Conținut

1. Run React app
2. Componente noi Shadcn
3. Get all courses - Axiox + TanStack Query
4. Create Course - TanStack Query [mutation]
5. React HookForm + zod
6. Paginare - extra

Resurse

🌐Shadcn: https://ui.shadcn.com/docs/components
🌐Axios: https://blog.logrocket.com/using-axios-with-react-native-manage-api-requests/
🌐TanStack Query DevTools: https://chromewebstore.google.com/detail/tanstack-query-devtools/annajfchloimdhceglpgglpeepfghfai
🌐ReactHookForm +Zod: https://www.youtube.com/watch?v=cc_xmawJ8Kg
🌐`Lucide:` `https://www.npmjs.com/package/lucide-react` ⇒ o sa aveti nevoie pentru iconote

1. Run React app

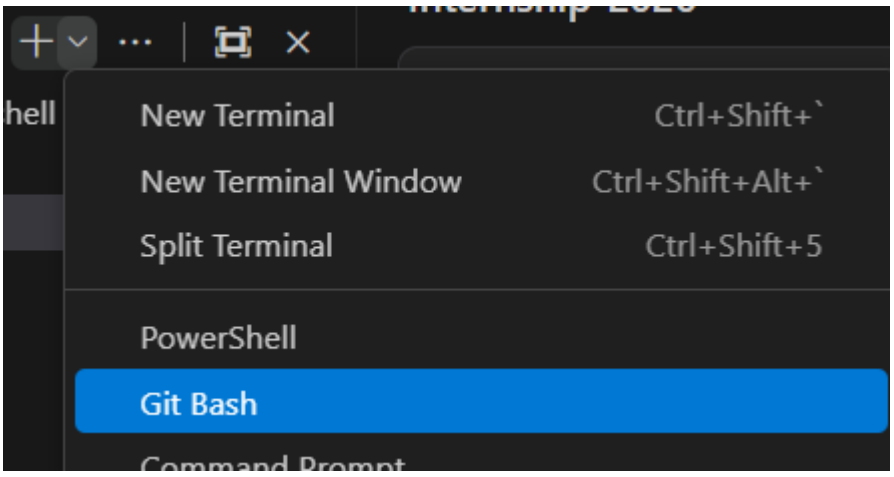
npm run install [daca abia ati clonat repository-ul]
npm run dev

2. Componente noi Shadcn

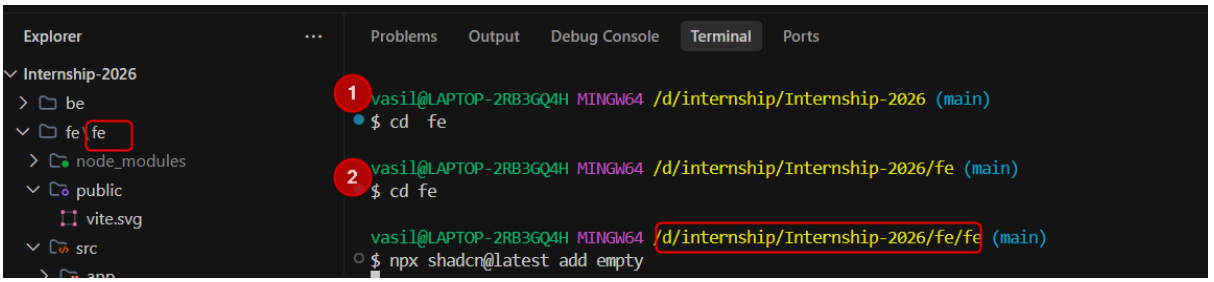
🌐 https://ui.shadcn.com/docs/components

Daca aveti nevoie de componente noi, vom folosi Shadcn(link mai sus).

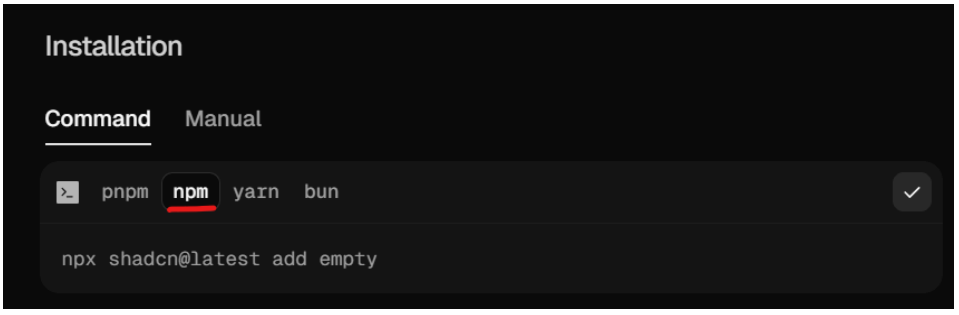
Exemplu: Empty: https://ui.shadcn.com/docs/components/radix/empty
Pas1: Deschide un Git Bash Terminal:



