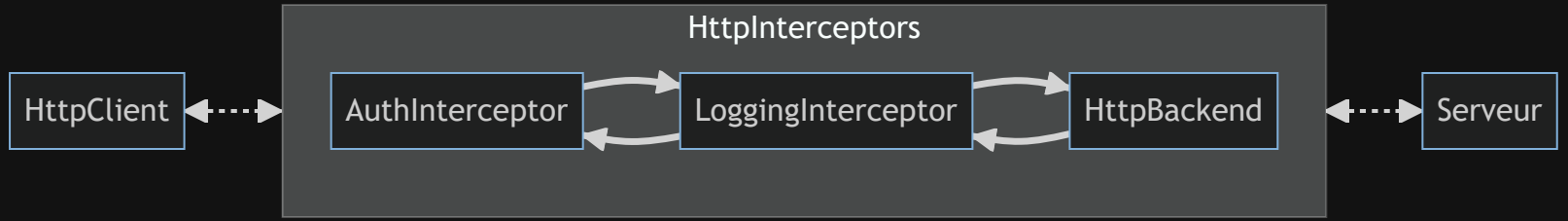
HttpInterceptor

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

- Dans une application, les intercepteurs sont des services injectés

```
@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);
}
```

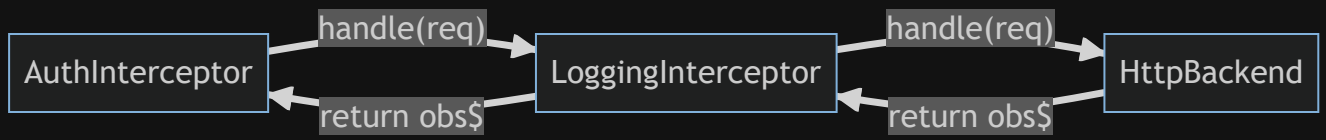
- La méthode intercept prend 2 paramètres :
  - req : la requète http envoyée par intercepteur précédent
  - next : le traitement de l'intercepteur suivant dans la liste

- 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

Chaine d’intercepteurs



- Dans la chaine, il y a un empilement des appels à handle
  - AuthInterceptor appelle le handle() de LoggingInterceptor qui appelle le handle() de HttpBackend
  - HttpBackend retourne un observable à LoggingInterceptor qui retourne un observable à HttpBackend
- 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

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

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

AuthInterceptor

- https://angular.io/guide/http-interceptor-use-cases#set-default-headers

```
intercept(req: HttpRequest<any>, next: HttpHandler) {
  // Get the auth token from the service.
  const authToken = this.auth.getAuthorizationToken();

  // Clone the request and replace the original headers with
  // cloned headers, updated with the authorization.
  const authReq = req.clone({
    headers: req.headers.set('Authorization', authToken)
  });

  // send cloned request with header to the next handler.
  return next.handle(authReq);
}
```

- Notre intercepteur peut aussi dépendre d'autres services :

```
  constructor(private auth: AuthService) {}
```

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)
})
```

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
- Exemple de NoopInterceptor fonctionnel :

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

- Utilisation de l'intercepteur :

```
provideHttpClient(withInterceptors(
  [intercept]
))
```

Exercice : LoggingInterceptor

- Créer un intercepteur LoggingInterceptor, qui va logger toutes les *réponses* retournées par le serveur
- Le LoggingInterceptor utilisera un service pour gérer le log
- Intércepteur classique ou fonctionnel
- Utiliser un HttpContextToken IS_LOGGING_ENABLED pour activer ou pas l'intércepteur