07

Padrão debounce com RxJS

Marina aprendeu durante a aula que é boa prática implementar um padrão de projeto chamado **debounce** toda vez que for executar uma operação que será disparada repetidas vezes de acordo com eventos gerados pelo usuário. Onde um claro exemplo de situação assim é a filtragem de fotos. Então ela fez o código abaixo:

```
// photo-list.component.ts

@Component({
    // omitidas as configurações do componente.
})
export class PhotoListComponent implements OnInit, OnDestroy {

    photos: Photo[] = [];
    filter: string = '';
    debounce: Subject<string> = new Subject<string>();

    constructor(private activatedRoute: ActivatedRoute) {}

    ngOnInit() {
        this.photos = this.activatedRoute.snapshot.data.photos;
        this.debounce
            .pipe(debounceTime(300))
            .subscribe(filter => this.filter = filter);
    }

    ngOnDestroy() {
        this.debounce.unsubscribe();
    }
}
```

```
<!-- photo-list.component.html -->

<div class="text-center mt-3 mb-3">
    <form>
        <input class="rounded" type="search" placeholder="search..."
            autofocus (keyup)="debounce.next($event.target.value)">
    </form>
</div>

<ap-photos [photos]="photos | filterByDescription: filter"></ap-photos>
```

Marque a afirmativa correta sobre o código de nossa colega!

Selecione uma alternativa

A O código irá funcionar corretamente e a listagem de fotos será filtrada apenas uma vez após 300ms depois da última tecla ter sido apertada!

B

O código não funciona corretamente pois a forma correta de implementar o debounce seria:

```
this.debounce
    .debounceTime(300)
    .subscribe(filter => this.filter = filter);
```

C

O código não funciona corretamente pois a forma correta de implementar o debounce seria:

```
this.debounce
    .debounce(300)
    .subscribe(filter => this.filter = filter);
```