Pas2: Asigura-te ca deschizi terminalul si te afli la path-ul corect, unde ai rulat npm run dev! Daca a fost nevoie sa schimbati directorul cand ati rulat npm, sa nu uitati si aici:
Ex:

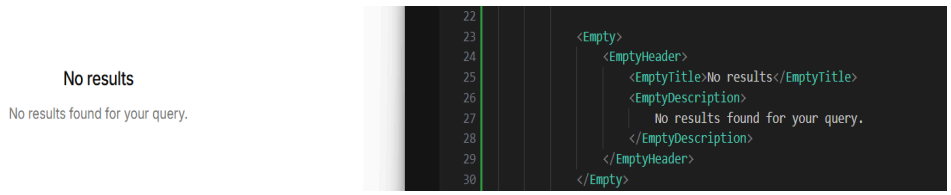


Pas3: instaleza componenta folosind comanda ce o gasesti in shadcn [npm]



Obs: componenta noua va aparea in \fe\src**components**

Pas4: Schimare culoare la EmptyTitle in tot sisitemul



Pentru a modifica in componemta, trebuie sa mergem direct in fiiserul de gnerat de shadcm, in componenta <EmptyTitle>, si saadaugi in className :

```
className={cn("text-lg font-medium tracking-tight text-red-500", className)}
```

Pas5: Schimbam culoarea la EmptyTitle intr-un singur loc
Este nevoie doar de adaugarea className:

```
<EmptyTitle className="text-green-500">No results</EmptyTitle>
```

!!La unele componente din shadcn, este nevoie sa puneti "!" ca sa suprascrieti o clasa

```
className="!text-green-500"
```

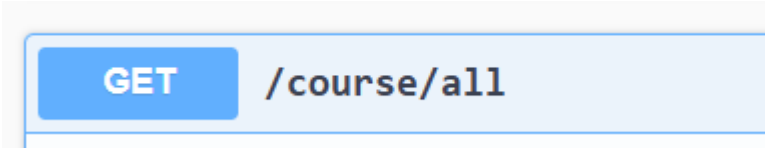
!!Atentie - daca deja componenta are deja setata o clasa si mai adaugati si voi una, se va suprascrie cu ultima existente→ in cazul de mai jos, va fi textul rosu ⇒ Deci fiti atenti si modificati dupa preferinta, nu adaugati una suplimentara

```
className={cn("text-lg text-green-500 font-medium tracking-tight text-red-500", className)}
```

3. Get all courses

Eu o sa fac un exemplu mai simplificat, e posibil sa nu fie 1 la 1 cu ce ati implementat voi 😊

Pas1: Mergem in Swagger la request-ul



Pas2: Ne uitam la response (e ultimul lucru pe care il vedeti cand faceti expanf la request),

Deci vedem ca avem o lista de CourseDto, care are: **id, title, description** si o lista numita **lessons** cu cateva campuri.

Pas3: Mergem in FE si ne facem type-urile:

```
export type Lesson = {
    id: string,
    content: string,
}
```

