State management

- Il existe des libraires dédiées au state management pour Angular (NgRx, NGXS), mais elles ne sont pas obligatoires
- Dans le futur, state management avec des Signals ?

Retour sur HttpClient

Méthodes HTTP

- Les méthodes pour les requètes HTTP courantes sont également implémentées dans le service HTTPClient :
  - post() / POST
  - put() / PUT
  - delete() / DELETE
  - patch() / PATCH
  - head() / HEAD
  - options() / OPTIONS

POST / post()

- La requète post permet d'envoyer des données au serveur

```
_httpClient.post(url, body, options): Observable<Object>
```

- Comme pour la requète get(), le type de retour de la méthode est un observable, qui renvoie le contenu du body de la réponse du serveur, supposée au format JSON et convertie en objet

Typage

- Par défaut le client http de Angular retourne uniquement le body de la réponse de serveur, au format JSON convertie en object

- On peut utiliser les options dans les méthodes du client pour changer cela :

```
responseType?: 'json' (default) | 'arraybuffer' | 'blob' | 'text'
```

```
observe?: 'body' (default) | 'events' | 'response'
```

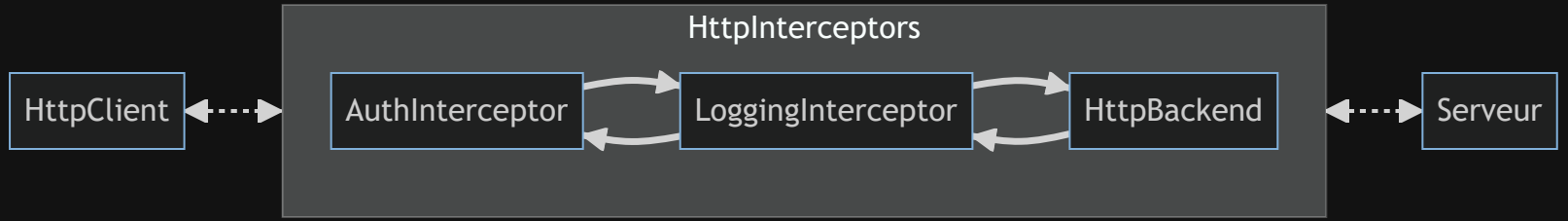
- Le type de retour de l'observable change avec les paramètres

HttpInterceptor

HttpInterceptor

- Dans une application Angular, il y a des traitements que l'on veut appliquer sur toutes les requètes que l'on envoie, ou sur toutes les réponses que l'on recoit

- Pour cela, on va définir des HttpInterceptor



- Les intercepteurs se composent sous forme de chaîne
- Le dernier intercepteur de la chaîne est HttpBackend, qui s'occupe de l'evnoi de la requète

HttpInterceptor

- La manière historique de faire un intercepteur est un service injecté :

```
@Injectable()
export class NoopInterceptor implements HttpInterceptor {
```

- Angular nous fournit l'interface HttpInterceptor comme modèle pour créer un intercepteur

```
interface HttpInterceptor {
  intercept(req: HttpRequest<any>, next: HttpHandler): Observable<HttpEvent<any>>
}
```

- Cette interface ne contient qu'une méthode, intercept, que nous allons étudier

- Nous alons d'abord voir un intercepteur qui ne fait rien :

```
intercept(req: HttpRequest<any>, next: HttpHandler): Observable<HttpEvent<any>> {
  return next.handle(req);
}
```

- La méthode intercept prend 2 paramètres :
  - req : la requète http envoyée par intercepteur précédent
  - next : le traitement de l'intercepteur suivant dans la liste

- HttpHandler est une classe abstraite qui représente le traitement d'un intercepteur

```
abstract class HttpHandler {
  abstract handle(req: HttpRequest<any>): Observable<HttpEvent<any>>
}
```

- Le retour de la méthode est l'observable avec la réponse
- L'observable est renvoyé à l'intercepteur précédent

Utilisation (avec DI)

- Nous avons maintenant défini un intercepteur, mais comment l'utiliser ?
- On a vu que notre provider est un service injectable, on va donc utiliser l'injection de dépendances
  - Pour cela, Angular nous met à disposition un token d'injection : HTTP_INTERCEPTORS

```
{ provide: HTTP_INTERCEPTORS, useClass: NoopInterceptor, multi: true }
```

- Quelques remarques :
  - Le paramètre "multi: true" indique qu'il est possible de spécifier plusieurs providers pour une dépendance (ici un provider par intercepteur)
  - ⚠️ L'ordre des providers à une importance
  - Les intercepteurs sont une dépendance optionelle de HttpClient, il faut donc les fournir à l'instanciation du HttpClient, dans le même provider ou un parent