Profile>(`/profiles/${id}`, 'GET');
}

export async function editProfile(id: string, profile: EditProfileSchema) {
    const result = await fetchClient<Profile>(`/profiles/edit`, 'PUT', {body: profile});

    revalidatePath(`/profiles/${id}`)

    return result;
}
```

4. Create a new page in app/profiles/page.tsx. This will display a list of users and their profiles in a table. Check the HeroUI docs on how to use a Table for this. You can consult the HeroUI docs which has good examples of using Tables for this.
    1. Each row should be clickable and be a link to the users individual profile.
    2. Each column heading should be clickable to allow sorting by the column name (hint: you may want to use pgAdmin to update one of the users reputation scores so you can see this visibly working in the UI).

```
import ProfilesList from "@/app/profiles/ProfilesList";
import {getUserProfiles} from "@/lib/actions/profile-actions";
import {handleError} from "@/lib/util";

type SearchParams = Promise<{sortBy?: string}>

export default async function Page({searchParams}: {searchParams: SearchParams}) {
    const {sortBy} = await searchParams;
    const {data: profiles, error} = await getUserProfiles(sortBy);

    if (error) handleError(error);
    if (!profiles) return;

    return (
        <div className="flex flex-col gap-3 px-6">
            <h3 className='text-3xl font-semibold'>User table</h3>
            <ProfilesList profiles={profiles} />
        </div>
    );
}
```

```
'use client';

import {
    getKeyValue,
    Table,
    TableBody,
    TableCell,
    TableColumn,
    TableHeader,
    TableRow
} from "@heroui/table";
import {Profile} from "@/lib/types";
import {useRouter} from "next/navigation";
import {SortDescriptor} from "@heroui/react";

type Props = {
    profiles: Profile[];
}

export default function ProfilesList({profiles}: Props) {
    const router = useRouter();
    const columns = [
        {key: 'displayName', label: 'Display Name'},
        {key: 'reputation', label: 'Reputation'},
    ]

    const onSortChange = (sort: SortDescriptor) => {
        router.push(`/profiles?sortBy=${sort.column}`);
    }

    return (
        <Table
            onSortChange={(sort) => onSortChange(sort)}
            aria-label='User profiles'
            selectionMode='single'
            onRowAction={(key) => router.push(`/profiles/${key}`)}
        >
            <TableHeader columns={columns}>
                {(column) =>
                    <TableColumn
                        key={column.key}
                        allowsSorting
                    >
                        {column.label}
                    </TableColumn>}
            </TableHeader>
            <TableBody items={profiles}>
                {(item) => (
                    <TableRow key={item.userId}
                              className='hover:cursor-pointer'>
                        {(columnKey) => <TableCell>{getKeyValue(item, columnKey)}</TableCell>}
                    </TableRow>
                )}
            </TableBody>
        </Table>
    );
}

```

5. Create an individual profile page which displays the user display name, their description and their reputation. I went for a very simple approach and here is the JSX for this one just using a simple HeroUI card. Feel free to be more creative than this!

Example Profile card.

```
        <Card>
            <CardHeader className='text-3xl font-semibold flex justify-between'>
                <div className='flex items-center gap-3'>
                    <Avatar className='h-30 w-30' color='secondary' />
                    <span>Display name</span>
                </div>
                <Button
                    variant='bordered'
                >
                    Edit profile
                </Button>
            </CardHeader>
            <Divider />
            <CardBody>
                <p>No profile description added yet</p>Ï
            </CardBody>
            <CardFooter className='text-xl font-semibold'>
                Reputation score: 42
            </CardFooter>
        </Card>
```

page.tsx

```
import {getProfileById} from "@/lib/actions/profile-actions";
import {notFound} from "next/navigation";
import {handleError} from "@/lib/util";
import ProfileDetailed from "@/app/profiles/[id]/ProfileDetailed";
import {getCurrentUser} from "@/lib/actions/auth-actions";

type Params = Promise<{id: string}>
export default async function Page({params}: {params: Params}) {
    const {id} = await params;
    const {data: profile, error} = await getProfileById(id);

    if (error) handleError(error);
    if (!profile) return notFound()

    return (
        <div className="px-6 flex flex-col gap-3">
            <h3 className='text-3xl font-semibold'>Profile details</h3>
            <ProfileDetailed
                profile={profile}
            />
        </div>
    );
}
```

```
'use client';

import {Card, CardBody, CardFooter, CardHeader} from "@heroui/card";
import {Profile} from "@/lib/types";
import {Button} from "@heroui/button";
import {useState} from "react";
import EditProfileForm from "@/app/profiles/[id]/EditProfileForm";
import {Avatar} from "@heroui/avatar";
import {Divider} from "@heroui/divider";

type Props = {
    profile: Profile
}

export default function ProfileDetailed({profile}: Props) {
    return (
        <Card>
            <CardHeader className='text-3xl font-semibold flex justify-between'>
                <div className='flex items-center gap-3'>
                    <Avatar className='h-30 w-30' color='secondary' />
                    <span>{profile.displayName}</span>
                </div>
                    <Button
                        variant='bordered'
                    >
                        Edit profile
                    </Button>
            </CardHeader>
            <Divider />
            <CardBody>
                    <p>{profile?.description || 'No profile description added yet'}</p>
            </CardBody>
            <CardFooter className='text-xl font-semibold'>
                Reputation score: {profile.reputation}
            </CardFooter>
        </Card>
    );
}
```

6. Create a form to edit the users profile. We are only updating the display name and description here so you can keep it simple and just use local state to swap the profile description with a form.
    1. The button will need to be visible only when the current user is viewing their own profile.
    2. Ensure on form submission the profile is visibly updated and the form is closed.