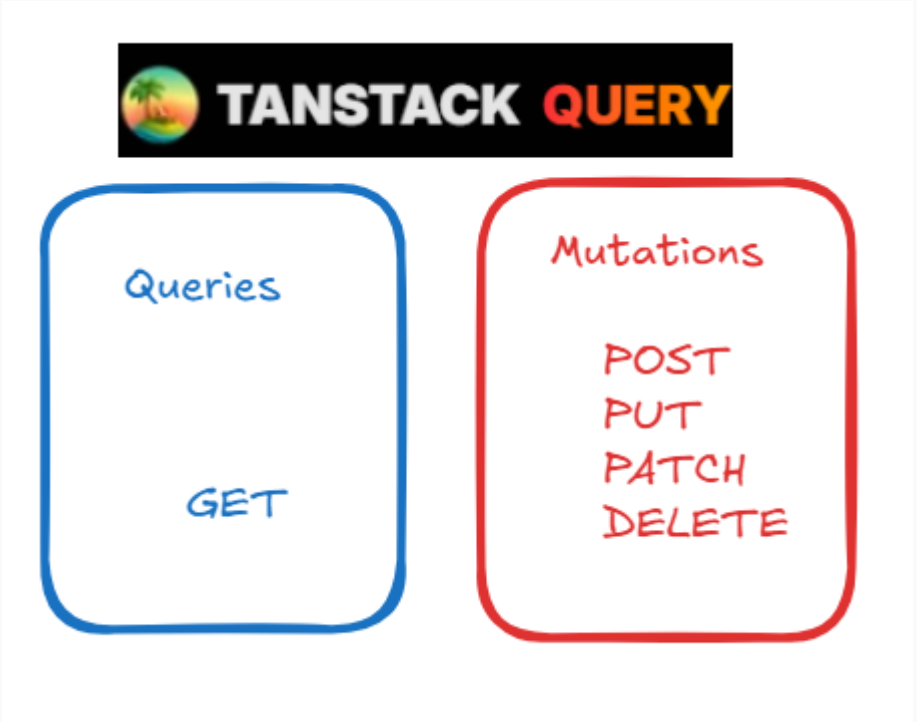
```
export type Course = {
    id: string,
    title: string,
    description: string,
    lessons: Lesson[],
}
```

Pas4: Ne definim API-ul

🌐Axios: https://blog.logrocket.com/using-axios-with-react-native-manage-api-requests/

```
const getAllCourses = async (): Promise<Course[]> => {
    const response = await axios.get<Course[]>("/course/all")
    return response.data
}
```

Pas5: Folosim Tanstack Query [in acelasi fisier unde s-a definit `getAllCourses`]




```
const useAllCourses = () => {
    return useQuery({
        queryKey: ["courses"], //cheie pt. cache unde e salvat
        queryFn: getAllCourses,
    })
}
```

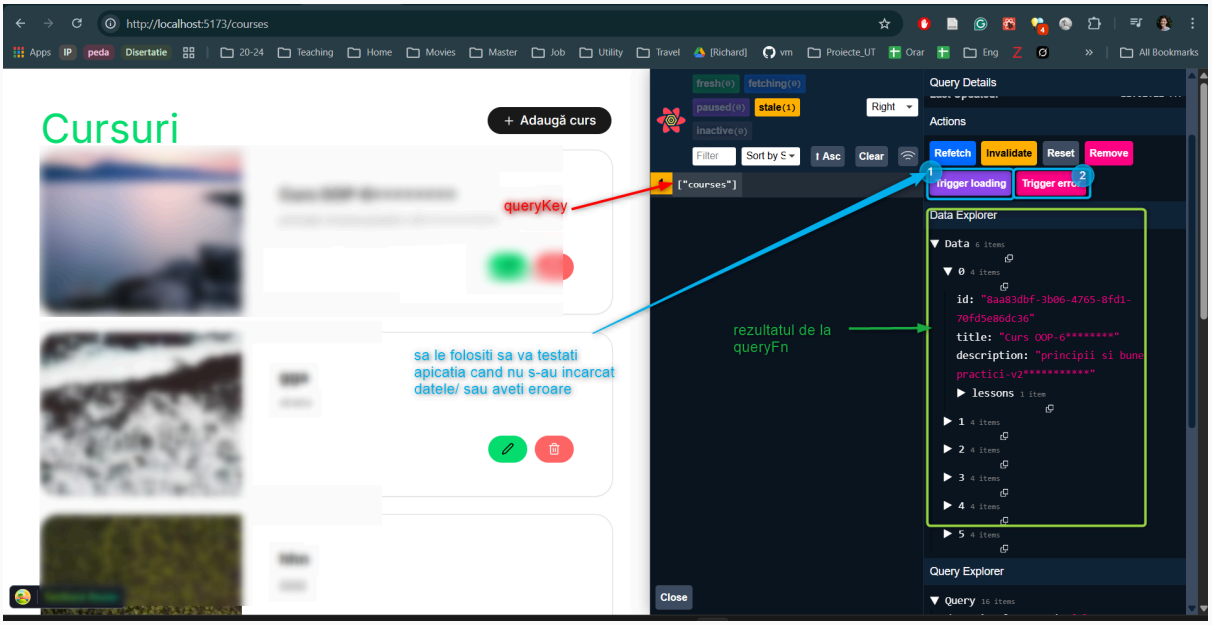
Pas6: Se va apela in pagina Courses hook-ul

```
  const { data, isLoading, error } = useAllCourses();
```

