Intercepteur fonctionnel

- Depuis la version 14 d'Angular, il est possible de déclarer les intercepteurs sous forme de fonction plutôt que sous forme de classe
- L'écriture est très similaire au service, sauf qu'on utilise directement des fontions
- Dans les versions antérieures d'Angular, les intercepteurs étaient sous forme de service car c'était la seule manière de pouvoir injecter d'autres services
- La fonction inject() a permit de pouvoir créer des intercepteurs fonctionnels

Intercepteur fonctionnel

- Exemple de NoopInterceptor fonctionnel :

```
export const noopIntercept: HttpInterceptorFn = (
  req: HttpRequest<unknown>, next: HttpHandlerFn
): Observable<HttpEvent<unknown>> => {
  return next(req)
}
```

- Le type HttpInterceptorFn ressemble à la méthode intercept de l'interface HttpInterceptor

```
export declare type HttpInterceptorFn =
  (req: HttpRequest<unknown>, next: HttpHandlerFn) => Observable<HttpEvent<unknown>>;
```

- Le type HttpHandlerFn ressemble à la méthode handle de l'interface HttpHandler

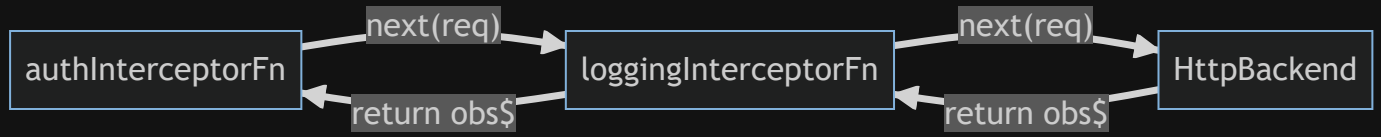
```
export declare type HttpHandlerFn = (req: HttpRequest<unknown>) => Observable<HttpEvent<unknown>>;
```

Intercepteur fonctionnel

- Utilisation de l'intercepteur :

```
provideHttpClient(withInterceptors(
  [noopIntercept]
))
```

Chaine d’intercepteurs



- Dans la chaine, il y a un empilement des appels des intercepteurs
  - authInterceptorFn appelle loggingInterceptor qui appelle le HttpBackend
  - HttpBackend retourne un observable à LoggingInterceptor qui retourne un observable à HttpBackend
- Pendant la chaine, les différents intercepteurs peuvent modifier la requète, mais aussi la réponse
- Cet empilement signifie que LoggingInterceptor va voir la requète après qu’elle soit passée dans tous les intercepteurs de la chaîne, mais la réponse en premier

HttpRequest

- HttpRequest est une classe représentant une requète Http. Une instance est crée par HttpClient et passe dans les différents intercepteurs

- L'objet HttpRequest de Angular est immutable, il est nécessaire de faire une copie à chaque intercepteur, pour cela, on utilise la méthode clone()

```
const clone = req.clone({
  headers: req.headers.set('Authorization', authToken)
});
```

- Les headers sont aussi un objet immutable, la requète clonée à aussi un clone des headers

HttpContext

- Les intercepteurs sont appelés pour chaque requète Http
- Parfois, on veut pouvoir avoir un traitement particulier pour certaines requètes
- Pour cela, peut définir un HttpContext dans une requète, qui est une map de HttpContextToken, auquel on peut définir une valeur particulier dans chaque requète
- HttpContextToken<T> est une classe générique

HttpContext

- Définition du token :

```
export const IS_LOGGING_ENABLED = new HttpContextToken<boolean>(() => false)
```

- Lorsque l'on définit un token, on doit définir une valeur par défaut dans le constructeur

- Utilisation dans l'intercepteur :

```
  if (req.context.get(IS_LOGGING_ENABLED) === true) {
    return ...;
  }
  ...
```

- Utilisation dans le HttpClient :

```
return this._http.get(this.baseUrl, {
  context: new HttpContext().set(IS_LOGGING_ENABLED, true)
})
```

TrackBy

TrackBy

- Tous les éléments à l'intérieur de la balise portant la directive *ngFor (ou @for) sont dupliqués pour chaque membre de la collection
- Lorsque un membre de la collection change, Angular détruit tous les éléments correspondants, et recrée des nouveaux éléments avec le nouveau membre
- Le fait de détruire et de recréer des éléments du DOM est une opération couteuse, qui peut ralentir les performances de l'applciation si elle est faite trop souvent
- Pour savoir si un membre de la collection par défaut, Angular compare les références des objets

- On peut définir une autre fonction pour comparer les objets

```
<div *ngFor="let item of collection; trackBy: trackByFunction">
  ...
```

```
...
trackByFunction(index: number, item: any): any {
  return item
}
...
```

- index est la position de l'objet dans la liste, item est une référence à l'objet
- Le retour le la fonction trackBy est utiliser pour déterminer si l'objet à changé
- Il est très imporant que la fonction trackBy renvoie un résultat unique pour chaque élément de la collection
- Avec @for, définir cette fonction est obligatoire