```
import {getProfileById} from "@/lib/actions/profile-actions";
import {notFound} from "next/navigation";
import {handleError} from "@/lib/util";
import ProfileDetailed from "@/app/profiles/[id]/ProfileDetailed";
import {getCurrentUser} from "@/lib/actions/auth-actions";
```

```tsx

type Params = Promise<{id: string}>
export default async function Page({params}: {params: Params}) {
    const currentUser = await getCurrentUser();
    const {id} = await params;
    const {data: profile, error} = await getProfileById(id);
    const currentUserProfile = currentUser?.id === id;

    if (error) handleError(error);
    if (!profile) return notFound()

    return (
        <div className="px-6 flex flex-col gap-3">
            <h3 className='text-3xl font-semibold'>Profile details</h3>
            <ProfileDetailed
                profile={profile}
                currentUserProfile={currentUserProfile}
            />
        </div>
    );
}
```

```tsx
'use client';

import {Card, CardBody, CardFooter, CardHeader} from "@heroui/card";
import {Profile} from "@/lib/types";
import {Button} from "@heroui/button";
import {useState} from "react";
import EditProfileForm from "@/app/profiles/[id]/EditProfileForm";
import {Avatar} from "@heroui/avatar";
import {Divider} from "@heroui/divider";

type Props = {
    profile: Profile
    currentUserProfile: boolean
}

export default function ProfileDetailed({profile, currentUserProfile}: Props) {
    const [editMode, setEditMode] = useState(false);
    return (
        <Card>
            <CardHeader className='text-3xl font-semibold flex justify-between'>
                <div className='flex items-center gap-3'>
                    <Avatar className='h-30 w-30' color='secondary' />
                    <span>{profile.displayName}</span>
                </div>
                {currentUserProfile &&
                    <Button
                        onPress={() => setEditMode(prev => !prev)}
                        variant='bordered'
                    >
                        {editMode ? 'Cancel' : 'Edit profile'}
                    </Button>}
            </CardHeader>
            <Divider />
            <CardBody>
                {editMode ? (
                    <EditProfileForm profile={profile} setEditMode={setEditMode} />
                ) : (
                    <p>{profile?.description || 'No profile description added yet'}</p>
                )}
            </CardBody>
            <CardFooter className='text-xl font-semibold'>
                Reputation score: {profile.reputation}
            </CardFooter>
        </Card>
    );
}
```

```
import {useForm} from "react-hook-form";
import {zodResolver} from "@hookform/resolvers/zod";
import {Input, Textarea} from "@heroui/input";
import {Button} from "@heroui/button";
import {Profile} from "@/lib/types";
import {useTransition} from "react";
import {editProfile} from "@/lib/actions/profile-actions";
import {handleError, successToast} from "@/lib/util";
import {editProfileSchema, EditProfileSchema} from "@/lib/schemas/editProfileSchema";

type Props = {
    profile: Profile;
    setEditMode: (value: boolean) => void;
}

export default function EditProfileForm({profile, setEditMode}: Props) {
    const [pending, startTransition] = useTransition();
    const {register, handleSubmit, formState: {isSubmitting, errors, isValid}} =
useForm<EditProfileSchema>({
        resolver: zodResolver(editProfileSchema),
        mode: 'onTouched',
        defaultValues: {
            displayName: profile.displayName,
            description: profile.description
        }
    })

    const onSubmit = (data: EditProfileSchema) => {
        startTransition(async () => {
            const {error} = await editProfile(profile.userId, data);
            if (error) handleError(error);
            successToast('Profile successfully updated');
            setEditMode(false);
        })
    }

    return (
        <form onSubmit={handleSubmit(onSubmit)} className='flex flex-col gap-4'>
            <Input
                {...register('displayName')}
                label='Display name'
                isInvalid={!!errors.displayName}
                errorMessage={errors.displayName?.message}
            />
            <Textarea
                {...register('description')}
                label='Description'
                rows={4}
                isInvalid={!!errors.description}
                errorMessage={errors.description?.message}
            />
            <Button
                isLoading={isSubmitting || pending}
                isDisabled={isSubmitting || !isValid}
                color='secondary'
                className='w-fit'
                type='submit'
            >Submit</Button>
        </form>
    );
}

```

7. Update the link in the UserMenu to take them to their own profile.

Challenge complete!!