Pasul 7: vor afisa elemente reprezentative pentru starile: isLoading [eg: spinner], error [un mesaj explicit], cand nu sunt elemente de afisat, + cursurile propriu-zise

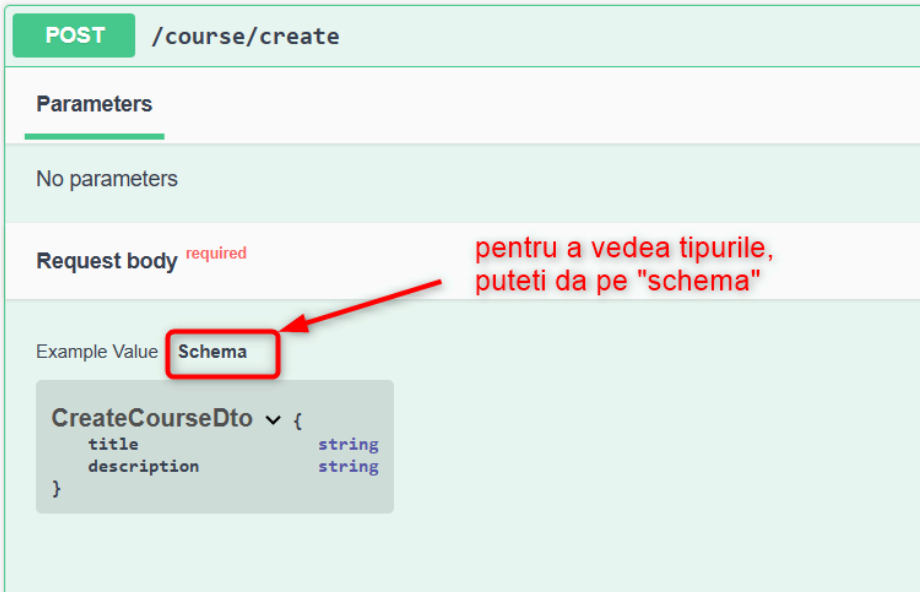
```tsx
const CourseList = () => {
    const { data, isLoading, error } = useAllCourses();

    if (isLoading) {
        return <div><Spinner /></div>;
    }

    if (error) {
        return <div>A apărut o eroare la încărcarea cursurilor.</div>;
    }

    return (
        <>
            <div className="flex flex-col gap-6">
                {data?.map((course: Course) => (
                    <CourseItem key={course.id} course={course} />
                ))}
            </div>
        </>
    );
};
```

Pasul 8: verificare date in UI cu **TanStack Query DevTools**
🌐https://chromewebstore.google.com/detail/tanstack-query-devtools/annajfchloimdhceglpgglpeepfghfai



4. Create Course

Pas1: ne uitam in Swagger + facem type pentru body!!!



Pas2: axios-POST + body

```
const crateCourse = async (body: CreateCourseBody) => {
    const response = await axios.post("/course/create", body)
    return response.data
}
```

Pas3: TanStackQuery-Mutation

```
const mutateCreateCourse = () => {
    const queryClient = useQueryClient(); // este hook folosit pentru mamagement al cache-ului

    return useMutation({
        mutationFn: (body: CreateCourseBody) => crateCourse(body),
        onSuccess: () => {
            queryClient.invalidateQueries({ queryKey: ['courses'] }) forteaza cash-ul "courses" sa se reactualizeze
        },                                                           <=> refresh
    })
```

5. React HookForm + zod

🌐 https://www.youtube.com/watch?v=cc_xmawJ8Kg

Acum vrem sa legam API-ul de formularul cu datele despre curs - pentru create:

Pas1: se adauga tag-ul <form> </form>

Pas2: se introduc field-urile title si description si butonul cu type="submit"

```
<Input placeholder="Titlu" />
<Textarea placeholder="Descriere"/>
<Button type="submit">Adaugă curs</Button>
```

Pas3: Se defineste schema de validare cu zod, unde se pun restrictii pe campuri: lungimi minime, mesaje de eroare

```
const courseSchema = z.object({
  title: z.string().min(2, 'Titlul trebuie sa aiba minim 2 caractere'),
  description: z.string().min(2, 'Descrierea > 2 caractere')
 })
```

