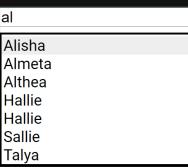
Exercice

- Dans la suite du cours, nous allons essayer d'implémeter le composant suivant :
- Considérons un serveur qui expose une liste de clients à l'url : http://localhost:3000/clients

```
export type Client = {
  id: string,
  firstname: string,
  lastname: string,
  age: number
}
```

Créer un composant "searchbar avec autocomplétion" (uniquement sur le firstname)



fromEvent()

```
fromEvent<T>(target: any, eventName: string): Observable<T>;
```

- Fonction fournie par RxJS
- Crée un observable à partir d'un évènement (eventName) d'un élément (target) du DOM
- L'observable généré émet une valeur à chaque fois que l'évènement est déclenché

fromEvent()

- Dans une application Angular, on peut utiliser une ViewQuery pour récupérer les éléments du DOM
- Template:

```
<input #input />
```

Composant:

```
@ViewChild('input')
input!: ElementRef<HTMLInputElement>
...

ngAfterViewInit() {
  fromEvent(this.input, 'input').subscribe( ... )
}
```

HttpClient

- Le service HttpClient d'Angular fournit des méthodes pour faire des requètes HTTP
- Un seul provider pour HttpClient dans l'application. On peut définir un provider sur le AppModule, ou dans la methode bootstrapApplication pour un composant standalone

```
bootstrapApplication(HttpClientComponent, {
    providers: [
        provideHttpClient()
    ]
}).catch(err \Rightarrow console.error(err));
```

Puis on injecte le service HttpClient

```
private _http = inject(HttpClient)
```

 En général le HttpClient n'est pas injecté directement dans les composants, on passe par un service intermédiare

Remarques sur les méthodes de HttpClient

- Le service HTTP client fournit des méthodes, permettant de faire des requètes HTTP
- Les méthodes renvoient des observables
- La requète HTTP n'est émise que lors de l'abonnement à l'observable (lazy)
- Une requète est envoyée pour chaque abonnement à l'observable
- L'observable emet la réponse une fois retournée par le serveur
- Le désabonnement de ces observables est géré par Angular

GET / get()

- La requête GET permet de récupérer des données
- La méthode get() de HttpClient prend deux paramètres d'entrée, l'url à requêter et des paramètres optionels

```
get<T>(url: string, options): Observable<T>
```

- Par défaut, les données de retour sont supposées au format JSON, et converties en Object
- Il est possible de spécifier un type de retour de l'observable de la methode get()

```
posts$ = this.http.get<Articles[]>("http://localhost:3000/articles")
```

■ 🛕 Il n'y a aucune garantie que le type des données renvoyées par le serveur corresponde au type spécifié

json-server

- Package node pour simuler un service REST facilement
- Installaton:

```
npm install -g json-server
```

Utilisation:

```
json-server --watch {file.json}
```

Le serveur est prêt à étre utilisé

json-server

- Quelques options qui vont nous servir pour notre exercice :
- Add _like to filter (RegExp supported)

```
GET /articles?title_like=server
```

Add _sort and _order (ascending order by default)

```
GET /articles?_sort=views&_order=asc
GET /articles/1/comments?_sort=votes&_order=asc
```

• For multiple fields, use the following format:

```
GET /articles?_sort=user, views&_order=desc, asc
```

HttpParams

Pour certaines requètes, il est possible de passer des paramètres dans l'url

```
return this._http.get<Articles[]>(`${this.baseUrl}?author_like=${filter}&_sort=title&public=true`)
```

- Plutot que de tout mettre directement dans l'url lors de l'appel a get(), il est possible d'utiliser un objet
 HttpParams dans les options
- La classe HttpParam permet de stocker les paramètres sous forme d'une liste clé/valeur
- L'objet HttpParam est immutable, toutes les opérations renvoient un nouvel objet

```
let options = {
  params: new HttpParams()
    .append('author_like', filter)
    .append('_sort', 'title')
    .append('public', true)
}
return this._http.get<Articles[]>(`${this.baseUrl}`, options)
```

Exercice

- Implémenter deux parties de notre barre de recherche :
 - Un composant avec un input, dans lequel le changement de cet input sera détecté et affiché dans la console
 - Un service ClientService, qui expose une méthode getFilteredClients(filter: string), qui fait une requète
 HTTP à notre serveur pour retourner la liste des clients filtrés