Pas4: generam un tip din schema de validare

```
type CourseSchema = z.infer<typeof courseSchema>
```

Pas5: cream useForm, unde adaugam

- resolver - care trimite datele la zod sa le verifice
- register - leaga input-urilr html cu logica din formular

```
const { register, handleSubmit, formState: { errors, isValid }, reset }
= useForm<CourseSchema>({
        resolver: zodResolver(courseSchema),
        values: {
            title: '',
            description: ''
        }
    });
```

Pas6:

```
// acces zod la campuri pentru a le valida + afisare erori
<Input placeholder="Titlu"  {...register('title')} />
{errors.title &&
 <p className="text-red-500">{errors.title.message}</p>}
```

Pas7: submit + hook-ul de create construit la Punctul 4

```
 <form onSubmit={handleSubmit(onSubmit)}>
```

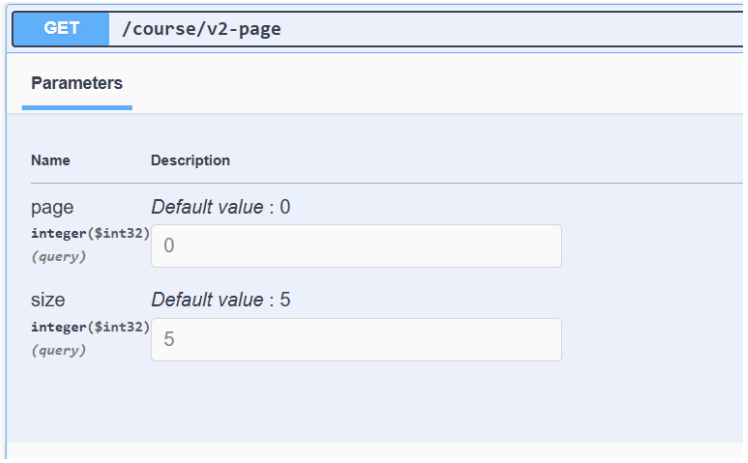
```
 const createCourseMutation = createCourseMutation();
 const onSubmit = (data: CourseSchema) => {
```

```
createCourseMutation .mutate(data, {
   onSuccess: () => {
   reset(); // sa nu ramana datele anterioare la next create
    }
   });
   }
```

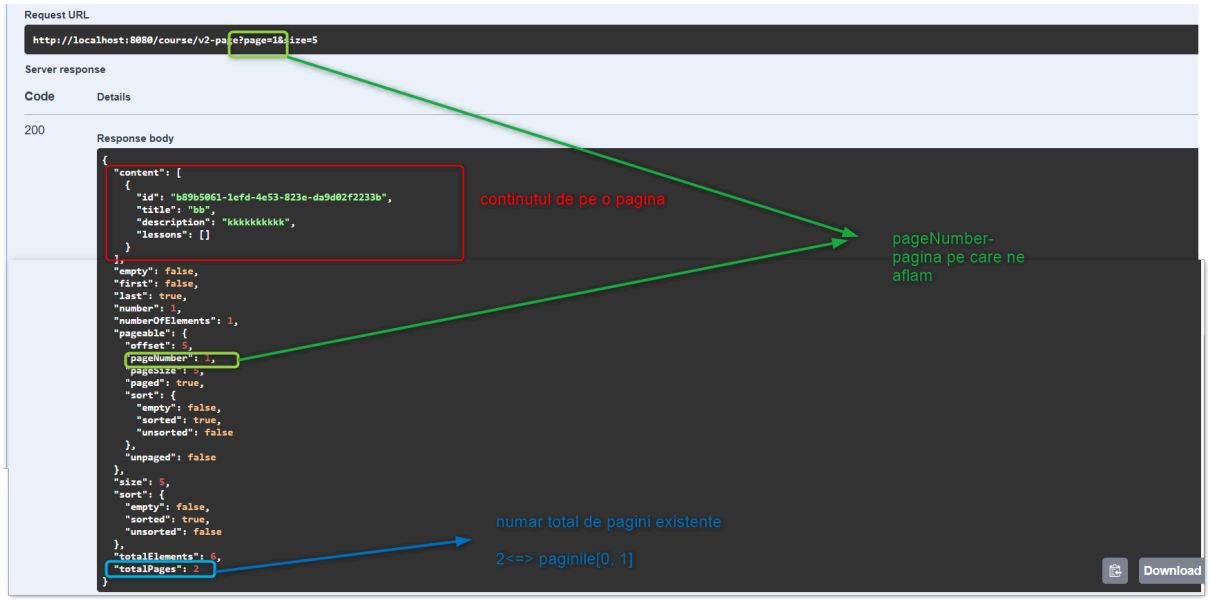
6. Paginare- extra

➡️doar daca aveti endpoint-ul in BE si doriti sa legati cu FE

Pas1: ne uitam in Swagger sa vedem structura end-point-ului + raspunsul



Raspuns- explicatii



Pas2: facem type nou pentru raspuns care sa cuprinda paginarea:

```
export type PaginatedCourse={
    content:Course[];
    pageable:{
        pageNumber: number
    };
    totalPages: number
}
```

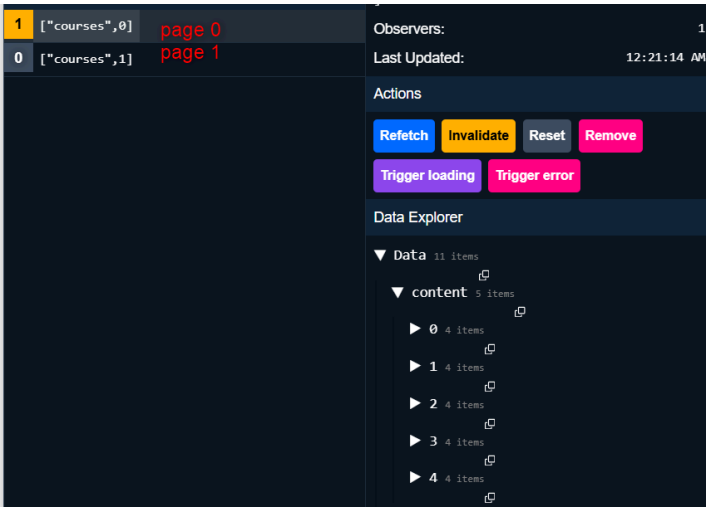
Pas3: Axios:

```
import type { PaginatedCourse } from "@/app/course/types/course"
import { useQuery } from "@tanstack/react-query";
import axios from "axios"

const getPaginatedCourse = async ({ page, size }: { page: number, size:
number }): Promise<PaginatedCourse> => {
    const response = await
axios.get<PaginatedCourse>(`/course/v2-page?page=${page}&size=${size}`)
;
    return response.data;
}
```

Pas4: TanStackQuery

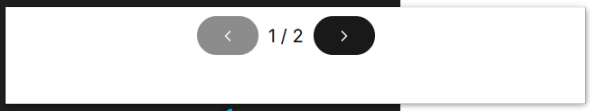
!!acum, fiecare pagina va fi salvata in cache separat, in functie de pagina la care se afla din acest motiv folosim un array ca queryKey , care cuprinde si pagina

```
const queryPaginatedCourse = ({ page, size }: { page: number, size:
number }) => {
    return useQuery({
        queryKey: ['courses', page],
        queryFn: () => getPaginatedCourse({ page, size }),
    })
}
```



Pas5: UI-paginare

```tsx
const CourseList = () => {
    const [page, setPage] = useState(0);

    const { data, isLoading, error } = queryPaginatedCourse({ page, size: 5 });

    if (isLoading) {
        return <div><Spinner /></div>;
    }

    if (error) {
        return <div>A apărut o eroare la încărcarea cursurilor.</div>;
    }

    return (
        <>
            <div className="flex flex-col gap-6">
                {data?.content?.map((course: Course) => (
                    <CourseItem key={course.id} course={course} />
                ))}
            </div>

            <div className="flex items-center justify-center gap-2 mt-4">
                <Button onClick={() => setPage(page - 1)} disabled={page === 0}>
                    <ChevronLeft />
                </Button>
                {page + 1} / {data?.totalPages}
                <Button onClick={() => setPage(page + 1)} disabled={page === (data?.totalPages ?? 1) - 1}>
                    <ChevronRight />
                </Button>
            </div>
        </>
    );
};
```



‼Foarte important

Invalidarea Query-urilor se face in functie de prefix, deci, trebuie sa folositi acelasi prefix la queryKey la delete, get, ca altfel, o sa stergeti un element si nu o sa se faca refresh